

Digital Signal Processing Control of Electric Machines and Drives Laboratory

(Implemented on Texas Instruments DSP TMS320F240 EVM)

Ali Keyhani

Professor of Electrical and Computer Engineering

Min Dai

Ph.D. Student, Lab Assistant

**Mechatronics Laboratory
Department of Electrical Engineering**

The Ohio State University

March 2002

Tel: (614) 292-4430

Fax: (614) 292-7596

E-mail: keyhani.1@osu.edu

Abstract

This laboratory is designed to provide first time users of TI DSP TMS320F240 with fundamental experiments in the use of digital signal processor based systems for control of electric machines and power electronic drives.

This report covers introduction to laboratory hardware, software, and assembly language programming, digital interfacing (general purpose digital I/O and relays and switches interfacing), timer operations and interrupt systems (programming the timer functions including comparison, input capture, and pulse-width modulator, application of timer functions including time delay, waveform generation and frequency measurement), analog interfacing (A/D and D/A operations and programming), stepper motor drive and shaft encoder (stepper motor drive circuits and encoders operations and interfacing), DC motor speed control (base drive and power converter circuits and open loop speed control), serial communication and programming for data transfer, and generation of a sine modulated PWM signal.

Chapters 1 and 2 are the introduction of TI DSP TMS320F240 and assembly language programming fundamentals with example programs. From Chapter 3, each chapter contains objectives and general introductions, detailed operating principles, device architectures, register descriptions, peripheral circuitry descriptions, example programs, and laboratory procedures and assignments. This report consists of 313 pages.

Table of Contents

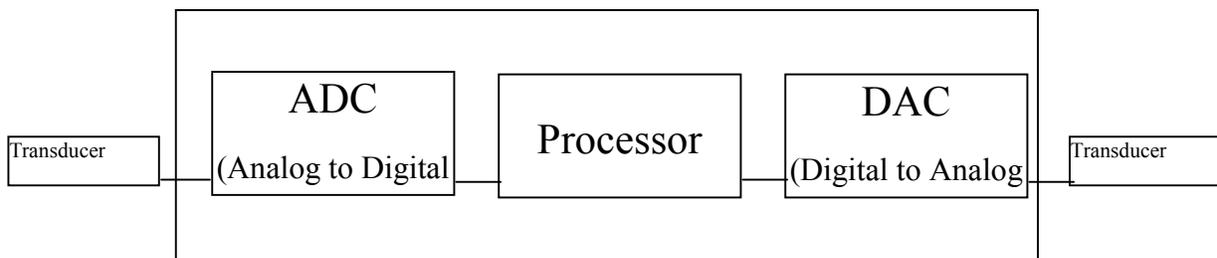
Chapter 1	Introduction to digital signal processing	1
Chapter 2	Overview of TMS320F240 DSP and introduction to assembly language programming	6
Chapter 3	Digital I/O ports, relays & switches interfacing	82
Chapter 4	Interrupts & Timer Operations	116
Chapter 5	Analog interfacing: A/D and D/A converters	173
Chapter 6	Stepper motor drives and shaft encoders	200
Chapter 7	DC motor drive	234
Chapter 8	Serial communications interface	268
Chapter 9	Creating a sine modulated PWM signal	294

CHAPTER 1

INTRODUCTION TO DIGITAL SIGNAL PROCESSING

This course discusses the design of electromechanical systems with the Texas Instruments DSP TMS320F240. In this chapter, the basic concepts of digital signal processors and their applications will be discussed. The basic features of the TMS320F240 will also be discussed.

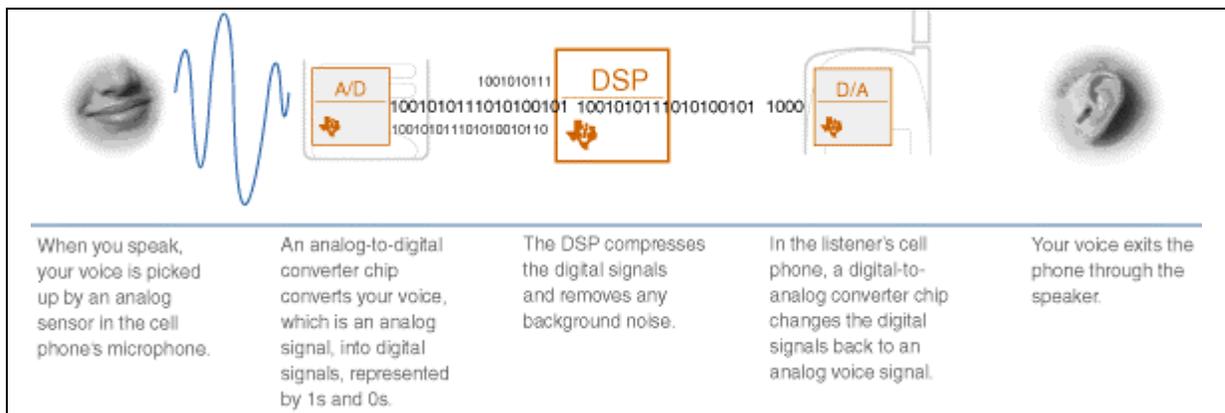
DSP processors are microprocessors designed to perform digital signal processing -- the mathematical manipulation of digitally represented signals. All the signals we need to process in the environment are analog in nature. In order to manipulate and efficiently process these signals, it is very important to convert them into digital data that can be handled by the processor. The device that converts an analog signal into binary code (digital data) is called an analog to digital converter. A DSP system primarily consists of a transducer to convert the physical quantity into an electrical signal, Analog to Digital Converter, a processor, a Digital to Analog converter and another transducer to convert the electrical signal into the physical form. The block diagram is as shown below:



A digital signal processor (DSP) is a specialized microprocessor - one that is incredibly fast and powerful. What makes a DSP so powerful and unique, is its capability to handle real-time data.

A typical example of a DSP system is the cellular telephone. We speak in real-world, analog signals. This signal needs to be converted into a digital signal, which is a language of 1s and 0s that can be processed by mathematics. This is essential for the digital signal processor, which can handle signals at incredible speeds. This is the signal that the DSP works upon. When it has completed the signal improvement, a digital to analog converter translates the digital signal back to analog.

The cellular phone example is illustrated in the figure below:



The features that make a DSP so much more powerful than an ordinary microprocessor or micro-controller are:

1. Ability to perform a multiply-accumulate operation (often called a "MAC") in a single instruction cycle. The multiply-accumulate operation is useful in DSP calculations such as the vector dot product used for digital filtering.
2. Ability to complete several accesses to memory in a single instruction cycle. This allows the processor to fetch an instruction while simultaneously fetching operands and/or storing the result of a previous instruction to memory

3. Harvard architecture of the DSP core provides separate internal buses for data and program memory, which results in faster execution of instructions. Almost every instruction can be completed in one machine cycle.
4. To allow low-cost, high-performance input and output, most DSP processors incorporate one or more serial or parallel I/O interfaces, and specialized I/O handling mechanisms such as low-overhead interrupts and direct memory access (DMA) to allow data transfers to proceed with little or no intervention from the rest of the processor.

The various applications of DSP processors in electromechanical systems include:

1. Industrial motor drives
2. Power inverters and controllers
3. Automotive systems, such as electronic power steering, anti-lock brakes, and climate control.
4. Appliance and HVAC blower/compressor motor controls
5. Printers, copiers and other office products
6. Tape drives, magnetic optical drives, and other mass storage products
7. Robotics and CNC milling machines

Through this course, we are going to perform the following labs to understand some specific applications of the Texas Instruments DSP, TMS320F240 to electromechanical systems-

1. Introduction to Laboratory hardware, software and assembly language programming.

Description:

Hardware - Fundamentals of DSP architecture. Features of the TMS320F240 DSP.

Software - Introduction to assembly language with several examples. Discussion of addressing modes.

Lab - Interfacing of EVB with the computer. Simple program to perform basic operations of addition, subtraction etc.

2. Digital Interfacing

Description:

General purpose digital I/O

Relays and switches interfacing

Fundamentals of digital I/O interfacing with DSP.

Lab - Interfacing with DIP switches for digital input and lamps through relays for digital output. Various examples for turning on lamps in particular sequences are implemented. The concept of delay generation is also introduced.

3. Timer operations

Description:

Interrupt systems

Programming the timer functions : compare, input capture, and pulse-width modulator

Application of timer functions : time delay, waveform generation, frequency measurement

Lab : Experiments to introduce the concept of interrupts are setup. Timer, input capture, PWM etc functions are studied by generating various square wave outputs that can be observed on the oscilloscope.

4. Stepper motor drive and shaft encoder

Description:

Stepper motor drive circuits

Encoders operations and interfacing: incremental encoders

Position and speed control of stepper motors

Lab : Experiment will be set up to study the position and speed control of a stepper motor with an encoder.

5. Analog interfacing

Description:

A to D operations and programming

D to A operations and programming

Lab : Experiments are set up to read value from an input source and display it in binary format. Also, the operation of DAC is studied by generating voltages of various levels that can be observed on the oscilloscope.

6. DC Motor speed control

Description:

Basedrive and power converter circuits

Open loop and closed loop speed control

Lab - The setup helps to study open loop control of DC brush motor. The power converter module from Semikron is employed in this set up.

7. Serial communication

Description:

Serial communication and peripheral interface operations and programming for asynchronous data transfer.

Lab - Experiment will be set up to carry put communication with the PC RS232 port using the RS232 port on the EVB.

CHAPTER 2

OVERVIEW OF TMS320F240 DSP AND

INTRODUCTION TO ASSEMBLY LANGUAGE PROGRAMMING

2.1 Introduction

The laboratory experiments developed in this book are intended to provide the students with hands-on experiences in the use of DSP for industrial applications, especially in electromechanical systems. The digital signal processor (DSP) employed for this purpose is the Texas Instruments TMS320F240. In this book, the development of the DSP system has been carried out using the TMS320F240 evaluation module. Using this module, a DSP designer or programmer can test, evaluate, and debug the hardware and software being developed with the help of a personal computer (PC).

Through out the book, the software for the DSP will be developed using TMS320Cxx assembly language.

This chapter aims at providing an overview of the TMS320F240 evaluation board and a brief introduction to the assembly language. Simple examples are provided to familiarize the user with the assembly language. As we progress through the various chapters, more details about the assembly language will be discussed.

2.2 TMS320F240 Overview

- This device belongs to the TMS320C2xx generation of 16-bit fixed-point digital signal processors (DSPs). The TMS320F240 is a device optimized for digital motor control and power conversion applications.

Following is a summary of the features of the TMS320F240:

- Central Processing Unit (CPU)
- 32-bit central arithmetic logic unit (CALU)

- 32-bit accumulator
- 16-bit × 16-bit parallel multiplier
- Three scaling shifters
- Eight 16-bit auxiliary registers with a dedicated arithmetic unit for indirect addressing of data memory

- Memory
 - 544 words × 16 bits of on-chip data/program dual-access RAM
 - 16K words × 16bits of on-chip program flash EEPROM
 - 224K × words of program space, 26K words of data space, 64K words of I/O space, and 32K words of global space)
 - External memory Interface Module with software wait-state generator, a 16-bit address bus, and a 16-bit data bus.
 - Support of hardware wait-states

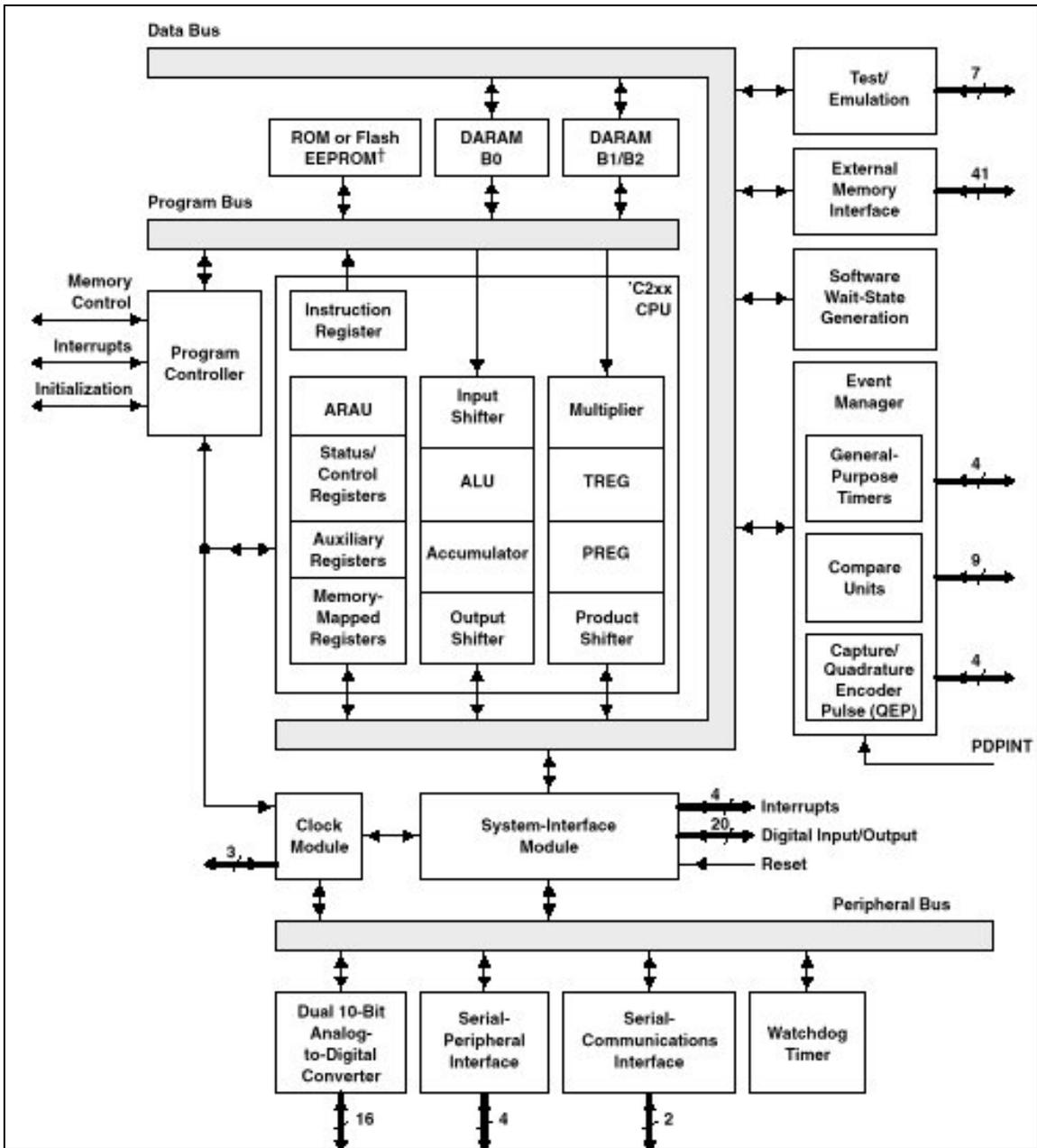
- Program Control
 - Four-level pipeline operation
 - Eight-level hardware stack
 - Six external interrupts: power-drive protection interrupt, reset, NMI, and three maskable interrupts

- Instruction Set
 - Source code compatibility with 'C2x, 'C2xx and 'C5x fixed-point generations of the TMS320 family.
 - Single-instruction repeat operation
 - Single-cycle multiply/accumulate instructions
 - Memory block move instructions for program/data management
 - Indexed-addressing capability
 - Bit-reversed indexed-addressing capability for radix-2 fast Fourier transforms (FFT)

- ❑ Power
 - Static CMOS technology
 - Four power-down modes to reduce power consumption
- ❑ Emulation: IEEE Standard 1149.1 test access port interface to on-chip scan based emulation logic
- ❑ Speed : 50-ns (20 MIPS) instruction cycle time, with most instructions single-cycle
- ❑ Event Manager:
 - 12 compare/pulse-width modulation (PWM) channels (9 independent)
 - Three 16-bit general-purpose timers with six modes, including continuous up counting and continuous up/down counting
 - Three 16-bit full compare units with dead band capability
 - Three 16-bit simple compare units
 - Four capture units, two of which have quadrature encoder-pulse interface capability
- ❑ Dual 10-bit analog-to-digital converter
- ❑ 28 individually programmable, multiplexed I/O pins
- ❑ Phase-locked loop (PLL)-based clock module
- ❑ Watchdog timer module with real-time interrupt
- ❑ Serial communication interface (SCI)
- ❑ Serial peripheral interface (SPI)

2.3 TMS320F240 ARCHITECTURE

The TMS320F240 is based on the modified Harvard architecture, which supports separate bus structures for program space and data space. This multiple bus structure allows simultaneous reading of data and instructions thus enabling the execution of most instructions in a single machine cycle. A third space, the input/output (I/O) space is also available and is accessed through the external bus interface.



TI Doc Ref: SPRS042D-OCTOBER 1996-REVISED NOVEMBER 1998

As shown in the figure above, the TMS320F240 is composed of three main functional units: the **DSP Core / CPU**, the **internal memory** and the **peripherals**. In addition to these basic units, there are a number of system features such as the memory map, device reset, interrupts, digital I/O, clock generation and low-power operation. The following section will discuss each of these in brief.

2.3.1 Internal Bus Structure:

The internal data and program bus structure is made up of 6 16-bit buses:

Program Address Bus (PAB): reads from and writes to program memory

Data Read Address Bus (DRAB): reads from data memory

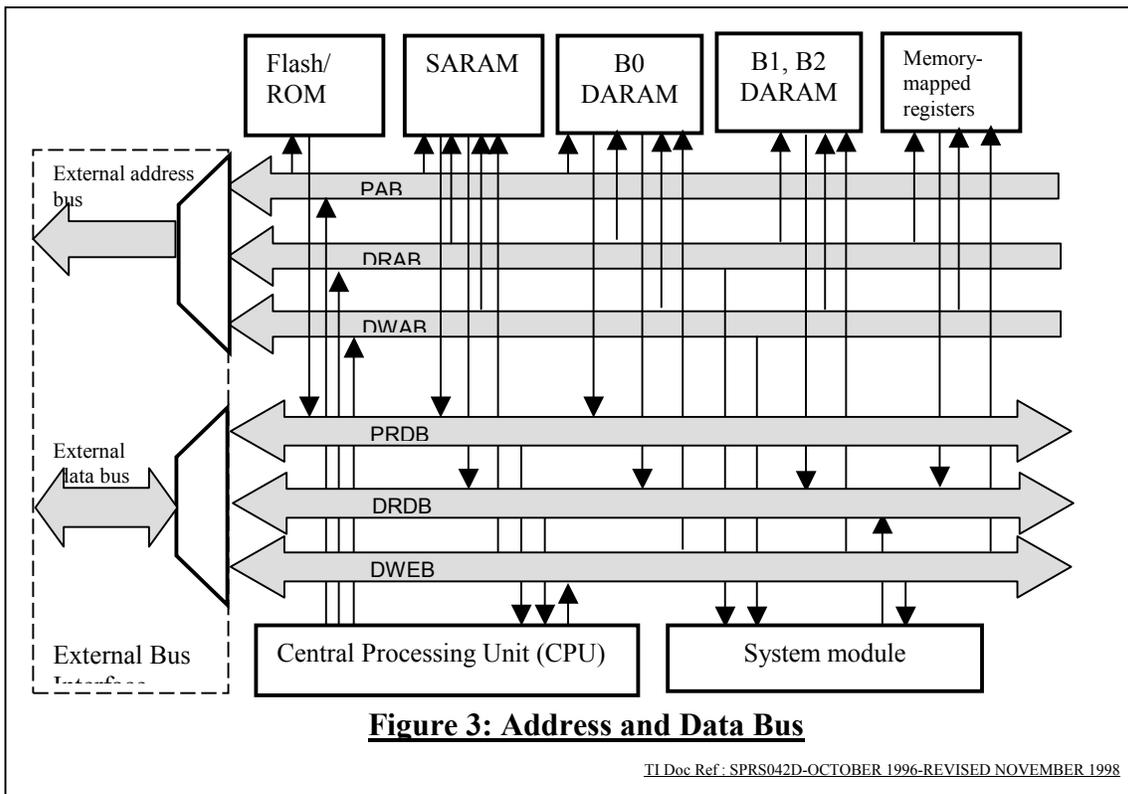
Data Write Address Bus (DWAB): writes to data memory

Program Read Bus (PRDB): carries instruction code and immediate operands, as well as table information, from program memory to the CPU

Data Read Bus (DRDB): carries data from the data memory to the central arithmetic logic unit (CALU) and the auxiliary register arithmetic unit (ARAU)

Data Write Bus (DWEB): carries data to data and program memory

This bus structure can be summarized as shown in Figure 3.



2.3.2 Memory

The TMS320F240 has two kinds of on-chip memory-

Dual-access RAM (DARAM)

The DARAM allows writes to and reads from the RAM in the same cycle. It has a total of 1056 words and is configured as 3 blocks - block B0, block B1 and block B2. Block B0 is a 256-word block that can be configured as data or program memory. Block B1 is 256 words of data memory and block B2 is 32 words of data memory.

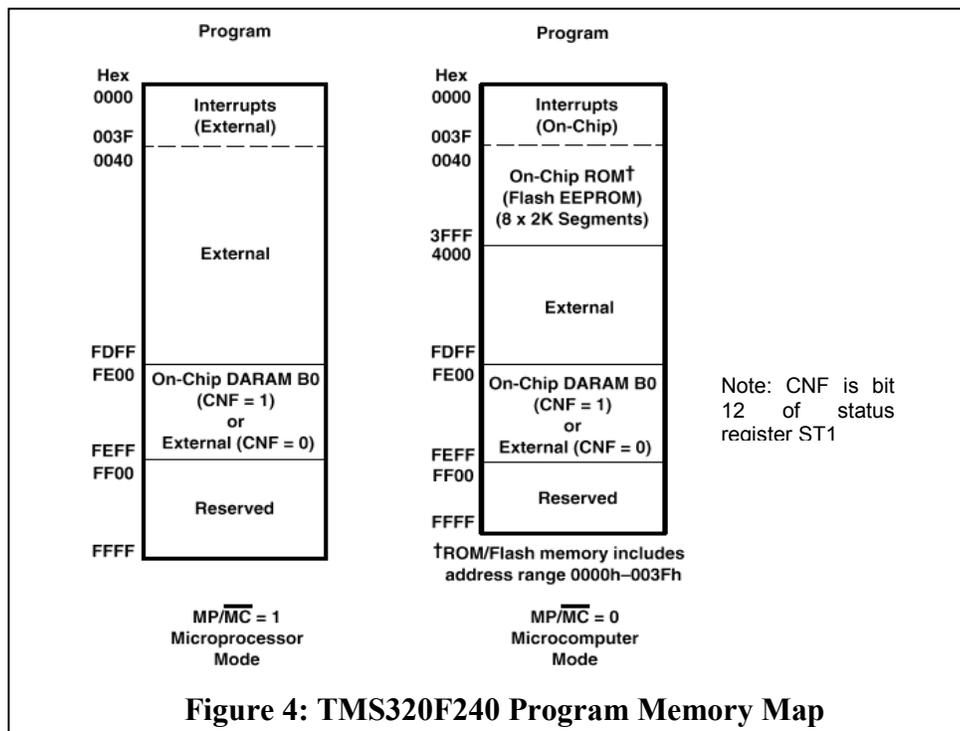
Flash EEPROM

The TMS320F240 includes 16K words of Flash EEPROM. This memory acts as a non-volatile storage for programs. It allows one access per cycle in read mode. In write mode (programming) it requires a 5-V supply that is generated by the chip itself.

The memory of TMS320F240 is organized into 4 selectable spaces:

1. Program Memory

This is the space where the application program code resides. The program space can address up to 64K 16-bit words of external and on-chip memory. The memory map is as shown in Figure 4.



The chip has 2 modes of operation:

Microprocessor Mode (pin MP/MC=1): In this case, when a reset occurs, the device fetches the reset vector from external memory.

Microcomputer Mode (pin MP/MC=0): In this mode, device fetches reset vector from on-chip flash memory.

2. Local Data Memory

This can address up to 64K 16bit words of memory. This includes the on-chip DARAM blocks B1 and B2 and external memory. Block B0 is included if CNF bit is set to 0. The memory map is shown in Figure 5 below.

The 0000h-005Fh space is mapped as follows-

0000h-0003h Reserved

0004h Interrupt Mask Register (IMR) *

0005h Global Memory Allocation Register (GREG) *

0006h Interrupt Flag Register (IFR) *

0023h-0027h Reserved

002Bh-002Fh Reserved for test/emulation systems for special information transfers.

0030h-005Fh Reserved

*These registers will be discussed in detail in later sections.

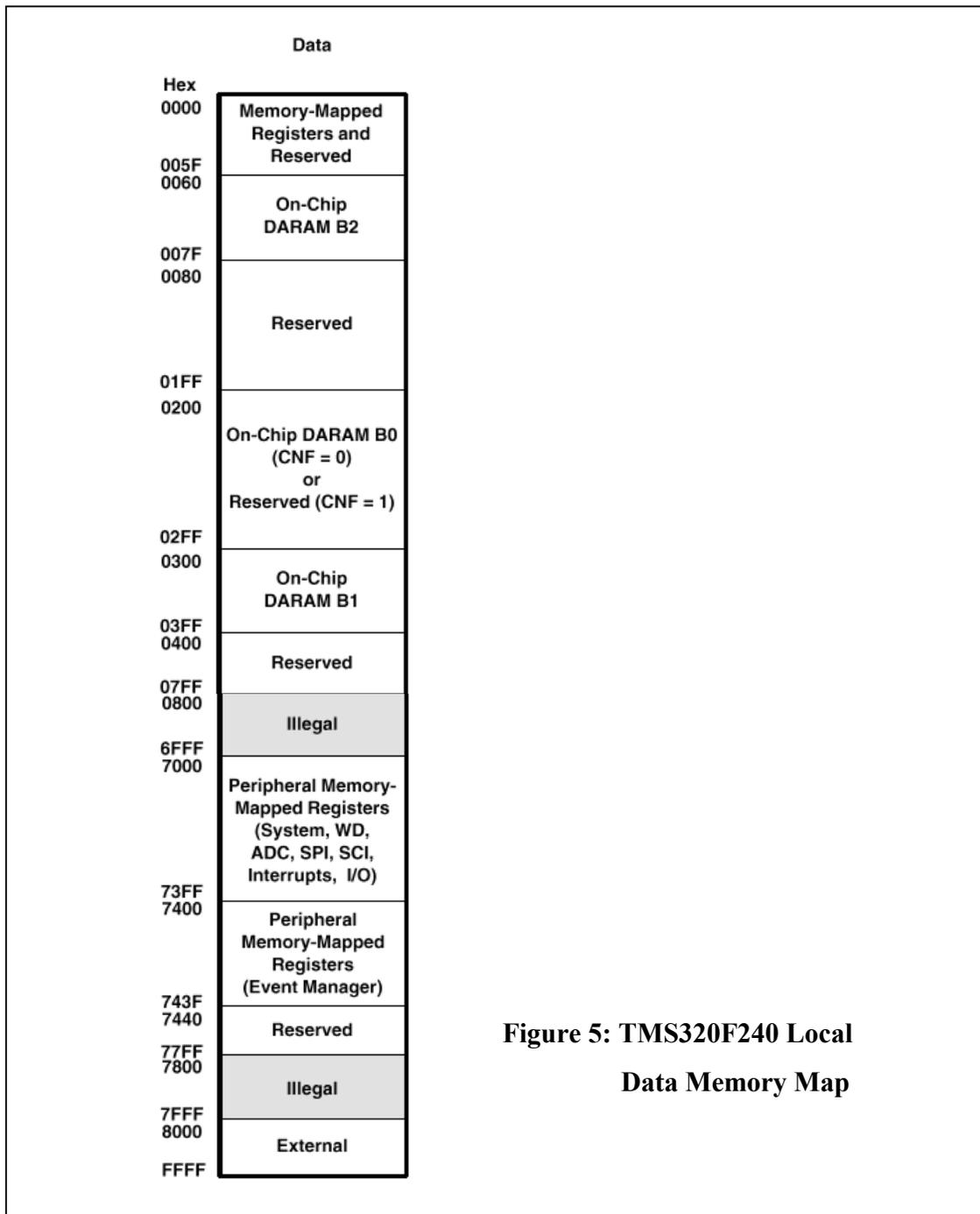


Figure 5: TMS320F240 Local Data Memory Map

3. Global Data Memory

Addresses in the upper 32K words of local data memory i.e. 8000h-FFFFh can be configured as global data memory. This is mainly used for multi-processor systems. The global memory allocation register (GREG) decides the size of the global data memory

that is between 256 to 32K words. The different possible memory configurations with the allowable values of the GREG are listed in the table below:

GREG Value		Local memory		Global memory	
High Byte	Low Byte	Range	Words	Range	Words
XXXX XXXX	0000 0000	0000h-FFFFh	65536	-	0
XXXX XXXX	1000 0000	0000h-7FFFh	32768	8000h-FFFFh	32768
XXXX XXXX	1100 0000	0000h-BFFFh	49152	C000h-FFFFh	16384
XXXX XXXX	1110 0000	0000h-DFFFh	57344	E000h-FFFFh	8192
XXXX XXXX	1111 0000	0000h-EFFFh	61440	F000h-FFFFh	4096
XXXX XXXX	1111 1000	0000h-F7FFh	63488	F800h-FFFFh	2048
XXXX XXXX	1111 1100	0000h-FBFFh	64512	FC00h-FFFFh	1024
XXXX XXXX	1111 1110	0000h-FDFFh	65024	FE00h-FFFFh	512
XXXX XXXX	1111 1111	0000h-FEFFh	65280	FF00h-FFFFh	256

Note : X = Don't care

4. I/O Space

The I/O space memory addresses up to 64K 16-bit words. The memory map is shown Figure 6 below.

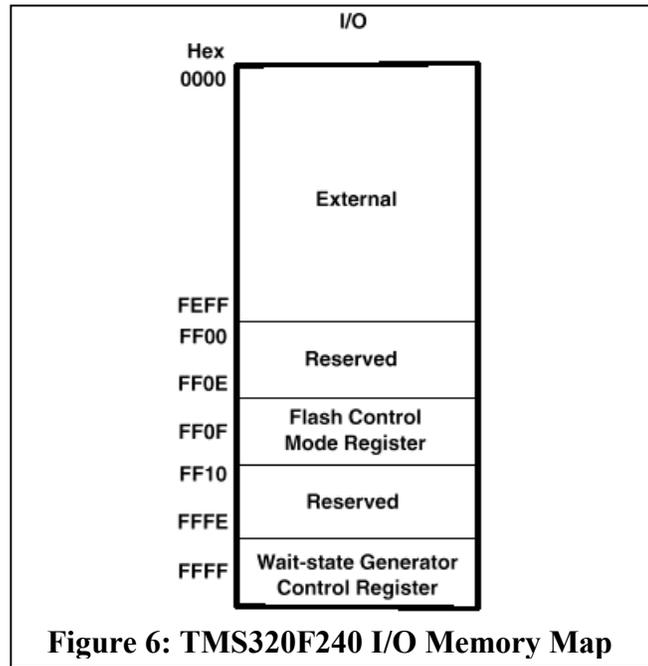


Figure 6: TMS320F240 I/O Memory Map

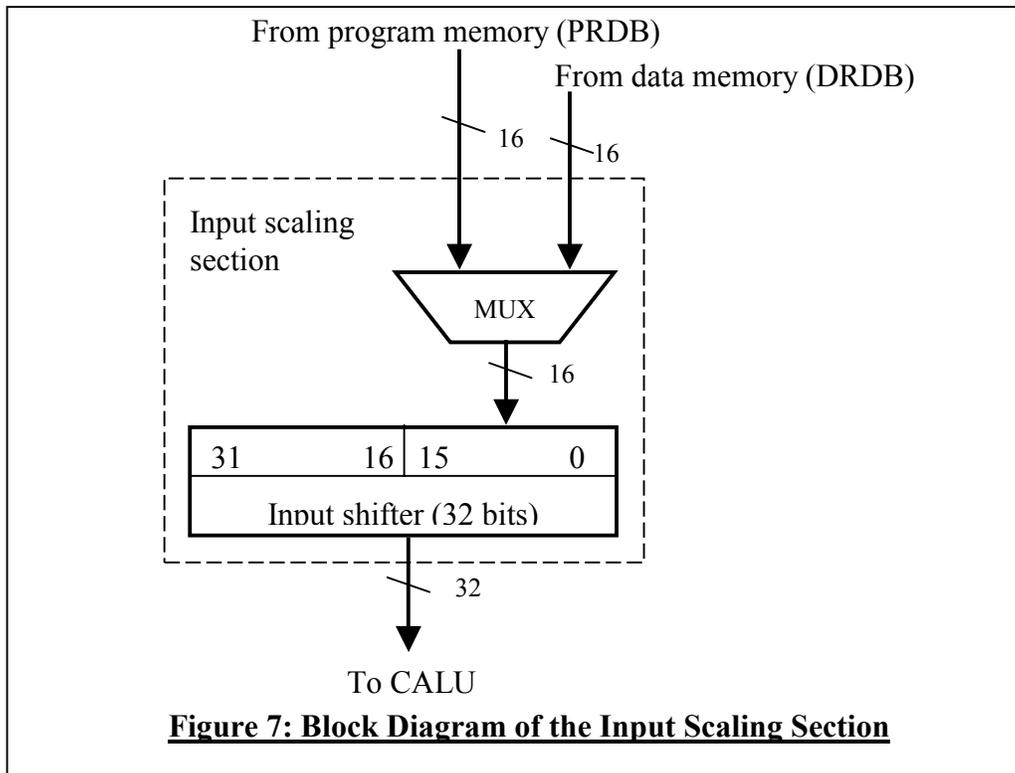
The *flash control mode register* offers two options: register access mode and array access mode. The register access mode gives access to four control registers in the memory space which are used to control erasing, programming and testing of the flash array. The array access mode allows only access to the data; the registers cannot be accessed in this mode.

2.3.3 CPU (Central Processing Unit)

The various components of the CPU as shown in figure 2 are discussed in the following section.

Input Scaling Shifter

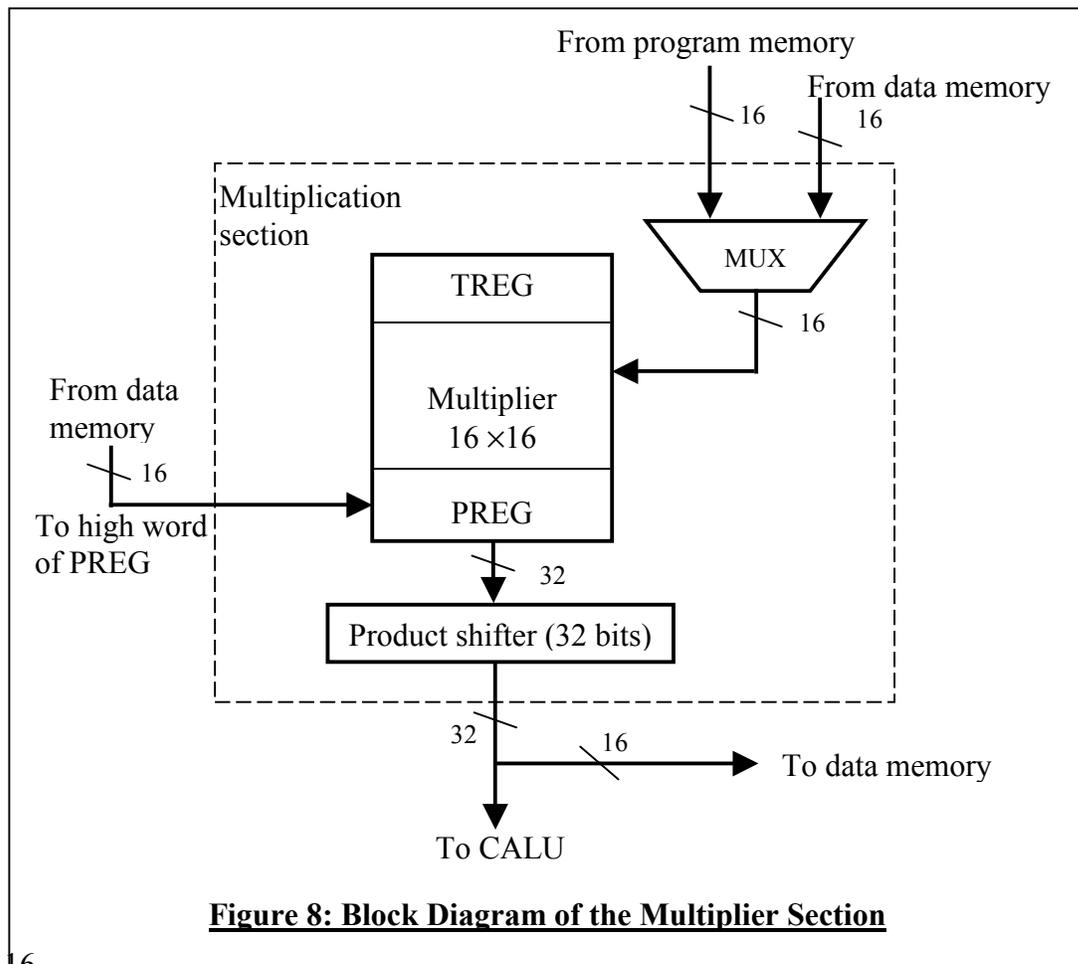
This unit forms the interface between the 16-bit data bus and the 32-bit central arithmetic logical unit (CALU). Its main function is to align the 16-bit data coming from the program and data space to the 32-bit CALU as required by the data scaling arithmetic as well as aligning masks for logical operations. The block diagram shown in figure 7 below briefly explains this scheme.



The shifter basically shifts the value left by 0 to 16 bits. The unused least significant bits (LSBs) are filled with 0s and the unused most significant bits (MSBs) are either filled with 0s or sign extended depending on the value of the sign extension mode (SXM) bit in the ST1 register. The shift count is decided either by a constant embedded in the instruction word or by the four LSBs of the TREG register.

Multiplier

This is a 16×16 -bit hardware multiplier capable of signed and unsigned multiplication in a single machine cycle. The 16-bit multiplicand is picked up from the TREG which, is loaded from the data memory. The multiplier is a 16-bit value which, can be either from the data or program memory. The 32-bit product is stored in the product register PREG. This 32-bit product is then passed through the product shifter, which passes a 32-bit value to the CALU or a 16-bit value to the data memory. The shifter decides the shift count based on the status of the product shift mode PM bits of the ST1 register, which will be discussed later in this section. Figure 8 below summarizes the function of the multiplier.



Central Arithmetic Logic Unit (CALU)

The CALU is referred to as central so as to differentiate it from the auxiliary register arithmetic unit (ARAU). The CALU performs many arithmetic and logic functions most of which it performs in a single clock cycle. These fall under the categories of : 16-bit addition, 16-bit subtraction, Boolean logic operations and Bit testing, shifting, rotating. The 32-bit accumulator always provides one input to the CALU. Either the product-scaling shifter or the input data-scaling shifter provides the second input. For some of the instructions, the sign extension (SXM) bit of the ST1 register determines if the sign extension is employed in the CALU calculations. The CALU transfers its output to the 32-bit accumulator.

Accumulator

The 32-bit accumulator accepts input from the output of the CALU and is capable of performing bit shifts or rotations on its contents. The 32-bit contents can be split into 2 16-bit segments for storage in the data memory. The contents of the accumulator are passed through the output data-scaling shifter before they are stored into the data memory. The carry bit (C) of ST1, the overflow mode bit (OVM) of ST0, the overflow flag bit (OV) of ST0 and the test/control flag bit (TC) of ST1 are associated with the accumulator. These are discussed in detail in a later section.

Output Data-Scaling Shifter

The output data-scaling shifter accepts a 32-bit input from the accumulator, performs a 0 to 7-bit shift on it and provides a 16-bit output to the data bus. The shift count is based on the value specified in the corresponding instruction. The MSBs are lost during the shift and the LSBs are loaded with 0s. It must be noted here that the contents of the accumulator remain unchanged during this process.

Auxiliary Register Arithmetic Unit (ARAU)

The main function of the ARAU is to perform arithmetic operations on the 8 auxiliary registers (AR0-7) in parallel with the operations occurring in the CALU. The auxiliary

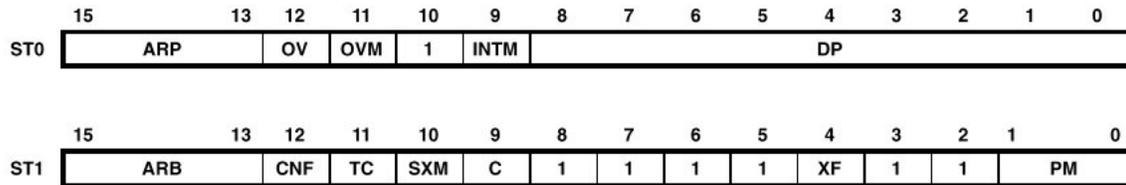
registers are primarily employed for indirect addressing which is discussed in a later section. The other functions include:

- For support of conditional branches, calls and returns.
- For temporary storage of data
- As software counters by incrementing and decrementing the registers as required.

A specific auxiliary register can be selected by specifying the 3-bit value of the auxiliary register pointer (ARP) in the status register ST0. Figure 9 shows the ARAU and related logic.

Status Registers

The device has 2 status/control register ST0 and ST1 which can be loaded from and stored the data memory; thus making the machine status accessible to subroutines. The various bits of these registers are discussed below.



FIELD	FUNCTION
ARB	Auxiliary register pointer buffer. When the ARP is loaded into ST0, the old ARP value is copied to the ARB except during an LST instruction. When the ARB is loaded by way of an LST #1 instruction, the same value is also copied to the ARP.
ARP	Auxiliary register (AR) pointer. ARP selects the AR to be used in indirect addressing. When the ARP is loaded, the old ARP value is copied to the ARB register. ARP can be modified by memory-reference instructions when using indirect addressing, and by the LARP, MAR, and LST instructions. The ARP is also loaded with the same value as ARB when an LST #1 instruction is executed.
C	Carry bit. C is set to 1 if the result of an addition generates a carry, or reset to 0 if the result of a subtraction generates a borrow. Otherwise, C is reset after an addition or set after a subtraction, except if the instruction is ADD or SUB with a 16-bit shift. In these cases, the ADD can only set and the SUB only reset the carry bit, but cannot affect it otherwise. The single bit shift and rotate instructions also affect C, as well as the SETC, CLRC, and LST #1 instructions. Branch instructions have been provided to branch on the status of C. C is set to 1 on a reset.
CNF	On-chip RAM configuration control bit. If CNF is set to 0, the reconfigurable data dual-access RAM blocks are mapped to data space; otherwise, they are mapped to program space. The CNF can be modified by the SETC CNF, CLRC CNF, and LST #1 instructions. \overline{RS} sets the CNF to 0.
DP	Data memory page pointer. The 9-bit DP register is concatenated with the seven LSBs of an instruction word to form a direct memory address of 16 bits. DP can be modified by the LST and LDP instructions.
INTM	Interrupt mode bit. When INTM is set to 0, all unmasked interrupts are enabled. When set to 1, all maskable interrupts are disabled. INTM is set and reset by the SETC INTM and CLRC INTM instructions. \overline{RS} and \overline{IACK} also set INTM. INTM has no effect on the unmaskable \overline{RS} and NMI interrupts. Note that INTM is unaffected by the LST instruction. This bit is set to 1 by reset. It is also set to 1 when a maskable interrupt trap is taken.
OV	Overflow flag bit. As a latched overflow signal, OV is set to 1 when overflow occurs in the arithmetic logic unit (ALU). Once an overflow occurs, the OV remains set until a reset, BCND/D on OV/NOV, or LST instructions clear OV.
OVM	Overflow mode bit. When OVM is set to 0, overflowed results overflow normally in the accumulator. When set to 1, the accumulator is set to either its most positive or negative value upon encountering an overflow. The SETC and CLRC instructions set and reset this bit, respectively. LST can also be used to modify the OVM.
PM	Product shift mode. If these two bits are 00, the multiplier's 32-bit product is loaded into the ALU with no shift. If PM = 01, the PREG output is left-shifted one place and loaded into the ALU, with the LSB zero-filled. If PM = 10, PREG output is left-shifted by four bits and loaded into the ALU, with the LSBs zero-filled. PM = 11 produces a right shift of six bits, sign-extended. Note that the PREG contents remain unchanged. The shift takes place when transferring the contents of the PREG to the ALU. PM is loaded by the SPM and LST #1 instructions. PM is cleared by \overline{RS} .
SXM	Sign-extension mode bit. SXM = 1 produces sign extension on data as it is passed into the accumulator through the scaling shifter. SXM = 0 suppresses sign extension. SXM does not affect the definitions of certain instructions; for example, the ADDS instruction suppresses sign extension regardless of SXM. SXM is set by the SETC SXM and reset by the CLRC SXM instructions, and can be loaded by the LST #1. SXM is set to 1 by reset.
TC	Test/control flag bit. TC is affected by the BIT, BITT, CMPR, LST #1, and NORM instructions. TC is set to a 1 if a bit tested by BIT or BITT is a 1, if a compare condition tested by CMPR exists between AR (ARP) and AR0, if the exclusive-OR function of the two MSBs of the accumulator is true when tested by a NORM instruction. The conditional branch, call, and return instructions can execute based on the condition of TC.
XF	XF pin status bit. XF indicates the state of the XF pin, a general-purpose output pin. XF is set by the SETC XF and reset by the CLRC XF instructions. XF is set to 1 by reset.

2.3.4 Peripherals

The TMS320F240 has a number of on-chip peripherals. Some of the peripherals are accessed through the data bus while the other are accessed via the peripheral bus which is mapped to the data bus through the system interface module. The different peripherals are discussed in this section.

External Memory Interface

The TMS320F240 can access 64K words of memory for I/O, program and data space, which includes on-chip, as well as external memory. The CPU schedules a program fetch, data read and data write on the same machine cycle; since this is possible in case of on-chip memory. However, for external memory access, the external memory interface multiplexes the internal to one address bus and one data bus and sequences these operations to complete the data write first, then the data read and finally the program read.

The interface provides a number of control signals such as R/W output signal to indicate whether the current cycle is read or write, the STRB signal to provide a timing reference for all external cycles. Interface with devices of varying speeds the READY input is employed. The bus request (BR) signal in conjunction with other signals is used to arbitrate external global memory accesses. These will be dealt with in greater detail in the chapter on peripherals.

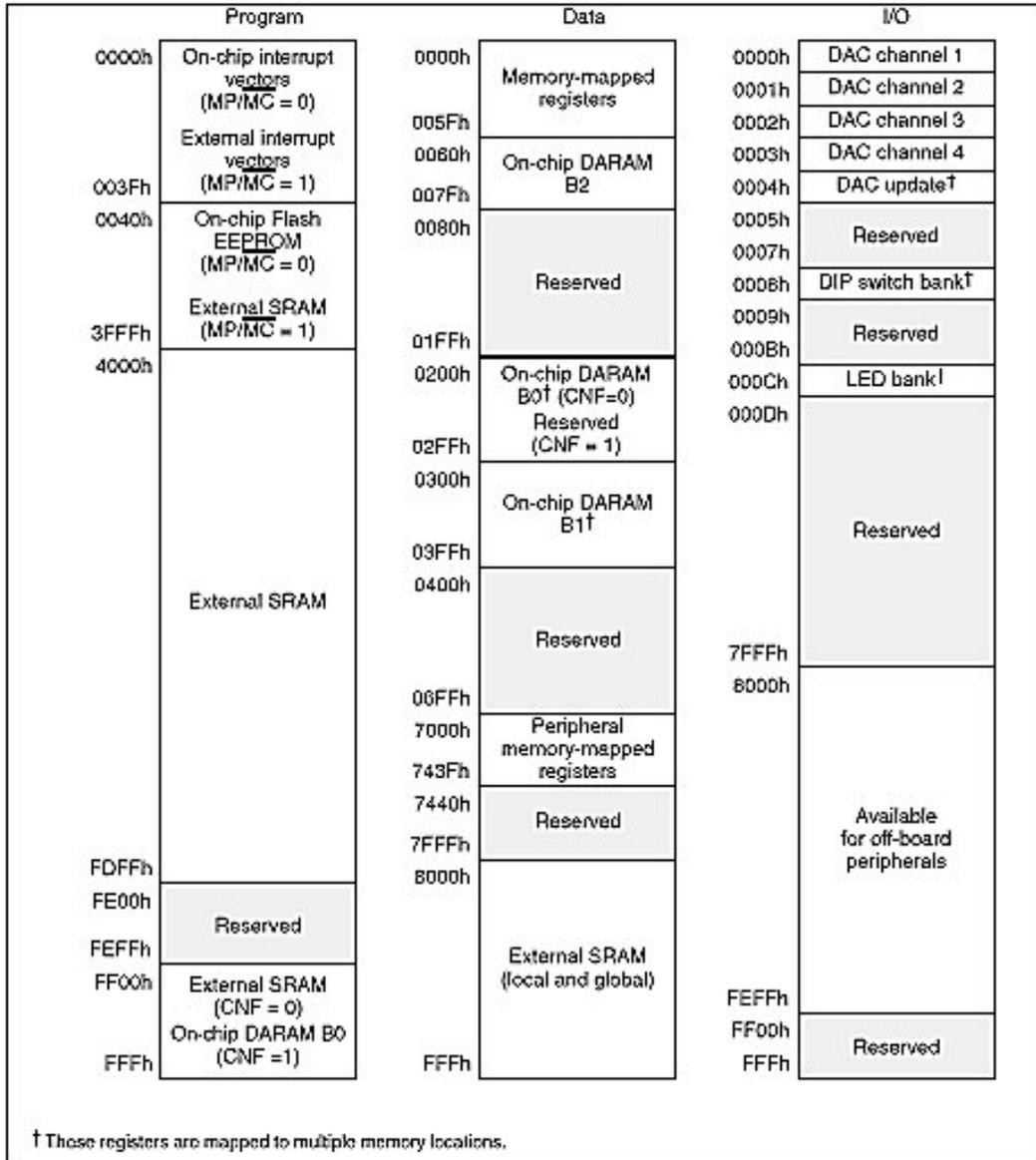


Figure 11: TMS320CF240 EVM Memory Map

The EVB has an on-board RS-232 compatible serial port for asynchronous communication and can be configured for various communication protocols.

The four 34-pin connectors give access to all the relevant signals on the evaluation board. The I/O connector P1, gives access to all event manager (EV), serial peripheral interface (SPI) and serial communication interface (SCI) signals. All analog signals including the four DAC output channels, 16 ADC input channels and the ADC reference voltages are brought out to the analog connector (P2). The address/data connector gives access to the

external address and data bus signals. All the external memory interface signals can be found on the control connector (P4). The evaluation port (P5), connects the EVB to the XDS510PP emulator which acts as the main interface between the debugger and the evaluation board. The detailed pin diagrams of all the connectors are shown in figure below:

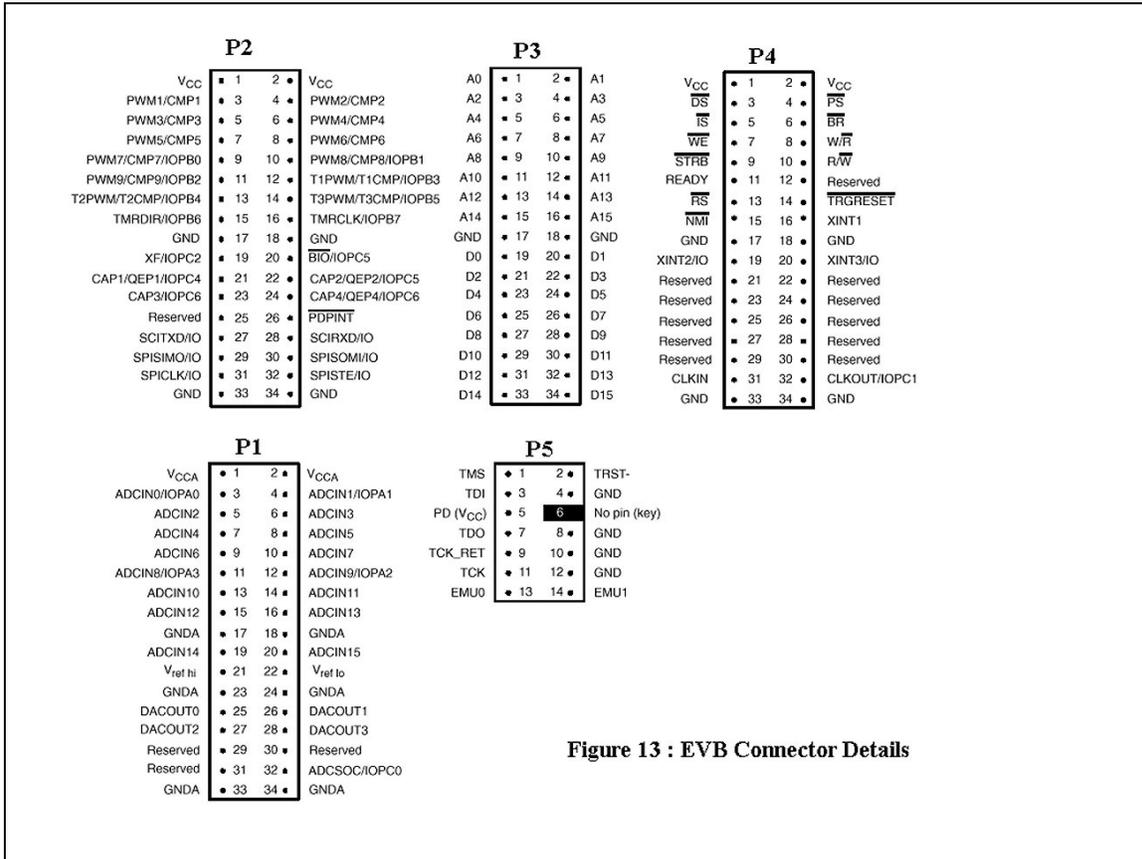


Figure 13 : EVB Connector Details

Programming the TMS320F240

Before programming a device, it is important to understand the basic program control for the device.

Program Address Generation

The execution of any instruction can be said to comprise of four independent stages - instruction-fetch, instruction-decode, opcode-fetch and instruction-execute. During any given cycle, one to four different instructions can be active, each at a different stage of completion given the 4-level deep pipeline of the TMS320F240. Program flow requires

that the processor generate the address of the next instruction while executing the current instruction. The 16-bit program counter holds the address of the next instruction. The program address generation is explained in the figure 13 below.

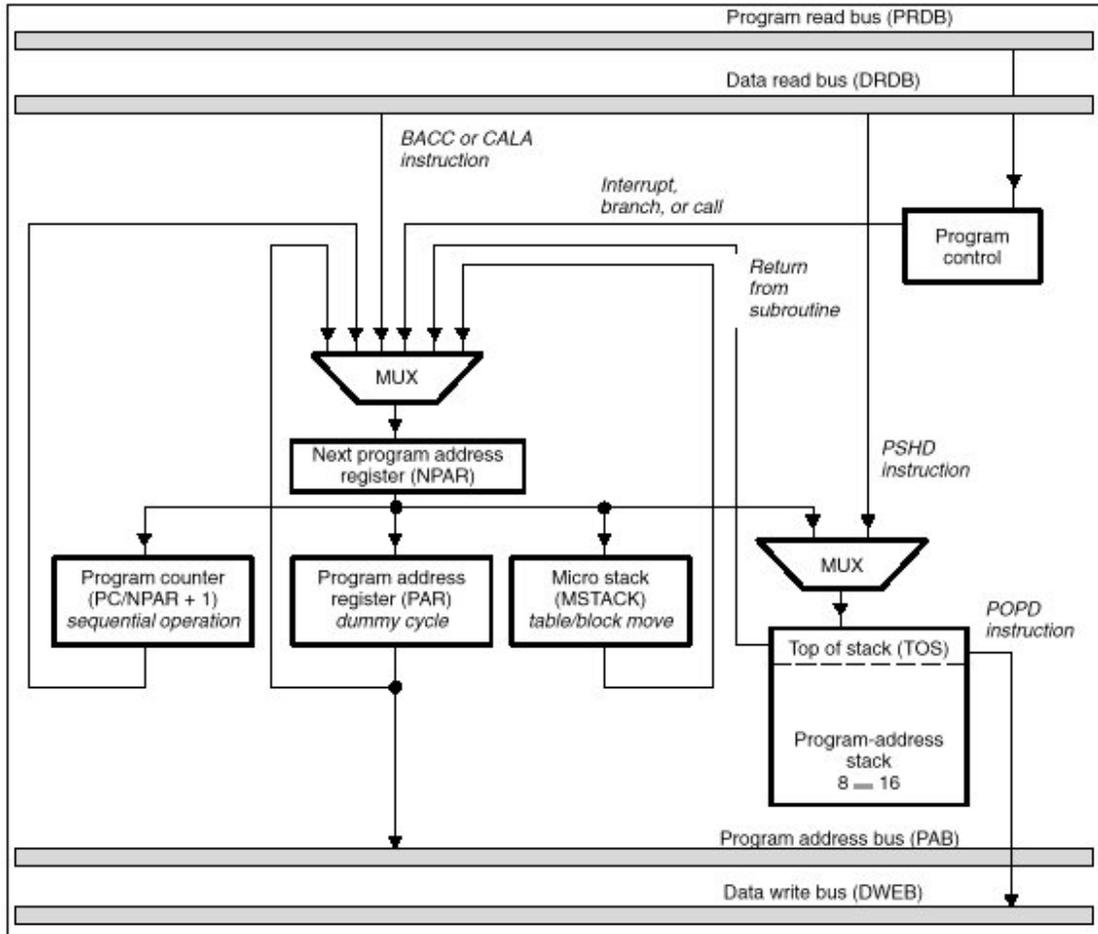


Figure 13: Program Address Generation Block Diagram

The program address generation logic uses the following hardware:

Program Counter (PC) - Holds the address of the next instruction to be executed.

Program Address Register (PAR) - Drives the program address bus (PAB). The PAB is a 16-bit bus that provides program addresses for both reads and writes.

Stack- The program generation logic includes a 16-bit wide, 8-level hardware stack for storing up to eight return addresses when a subroutine call or interrupt occurs. The stack

can also be used for temporary storage. And for saving context data during a subroutine or interrupt service routine.

Microstack (MSTACK) - For certain instructions, the program-address generation logic uses the 16-bit wide, 1-level MSTACK to store one return address. These instructions use the program-address generation logic to provide a second address in a 2-operand instruction.

Repeat Counter (RPTC) - The 16-bit RTPC is used with the repeat (RPT) instruction to determine how many times the instruction following RPT is repeated.

Program Address Generation Summary -

Operation	Program-Address Source
Sequential Operation	PC (contains program address +1)
Dummy Cycle	PAR (contains program address)
Return from subroutine	Top of stack (TOS)
Return from table move or block move	Microstack (MSTACK)
Branch or call to address specified in lower half of the accumulator	Branch or call instruction by way of the program read bus (PRDB)
Branch or call to address specified in lower half of the accumulator	Low accumulator by way of the data read bus (DRDB)
Branch to interrupt service routine	Interrupt vector location by way of the program read bus (PRDB)

Addressing Modes

The 3 addressing modes used by the TMS320F240 instruction set are -

- Immediate Addressing Mode
- Direct Addressing Mode
- Indirect Addressing Mode

Immediate Addressing Mode

In this mode, the instruction word contains a constant to be manipulated by the instruction. There are 2 types of immediate addressing mode:

Short immediate addressing - Instructions that use this mode take an 8-bit, 9-bit or 13-bit constant as an operand. These instructions require a single word with the constant embedded in that word.

Example 2.1:

```
LACC #99          ; Load the number 99 into the accumulator
```

Long immediate addressing - Instructions that use long-immediate addressing take a 16-bit constant as an operand and require two instruction words. The constant is sent in the second instruction word. This 16-bit value can be used as an absolute constant or as a 2's complement value.

Example 2.2:

```
ADD #16384,2      ; Shift the value 16384 left by two bits  
                  ; and add the result to the accumulator
```

Direct Addressing Mode

In this mode, the data memory is addressed in blocks of 128 words called data pages. Thus the entire 64K of data memory can be addressed by 512 pages, which are labeled from 0 to 511 (000000000b to 111111111b). The value in the 9-bit data page pointer (DP) in the status register ST0 determines the current data page. The particular being referenced within a page is determined by a 7-bit offset, which is specified by the seven LSBs of the instruction register.

When using the direct addressing mode, the steps to be followed are-

1. *Set the data page* - Load the appropriate value between 0-511 in the DP register using the LDP instruction. For example, to set the current data page to 2 i.e. addresses 0100h-017Fh the following instruction should be used -

```
LDP    #2    ; Initialize data page pointer
```

2. *Specify the Offset* - Supply the 7-bit offset as an operand of the instruction. For example, if you want to use the ADD instruction for the second value in the current page, the command is

```
ADD    1h    ; Add to accumulator the value in the current
           ; data page, offset of 1
```

Indirect Addressing Mode

As mentioned earlier, the eight auxiliary registers (AR0-AR7) are employed for indirect addressing. The address of the operand is contained in the currently selected auxiliary register. A specific auxiliary register is selected by loading a 3-bit value in the auxiliary register pointer (ARP) of the status register ST0. The register pointed to by the ARP is referred to as the *current auxiliary register* or the *current AR*. The data address (i.e. contents of AR) is passed either to the data-read bus or data-write bus by the ARAU depending on the instruction. The ARAU performs arithmetic operation on the contents of the AR during the decode phase depending upon the mode of addressing used in the instruction.

There are seven indirect addressing modes:

Operand	Option	Example
*	No increment or decrement	LT* loads the temporary register (TREG) with the contents of the data memory address referenced by the current AR.
+	Increment by 1 (Auto-increment)	LT+ loads the temporary register (TREG) with the contents of the data memory address referenced by the

		current AR and then adds 1 to the contents of the current AR.
-	Decrement by 1 (Auto-decrement)	LT- loads the temporary register (TREG) with the contents of the data memory address referenced by the current AR and then subtracts 1 to the contents of the current AR.
*0+	Increment by index amount (Post-indexing by adding contents of AR0)	LT*0+ loads the temporary register (TREG) with the contents of the data memory address referenced by the current AR and then adds the contents of AR0 to the contents of the current AR
*0-	Decrement by index amount (Post-indexing by subtracting contents of AR0)	LT*0- loads the temporary register (TREG) with the contents of the data memory address referenced by the current AR and then subtracts the contents of AR0 from the contents of the current AR
* BRO+	Increment by index amount, adding with reverse carry (used in FFTs)	LT *BRO+ loads the temporary register (TREG) with the contents of the data memory address referenced by the current AR and then

		adds the content of AR0 to the content of the current AR, adding with reverse carry propagation
*BRO-	Decrement by index amount, subtracting with reverse carry (used in FFTs)	LT *BRO- loads the temporary register (TREG) with the contents of the data memory address referenced by the current AR and then subtracts the content of AR0 to the content of the current AR, subtracting with reverse carry propagation

Many instructions also specify a value of next AR, in addition to the current AR. This is current AR after the instruction is complete. Then the APR is loaded with the value of next AR, the previous value is loaded into the auxiliary register pointer buffer (ARB).

Modifying the Auxiliary Register Content

The LAR, ADRK, SBRK and MAR are specialized instructions for changing the contents of an auxiliary register.

- ❑ The LAR instruction loads an AR.
- ❑ The ADRK instruction adds an immediate value to an AR; SBRK subtracts an immediate value.
- ❑ The MAR instruction can increment or decrement an AR value by 1 or by an index amount.

Assembly Language Instructions

Before we start with the instruction set, here are a few tips on how to use the instruction descriptions.

Syntax

The notations used in the syntax expressions are -

italic Italic symbols in an instruction syntax represent variables.

symbols Example : For the syntax

ADD *dma*

Any value can be used for *dma* such as

ADD DAT, ADD 21 etc.

boldface Boldface characters in an instruction syntax must be typed as shown

characters Example : For the syntax

ADD *dma*, **16**

A variety of values may be used for *dma*, but **ADD** and **16** must be typed shown. ADD 7h, 16 or ADD X, 16

[,x] Operand x is optional

Example : For the syntax

ADD *dma*, [,*shift*]

dma must be supplied as in the instruction:

ADD 7h

There is an option to provide a shift value as in the instruction:

ADD 7h, 5

[,x1 [,x2]] Operands x1 and x2 are optional. However, x2 cannot be included without including x1.

Example : For the syntax

ADD *ind*, [,*shift* [,ARN]]

ind has to be supplied as in the instruction

```
ADD *+
```

Including of *shift* in the instruction is optional.

```
ADD *+, 5
```

Once the *shift* is included, you have an option of including **ARn** too

```
ADD *+, 5 ,AR1
```

The # is a prefix for constants used in immediate addressing.

Example :

```
RPT #15 causes the next instruction to be repeated 16 times
```

RPT 15 causes the next instruction to be repeated a number of times determined by the value in that memory location.

The instruction set summary is attached. As we progress through the various chapters, the relevant instructions will be discussed in detail.

The assembly language source files are translated into machine language COFF (common object file format) files. Apart from the various instructions discussed above, the source files also contain assembler directives, which control various aspects of the assembly process such as source listing format, data alignment listing and section content. A source statement can contain four ordered fields and has the general syntax as follows ;

```
[label] [:] mnemonic [operand list] [;comment]
```

Example:

```
SYM1 .set 2 ;Set SYM1 = 2  
Start: LDPK SYM1 ;Load DP with 2
```

Label Field : A label can contain up to 32 alpha-numeric characters, should not begin with a number and is case-sensitive. The value of a label is the current value of the section program counter. The section program counter (SPC) represents the current address within a section of code or data. Thus in the example, `Start` points to the instruction `LDPK SYM1`.

Mnemonic Field: The mnemonic field can contain machine instructions (LDPK in example above) or assembler directives (.set in example above). This field should not start in column 1 or it will be interpreted as a label.

Operand Field: This is the list of operands that follow the mnemonic field. An operand can be a constant, a symbol or a combination of the two depending on the mnemonic preceding it.

Comment Field: A comment can begin in any column and extends to the end of the source line. A comment can contain any ASCII character. Comments are printed in the assembly source listing, but do not affect the assembly.

Assembler Directives

A summary of the various assembler directives is attached. The most commonly used directives will be discussed in this chapter.

Directives that define sections

The smallest unit of an object file is called a section. A section is a block of data or code that occupies a contiguous space in the memory map. COFF files have three default sections:

.text section usually contains executable code

.data section usually contains initialized data

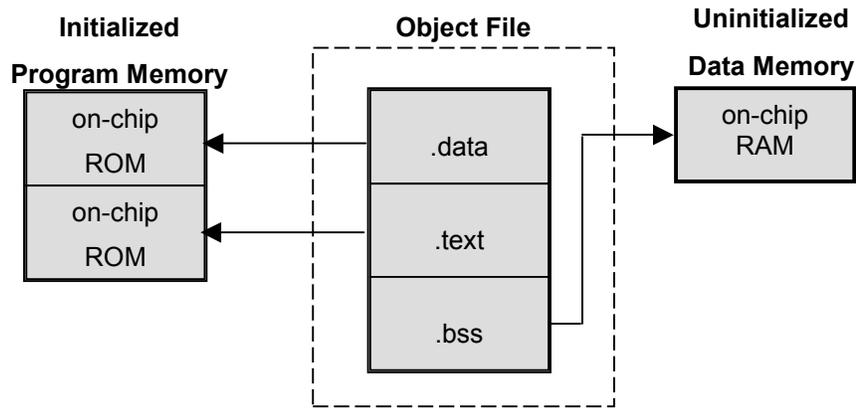
.bss section usually reserves space for uninitialized variables

There are 2 basic types of sections-

Initialized sections contain data or code. .text and .data sections and named section created with the .sect directive lie in this category.

Uninitialized sections reserve space in the memory map for uninitialized data. .bss sections and named sections created with the .usect directive lie in this category

Since all sections are independently relocatable, they enable efficient use of the target memory since any section can be placed in any allocated block of target memory. This partitioning of memory into logical blocks is illustrated in the figure below.



The assembler has 6 directives for handling sections-

- ❑ .bss
- ❑ .usect
- ❑ .text
- ❑ .data
- ❑ .sect
- ❑ .asect

The .bss and .usect create uninitialized sections while the others create initialized sections. If no directive is used, the assembler assembles everything into the .text directive.

Uninitialized sections-

These reserve space in the RAM, which the program can use at runtime for creating and storing variables. The syntax for the relevant directives is:

```
.bss symbol, size in words [blocking flag]
symbol .usect "section name", size in words, [blocking flag]
```

symbol This corresponds to the name of the variable for which the space is being reserved. It can be referenced by any other section and can also be declared as global.

size in words It is an absolute value which determines the words to be reserved in the section.

blocking flag This is an optional parameter. If a value greater than 0 is specified, the assembler associates size words contiguously; the allocated space will not cross a page boundary, unless size is greater than a page in which case the object will start on a page boundary.

section name This is a 8 character name that tells the assembler what named section to reserve space in. A named section is created by the user and can be used like the default .text, .data and .bss sections except that they are assembled separately.

Initialized sections-

These contain executable code or initialized data. The contents of these sections are stored in the object file and placed in the device memory where the program is stored. The syntax for the relevant directives is:

```
.text  
.data  
.sect "section name"  
.asect "section name", address
```

When the assembler encounters any of these directives, it stops assembling in the current section and assembles the subsequent code into the designated section until it again encounters any of the above 4 directives. It is important to note here that the .bss and the .usect directives do not end the current section or begin a new one. They merely reserve

the specified amount of space and the assembler resumes assembly of code or data in the current section.

