What is VHDL?

- A *hardware description language* that can be used to model a digital system.

- VHDL = VHSIC Hardware Description Language
- Very High Speed Integrated Circuit

- Can describe:
  - behaviour,
  - structure, and
  - timing

  of a logic circuit.

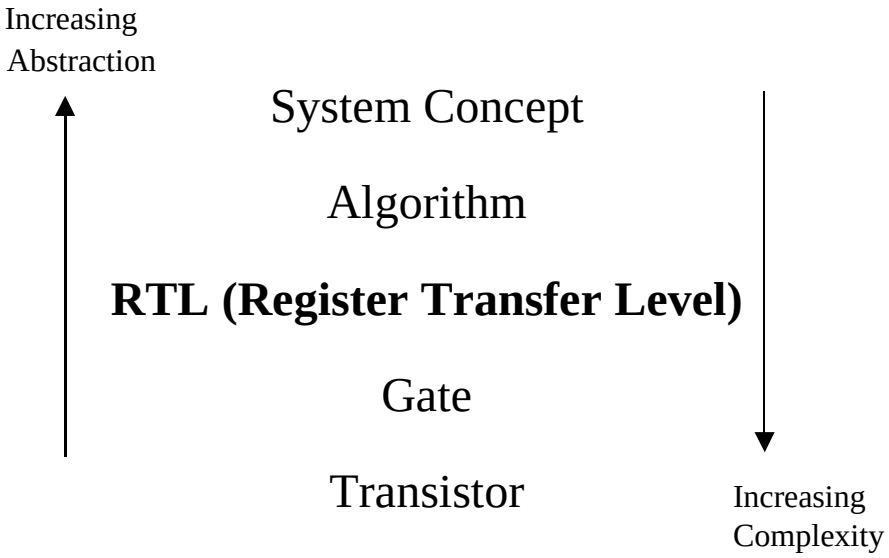
Hardware Modelling in VHDL

- VHDL is **NOT** a programming language like C or Java.

- It is used to model the physical hardware used in digital systems.

- Therefore you must always think about the hardware you wish to implement when designing systems using VHDL.

Abstraction Hierarchy



Data Objects

- A *data object* holds a value of a specified type.

- The data objects that can be synthesized directly in hardware are:

1. **Signal**: Represents a physical wire in a circuit. It holds a list of values which includes its current value and a set of possible future values.

2. **Variable**: Used to hold results of computations. It does not necessarily represent a wire in a circuit.

3. **Constant**: Contains a value that cannot be changed. It is set before the beginning of a simulation.

Predefined Data Types

- Standard logic type: STD_LOGIC, STD_LOGIC_VECTOR (Can hold 0, 1, Z, and −−.)
- Bit type: BIT, BIT_VECTOR
- Integer type: INTEGER
- Floating−point type: REAL
- Physical type: TIME
- Enumeration type: BOOLEAN, CHARACTER

- To use the STD_LOGIC and STD_LOGIC_VECTOR types, the *std_logic_1164* package must be included in the VHDL design file.

- We can define our own data types. This is especially useful when designing finite−state machines (FSMs).

- An object declaration is used to declare an object, its type, its class, and optionally to assign it a value.

Object Declaration Examples

- Signal declarations:

```
SIGNAL sresetn : STD_LOGIC;
SIGNAL address : STD_LOGIC_VECTOR(7
    downto 0);
```

- Variable declarations:

```
VARIABLE index : INTEGER range 0 to 99 :=
    20;
VARIABLE memory : BIT_MATRIX(0 to 7, 0
    to 1023);
```

- Constant declarations:

```
CONSTANT cycle_time : TIME := 100 ns;
CONSTANT cst : UNSIGNED(3 downto 0);
```

Operators

	Operator Class	Operator
Highest Precedence	Miscellaneous	**, ABS, NOT
	Multiplication	*, /, MOD, REM
	Unary Arithmetic (Sign)	+, −
	Addition	+, −, &
	Shift/Rotate	sll, srl, sla, sra, rol, ror
	Relational	=, /=, <, <=, >, >=
Lowest Precedence	Logical	and, or, nand, nor, xor, xnor

- The individual operators in each class have the same precedence.

