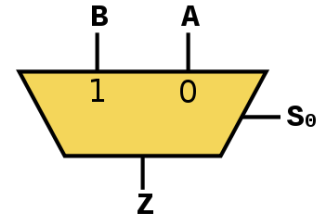


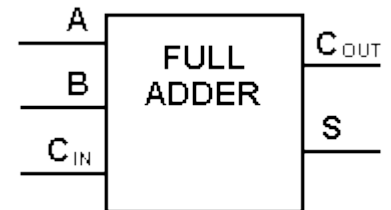
Pre-lab Questions

- Q1.** Consider the logic statement $(A*B)+(C*A')$
- Write the truth table for that logic statement
 - Draw the gates that implement that logic statement (without simplification).

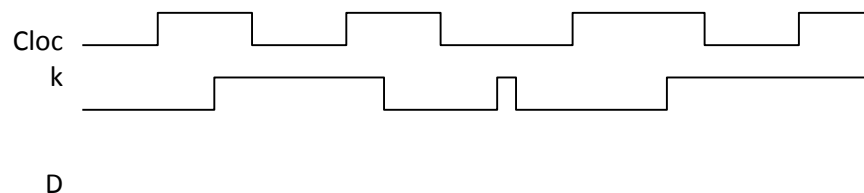
- Q2.** Consider a 2-to-1 MUX as drawn on the right.
- Write a truth table for this circuit.
 - Using standard 2-input gates (AND, OR, XOR) and the NOT gate, draw a circuit which implements the same function (has the same truth table).



- Q3.** Consider a full-adder as drawn on the right.
- Write a truth table for this circuit.
 - Using standard 2-input gates (AND, OR, XOR) and the NOT gate, draw a circuit which implements the same function (has the same truth table).



- Q4.** Using an assign statement, implement the following (including declaring all outputs)
- $X=(A*B)+C'+(B*D)'$
 - The logic to implement the 2-to-1 MUX. Use A, B, S₀ as the inputs and Z as the output.
- Q5.** Using the adder in the tutorial as a template, implement
- A module implementing a 2-to-1 MUX called MUX21 which takes three inputs (A,B,S) and generates one output (Z).
 - A module implementing a 4-to-1 MUX called MUX41 which takes 6 bits of input (A[3:0], S[1:0]) and generates one output (Z). It should do all its work by instantiating the 2-to-1 MUX of part a.
 - A module implementing a 4-to-2 MUX called MUX42 which takes 5 bits of input (A[1:0], B[1:0], S) and generates two bits of output (Z[1:0]). It should do all its work by instantiating the 2-to-1 MUX of part a.
- Q6.** Redo problem **Q4.a** using an `always @*` block.
- Q7.** Neatly complete the following timing diagram for a D flip-flop. Indicate that the value is unknown before the first rising edge in any way you see fit.



- Q8.** Using full adders and D flip-flops as building blocks, draw a 2-bit modulo counter. That is, on each rising edge the counter's output value is incremented by 1, but where 3 wraps around to 0 (so it is a modulo-4 counter). You do not need to worry about reset.
- Q9.** Answer the following questions about the state diagram in figure 2 from the tutorial on the website.
- How many bits are used to store the state? How many are the least that could have been used?
 - Section 3.2 in the tutorial has two implementations of the next state logic. What are the pros and cons of each?
- Q10.** Draw the state transition diagram which is implemented by the following code. Your diagram should resemble figure 2a from the tutorial. You need not include reset.

```

module statem(clk, in, reset, out);

input clk, in, reset;
output [3:0] out;

reg [3:0] out;
reg [1:0] state;
reg [1:0] next_state;

parameter zero=2'd0, one=2'd1, two=2'd2, three=2'd3;

always @*
begin
case (state)
zero:
begin
out = 4'b0000;
next_state = one;
end
one:
begin
out = 4'b0001;
if(in)
next_state = one;
else
next_state = two;
end
two:
begin
out = 4'b0010;
if(in)
next_state = one;
else
next_state = three;
end
three:
begin
out = 4'b0100;
next_state = zero;
end
default:
begin
out = 4'b0000;
next_state = zero;
end
end
end

```

```
                end
            endcase
        end

        always @(posedge clk)
            begin
                if (reset)
                    state <= zero;
                else
                    state <= next_state;
                end
            end
        endmodule
```

- Q11.** Consider the above Verilog code and respond to the questions/requests below.
- Provide the above code with the blanks correctly filled in.
 - At present the values for the hex displays are listed in binary while the labels for the case statements are in hex. Why do you think that was done?
- Q12.** Fill in the blank in the MATLAB code.
- What would be the code to calculate the FTV for each sinusoid
 - Write down the FTV values you calculated for this set-up of the code
- Q13.** Change the order of your Fourier synthesis to 20 and attach the plot to your pre-lab report.

In-lab & Post-lab Questions

- Q1.** What output do you get if, after reset, you input a “1010”?
- Q2.** Observe the output when only SW0 is a “1”.
- Measure the frequency of the waveform it generates.
 - Is there anything odd or unexpected about the waveform?
- Q3.** Now look over the code.
- What is the value of FTV for the situation you observed in **Q2**?
 - What is the sampling rate?
 - Compute the expected frequency for the case you observed in **Q2**. Show your work.
- Q4.** What is the lowest frequency signal you can now generate? The highest?
- Q5.** What’s the biggest difference between the two spectra you see? Your answer should discuss a certain kind of noise.
- Q6.** Observe the output when only SW2 is a “1”. About how many periods do you observe?
- Q7.** Now look over the code.
- Notice we are using the same DDS as in the audio part however the range of the values outputted to the VGA is completely different (it’s now from 40 to 450 vs. -32768 to 32767 in the audio case). Find the code that does this conversion and explain what’s going on.
 - Notice that VGA_R, VGA_B and VGA_G are all 10 bits but only take the values 0 and -1. What color do you suppose is associated with each of those two?
- Q8.** Print out your code for the sawtooth and turn it in with your other questions.
- Q9.** You likely noticed that the waveform generated by the table had some unusual characteristics when put on the oscilloscope.
- At lower frequencies you may have seen a bit of a slope on some lines you’d have expected to be straight
 - At lower frequencies the corners were not overly sharp.
 - At higher frequencies the amplitude was attenuated.
 - In some cases the output appears to saturate.
Why do you think these effects are occurring? (Note: there may be different reasons for each and perhaps multiple reasons for some. This is a tricky question.)
- Q10.** Read a bit about VGA (<http://www.nathandumont.com/node/241> and perhaps <http://www.eecg.toronto.edu/~tm4/rgbout.html> may be useful).
- What is a “porch” in this context?
 - Explain why this version of the DDS module has “lineend”.
 - Looking at the code Q7, explain why the output doesn’t seem to start at the beginning of the sine wave.