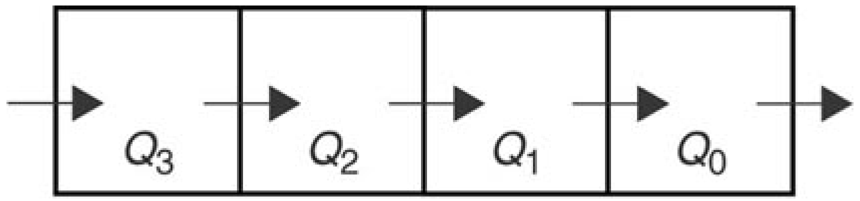
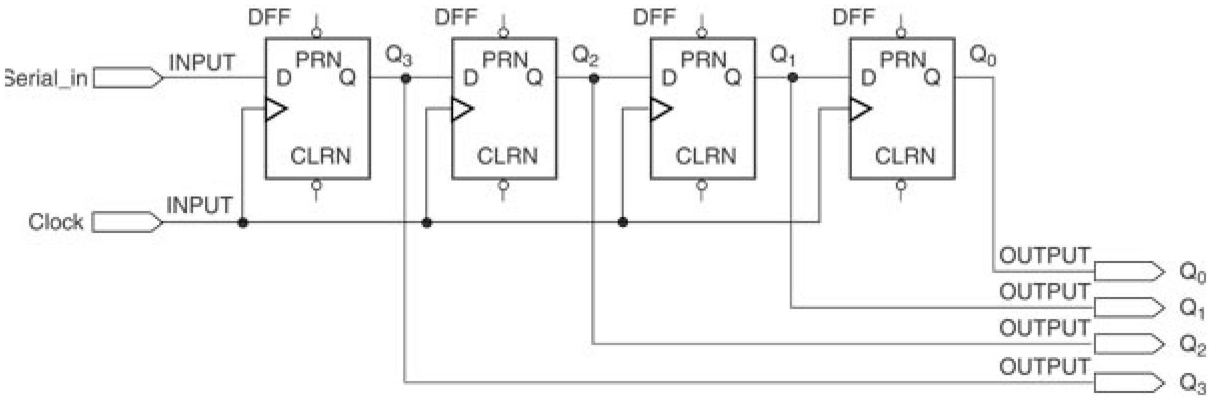
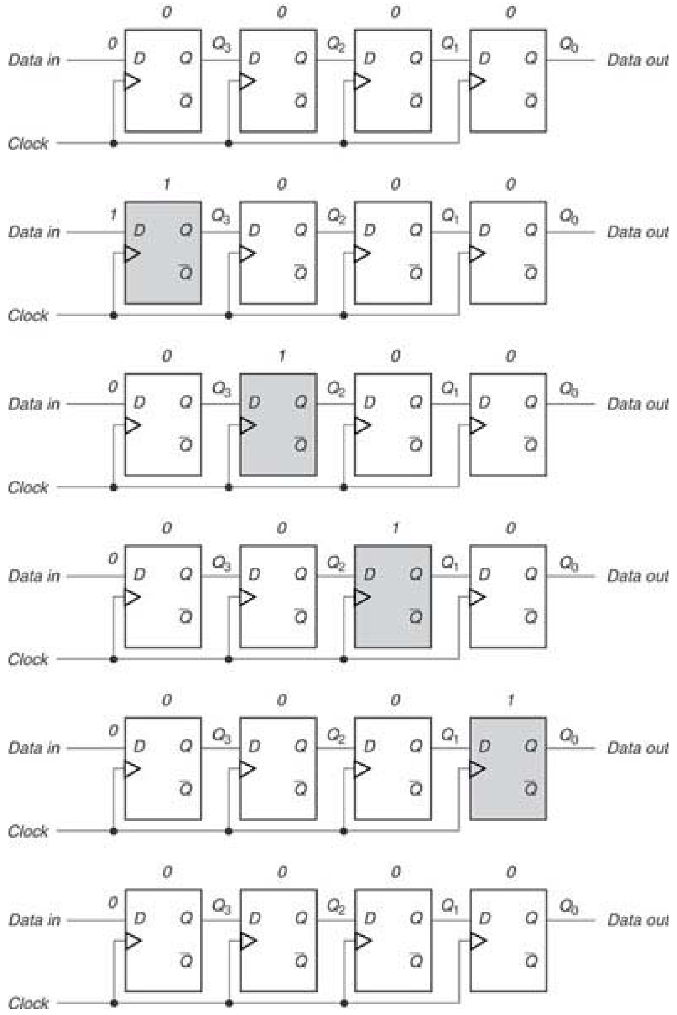
Class 11
Shift Registers