Example:

```
.text
.word 1,2          ; Initialize words with values 1
                  ; and 2 in the .text section

.sect "sect1"
.word 3,4          ; Initialize words with values 3
                  ; 4 in the named section sect1.

.data
.word 5,6          ; Initialize words with values 1
                  ; and 2 in the .data section

.bss    sym,20     ; Reserve 20 words in .bss
.word 7,8          ; Initialize words with values 7
                  ; and 8 in the .data section

.text          ; Resume assembly in .text sect
usym .usect "sect2", 25 ; Reserve 20 words in named
                  ; section sect2
.word 9,10       ; Initialize words with values 9
                  ; and 10 in the .text section
```

Directives that initialize constants

The various directives that assemble values for the current section are as follows:

.word places one or more consecutive 16-bit values into words of the current section

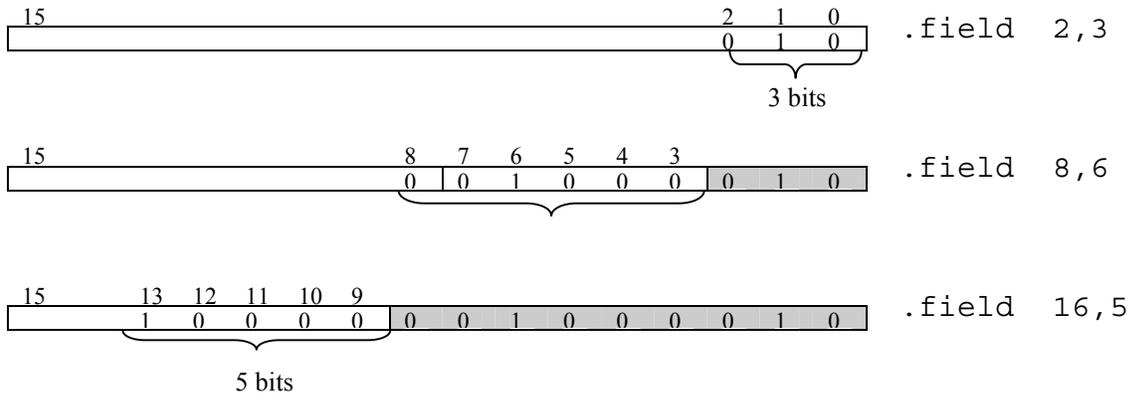
.int same as **.word**

.byte places one or more consecutive 8-bit values into words of the current section

.string similar to **.byte**, except that two characters are packed into each word.

.field places a specified value into a specified number of bits in the current word.
 The assembler does not increment the SPC until the entire word is filled.

Example:

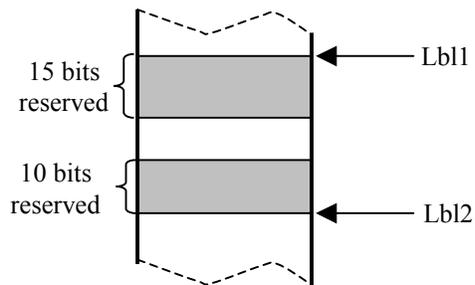


.space reserves a specified no. of bits in the current section(i.e. fills them with 0s)
 When a label is used with this directive, it points to the first word that contains the reserved bits.

.bes reserves a specified no. of bits in the current section(i.e. fills them with 0s)
 When a label is used with this directive, it points to the last word that contains the reserved bits.

Example:

```
Lbl1: .space 15
      .word 20
Lbl2: .bes      10
      .word 30
```



.float calculates the single-precision 32-bit iee floating-point representation of a single floating-point value and stores it in two consecutive words in the current section

- .bfloat** same as `.float` except that it guarantees the object will not span a page boundary.
- .long** places 32-bit values into consecutive two-word blocks in the current section current.
- .blong** same as `.long` except that it guarantees that the object will not span the page boundary

Directives that align the section program counter

- .align** Aligns the SPC at a 128-word boundary thus ensuring that the code following this directive begins on a new page boundary.
- .even** Aligns the SPC so that it points to the next full word. This can be used after a `.field` directive. If the `.field` directive does not fill the word, the `.even` directive fills the unused bits with 0s.

Conditional Assembly Directives

The **.if/.elseif/.else/.endif** directives tell the assembler to conditionally assemble a block of code.

- .if *expression*** marks the beginning of a conditional block and assembles code if the `.if` condition is true
- .elseif *expression*** marks a block of code to be assembled if the `.if` condition is false and the `.elseif` condition is true
- .else** marks the block of code to be executed to be assembled if the `.if` condition is false
- .endif** marks the end of a conditional block and terminates the block

Example:

```
var1 .set 2
```

```

var2 .set 3
var3 .set 6
lbl_if : .if var3 = var1 * var2 ;Set the value equal
                                         ;to var1*var2

        .byte var3
        .else
        .byte var1*var2
        .endif

```

The **.loop/.break/.endloop** directives make the assembler repeatedly assemble a block of code according to the evaluation of a particular expression.

.loop *expression* marks the beginning of a repeatable block of code

.break *expression* tells the assembler to repeatedly assemble the block of code if the *expression* is false and to jump to the code immediately after the **.endloop** in case the *expression* is false.

.endloop marks the end of a repeatable block.

Example:

```

        .eval 0,x ; Set x=0. Initialize count
loop1: .loop
        .word x*100 ; store x*100 at the current
                    ; location
        .eval x+1,x ; Increment x i.e count
        .break x=6 ; If x=6 quit else goto loop1
        .endloop ; the word has a value 500
                    ;when program quits the loop

```

Miscellaneous Directives

.asg assigns a character string to a substitution symbol. The value is stored in the substitution symbol table so that whenever the assembler encounters this symbol, it substitute it with the character string. Substitution symbols can be redefined

Example:

```
.asg "1, 2, 3, 4, 5", char_sym
```

.set sets a constant value to a symbol and cannot be redefined.

Example:

```
bval .set 0020h
```

.equ same as .set

.global a symbol defined as global in a current module allows it to be accessed from an external module. If the symbol is not defined in an external module, then the current module can access it by defining it as global.

.end it is optional and terminates assembly. It should be the last source statement of a program.

REVIEW QUIZ

1. How is the on-chip memory of the TMS320F240 distributed?
2. What are the main components of the CPU?
3. What are the different addressing modes?
4. Explain how the address is generated in case of direct addressing.
5. What are the various modes in indirect addressing?
6. In each of the following instructions, identify the addressing mode and explain what each instruction will do.
 - a. MAR *,AR1
LT*+, AR2
 - b. LDP #500
ADDC 6h
 - c. ADD *-, 8
 - d. ADD *BRO-, 8
 - e. SUB #7226, 5

7. Explain the operation of the following segment of code.

```
var1 .set 1
var2 .set 3
    .asg "var1>var2", msg1
    .asg "var1<var2", msg2
    .asg "var1=var2", msg3
if_1: .if var1 > var2
    .byte msg1
    .elseif var1 < var2
    .byte msg2
    .else
    .byte msg3
    .endif
```

8. The contents of the GREG are 1111 0000 1111 0000. What are the local and global memory ranges?

Answer Key -

1. DARAM has a total of 1056 words and is configured as 3 blocks - block B0 - 256 words can be configured as data/program, block B1 - 256 words can be configured as data and block B2 - 32 words can be configured as data. 16K words of Flash EEPROM can be configured as program.
2. Instruction Register, CALU, ARAU, Input Shifter, Output Shifter, PREG, TREG, Multiplier, ST0, ST1, Accumulator.
3. a. Immediate Addressing (Short and Long)
b. Direct Addressing
c. Indirect Addressing
4. The 9 MSBs are taken from the value of the 9-bit Data Pointer in ST0. The 7-LSBs, which comprise the offset are provided by the operand of the instruction.
5. There are 7 types of indirect addressing: auto-increment, auto-decrement, post-indexing by adding of AR0, post-indexing by subtracting contents of AR0, single-

indirect addressing with no increment/decrement, bit reversed addressing with increment, bit-reversed addressing with decrement.

6. a. `MAR *, AR1`

Mode of addressing: Immediate

Operation: Loads ARP with 1 to make AR1 the current auxiliary register.

`LT*+, AR2`

Mode of addressing: Indirect with increment by index amount

Operation: Make AR2 the next auxiliary register. Load the TREG with the content of the address referenced by AR1, add 1 to the content of AR1, and then make AR2 the current auxiliary register.

b. `LDP #500`

Mode of addressing: Immediate

Operation: Load data pointer with value 500 i.e. to address space FA00h - FA7Fh

`ADDC 6h`

Mode of addressing: Direct

Operation: The contents of the data address FA06h and the value of the carry flag © are added to the contents of accumulator.

c. `ADD *- , 8`

Mode of addressing: Indirect with auto-decrement (decrement by 1)

Operation: Add the accumulator to the contents of the data memory address referenced by the current auxiliary register. The data is left shifted by 8 bits before being added. After the operation, the current auxiliary register is decremented by 1.

d. `ADD *BRO-, 8`

Mode of addressing: Indirect with decrement by index amount with reverse carry propagation.

Operation: Add the accumulator to the contents of the data memory address referenced by the current auxiliary register. The data is left shifted by 8 bits before being added. After

the operation, contents of register AR0 are subtracted from the current auxiliary register with reverse carry propagation.

e. SUB #7226, 5

Mode of addressing: Immediate

Operation: Shift 7226 i.e. 1C3Ah left by 5 bits and then subtract the result from the accumulator.

7. The main function of this code is to compare two variables and store the result in memory location (current SPC). With the values given in the example, the result is $var1 < var2$, I.e. the elseif condition is true.

```
var1 .set 1
var2 .set 3
```

Assign values 1 and 3 to variables var1 and var2. These are entered in the symbol table and cannot be modified.

```
.asg "var1>var2", msg1
.asg "var1<var2", msg2
.asg "var1=var2", msg3
```

Assign character strings for the substitution symbols msg1, msg2 and msg3.

```
if_1: .if var1 > var2
      .byte msg1
```

If the condition $var1 > var2$ is true, then store the contents of msg1 at the current location.

```
.elseif var1 < var2
      .byte msg2
```

If the condition $var1 < var2$ is true and $var1 > var2$ is false, then store the contents of msg2 at the current location.

```
.else
      .byte msg3
```

If the condition $var1 > var2$ is false, then store the contents of msg3 at the current location.

```
.endif
```

end the conditional statement.

8. Local Memory Range : 0000h-EFFFh
Global Memory Range : F000h-FFFFh

LABORATORY EXPERIMENT 1

LABORATORY HARDWARE AND SOFTWARE FOR THE DEVELOPMENT OF

TMS320F240 DSP-BASED SYSTEMS

Objectives

The objective of this lab is to introduce the students to the hardware and software used in the laboratory for developing TMS320F240 DSP-based systems. The students will learn about the TMS320F240 evaluation board, and familiarize themselves with the various development software required. At the end of this lab, the students should be able to do the following:

- Understand the working principle of the evaluation board and explain how it can be used to assist in the development of the DSP system.
- Become familiar with the MS-Windows environment used to run the various software needed including TMS320C2xx Assembler and Linker, TMS320C2xx C Source debugger.
- Write simple assembly language programs, assemble and link them, download the assembled code to the evaluation board; and execute it.

Equipment Required

Hardware :

- PC Specifications -
 - '386 or higher IBM PC/AT
 - 1.44Mb 3.5-inch floppy drive
 - 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
 - Minimum 4Mb memory
 - Color VGA Monitor

- TMS320C24x Evaluation Board
- XDS510PP Emulator
- +5V power supply.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable

Software :

- MS Windows 3.1 or Windows 95
- ASCII Editor
- TMS320C2xx Assembler
- TMS320C2xx C Source Debugger

Discussion

The TMS320F240 evaluation board (EVB)

The TMS320F240 EVB is a circuit board from Texas Instruments that can be used for designing, debugging, and evaluating the operation of a TMS320F240 based system. The EVB is built around the TMS320F240 chip along with the necessary circuitry such as timing, memory, and I/O circuitry to aid the development and prototyping of a TMS320F240 DSP-based system. The EVB provides a two 128K × 8 bit SARAM chips that can be used for program as well as data memory. An on-board 4 channel is also included as an added feature for development and testing.

For operation, the EVB requires a +5 volt power supply, a personal computer (PC) which acts as a host computer for the evaluation board and an emulator. The software controls the communication between the PC and the EVB, and allows the following tasks to be performed:

- Downloading code from the PC to the EVB flash EEPROM.
- Execution of programs on the EVB

❑ Debugging the programs

The communication between the PC and the evaluation board is carried out through the XDS510PP emulator via the PC parallel port.

Laboratory Software

The software packages used for the laboratory are as follows:

1. *TMS320C2xx Assembler*

This software translates the assembly language source code files into machine code COFF (common object file format) object files.

2. *TMS320C2xx Linker*

This combines several object files into a single executable COFF object module.

3. *TMS320C2xx C Source Debugger*

This software serves the following purposes -

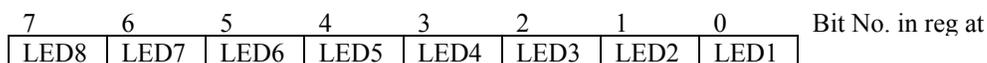
- Downloading of code to the EVB.
- Running and testing the code on the EVB
- Debugging of the code

All this software is compatible with Windows 3.1 and Windows 95.

EXAMPLE PROGRAM 1:

Write a program to turn on consecutive LEDs i.e. LEDs 1, 3, 5 and 7 on the EVB.

The EVB has 8 LEDs that are mapped at address 000Ch of I/O memory space. Thus each LED can be turned on or off by setting or clearing the corresponding bit in the register that is mapped at 000Ch in the I/O space.



Thus to turn on LEDs 1, 3, 5 and 7, the register contents should be 0055h. Thus our task is to write these contents to the register 000Ch in I/O space. To write to I/O space, the OUT instruction is used.

The OUT instruction writes a 16-bit value from a data memory location to a specified I/O location. Only direct and indirect addressing can be used for this instruction. Thus, the data first has to be written to a register in data memory and then output to the I/O space.

The instruction to store data to a data memory register is the SPLK instruction that used. Thus the program segment to write 0055h to the location 000Ch is:

```
SPLK #0055h, LED_STATUS ;Load value into the
                                ;uninitialized register
                                ;LED_STATUS
OUT LED_STATUS, LEDS ;Write the value in LEDS to
                    ;address 000Ch in the I/O
                    ;memory space
```

LED_STATUS is defined as an uninitialized register i.e 16-bits of space is reserved for it in memory. This is similar to the concept of a variable in programming languages. The definition of LED_STATUS is done as follows -

```
.bss LED_STATUS,1 ;LED Status Register
```

The register LEDS refers to the address 000Ch. It is defined as a symbol with constant with the .set directive as follows. Unlike the .bss, the .set defines a memory location with a constant/ initialized value.

```
LEDS .set 000Ch ;LEDs Register
```

A listing of the complete program is shown below:

```
*****
; File Name: ch2_e1.asm
; Target System: C24x Evaluation Board
; Description: This sample program helps you get familiar with
```

```

;           manipulating the I/O mapped LEDS (DS1-DS8) on the
;           F240 EVM Development Board
;*****
        .include f240regs.h
;-----
; I/O Mapped EVM Register Declarations
;-----
LEDS      .set      000Ch      ;LEDS Register
;-----
; Variable Declarations for on chip RAM Blocks
;-----
        .bss GPR0,1      ;General purpose register.
        .bss LED_STATUS,1 ;LED Status Register
;-----
; Vector address declarations
;-----
        .sect      ".vectors"

RSVECT B      START      ; Reset Vector
INT1      B      PHANTOM  ; Interrupt Level 1
INT2      B      PHANTOM  ; Interrupt Level 2
INT3      B      PHANTOM  ; Interrupt Level 3
INT4      B      PHANTOM  ; Interrupt Level 4
INT5      B      PHANTOM  ; Interrupt Level 5
INT6      B      PHANTOM  ; Interrupt Level 6
RESERVED  B      PHANTOM  ; Reserved
SW_INT8   B      PHANTOM  ; User S/W Interrupt
SW_INT9   B      PHANTOM  ; User S/W Interrupt
SW_INT10  B      PHANTOM  ; User S/W Interrupt
SW_INT11  B      PHANTOM  ; User S/W Interrupt
SW_INT12  B      PHANTOM  ; User S/W Interrupt
SW_INT13  B      PHANTOM  ; User S/W Interrupt
SW_INT14  B      PHANTOM  ; User S/W Interrupt
SW_INT15  B      PHANTOM  ; User S/W Interrupt
SW_INT16  B      PHANTOM  ; User S/W Interrupt
TRAP      B      PHANTOM  ; Trap vector
NMINT     B      PHANTOM  ; Non-maskable Interrupt
EMU_TRAP  B      PHANTOM  ; Emulator Trap
SW_INT20  B      PHANTOM  ; User S/W Interrupt
SW_INT21  B      PHANTOM  ; User S/W Interrupt
SW_INT22  B      PHANTOM  ; User S/W Interrupt
SW_INT23  B      PHANTOM  ; User S/W Interrupt

```

```

;=====
; M A I N   C O D E   - starts here
;=====

    .text
    NOP
START: SETC INTM          ;Disable interrupts
       SPLK #0000h,IMR    ;Mask all core interrupts
       LACC IFR          ;Read Interrupt flags
       SACL IFR          ;Clear all interrupt flags

       CLRC SXM          ;Clear Sign Extension Mode
       CLRC OVM          ;Reset Overflow Mode
       CLRC CNF          ;Config Block B0 to Data mem

;-----PLL code begins-----
       LDP  #00E0h        ;DP for addresses 7000h-707Fh
       SPLK #00BBh,CKCR1 ;CLKIN(OSC)=10MHz, CPUCLK=20MHz
       SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable, SYSCLK=CPUCLK/2
       SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK

       SPLK #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 2-3)
       KICK_DOG          ;Reset Watchdog

;-----PLL code ends-----
       SPLK #0h,GPR0     ;Set wait state generator for:
       OUT  GPR0,WSGR    ;Program Space, 0 wait states
                   ;Data Space, 0 wait states
                   ;I/O Space, 0 wait states
       SPLK #00aah,LED_STATUS ;Turn on LEDs DS2, DS4, DS6, DS8
       OUT  LED_STATUS,LEDS ;Turn off LEDs DS1, DS3, DS5, DS7
END     B END

;=====
; I S R   - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM KICK_DOG        ;Resets WD counter
       B PHANTOM

```

Following is a detailed discussion of each statement in the above listing -

```
.include f240regs.h
```

This is the header file for the TMS320F240 processor. It contains all peripheral register declarations as well as other useful definitions. This file must be included for all programs. Here is a brief description of some of the portions of the above program:

The ".sect ".vector" portion defines the interrupt vectors for the various interrupts that can occur in the processor. Since, our program does not use any interrupts, all the vectors branch to the PHANTOM interrupt sub-routine. This routine is basically meant to trap any spurious interrupts.

The first four instructions in the "PLL code", set the CPUCLK to 20 MHz.

The CPU has a watchdog timer unit. This is basically a free running timer which reset sends a system reset on overflow. This is very important in case of software failures, since a system reset takes the CPU to a known location. Thus, typically a program resets the WD timer (with the KICKDOG macro) at regular intervals (before it overflows) so as to avoid unnecessary resets. However, in this program and for development purposes, the watchdog is disabled.

The SPLK #0h,GPR0 and OUT GPR0, WSGR instructions, write 0 wait states in all the data, memory and I/O spaces. Wait states need to be added when the CPU is accessing a slower memory or peripheral.

Connecting the TMS320C240 Evaluation Board and the XDS510PP Emulator

1. Turn off power to the PC.
2. Connect one end of the DB25 printer cable to the parallel port of the PC.
3. Connect the other end of the DB25 printer cable to the XDS510PP emulator.
4. Connect the 14-pin female header on the XDS510PP emulator to the 14-pin right angle emulation port on the evaluation board (P5).

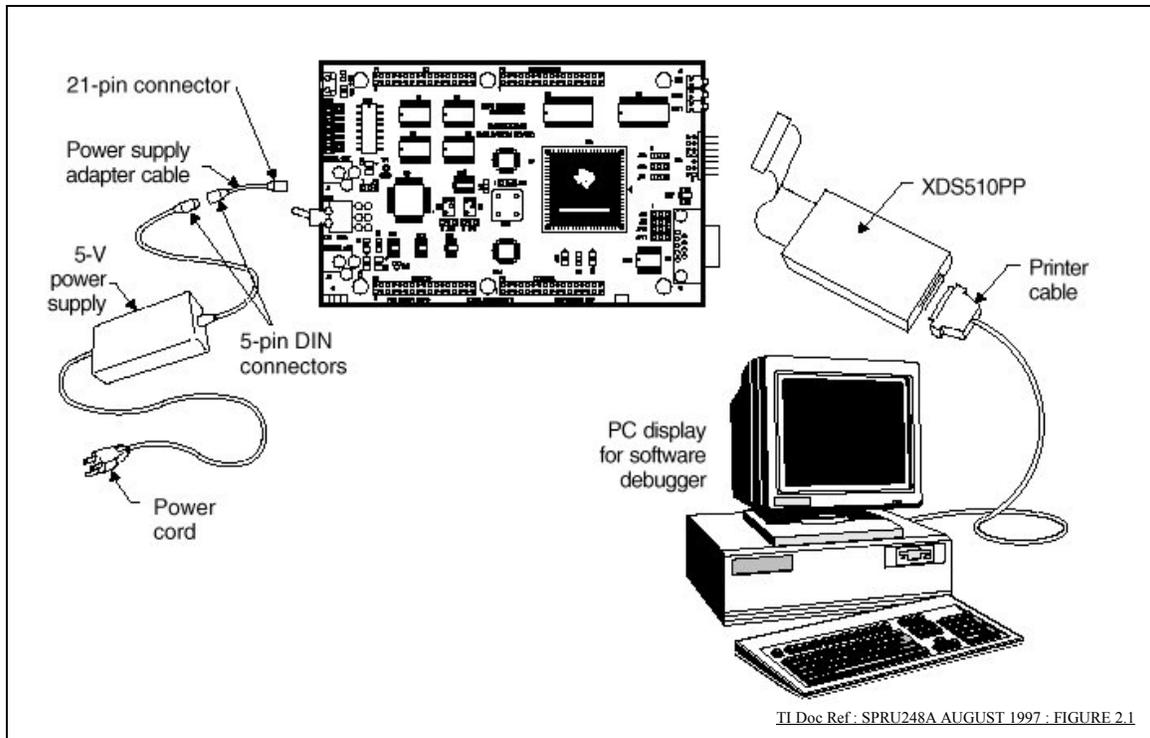


Figure L1.1: Connecting the TMS320F240 EVB and XDS510PP Emulator

Supplying Power to the TMS320C240 Evaluation Board and the XDS510PP Emulator

1. Make sure the power switch on the EVB is in the OFF position.
2. Use the black power cord to plug the power supply into the wall outlet.
3. Connect the 5-pin DIN end of the power supply adapter cable into the 5-pin DIN connector on the power supply.
4. Connect the 2.1-mm end of the power supply adapter cable into the digital power supply jack (J1) on the evaluation board.
5. Turn on the power to the PC and the evaluation board.

Verifying the Emulator Driver

After turning on the computer, you can power on the EVB by pressing the power switch on the EVB power supply. If the power supply is working properly, you should be able to see the red LED in the evaluation board turning on.

Reset the XDS510PP emulator by double clicking the "XDS510PP Reset" icon on the desktop of the computer. After resetting the XDS510PP emulator, you should see a message similar to the following in a system prompt window:

```
EMURST FOR THE XDS510PP VERSION 1.0
XDS510PP IS RESET, HARDWARE VERSION 1
```

This message confirms the working of the emulator, the PC parallel port and in effect it verifies the communication channel between the PC and the EVB.

However if the following message appears, then it means the emulator has not been reset properly:

```
EMURST FOR THE XDS510PP VERSION 1.0
COMMUNICATIONS ERROR, OR POD HAS NO POWER
```

In this case, check the following:

- Be sure power is supplied to the EVB and the power switch on the board (SW3) is in the ON position.
- Verify that the parallel port cable is firmly connected to the parallel port on the PC and to the XDS510PP emulator pod.
- Verify that the 14-pin header on the XDS510PP emulator pod is firmly connected to the emulation port (P5) on the evaluation board.

Verifying the TMS320F240 EVM C Source Debugger

After successfully resetting the XDS510PP emulator, invoke the debugger by double clicking on the "C24x EVM Debugger" icon on the computer desktop. A display similar to the following one should appear:



Figure L1.2: TMS320F240 C Source Debugger Display

This display verifies the communication of the debugger software with the emulator and the evaluation board.

- If you see a display and the lines of code say *Invalid Address* or all the fields in the MEMORY window are shown in red, contact the instructor.
- If the following message appears in the debugger display :

CANNOT INITIALIZE THE TARGET!!

- Check the I/O configuration
- Check cabling and target power.

In this case, do the following:

- Be sure power is supplied to the EVM and the power switch SW3 is in the ON position
- Verify that the parallel port is firmly connected to the parallel port on the PC and to the XDS510PP emulator pod.
- Verify that the 14-pin header on the XDS510PP emulator pod is firmly connected to the emulation port (P5) on the evaluation board.

Assembling and Linking Your First Program

In the directory `c:\dsp-lab`, save your assembly source file in this directory under the name "leds.asm". To assemble the program, type the following command on the MSDOS command prompt:

```
cd c:\dsp-lab\name ↵  
dspa leds.asm -v2xx -s ↵
```

The `-v2xx` option tells the assembler to produce code for the 'C2xx family of devices. The `-s` option puts all defined symbols, including labels, in the object file's symbol table.

The following message should appear if the assembly is successful:

```
TMS320C1x/C2x/C2xx/C5x COFF Assembler Version 6.60  
Copyright © 1987-1995 Texas Instruments Incorporation  
Pass 1  
Pass2  
No Errors, No Warnings
```

The assembler produces a relocatable object file, `leds.obj` that is then used by the linker.

The linker generates a `leds.out` file that can be directly downloaded to the TMS320F240 device. Execute the following command from the MSDOS command prompt to link the `leds.obj` and the `f240lnk.cmd` files:

```
dsplnk leds.obj f240lnk.cmd -o leds.out
```

This command creates the `leds.out` file. The `-o` option specifies the name of the output file. The following message should appear if the linking is successful:

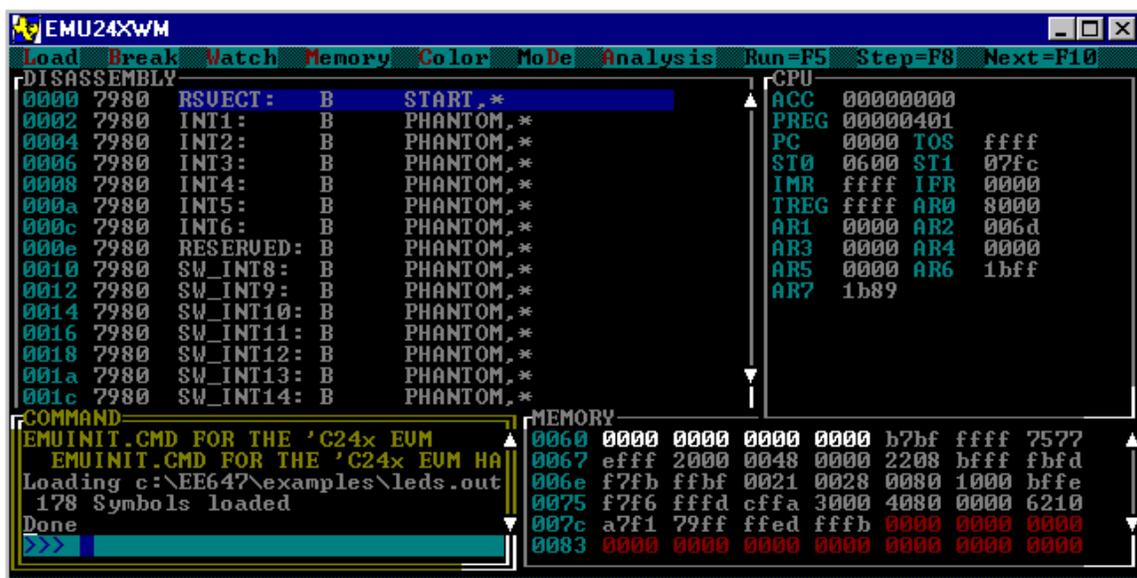
```
TMS320C1x/C2x/C2xx/C5x COFF Linker Version 6.60  
Copyright © 1987-1995 Texas Instruments Incorporation
```

Downloading and Running Your First Program

Invoke the C source debugger by double clicking on the "C24x EVM Debugger" icon on the computer desktop. To download the program to the TMS320F240, type the following command at the prompt in the command window of the debugger:

```
load c:\dsp-lab\name\leds.out
```

If the loading is successful, a display similar to the following should appear:



The screenshot shows the EMU24XWM debugger interface. The main window is divided into several panes:

- DISASSEMBLY:** Shows assembly instructions starting with address 0000 and instruction RSUECT: B START.*. Subsequent instructions are INT1: B PHANTOM,*, INT2: B PHANTOM,*, INT3: B PHANTOM,*, INT4: B PHANTOM,*, INT5: B PHANTOM,*, INT6: B PHANTOM,*, RESERVED: B PHANTOM,*, SW_INT8: B PHANTOM,*, SW_INT9: B PHANTOM,*, SW_INT10: B PHANTOM,*, SW_INT11: B PHANTOM,*, SW_INT12: B PHANTOM,*, SW_INT13: B PHANTOM,*, and SW_INT14: B PHANTOM,*.
- CPU:** Shows register values: ACC 00000000, PREG 00000401, PC 0000 TOS ffff, ST0 0600 ST1 07fc, IMR ffff IFR 0000, TREG ffff AR0 0000, AR1 0000 AR2 006d, AR3 0000 AR4 0000, AR5 0000 AR6 1bff, and AR7 1b89.
- COMMAND:** Shows the command prompt with the following text: EMUINIT.CMD FOR THE 'C24x EUM, EMUINIT.CMD FOR THE 'C24x EUM HA, Loading c:\EE647\examples\leds.out, 178 Symbols loaded, Done, and a prompt >>>.
- MEMORY:** Shows a memory dump starting at address 0060 with values: 0000 0000 0000 0000 0000 b7bf ffff 7577, 0067 efff 2000 0048 0000 2208 bfff fbfd, 006e f7fb ffbf 0021 0028 0080 1000 bffe, 0075 f7f6 fffd cffa 3000 4080 0000 6210, 007c a7f1 79ff ffed fffb 0000 0000 0000, and 0083 0000 0000 0000 0000 0000 0000 0000.

Press F5 to run the program after it has been loaded. The LEDs DS1, DS3, DS5 and DS7 should turn on if the program is running successful.

EXAMPLE PROGRAM 2:

Write a program to check the status of the DIP switches and accordingly manipulate the corresponding LEDs i.e. if DIP switch 1 is ON, turn LED1 (DS1) ON and vice versa etc.

The EVB has 8 DIP switches that are mapped at address 0008h of I/O memory space. As mentioned earlier, the LEDs are mapped at address 000Ch. Thus, we first define these two registers so that they can be referred to as symbols as follows:

```
LEDS      .set 000Ch    ;LEDs Register
SWITCHES .set 0008h    ;DIP SWITCH Register
```

The first task is to read the status of the DIP switches. This is done using the IN instruction. The status is read into a register DIP_STATUS which is defined as an uninitialized variable. To write this data to the I/O space assigned to the LEDs, the OUT instruction is used. The relevant code segment is as follows:

```
.bss DIP_STATUS,1          ;DIP SWITCH Register

IN   DIP_STATUS, SWITCHES ; Get the status of each DIP
                                ; switch status
OUT  DIP_STATUS, LEDES    ; Turn LEDs on/off depending
                                ; on status of corresponding
                                ; switch
```

A listing of the complete program is shown below:

```
*****
; File Name:    ch2_e2.asm
; Description: This sample program helps you get familiar with
;              manipulating the I/O mapped LEDES (DS1-DS8) on the
;              F240 EVM Development Board depending on the status of
;              the DIP switches
;*****
        .include f240regs.h
;-----
; I/O Mapped EVM Register Declarations
;-----

LEDES      .set    000Ch    ;LEDs Register
SWITCHES .set    0008h    ;DIP SWITCH Register
```

```

;-----
; Variable Declarations for on chip RAM Blocks
;-----
        .bss GPR0,1      ;General purpose register.
        .bss DIP_STATUS,1 ;DIP SWITCH Register

;-----
; Vector address declarations
;-----
        .sect ".vectors"

RSVECT B    START      ; Reset Vector
INT1      B    PHANTOM  ; Interrupt Level 1
INT2      B    PHANTOM  ; Interrupt Level 2
INT3      B    PHANTOM  ; Interrupt Level 3
INT4      B    PHANTOM  ; Interrupt Level 4
INT5      B    PHANTOM  ; Interrupt Level 5
INT6      B    PHANTOM  ; Interrupt Level 6
RESERVED  B    PHANTOM  ; Reserved
SW_INT8   B    PHANTOM  ; User S/W Interrupt
SW_INT9   B    PHANTOM  ; User S/W Interrupt
SW_INT10  B    PHANTOM  ; User S/W Interrupt
SW_INT11  B    PHANTOM  ; User S/W Interrupt
SW_INT12  B    PHANTOM  ; User S/W Interrupt
SW_INT13  B    PHANTOM  ; User S/W Interrupt
SW_INT14  B    PHANTOM  ; User S/W Interrupt
SW_INT15  B    PHANTOM  ; User S/W Interrupt
SW_INT16  B    PHANTOM  ; User S/W Interrupt
TRAP      B    PHANTOM  ; Trap vector
NMINT     B    PHANTOM  ; Non-maskable Interrupt
EMU_TRAP  B    PHANTOM  ; Emulator Trap
SW_INT20  B    PHANTOM  ; User S/W Interrupt
SW_INT21  B    PHANTOM  ; User S/W Interrupt
SW_INT22  B    PHANTOM  ; User S/W Interrupt
SW_INT23  B    PHANTOM  ; User S/W Interrupt

;=====
; M A I N   C O D E   - starts here
;=====

        .text
        NOP

```

```

START: SETC INTM          ;Disable interrupts
      SPLK #0000h,IMR      ;Mask all core interrupts
      LACC IFR            ;Read Interrupt flags
      SACL IFR            ;Clear all interrupt flags

      CLRC SXM            ;Clear Sign Extension Mode
      CLRC OVM            ;Reset Overflow Mode
      CLRC CNF            ;Config Block B0 to Data mem
;-----PLL code begins-----
      LDP  #00E0h          ;DP for addresses 7000h-707Fh
      SPLK #00BBh,CKCR1    ;CLKIN(OSC) =10MHz, CPUCLK =
                          ;20MHz
      SPLK #00C3h,CKCR0    ;CLKMD=PLL Enable, SYSCLK =
                          ;CPUCLK/2
      SPLK #40C0h,SYSCR    ;CLKOUT=CPUCLK
      SPLK #006Fh, WDCR    ;Disable WD if VCCP=5V (JP5 2-3)
      KICK_DOG             ;Reset Watchdog
;-----PLL code ends-----
      SPLK #0h,GPR0        ;Set wait state generator for:
      OUT  GPR0,WSGR        ;Program Space, 0 wait states
                          ;Data Space, 0 wait states
                          ;I/O Space, 0 wait states
      IN   DIP_STATUS, SWITCHES ;Get the status of each DIP
                          ;switch
      OUT  DIP_STATUS, LEDS  ;Turn on/off LED depending on
                          ;status of corresponding switch
END      B END
;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM  KICK_DOG          ;Resets WD counter
      B PHANTOM

```

To run the program, do the following:

Assemble, link and load the program as described earlier. Adjust the DIP switches such that switched 1, 2, 3 and 4 are ON and switches 5,6,7 and 8 are OFF. Now press the F5 key to run the program. If the execution is successful, LEDs DS1, DS2, DS3 and DS4 will be ON and DS5, DS6, DS7 and DS8 will be OFF.

EXAMPLE PROGRAM 3:

Write a program to add 2 numbers that are stored as uninitialized variables.

We will use the ADD instruction with the direct addressing in this example. Let the variables to be added, be initialized as 'var1' and 'var2'. The ADD instruction adds the contents of a register to the contents of the accumulator and stores the result in the accumulator. The relevant code segment is:

```
.bss var1,1
.bss var2,1

SPLK #0002h,var1 ;Store a value 2h in var1
SPLK #0003h,var2 ;Store a value 3h in var2
LACC var1 ;Load contents of var1 in
                ;accumulator
ADD var2 ;Add contents of var2 to
                ;contents of accumulator.
                ;Store results in accumulator
```

Running the program:

Assemble, link and load the program. To check the program, the values of var1, var2 and accumulator need to be checked. The CPU window of the debugger shows the contents of the accumulator (ACC). To observe the values of var1 and var2, open the watch window of the debugger with the following command:

```
wa *var1 ↵
```

```
wa *var2 ↵
```

wa is the command to add a variable in the watch window. The ***** tells the debugger that it is the data value of the variable that you wish to observe. If the ***** is omitted, then the debugger keeps a watch on the address of the variable rather than its value. A detailed discussion of the various debugger commands can be found in the appendix to this chapter.

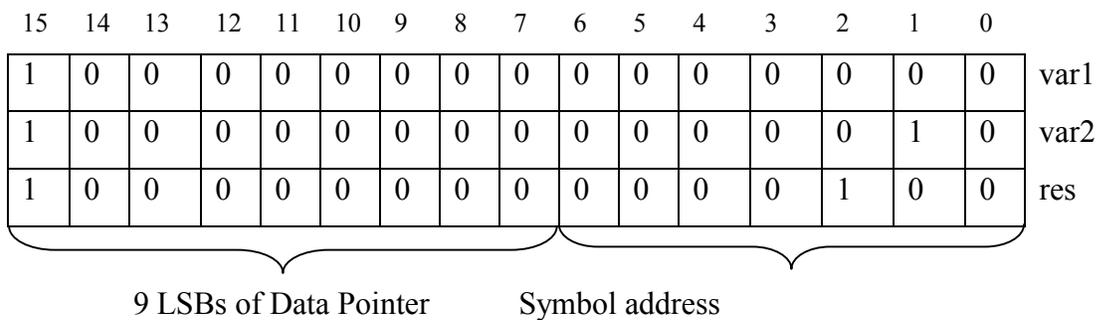
Now run the program by pressing the F5 key. Now observe the contents of the accumulator and variables. For successful execution of the program the following values are expected:

ACC	0x0005
var1	0x0002
var2	0x0003

EXAMPLE PROGRAM 4:

Write a program to add two numbers stored at specific memory locations. Store the result in a third memory location. All these memory locations should be on the external SARAM.

Refer the memory map of the EVB as shown in Figure 11. The address range for SARAM is 8000h - FFFFh. Let us select the addresses 8000h, 8002h and 8004h for this program. The first task is to define symbols for these memory locations - var1, var2 and res. When we use direct addressing, the address is formed with the 9 MSBs taken from the data pointer and the 7 LSBs are taken from the operand of the instruction. This break-up is as shown below-



Thus the DP value will be 0100h. The relevant code is:

```
var1      .set      0000h
var2     .set      0002h
res      .set      0004h
```

```

LDP      #0100h ;Load Data Pointer
LACC     var1 ;Load value of var1 in accumulator
ADD      var2
SACC     res  ;Store result in res i.e. memory
           ;location 8004h

```

Run the program. The checking of variables can be done using the **mem** command as shown below-

```
mem 0x8000
```

This will display the contents of memory addresses 8000h onwards in the memory window of the debugger.

EXAMPLE PROGRAM 5:

Write a program to multiply two numbers. Both the numbers are stored in memory location in the SARAM.

Let the memory locations be 8000h and 8002h. For the multiply instruction, the multiplicand needs to be in the TREG and the result is stored in PREG. Accordingly the relevant code is:

```

mtplr   .set  0000h
mtplcnd .set  0002h

LDP     #0100h
SPLK    #0002h,mtplr
SPLK    #0003h,mtplcnd
LT      mtplcnd ;Load multiplicand into TREG
MPY     mtplr   ;Multiply contents of TREG
           ;with multiplier

```

To view the contents of TREG and PREG, check the CPU window of the debugger. The values of `mtplr` and `mtplcnd` can be checked with the `wa` command. The contents should be:

	After Instruction
8000h / <code>mtplr</code>	0x0002
80002h/ <code>mtplcnd</code>	0x0003
TREG	0x0003
PREG	0x00000006

EXAMPLE PROGRAM 6:

Check the value of a temporary register `TEMP`. If `TEMP = 1`, add the contents of `var1` and `var2`. Store the result in `res`. If `TEMP=2`, subtract contents of `var2` from `var1` and store result in `res`. For any other value of `TEMP`, multiply the contents of `var1` and `var2` and store result in `res`.

The purpose of this example is to introduce conditional `.if/elseif/else/endif` directives.

The relevant code is:

```

        TEMP    .set  1
        var1    .set  000eh
        var2    .set  000fh
        res     .set  0010h
one     .set  1
two     .set  2

        LDP    #0100h
        SPLK   #0005h,var1
        SPLK   #0002h,var2
lbl_if: .if     TEMP = one
        LACC   var1
        ADD    var2

```

```

SACL    res    ;res = var1 + var2
.elseif TEMP = two
LACC    var1
SUB     var2
SACL    res    ;res = var1 - var2
.else
LT      var1
MPY     var2
SPL     res    ;res = var1 * var2
.endif

```

EXAMPLE PROGRAM 7:

Write a program to increment a variable 10 times.

The main purpose of this program is to introduce the `.loop/.break/.endloop` directive.

The relevant code is:

```

.bss   ctr,1
.bss   var,1

SPLK   #0,var
.eval  0,ctr ;Set counter = 0
lbl    .loop
LACC   #1
ADD    var
SACL   var    ;var = var + 1
.eval  ctr+1,ctr ;Increment counter
.break ctr=10 ;If counter=10, exit loop
.endloop

```

Since the initial value of `var` is 0, the value of `var` after the completion of the program will be 10 or 000Ah.

EXAMPLE PROGRAM 8:

Write a program to turn on the LEDs DS8 to DS1, one after the other.

The purpose of this program is to introduce the bit-shift operation. The Shift Right SFR instruction will be employed. The program is listed below with infinite iteration.

```
;*****
; File Name:      ch2_e8.asm
; Target System: C24x Evaluation Board
; Description:    Turn on the LEDs DS8 to DS1, one after
;the other using shift right instruction
;*****
        .include  f240regs.h
;-----
; I/O Mapped EVM Register Declarations
;-----
LEDS    .set  000ch    ; Address of LEDs
;-----
; Variable Declarations for on chip RAM Blocks
;-----
        .bss  GPR0,1    ; General purpose register.
        .bss  ctr,1     ; LED number counter
        .bss  LED_STATUS,1 ; LED status
        .bss  RPT_NUM,1 ; for delay subroutine
        .bss  mSEC,1   ; for delay subroutine
;-----
; Vector address declarations
;-----
        .sect  ".vectors"

RSVECT   B    START    ; Reset Vector
INT1     B    PHANTOM  ; Interrupt Level 1
```

```

INT2      B      PHANTOM ; Interrupt Level 2
INT3      B      PHANTOM ; Interrupt Level 3
INT4      B      PHANTOM ; Interrupt Level 4
INT5      B      PHANTOM ; Interrupt Level 5
INT6      B      PHANTOM ; Interrupt Level 6
RESERVED  B      PHANTOM ; Reserved
SW_INT8   B      PHANTOM ; User S/W Interrupt
SW_INT9   B      PHANTOM ; User S/W Interrupt
SW_INT10  B      PHANTOM ; User S/W Interrupt
SW_INT11  B      PHANTOM ; User S/W Interrupt
SW_INT12  B      PHANTOM ; User S/W Interrupt
SW_INT13  B      PHANTOM ; User S/W Interrupt
SW_INT14  B      PHANTOM ; User S/W Interrupt
SW_INT15  B      PHANTOM ; User S/W Interrupt
SW_INT16  B      PHANTOM ; User S/W Interrupt
TRAP      B      PHANTOM ; Trap vector
NMINT     B      PHANTOM ; Non-maskable Interrupt
EMU_TRAP  B      PHANTOM ; Emulator Trap
SW_INT20  B      PHANTOM ; User S/W Interrupt
SW_INT21  B      PHANTOM ; User S/W Interrupt
SW_INT22  B      PHANTOM ; User S/W Interrupt
SW_INT23  B      PHANTOM ; User S/W Interrupt

;=====
; M A I N   C O D E   - starts here
;=====

        .text
        NOP
START:  SETC  INTM          ;Disable interrupts
        SPLK #0000h,IMR    ;Mask all core interrupts
        LACC  IFR          ;Read Interrupt flags
        SACL  IFR          ;Clear all interrupt flags

        CLRC  SXM         ;Clear Sign Extension Mode
        CLRC  OVM         ;Reset Overflow Mode

```

```

CLRC  CNF      ;Config Block B0 to Data mem

LDP   #00E0h   ;DP for addresses 7000h-707Fh
SPLK  #00BBh,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
SPLK  #00C3h,CKCR0 ;CLKMD=PLL Enable,
      ;SYSCLK=CPUCLK/2
SPLK  #40C0h,SYSCR ;CLKOUT=CPUCLK

SPLK  #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 2-3)
KICK_DOG      ;Reset Watchdog

SPLK  #0h,GPR0   ;Set wait state generator for:
OUT   GPR0,WSGR  ;Program Space, 0 wait states
      ;Data Space, 0 wait states
      ;I/O Space, 0 wait states

strt  SPLK      #0080h,LED_STATUS

      .eval 0,ctr ; Set counter = 0
lbl   .loop
OUT   LED_STATUS,LEDS ; Turn on LEDs
      ; based on status
LACC  LED_STATUS   ; Load status into ACC
SFR   ; Shift right ACC
SACL  LED_STATUS   ; Update LED status
      CALL      mS_DELAY ; Delay for 1 sec
      .eval ctr+1,ctr ; Increment counter
      .break ctr=8 ; If counter=8, exit loop
      .endloop
      B        strt ; Repeat shifting
      ; from beginning

END   B END

;=====
; Subroutine: mS_DELAY,

```

```

; Discription: implement a delay of approximately 1 sec
;=====
mS_DELAY:      LDP      #0h    ; DP->0000h-007Fh
                LACC     #4000  ; Load repeat number
                SACL    RPT_NUM ; Store repeat number
                SPLK    #5000,mSEC ; Initialize loop
counter
mS_LOOP:      LDP      #0h    ; DP->0000h-007Fh
                RPT     RPT_NUM ; Repeat next instruction
                NOP     ; 4000 cycles = 0.2ms
                LACC    mSEC    ; Load value of counter
                SUB     #1      ; Decrement ACC
                SACL    mSEC    ; Update loop counter
                BCND    mS_LOOP,NEQ
                ; Jump to mS_LOOP
                ; if not zero
                RET     ; Return
;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM      KICK_DOG    ;Resets WD counter
                B PHANTOM

```

Look-up tables play a very important role in any programming language. In the next few examples, we shall see how to access data from a look-up table, how to write data in tabular form etc.

EXAMPLE PROGRAM 9:

Write a program that reads data from the program memory and writes it to the address 8000h of external data memory. The total number of words written is 10.

The TBLR instruction allows a word from a location in program memory to be transferred to a specific location in data memory. We will use this instruction in order to achieve the above objective. The table in program memory is defined as TABLEA and the destination table in data memory is defined as TABLEB. A counter (CTR) is setup in order to transfer 10 words. The BNZ (Branch if ACC > 0) conditional branch is maintains the loop for 10 word transfer. The complete program is as below -

```
;*****  
; File Name:    tblp-d.asm  
; Target System: C24x Evaluation Board  
;  
; Description:  This program transfers a block of data from  
;              the program memory to a particular location  
;              in data memory. Data transfer is from TABLEA  
;              to TABLEB.  
;*****  
  
    .include    f240regs.h  
  
TABLEB .set    8000h    ;Starting address of the  
                        ;destination table  
COUNT .set    10      ;Defines the number of entries  
                        ;in the table
```

```

;-----
; Variable Declarations for on chip RAM Blocks
;-----
        .bss  GPR0,1      ;General purpose register.
        .bss  SRCTBL,1
        .bss  CNT,1

;-----
; Vector address declarations
;-----
        .sect ".vectors"

RSVECT  B      START      ; Reset Vector
INT1    B      PHANTOM    ; Interrupt Level 1
INT2    B      PHANTOM    ; Interrupt Level 2
INT3    B      PHANTOM    ; Interrupt Level 3
INT4    B      PHANTOM    ; Interrupt Level 4
INT5    B      PHANTOM    ; Interrupt Level 5
INT6    B      PHANTOM    ; Interrupt Level 6
RESERVED B      PHANTOM    ; Reserved
SW_INT8 B      PHANTOM    ; User S/W Interrupt
SW_INT9 B      PHANTOM    ; User S/W Interrupt
SW_INT10 B     PHANTOM    ; User S/W Interrupt
SW_INT11 B     PHANTOM    ; User S/W Interrupt
SW_INT12 B     PHANTOM    ; User S/W Interrupt
SW_INT13 B     PHANTOM    ; User S/W Interrupt
SW_INT14 B     PHANTOM    ; User S/W Interrupt
SW_INT15 B     PHANTOM    ; User S/W Interrupt
SW_INT16 B     PHANTOM    ; User S/W Interrupt
TRAP    B      PHANTOM    ; Trap vector
NMINT   B      PHANTOM    ; Non-maskable Interrupt
EMU_TRAP B     PHANTOM    ; Emulator Trap
SW_INT20 B     PHANTOM    ; User S/W Interrupt
SW_INT21 B     PHANTOM    ; User S/W Interrupt
SW_INT22 B     PHANTOM    ; User S/W Interrupt

```

```

SW_INT23  B      PHANTOM      ; User S/W Interrupt
;=====
; M A I N  C O D E  - starts here
;=====

        .text
        NOP
START:  SETC  INTM              ;Disable interrupts
        SPLK  #0000h,IMR      ;Mask all core interrupts
        LACC  IFR              ;Read Interrupt flags
        SACL  IFR              ;Clear all interrupt flags

        CLRC  SXM              ;Clear Sign Extension Mode
        CLRC  OVM              ;Reset Overflow Mode
        CLRC  CNF              ;Config Block B0 to Data mem

        LDP   #00E0h          ;DP for addresses 7000h-707Fh
        SPLK  #00BBh,CKCR1    ;CLKIN(OSC)=10MHz, CPUCLK=20MHz
        SPLK  #00C3h,CKCR0    ;CLKMD=PLL Enable, SYSCLK =
                                ;CPUCLK/2
        SPLK  #40C0h,SYSCR    ;CLKOUT=CPUCLK

        SPLK  #006Fh, WDCR    ;Disable WD if VCCP=5V (JP2-3)
        KICK_DOG          ;Reset Watchdog

        SPLK  #0h,GPR0        ;Set wait state generator for:
        OUT   GPR0,WSGR      ;Program Space, 0 wait states
                                ;Data Space, 0 wait states
                                ;I/O Space, 0 wait states

        LACC  #COUNT
        SACL  CNT            ;Store the no. of data entries
                                ;in CNT

        LARP  1              ;Select AR1 as the current AR
        LDP  #0h            ;DP=0, since the source data

```

```

                                ;table is in program memory
LAR AR1,#TABLEB                ;Load the starting address of
                                ;the destination table in AR1

LACC #TABLEA
SACL SRCTBL                    ;Point the data pointer SRCTBL
                                ;to the top of the source data
LOOP LACC SRCTBL
    TBLR *+                    ;Read the value from the table
                                ;and store the destination
                                ;table. Increment AR1 to point
                                ;to the next address in the
                                ;destination table

    ADD #1
    SACL SRCTBL                ;Increment source data pointer
    LACC CNT
    SUB #1
    SACL CNT                    ;Decrement loop count
    BNZ LOOP                    ;Continue if CNT > 0; i.e.
                                ;until the end of the source
                                ;data table is reached.

END B END                      ;End Program

;-----
; Data look-up table
; No. Entries : 10
;-----
TABLEA .word 0
        .word 1h
        .word 2h
        .word 3h
        .word 4h
        .word 5h
        .word 6h
        .word 7h

```

```
        .word    8h
        .word    9h

;=====
; I S R   -   PHANTOM
;
; Description:  Dummy ISR, used to trap spurious interrupts.
;
; Modifies:  Nothing
;=====
PHANTOM    KICK_DOG            ;Resets WD counter
          B PHANTOM
```

To check this program, type the following commands in the debugger command window and run the program.

```
mem2 0x8000
```

The screen should appear as follows-
Observe the locations 8000h to 8009h.



EXAMPLE PROGRAM 10:

Write a program that reads data from the data memory and writes it to the address 7000h of program memory. The total number of words written is 10.

Here we use the TBLW instruction which transfers a word from data memory to program memory. The rest of the logic for the program remains the same as in the previous example. Thus, data is transferred from TABLEB ; to TABLEA.

```

;*****
; File Name:    tbld-p.asm
; Target System:C24x Evaluation Board
; Description:  This program transfers a block of data from
;              the data memory to a particular location in ;
;              program memory. Data transfer is from TABLEB ;      to
;              TABLEA.
;*****

```

```

        .include  f240regs.h

TABLEA    .set  7000h ;Starting address of the
           ;destination table
TABLEB    .set  8000h ;Starting address of the
           ;source table
COUNT    .set  10   ;No. of entries in the table

;-----
; Variable Declarations for on chip RAM Blocks
;-----

        .bss  GPR0,1   ;General purpose register.
        .bss  DESTBL,1
        .bss  CNT,1

;-----
; Vector address declarations
;-----

        .sect  ".vectors"

RSVECT    B    START  ; Reset Vector
INT1      B    PHANTOM ; Interrupt Level 1
INT2      B    PHANTOM ; Interrupt Level 2
INT3      B    PHANTOM ; Interrupt Level 3
INT4      B    PHANTOM ; Interrupt Level 4
INT5      B    PHANTOM ; Interrupt Level 6
RESERVED  B    PHANTOM ; Reserved
SW_INT8   B    PHANTOM ; User S/W Interrupt
SW_INT9   B    PHANTOM ; User S/W Interrupt
SW_INT10  B    PHANTOM ; User S/W Interrupt
SW_INT11  B    PHANTOM ; User S/W Interrupt
SW_INT12  B    PHANTOM ; User S/W Interrupt
SW_INT13  B    PHANTOM ; User S/W Interrupt
SW_INT14  B    PHANTOM ; User S/W Interrupt

```

```

SW_INT15    B    PHANTOM    ; User S/W Interrupt
SW_INT16    B    PHANTOM    ; User S/W Interrupt
TRAP        B    PHANTOM    ; Trap vector
NMINT       B    PHANTOM    ; Non-maskable Interrupt
EMU_TRAP    B    PHANTOM    ; Emulator Trap
SW_INT20    B    PHANTOM    ; User S/W Interrupt
SW_INT21    B    PHANTOM    ; User S/W Interrupt
SW_INT22    B    PHANTOM    ; User S/W Interrupt
SW_INT23    B    PHANTOM    ; User S/W Interrupt

;=====
; M A I N    C O D E    - starts here
;=====

    .text
    NOP
START:  SETC    INTM            ;Disable interrupts
        SPLK   #0000h,IMR      ;Mask all core interrupts
        LACC   IFR            ;Read Interrupt flags
        SACL   IFR            ;Clear all interrupt flags

        CLRC   SXM            ;Clear Sign Extension Mode
        CLRC   OVM            ;Reset Overflow Mode
        CLRC   CNF            ;Config Block B0 to Data mem

        LDP    #00E0h          ;DP for addresses 7000h-707Fh
        SPLK   #00BBh,CKCR1    ;CLKIN(OSC)=10MHz, CPUCLK=20MHz
        SPLK   #00C3h,CKCR0    ;CLKMD=PLL Enable , SYSCLK =
                                ;CPUCLK/2
        SPLK   #40C0h,SYSCR    ;CLKOUT=CPUCLK

        SPLK   #006Fh, WDCR    ;Disable WD if VCCP=5V(JP52-3)
        KICK_DOG          ;Reset Watchdog

        SPLK   #0h,GPR0        ;Set wait state generator for:
        OUT    GPR0,WSGR      ;Program Space, 0 wait states

```

```

;Data Space, 0 wait states
;I/O Space, 0 wait states
LACC #COUNT
SACL CNT ;Store the number of data
;entries in CNT
LARP 1 ;Select AR1 as the current AR
LDP #0h ;DP=0, since the source data
;table is in program memory
LAR AR1,#TABLEB ;Load the starting address of
;the source table in AR1
LACC #TABLEA
SACL DESTBL ;Point the data pointer DESTBL
;to the top of the destination
;data table
LOOP LACC DESTBL
TBLW *+ ;Read the value from the table
;and store the destination
;table. Increment AR1 to point
;to the next address in the
;source table
ADD #1
SACL DESTBL ;Increment destination data
;pointer
LACC CNT
SUB #1
SACL CNT ;Decrement loop count
BNZ LOOP ;Continue if CNT > 0; i.e.
;until the end of the source
;data table is reached.
END B END ;End Program

;=====
; I S R - PHANTOM
;

```

```

; Description: Dummy ISR, used to trap spurious interrupts.
;
; Modifies: Nothing
;=====
PHANTOM KICK_DOG ;Resets WD counter
        B PHANTOM

```

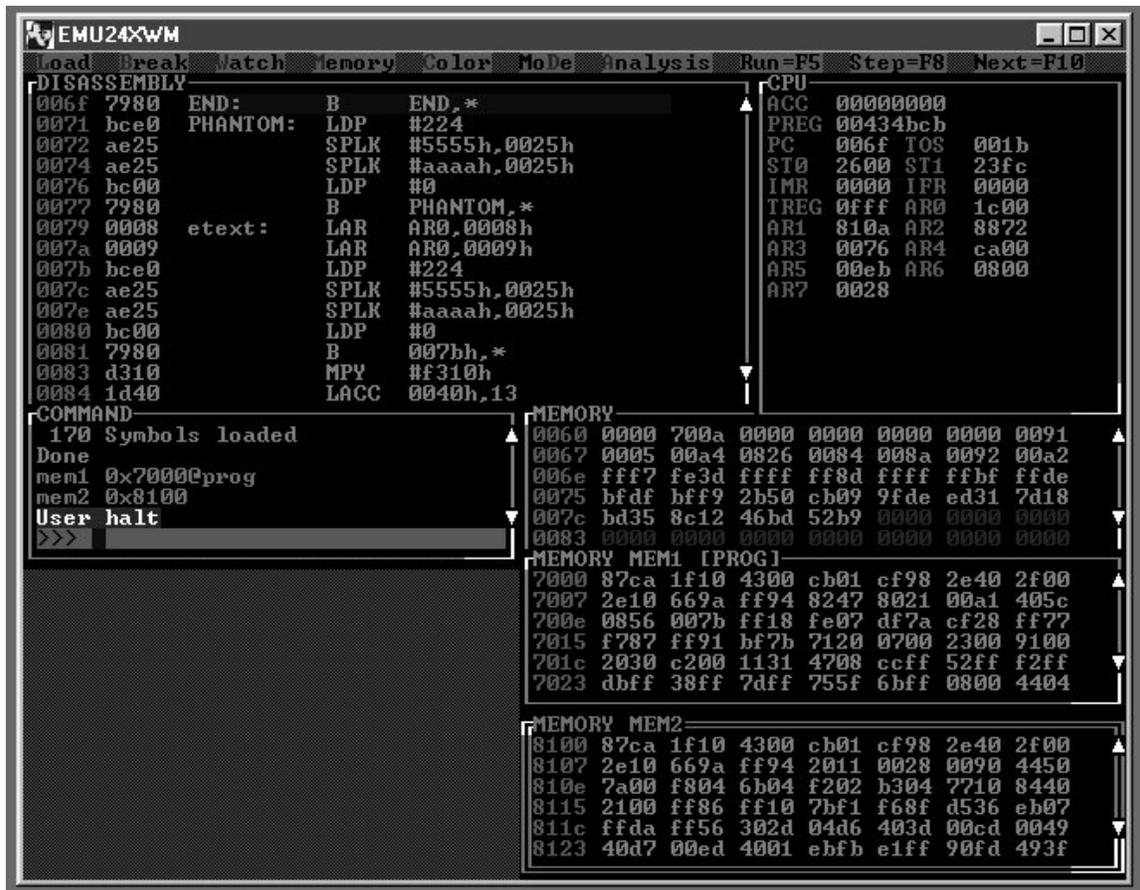
To check this program, type the following commands in the debugger command window and run the program.

mem1 0x7000@prog - Destination

mem2 0x8000 - Source

The screen should appear as follows-

Observe the locations 8000h to 8009h.



EXAMPLE PROGRAM 11:

Write a program that reads data from a location in data memory (8310h) and writes it to another location in data memory (8410h). The total number of words written is 10.

The BLDD instruction allows a word in data memory pointed to by *source* to be copied into another data memory location pointed by *destination*. The various addressing modes allowed for this instruction are -

BLDD #lk, dma

BLDD #lk,ind[, ARn]

BLDD dma, #lk,

BLDD ind, #lk [, ARn]

In this example, we'll use the **BLDD** #lk,ind[, ARn] mode. The RPT instruction is employed to repeat the transfer 10 times. The number of words to be transferred is stored in the RPTCNT. When the BLDD instruction is repeated, the source address specified by the long immediate constant is stored in the PC. Since, the PC is incremented after every repetition, the source address is also incremented. In case of the destination, the auto-increment option of indirect addressing is used. The complete code listing is given below-

```
*****
; File Name:    BLDD-d.asm
; Target System: C24x Evaluation Board
;
; Description:  This program transfers a block of data from
;               the data memory to a another location in
;               data memory.
;*****
                .include f240regs.h

TABLEA        .set  8310h ;Starting address of the
                ;source table
TABLEB        .set  8410h ;Starting address of the
```

```

;destination table
RPTCNT      .set  10      ;Defines the number of entries
;in the table
;-----
; Variable Declarations for on chip RAM Blocks
;-----
        .bss  GPR0,1      ;General purpose register.
;-----
; Vector address declarations
;-----
        .sect ".vectors"

RSVECT      B      START  ; Reset Vector
INT1        B      PHANTOM ; Interrupt Level 1
INT2        B      PHANTOM ; Interrupt Level 2
INT3        B      PHANTOM ; Interrupt Level 3
INT4        B      PHANTOM ; Interrupt Level 4
INT5        B      PHANTOM ; Interrupt Level 5
INT6        B      PHANTOM ; Interrupt Level 6
RESERVED    B      PHANTOM ; Reserved
SW_INT8     B      PHANTOM ; User S/W Interrupt
SW_INT9     B      PHANTOM ; User S/W Interrupt
SW_INT10    B      PHANTOM ; User S/W Interrupt
SW_INT11    B      PHANTOM ; User S/W Interrupt
SW_INT12    B      PHANTOM ; User S/W Interrupt
SW_INT13    B      PHANTOM ; User S/W Interrupt
SW_INT14    B      PHANTOM ; User S/W Interrupt
SW_INT15    B      PHANTOM ; User S/W Interrupt
SW_INT16    B      PHANTOM ; User S/W Interrupt
TRAP        B      PHANTOM ; Trap vector
NMINT       B      PHANTOM ; Non-maskable Interrupt
EMU_TRAP    B      PHANTOM ; Emulator Trap
SW_INT20    B      PHANTOM ; User S/W Interrupt
SW_INT21    B      PHANTOM ; User S/W Interrupt
SW_INT22    B      PHANTOM ; User S/W Interrupt

```

```

SW_INT23    B    PHANTOM    ; User S/W Interrupt

;=====
; M A I N    C O D E    - starts here
;=====

    .text
    NOP
START:  SETC    INTM            ;Disable interrupts
        SPLK   #0000h,IMR      ;Mask all core interrupts
        LACC   IFR            ;Read Interrupt flags
        SACL   IFR            ;Clear all interrupt flags

        CLRC   SXM            ;Clear Sign Extension Mode
        CLRC   OVM            ;Reset Overflow Mode
        CLRC   CNF            ;Config Block B0 to Data mem
        LDP    #00E0h          ;DP for addresses 7000h-707Fh
        SPLK   #00BBh,CKCR1    ;CLKIN(OSC)=10MHz, CPUCLK=20MHz
        SPLK   #00C3h,CKCR0    ;CLKMD=PLL Enable, SYSCLK =
                                ;CPUCLK/2
        SPLK   #40C0h,SYSCR    ;CLKOUT=CPUCLK

        SPLK   #006Fh, WDCR    ;Disable WD if VCCP=5V(JP52-3)
        KICK_DOG            ;Reset Watchdog

        SPLK   #0h,GPR0        ;Set wait state generator for:
        OUT    GPR0,WSGR      ;Program Space, 0 wait states
                                ;Data Space, 0 wait states
                                ;I/O Space, 0 wait states

        LARP   1              ;Select AR1 as the current AR
        LDP    #0100h          ; DP for addresses 8000h-807Fh
        LAR    AR1,#TABLEB     ;Load the starting address of
                                ;the destination table in AR1
        RPT    #COUNT        ;Perform the following
                                ;operation 10 times

```

```

BLDD #TABLEA,*+ ;Transfer data from TABLEA to
;TABLEB. After the
;instruction, pointers to data
;in both the tables are
;incremented.

END B END ;End Program

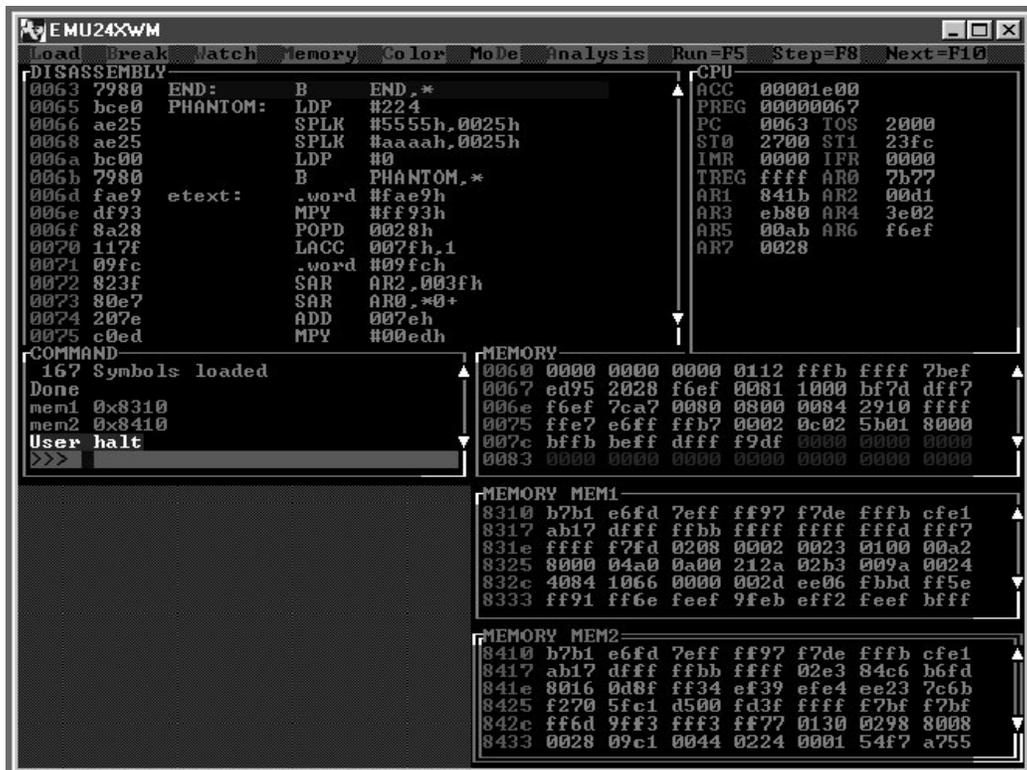
;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====

PHANTOM KICK_DOG ;Resets WD counter

B PHANTOM

```

After running the program, the debugger screen should appear as follows-



Laboratory Assignments

1. Read and run all the example programs.
2. Write a program to turn on the LEDs from DS1 to DS8 or inversely, one after the other and repeat for N ($N < 8$) times (cycles). Use DIP switches to set the value of N . If a variable TEMP=0, turn on the LEDs from DS1 to DS8, otherwise, reverse the sequence. Use assembler directives and branch instructions to control the flow.
3. Write a program to store integer vector $A=[1, \dots, 9]^T$ into data memory starting from address 8000h and integer vector $B=[9, \dots, 1]^T$ into data memory starting from address 8100h. Then compute the inner product of these two vectors ($A^T B$) and store the result in 8200h. Display the result using LEDs. (Hint: use indirect addressing mode)

CHAPTER 3

DIGITAL I/O PORTS

RELAYS & SWITCHES INTERFACING

3.1 Introduction

This chapter discusses the use of digital I/O ports in the TMS320F240 to control simple digital I/O devices such as relays and switches. The discussion covers the hardware and programming aspects of the digital I/O ports in the DSP.

Relay has been chosen as an example of electromechanical device that can be controlled from the digital output ports of a microcontroller. In this chapter, a typical circuit configuration used to interface relays to a DSP is described.

3.2 Digital I/O ports in TMS320F240

3.2.1 General purpose I/O ports

The processor has a total of 28 pins sharing primary functions and I/O that can be divided into 2 categories:

Group 1: Primary functions are shared with I/Os belonging to dedicated ports, Port A, Port B and Port C.

Group 2: Primary functions belong to peripheral modules having built-in I/O features as a secondary function (eg. SCI, SPI, external interrupt and PLL clock modules).

In this chapter we will study in detail the I/O functions of the Group 1 pins.

In order to perform I/O at any pin, the following is required:

1. Select the function of the pin (primary or I/O). This is done by the **Mux control bit**.
2. If the I/O function is selected, decide the direction of data transfer i.e. whether the pin is an input or output. This is done with the **I/O direction bit**.
3. Once the direction of the pin is selected, data has to be read from or written to the pin.

This data to be output to or input from the pin is stored in the **I/O data bit**.

All the bits discussed above are present in the I/O control registers.

