COE 405
Dataflow Descriptions in VHDL

Dr. Aiman H. El-Maleh
Computer Engineering Department
King Fahd University of Petroleum & Minerals

Outline

- **Constructs for Dataflow Descriptions**
- **Selection Constructs**
- **Multiplexing and Clocking**
- **Block Statement & Nested Blocks**
- **Guarded Constructs**
- **Multiple Assignments**
- **Resolution Function**
- **Resolution of Guarded Signals**
- **State Machines**
- **Complete Dataflow Examples**

DATA FLOW MODEL

- **Represents Register Transfer operations**
- **There is Direct Mapping between Data Flow Statements && Register Structural Model**
  - Implied Module Connectivity
  - Implied Muxes & Buses
- **Main Data Flow VHDL Constructs:**
  - Concurrent Signal Assignment Statements
  - Block Statement

Signal Assignment …

- **Unconditional: Both Sequential & Concurrent**
- **Conditional: Only Concurrent; Conditions Must Be Boolean, May Overlap and Need Not Be Exhaustive**
- **Selected: Only Concurrent; Cases Must Not Overlap and Must Be Exhaustive**
- **Conditional Signal Assignment**

```
[ Label: ]    target <= [Guarded] [Transport ]
                         Wave1     when  Cond1     Else
                         Wave2     when  Cond2     Else
                         ………………………………..
                          Waven-1  when  Condn-1  Else
                          Waven  ; -- Mandatory Wave
```

… Signal Assignment

- **Selected Signal Assignment**

```
With Expression  Select
                    target <= [Guarded] [Transport]
                              Wave1     when  Choice1 ,
                              Wave2     when  Choice2 ,
                              ……………………………
                              Waven-1  when  Choicen-1 ,
                              Waven     when  OTHERS ;
```

VHDL-93: Any *Wavei* Can Be Replaced By the Keyword UNAFFECTED (Which Doesn't Schedule Any Transactions on the Target Signal.)

Signal Assignment Examples

Example: A 2x4 Decoder

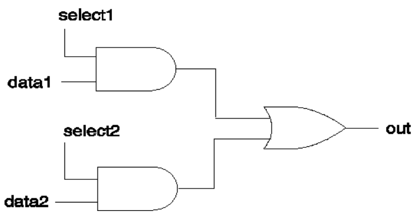
```
Signal  D  :  Bit_Vector(1 To 4) := "0000";
Signal  S0, S1  :  Bit;
……………………………………………
Decoder:   D <= "0001" after T  When S1='0' and S0='0' else
                "0010"  after T  When S1='0' else
                 "0100" after T  When S0='0' else  "1000" ;
```

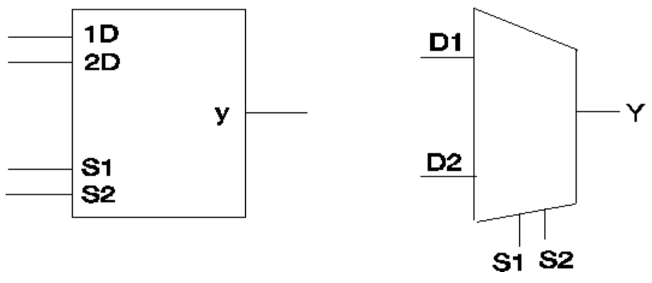
Example: 4-Phase Clock Generator

```
Signal Phi4  :  Bit_Vector(1 To 4) := "0000";
……………………………………………
ClkGen:   With  Phi4  Select
                    Phi4   <=  "1000" after T  When "0000" ,
                               "0100" after T  When "1000" ,
                               "0010" after T  When "0100" ,
                              "0001" after T  When "0010" ,
                              "1000" after T  When "0001" ,
                              "0000" When   Others  ; -- Exhaustive
```

Multiplexing …

- Multiplexers are used for data selection

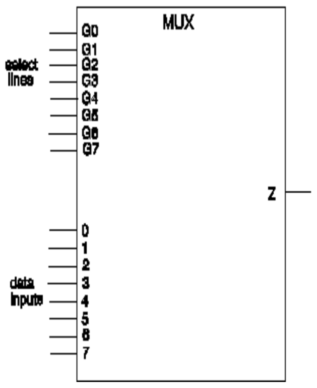




9-7

… Multiplexing

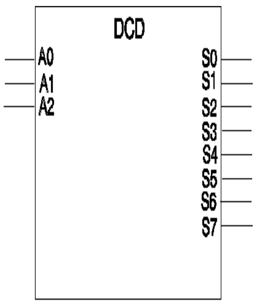
```
USE WORK.basic_utilities.ALL;
-- FROM PACKAGE USE: qit, qit_vector
ENTITY mux_8_to_1 IS
PORT (i7, i6, i5, i4, i3, i2, i1, i0 : IN qit;
s7, s6, s5, s4, s3, s2, s1, s0 : IN qit; z : OUT qit );
END mux_8_to_1;
--
ARCHITECTURE dataflow OF mux_8_to_1 IS
SIGNAL sel_lines : qit_vector ( 7 DOWNTO 0);
BEGIN
sel_lines <= s7&s6&s5&s4&s3&s2&s1&s0;
WITH sel_lines SELECT
z <= '0' AFTER 3 NS WHEN "00000000",
i7 AFTER 3 NS WHEN "10000000" | "Z0000000",
i6 AFTER 3 NS WHEN "01000000" | "0Z000000",
i5 AFTER 3 NS WHEN "00100000" | "00Z00000",
i4 AFTER 3 NS WHEN "00010000" | "000Z0000",
i3 AFTER 3 NS WHEN "00001000" | "0000Z000",
i2 AFTER 3 NS WHEN "00000100" | "00000Z00",
i1 AFTER 3 NS WHEN "00000010" | "000000Z0",
i0 AFTER 3 NS WHEN "00000001" | "0000000Z",
'X' WHEN OTHERS;
END dataflow;
```



9-8

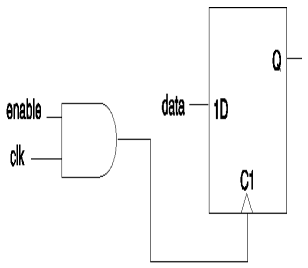
3-to-8 Decoder

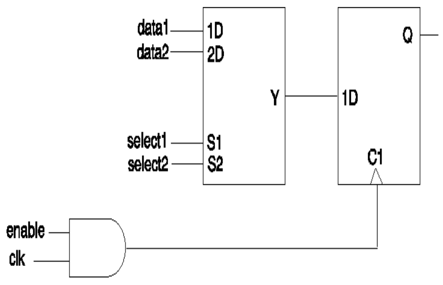
```
USE WORK.basic_utilities.ALL;
-- FROM PACKAGE USE : qit_vector
ENTITY dcd_3_to_8 IS
PORT (adr : IN qit_vector (2 DOWNTO 0);
so : OUT qit_vector (7 DOWNTO 0));
END dcd_3_to_8;
--
ARCHITECTURE dataflow OF dcd_3_to_8 IS
BEGIN
WITH adr SELECT
so <= "00000001" AFTER 2 NS WHEN "000",
"00000010" AFTER 2 NS WHEN "00Z" | "001",
"00000100" AFTER 2 NS WHEN "0Z0" | "010",
"00001000" AFTER 2 NS WHEN "0ZZ" | "0Z1" | "01Z" |
"011",
"00010000" AFTER 2 NS WHEN "100" | "Z00",
"00100000" AFTER 2 NS WHEN "Z0Z" | "Z01" | "10Z" |
"101",
"01000000" AFTER 2 NS WHEN "ZZ0" | "Z10" | "1Z0" |
"110",
"10000000" AFTER 2 NS WHEN "ZZZ" | "ZZ1" | "Z1Z" |
"Z11" | "1ZZ" | "1Z1" | "11Z" | "111",
"XXXXXXXX" WHEN OTHERS;
END dataflow;
```