VHDL Design Entity

Design Entity

Entity Declaration

− Specifies the interface of entity to the outside world.

− Has a name.

− Includes PORT statement which specifies the entity's input and output signals (ports)

− Ports can have different modes:
IN
OUT
INOUT
BUFFER

Architecture

− Provides circuit details for an entity

− General form:

```
ARCHITECTURE arch_name OF entity_name IS
        Signal declarations
        Constant declarations
        Type declarations
        Component declarations
BEGIN
        Component instantiations
        Concurrent assignment statements
        Process statements
END arch_name
```

Concurrent Assignment Statements

- A concurrent assignment statement is used to assign a value to a signal in an architecture body.

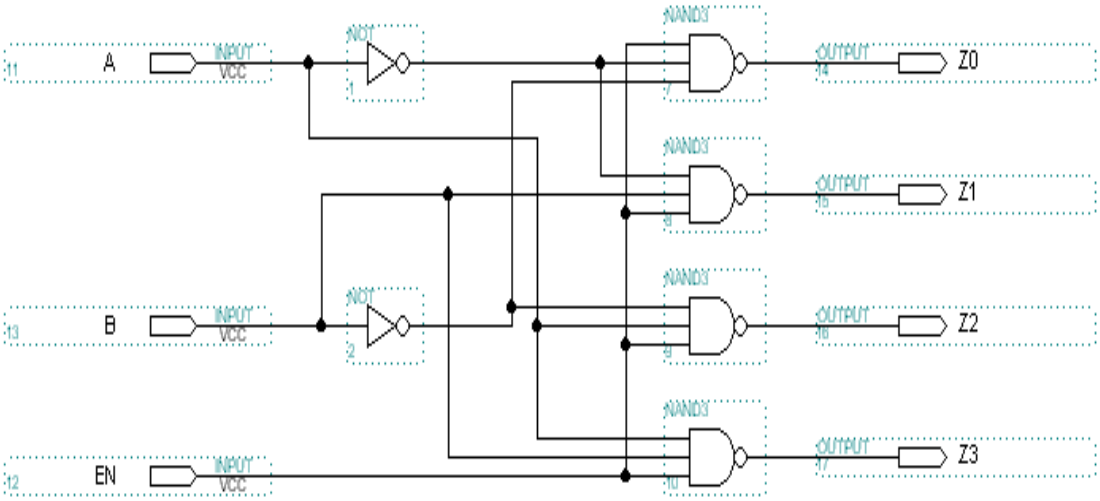
- Used to model combinational circuits.

- The order in which these statements occur does not affect the meaning of the code.

2x4 Decoder Example


```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY decoder2x4 IS
      PORT (A, B, EN : IN STD_LOGIC;
            Z : OUT STD_LOGIC_VECTOR(3 downto 0));
END decoder2x4;
−−this is a comment
```

Architecture Body of 2x4 Decoder




```
ARCHITECTURE dec_df OF decoder2x4 IS
    SIGNAL ABAR, BBAR : STD_LOGIC;
BEGIN
    −−Order of concurrent signal assignment
    statements is not important.
    Z(3) <= not (A and B and EN);
    Z(0) <= not (ABAR and BBAR and EN);
    BBAR <= not B;
    Z(2) <= not (A and BBAR and EN);
    ABAR <= not A;
    Z(1) <= not (ABAR and B and EN);
END dec_df;
```

Sequential Assignment Statements

- Sequential assignment statements assign values to signals and variables. The order in which these statements appear can affect the meaning of the code.

- Can be used to model combinational circuits and sequential circuits.

- Require use of the PROCESS statement.

- Include three variants: IF statements, CASE statements, and LOOP statements.

2x4 Decoder Revisited

```
ARCHITECTURE dec_seq OF decoder2x4 IS
BEGIN
    PROCESS(A, B, EN)  −−Sensitivity list
        VARIABLE ABAR, BBAR : STD_LOGIC;
    BEGIN
        −−Variable values assigned immediately.
        ABAR := not A;
        BBAR := not B;
        IF (EN = ’1’) THEN
            Z(3) <= not(A and B);
            Z(2) <= not(A and BBAR);
            Z(1) <= not(ABAR and B);
            Z(0) <= not(ABAR and BBAR);
        ELSE
            Z <= "1111";
        END IF;
    END PROCESS;
END dec_seq;
```

IF and CASE Statements

- IF and CASE statements are used to model multiplexers, decoders, encoders, and comparators.