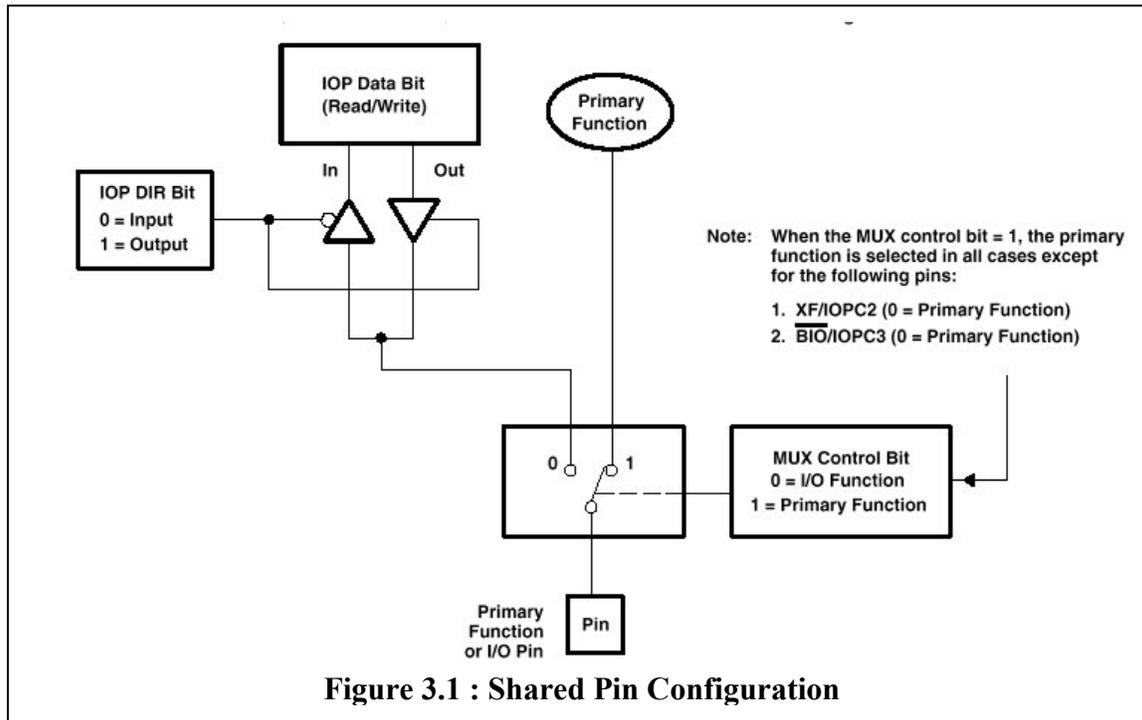


Table 1 below, gives a summary of the Group 1 pin configurations and associated bits.

PIN #	MUX CONTROL REGISTER (name.bit #)	PIN FUNCTION SELECTED		I/O PORT DATA AND DIRECTION†		
		(CRx.n = 1)	(CRx.n = 0)	REGISTER	DATA BIT #	DIR BIT #
72	OCRA.0	ADCIN0	IOPA0	PADATDIR	0	8
73	OCRA.1	ADCIN1	IOPA1	PADATDIR	1	9
90	OCRA.2	ADCIN9	IOPA2	PADATDIR	2	10
91	OCRA.3	ADCIN8	IOPA3	PADATDIR	3	11
100	OCRA.8	PWM7/CMP7	IOPB0	PBDATDIR	0	8
101	OCRA.9	PWM8/CMP8	IOPB1	PBDATDIR	1	9
102	OCRA.10	PWM9/CMP9	IOPB2	PBDATDIR	2	10
105	OCRA.11	T1PWM/T1CMP	IOPB3	PBDATDIR	3	11
106	OCRA.12	T2PWM/T2CMP	IOPB4	PBDATDIR	4	12
107	OCRA.13	T3PWM/T3CMP	IOPB5	PBDATDIR	5	13
108	OCRA.14	TMRDIR	IOPB6	PBDATDIR	6	14
109	OCRA.15	TMRCLK	IOPB7	PBDATDIR	7	15
63	OCRB.0	ADCSOC	IOPC0	PCDATDIR	0	8
64	SYSCR.7-6					
	0 0		IOPC1	PCDATDIR	1	9
	0 1		WDCLK	—	—	—
	1 0		SYSCLK	—	—	—
	1 1		CPUCLK	—	—	—
65	OCRB.2	IOPC2	XF	PCDATDIR	2	10
66	OCRB.3	IOPC3	BIO	PCDATDIR	3	11
67	OCRB.4	CAP1/QEP1	IOPC4	PCDATDIR	4	12
68	OCRB.5	CAP2/QEP2	IOPC5	PCDATDIR	5	13
69	OCRB.6	CAP3	IOPC6	PCDATDIR	6	14
70	OCRB.7	CAP4	IOPC7	PCDATDIR	7	15

† Valid only if the I/O function is selected on the pin.

Table 1: Group 1 Shared Pin Configurations

The addresses of the various I/O control registers are as below:

ADDRESS	REGISTER	NAME
7090h	OCRA	I/O mux control register A
7092h	OCRB	I/O mux control register B
7098h	PADATDIR	I/O port A data and direction register
709Ah	PBDATDIR	I/O port B data and direction register
709Ch	PCDATDIR	I/O port C data and direction register

We will now discuss each of these registers in detail.

Output Control Register A (OCRA)

15	14	13	12	11	10	9	8
OPA15	OPA14	OPA13	OPA12	OPA11	OPA10	OPA9	OPA8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
OPA7	OPA6	OPA5	OPA4	OPA3	OPA2	OPA1	OPA0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 -= value after reset

OPA15 - OPA0

0 = Set corresponding pin for general I/O

1 = Set corresponding pin for primary function.

Output Control Register B (OCRB)

15	14	13	12	11	10	9	8
OPB15	OPB14	OPB13	OPB12	OPB11	OPB10	OPB9	OPB8
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
OPB7	OPB6	OPB5	OPB4	OPB3	OPB2	OPB1	OPB0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = Read access, W = Write access, -0 -= value after reset

OPB15 - OPB0

0 = Set corresponding pin for general I/O

1 = Set corresponding pin for primary function.

Data and Direction Control Registers (PxDATDIR; x=A,B,C)

15	14	13	12	11	10	9	8
x7DIR	x6DIR	x5DIR	x4DIR	x3DIR	x2DIR	x1DIR	x0DIR
RW-0							
7	6	5	4	3	2	1	0

IOPx7	IOPx6	IOPx5	IOPx4	IOPx3	IOPx2	IOPx1	IOP0
RW-0	RW-0						

Note: R = Read access, W = Write access, -0 -= value after reset; x=ports A, B or C; n=0-7.

x7DIR-x0DIR

0 = Configure corresponding pin as an INPUT

1 = Configure corresponding pin as an OUTPUT

IOPx7-IOPx0

If xnDIR = 0, then

0 = Corresponding I/O pin is read as a LOW.

1 = Corresponding I/O pin is read as a HIGH.

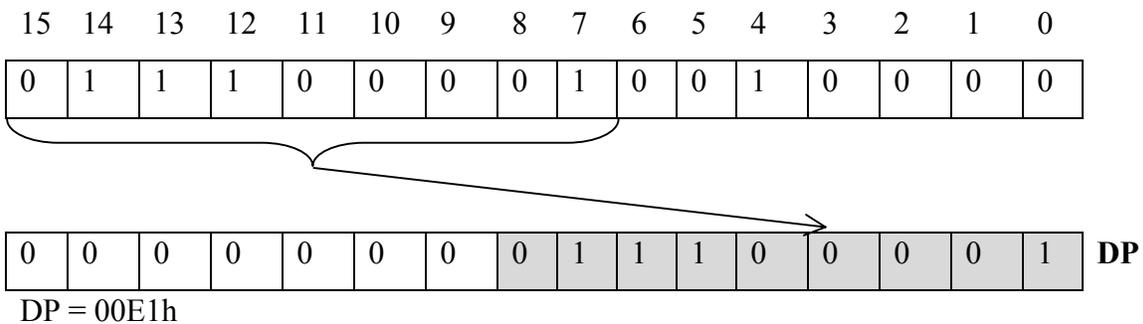
If xndir = 1, then

0 = Set corresponding I/O pin LOW.

1 = Set corresponding I/O pin HIGH.

Let us now discuss the programming involved in configuring a pin for digital I/O. As an example, let us configure bit 0 of PORT A as an input pin.

The first task is to set the corresponding bit in the mux control register to 0 in order to configure the pin for I/O. This is bit 0 of the OCRA register. The OCRA register is located at address 7090h. Thus, the value of the data pointer will be :



The file "f240regs.h" defines all the registers in the TMS320F240. Thus by including this file, the OCRA register address can be referred to as OCRA in the program. Following is the code segment for setting the OCRA register:

```

        .include f240regs.h

        LDP    #00E1h    ;Point to appropriate Data Page
        LACC  #0h        ;Pins IOPA0-IOPA3 and IOPB0-IOPB7
        SACL  OCRA      ;are configured as I/O pins
    
```

The next job is to set the pin as input in the PADATDIR register, which is done as follows-

```

        LACC  #0h
        SACL  PADATDIR  ;Pins IOPA0-IOPA3 are configured as inputs
        LACC  PADATDIR  ;Bit 0 of accumulator gives the status of the
                        ;IOPA0 pin.
    
```

Example Program 1:

Write a program to output 1010b to pins IOPA3-0.

Solution:

Registers involved -

OCRA - Bits 3-0

PADATDIR - Bits 3-0 (data)

Bits 11-8 (direction)

```

        LDP    #00E1h    ;Point to appropriate Data Page
        LACC  #0h        ;Pins IOPA0-IOPA3 are
        SACL  OCRA      ;configured as I/O pins
        LACC  #0f0ah    ;Pins IOPA3-IOPA0 are configured as outputs and
        SACL  PADATDIR  ;IOPA3, IOPA1 are set=1 & IOPA2, IOPA0 are set=0.
    
```

Example Program 2:

Write a program to do the following-

Read data from pins IOPA3-0 of port A and output the data to pins IOB3-0 of port B.

Solution:

Registers involved :

OCRA - Bits 3-0 (IOPA3-0)

Bits 11-8 (IOPB3-0)

PADATDIR - Bits 3-0 (data for IOPA3-0)

Bits 11-8 (direction for IOPA3-0)

PBDATDIR - Bits 3-0 (data for IOPB3-0)

Bits 11-8 (direction for IOPB3-0)

LDP	#00E1h	;Point to appropriate Data Page
LACC	#0h	;Pins IOPA0-IOPA3 and IOPB0-IOPB7 are
SACL	OCRA	;configured as I/O pins
LACC	#0h	
SACL	PADATDIR	;Pins IOPA3-IOPA0 are configured as inputs
LACC	PADATDIR	;Read contents of IOPA3-0 into accumulator
AND	#000Fh	;Mask the bits IOPA3-0
OR	#0F00h	;Set IOPB3-0 as outputs and
SACL	PBDATDIR	;write data read from IOPA3-0 to IOPB3-0.

3.3 Interfacing Relays and Switches

3.3.1 Electromagnetic Relays

3.3.1.1 Principles of electromagnetic relays

Figure 3.2 shows the essential parts of an electromagnetic relay and its simplified circuit diagram. The working principle of a relay is based on magnetic force produced by a current-carrying conductor. When voltage is applied to the coil terminal, current flows in the coil conductor. The current produces magnetic field, which magnetizes the magnetic core. The magnetic force produced then attracts the armature and closes the switch. At this state, the relay is said to be in energized state, otherwise it is de-energized.

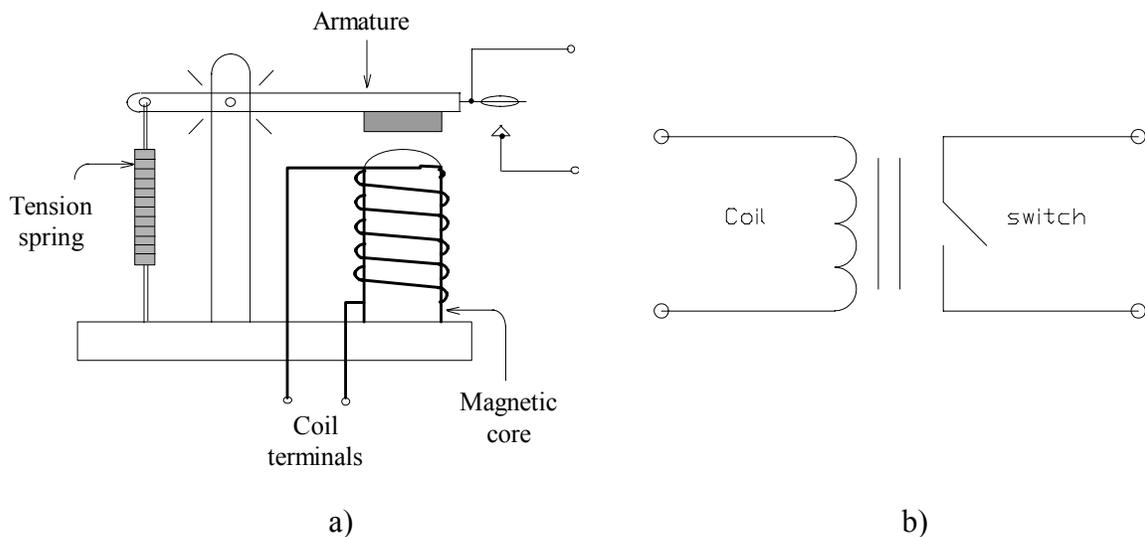


Figure 3.2 a) *Essential parts of an electromagnetic relay*
b) *Simplified circuit diagram*

Depending on the position of the switch when the relay is energized, we recognized two types of relay connections, a normally open (NO) or a normally closed (NC) connections. In a normally open connection, energizing the relay will cause the

switch to be closed, while de-energizing it will leave it open. The opposite applies to the normally closed connection. A relay can have both NO and NC connections in a single device. The schematic diagram of a relay with NO and NC connection is shown in Figure 3.3. The relay shown in this figure is also known as SPDT (*single pole dual throw*) relay, while the one shown in Figure 3.2 is usually called SPST (*single pole single throw*) relay.

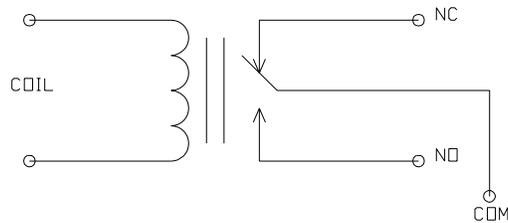


Figure 3.3 A relay with NO and NC connections (SPDT relay)

The equivalent circuit of the coil part of a relay is shown in Figure 3.4. It consists of a series connection of a resistance and an inductance. A relay usually requires a specified input voltage applied to the coil terminal. The input voltage can be of a DC or AC type. When the input voltage is DC, the current required to energize the relay coil is determined by the coil resistance. For example, a 12 V DC relay with 120 ohm coil resistance requires $12V/120ohm = 100mA$ energizing current. The current requirement of the relay coil is very important in designing circuits to drive the relay. Another important consideration in using relays is the contact resistant of its armature. This is usually specified as the maximum current it can withstand at a specified voltage (usually AC). For example, a relay with 2A, 125V AC contact rating can be used to switch a 125V AC circuit that draws current up to 10 A.

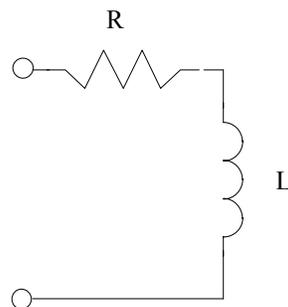


Figure 3.4 Equivalent circuit of the relay coil

3.3.1.2 Interfacing relays to the TMS320F240 EVB ports

Due to the relatively large current required to energize a relay coil, an interface circuit is needed to connect a relay to the digital I/O port of a DSP. Ports A, B and C in the TMS320F240 can only source and sink current up to a few mA while a typical relay coil usually requires current larger than these values.

Figure 3.5 shows a circuit, which will be used in the lab experiment to interface a relay to one of the TMS320F240 output port pin. The relay is a 12Vdc, 400 mA relay with a contact rating of 10A, 125V.

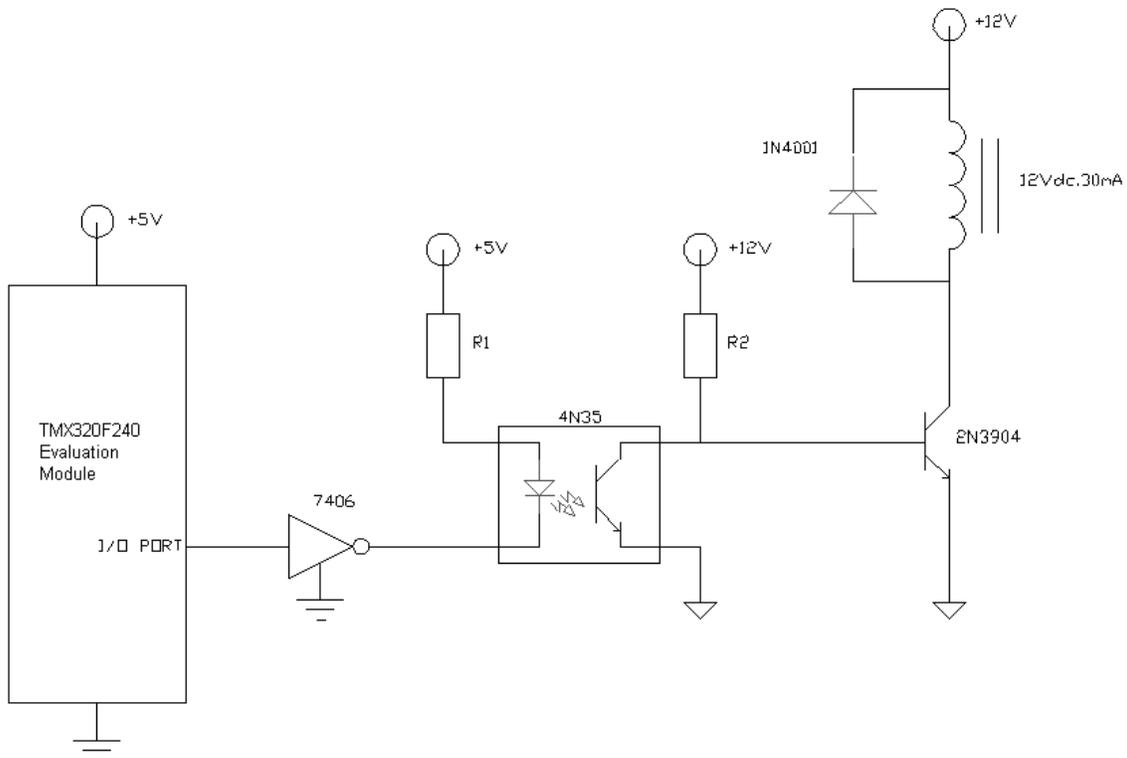


Figure 3.5 Relay interface circuit

The relay coil is energized or de-energized using an output transistor (2N3904), which has a maximum collector current of 200 mA. If the transistor is turned on, the lower end of the relay coil terminal will be pulled to the ground voltage and the relay is energized. When the transistor is turned off the relay coil will be disconnected from the supply voltage and become de-energized. The diode (1N4001) is used to protect the

transistor from a large voltage appearing at the collector of the transistor when it is turned off. This surge voltage is due to the inductive nature of the relay coil and might cause damage to the transistor. The diode avoids this to happen by providing a return path for the current in the relay coil when the transistor is turned off.

The optocoupler (4N35) shown in Figure 3.5 is used to separate the power supplies of the microcontroller and the external relay. This is desirable because of the large fluctuation in load currents found at the relay coil. This fluctuation can be coupled back to the logic power supply, causing major problem in the reliability of the DSP-based system.

The optocoupler consists of an LED and a phototransistor packaged in a light insensitive case. If the LED is turned on by applying a forward bias current, the light produced will turn on the phototransistor and pull the base of the output transistor to ground causing the output transistor to turn off. If the LED is turned off, the phototransistor will be turned off and a forward bias current will flow from the supply voltage to the base of the output transistor through resistor R2. This will cause the output transistor to turn on and energize the relay coil.

The value of the resistor R2 is chosen in order to cause the output transistor to be fully turned on when its base is forward biased. The 2N3904 has a dc current gain h_{fe} of 60 at 30 mA collector current. In order to turn on (saturate) the transistor, a minimum base current of $30mA / 60 = 0.5mA$ is required. To guarantee a full turn on of the transistor we choose a base current greater than this minimum base current, say $3mA$. The required value of resistor R2 can then be calculated by looking at Figure 3.6.

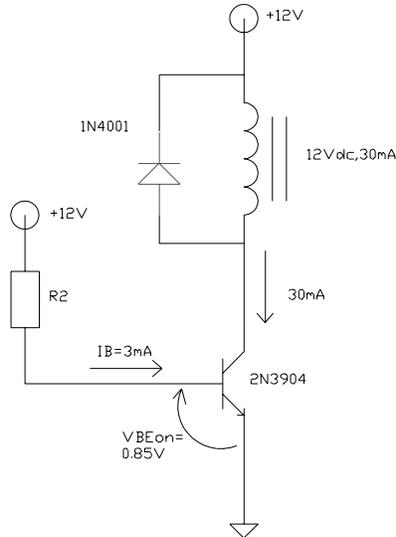


Figure 3.6 Calculating the value of R2

From Figure 3.6, R2 can be calculated as :

$$R_2 = \frac{+12 - V_{BE(on)}}{I_B} = \frac{12 - 0.85}{3mA} = 3.71k\Omega$$

We can use the closest standard value resistor of 3.3k Ω for R2.

To turn on the LED of the 4N35, a forward diode current of 10 mA is required. This current can not be supplied directly from the microcontroller port. Therefore, an inverter buffer (7406) which can sink current up to 40 mA is used to drive the LED. If the microcontroller outputs logic HIGH at the I/O port, the output of the inverter buffer is LOW. This will cause a forward current to flow through the LED of the optocoupler, causing it to turn on. Here, resistor R1 is used to limit the forward current to 10 mA. Its value can be calculated by considering Figure 3.12 below.

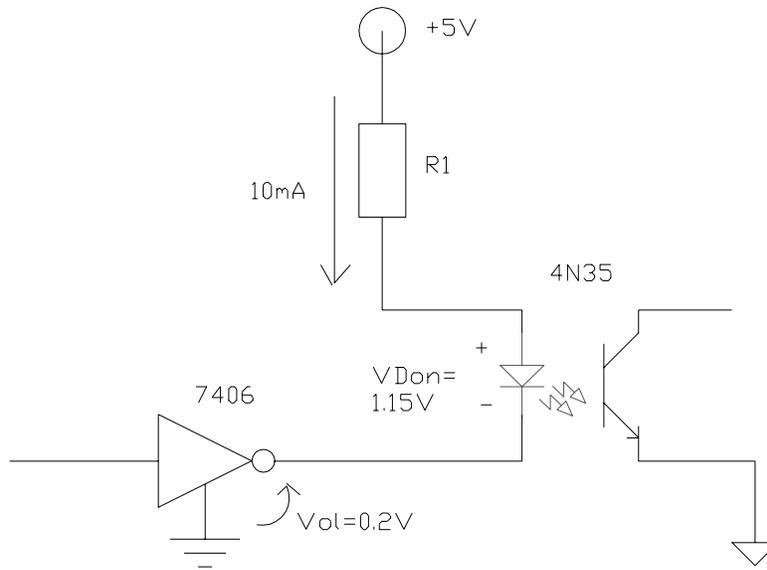


Figure 3.7. Calculating the value of R1

The value of R1 can be calculated from:

$$R_1 = \frac{+5V - V_{D(on)} - V_{ol}}{10mA} = \frac{+5V - 1.15V - 0.2V}{10mA} = 365\text{ohm}$$

A standard resistor of 330 ohm can be used for R1.

Note that the overall net effect of the interface circuit in Figure 3.2 is an inverting logic, which means that a logic 0 (LOW) at the dsp port will energize the relay, while a logic 1 (HIGH) causes the relay to be de-energized.

3.3.2 Interfacing switches to TMS320F240 EVB ports

Switches are perhaps the simplest digital input devices that can be interfaced to the DSP. Figure 3.8 shows a 4 position DIP switches connected to the digital port A of the TMS320F240.

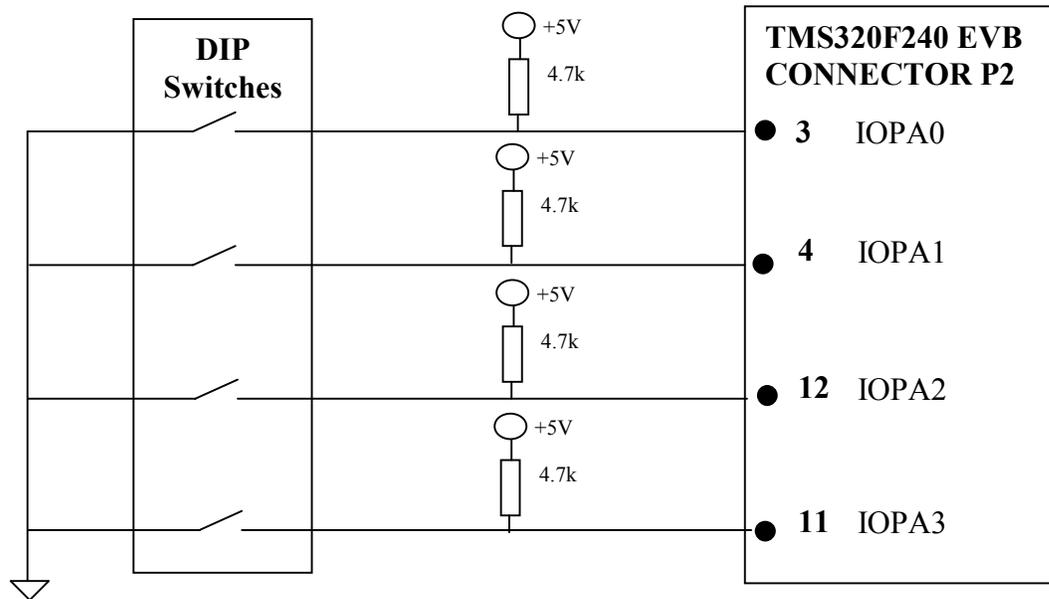


Figure 3.8 *DIP switches interface circuit*

When a switch is closed, a logic HIGH voltage (+5V) is applied to the corresponding input port; when it is opened the voltage at the input port is pulled to logic LOW by the 4.7k resistor.

3.4 Example programs for controlling relays and switches from the DSP EVB

The following are example programs that uses the general purpose I/O port to turn on 4 AC light bulbs using relays, and input digital data from a 2 position DIP switches. The relays and their interface circuits are connected to port B pin IOPB0, IOPB1, IOPB2, and IOPB3, while the DIP switches uses port A pins IOPB0, IOPB1, IOPB2 and IOPB3. The interface circuits are shown in Figure 3.9.

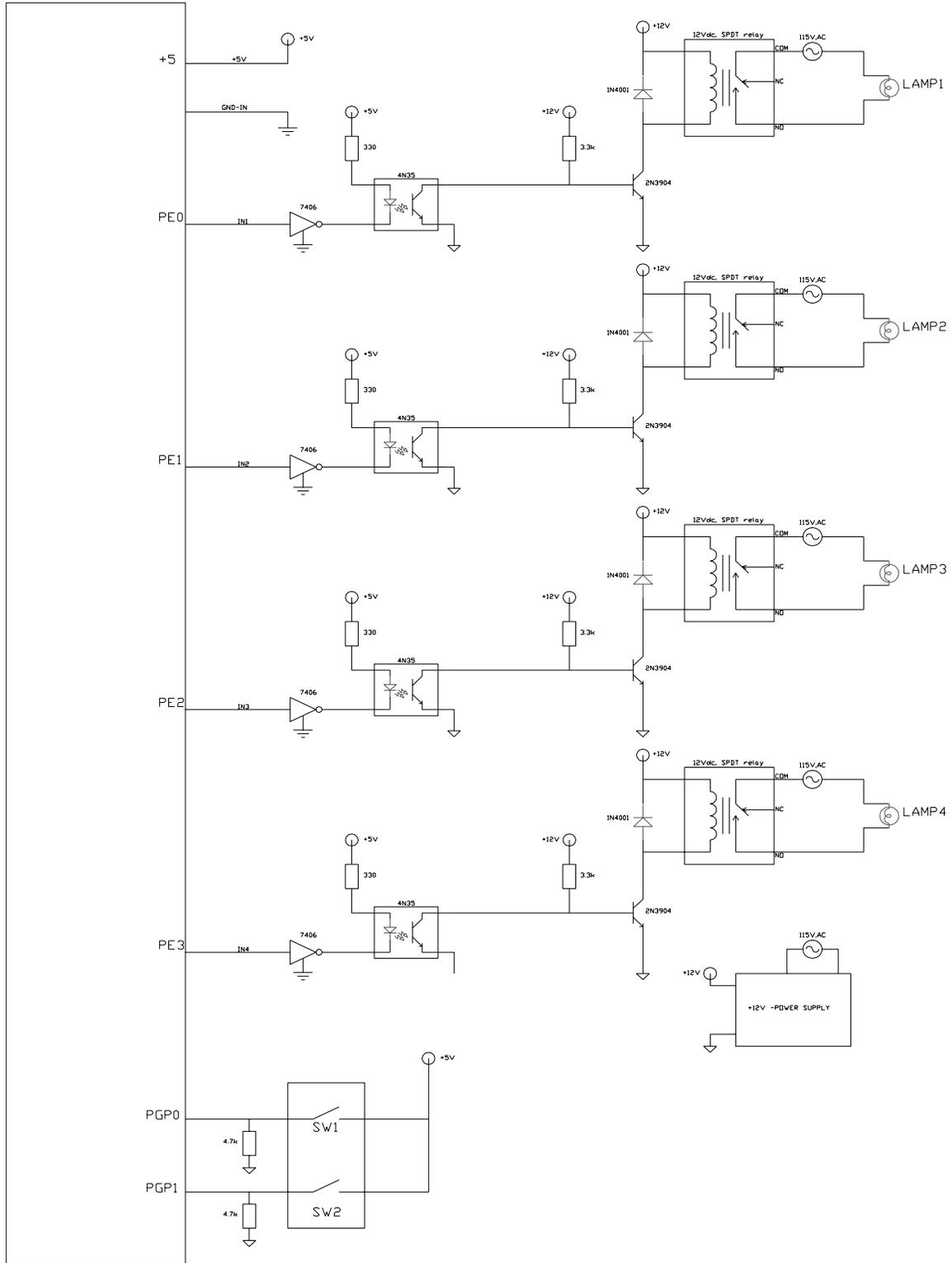


Figure 3.9 Interface circuit for example programs

Example Program 3

Write a program that reads the status of the DIP switches and accordingly turns on the lamps; i.e. if SW1 = 1, LAMP1 = OFF; SW2=1, LAMP2=OFF etc.

Let us connect the DP switches 1-4 to IOPA0-3 and LAMPS1-4 to IOPB0-3. Thus IOPA3-0 will be configured as inputs and IOPB3-0 will be configured as outputs.

Registers involved -

OCRA - Bits 3-0 (IOPA3-0)

Bits 11-8 (IOPB3-0)

PADATDIR - Bits 3-0 (data for IOPA3-0)

Bits 11-8 (direction for IOPA3-0)

PBDATDIR - Bits 3-0 (data for IOPB3-0)

Bits 11-8 (direction for IOPB3-0)

```

.include f240regs.h

LDP  #00E1h    ;Point to appropriate Data Page
LACC #0h      ;Pins IOPA0-IOPA3 and IOPB0-IOPB7
SACL OCRA     ;are configured as I/O pins
SACL PADATDIR ;Pins IOPA3-IOPA0 are configured as inputs
LACC PADATDIR ;Read status of DIP switches into accumulator
AND  #000Fh   ;Mask the bits IOPA3-0
OR   #0F00h   ;Set IOPB3-0 as outputs and
SACL PBDATDIR ;write data read from DIP switches to the lamps

```

Example Program 4

Write a program that reads the status of the DIP switches 1 and 2 and turns on/off the lamps as per the following logic.

Inputs		Outputs			
SW2	SW1	LAMP4	LAMP3	LAMP2	LAMP1
0	0	OFF	OFF	OFF	ON
0	1	OFF	OFF	ON	OFF

1	0	OFF	ON	OFF	OFF
1	1	ON	OFF	OFF	OFF

Let the switches 1 and 2 be connected to IOPA0 and IOPA1 respectively. The LAMPS1-4 be connected to IOPB0-3. The logic table therefore becomes-

IOPA1-0	IOPB3-0
00b	1110b
01b	1101b
10b	1011b
11b	0111b

Note: Logic 0 turns a lamp ON. Logic 1 turns the lamp OFF

Registers involved:

OCRA - Bits 1-0 (IOPA1-0)

Bits 9-8 (IOPB3-0)

PADATDIR - Bits 1-0 (data for IOPA1-0)

Bits 9-8 (direction for IOPA1-0)

PBDATDIR - Bits 3-0 (data for IOPB3-0)

Bits 11-8 (direction for IOPB3-0)

Here we will introduce the branch instruction. The TMS320C240 has 2 branch instructions: unconditional branch and unconditional branch.

B *pma*[, *ind*[, **AR***n*]]

Where-

pma: 16-bit program memory address

n: value from 0 to 7 designating the next auxiliary register

ind: one of the 7 indirect addressing options

The program control is unconditionally transferred to the specified pma which can be either a symbolic or numeric address.

BCND *pma, cond1* [*cond2*] [...]

Where-

pma: 16-bit program memory address

<i>Cond</i>	Condition
EQ	ACC = 0
NEQ	ACC ≠ 0
LT	ACC < 0
LEQ	ACC ≤ 0
GT	ACC > 0
GEQ	ACC ≥ 0
NC	C = 0
C	C = 1
NOV	OV = 0
OV	OV = 1
BIO	BIO low
NTC	TC = 0
TC	TC = 1
UNC	Unconditionally

The program control is unconditionally transferred to the specified pma when all the required conditions are met.

The basic logic that will be followed in this program is:

If DIP switch status = 0, then branch to location which turns on lamp 1, exit

Else if DIP switch status=2, then branch to location which turns on lamp 3, exit

Else if DIP switch status=1, then branch to location which turns on lamp 2, exit

Else, turn on lamp 4, exit.

The code is as below (the initialization and PHANTOM ISR sections should be added for practical operations)-

```
*****
; File Name:    ch3_e4.asm
; Description:  This is a program to turn the lamps as per a specific
; logic depending on the status of the two DIP switches.
;*****
        .include f240regs.h
;-----
; Variable Declarations for on chip RAM Blocks
;-----
        .bss  GPR0,1
        .bss  invar,1      ;A temporary memory location
;=====
; M A I N   C O D E   - starts here
;=====

        .text
        NOP
START:  SETC  INTM          ;Disable interrupts
        SPLK #0000h,IMR    ;Mask all core interrupts
        LACC  IFR          ;Read Interrupt flags
        SACL  IFR          ;Clear all interrupt flags

        CLRC  SXM          ;Clear Sign Extension Mode
        CLRC  OVM          ;Reset Overflow Mode
        CLRC  CNF          ;Config Block B0 to Data mem

        LDP   #00E0h       ;DP for addresses 7000h-707Fh
        SPLK #00BBh,CKCR1  ;CLKIN(OSC)=10MHZ,CPUCLK=20MHZ
        SPLK #00C3h,CKCR0  ;CLKMD=PLL Enable,SYSClk=CPUCLK/2
        SPLK #40C0h,SYSCR  ;CLKOUT=CPUCLK

        SPLK #006Fh, WDCR  ;Disable WD if VCCP=5V (JP5 in pos. 2-3)
        KICK_DOG          ;Reset Watchdog

        SPLK #0h,GPR0      ;Set wait state generator for:
        OUT   GPR0,WSGR    ;Program Space, 0 wait states
                          ;Data Space, 0 wait states
```

```

;I/O Space, 0 wait states

LDP #00E1h ;Point to appropriate Data Page
LACC #0h ;Pins IOPA0-IOPA3 and IOPB0-IOPB7
SACL OCRA ;are configured as I/O pins
SACL PADATDIR ;Pins IOPA3-IOPA0 are configured as inputs
LACC PADATDIR ;Read status of DIP switches into ACC
AND #0003h ;Mask the bits IOPA1-0
LDP #0h ;Data page points to program memory
SACL invar ;Store the switch status in memory location
BCND addr1,EQ ;If switch status = 0, goto addr1
AND #01h ;check if switch status = 2
BCND addr2,EQ ;If switch status = 2, goto addr2
LACC invar
AND #02h ;check if switch status = 1
BCND addr3,EQ ;If switch status = 1, goto addr3
LACC #7h ;Load valid lamp data for switch status=3
;in ACC

B addr4
addr1: LACC #0eh ;Load valid lamp data for switch status=0
; in ACC

B addr4
addr2: LACC #bh ;Load valid lamp data for switch status=2
;in ACC

B addr4
addr3: LACC #0dh ;Load valid lamp data for switch status=1
;in ACC

B addr4
addr4: LDP #00E1h ;Point to appropriate data page
OR #0F00h ;Set IOPB3-0 as outputs and
SACL PBDATDIR ;write appropriate data to tun on/off the
;lamps

```

Example Program 5

Write a program that turns on the four lamps sequentially from lamp 1 until lamp 4 one by one. Use software delay to keep the lamp on for approximately one second..

Solution:

In this program we will introduce the concept of a sub-routine. A delay subroutine is written to implement a delay of approximately 1 sec. The code segment for the delay subroutine is -

```

mS_DELAY:  LDP    #0h                ;DP-->0000h-007Fh
           LACC   #4000             ;Load RPT value to GPR0
           SACL  RPT_NUM
           SPLK  #5000,mSEC

mS_LOOP:   LDP    #0h                ;DP-->0000h-007Fh
           RPT   RPT_NUM            ;4000 cycles = 0.2 ms
           NOP
           LACC  mSEC               ;Load mSEC or count in ACC
           SUB   #1                 ;Decrement ACC
           SACL  mSEC               ;Update mSEC
           BCND  mS_LOOP,NEQ        ;Repeat DELAY_LOOP
           RET    ;Return from DELAY SUBROUTINE

```

The approximate delay can be computed as follows -

Since the CPU clock is 20MHz, each cycle is 50ns. The NOP instruction takes one cycle for execution i.e. it takes 50ns. The RPT instruction repeats the next instruction RPT_NUM times. Thus, the NOP is executed 2000 times which amounts to a delay of $50\text{ns} \times 4000 = 200\mu\text{s}$. As per the ms_LOOP, this is executed mSEC i.e. 5000 times which gives a total delay of $200\mu\text{s} \times 5000 = 1 \text{ sec}$.

The code for the entire program is as follows (the initialization and PHANTOM ISR sections should be added for practical operations)-

```

;*****
; File Name:      ch3_e5.asm
; Description:    This is a program to turn the lamps one after the other
;                for 1 second.
;*****
        .include      f240regs.h
;-----

```

```

; Variable Declarations for on chip RAM Blocks
;-----
        .bss      mSEC,1
        .bss      RPT_NUM,1
        .bss      GPR0,1
;=====
; M A I N   C O D E   - starts here
;=====
        .text
        NOP

START:  SETC INTM          ;Disable interrupts
        SPLK #0000h,IMR    ;Mask all core interrupts
        LACC IFR          ;Read Interrupt flags
        SACL IFR          ;Clear all interrupt flags

        CLRC SXM          ;Clear Sign Extension Mode
        CLRC OVM          ;Reset Overflow Mode
        CLRC CNF          ;Config Block B0 to Data mem

        LDP   #00E0h       ;DP for addresses 7000h-707Fh
        SPLK #00BBh,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
        SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2
        SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK

        SPLK #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 in pos. 2-3)
        KICK_DOG          ;Reset Watchdog

        SPLK #0h,GPR0     ;Set wait state generator for:
        OUT   GPR0,WSGR   ;Program Space, 0 wait states
                           ;I/O Space, 0 wait states

LAMP1:  LDP   #00E1h       ;Point to appropriate Data Page
        LACC #0h          ;Pins IOPA0-IOPA3 and IOPB0-IOPB7
        SACL OCRA         ;are configured as I/O pins
        LACC #0F0eh       ;Configure IOPB3-0 as outputs and turn
        SACL PBDATDIR     ;LAMP1 ON.
        CALL mS_DELAY     ;1 sec delay

        LDP   #00E1h       ;point to appropriate data page. this is very
                           ;important since the delay sub-routine
                           ;changes the value of DP
        LACC #0F0dh       ;Configure IOPB3-0 as outputs and turn

```

```

SACL PBDATDIR      ;LAMP2 ON.
CALL mS_DELAY      ;1 sec delay

LDP #00E1h         ;point to appropriate data page.
LACC #0F0bh        ;Configure IOPB3-0 as outputs and turn
SACL PBDATDIR      ;LAMP3 ON.
CALL mS_DELAY      ;1 sec delay

LDP #00E1h         ;point to appropriate data page.
LACC #0F07h        ;Configure IOPB3-0 as outputs and turn
SACL PBDATDIR      ;LAMP4 ON.
CALL mS_DELAY      ;1 sec delay
B    LAMP1

;=====
; Routine Name:  mS_DELAY
; Description:   Produces a delay of approximately 1 sec.
; Calling Convention:
; Variables      on Entry      on Exit
; -----
;  DP            XX            0x0000
; -----
;=====

mS_DELAY: LDP #0h          ;DP-->0000h-007Fh
          LACC #4000       ;Load RPT value to GPR0
          SACL RPT_NUM
          SPLK #5000,mSEC

mS_LOOP:  LDP #0h          ;DP-->0000h-007Fh
          RPT RPT_NUM      ;4000 cycles = 0.2 ms
          NOP
          LACC mSEC        ;Load mSEC or count in ACC
          SUB #1           ;Decrement ACC
          SACL mSEC        ;Update mSEC
          BCND mS_LOOP,NEQ ;Repeat DELAY_LOOP
          RET              ;Return from DELAY SUBROUTINE

```

LABORATORY EXPERIMENT 2

DIGITAL INTERFACING: RELAYS AND SWITCHES

Objectives

The objective of this lab session is to familiarize the students with the DSP digital I/O interfacing. In this session, the students will write and test assembly language programs that use TMS320F240 general-purpose I/O ports to turn on AC light bulbs using relay circuits and read digital data from DIP switches.

Equipment Required

Hardware :

- PC Specifications -
 - ❑ '386 or higher IBM PC/AT
 - ❑ 1.44Mb 3.5-inch floppy drive
 - ❑ 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
 - ❑ Minimum 4Mb memory
 - ❑ Color VGA Monitor

- TMS320C24x Evaluation Board
- XDS510PP Emulator
- +5V power supply.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable
- Optocoupler interface circuit board.
- Relays and Lamp Box

- DIP switches circuit board

Software :

- MS Windows 3.1 or Windows 95
- ASCII Editor
- TMS320C2xx Assembler
- TMS320C2xx C Source Debugger

Discussion

General purpose digital I/O ports in the TMS320F240

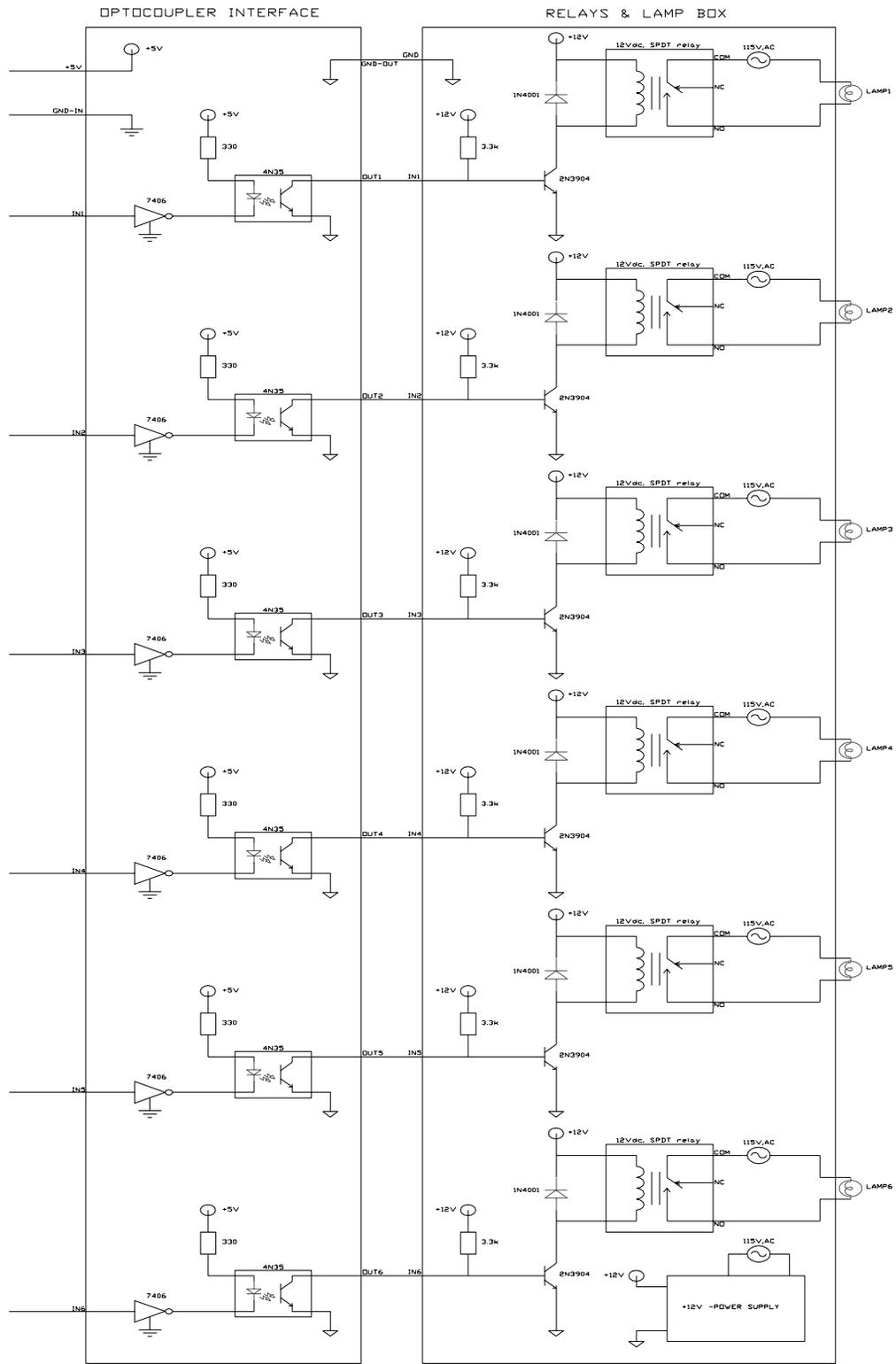
The Relays Interface Circuit

In the lab experiment you will utilize the general-purpose I/O ports of the TMS320F240 EVB to turn on or off AC light bulbs using relays. The interface circuit is shown in Lab Figure 3.1. The circuit is built of two modules i.e. the *optocoupler interface circuit board* and the *relays & lamps box*.

The *optocoupler interface circuit board* contains a hex inverter buffers (7406) and six optocouplers (4N35). The optocouplers are used to separate the power supplies of the microcontroller and the relays. This separation is necessary in order to avoid the large load current fluctuation in the relays circuit being coupled back to the logic power supply.

The *relays & lamps box* encloses the relays and lamps circuitry. The relays used here are 12 Vdc, 30 mA SPDT relays with contact ratings of 10A, 125Vac. The lamps are 25 watts, 115 Vac colored light bulbs. These lamps are connected to the NO (normally open) terminal of the relay switches. The box is powered from the 115Vac source, which is used to supply the lamps and a 12Vdc-power supply.

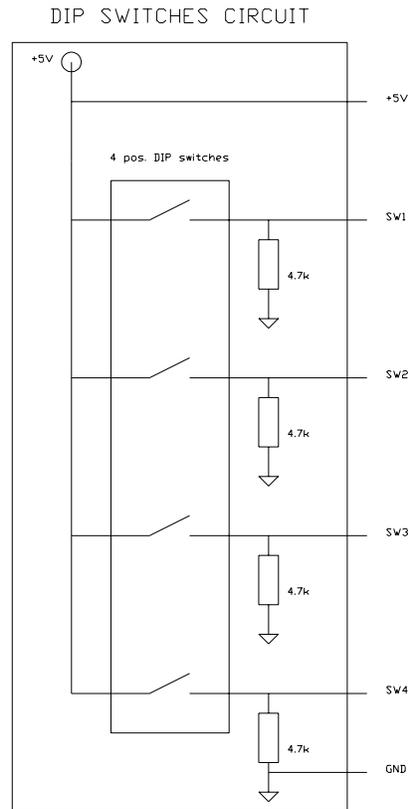
Note that the overall net effect of the interface circuit shown in Lab Figure 3.1 is an inverting logic. This means that a logic 0 (LOW) at the input of the optocoupler interface circuit will energize the relay, while a logic 1 (HIGH) cause the relay to be de-energized.



Lab Figure 3.1 *Relays interface circuit*

DIP switches interface circuit

Lab Figure 3.2 shows the circuit that will be used to interface a 4 position DIP switches to the input port of the DSP EVB.



Lab Figure 3.2 DIP Switches Interface Circuit Board

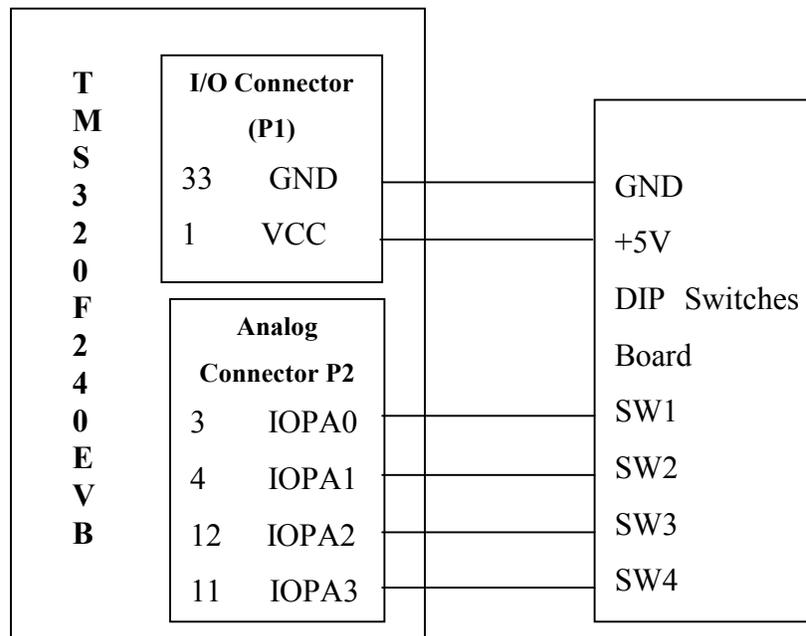
Closing a switch causes the voltage at the corresponding terminal (SW1-SW4) to be pulled to logic HIGH, while opening a switch causes the terminal to be at logic LOW.

Procedure

Setup

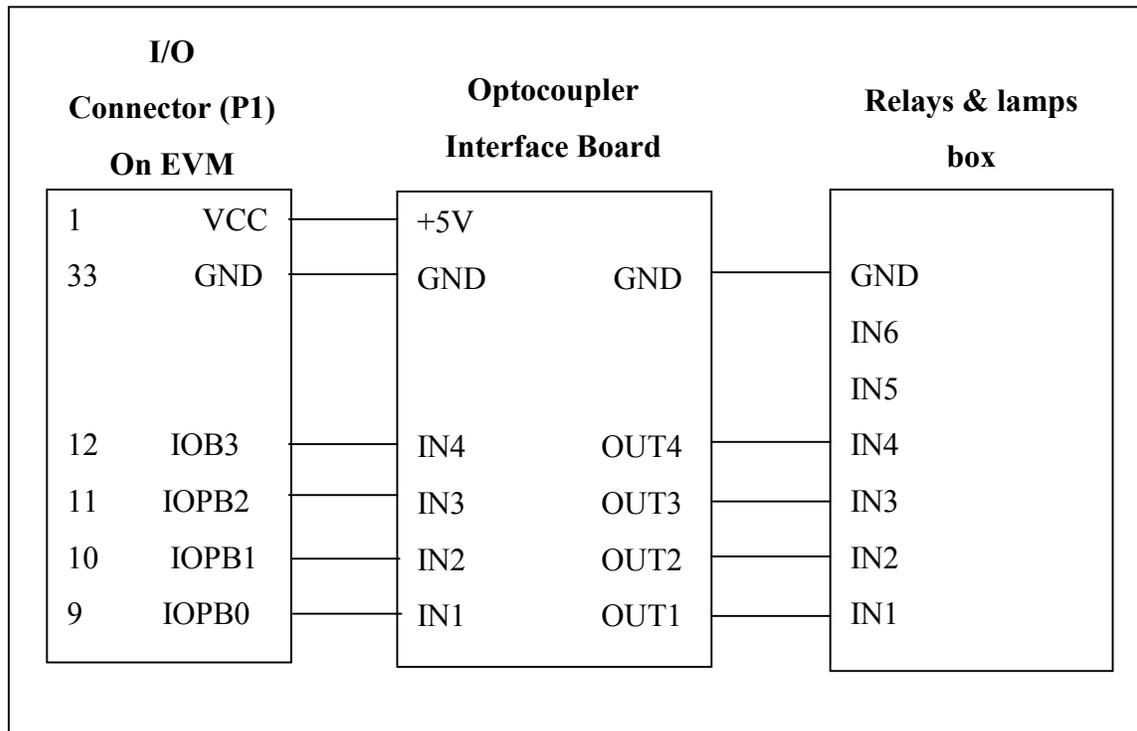
1. Setup the EVB and PC as discussed in LAB1.
2. Connect the DIP switches circuit board to the EVB terminal Blocks as shown in Lab Figure 3.3. This will connect the DIP switches to the Port A pins IOPA3:0 to SW4-1.

Note: The EVB connector details are attached at the end of this lab session.



Lab Figure 3.3 DIP switches board & EVB interconnection diagram

3. Connect the relays interface circuit (optocoupler circuit board and relays & lamps box) as shown in Lab Figure 3.4. These interconnections cause the relays to be interfaced to the port B pins IOPB5-0.



Lab Figure 3.4 Relays interface circuits & EVB interconnection diagram

4. Show all the interconnections you made in step 1, 2 and 3 to the TA. If the TA agrees, you can proceed to the next step.
5. Turn on the PC and wait until the MS-Windows has started.
6. Power on the EVB power supply and run the EVB Testing program in PC to check the proper operation of the EVB and its connection to PC. If the test fails, turn off the EVB power supply, quit MS-Windows, and turn off the PC. Check the cable connections from the EVB to the PC and repeat step 5 and 6. If the problem persists, report to the lab TA. If the test succeeds, you are ready to use the EVB and proceed to the next procedure.
7. Power on the relays and lamps box by connecting its power cord to the AC outlet. If no problem exists, you should see all the lamps are on. Otherwise, check all the connections you have made. If the problem persists, report to the TA.

Laboratory Assignments

1. Write a program to that reads data from the DIP switches, inverts the data and outputs it to the lamps i.e. if SW1=0, LAMP1=OFF etc. Connect the DIP switches to IOPA3-0. Connect lamps 1-4 to IOPB0-3.
2. Write a program that will read the status of the switches and turn on the lamps as per the following logic.

Inputs		Outputs					
SW2	SW1	LAMP6	LAMP5	LAMP4	LAMP3	LAMP2	LAMP1
0	0	OFF	OFF	OFF	OFF	ON	ON
0	1	OFF	OFF	OFF	ON	ON	OFF
1	0	OFF	ON	ON	OFF	OFF	OFF
1	1	ON	ON	OFF	OFF	OFF	OFF

3. Write a program that turns on the six lamps **one by one** according to the following sequence:

1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, ...

which repeats 'forever'. Use software delay to keep each lamp on for approximately two seconds.

4. Write a program that changes the pattern of the lamps every about five seconds, from pattern #1 until pattern #4, then repeats from pattern #1 and continues forever. The patterns are:

Pattern	LAMP6	LAMP5	LAMP4	LAMP3	LAMP2	LAMP1
#1	ON	ON	OFF	ON	OFF	ON
#2	OFF	ON	OFF	OFF	ON	ON
#3	ON	OFF	ON	OFF	OFF	OFF
#4	ON	ON	OFF	OFF	ON	ON

P1

V _{CC}	■ 1	2 ●	V _{CC}
PWM1/CMP1	◆ 3	4 ◆	PWM2/CMP2
PWM3/CMP3	◆ 5	6 ◆	PWM4/CMP4
PWM5/CMP5	◆ 7	8 ◆	PWM6/CMP6
PWM7/CMP7/IOPB0	◆ 9	10 ◆	PWM8/CMP8/IOPB1
PWM9/CMP9/IOPB2	◆ 11	12 ◆	T1PWM/T1CMP/IOPB3
T2PWM/T2CMP/IOPB4	■ 13	14 ●	T3PWM/T3CMP/IOPB5
TMRDIR/IOPB6	◆ 15	16 ◆	TMRCLK/IOPB7
GND	◆ 17	18 ◆	GND
XF/IOPC2	■ 19	20 ◆	<u>BIO</u> /IOPC5
CAP1/QEP1/IOPC4	■ 21	22 ●	CAP2/QEP2/IOPC5
CAP3/IOPC6	■ 23	24 ●	CAP4/QEP4/IOPC6
Reserved	◆ 25	26 ◆	<u>PDPINT</u>
SCITXD/IO	◆ 27	28 ◆	SCIRXD/IO
SPISIMO/IO	◆ 29	30 ◆	SPISOMI/IO
SPICLK/IO	■ 31	32 ●	SPISTE/IO
GND	◆ 33	34 ◆	GND

P2

V_{CCA}	● 1	2 ■	V_{CCA}
ADCIN0/IOPA0	● 3	4 ■	ADCIN1/IOPA1
ADCIN2	● 5	6 ■	ADCIN3
ADCIN4	● 7	8 ■	ADCIN5
ADCIN6	● 9	10 ■	ADCIN7
ADCIN8/IOPA3	● 11	12 ■	ADCIN9/IOPA2
ADCIN10	● 13	14 ■	ADCIN11
ADCIN12	● 15	16 ■	ADCIN13
GNDA	■ 17	18 ▼	GNDA
ADCIN14	● 19	20 ■	ADCIN15
$V_{ref\ hi}$	● 21	22 ■	$V_{ref\ lo}$
GNDA	● 23	24 ■	GNDA
DACOUT0	● 25	26 ▼	DACOUT1
DACOUT2	■ 27	28 ■	DACOUT3
Reserved	■ 29	30 ▼	Reserved
Reserved	● 31	32 ■	ADCSOC/IOPC
GNDA	● 33	34 ■	GNDA

CHAPTER 4

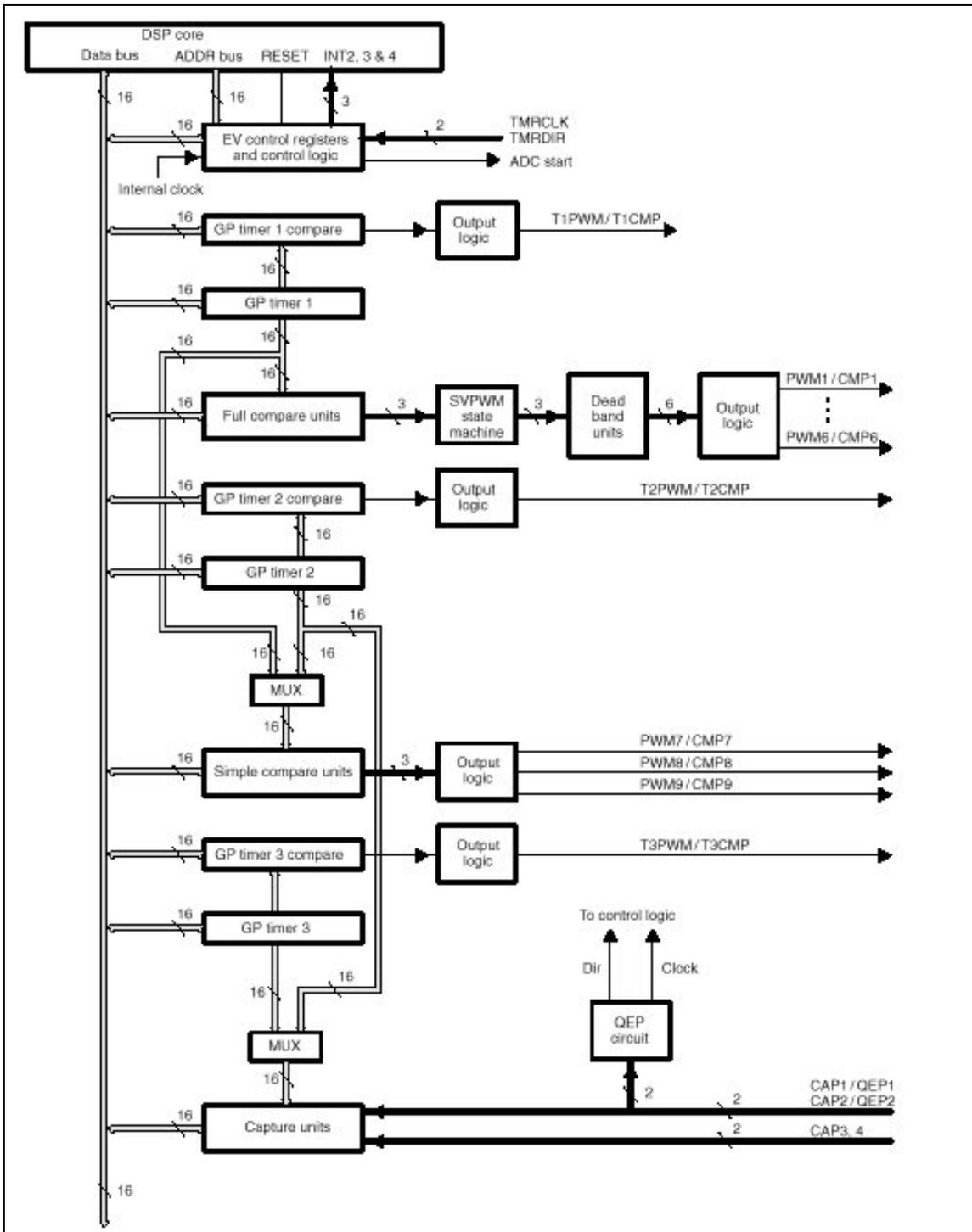
Timer and Interrupt Operations

4.1 Introduction

In this chapter we will explore the Event Manager (EV) module of the TMS320F240, which provides numerous features applicable in motion control. The EV comprises the following functional blocks:

- Three general-purpose timers (GPT).
- Three full compare units
- Three simple compare units
- Pulse-width modulation (PWM) circuits
- Four capture units
- Quadrature encoder pulse (QEP) circuit
- Interrupt logic

The block diagram is shown in Figure 4.1 below:



Ref: Texas Instruments Literature # SPRU160A, March 1997

Figure 4.1: Event Manager (EV) Block Diagram

The *GP timers* (general-purpose timers) provide the ability to make precise control decisions in real time. The *compare, capture* and *PWM* (pulse-width modulators)

units are very useful in motor control and electromechanical applications. We will discuss the principle of operation and programming of each of the units in detail in this chapter. There are numerous applications of the timer function. The simplest application that will be discussed here is to generate time delay for variety of purposes. We will also examine two applications of the timer units, which are very important in electromechanical system. One application is to generate signals with various waveforms, such as square wave signals with different frequency and pulse width (PWM signals). Another application is to measure pulse widths and related variables.

In order to utilize the timer units of the GPT, we need first to understand one important mechanism in the processor called *interrupts*. An interrupt is a hardware or software driven signal that can cause the processor to suspend its present execution sequence and execute another subroutine called an *Interrupt Service Routine*, or *interrupt handler*.

4.2 Interrupts

4.2.1 What is an interrupt?

An interrupt is a signal that tells the CPU to suspend the execution of an application and causes it to be switched to an appropriate subroutine called interrupt service routine or interrupt handler. When interrupt occurs, the CPU saves the state of the machine and jumps to the interrupt service routine. After the interrupt handler has completed its task, the CPU can continue executing the application programs from where it is left off. Figure 4.2 illustrates how the program flows when an interrupt occurs. In this figure, an interrupt occurs when the CPU is executing instruction #3 in the main part of the application program. The CPU finishes this instruction, save the state of the machine and jumps to the interrupt service routine. The CPU then starts executing instructions in the interrupt service routine (instruction #1a -6a) until it encounters the RET (return) instruction. This instruction causes the CPU to return to the main program and continue executing instruction #4, #5 and so on.

4.2.3 How does the processor handle interrupts?

The interrupt operation can be summarized in the following three main phases:

1. **Receive interrupt request.** The suspension of the main program is requested by software (program code) or hardware (a pin or an on-chip peripheral)
2. **Acknowledge interrupt.** Once an interrupt is received, the processor checks whether it is maskable. If this is the case, it checks certain conditions (which are discussed later) and then sends an acknowledgement. In case of non-maskable interrupts, the processor acknowledges the interrupt immediately.
3. **Execute the ISR.** After acknowledging the interrupt, the processor branches to the appropriate vector address location where it executes the ISR.

4.2.3 How does the CPU know which interrupt service routine to execute?

Interrupt Vectors

The interrupt vectors direct the CPU to jump to the intended interrupt service routines each time an interrupt occurs. *An interrupt vector is a 2-byte data that represents the start address of an interrupt service routine. Interrupt vectors in the TMS320F240 are assigned numbers from 0h through 3Eh.* These interrupt vectors are stored in a specific block of memory called the **interrupt vector table** starting at address 0000h.

When a device sends an interrupt signal to the CPU, it also sends the vector number to identify the interrupt. According to the vector number received, the CPU fetches the interrupt vector at the corresponding vector address. The CPU then executes the interrupt service routine which starts at the address specified by the interrupt vector. Using this mechanism, the CPU is able to identify the type of interrupt that occurs and execute the correct interrupt service routine.

Note that it is the responsibility of the programmer to set up the interrupt vector table prior to using an interrupt. If one wishes to use an interrupt from a device, one must find out the vector number corresponding to that interrupt and fill in the interrupt vector at the appropriate location in the vector table. As an example, suppose that the interrupting device generates an interrupt with a vector number 0x50 (80 decimal). And suppose also that the interrupt service routine is placed at address 0x1234 in memory so

that the interrupt vector in this case is 0x1234. The vector number 0x50 has a vector address 0x00A0 (twice 0x50) in the interrupt vector table. Therefore, memory location 0x00A0 and 0x00A1 have to be filled in with 0x12 (high byte of the interrupt vector) and 0x34 (lower byte of the interrupt vector table). This setup is illustrated in Figure 4.3 below.

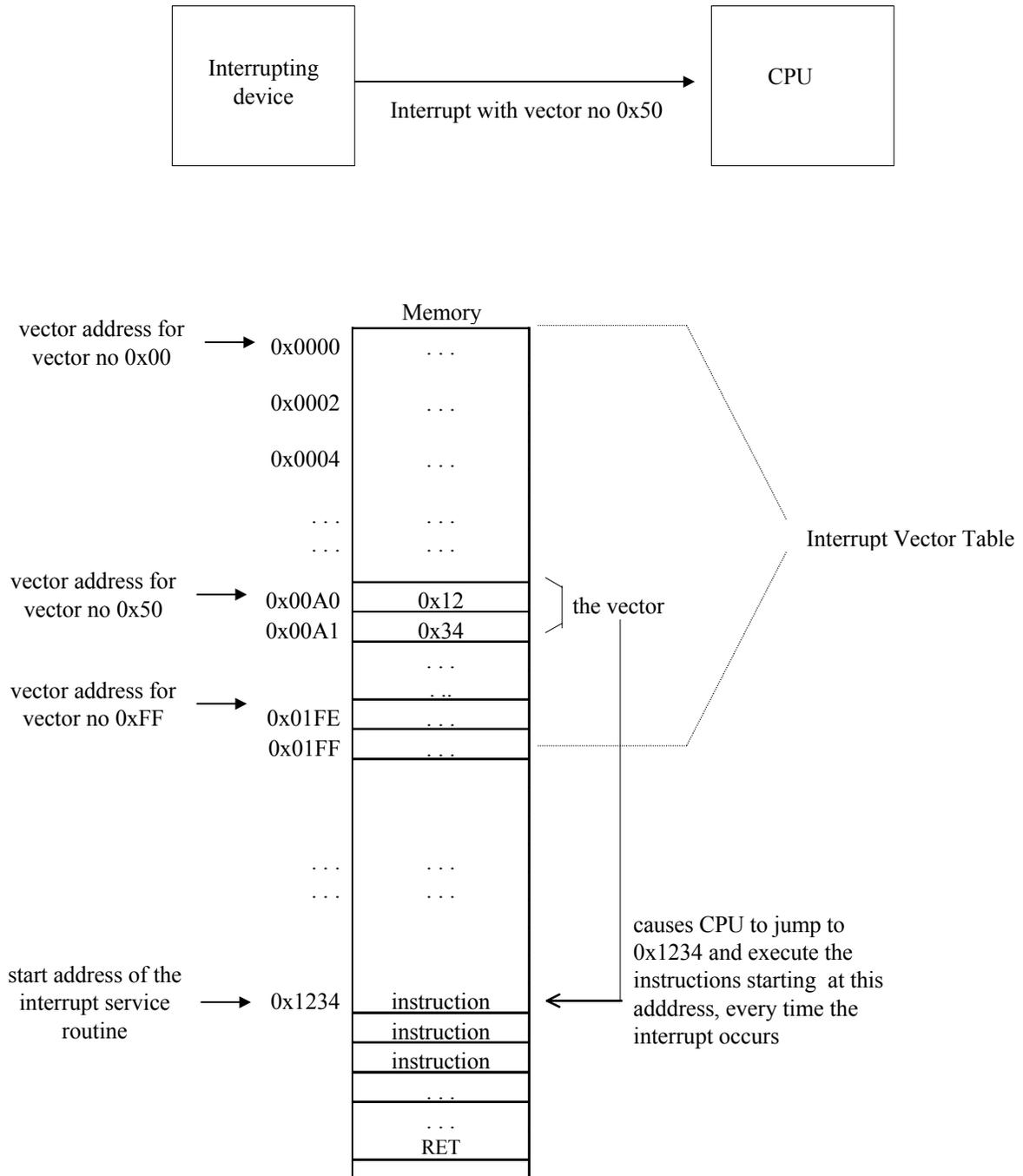


Figure 4.3: Interrupt Vectoring example

Executing the ISR for a Non-maskable interrupt:

After acknowledging a non-maskable interrupt, the CPU:

1. Stores the program counter (PC) value i.e. the return address to the top of the hardware stack.
2. Loads the PC with the address of the interrupt vector.
3. Fetches the branch instruction that is stored at the vector location by the programmer. If the interrupt was a hardware interrupt or was requested either by the INTR or NMI instructions, the CPU also sets the INTM bit to 1 to disable the maskable interrupts.
4. Execute the branch, which leads it to the address of the ISR.
5. Executes the ISR until a return instruction concludes the ISR.
If INTM is 1, all maskable interrupts are disabled during the execution of the ISR.
6. Pops the return address off the stack into the PC.
7. Continues executing the main program.