Serial Shift Register

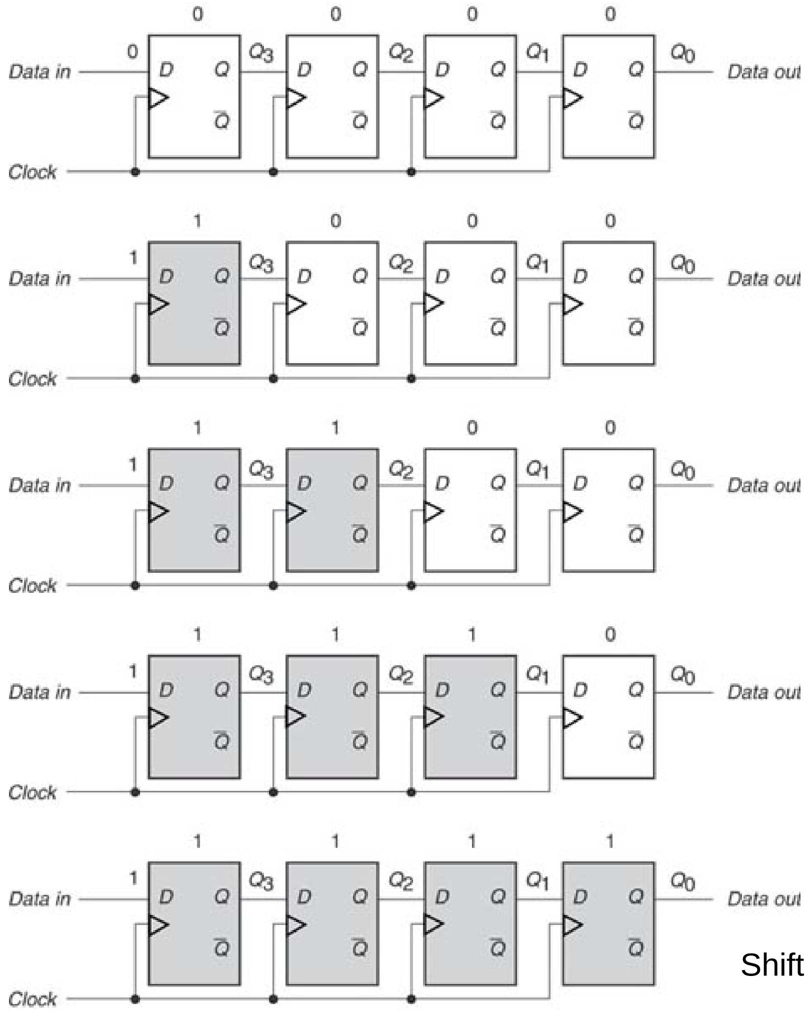




Serial Shift Register (Cont.)



Shift a “1”



