#### **Undergraduate Research**

Today 3:30-5pm @ Tucker Hall  $\rightarrow$  Summer Research Info Session.

Today 7-8pm @ Tucker Hall  $\rightarrow$  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



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).

А	В	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 (la)**

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	<pre>primitive mux_primitive</pre>	(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	000:0;	
8	001:0;	
9	010:0;	
10	100:1;	
11	110:1;	
12	011:0;	
13	101:1;	
14	111:0;	
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	<pre>module mux_always_if(input_3bit, output_4bit); // 3-to-4 multiplexer</pre>
2	<pre>input [2:0] input_3bit; // 3 input lines (bits)</pre>
3	<pre>output reg [3:0] output_4bit; // 4-bit output register</pre>
4	
5	// always block with "if" statement for each input case
6	always
7	E begin
8	if(input_3bit == 3'b000)
9	<pre>if(input_3bit == 3'b001) output_4bit &lt;= 4'b0010;</pre>
10	<pre>if(input_3bit == 3'b010) output_4bit &lt;= 4'b0100;</pre>
11	<pre>if(input_3bit == 3'b100) output_4bit &lt;= 4'b1000;</pre>
12	<pre>if(input_3bit == 3'b110) output_4bit &lt;= 4'b1100;</pre>
13	<pre>if(input_3bit == 3'b011) output_4bit &lt;= 4'b0110;</pre>
14	<pre>if(input_3bit == 3'b101) output_4bit &lt;= 4'b1010;</pre>
15	<pre>if(input_3bit == 3'b111) output_4bit &lt;= 4'b0001;</pre>
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	<pre>module mux_always_case(input_3bit,output_4bit); // 3-to-4 multiplexer</pre>
2	<pre>input [2:0] input_3bit; // 3-bit input</pre>
3	<pre>output reg [3:0] output_4bit; // 4-bit output</pre>
4	
5	always // always block with "case" construct
6	E begin
7	<pre>case(input_3bit)</pre>
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	<pre>module multiple_modules(input_clock, output</pre>	t_FourBi	its); // top-level module
	2	input input_clock; //	input w	Jire
	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 w	vires of multiplexer
	7			
	8	<pre>assign mux_input = counter_output; //</pre>	connect	the counter output and multiplexer input wires
	9			
	10	counter counter result(input clock, co	unter ou	<pre>stput); // call the "counter" module</pre>
	11		_	<pre>// with instance "counter result"</pre>
	12			_
	13	mux always case mux output(mux input,	output F	<pre>fourBits); // call the "mux always case" module</pre>
	14		-	// with instance " mux output"
	15			_
	16	endmodule	mux alwai	
	17			ra_0000. r
		·	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
counter. v	,		4	
			5	always // always block with "case" construc
1	= n	<pre>module counter(input clk, output_3bit);</pre>	6	begin begin
2		input input_clk;	7	case(input 3bit)
3		output reg [2:0] output 3bit;	8	3'b000: output 4bit <= 4'b0001;
4		_	9	3'b001: output 4bit <= 4'b0010;
5		always@(posedge input_clk)	10	3'b010: output 4bit <= 4'b0100;
6		begin	11	3'b100: output 4bit <= 4'b1000;
7		output_3bit <= output_3bit + 3'b001;	12	3'b110: output 4bit <= 4'b1100;
8		end	13	3'b011: output 4bit <= 4'b0110;
9			14	3'b101: output 4bit <= 4'b1010;
10	e	ndmodule	15	3'b111: output 4bit <= 4'b0001;
11			16	endcase
			17	end
			18	
			19	endmodule
			20	

# **Multiple Modules**

multiple	e_modules.v			
1	1 module multiple_modules(input_clock, output	t_FourBi	ts	); // top-level module
2	input input_clock; //	input w	ir	e
3	output [3:0] output FourBits;	output	wi	Tenanting the 2 lower lovel medules
4	4 Wires	s for c	;0	nnecting the z lower level modules
5	5 wire [2:0] counter_output; //	output	wi	res of counter
6	5 wire [2:0] mux_input;	input w	ir	es of multiplexer
8	<pre>3 assign mux_input = counter_output; //</pre>	connect	t	he counter output and multiplexer input wires
9				
10	) counter counter_result(input_clock, co	unter_ou	ıtp	ut); // call the "counter" module
11	modulo namo instanco namo			<pre>// with instance "counter_result"</pre>
12				
13	3 mux_always_case mux_output(mux_input,	output_F	ou	rBits); // call the "mux_always_case" module
14	1			<pre>// with instance " mux_output"</pre>
15	5			
16	5 endmodule	mux_always_case.v*		
17	7			
		1		<pre>module mux_always_case(input_3bit,output_4bit); //</pre>
		2		<pre>input [2:0] input_3bit; // 3-bit input</pre>
		3		output reg [3:0] output_4bit; // 4-bit outpu
inter. v		4		
4		5		always // always block with "case" constru
1	<pre>module counter(input_cik, output_spit);</pre>	6		begin
2	input input_cik;	7		case(input_3bit)
Д	output reg [2:0] output_spit;	8		3'b000: output_4bit <= 4'b0001;
-	elwewe@(nocedge_innut_clk)	9		3'b001: output_4bit <= 4'b0010;
5	alwayse(posedge input_cik)	10		3'b010: output_4bit <= 4'b0100;
7	$= \frac{\text{pegin}}{\text{output 3bit } <= \text{output 3bit } + 3!b001;$	11		3'b100: output_4b1t <= 4'b1000;
ģ	end	12		3'b110: output_4b1t <= 4'b1100;
9	enu	13		3'DOII: OUTPUT_4DIT <= 4'DOIIO;
10	endmodule	14		3'DIOI: OUTPUT_4DIT <= 4'DIOIO;
11		15		s.piii: output_tait <= 4.p0001;
11		10		enucase
		10		enu
		10		endmodule
		20		
		20		1