- Can only be used in a PROCESS.

Modelling a 4−1 Multiplexer

Using an IF statement:

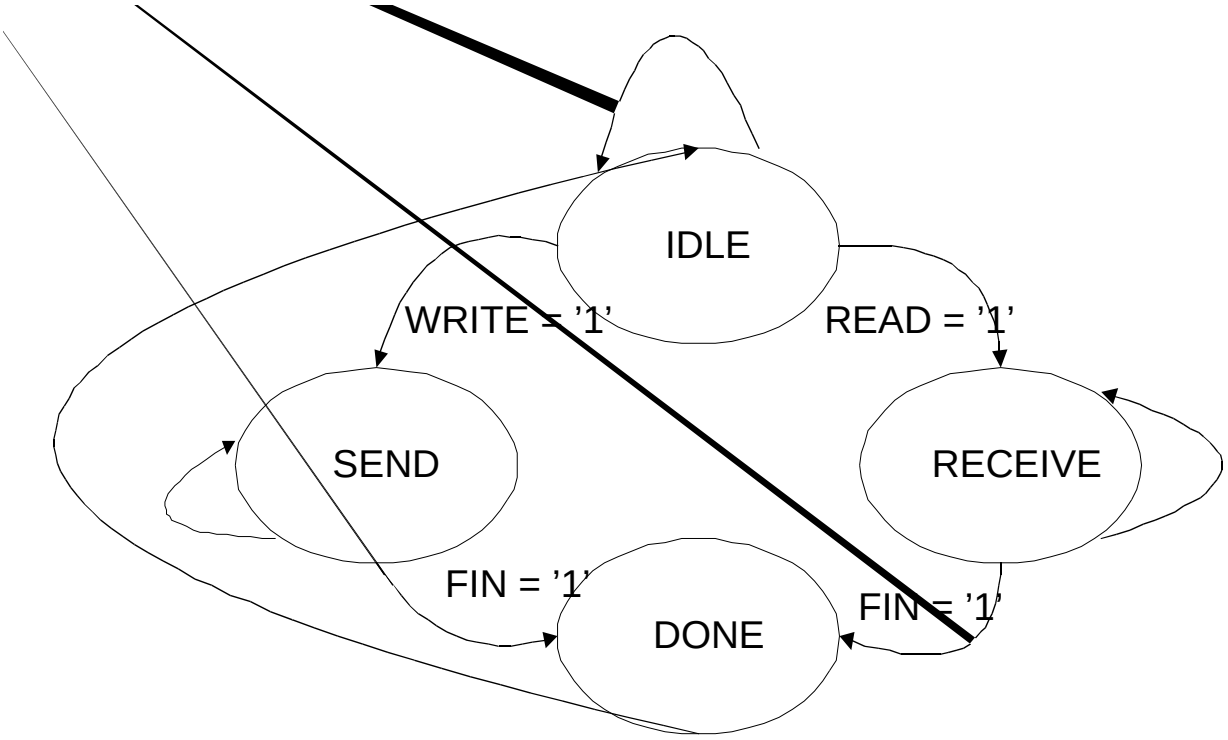
```
PROCESS (Sel, A, B, C, D)
BEGIN
        IF (Sel = "00") THEN
                Y <= A;
        ELSIF (Sel = "01")
THEN
                Y <= B;
        ELSIF (Sel = "10")
THEN
                Y <= C;
        ELSE
                Y <= D;
        END IF;
END PROCESS;
```

Using a CASE statement:

```
PROCESS (Sel, A, B, C, D)
BEGIN
        CASE Sel IS
                WHEN "00" => Y <= A;
                WHEN "01" => Y <= B;
                WHEN "10" => Y <= C;
                WHEN "11" => Y <= D;
                WHEN OTHERS =>Y
<=A;
        END CASE;
END PROCESS;
```