Clocking …

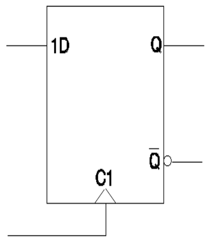
- Flip flop clocking selects data
- Various forms of data selection may be combined





… Clocking …

```
ENTITY d_flipflop IS
GENERIC (delay1 : TIME := 4 NS; delay2 : TIME := 5 NS);
PORT (d, c : IN BIT; q, qb : OUT BIT);
END d_flipflop;
--
ARCHITECTURE assigning OF d_flipflop IS
SIGNAL internal_state : BIT;
BEGIN
internal_state <= d WHEN ( c = '1' AND NOT c'STABLE )
ELSE  internal_state;
q <= internal_state AFTER delay1;
qb <= NOT internal_state AFTER delay2;
END assigning;
```



- *A simple flipflop uses internal_state*
- *On clock edge d is transferred to internal_state*
- *Events on internal_state cause assignments to q and qb*
    - *Two transactions on internal_state for every clock edge*

Clocking …

```
ENTITY d_flipflop IS
GENERIC (delay1 : TIME := 4 NS; delay2 : TIME := 5 NS);
PORT (d, c : IN BIT; q, qb : OUT BIT);
END d_flipflop;
--
ARCHITECTURE guarding OF d_flipflop IS
      BEGIN
      ff: BLOCK ( c = '1' AND NOT c'STABLE )
      BEGIN
            q <= GUARDED d AFTER delay1;
            qb <= GUARDED NOT d AFTER delay2;
      END BLOCK ff;
END guarding;
```

- *Better representation of clocking* ***disconnects*** *d from q*
- *Disconnection is specified by GUARDED*
- *GUARDED assignments are guarded by guard expression*
    - *Guard expression is only TRUE for 1 delta*
- *Can also guard selected and conditional signal assignments*

Block Statement

- Block Statement is a Concurrent VHDL Construct Which is Used Within an Architectural Body to Group (Bind) a Set of Concurrent Statements

```
Block_Label: Block [ (Guard_Condition) ] [ IS ]
                        Block Header;
                        Block_Declarations;
                   Begin
                        Concurrent_Statements;
                   END Block  Block_Label  ;
```

- **A Guard Condition May be Associated with a Block Statement to Allow Enabling/Disabling of Certain Signal Assignment Statements.**
- **The Guard Condition Defines an *Implicit Signal* Called** *GUARD.*
- **In the Simplest Case, Binding (*Packing !*) Statements Within A** Block **Has No Effect On the Model.**
- **Blocks Can Be Nested.**

9-13

Block Statement Example

```
Architecture DF of D_Latch is
         Begin
        B : Block (Clk = `1`)
             Signal  I_State :Bit;  Block Local Signal
              Begin
                   I_State <=  Guarded  D ;
                    Q     <= I_State after 5 ns;
                    QB  <= not I_State after 5 ns;
              END  Block B ;
END DF ;
```

- **UnGuarded Signal Targets (e.g.,** Q, QB**) are independent of the Guard Condition**

- **If *Guard Condition* (Clk=`1`) is TRUE, *Guarded Statements* within block are <u>Enabled</u> (<u>*Made Active*</u>)**
- **Guarded Statements (e.g.,** *I_State***) execute when**
  - **Guard Condition Becomes True, <u>AND</u>**
  - **While Guard Condition is True, a Signal on the RHS Changes Value**

9-14

Positive-Edge-Triggered DFF …

```
Library IEEE;
Use IEEE.Std_Logic_1164.ALL;
Entity DFF is
       Generic(TDel: Time:= 5 NS);
       Port(D, Clk: in Std_Logic; Q, QB: out Std_Logic);
End DFF;
```

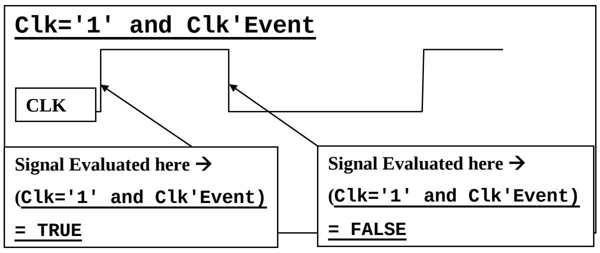
- *We will show several dataflow architectures with and without Block statement*
- *Will show why some of these architectures do not work*

… Positive-Edge-Triggered DFF …

Arch 1

```
Architecture DF1_NO_Block of DFF is
Signal I_State: Std_Logic:='1';
Begin
       I_State <= D when (Clk='1' and
Clk'Event) else I_state;
       Q <= I_state after TDel ;
       QB <= not I_state after TDel ;
End  DF1_NO_Block ;
```

Works



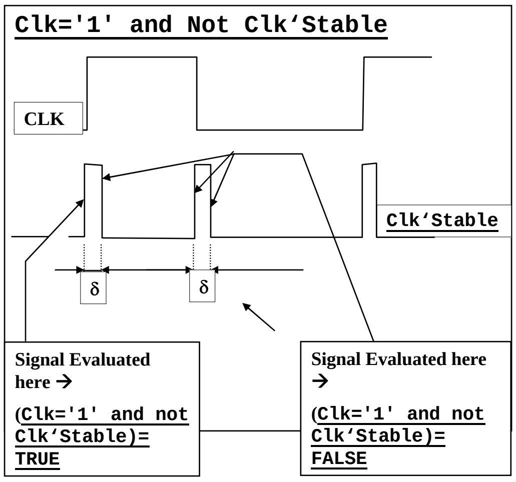
Signal Evaluated 2-Times Per Clock Cycle

… Positive-Edge-Triggered DFF …

Doesn't Work

Arch 2

```
Architecture
DF2_NO_Block of DFF is
Signal I_State:
Std_Logic:='1';
Begin
I_State <= D after TDel
when (Clk='1' and
(not(Clk'Stable))) else
I_state;
       Q <= I_state;
       QB <= not
I_state;
End  DF2_NO_Block ;
```


Clk='1' and Not Clk'Stable
CLK
Clk'Stable
δ
δ
Signal Evaluated here →
(Clk='1' and not Clk'Stable)= TRUE
Signal Evaluated here →
(Clk='1' and not Clk'Stable)= FALSE

Signal Evaluated 4-Times Per Clock Cycle

… Positive-Edge-Triggered DFF …

Works

Arch 3

```
Architecture DF3_NO_Block of DFF
is
Signal I_State: Std_Logic:='1';
Begin
       I_State <= D when
(Clk='1' and (not(Clk'Stable)))
else I_state;
       Q <= I_state after TDel;
       QB <= not I_state after
TDel ;
End DF3_NO_Block ;
```

I_State gets the value of D after 1 delta and Its value does not get overwritten

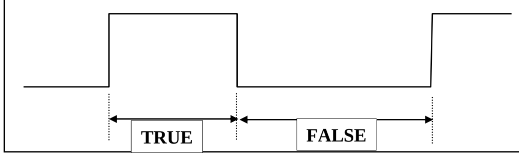
… Positive-Edge-Triggered DFF …

Arch4

```
Architecture DF1_Block of DFF is
Signal I_State: Std_Logic:='1';
Begin
 D_Blk: Block(Clk='1' and Clk'Event)
    Begin
       Q  <= Guarded D after Tdel;
       QB <= Guarded not D after Tdel;
    End Block;
End  DF1_Block ;
```

Doesn't Work

GUARD <= Clk='1' and Clk'Event



Signal Evaluated Continuously while Clk = '1' !!!

