Lab #13

4-Bit Binary Sequential Multiplier

Objectives

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

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 # of FFs including FFs used in all the registers and counters of the system).
- Generally, large digital systems are partitioned into two units:
  1. **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.
  2. **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, adder) should be adjusted accordingly.

Design Specifications

Inputs:

- **A**: First 4-Bit operand (*multiplier*).
- **B**: Second 4-Bit operand (*multiplicand*).
- **S**: Start signal which initiates the multiplication operation
- **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 content of the count register on one of the 7-segment display digits.
- **Data Path Control Signals**: Displayed using LED's.

Approach:

The block diagram of the sequential multiplier is shown in Figure 1.

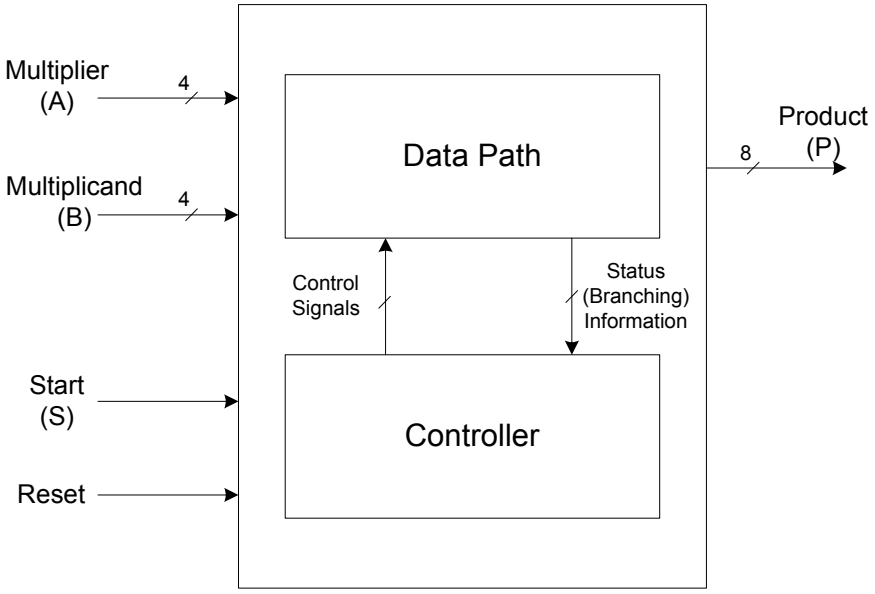

Figure 1: Block Diagram of Sequential Multiplier

As in hand multiplication (see Figure 2), 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 4 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 2.