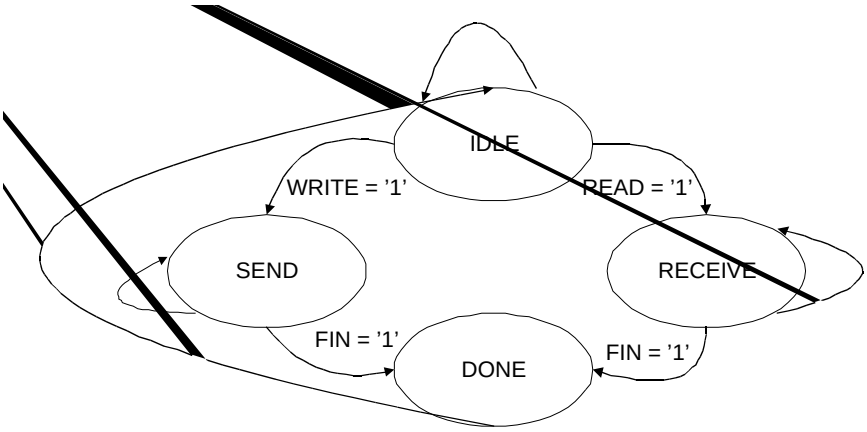
- Can also model multiplexers with WHEN/ELSE clause and WITH/SELECT clause. These can only be used outside of a PROCESS.

FSM Example



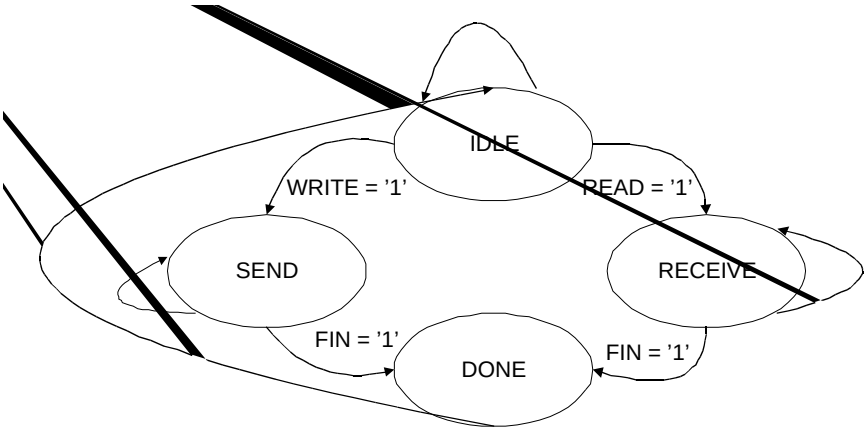

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Controller IS
      PORT ( Write, Read, Fin      : IN STD_LOGIC;
                  Busy, Snd, Rec, Dn : OUT STD_LOGIC;
                  Clock              : IN STD_LOGIC;
                  Reset              : IN STD_LOGIC);
END Controller;
```

FSM Example Continued




```
ARCHITECTURE My_FSM OF Controller IS
    TYPE State_Type IS (Idle, Send, Receive,
                                    Done);
    SIGNAL State : State_Type;

BEGIN
    PROCESS(Reset, Clock)
    BEGIN
        IF Reset='1' THEN
            State <= Idle;
        ELSIF(Clock'EVENT and Clock='1') THEN
            CASE State IS
                WHEN Idle =>
                    IF Read='1' THEN
                        State <= Receive;
                    ELSIF Write='1' THEN
                        State <= Send;
                    ELSE
                        State <= Idle;
                    END IF;
```

FSM Example Continued




```
                    WHEN Receive =>
                        IF Fin='1' THEN
                            State <= Done;
                        END IF;

                    WHEN Send =>
                        IF Fin='1' THEN
                            State <= Done;
                        END IF;

                    WHEN Done =>
                        State <= Idle;
                END CASE;
            END IF;
        END PROCESS;

        Busy <= '0' WHEN State = Idle ELSE '1';
        Rec  <= '1' WHEN State = Receive ELSE '0';
        Snd  <= '1' WHEN State = Send ELSE '0';
        Dn   <= '1' WHEN State = Done ELSE '0';

