EXPERIMENT 12

12. 4-Bit Binary Sequential Multiplier

12.1 Objectives

- Introduction of large digital system design, i.e. data path and control path.
- To apply the above concepts to the design of a sequential multiplier.

12.2 Introduction: Design of Large Digital Systems

Large and medium size digital systems are mostly sequential systems with large number of registers and counters.

It is not practical to design such systems using FSM design techniques since this will result in a huge number of states (2^n, with n being the number of Flip-Flops (FF) including FFs used in all the registers and counters of the system).

Generally, large digital systems are partitioned into two units:

- **Data Path**: This is the data processing unit. It includes both combinational and sequential modules of well-defined functions, e.g. registers, counters, adders, multiplexers, decoders, etc.
- **Control Path**: This is the control unit or simply "*controller*" which controls the operations performed by the *data path* and the proper sequencing of these operations. The controller is implemented as an FSM that may be designed in the conventional manner.

In this lab, you will design the data path and controller of a 4-bit sequential multiplier. The design can be easily extended to an n-bit multiplier which uses the same controller and the same data path configuration but sizes of data path components (e.g. registers, and adder) should be adjusted accordingly.

12.3 Design Specifications

Inputs:

- **Multiplier**: First 4-Bit operand (**A**).
- **Multiplicand**: Second 4-Bit operand (**B**).
- **Reset**: Reset signal which puts the controller into the initial state.

Outputs:

- **P**: The 8-bit product result ($P = A \times B$).

Display:

- **P**: Displayed using 7-segment display digits.
- **Cnt**: The current counter content on one of the 7-segment display digits.
- **Control Signals**: Displayed using LED's.

Approach:

The block diagram of the sequential multiplier is shown in Figure 12.1.

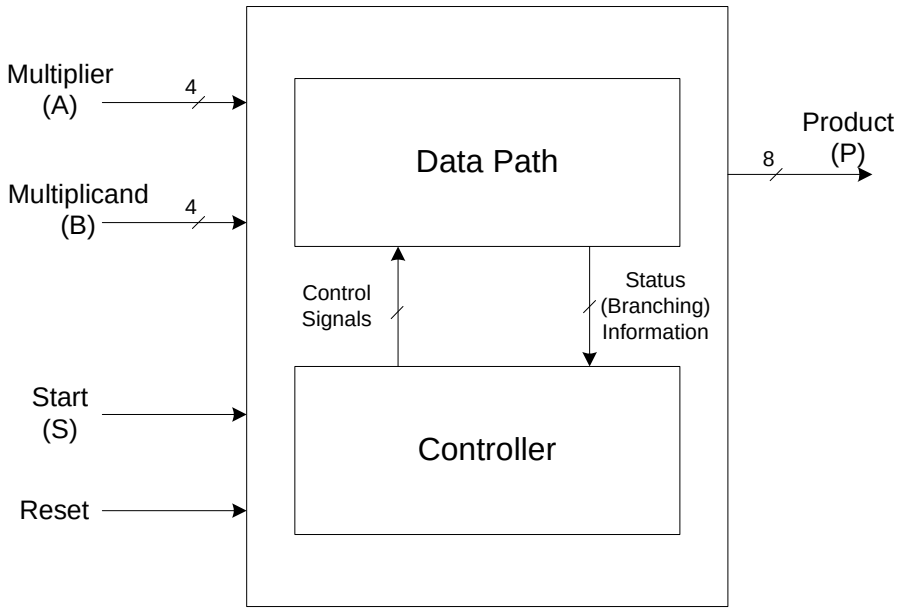

Figure 12.1: Block Diagram of Sequential Multiplier.

As in hand multiplication, we multiply the bits of the multiplier A ($a_3a_2a_1a_0$) by the multiplicand (B) starting from the LSB (a_0) to the MSB (a_3). This forms four partial products a_0B, a_1B, a_2B, and a_3B. The resulting partial products are added with each product shifted lefts by one bit position from its predecessor as shown in Figure 12.2.