Shift with 1’s

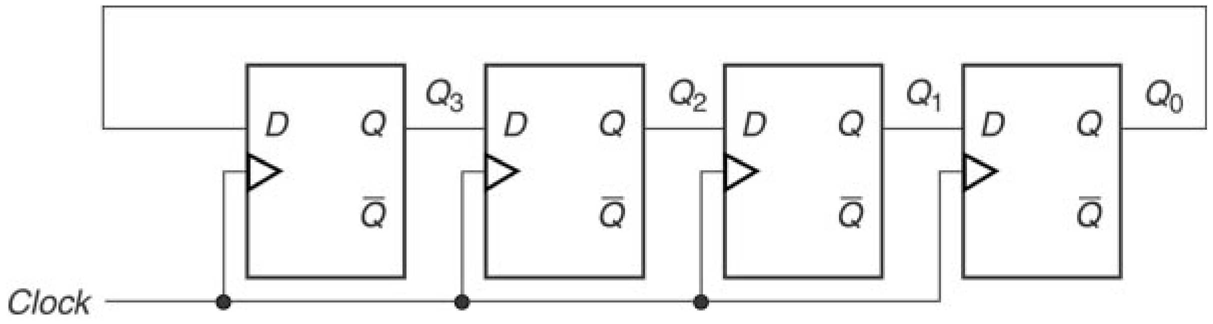
Universal Shift Register

S_1	S_0	Function	D_3	D_2	D_1	D_0
0	0	Hold	Q_3	Q_2	Q_1	Q_0
0	1	Shift Right	RSI	Q_3	Q_2	Q_1
1	0	Shift Left	Q_2	Q_1	Q_0	LSI
1	1	Load	P_3	P_2	P_1	P_0

RSI: Right-Shift Input
LSI: Left-Shift Input

Ring Counter

- A serial shift register with feedback from the output of the last flip-flop to the input of the first.



Shift Register – Structural Design

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
LIBRARY altera;
USE altera.maxplus2.ALL;

ENTITY srg4strc IS
  PORT(
    serial_in, clk : IN        STD_LOGIC;
    qo : BUFFER                STD_LOGIC_VECTOR(3
downto 0));
END srg4strc;

ARCHITECTURE right_shift of srg4strc IS
  COMPONENT dff
    PORT (d   : IN STD_LOGIC;
      clk : IN STD_LOGIC;
      q   : OUT STD_LOGIC);
  END COMPONENT;
BEGIN
  flip_flop_3: dff PORT MAP(serial_in, clk, qo(3));
  dffs:  FOR i IN 2 downto 0 GENERATE
    flip_flops_2_to_0: dff PORT MAP (qo(i+1), clk, qo(i));
  END GENERATE;
END right_shift;
```

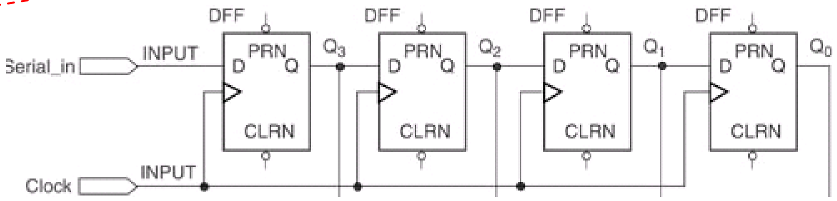
Include Altera packages for D Flip-Flop

BUFFER can be a feedback to the circuit

Declare the DFF component

Flip Flop 3

Flip Flop 2-0



Shift Register – Dataflow Design / Behavioral Design

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY srg4dflw IS
  PORT(
      serial_in, clk : IN STD_LOGIC;
      q: BUFFER STD_LOGIC_VECTOR(3 downto 0));
END srg4dflw;

ARCHITECTURE right_shift  OF srg4dflw IS
  SIGNAL d : STD_LOGIC_VECTOR(3 downto 0);
BEGIN
  PROCESS (clk)
  BEGIN
    -- Define a 4-bit D flip-flop
    IF (clk'EVENT and clk = '1‘) THEN
      q <= d;
    END IF;
  END PROCESS;
  d <= serial_in & q(3 downto 1);
END right_shift;
```

Dataflow design: describe a design entity in terms of the Boolean relationships between different parts of the circuit.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY srg4behv IS
  PORT(
      serial_in, clk : IN STD_LOGIC;
      q: BUFFER STD_LOGIC_VECTOR(3 downto 0));
END srg4behv;

