# EEE 2243
# Digital System Design
# Semester II 2011/12

## Tutorial 5

Data Flow Algorithm Design

1. Using only shifters, adders and multiplexers, draw the diagram of a data flow system to multiply 3 or 5 to a 4 bit input and give the result as a 6 bit value. If you are able to make any optimization to the original system, please also draw the optimized version. Write the Verilog code for both original and optimized system.

Multiply with 3 ($0011_2$) equals to adding a number with a copy of itself shifted 1 bit to the left:

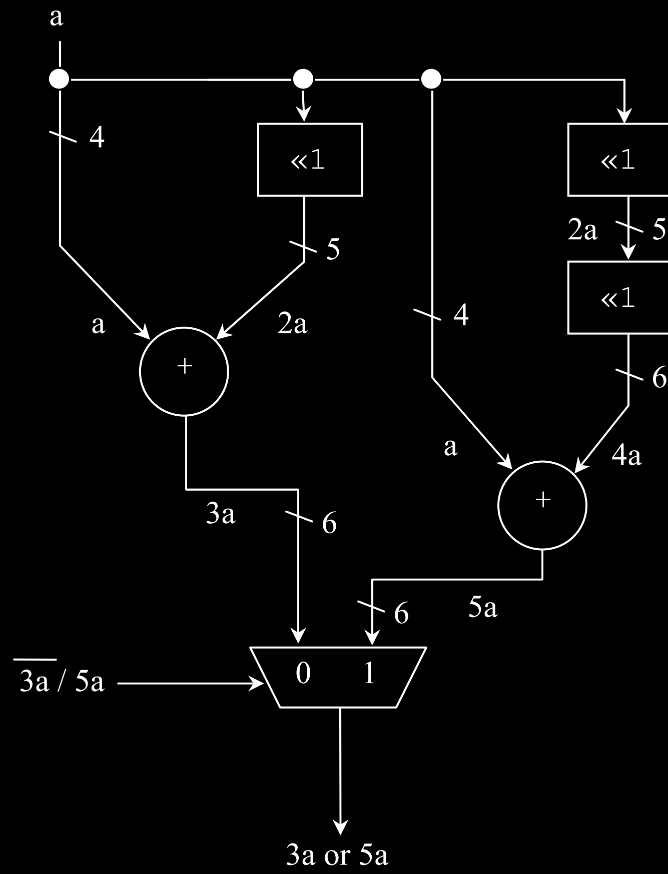
```
    XXXX
  × 0011
    ════
    XXXX
 + XXXX0
   ═════
   AAAAA
```

Multiply with 5 ($0101_2$) equals to adding a number with a copy of itself shifted 2 bits to the left:

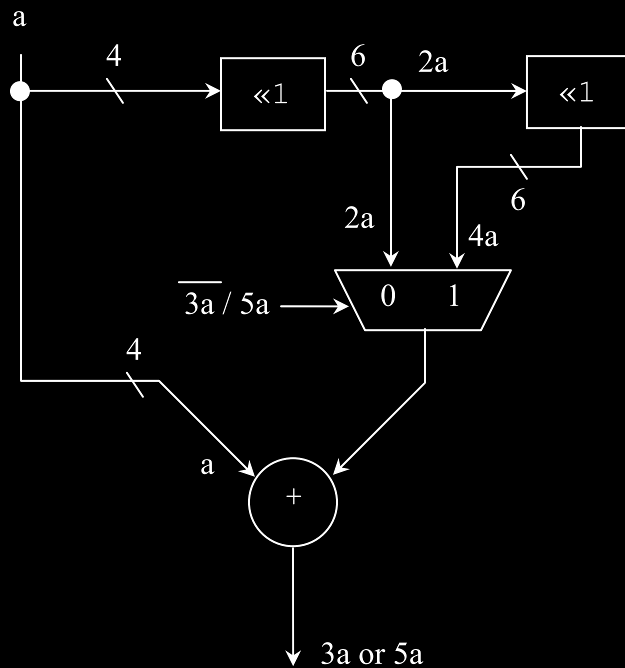
```
    XXXX
  × 0101
    ════
    XXXX
+ XXXX00
  ══════
  BBBBBB
```

We will need to construct a combined dataflow design for these 2 calculations.

