Implementing a Serial Adder in Quartus

Munim Zabidi

June 10, 2016

Abstract

This documents gives the basic steps in implementing a serial adder in Altera Quartus.

1 Overview

A digital system is first defined with the specification of the top level component (entity) and the algorithm to be performed. From the specifications, the appropriate port interface is defined.

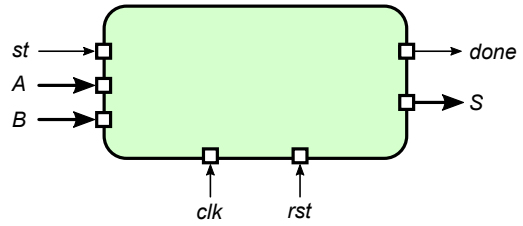

Figure 1: Serial Adder entity.

Inputs

- **A**: First n-bit Operand (Addend)
- **B**: Second n-bit Operand (Augend)
- **st**: Start signal which initiates the addition operation
- **rst**: Reset signal which puts the controller into the initial state

Outputs

- **S**: The (n+1)-bit sum with carry ($S = A + B$)
- **done**: Active when the operation is complete

The algorithm is as follows:

```
Load A, Load B
c = 0, k = 0
while(k < n)
  {c,S[k]} = A[k] + B[k] + c
  k = k + 1
endwhile
done = 1
```

where n is the number of bits, c is the carry-in bit and k is a counter variable which goes from 0 to $n-1$. Variables on the left-hand side of the equations are normally need to be stored in registers, hence, we infer that two n-bit registers A and B, and a 1-bit carry register, c are needed. An $(n+1)$-bit register S stores the sum. Since the addition is performed in a serial fashion, a 1-bit adder is needed.

In practical designs, a down counter is preferred over an up counter because detecting a final value of 0 is trivial. Therefore, the pseudo-code is modified as follows:

```
Load A, Load B
c = 0, k = n
while (k > 0)
  {c,S[n-k]} = A[n-k] + B[n-k] + c
  k = k - 1
endwhile
done = 1
```

2 First-Cut Circuit

A serial adder consists of three n-bit shift registers, a full-adder and a D flip-flop. Two parallel-in-serial-out (PISO) shift registers holds the numbers (A and B) to be added, while a serial-in-parallel-out (PISO) register holds the sum (S). The full-adder performs the addition operation on the values stored in the input shift registers and the carry flip-flp in n clock cycles. A "first-cut" block diagram of a serial adder is shown in Figure 2.

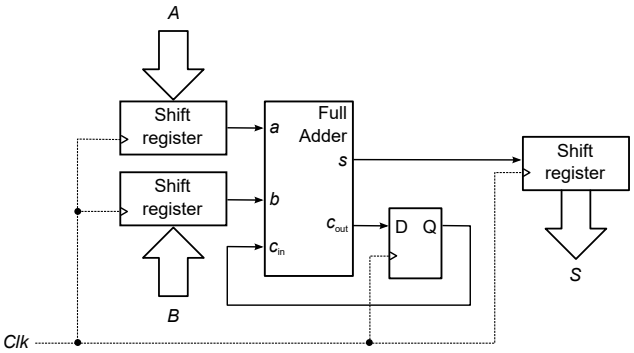

Figure 2: Overview of the serial adder circuit.

Table 1: Sequence of 4-bit serial adder operation performing 1011 + 0011 + 0 → 01110.

A	B	c_i	c_{i+1}	s_i	S
1011	0011	0	1	0	----
-101	-001	1	1	1	0---
--10	--00	1	0	1	10--
---1	---0	0	0	1	110-
----	----	0	-	-	1110

During each clock cycle, the current least significant bits of the A and B registers, a_0 and b_0, with the current carry-in, c_i are summed by the full-adder. This produces