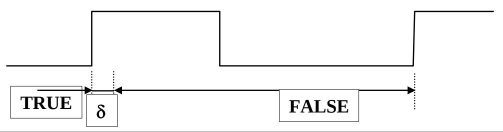
Positive-Edge-Triggered DFF …

Arch5

```
Architecture DF2_Block of DFF is
Signal I_State: Std_Logic:='1';
Begin
 D_Blk: Block(Clk='1' and not Clk'Stable)
    Begin
       Q  <= Guarded D after Tdel;
       QB <= Guarded not D after Tdel;
    End Block;
End  DF2_Block ;
```

Works

GUARD <= Clk='1' and not Clk'Stable



Signal Evaluated Once Per Clock Cycle

(At Rising Edge of the Clock)

Nested Block Statements

```
Architecture  Block_Structure of   Demo  is
begin
 A: Block -- 1
      Outer Block Declarative Section;
  Begin
       Concurrent Statements of Outer Block;
    B:Block -- 1.1
          Inner Block ``A`` Declarative Section;
       begin
             Concurrent Statements of Inner Block ``A``;
             .................................
       end Block B;
    C:Block -- 1.2
          Inner Block ``B`` Declarative Section;
       begin
             Concurrent Statements of Inner Block ``B``;
             .................................
       end Block C;
 end Block A;
 D: Block -- 2
             .................................
 end Block D;
end   Block_Structure;
```

Use of Nested Blocks For Composite Enabling Conditions

```
ARCHITECTURE guarding OF DFF IS
    BEGIN
    edge: BLOCK ( c = '1' AND NOT c'STABLE )
    BEGIN
           gate: BLOCK ( e = '1' AND GUARD )
           BEGIN
           q <= GUARDED d AFTER delay1;
           qb <= GUARDED NOT d AFTER delay2;
           END BLOCK gate;
    END BLOCK edge;
    END guarding;
```

- Inner Guard Signal <= (e= '1') AND (c= '1' AND NOT c'STABLE)
- Can nest block statements
- Combining guard expressions must be done explicitly
- Implicit GUARD signals in each block

Data Flow Example …

Model A System with 2 8-Bit Registers R1 and R2, a 2-Bit Command signal "**COM**" and an external 8-Bit Input "**INP**"

- When Com= "00"→ R1 is Loaded with External Input
- When Com= "01"→ R2 is Loaded with External Input
- When Com= "10"→ R1 is Loaded with **R1+R2**
- When Com= "11"→ R1 is Loaded with **R1-R2**

```
Use Work.Utils_Pkg.ALL
Entity DF_Ex is
    Port (Clk: IN Bit; Com: IN Bit_Vector (1 DownTo
    0); INP: IN  Bit_Vector(7 DownTo 0)
    R1, R2: BUFFER Bit_Vector(7 DownTo 0));
End DF_Ex;
```

9-23

… Data Flow Example …

```
Architecture DF of DF_Ex is
Signal  Mux_R1, R1, R2, R2C, R2TC, Mux_Add,
        Sum: Bit_Vector(7 DownTo 0);
Signal  D00, D01, D10, D11, R1E: Bit;
Begin
   D00 <= not Com(0) and not Com(1); -- Decoder
   D01 <= not Com(0) and Com(1); -- Decoder
   D10 <= Com(0) and not Com(1); -- Decoder
   D11 <= Com(0) and Com(1); -- Decoder
   R2C  <= not R2;
   R2TC <= INC(R2C); -- Increment Function Defined  in the Package
   Mux_Add <=R2TC when D11 = '1' Else   R2 ;
```

9-24

… Data Flow Example

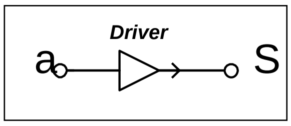
```
Sum <= ADD(R1,  Mux_Add); -- ADD Function-- Defined in Package
Mux_R1 <= INP when D00 = '1' Else  Sum;
R1E  <= D00 OR D10  OR D11;
Rising_Edge: BLOCK(Clk='1' and not Clk'Stable) Begin
   R1_Reg: BLOCK(R1E='1' AND GUARD) Begin
       R1 <= Guarded Mux_R1 ;
   End Block R1_Reg ;

   R2_Reg: BLOCK(D01='1' AND GUARD) Begin
       R2 <= Guarded INP ;
   End Block R2_Reg ;
End Block Rising_Edge;
End DF;
```

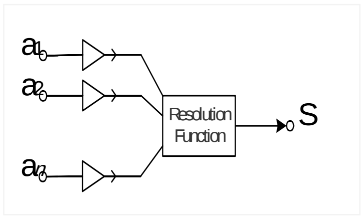
9-25

Signal Resolution Function …

- **Each signal assignment statement defines a Signal Driver (Source)**
- **Example:**
  - S <= a After T ;



- **Multiple Concurrent Assignment Statements to the Same Signal defines Multiple Drivers (Signal Sources).**
- **Such Multi-Driver Signals are commonly encountered in Buses with Multiple Drivers**
- **Electrically, Tri-State or Open-Collector Drivers are used to Resolve Conflicts of the Different Drivers**
- **VHDL Model requires the definition of a Resolution Function to Resolve values being assigned to the Common Signal by all its Drivers**



9-26

… Signal Resolution Function …

- A User-Defined *Resolution Function* (RF) Must be Defined to Resolve the Final Value of Such Signals. Such Signals are Called *Resolved* Signals.
- AN Error is Reported whenever an RF is not Defined for Multi-Driver Signals.
- The RF is Invoked Each time any of the signal Drivers Receives a New Value.

RF Example

Input : 1-D *Unconstrained* Array of Values of the Type to be Resolved

Output (Returned Value): A Value of the *Resolved* Type.

```
Type MVL4 IS (` X `, `0`, `1`, `Z`);
Type MVL4_Vector IS Array (Natural Range <>) OF MVL4;
Function Wired_OR(Vin: MVL4_Vector) Return MVL4 IS
          Variable R: MVL4 :=`0`;  -- Initial Default Result
Begin
    For  I  IN Vin`Range
        Loop
            IF  Vin(i) = `1` Then  R:= `1`; Exit;
            Elsif Vin(i) = `X` Then  R:= `X`;
            Else Null;
            EndIF;
        End Loop;
Return R;
End Wired_Or
```

9-27

… Signal Resolution Function

Notes

1. The RF Should Handle **Any** # of Signal Drivers. Thus The Input To RF Must Be an *Unconstrained Array*.
2. RF Should Not Depend on The *Order of Values in the Input Vector*. Thus, an Input (‘0’, ‘1’) Should Return Identical Value Like The Vector (‘1’, ‘0’).
3. Multiple Assignments To the Same Target Signal in A Process Body *Does Not Require an RF* Since a Process Body Is Sequential Not Concurrent.

