CS203: Digital Logic Design

Lab 6 Report

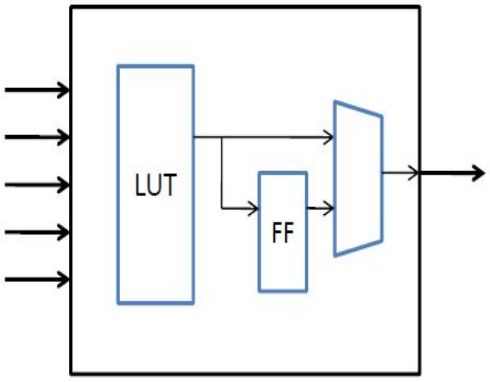
Name: Tarun Singla	**Entry Number:** 2019CSB1126
Lab IDs: I(4-bit Adder), IV(3:8 Decoder), VII(8-bit Universal Shift Register)	

In this lab, I have implemented three circuits using a single configurable logic/fabric. The fabric consists of two components: logic tile and switch box.

Components

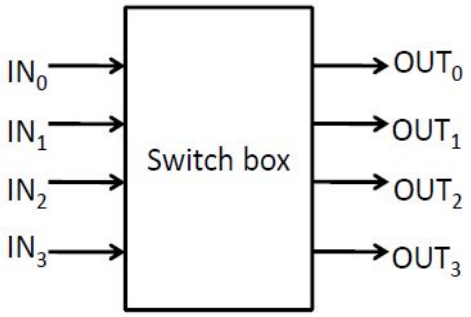
1. Logic Tile

The logic tiles are based on LUTs (Look-up-tables). Here, I have used five input LUTs. Overall, there is one output that can be synchronous or non-latched based on the configuration. The flip-flop used is a D flip-flop, and a clock is connected to the flip-flop. The D flip-flop also has a clear input. Using this logic tile, we can implement any five or fewer input function. The configuration data consists of 32 bits for the LUT and 1 bit for the mux.



2. Switch Box

I have used 4x4 switch boxes using which we can connect an input wire to any of the four output wires. There are four one-hot encoded muxes, one for each output, each with all four inputs. The configuration data consists of 16 bits, 4 bits for each mux.



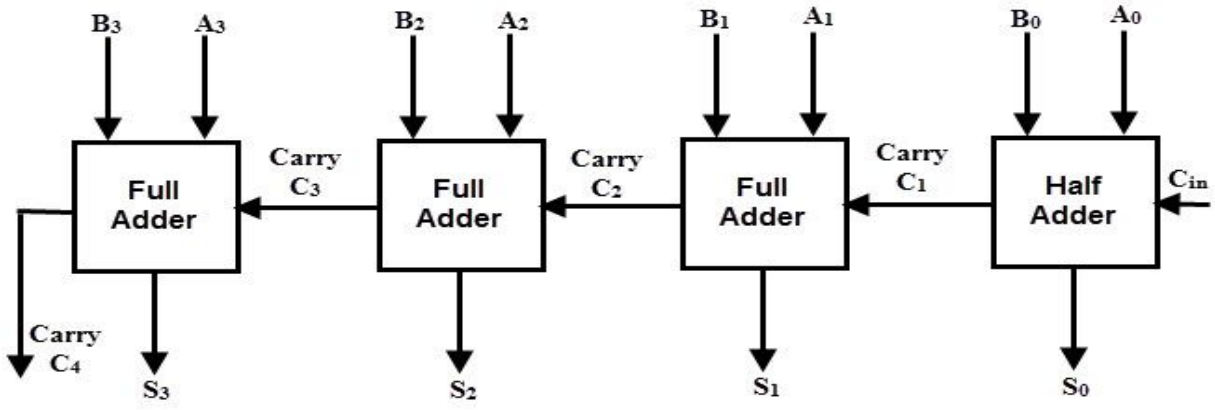
Circuits

1. 4-bit Adder

A 4-bit adder is used to add two 4-bit numbers. Here, we have implemented a 4-bit ripple carry adder. It consists of four full adders. Each full adder takes A, B, and C_{in} as inputs where A, B are a bit from 4-bit numbers and C_{in} is the carry from the previous stage. The outputs are S and C_{out}. They are given as:

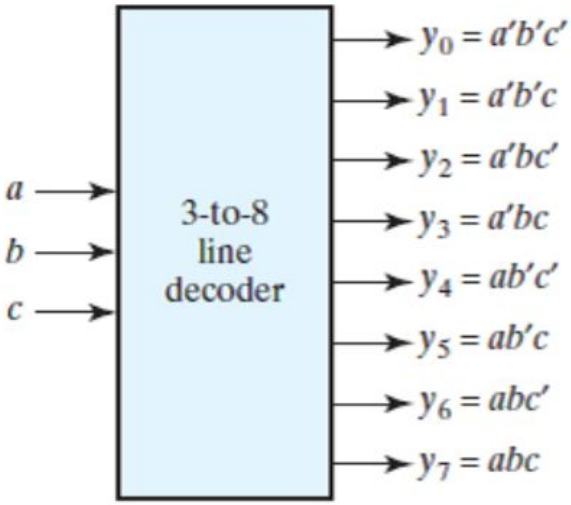
$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = A.B + B.C_{in} + C_{in}.A$$

where (.) is the AND operator, (+) is the OR operator, and $\oplus$ is the XOR operator.



2. 3:8 Decoder

A 3:8 decoder has three inputs and eight outputs. Based on the three inputs, one of the outputs is selected. The expression for each output is as shown in the figure. We can have an additional enable input also. Here, I have not used an enable input.