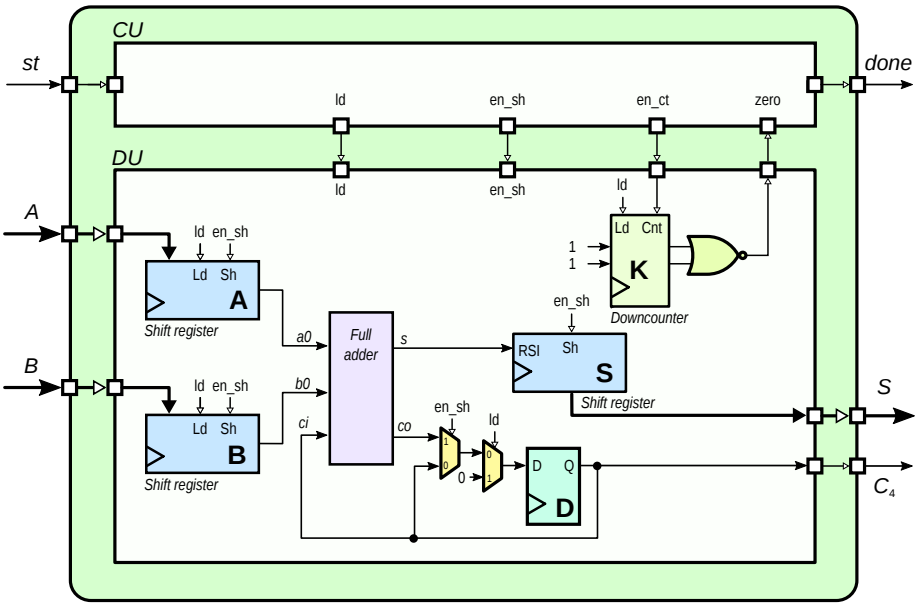

Figure 3: Serial adder datapath.

sum bit s_i and the carryout bit c_{i+1}. On the next clock edge, s_i are shifted into the result register S. At the same time, c_{i+1} is stored into the DFF, to prepare for the next set of bits. The sequence of the operation of a 4-bit serial adder is illustrated in Table 1.

The value at the D flip-flop, c_i, is the carry-out from the whole addition. This bit can be combined with S to form the 5-bit result.

The serial adder completes the addition process in 4 clock pulses. Shifting of the registers must be stopped after completion. Otherwise, the computed result will be shifted out and disappear.

3 Datapath Overview

Figure 3 shows the top level circuit diagram for a 4-bit serial adder based in on the first-cut circuit. A 2-bit counter and muxes at the DFF input are now part of the datapath. The signals needed for controlling this DPU are:

- **en_ld**: Load the register/counter
- **en_sh**: Enable shift register operation
- **en_ct**: Enable counter operation. Later we found out we do not need this signal.

Here's how it operates:

- In idle state, the FSM waits for *st* (start) signal while continuously loading A, B registers, clearing D and loading counter K with 3.
- When start signal is received, repeat the following 4 times:
  - Shift registers A, B, S and flip-flop D
  - Decrement downcounter K
- After the bits have been shifted 4 times (signaled by Zero signal from downcounter), stop shifting and output the *done* signal. The 5-bit sum is now stable and correct.

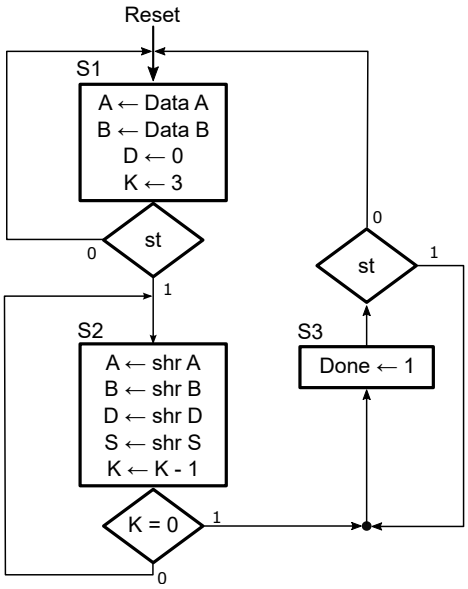

Figure 4: High-level ASM for serial adder.

The high-level ASM as shown in Fig. 4 describes the tasks done by the circuit.

4 Building Blocks

After the top-down design stage comes the bottom-up implementation stage. The modules to be built are:

- Full adder
- DFF with enable and clear
- PIPO register
- SIPO register
- Downcounter

4.1 Full adder

The full adder is constructed by combining two half-adders and an OR gate (Fig. 5 and Fig. 6).