# **Multiple Modules**

1	module multiple modules(input clock, outpu	t FourBi	ts); // top-level module
2	input imput clock, //	input w	vire
3	output [3:0] output FourBits; //	output	wires
4	Warni	na: D	O NOT make these registers !!!
5	wire [2:0] counter_output; //	oucput	wires of counter
6	wire [2:0] mux_input; //	input w	vires of multiplexer
7			
8	<pre>assign mux_input = counter_output; //</pre>	connect	the counter output and multiplexer input wires
9			
10	counter counter_result(input_clock, co	unter_ou	tput); // call the "counter" module
11	module name instance name		<pre>// with instance "counter_result"</pre>
12			
10	mux_always_case mux_output(mux_input,	output_f	ourbits); // call the "mux_always_case" module
15			// with instance ~ mux_output~
16	endmodule		
17		mux_alway	/s_case.v <sup>-</sup>
		1	module mux always case(input 3bit,output 4bit); //
		2	input [2:0] input 3bit; // 3-bit input
		3	output reg [3:0] output 4bit; // 4-bit output
iter. v		4	_
		5	always // always block with "case" constru
1	<pre>module counter(input_clk, output_3bit);</pre>	6	E begin
2	input input_clk;	7	<pre>case(input_3bit)</pre>
3	output reg [2:0] output_3bit;	8	3'b000: output_4bit <= 4'b0001;
4		9	3'b001: output_4bit <= 4'b0010;
E	alwayse(poseuge input cik)	1 10	3' builds output 4bit <= 4' builds
5	begin		21b100, cutput 4bit (= 41b1000;
5 6 <b>E</b>	begin output 3bit <= output 3bit + 3'b001.	11	3'b100: output_4bit <= 4'b1000; 3'b110: output_4bit <= 4'b1000;
5 6 <b>1</b> 7	<pre>begin output_3bit &lt;= output_3bit + 3'b001; end</pre>	11 12 13	3'b100: output_4bit <= 4'b1000; 3'b110: output_4bit <= 4'b1100; 3'b110: output_4bit <= 4'b1100;
5 6 7 8 9	begin output_3bit <= output_3bit + 3'b001; end	11 12 13	3'b100: output_4bit <= 4'b1000; 3'b110: output_4bit <= 4'b1100; 3'b011: output_4bit <= 4'b0110; 3'b011: output_4bit <= 4'b0110; 3'b101: output_4bit <= 4'b1010;
5 6 7 8 9	<pre>begin output_3bit &lt;= output_3bit + 3'b001; end endmodule</pre>	11 12 13 14	3'b100: output_4bit <= 4'b1000; 3'b110: output_4bit <= 4'b1100; 3'b011: output_4bit <= 4'b0110; 3'b101: output_4bit <= 4'b0110; 3'b101: output_4bit <= 4'b1010; 3'b111: output_4bit <= 4'b0001:
5 6 7 8 9 10 11	<pre>begin output_3bit &lt;= output_3bit + 3'b001; end endmodule</pre>	11 12 13 14 15 16	3'b100: output_4bit <= 4'b1000; 3'b110: output_4bit <= 4'b1100; 3'b011: output_4bit <= 4'b0110; 3'b101: output_4bit <= 4'b1010; 3'b111: output_4bit <= 4'b0001; endcase
5 6 7 8 9 10 11	<pre>begin output_3bit &lt;= output_3bit + 3'b001; end endmodule</pre>	11 12 13 14 15 16 17	3'b100: output_4bit <= 4'b1000; 3'b110: output_4bit <= 4'b1100; 3'b011: output_4bit <= 4'b0110; 3'b101: output_4bit <= 4'b1010; 3'b101: output_4bit <= 4'b0001; endcase end
5 6 7 8 9 10 11	<pre>begin output_3bit &lt;= output_3bit + 3'b001; end endmodule</pre>	11 12 13 14 15 16 17 18	<pre>3'b100: output_4bit &lt;= 4'b1000; 3'b110: output_4bit &lt;= 4'b1100; 3'b011: output_4bit &lt;= 4'b0110; 3'b101: output_4bit &lt;= 4'b0100; 3'b111: output_4bit &lt;= 4'b0001; endcase end</pre>
5 6 7 8 9 10 11	<pre>begin output_3bit &lt;= output_3bit + 3'b001; end endmodule</pre>	11 12 13 14 15 16 17 18 19	<pre>3'b100: output_4bit &lt;= 4'b1000; 3'b110: output_4bit &lt;= 4'b1100; 3'b011: output_4bit &lt;= 4'b0110; 3'b101: output_4bit &lt;= 4'b0110; 3'b101: output_4bit &lt;= 4'b0001; endcase end endmodule</pre>