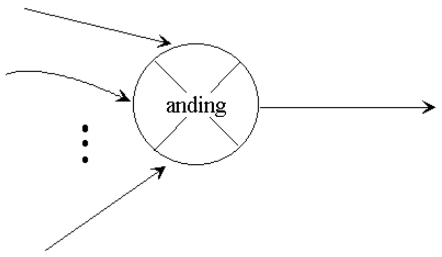
Declaring Resolved Signals

1. Declare A Resolved **SubType** 2. Declare Signal (Resolved Signal) To Be of this Resolved **SubType**. Example **SubType** *Resolved_Wire* **IS** *Wired_Or* **MVL4;** **Signal** Resolved_Sig: *Resolved_Wire*;	• Declare The RF Directly in the Resolved Signal Declaration Example **Signal** Resolved_Sig: *Wired_Or* MVL4**;**

9-28

“ANDING” Signal Resolution Function

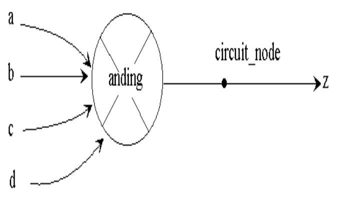
```
-- FROM basic_utilities USE qit, qit_vector, "AND"
FUNCTION anding ( drivers : qit_vector) RETURN qit IS
VARIABLE accumulate : qit := '1';
BEGIN
FOR i IN drivers'RANGE LOOP
accumulate := accumulate AND drivers(i);
END LOOP;
RETURN accumulate;
END anding;
```



9-29

“ANDING” Signal Resolution Function

```
USE WORK.basic_utilities.ALL;
-- FROM PACKAGE USE: qit
ARCHITECTURE wired_and OF y_circuit IS
FUNCTION anding ( drivers : qit_vector)
RETURN qit IS
VARIABLE accumulate : qit := '1';
BEGIN
FOR i IN drivers'RANGE LOOP
accumulate := accumulate AND drivers(i);
END LOOP;
RETURN accumulate;
END anding;
SIGNAL circuit_node : anding qit;
BEGIN
circuit_node <= a;
circuit_node <= b;
circuit_node <= c;
circuit_node <= d;
z <= circuit_node;
END wired_and;
```



9-30

“ORING” Signal Resolution Function

```
FUNCTION oring ( drivers : qit_vector) RETURN qit IS
VARIABLE accumulate : qit := '0';
BEGIN
FOR i IN drivers'RANGE LOOP
accumulate := accumulate OR drivers(i);
END LOOP;
RETURN accumulate;
END oring;
```

```
SUBTYPE ored_qit IS oring qit;
. . .
-- The following declarations are equivalent
SIGNAL t : ored_qit;
SIGNAL t : oring qit;
```

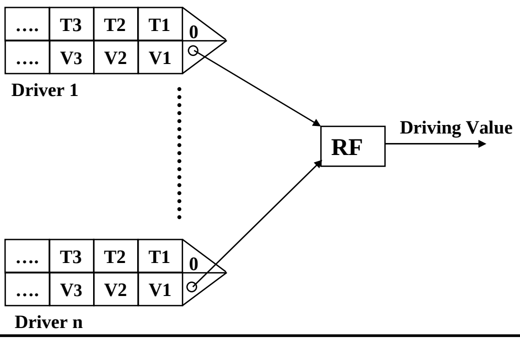
Multiplexer uses implicit ORing

```
ARCHITECTURE multiple_assignments OF mux_8_to_1 IS
SIGNAL t : ored_qit;
BEGIN
t <= i7 AND s7;
t <= i6 AND s6;
t <= i5 AND s5;
t <= i4 AND s4;
t <= i3 AND s3;
t <= i2 AND s2;
t <= i1 AND s1;
t <= i0 AND s0;
z <= t;
END multiple_assignments;
```

- *Multiplexer uses implicit ORing on t*
- *AND_OR logic is realized*

Resolution of Non-Guarded Signals

- Resolved Non-Guarded LHS Signal Values Are Determined by ***The Resolution Function*** from CVs of all Driver Signals on the RHS
- ***Expired Transactions*** on Any of the Signal Drivers, Activates the RF to Determine the new value of the output signal
- ***Pending Transaction*s** on the PWFM of the Signal Continue to Affect the Signal Value (Through RF) as they Expire.

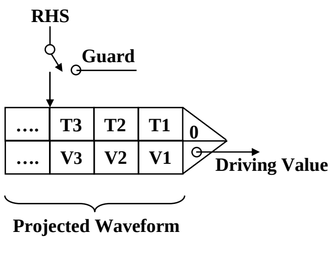


Resolution of Guarded Signals

- **(GUARD = False) → LHS Signal is *Disconnected* from its Driver Signals on the RHS**

$\Downarrow$

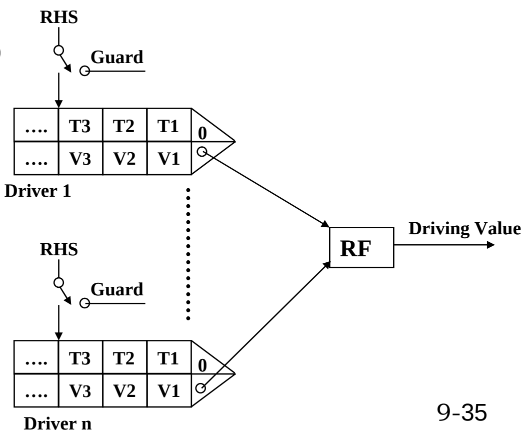
- ***No New Transactions* May Be Placed on the LHS Signal Driver**
- ***Pending Transaction*s on the PWFM of the Signal Continue to Affect the Signal Value as they Expire.**



Resolution of Guarded Signals

- A ***Resolved Guarded Signal*** is Declared to Be of either ***REGISTER*** *kind* or ***BUS*** Kind.
- Only Drivers with (GUARD = True) Participate in Determining Value of Target Signal
- If a Driver has (GUARD = False)

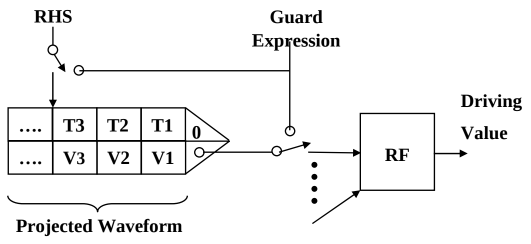
  It is Considered **Turned-Off**
- ***Register Signals*** drivers *DO NOT Invoke* the *RF in Case All Drivers Are Turned Off* → Signal **Retains its Previous Value**.
- Signals of ***BUS*** Kind *Invoke the RF in case All Signal Drivers Are Turned Off* → *RF is Invoked with a NULL input* → ***Default Value is Returned***.



9-35

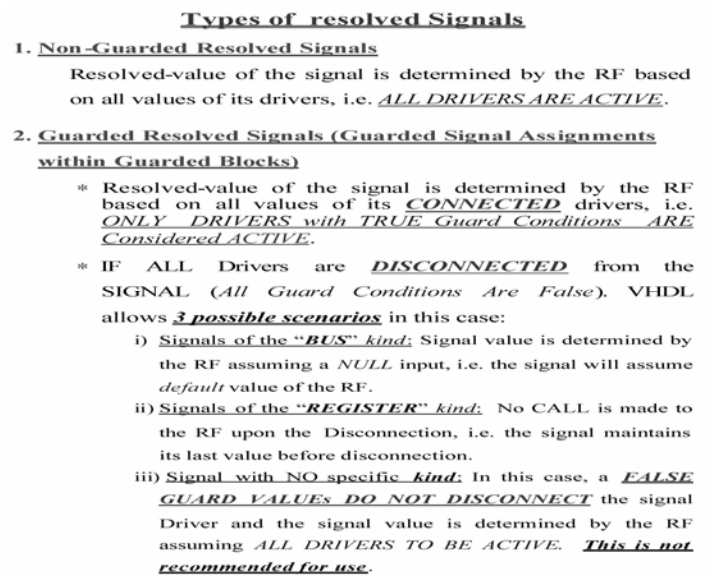
Resolution of Guarded Signals

- **Events and Transactions on Signals of BUS & Register kind are exactly the same as long as at least One Driver is ON.**
- **When all Drivers are OFF, Register signals will maintain their Previous Values while BUS signals will assume the Default Value of the RF.**



9-36

Types of Resolved Signals



Types of resolved Signals

1. **Non-Guarded Resolved Signals**

   Resolved-value of the signal is determined by the RF based on all values of its drivers, i.e. *ALL DRIVERS ARE ACTIVE*.