3. 8-bit Universal Shift Register

An 8-bit universal shift register can perform all operations(serial-in, serial-out, parallel-in, and parallel-out). The inputs are 8-bit number(a_i), s_i(serial-in), shl(shift left), shr(shift right), l(load), clear and clock. The outputs are the current values in the register and s_o which is the content of flip-flop at LSB position. It can be implemented using D flip-flops and muxes. The next state of a flip-flop(out of 8) in a register is given as:

$$Q_i^+ = (shl)'.(shr)'.(l)'.Q_i + (shl)'.(shr)'.(l).a_i + (shl)'.(shr).(l)'.Q_{i+1} + (shl).(shr)'.(l)'.Q_{i-1}$$

where Q_i^+ is the next state of the flip-flop into consideration, Q_{i-1} is the current state of the previous flip-flop, Q_{i+1} is the current state of the next flip-flop. Here, for flip-flop at the LSB position, Q_{i-1} is always taken as zero, and for flip-flop at MSB position, Q_{i+1} is taken as s_i.

Here, clearly the number of inputs to Q_i^+ are more than five. Therefore, I considered two variables as:

$$s = shl + shr;$$
$$Z = (shl)'.(shr).Q_{i-1} + (shl).(shr)'.Q_{i+1};$$

where s tells whether we have to shift or not because either both of shl and shr will be zero or any one of them will be one. Z tells us about the value to be stored in the flip-flop if we have to shift by considering both the cases, i.e., shift left and shift right.

The expression of Q_i^+ now reduces to:

$$Q_i^+ = (s)'.(l)'.Q_i + (s)'.(l).a_i + (s).(l)'.Z;$$

It is now clearly a five input function and can be implemented using a single logic tile after preprocessing as above.

Here, we have considered that only one of shl, shr, and l can be one in our implementation. However, all of these can be zero, in which case, the register will retain its value. When the clear input is zero, the register resets, and all flip-flops store zero value.

Fabric

All three circuits are divided into eight parts. The 4-bit adder has four parts for calculating the carry and four parts for calculating the sum bit. The 3:8 decoder has eight outputs; hence, it can be divided into eight parts. The 8-bit universal shift register also has eight parts, one for each flip-flop.

Each part looks similar to the following verilog implementation:

```
logic_tile lta_2(w[2], clock, clear, o3, o1, i10, i9, 0);
switch_box_4x4 sb2a({x, y, z, a[2]}, {1'b0, i1, w[2], i0});
switch_box_4x4 sb2b({x, y, z, b[2]}, {1'b0, i5, i2, i1});
switch_box_4x4 sb2c({x, y, z, c[2]}, {1'b0, out[1], out[2], i2});
switch_box_4x4 sb2d({x, y, e[2], d[2]}, {1'b0, 1'b0, i11, s});
logic_tile lt_2(out[2], clock, clear, a[2], b[2], c[2], d[2], e[2]);
switch_box_4x4 sb2e({x, y, z, o2}, {1'b0, 1'b0, out[4], out[2]});
```

The function of each logic tile and switch box in the above part is as follows:

1. The logic tile lta_2 calculates Z as described above, i.e., the value to be considered if we were to perform a shift operation (left or right).

2. The switch boxes sb2a, sb2b, sb2c, and sb2d, select inputs for the logic tile lt_2 in the following way:
    a. In the case of 3:8 decoder, three input bits will be selected.
    b. In the 4-bit adder case, two bits from the input and one bit of carry (either from another logic tile or from input) will be selected.
    c. In the 8-bit universal shift register case, two bits from the input (a_i, load, i.e., l), the current value of the flip-flop, Z, and s, will be selected. Here, s will be as shl + shr (as described above and calculated beforehand in another logic tile).

    ```
    logic_tile select(s, clock, clear, i9, i10, 0, 0, 0);
    ```

3. The logic tile lt_2 does the following operations:
    a. In the case of 3:8 decoder, it generates the output corresponding to its expression (out of 8).

    b. In the 4-bit adder case, it generates the carry or generates the sum based on the configuration.
    c. In the 8-bit universal shift register case, it does the operation based on load(l) and s inputs.

4. The last switch box, sb2e, assigns the output. In the case of 3:8 decoder and 8-bit universal shift register, the direct result of logic tile lt_2 is given. In the 4-bit adder case, each switch box's output is assigned accordingly since there are only five outputs.

There will be eight such parts in the fabric.

Conclusion

In this way, we have made the configurable fabric for performing three operations, meanwhile optimizing resource usage.

Note - Some of the images used above are taken from the internet.

Truth Tables

1. 4-bit Adder

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

2. 3:8 Decoder

A	B	C	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

3. 8-bit Universal Shift Register
    a. For calculating s (= shl + shr)

shl	shr	s
0	0	0
0	1	1
1	0	1
1	1	X

    b. For calculating Z

shl	shr	Q_{i-1}	Q_{i+1}	Z
0	0	0	0	X
0	0	0	1	X
0	0	1	0	X
0	0	1	1	X
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

c. For calculating Q_i^+

s	l	Q_i	a_i	Z	Q_i^+
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	1
0	1	1	1	1	1

s	l	Q_i	a_i	Z	Q_i^+
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	X
1	1	0	0	1	X
1	1	0	1	0	X
1	1	0	1	1	X
1	1	1	0	0	X
1	1	1	0	1	X
1	1	1	1	0	X
1	1	1	1	1	X