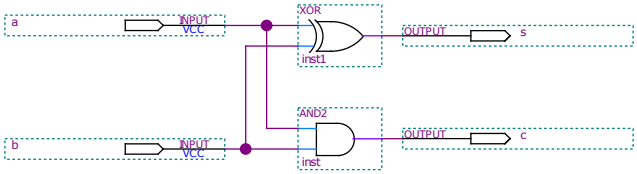

Figure 5: Half adder component.

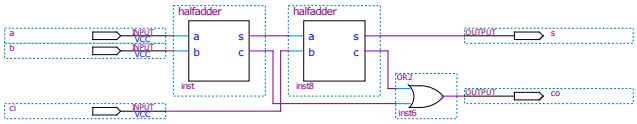

Figure 6: Full adder component.

4.2 DFF with enable and clear

A basic DFF loads a new value every time a clock pulse arrives. For this serial adder, a custom DFF circuit is required because we want to stop loading when the operation is completed. The *D flip-flop with enable and clear* (DFFEC) shown in Fig. 7 has two additional controls:

- **load**: to load a new value into the flip-flop
- **clear**: to clear the flip-flop

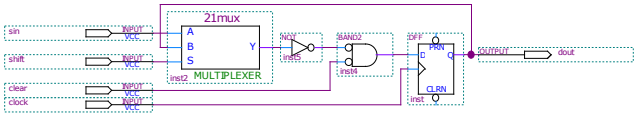

Figure 7: D flip-flop with enable and clear.

In the serial adder datapath architecture, the **ld** signal initializes all registers to prepare for adding a new set of numbers. Connecting the **ld** signal to the **clear** input of the DFF modules prepares the DFF adding the first bit pair.

4.3 SIPO Shift Register

The serial-in-parallel-out (SIPO) shift register receives a sum bit from the full adder, one bit at a time. Data is inserted serially through the right shift input.

In Quartus, first choose the **lpm_shiftreg** megafunction. On the first pop-up screen (Fig. 8), choose the following settings:

- Output bus width: *4*
- Shift direction: *right*
- *Data output*
- *Clock enable* input
- *Serial shift data input*

Then click Finish.

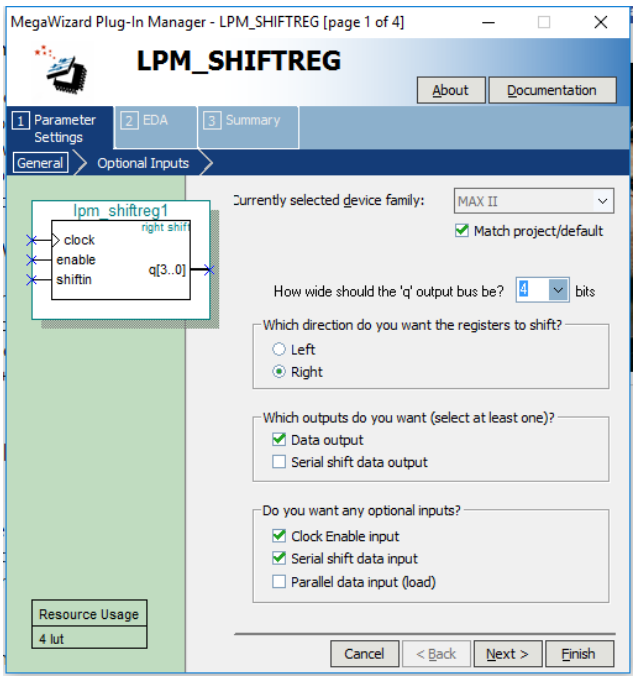

Figure 8: Setting the parameters SIPO shift register.

4.4 PISO Shift Register

Two parallel-in-serial-output (PISO) registers are required. Repeat the following procedure in Quartus twice.

First choose the **lpm_shiftreg** megafunction. On the first pop-up screen (Fig. 9), choose the following settings:

- Output bus width: *4*
- Shift direction: *right*
- *Data output* (Not strictly needed for this project but choosing it allows us to see the internal operation of the SIPO.)
- *Serial shift data output*
- *Clock enable input*
- *Parallel data input* (load)

Then click Finish.