Un-optimized solution:



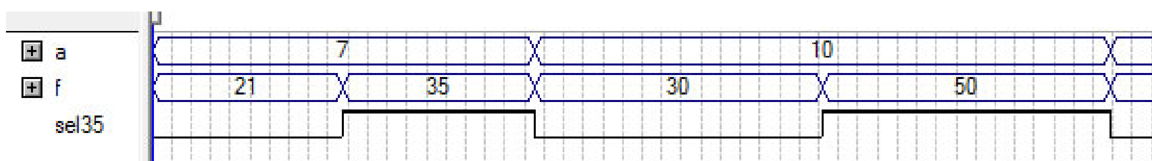
Optimization can be made by making the shifter of ×3 also the first shifter for ×5. The addition can then be made between the original value and the output of either the first shifter or the second one:




The optimized version reduced no. shifters from 3 to 2, and no. adders from 2 to 1.

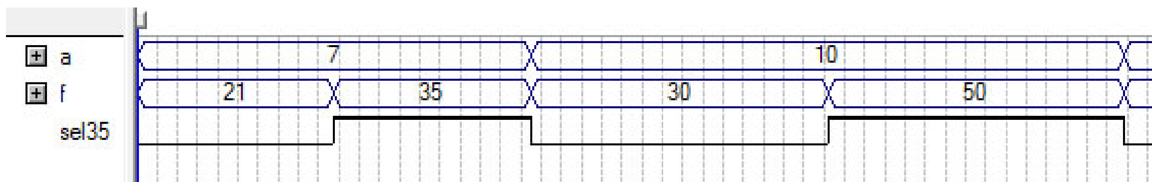
```
module NonOptimize35(a, sel35, f);

input [3:0] a;
input sel35;
output [5:0] f;
wire w;

assign w = a << 1;
assign f = (sel35) ?
           (a + (w << 1)) :
           (a + (a << 1));

endmodule
```

Simulation:



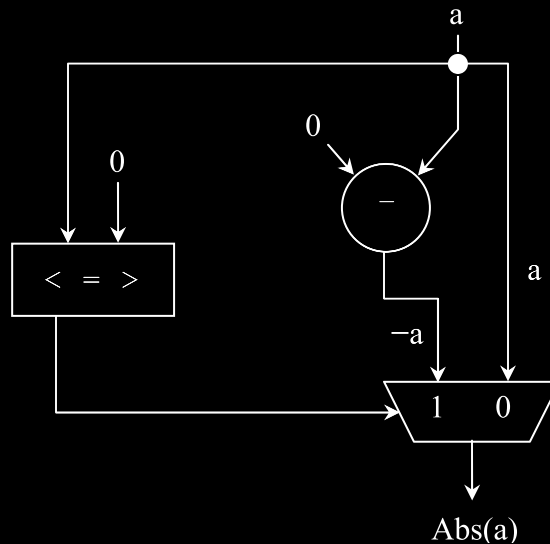
```
module Optimize35(a, sel35, f);

input [3:0] a;
input sel35;
output [5:0] f;
wire w;

assign w = a << 1;
assign f = a + ((sel35) ? (w + (a << 1)) : (w));

endmodule
```

Simulation:




2. Using any available Verilog operators in Quartus II platform, draw the diagram of a data flow system that is able to compute absolute value of an 8 bit 2's complement formatted input. Write the Verilog code.

Absolute value of an integer number, if it is positive or zero, is the same as the number's value. If it is negative, the absolute value is the positive part of it (equal to zero minus that number). The data flow implementation is then as follows:



```
module Absolute(a, abs);

input [7:0] a;
output [7:0] abs;

assign abs = (a < 0) ? (0-a) : a;

endmodule
```

Simulation:

3. Re-design the problem in question 2 using adder/subtractor circuit in page 51 of the slides for Chapter 4.

The adder/subtractor circuit can be operated as adder if the adder/subtractor control line is set to 0, or be operated as subtractor if the line is 1. Since the MSB value in a 2's complement number is 0 or 1 if it is positive or negative respectively, we can put that value to the control line to set the circuit to into an adder or subtractor respectively. Setting the circuit's control line to 0 (adder mode) and adding with 0 will keep a positive number unchanged. While setting the circuit's control line to 1 (subtractor mode) and subtracting from 0 will transform a negative number to its positive value. This is actually the process of computing the absolute value of a number. Obviously we can implement this by putting the MSB to the control line of the circuit:

MSB

7 6 5 4 3 2 1 0

0 0 0 0 0 0 0 0

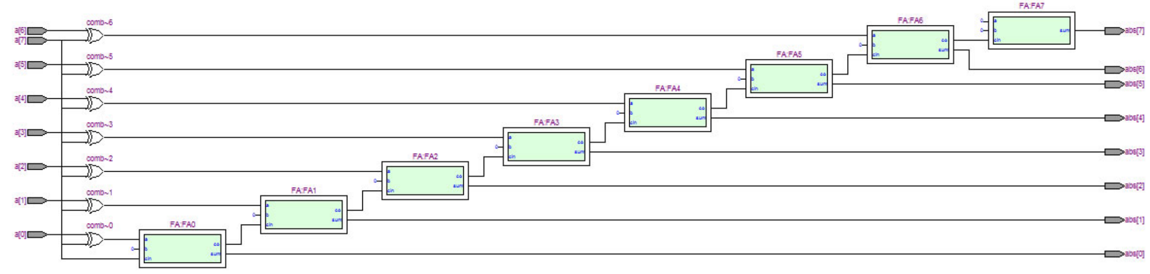
FA FA FA FA FA FA FA FA

Cin

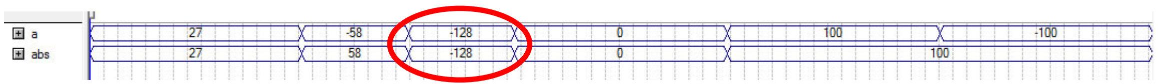
Abs[7] Abs[6] Abs[5] Abs[4] Abs[3] Abs[2] Abs[1] Abs[0]

```
module Absolute2(a, abs);

input [7:0] a;
output [7:0] abs;
wire [7:0] co;

FA FA0(.a(a[0]^a[7]), .b(0), .cin(a[7]),  .sum(abs[0]), .co(co[0]));
FA FA1(.a(a[1]^a[7]), .b(0), .cin(co[0]), .sum(abs[1]), .co(co[1]));
FA FA2(.a(a[2]^a[7]), .b(0), .cin(co[1]), .sum(abs[2]), .co(co[2]));
FA FA3(.a(a[3]^a[7]), .b(0), .cin(co[2]), .sum(abs[3]), .co(co[3]));
FA FA4(.a(a[4]^a[7]), .b(0), .cin(co[3]), .sum(abs[4]), .co(co[4]));
FA FA5(.a(a[5]^a[7]), .b(0), .cin(co[4]), .sum(abs[5]), .co(co[5]));
FA FA6(.a(a[6]^a[7]), .b(0), .cin(co[5]), .sum(abs[6]), .co(co[6]));
FA FA7(.a(a[7]^a[7]), .b(0), .cin(co[6]), .sum(abs[7]), .co(co[7]));

endmodule


module FA(a, b, cin, sum, co);

input a, b, cin;
output co, sum;

assign {co, sum} = a+b+cin;

endmodule
```

RTL View:



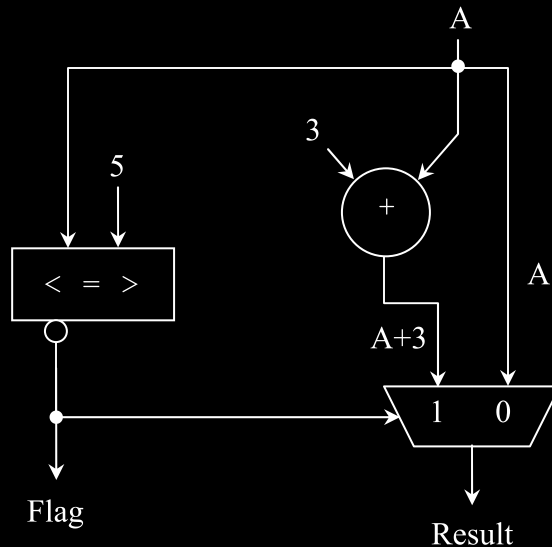
Simulation:




Can you explain this?