ARCHITECTURE right_shift  OF srg4behv IS

BEGIN
  PROCESS (clk)
  BEGIN
    -- Define a 4-bit D flip-flop
    IF (clk'EVENT and clk = '1‘) THEN
      q <= serial_in & q(3 downto 1);
    END IF;
  END PROCESS;
END right_shift;
```

Behavioral design: describe a design entity in terms of the behavior of the circuit.

Shift Registers of Generic Width

Use the GENERIC clause to define a parameter

Default value is 4

Clear every bit of q() to ‘0’

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY srg4dflw IS
  GENERIC (width: POSITIVE := 4)
  PORT(
    serial_in, clk, clear : IN STD_LOGIC;
    q: BUFFER
      STD_LOGIC_VECTOR(width-1 downto 0));
END srg4dflw;
ARCHITECTURE right_shift  OF srg4dflw IS
 SIGNAL d : STD_LOGIC_VECTOR(3 downto 0);
BEGIN
  PROCESS (clk)
  BEGIN
   IF (clear = ‘0’) THEN
    q <= (others => ‘0’); -- clear every bit of q() to ‘0’
   ELSIF (clk'EVENT and clk = '1‘) THEN
    q <= d;
   END IF;
  END PROCESS;
  d <= serial_in & q(width-1 downto 1);
END right_shift;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;
ENTITY srg4dflw IS
  GENERIC (width: POSITIVE := 4)
  PORT(
    serial_in, clk, clear : IN STD_LOGIC;
    q: BUFFER
      STD_LOGIC_VECTOR(width-1 downto 0));
END srg4dflw;

ARCHITECTURE right_shift  OF srg4dflw IS
BEGIN
  PROCESS (clk)
  BEGIN
   IF (clear = ‘0’) THEN
    q <= CONV_STD_LOGIC_VECTOR(0, width);
   ELSIF (clk'EVENT and clk = '1‘) THEN
    q <= d;
   END IF;
  END PROCESS;
  d <= serial_in & q(width-1 downto 1);
END right_shift;
```

Possible Design Errors in PROCESS

- In VHDL, a PROCESS statement is *concurrent*, but statements inside the PROCESS are sequential.
  - Anything described by a PROCESS acts like a separate component in a design entity.
- Possible design errors:
  - Only one instance of the EVENT express (e.g., clk’EVENT and clk=‘1’) is allowed in a PROCESS statement.
  - No other port, signal, or variable is allowed to be included with the expression that evaluates the clock.
  - For the statements in a process, it is only possible to assign one value to a port, variable, or signal for each time the process executes.

Possible Design Errors in PROCESS (Cont.)

- Only one instance of the EVENT express (e.g., clk'EVENT and clk='1') is allowed in a PROCESS statement.

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (load='1') THEN
         q <= p;
      END IF;
   END IF;
   IF (clk'EVENT and clk='1') THEN
      IF (count_enable='1') THEN
         q <= q+1;
      END IF;
   END IF;
END PROCESS;
```

Illegal syntax: more than one clock per process

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (load='1') THEN
         q <= p;
      ELSIF (count_enable='1') THEN
         q <= q+1;
      END IF;
   END IF;
END PROCESS;
```

Legal syntax

Possible Design Errors in PROCESS (Cont.)

- No other port, signal, or variable is allowed to be included with the expression that evaluates the clock.

```
PROCESS(clk)
BEGIN
   IF (clk’EVENT and clk=‘1’ and load=‘1’) THEN
       q <= p;
   ELSE
      q <= q+1;
   END IF;
END PROCESS;
```

Illegal syntax: load evaluated in same statement as clk

```
PROCESS(clk)
BEGIN
   IF (clk’EVENT and clk=‘1’) THEN
      IF (load=‘1’) THEN
         q <= p;
      ELSE
         q <= q+1;
      END IF;
   END IF;
END PROCESS;
```

Legal syntax

Possible Design Errors (Cont.)

- For the statements in a process, it is only possible to assign one value to a port, variable, or signal for each time the process executes.

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (count_enable = '1') THEN
         q <= q+1;
      END IF;
      IF (load = '1') THEN
         q<= p;
      END IF;
      IF (clear = '0') THEN
         q <= (others =>'0');
      END IF;
   END IF;
END PROCESS;
```

Ambigous (but not illegal) syntax: q assigned more than once in a process. May have an unexpected result.

$\Rightarrow$