4.5 Loop control counter

This counter controls how many times a task is repeated. The counter is initialized with the number of loops required minus 1 and counts down to 0. To repeat a task 3 times, load the counter with 3. Each time a task is performed, the counter is decremented by 1. At zero, a NOR gate sends a *zero* signal to the controller.

In Quartus, first choose the **lpm_counter** megafunction. On the first pop-up screen (Fig. 10), choose the following settings:

- Output bus width: *2*
- Count direction: *down only*

Click Next to get the second pop-up screen. On the second pop-up screen, just click Next to skip the options and get the third pop-up screen.

On the third pop-up screen (Fig. 11), in the synchronous inputs box, select:

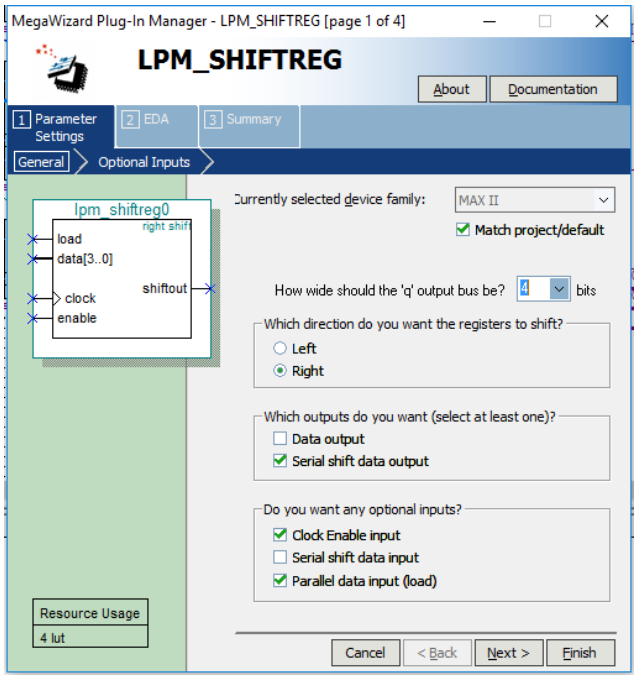

Figure 9: Setting the parameters for a PISO shift register.

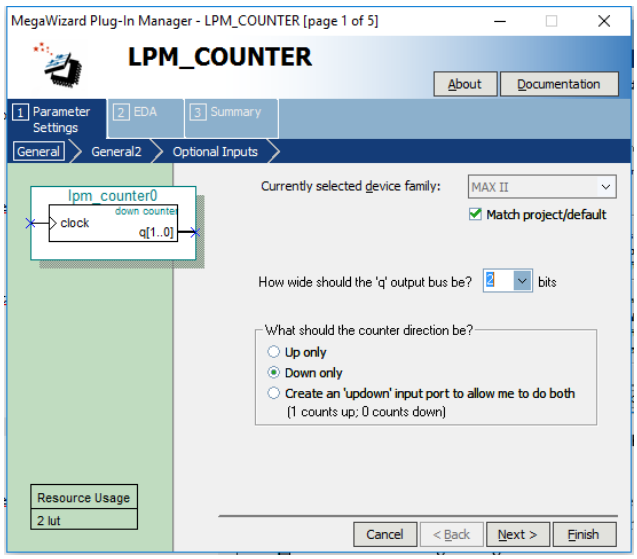

Figure 10: Setting the parameters for a simple down counter.

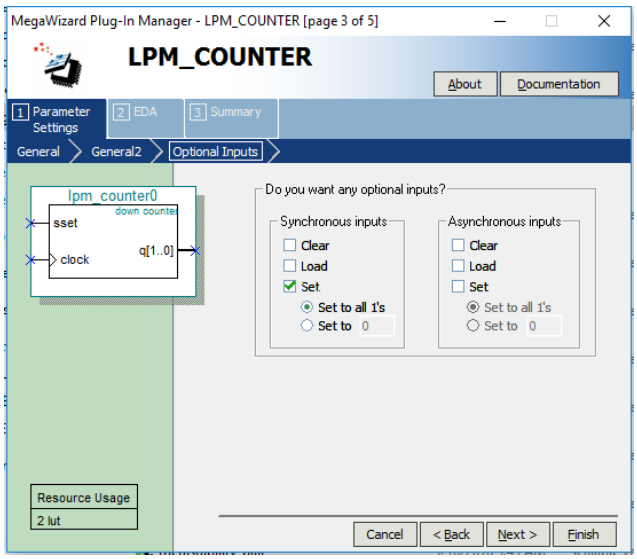

Figure 11: Choosing a pre-load value of 3.