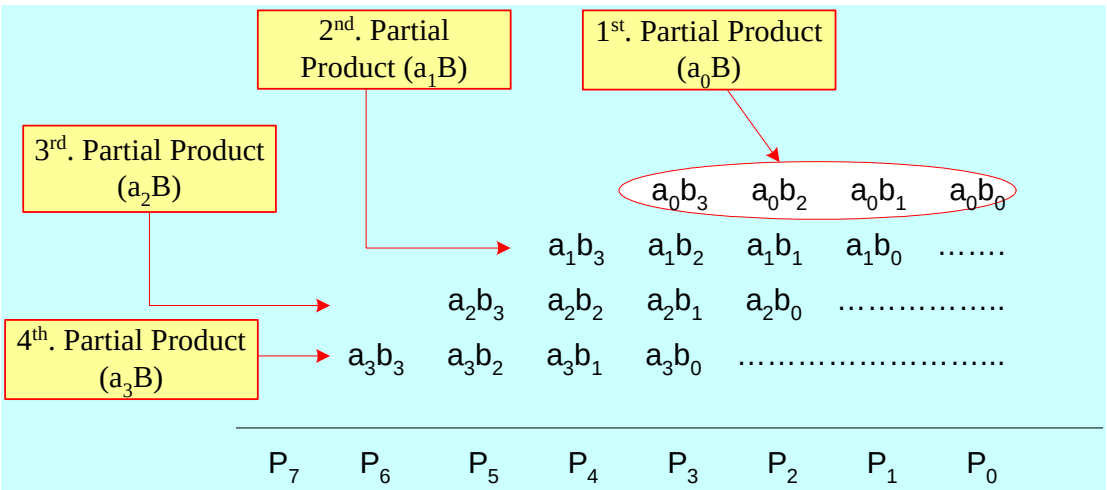

Figure 12.2: Binary Multiplication.

12.3.1 Multiplier Data Path

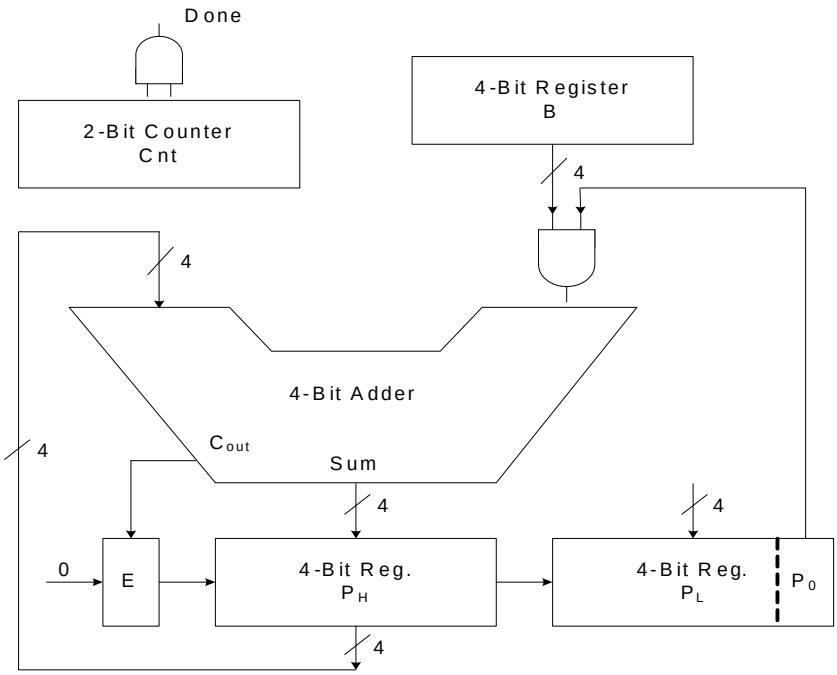

Figure 12.3: Multiplier Data Path.