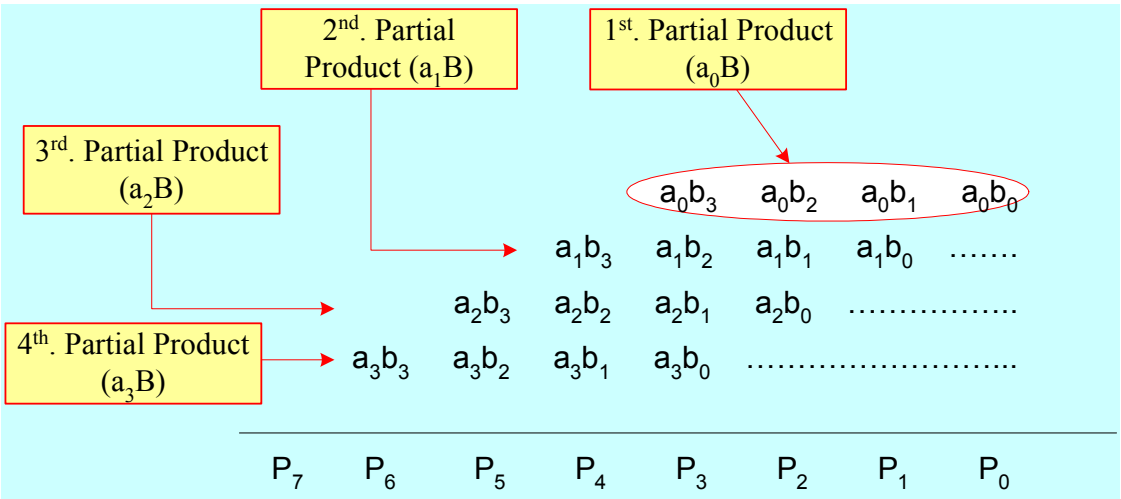

Figure 2: Binary Multiplication

Multiplier Data Path

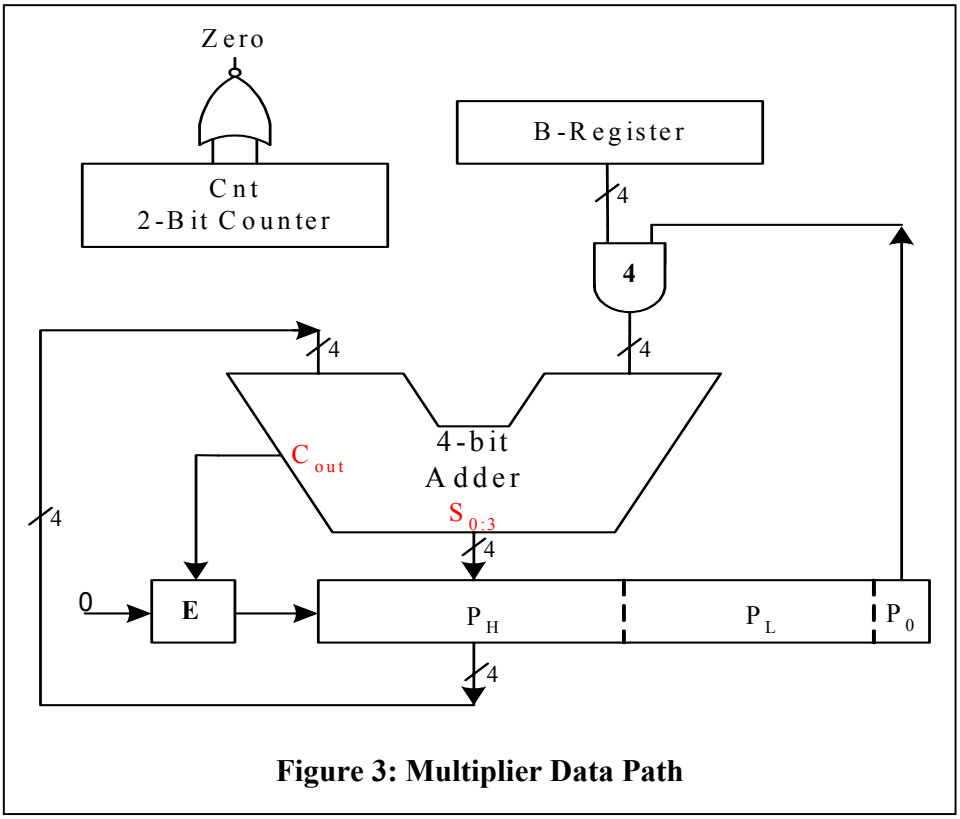

Figure 3: Multiplier Data Path

The data path for the sequential multiplier is shown in Figure 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, while P_H is cleared
  - The final result (product) is stored in P = ($\mathbf{P_H}$, $\mathbf{P_L}$).
- **E-Register**: A 1-bit register, that is used to hold the carryout output of the adder
  - Initially E is cleared
  - It may be considered the 9^{th} Bit of P, i.e. P_8.
  - In the final step, E will hold a 0 value.
- **Cnt**: A 2-bit down counter used to control the number of steps to be performed (total of 4 steps). The counter counts from 3 down to 0.

The operation is stopped when the count reaches 0. This zero condition (**Zero**) is detected by a NOR gate.

Notation:

- (**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:

1. Initialize: $i=0$, **$P_H \leftarrow 0$**, **$P_L \leftarrow$** A, **B-Reg** ← B, **Cnt** ← $n-1$, where n = number of operand bits.
2. **$(E,P_H) \leftarrow P_H + a_i B = P_H + P_0 B$;**
3. Shift (**E, P_H, P_L**) right by one bit; **Cnt ← Cnt -*1***; *and* $i=i+1$.
4. **IF Cnt = 0 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 output of the adder are the parallel input 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-Reg, 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 doesn't take place except **AFTER the NEXT ACTIVE CLOCK EDGE arrives**.

Example: A=1011, B=1101, then $n = 4$.