- *Set*
- *Set to all 1's*

This last option creates the **sset** input. Whenever this input is high, the counter is loaded with 3. When this input is low, the counter immediately counts down.

5 Datapath Unit

When all building blocks are completed, they are integrated to form the datapath circuit as shown in Fig. 12. The datapath is tested by setting the inputs via a waveform editor. Once the datapath design is verified, we are ready to design the controller.

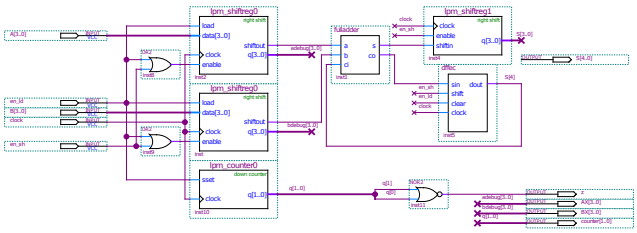

Figure 12: Serial adder dapath unit.

Input port types:

- Clock
- Data inputs A and B directly from an external system.
- Inputs from the control unit: **en_sh** to enable shifting and adding, and **en_ld** to initialize all registers.

Output port types:

- **Zero** tells the control unit that counter has reached 0
- Data outputs to an external system.

Two OR gates are added to the enable input of the PISO because to perform shifting, **enable** must be high but to perform loading, both **enable** and **load** inputs must high at the same time.

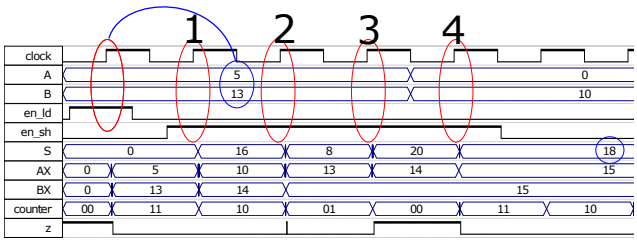

Figure 13: Simulation of dapath unit.

Outputs AX[3..0], BX[3..0] and counter[1..0] are optional outputs to help us understand what is happening the circuit.

Output ports are two types.

- **Zero** to tell the control unit that counter has reached 0
- Data outputs to an external system.

Simulation of the datapath unit is shown in Fig. 13. On the first clock edge, **en_ld** is high to initialize all registers ($A \leftarrow 5, B \leftarrow 13, D \leftarrow 0, K \leftarrow 3$). On the four subsequent clock edges, **en_sh** is high to enable adding and storage of result bits. It is important to disable this signal after four clock edges to freeze the result.

6 Control Unit

The controller automates the datapath operation. The low-level ASM in Fig. 14 defines the actual status and control signals required for control unit operation.

The control unit is implemented using the almost-one-hot state assignment. We get the following next state equations:

$$S0 = \overline{S0} \cdot \overline{S1} \cdot \overline{S2} + S0 \cdot \overline{st} + S2 \cdot \overline{st}$$
$$S1 = S0 \cdot st + S1 \cdot \overline{zero}$$
$$S2 = S1 \cdot zero + S2 \cdot st$$

The output equations are read directly from the ASM chart:

$$en\_ld = S0$$
$$en\_sh = S1$$
$$done = S2$$

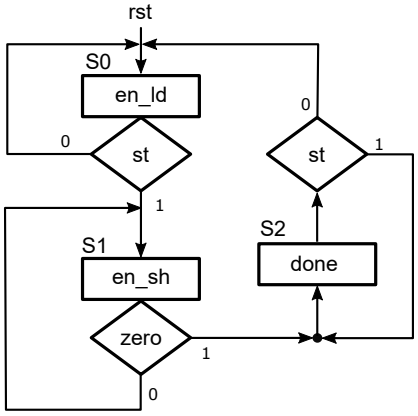

Figure 14: Low-level ASM for serial adder.