Ref: Texas Instruments Literature # SPRU160A, March 1997

Interrupt Priority

All hardware interrupts are given a priority rank from 1 to 10 (1 being the highest priority). When more than one hardware interrupt occurs at the same time, the interrupt with the highest priority gets serviced first. The other interrupts are serviced after that sequentially in the order of their priority. The interrupt priorities are summarized in table 4.1 below:

K	Vector Location	Name	Priority	Function
0	0h	RS	1	Hardware reset (non maskable)
1	2h	INT1	4	Maskable interrupt level #1
2	4h	INT2	5	Maskable interrupt level #2
3	6h	INT3	6	Maskable interrupt level #3
4	8h	INT4	7	Maskable interrupt level #4

5	0Ah	INT5	8	Maskable interrupt level #5
6	0Ch	INT6	9	Maskable interrupt level #6
7	0Eh		10	Reserved
8	10h	INT8	-	User defined software interrupt
9	12h	INT9	-	User defined software interrupt
10	14h	INT10	-	User defined software interrupt
11	16h	INT11	-	User defined software interrupt
12	18h	INT12	-	User defined software interrupt
13	1Ah	INT13	-	User defined software interrupt
14	1Ch	INT14	-	User defined software interrupt
15	1Eh	INT15	-	User defined software interrupt
16	20h	INT16	-	User defined software interrupt
17	22h	TRAP	-	TRAP instruction vector
18	24h	NMI	3	Non maskable interrupt
19	26h		2	Reserved
20	28h	INT20	-	User defined software interrupt
21	2Ah	INT21	-	User defined software interrupt
22	2Ch	INT22	-	User defined software interrupt
23	2Eh	INT23	-	User defined software interrupt
24	30h	INT24	-	User defined software interrupt
25	32h	INT25	-	User defined software interrupt
26	34h	INT26	-	User defined software interrupt
27	36h	INT27	-	User defined software interrupt
28	38h	INT28	-	User defined software interrupt
29	3Ah	INT29	-	User defined software interrupt
30	3Ch	INT30	-	User defined software interrupt
31	3Eh	INT31	-	User defined software interrupt

Ref: Texas Instruments Literature # SPRU160A, March 1997

From the table we can observe that the RESET interrupt has the highest priority. It stops the program flow, returns the processor to a pre-determined state and begins program execution from address 0000h. This is the *hardware reset*.

Path of a Maskable Interrupt Request:

As mentioned earlier, software and non-maskable interrupts are acknowledged immediately. In case of maskable interrupts, the following conditions have to be met:

- The interrupt with the highest priority is serviced first.
- The interrupt mode (INTM) bit, bit 9 of status register STO is set. This is to avoid any maskable interrupts during execution of the ISR.
 - INTM = 0, all unmasked interrupts are enabled.
 - INTM = 1, all maskable interrupts are disabled.
- Set IMR mask bit to 1. Each maskable interrupt level has a mask bit in the IMR (interrupt mask register). To unmask an interrupt level, its IMR bit is set to 1.

Let us take a short detour here to discuss the CPU interrupt registers:

The 2 CPU registers for controlling interrupts are:

- The interrupt flag register (IFR) which contains flag bits that indication of the maskable interrupt that have reached the CPU on level 1 to 6.
- The interrupt mask register (IMR) which contains mask bits to enable/disable each of the interrupt levels INT1 through 6.

Interrupt Flag Register (IFR) - Address 0006h

15-6	5	4	3	2	1	0
Reserved	INT6	INT5	INT4	INT3	INT2	INT1
0	RW1C -0	RW1C-0				

Note: 0 = Always read as zeros, R = Read access, W1C = Write 1 to this bit to clear it, -0 = value after reset

Bits 15-6 Reserved. These bits are always read as 0s.

Bit n INT x. Interrupt x flag. Where x = 1 to 6: interrupt level.

(n = 0to5) 0 = No INT x interrupt is pending.

1 = At least one INT x interrupt is pending. *Write a 1 to this bit to clear it to 0 and clear the interrupt request.*

Interrupt Mask Register (IMR) - Address 0004h

15-6	5	4	3	2	1	0
Reserved	INT6	INT5	INT4	INT3	INT2	INT1
0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: 0 = Always read as zeros, R = Read access, W = Write access, -0 = value after reset

Bits 15-6 Reserved. These bits are always read as 0s.

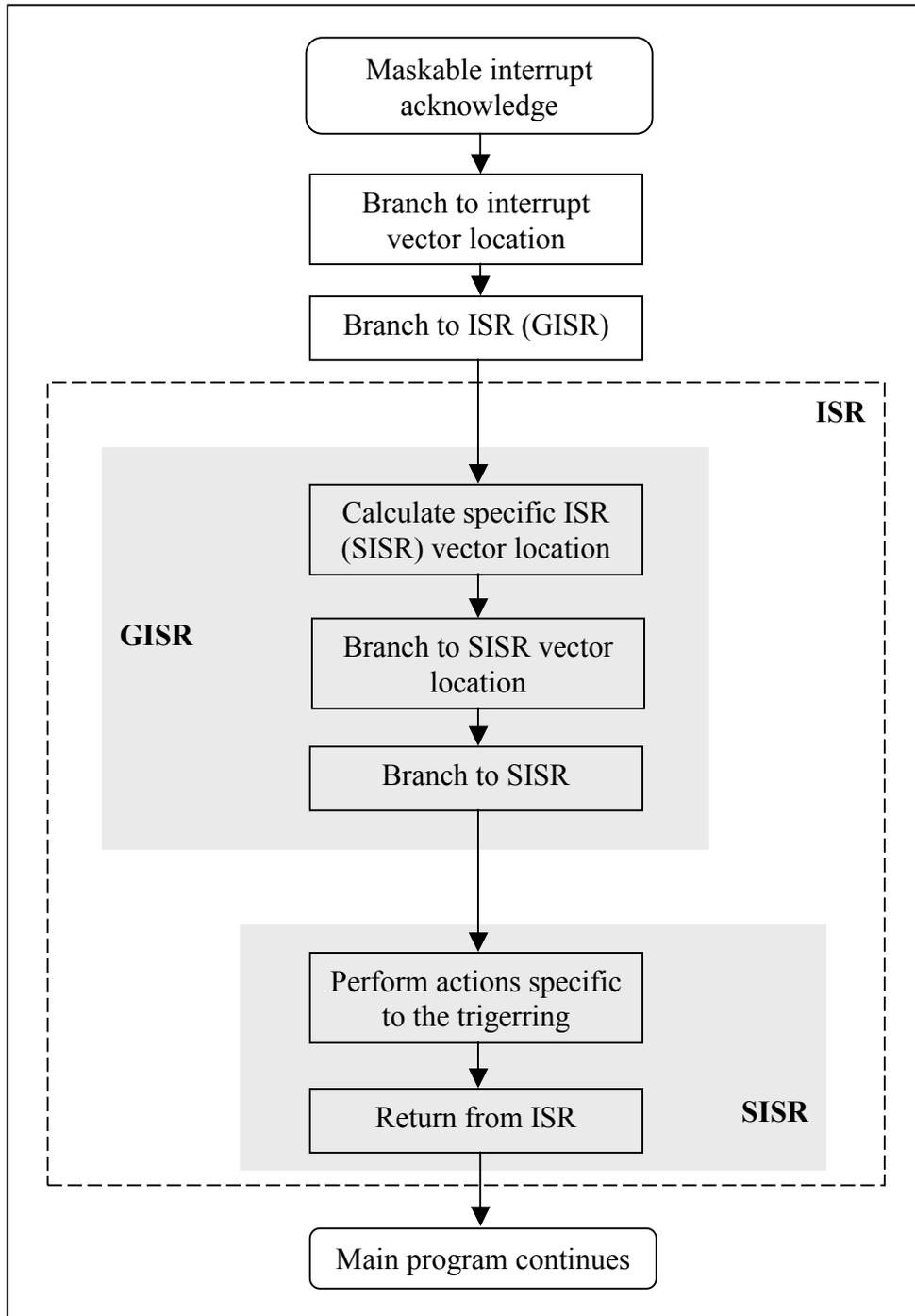
Bit n INT x. Interrupt x flag. Where x = 1 to 6 : interrupt level.

(n = 0to5) 0 = Level INT x is masked.

1 = Level INT x is unmasked.

Interrupt Service Routine for a Maskable Interrupt

Each of the 6-interrupt levels has associated with it a specific general interrupt service routine (GISR). For example, INT1 has associated with it GISR1. Thus, the CPU first branches to the appropriate GISR, which performs the common functions such as context saves etc. After this, the GISR identifies and branches to the specific interrupt service routine (SISR). The SISR performs functions specific to the triggering interrupt and then returns program control to the main program. This is represented in the flowchart in figure below:



Ref: Texas Instruments Literature # SPRU160A, March 1997

Figure 4.4: Interrupt Subroutine Flowchart

How is the SISR address generated?

In response to an interrupt acknowledge, a peripheral generates a vector address offset that corresponds to the particular interrupt event. This vector is latched in the

system interrupt vector register (SYSIVR) in most cases, while some peripherals have store it in their register. The details of the location of the interrupt vector register (IVR) will be discussed shortly. The GISR reads the value stored in the IVR and used it to generate the branch target address for the SISR.

Location of IVR:

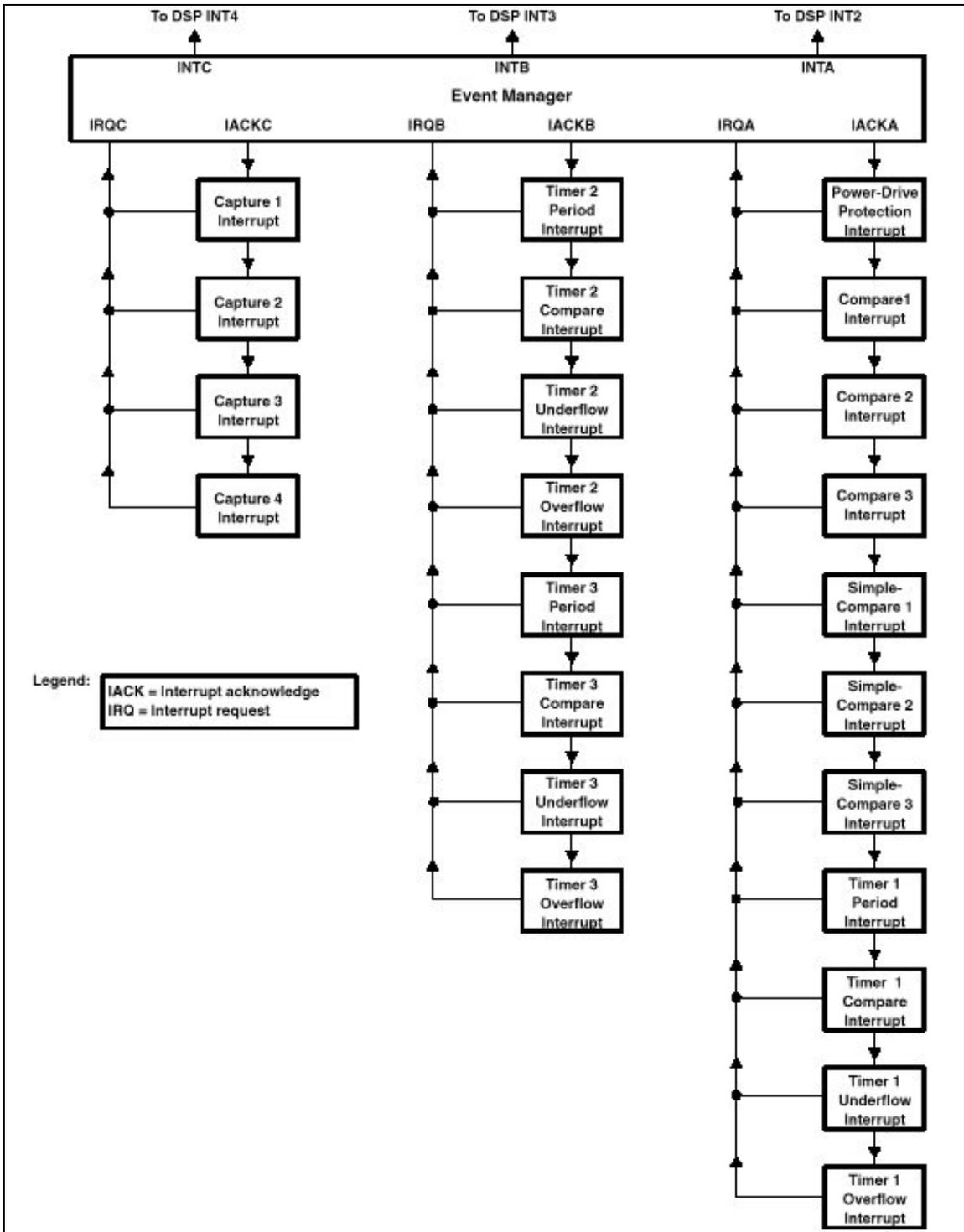
Control of inputs to all the maskable interrupt lines is distributed between the two peripheral control modules as follows:

PERIPHERAL	INTERRUPT LINES
System Module	INT1
	INT5
	INT6
Event manager	INT2
	INT3
	INT4

At each level of the system module and the event manager, each of the maskable interrupt lines (INT1-INT6) is connected to multiple maskable interrupt sources. Each of these sources has a priority ranking. These interrupts are also classified as type A, B and C. The features of each of these are -

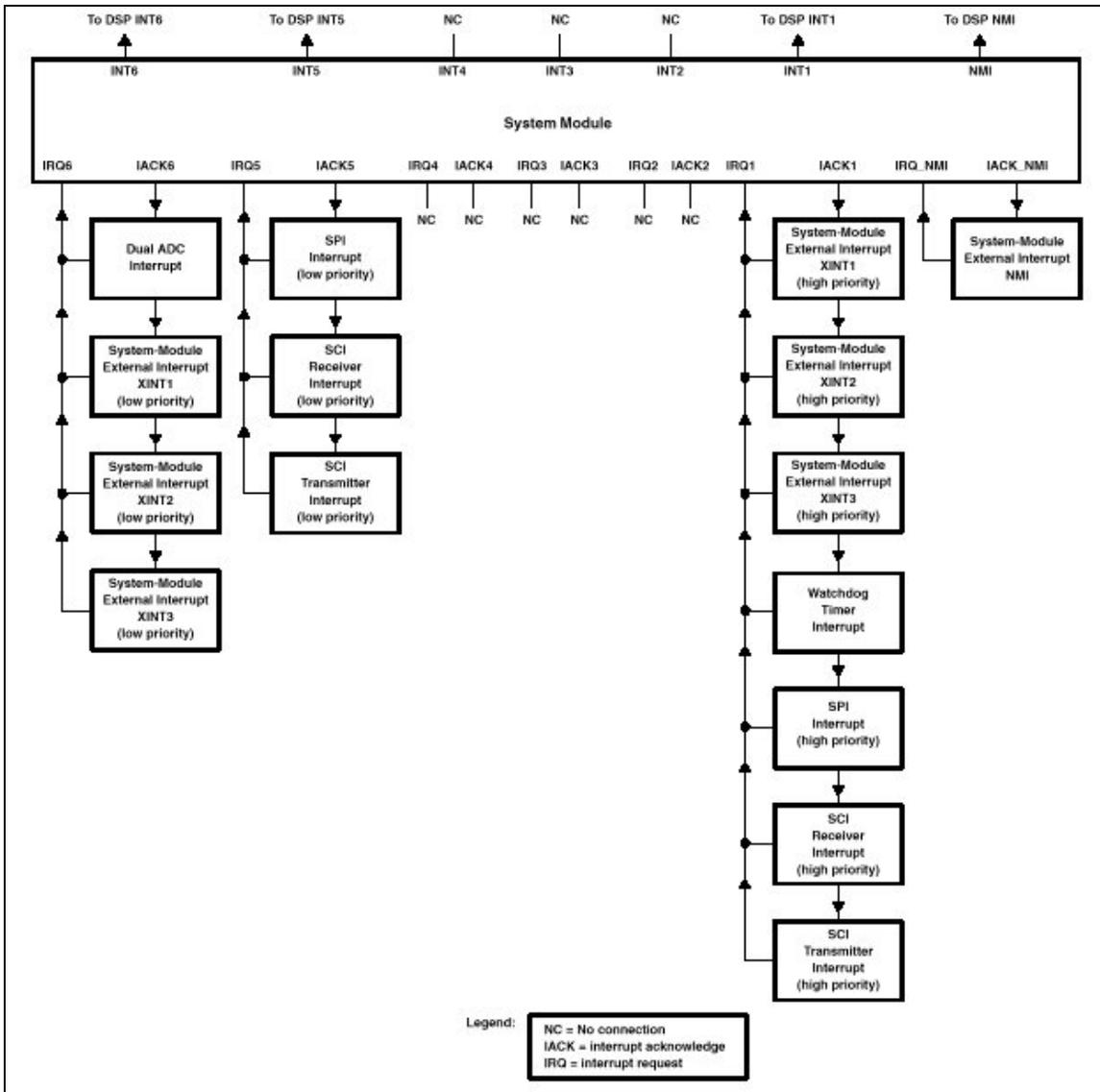
- Type A interrupt pins allow for digital input only. One Type A pin is a maskable interrupt pin, the other is a non-maskable interrupt pin.
- Type B pins can be programmed as maskable or non-maskable interrupts. These can be used as digital input and output.
- Type C inputs can only be maskable. These can be used as digital input and output.

The interrupt structures of the event-manager and the system module are shown in Figure 4.5 and 4.6 respectively. Table 4.3 summarizes the interrupt sources, overall priority, vector address/ offset source and function of each interrupt.



Ref: Texas Instruments Literature # SPS0420D, Nov 1998

Figure 4.5: Event -Manager Interrupt Structure



Ref: Texas Instruments Literature # SPS0420D, Nov 1998

Figure 4.6: System-Module Interrupt Structure

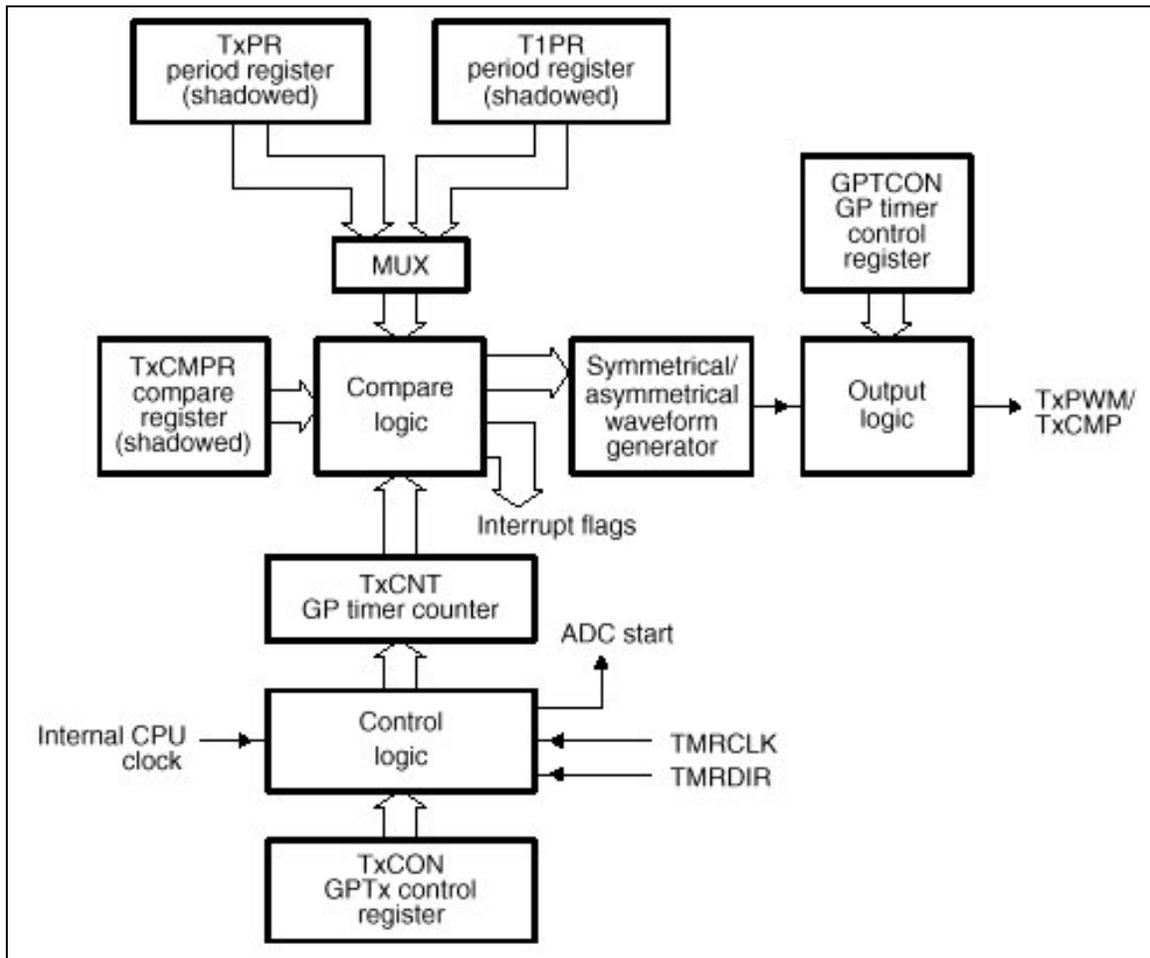
INTERRUPT NAME	OVERALL PRIORITY	DSP-CORE INTERRUPT, AND ADDRESS	PERIPHERAL VECTOR ADDRESS	PERIPHERAL VECTOR ADDRESS OFFSET	MASKABLE?	'x240 SOURCE PERIPHERAL MODULE	FUNCTION INTERRUPT
RS	1 Highest	RS 0000h	N/A		N	Core, SD	External, system reset (RESET)
RESERVED	2	INT7 0026h	N/A	N/A	N	DSP Core	Emulator trap
NMI	3	NMI 0024h	N/A	0002h	N	Core, SD	External user interrupt
XINT1 XINT2 XINT3	4 5 6	INT1 0002h (System)	SYSIVR (701Eh)	0001h 0011h 001Fh	Y	SD	High-priority external user interrupts
SPIINT	7			0005h	Y	SPI	High-priority SPI interrupt
RXINT	8			0006h	Y	SCI	SCI receiver interrupt (high priority)
TXINT	9			0007h	Y	SCI	SCI transmitter interrupt (high priority)
WDTINT	10			0010h	Y	WDT	Watchdog timer interrupt
PDPINT	11			0020h	Y	External	Power-drive protection Int.
CMP1INT	12			0021h	Y	EV.CMP1	Full Compare 1 interrupt
CMP2INT	13	0022h	Y	EV.CMP2	Full Compare 2 interrupt		
CMP3INT	14	0023h	Y	EV.CMP3	Full Compare 3 interrupt		
SCMP1INT	15	INT2 0004h (Event Manager Group A)	EVIVRA (7432h)	0024h	Y	EV.CMP4	Simple compare 1 interrupt
SCMP2INT	16			0025h	Y	EV.CMP5	Simple compare 2 interrupt
SCMP3INT	17			0026h	Y	EV.CMP6	Simple compare 3 interrupt
TPINT1	18			0027h	Y	EV.GPT1	Timer1-period interrupt
TCINT1	19			0028h	Y	EV.GPT1	Timer1-compare interrupt
TUFINT1	20			0029h	Y	EV.GPT1	Timer1-underflow interrupt
TOFINT1	21			002Ah	Y	EV.GPT1	Timer1-overflow interrupt
TPINT2	22			002Bh	Y	EV.GPT2	Timer2-period interrupt
TCINT2	23			002Ch	Y	EV.GPT2	Timer2-compare interrupt
TUFINT2	24			002Dh	Y	EV.GPT2	Timer2-underflow interrupt
TOFINT2	25	002Eh	Y	EV.GPT2	Timer2-overflow interrupt		
TPINT3	26	INT3 0006h (Event Manager Group B)	EVIVRB (7433h)	002Fh	Y	EV.GPT3	Timer3-period interrupt
TCINT3	27			0030h	Y	EV.GPT3	Timer3-compare interrupt
TUFINT3	28			0031h	Y	EV.GPT3	Timer3-underflow interrupt
TOFINT3	29			0032h	Y	EV.GPT3	Timer3-overflow interrupt
CAPINT1	30			0033h	Y	EV.CAP1	Capture 1 interrupt
CAPINT2	31			0034h	Y	EV.CAP2	Capture 2 interrupt
CAPINT3	32			0035h	Y	EV.CAP3	Capture 3 interrupt
CAPINT4	33	0036h	Y	EV.CAP4	Capture 4 interrupt		
SPIINT	34	INT5	SYSIVR (701Eh)	0005h	Y	SPI	Low-priority SPI interrupt
RXINT	35	000Ah (System)		0006h	Y	SCI	SCI receiver interrupt (low priority)
TXINT	36			0007h	Y	SCI	SCI transmitter interrupt (low priority)
ADCINT	37	INT6	SYSIVR	0004h	Y	ADC	Analog-to-digital interrupt
XINT1	38	000Ch (System)	(701Eh)	0001h	Y	External pins	Low-priority external user interrupts
XINT2	39			0011h	Y		
XINT3	40			001Fh	Y		
RESERVED	41	000Eh	N/A		Y	DSP Core	Used for analysis
TRAP	N/A	0022h	N/A		N/A		TRAP instruction vector

The interrupt operation for a maskable interrupt can be summarized as follows-

1. The flag bit in the individual control register is set. If the individual mask bit is also set, the corresponding IFR bit is set.
2. Once the IFR bit is set, the acknowledgement conditions (INTM bit =0 and IMR bit = 1) are tested. If the conditions are true, the CPU services the interrupt, generating the interrupt acknowledge signal; otherwise, it ignores the interrupt and continues with the current code sequence.
3. When the interrupt has been acknowledged, the IFR bit is cleared to 0 and the INTM bit is set to 1 (to block other maskable interrupts). The flag bit in the corresponding control register is not cleared.
4. The return address (incremented PC value) is saved on the stack.
5. The CPU branches to and executes the interrupt service routine (ISR). The ISR is concluded by the return instruction, which pops the return address off the stack. The CPU continues with the interrupt code sequence.

4.4.1 The General Purpose Timers

The TMS320F240 has 3 general-purpose timers. The block diagram is shown in figure below-



Ref: Texas Instruments Literature # SPRU161A, March 1997

Figure 4.7: General Purpose (GP) Timer Block Diagram (x = 1, 2, 3)

GP Timer Inputs

1. Internal CPU Clock - This comes directly from the DSP core.
2. TMRCLK - This is the external clock. It can have a maximum frequency one-fourth that of the CPU clock.

3. TMRDIR - This is the directional input, which is useful for the timer in the directional up/ down counting mode.
4. RESET - This is timer-reset signal.

GP Timer Outputs

1. TxPWM/TxCMP (x = 1, 2, 3) - These are the timer compare outputs. These can be specified to be active high, active low, forced high or forced low.
2. ADC Start - This is the start signal for the ADC module.
3. Underflow, overflow, compare match and period match signals to the compare logic and full and simple compare units.
4. Counting direction indication bits.

Control Registers for Timer Operation

The two registers, which control the timer operation, are:

TxCON (x = 1, 2, 3) - GP Timer Control Register

This register basically controls the mode of operation, input clock frequency, timer enable/ disable, clock source etc. for each individual timer. The detailed description for each bit of this register can be found in the appendix to this chapter.

GPTCON - GP Timer Control register

This register specifies the action to be taken by the GP timers on different timer events and gives an indication of the counting directions of each timer. The detailed description for each bit of this register can be found in the appendix to this chapter.

GP Timer Compare Registers

The compare register associated with a timer stores a value, which is continuously compared with that of the timer counter. When the two values match, there is a transition on the associated compare output, the corresponding interrupt flag is set and an interrupt request is sent to the CPU. The compare operation for each timer can be enabled or disabled with a bit in the appropriate TxCON register.

This register is shadowed, which means that a new value can be written to it at any time during the period, in order that the modification to become effective at the next period. This is possible because each compare register has associated with it a shadow register, which stores the updated contents.

GP Timer Period Register

The period register associated with a timer stores a count equivalent to the period of the timer. Depending on the counting mode of the timer, the GP timer stops and holds its current value, resets to 0 or starts counting downwards when the timer counter value reaches that in the period register. The period register too is shadowed.

GP Timer Interrupts

There are a total of 12 interrupt flags in EVIFRA and EVIFRB for the three GP timers. Each timer generates the following 4 interrupts-

Overflow: TxOFINT (x = 1,2,3)

This occurs when the timer counter reaches FFFFh.

Underflow: TxUFINT (x=1, 2, 3)

This occurs when the timer counter reaches 0000h.

Compare Match: TxCINT (x=1,2, 3)

This occurs when the value of the timer counter matches with that of the compare register. Period Match: TxPINT (x=1, 2, 3)

This occurs when the value of the timer counter is the same as that of the period register.

In each of the above cases, the corresponding interrupt flag is set two CPU cycles after the event occurs.

GP Timer Counting Operation

Each GP timer has the following 6 modes of operation-

- ❑ Stop/Hold mode
- ❑ Single-Up counting mode

- ❑ Continuous-Up counting mode
- ❑ Directional-Up/Down counting mode
- ❑ Single-Up/Down counting mode
- ❑ Continuous-Up/Down counting mode

Stop/Hold Mode

The operation of the GP timer stops and holds its current state. The timer counter, the compare output and the pre-scale counter all remain unchanged.

Single-Up Counting Mode

Basic Operation:

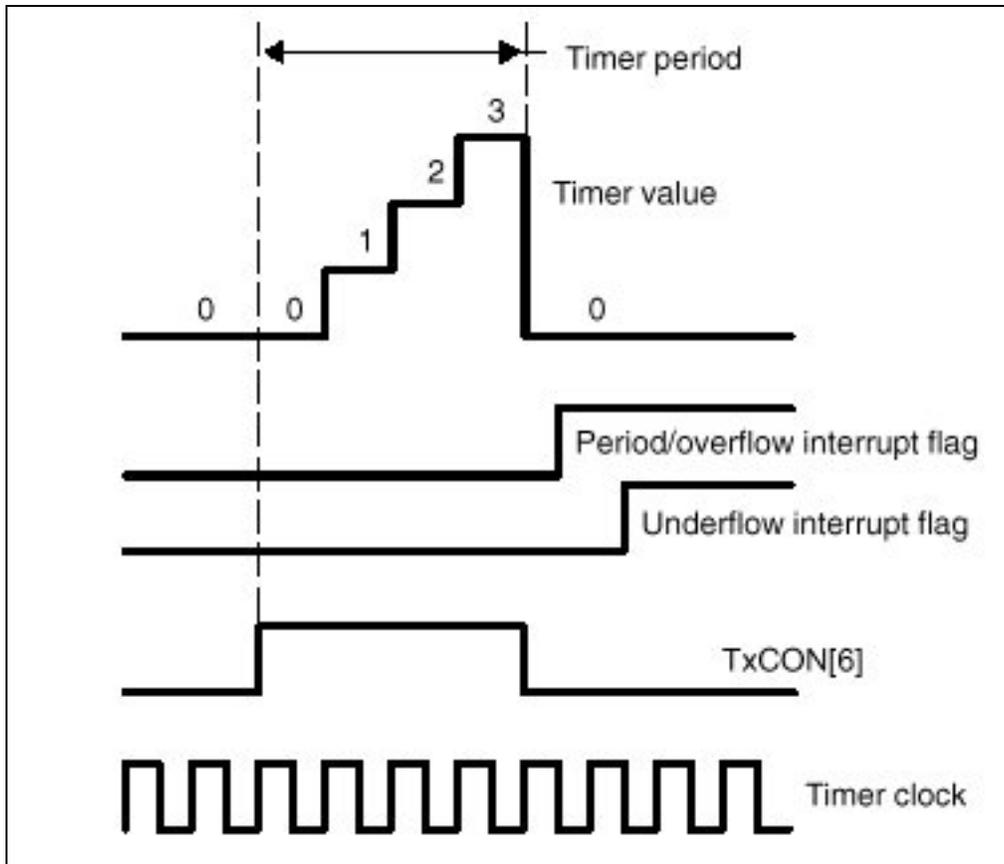
In this mode the timer counts up according to the scaled clock input until it reaches the value in the period register. Then, on the next rising clock edge, the timer resets to zero and disables the counting operation by resetting TxCON[6] bit. The initial value of the timer can be between 0 to FFFFh.

Interrupts Generated:

Two clock cycles after the match between the timer counter (TxCNT) and the period register occurs, the period interrupt is generated. At the same time, an ADC start signal is sent to the ADC module if this timer has been selected by the appropriate bits in GPTCON to start the ADC.

Two clock cycles after the timer counter reaches 0, the underflow interrupt is generated. At the same time, an ADC start signal is sent to the ADC module if this timer has been selected by the appropriate bits in GPTCON to start the ADC.

Two clock cycles after the timer counter reaches FFFFh, the overflow interrupt is generated. At the same time, an ADC start signal is sent to the ADC module if this timer has been selected by the appropriate bits in GPTCON to start the ADC.



Ref: Texas Instruments Literature # SPRU161A, March 1997

Figure 4.8: GP Timer Single-Up Counting Mode

Following is an example of a code segment for initialization of GP timer 1 in single-up counting mode.

```

.include f240regs.h

LDP    #00E8h    ;Point to appropriate Data Page for
                ;EV registers
;-----
;                CONFIGURE GPTCON
;-----

SPLK   #0041h,GPTCON ;Set bits 1-0 to 01 to program the
                ;timer 1 compare o/p as active low
                ;Set bit 6 to 1 to enable GP timer
                ;compare outputs
;-----
;                CONFIGURE T1PR and T1COM

```

```

;-----
SPLK    #10, T1PR    ;Set timer 1 period to 10 clock
                        ;cycles
SPLK    #5, T1COM    ;Set compare register to 5 clock
                        ;cycles
;-----
;
;                CONFIGURE T1CON
;-----
SPLK    #8942h      ;Set Timer 1 control
; Description of each bit in T1CON
;bit 0 0: Use own period register
;bit 1 1: GP timer 1 compare enabled
;bits 2-3 00: Load GP timer cmp register on underflow
;bits 4-5 00: Select internal clock
;bit 6 1: Timer counting operation enabled
;bit 7 0: Use own timer enable
;bits 8-10 001: Prescaler = 2
;bits 11-13 001: Single up count
;bit 14 0: SOFT = 0
;bit 15 1: FREE = 1

```

Continuous-Up Counting Mode

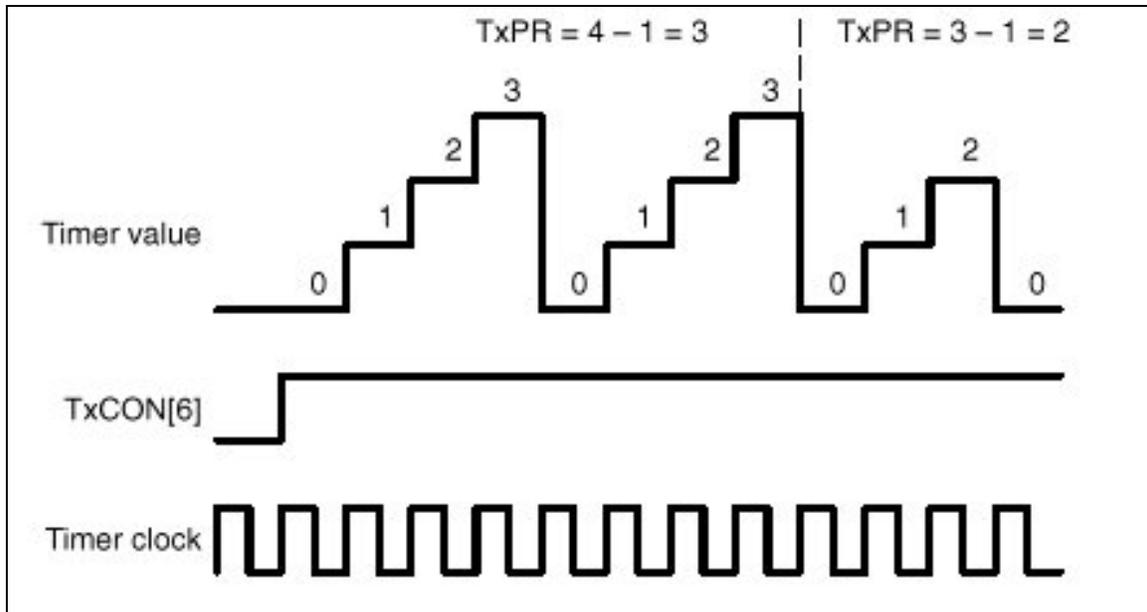
Basic Operation:

The operation is the same as the single-up counting mode repeated each time the counter is reset to 0. Thus the timer up counts until the value in the period register is reached, then resets to zero and starts another counting period. No clock cycle is missed from the time the counter reaches the value in period register, to the time it starts another counting cycle. The initial value of the timer can be between 0 to FFFFh.

Interrupts Generated:

The period, overflow and underflow interrupts are generated the same way as in the single-up counting mode.

This mode finds applications in the generation of edge-triggered or asynchronous PWM waveforms and sampling periods in many motor and motion control systems.



Ref: Texas Instruments Literature # SPRU161A, March 1997

Figure 4.9: GP Timer Continuous-Up Counting Mode

Directional-Up/Down Counting Mode

Basic Operation:

The timer counts up or down according to the scaled input clock and the TMDIR input. When the corresponding TMDIR pin is held HIGH, the GP timer counts up until it reaches the value in the period register or FFFFh. Once this condition is reached, the timer holds at that value.

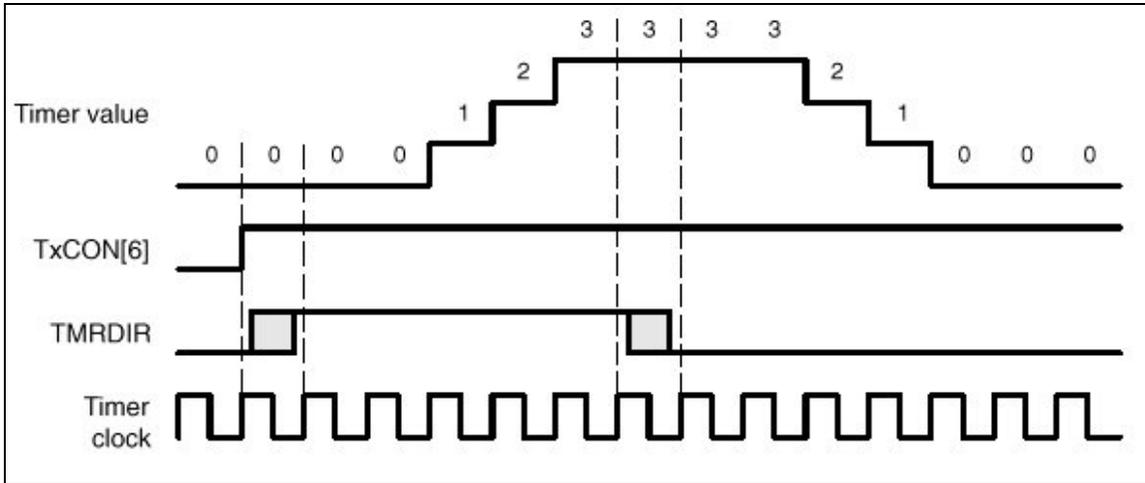
When the corresponding TMDIR pin is held LOW, the GP timer counts down until it reaches the value 0 and then, the timer holds at that value.

The initial value can be between 0 to FFFFh. If the initial value is greater than that of the period register, the timer will count up to FFFFh and hold the value if TMDIR = HIGH, the timer. If the initial value is equal to the period register, the timer will hold at that value if TMDIR = HIGH and count down if TMDIR = LOW.

Interrupts Generated:

The period, overflow and underflow interrupts are generated the same way as in the single-up counting mode.

This counting mode can be used with the Quadrature Encoder Pulse (QEP) circuits in the EV module that are discussed in a later section. This mode can also be used to time the occurrence of specific external events in motor control and power electronic applications.



Ref. Texas Instruments Literature # SPRU161A, March 1997

Figure 4.10: GP Timer Directional-Up/Down Counting Mode

Single-Up/Down Counting Mode

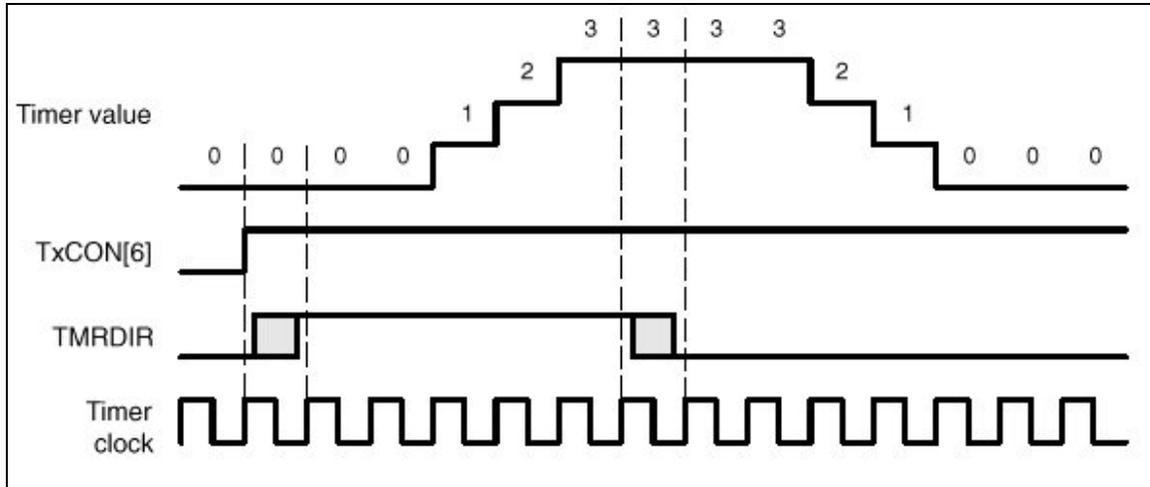
In this mode the timer counts up until it reaches the value in the period register. It then changes its counting direction and starts down counting until it reaches a value of 0. Once this value is reached, the timer resets the corresponding TxCON[6] bit to disable counting, resets its prescale counter, stops counting and holds its current value.

The initial value can be between 0 to FFFFh. If the initial timer value is greater than that of the period register, the counter will count up to FFFFh, reset to 0 and then continue as if the initial value of the timer is 0.

If initial value of the timer is the same as that of the period register, the timer will count down to 0 and end the period.

Interrupts Generated:

The period, overflow and underflow interrupts are generated the same way as in the single-up counting mode. However, the period event happens when the timer counter and period register values match, which is in the middle of the period.



Ref: Texas Instruments Literature # SPRU161A, March 1997

Figure 4.11: GP Timer Single-Up/Down Counting Mode

Continuous-Up/Down Counting Mode

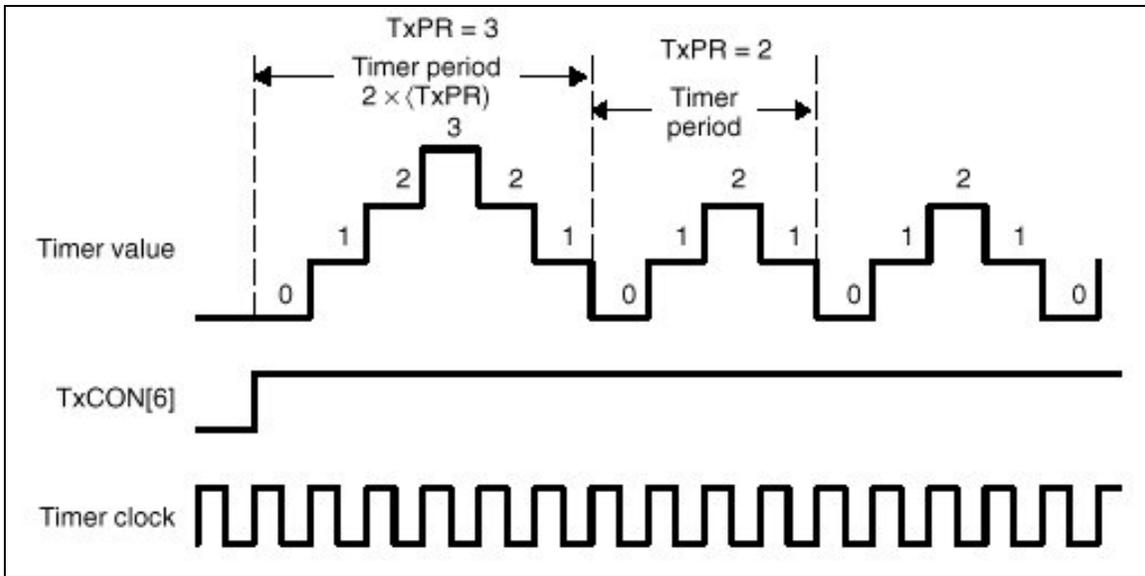
Basic Operation:

This mode of operation is the same as the single-up/down counting mode, repeated each time the timer is reset to 0.

Interrupts Generated:

The period, overflow and underflow interrupts are generated the same way as in the single-up/down counting mode.

This mode is useful in generation of centered or symmetric PWM waveforms found in a broad range of motor/ motion control and power electronic applications.



Ref: Texas Instruments Literature # SPRU161A, March 1997

Figure 4.12: GP Timer Continuous-Up/Down Counting Mode

The following table summarizes the contents of the timer control registers for the various counting modes for timer 1.

Mode	GPTCON	TxCON
Single-Up	0041h	8942h
Continuous-Up	006Ah	8142h
Directional-Up/Down	006Ah	9852h
Single-Up/Down	0041h	8924h
Continuous-Up/Down	003Fh	A840h

GP Timer Compare Operation

Each GP timer has associated with it a compare register and a TxPWM/TxCMP output. When the compare operation is enabled, (by setting TxCON[1] bit to 1), the value in the timer counter is compared with that in the compare register and when a match occurs,

- The compare interrupt flag of the corresponding timer is set after two clock cycles.

- ❑ A transition occurs on the TxPWM/TxCMP output depending on the bit configuration in the GPTCON one CPU cycle after the match only if the timer is not in directional-up/down counting mode.
- ❑ An ADC signal is sent to the ADC module at the same time when the compare interrupt flag is set if the flag has been selected in the GPTCON to start the ADC.

The transition on the compare/PWM pin depends on the following-

- Symmetric/ Asymmetric waveform generator
- Associated output logic.
- Definition of bits 1-0 in GPTCON.
- Counting mode of timer. No transition occurs in the directional-up/down counting mode.
- Counting direction when the counting is in single- or continuous-up/down mode.

Symmetric/Asymmetric Generator-

This generator generates a symmetric or asymmetric waveform depending on the counting mode the timer is in.

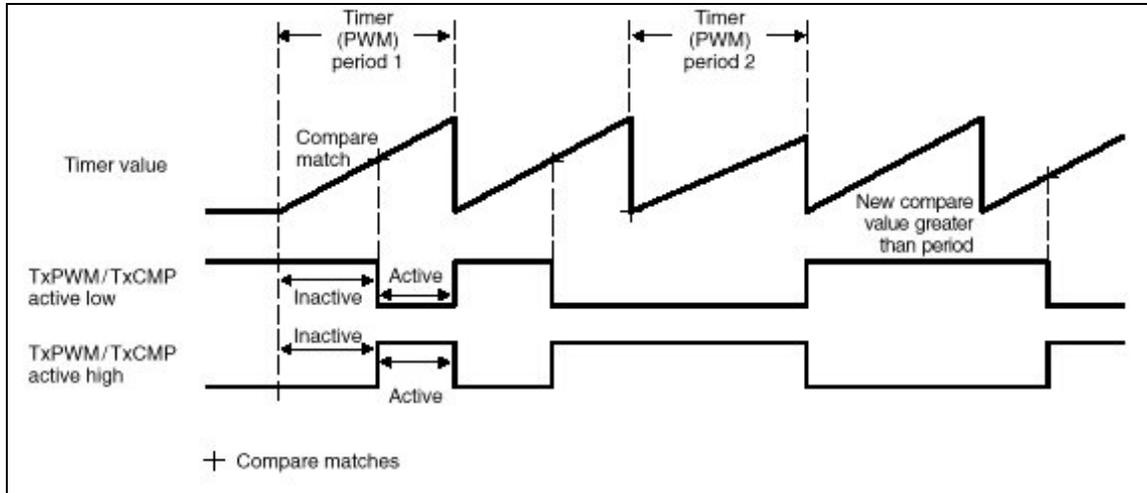
Waveform	Counting Mode
Asymmetric	Single-up
	Continuous-up
Symmetric	Single-up/down
	Continuous-up/down

Asymmetric Waveform Generation:

This waveform is generated when the timer is in single-up and continuous-up counting modes. The output of the waveform generator changes as per the following sequence:

- 0 before the counter operation starts
- remains unchanged until the compare match occurs
- toggles on compare match

- remains unchanged until end of period
- resets to 0 at the end of a period match, if the new compare value for the following period is not 0.



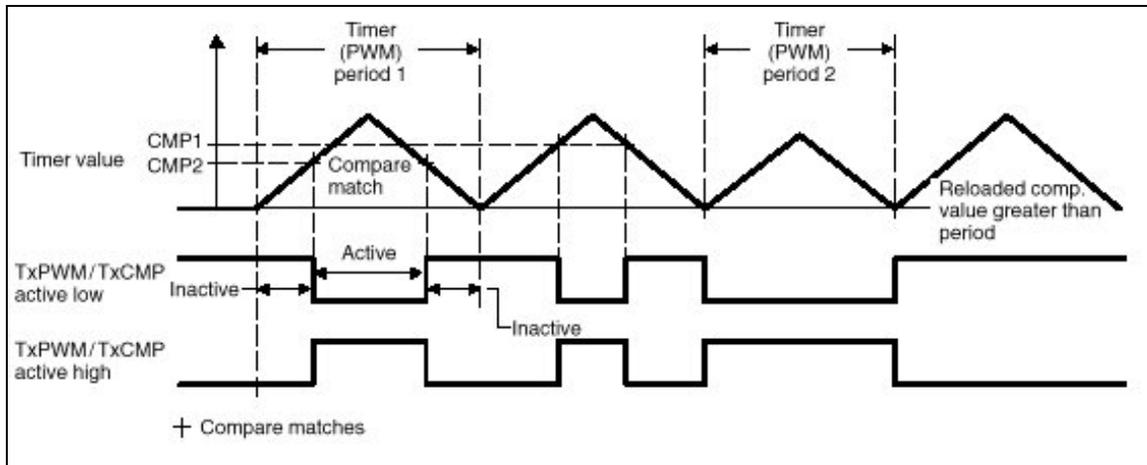
Ref. Texas Instruments Literature # SPRU161A, March 1997

Figure 4.13: GP Timer Compare/PWM Output in Up Counting Modes

Symmetric Waveform Generation:

This waveform is generated when the timer is in single-up/down and continuous-up/down counting modes. The output of the waveform generator changes as per the following sequence:

- 0 before the counter operation starts
- remains unchanged until the compare match occurs
- toggles on the first compare match
- remains unchanged until the second compare match
- Toggles on the second compare match
- Remains unchanged until the end of the period
- resets to 0 at the end of a period match, if there is no second compare match and the new compare value for the following period is not 0.



Ref: Texas Instruments Literature # SPRU161A, March 1997

Figure 4.14: GP Timer Compare/PWM Output in Up/Down Counting Modes

Output Logic -

The output logic conditions the output of the waveform generator to form the ultimate compare/PWM output to control different kinds of devices. The compare/PWM output can be defined as active high, active low, forced high or forced low.

Compare Units

The EV has 3 full compare units and 3 simple compare units. Each full compare module has 2 associated compare/PWM outputs. Each simple compare unit has 1 associated compare/PWM output. GP timer 1 provides the time-base for the full compare units, while GP timers 1 or 2 provide it for the simple compare units.

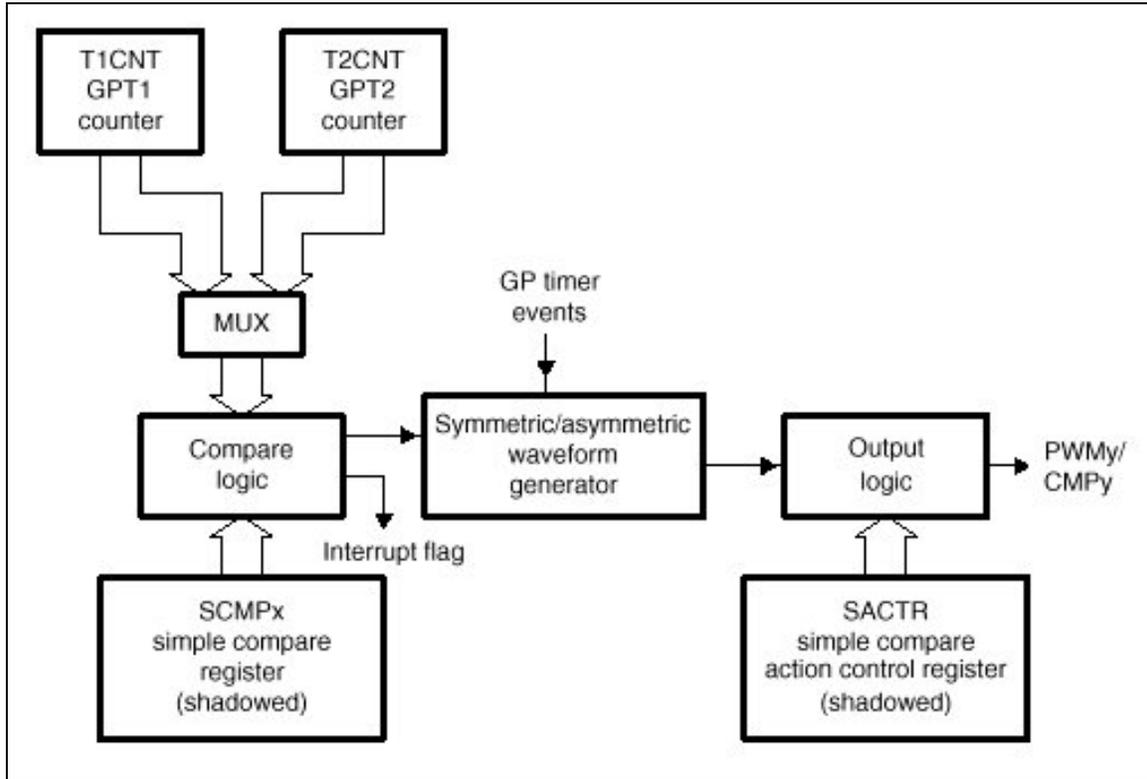
Simple Compare Units

The 3 simple compare units include:

- ❑ Three 16-bit compare registers (SCMPRx, x = 1, 2, 3)
- ❑ One compare control register (COMCON) which is shared with the full compare units.
- ❑ One 16-bit action control register (SACTR)
- ❑ Three symmetric/asymmetric waveform generators.

- ❑ Three compare/PWM (3-state) outputs, PWM_y/CMP_y, y=7, 8, 9, one for each simple compare unit, with programmable polarity.
- ❑ Compare and Interrupt logic

The block diagram for the simple compare units is shown in Figure 4.15 below.



Ref: Texas Instruments Literature # SPRU161A, March 1997

Figure 4.15: Simple Compare Unit Block Diagram (x = 1, 2, or 3; y = 7, 8, or 9)

The operation of the simple compare unit is similar to the GP timer except that:

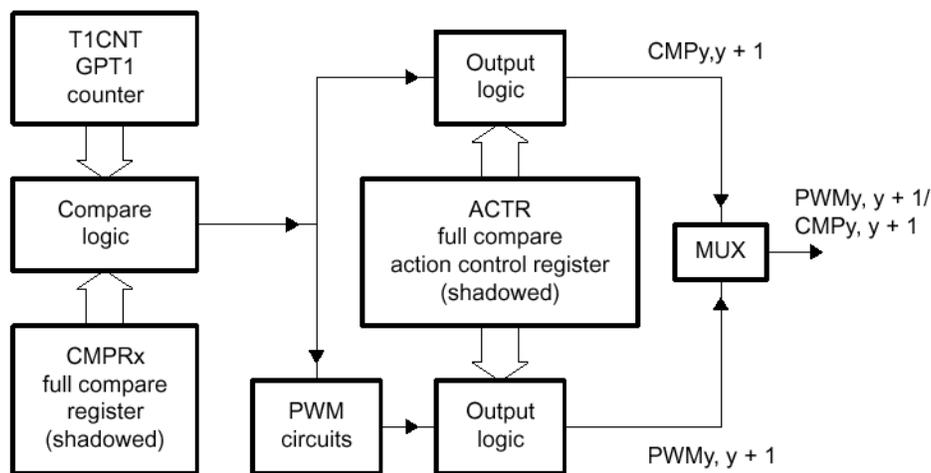
- ❑ The time base can be GP timer 1 or 2
- ❑ The enabling/ disabling of simple compare operation, the enabling/disabling of simple compare outputs, the condition when simple compare register is updated, and selection of time base for simple compare operation are controlled by appropriate bits in the COMCON register.
- ❑ The behavior of compare outputs of simple compare units is individually defined by corresponding bits in simple compare action control register SACTR.

Full Compare Units

The 3 full compare units include:

- ❑ Three 16-bit compare registers (CMPRx, $x = 1, 2, 3$).
- ❑ One 16-bit compare control register (COMCON)
- ❑ One 16-bit action control register (ACTR)
- ❑ Six compare/PWM (3-state) output pins, PWM_y/CMP_y, $y = 1, 2, 3, 4, 5, 6$.
- ❑ Control and interrupt logic.

The functional diagram of the full compare unit is shown in Figure 4.16 below.



Ref: Texas Instruments Literature # SPRU161A, March 1997

Figure 4.16: Full Compare Unit Block Diagram ($x = 1, 2, \text{ or } 3$; $y = 1, 3, \text{ or } 5$)

The full compare units can operate in two modes:

Compare Mode-

In this mode,

- ❑ The value in GP timer 1 is continuously compared with that in the compare register.
- ❑ Once a match occurs, a transition appears on the two outputs of the compare unit depending on the status of the bits in the ACTR. The ACTR can individually specify each output to Hold, Reset (go low), Set (go high) or Toggle on a compare match.
- ❑ When the compare match is made, the compare interrupt flag of the associated full compare unit is set if compare is enabled.

- The timing of output transitions, setting of interrupt flags, and generation of interrupt requests is the same as that of GP timer compare operation.

PWM Mode -

Each full compare unit can be individually put into PWM mode. The operation in this mode is similar to the GP timer compare operation with a few exceptions that are discussed in a later section.

- APPENDIX -

GP Timer Control Registers

GP Timer Control Register (TxCON; x = 1, 2, and 3)							
15	14	13	12	11	10	9	8
Free	Soft	TMODE2	TMODE1	TMODE0	TPS2	TPS1	TPS0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0
7	6	5	4	3	2	1	0
TSWT1	TENABLE	TCLKS1	TCLKS0	TCLD1	TCLD0	TECMPR	SELT1PR
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: R = read access, W = write access, -0 = value after reset

Bits 15–14 Free, Soft. Emulation control bits

- 00 = Stop immediately on emulation suspend
- 01 = Stop after current timer period is completed on emulation suspend
- 10 = Operation is not affected by emulation suspend
- 11 = Operation is not affected by emulation suspend

Bits 13–11 TMODE2–TMODE0. Count mode selection

- 000 = Stop/hold
- 001 = Single up counting mode
- 010 = Continuous up counting mode
- 011 = Directional up/down counting mode
- 100 = Single up/down counting mode
- 101 = Continuous up/down counting mode
- 110 = Reserved. Result is unpredictable
- 111 = Reserved. Result is unpredictable

Bits 10–8 TPS2–TPS0. Input clock prescaler

- 000 = x/1
- 001 = x/2
- 010 = x/4
- 011 = x/8
- 100 = x/16
- 101 = x/32
- 110 = x/64
- 111 = x/128

x = CPU clock frequency

Bit 7 TSWT1. (GP timer start with GP timer 1). Start timer with GP timer 1 timer enable bit. This bit is reserved in T1CON.

- 0 = Use own TENABLE bit
- 1 = Use TENABLE bit of T1CON to enable and disable operation ignoring own TENABLE bit

Bit 6 TENABLE. Timer enable. In single up counting and single up/down counting modes, this bit is cleared to 0 by the timer after it completes a period of operation.

- 0 = Disable timer operation; that is, put the timer in hold and reset the prescaler counter
- 1 = Enable timer operations

Bits 5–4 TCLKS1, TCLKS0. Clock source select

- 00 = Internal
- 01 = External
- 10 = Rollover from GP timer 2 (only applicable to T3CON when GP Timers 2 and 3 are cascaded into a 32-bit timer; reserved in T1CON and T2CON; ILLEGAL if SELT1PR=1.)
- 11 = Quadrature encoder pulse circuit (only applicable to T2CON and T3CON; reserved in T1CON; ILLEGAL if SELT1PR is 1)

ILLEGAL means result is unpredictable.

- Bits 3–2** **TCLD1, TCLD0.** Timer compare (active) register reload condition
- 00 = When counter is 0
 - 01 = When counter value is 0 or equals period register value
 - 10 = Immediately
 - 11 = Reserved
- Bit 1** **TECMPR.** Timer compare enable
- 0 = Disable timer compare operation
 - 1 = Enable timer compare operation
- Bit 0** **SELT1PR.** Period register select. This bit is reserved in T1CON.
- 0 = Use own period register
 - 1 = Use T1PR as period register ignoring own period register

Note: Synchronization of the GP Timers

Two consecutive writes to T1CON are required to ensure the synchronization of the GP timers when T1CON[6] is used to enable GP timer 2 or 3:

- 1) Configure all other bits with T1CON[6] set to 0.
- 2) Enable GP timer 1 and, thus, GP timer 2 or GP timers 2 and 3, by setting T1CON[6] to 1.

GP Timer Control Register (GPTCON)

Figure 6–11. GP Timer Control Register (GPTCON) — Address 7400h

15	14	13	12–11	10–9	8–7
T3STAT	T2STAT	T1STAT	T3TOADC	T2TOADC	T1TOADC
R–1	R–1	R–1	RW–0	RW–0	RW–0
6	5–4	3–2	1–0		
TCOMPOE	T3PIN	T2PIN	T1PIN		
RW–0	RW–0	RW–0	RW–0		

Note: R = read access, W = write access, –n = value after reset

- Bit 15** **T3STAT.** GP timer 3 status. Read only
- 0 = Count down
 - 1 = Count up
- Bit 14** **T2STAT.** GP timer 2 status. Read only
- 0 = Count down
 - 1 = Count up
- Bit 13** **T1STAT.** GP timer 1 status. Read only
- 0 = Count down
 - 1 = Count up
- Bits 12–11** **T3TOADC.** Start ADC by GP timer 3 event
- 00 = No event starts ADC
 - 01 = Setting of underflow interrupt flag
 - 10 = Setting of period interrupt flag
 - 11 = Setting of compare interrupt flag
- Bits 10–9** **T2TOADC.** Start ADC by GP timer 2 event
- 00 = No event starts ADC
 - 01 = Setting of underflow interrupt flag. Starts ADC
 - 10 = Setting of period interrupt flag. Starts ADC
 - 11 = Setting of compare interrupt flag. Starts ADC

Bits 8–7 **T1TOADC.** Start ADC by GP timer 1 event

- 00 = No event starts ADC
- 01 = Setting of underflow interrupt flag
- 10 = Setting of period interrupt flag
- 11 = Setting of compare interrupt flag

Bit 6 **TCOMPOE.** Compare output enable. Active PDPINT writes 0 to this bit.

- 0 = Disable all three GP timer compare outputs (all three compare outputs are put in the high-impedance state)
- 1 = Enable all three GP timer compare outputs

Bits 5–4 **T3PIN.** Polarity of GP timer 3 compare output

- 00 = Forced low
- 01 = Active low
- 10 = Active high
- 11 = Forced high

Bits 3–2 **T2PIN.** Polarity of GP timer 2 compare output

- 00 = Forced low
- 01 = Active low
- 10 = Active high
- 11 = Forced high

Bits 1–0 **T1PIN.** Polarity of GP timer 1 compare output

- 00 = Forced low
- 01 = Active low
- 10 = Active high
- 11 = Forced high

Example 1

Write a program that generates a 10 ms pulse.

Solution:

To generate the pulse, we can operate the general-purpose timer 1 (GPT1) in the single up/down counting mode. We load a count in the period register corresponding to half the width of the pulse. The timer is enabled with the value 0 loaded into the counter register. The counter resets to zero after it reaches the value in the period register. The output can be observed on an oscilloscope at the T1PWM/T1CMP/IOPB3 pin that is the pin 12 on I/O connector P1 of the EVB.

Registers involved:

T1PR = 50000

The calculation of the value to be loaded in T1PR is as follows-

Period Value = $\{(\text{CLKINOSC}/\text{PRESCALER}) \times (\text{PLL MTPLN RATIO} / \text{PLL DIVIDE})\} / \text{DESIRED FREQ}$
CLKINOSC = 10MHz. This is the input oscillator frequency and is select by bits 7-4 of the CKCR1 register.

PLL DIVIDE = 2. Bit 0 of CKCR0 decides whether this system pre-scale value is 1 or 2.

PLL MTPLN RATIO = 4. This is selected by bits 2-0 of CKCR1 which are 011 in this case.

PRESCALER = 2. This is set by bits 10-8 of T1CON.

Thus, Period Value = $\{(10^7/2) \times (4/2)\} / 100 = 10^5$.

We load T1PR with half the value,

\therefore T1PR = 50000.

GPTCON = 0069h

Enable Compare outputs of all GPTs. GPT1 compare output - active low

T1CON = A142h

Select single up/down counting mode, Prescaler = 2, enable timer compare operation, enable timer operation.

```

*****
; File Name:   ch4_e1.asm
; Description: This program a 10ms pulse.
;*****
    .include f240regs.h

    .bss ctr,1
    .bss GPR0,1

    .text
NOP
START: SETC INTM      ;Disable interrupts
        SPLK #0000h,IMR ;Mask all core interrupts
        LACC IFR      ;Read Interrupt flags
        SACL IFR      ;Clear all interrupt flags

        CLRC SXM      ;Clear Sign Extension Mode
        CLRC OVM      ;Reset Overflow Mode
        CLRC CNF      ;Config Block B0 to Data mem

        LDP  #00E0h    ;DP for addresses 7000h-707Fh
        SPLK #0041h,CKCR0 ;Disable PLL, necessary for
                        ;modifying CKCR1
        SPLK #00BBh,CKCR1 ;CLKIN(OSC)=10MHz, CPUCLK=20MHz
        SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable, SYSCLK=CPUCLK/2
        SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK
        SPLK #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 2-3)
        KICK_DOG      ;Reset Watchdog

        SPLK #0h,GPR0 ;Set wait state generator for:
        OUT  GPR0,WSGR ;Program Space, 0 wait states
                        ;Data Space, 0 wait states
                        ;I/O Space, 0 wait states
        LDP  #00E1h    ;Point to appropriate Data Page
        LACC OCRA
        OR #0800h     ;The primary function for the
        SACL OCRA     ;T1PWM/T1CMP/IOPB3 pin
        LDP  #00E8h
        SPLK #0069h,GPTCON ;Enable all three GP timer compare
                                ;outputs. Set polarity of GP timer1
                                ; to "Active Low"

```

```

SPLK #50000,T1PR    ;Load period register
SPLK #0h,T1CNT     ;Set initial count = 0
SPLK #0A142h,T1CON ;Select Single-Up/Down counting Mode
                    ;Set input prescaler to 2
                    ;Select internal clock
                    ;Enable Timer Operation
END      B  END

```

Example 2

Write a program that generates a 1 kHz square-wave output at pin T1PWM/T1CMP/IOPB3.

Solution:

Since the output is required at the T1PWM/T1CMP/IOPB3 pin, we have to use the GP timer1. There are many ways of generating a square wave. In this example, we will use the continuous up counting mode of the timer. The period register is loaded with the appropriate value. The compare register is loaded with a count corresponding to half the period - so as to get a square wave.

Registers involved:

T1PR = 5000

The calculation of the value to be loaded in T1PR is as follows-

Period Value = {(CLKINOSC/PRESCALER) × (PLL MTPLN RATIO / PLL DIVIDE)} / DESIRED FREQ

CLKINOSC = 10MHz. This is the input oscillator frequency and is select by bits 7-4 of the CKCR1 register.

PLL DIVIDE = 2. Bit 0 of CKCR0 decides whether this system pre-scale value is 1 or 2.

PLL MTPLN RATIO = 4. This is selected by bits 2-0 of CKCR1 which are 011 in this case.

PRESCALER = 4. This is set by bits 10-8 of T1CON.

Thus, Period Value = $\{(10^7/4) \times (4/2)\} / 1000 = 5000$.

T1CMPR = 2500

GPTCON = 006Ah

Enable Compare outputs of all GPTs. GPT1 compare output - active high

T1CON = 9242h

Select continuous up counting mode, Prescaler = 4, enable timer compare operation, enable timer operation.

OCRA = 0800h

Configure the o/p pin T1PWM/T1CMP/IOPB3 pin for the T1PWM function.