The data path for the sequential multiplier is shown in Figure 12.3. It consists of several registers and an adder. The required registers include:

- **B-Register**: A 4-bit register which holds the *multiplicand* (B)
- **P-Register**: An 8-bit register which consists of two 4-bit registers P_L (P-Low) and P_H (P-High).
  - Initially the *multiplier* (A) is loaded in P_L.
  - The final result (*product*) is stored in P = (P_H, P_L).
- **E-Register**: A 1-bit register (FF), that is used to hold the carryout output of the adder
- **Cnt**: A 2-bit counter used to control the number of steps to be performed (total of 4 steps). The counter counts from 0 to 3. The operation is stopped when the counter reaches 3. This done condition (*Done*) is detected by an AND gate.

<u>Notation:</u>

- (E, P_H) refers to the 5-bit register consisting of E as the MSB and P_H.
- (E, P_H, P_L) refer to the 9-bit register consisting of E as the MSB, P_H. and P_L.

Computation Steps:

Note that i=Cnt all the time.

1. Initialize: $P_H \leftarrow 0$, $P_L \leftarrow$ A, B-Reg $\leftarrow$ B, Cnt $\leftarrow 0$.
2. $(E,P_H) \leftarrow P_H + a_iB = P_H + P_0B$.
3. Shift (E, P_H, P_L) right by one bit, and increment Cnt.
4. IF Cnt = 3 then STOP else Loop back to step 2.

Notes:

- The 1-bit register (E) may be considered as the MSB bit of the P_H register. The 4-bit sum outputs of the adder are the parallel inputs of P_H, while E is loaded with the adder carry out output (C_{out}). This expanded 5-bit register will be referred to as the (E, P_H) register.
- After initializing the P-Register, the partial products (a_iB) are accumulated into the P_H -register one by one.
- Instead of shifting the partial products *lefts* before accumulating into P, the P-Register (E, P_H, P_L) is shifted *rights* and the partial products are then added to it.
- In the shift right step, the three registers (E, P_H, P_L) are treated as one 9-bit shift register (P). In this *shift right step*, *a zero is shifted into E*.
- Since P_L is initially loaded with A, and P is shifted rights one bit per iteration, then the LSB P_0 will *always* equal to a_i, i.e. $P_0 = a_i \ \forall$ *iteration steps*.
- **Important Note**: Control signals are asserted in some state but **ACTUAL EXECUTION OF CORRESPONDING OPERATION DOES NOT TAKE PLACE EXCEPT AFTER THE NEXT ACTIVE CLOCK EDGE ARRIVES.**

Example:

To perform the multiplication of A=**1011** by B=**1101**, you can do it step-by-step as in Table 12.1. Note that the state entry shows the state after the clock.

Table 12.1: Multiplication Example.

Action	State	Cnt	E	P_H	P_L	P_0B
Reset	0	0	0	0000	0000	0000
Initiation	1	0	0	0000	1011	1101
Load	2	0	0	1101	1011	1101
Shift	1	1	0	0110	1101	1101
Load	2	1	1	0011	1101	1101
Shift	1	2	0	1001	1110	0000
Load	2	2	0	1001	1110	0000
Shift	1	3	0	0100	1111	1101
Load	2	3	1	0001	1111	1101
Shift	0	3	0	1000	1111	1101

Controlling Data Path Registers

Following is a list of data path registers and the various control signals:

- **B-Register**: Requires a parallel load capability to load the multiplicand (B)
- **P_H&P_L-Registers**: Requires the following features:
  - Asynchronous clear for initialization.
  - Parallel load to allow loading of the adder output sum bits.
  - Shift right.
- **E-Register**: Requires the following features:
  - Asynchronous clear for initialization.
  - Load to allow loading of the adder carry-out bit.
- **Cnt**: Requires the following features:
  - Asynchronous clear for initialization.
  - Increment by one.

Figure 12.4 shows the control signals of data path registers.