END My_FSM;
```

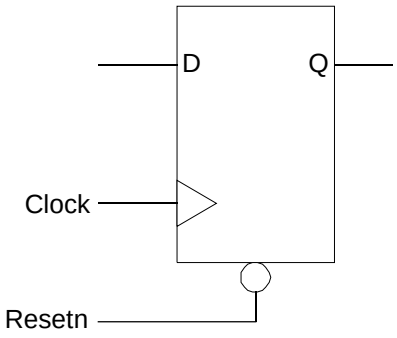
Behavioural vs. Structural Modelling

- With VHDL, we can describe the behaviour of simple circuit building blocks and then use these to build up the structure of a more complex circuit.

- Behavioural modelling is useful because it allows the designer to build a logic circuit without having to worry about the low−level details.

- Structural modelling is useful because it tells the synthesis tools exactly how to construct a desired circuit.

Behavioural Model of a D Flip−Flop

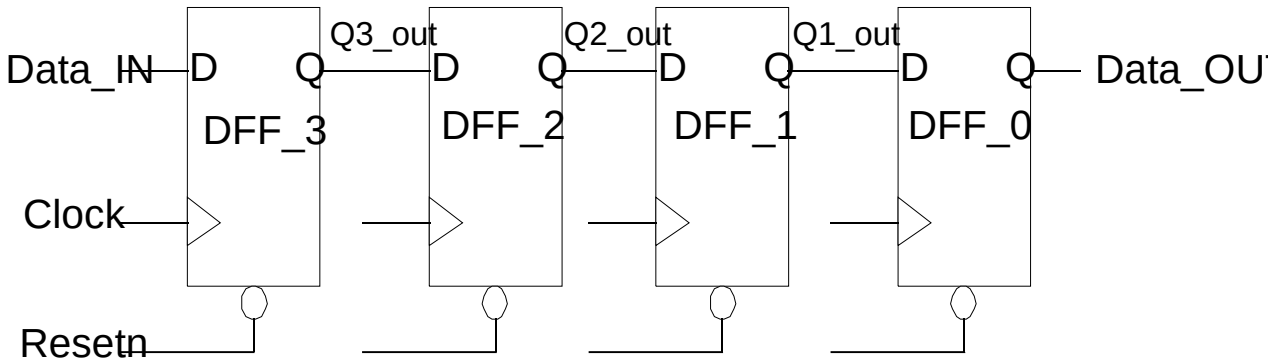



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY my_dff IS
      PORT ( D : IN  STD_LOGIC;
                  Q : OUT STD_LOGIC;
                  Clock : IN  STD_LOGIC;
                  Resetn: IN   STD_LOGIC);
END my_dff;

ARCHITECTURE Behaviour OF my_dff IS
BEGIN
      PROCESS(Clock, Resetn)
      BEGIN
            IF Resetn='0' THEN
                  Q <= '0';
            ELSIF (Clock'EVENT AND Clock='1') THEN
                  Q <= D;
            END IF;
      END PROCESS;
END Behaviour;
```

Behavioural Model of a 4−Bit Shift Register

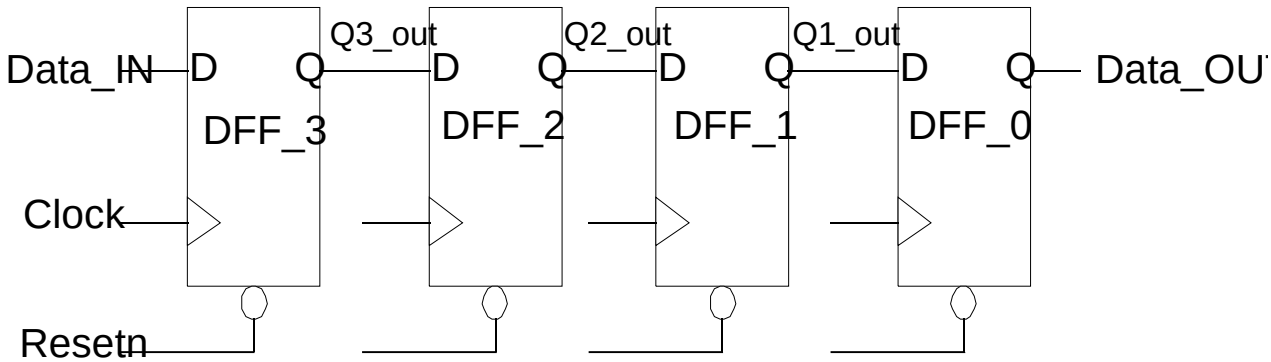
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift_reg_behav IS
      PORT ( Data_IN : IN STD_LOGIC;
                  Data_OUT : OUT STD_LOGIC;
                  Clock, Resetn : IN STD_LOGIC);
END shift_reg_behav;