Following is the complete code-

```
*****
; File Name:   ch4_e2.asm
; Description: This program generates a 1 kHz square-wave output at pin
;             T1PWM/T1CMP/IOPB3.
*****
    .include f240regs.h
    .bss GPR0,1
    .text
    NOP
START: SETC INTM          ;Disable interrupts
        SPLK #0000h,IMR   ;Mask all core interrupts
        LACC IFR          ;Read Interrupt flags
        SACL IFR          ;Clear all interrupt flags

        CLRC SXM         ;Clear Sign Extension Mode
        CLRC OVM         ;Reset Overflow Mode
        CLRC CNF         ;Config Block B0 to Data mem

        LDP #00E0h       ;DP for addresses 7000h-707Fh
        SPLK #0041h,CKCR0 ;Disable PLL, necessary for
                        ;modifying CKCR1
        SPLK #00BBh,CKCR1 ;CLKIN(OSC)=10MHz, CPUCLK=20MHz
        SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable, SYSCLK=CPUCLK/2
        SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK
        SPLK #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 2-3)
        KICK_DOG         ;Reset Watchdog
        SPLK #0h,GPR0    ;Set wait state generator for:
        OUT GPR0,WSGR    ;Program Space, 0 wait states
                        ;Data Space, 0 wait states
                        ;I/O Space, 0 wait states
        LDP #00E1h      ;Point to appropriate Data Page
        OR #0800h       ;The primary function for the
        SACL OCRA       ;T1PWM/T1CMP/IOPB3 pin
```

```

LDP #00E8h
SPLK #006Ah,GPTCON ;Enable all three GP timer compare
                    ;outputs. Set polarity of GP timer1
                    ; to "Active High"

SPLK #5000,T1PR    ;Load count in period register
SPLK #2500,T1CMPR ;Load count in compare register
SPLK #0,T1CNT     ;Set initial count to 0
SPLK #9242h, T1CON ;Select Continuous-Up counting Mode
                    ;Set input prescaler to 4
                    ;Select internal clock
                    ;Enable Timer Operation
                    ;Program the counter to stop
                    ;immediately on emulation suspend

WAIT B WAIT

```

The contents of the relevant timer registers can be checked at the following addresses:

- GPTCON - 7400h
- T1CNT - 7401h
- T1CMPR - 7402h
- T1PR - 7403h
- T1CON - 7404h

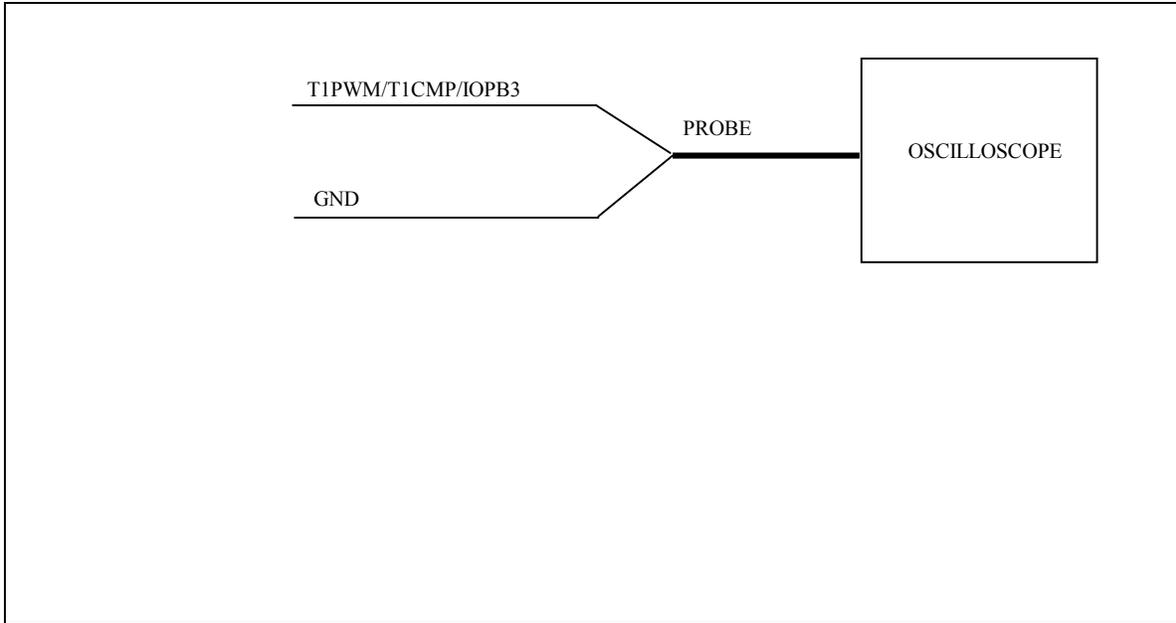
In the debugger this can be done in a number of ways such as:

- mem 0x7400** - Shows contents of addresses starting from 7400h in the memory window.
- ? *0x7400** - Shows the contents of memory location 7400h only. Note: Do not forget the *, without it the debugger just returns the decimal equivalent of the hex address entered.
- wa *0x7400** - Shows contents of location 7400h only in the watch window.
- wa *0x7400,GPTCON** - Shows contents of location 7400h with alias as GPTCON.

Check the bit 13 of the GPTCON after running the program. It should be '1' in order to indicate the up counting status of the GP timer1.

To check the square wave output on the oscilloscope, the following connections should be made-

I/O Connector P1 on EVB	
1	2
3	4
5	6
7	8
9	10
11	12
13	14
15	16
17	18



Generating Pulse-Width Modulation outputs

A pulse width modulation signal is a fixed frequency on-off signal with variable duty cycle. This signal plays important role in electromechanical system, especially for electric machine drives. Recall that the duty cycle is defined as the percentage of time the signal is high compared to the signal's period. Figure 4.14 illustrate PWM signals with different duty cycle. Note that a square wave is a special case of PWM signal with 50% duty cycle.

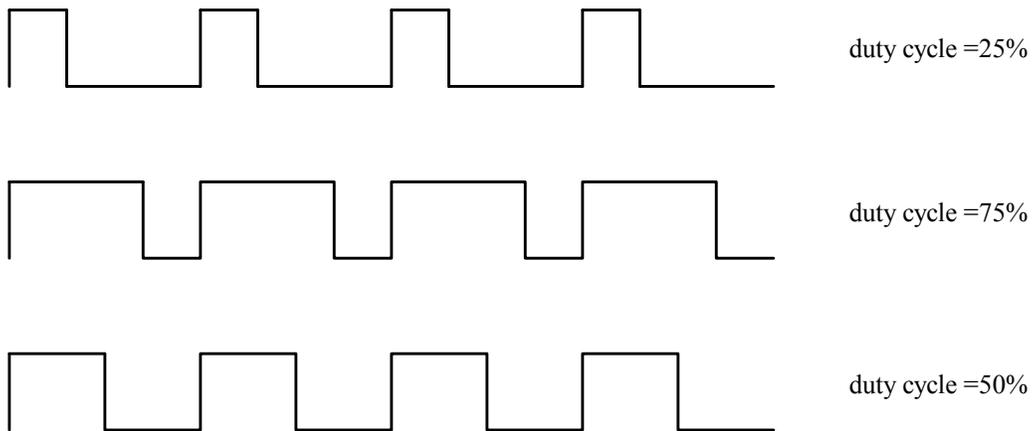


Figure 4.14 *PWM signals*

The TMS320F240 offers a number of ways for generation of the PWM wave.

1. Using the general-purpose timers.
2. Using the simple compare unit.
3. Using the full compare unit.

We shall study each of these below -

1. Generation of PWM wave using GP timer. Here again, two counting modes can be employed - the continuous up-counting mode for generation of asymmetric PWM waveform i.e. there is control over only one edge of the waveform: either the rising edge or the falling edge. Thus an asymmetric PWM has control over only switching on or off a device. The continuous up/down counting mode is used for the generation of a symmetric PWM waveform, which gives control over the rising as well as falling edge of the waveform. Thus symmetric PWM allows control over switching on and switching off a device.

Example 3

Write a program that generates an asymmetric PWM waveform of frequency 1kHz at 25% duty cycle.

$$\begin{aligned} \text{Period Value} &= \{(\text{CLKINOSC}/\text{PRESCALER}) * (\text{PLL MTPLN RATIO} / \text{PLL DIVIDE})\} / \text{DESIRED FREQ} \\ &= \{(10^7 / 1) * (4 / 2)\} / 1000 = 20000 \end{aligned}$$

```
*****
; File Name:      ch4_e3.asm
; Description:   This sample program generates a PWM wave at the
;               output of 75% duty cycle at 1kHz.
;*****
        .include f240regs.h
cmpr_val .set 5000
per_val .set 20000
;-----
; Variable Declarations for on chip RAM Blocks
;-----
        .bss GPR0,1      ;General purpose register.
;-----
```

```

; Vector address declarations
;-----
        .sect  ".vectors"
RSVECT B    START    ; Reset Vector
INT1   B    PHANTOM  ; Interrupt Level 1
INT2   B    PHANTOM  ; Interrupt Level 2
INT3   B    PHANTOM  ; Interrupt Level 3
INT4   B    PHANTOM  ; Interrupt Level 4
INT5   B    PHANTOM  ; Interrupt Level 5
INT6   B    PHANTOM  ; Interrupt Level 6
RESERVED B    PHANTOM ; Reserved
SW_INT8 B    PHANTOM ; User S/W Interrupt
SW_INT9 B    PHANTOM ; User S/W Interrupt
SW_INT10 B    PHANTOM ; User S/W Interrupt
SW_INT11 B    PHANTOM ; User S/W Interrupt
SW_INT12 B    PHANTOM ; User S/W Interrupt
SW_INT13 B    PHANTOM ; User S/W Interrupt
SW_INT14 B    PHANTOM ; User S/W Interrupt
SW_INT15 B    PHANTOM ; User S/W Interrupt
SW_INT16 B    PHANTOM ; User S/W Interrupt
TRAP   B    PHANTOM ; Trap vector
NMINT  B    PHANTOM ; Non-maskable Interrupt
EMU_TRAP B    PHANTOM ; Emulator Trap
SW_INT20 B    PHANTOM ; User S/W Interrupt
SW_INT21 B    PHANTOM ; User S/W Interrupt
SW_INT22 B    PHANTOM ; User S/W Interrupt
SW_INT23 B    PHANTOM ; User S/W Interrupt

;-----
; M A I N   C O D E   - starts here
;-----

        .text
        NOP
START:  SETC INTM          ;Disable interrupts
        SPLK #0000h,IMR    ;Mask all core interrupts
        LACC IFR          ;Read Interrupt flags
        SACL IFR          ;Clear all interrupt flags

        CLRC SXM          ;Clear Sign Extension Mode
        CLRC OVM          ;Reset Overflow Mode
        CLRC CNF          ;Config Block B0 to Data mem

```

```

LDP #00E0h ;DP for addresses 7000h-707Fh
SPLK #0041h,CKCR0 ;Disable PLL, necessary for
;modifying CKCR1
SPLK #00BBh,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2
SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK
SPLK #006Fh, WDCR ;Disable WD if VCCP=5V(JP5 pos. 2-3)
KICK_DOG ;Reset Watchdog
;-----
; SET UP DIGITAL I/O PORT
;-----
LDP #00E1h
SPLK #0800h, OCRA;configure gpt1 timer pwm o/p pin
LDP #00E8h
SPLK #0041h,GPTCON ;Enable all 3 GP timer
;compare output
;GP timer1 compare output
;"Active Low"
SPLK #cmpr_val, T1CMPR;Set compare value
SPLK #per_val, T1PR ;Period = 1 ms
SPLK #1042h, T1CON ;Operation not affected by
;emulation suspend
;Input clock prescaler = 1
;Enable timer operations
;Enable timer compare operation
WAIT B WAIT
;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM KICK_DOG ;Resets WD counter
B PHANTOM

```

Measuring Period (Frequency, Speed)

The period of a repetitive signal includes both the high and low parts of the cycle. To measure period, a program needs to capture the time of two successive rising (or

falling) edges. Below is an example program that measures the frequency of a signal. The basic scheme is to measure the number of rising edges encountered in one second. The rising edges are captured by the CAP1 input. The capture interrupt is enabled. Thus, the processor is interrupted at every rising edge. The ISR for the capture interrupt increments its counter CNT thus keeping track of the number of rising edges. The Timer1 is set for a period of 1 second and the period interrupt is enabled. This ISR stores the number of rising edges in "FREQ" and resets the counter for rising edges CNT.

Example 4

Write a program that measures the frequency of a square wave signal at CAP1 pin.

Solution:

Registers involved:

T1PR = 39062.5 \approx 9897h

The calculation of the value to be loaded in T1PR is as follows-

Period Value = $\{(\text{CLKINOSC}/\text{PRESCALER}) \times (\text{PLL MTPLN RATIO} / \text{PLL DIVIDE})\} / \text{DESIRED FREQ}$

CLKINOSC = 10MHz. This is the input oscillator frequency and is select by bits 7-4 of the CKCR1 register.

PLL DIVIDE = 2. Bit 3 of CKCR1, which is 1, determines PLL input is divided by 2.

PLL MTPLN RATIO = 1. This is selected by bits 2-0 of CKCR1 which are 000 in this case.

PRESCALER = 128. This is set by bits 10-8 of T1CON.

Thus, Period Value = $\{(10^7/128) \times (1/2)\} / 1 = 39062.5 \approx 39063 = 9896h$

GPTCON = 0055h

Enable Compare outputs of all GPTs. GPT1 compare output - active high

T1CON = 1746h

Select continuous up counting mode, Prescaler = 128, enable timer compare operation, enable timer operation.

OCRA = 3800h

Configure pin TxPWM/TxCMP (x=1, 2, 3) to be primary function

OCRB = 00F0h

Configure the CAP1 input pin.

CAPCON = A040h

Bits 14-13: 01 - Enable capture units 1 and 2.

Bit 9: 0 - Select GP Timer 2 as time base for capture unit 1.

Bits 7-6: 01- Detect rising edge on capture unit 1.

CAPFIFO = 00FFh

This clears the capture unit FIFO initially.

EVIMRA = 0080h

Enable timer1 period interrupt

EVIMRC = 1

Enable capture unit 1 interrupt

```
*****  
; File Name:      ch4_e4.asm  
; Description:   This sample program measures the frequency of an  
;               input square wave.  
*****  
    .include f240regs.h  
-----  
; Variable Declarations for on chip RAM Blocks  
-----  
    .bss GPR0,1 ;General purpose register.  
    .bss CNT,1  ;Counter for rising edges  
    .bss FREQ,1 ;Variable to store frequency value  
-----  
; Vector address declarations  
-----  
    .sect ".vectors"  
RSVECT B    START    ; Reset Vector  
INT1      B    PHANTOM ; Interrupt Level 1  
INT2      B    GPT1_ISR ; Interrupt Level 2  
INT3      B    PHANTOM ; Interrupt Level 3  
INT4      B    CAP1_ISR ; Interrupt Level 4  
INT5      B    PHANTOM ; Interrupt Level 5  
INT6      B    PHANTOM ; Interrupt Level 6  
RESERVED  B    PHANTOM ; Reserved  
SW_INT8   B    PHANTOM ; User S/W Interrupt
```

```

SW_INT9      B    PHANTOM ; User S/W Interrupt
SW_INT10     B    PHANTOM ; User S/W Interrupt
SW_INT11     B    PHANTOM ; User S/W Interrupt
SW_INT12     B    PHANTOM ; User S/W Interrupt
SW_INT13     B    PHANTOM ; User S/W Interrupt
SW_INT14     B    PHANTOM ; User S/W Interrupt
SW_INT15     B    PHANTOM ; User S/W Interrupt
SW_INT16     B    PHANTOM ; User S/W Interrupt
TRAP         B    PHANTOM ; Trap vector
NMINT        B    PHANTOM ; Non-maskable Interrupt
EMU_TRAP     B    PHANTOM ; Emulator Trap
SW_INT20     B    PHANTOM ; User S/W Interrupt
SW_INT21     B    PHANTOM ; User S/W Interrupt
SW_INT22     B    PHANTOM ; User S/W Interrupt
SW_INT23     B    PHANTOM ; User S/W Interrupt
;=====
; M A I N   C O D E   - starts here
;=====

        .text
        NOP
START:  SETC INTM          ;Disable interrupts
        SPLK #000Ah,IMR    ;Mask all core interrupts
        LACC IFR          ;Read Interrupt flags
        SACL IFR          ;Clear all interrupt flags

        CLRC SXM          ;Clear Sign Extension Mode
        CLRC OVM          ;Reset Overflow Mode
        CLRC CNF          ;Config Block B0 to Data mem

        LDP #00E0h        ;DP for addresses 7000h-707Fh
        SPLK #0041h,CKCR0 ;Disable PLL, necessary for
                        ;modifying CKCR1
        SPLK #00B8h,CKCR1 ;CLKIN(OSC)=10MHz, CPUCLK=5MHz
        SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable, SYSCLK=CPUCLK/2
        SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK
        SPLK #006Fh, WDCR ;Disable WD if VCCP=5V(JP5 pos. 2-3)
        KICK_DOG          ;Reset Watchdog
;-----
;          SET UP DIGITAL I/O PORT
;-----

        LDP #225

```

```

SPLK #3800h, OCRA;Configure pin TxPWM/TxCMP, x=1,2,3
SPLK #00F0h, OCRB;Configure CAP1 input pin
;-----
; RESET SECTION - BEGINS
;-----

LDP #232
SPLK #0, GPTCON
SPLK #0, T1CON
SPLK #0, T2CON
SPLK #0, T3CON

SPLK #0, COMCON
SPLK #0, ACTR
SPLK #0, SACTR
SPLK #0, DBTCON
SPLK #0, CAPCON

SPLK #0FFFFh, EVIFRA ;Reset interrupt flag registers
SPLK #0FFFFh, EVIFRB
SPLK #0FFFFh, EVIFRC

SPLK #0, EVIMRA ;Reset interrupt mask registers
SPLK #0, EVIMRB
SPLK #0, EVIMRC
;-----
; RESET SECTION - ENDS
;-----

LDP #00E8h
SPLK #9897h, T1PR;Period=1 second (39063)
SPLK #0055h, GPTCON ;Enable compare outputs of all
;GPTs. GPT1 compare output "Active
;High"
SPLK #0A040h, CAPCON ;Enable capture units 1 and 2
;Select GP Timer 2 as time base for
;capture unit 1
;Detect rising edge
SPLK #00FFh, CAPFIFO ;Clear capture unit FIFO initially
SPLK #0080h, EVIMRA ;Enable timer1 period interrupt
SPLK #1, EVIMRC ;Enable capture unit 1 interrupt

LDP #0

```

```

SPLK  #0, CNT    ;Initialize the edge counter

LDP   #00E8h
LACC  EVIFRC    ;ACC=Interrupt Flags of EVIFRC
SACL  EVIFRC    ;EVIFRC=ACC => clear all flags
LACC  EVIFRA    ;ACC=Interrupt Flags of EVIFRA
SACL  EVIFRA    ;EVIFRA=ACC => clear all flags

LDP   #0
LACC  IFR       ;ACC=Interrupt Flags of IFR
SACL  IFR       ;IFR=ACC => clear all flags

LDP   #00E8h
SPLK  #1746h, T1CON ;ENABLE GPT1
      ;Input clock prescaler 128
      ;Enable timer operations

CLRC  INTM      ;Enable maskable interrupts
WAIT  B        WAIT

;=====
; ISR - GPT1_ISR
; Description: Store the number of rising edges of input signal
;              every 1 second. Resets the counter which counts
;              the number of rising edges.
; Modifies:  FREQ, CNT
;=====
GPT1_ISR LDP #0
        LACC CNT      ;Load rising edge number into
        SACL FREQ     ;Store the product as frequency
        SPLK #0, CNT  ;Reset rising edge counter
        LDP #00E8h
        LACC EVIVRA   ;Reading Vector Register clears
                    ;Interrupt Flags
        CLRC INTM     ;Enable maskable interrupts
        RET          ;Return from interruption
;=====
; ISR - CAP1_ISR
; Description: Counts the number of rising edges of input signal
; Modifies:  FREQ, CNT
;=====

```

```

CAP1_ISR LDP #0
        LACC CNT
        ADD #1
        SACL CNT      ;Increment the edge counter by 1
        LDP #00E8h
        LACC EVIVRC   ;Reading Vector Register clears
                    ;Interrupt Flags
        CLRC INTM     ;Enable maskable interrupts
        RET          ;Return from interruption
;=====
ISR - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM KICK_DOG      ;Resets WD counter
        B PHANTOM

```

Example 5

Write a program that displays the number of seconds on the 8 LEDs on EVB.

Solution:

The GP timer1 is set to interrupt the processor every 1ms. The ISR has 2 counters, MSEC_CTR that counts the number of milliseconds and SEC_CTR that counts the number of seconds and outputs the number to the LEDs.

Registers involved:

T1PR = 5000

The calculation of the value to be loaded in T1PR is as follows-

Period Value = {(CLKINOSC/PRESCALER) × (PLL MTPLN RATIO /PLL DIVIDE)} / DESIRED FREQ

CLKINOSC = 10MHz. This is the input oscillator frequency and is select by bits 7-4 of the CKCR1 register.

PLL DIVIDE = 2. Bit 0 of CKCR0 decides whether this system pre-scale value is 1 or 2.

PLL MTPLN RATIO = 4. This is selected by bits 2-0 of CKCR1 which are 011 in this case.

PRESCALER = 4. This is set by bits 10-8 of T1CON.

Thus, Period Value = $\{(10^7/4) \times (4/2)\} / 1000 = 5000$.

GPTCON = 0h

T1CON = 1240h

Select continuous up counting mode, Prescaler = 4, enable timer compare operation, enable timer operation.

```
;*****
; File Name:   ch4_e5.asm
; Description: This counts the number of seconds elapsed and displays
;              the number on the LEDs on EVB.
;*****
        .include f240regs.h
LEDS     .set   000Ch   ;LEDS Register

        .bss GPR0,1
        .bss MSEC_CTR,1
        .bss SEC_CTR,1

;-----
; Vector address declarations
;-----

        .sect   ".vectors"

RSVECT B   START      ; Reset Vector
INT1    B   PHANTOM   ; Interrupt Level 1
INT2    B   GISR2     ; Interrupt Level 2
INT3    B   PHANTOM   ; Interrupt Level 3
INT4    B   PHANTOM   ; Interrupt Level 4
INT5    B   PHANTOM   ; Interrupt Level 5
INT6    B   PHANTOM   ; Interrupt Level 6
RESERVED B   PHANTOM   ; Reserved
SW_INT8 B   PHANTOM   ; User S/W Interrupt
SW_INT9 B   PHANTOM   ; User S/W Interrupt
SW_INT10 B  PHANTOM   ; User S/W Interrupt
SW_INT11 B  PHANTOM   ; User S/W Interrupt
SW_INT12 B  PHANTOM   ; User S/W Interrupt
SW_INT13 B  PHANTOM   ; User S/W Interrupt
SW_INT14 B  PHANTOM   ; User S/W Interrupt
```

```

SW_INT15    B    PHANTOM    ; User S/W Interrupt
SW_INT16    B    PHANTOM    ; User S/W Interrupt
TRAP        B    PHANTOM    ; Trap vector
NMINT       B    PHANTOM    ; Non-maskable Interrupt
EMU_TRAP    B    PHANTOM    ; Emulator Trap
SW_INT20    B    PHANTOM    ; User S/W Interrupt
SW_INT21    B    PHANTOM    ; User S/W Interrupt
SW_INT22    B    PHANTOM    ; User S/W Interrupt
SW_INT23    B    PHANTOM    ; User S/W Interrupt

;=====
; M A I N    C O D E    - starts here
;=====

        .text
        NOP
START:  SETC INTM          ;Disable interrupts
        SPLK #0002h,IMR    ;Mask all core interrupts
                        ;except INT2
        LACC IFR          ;Read Interrupt flags
        SACL IFR          ;Clear all interrupt flags

        CLRC SXM         ;Clear Sign Extension Mode
        CLRC OVM         ;Reset Overflow Mode
        CLRC CNF         ;Configure Block B0 to Data memory

        LDP #00E0h        ;DP for addresses 7000h-707Fh
        SPLK #0041h,CKCR0 ;Disable PLL, necessary for
                        ;modifying CKCR1
        SPLK #00BBh,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
        SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2
        SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK
        SPLK #006Fh,WDCR  ;Disable WD if VCCP=5V
                        ;(JP5 in pos. 2-3)
        KICK_DOG         ;Reset Watchdog

        SPLK #0h,GPR0     ;Set wait state generator for:
        OUT  GPR0,WSGR    ;Program Space, 0 wait states
                        ;Data Space, 0 wait states
                        ;I/O Space, 0 wait states

        SPLK #0h,SEC_CTR
        OUT  SEC_CTR,LEDS ;Turn off all LEDs as initialization

```

```

LDP #00E1h
SPLK #0FFFh, OCRA ;Configure pins to primary functions

LDP #00E8h
SPLK #0080h, EVIMRA ;Enable timer 1 period interrupt
LACC EVIFRA ;ACC=Interrupt Flags of EVIFRA
SACL EVIFRA ;EVIFRA=ACC => clear all flags

LDP #00E8h
SPLK #0h,GPTCON ;GP timers configured (no compare)
SPLK #5000,T1PR ;Period = 1 ms
SPLK #0h,T1CNT ;Initial value of the counter
SPLK #1240h,T1CON ;Input clock prescaler = 4
;Disable timer compare operation
;Enable timer operations
; (1244h OK too)

CLRC INTM
WAIT B WAIT
;=====
; I S R - GISR2
; Description: Calculates the number of seconds elapsed and
; accordingly outputs the numbers to the LEDs
; Modifies: MSEC_CTR, SEC_CTR
;=====
GISR2 LDP #0h
LACC MSEC_CTR
ADD #1
SACL MSEC_CTR ;Increment millisecond counter
SUB #1000 ;Test if it reaches 1000
BCND BR1,NEQ ;If not, branch to BR1
SPLK #0, MSEC_CTR ;Else reset millisec counter
LACC SEC_CTR
ADD #1
SACL SEC_CTR ;Increment second counter
OUT SEC_CTR,LEDS ;Display second counter content
SUB #0FFh ;Test if it reaches its maximum
BCND BR1, NEQ ;If not, branch to BR1
SPLK #0, SEC_CTR ;Else reset second counter
BR1 LDP #00E8h
LACC EVIVRA ;Reading Vector Register clears

```

```

                                ;Interrupt Flags
CLRC INTM          ;Enable maskable interrupts
RET                ;Return from interruption

;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM KICK_DOG      ;Resets WD counter
        B PHANTOM
```

LABORATORY EXPERIMENT 3

TIMER OPERATIONS

Objectives

In this lab, the students will learn the operations of timer functions found in the Event module of TMS320F240. The timer functions covered in this lab are compare, input capture, and pulse-width modulation functions. Students will write programs to control the operations of these timer functions for various applications including waveforms generations and period/frequency measurements.

Equipment Required

Hardware :

- PC Specifications -
 - ❑ '386 or higher IBM PC/AT
 - ❑ 1.44Mb 3.5-inch floppy drive
 - ❑ 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
 - ❑ Minimum 4Mb memory
 - ❑ Color VGA Monitor

- TMS320C24x Evaluation Board
- XDS510PP Emulator
- +5V power supply.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable

Discussion

Timer Operations

The DSP provides various timer functions in its General Purpose Timers. In this lab, we will be performing experiments related to the operations of compare, input capture, and pulse width modulation functions. To be able to perform the experiments, you should study the materials in Chapter 4 that describes in detail the operations of the timer functions and provides several useful example programs. For a quick reference on programming the timers, refer to Appendix A.

Procedure

Setup

1. Make sure that the EVB system has been properly setup as in the previous lab.
2. Turn on the PC and run the EVB Testing program.

Laboratory Assignments

1. Square-wave signals generation using output compare function

Write a program for that outputs a square-wave with frequency of 2 kHz at pin T2PWM/T2CMPR/IOPB4.

- 1) Compile the program and download it to the EVB.
- 2) Connect the pin T2PWM/T2CMPR/IOPB4 (pin 13 on connector P1) and GND (pin 33 on connector P1) to oscilloscope.
- 3) Run the downloaded program.
- 4) Verify that the signal has the desired frequency on oscilloscope.
- 5) Repeat step 1 to 4 above for the following square wave signal frequencies :
 - (a) 20 kHz
 - (b) 100 kHz
 - (c) 500 Hz
 - (d) 60 Hz

2. PWM signals generation

- 1) Write a program that outputs a 500 Hz PWM signal with 30% duty cycle.
- 2) Compile the program and download it.
- 3) Run the downloaded program.
- 4) Verify that the signal has the desired frequency and duty cycle on oscilloscope.
- 5) Change the values of the duty cycle in your program to the following:
(a) 75% (b) 50% (c) 25% (d) 10 %
and repeat step 2 to 4
- 6) Investigate the result of your program for high duty cycle PWM signals. Try different duty cycle values which are close to 100 % and observe the results. Does your program produce the intended results? Explain what happens when your program tries to generate PWM signals with duty cycle close to 100%. Obtain the maximum duty cycle your program can generate! Does your program have the same problems when generating PWM signal with low duty cycle? Explain.

3. Period and Frequency Measurements using Capture Functions

- 1) Write a program that measures the period and frequency of a square wave signal. Use the input capture pin CAP2 as the signal input.
- 2) Connect a square wave input from the signal generator to CAP2 (pin 22 of connector P1) and GND (pin 33 of connector P1)
- 3) Compile, download and run your program.
- 4) Tabulate results for the following input frequencies from signal generator:
 - 100 Hz
 - 1 kHz
 - 50 Hz

CHAPTER 5

ANALOG INTERFACING: Analog to Digital Converter

3.1 Introduction

This chapter discusses the ADC (Analog to Digital Converter) module of the TMS320F240. The evaluation board has an on-board Digital to Analog converter module, which will be discussed here too.

3.2 Analog to Digital Converter

The TMS320F240 has two ADC modules, each of which has a 10-bit A to D converter. The ADC accepts an analog input voltage of up to 5V and converts it into a proportional 10-bit digital value. The basic principle of an ADC can be explained by the following block diagram-

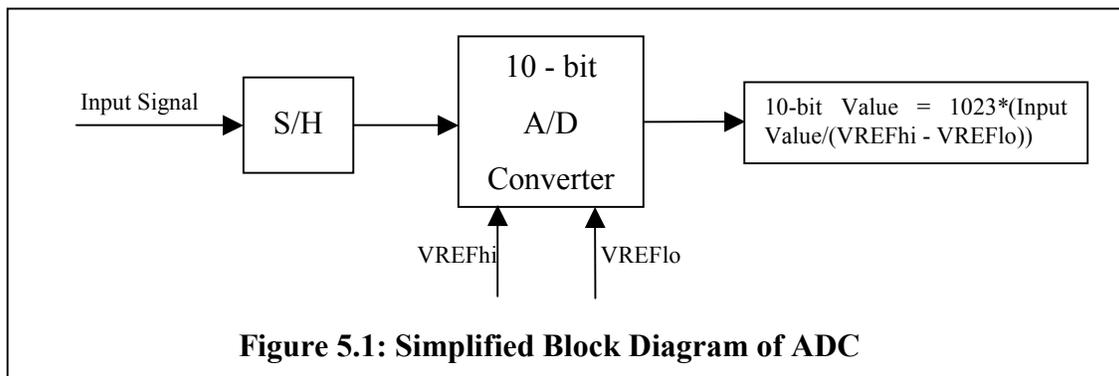


Figure 5.1: Simplified Block Diagram of ADC

The input signal is sampled by the S/H (sample and hold) block, which passes a sample of input to the A/D converter. The A/D converter reads this value and converts it into a 10-bit digital value. The voltage at VREFlo i.e. the low reference voltage corresponds to 0000000000b (0h) and the voltage at VREFhi i.e. high reference voltage is 111111111b

(03FFh or 1023d). Thus, the digital value of the input voltage can be computed as follows-

$$\text{Digital Result} = 1023 * \{\text{Input Voltage} / (\text{VREFhi} - \text{VREFlo})\}$$

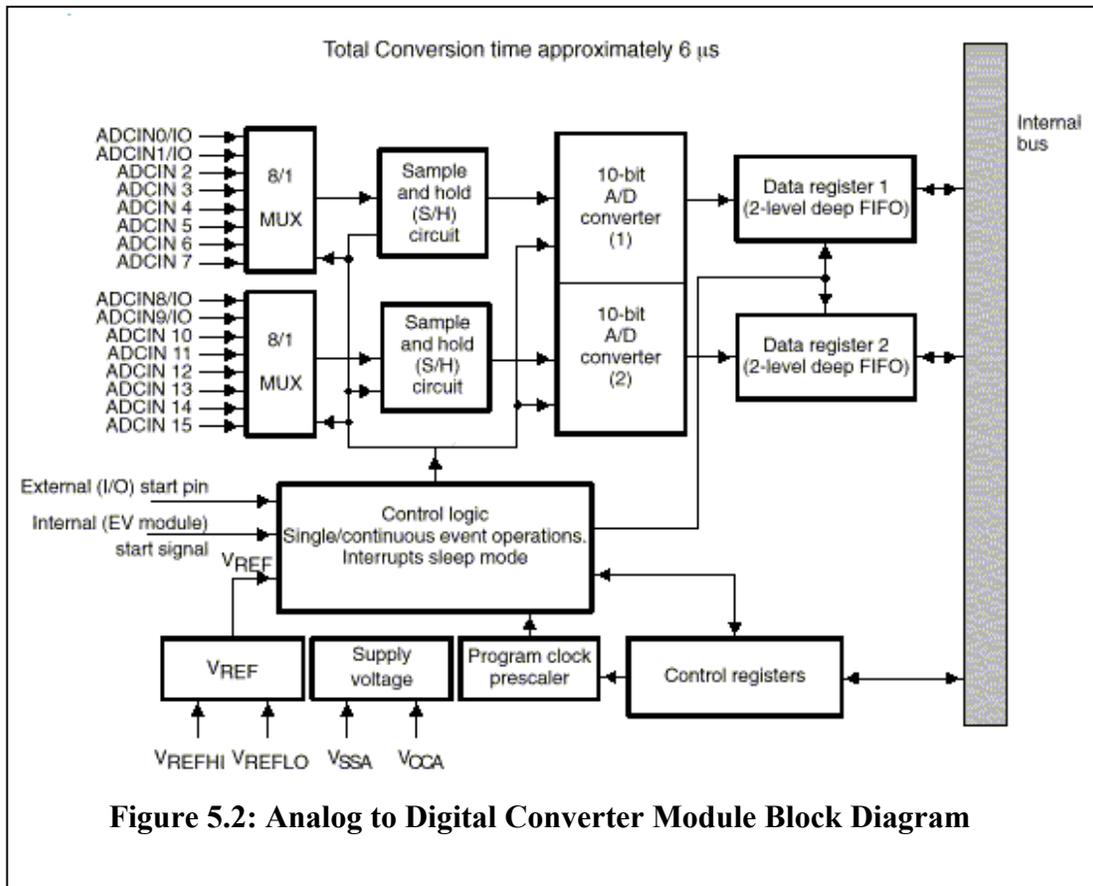
The various features of the TMS320F240 ADC module are listed below-

- ❑ 16 multiplexed analog input channels with 8 channels per ADC units.
- ❑ Simultaneous measurement of two analog channels using two ADC units.
- ❑ Single and Continuous conversion.
- ❑ Conversion can be software, internal event or external event initiated.
- ❑ Upper and Lower reference voltages up to 5V can be supplied externally.
- ❑ Two level deep digital registers to store the values of the completed conversions.
- ❑ Total maximum conversion time of 6.6µs.
- ❑ Programmable pre-scaler select to choose the sampling frequency. The ADC module performs input sampling in one ADC pre-scaled clock cycle and conversion in five pre-scaled clock cycles i.e. a total of 6 prescaled clock cycles. Thus, the pre-scaler must satisfy the following formula -

$$\text{SYSCLK Period} * \text{Prescaler Value} * 6 \geq 6.6\mu\text{s}.$$

- ❑ Two programmable ADC module control registers.
- ❑ Interrupt or polled operation i.e. an interrupt can be generated at the end of each conversion or else the end of conversion (EOC) flag can be polled.

The features are shown in Figure 5.2 below:



ADC Module Pin Description

This module has a total of 20 pins that interface with the external circuitry. ADC0-15 are for the 16 input channels, 2 pins VREFHI and VREFLO for the reference voltages and V_{CCA}, V_{SSA} are the analog supply pins.

ADC Registers

The various registers used by the ADC module are-

Address	Register	Name
7032h	ADCTRL1	ADC Control Register 1
7034h	ADCTRL2	ADC Control Register 2
7036h	ADCFIFO1	ADC 2-Level Deep Data Register FIO for ADC 1
7038h	ADCFIFO2	ADC 2-Level Deep Data Register FIO for ADC 2

ADC Control Register 1 (ADCTRL1)

15	14	13	12	11	10	9	8
Suspend-soft	Suspend-free	ADCIMSTART	ADC2EN	ADC1EN	ADCCONRUN	ADCINTEN	ADCINTFLAG
RW-0	RW-0	RW-0	SRW-0	SRW-0	SRW-0	SRW-0	RW-0
7	6-4			3-1		0	
ADCEOC	ADC2CHSEL			ADC1CHSEL			ADCSOC
R-0	SRW-0			SRW-0			SRW-0

Note: R = Read access, W = Write access, S = Shadowed, -0 -= value after reset

Suspend-soft (Bit 15)

0 = Stop immediately when suspend-free = 0.

1 = Complete conversion before halting emulator.

Suspend-free (Bit 14)

0 = Operation is determined by suspend-soft.

1 = Keep running on emulator suspend.

ADCIMSTART (Bit 13)

0 = No action

1 = Immediate start of conversion.

ADC2EN (Bit 12)

This bit can be written while a previous conversion is still going on. However, the effect of writing to this bit will take place from the next conversion.

0 = ADC2 is disabled

1 = ADC2 is enabled

ADC1EN (Bit 11)

This bit can be written while a previous conversion is still going on. However, the effect of writing to this bit will take place from the next conversion.

0 = ADC1 is disabled

1 = ADC1 is enabled

ADCONRUN (Bit 10)

This bit can be written while a previous conversion is still going on. However, the effect of writing to this bit will take place from the next conversion.

0 = No action

1 = Continuous conversion

ADCINTEN (Bit 9)

This bit can be written while a previous conversion is still going on. However, the effect of writing to this bit will take place from the next conversion. This bit is cleared on reset.

0 = Disable all ADC interrupts

1 = Enable all ADC interrupts

ADCINTFLAG (Bit 8)

Writing a 1 clears this bit.

0 = No interrupt event occurred.

1 = An interrupt event occurred.

ADCEOC (Bit 7)

0 = End of conversion.

1 = Conversion is in progress.

ADC2CHSEL (Bit 6-4)

These bits can be written while a previous conversion is still going on. However, the effect of writing to these bits will take place from the next conversion. These bits select the channels for ADC2.

000 = Channel 9 (ADCIN8)

100 = Channel 13 (ADCIN12)

001 = Channel 10 (ADCIN9)

101 = Channel 14 (ADCIN13)

010 = Channel 11 (ADCIN10)

110 = Channel 15 (ADCIN14)

011 = Channel 12 (ADCIN11)

111 = Channel 16 (ADCIN15)

ADC1CHSEL (Bit 3-1)

These bits can be written while a previous conversion is still going on. However, the effect of writing to these bits will take place from the next conversion. These bits select the channels for ADC1.

000 = Channel 1 (ADCIN0)

100 = Channel 5 (ADCIN4)

001 = Channel 2 (ADCIN1)

101 = Channel 6 (ADCIN5)

010 = Channel 3 (ADCIN2)

110 = Channel 7 (ADCIN6)

011 = Channel 4 (ADCIN3)

111 = Channel 8 (ADCIN7)

ADCSOC (Bit 0)

This bit can be written while a previous conversion is still going on. However, the effect of writing to this bit will take place from the next conversion.

0 = No action

1 = Start converting

ADC Control Register 2 (ADCTRL2)

This register selects ADC input clock prescaler, conversion mode, emulation operation and shows FIFO status.

15-11				10		9		8	
Reserved				ADCEVSOC		ADCEXTSOC		Reserved	
				SRW-0		SRW-0			
7		5		4-3		2-0			
ADCFIFO2		Reserved		ADCFIFO1		ADCPSCALE			
R-0				R-0		SRW-0			

Note: R = Read access, W = Write access, S = Shadowed, -0 -= value after reset

Reserved

Reads are indeterminate and writes have no effect.

ADCEVSOC (Bit 10)

ADC conversion can be synchronized with an event manager signal depending on a compare match.

0 = Disable conversion start by EV

1 = Enable conversion start by EV

ADCEXTSOC (Bit 9)

0 = ADC conversion cannot be synchronized by external signal

1 = ADC conversion can be synchronized by external signal

ADCFIFO2 (Bit 7-6)

00 = FIFO2 is empty

01 = FIFO2 has one entry

10 = FIFO2 has two entries

11 = FIFO2 had two entries and another entry was received; first entry has been lost.

ADCFIFO1 (Bit 4-3)

00 = FIFO1 is empty

01 = FIFO1 has one entry

10 = FIFO1 has two entries

11 = FIFO1 had two entries and another entry was received; first entry has been lost.

ADCPSCALE (Bit 2-0)

Bit 2-0	Prescale Value	Bit 2-0	Prescale Value
000	4	100	12
001	6	101	16
010	8	110	20
011	10	111	32

ADC Data Registers ADCFIFO1 and ADCFIFO2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0
MS										LSB					

B

D9-D0 Actual 10-bit converted data.

Reserved Always read as 0.

In order to understand how all these registers are configured/ accessed, let us take a simple programming example.

Example Program 1:

Configure channel 0 of ADC1 for single conversion. No interrupts need to be enabled. When the conversion ends, the FIFO value should be loaded into a memory location "RESULT".

Solution:

The registers involved are- ADCTRL1, ADCTRL2 and ADCFIFO1.

We will first program ADCCTRL1 for the following:

- To complete conversion before halting emulator.
- Select Channel 0 of ADC1.
- Disable all interrupts
- Single conversion
- Command to start conversion

Thus we have, ADCTRL1 = 0901h

We program ADCCTRL2 for:

- Do not select SOC by event manager or external signal
- Prescaler = 16. This value is selected so as to satisfy the following relation:

System Clock = 10MHz \Rightarrow 0.1 μ sec. Since each conversion takes 6 ADC clock cycles and a minimum of 6.6 μ sec, the prescaler should satisfy,

$$0.1\mu\text{sec} \times 16 \times 6 \geq 6.6\mu\text{sec}$$

Thus, ADCTRL2 = 0005h

Another important point to remember is that, the FIFO register is 16-bit long whereas the ADC is only 10-bit. The converted 10-bit value is stored in the upper 10 bits of the FIFO register. Thus, the contents of the FIFO should be shifted right in order to get the right value.

The code relevant segment is -

```

LDP #00E0h          ;Load DP with appropriate value
SPLK #0005h, ADCTRL2 ;Load control registers and
SPLK #3B60, ADCTRL1 ;start conversion
                    ;#8901h, #2900h, #0901h are also valid here
UP   LACC ADCTRL1
     AND #0080h      ;Select bit 7
     BCND UP, NEQ    ;Check for end of conversion
     LDP #00E0h
UP1  LACC ADCTRL2
     AND #00C0h      ;Select bit 7-6
     BCND UP1, EQ    ;Check for non-empty FIFO1
     LACC ADCFIFO1   ;Load from FIFO1
     RPT #5
     SFR              ;Shift contents before storing
     LDP #0
     SACL RESULT     ;Store digital result

```

Example Program 2:

Configure channel 0 of ADC1 for continuous conversion. An interrupt should be generated at the end of each conversion. The FIFO value must be loaded into a memory location "VALUE" after each conversion.

Solution:

The registers involved are:

ADCTRL1, ADCTRL2 and ADCFIFO1.

We will first program ADCCTRL1 for the following:

- To complete conversion before halting emulator.
- Select Channel 0 of ADC1.
- Enable all ADC interrupts
- Continuous conversion

Thus we have, ADCTRL1 = 8E00h

We program ADCCTRL2 for:

- Do not select SOC by event manager or external signal
- Prescaler = 16

Thus, ADCTRL2 = 0005h

INT6 is the level for ADC interrupts. When an ADC interrupt occurs, it branches to the EOC_ISR interrupt subroutine. Here, the conversion result is stored and another SOC command is given.

The complete program listing is -

```
;*****  
; Program: Ch5_e2.asm  
; Description: Channel 0 of ADC1 for continuous conversion.  
; An interrupt should be generated at the end of  
; each conversion. The FIFO value must be loaded  
; into a memory location "VALUE" after each conversion  
;*****  
    .include f240regs.h  
  
    .bss GPR0,1  
    .bss VALUE,1
```

```

;-----
; Vector address declarations
;-----
        .sect  ".vectors"

RSVECT  B    START    ; Reset Vector
INT1    B    PHANTOM  ; Interrupt Level 1
INT2    B    PHANTOM  ; Interrupt Level 2
INT3    B    PHANTOM  ; Interrupt Level 3
INT4    B    PHANTOM  ; Interrupt Level 4
INT5    B    PHANTOM  ; Interrupt Level 5
INT6    B    EOC_ISR  ; Interrupt Level 6
RESERVED B    PHANTOM  ; Reserved
SW_INT8 B    PHANTOM  ; User S/W Interrupt
SW_INT9 B    PHANTOM  ; User S/W Interrupt
SW_INT10 B    PHANTOM ; User S/W Interrupt
SW_INT11 B    PHANTOM ; User S/W Interrupt
SW_INT12 B    PHANTOM ; User S/W Interrupt
SW_INT13 B    PHANTOM ; User S/W Interrupt
SW_INT14 B    PHANTOM ; User S/W Interrupt
SW_INT15 B    PHANTOM ; User S/W Interrupt
SW_INT16 B    PHANTOM ; User S/W Interrupt
TRAP    B    PHANTOM  ; Trap vector
NMINT   B    PHANTOM  ; Non-maskable Interrupt
EMU_TRAP B    PHANTOM ; Emulator Trap
SW_INT20 B    PHANTOM ; User S/W Interrupt
SW_INT21 B    PHANTOM ; User S/W Interrupt
SW_INT22 B    PHANTOM ; User S/W Interrupt
SW_INT23 B    PHANTOM ; User S/W Interrupt

;-----
; Main program
;-----
        .text

        NOP
START:   SETC INTM      ;Disable interrupts
        SPLK #0020h,IMR ;Mask all core interrupts except INT6
        LACC IFR       ;Read Interrupt flags
        SACL IFR       ;Clear all interrupt flags

```

```

CLRC SXM          ;Clear Sign Extension Mode
CLRC OVM          ;Reset Overflow Mode
CLRC CNF          ;Config Block B0 to Data mem

LDP #00E0h        ;DP for addresses 7000h-707Fh
SPLK #00BBh,CKCR1 ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2
SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK

SPLK #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 in pos. 2-3)
KICK_DOG          ;Reset Watchdog
SPLK #0h,GPRO     ;Set wait state generator for:
OUT GPRO,WSGR     ;Program Space, 0 wait states
                  ;Data Space, 0 wait states
                  ;I/O Space, 0 wait states

LDP #00E0h
SPLK #8E00h, ADCTRL1
SPLK #0005h, ADCTRL2
SBIT1 ADCTRL1, B0_MSK ;Give start of conversion SOC command
                  ;This is same as ADCTRL1 OR 0001h
                  ;B0_msk to B15_MSK are defined in
                  ;f240regs.h

CLRC INTM        ;Enable interrupts
END B END
;=====
; I S R - EOC_ISR
; Description: Store the conversion result to variable "VALUE"
;=====
EOC_ISR LDP #00E0h
        LACC ADCTRL1
        SACL ADCTRL1 ;Clear any ADC interrupt that occurred
        LACC ADCFIFO1
        RPT #5
        SFR ;Shift contents of FIFO before storing
        LDP #0
        SACL VALUE ;Store result
        LDP #00E0h
        SBIT1 ADCTRL1, B0_MSK ;Give SOC
        CLRC INTM
        RET

```

```

;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM    KICK_DOG          ;Resets WD counter
           B PHANTOM

```

Example Program 3:

Configure channel 0 of ADC1 for continuous conversion. Generate a PWM output of constant frequency using timer 1. The duty cycle of the PWM wave varies directly as per the input.

Solution:

The basic scheme here is to enable the timer period interrupt. Thus, every period, the timer compare register is loaded with the value in the ADCFIFO (shifted appropriately) for the most recent conversion. Thus, every period the compare register is loaded with a new value. This operation is performed in the ISR for the timer period interrupt.

The registers involved are: ADCTRL1, ADCTRL2, ADCFIFO1, T1CON, GPTCON and T1PER..

We will first program ADCCTRL1 for the following:

- To complete conversion before halting emulator.
- Select Channel 0 of ADC1.
- Disable all ADC interrupts (only use timer interrupt)
- Continuous conversion

Thus we have, ADCTRL1 = 8D00h

We program ADCCTRL2 for:

- Do not select SOC by event manager or external signal
- Prescaler = 16

Thus, ADCTRL2 = 0005h

We program GPTCON for:

- Enable compare outputs of all timers
- Set all timer compare outputs to active low.

Thus, GPTCON = 0055h

We program T1CON for:

- Stop immediately on emulation suspend.
- Continuous up counting mode
- Prescaler = 1
- Internal clock source
- Reload timer compare register immediately.
- Enable timer compare operation

Thus, T1CON = 100Ah

Also, we enable the Timer Period interrupt by setting the appropriate bits in the IMR and EVIMRA registers.

The complete program listing is -

```
*****  
; Program: Ch5_e3.asm  
; Description: Channel 0 of ADC1 for continuous conversion.  
;           Generate a PWM output of constant frequency using  
;           timer 1. The duty cycle of the PWM wave varies  
;           directly as per the input  
;*****  
    .include f240regs.h  
  
T1COMPARE .set 0h  
T1PERIOD  .set 03FFh  
  
    .bss GPR0,1  
  
;-----  
; Vector address declarations  
;-----  
    .sect  ".vectors"
```

```

RSVECT   B   START       ; Reset Vector
INT1     B   PHANTOM    ; Interrupt Level 1
INT2    B   CHG_CMPR   ; Interrupt Level 2
INT3     B   PHANTOM    ; Interrupt Level 3
INT4     B   PHANTOM    ; Interrupt Level 4
INT5     B   PHANTOM    ; Interrupt Level 5
INT6     B   PHANTOM    ; Interrupt Level 6
RESERVED B   PHANTOM    ; Reserved
SW_INT8  B   PHANTOM    ; User S/W Interrupt
SW_INT9  B   PHANTOM    ; User S/W Interrupt
SW_INT10 B   PHANTOM    ; User S/W Interrupt
SW_INT11 B   PHANTOM    ; User S/W Interrupt
SW_INT12 B   PHANTOM    ; User S/W Interrupt
SW_INT13 B   PHANTOM    ; User S/W Interrupt
SW_INT14 B   PHANTOM    ; User S/W Interrupt
SW_INT15 B   PHANTOM    ; User S/W Interrupt
SW_INT16 B   PHANTOM    ; User S/W Interrupt
TRAP     B   PHANTOM    ; Trap vector
NMINT    B   PHANTOM    ; Non-maskable Interrupt
EMU_TRAP B   PHANTOM    ; Emulator Trap
SW_INT20 B   PHANTOM    ; User S/W Interrupt
SW_INT21 B   PHANTOM    ; User S/W Interrupt
SW_INT22 B   PHANTOM    ; User S/W Interrupt
SW_INT23 B   PHANTOM    ; User S/W Interrupt
;-----
; Main program
;-----
        .text
        NOP
START:  SETC INTM          ;Disable interrupts
        SPLK #0002h,IMR    ;Mask all core interrupts except timer 1
                                ;period interrupt
        LACC IFR          ;Read Interrupt flags
        SACL IFR          ;Clear all interrupt flags

        CLRC SXM          ;Clear Sign Extension Mode
        CLRC OVM          ;Reset Overflow Mode
        CLRC CNF          ;Config Block B0 to Data mem

        LDP  #00E0h        ;DP for addresses 7000h-707Fh
        SPLK #00BBh,CKCR1 ;CLKIN(OSC)=10MHz, CPUCLK=20MHz

```

```

SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable,SYSClk=CPUCLK/2
SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK

SPLK #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 in pos. 2-3)
KICK_DOG           ;Reset Watchdog

SPLK #0h,GPR0      ;Set wait state generator for:
OUT   GPR0,WSGR    ;Program Space, 0 wait states
                        ;Data Space, 0 wait states
                        ;I/O Space, 0 wait states

LDP #00E1h         ;Point to appropriate data page
SPLK #380Fh, OCRA ;Select the T1PWM function for the I/O pin

```

```

;-----
; RESET SECTION - BEGINS
;-----

```

```

LDP #00E8h
SPLK #0, GPTCON
SPLK #0, T1CON
SPLK #0, T2CON
SPLK #0, T3CON

SPLK #0, COMCON
SPLK #0, ACTR
SPLK #0, SACTR
SPLK #0, DBTCON
SPLK #0, CAPCON

SPLK #0FFFFh, EVIFRA
SPLK #0FFFFh, EVIFRB
SPLK #0FFFFh, EVIFRC

SPLK #0, EVIMRA
SPLK #0, EVIMRB
SPLK #0, EVIMRC

```

```

;-----
; RESET SECTION - ENDS
;-----

```

```

LDP #00E8h           ;Load DP for timer registers
SPLK #T1COMPARE, T1CMPR ;Set initial value for the

```

```

;compare register
SPLK #0055h, GPTCON
SPLK #T1PERIOD, T1PR ;Set period value=3FFh
SPLK #0, T1CNT
SPLK #0, T2CNT
SPLK #0, T3CNT

SPLK #100Ah, T1CON ;Continuous up-counting
;Prescaler=1
;Disable timer operation
;Compare register reload immediately
;Enable timer compare operation

SPLK #0, T2CON
SPLK #0, T3CON
SPLK #0080h, EVIMRA ;Enable timer 1 period interrupt

LDP #00E0h ;Load DP for ADC registers
SPLK #8D00h, ADCTRL1 ;ADC1 is enabled
;Continuous conversion
;Disable all ADC interrupts

SPLK #0005h, ADCTRL2

LDP #00E8h ;Load DP for timer registers
SBIT1 T1CON, B6_MSK ;Enable timer 1
LDP #00E0h ;Load DP for ADC registers
SBIT1 ADCTRL1, B0_MSK ;Start conversion for ADC1

CLRC INTM ;Enable interrupts
END B END

;=====
; I S R - CHG_CMPR
; Description: Timer 1 Period ISR, modifies the value of timer 1 compare
; register depending upon analog input.
;=====
CHG_CMPR LDP #224 ;Load DP for ADC registers
LACC ADCTRL1
SACL ADCTRL1 ;Clear ADC interrupts
LACC ADCFIFO1
RPT #5

```

```

SFR                                ;Read FIFO and shift appropriately

LDP  #232                              ;Load DP for timer registers
SACL T1CMPR                          ;Load ADC conversion result in timer
                                          ;compare register to adjust duty cycle.

LACC EVIFRA
SACL EVIFRA                            ;Clear timer period interrupts

CLRC INTM                              ;Enable interrupts
RET

;=====
; I S R  -  PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM  KICK_DOG                      ;Resets WD counter
        B  PHANTOM

```

3.3 Digital to Analog Converter

The TMX320x24x EVB has an on-board 12-bit digital to analog converter (DAC) module. As the name suggests, a DAC generates an analog voltage proportional to the digital input fed to it. We shall employ this module for generating different voltages that will act as inputs to the ADC.

The DAC module has four channels. These four channels and the DAC update register are mapped onto the DSP I/O space as follows-

Register Name	Register Address	Description
DAC0	0000h	Input data register for DAC0
DAC1	0001h	Input data register for DAC1
DAC2	0002h	Input data register for DAC2
DAC3	0003h	Input data register for DAC3
DAC update	0004h	DAC update register

The DAC module requires that wait states be generated for proper operation. The DSP must therefore be programmed to generate one software wait state for I/O space accesses and the 20-MHz CPUCLK signal must be output on the CLKOUT pin of the device. The following code shows how to generate the appropriate number of wait states. The following code segment does this -

```

LDP    #00E0h          ;DP for addresses 7000h-707Fh
SPLK   #00BBh,CKCR1   ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
SPLK   #00C3h,CKCR0   ;CLKMD=PLL Enable,SYSCLK=CPUCLK/2
SPLK   #40C0h,SYSCR    ;CLKOUT=CPUCLK

LDP    #0000h
SPLK   #4h,GPR0       ;Set wait state generator for:
OUT    GPR0,WSGR      ;Program Space, 0 wait states
                          ;Data Space, 0 wait states
                          ;I/O Space, 1 wait states

```

The digital value to be converted must be written to the appropriate DAC input data register. A value must be written to the DAC update register in order to start the conversions. The outputs can be observed some pins on the P2 connector as discussed in a later section.

The code for writing data to the DAC and outputting the analog voltages is as follows-

```

LDP    #0000h
SPLK   #03FFFh,DAC0VAL ;Load 03FFFh into DAC0VAL register
SPLK   #07FFFh,DAC1VAL ;Load 07FFFh into DAC1VAL register
SPLK   #0BFFFh,DAC2VAL ;Load 0BFFFh into DAC2VAL register
SPLK   #0FFFFh,DAC3VAL ;Load 0FFFFh into DAC3VAL register

OUT    DAC0VAL,0000h   ;Write 03FFFh to the DAC0 register
OUT    DAC1VAL,0001h   ;Write 07FFFh to the DAC1 register
OUT    DAC2VAL,0002h   ;Write 0BFFFh to the DAC2 register
OUT    DAC3VAL,0003h   ;Write 0FFFFh to the DAC3 register

OUT    DAC3VAL,0004h   ;Start DAC conversions by writing a
                          ;value to the DAC update register

```

Ref: Texas Instruments Literature # SPRU248A, August 1997

Assuming that VREFhi and VREFlo pins are 5V and 0V respectively, the voltages at the DACOUT pins for the various values are-

DAC Register	Value in Register	Output Pin	Voltage at Pin
DAC0	03FFh	DACOUT0	1.25V
DAC1	07FFh	DACOUT1	2.50V
DAC2	0BFFh	DACOUT2	3.75V
DAC3	0FFFh	DACOUT3	5.00V

Example Program 4:

Write a program to output a staircase waveform with steps 0V, 1.25V, 2.5V, 3.75V, 5V and frequency 50Hz at the pin DAC0OUT.

Solution:

Timer 1 interrupt is used to specify the timing of each state. In the ISSR, a counter is maintained for the state and the appropriate voltage is output at the DAC. The relevant portion of the code is listed below.

The frequency for the staircase waveform is 50Hz, which means the period is 20ms. Since there are 5 stairs in total, the period for each stair should be 4ms – stair changes once every 4ms. Therefore, the frequency for stair change is 250Hz. From last chapter,

Period Value = $\{(CLKINOSC/PRESCALER) \times (PLL\ MTPLN\ RATIO / PLL\ DIVIDE)\} / DESIRED\ FREQ$

CLKINOSC=10MHz;

PRESCALER=32;

PLL MULTIPLICATION RATIO=1;

PLL DIVIDE=2;

DESIRED FREQ=250Hz

Period Value= $\{(10^7/32) \times (1/2)\} / 250 = 625$.

The complete program is shown below:

```

;*****
; File Name:    ch5_e4.asm
; Description:  This sample program outputs a staircase waveform
;              with steps 0V, 1.25V, 2.5V, 3.75V, and 5V of 50Hz
;              at the pin DAC0OUT.
;*****
        .include f240regs.h

DAC0    .set 0h
DACUPDATE .set 0004h

;-----
; Variable Declarations for on chip RAM Blocks
;-----

        .bss GPR0,1    ;General purpose register.
        .bss CNT,1     ;Counter of the stairs
        .bss DAC0VAL, 1 ;Digital value to be converted

;-----
; Vector address declarations
;-----

        .sect ".vectors"
RSVECT   B   START    ; Reset Vector
INT1     B   PHANTOM  ; Interrupt Level 1
INT2     B   GPT1_ISR ; Interrupt Level 2
INT3     B   PHANTOM  ; Interrupt Level 3
INT4     B   PHANTOM  ; Interrupt Level 4
INT5     B   PHANTOM  ; Interrupt Level 5
INT6     B   PHANTOM  ; Interrupt Level 6
RESERVED B   PHANTOM  ; Reserved
SW_INT8  B   PHANTOM  ; User S/W Interrupt
SW_INT9  B   PHANTOM  ; User S/W Interrupt
SW_INT10 B   PHANTOM  ; User S/W Interrupt
SW_INT11 B   PHANTOM  ; User S/W Interrupt
SW_INT12 B   PHANTOM  ; User S/W Interrupt
SW_INT13 B   PHANTOM  ; User S/W Interrupt
SW_INT14 B   PHANTOM  ; User S/W Interrupt
SW_INT15 B   PHANTOM  ; User S/W Interrupt
SW_INT16 B   PHANTOM  ; User S/W Interrupt
TRAP     B   PHANTOM  ; Trap vector
NMINT    B   PHANTOM  ; Non-maskable Interrupt
EMU_TRAP B   PHANTOM  ; Emulator Trap

```

```

SW_INT20 B    PHANTOM ; User S/W Interrupt
SW_INT21 B    PHANTOM ; User S/W Interrupt
SW_INT22 B    PHANTOM ; User S/W Interrupt
SW_INT23 B    PHANTOM ; User S/W Interrupt

;=====
; M A I N   C O D E   - starts here
;=====

    .text
    NOP
START: SETC INTM           ;Disable interrupts
       SPLK #0002h,IMR     ;Mask all core interrupts
       LACC IFR           ;Read Interrupt flags
       SACL IFR           ;Clear all interrupt flags

       CLRC SXM           ;Clear Sign Extension Mode
       CLRC OVM           ;Reset Overflow Mode
       CLRC CNF           ;Config Block B0 to Data mem

       LDP  #00E0h        ;DP for addresses 7000h-707Fh
       SPLK #00B8h,CKCR1 ;CLKIN(OSC)=10MHz, CPUCLK=5MHz
       SPLK #0041h,CKCR0 ;CLKMD=PLL Disable, SYSCLK=CPUCLK/2
       SPLK #00C3h,CKCR0 ;CLKMD=PLL Enable
       SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK

       SPLK #006Fh, WDCR ;Disable WD if VCCP=5V (JP5 in pos. 2-3)
       KICK_DOG           ;Reset Watchdog

       LDP  #0
       SPLK #0004h, GPR0 ;wait states for DAC
       OUT  GPR0, WSGR

;-----
;   RESET SECTION - BEGINS
;-----

       LDP  #00E8h
       SPLK #0, GPTCON
       SPLK #0, T1CON
       SPLK #0, T2CON
       SPLK #0, T3CON

       SPLK #0, COMCON
       SPLK #0, ACTR

```

```

SPLK #0, SACTR
SPLK #0, DBTCON
SPLK #0, CAPCON

SPLK #0FFFFh, EVIFRA
SPLK #0FFFFh, EVIFRB
SPLK #0FFFFh, EVIFRC

SPLK #0, EVIMRA
SPLK #0, EVIMRB
SPLK #0, EVIMRC

;-----
; RESET SECTION - ENDS
;-----

LDP #00E8h
SPLK #625, T1PR ;Period=4ms, for one stair
;so 5 stairs in total=20ms

SPLK #0055h, GPTCON
SPLK #0h, T2CON
SPLK #0h, T3CON

SPLK #0080h, EVIMRA ;Enable timer 1 period interrupt
LACC EVIFRA
SACL EVIFRA

LDP #0
LACC IFR
SACL IFR
SPLK #0, CNT

LDP #00E8h
SPLK #1546h, T1CON ;Continuous up-counting, Prescaler=32
;Enable timer operation
;Enable timer compare operation
CLRC INTM ;Enable interrupts

WAIT B WAIT

;=====
; I S R - GPT1_ISR

```

```

; Description: Keep track of the state of the wave and accordingly
;           output a voltage at channel 0 of DAC
;=====
GPT1_ISR  LDP  #0
          LACC #DACTBL  ;Set pointer to start of table
          ADD  CNT      ;Point to appropriate value in table
          TBLR DAC0VAL  ;depending on the stair counter CNT
          OUT  DAC0VAL, DAC0
          OUT  DAC0VAL, DACUPDATE ;Trigger the D/A conversion
          LACC CNT
          SUB  #4
          BCND NXT, NEQ ;Check if CNT reaches 4
          SACL CNT      ;If CNT=4, reset it to 0
          B   NXT1
NXT       LACC CNT      ;If CNT<4, increment by 1
          ADD  #1
          SACL CNT      ;Store the new CNT value
NXT1      LDP  #00E8h
          LACC EVIVRA
          CLRC INTM     ;Enable interrupts
          RET
;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM   KICK_DOG      ;Resets WD counter
          B   PHANTOM

DACTBL    .word  0000h
          .word  03FFh
          .word  07FFh
          .word  0BFFh
          .word  0FFFh

```

LABORATORY EXPERIMENT 4

Objectives

The objective of this lab session is to familiarize the students with the analog to digital converter (ADC) of the TMS320F240. In this session, the students will write and test assembly language programs that use TMS320F240 ADC to read several analog inputs. Also, the student will generate different analog signals employing the digital to analog converter (DAC) on the evaluation board.

Equipment Required

Hardware :

- PC Specifications -
 - ❑ '386 or higher IBM PC/AT
 - ❑ 1.44Mb 3.5-inch floppy drive
 - ❑ 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
 - ❑ Minimum 4Mb memory
 - ❑ Color VGA Monitor
- TMS320C24x Evaluation Board
- XDS510PP Emulator
- +5V power supply.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable

Software :

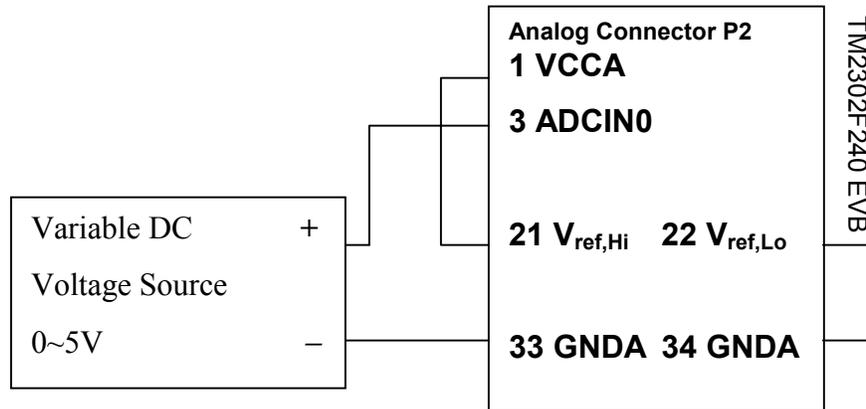
- MS Windows 3.1 or Windows 95
- ASCII Editor
- TMS320C2xx Assembler
- TMS320C2xx C Source Debugger

Discussion

Procedure

Setup

1. Setup the EVB and PC as discussed in LAB1.
2. Make the following connections for problem 1.



Laboratory Assignments

Problem 1

1. Configure channel 0 of ADC1 for continuous conversion. No interrupts need to be enabled. When the conversion ends, the FIFO value should be loaded into a memory location "RESULT". Connect the dc source as analog input. Observe the value of RESULT for different values of RESULT in the "Watch" window of the debugger. Tabulate the readings as follows-

Analog Input (volts)	RESULT
0	
1	
2	
3	
4	
5	

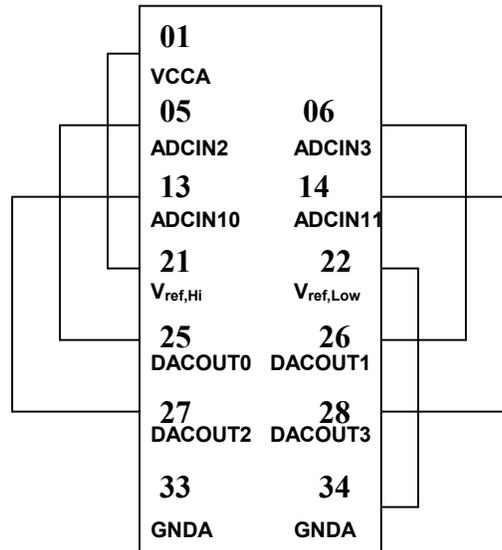
Problem 2

Write a program to generate the following voltages at the four DAC channels.

DAC Channel	Voltage at Pin
DAC0	1.25V
DAC1	2.50V
DAC2	3.75V
DAC3	5.00V

Connect these to channels ADCIN2, ADCIN3, ADCIN10 and ADCIN11 respectively. Set up both the ADCs for single conversion and store the converted values in ADC2VAL, ADC3VAL, ADC10VAL and ADC11VAL respectively. Make the connections as in figure below and tabulate results as shown in the table.

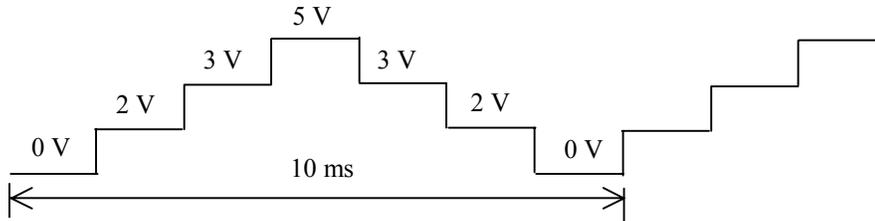
Analog Connector P2



ADC CHANNEL	RESULT
ADCIN2 (ADC2VAL)	
ADCIN3 (ADC3VAL)	
ADCIN10 (ADC10VAL)	
ADCIN11 (ADC11VAL)	

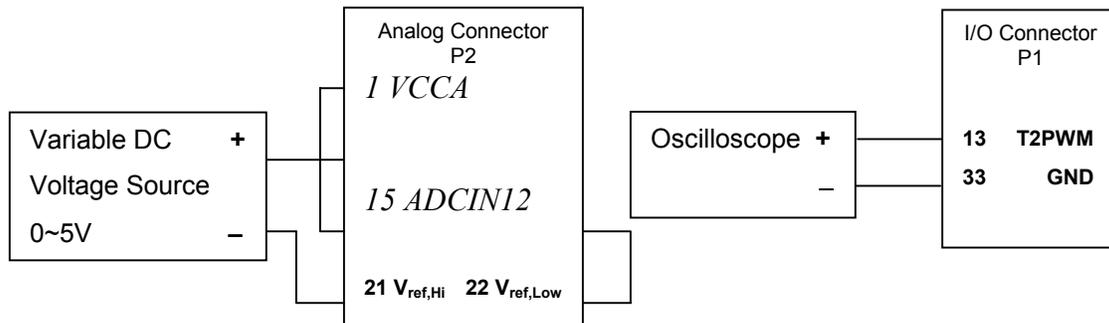
Problem 3

Write a program to output a waveform at the DAC2OUT as shown below. Observe it on the oscilloscope.