```
PROCESS(clk)
BEGIN
   IF (clk'EVENT and clk='1') THEN
      IF (count_enable = '1') THEN
         q <= q+1;
      ELSIF (load = '1') THEN
         q<= p;
      ELSIF (clear = '0') THEN
         q <= (others =>'0');
      END IF;
   END IF;
END PROCESS;
```

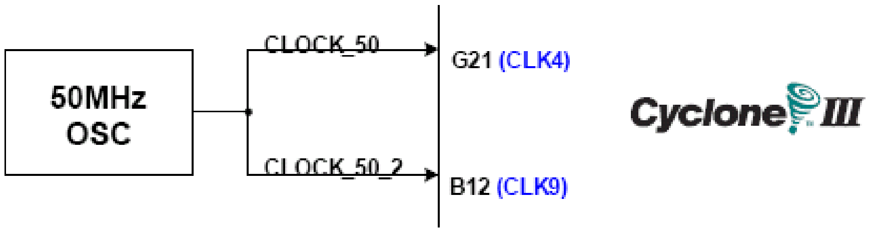
Legal syntax

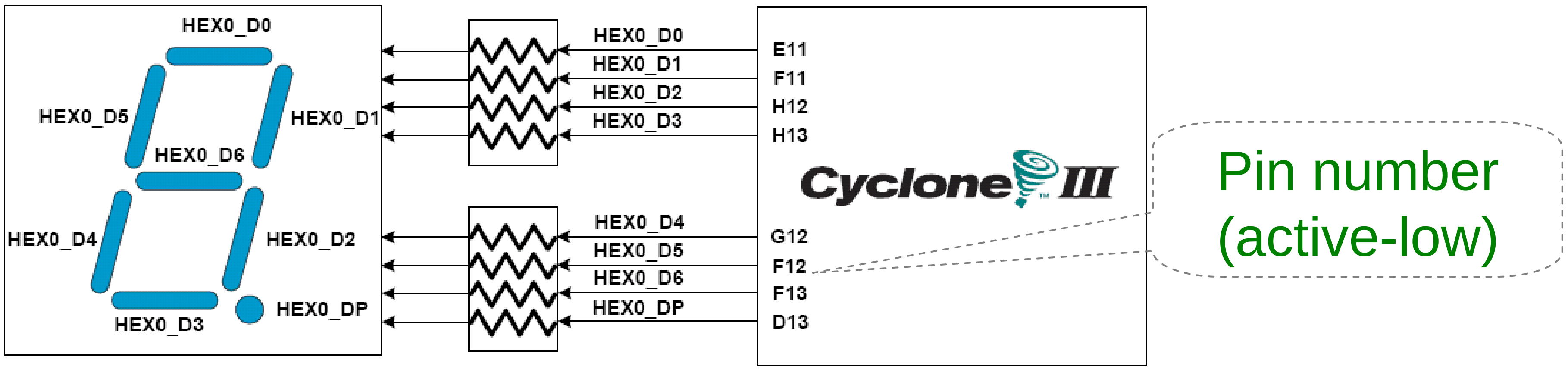
Lab 11

- Design a universal shift register with feedback from the output. The shift frequency is 5Hz.

PB_2	**PB_1**	**Function**	**LED_9**	**LED_8**	**…**	**LED_1**	**LED_0**
1	1	Hold	LED_9	LED_8	…	LED_1	LED_0
1	0	Shift Right	LED_0	LED_9	…	LED_2	LED_1
0	1	Shift Left	LED_8	LED_7	…	LED_0	LED_9
0	0	Load	SW_9	SW_8		SW_1	SW_0

