Undergraduate Research

Today 3:30-5pm @ Tucker Hall → Summer Research Info Session.

Today 7-8pm @ Tucker Hall → Graduate School Fellowships.

More info on-line at:

http://web.wm.edu/charlescenter/FOURweek08.php

The summer between Junior and Senior year is the best time to get into an REU summer program !!!

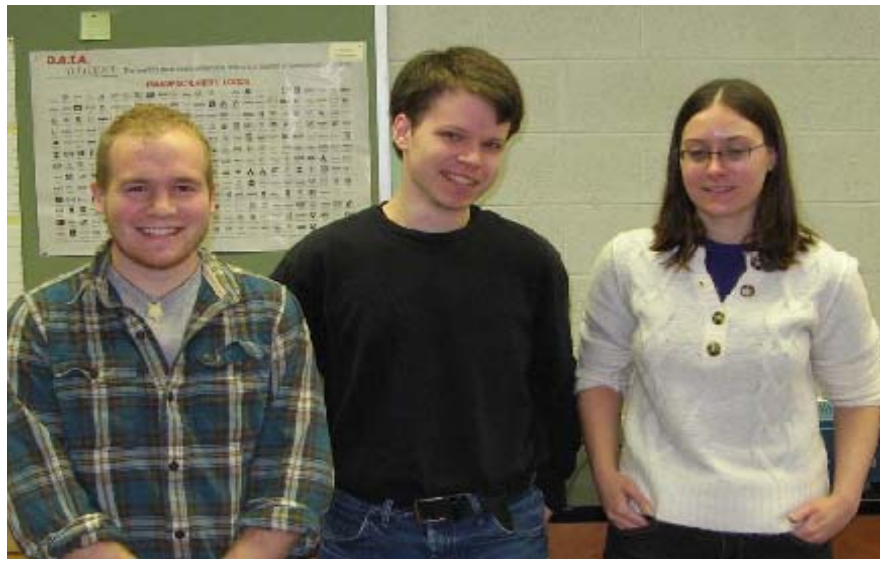
Project Teams

Reminder:

Choose project teams on or before lab on Wednesday.

Max number of teams: 5

New budget: $200/team

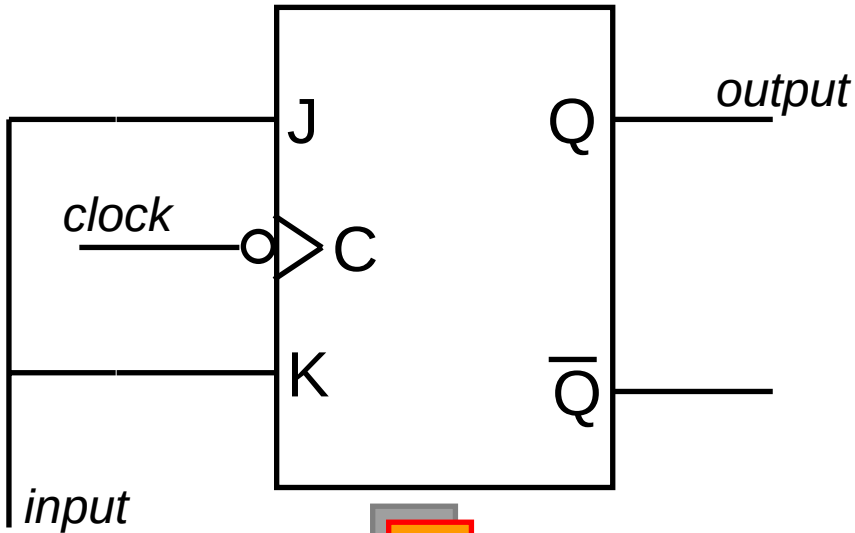


Last year's DSP Design Competition winners
Trevor Harrison, William Ames, and Sarah Mohon

Current teams:

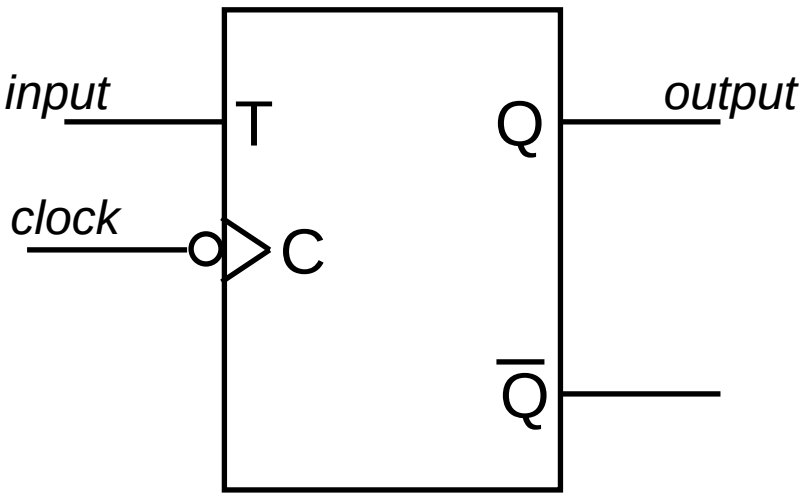
- Jordan Gates, Ryan Zielinski, Justin Vazquez.