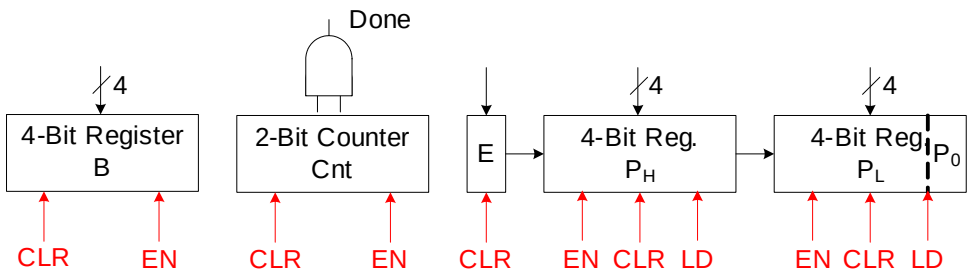

Figure 12.4: Data path Register Control Signals.

12.3.2 Multiplier Controller

The data path control signals are generated by the controller in a proper sequence to compute the required result ($P = A \times B$). Controllers in general need status information to decide on the next proper actions. In the case of multiplier, the controller uses the signal "*Done*" to determine when to stop the computation.

The state diagram of the controller is shown in Figure 12.5. The controller FSM has 3 states (S0, S1, and S2). The *Reset* signal resets the system to the first state S0 and clears all the components.

The next clock edge moves the FSM to the second state S1, loads the *multiplier* (A) into P_L, and the *multiplicand* (B) into the B-register.

The next clock will take the FSM to the third state S2, loads the adder's sum and carry-out bits into the P_H and E registers.

Cnt $\neq$ 3 indicates that there are more multiplier bits to be processed. In this case, the next state of S2 is S1, otherwise the computation is done and controller returns back to state S0. In either case, the counter is incremented and the (E, P_H, P_L) register is shifted right.

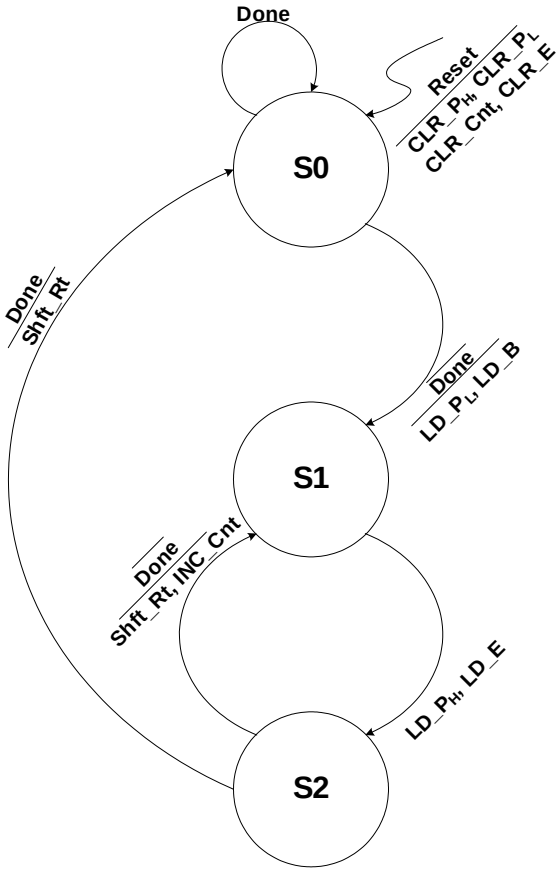

Figure 12.5: Multiplier Controller.

12.4 Pre-Lab

12.4.1 First Week – Phase 1

- Bring the *Multiplexed Seven-Segment Displays* macro and its clock.
- Study and *understand* the design strategy explained in this document.
- On table below, show how the product of A=1010 and B=1110 is computed according to the above multiplier circuit.

Action	State	Cnt	E	P_H	P_L	P_0B

12.4.2 Second Week – Phase 2

- Design the controller FSM using D-FFs and generate all data path control signals.
  - Clr_P_H =
  - Clr_P_L =
  - Clr_Cnt =
  - Clr_E =
  - Ld_B =
  - Ld_ P_L =
  - Ld_P_H =
  - EN_P_L =
  - EN_P_H =
  - EN_Cnt =
- Bring the work of phase 1 with you completed.