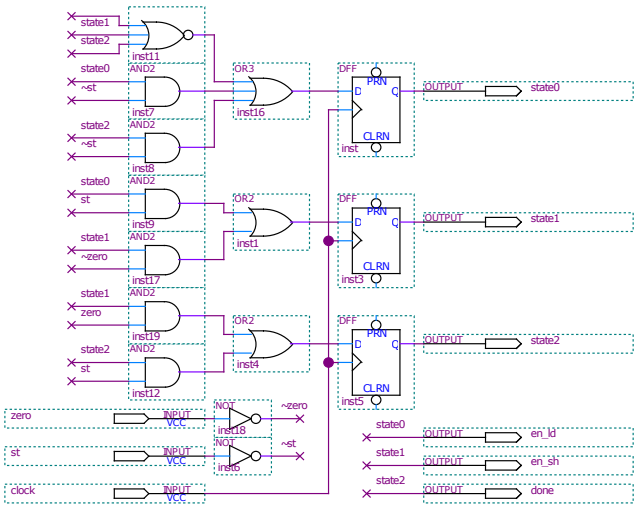

Figure 15: Control unit.

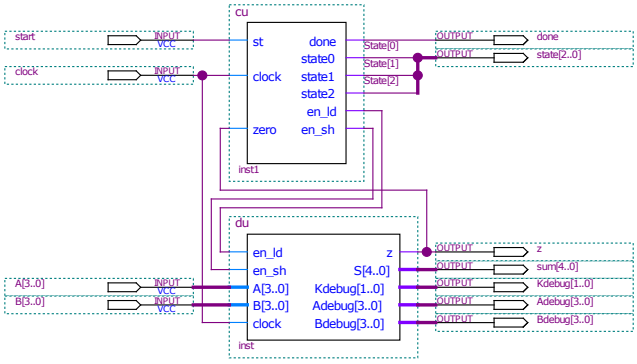

Figure 16: Serial adder top level entity.

The control unit is then combined with the datapath unit in the top level entity circuit in Fig. 16. The simulation of the complete serial adder is shown in Fig. 17.

When using a controller based on the almost one-hot assignment, remember to wait one clock cycle for the flip-flops to transition from 000 (“boot” or initial state) to 001 (state S0).

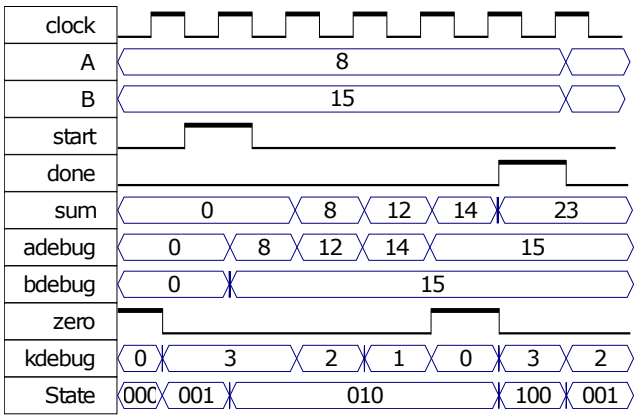

Figure 17: Simulation of serial adder for 8 + 15.

7 Version 2 of Serial Adder

This section gives an introduction to a version 2 of the serial multiplier. This version does not use a counter to keep track of the number of bits added so far. Instead, the progress of the addition is kept track using the state diagram. Fig. 18 shows the state diagram where in each state the control unit outputs a different control signal.

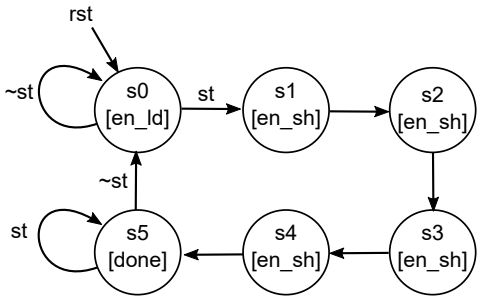

Figure 18: FSM for serial adder version 2.

