Resolved Signals

What are resolved signals and how do they work.
Overview – Resolved Signals

- Why resolved signals?
- Step to creating a resolved signal in VHDL
- Resolution Functions
Busses and Wires

- What is the difference between a bus and a wire?
- Wires have only one driving source
Busses and Wires

- Busses on the other hand can be driven by one or more sources.
- In both cases there can be more than one destination for the signal.
- With busses, only the device acting as source will actually drive a value. All others will have their output at high impedance (Z).
How do you handle Busses in an HDL?

- First must consider the information present on a wire and on a bus in a digital circuit.

- Information present on a wire:
  - Wire is limited to 2 states
  - High or 1 or ‘1’
  - Low or 0 or ‘0’
  - There is a transition period between the two but High and Low are the only 2 stable states.
Information on a Bus

- Possible state for a BUS
  - Driven high (driven to a 1)
  - Driven low (driven to a 0)
  - No driving value (Z or high impedance)
  - Capacitive high (H)
  - Capacitive low (L)
  - Conflict (one driver driving it to a 1, another a 0) (X)
  - Conflict of capacitive values (W)

- And other useful values
  - U – Uninitialized
  - – - a Don’t Care
In an HDL need Resolution

- With multiple drivers of a signal how do you resolve the value seen by devices using the bus?

- RESOLUTION
  - How to determine the value when two or more drivers are driving the same signal

- Must look at all drivers and determine the appropriate value to use.
Step needed in VHDL

- Declare a type for the multi-value logic system
  - type mv4_logic is ('X','Z','0','1');
  - type mv4_logic_vector is array (natural range <>) of mv4_logic;

- Then declare a resolution function
  - function resolved (s:mv4_logic_vector) RETURN mv4_logic;

- And then the resolved signal
  - subtype mv4r_logic is resolved mv4_logic;
  - type mv4r_logic_vector is array (natural range <>) of mv4r_logic;

- Note that you need to use a subtype declaration to incorporate the resolution function.

- So there will be both a resolved and unresolved type for the multi-value system
Type and Subtype Declaration BNF

- `type_declaration ::= full_type_declaration | incomplete_type_declaration`
- `full_type_declaration ::= TYPE identifier IS type_definition`
- `incomplete_type_definition ::= TYPE identifier`
- `type_definition ::= scalar_type_definition | composite_type_definition | access_type_definition | file_type_definition`

- Can follow the BNF for a TYPE definition but find no resolution function here
It is, however, in the SUBTYPE definition

subtype_declaration ::= 
  SUBTYPE identifier IS subtype_indication

subtype_indication ::= [resolution_function_name] type_mark [constraint]

  type_mark ::= type_name | subtype_name

constraint ::= range_constraint | index_constraint
VHDL specification cont.

- Had indication that there was a function resolved
  ```vhdl
  function resolved(s : mv4_logic_vector) RETURN mv4_logic;
  ```

- Then the body of the resolution function is given in the package body where you will find:
  ```vhdl
  TYPE mv4_logic_table IS array (mv4_logic,mv4_logic) of mv4_logic;
  CONSTANT resolution_table : mv4_logic_table :=(
      -- ---------------------------------
      -- |    X    Z     0      1         |     |
      -- ---------------------------------
      ( 'X', 'X', 'X', 'X' ), --| X |
      ( 'X', 'Z', '0',  '1' ),  --| Z |
      ( 'X', '0', '0',  'X' ),  --| 0 |
      ( 'X', '1', 'X',  '1' )); --| 1 |
  ```

- And having the resolution table can write the body of the resolution func.
The resolution function

function resolved (s : mv4_logic_vector) RETURN mv4_logic IS
    variable result : mv4_logic := Z – weakest state
    BEGIN
    IF (s’length = 1) then return s(s’low)
    ELSE
    FOR i IN s’range LOOP
    result := resolution_table(result,s(i));
    END LOOP;
    END IF;
    END IF;
    return result;
    END resolved;

Execution could be shortened by adding exit when result=‘X’
The big picture

- After posting of a transaction(s) to the current value of one or more of the drivers, a vector composed of the current values of the drivers is sent to the resolution function for determination of the resolved value.