12.5 In-Lab

You need to complete the lab by breaking it to two stages.

- Design and verification of data path. You need to design the data path and provide the control signals manually and verify its functionality.
- Design and verification of controller. Design the controller using D-Flip-Flops. Verify that it is functionally correct by giving the external signals and observing the data path control signals.
- Complete the design. Join the two components of the multiplier together. Verify its functioning after downloading on the FPGA prototype board.

12.5.1 Multiplier Data Path – Phase 1

Note that if an enabled shift register is clocked and its load (LD) control signal is low, the register will be shifted. Therefore, only one signal is used to determine the LOAD/SHIFT operation.

1. Use a counter with asynchronous clear and enabling capability.
2. You also need the following components:
   - Two shift registers (SR4CLED) for P_H and P_L.
   - A 4-Bit register to store B (FD4CE).
   - A 4-Bit adder with carry out.
   - One-Bit register to store E (FDC).
3. Construct a design similar to Figure 12.3 constraining the inputs of the *multiplier* (A) to be at decimal value “10”, and constraining the inputs of the *multiplicand* (B) to be at decimal value “6”.
4. E-Register should get the carry-out only when P_H is loaded
5. Import the macro of the *Multiplexed Seven-Segment Displays* and its clock.
6. See Figure 12.6 for inputs/outputs connectivity.
7. Download your design and test it by following Table 12.2 . Note that if LD of the shift register (SR4CLED) is high, the register will load regardless of the enable signal.

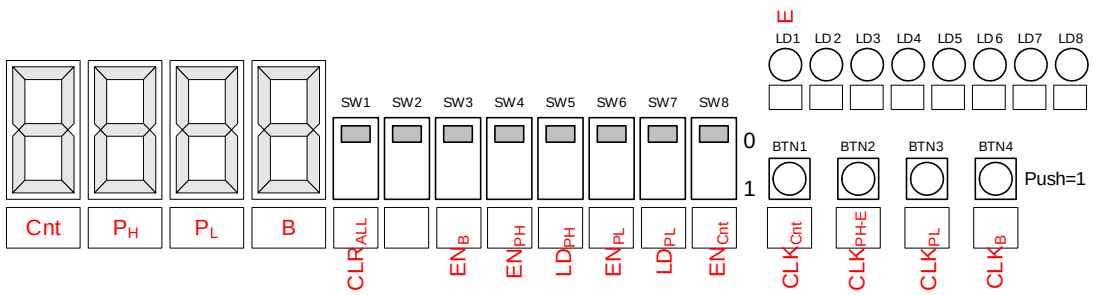

Figure 12.6: Data Path Inputs/Outputs Assignment.

Table 12.2: Testing the Data Path Logic.

No	SW1 CLR_{ALL}	SW3 EN_B	SW4 EN_{PH}	SW5 LD_{PH}	SW6 EN_{PL}	SW7 LD_{PL}	SW8 ED_{Cnt}	BTN CLK_{Cnt}	BTN2 CLK_{PH-E}	BTN3 CLK_{PL}	BTN4 CLK_B	7-Seg. Displays Cnt	P_H	P_L	B	LD1 E
1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
2	0	1	0	0	0	0	0	X	0	0	1	0	0	0	6	0
3	0	0	1	1	1	1	0	X	1	1	X	0	0	A	6	0
4	0	0	0	0	1	0	1	0	X	1	X	0	0	5	6	0
5	0	0	0	1	0	0	1	1	1	X	X	1	6	5	6	0
6	0	0	0	1	0	0	1	1	1	X	X	2	C	5	6	0
7	0	0	0	1	0	0	0	X	1	X	X	2	2	5	6	1
8	0	0	1	0	0	0	0	X	1	X	X	2	9	5	6	0
9	1	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0

12.5.2 Multiplier Controller – Phase 2

To design the multiplier data path you can follow these steps:

8. Use a push button for *Reset* and use it to clear all design components (BTN1).
9. Connect another push button to BUFGS and use it to clock all sequential components in the design (BTN4).
10. You need two bits to store you states (S0, S1, and S2). Use two FFs with clearing capability for that (FDC).
11. Connect the outputs of the FFs to two LEDs (LD7 and LD8). LD8 should be the least significant FF.
12. The Boolean expressions of the various control signals can be easily derived from the state diagram. See Figure 12.7 for inputs/outputs connectivity.
13. Download your design and test it. It should follow Table 12.3. Notice that if you press *Reset* at any time the system will go to S0.

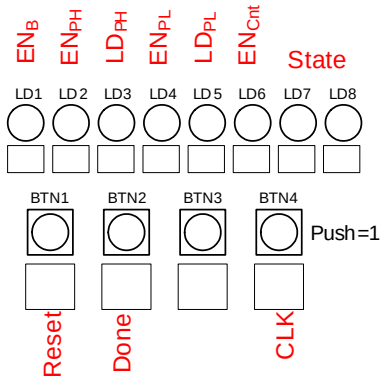

Figure 12.7: Controller Inputs/Outputs Assignment.

Table 12.3: Testing the Controller Logic.

LD7&8 State In	BTN4 CLK	BTN2 Done	LD1 EN_B	LD2 LD_{PL}	LD3 EN_{PL}	LD4 LD_{PH}	LD5 EN_{PH}	LD6 EN_{Cnt}	LD7&8 State Out
0	X	1	0	0	0	0	0	0	0
0	0	0	**1**	**1**	**1**	0	0	0	0
0	1	0	0	0	0	**1**	**1**	0	1
1	0	X	0	0	0	**1**	**1**	0	1
1	1	X	0	0	**1**	0	**1**	**1**	2
2	0	0	0	0	**1**	0	**1**	**1**	2
2	0	1	0	0	**1**	0	**1**	0	2
2	1	0	0	0	0	**1**	**1**	0	1
2	1	1	0	0	0	0	0	0	0

12.5.3 Multiplier Data Path and Controller

Combine the multiplier data path and controller taking into account the following:

14. The *Done* signal should be high if the counter has the value 3
15. Use four switches (SW5-SW8) to load the *multiplier* (A) into P_L, and four switches (SW1-SW4) to load the *multiplicand* (B) into its register as in Figure 12.8.
16. Download your design and show it to the lab instructor.

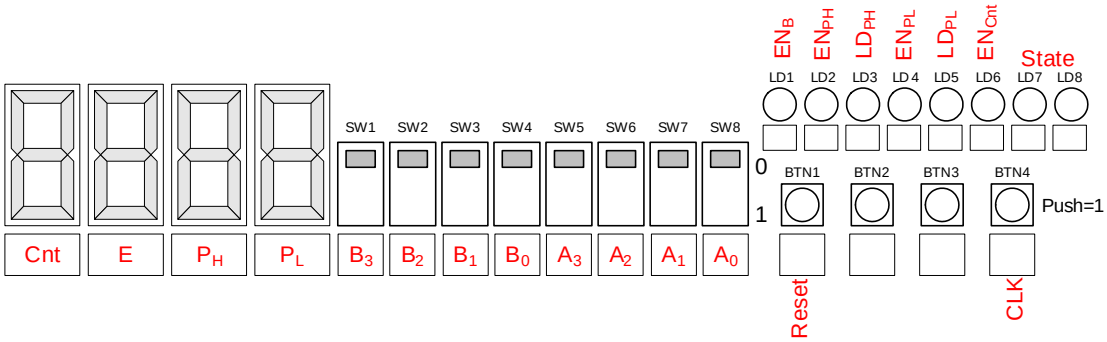

Figure 12.8: Multiplier Inputs/Outputs Assignment.

12.6 Post-Lab

You have to hand in a lab report that contains the following:

- Work sheet of the multiplication example on the Pre-lab
- All tables, k-maps, and Boolean expirations used in your design.