UNIVERSITI TEKNOLOGI MALAYSIA

Serial Adder with Custom Modules

Mun'im Zabidi

Abstract
This documents gives directions forn implementing a serial adder in Altera Quartus using custom modules as building blocks.

[1] *Department of Electronic and Computer Engineering, School of Electrical Engineering, Faculty of Engineering*
*__Corresponding author__: munim@utm.my

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.

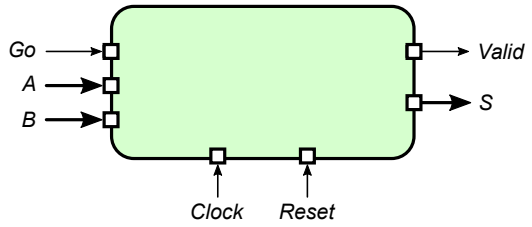

Figure 1. *Serial Adder entity.*

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.

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 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

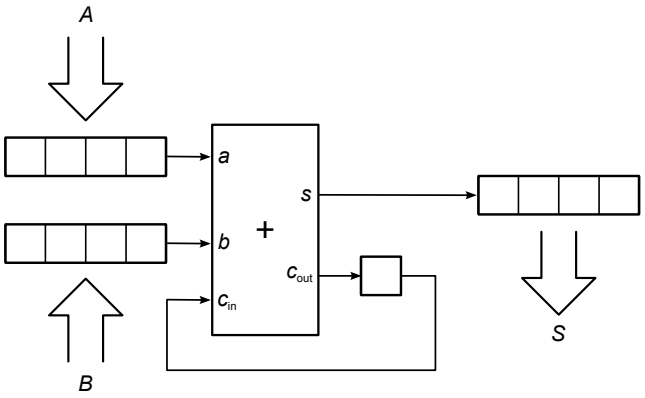

Figure 2. *Overview of the serial adder circuit.*

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

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.

This multiplier 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. 4 shows the state diagram where in each state the control unit outputs a different control signal.

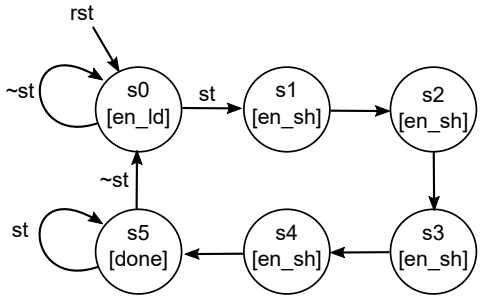

Figure 4. *FSM for serial adder.*

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.

4. Building Blocks

This section shows the bottom-up implementation of datapath components besides the full adder.

4.1 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. 5 has two additional controls:

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

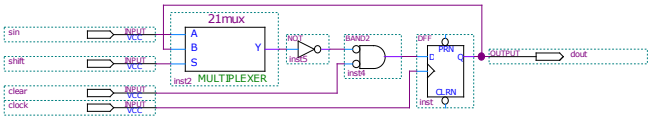

Figure 5. *D flip-flop with enable and clear.*

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.

4.2 DFF with enable and load

The *D flip-flop with enable and load* (DFFEL) functions the same way as the DFFEC in Section 4.1 but it uses two 2:1 muxes.