2. **Guarded Resolved Signals (Guarded Signal Assignments within Guarded Blocks)**

   * Resolved-value of the signal is determined by the RF based on all values of its ***CONNECTED*** drivers, i.e. *ONLY DRIVERS with TRUE Guard Conditions ARE Considered ACTIVE*.

   * IF ALL Drivers are ***DISCONNECTED*** from the SIGNAL (*All Guard Conditions Are False*). VHDL allows ***3 possible scenarios*** in this case:

   i) Signals of the "***BUS***" *kind*: Signal value is determined by the RF assuming a *NULL* input, i.e. the signal will assume *default* value of the RF.

   ii) Signals of the "***REGISTER***" *kind*: No CALL is made to the RF upon the Disconnection, i.e. the signal maintains its last value before disconnection.

   iii) Signal with NO specific ***kind***: In this case, a ***FALSE GUARD VALUEs DO NOT DISCONNECT*** the signal Driver and the signal value is determined by the RF assuming *ALL DRIVERS TO BE ACTIVE*. ***This is not recommended for use***.

Syntax

Signal <*sig_name*> **:** <***resolved sig_subtype***> ***[Signal_kind] [:=Initial_Value]*** **;**

Signal_kind **::= BUS | Register**

Examples:

Signal x : Wired_MVL4 *BUS* ;

Signal y : Wired_MVL4 *Register* ;

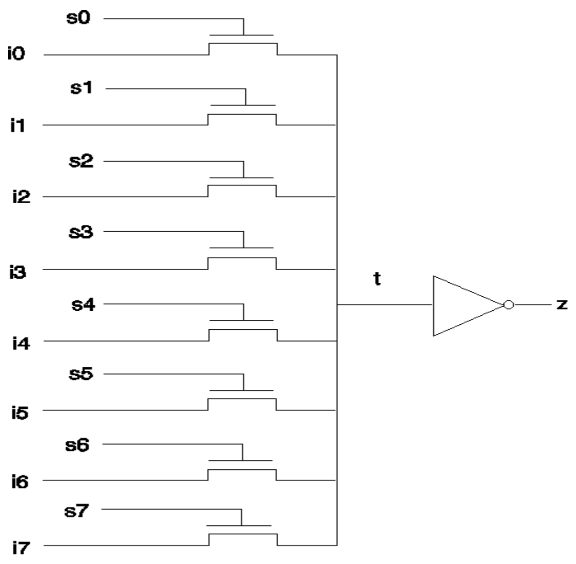
Note:

1. Only Signals of Kind* BUS *May be Specified for as Port Signals

***2. Signals of* Register *Kind May NOT be Specified as Port Signals*)**

```
Entity ex is
Port( s1, s2 : in MVL4; Z: out  wired_MVL4  BUS) ;
End ex;
```

Example MOS (PTL) Multiplexer …

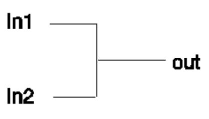


… Example MOS (PTL) Multiplexer …

```
FUNCTION wire (a, b : qit) RETURN qit IS
CONSTANT qit_and_table : qit_2d := (
                                        ('0','X','0','X'),
                                        ('X','1','1','X'),
                                        ('0','1','Z','X'),
                                        ('X','X','X','X'));
BEGIN
          RETURN qit_and_table (a, b);
END wire;
```

In2: \ In1:	0	1	Z	X
0	0	X	0	X
1	X	1	1	X
Z	0	1	Z	X
X	X	X	X	X

Out



… Example MOS (PTL) Multiplexer …

```
FUNCTION wiring ( drivers : qit_vector) Return qit IS
Variable accumulate : qit := 'Z'; -- Default
BEGIN
FOR i IN drivers'RANGE LOOP
            accumulate := wire (accumulate, drivers(i));
END LOOP;
            RETURN accumulate;
END wiring;
```

```
SUBTYPE wired_qit IS wiring qit;
TYPE wired_qit_vector IS Array (Natural Range <>) OF wired_qit;
```

… Example MOS (PTL) Multiplexer …

Model 1 (BUS Signal Kind)

```
USE WORK.basic_utilities.ALL;
    -- FROM PACKAGE USE: wired_qit
Architecture multiple_guarded_assignments OF mux_8_to_1 IS
SIGNAL t : Wired_qit  BUS;
BEGIN
b7: Block (s7 = '1' OR s7 = 'Z') Begin t <= Guarded  i7; End Block;
b6: Block (s6 = '1' OR s6 = 'Z') Begin t <= Guarded i6; End Block ;
b5: Block (s5 = '1' OR s5 = 'Z') Begin t <= Guarded i5; End Block ;
b4: Block (s4 = '1' OR s4 = 'Z') Begin t <= Guarded i4; End Block ;
b3: Block (s3 = '1' OR s3 = 'Z') Begin t <= Guarded i3; End Block ;
b2: Block (s2 = '1' OR s2 = 'Z') Begin t <= Guarded i2; End Block ;
b1: Block (s1 = '1' OR s1 = 'Z') Begin t <= Guarded i1; End Block ;
b0: Block (s0 = '1' OR s0 = 'Z') Begin t <= Guarded i0; End Block ;
 z <= not t after 1 NS;
 END multiple_guarded_assignments;
```

- **Disconnection** is realized by **block** statements
- *If all drivers are disconnected*

Hardware returns to 'Z' → Modeling This Requires Using **BUS** Signal Kind.

… Example MOS (PTL) Multiplexer

Model 2 (Register Signal Kind)

```
USE WORK.basic_utilities.ALL;
    -- FROM PACKAGE USE: wired_qit
Architecture multiple_guarded_assignments OF mux_8_to_1 IS
SIGNAL t : Wired_qit  REGISTER;
BEGIN
b7: Block (s7 = '1' OR s7 = 'Z') Begin t <= Guarded  i7; End Block;
b6: Block (s6 = '1' OR s6 = 'Z') Begin t <= Guarded i6; End Block ;
b5: Block (s5 = '1' OR s5 = 'Z') Begin t <= Guarded i5; End Block ;
b4: Block (s4 = '1' OR s4 = 'Z') Begin t <= Guarded i4; End Block ;
b3: Block (s3 = '1' OR s3 = 'Z') Begin t <= Guarded i3; End Block ;
b2: Block (s2 = '1' OR s2 = 'Z') Begin t <= Guarded i2; End Block ;
b1: Block (s1 = '1' OR s1 = 'Z') Begin t <= Guarded i1; End Block ;
b0: Block (s0 = '1' OR s0 = 'Z') Begin t <= Guarded i0; End Block ;
 z <= not t after 1 NS;
 END multiple_guarded_assignments;
```

•**Disconnection** is realized by **block** statements

•*If all drivers are disconnected*

Real hardware Maintains State for few milliseconds (As Charge on the Capacitance of Node "t".