Diagram:
- 1st Process for a <= equation ....
  - 1st Driver for a
    - CV 'Z' t

- 2nd Process for a <= 2nd equation ....
  - 2nd Driver for a
    - CV '0' t

- nth Process for a <= nth equation ....
  - nth Driver for a
    - CV 'Z' t

Resolved Value
Completeness of a MVL package

- Having a MVL type with resolution is only part of creating a MVL system.
- ALSO need
  - Overloaded function for standard operators
  - Type conversion functions to convert from other type to this type and the reverse
- ieee_1164 standard MVL package is a standard package for a multi-value logic system and contains all of these.
Example of overloading for our 4 state logic system

- In the declarative part of the MVL package
  
  function “and” (l,r : mv4_logic) RETURN mv4_logic;

- In the body of the package for this MVL
  
  type mv4logic_table is array (mv4_logic,mv4_logic) of mv4_logic;
  
  CONSTANT and_table : mv4logic_table := (  
  -- ---------------- -----------------
  -- |    X    Z     0      1         |     |
  -- ---------------- -----------------
  ( 'X', 'X', '0', 'X' ), --| X |  
  ( 'X', 'X', '0', 'X' ), --| Z |  
  ( '0', '0', '0', '0' ), --| 0 |  
  ( 'X', 'X', '0', '1' )); --| 1 |  
  
  function “and” (l,r : mv4_logic) RETURN mv4_logic IS
  
  BEGIN
  
  return (and_table(l,r));
  
  END “and”;
Use of Resolved signals

- Must use a resolved signal type for any signals of mode INOUT
  - PORT ( ABUS : INOUT mv4r_logic; …

- Within an ARCHITECTURE they are needed whenever the signal will have more than one driver

ARCHITECTURE abcd OF wxyz IS
  ...
  SIGNAL a,b,c : mv4r_logic;
  ...
  BEGIN
  a <= ‘0’, ‘1’ after 10 ns, ‘0’ after 20 ns;
  p1: process
    begin
      a <= ‘Z’;
      wait …
      end process;
  p2: process
    begin
      a <= ‘0’;
      wait …
      end process;
  END abcd;
Standard logic 1164

- Package is online in the course directory
- Opens with comments on the code
- What is the first part of declaring an MVL system in a package?
Declaration Part of the Package

- Declare MVL logic system types
- Declare the resolution function
- Declare the resolved type
- And declare the array types for vectors
- Declare subtype of reduced logic systems
Overload operators

- All operators are overloaded
- Can find the package in ~degroat/ee762_assign/std_1164.vhd
Type conversion functions and edge detection

- To convert from built-in logic types of BIT and BIT_VECTOR
- And there are similar conversion functions for the reduced logic systems
- Also have functions for rising and falling edge

```vhdl
-- conversion functions
FUNCTION To_bit (s : std_logic; xmap : BIT := '0') RETURN BIT;
FUNCTION To_bitvector (s : std_logic_vector; xmap : BIT := '0') RETURN BIT_VECTOR;
FUNCTION To_bitvector (s : std_logic_vector; xmap : BIT := '0') RETURN BIT_VECTOR;
FUNCTION To_STD_ULogic (b : BIT) RETURN std_logic;
FUNCTION To_STD_LogicVector (b : BIT_VECTOR) RETURN std_logic_vector;
FUNCTION To_STD_LogicVector (s : std_logic_vector) RETURN std_logic_vector;
FUNCTION To_STD_LogicVector (b : BIT_VECTOR) RETURN std_logic_vector;
FUNCTION To_STD_LogicVector (s : std_logic_vector) RETURN std_logic_vector;

-- edge detection
FUNCTION rising_edge (SIGNAL s : std_logic) RETURN BOOLEAN;
FUNCTION falling_edge (SIGNAL s : std_logic) RETURN BOOLEAN;
```
Now have the package body

- Starts with a header
- First code is for the resolution function
- Note initial state
- Note sink state
  - Sink state is state that overrides all others
The various functions