Keep in mind that if we modify the adder to add more bits, we must add states to the FSM because we do not have a counter for keeping track of the number of bits added.

8 Building Blocks of Version 2

This section shows alternae versions of datapath components other than the full adder.

8.1 DFF with enable and load

The DFF module presented here follows the design in the textbook. The *D flip-flop with enable and load* (DFFEL) functions the same way as the DFFEC in Section 4.2 but it uses two 2:1 muxes.

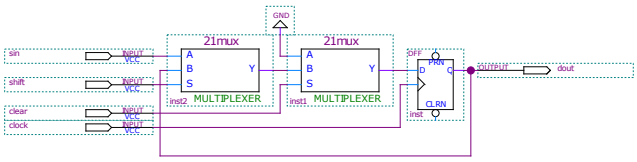

Figure 19: D flip-flop with enable and clear.

8.2 SIPO Shift Register

The alternate version of the serial-in-parallel-out (SIPO) shift register is built using 1-bit SIPO cell.

Four 1-bit cells are combined to build the 4-bit SIPO needed for the project.

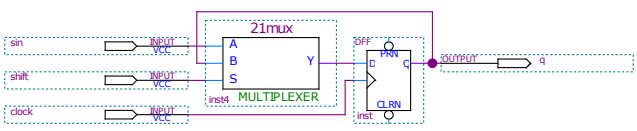

Figure 20: SIPO shift register cell.

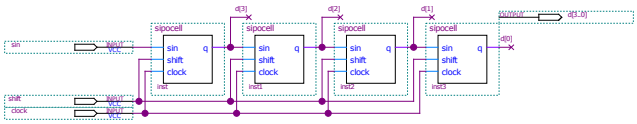

Figure 21: 4-bit SIPO shift register.

8.3 PISO Shift Register

The alternate version of the parallel-in-serial-output (PISO) registers are build using 1-bit PISO cells.

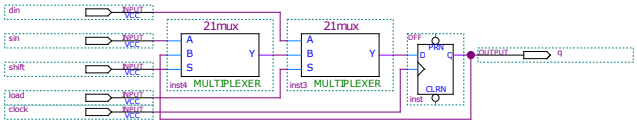

Figure 22: SIPO shift register cell.

Four 1-bit cells are combined to build the 4-bit PISO needed for the project.

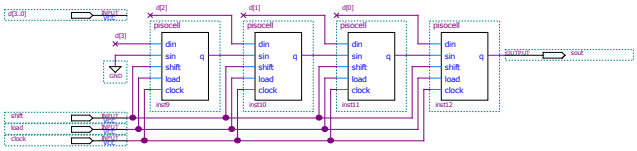

Figure 23: 4-bit PISO shift register.

9 Datapath Unit for Version 2

The datapath unit is almost the same as the standard version except for the missing counter. See Fig. 24.

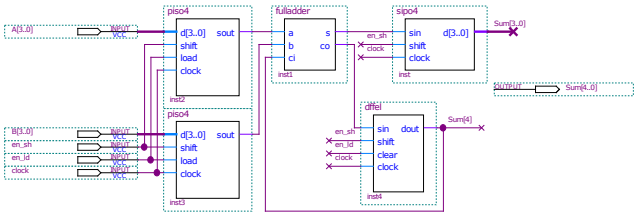

Figure 24: Serial adder alternate dapath unit.

The simulation waveform for the datapath unit is shown in Fig. 25.

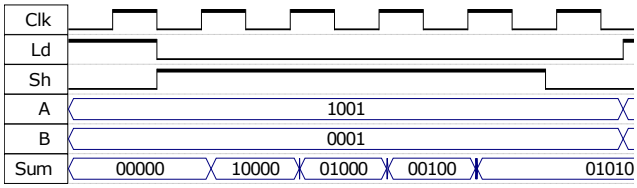

Figure 25: Simulation of dapath unit.