•Use **Register** to implement this behavior

General Multiplexer

```
USE WORK.basic_utilities.ALL;
-- FROM PACKAGE USE : qit, qit_vector, wired_qit
ENTITY mux_n_to_1 IS
PORT (i, s : IN qit_vector; z : OUT wired_qit BUS);
END mux_n_to_1;
--
ARCHITECTURE multiple_guarded_assignments OF mux_n_to_1 IS
BEGIN
bi: FOR j IN i'RANGE GENERATE
            bj: BLOCK (s(j) = '1' OR s(j) = 'Z')
            BEGIN
                        z <= GUARDED i(j);
            END BLOCK;
    END GENERATE;
END multiple_guarded_assignments;
```

Disconnection Delay …

- **Signal assignment can specify connection delay**
  - If i5 changes value while guard expression is true or guard expression changes from False to True, i5 will be assigned to t after 4 ns.
  - The 4ns delay only applies when a driver is connected and does not apply when it is disconnected.

```
b5: BLOCK (s5='1' OR s5='Z')
BEGIN t <= GUARDED i5 AFTER 4 NS;
END BLOCK;
```

- **A disconnection specification statement can be used to specify the disconnection delay for a guarded signal**
  - Disconnect t: wired_qit AFTER 6 ns;
  - If a driver turns off, its effect remains on t for 6 ns after it has been turned off.

9-45

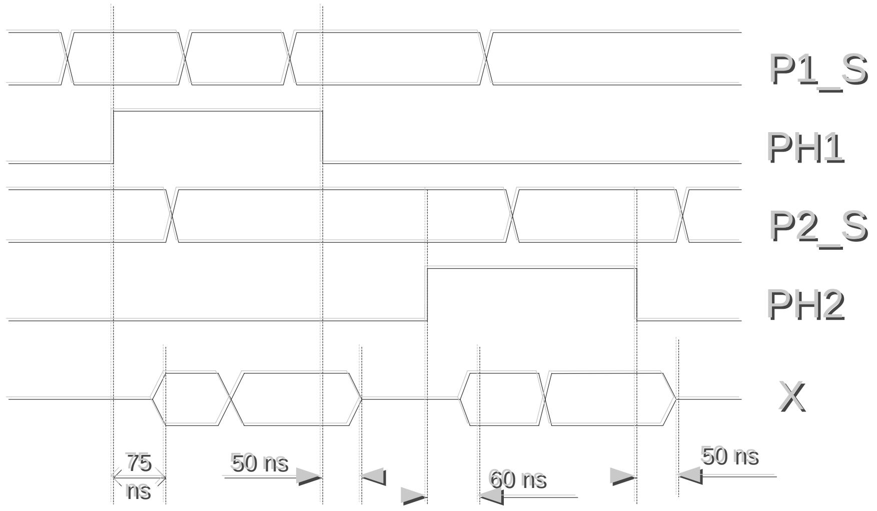
… Disconnection Delay

- **The DISCONNECTION statement is placed in the declarative part of the Architecture and applies to all assignments to this signal**
- **EXAMPLE**

```
Architecture_ DF of  Example is
Signal_              X  : WX_Vector (7 downTo 0) BUS ;
DISCONNECT    X  : WX_Vector  after  50 ns ;
Begin
  B1: Block (Ph1=`1`)
                              Signal  P1_S :  WX_Vector(7 downTo 0)  ;
               Begin
                              P1_S <= ….
                              X <= Guarded  P1_S after 75 ns;
               End Block B1   ;
  B2: Block (Ph2=`1`)
                              Signal  P2_S :  WX_Vector(7 downTo 0)  ;
               Begin
                              P2_S <= ….
                              X <= Guarded  P2_S after 60 ns;
               End Block B2  ;
END DF ;
```

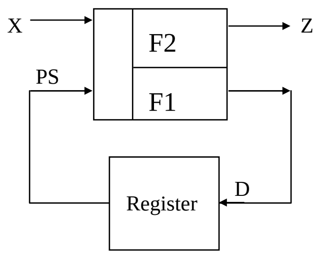
9-46

Disconnection of BUS Signals



Mealy Machine Example Using Block Statements …

```
entity Mealy_Mc is
            Port (Clk, X: in  Bit;
                          Z : out Bit);
end Mealy_Mc;
```



```
Architecture Mealy_Block of  Mealy_Mc is
    Type State is (St0, St1, St2);
    Type St_Vector is array (Natural range <>) of State ;
    Function  State_RF (X: St_Vector ) Return State is
     Begin
        Return X(X`Left);
     End State_RF  ;
    Signal PS: State_RF State REGISTER := St0;
```

… Mealy Machine Example Using Block Statements …

```
Begin
  B1: Block(not Clk`STABLE  and  Clk = `1`)
       begin
          S0:Block((PS = St0) and Guard)
               begin
                      PS <= Guarded St1 when X=`0`  else St2;
               end block S0;
           S1:Block((PS = St1) and Guard)
                begin
                     PS <= Guarded St2 when X=`0` else St0;
                end block S1;
           S2:Block((PS = St2) and Guard)
                begin
                     PS <= Guarded St1 when X=`1`  else St2;
                end block S2;
        End Block B1;
         Z <= `1` when PS =St1 and X=`0` else  `0` when PS =St1 and X=`1` else
            `0` when PS =St2 and X=`0` else  `1` when PS =St2 and X=`1` else `0` ;
 End Mealy_Block;
```

… Mealy Machine Example Using Block Statements

- **Since there are 3 concurrent signal assignments to the signal PS , it is declared as a Resolved Signal with the RF being State_RF.**
- **Signal PS is also declared to be of REGISTER kind. This Means that the signal is Guarded and Resolved and that the RF is not invoked in case all its Drivers are Turned Off (e.g. when CLK = ‘0’) in which case the signal retains its Previous Value.**
- **The outer Block statement ``B1`` defines an IMPLICIT Guard signal Which is TRUE only on the Rising Edge of the Clock.**
- **The Implicit Guard Signal ANDed with the Present State define the Guard Condition for the Nested Block Statements.**
- **ONE Inner Block Statement is assigned to each possible Present State.**
- **The State Machine Model used allows only One Driver of the Resolved Signal PS to be Active at any Given Time. Thus the `Left Attribute is used in the RF to derive the signal value forced by this driver.**

Sequence Detector Example …

- Overlapped Detection of the Sequence “1011”

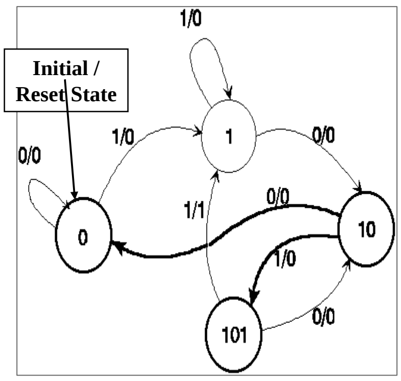
•A simple 1011 Mealy Sequence Detector

•Single Input x and A single Output z

•For x= 011011011011110111

z= 000001001001000010

```
Entity detector IS
PORT (x, clk : IN Bit; z : out Bit);
END detector;
```