- The AND table
- The OR table
For operation on single logic values and vectors

- Single values and vectors

```plaintext
-- overloaded logical operators (with optimizing hints)

FUNCTION "and" ( l : std_logic; r : std_logic ) RETURN UX01 IS
BEGIN
RETURN (and_table(l, r));
END "and";

FUNCTION "nand" ( l : std_logic; r : std_logic ) RETURN UX01 IS
BEGIN
RETURN  (not_table ( and_table(l, r)));
END "nand";

FUNCTION "or" ( l : std_logic; r : std_logic ) RETURN UX01 IS
BEGIN
RETURN (or_table(l, r));
END "or";

FUNCTION "and" ( l : std_logic_vector; r : std_logic_vector ) RETURN std_logic_vector IS
ALIAS lv : std_logic_vector (1 TO l'LENGTH ) IS l;
ALIAS rv : std_logic_vector (1 TO r'LENGTH ) IS r;
VARIABLE result : std_logic_vector (1 TO l'LENGTH );
BEGIN
IF ( l'LENGTH /= r'LENGTH ) THEN
ASSERT FALSE
REPORT "arguments of overloaded 'and' operator are not of the same length"
SEVERITY FAILURE;
ELSE
FOR i IN result'RANGE LOOP
result(i) := and_table (lv(i), rv(i));
END LOOP;
END IF;
RETURN result;
END "and";

FUNCTION "and" ( l : std_logic_vector; r : std_logic_vector ) RETURN std_logic_vector IS
ALIAS lv : std_logic_vector (1 TO l'LENGTH ) IS l;
ALIAS rv : std_logic_vector (1 TO r'LENGTH ) IS r;
VARIABLE result : std_logic_vector (1 TO l'LENGTH );
BEGIN
IF ( l'LENGTH /= r'LENGTH ) THEN
ASSERT FALSE
REPORT "arguments of overloaded 'and' operator are not of the same length"
SEVERITY FAILURE;
ELSE
FOR i IN result'RANGE LOOP
result(i) := and_table (lv(i), rv(i));
END LOOP;
END IF;
RETURN result;
END "and";
```

Copyright 2006 - Joanne DeGroat, ECE, OSU
Example of transactions on resolved values

- The model
- Note that there is a resolved signal res that is driver both by a concurrent signal assignment statement and in a process. Each is a driver.
- Even though the process has two statements that generate transactions for res, it is one driver of res.

ENTITY ex1 IS END ex1; -- the entity
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ARCHITECTURE one OF ex1 IS
    SIGNAL res : std_logic; -- res is a resolved signal
    BEGIN
        -- concurrent signal assignment statement for signal res
        -- this concurrent signal assignment statement is driver 1
        res <= '0' after 1 ns, '1' after 10 ns, '0' after 20 ns;
        -- this process assigns values to res and is driver 2
        PROCESS
            WAIT FOR 5 ns;
            res <= 'L' after 1 ns;
            WAIT FOR 20 ns;
            res <= '1' after 1 ns;
        END PROCESS;
    END one;
Time 0- and time 0

- At startup simulate each process till it suspends
- Each driver is set to the initial value of ‘U’
- Concurrent stmt is driver 1 of res and generates the transactions shown
- The process immediately suspends until 5 ns
Advance time till time when something occurs = 1 ns

- At 1 ns driver 1 (the concurrent stmt) goes to a value of ‘0’
- Resolved value is still a ‘U’
Avoidance time to 5 ns when the process resumes

- Process generates transaction of an ‘L’ for res at 6 ns
- Process then suspends until 25 ns
Advance time to 6 ns

- Transaction on driver 2 becomes the current value on driver 2
- Re-evaluate resolution function
Advance time to next time

- Process is suspended until 25 ns
- Transaction on Driver 1 at 10 ns
- Advance time to 10 ns and post transaction and re-evaluate resolution function
Advance time to next time at which something occurs

- At 20 ns last transaction for Driver 1 is posted
- Process is suspended until 25 ns
Adance time again, to 25 ns

- At 25 ns process resumes
- Generates transaction for res of (‘1’,26ns)
- Process then suspends until 30 ns
Advance to 26 ns