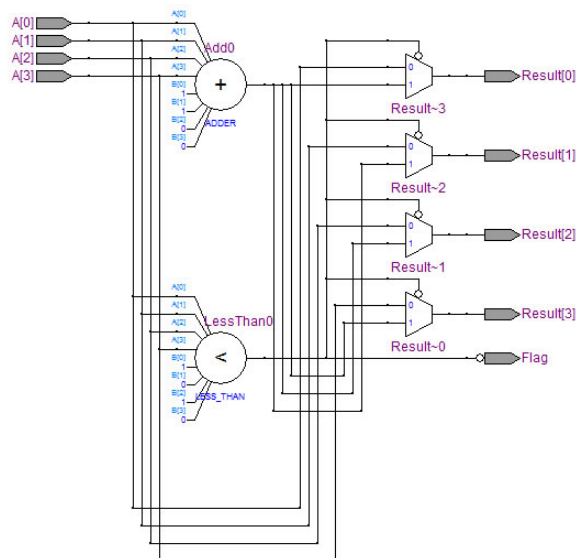
4. Using any available Verilog operators in Quartus II platform, draw the diagram of a data flow system that is able to add 3 to a 4 bit input if the input is greater than or equal to 5, otherwise nothing is done. Return the output as another 4 bit value. If the addition took place, a single bit flag will be set to 1 otherwise it is 0. Write the Verilog code as well.

[This is actually a design for binary to BCD digit-by-digit converter.] If we need a combination of 2 outputs from a comparator, like in “greater than” or “equal to”, the straightforward solution is that we can take the 2 outputs and combine using an OR gate. But a better solution would be to take the inverted output of “less than” (because NOT “less than” means “greater than” OR “equal to”).

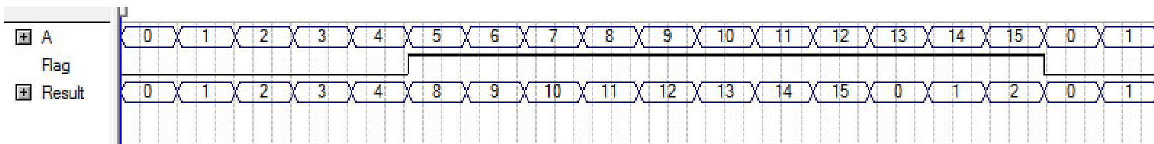



```
module Bin4Bit_to_BCD1Dgt(A, Result, Flag);

input [3:0] A;
output [3:0] Result;
output Flag;
wire not_lt;

assign not_lt = !(A < 5);
assign Result = not_lt ? (A+3) : A;
assign Flag = not_lt;

endmodule
```

RTL View:

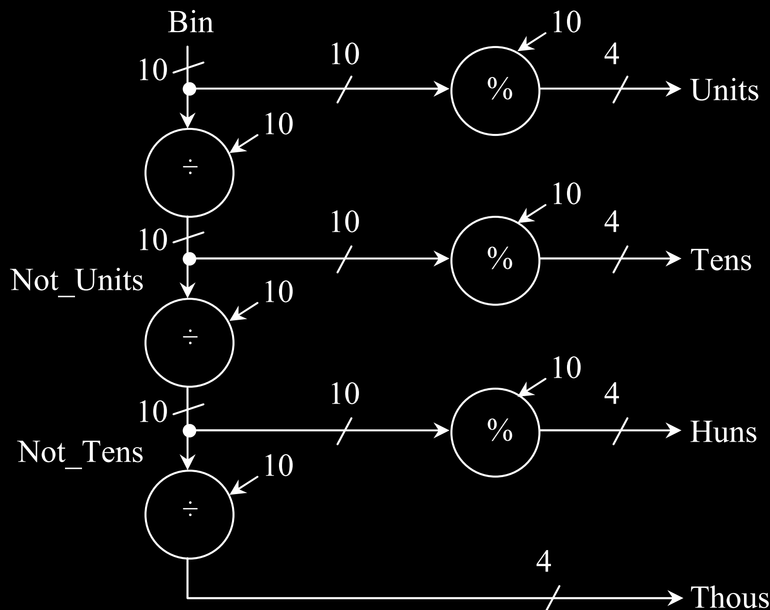


Simulation:



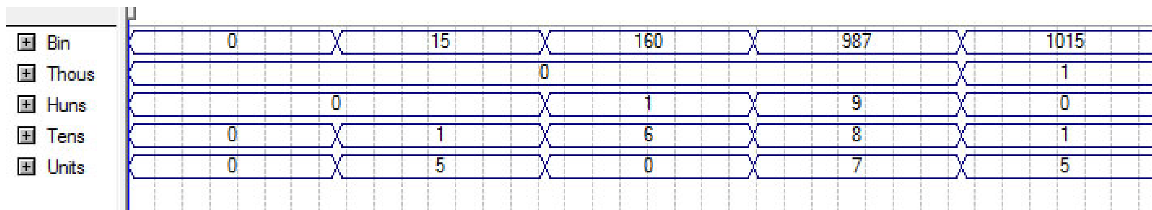

5. Using Verilog's division and modulus operators, draw the diagram of a data flow system that converts a 10 bit binary input into 4 digits BCD output. Each BCD digit is 4 bits. Write the Verilog code as well.

Binary numbers can be converted into BCD by repeatedly dividing it with 10 until we get a zero. This is similar to converting binary number into decimal. We keep the remainders as the decimal digits and re-write them from the last remainder to the first one (backward). For a 10-bit binary input, we just need to do it for 4 times (for 4 BCD digits) as the maximum value in decimal is 1023 (4 decimal digits).



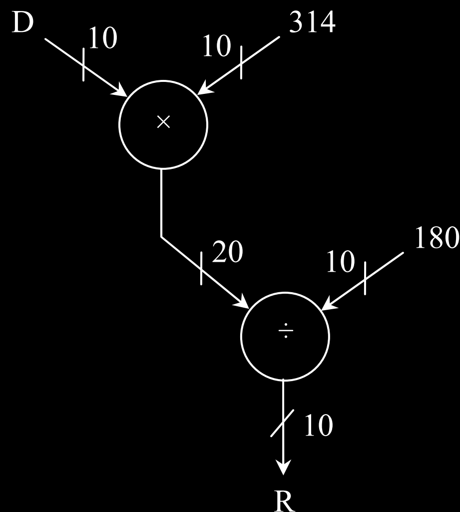

```
module Bin2BCD(Bin, Thous, Huns, Tens, Units);

input [9:0] Bin;
output [3:0] Units;
output [3:0] Tens;
output [3:0] Huns;
output [3:0] Thous;
wire [9:0] Not_Units;
wire [9:0] Not_Tens;

assign Units     = Bin%10;
assign Not_Units = Bin/10;
assign Tens      = Not_Units%10;
assign Not_Tens  = Not_Units/10;
assign Huns      = Not_Tens%10;
assign Thous     = Not_Tens/10;

endmodule
```

Simulation:




6. Angles, D, in degree can be converted into radian, R, using the formula R = πD/180. This formula cannot be implemented using the Verilog version in Quartus II version 9 that we are using. The reason is because π and the angle in radian are not integers. One workaround for this is by multiplying π with 100 (= 314) which will give us the value of R 100 times larger too. Draw the data flow diagram of the system that can perform the function: Input of 10 bit binary data representing an angle in degree, then output its corresponding radian value multiplied 100 times. Write the Verilog code as well. Decide the best size for the output in bits.

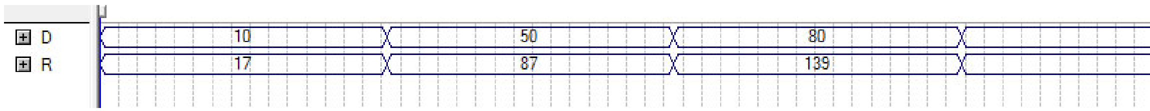
The value of degree will be restricted to integer values in 10 bit. No bit reduction division is needed as the multiplication is followed by the division by 180. The best no. bits at output is 10 (= 20−10).




```
module Deg2Rad(D, R);

input [9:0] D;
output [9:0] R;

assign R = (D*314)/180;

endmodule
```

The above code works only for positive values as the division module used in Quartus II works only with positive operands.