… Sequence Detector Example …

```
Architecture Singular_state_machine OF Detector IS
TYPE State IS (Reset, Got1, Got10, Got101);
Type State_vector Is Array (Natural Range <>) Of State;
Function One_of (Sources : State_vector) Return State Is
BEGIN
   RETURN Sources(Sources’Left);
End One_of;
Signal PS : One_of  State  Register := Reset;
Begin
Clocking : BLOCK (Clk = '1' AND NOT Clk‘Stable)
Begin
S1: BLOCK ( PS = Reset AND GUARD )
BEGIN
PS <= GUARDED Got1 When X = '1' Else Reset;
End Block S1;
```

… Sequence Detector Example …

```
S2: Block ( PS = Got1 And Guard )
Begin
PS <= GUARDED Got10 When X = '0' Else Got1;
End Block S2;
S3: Block ( PS = Got10 And Guard )
Begin
PS <= Guarded Got101 When X = '1' Else Reset;
End Block S3;
S4: Block ( PS = Got101 And Guard)
Begin
PS <= Guarded Got1 When X = '1' Else Got10;
End Block S4;
End Block Clocking;

Z <= '1' When ( PS = Got101 And X = '1') Else '0';

End Singular_state_machine;
```

- **PS receives four concurrent assignments**
- **PS must be resolved; use *one_of* as an RF**

… Sequence Detector Example …

```
--States are: 1 = reset, 2 = got1, 3 = got10, 4 = got101
--use a signal for each state
ARCHITECTURE multiple_state_machine OF detector IS
SIGNAL s : ored_bit_vector (1 TO 4) REGISTER := "1000";
BEGIN
clocking : BLOCK (clk = '1' AND NOT clk'STABLE)
BEGIN
   s1: BLOCK (s(1) = '1' AND GUARD)
    BEGIN
            s(1) <= GUARDED '1' WHEN x = '0' ELSE '0';
            s(2) <= GUARDED '1' WHEN x = '1' ELSE '0';
    END BLOCK s1;
   s2: BLOCK (s(2) = '1' AND GUARD)
    BEGIN
            s(3) <= GUARDED '1' WHEN x = '0' ELSE '0';
            s(2) <= GUARDED '1' WHEN x = '1' ELSE '0';
    END BLOCK s2;
```

… Sequence Detector Example

```
    s3: BLOCK (s(3) = '1' AND GUARD)
     BEGIN
            s(1) <= GUARDED '1' WHEN x = '0' ELSE '0';
            s(4) <= GUARDED '1' WHEN x = '1' ELSE '0';
     END BLOCK s3;
    s4: BLOCK (s(4) = '1' AND GUARD)
     BEGIN
            s(3) <= GUARDED '1' WHEN x = '0' ELSE '0';
            s(2) <= GUARDED '1' WHEN x = '1' ELSE '0';
            z <= '1' WHEN (s(4) = '1' AND x = '1') ELSE '0';
     END BLOCK s4;
  s <= GUARDED "0000";
  END BLOCK clocking;
  END multiple_state_machine;
```

•*S <= GUARDED "0000";*
Causes removal of retained value upon last disconnection

General Mealy State Machine …

```
ENTITY detector_m IS
   PORT (x,clk : IN BIT; z : OUT BIT);
END detector_m;
ARCHITECTURE multiple_moore_machine_1 OF detector_m IS
FUNCTION oring( drivers : BIT_VECTOR) RETURN BIT IS
VARIABLE accumulate : BIT := '0';
BEGIN
FOR i IN drivers'RANGE LOOP
accumulate := accumulate OR drivers(i);
END LOOP;
RETURN accumulate;
END oring;
SUBTYPE ored_bit IS oring BIT;
TYPE ored_bit_vector IS ARRAY (NATURAL RANGE <>) OF ored_bit;
TYPE next_table IS ARRAY (1 TO 6, BIT) OF INTEGER;
TYPE out_table IS ARRAY (1 TO 6, BIT) OF BIT;
```

… General Mealy State Machine …

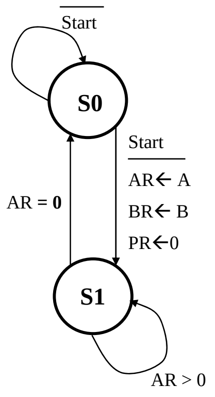
```
-- Fill in next_val, out_val, and s arrays
SIGNAL o : ored_bit REGISTER;
BEGIN
clocking : BLOCK (clk = '1' AND (NOT clk'STABLE))
BEGIN
g: FOR i IN s'RANGE GENERATE
si: BLOCK (s(i) = '1' AND GUARD)
BEGIN
s(next_val(i,'0')) <= GUARDED '1' WHEN x='0' ELSE '0';
s(next_val(i,'1')) <= GUARDED '1' WHEN x='1' ELSE '0';
o <= GUARDED out_val(i, x);
END BLOCK si;
s (i) <= GUARDED '0';
END GENERATE;
END BLOCK clocking;
z <= o;
END multiple_moore_machine_1;
```

… General Mealy State Machine

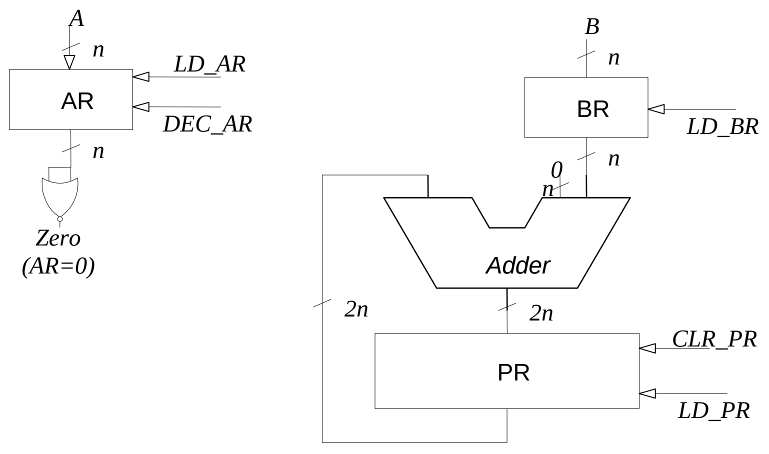
```
--The folowing tables program the general purpose Moore description--
-- -- Next States: ----- x=0, x=1 --
CONSTANT next_val : next_table := ( (1 , 2), --S1: -> S1, S2 --
(1 , 3), --S2: -> S1, S3 --
(1 , 4), --S3: -> S1, S4 --
(1 , 1), --S4: -> S1, S1 --
(5 , 6), --S5: -> S5, S6 --
(5 , 6) ); --S6: -> S5, S6 --
-- -- Output Values: ----- x=0, x=1 --
CONSTANT out_val : out_table := ( ('0' , '0'), --S1:  z=0, 0 --
('0' , '0'), --S2:  z=0, 0 --
('0' , '0'), --S3:  z=0, 0 --
('0' , '0'), --S4:  z=1, 1 --
('0' , '0'), --S5:  z=0, 0 --
('1' , '1') );--S6:  z=1, 1 --
-- -- Initial Active States: --
SIGNAL s : ored_bit_vector (1 TO 6) REGISTER := "100000"; --
```

Multiplier Design

- Design a Multiplier Circuit which Multiplies 2 Unsigned n-bit numbers A (multiplicand) & B (multiplier).
- The Product (P) is evaluated by repeated additions of the Multiplicand (B) to itself a number of times equals the Multiplier (A) value.
- Example
  1. A=3, B=4 → P = 4 + 4 +4
  2. A=0, B=4 → P = 0
  3. A=3, B=0 → P = 0 + 0 + 0
- Required Data Path Modules:
  - A-Register (n-Bits) → AR
  - B-Register (n-Bits) → BR
  - P-Register (2n-Bits) → PR
  - Adder