- Post a value of ‘1’ to res
- Now resolving a ‘0’ with a ‘1’
- Simulation never goes quiescent as the process will always be waiting
A 2\textsuperscript{nd} example of transactions on resolved values

- The model is similar to the first

- Again there is a resolved signal res that is driver both by a concurrent signal assignment statement and in a process. Each is a driver. However, this time res is initialized to ‘1’

- Again there are two drivers of res

```vhdl
ENTITY ex2 IS END ex2;  -- the entity
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ARCHITECTURE one OF ex2 IS
  SIGNAL res : std_logic := '1';  -- res is a resolved signal
BEGIN
  -- this concurrent signal assignment statement is driver 1
  res <= '0' after 1 ns, '1' after 10 ns, '0' after 20 ns;

  -- this process assigns values to res and is driver 2
  -- and slightly different from the process in ex1
  PROCESS
    BEGIN
      res <= 'Z' after 2 ns;
      WAIT FOR 5 ns;
      res <= 'L' after 25 ns;
      WAIT FOR 20 ns;
      res <= '1' after 1 ns;
  END PROCESS;
END one;
```
Time 0- and time 0

- At startup simulate each process till it suspends
- Each driver is set to the initial value of ‘1’
- Concurrent stmt is driver 1 of res and generates the transactions shown
- Process generates the signal transaction, suspends until 5 ns
Advance time till time when something occurs = 1 ns

- At 1 ns driver 1 (the concurrent stmt) goes to a value of ‘0’
- Resolved value is now an ‘X’
Advance time till time when something occurs = 2 ns

- At 2 ns driver 2, the process, is updated with a value of ‘Z’ for res.
- Resolved value is now an ‘0’
- Still have process suspended until 5 ns
Avdance time to 5 ns when the process resumes

- Process generates transaction of an ‘L’ for res after 25 ns or 30 ns into simulation
- Process then suspends until 25 ns
Advance time to 10 ns

- Posting transaction on Driver 1.
- Re-evaluate resolution function
- Process 2 still suspended until 25 ns.
Advance time to next time, 20 ns

- Post transaction on Driver 1
- Process still suspended till 25 ns
Advance time to next time, 25 ns

- Time resume process
- First generate a transaction of res(‘1’,26ns)
- This is before the transaction on the project output waveform so that old transaction is deleted and this new transaction posted.
Advance time to next time, 25 ns

- Process continues at top and executes res<=‘Z’ after 5 ns;
- Generates transaction res(‘Z’,27ns)
- Now must add this transaction to driver 2
- Process then suspends until 30 ns
Adding transaction to driver 2

- Run the algorithm for posting transactions to drivers (Lect 16).
- Step 1 no at or after to delete
- Step2 – append the transaction
  - POW – CV - res(‘1’,26ns) - res(‘Z’,27ns)
- Now run part II of the algorithm
Part II of the Update algorithm

- Step 1 - Mark new transactions
  - POW – CV - res(‘1’,26ns) - Xres(‘Z’,27ns)

- Step 2 - An old transaction is marked if it immediately precedes a marked transaction and its value component is the same as that of the marked transaction.
  - Here the values are different so it is not marked
  - POW – CV - res(‘1’,26ns) - Xres(‘Z’,27ns)

- Step 3 – aThe CV is marked
  - POW – xCV - res(‘1’,26ns) - Xres(‘Z’,27ns)

- Step 3 – All unmarked are deleted

- POW – CV - res(‘Z’,27ns)
Resulting in

- \( \text{res} \)
  - ‘1’ \( \times \) ‘X’ \( \times \) ‘0’ \( \times \) ‘1’

- \( \text{Dr1} \)
  - ‘1’ \( \times \) ‘0’ \( \times \) ‘1’ \( \times \) ‘0’

- \( \text{Dr2} \)
  - ‘1’ \( \times \) ‘Z’

- CV – ‘0’

- CV – ‘Z’ ('Z', 27ns)
Advance time to 27ns

- Post transaction on Driver 2
- Process suspended until 30 ns
- As values assigned by process will result in no change of value will end here.