1. Initialization: P = $(\mathbf{P_H}, \mathbf{P_L})$ = 0000_1011
   - B = 1101
   - Cnt = 3
   - $i = 0$
2. **E, $\mathbf{P_H} \leftarrow \mathbf{P_H} + P_0$B** = (0000) + (1101) = 01101 → **(E, $\mathbf{P_H}$, $\mathbf{P_L}$) = (0, 1101, 1011)**
3. Shift P Right –-- **P = (0110, 1101) , Cnt** = 2, and i=1
4. **E, $\mathbf{P_H} \leftarrow \mathbf{P_H} + P_0$B** = (**0110**) + (1101) = 10011 → **(E, $\mathbf{P_H}$, $\mathbf{P_L}$) = (1, 0011, 1101)**
5. Shift P Right –-- **P = (1001, 1110) , Cnt** = 1, and i=2
6. **E, $\mathbf{P_H} \leftarrow \mathbf{P_H} + P_0$B** = (**1001**) + (0000) = 01001→ **(E, $\mathbf{P_H}$, $\mathbf{P_L}$) = (0, 1001, 1110)**
7. Shift P Right –-- **P = (0100, 1111) , Cnt** = 0, and i=3 (*Note that **Cnt** becomes 0 only after the next clock not while being in state S2*)
8. **E, $\mathbf{P_H} \leftarrow \mathbf{P_H} + P_0$B** = (**0100**) + (1101) = 10001 → **(E, $\mathbf{P_H}$, $\mathbf{P_L}$) = (1, 0001, 1111)**
9. Shift P Right –-- **P = (1000, 1111) , Cnt** = 0
10. STOP.

Controlling Data Path Registers

- Operations of the data path registers are controlled by the states of its control signals. These control signals are produced by the controller.
- Again, it is important to stress that the control signals asserted in some state are executed only **AFTER the arrival of the NEXT ACTIVE CLOCK EDGE**.

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)
- **$\mathbf{P_H}$-Register**: Requires the following features:
  - Synchronous clear for initialization ($P_H \leftarrow 0$)
  - Parallel load to allow loading of the adder output sum bits.
  - Shift right..
- **$\mathbf{P_L}$-Register**: Requires the following features:
  - Parallel load to allow loading of the multiplier (A).
  - Shift right.
- **E-Register**: Requires the following features:

- Parallel load to allow loading of the adder carryout output bit.
- Shift right capability with 0 serial input.

➢ **Cnt**: Requires the following features:

- Parallel load to allow initial loading of *n*-1.
- Decrement by 1.

Figure 4 shows the control signals of data path registers.

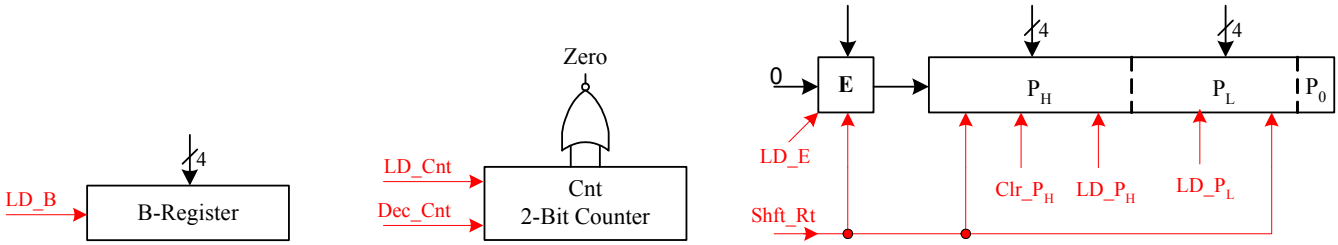

Figure 4: Data path Register Control Signals

Multiplier Controller

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

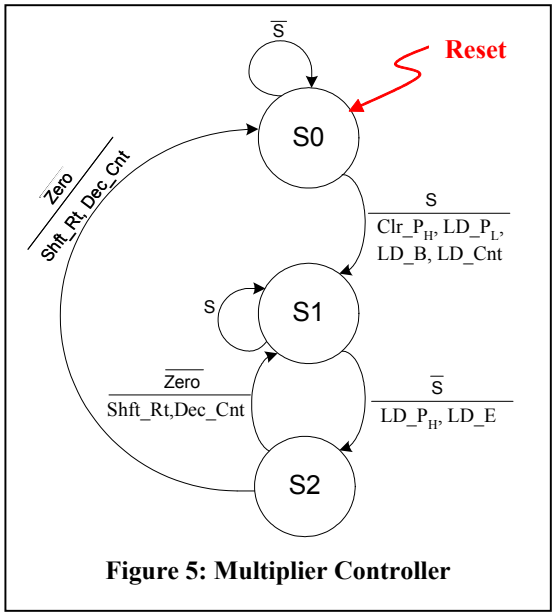

Figure 5: Multiplier Controller

The state diagram of the controller is shown in Figure 5. The controller FSM has 3 states (S0, S1, and S2). The initial state is the reset state S0. The state machine stays in the reset state until a *Start* signal "S" is asserted high indicating the availability of new input data. Thus, when S is asserted high, the next clock edge moves the FSM to the second state S1, loads multiplier input (A) into $\mathbf{P_L}$, clears $\mathbf{P_H}$, loads multiplicand (B) into the B-register, and loads the value '*n-1*' (= 3) in the down counter "**Cnt**".