Problem 4

Write a program that reads an analog voltage at ADCIN12 and outputs a proportional variable duty cycle PWM wave of frequency 1kHz at the pin T2PWM/T2CMP/IOPB4. Observe the output on oscilloscope for various values of input. Make connections as below:



CHAPTER 6

Stepper Motor Drives and Shaft Encoders

6.1 Stepper Motors

6.1.1 Basic Operation

A stepper motor is an electric machine that rotates in discrete angular increments or steps. It is operated by applying pulses of a specific frequency to the input of the motor. Each pulse applied to the motor causes its shaft to move a certain angle of rotation, called a *stepping angle*. Figure 6.1 shows a simplified construction of *bifilar permanent magnet* stepper motor. The rotor of the motor is made of a permanent magnet material and has six teeth equally spaced around the circumference of the rotor with alternating north (N) and south(S) magnet polarities. The stator has four poles, each of which has a center-tapped winding. The windings on opposing poles are connected together so that only five wires - A, B, C, D, and V+ - leave the motor. A winding is excited by sending a current into the V+ wire and out one of the other wires. The windings are wound in the stator teeth in such way so that the following results are obtained.

- ❑ If winding B is excited, pole 1 is energized as North and pole 2 as South; if winding A is excited, pole 1 becomes South and pole 2 becomes North instead.
- ❑ If winding C is excited, pole 3 is energized as North and pole 4 as South; if winding D is excited, pole 3 becomes South and pole 4 becomes North instead.

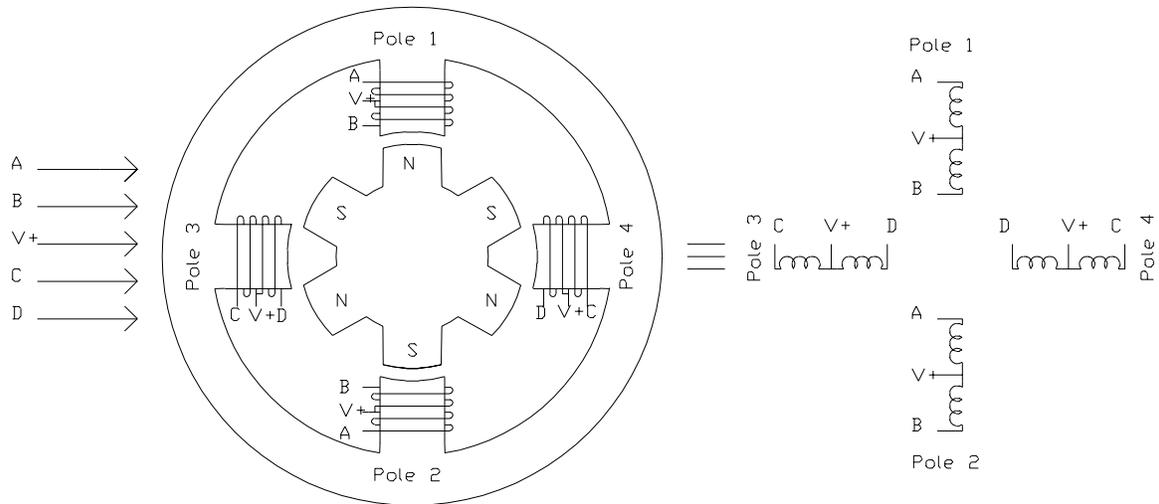


Figure 6.1 Construction of a bifilar PM stepper motor

The operation of the stepper motor relies on the simple principle that, opposite magnetic poles attract while like poles repel. If the windings are excited in a correct sequence, the rotor will rotate to a certain direction Figure 6.2 illustrates how the rotor rotates when the windings are excited with the sequence given in Table 1

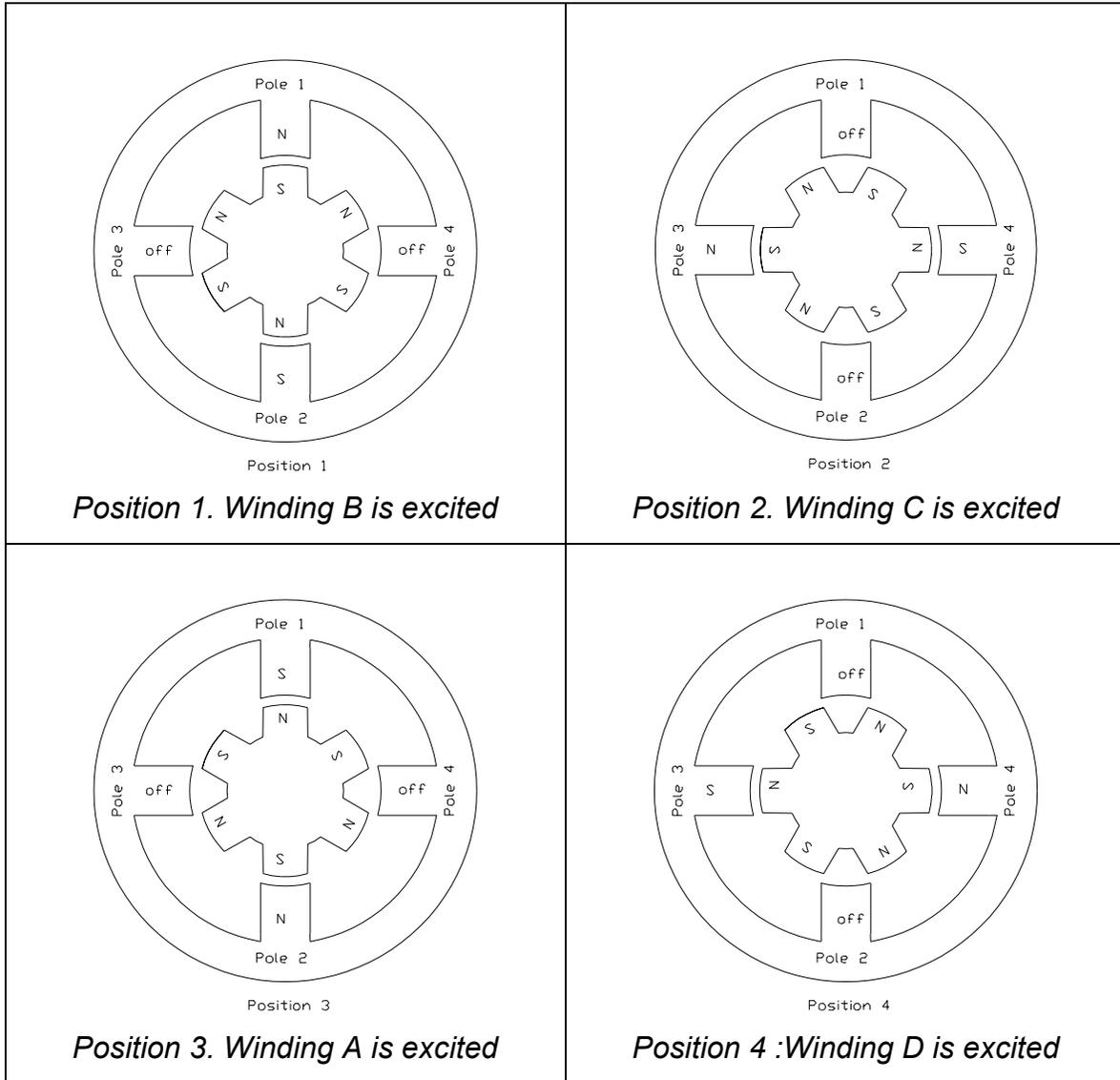


Figure 6.2 The rotor rotates in clockwise direction

	Winding A	Winding B	Winding C	Winding D
Position 1	Off	On	Off	Off
Position 2	Off	Off	On	Off
Position 3	On	Off	Off	Off
Position 4	Off	Off	Off	On

Table 1. *Winding excitation sequence for Figure 6.2*

As can be seen in Figure 6.2, the excitation sequence given in Table 1 causes the rotor to rotate in clockwise direction. If the excitation sequence is reversed, the direction of motion will also be reversed. If the excitation is removed, there is still some attraction between the poles and the teeth due to the permanent magnet in the rotor. As a result there is a residual *holding torque* even when there is no power applied to the motor.

From Figure 6.2 it can be seen, that the motor has a 30 degrees stepping angle, and it requires 12 steps to complete one revolution. The number of steps per revolution in a stepper motor can be increased by adding more teeth on the rotor and by having additional teeth machined into the stator poles. The stepping angle of a stepper motor can be made to be as small as 1.8 degree so that 200 steps are required per revolution.

The excitation scheme in Figure 6.2 is referred to as *single phase excitation* since only one of the four windings is excited at a time. At each step the rotor teeth is aligned exactly with the active stator teeth. It is, however, possible to operate the motor with two windings carrying current at the same time (*two-phase excitation*). In that case the rotor teeth align themselves between the two active stator teeth. Table 2 shows the actuation sequences and the rotor positions for single-phase and two phase excitation. Note that the stepping angles for the two kinds of excitation are the same but that the rotor positions are offset by half the stepping angle. These two actuation scheme are sometimes called *full-stepping* actuation modes

Single Phase Excitation

Rotor position	Winding A	Winding B	Winding C	Winding D
0	Off	On	Off	Off
θ	Off	Off	On	Off
2θ	On	Off	Off	Off
3θ	Off	Off	Off	On

Two-Phase Excitation

Rotor position	Winding A	Winding B	Winding C	Winding D
$\theta/2$	Off	On	On	Off
$3\theta/2$	On	Off	On	Off
$5\theta/2$	On	Off	Off	On
$7\theta/2$	Off	On	Off	On

Table 2. Full step actuation mode: single-phase and two-phase excitation

If the single-excitation and the two-phase actuation sequence are combined, a *half-step* mode results. In this mode the number of steps per revolution is doubled, so that a motor requiring 200 steps per revolution with the full-step mode will require 400 steps per revolution when operated in the half step mode. Table 3 shows the actuation sequence for the half step mode.

Half-step mode

Rotor position	Winding A	Winding B	Winding C	Winding D
0	Off	On	Off	Off
$\theta/2$	Off	On	On	Off
θ	Off	Off	On	Off
$3\theta/2$	On	Off	On	Off
2θ	On	Off	Off	Off
$5\theta/2$	On	Off	Off	On
3θ	Off	Off	Off	On
$7\theta/2$	Off	On	Off	On

Table 3. Half-step actuation mode

The stepper motor described above uses two windings with opposing magnetizing effect in each pole. This is the reason why it is called '*bifilar*' stepper motor. Some stepper motors use only one winding per pole, and are referred to as '*unifilar*' type. Unlike the bifilar type, the unifilar stepper motor requires a negative voltage to reverse the magnetic polarity of the pole. Besides unifilar and bifilar, stepper motors are also classified from the material used to build the rotor. There are some stepper motors that use a simple iron rotor with no permanent magnet. This type of stepper motor is called *variable reluctance* stepper motor. In this type of stepper motor, the rotor is still moved by the attraction of the rotor to the energized poles of the stator. However, a variable reluctance stepper motor has no residual holding torque when the winding is not energized.

6.1.2 Stepper-Motor Drive Circuits

In the lab experiments, we will control the operation of a stepper motor directly from the DSP output port. For this purpose we need a drive circuit for the motor. The type of stepper motor used is very important in designing the drive circuit for the motor. Different type of stepper motor requires different drive circuit. In the laboratory experiment, we will use a bifilar permanent magnet type stepper motor, similar to the stepper motor in Figure 6.1, except that it has a stepping angle of 1.8 degree or 200 step/revolution. Other considerations in designing the drive circuit are the voltage and current required to energize the stator windings of the motor. The stepper motor we use in the lab requires a 4V voltage and 1A current for each phase winding. Figure 6.3 shows the stepper motor drive circuit we will use for the lab experiments.

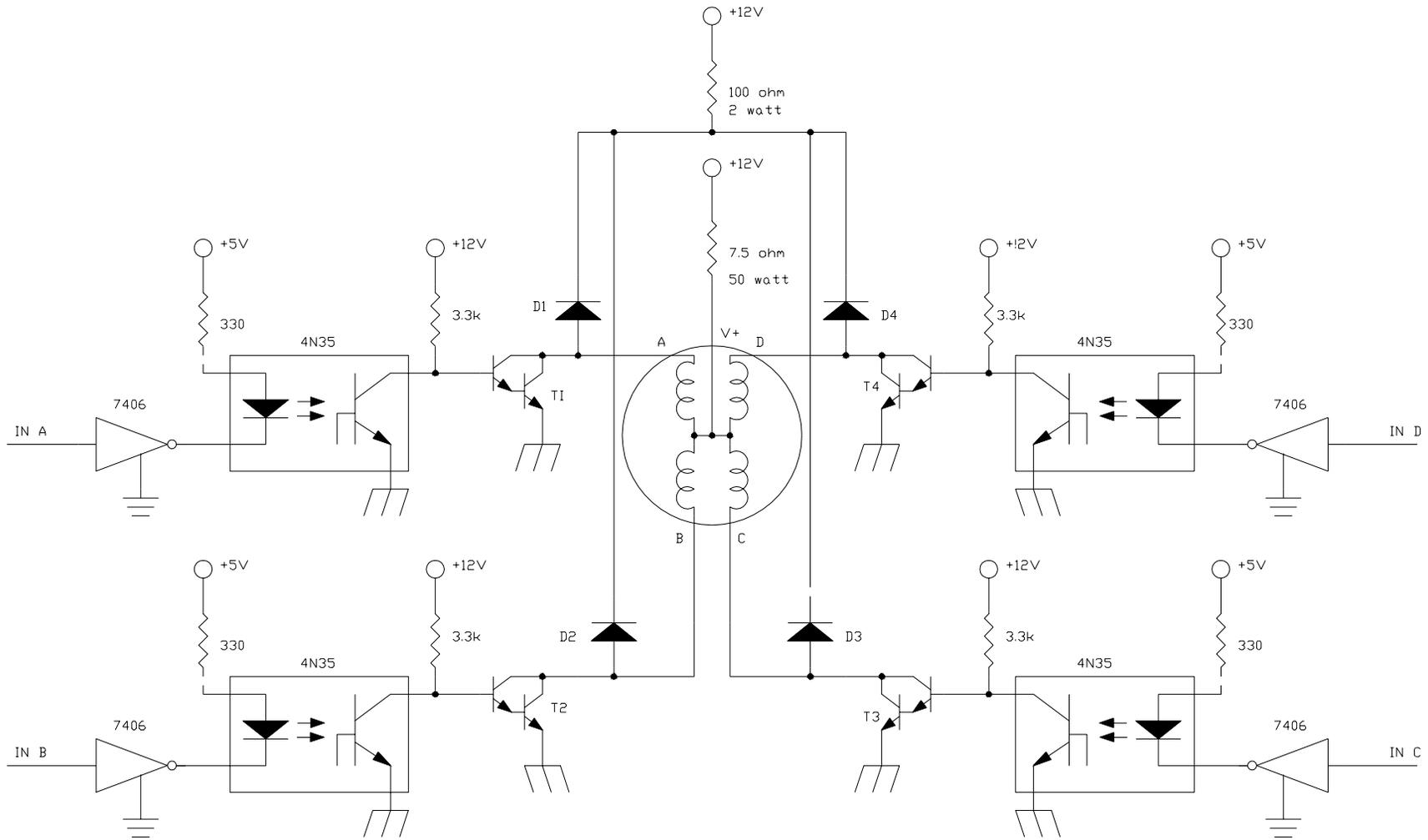


Figure 6.3 Stepper motor drive circuit

In Figure 6.3, wires A, B, C, and D from the stepper motor are connected to the Darlington power transistor T1, T2, T3, and T4 which act as switches. The V+ wire is connected to the +12V power supply via a series resistor. When the power transistor is switched on, a circuit is completed and current flows through one of the windings.

The winding in a stepper motor has substantial inductance. This inductance presents two problems. First, when a winding is switched on, the equation for the current flowing in the winding is:

$$i(t) = \frac{V}{R} \left(1 - \exp \frac{-t \cdot R}{L} \right)$$

where V = supply voltage

R = total circuit resistance

L = total circuit inductance

The current needs approximately $3L/R$ seconds (three time constants) to reach 95 percent of its final value. The stepper motor we use in the lab has a winding resistance of 4 ohms (4V/1A) and winding inductance of approximately 20 mH. Therefore, the time required for the current to reach its final value will be about 15 ms. This certainly limits the stepping rate of the motor. For this reason a 7.5-ohm series resistor is added in the V+ line in Figure 4. This series resistor provides additional resistance to the circuit so that the time constant is decreased. Note that the addition of the series resistance is responsible for the higher supply voltage used in the circuit. Assuming the collector-emitter saturation voltage is 0.5V and the winding voltage drop is 4V, the required 1 A winding current causes 7.5 V voltage drop on the series resistor. Therefore, a +12V (0.5V+4V+7.5V) supply is used.

The second problem created by the winding inductance occurs when a winding is switched off. Recall that when a circuit containing an inductive load is switched off, the collapsing field attempts to maintain the flow of current. If no path is provided for the current. Damaging voltage spikes will be generated across the switching devices. Therefore, freewheeling diodes (D1 to D4) must be included in the driver circuit. The freewheeling diode circuit used in Figure 6.3 includes a series resistor to increase the

decay rate of the winding currents. If this resistor is not used, the maximum stepping rate will be limited by the time it takes the winding current to decay.

The opto-coupler and buffer circuits that drive the base of the power transistors are the same as those we used previously for the relay drive circuit. The opto-couplers are used to provide the isolation between the microcontroller ground and the high power circuit ground, while the inverting buffers (7406) are used to supply enough current to the opto-coupler diodes. *Note that the stepper motor drive circuit in Figure 6.4 has an inverting property. This means that logic 1 at an input terminal will cause the corresponding winding to turn off, while logic 0 causes it to turn on.*

6.2 Generation of the stepper motor drive signals

Program :

Write a program that drive the stepper motor in clockwise direction for 180 degrees in 5 sec.

Solution:

The stepper motor operates with 200 steps per revolution. Thus, to move through 180 degrees, 100 steps are required. This is to be achieved in 5 seconds, thus a step is output to the motor every $5/100 = 50\text{msec}$. The timer1 is set up in continuous up-counting mode to interrupt every 50msec. The timer ISR increments the step counter and the state counter and outputs the proper sequence. After 100 steps have been output, the ISR stops the motor and disables the timer interrupt so that no more commands are sent to the motor. The complete program listing is as below-

$$\text{Timer 1 period value} = \frac{10^7}{32} \times \frac{4}{2} / 20 = 31250$$

```

;*****
; File Name:    Ch6_e1.asm
; Description:  This program moves the stepper motor by 180 deg in 5s
;              T1 - T4 -> IOPB0-3 (Connector P1 Pin 9, 10, 11, 12)
;*****
        .include f240regs.h

```

```

T1PERIOD .set 31250 ;for 50ms

    .bss STEPCTR,1 ;variable for step counter
    .bss STATECTR,1 ;variable for state counter
    .bss STATEVAL,1 ;variable for state value
    .bss GPR0,1

;-----
; Vector address declarations
;-----

    .sect ".vectors"
RSVECT B    START    ; Reset Vector
INT1    B    PHANTOM ; Interrupt Level 1
INT2    B    TMR_ISR ; Interrupt Level 2
INT3    B    PHANTOM ; Interrupt Level 3
INT4    B    PHANTOM ; Interrupt Level 4
INT5    B    PHANTOM ; Interrupt Level 5
INT6    B    PHANTOM ; Interrupt Level 6
RESERVED B    PHANTOM ; Reserved
SW_INT8 B    PHANTOM ; User S/W Interrupt
SW_INT9 B    PHANTOM ; User S/W Interrupt
SW_INT10 B    PHANTOM ; User S/W Interrupt
SW_INT11 B    PHANTOM ; User S/W Interrupt
SW_INT12 B    PHANTOM ; User S/W Interrupt
SW_INT13 B    PHANTOM ; User S/W Interrupt
SW_INT14 B    PHANTOM ; User S/W Interrupt
SW_INT15 B    PHANTOM ; User S/W Interrupt
SW_INT16 B    PHANTOM ; User S/W Interrupt
TRAP    B    PHANTOM ; Trap vector
NMINT   B    PHANTOM ; Non-maskable Interrupt
EMU_TRAP B    PHANTOM ; Emulator Trap
SW_INT20 B    PHANTOM ; User S/W Interrupt
SW_INT21 B    PHANTOM ; User S/W Interrupt
SW_INT22 B    PHANTOM ; User S/W Interrupt
SW_INT23 B    PHANTOM ; User S/W Interrupt

;-----
; Main program
;-----

    .text

```

```

NOP
START: SETC INTM      ;Disable interrupts
      SPLK #0002h,IMR ;Mask all core interrupts
      LACC IFR      ;Read Interrupt flags
      SACL IFR      ;Clear all interrupt flags

      CLRC SXM      ;Clear Sign Extension Mode
      CLRC OVM      ;Reset Overflow Mode
      CLRC CNF      ;Config Block B0 to Data mem

      LDP #00E0h    ;DP for addresses 7000h-707Fh
      SPLK #00BBh,CKCR1;CLKIN(OSC)=10MHz,CPUCLK=20MHz
      SPLK #00C3h,CKCR0;CLKMD=PLL Enable,SYSClk=CPUCLK/2
      SPLK #40C0h,SYSCR;CLKOUT=CPUCLK

      SPLK #006Fh, WDCR;Disable WD if VCCP=5V (JP5 in pos. 2-3)
      KICK_DOG      ;Reset Watchdog

      SPLK #0h,GPR0 ;Set wait state generator for:
      OUT  GPR0,WSGR ;Program Space, 0 wait states
           ;Data Space, 0 wait states
           ;I/O Space, 0 wait states

      LDP #0
      SPLK #0, STATECTR;Initialize state counter
      SPLK #0, STEPCTR ;Initialize step counter

      LDP #00E1h
      SPLK #0000h, OCRA ;Set pins to be I/O
      SPLK #0F0Fh,PBDATDIR ;Select IOPB0-3 as outputs
;-----
;      RESET SECTION - BEGINS
;-----

      LDP #00E8h
      SPLK #0, GPTCON
      SPLK #0, T1CON
      SPLK #0, T2CON
      SPLK #0, T3CON

      SPLK #0, COMCON

```

```

SPLK #0, ACTR
SPLK #0, SACTR
SPLK #0, DBTCON
SPLK #0, CAPCON

SPLK #0FFFFh, EVIFRA
SPLK #0FFFFh, EVIFRB
SPLK #0FFFFh, EVIFRC

SPLK #0, EVIMRA
SPLK #0, EVIMRB
SPLK #0, EVIMRC
;-----
; RESET SECTION - ENDS
;-----

LDP #00E8h
SPLK #0055h, GPTCON
SPLK #T1PERIOD, T1PR ;Set timer 1 period 50ms
SPLK #0, T1CNT
SPLK #0, T2CNT
SPLK #0, T3CNT

SPLK #150Ah, T1CON ;Set PRESCALER=32

SPLK #0080h, EVIMRA
LACC EVIFRA
SACL EVIFRA

LDP #0
LACC IFR
SACL IFR

LDP #232
SBIT1 T1CON, B6_MSK ;Enable timer 1
CLRC INTM ;Enable interrupts
END B END

STATE .word 0F0Dh ;Position 2
.word 0F0Bh ;Position 1
.word 0F0Eh ;Position 4
.word 0F07h ;Position 3

```

```

;=====
; I S R - TMR_ISR
; Description: Determines the current state and outputs the
; appropriate sequence to the transistors every
; 50ms. It also keeps track of the number of steps
; and stops the motor when it has covered 180 degrees.
; Modifies: STATEVAL, STATECTR, STEPCTR, PBDATDIR, EVIMRA
;=====
TMR_ISR LDP #0
        LACC STEPCTR
        ADD #1
        SACL STEPCTR ;Increment step counter
        SUB #101
        BCND NXT1,NEQ ;If number of steps < 100, goto NXT1
        LDP #00E1h ;Load DP for I/O registers
        SPLK #0F0Fh,PBDATDIR ;Turn off all transistors,
                ;stop the motor if step=100
        LDP #00E8h ;Load DP for event manager
        SPLK #0000h, EVIMRA ;Disable timer 1 period interrupt
        B FIN
NXT1 LACC #STATE
      ADD STATECTR ;ACC=address of current state
      TBLR STATEVAL ;Move the state to a variable
      LACC STATEVAL ;Load the state
      LDP #00E1h
      SACL PBDATDIR ;Output the current state
      LDP #0
      LACC STATECTR
      SUB #3
      BCND NXT, NEQ ;If state counter < 3, jump to NXT
      SPLK #0,STATECTR ;Reset state counter
      B FIN
NXT LACC STATECTR
    ADD #1
    SACL STATECTR ;Increment state counter
FIN LDP #00E8h
    LACC EVIFRA
    SACL EVIFRA ;Enable interrupts
    CLRC INTM
    RET

```

```
=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
; Last Update: 16 June 95
=====
PHANTOM KICK_DOG ;Resets WD counter
        B PHANTOM
```

6.3 Shaft Encoders

Shaft encoders are electromechanical devices that convert the angular position of the shaft to some form of digital signal. Depending on the form of signals they produce, there are two types of encoders, *incremental encoders* and *absolute encoders*. An absolute encoder produces an output consisting of a *coded n-bit digital data* that directly represents the absolute position of the shaft. An incremental encoder, on the other hand, generates a train of pulses which must be counted in some way to obtain the information about the relative position of the shaft. Although both encoders function as position sensors, the information about the speed of the shaft can be easily deduced through some additional processing steps. Therefore these encoders are also used for applications that require measurement of the shaft speed. Below we will discuss the basic principles of both type of encoders and how to interface them to a DSP, specifically the TMS320F240 DSP.

6.3.1 Principles of operations

There are two types of incremental encoders: *the contacting type* and *the optical type*. Figure 6.1 and 6.2 shows a typical structure for both types. Conversion of the shaft angular position to pulse trains is achieved by using a disk that rotates with the shaft of the motor.

In the *contacting type* (Figure 6.4), metallic strips (electrical conductors) are deposited along three separate tracks (zero index, A, and B) on the surface of the disk which is made of electrically non-conductive material. The metallic strips are connected to ground potential. For each track there is a contacting spring loaded pin which make a pressure contact with the disk and rides along the track as the disk rotates. Every time the pin makes contact with the metallic strip the output voltage becomes 0V (since the strip is maintained at ground potential), and during the time the pin is between the strips, the output rises to the full supply voltage. As the disk rotates a pulse train will be generated at each output of the encoders.

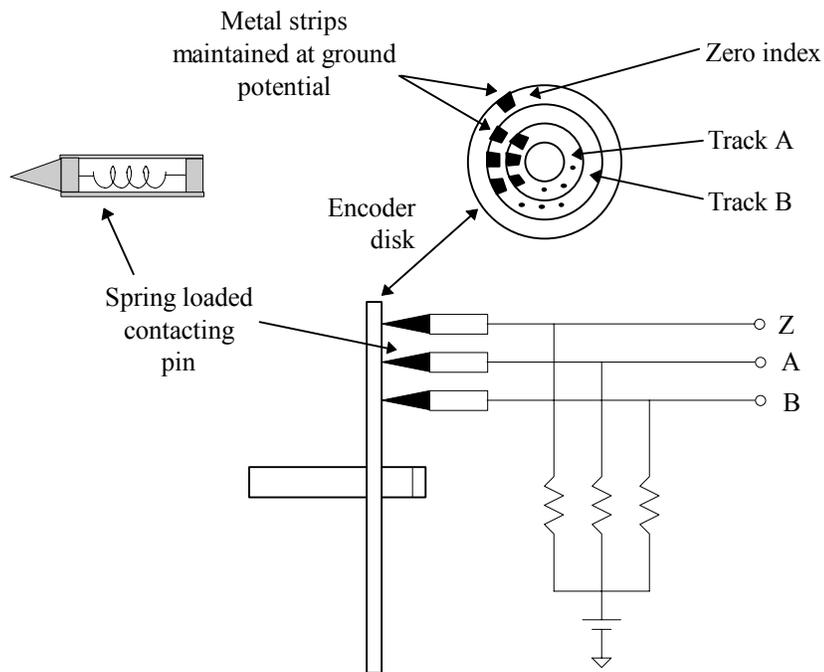


Figure 6.4 *Contacting type incremental encoders*

In the *optical* type (Figure 6.5), the metallic strips are replaced by the apertures or holes on the disk and the sensing unit consists of the LED-phototransistor combination. The phototransistors are put in common emitter configurations. Recall that a phototransistor will conduct current if it is excited by a light beam with the conducting state being proportional to the amount of light received, As the disk rotates, the conducting state of the phototransistor will be modulated by the relative position of the apertures and LED-phototransistor pairs. When an aperture is in complete alignment with the pair, the phototransistor receives the maximum amount of light beam from the LED so that the conducting state of the phototransistor reaches its maximum value (the phototransistor presents minimum resistance to ground). This causes the collector voltage of the phototransistor to be in its minimum value. When the light beam is completely blocked by the disk, the conducting state will reach the minimum value so that the collector voltage is at its maximum. As the disk rotates the collector voltage of the phototransistor will take the form as shown in Figure 6.5. This voltage is supplied to a

comparator so that a pulse train is generated (a comparator produces a HIGH voltage whenever the voltage difference between its + and - terminal is positive and a LOW voltage when the difference is negative).

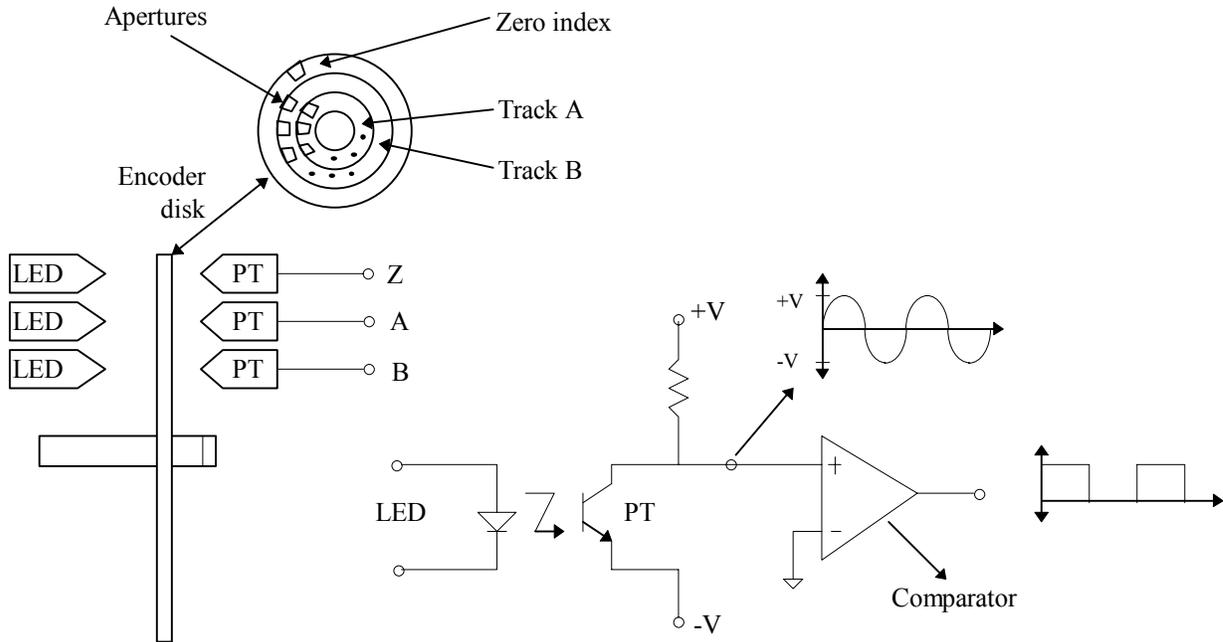


Figure 6.5 Optical type incremental encoder

Both the contacting type and optical type incremental encoders produce the same A and B output waveforms as shown in Figure 6.6. The strips or apertures on track A and track B are designed such that waveforms A and B are in quadrature (90 degree phase difference). Such A and B signal waveforms provides us with a way to determine the direction of shaft rotation. In one direction (say CW) A leads B, and in the opposite direction (CCW) A lags B.

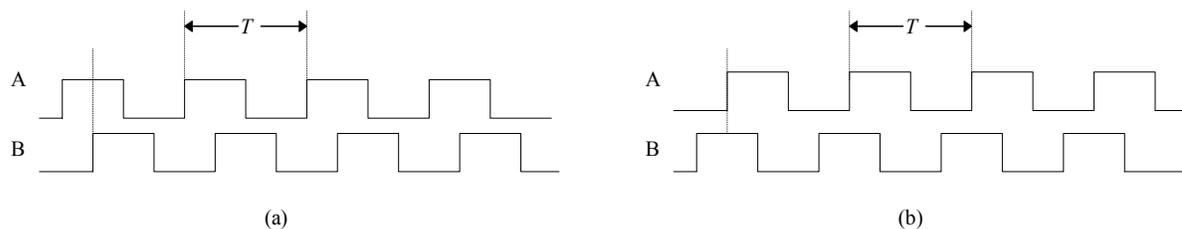


Figure 6.6 A and B output signal waveforms (a) CW direction (b) CCW direction

The number of or apertures present on track A or B determine the number of pulses produced in one revolution of the disk. If there are N strips or apertures on a given

track, one revolution of the disk produces N pulses per revolution. The resolution that represent the maximum rotation of the shaft before the next pulse is produced depends on N , and may be expressed as

$$R = \frac{360^\circ}{N}$$

Thus for a 1 degree resolution, 360 strips or apertures per track are required. To determine the angular displacement of the shaft, one simply needs to count the pulses produced by the encoders. We will see later how to accomplish this using a DSP.

The speed of rotation, n (rpm) can also be deduced from signal A or B. If T represents the period of the signal in second and f the frequency in Hertz, then the speed of rotation can be determined from the following relation:

$$n = \frac{60 \times f}{N} = \frac{60}{T \cdot N}$$

Therefore, we can use the encoder to determine the speed of rotation by measuring the period of signal A or B and calculating the speed with the above formula.

Note that incremental encoders available in the market do not necessarily come with both signal A and B present. Encoders that come with both signal A and B as above are normally called *bi-directional* incremental encoders. Encoders that come with only signal A are termed *unidirectional* encoders due to the fact that the direction of rotation can not be determined using these encoders.

The zero index track contains one strip or aperture so that only one pulse is generated every one revolution of the shaft as shown in Figure 6.7. This signal serves as a convenient marker for synchronization purposes. Since the position of the shaft at which the index pulse occurs is fixed, this position can be used as a starting point of known position from which counting or position counting begins.

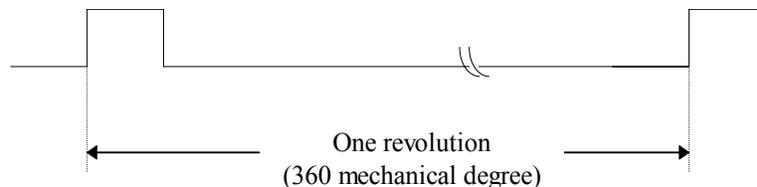


Figure 6.7 Zero index signal

6.3.2 Interface circuit

The stepper motor used in the lab (ASTROSYS C245) is equipped with an incremental encoder which is mounted at one end of its shaft. The encoder is a SHARP 3GP1R26W bi-directional optical incremental encoder which has 49 pulses per revolution and a zero index signal and requires bipolar supply voltages of $\pm 15V$ for its operation. Figure 6.8 shows the construction of the stepper motor with the encoder. In the following paragraph, this encoder will be used to illustrate how to interface an incremental encoder to the microcontroller and how to write a program for the microcontroller to obtain the position and speed of the shaft from the output of the encoders.

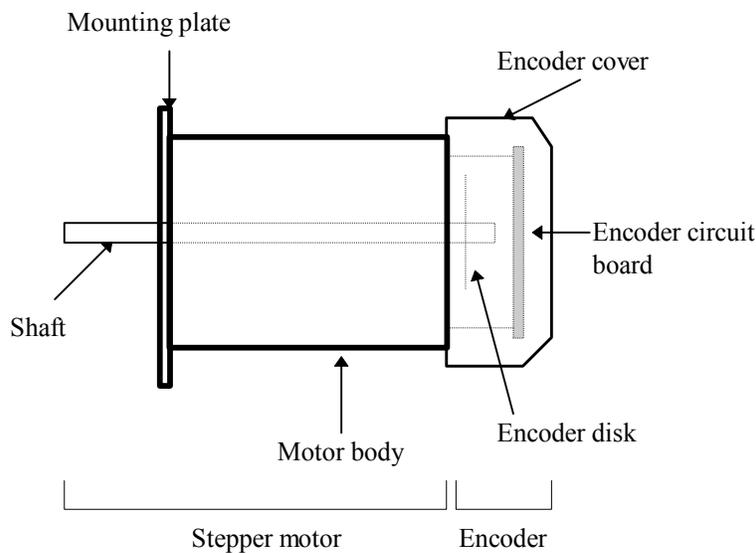


Figure 6.8 Construction of the ASTROSYS stepper motor

The output voltages of the SHARP encoder are shown in Figure 6.6. It can be seen from this figure that the output voltages are still in the form of alternating wave. An interface circuit has been designed to interface the encoder to the DSP. The interface circuit converts the output waveforms of the encoder to digital pulse trains with voltage level compatible with the logic level of the DSP (Figure 6.9).

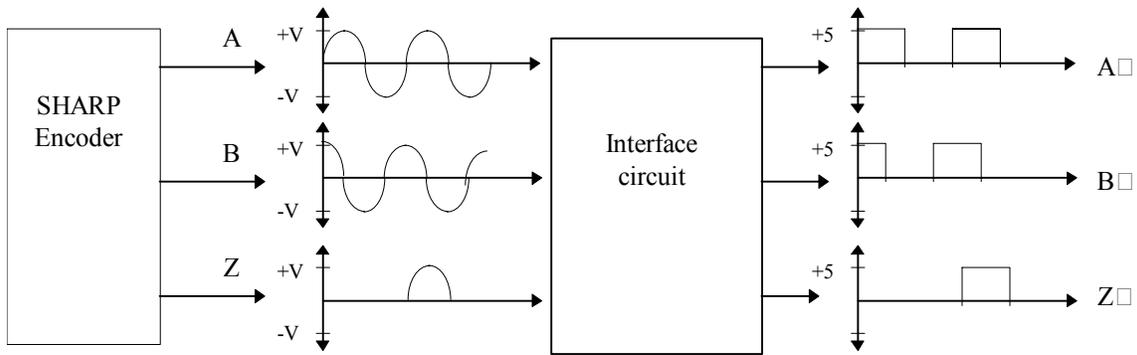


Figure 6.9. Encoder output waveforms and interface circuit

Figure 6.10 shows the schematic diagram of the encoder interface circuit. The circuit consists of zero crossing detectors and inverting Schmitt Trigger gates. The zero crossing detector comprising of a comparator and a transistor converts the input voltage into a pulse train of 0-5V magnitude. When the input voltage is positive the output of the zero crossing detector is +5V; when it is negative the output is 0V. The inverting Schmitt trigger is used to reject the noise that might interfere with the voltage waveforms,

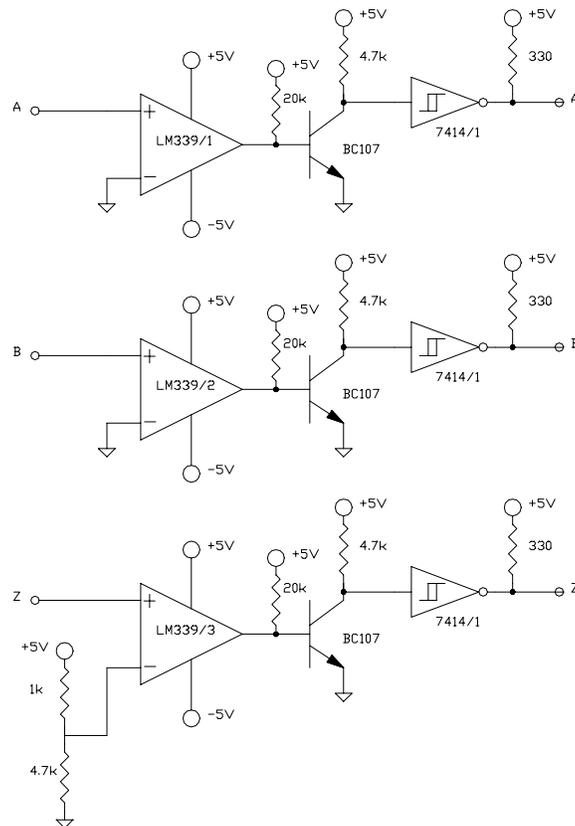


Figure 6.10 Schematic diagram of the encoder interface circuit

Note that most encoders currently available in the market provide output voltages which are already compatible with the logic voltage used by most microprocessors or DSPs, i.e. TTL or CMOS logic. The interface for these encoders are simpler; some of them can even be directly interfaced to the DSP.

6.3.3 Programming

Pulse counting

To obtain the position information we need to count the number of pulses generated by the incremental encoder. One way to implement pulse counting using the TMS320F240 DSP is shown in Figure 6.11. The output signal from the encoder is connected to one of the input capture channel in the DSP. The input capture is configured to generate interrupts when either the falling or rising edges of the signal occur. When the interrupt from the input capture occurs, the interrupt service routine will increment a variable in memory (`pulse_count`) that represent the number of pulses received.

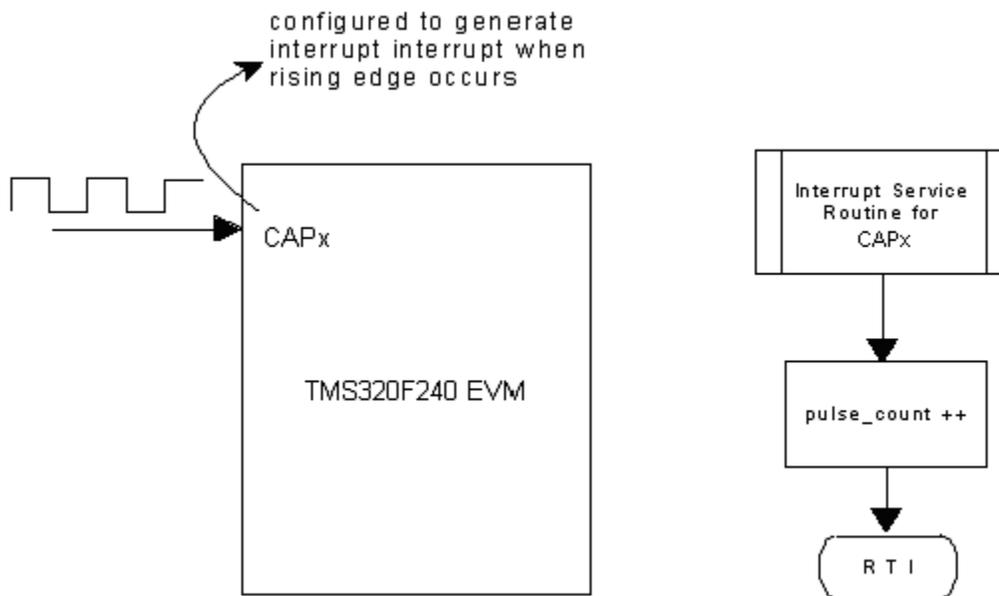


Figure 6.11 *Pulse counting using input capture interrupt and software counter*

Direction determination

As mentioned previously, the direction of rotation can be determined by looking at the phase of signal A with respect to signal B. Whether signal A leads or lags signal B can be resolved by looking at the status of signal B at the rising or falling edges of signal A as shown in Figure 6.12. If A leads B, B is LOW at the rising edge of signal A; if A lags B, B is HIGH at the rising edge of signal A.

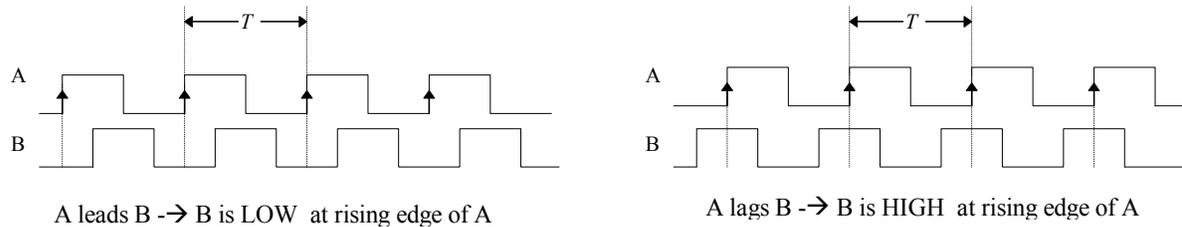


Figure 6.12. *Determination of the direction of rotation*

Figure 6.13 shows how to include the direction determination into the previous pulse counting scheme. In this case, signal A is connected to the input capture pin while signal B is connected to one of the general purpose I/O port pin of the microcontroller. The input capture is configured to detect the rising edge of A and to generate interrupts as before. However, this time when interrupts occur the interrupt service routine first read the status of signal B; if B equals to 0 then the `pulse_count` is incremented, otherwise it is decremented to reflect rotation in the other direction.

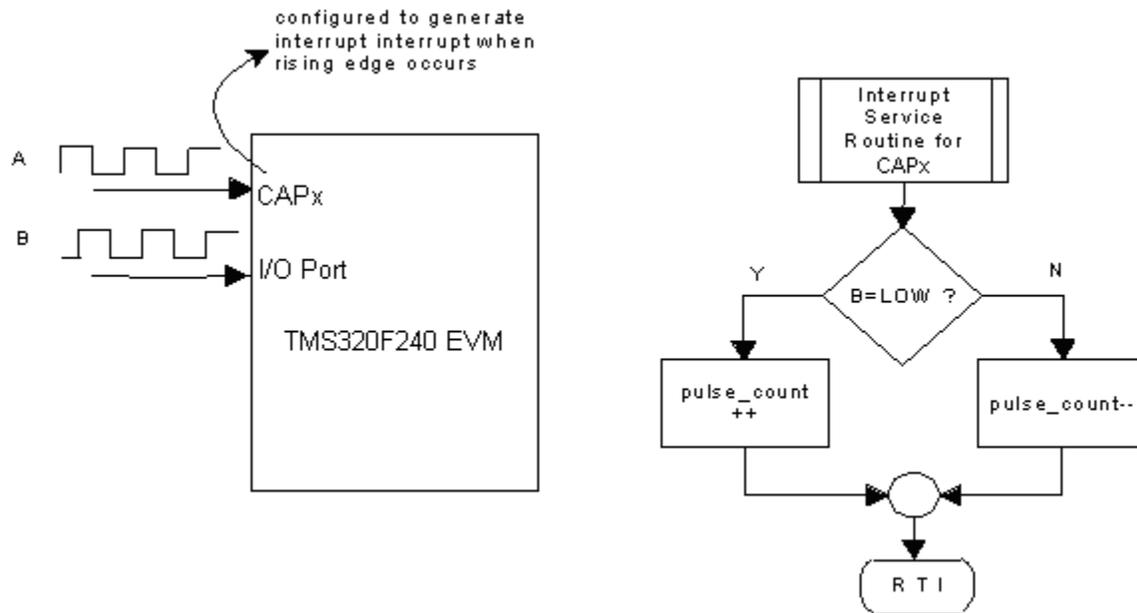


Figure 6.13 *Pulse counting with direction determination*

Zero position initialization

The pulse counts of signal A or B only provide information about the relative movement of the motor shaft. In order to obtain the absolute position of the shaft, a zero position reference is needed. One way to establish the zero position reference is by using the zero index signal. Prior to using the motor, the shaft should be brought to the position at which the zero index signal occurs. This position will then become the starting point of known position from which counting or position counting begins. We refer to this process as *zero position initialization*.

Figure 6.14 illustrates how to implement a zero position initialization for the stepper motor. The zero index signal is connected to one of the input capture pin which is configured to generate an interrupt when a rising edge signal is detected. The program first starts the motor to rotate. When the rising edge of the signal is detected, an interrupt is generated by the input capture, at which time the zero position is reached and the interrupt service routine stops the motor.

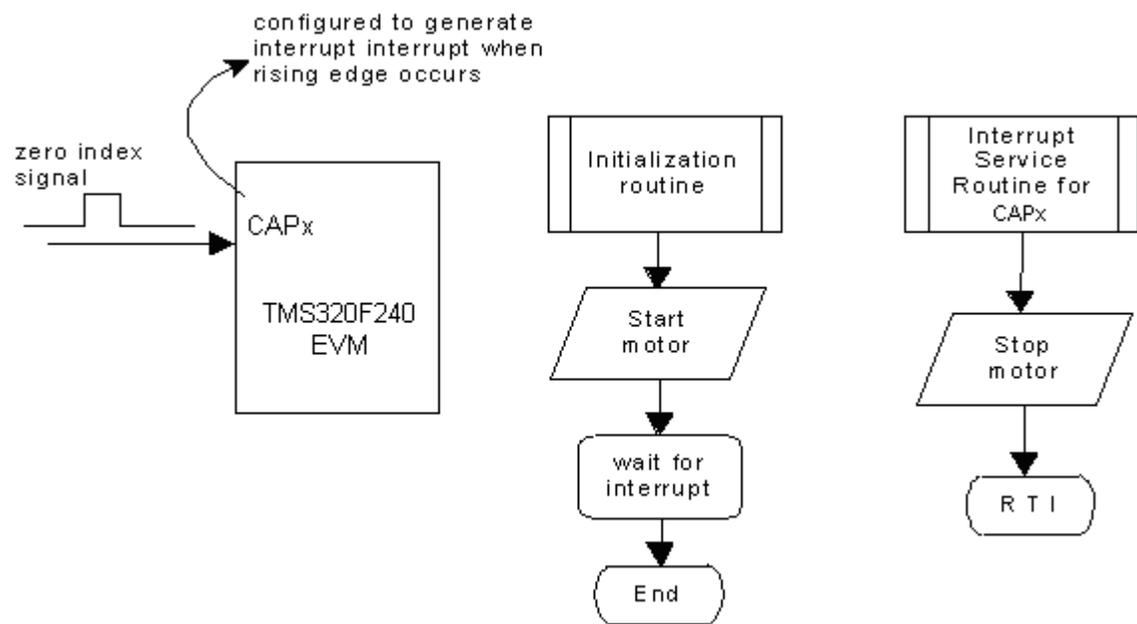


Figure 6.14 Zero initialization process

6.4 An Example Program

Below is an example program that drives the stepper motor for certain degree of rotation and determines the position information from the encoder output signal. The stepper motor is driven from the general purpose port F pins PF0 to PF3 and output compare OC2 is used for the step timing. Signal A of the encoder is connected to pin IC1, while signal B is connected to pin PGP1/IC2 which is programmed for general purpose port GP. The zero index signal is connected to the IC3 pin of the microcontroller.

Program :

```
#include <stdio.h>
#include <sim.h>          /* for port F registers declarations */
#include <gpt.h>          /* for GPT registers declarations */
#include <inerrup.h>

#define TRUE 1
#define FALSE 0

unsigned char StepTable[4]={~2,~4,~1,~8};
unsigned int StepPeriod;
unsigned char zero_pos=FALSE;

long pulse_count;
int StatePtr;
long StepCount;

@port void OC2_Isr()
{
    TFLG1&=~0x10;          /* clear OC2F flag */
    PORTF=StepTable[StatePtr]; /* output the sequence to the
                               ports */
    if(StatePtr==3)        /* if the lastreached */
        StatePtr=0;        /* the end of the table */
    else
        StatePtr++;
    StepCount++;           /* increment step count */
    TOC2+=StepPeriod;      /* update TOC2 to schedule for next
                             step */
}

@port void IC3_Isr()
{
    TFLG1&=~0x04;          /* clear IC3 flag */
    TMSK1=0;               /* disable OC2 and IC3 interrupt */
    zero_pos=TRUE;
}
```

```

@port void IC2_Isr()
{
    TMSK1&=~0x02;
    TFLG1&=~0x02;          /* clear IC3 flag */
    pulse_count++;
    TMSK1|=0x02;
}

void InitPosition()
{
    TCTL2=0x10;          /* configure IC3 to capture on rising edge*/
    StepPeriod=100;
    StepCount=0;
    StatePtr=0;
    zero_pos=FALSE;
    PORTF=StepTable[StatePtr]; /* output the sequence to the */
                                /* ports */
    TOC2+=StepPeriod;    /* delay */
    TFLG1&=~0x14;       /*clear OC2 and IC3 flag */
    TMSK1|=0x14;        /* enable and IC3 and OC2 interrupt */
    EnableAllInt() ;
    while(!zero_pos);
    DisableAllInt() ;
}

void Rotate(long TotalStep)
{
    TCTL2=0x04;          /* configure IC2 to capture on rising edge*/
    StepPeriod=100;
    StepCount=0;
    pulse_count=-1;
    TOC2+=TCNT;         /* update TOC2 to schedule for next step */
    TFLG1&=~0x12;       /*clear OC2 and IC3 flag */
    TMSK1|=0x12;        /* enable IC2 and OC2 interrupt */
    EnableAllInt() ;
    while(StepCount<=TotalStep);
    DisableAllInt() ;
    TCTL2=0;           /* disable capture */
    TMSK1=0;           /* disable OC2 and IC2 interrupt */
    PORTF=0x0f;        /* turn off all windings */
}

void main()
{
    char c;
    int i;

    /*** configure port F pins **/
    PFPAR=0;          /* assign pins for port F use */
    DDRF|=0x0f;       /* configure PF0-PF3 as output */
    PORTF=0x0f;       /* turn all windings off initially*/

    /* set TCNT clock freq = system clock /256 */
    TMSK2=0x06;
}

```

```

/* initialize interrupt system */
GPTMCR=5;          /* assign GPT int. arb = 5 */
ICR=0x0640;       /* IRL=6, IVBA=4 */
SetVector(OC2_Isr,0x45);
SetVector(IC3_Isr,0x43);
SetVector(IC2_Isr,0x42);

/*configure OC2 operation , no OC2 output is necessary */
TCTL1=0x0;        /* disable OC2 pin output */

InitPosition();

for(i=1;i<=20000;i++);
Rotate(400);
printf("Pulse detected :%ld \n",pulse_count);
}

```

LABORATORY EXPERIMENT 5

Stepper Motor Drives and Encoders

Objectives

In this lab, the students will learn the basic operation of a stepper motor and shaft encoders and their interfaces to a DSP. Upon completion of this lab the students should be able to understand and write assembly language programs to control the stepper motor and obtain the shaft position and speed information using the encoders.

Equipment required

Hardware :

- PC Specifications -
 - ❑ '386 or higher IBM PC/AT
 - ❑ 1.44Mb 3.5-inch floppy drive
 - ❑ 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
 - ❑ Minimum 4Mb memory
 - ❑ Color VGA Monitor

- TMS320C24x Evaluation Board
- XDS510PP Emulator
- +5V power supply.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable
- Opto-coupler interface circuit board
- Power transistor switches circuit board
- Encoder interface circuit board

Software :

- MS Windows 3.1 or Windows 95
- ASCII Editor
- TMS320C2xx Assembler
- TMS320C2xx C Source Debugger

Discussion

In this lab, we will use an ASTROSYS C245 stepper motor. This motor is a 4-phase bifilar stepper motor with stepping angle of 1.8 degree and 1 A per phase current. It is equipped with a SHARP 3GP1R26W bi-directional optical incremental encoder which has 49 pulse/ revolution resolution and a zero index signal. The construction of the stepper motor and its encoder is shown in Figure 6.8.

An interface circuit is needed to drive the stepper motor from the DSP port. Figure 1 shows the schematic diagram of the stepper motor interface circuit. In the lab the interface circuit is broken down into two separate board as shown in Figure 2: *the opto-coupler interface circuit board* and the *power transistor switches circuit board*. The opto-coupler interface circuit board is the same as the one we have used for the relay interface in the previous lab, while the power transistor switches circuit board contain the rest of the stepper motor interface circuit.

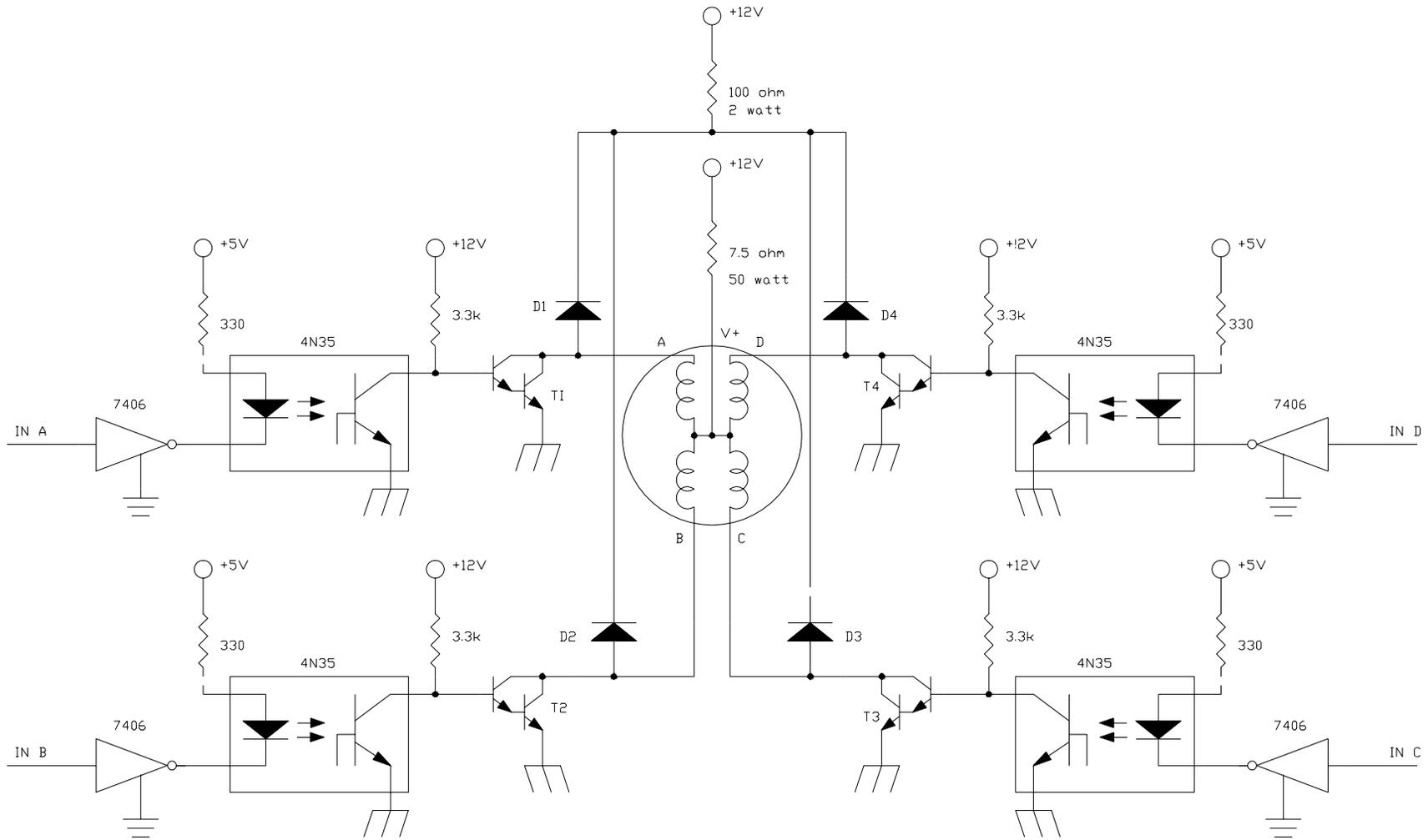


Figure 1. Stepper motor interface circuit

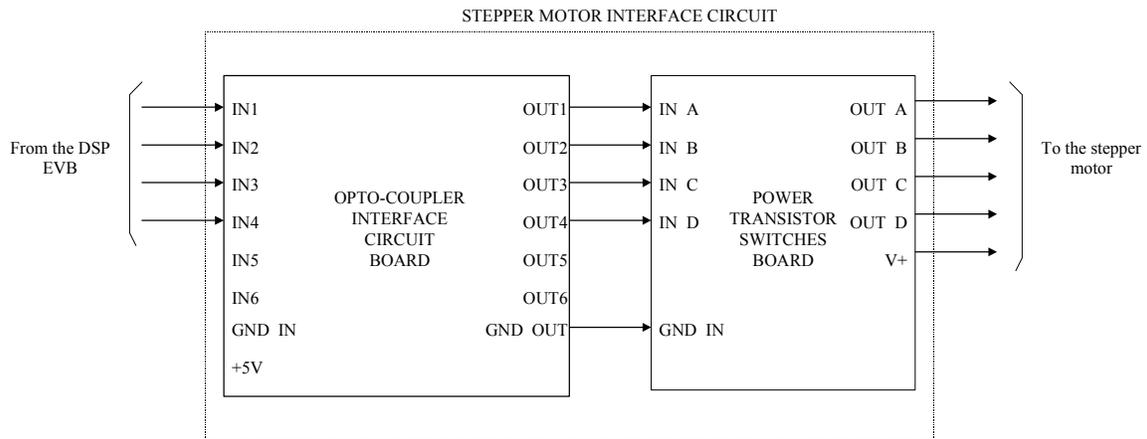


Figure 2. The opto-coupler interface circuit board and the power transistor switches board

The interface circuit for the encoder is shown in Figure 3. This circuit is placed into a board called *encoder interface circuit board*. The block diagram showing the input-output connection of the encoder circuit board is shown in Figure 4.

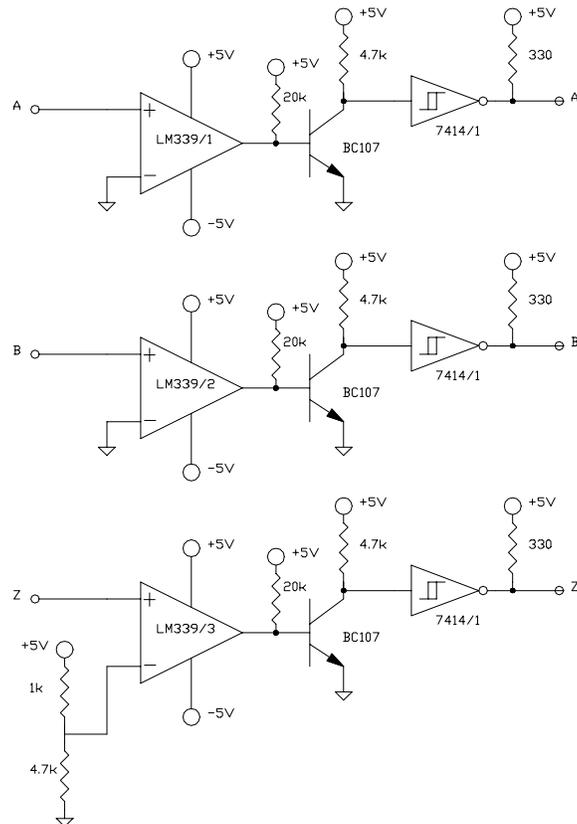


Figure 3 Schematic diagram of the encoder interface circuit

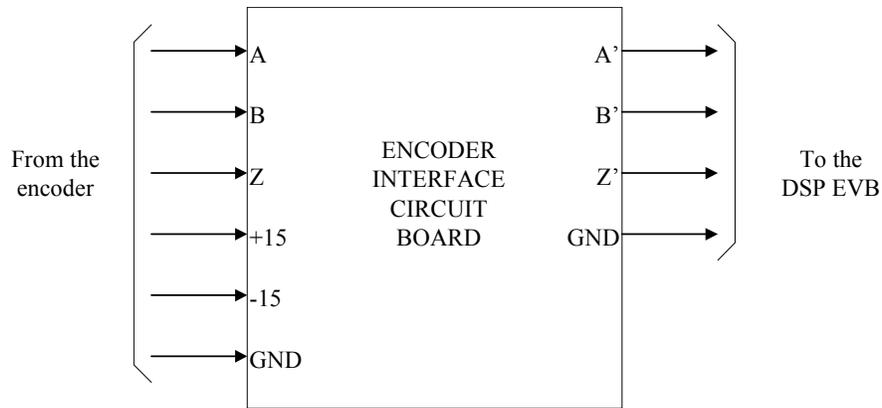


Figure 4 *Input-output connection of the encoder interface circuit board*

Procedure

Setup

1. Make sure that the EVB system has been properly setup as in the previous lab.
2. Make sure that the LabPC+ board has been installed in the PC you are using and connect the LabPC+ board to the NITB terminal block using the ribbon wire provided.
3. Connect the power transistor switches circuit board to the stepper motor using the connectors and wires provided.
4. Establish the interconnection between the power transistor switches circuit board and the opto-coupler interface circuit board.
5. Connect the opto-coupler board inputs IN1 to IN4 to pins IOPB0 to IOPVB4 of the EVB respectively and the GND and +5V connections.
6. Connect the encoder interface circuit board to the motor encoder using the connector provided.
7. Connect the output of the encoder interface circuit to the EVB as follows: A' to CAP1, B' to IOPB4, and Z' to CAP2. Also connect the GND to the GND of the EVB.
8. Turn on the PC and wait until the MS-Windows starts.
9. Turn on the EVB power supply and the power for all the boards.
10. Run the EVB Testing program.

Laboratory Assignments

Stepper Motor Drive

1. Write a program that drives the stepper motor to rotate in CW direction using the single-phase excitation scheme and perform the following experiments :
 - a) Drive the stepper motor to rotate for 18 degrees in 2 sec
 - b) Drive the stepper motor to rotate for 90 degrees in 1 sec
 - c) Drive the stepper motor to rotate for 1 revolution in 5 sec
 - d) Drive the stepper motor to rotate for 10 revolution in 10 sec
 - e) Drive the motor to run continuously with stepping rate of 100 ms/step
 - f) Drive the motor to run continuously with stepping rate of 2 ms/step

Verify the result of your program and observe the output wave-forms from the EVB using the oscilloscope.

2. Repeat 1 and 2 using the dual-phase excitation scheme.
3. Repeat 1 and 2 using the half-step-mode excitation scheme.
4. Repeat step 1 to 3 but this time for driving the motor in CCW direction

Incremental Encoder

Position decoding

1. Write a program that performs the following :
 - a) Initializes the position of the stepper motor to the zero reference position
 - b) Drives the stepper for 180 degree CW rotation in 1 sec, counts the number of pulses generated by signal A of the encoder, and determine the direction of rotation from the encoder signals.
 - c) Display the number of pulses generated by the encoder and the direction in the terminal emulator window.
2. Verify your program by executing it in the EVB and observe the result. Compare the result of your program with the actual position of the stepper motor.

3. Repeat step 1 and 2 but for part B drive the stepper motor for 2 CCW revolutions in 2 sec.

Speed decoding

1. Write a program that performs the following :
 - a) Initializes the position of the stepper motor to the zero reference position.
 - b) Drives the stepper continuously in CW direction with stepping rate of 2 ms/step and determine the steady state speed and direction of the stepper motor from the encoders signals
 - c) Display the steady speed and the direction in the terminal emulator window.
2. Run your program in the EVB.
3. Observe the output of the encoder and the encoder interface circuit using the oscilloscope. Compare the steady speed obtained by your with the intended stepping rate.
4. Repeat step 1 and 2 but for part B drive the stepper motor in CCW direction.

CHAPTER 7

DC MOTOR DRIVE

In this chapter, a DSP-based DC motor speed control system is described. The DC motor used is of permanent magnet type. The model of the permanent magnet DC motor is first developed, followed by the discussion on the mechanism that will be used to drive the motor.

7.1 Permanent Magnet DC Motor

A permanent magnet DC motor consists of a permanent magnet stator and armature windings in the rotor. The armature winding is supplied with a DC voltage that causes a DC current to flow in the windings. Interaction between the magnetic field produced by the armature current and that of the permanent magnet stator causes the rotor to rotate. The equivalent circuit of a PM DC motor is shown in Figure 7.1.

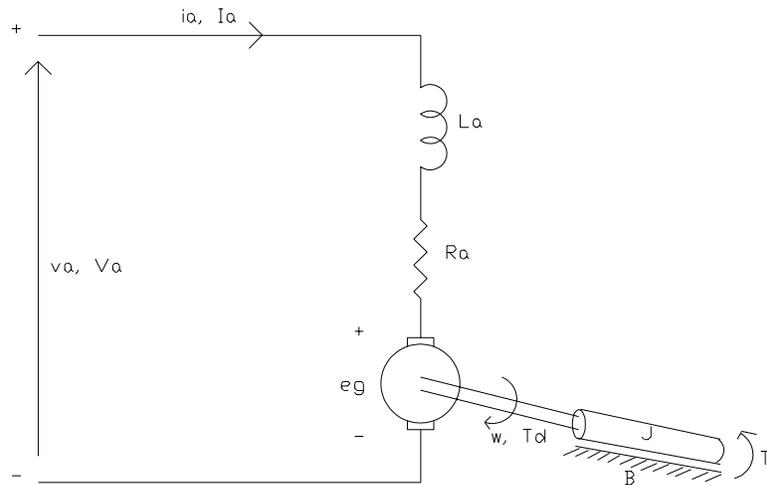


Figure 7.1. *Equivalent circuit of a permanent magnet DC motor*

The equivalent circuit in Figure 7.1 includes the resistance R_a and inductance L_a of the motor's armature windings. When a DC supply voltage V_a is applied to the armature,

current i_a flows in the armature and the motor torque to balance the load torque. Due to the movement of the armature in the magnetic field a back-emf voltage e_g is generated. The back-emf voltage opposes the voltage applied to the motor terminal and is proportional to the speed of the motor.

The equations describing the characteristic of a PM DC motor can be determined from Figure 7.1. The instantaneous armature current i_a can be found from:

$$v_a = R_a i_a + L_a \frac{di_a}{dt} + e_g \quad (7.1)$$

The back-emf is expressed as:

$$e_g = K_E \omega \quad (7.2)$$

The torque developed by the motor is proportional to the armature current and is expressed as

$$T_m = K_t i_a \quad (7.3)$$

The developed torque must be equal to the load torque:

$$T_m = J \frac{d\omega}{dt} + B\omega + T_L \quad (7.4)$$

where:

ω = motor speed, rad/s

B = viscous friction constant. N.m/rad/s

K_E = back emf constant

K_t = torque constant

L_a = armature circuit inductance, H

R_a = armature circuit resistance, ohm

T_L = load torque, N-m

T_m = motor developed torque, N-m

Under the steady state conditions, the time derivatives in these equations are zero and the steady state average quantities are:

$$E_g = K_E \omega \quad (7.5)$$

$$\begin{aligned} V_a &= R_a I_a + E_g \\ &= R_a I_a + K_E \omega \end{aligned} \quad (7.6)$$

$$\begin{aligned} T_m &= K_t I_a \\ &= B\omega + T_L \end{aligned} \quad (7.7)$$

From equation 7.6, the steady state speed of a permanent magnet motor can be found from:

$$\omega = \frac{V_a - I_a R_a}{K_E} \quad (7.8)$$

It can be noticed from Equation 7.8 that **the motor speed can be varied by controlling the armature voltage V_a or the armature current I_a .**

The direction of rotation of the motor can be reversed by reversing the polarity of the voltage applied to the terminal. If the voltage applied to the motor terminal is reversed, the direction of current flowing in the armature winding is also reversed. This will cause the motor to produce torque in the reverse direction. Note here that the polarity of the back-emf voltage is also reversed.