7-Segment Displays & DE0 – External Clock





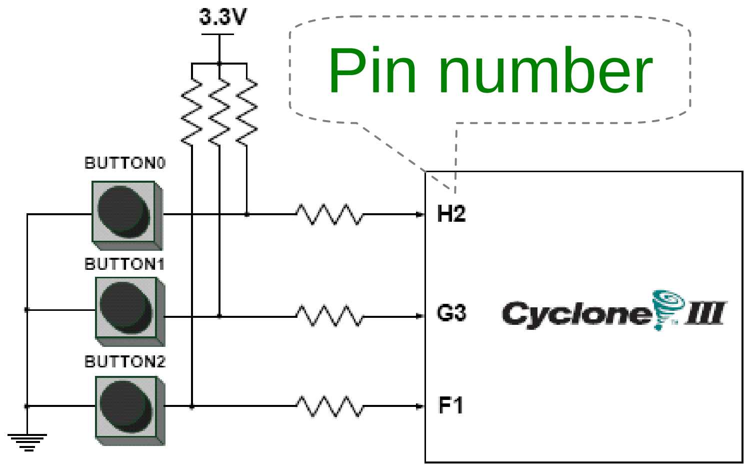
Signal Name	FPGA Pin No.
HEX0_D[0]	PIN_E11
HEX0_D[1]	PIN_F11
HEX0_D[2]	PIN_H12
HEX0_D[3]	PIN_H13
HEX0_D[4]	PIN_G12
HEX0_D[5]	PIN_F12
HEX0_D[6]	PIN_F13
HEX0_DP	PIN_D13

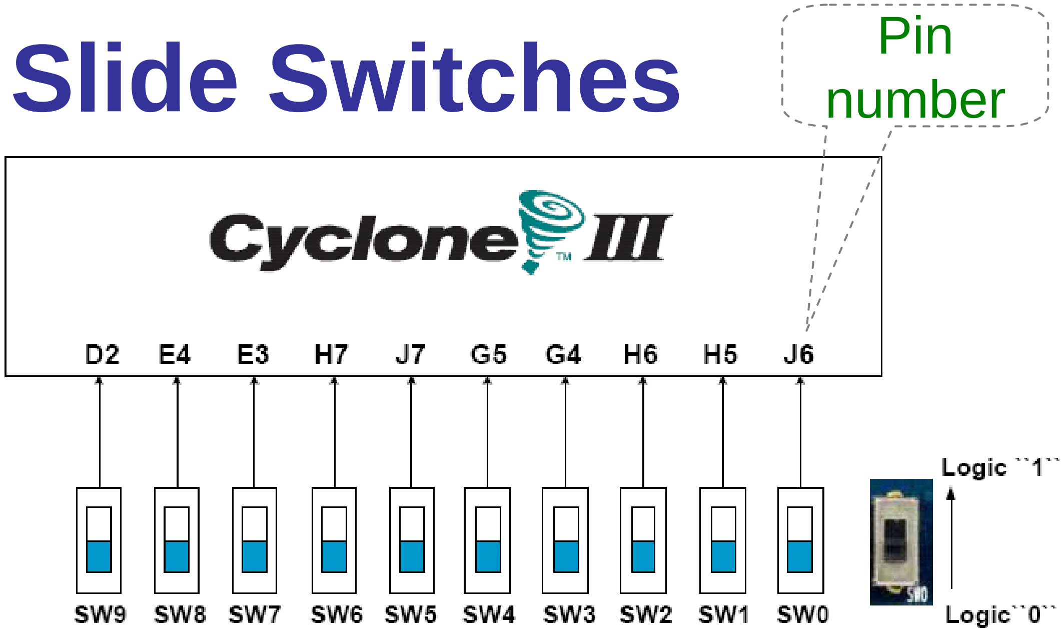
HEX1_D[0]	PIN_A13
HEX1_D[1]	PIN_B13
HEX1_D[2]	PIN_C13
HEX1_D[3]	PIN_A14
HEX1_D[4]	PIN_B14
HEX1_D[5]	PIN_E14
HEX1_D[6]	PIN_A15
HEX1_DP	PIN_B15

HEX2_D[0]	PIN_D15
HEX2_D[1]	PIN_A16
HEX2_D[2]	PIN_B16
HEX2_D[3]	PIN_E15
HEX2_D[4]	PIN_A17
HEX2_D[5]	PIN_B17
HEX2_D[6]	PIN_F14
HEX2_DP	PIN_A18

HEX3_D[0]	PIN_B18
HEX3_D[1]	PIN_F15
HEX3_D[2]	PIN_A19
HEX3_D[3]	PIN_B19
HEX3_D[4]	PIN_C19
HEX3_D[5]	PIN_D19