ARCHITECTURE Behaviour OF shift_reg_behav IS
      SIGNAL Shift : STD_LOGIC_VECTOR(3 downto
            0);
BEGIN
      PROCESS(Clock, Resetn)
      BEGIN
            IF Resetn='0' THEN
                  Shift <= "0000";
            ELSIF (Clock'EVENT AND Clock='1') THEN
                  Shift(3) <= Data_IN;
                  --shift data to the right
                  Shift(2 downto 0) <= Shift(3 downto 1);
            END IF;
      END PROCESS;

      Data_OUT <= Shift(0);

END Behaviour;
```

Structural Model of a 4−Bit Shift Register




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift_reg_struct IS
      PORT (Data_IN : IN STD_LOGIC;
                  Data_OUT: OUT STD_LOGIC;
                  CLK, RESN : IN STD_LOGIC);
END shift_reg_struct;
```

Structural Model of a 4−Bit Shift Register Cont.




```
ARCHITECTURE Structure OF shift_reg_struct IS
      COMPONENT my_dff
            PORT (    D : IN  STD_LOGIC;
                      Q : OUT   STD_LOGIC;
                      Clock : IN  STD_LOGIC;
                      Resetn: IN       STD_LOGIC);
      END COMPONENT;

      SIGNAL Q3_out, Q2_out, Q1_out : STD_LOGIC;

BEGIN
      DFF_3 : my_dff PORT MAP (Data_IN, Q3_out, CLK, RESN);

      DFF_2 : my_dff PORT MAP (Q3_out, Q2_out, CLK, RESN);

      DFF_1 : my_dff PORT MAP (D=>Q2_out, Q=>Q1_out,
            Clock=>CLK, Resetn=>RESN);

      DFF_0 : my_dff PORT MAP (D=>Q1_out, Q=>Data_OUT,
            Clock=>CLK, Resetn=>RESN);

END Structure;
```

Using Altera's Library of Parameterized Modules (LPMs)

- Altera MAX+plus II has numerous predefined circuit building blocks in its LPMs.

- These libraries include everything from full−adders to ROMs.

Altera LPM Flip−Flop

- Must include library and use:

```
LIBRARY lpm;
USE lpm.lpm_components.all;

COMPONENT lpm_ff
  GENERIC (LPM_WIDTH: POSITIVE;
    LPM_AVALUE: STRING := "UNUSED";
    LPM_FFTYPE: STRING := "FFTYPE_DFF";
    LPM_TYPE: STRING := "L_FF";
    LPM_SVALUE: STRING := "UNUSED";
    LPM_HINT: STRING := "UNUSED");
  PORT (data: IN STD_LOGIC_VECTOR(LPM_WIDTH−1
            DOWNTO 0);
    clock: IN STD_LOGIC;
    enable: IN STD_LOGIC := '1';
    sload: IN STD_LOGIC := '0';
    sclr: IN STD_LOGIC := '0';
    sset: IN STD_LOGIC := '0';
    aload: IN STD_LOGIC := '0';
    aclr: IN STD_LOGIC := '0';
    aset: IN STD_LOGIC := '0';
    q: OUT STD_LOGIC_VECTOR(LPM_WIDTH−1 DOWNTO
      0));
END COMPONENT;
```

Altera LPM Flip−Flop Device Description

Ports:

INPUTS

Port Name	Required	Description	Comments
data[]	No	T-type flipflop: Toggle enable D-type flipflop: Data input	Input port LPM_WIDTH wide. If the data[] input is not used, at least one of the aset, aclr, sset, or sclr ports must be used. Unused data inputs default to GND.
clock	Yes	Positive-edge-triggered Clock.	
enable	No	Clock Enable input.	Default = 1.
sclr	No	Synchronous Clear input.	If both sset and sclr are used and both are asserted, sclr is dominant. The sclr signal affects the output q[] values before polarity is applied to the ports.
sset	No	Synchronous set input.	Sets q outputs to the value specified by LPM_SVALUE, if that value is present, or sets the q outputs to all 1's. If both sset