7.2. The Motor Drive

As described in the previous section, the speed of a permanent magnet DC motor can be altered by varying the voltage applied to its terminal. One way of varying the applied voltage is by using the pulse-width modulation (PWM) technique. Using this technique, a fixed frequency voltage signal with varying pulse-width is applied to the motor terminal. Figure 7.2 shows an example of a PWM signal where T is the signal period, t_d is the pulse-width, and V_m is the signal amplitude. The average voltage can be calculated from

$$V_{avg} = \frac{1}{T} \cdot \int_0^T v(t) dt = \frac{t_d}{T} \cdot V_m = k \cdot V_m \quad (7.9)$$

where k is the duty cycle defined as :

$$k = \frac{t_d}{T} \quad (7.10)$$

From equation (7.9) it can be seen that the average (DC component) of the voltage signal is linearly related to the pulse-width of the signal, or the duty cycle of the signal since the period is fixed. Therefore, varying the duty cycle of the signal can alter the voltage applied to the motor terminal.

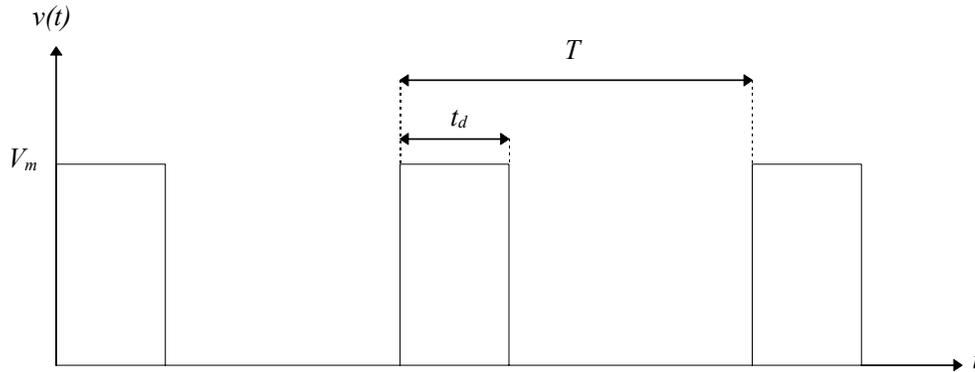


Figure 7.2 A PWM signal

The PWM voltage waveforms for the motor can be obtained using a special power electronic circuit called a *DC chopper*. A DC chopper basically uses power switching devices to switch a constant DC voltage on and off according to a specified switching scheme in order to obtain the required voltage and current waveforms. There are various types of DC chopper configurations, which can be found in textbooks on power electronics. Reference [1] given at the end of this chapter gives excellent explanations on different types of DC chopper circuits and their theories of operations.

In this section, we will discuss one type of DC chopper configuration called *bridge power converter* also known as *H-bridge converter*. The schematic diagram of this converter is shown in Figure 7.3. T1 to T4 are controlled switches that can be implemented using power semiconductor devices such as BJT, Power MOSFET, or IGBT. These devices provide low resistance for the current flow when they are turned on and very high resistance when turned off. Diodes D1 through D4 provide a path for preserving the continuity of the current flow when one or more of the switches are turned off. This is necessary to protect the power switches from excessive voltage spike due to

the inductive load presented by the DC motor. These diodes are also known as *freewheeling diodes*.

The DC voltage supply V_m can be obtained from a rectified ac signal or a DC voltage source such as a car battery.

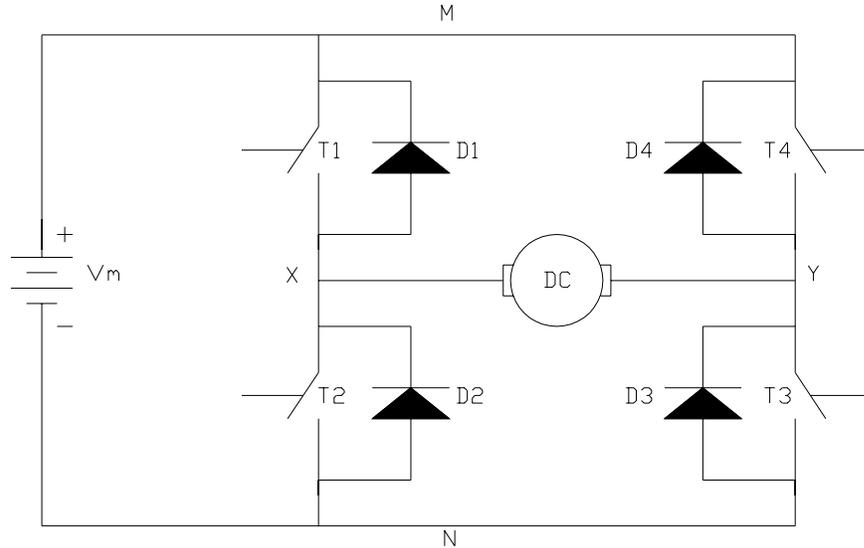
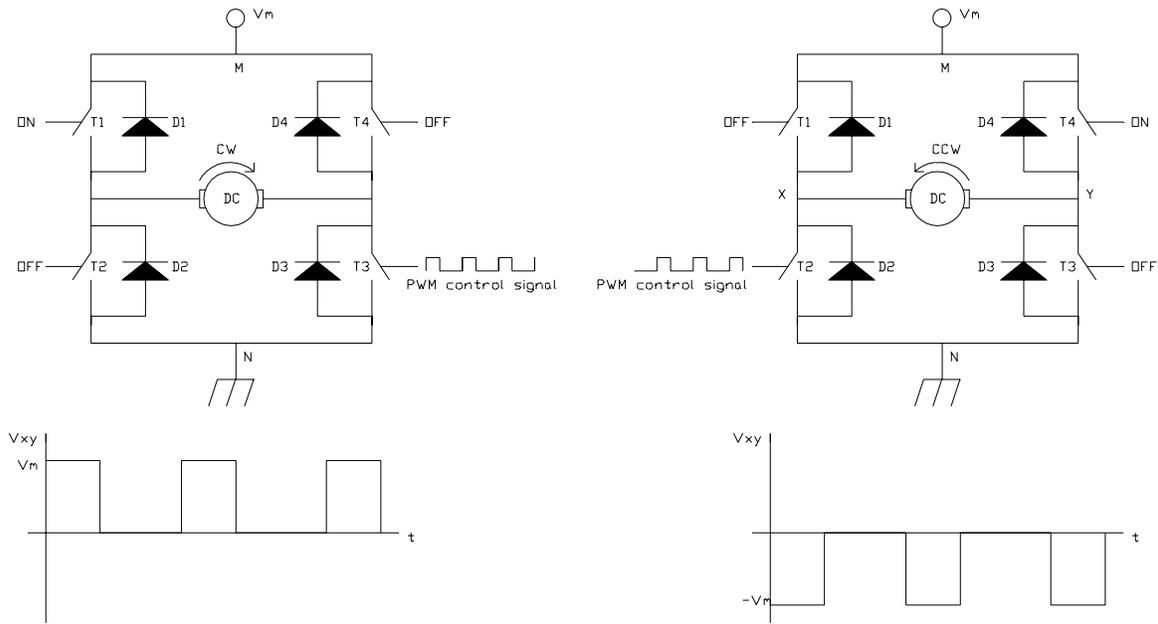


Figure 7.3 H-bridge converter for DC motor drive

In this section, we will examine two switching schemes for the bridge converter that can be used to drive the motor. Both schemes allow polarity reversal of the voltage applied across the motor and hence, provide for bi-directional control of the motor. The following paragraphs discuss the basic principles for the two switching schemes.

Scheme 1

This scheme uses only two of the switches of the bridge converter to drive the motor in one direction. Figure 7.4 shows how the switches are controlled to obtain the pulse-width modulated voltage waveform at the motor terminal. Figure 7.4(a) shows how the switches are controlled to drive the motor in one direction (say CW), while figure 4(b) to drive the motor in the other direction (CCW). The resulting voltages across the motor, for both cases are also shown in this figure.



(a)

(b)

Figure 7.4 Switching control for Scheme 1
 (a) CW direction ($V_{XY} > 0$)
 (b) CCW direction ($V_{XY} < 0$)

In Figure 7.4 (a) switches T2 and T4 are kept off, and T1 is kept on all the time, while the PWM control signal is applied to switch T3. Figure 7.5 shows the flow of current when T3 is driven ON and OFF. In this figure the motor is replaced with its equivalent circuit and only devices that take part in current conduction are drawn. When T3 is driven ON, node Y is connected to ground which causes current to flow from the supply voltage to ground through T1, the motor, and T3. When T3 is turned OFF, the current will continue to flow due to the energy stored in the inductance of the motor. In this case, diode D4 will turn on and the current flows through closed path consisting of the T1, the motor, and D4.

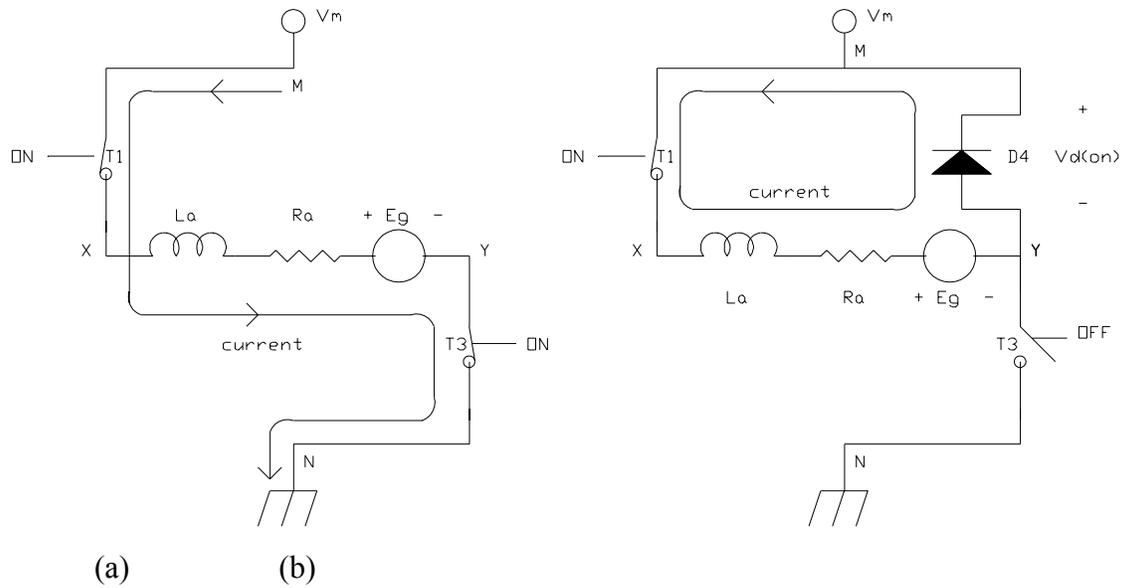


Figure 7.5 Current flow paths when T3 is ON (a) and OFF (b) in Figure 7.4(a)

The steady state voltage and current waveforms resulting from the switching scheme of Figure 7.4 (a) are shown in Figure 7.6. When T3 is ON, the current ramps up exponentially with a time constant determined by the motor inductance and the total resistance of the current flow path. The voltage across the motor terminal V_{XY} , in this case, is equal to the supply voltage V_m . When T3 is OFF, the current decays exponentially through T1 and D4 as in Figure 7.5 (b) and the voltage across the motor terminal V_{XY} will be equal to the negative of the diode turn on voltage (normally in the order of 0.6V - 0.8V).

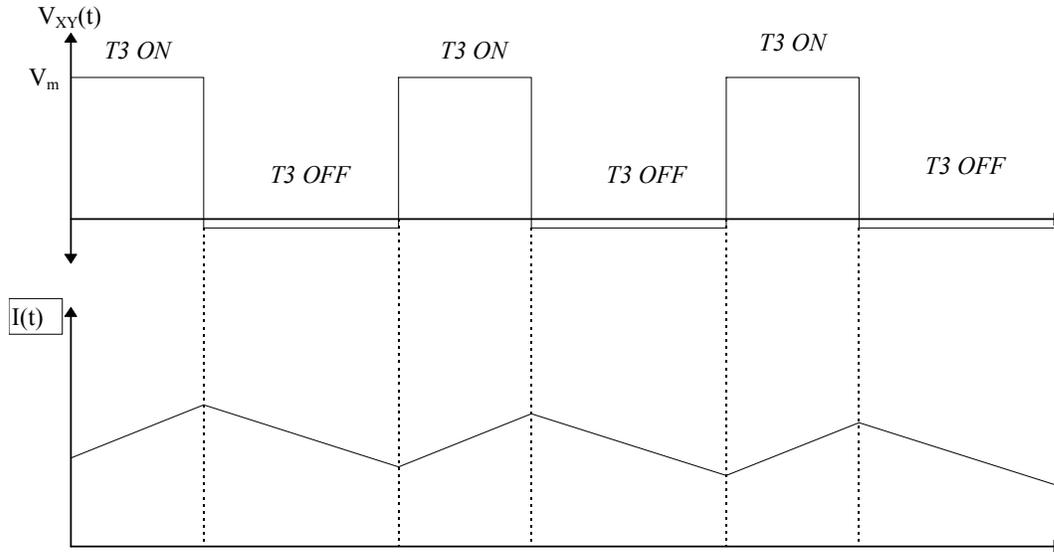


Figure 7.6. Voltage and current waveforms for Figure 7.4 (a)

When the motor is driven in the CCW direction, switches T1 and T3 are turned OFF, and T4 is turned ON all the time, while the PWM control signal is applied to switch T2. When T2 is turned ON the current now flows from the supply voltage to the ground through T4 the motor, and T2. This causes the motor terminal voltage V_{XY} to be equal to the negative of the supply voltage V_m . When T2 is turned OFF, the current flows through diode D1, and the voltage across the motor is clamped to the diode turn-on voltage. The voltage and current waveforms for the case when the motor is driven in the CCW direction (Figure 7.4(a)) are the same as in Figure 7.6 except that the polarities are reversed.

Scheme 2

In this scheme, all the four switches of the bridge converter are utilized to drive the motor in one direction. Looking at the bridge configuration in Figure 7.3, we can see that there are two ways to force the voltage at the motor terminal (V_{XY}) to zero i.e. either by turning on T1 and T4 or by turning on T2 and T3 simultaneously. To drive the motor in the CW direction the switches are controlled as shown in Figure 7.7. The current flow paths are also shown in Figure 7.7.

The scheme uses four states of the switches to obtain the PWM voltage waveform at the motor terminal. To make $V_{XY} = V_m$, switches T1 and T3 are turned ON while T2 and

T4 are turned off (state I and state III). To force $V_{XY}=0$, either T1 and T4 are turned ON (state II) or T2 and T3 are turned ON (state IV). The four switches states are summarized in Table 7.1 below.

State	T1	T2	T3	T4
I	ON	OFF	ON	OFF
II	ON	OFF	OFF	ON
III	ON	OFF	ON	OFF
IV	OFF	ON	ON	OFF

Table 7.1 *Switches states for CW drive*

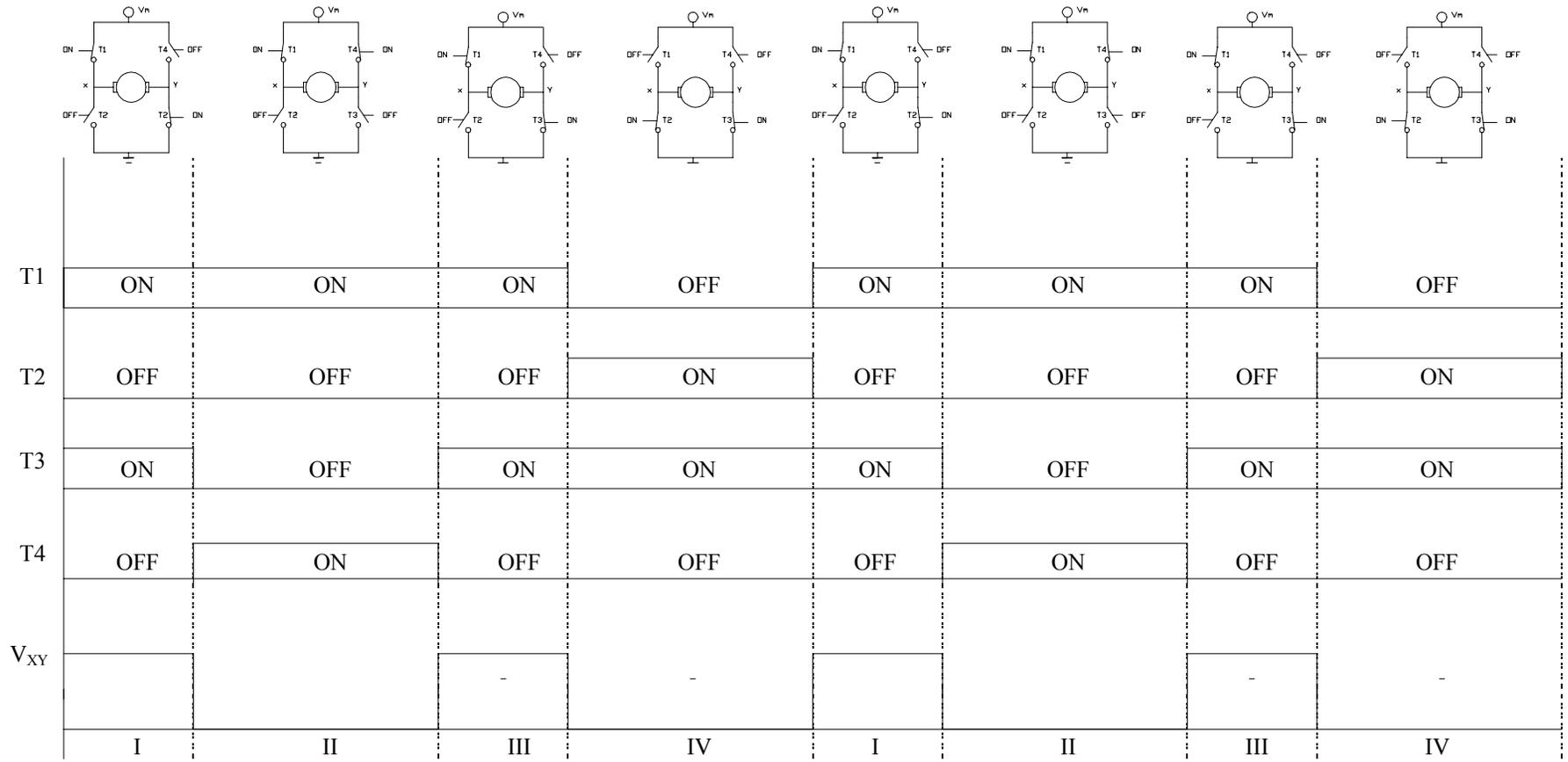


Figure 7.7 Switches control for CW drive (scheme 2)

To drive the motor in the other direction, we need to obtain $V_{XY} = -V_m$ during the high period of the signal (state I and state III). This can be accomplished by turning on switches T2 and T4. The switch states for this case are shown in Table 7.2.

State	T1	T2	T3	T4
I	OFF	ON	OFF	ON
II	ON	OFF	OFF	ON
III	OFF	ON	OFF	ON
IV	OFF	ON	ON	OFF

Table 7.2 Switches states for CCW drive

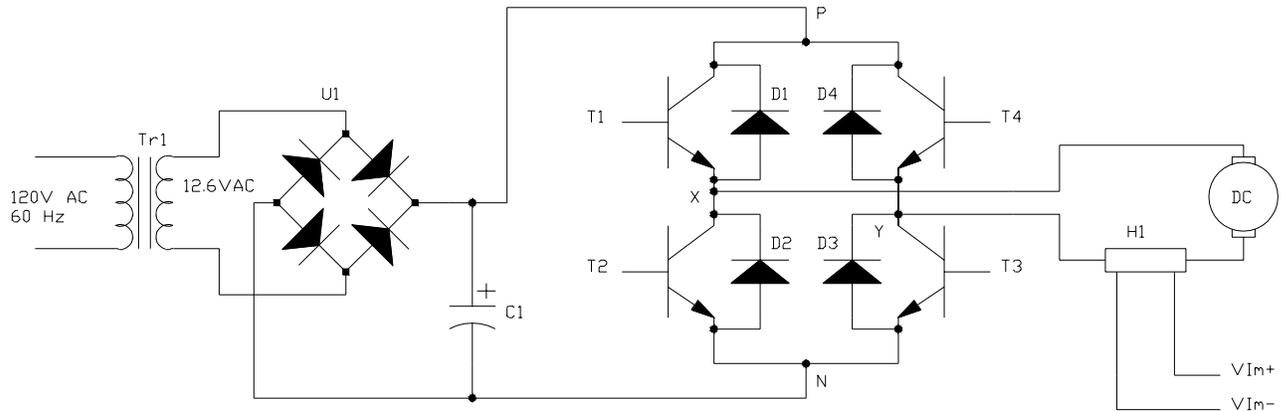
Note that in the above schemes, two switches that reside in the same leg of the bridge (T1 and T2 or T3 and T4) should not be turned on at the same time since this would result in the supply voltage being short circuited to ground.

7.3 Circuits realizations

As mentioned earlier, the switches in the H-bridge are implemented using power semiconductor devices. Below the realization of an H-bridge power converter using BJT power transistors as the switching devices is discussed. In this section, we will also discuss how to interface an H-bridge power converter to the DSP ports. In this case, a special interface circuit called *base-drive circuit* will be developed.

7.3.1 BJT Bridge Converter

Figure 7.8 shows the H-bridge converter implemented using BJT power transistors. It uses 4 BJT power transistors of the type 2N6547. The 2N6547 is an NPN power transistor which can handle collector current up to 10 amp and is specifically designed for use with inductive loads such motor or solenoid. The freewheeling diodes are implemented using MUR410, 10A ultra-fast switching diodes.



Component list:

T1, T2, T3, T4: 2N6547 Tr1 : 12.6V, 3 A transformer
D1, D2, D3, D4 : MUR410 C1 : 5000uF electrolyte capacitor
U1 : KBL01 diode bridge rectifier H1 : SY-05 current sensor.

Figure 7.8 H-Bridge Power Converter

In Figure 7.8, the DC supply for the motor is obtained from a rectified ac-signal. The 120 V ac voltage source is first converted to a lower level ac voltage using a step down transformer. A diode bridge rectifier (U1) then rectifies the secondary output of the transformer. A large capacitor C1 in Figure 7.8 is used to filter the high frequency content of the rectified signal.

The circuit in Figure 7.8 also includes an SY-05 Hall effect current sensor (H1) which can be used to detect or observed the motor current. This device produces an output voltage proportional to the current flowing through its sensing terminal.

The data-sheets for the devices used in the circuit are given at the end of the chapter.

7.3.2 Interface circuit (Base-drive Circuit)

The base drive circuit is used to interface the bridge converter to the DSP. Following are functions that must be carried out by the base drive circuits:

- 1) The base drive circuit must be able to supply sufficient current and apply necessary voltage to the base of each transistor in the bridge converter to ensure proper turn on and off of the devices. Note that the emitters of the upper transistors T1 and T4 are connected to the floating nodes X and Y respectively, while the emitter of the lower transistors T2 and T3 are both connected to node N. This interconnection requires that the voltage signal that drives the base of each transistor be referenced to its corresponding emitter nodes.
- 2) The base drive circuit must provide a mechanism that prevents the transistors that reside at the same leg of the bridge converter to turn on at the same instant of time. Recall that in theory the switching schemes described earlier never turns on the transistors at one leg at the same time. In the scheme 2, for example, T1 and T2 (or T3 and T4) are always switched in the opposite states, one being ON and the other OFF. However, a problem might occur during the transitions between the ON and OFF states, due to the fact that the switching devices used in practice can not turn ON or turn OFF instantaneously. To prevent short-circuiting during this period, we must allow one transistor to turn OFF completely before turning ON the other transistor by delaying the turn ON process. A circuit that accomplishes this is called a *lock out circuit*. The implementation of the base drive circuit, therefore, must include the lock out circuits.
- 3) The base drive circuit must provide isolation between the power converter that carries high power signals, and the DSP circuitry that carries low level digital signals. This can be accomplished using opto-couplers as we have seen in the previous chapters.
- 4) The base drive circuit must ensure that currents drawn from the DSP output ports do not exceed the maximum permissible current dictated by the DSP chip manufacturer. The circuit, therefore, must contain buffer devices that enhance the DSP output current capability.

Figure 7.9 shows a base drive circuit that can accomplish the requirements stated above. The base drive contains four drives each for the four transistors in the bridge converter. The output stage of each drive consists of a fast high power op-amp, OPA633 (U3), which can supply current up to 100mA maximum. To satisfy the requirement that the emitter node of each transistor is referenced to the different nodes at the converter, three pairs of power supplies with three separate ground references are used to drive the output stages. Each power supply pair denoted by VPPx (+5V) and VNNx (-5V) (x=1,2,3) is referenced to its own ground point denoted by GNDx (x=1,2,3). VPP1 and VNN1 are used for T1 drive (GND1=node X), VPP2 and VNN2 are used for T4 drive (GND2=node Y), while VPP3 and VNN3 are used for T3 and T4 which share a common emitter node (node N at the bridge converter = GND3).

The output stage of each drive that carries relatively high current is separated from the rest of the circuits using a high-speed opto-coupler HP2200 (U1). The output of the each opto-coupler is pulled to the corresponding VPPx and VNNx supplies. This causes the magnitudes of the opto-coupler output voltages to be $\pm 5V$. The $\pm 5V$ applied to the input of the power op-amps will cause the op-amps to saturate and produce output voltages of magnitude approximately $\pm 3.5V$ that will turn on and off the power transistors.

The lock out function in each drive is implemented using a circuit consisting of an IC comparator (U2: LM339) and a network of resistors, capacitors and a diode. The operation of the lock out circuit can be explained by looking at Figure 7.10 where one of the lockout circuit is redrawn. Figure 7.11 shows the effect of the lockout circuit on the output of the opto-coupler that drives the power op-amp.

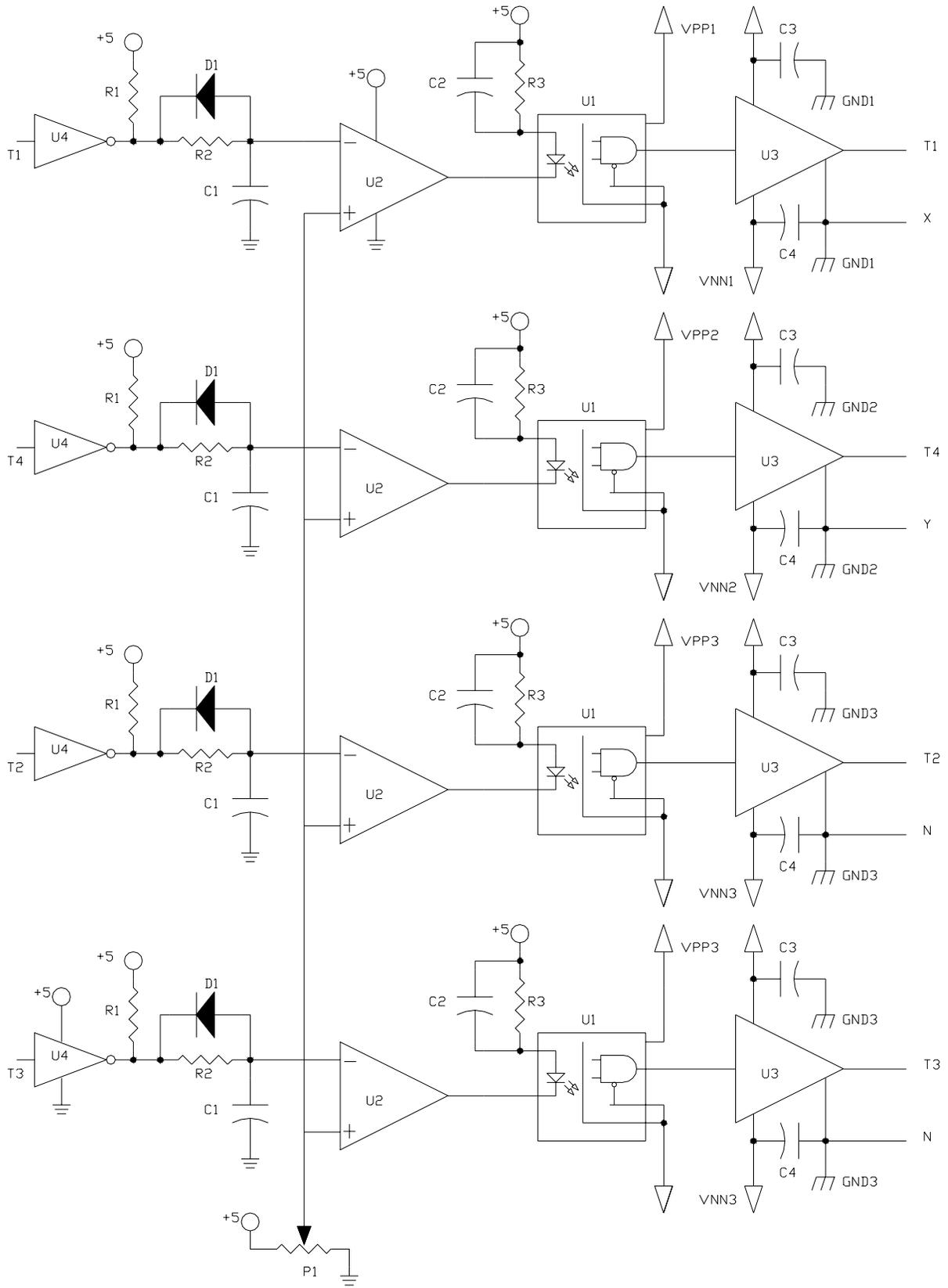


Figure 7.9. Base drive circuit for the BJT H-bridge converter

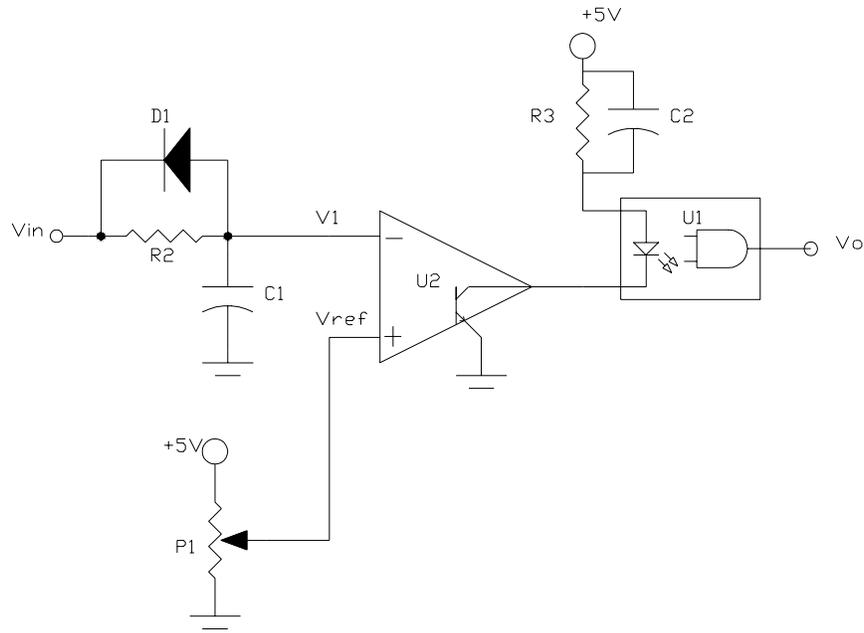


Figure 7.10 Lockout circuit

The IC comparator LM339 (U2) used in the lockout circuit has an open collector output as shown in Figure 7.10. When the difference between the non-inverting and the inverting input of the comparator ($V_{ref} - V_1$) is positive the output transistor inside the IC is turned ON pulling the output to ground. This causes the LED of the opto-coupler to turn ON and the output of the opto-coupler becomes +5V (VPP). When the difference is negative the output transistor is turned off causing the LED to turn OFF and the output of the opto-coupler becomes -5V(VNN).

A reference input V_{ref} obtained from the +5V supply and a variable resistor P1 is applied to the non-inverting output of the comparator. When the input voltage V_{in} is LOW ($\sim 0V$), the inverting input is also zero and the output of the opto-coupler is -5V. When the input voltage changes from 0 to +5V, diode D1 turns off (since it is reversed biased), causing the current to flow through resistor R_2 and charge the capacitance C_1 . This causes the voltage at the inverting input of the comparator (V_1) to rise exponentially according to the equation:

$$V_1 = +5 \cdot \left(1 - e^{-\frac{t}{R_2 C_1}} \right)$$

When V_1 is still less than V_{ref} , the output transistor of the comparator is turned off preventing the opto-coupler output V_o to change immediately. Only when V_1 raises to a value greater than V_{ref} , will the output transistor turn ON and the output V_o change to +5V as shown in Figure 7.11. This way the process of turning on the power transistor is delayed and the amount of delay is determined by the time constant $R_2 C_1$ and the voltage reference V_{ref} . Since the values of the resistor R_2 and capacitor C_1 are fixed, the delay time can be adjusted by varying the value of V_{ref} using the potentiometer P1. Larger value of V_{ref} results in longer delay time, lower value causes shorter delay time.

On the other hand, if the input voltage V_{in} changes from HIGH ($\sim 5V$) to LOW ($\sim 0V$), diode D1 turns ON (since it becomes forward biased) providing a low resistance path for the charge stored in capacitor C_1 to discharge quickly to ground. This causes the value of V_1 to drop quickly below the reference voltage V_{ref} . This mechanism speeds up the change of output voltage V_o to VNN ($-5V$) and the turning OFF process the power transistor.

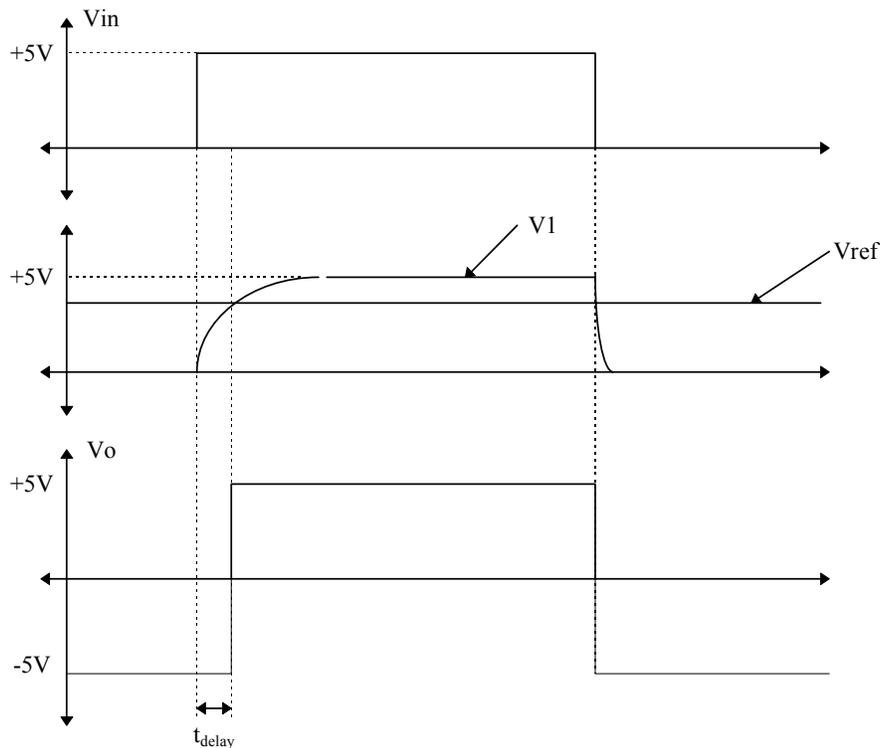


Figure 7.11. Effects of the lock-out circuit on the turning ON and turning OFF processes

The last requirement the base drive circuit must perform is to ensure the current drive capability of the DSP port is not exceeded. For this purpose the base drive circuit in Figure 7.9 includes inverting buffer gates U4 (7406 IC). *Note that the inclusion of the inverting buffers at the input of the base drive circuit inverts the logic for turning on the power transistor. Logic 0 at one of the inputs of the base drive circuit causes the corresponding power transistor to turn ON, logic 1 causes it to turn OFF.*

7.4 DSP Program

To control the operation of the H-bridge converter for the DC motor drive, the DSP must generate the required base drive signals. The base drive signals for the switching scheme 1 is relatively simple to implement, since it requires only a single PWM control signal to drive the motor in one direction. For this switching scheme the power converter and the base drive circuit may be connected to the DSP ports as shown in Figure 7.12. Since the lower transistors T2 and T3 need to be driven by PWM control signals (see Figure 7.4), the drives for these transistors are connected to the output pins of the DSP T1PWM and T2PWM. The upper transistors T1 and T4 do not require PWM control signals and therefore can be connected to any general purpose port pins. In Figure 12, T1 and T4 are connected to the I/O port B pins IOPB2 and IOPB5.

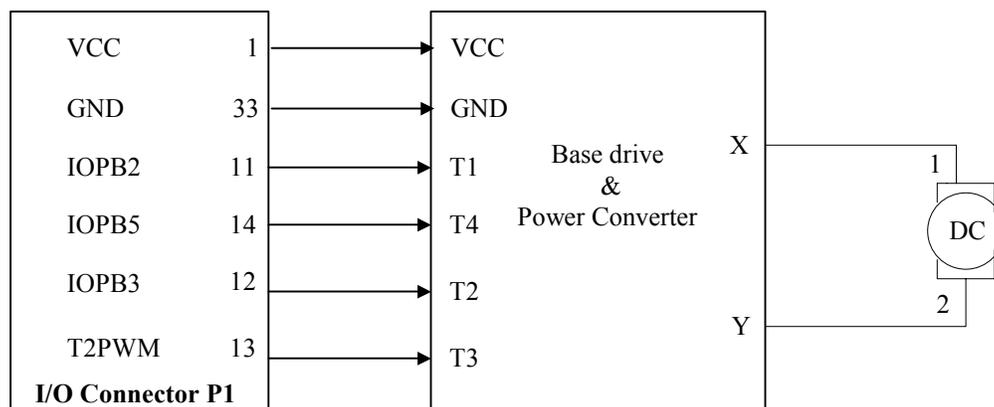


Figure 7.12 DSP EVB ports interconnections for switching scheme 1

Below is a program for the DSP that can be used to drive the motor in CW direction using the switching scheme 1 and the port interconnections as shown in Figure 7.12. The PWM signal used has 50% duty cycle and 20 kHz frequency.

Program 1

The basic scheme is to output a "0" at T1 to turn it ON. A PWM wave of duty cycle 50% and frequency 20 kHz at T1PWM. This PWM wave is generated using Timer 1 in continuous up-counting mode. A "1" is output at IOPB3 and IOPB5 in order to turn OFF T1 and T4.

The registers involved, are: OCRA, PADATDIR, GPTCON, TICON, T1PR, T1CMP.

OCRA = 1000h

- Configure the o/p pin T2PWM for timer2 PWM output to T3
- Configure o/p pins IOPB2, IOPB3, IOPB5 as digital outputs for T1, T2 and T4 respectively.

T2PR = 1000

Calculation of period value:

Period Value = {(CLKINOSC/PRESCALER) × (PLL MTPLN RATIO / PLL DIVIDE)} / DESIRED FREQ

Period Value = $\{(10^7 / 1) \times (4 / 2)\} / (20 \times 10^3) = 1000$

T2CMPR = 500 = T2PR/2

GPTCON = 0045h

Enable Compare outputs of all GPTs. GPT2 compare output - active low

T2CON = 1042h

Select continuous up counting mode, Prescaler = 1, enable timer compare operation, enable timer operation.

The complete code is given below -

```

;*****
; File Name:    Ch7_e1.asm
; Description: Program to drive motor in clockwise direction as per
;              scheme 1.
;*****
                .include f240regs.h

```

```

cmpr_val .set 500      ;Value to be loaded in the GPT
          ;compare register
per_val .set 1000     ;Value to be loaded in the GPT
          ;period register
;-----
; Variable Declarations for on chip RAM Blocks
;-----
        .bss GPR0,1 ;General purpose register.
;-----
; Vector address declarations
;-----
        .sect ".vectors"

RSVECT B    START      ; Reset Vector
INT1    B    PHANTOM   ; Interrupt Level 1
INT2    B    PHANTOM   ; Interrupt Level 2
INT3    B    PHANTOM   ; Interrupt Level 3
INT4    B    PHANTOM   ; Interrupt Level 4
INT5    B    PHANTOM   ; Interrupt Level 5
INT6    B    PHANTOM   ; Interrupt Level 6

RESERVED B    PHANTOM   ; Reserved
SW_INT8 B    PHANTOM   ; User S/W Interrupt
SW_INT9 B    PHANTOM   ; User S/W Interrupt
SW_INT10 B    PHANTOM  ; User S/W Interrupt
SW_INT11 B    PHANTOM  ; User S/W Interrupt
SW_INT12 B    PHANTOM  ; User S/W Interrupt
SW_INT13 B    PHANTOM  ; User S/W Interrupt
SW_INT14 B    PHANTOM  ; User S/W Interrupt
SW_INT15 B    PHANTOM  ; User S/W Interrupt
SW_INT16 B    PHANTOM  ; User S/W Interrupt
TRAP    B    PHANTOM   ; Trap vector
NMINT   B    PHANTOM   ; Non-maskable Interrupt
EMU_TRAP B    PHANTOM  ; Emulator Trap
SW_INT20 B    PHANTOM  ; User S/W Interrupt
SW_INT21 B    PHANTOM  ; User S/W Interrupt
SW_INT22 B    PHANTOM  ; User S/W Interrupt
SW_INT23 B    PHANTOM  ; User S/W Interrupt

;=====
; M A I N   C O D E   - starts here
;=====

```

```

.text
NOP
START: SETC INTM          ;Disable interrupts
      SPLK #0000h,IMR      ;Mask all core interrupts
      LACC IFR             ;Read Interrupt flags
      SACL IFR            ;Clear all interrupt flags

      CLRC SXM            ;Clear Sign Extension Mode
      CLRC OVM            ;Reset Overflow Mode
      CLRC CNF            ;Config Block B0 to Data mem

      LDP #00E0h          ;DP for addresses 7000h-707Fh
      SPLK #00BBh,CKCR1    ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
      SPLK #00C3h,CKCR0    ;CLKMD=PLL Enable, SYSCLK =
                          ;CPUCLK/2
      SPLK #40C0h,SYSCR    ;CLKOUT=CPUCLK

      SPLK #006Fh, WDCR    ;Disable WD if VCCP=5V(JP52-3)
      KICK_DOG            ;Reset Watchdog

;-----
;   SET UP DIGITAL I/O PORT
;-----

      LDP #00E1h          ;DP for addresses 7080-70FFh
      SPLK #0000h, OCRA    ;CONFIGURE ALL PINS FOR I/O
      SPLK #3C3Ch, PBDATDIR ;Turn off T1, T2, T3 and T4 by
                          ;outputting 1 on all pins.

      SPLK #1000h, OCRA    ;Configure the o/p pins for T2PWM,
                          ;IOPB2, IOPB3, IOPB5. i.e. output
                          ;PWM at input of T3.
      SPLK #2C28h, PBDATDIR ;Turn on T1

      LDP #00E8h          ;DP for addresses 7400-747Fh
      SPLK #0045h,GPTCON  ;Enable Compare outputs of all GPTs
                          ;GPT1 compare output - active low
      SPLK #cmpr_val, T2CMPR ;Load compare register
      SPLK #per_val, T2PR  ;Load period register
      SPLK #1042h, T2CON  ;Select continuous up counting mode
                          ;Prescaler = 1, enable timer compare

```

```

                                ;operation, enable timer operation.
END      B      END      ;End of program
;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM  KICK_DOG      ;Resets WD counter
      B PHANTOM

```

It can be seen that the program that drives the motor using scheme 1 is very straightforward and simple. This is due to the simplicity of programming the PWM unit in the DSP. For switching scheme 2, this advantage cannot be utilized since we need to control the switching of the four transistors at the same time. In this case the base drive and power converter can be connected as shown in Figure 7.13.

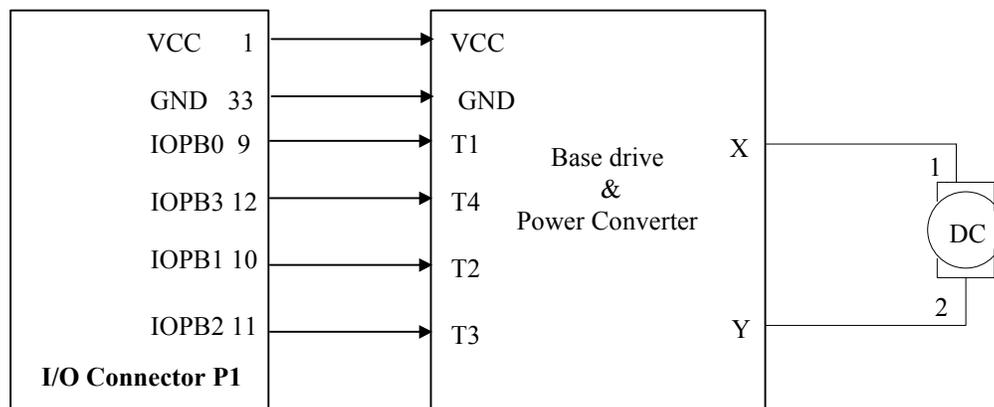


Figure 7.13 DSP EVB ports interconnections for switching scheme 2

Example 2:

Write a program to drive the motor in clockwise direction using scheme 2. The frequency of the PWM is 1 kHz at a duty cycle of 50%. Also, compute the speed by reading the input of an incremental encoder. The encoder outputs 2950 pulses/revolution. Store the number of pulses generated per second in a memory location *SPEED*.

Solution:

The pulses are output in four states as per the following sequence:

State	T1		T2		T3		T4	
	Command	Value	Command	Value	Command	Value	Command	Value
I	ON	0	OFF	1	ON	0	OFF	1
II	ON	0	OFF	1	OFF	1	ON	0
III	ON	0	OFF	1	ON	0	OFF	1
IV	OFF	1	ON	0	ON	0	OFF	1

Thus in each period, there are four states. The period interval is split into four equal sub-intervals. At the end of each interval, the pattern for a state is output. Timer T1 keeps track of each sub-interval. The period of the timer is set to a value corresponding to a frequency of $1 \text{ kHz} / 4 = 250\text{Hz}$. The timer period interrupt is set. Thus, the timer period ISR decides which sub-interval it is, and accordingly outputs the right pattern.

The incremental encoder is connected to the capture unit 4 i.e. the CAP4 input. The capture unit is programmed to generate an interrupt every time a rising edge is detected at CAP4. The CAP4_ISR counts the pulses from the encoder. At the end of every 1 second interval, the timer 1 ISR stores the number of pulses in SPEED and resets the pulse counter. The actual speed of the motor in rpm can be computed as -

$$\text{RPM} = (\text{SPEED}/2950) * 60$$

The program listing is as below -

```

;*****
; File Name:   Ch7_e2.asm
; Description:   Program to drive motor in clockwise direction as per
;               scheme 2.The program also counts the number of pulses ;   of
encoder in one second as a measure of the speed.
;*****
        .include f240regs.h

STATE1  .set 0F0Ah
STATE2  .set 0F06h
STATE3  .set 0F0Ah
STATE4  .set 0F09h

        .bss ENCDR_CNT,1 ;Encoder pulse counter

```

```

        .bss SEC_CTR,1 ;Second counter
        .bss SPEED,1  ;Value of speed
        .bss CTR,1    ;State counter
        .bss GPR0,1

;-----
; Vector address declarations
;-----

        .sect ".vectors"

RSVECT B    START    ; Reset Vector
INT1    B    PHANTOM ; Interrupt Level 1
INT2    B    TMR_ISR ; Interrupt Level 2
INT3    B    PHANTOM ; Interrupt Level 3
INT4    B    CAP4_ISR ; Interrupt Level 4
INT5    B    PHANTOM ; Interrupt Level 5
INT6    B    PHANTOM ; Interrupt Level 6
RESERVED B    PHANTOM ; Reserved
SW_INT8 B    PHANTOM ; User S/W Interrupt
SW_INT9 B    PHANTOM ; User S/W Interrupt
SW_INT10 B    PHANTOM ; User S/W Interrupt
SW_INT11 B    PHANTOM ; User S/W Interrupt
SW_INT12 B    PHANTOM ; User S/W Interrupt
SW_INT13 B    PHANTOM ; User S/W Interrupt
SW_INT14 B    PHANTOM ; User S/W Interrupt
SW_INT15 B    PHANTOM ; User S/W Interrupt
SW_INT16 B    PHANTOM ; User S/W Interrupt
TRAP    B    PHANTOM ; Trap vector
NMINT   B    PHANTOM ; Non-maskable Interrupt
EMU_TRAP B    PHANTOM ; Emulator Trap
SW_INT20 B    PHANTOM ; User S/W Interrupt
SW_INT21 B    PHANTOM ; User S/W Interrupt
SW_INT22 B    PHANTOM ; User S/W Interrupt
SW_INT23 B    PHANTOM ; User S/W Interrupt

;=====
; M A I N   C O D E   - starts here
;=====

        .text
        NOP
START: SETC INTM    ;Disable interrupts

```

```

SPLK #000Ah,IMR ;Mask all core interrupts
LACC IFR ;Read Interrupt flags
SACL IFR ;Clear all interrupt flags

CLRC SXM ;Clear Sign Extension Mode
CLRC OVM ;Reset Overflow Mode
CLRC CNF ;Config Block B0 to Data mem

LDP #00E0h ;DP for addresses 7000h-707Fh
SPLK #00BBh,CKCR1;CLKIN(OSC)=10MHz,CPUCLK=20MHz
SPLK #00C3h,CKCR0;CLKMD=PLL Enable,SYSClk=CPUCLK/2
SPLK #40C0h,SYSCR;CLKOUT=CPUCLK

SPLK #006Fh, WDCR;Disable WD if VCCP=5V (JP5 in pos. 2-3)
KICK_DOG ;Reset Watchdog

SPLK #0h,GPR0 ;Set wait state generator for:
OUT GPR0,WSGR ;Program Space, 0 wait states
;Data Space, 0 wait states
;I/O Space, 0 wait states

LDP #0
SPLK #0, ENCDR_CNT
SPLK #0, CTR
SPLK #0, SEC_CTR
SPLK #0, SPEED

LDP #00E1h
SPLK #0000h, OCRA ;Select IOB0-3 as outputs
SPLK #0080h, OCRB ;sELECT CAP4 FUNTION OF PIN
;-----
; RESET SECTION - BEGINS
;-----

LDP #00E8h
SPLK #0, GPTCON
SPLK #0, T1CON
SPLK #0, T2CON
SPLK #0, T3CON

SPLK #0, COMCON
SPLK #0, ACTR

```

```

SPLK #0, SACTR
SPLK #0, DBTCON
SPLK #0, CAPCON

SPLK #0FFFFh, EVIFRA
SPLK #0FFFFh, EVIFRB
SPLK #0FFFFh, EVIFRC

SPLK #0, EVIMRA
SPLK #0, EVIMRB
SPLK #0, EVIMRC

;-----
; RESET SECTION - ENDS
;-----

T1PERIOD .set 2500
LDP #00E8h
SPLK #0055h, GPTCON
SPLK #T1PERIOD, T1PR
SPLK #0, T1CNT
SPLK #0, T2CNT
SPLK #0, T3CNT

SPLK #150Ah, T1CON ;SET PRESCALER =32
SPLK #8801h, CAPCON
SPLK #00FFh, CAPFIFO
SPLK #0080h, EVIMRA
SPLK #0008h, EVIMRC

LACC EVIFRC
SACL EVIFRC
LACC EVIFRA
SACL EVIFRA

LDP #0
LACC IFR
SACL IFR

LDP #00E8h
SBIT1 T1CON, B6_MSK ;Enable timer 1
CLRC INTM
END B END

```

```

;=====
; TMR_ISR -
; Description: Outputs the proper sequence for all the transistors.
;           Loads the number of pulses in SPEED at the end of
;           every 1 sec interval.
; Modifies:  SPEED, ENCDR_CNT, SEC_CTR
;=====
TMR_ISRLDP #0
    LACC SEC_CTR
    ADD #1
    SUB #250
    BCND NXT, NEQ      ;Check if 1 sec
    LACC ENCDR_CNT     ;If 1 sec then SPEED=ENCDR_CNT
    SACL SPEED
    SPLK #0, ENCDR_CNT ;Reset ENCDR_CNT and SEC_CTR
    SPLK #0, SEC_CTR
    B NXT1
NXT    LACC SEC_CTR     ;If not 1 sec increment SEC_CTR
    ADD #1
    SACL SEC_CTR
NXT1   LACC CTR
    BCND ST2, NEQ      ;Check the state number if 0
    LDP #00E1h
    SPLK #STATE1, PBDATDIR ;Output sequence for STATE1
    LDP #0
    SPLK #1,CTR        ;Update state counter
    B FIN
ST2    SUB #1
    BCND ST3,NEQ      ;Check the state number if 1
    LDP #00E1h
    SPLK #STATE2, PBDATDIR ;Output sequence for STATE2
    LDP #0
    SPLK #2,CTR        ;Update state counter
    B FIN
ST3    SUB #1
    BCND ST4,NEQ      ;Check the state number if 2
    LDP #00E1h
    SPLK #STATE3, PBDATDIR ;Output sequence for STATE3
    LDP #0
    SPLK #3,CTR        ;Update state counter
    B FIN

```

```

ST4    LDP    #00E1h
        SPLK  #STATE4, PBDATDIR    ;Output sequence for STATE4
        LDP    #0
        SPLK  #0, CTR    ;Reset state counter
FIN    LDP    #00E8h
        LACC  EVIFRA
        SACL  EVIFRA

        CLRC  INTM
        RET

;=====
; CAP4_ISR -
; Description: Counts the number of pulses from encoder.
; Modifies: ENCDR_CNT
;=====
CAP4_ISR  LDP    #0
          LACC  ENCDR_CNT
          ADD   #1
          SACL  ENCDR_CNT    ;Increment the pulse counter
          LDP    #00E8h
          LACC  EVIFRC
          SACL  EVIFRC
          CLRC  INTM
          RET

;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
;=====
PHANTOM  KICK_DOG    ;Resets WD counter
          B    PHANTOM

```

Laboratory Experiment 6

Objectives

To understand the principles of DC motor drives using an H-bridge converter and its interface to the DSP (base drive circuit)

To write programs for the DSP to control the speed of the DC motor using the H-bridge converter and its base drive circuit (open loop control of DC-motor)

Equipment

Hardware :

- PC Specifications -
 - ❑ '386 or higher IBM PC/AT
 - ❑ 1.44Mb 3.5-inch floppy drive
 - ❑ 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
 - ❑ Minimum 4Mb memory
 - ❑ Color VGA Monitor

- TMS320C24x Evaluation Board
- XDS510PP Emulator
- +5V power supply.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable
- Encoder interface circuit board
- BJT H-Bridge Power converter
- Base drive circuit
- Pittman 24 V PM DC Motor with HP incremental encoder

Software :

- MS Windows 3.1 or Windows 95
- ASCII Editor
- TMS320C2xx Assembler
- TMS320C2xx C Source Debugger

Discussion

The DC motor used in the experiments is a Pittman PM DC Motor with HP incremental encoder. The cross section of the motor is shown in Laboratory Figure 7.1 below. The PM DC motor can be operated from 1.5V to 24V terminal voltage and is reversible.

The motor is equipped with an incremental encoder from HP, HEDS9100. The encoder produces a single channel output of 2950 pulses per revolution of the shaft. The encoder is powered from a +5V DC voltage and has output voltage compatible with the TTL logic devices.

Additional Reading

Chapter 7. (DC Motor Drive)

Chapter 3 (Interrupts and Timer Operations)

Procedure

I. Setup

1. Make sure that the EVB system has been properly setup as in the previous lab.
2. Make sure that the connections between the base drive and power converter are properly installed. Do not make any connections from the base drive circuit to the EVB yet!
3. Connect the motor terminal to the X and Y outputs of the power converter.

4. Turn on the PC and wait until the MS-Windows starts
5. Turn on the EVB power supply and the power for all the boards.
6. Run the EVB Testing program.

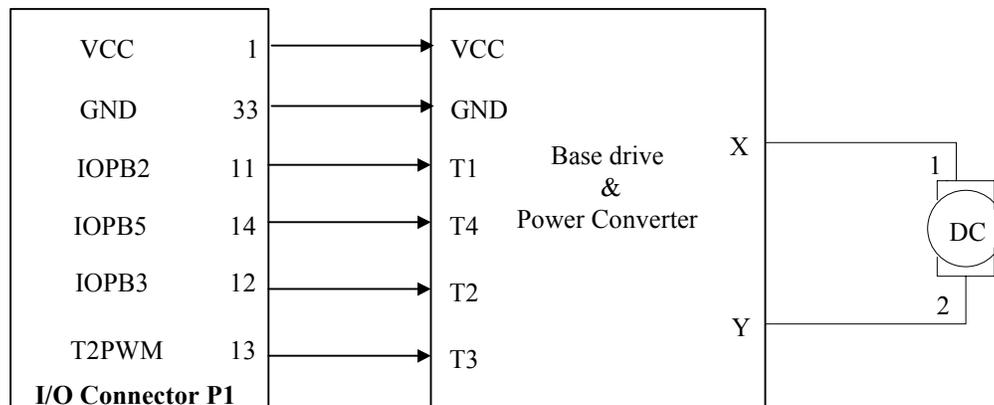
II. Reset conditions of the DSP ports

1. Using a multimeter measure the output voltages at the following DSP ports: IOPB2, IOPB3, T2PWM and IOPB5.
2. The states of the output ports you measured in 1 represent the reset condition of the ports. Using the information obtained, can you explained the reason why the base drive circuit is designed to have an inverting logic (0 = Transistor ON, 1 = Transistor OFF)

Laboratory Assignments

I. Switching scheme 1

1. Connect the base drive and power converter circuit to the DSP as shown in Laboratory Figure 7.2



Laboratory Figure 7.2 DSP ports interconnections for switching scheme 1

2. Power On the Base drive & Power converter circuit.
3. Adjust the voltage V_{ref} of the lock out circuit in the base drive to zero. Briefly explain why for switching scheme 1, delaying the turn ON process is not necessary!

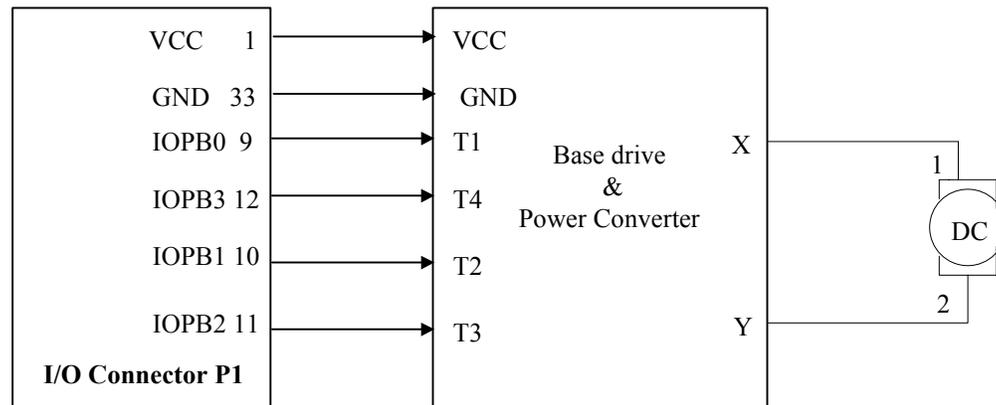
4. Write a program for the DSP that drives the motor with a PWM signal with the following duty cycles at frequency of 1 kHz
 - a) 25 %
 - b) 50%
 - c) 75%
5. Download and run your programs in the EVB and observe the following using the oscilloscope :
 - a) voltage at the motor terminal (V_{xy})
 - b) output voltage of the current sensor (current waveform)
 - c) output waveforms of the encoder.

For each case determine the speed of the motor from the period of the encoder output signal. Answer the following questions:

- a) Does the speed change proportionally with the duty cycle?
 - b) Did you obtain the terminal voltage and current waveforms as explained in the theory in Chapter 7. *Hint: Use the current flow paths as shown in Figure 7.5, to explain any peculiarities in the observed waveforms.*
6. Repeat step 4 and 5 for the following PWM signal frequencies:
 - (a) 2.05 kHz
 - (b) 4.09 kHz
 - (c) 8.19 kHz
 - (d) 16.4 kHz
 7. Rewrite your program to drive the motor in the other direction and experiment with different duty cycles and frequencies

II. Switching scheme 2

1. Connect the base drive and power converter circuit to the DSP as shown in Laboratory Figure 7.3



Laboratory Figure 7.3 DSP ports interconnections for switching scheme 2

2. Calculate the voltage V_{ref} required to obtain a 10 microseconds turn ON delay for the lockout circuit. Adjust V_{ref} in the base drive circuit to this calculated value.
3. Write a program for the DSP that drives the motor with a PWM signal with the following duty cycles at frequency of 1.0 kHz.
 - a) 25 %
 - b) 50%
 - c) 75%
5. Download and run your programs in the EVB and observe the following using the oscilloscope:
 - a) voltage at the motor terminal (V_{xy})
 - b) output voltage of the current sensor (current waveform)
 - c) output waveforms of the encoder.

For each case determine the speed of the motor from the period of the encoder output signal. Answer the following questions:

- a) Does the speed change proportionally with the duty cycle?

- b) Did you obtain the same output voltage and current waveforms as in the previous experiment?
- 6. Repeat step 4 and 5 for the following PWM signal frequencies:
 - (a) 2.kHz
 - (b) 4 kHz
 - (c) 8 kHz
- 7. Rewrite your program to drive the motor in the other direction and experiment with different duty cycles and frequencies

CHAPTER 8

SERIAL COMMUNICATIONS INTERFACE

In this chapter, we will discuss the Serial Communication Interface (SCI) of the TMS320F240. These interfaces allow digital communication between the controller and external peripherals / other controller, in asynchronous and isosynchronous modes.

8.1 Introduction to Serial Communication

In this laboratory, the serial communication between a TMS320F240 EVM and a PC will be implemented and tested although the term “serial communication” itself can be generalized to PC-to-PC, DSP-to-DSP, and so on. Without losing generality, the introduction to the fundamentals of serial communication starts from what you may be more familiar with – a PC:

All IBM PC and compatible computers are typically equipped with two serial ports and one parallel port. Although these two types of ports are used for communicating with external devices, they work in different ways.

A parallel port sends and receives data eight bits at a time over 8 separate wires. This allows data to be transferred very quickly; however, the cable required is more bulky because of the number of individual wires it must contain. Parallel ports are typically used to connect a PC to a printer and are rarely used for much else. A serial port sends and receives data one bit at a time over one wire. While it takes eight times as long to transfer each byte of data this way, only a few wires are required. In fact, two-way (full duplex) communications is possible with only three separate wires - one to send, one to receive, and a common signal ground wire.

Bi-Directional Communications

The serial port on your PC is a full-duplex device meaning that it can send and receive data at the same time. In order to be able to do this, it uses separate lines for transmitting and receiving data. Some types of serial devices support only one-way

communications and therefore use only two wires in the cable - the transmit line and the signal ground.

Communicating by Bits

Once the start bit has been sent, the transmitter sends the actual data bits. There may either be 5, 6, 7, or 8 data bits, depending on the number you have selected. Both receiver and the transmitter must agree on the number of data bits, as well as the baud rate. Almost all devices transmit data using either 7 or 8 data bits.

Notice that when only 7 data bits are employed, you cannot send ASCII values greater than 127. Likewise, using 5 bits limits the highest possible value to 31. After the data has been transmitted, a stop bit is sent. A stop bit has a value of 1 - or a mark state - and it can be detected correctly even if the previous data bit also had a value of 1. This is accomplished by the stop bit's duration. Stop bits can be 1, 1.5, or 2 bit periods in length.

The Parity Bit

Besides the synchronization provided by the use of start and stop bits, an additional bit called a parity bit may optionally be transmitted along with the data. A parity bit affords a small amount of error checking, to help detect data corruption that might occur during transmission. You can choose either even parity, odd parity, mark parity, space parity or none at all. When even or odd parity is being used, the number of marks (logical 1 bits) in each data byte are counted, and a single bit is transmitted following the data bits to indicate whether the number of 1 bits just sent is even or odd.

For example, when even parity is chosen, the parity bit is transmitted with a value of 0 if the number of preceding marks is an even number. For the binary value of 0110 0011 the parity bit would be 0. If even parity were in effect and the binary number 1101 0110 were sent, then the parity bit would be 1. Odd parity is just the opposite, and the parity bit is 0 when the number of mark bits in the preceding word is an odd number. Parity error checking is very rudimentary. While it will tell you if there is a single bit error in the character, it doesn't show which bit was received in error. Also, if an even number of bits are in error then the parity bit would not reflect any error at all.

Mark parity means that the parity bit is always set to the mark signal condition and likewise space parity always sends the parity bit in the space signal condition. Since these two parity options serve no useful purpose whatsoever, they are almost never used.

RS-232C

RS-232 stands for Recommend Standard number 232 and C is the latest revision of the standard. The serial ports on most computers use a subset of the RS-232C standard. The full RS-232C standard specifies a 25-pin "D" connector of which 22 pins are used. Most of these pins are not needed for normal PC communications, and indeed, most new PCs are equipped with male D type connectors having only 9 pins.

Baud vs. Bits per Second

The baud unit is named after Jean Maurice Emile Baudot, who was an officer in the French Telegraph Service. He is credited with devising the first uniform-length 5-bit code for characters of the alphabet in the late 19th century. What baud really refers to is modulation rate or the number of times per second that a line changes state. This is not always the same as bits per second (BPS). If you connect two serial devices together using direct cables then baud and BPS are in fact the same. Thus, if you are running at 19200 BPS, then the line is also changing states 19200 times per second. But when considering modems, this isn't the case.

Because modems transfer signals over a telephone line, the baud rate is actually limited to a maximum of 2400 baud. This is a physical restriction of the lines provided by the phone company. The increased data throughput achieved with 9600 or higher baud modems is accomplished by using sophisticated phase modulation, and data compression techniques.

Synchronous and Asynchronous Communications

There are two basic types of serial communications, synchronous and asynchronous. With Synchronous communications, the two devices initially synchronize themselves to each other, and then continually send characters to stay in sync. Even when data is not really being sent, a constant flow of bits allows each device to know where the

other is at any given time. That is, each character that is sent is either actual data or an idle character. Synchronous communications allows faster data transfer rates than asynchronous methods, because additional bits to mark the beginning and end of each data byte are not required. The serial ports on IBM-style PCs are asynchronous devices and therefore only support asynchronous serial communications.

Asynchronous means "no synchronization", and thus does not require sending and receiving idle characters. However, the beginning and end of each byte of data must be identified by start and stop bits. The start bit indicates when the data byte is about to begin and the stop bit signals when it ends. The requirement to send these additional two bits cause asynchronous communications to be slightly slower than synchronous however it has the advantage that the processor does not have to deal with the additional idle characters.

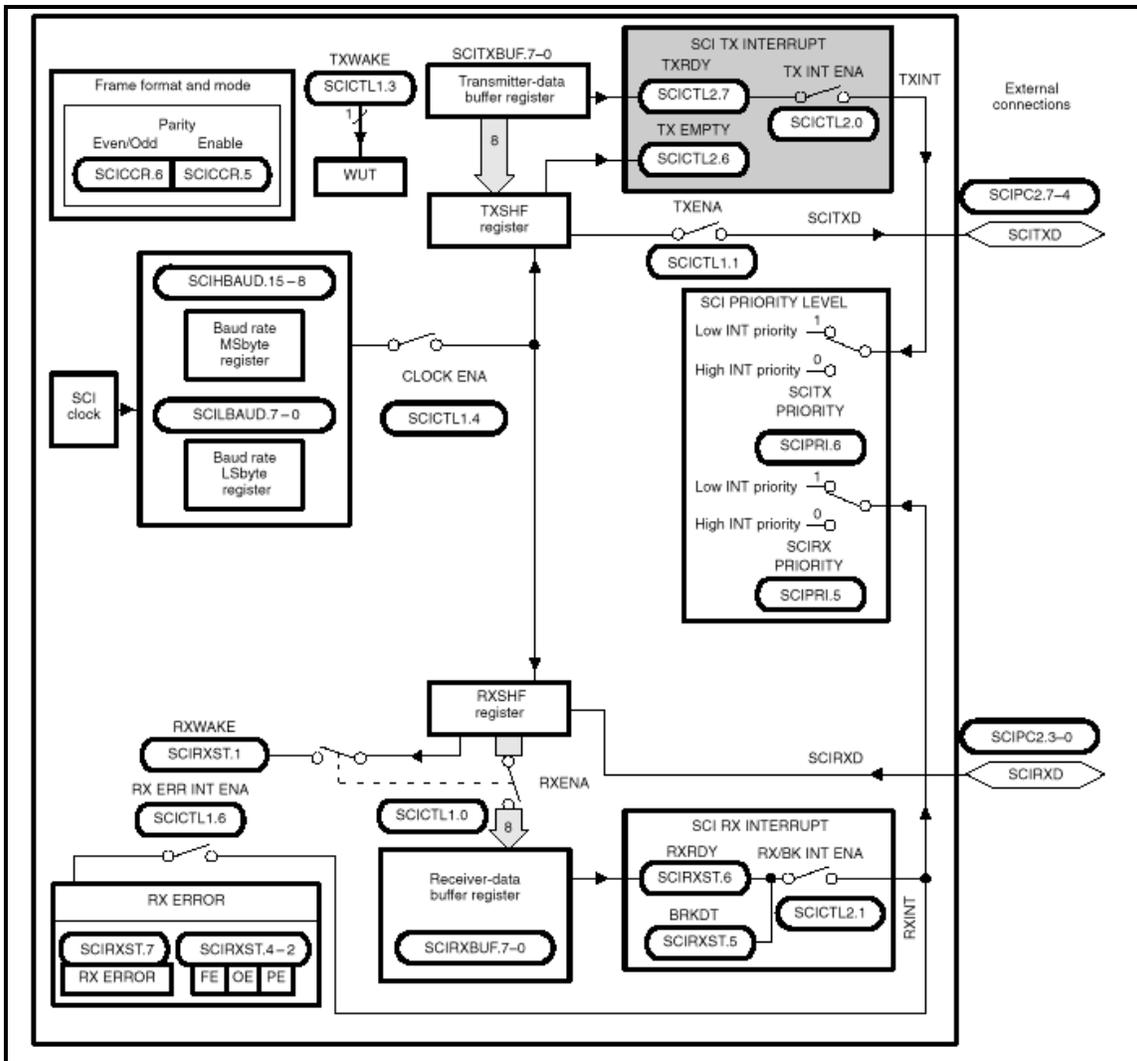
An asynchronous line that is idle is identified with a value of 1, (also called a mark state). By using this value to indicate that no data is currently being sent, the devices are able to distinguish between an idle state and a disconnected line. When a character is about to be transmitted, a start bit is sent. A start bit has a value of 0, (also called a space state). Thus, when the line switches from a value of 1 to a value of 0, the receiver is alerted that a data character is about to come down the line.