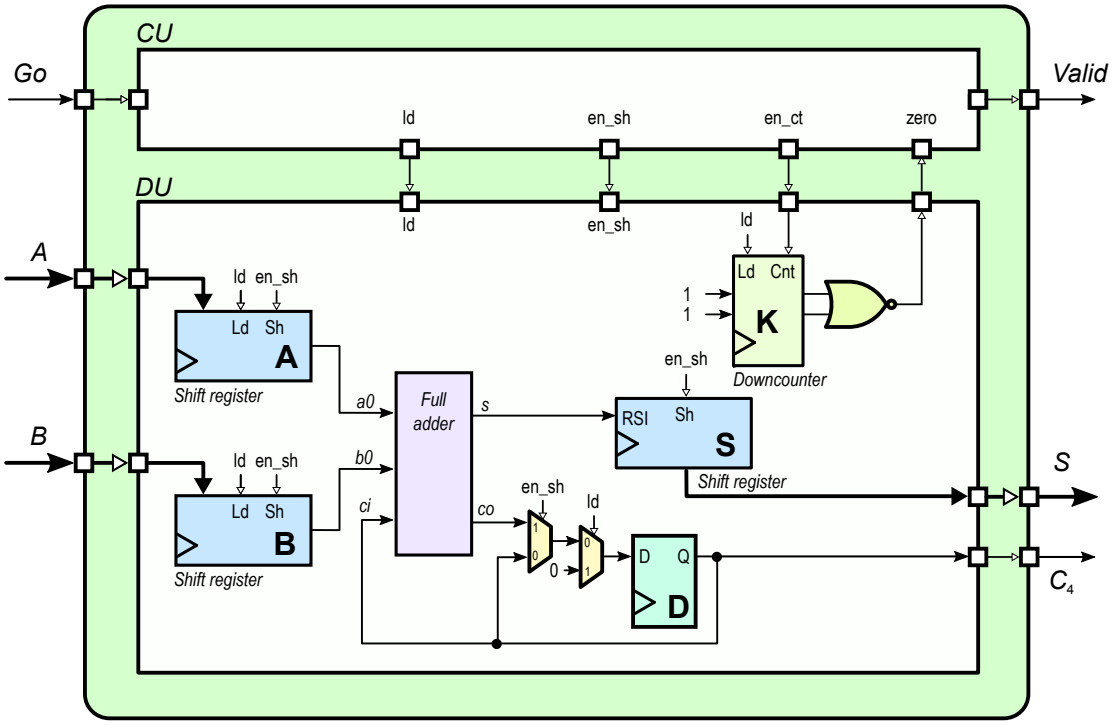

Figure 3. *Serial adder datapath.*

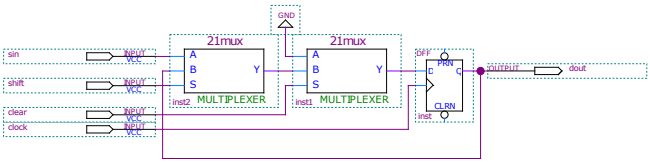

Figure 6. *D flip-flop with enable and clear.*

4.3 SIPO Shift Register

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

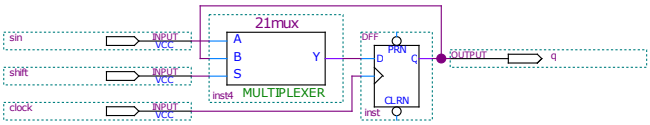

Figure 7. *SIPO shift register cell.*

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

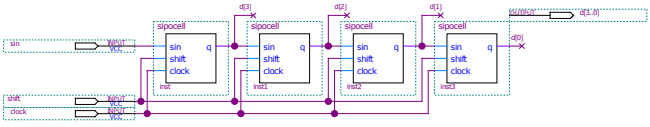

Figure 8. *4-bit SIPO shift register.*

4.4 PISO Shift Register

The parallel-in-serial-output (PISO) registers are built using 1-bit PISO cells.

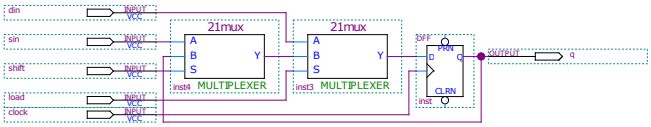

Figure 9. *SIPO shift register cell.*

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

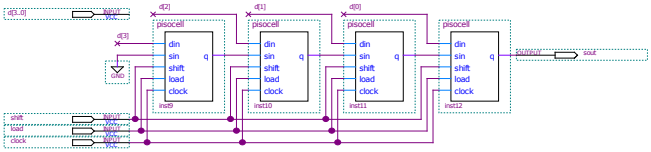

Figure 10. *4-bit PISO shift register.*

5. Datapath Unit

The datapath unit is almost the same as the standard version except for the missing counter. See Fig. 11.

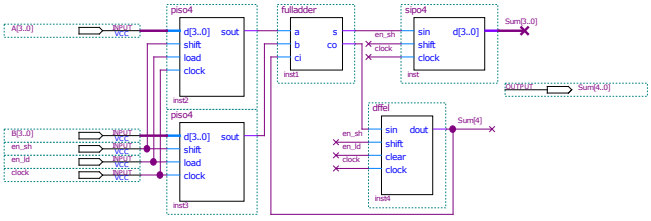

Figure 11. *Serial adder alternate dapath unit.*

The simulation waveform for the datapath unit is shown in Fig. 12.

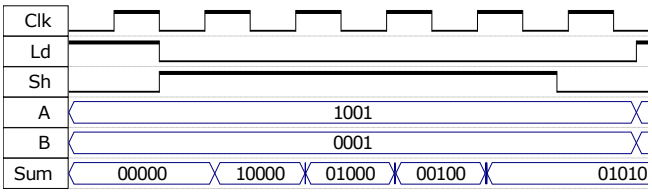

Figure 12. *Simulation of dapath unit.*

6. Control Unit

The controller implements the FSM in Fig. 4. 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. 13.