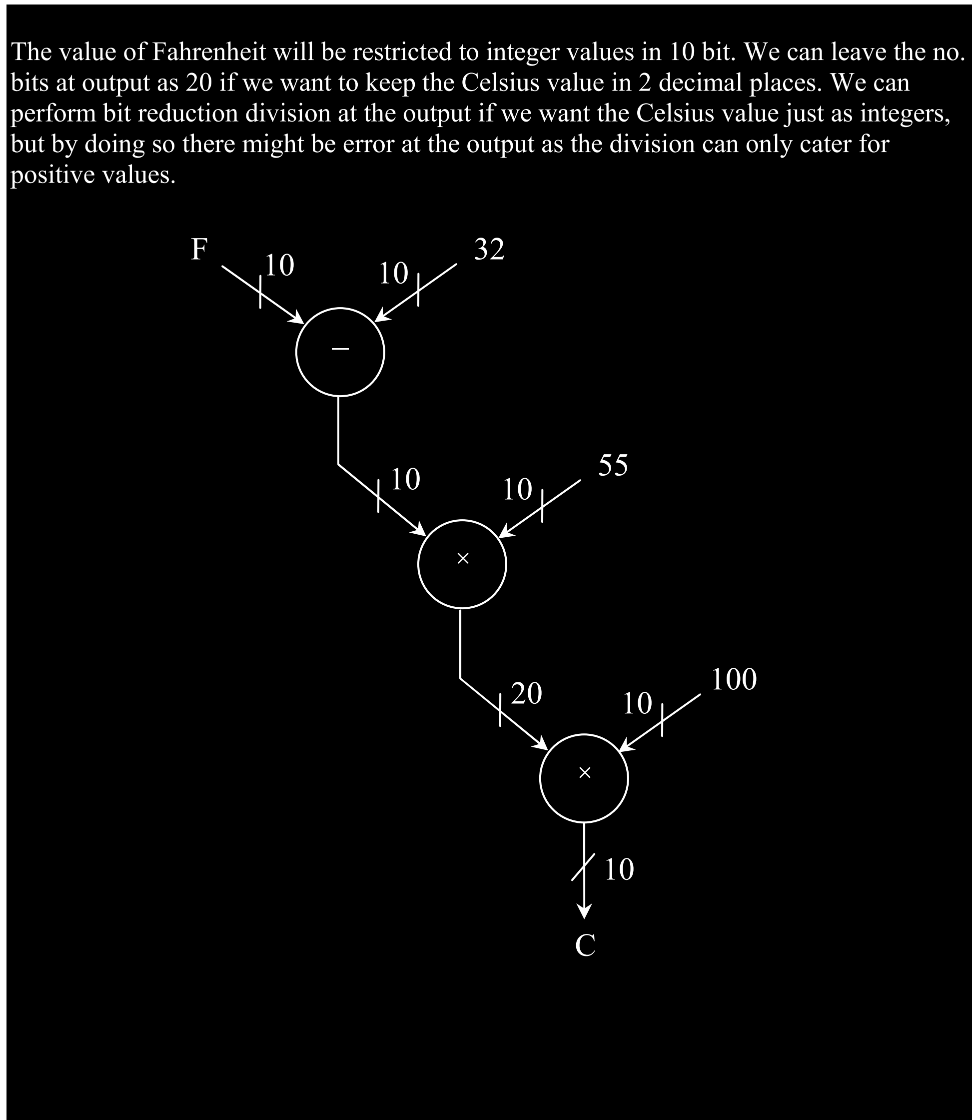
Simulation (the value of radian is 100 times larger):




7. Repeat a similar workaround as in question 6 for the problem of converting temperature in Fahrenheit into Celsius,

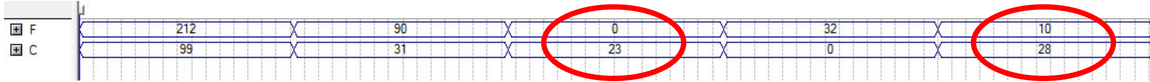
$$
\begin{aligned}
C &= (F - 32) \times \frac{5}{9} \\
&= (F - 32) \times 0.55
\end{aligned}
$$

in 2 decimal fraction digits.



```
module Fah2CelInt(F, C);

input [9:0] F;
output [9:0] C;

assign C = ((F-32)*55)/100;

endmodule
```

Simulation for the integer Celsius code (error when division involves negative values):




```
module Fah2Cel(F, C);

input [9:0] F;
output [19:0] C;

assign C = (F-32)*55;

endmodule
```

Simulation for the fraction Celsius code (Celsius is 100 times larger):