T-type flip-flop



JK Logic table

J	K	Q_{n+1}
0	0	Q_n
1	0	0
0	1	1
1	1	$\overline{Q_n}$



T Logic table
(clock falling edge)

T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

T-type flip-flops are used in counters.

Multiplexers

What's a multiplexer ?

A multiplexer is a generalized multi-input and multi-output gate. It will produce a specific multiple line output for each specific multiple line input.

Example: 3-line input and 4-line output (i.e. 3-to-4 multiplexer).

A	B	C	1	2	3	4
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	1	0	0
1	0	0	1	0	0	0
1	1	0	1	1	0	0
0	1	1	0	1	1	0
1	0	1	1	0	1	0
1	1	1	0	0	0	1

N.B. Multiplexers can be very useful for converting a binary number to a HEX display code.

Multiplexers with FPGAs (Ia)

You could build a multiplexer out of logic gates using Karnaugh maps … or you could let the Verilog compiler figure it out.

N-to-1 multiplexer:

mux_primitive.v

```
1  primitive mux_primitive(out1,A,B,C);    // 3-to-1 multiplexer
2      input A, B, C;       // 3 input wires
3      output out1;         // 1 output wire
4
5      table                // table defines the output based on the 3 inputs
6      // A B C out1
7      0 0 0 : 0 ;
8      0 0 1 : 0 ;
9      0 1 0 : 0 ;
10     1 0 0 : 1 ;
11     1 1 0 : 1 ;
12     0 1 1 : 0 ;
13     1 0 1 : 1 ;
14     1 1 1 : 0 ;
15     endtable
16
17 endprimitive
18
```

Multiplexers with FPGAs (IIa)

An always block with “if” statements can be used for an N-to-M multiplexer:

mux_always_if.v

```
1  module mux_always_if(input_3bit, output_4bit);  // 3-to-4 multiplexer
2      input [2:0] input_3bit;          // 3 input lines (bits)
3      output reg [3:0] output_4bit;    // 4-bit output register
4
5  // always block with "if" statement for each input case
6      always
7          begin
8              if(input_3bit == 3'b000) output_4bit <= 4'b0001;
9              if(input_3bit == 3'b001) output_4bit <= 4'b0010;
10             if(input_3bit == 3'b010) output_4bit <= 4'b0100;
11             if(input_3bit == 3'b100) output_4bit <= 4'b1000;
12             if(input_3bit == 3'b110) output_4bit <= 4'b1100;
13             if(input_3bit == 3'b011) output_4bit <= 4'b0110;
14             if(input_3bit == 3'b101) output_4bit <= 4'b1010;
15             if(input_3bit == 3'b111) output_4bit <= 4'b0001;
16         end
17 endmodule
18
```

An always block guarantees that you won’t have any signal races or glitches.

Multiplexers with FPGAs (III)

An always block with “case” constructs can be used for an N-to-M multiplexer:

```
1  module mux_always_case(input_3bit,output_4bit);     // 3-to-4 multiplexer
2      input [2:0] input_3bit;          // 3-bit input
3      output reg [3:0] output_4bit;    // 4-bit output
4
5      always      // always block with "case" construct
6          begin
7              case(input_3bit)
8              3'b000: output_4bit <= 4'b0001;
9              3'b001: output_4bit <= 4'b0010;
10             3'b010: output_4bit <= 4'b0100;
11             3'b100: output_4bit <= 4'b1000;
12             3'b110: output_4bit <= 4'b1100;
13             3'b011: output_4bit <= 4'b0110;
14             3'b101: output_4bit <= 4'b1010;
15             3'b111: output_4bit <= 4'b0001;
16             endcase
17         end
18
19 endmodule
```