The entity level circuit is shown in Fig. 14. The simulation of the whole serial adder is shown in Fig. 15.

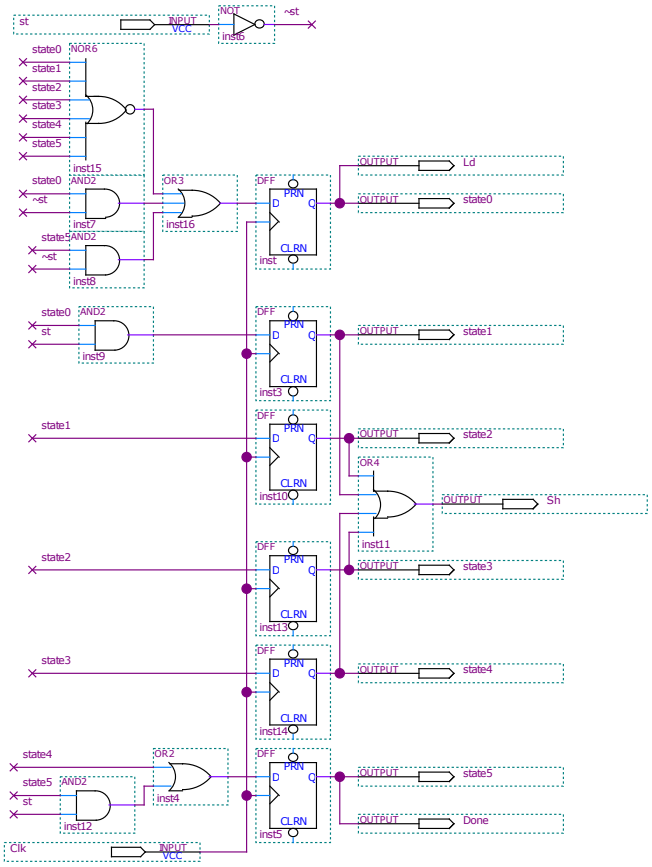

Figure 13. *Control unit.*

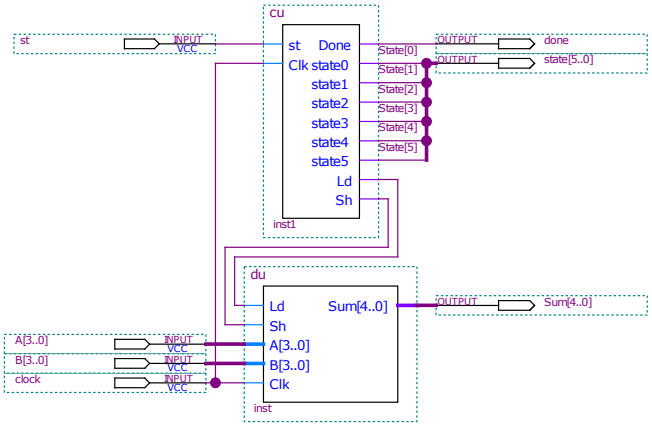

Figure 14. *Serial adder top level entity.*

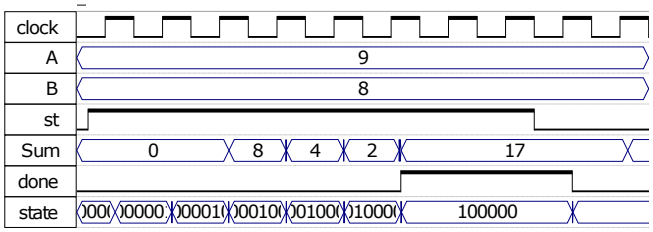

Figure 15. *Simulation of serial adder for 8 + 15.*

References

[1] *Designing State Machines for FPGAs*. Actel Corporation. Sept. 1997.

[2] Steve Golson. "One-hot state machine design for FPGAs". In: *Proc. 3rd Annual PLD Design Conference & Exhibit*. Vol. 1. 3. 1993.

[3] David A Huffman. "The synthesis of sequential switching circuits". In: *Journal of the Franklin Institute* 257.3 (1954).

[4] Munim Zabidi, Izam Kamisian, and Ismahani Ismail. *The Art of Digital Design*. 2019.

[5] *Introduction to the Quartus® II Software*. Version 10. Altera. 2010. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/intro_to_quartus2.pdf.

[6] Stephen Brown and Zvonko Vranesic. *Fundamentals of Digital Logic with Verilog Design*. 3rd ed. McGraw-Hill Education, 2013.