HEX3_D[6]	PIN_G15
HEX3_DP	PIN_G16

Pushbutton and Slide Switches





3 Pushbutton switches:
Not pressed → Logic High
Pressed → Logic Low

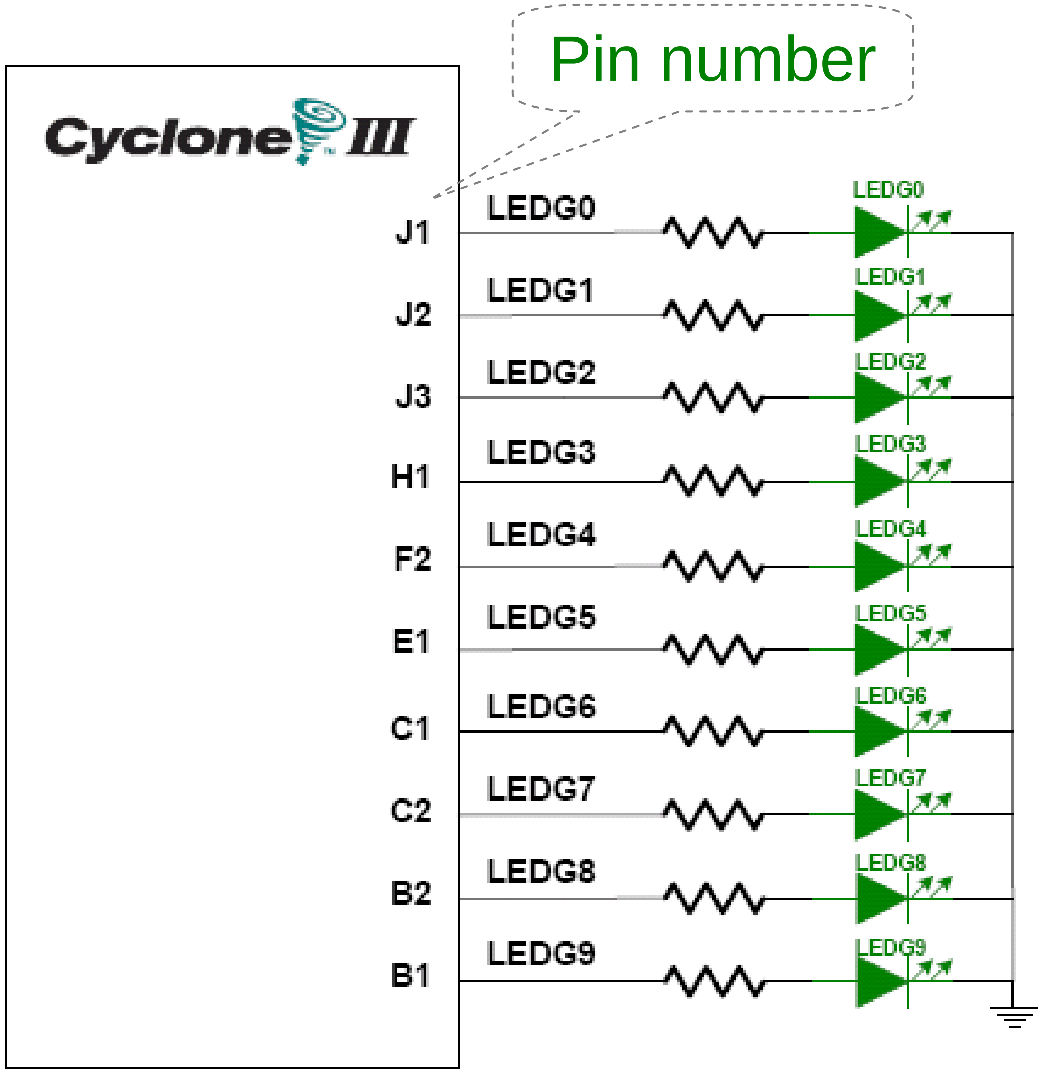
10 Slide switches (Sliders):
Up → Logic High
Down → Logic

Signal Name	FPGA Pin No.
BUTTON [0]	PIN_ H2
BUTTON [1]	PIN_ G3
BUTTON [2]	PIN_ F1

SW[0]	PIN_J6	SW[5]	PIN_J7
SW[1]	PIN_H5	SW[6]	PIN_H7
SW[2]	PIN_H6	SW[7]	PIN_E3
SW[3]	PIN_G4	SW[8]	PIN_E4
SW[4]	PIN_G5	SW[9]	PIN_D2

LEDs

10 LEDs
Opuput high → LED on
Output low → LED off



Signal Name	FPGA Pin No.
LEDG[0]	PIN_J1
LEDG[1]	PIN_J2
LEDG[2]	PIN_J3
LEDG[3]	PIN_H1
LEDG[4]	PIN_F2
LEDG[5]	PIN_E1
LEDG[6]	PIN_C1
LEDG[7]	PIN_C2
LEDG[8]	PIN_B2
LEDG[9]	PIN_B1