Multiple Modules

multiple_modules.v

```
1  module multiple_modules(input_clock, output_FourBits);  // top-level module
2      input input_clock;                    // input wire
3      output [3:0] output_FourBits;         // output wires
4
5      wire [2:0] counter_output;            // output wires of counter
6      wire [2:0] mux_input;                 // input wires of multiplexer
7
8      assign mux_input = counter_output;  // connect the counter output and multiplexer input wires
9
10     counter counter_result(input_clock, counter_output);     // call the "counter" module
11                                                              // with instance "counter_result"
12
13     mux_always_case mux_output(mux_input, output_FourBits); // call the "mux_always_case" module
14                                                              // with instance " mux_output"
15
16 endmodule
17
```

counter.v

```
1  module counter(input_clk, output_3bit);
2      input input_clk;
3      output reg [2:0] output_3bit;
4
5      always@(posedge input_clk)
6          begin
7          output_3bit <= output_3bit + 3'b001;
8          end
9
10 endmodule
11
```

mux_always_case.v*

```
1  module mux_always_case(input_3bit,output_4bit); // 3-
2      input [2:0] input_3bit;          // 3-bit input
3      output reg [3:0] output_4bit;    // 4-bit output
4
5      always      // always block with "case" construct
6          begin
7              case(input_3bit)
8              3'b000: output_4bit <= 4'b0001;
9              3'b001: output_4bit <= 4'b0010;
10             3'b010: output_4bit <= 4'b0100;
11             3'b100: output_4bit <= 4'b1000;
12             3'b110: output_4bit <= 4'b1100;
13             3'b011: output_4bit <= 4'b0110;
14             3'b101: output_4bit <= 4'b1010;
15             3'b111: output_4bit <= 4'b0001;
16             endcase
17         end
18
19 endmodule
20
```

Multiple Modules

multiple_modules.v

```
1  module multiple_modules(input_clock, output_FourBits);  // top-level module
2      input input_clock;                    // input wire
3      output [3:0] output_FourBits;         // output wires
4
5      wire [2:0] counter_output;            // output wires of counter
6      wire [2:0] mux_input;                 // input wires of multiplexer
7
8      assign mux_input = counter_output;    // connect the counter output and multiplexer input wires
9
10     counter counter_result(input_clock, counter_output);     // call the "counter" module
11                                                              // with instance "counter_result"
12
13     mux_always_case mux_output(mux_input, output_FourBits); // call the "mux_always_case" module
14                                                              // with instance " mux_output"
15
16 endmodule
17
```

Wires for connecting the 2 lower level modules

module name instance name

counter.v

```
1  module counter(input_clk, output_3bit);
2      input input_clk;
3      output reg [2:0] output_3bit;
4
5      always@(posedge input_clk)
6          begin
7          output_3bit <= output_3bit + 3'b001;
8          end
9
10 endmodule
11
```

mux_always_case.v*

```
1  module mux_always_case(input_3bit,output_4bit); // 3-
2      input [2:0] input_3bit;          // 3-bit input
3      output reg [3:0] output_4bit;    // 4-bit output
4
5      always      // always block with "case" construct
6          begin
7              case(input_3bit)
8              3'b000: output_4bit <= 4'b0001;
9              3'b001: output_4bit <= 4'b0010;
10             3'b010: output_4bit <= 4'b0100;
11             3'b100: output_4bit <= 4'b1000;
12             3'b110: output_4bit <= 4'b1100;
13             3'b011: output_4bit <= 4'b0110;
14             3'b101: output_4bit <= 4'b1010;
15             3'b111: output_4bit <= 4'b0001;
16             endcase
17         end
18
19 endmodule
20
```

Multiple Modules

multiple_modules.v

```
1  module multiple_modules(input_clock, output_FourBits);  // top-level module
2      input input_clock;                                  // input wire
3      output [3:0] output_FourBits;                       // output wires
4
5      wire [2:0] counter_output;                          // output wires of counter
6      wire [2:0] mux_input;                               // input wires of multiplexer
7
8      assign mux_input = counter_output;  // connect the counter output and multiplexer input wires
9
10     counter counter_result(input_clock, counter_output);    // call the "counter" module
11                                                             // with instance "counter_result"
12
13     mux_always_case mux_output(mux_input, output_FourBits); // call the "mux_always_case" module
14                                                             // with instance " mux_output"
15
16 endmodule
17
```

Warning: DO NOT make these registers !!!

module name instance name

counter.v

```
1  module counter(input_clk, output_3bit);
2      input input_clk;
3      output reg [2:0] output_3bit;
4
5      always@(posedge input_clk)
6          begin
7          output_3bit <= output_3bit + 3'b001;
8          end
9
10 endmodule
11
```

mux_always_case.v*

```
1  module mux_always_case(input_3bit,output_4bit); // 3-
2      input [2:0] input_3bit;          // 3-bit input
3      output reg [3:0] output_4bit;    // 4-bit output
4
5      always      // always block with "case" construct
6          begin
7              case(input_3bit)
8              3'b000: output_4bit <= 4'b0001;
9              3'b001: output_4bit <= 4'b0010;
10             3'b010: output_4bit <= 4'b0100;
11             3'b100: output_4bit <= 4'b1000;
12             3'b110: output_4bit <= 4'b1100;
13             3'b011: output_4bit <= 4'b0110;
14             3'b101: output_4bit <= 4'b1010;
15             3'b111: output_4bit <= 4'b0001;
16             endcase
17         end
18
19 endmodule
20
```