10 Control Unit for Version 2

The controller implements the FSM in Fig. 18. The control unit is implemented using the almost-one-hot state

assignment. We get the following next state equations:

$$S0 = \overline{S0} \cdot \overline{S1} \cdot \overline{S2} \cdot \overline{S3} \cdot \overline{S4} \cdot \overline{S5} + S0 \cdot \overline{st} + S5 \cdot \overline{st}$$
$$S1 = S0 \cdot st$$
$$S2 = S1$$
$$S3 = S2$$
$$S4 = S3$$
$$S5 = S4 + S5 \cdot st$$

The output equations are read directly from the state diagram:

$$en\_ld = S0$$
$$en\_sh = S1 + S2 + S3 + S4$$
$$done = S5$$

The controller unit produced from these equations is shown in Fig. 26.

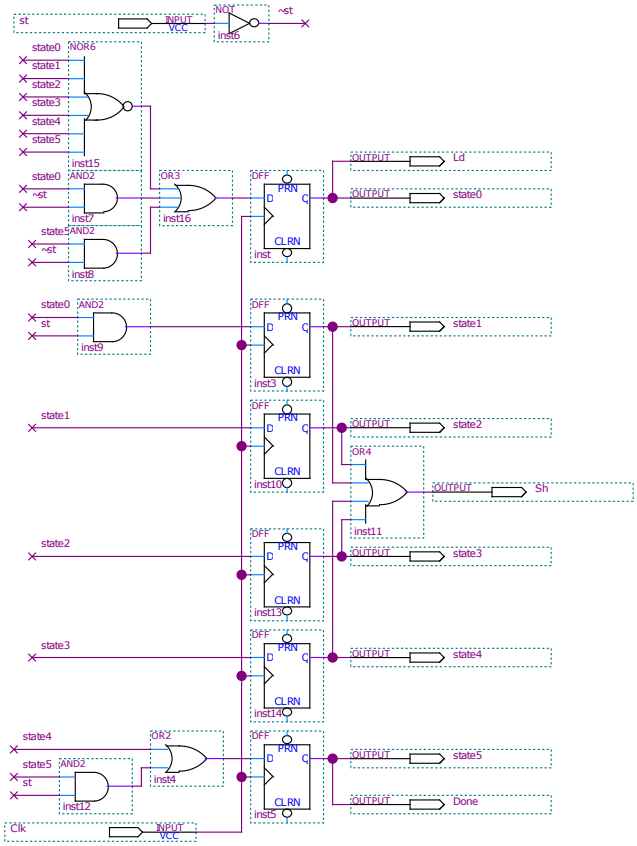

Figure 26: Control unit.

The entity level circuit is shown in Fig. 27. The simulation of the whole serial adder is shown in Fig. 28.

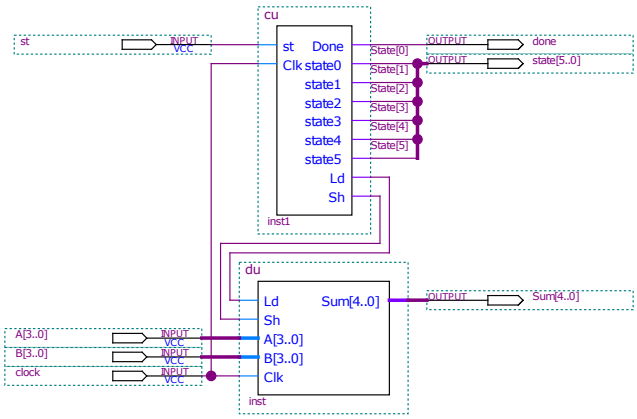

Figure 27: Serial adder top level entity.

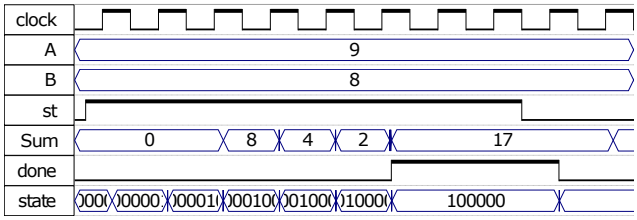

Figure 28: Simulation of serial adder for 8 + 15.

11 Conclusion

We hope what you learned from course can be applied in your future projects. For the SKEE2263 assignment, briefly reflect what you learned from assignment in the conclusion section.

12 References

List your references here. List at least four references in IEEE format.