The FSM remains in the second state (S1), as long as the start signal S is high. If the start signal is reset low the next clock will take the FSM to the next state S2 while asserting the following data path control signals:

- LD_P_H and LD_E, which load the adder sum and carry-out bits into the P_H and E registers.

Cnt ≠ 0 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, and the following data path control signals are asserted:

- Decrement the counter, *Dec_Cnt*.
- *Shft_Rt*, which causes a 1-bit right shift of the (**E, P_H**, **P_L**) register.

It should be noted that a "**Reset**" input will put the FSM in the initial state S0.

Generating Data Path Control Signal

The Boolean expressions of the various control signals can be easily derived from the state diagram as follows:

- Clr_ **P_H** = S0.S
- Ld_ **P_L** = S0.S
- Ld_ **B** = S0.S
- Ld_ **Cnt** = S0.S
- Ld_ **P_H** = S1. S'
- Ld_ **E** = S1. S'
- Shft_Rt = S2
- Dec_Cnt =S2

Implementation Issues

There are various choices available for implementing the counter. You can either use a simple up counter to produce the flag zero when it reaches a value '3' or you can use a down counter and preload it with value '3' in it. In the former case, you might need to clear the counter instead of loading it with value (*n-1*) (=3) as well as change the zero flag logic.

You may use the 4-bit adder available in the Spartan device library.

Similar to the adder, shift registers are also available in the library.

Inputs:

- **A**: First 4-Bit operand (*multiplier*)P: → Use 4 Switches
- **B**: Second 4-Bit operand (*multiplicand*). : → 4 Switches
- **S**: Start signal which initiates the multiplication operation: → Push button
- **Reset**: Reset signal which puts the controller into the initial state:→ Push button
- **Clock**:→ Use one push button

Outputs:

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

Display:

- **Cnt**: The current content of the count register to be displayed on the leftmost 7-segment display digit.

- **(E, P_H, P_L)**: Displayed using the remaining THREE 7-segment display digits. Leftmost digit displays the contents of E, while the second & third digits display **P = (P_H, P_L)**.
- **Data Path Control Signals**: Displayed using LED's in the following order from left to right:
    - **i. Ld_ P_H**
    - **ii. Ld_ E**
    - **iii. Shft_Rt**
    - **iv. Dec_Cnt**
    - **v. Clr_ P_H**
    - **vi. Ld_ P_L**
    - **vii. Ld_ B**
    - **viii. Ld_ Cnt**

You already have a good exposure utilizing the on-board seven-segment display digits. In this lab, you need to use a seven-segment decoder that displays result in hex-format. The format for the six hexadecimal digits is shown in Figure 6.

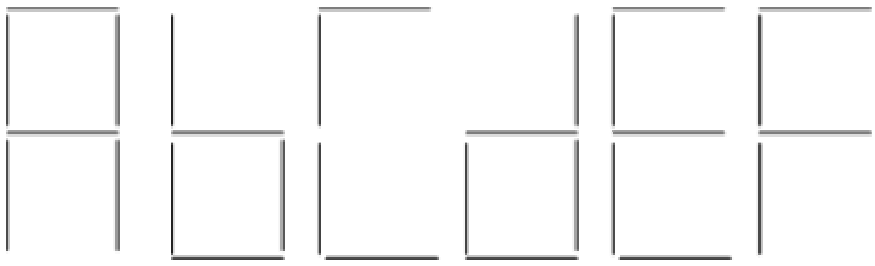

Figure 6: Hexadecimal Display

.

Pre-Lab

1. Study and *understand* the design strategy explained in this document.
2. On a worksheet, show how the product of A=1010 and B=1110 is computed according to the above multiplier circuit.
3. Design the controller FSM using D-FFs and generate all data path control signals.
4. Design the data path control signal logic.
5. Design/plan strategy for displaying the required information.

Before coming to the lab, make sure you have the complete design with you.

In-Lab

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

- Design and verification of data path.

You need to design the data path and provide the control signals manually. Verify your data path by simulating it through the embedded simulator.

- 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 in the embedded simulator.
- Design, verification and implementation of display decoder.
  You need to design, verify and implement the seven-segment decoder required for displaying the result of the multiplier. For this purpose, you can use the 8 switches available on the board for producing a hex digit. For verification purposes, you can use this hex digit for both the hex digits to be displayed.
- Complete the design.
  Join the three components of the multiplier together. Verify its functioning in the simulator and after downloading on the FPGA prototype board.

Hand-in

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

- Section on the Pre-lab explaining the design of each block and the work sheet of the multiplication example.
- Discussion of implementation results commenting on the time a control signal becomes valid and the time it actually gets executed.

- **Conclusions.**