8.2 Serial Communication Interface (SCI)

Now we come back talking about the features related to serial communications on the DSP – TMS320F240:

The SCI supports two communication modes. Multiprocessor communication mode which allows data transfer between multiple processors. Universal Asynchronous Receiver/Transmitter (UART) communications mode, for interfacing with various peripherals. The receiver and transmitter each have their separate enable and interrupt control bits. The baud rate of the transfer can be selected from among 65000 different values.

The block diagram of the SCI module is shown in Figure 8.1 below.



Ref : Texas Instruments Literature # SPRU161A, March 1997

Figure 8.1: SCI Block Diagram

The key features of the module are:

- ❑ Two I/O pins
 - SCIRXD - SCI receive data input
 - SCITXD - SCI transmit data output
- ❑ 65000 programmable Baud rate. The range is from 19.07 bps to 625.0 kbps for a 10MHz system clock.
- ❑ Programmable data word length from one to eight bits.
- ❑ Programmable stops bits of either one or two bits in length.
- ❑ Internally generated serial clock.

- ❑ Error detection flags for Parity, Overrun, Framing and Break detect errors.
- ❑ Two wake-up modes for multiprocessor communication.
 - Idle-line wake-up
 - Address-bit wake-up
- ❑ Half duplex and full duplex operation
- ❑ Transmitter and receiver operation through separate interrupts (RXINT and TXINT) or through polling as follows-
 - Transmitter: TXRDY and TXEMPTY flags
 - Receiver: RXRDY, BRKDT and RXERROR flags

As per the figure, the major elements employed for full duplex data transfer are:

- ❑ The transmitter (TX) and its major registers.
 - SCITXBUF - Transmitter Data Buffer Register. It contains the data to be transmitted.
 - TXSHF - Transmitter Shift Register. It loads data from the SCITXBUF and shifts it onto the SCITXD pin, one bit at a time.
- ❑ The receiver (RX) and its major registers.
 - RXSHF - Receiver Shift Register. It shifts data in from the SCIRXD pin, one bit at a time.
 - SCIRXBUF - Receiver Data Buffer Register. It contains data to be read by the CPU. Data from a remote processor is loaded into RXSHF and then into SCIRXBUF and SCIRXEMU.
- ❑ The programmable baud generator.
- ❑ Data memory mapped control and status registers.

The following table summarizes the function of each register associated with the SCI module.

Address	Register	Name	Description
7050h	SCICCR	SCI Communication Control Register	Defines the character format, protocol and communications mode used by the SCI.

7051h	SCICTL1	SCI Control Register 1	Controls the RX/TX and receiver error interrupt enable, TXWAKE and SLEEP functions, internal clock enable, and the SCI software reset.
7052h	SCIHBAUD	SCI baud Register, high bits.	Stores the data (MSbyte) required to generate the bit rate.
7053h	SCILBAUD	SCI baud Register, low bits.	Stores the data (LSbyte) required to generate the bit rate.
7054h	SCICTL2	SCI Control Register 2	Contains the transmitter interrupt enable, the receiver-buffer/ break interrupt enable, the transmitter ready flag and the transmitter empty flag.
7055h	SCIRXST	SCI receiver status register	Contains seven receiver status flags.
7056h	SCRXEMU	SCI Emulation Data Buffer Register	Contains data received for screen updates principally used by the emulator.
7057h	SCIRXBUF	SCI Receiver Data Buffer Register	Contains the current data from the receiver shift register.
7059h	SCITXBUF	SCI Transmit Data Buffer Register	Stores the data bits to be transmitted by the SCITX.
705Eh	SCIPC2	SCI Port Control Register 2	Controls the SCIRXD and SCITXD pin functions.
705Fh	SCIPRI	SCI Priority Control Register	Contains the receiver and transmitter interrupt priority select bits and the emulator suspend enable bit.

Ref: Texas Instruments Literature # SPRU161A, March 1997

Table 8.1 : SCI Registers Summary

8.3 SCI Programmable Data Format

SCI receive and transmit data is in NRZ (nonreturn-to-zero) format. This format consists of:

- ❑ One start bit
- ❑ One to eight data bits
- ❑ One even/ odd parity bit (optional)
- ❑ One or two stop bits
- ❑ One extra bit to distinguish between address and data (for address-bit mode only)

This is shown in figure below.

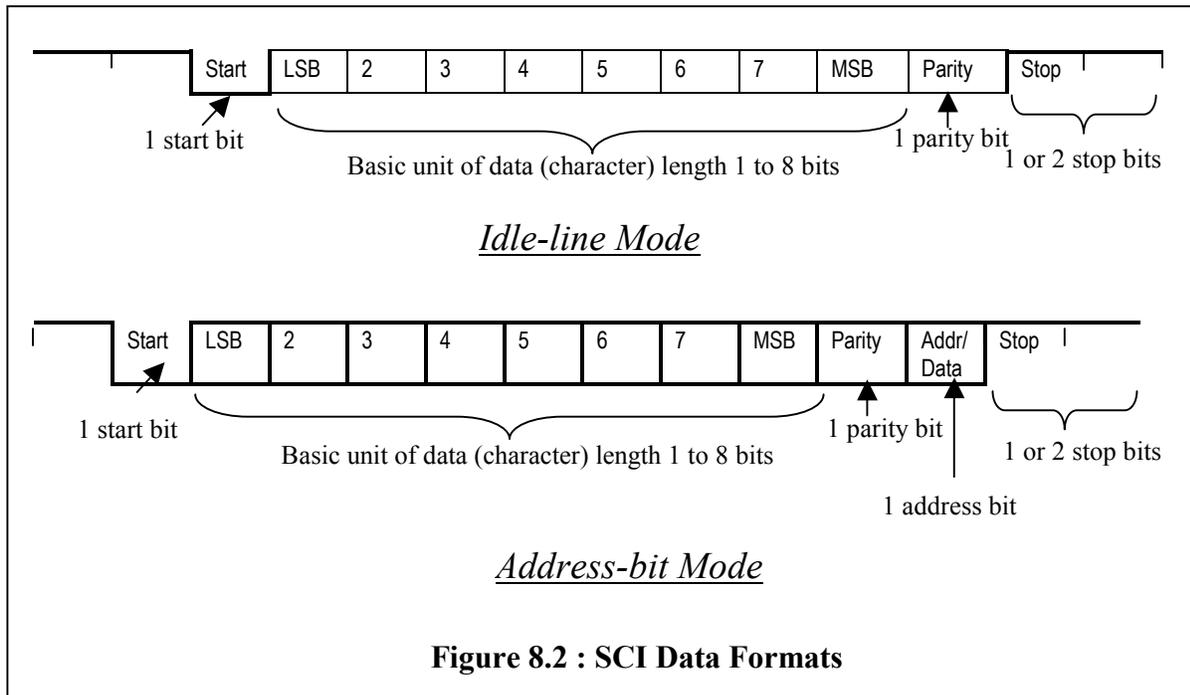


Figure 8.2 : SCI Data Formats

8.4 SCI Multiprocessor Communication

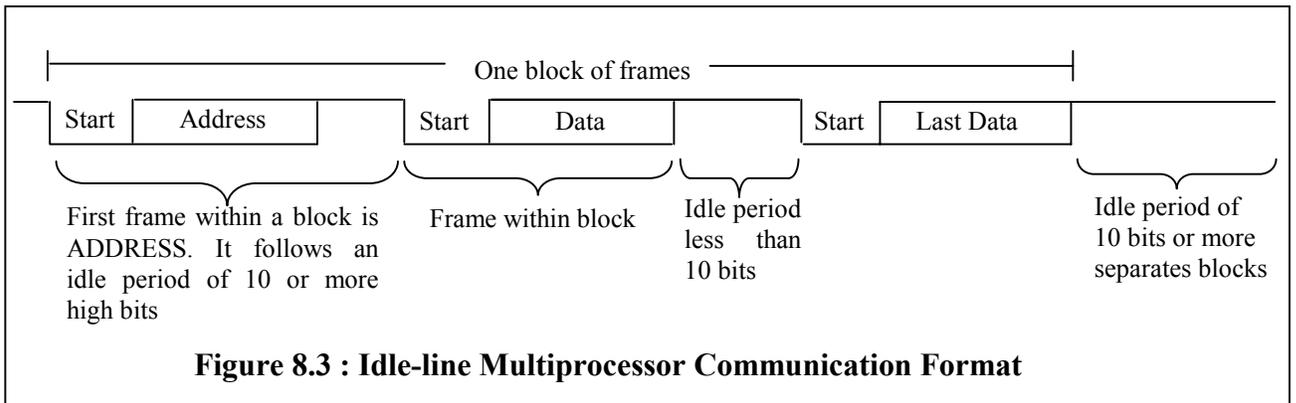
Multiprocessor allows data transfer between two processors on a serial link. At a time, there can be only one talker on the serial line, and several listeners. The first byte of a block of information sent by the talker contains the address. All the listeners read this byte. Only the listeners with the correct address are interrupted by the following data

bytes. The other listeners remain uninterrupted until the next address byte. All processors have a SLEEP bit which they set to 1, so that they are interrupted only when an address byte is detected. When the address of a processor matches with that sent on the serial link, the SLEEP bit for that processor must be cleared by software.

A processor recognizes an address byte according to the multiprocessor mode.

Idle-line Multiprocessor Mode

This mode a quiet space or idle time of 10 or more, high level bits after a frame indicate the beginning of a new block of information implying the first frame will contain an address byte. This mode does not contain the extra address bit and is therefore more efficient for blocks that contain more than 10 bytes of data. Figure 8.3 below shows this multiprocessor communication format.



The sequence of events in the idle-line mode is as follows-

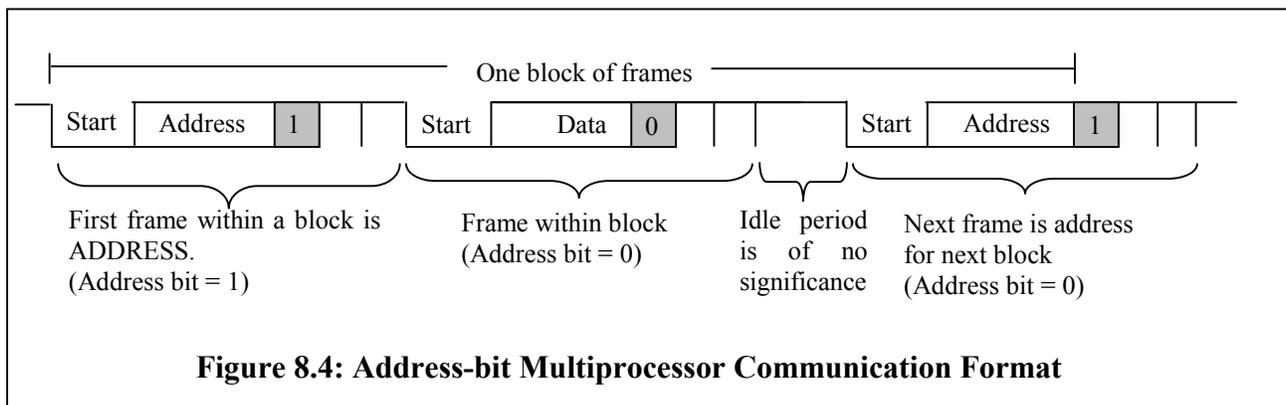
1. SCI wakes up after the receipt of the block-start signal.
2. The processor now recognizes the next SCI interrupt.
3. The service routine compares the received address with its own.
4. If the CPU is being addressed, the service routine clears the SLEEP bit and receives the rest of the data.
5. If the CPU is not being addressed, the SLEEP bit remains set. This lets the CPU continue to execute its main program without being interrupted by the SCI port until the detection of a new block start.

There are two ways of sending a block start signal:

- a) Deliberately leave an idle time of ten bits or more by delaying the time between transmission of the last data frame in a block and the address frame in the next block.
- b) The SCI port first sets the TXWAKE bit to 1 before writing to the SCITXBUF. This sends an idle time of exactly 11 bits.

Address-bit Multiprocessor Mode

In this mode, frames have an extra bit called the address bit that immediately follows the last data bit. The address is set to 1 for the first frame in every block and to 0 in all other frames. This mode is typically used for data frames of 11 bytes or less. Figure 8.4 below shows the address-bit multiprocessor communication format.



The sequence of events to send an address in this mode is as follows:

1. Set the TXWAKE bit to 1 and write the appropriate address value to SCITXBUF.
2. Once this address value is transferred to TXSHF and shifted out, the address bit is sent as a 1. This wakes the other processors on the serial link to read the address.
3. To transmit the non-address frames in the block, leave the TXWAKE bit set to 0.

8.5 SCI Port Interrupts

The SCI receiver and transmitter can be interrupt controlled. The interrupt requests and vectors for the receiver and transmitter are as follows-

- If the R/BK INT ENA bit is set, the receiver interrupt is asserted if when one of the following events occurs:

- The SCI receives a complete frame and transfers the data in the RXSHF to the SCIRXBUF. This action sets the RXRDY flag and initiates an interrupt.
 - When a break detect condition occurs, the BRKDT flag is set and an interrupt is initiated. A break detect condition occurs when the SCIRXD is low for 10 bit periods following a stop bit.
- If the TX INT ENA bit is set, the transmitter interrupt is asserted whenever the data in SCITXBUF is transferred to TXSHF, indicating that the CPU can write to TXBUF. This action sets the TXRDY flag and initiates an interrupt.

When the interrupts are not enabled, the condition flags still remain active reflecting the status of transmission and reception.

8.6 SCI Baud Rate Calculation

The baud rate for the different communication modes is determined in the following ways:

SCI Asynchronous Baud Rate for BRR = 1 to 65535

$$\text{SCI Asynchronous Baud} = \text{SYSCLK} / [(\text{BRR}+1) * 8]$$

$$\text{BRR} = (\text{SYSCLK} / \text{SCI Asynchronous Baud} * 8) - 1$$

SCI Asynchronous Baud Rate for BRR = 0

$$\text{SCI Asynchronous Baud} = \text{SYSCLK} / 16$$

Where, BRR = The 16-bit value in the baud-select registers.

8.7 Signals in Communications Modes

Figure 8.5 below, gives an example of receiver signal timing for address-bit mode with 6 bits per character.

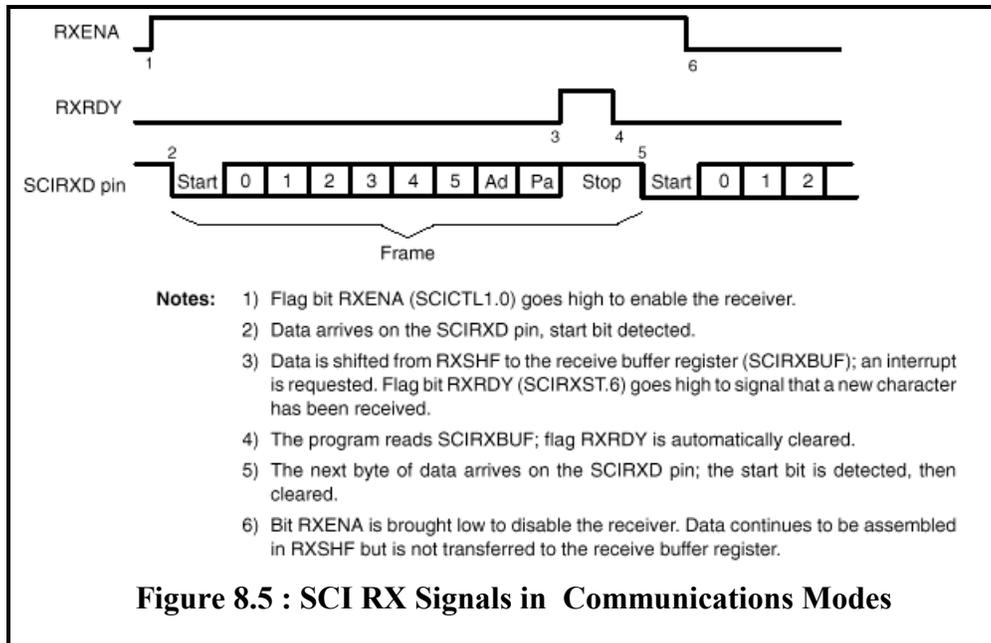
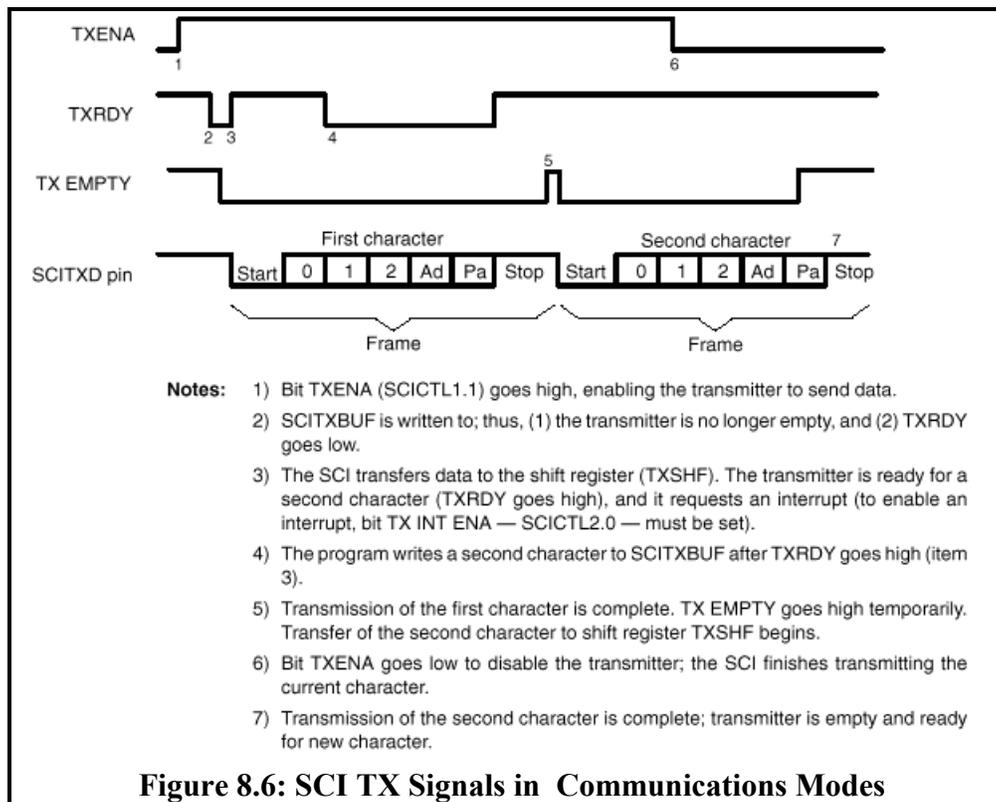


Figure 8.6 below, gives an example of transmitter signal timing for address-bit mode with 3 bits per character.



The SCICCR (described below) is employed for programming the data format.

SCI Communication Control Register (SCICCR) - Address 7050h

7	6	5	4	3	2	1	0
STOP BITS	EVEN/ODD PARITY	PARITY ENABLE	SCI ENA	ADDR/ID LE MODE	SCI CHAR2	SCI CHAR1	SCI CHAR0
RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0

Note: 0 = Always read as zeros, R = Read access, W = Write access, -0 = value after reset

Bit 7 STOP BITS. This bit specifies the number of stop bits to be transmitted. The receiver checks for only one stop bit.

0 = 1 stop bit

1 = 2 stop bits

Bit 6 PARITY. This bit selects odd or even parity.

0 = Odd parity

1 = Even parity

Bit 5 PARITY ENABLE. This bit enables or disables the parity function.

0 = Parity disabled. No parity is generated during transmission or expected during reception.

1 = Parity enabled.

Bit 4 SCI ENA. This bit enables or disables the SCI communication.

0 = Disable SCI communication.

1 = Enable SCI communication.

Bit 3 ADDR/IDLE MODE. This bit selects the multiprocessor mode.

0 = Idle-line mode protocol is selected.

1 = Address-bit mode protocol is selected.

Bit 2-0 SCI CHAR 2-0. These bits control the character length as shown below.

Characters less than 8 bits in length are filled with leading zeros in SCIRXBUF, but not so in SCITXBUF.

SCI CHAR 2-0 Bit Values			
SCI CHAR2	SCI CHAR1	SCI CHAR0	Character Length (Bits)
0	0	0	1
0	0	1	2

0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

8.8 DSP Program

Having studied the details of the SCI communication, let us now take a look at an example in order to understand how the SCI is programmed.

Example 1:

Write a program to initialize the SCI in idle-line mode with 8 character bits, 1 stop bit and odd parity. The baud rate of transmission and reception should be 19200 bps.

Solution:

Registers involved:

SCICCR = 0037h

Number of stop bits is 1. This is set by bit 7 = 0.

Odd parity is set by bit 6 = 0.

Parity is enabled by setting bit 5 = 1.

SCI communication is enabled by setting bit 4 = 1.

The IDLE MODE is selected by setting bit 3 = 0.

8 bits per character are selected by setting bits 2-0 to 111.

SCICTL1 = 0013h

Bit 7 Reserved.

Bit 6 = 0. Receive error interrupt is disabled.

Bit 5 = 0. SCI software reset initializes the SCI state machines and operating flags (SCICTL2 and SCIRXST) to the reset condition.

Bit 4 = 1. Enables SCI internal clock.
Bit 3 = 0. Transmit feature is not selected
Bit 2 = 0. SCI Sleep mode is disabled.
Bit 1 = 1. SCI transmitter enable.
Bit 0 = 1. SCI receiver enable.

SCICTL2 = 0002h

Bit 1 = 1. Receiver-buffer/break interrupt enabled.
Bit 0 = 0. Disable TXRDY interrupt.

SCIHBAUD = 0000h

SCILBAUD = 0040h

SCI 16-bit baud selection. SCIHBAUD (MSbyte) and SCILBAUD (LSbyte) concatenate to form a 16-bit baud value. The internally generated serial clock is determined by the SYSCLK and the two baud select registers. The SCI uses the 16-bit value of these registers to select one of 64K serial clock rates for the communication modes. The SCI baud rate for the different communication modes is determined in the following ways:

- For BRR = 1 to 65 535

SCI asynchronous baud = $\{\text{SYSCLK} / [(\text{BRR} + 1) * 8]\}$

- For BRR = 0

SCI asynchronous baud = $\text{SYSCLK} / 16$

Thus, in our problem,

$\text{BRR} = \{10\text{e}7 / (19200 * 8)\} - 1 = 0040\text{h}$

The code segment for the SCI initialization is given below. The rest of the program is self-explanatory with detailed comments.

```
SCI_INIT:  LDP #00E0h
           SPLK #0037h, SCICCR  ;1 stop bit,odd parity,8 char bits,
           ;async mode, idleline protocol
           SPLK #0013h, SCICTL1 ;Enable TX, RX, internal SCICLK,
           ;Disable RX ERR, SLEEP, TXWAKE
           SPLK #0002h, SCICTL2 ;Enable RX INT,disable TX INT
```

```

SPLK #0000h, SCIHBAUD
SPLK #0040h, SCILBAUD ;Baud Rate=19200 b/s (10 MHz SYSCLK)
SPLK #0022h, SCIPC2 ;Enable TXD & RXD pins
SPLK #0033h, SCICTL1 ;Relinquish SCI from Reset.

```

The complete program listing is as follows – This program is about serial communication between a DSP and a PC. The software running on the PC side and the relative settings are described in Laboratory 7.

```

;*****
; File Name: Ch8_e1.asm
; Description: This program uses the SCI module to implement a simple
; asynchronous communications routine. The SCI is initialized to operate
; in idleline mode with 8 character bits, 1 stop bit, and odd parity.
; The SCI Baud Rate Registers (BRR) are set to transmit and receive data
; at 19200 baud. The SCI generates an interrupt every time a character
; is loaded into the receive buffer (SCIRXBUF). The interrupt service
; routine(ISR) reads the receive buffer and determines if the carriage
; return button, <CR>,has been pressed. If so, the character string
; is compared with the value f password expected. If the password is
; correct, the character string "Right" is transmitted otherwise, the
; character string "Wrong" is transmitted.
;*****

        .include f240regs.h

LENGTH      .set 16h ; length of the data stream to be transmitted
LENGTH2     .set 8
PASWD       .set 4321h ; storage for 4 digit numeric password: 1234

        .bss DATA_OUT,LENGTH ;Location of LENGTH byte character
                                ;stream to be transmitted

        .bss GPR0,1 ;General purpose register.
        .bss RX_PASWD,1 ;Storage for received password
        .bss CHR_CNT,1 ;Counter for characters received

; Initialize Transmit Data for Interrupt Service Routine
        .data
TXDATA     .word 0052h ;Hex equivalent of ASCII character 'R'
           .word 0069h ;Hex equivalent of ASCII character 'i'

```

```

        .word 0067h    ;Hex equivalent of ASCII character 'g'
        .word 0068h    ;Hex equivalent of ASCII character 'h'
        .word 0074h    ;Hex equivalent of ASCII character 't'
        .word 000Ah;Hex equivalent of line feed
        .word 000Dh;Hex equivalent of carriage return
        .word 0000h;Hex equivalent of NULL

WRONG    .word 0057h    ;Hex equivalent of ASCII character 'W'
        .word 0072h    ;Hex equivalent of ASCII character 'r'
        .word 006fh    ;Hex equivalent of ASCII character 'o'
        .word 006eh    ;Hex equivalent of ASCII character 'n'
        .word 0067h    ;Hex equivalent of ASCII character 'g'
        .word 000Ah;Hex equivalent of line feed
        .word 000Dh;Hex equivalent of carriage return
        .word 0000h;Hex equivalent of NULL

;-----
; Vector address declarations
;-----

        .sect    ".vectors"

RSVECT B    START            ; Reset Vector
INT1      B    INT1_ISR    ; Interrupt Level 1
INT2       B    PHANTOM      ; Interrupt Level 2
INT3       B    PHANTOM      ; Interrupt Level 3
INT4       B    PHANTOM      ; Interrupt Level 4
INT5       B    PHANTOM      ; Interrupt Level 5
INT6       B    PHANTOM      ; Interrupt Level 6
RESERVED   B    PHANTOM      ; Reserved
SW_INT8    B    PHANTOM      ; User S/W Interrupt
SW_INT9    B    PHANTOM      ; User S/W Interrupt
SW_INT10   B    PHANTOM      ; User S/W Interrupt
SW_INT11   B    PHANTOM      ; User S/W Interrupt
SW_INT12   B    PHANTOM      ; User S/W Interrupt
SW_INT13   B    PHANTOM      ; User S/W Interrupt
SW_INT14   B    PHANTOM      ; User S/W Interrupt
SW_INT15   B    PHANTOM      ; User S/W Interrupt
SW_INT16   B    PHANTOM      ; User S/W Interrupt
TRAP       B    PHANTOM      ; Trap vector
NMINT      B    PHANTOM      ; Non-maskable Interrupt
EMU_TRAP   B    PHANTOM      ; Emulator Trap

```

```

SW_INT20    B    PHANTOM    ; User S/W Interrupt
SW_INT21    B    PHANTOM    ; User S/W Interrupt
SW_INT22    B    PHANTOM    ; User S/W Interrupt
SW_INT23    B    PHANTOM    ; User S/W Interrupt

;=====
; M A I N C O D E starts here
;=====

    .text
START: SETC INTM            ;Disable interrupts
      CLRC SXM              ;Clear Sign Extension Mode
      CLRC OVM              ;Reset Overflow Mode
      CLRC CNF              ;Config Block B0 to Data mem.
      LDP #00E0h
      SPLK #006Fh, WDCR     ;Disable Watchdog if VCCP=5V
      KICK_DOG              ;Reset Watchdog counter
      LDP #00E0h
      SPLK #00BBh, CKCR1    ;CLKIN(XTAL)=10MHz, CPUCLK=20MHz
      SPLK #00C3h, CKCR0    ;CLKMD=PLL Enable, SYSCLK=CPUCLK/2,
      SPLK #40C0h, SYSCR    ;CLKOUT=CPUCLK

      LDP #0000h
      SPLK #4h, GPR0
      OUT GPR0, WSGR        ;Set XMIF to run w/no wait states

      SPLK #0, RX_PASWD
      SPLK #0, CHR_CNT

;=====
; Initialize B2 RAM to zero's.
;=====

      LAR AR2, #B2_SADDR    ;AR2 B2 start address
      MAR *, AR2            ;Set ARP=AR2
      LACL #0               ;Set ACC = 0
      RPT #1fh              ;Set repeat cntr for 31+1 loops
      SACL *+               ;Write zeros to B2 RAM

;=====
; Initialize DATAOUT with data to be transmitted.
;=====

      LAR AR2, #B2_SADDR    ;Reset AR2 B2 start address
      LAR AR1, #DATA_OUT    ;AR1 DATAOUT start address
      RPT #(LENGTH - 1)    ;set repeat counter for LENGTH loops

```

```

        BLPD #TXDATA,*+      ;loads B2 with TXDATA
;=====
; INITIALIZATION OF INTERRUPT DRIVEN SCI ROUTINE
;=====
SCI_INIT:  LDP #00E0h
           SPLK #0037h, SCICCR   ;1 stop bit,odd parity,8 char bits,
           ;async mode, idleline protocol
           SPLK #0013h, SCICTL1  ;Enable TX, RX, internal SCICLK,
           ;Disable RX ERR, SLEEP, TXWAKE
           SPLK #0002h, SCICTL2  ;Enable RX INT,disable TX INT
           SPLK #0000h, SCIHBAUD
           SPLK #0040h, SCILBAUD  ;Baud Rate=19200 b/s (10 MHz SYSCLK)
           SPLK #0022h, SCIPC2   ;Enable TXD & RXD pins
           SPLK #0033h, SCICTL1  ;Relinquish SCI from Reset.
           LAR AR0, #SCITXBUF   ;Load AR0 with SCI_TX_BUF address
           LAR AR1, #SCIRXBUF   ;Load AR1 with SCI_RX_BUF address
           LAR AR2, #B2_SADDR   ;Load AR2 with TX data start address

           LDP #0
           LACC IFR             ;Load ACC with Interrupt flags
           SACL IFR             ;Clear all pending interrupt flags
           SPLK #0001h,IMR      ;Unmask interrupt level INT1
           CLRC INTM            ;Enable interrupts

WAIT:     B      WAIT

;=====
; I S R  INT1_ISR
; Description: The INT1_ISR first determines if the SCI RXINT caused
; the interrupt. If so, the SCI Specific ISR reads the character in the
; RX buffer. If the character received corresponds to a carriage return,
; <CR>, the character string received is compared with the expected
; value of password. If the value matches, "Right" is transmitted else
; "Wrong" is transmitted. If the character received does NOT correspond
; to a carriage return, <CR>, then the ISR returns to the main program
; without transmitting a character string. If the SCI RXINT did not
; cause an interrupt, then the value '0x0bad' is stored in the
; accumulator and program gets caught in the BAD_INT endless loop.
;=====

```

```

INT1_ISR: LDP #00E0h
          LACL SYSIVR      ;Load peripheral INT vector address
          SUB #0006h      ;Subtract RXINT offset from above
          BCND SCI_ISR,EQ  ;verify RXINT initiated interrupt
          B BAD_INT       ;Else, bad interrupt occurred
SCI_ISR:  LDP #0
          MAR *,AR1       ;Set ARP=AR1
          LACC *          ;Load ACC w/RX buffer character
          SUB #000Dh      ;Check if <CR> has been pressed:
          BCND XMIT_CHAR,EQ ;YES? Transmit data.
          LACC *, AR2
          SUB #30h        ;Extract the actual value from the
                        ; ASCII equivalent of the number.
          RPT CHR_CNT
          SFL
          SFR
          OR  RX_PASWD
          SACL RX_PASWD
          LACC CHR_CNT
          ADD #4
          SACL CHR_CNT
          B ISR_END       ;NO? Return from INT1_ISR.

XMIT_CHAR: SPLK #0, CHR_CNT
           LAR AR2, #B2_SADDR
           LACC RX_PASWD
           SUB #PASWD
           SPLK #0, RX_PASWD ;Check if received no. = password
           BCND XMIT_CHAR1, EQ
           LAR AR2, #(B2_SADDR + LENGTH2) ; if not point to starting
                        ; address of the string "Wrong"

XMIT_CHAR1: LACC *+,AR0 ;Load char to be xmitted into ACC
            BCND ISR_END,EQ ;Check for Null Character
                        ;YES? Return from INT1_ISR.
            SACL *,AR2 ;NO? Load char into xmit buffer.

XMIT_RDY:  LDP #00E0h
           BIT SCICTL2,BIT7 ;Test TXRDY bit
           BCND XMIT_RDY,NTC ;If TXRDY=0,then repeat loop
           B XMIT_CHAR1 ;else xmit next character
ISR_END:   LAR AR2, #B2_SADDR ;Reload AR2 w/ TX data start address

```

```

        LAR AR1, #SCIRXBUF ;Reload AR1 with SCI_RX_BUF address      LDP
#0

        CLRC INTM          ;Clear INT Mask flag
        RET                ;Return from INT1_ISR

BAD_INT:  LACC #0BADh      ;Load ACC with "bad"
          B BAD_INT        ;Repeat loop

;=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
;=====
PHANTOM: B PHANTOM

```

Laboratory Experiment 7

Objectives

To use serial communication using the SCI module of the DSP and the RS232 interface of the EVB to establish communication with the PC COM port.

Equipment

Hardware :

- PC Specifications -
 - ❑ '386 or higher IBM PC/AT
 - ❑ 1.44Mb 3.5-inch floppy drive
 - ❑ 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
 - ❑ Minimum 4Mb memory
 - ❑ Color VGA Monitor
- TMS320C24x Evaluation Board
- XDS510PP Emulator

- +5V power supply.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable

Software :

- MS Windows 3.1 or Windows 95
- ASCII Editor
- TMS320C2xx Assembler
- TMS320C2xx C Source Debugger

Discussion

RS232 Interface

The 'C24x evaluation board has an RS-232 compatible DB-9 serial port for asynchronous communication. The DB-9 serial port (P6) interfaces to the SCI peripheral on the 'F240 device through an RS-232 transceiver. Five RS-232 signals can be used to implement various communications protocols with software and hardware handshaking on the 'C24x evaluation board. These signals are:

- _ Receive data (RX)
- _ Transmit data (TX)
- _ Clear to send (CTS)
- _ Request to send (RTS)
- _ Data terminal ready (DTR)

An RS-232 cable is required to connect the serial port on the evaluation board to a host. The pin-out for the evaluation board and host serial ports is provided in Table below. The cable labeled RS232 Cable conforms to these specifications.

Function	Evaluation Board Serial Port			Host Serial Port		
	Pin (DB-9)	Signal		Signal	Pin (DB-9)	Pin (DB-25)
Carrier detect (not used)	1	NC		DCD	1	8
Transmit data to host	2	SCITX/IO	—	RX	2	3
Transmit data to evaluation board	3	SCIRX/IO	—	TX	3	2
Reset evaluation board via host†	4	HOSTRESET	—	DTR	4	20
Signal ground	5	GND	—	GND	5	7
Data set ready (not used)	6	NC		DSR	6	6
Request to send‡	7	$\overline{\text{BIO}}/\text{IOPC3}$	—	RTS	7	4
Clear to send	8	XF/IOPC2	—	CTS	8	5
Ring indicator (not used)	9	NC		RI	9	22

Note: NC = No connection
† This line should only be connected if you are implementing the host reset function.
‡ This line should only be connected if you are implementing hardware handshaking.

Ref : Texas Instruments Literature # SPRU248A, August 1997

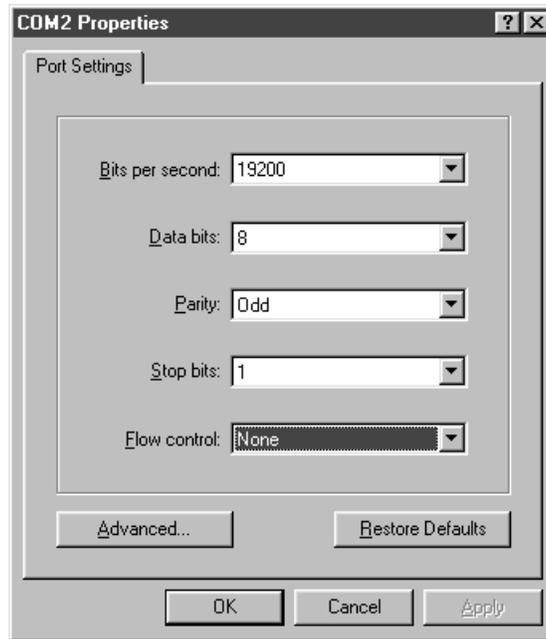
In order to test the program example discussed in the previous section, we employ the Windows HyperTerminal. This is a program that allows asynchronous communication with the PC. The protocol can be specified in this software.

Procedure:

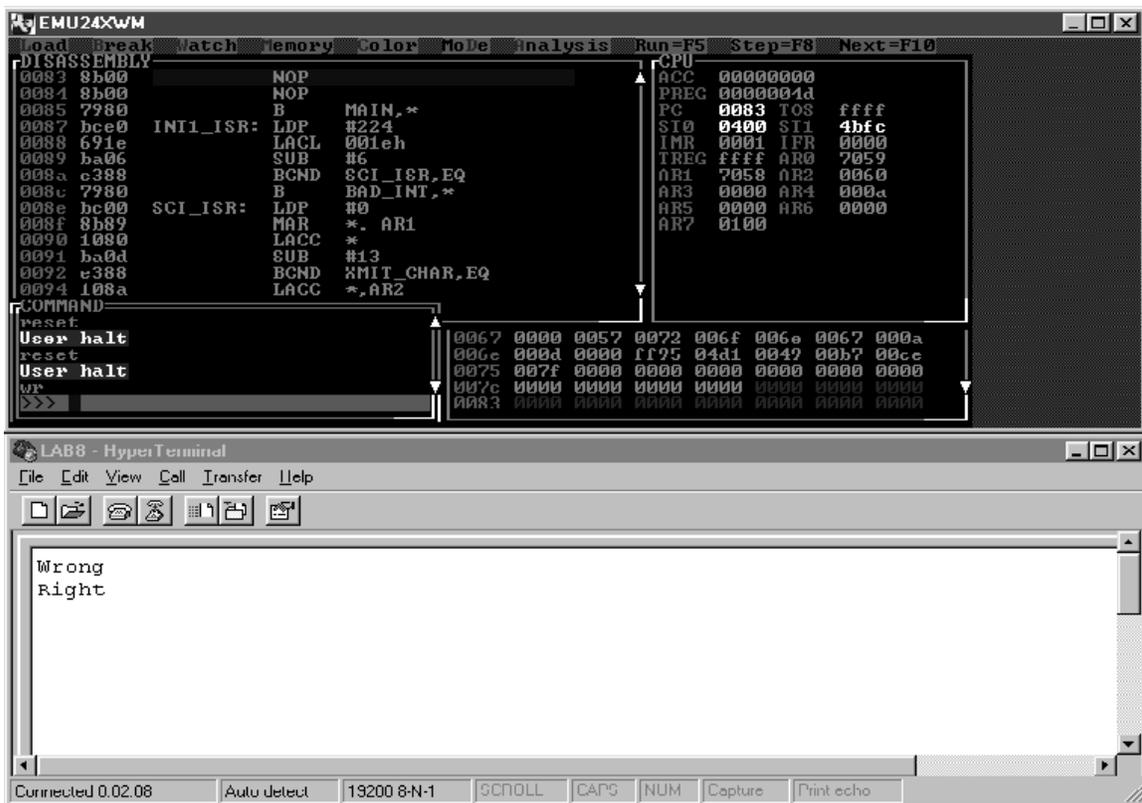
1. Turn off PC and EVB. Connect the cable labeled "RS232 Cable" such that the 9-pin connector end is connected to DB-9 on EVB, and the 25-pin connector end is connected to the COM2 port on the PC. Now power on the PC and EVB.
2. Double Click the icon "Short Cut to Hyperterm.exe" on the desktop. The following menu pops up. Enter the name of the session: e.g. LAB7. Select the port COM2, since this is the port we are going to use for communication.



3. Another menu pops up asking for the configuration details. Set up the port as shown in the figure below:



- Once the Hyperterm settings are done, start the C-source debugger and run the program. Now enter a password 1234, the terminal responds with a "Right". Try another password e.g. 5476, the terminal responds with a "Wrong". This is clearly shown in figure below:



Laboratory Assignments

1. Add comments for all lines in the interrupt service routine of the example program Ch8_e1.asm where there are no comment lines existing.
2. Write a program to enhance the password checking mechanism shown in the example – add features to simulate a real password checking interface:
 - Display a password input prompt on the Hyperterm screen at the beginning;
 - Display an asterisk “*” for each digit of password input;
 - If wrong, give prompt to reenter;
 - If wrong for 3 times, display some fail information;
 - If correct, display some information indicating the success.

Hint: use short words for prompts since the memory space in B2 is only 1Fh.

The ASCII code table is attached below:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

CHAPTER 9

CREATING A SINE MODULATED PWM SIGNAL

9.1 Overview

This chapter introduces one method to generate sine modulated PWM signals. This application generates an asymmetrical pulse width modulated (PWM) signal with a varying duty cycle. The period of the PWM signal is 0.05ms, which is equivalent to a 20kHz signal. The duty cycle is modulated with a sine function that can be varied in frequency. The implementation of the sine wave modulation is through a look-up table. This application is implemented using C2xx Assembly code. The algorithm described in this application report was implemented using the TI TMS320F240 EVM.

This application uses the Event Manager Module and General Purpose Timer 1 of the DSP. The frequency of the sinusoidal function modulated by the PWM signal is determined by variable declaration in the program and therefore no input signal is needed. The output PWM signal goes from pin T1PWM / T1CMP on P1 connector.

9.2 Methodology

The generation of the sine wave is performed using a look up table. To be able to control the frequency of the modulation with some accuracy, a method based on the modulo mathematical operation is used (i.e. any overflow is disregarded and only the remainder is kept).

In this application a 16-bit counter is used to determine the location of the next value. A step value is added to the counter every time a new value from the sine table is to be loaded. By changing the value of the step, one can accurately control the frequency of the sine wave.

Although a 16-bit counter is used, the upper byte determines the location of the next sine value to be used; thus, by changing how quickly values overflow from the lower byte (i.e., manipulating the step value), the frequency of the sine wave can be changed. The modulo mathematical operation is used when there is overflow in the accumulator from the lower word to the upper word. When an overflow occurs, only the remainder (lower word) is stored.

For example, the counter is set to 0000h and the step value is set to 40h. Every time a value is to be looked up in the table, the value 40h is added to the counter; however, since the

upper byte is used as the pointer on the look up table, the first, second, and third values will point to the same location. In the fourth step, which results in an overflow into the upper byte, the value that is loaded will change. Since the upper byte is used as the pointer, the look-up table has 256 values, which is equivalent to the number of possibilities for an 8-bit number: 0 to 255. Additionally, since the upper word of the accumulator is disregarded, the pointer for the sine look up table does not need to be reset.

Table 1. Look-Up Table Example 1

Step	Accumulator	Counter	Pointer	Step Value = 40h
0	0000 0000h	0000h	00h	1 st value of sine table
1	0000 0040h	0040h	00h	
2	0000 0080h	0080h	00h	
3	0000 00C0h	00C0h	00h	
4	0000 0100h	0100h	01h	2 nd value of sine table
....				
....				
....				
n	0000 FFC0h	FFC0h	FFh	256 th value of sine table
n+1	0001 0000h	0000h	00h	1 st value of sine table
n+2	0000 0040h	0040h	00h	

The step size controls the frequency that is output; as a result, the larger the step, the quicker the overflow into the upper byte, and the faster the pointer traverses through the sine look-up table.

Table 2. Look-Up Table Example 2

Step	Counter	Pointer	Step Value = C0h
0	0000h	00h	1 st value of sine table
1	00C0h	00h	
2	0180h	01h	2 nd value of sine table
3	0240h	02h	3 rd value of sine table
4	0300h	03h	4 th value of sine table

Although the step size indicates how quickly the pointer moves through the look up table, the step size does not provide much information about the approximate frequency that the sine wave will be modulating the PWM signal. To determine the frequency of the sine wave, determine how often the value in the compare register will be modified.

In this application, the routine to load a new value in the compare register is accessed every time that the timer value matches the value in the period register. Consequently, the routine will be accessed at the same frequency as the PWM signal (20kHz). Because the compare register will be updated each time that the period register and the timer values are equal, the routine that modifies the compare register will be implemented as an interrupt service routine. As a result, the proper registers, EVIMRA and the core IMR need to unmask the proper interrupt levels so that the compare register can be updated.

The frequency that the sine wave will be modulated at can be calculated from the following formula

$$f(step) = \frac{step}{T_s \times 2^n}$$

where:

$f(step)$ = desired frequency

T_s = the time period between each update (in this case, the PWM period)

n = the number of bits in the counter register ($n = 16$ here)

$step$ = the step size used

The frequency that the PWM signal will be modulated is proportional to the step size and inversely proportional to the size of the counter register and the period at which the routine is accessed. Thus, to increase the resolution that one can increment or decrement the frequency of the PWM modulation, one needs to have a larger counting register or access the routine at a slower frequency by increasing the period.

Since this program is interrupt driven, once the registers have been set for the PWM signal, the program can be ended with an unconditional branch and the output will continue because of the interrupt structure. The output will stop when the user halts the program or the software masks the corresponding interrupt levels.

Adding the following watch in the debugger environment

```
wa *FREQSTEP, , u
```

will allow one to modify the step size to change the frequency of modulation.

The complete code is given below -

```

;*****
; File Name: Ch9_e1.asm
; Originator: Digital Control systems Apps group - Houston
; Target System: 'C240 Evaluation Board
;
; Description: Pulse Width Modulator - Sets up the registers
; for an asymmetric PWM output. The output is a
; square wave with a sine wave modulated duty cycle.
; PWM Period is 0.05ms => 20kHz
;
; Entering the command
; wa *FREQSTEP,,u
; Allows one to change the step size to change
; the frequency in the debugger environment
;
; Last Updated: 20 June 1997
;
;*****
        .include f240regs.h
;-----
; Vector address declarations
;-----
        .sect ".vectors"

RSVECT B START    ; Reset Vector
INT1   B PHANTOM  ; Interrupt Level 1
INT2   B SINE     ; Interrupt Level 2
INT3   B PHANTOM  ; Interrupt Level 3
INT4   B PHANTOM  ; Interrupt Level 4
INT5   B PHANTOM  ; Interrupt Level 5
INT6   B PHANTOM  ; Interrupt Level 6
RESERVED B PHANTOM ; Reserved
SW_INT8 B PHANTOM ; User S/W Interrupt
SW_INT9 B PHANTOM ; User S/W Interrupt
SW_INT10 B PHANTOM ; User S/W Interrupt
SW_INT11 B PHANTOM ; User S/W Interrupt
SW_INT12 B PHANTOM ; User S/W Interrupt
SW_INT13 B PHANTOM ; User S/W Interrupt
SW_INT14 B PHANTOM ; User S/W Interrupt
SW_INT15 B PHANTOM ; User S/W Interrupt
SW_INT16 B PHANTOM ; User S/W Interrupt

```

```

TRAP    B PHANTOM ; Trap vector
NMINT   B PHANTOM ; Non-maskable Interrupt
EMU_TRAP B PHANTOM ; Emulator Trap
SW_INT20 B PHANTOM ; User S/W Interrupt
SW_INT21 B PHANTOM ; User S/W Interrupt
SW_INT22 B PHANTOM ; User S/W Interrupt
SW_INT23 B PHANTOM ; User S/W Interrupt

;=====
; M A I N C O D E - starts here
;=====

    .text
    NOP
START: SETC INTM    ;Disable interrupts
        SPLK #0002h,IMR ;Mask all core interrupts
        ; except INT2

        LACC IFR    ;Read Interrupt flags
        SACL IFR    ;Clear all interrupt flags
        CLRC SXM    ;Clear Sign Extension Mode
        CLRC OVM    ;Reset Overflow Mode
        CLRC CNF    ;Config Block B0 to Data mem

;-----
; Set up PLL Module
;-----

        LDP #00E0h

;The following line is necessary if a previous program set the PLL
;to a different ;setting than the settings which the application
;uses. By disabling the PLL, the CKCR1 register can be modified so
;that the PLL can run at the new settings when it is re-enabled.
        SPLK #000000001000001b,CKCR0 ;CLKMD=PLL Disable
        ;SYSCLK=CPUCLK/2
;        5432109876543210

        SPLK #0000000010111011b,CKCR1
        ;CLKIN(OSC)=10MHz,CPUCLK=20MHz
        ;CKCR1 - Clock Control Register 1
;        5432109876543210

```

```

SPLK #0000000011000001b,CKCRO ;CLKMD=PLL Enable
;
5432109876543210

SPLK #0100000011000000b,SYSCR ;CLKOUT=CPUCLK
SPLK #006Fh, WDCR ;Disable WD if VCCP=5V
; (JP5 in pos. 2-3)
KICK_DOG ;Reset Watchdog

;-----
; Set up Digital I/O Port
;-----

LDP #225 ;DP=225, Data Page to Configure OCRA
;
5432109876543210
SPLK #0011100000000000b,OCRA

;*****
;- Event Manager Module Reset
;*****

LDP #232 ;DP=232 Data Page for the Event Manager
SPLK #0000h,GPTCON ;Clear General Purpose Timer Control
SPLK #0000h,T1CON ;Clear GP Timer 1 Control
SPLK #0000h,T2CON ;Clear GP Timer 2 Control
SPLK #0000h,T3CON ;Clear GP Timer 3 Control
SPLK #0000h,COMCON ;Clear Compare Control
SPLK #0000h,ACTR

;Clear Full Compare Action Control Register
SPLK #0000h,SACTR

;Clear Simple Compare Action Control Register
SPLK #0000h,DBTCON

;Clear Dead-Band Timer Control Register
SPLK #0000h,CAPCON ;Clear Capture Control
SPLK #0FFFFh,EVIFRA;Clear Interrupt Flag Register A
SPLK #0FFFFh,EVIFRB;Clear Interrupt Flag Register B
SPLK #0FFFFh,EVIFRC;Clear Interrupt Flag Register C
SPLK #0000h,EVIMRA ;Clear Event Manager Mask Register A
SPLK #0000h,EVIMRB ;Clear Event Manager Mask Register B
SPLK #0000h,EVIMRC ;Clear Event Manager Mask Register C

```

```

;*****
;- End of RESET section for silicon revision 1.1 *
;*****

;-----
; Set up Event Manager Module
;-----
T1COMPARE .set 0 ; T1Compare Initialized to 0
T1PERIOD .set 1000 ; T1Period Initialized to
;1000 = 20kHz value

.text
LDP #232 ;DP=232, Data Page for
;Event Manager Addresses
SPLK #T1COMPARE,T1CMPR ;T1CMPR = 0
;
; 2109876543210
SPLK #0000001010101b,GPTCON
SPLK #T1PERIOD,T1PR ; T1PR = 1000
SPLK #0000h,T1CNT ; Initialize Timer 1
SPLK #0000h,T2CNT ; Initialize Timer 2
SPLK #0000h,T3CNT ; Initialize Timer 3
;
; 5432109876543210
SPLK #0001000000000110b,T1CON
SPLK #000000000000000b,T2CON ;Not used
SPLK #000000000000000b,T3CON ;Not Used

SBIT1 T1CON,B6_MSK ;Sets Bit 6 of T1CON
SPLK #0FFFFh,EVIFRA ;Clear all pending interrupts
SPLK #0080h,EVIMRA ;Enable Timer 1 Period Interrupt

;-----
; Generate Sine Wave Modulated PWM
;-----
.bss TABLE,1 ;Keeps address of the pointer in
;the SINE Table
.bss TOPTABLE,1 ;Keeps the reset value for the
;pointer
.bss COMPARET1,1 ;A register to do calculations since the
;T1CMPR register is double buffered

```

```

        .bss FREQSTEP,1 ;Frequency modulation of the sine wave
        .bss MODREG,1 ;Rolling Modulo Register
        .bss SINEVAL,1 ;Value from look up table

NORMAL .set 500
        .text
        LDP #0 ;DP = 0
        SPLK #0000h, TABLE ;Initialize Pointer to Top
        SPLK #STABLE, TOPTABLE ;Initialize TOPTABLE to
            ; address of sine table
        SPLK #4, FREQSTEP ;Set the step size to 4
        SPLK #0000h, MODREG ;Initialize the 16 bit
            ; counter register
        CLRC INTM ;Enable interrupts
END     B END ;End of Program

;-----
; Generate PWM Sine Wave ISR
;-----

SINE   LDP #0 ; DP = 0
        LACC MODREG ; ACC = 16 bit Counter Register
        ADD FREQSTEP ; ACC = Counter + Step
        SACL MODREG ; Counter assigned new value
        LACC MODREG, 8 ; ACC = Counter shifted to left
            ; by 8
        SACH TABLE ; TABLE = upper byte of
            ; counter = pointer
        LACC TABLE ; ACC = TABLE = Pointer
        ADD TOPTABLE ; Offset Addr from top of table
        TBLR SINEVAL ; Read sine value and store to
            ; SINEVAL

;Normalization of the Sine value to prevent the compare value from
; being negative
        LT SINEVAL ; TREG = SINEVAL (Q15)
        MPY #NORMAL ; PREG = TREG * NORMAL (Q30)
            ; NORMAL = T/2
        PAC ; ACC = PREG (Q30)
        SACH COMPARET1, 1 ; COMPARET1 = PREG (Q15)
        LACC COMPARET1 ; ACC = COMPARET1

```

```

ADD #NORMAL      ; ACC = COMPARET1 + NORMAL
LDP #232         ; DP = 232
SACL T1CMPR      ; T1CMPR = ACC = Normalize Sine
                  ; Value

;Clear the interrupt flags of the Event Manager Module
LACC EVIFRA      ; ACC = EVIFRA
SACL EVIFRA      ; EVIFRA = ACC; resets the
                  ; interrupt flag
CLRC INTM        ; Enable core interrupts
RET              ; Return to end of program

;-----
; Sine look-up table
; No. Entries : 256
; Angle Range : 360 deg
; Number format : Q15 with range -1 < N < +1
;-----
;SINVAL ; Index Angle Sin(Angle)
STABLE .word 0 ; 0 0 0.0000
        .word 804 ; 1 1.41 0.0245
        .word 1608 ; 2 2.81 0.0491
        .word 2410 ; 3 4.22 0.0736
        .word 3212 ; 4 5.63 0.0980
        .word 4011 ; 5 7.03 0.1224
        .word 4808 ; 6 8.44 0.1467
        .word 5602 ; 7 9.84 0.1710
        .word 6393 ; 8 11.25 0.1951
        .word 7179 ; 9 12.66 0.2191
        .word 7962 ; 10 14.06 0.2430
        .word 8739 ; 11 15.47 0.2667
        .word 9512 ; 12 16.88 0.2903
        .word 10278 ; 13 18.28 0.3137
        .word 11039 ; 14 19.69 0.3369
        .word 11793 ; 15 21.09 0.3599
        .word 12539 ; 16 22.50 0.3827
        .word 13279 ; 17 23.91 0.4052
        .word 14010 ; 18 25.31 0.4276
        .word 14732 ; 19 26.72 0.4496
        .word 15446 ; 20 28.13 0.4714
        .word 16151 ; 21 29.53 0.4929

```

.word 16846 ; 22 30.94 0.5141
.word 17530 ; 23 32.34 0.5350
.word 18204 ; 24 33.75 0.5556
.word 18868 ; 25 35.16 0.5758
.word 19519 ; 26 36.56 0.5957
.word 20159 ; 27 37.97 0.6152
.word 20787 ; 28 39.38 0.6344
.word 21403 ; 29 40.78 0.6532
.word 22005 ; 30 42.19 0.6716
.word 22594 ; 31 43.59 0.6895
.word 23170 ; 32 45.00 0.7071
.word 23731 ; 33 46.41 0.7242
.word 24279 ; 34 47.81 0.7410
.word 24811 ; 35 49.22 0.7572
.word 25329 ; 36 50.63 0.7730
.word 25832 ; 37 52.03 0.7883
.word 26319 ; 38 53.44 0.8032
.word 26790 ; 39 54.84 0.8176
.word 27245 ; 40 56.25 0.8315
.word 27683 ; 41 57.66 0.8449
.word 28105 ; 42 59.06 0.8577
.word 28510 ; 43 60.47 0.8701
.word 28898 ; 44 61.88 0.8819
.word 29268 ; 45 63.28 0.8932
.word 29621 ; 46 64.69 0.9040
.word 29956 ; 47 66.09 0.9142
.word 30273 ; 48 67.50 0.9239
.word 30571 ; 49 68.91 0.9330
.word 30852 ; 50 70.31 0.9415
.word 31113 ; 51 71.72 0.9495
.word 31356 ; 52 73.13 0.9569
.word 31580 ; 53 74.53 0.9638
.word 31785 ; 54 75.94 0.9700
.word 31971 ; 55 77.34 0.9757
.word 32137 ; 56 78.75 0.9808
.word 32285 ; 57 80.16 0.9853
.word 32412 ; 58 81.56 0.9892
.word 32521 ; 59 82.97 0.9925
.word 32609 ; 60 84.38 0.9952
.word 32678 ; 61 85.78 0.9973
.word 32728 ; 62 87.19 0.9988

.word 32757 ; 63 88.59 0.9997
.word 32767 ; 64 90.00 1.0000
.word 32757 ; 65 91.41 0.9997
.word 32728 ; 66 92.81 0.9988
.word 32678 ; 67 94.22 0.9973
.word 32609 ; 68 95.63 0.9952
.word 32521 ; 69 97.03 0.9925
.word 32412 ; 70 98.44 0.9892
.word 32285 ; 71 99.84 0.9853
.word 32137 ; 72 101.25 0.9808
.word 31971 ; 73 102.66 0.9757
.word 31785 ; 74 104.06 0.9700
.word 31580 ; 75 105.47 0.9638
.word 31356 ; 76 106.88 0.9569
.word 31113 ; 77 108.28 0.9495
.word 30852 ; 78 109.69 0.9415
.word 30571 ; 79 111.09 0.9330
.word 30273 ; 80 112.50 0.9239
.word 29956 ; 81 113.91 0.9142
.word 29621 ; 82 115.31 0.9040
.word 29268 ; 83 116.72 0.8932
.word 28898 ; 84 118.13 0.8819
.word 28510 ; 85 119.53 0.8701
.word 28105 ; 86 120.94 0.8577
.word 27683 ; 87 122.34 0.8449
.word 27245 ; 88 123.75 0.8315
.word 26790 ; 89 125.16 0.8176
.word 26319 ; 90 126.56 0.8032
.word 25832 ; 91 127.97 0.7883
.word 25329 ; 92 129.38 0.7730
.word 24811 ; 93 130.78 0.7572
.word 24279 ; 94 132.19 0.7410
.word 23731 ; 95 133.59 0.7242
.word 23170 ; 96 135.00 0.7071
.word 22594 ; 97 136.41 0.6895
.word 22005 ; 98 137.81 0.6716
.word 21403 ; 99 139.22 0.6532
.word 20787 ; 100 140.63 0.6344
.word 20159 ; 101 142.03 0.6152
.word 19519 ; 102 143.44 0.5957
.word 18868 ; 103 144.84 0.5758

.word 18204 ; 104 146.25 0.5556
.word 17530 ; 105 147.66 0.5350
.word 16846 ; 106 149.06 0.5141
.word 16151 ; 107 150.47 0.4929
.word 15446 ; 108 151.88 0.4714
.word 14732 ; 109 153.28 0.4496
.word 14010 ; 110 154.69 0.4276
.word 13279 ; 111 156.09 0.4052
.word 12539 ; 112 157.50 0.3827
.word 11793 ; 113 158.91 0.3599
.word 11039 ; 114 160.31 0.3369
.word 10278 ; 115 161.72 0.3137
.word 9512 ; 116 163.13 0.2903
.word 8739 ; 117 164.53 0.2667
.word 7962 ; 118 165.94 0.2430
.word 7179 ; 119 167.34 0.2191
.word 6393 ; 120 168.75 0.1951
.word 5602 ; 121 170.16 0.1710
.word 4808 ; 122 171.56 0.1467
.word 4011 ; 123 172.97 0.1224
.word 3212 ; 124 174.38 0.0980
.word 2410 ; 125 175.78 0.0736
.word 1608 ; 126 177.19 0.0491
.word 804 ; 127 178.59 0.0245
.word 0 ; 128 180.00 0.0000
.word 64731 ; 129 181.41 -0.0245
.word 63927 ; 130 182.81 -0.0491
.word 63125 ; 131 184.22 -0.0736
.word 62323 ; 132 185.63 -0.0980
.word 61524 ; 133 187.03 -0.1224
.word 60727 ; 134 188.44 -0.1467
.word 59933 ; 135 189.84 -0.1710
.word 59142 ; 136 191.25 -0.1951
.word 58356 ; 137 192.66 -0.2191
.word 57573 ; 138 194.06 -0.2430
.word 56796 ; 139 195.47 -0.2667
.word 56023 ; 140 196.88 -0.2903
.word 55257 ; 141 198.28 -0.3137
.word 54496 ; 142 199.69 -0.3369
.word 53742 ; 143 201.09 -0.3599
.word 52996 ; 144 202.50 -0.3827

.word 52256 ; 145 203.91 -0.4052
.word 51525 ; 146 205.31 -0.4276
.word 50803 ; 147 206.72 -0.4496
.word 50089 ; 148 208.13 -0.4714
.word 49384 ; 149 209.53 -0.4929
.word 48689 ; 150 210.94 -0.5141
.word 48005 ; 151 212.34 -0.5350
.word 47331 ; 152 213.75 -0.5556
.word 46667 ; 153 215.16 -0.5758
.word 46016 ; 154 216.56 -0.5957
.word 45376 ; 155 217.97 -0.6152
.word 44748 ; 156 219.38 -0.6344
.word 44132 ; 157 220.78 -0.6532
.word 43530 ; 158 222.19 -0.6716
.word 42941 ; 159 223.59 -0.6895
.word 42365 ; 160 225.00 -0.7071
.word 41804 ; 161 226.41 -0.7242
.word 41256 ; 162 227.81 -0.7410
.word 40724 ; 163 229.22 -0.7572
.word 40206 ; 164 230.63 -0.7730
.word 39703 ; 165 232.03 -0.7883
.word 39216 ; 166 233.44 -0.8032
.word 38745 ; 167 234.84 -0.8176
.word 38290 ; 168 236.25 -0.8315
.word 37852 ; 169 237.66 -0.8449
.word 37430 ; 170 239.06 -0.8577
.word 37025 ; 171 240.47 -0.8701
.word 36637 ; 172 241.88 -0.8819
.word 36267 ; 173 243.28 -0.8932
.word 35914 ; 174 244.69 -0.9040
.word 35579 ; 175 246.09 -0.9142
.word 35262 ; 176 247.50 -0.9239
.word 34964 ; 177 248.91 -0.9330
.word 34683 ; 178 250.31 -0.9415
.word 34422 ; 179 251.72 -0.9495
.word 34179 ; 180 253.13 -0.9569
.word 33955 ; 181 254.53 -0.9638
.word 33750 ; 182 255.94 -0.9700
.word 33564 ; 183 257.34 -0.9757
.word 33398 ; 184 258.75 -0.9808
.word 33250 ; 185 260.16 -0.9853

.word 33123 ; 186 261.56 -0.9892
.word 33014 ; 187 262.97 -0.9925
.word 32926 ; 188 264.38 -0.9952
.word 32857 ; 189 265.78 -0.9973
.word 32807 ; 190 267.19 -0.9988
.word 32778 ; 191 268.59 -0.9997
.word 32768 ; 192 270.00 -1.0000
.word 32778 ; 193 271.41 -0.9997
.word 32807 ; 194 272.81 -0.9988
.word 32857 ; 195 274.22 -0.9973
.word 32926 ; 196 275.63 -0.9952
.word 33014 ; 197 277.03 -0.9925
.word 33123 ; 198 278.44 -0.9892
.word 33250 ; 199 279.84 -0.9853
.word 33398 ; 200 281.25 -0.9808
.word 33564 ; 201 282.66 -0.9757
.word 33750 ; 202 284.06 -0.9700
.word 33955 ; 203 285.47 -0.9638
.word 34179 ; 204 286.88 -0.9569
.word 34422 ; 205 288.28 -0.9495
.word 34683 ; 206 289.69 -0.9415
.word 34964 ; 207 291.09 -0.9330
.word 35262 ; 208 292.50 -0.9239
.word 35579 ; 209 293.91 -0.9142
.word 35914 ; 210 295.31 -0.9040
.word 36267 ; 211 296.72 -0.8932
.word 36637 ; 212 298.13 -0.8819
.word 37025 ; 213 299.53 -0.8701
.word 37430 ; 214 300.94 -0.8577
.word 37852 ; 215 302.34 -0.8449
.word 38290 ; 216 303.75 -0.8315
.word 38745 ; 217 305.16 -0.8176
.word 39216 ; 218 306.56 -0.8032
.word 39703 ; 219 307.97 -0.7883
.word 40206 ; 220 309.38 -0.7730
.word 40724 ; 221 310.78 -0.7572
.word 41256 ; 222 312.19 -0.7410
.word 41804 ; 223 313.59 -0.7242
.word 42365 ; 224 315.00 -0.7071
.word 42941 ; 225 316.41 -0.6895
.word 43530 ; 226 317.81 -0.6716

```
.word 44132 ; 227 319.22 -0.6532
.word 44748 ; 228 320.63 -0.6344
.word 45376 ; 229 322.03 -0.6152
.word 46016 ; 230 323.44 -0.5957
.word 46667 ; 231 324.84 -0.5758
.word 47331 ; 232 326.25 -0.5556
.word 48005 ; 233 327.66 -0.5350
.word 48689 ; 234 329.06 -0.5141
.word 49384 ; 235 330.47 -0.4929
.word 50089 ; 236 331.88 -0.4714
.word 50803 ; 237 333.28 -0.4496
.word 51525 ; 238 334.69 -0.4276
.word 52256 ; 239 336.09 -0.4052
.word 52996 ; 240 337.50 -0.3827
.word 53742 ; 241 338.91 -0.3599
.word 54496 ; 242 340.31 -0.3369
.word 55257 ; 243 341.72 -0.3137
.word 56023 ; 244 343.13 -0.2903
.word 56796 ; 245 344.53 -0.2667
.word 57573 ; 246 345.94 -0.2430
.word 58356 ; 247 347.34 -0.2191
.word 59142 ; 248 348.75 -0.1951
.word 59933 ; 249 350.16 -0.1710
.word 60727 ; 250 351.56 -0.1467
.word 61524 ; 251 352.97 -0.1224
.word 62323 ; 252 354.38 -0.0980
.word 63125 ; 253 355.78 -0.0736
.word 63927 ; 254 357.19 -0.0491
.word 64731 ; 255 358.59 -0.0245
.word 65535 ; 256 360.00 0.0000
```

```
=====
; I S R - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Modifies: Nothing
; Last Update: 16 June 95
=====
PHANTOM    KICK_DOG        ;Resets WD counter
           B PHANTOM
```

Laboratory Experiment 8

Objectives

To understand one technique to create a sine modulated PWM signal using the DSP and control the frequency of the sinusoidal modulated function.

To observe the output sinusoidal waveform by applying a simple RC low-pass filter.

Equipment

Hardware :

- PC Specifications -
 - ❑ '386 or higher IBM PC/AT
 - ❑ 1.44Mb 3.5-inch floppy drive
 - ❑ 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
 - ❑ Minimum 4Mb memory
 - ❑ Color VGA Monitor

- TMS320C24x Evaluation Board
- XDS510PP Emulator
- +5V power supply.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable
- Optical isolation (optocoupler) board
- DC power supply
- Oscilloscope
- Breadboard
- Resistors and capacitors

Software :

- MS Windows 3.1 or Windows 95

- ASCII Editor
- TMS320C2xx Assembler
- TMS320C2xx C Source Debugger

Laboratory Assignments

2. Run the example program Ch9_e1.asm and observe the output PWM waveform from the pin T1PWM on P1 connector using an oscilloscope.
5. Connect the output signal to an RC filter through an optical isolation board as shown in the figure below. Power On the Base drive & Power converter circuit. Select filter parameters $R=6.8k\Omega$, $C=1\mu F$. Reduce the PWM frequency to 5kHz and change `FREQSTEP` to 1024. Observe the output signal V_{OUT} using an oscilloscope. Answer:
 - What is the frequency of V_{OUT} ? How close is it from the command frequency given in the program?
 - Why the PWM frequency has to be reduced?
 - Why V_{OUT} is not purely sinusoidal?
 - What should be done to maintain the quality of the waveform of V_{OUT} if its frequency increases?