Data Path Design



Controller Model …

```
Entity CPath_Mult is
Port (clk, start, zero: IN Bit ;
LD_AR, LD_BR, CLR_PR, LD_PR, Dec_AR: OUT BIT);
End CPath_Mult ;
Architecture DF of CPath_Mult is
Type States is (Initial, Iterative);
Type State_Vector is Array (Natural Range <>) of States;
Function RF(V:State_Vector) Return States is
Begin
   Return V(V'Left);
end RF;
Signal PS:  RF States Register := Initial;
Begin
edge: Block(Clk='1'  and  not Clk'Stable)
  Begin
```

… Controller Model

```
S0: Block(PS= Initial and Guard)
   Begin
   PS <= Guarded Iterative when Start='1' Else  Initial;
   end Block S0;
S1: Block(PS= Iterative and Guard)
   Begin
     PS <= Guarded Iterative when Zero /='1' Else Initial;
   end Block S1;
   LD_AR  <= '1' when PS= Initial and Start='1' else '0';
   LD_BR  <= '1' when PS= Initial and Start='1' else '0';
   Clr_PR  <= '1' when PS= Initial and Start='1' else '0';
   LD_PR  <= '1' when PS=Iterative and Zero /= '1' else '0';
   DEC_AR <= '1' when PS=Iterative and Zero /= '1' else '0';
End Block edge;
End  DF;
```

Data Path Model …

```
Entity DPath_Mult is
Generic(N: Positive:= 8);
Port(LD_AR, LD_BR, CLR_PR, LD_PR,Dec_AR, Clk: in Bit ;  A, B: in Bit_Vector(N-1 DownTo
    0);    Zero: out Bit :='0'; P: out Bit_Vector(2*N-1 DownTo 0));
End DPath_Mult ;
Architecture DF of DPath_Mult is
Signal AR, BR      : Bit_Vector(N-1 DownTo 0);
Signal PR          : Bit_Vector(2*N-1 DownTo 0);
Signal ARE,BRE,PRE : Boolean:=False ;
Begin
ARE <= LD_AR='1' or DEC_AR='1' ;
BRE <= LD_BR='1' ;                              -- Inner Block (Register) Enable Signals
PRE <= LD_PR='1' or CLR_PR='1' ;
edge: Block(Clk='1' and not Clk'Stable)
Begin
```

Data Path Model …

```
AReg: Block(ARE and Guard)
  Begin
    AR <= Guarded A when LD_AR='1' else AR+(N-1 Downto 0=>'1')       when  Zero /= '1' else
    Unaffected;
    Zero <= '1' when (Bin2Int(AR)=0) else '0';
end Block AReg;
BReg: Block(BRE and Guard)
    Begin
          BR <= Guarded B;
    end Block BReg;
PReg: Block(PRE and Guard)
  Begin
    PR <= Guarded PR+ (N-1 Downto 0 => '0')&BR when LD_PR='1' else (2*N-1 Downto 0 => '0');
  end Block PReg;
End Block edge;
    P <= PR ;
End  DF;
```

+ive Edge-Triggered Shift Register with Parallel Load …

Register INPUTS In Order of Priority

- **Ena : If Ena=0, The register Cannot Change its state.**
- **LD : IF LD = 1, Data on the parallel inputs (Din) are Loaded into the Register independent of the Clock Signal (Asynchronous Load)**
- **Dir : Determines the Direction of the Shift or Rotate Operation. Dir=0 indicates a Left shift/Rotate while Dir = 1, indicates a Right Shift /Rotate.**
- **Shift Mode Signals M1 & M2**
  - M1M2 : 00 A 0 is Shifted-In
  - M1M2 : 01 A 1 is Shifted-In
  - M1M2 : 10 The Sin input is Shifted-In
  - M1M2 : 11 Rotate Operation.

… +ive Edge-Triggered Shift Register with Parallel Load …

```
Type  MVL4 is ('X', '0', '1', 'Z');
Type MVL4_Vec is Array(Natural range <>) of MVL4 ;
Type MVL4_Tab is array(MVL4 , MVL4) of MVL4;
Constant Tab_X  :   MVL4_Tab  :=
--                              'x', '0', '1', 'Z'
                             -----------------------
                             (('x', 'x', 'x', 'x'),  -- 'x'
                              ('x', '0', 'x', '0'),  -- '0'
                              ('x', 'x', '1', '1'),  -- '1'
                              ('x', '0', '1', 'z')); -- 'z'
Function WiredX (INP : MVL4_Vec) Return MVL4 is
Variable Result: MVL4:='z';-- Initialize
Begin
    For i in INP'Range Loop
      Result:= TAB_X(Result , INP(i));
    End Loop;
    Return Result;
end WiredX ;
SubType WX is   WiredX    MVL4 ;
Type WX_Vector is Array(Natural range <>) of WX ;
```

… +ive Edge-Triggered Shift Register with Parallel Load …

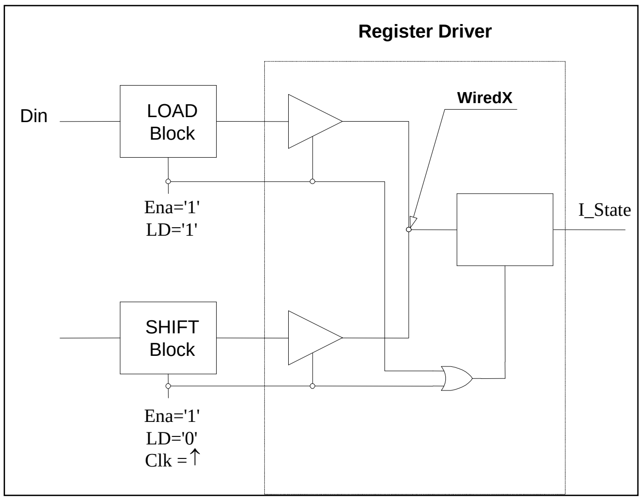
```
Entity ShiftReg is
    Port ( Ena, Ld, Clk, Dir, M1, M2 : in Bit;
           Sin : in MVL4 ;
           Din : in WX_Vector(7 downto 0);
           Q   : Out WX_Vector(7 downto 0));
END ShiftReg ;

Architecture DF of ShiftReg is
    Signal I_State : WX_Vector(7 downto 0) Register;
    signal t: bit_vector(2 downto 0);
Begin
t <=  Dir & M1 & M2 ;
Load: Block(Ena='1' and Ld='1')
        begin
            I_State <= Guarded Din;
        end block Load;
```

… +ive Edge-Triggered Shift Register with Parallel Load …

```
Shift: Block(Ena='1' and Ld='0' and Clk='1' and not  Clk`Stable)
        begin
            With t Select
             I_State <= Guarded
                  I_State(6 downto 0) & '0' When "000" ,
                  I_State(6 downto 0) & '1' When "001" ,
                  I_State(6 downto 0) & Sin When "010" ,
                  I_State(6 downto 0) & I_State(7) When "011" ,
                  '0' & I_State(7 downto 1)  When "100" ,
                  '1' & I_State(7 downto 1)  When "101" ,
                  Sin & I_State(7 downto 1)  When "110" ,
                  I_State(0)& I_State(7 downto 1)  When "111";
        end block Shift;
        Q <= I_State After 5 ns ;
End DF ;
```

… +ive Edge-Triggered Shift Register with Parallel Load



Register Driver
WiredX
Din
LOAD Block
Ena='1'
LD='1'
I_State
SHIFT Block
Ena='1'
LD='0'
Clk =↑