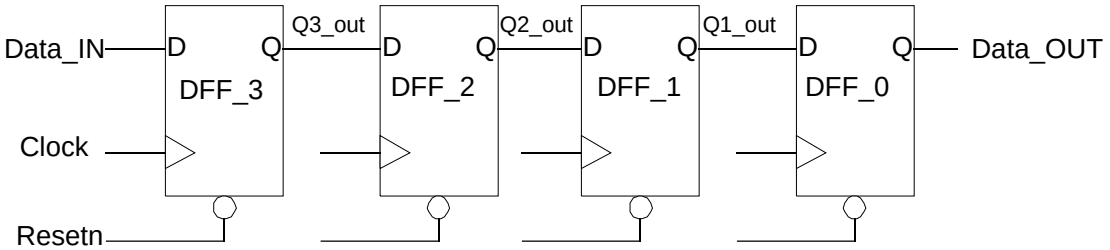
OUTPUTS

Port Name	Required	Description	Comments
q[]	Yes	Data output from D or T flipflops.	Output port LPM_WIDTH wide.

Parameters:

Parameter	Type	Required	Description
LPM_WIDTH	Integer	Yes	Width of the data[] and q[] ports.
LPM_AVALUE	Integer	No	Constant value that is loaded when aset is high. If omitted, defaults to all 1's. The LPM_AVALUE parameter is limited to a maximum of 32 bits.
LPM_SVALUE	Integer	No	Constant value that is loaded on the rising edge of clock when sset is high. If omitted, defaults to all 1's.
LPM_FFTYPE	String	No	Values are "DFF", "TFF", and "UNUSED". Type of flipflop. If omitted, the default is "DFF". When the LPM_FFTYPE parameter is set to "DFF", the sload port is ignored.
LPM_HINT	String	No	Allows you to specify Altera-specific parameters in VHDL Design Files. The default is "UNUSED".
LPM_TYPE	String	No	Identifies the LPM entity name in VHDL Design Files.

Structural Description of Shift Register Using Altera's Flip−Flop LPM




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY shift_reg_struct2 IS
PORT (
      Data_IN : IN STD_LOGIC_VECTOR(0 downto 0);
      Data_OUT: OUT STD_LOGIC_VECTOR(0
            downto 0);
      CLK, RESN : IN STD_LOGIC);
END shift_reg_struct2;
```

Structural Description of Shift Register Using Altera's Flip−Flop LPM Cont.

```
ARCHITECTURE Structure OF shift_reg_struct2 IS
    SIGNAL Q3_out, Q2_out, Q1_out :
            STD_LOGIC_VECTOR(0 downto 0);
    SIGNAL Reset_Internal : STD_LOGIC;

BEGIN
    Reset_Internal <= not RESN;

    DFF_3 : lpm_ff GENERIC MAP (LPM_WIDTH=>1,
                LPM_FFTYPE=>"DFF")
            PORT MAP (data=>Data_IN, q=>Q3_out,
                clock=>CLK, aclr=>Reset_Internal);

    DFF_2 : lpm_ff GENERIC MAP (LPM_WIDTH=>1,
                LPM_FFTYPE=>"DFF")
            PORT MAP (data=>Q3_out, q=>Q2_out,
                clock=>CLK, aclr=>Reset_Internal);

    DFF_1 : lpm_ff GENERIC MAP (LPM_WIDTH=>1,
                LPM_FFTYPE=>"DFF")
            PORT MAP (data=>Q2_out, q=>Q1_out,
                clock=>CLK, aclr=>Reset_Internal);

    DFF_0 : lpm_ff GENERIC MAP (LPM_WIDTH=>1,
                LPM_FFTYPE=>"DFF")
            PORT MAP (data=>Q1_out, q=>Data_OUT,
                clock=>CLK, aclr=>Reset_Internal);

END Structure;
```