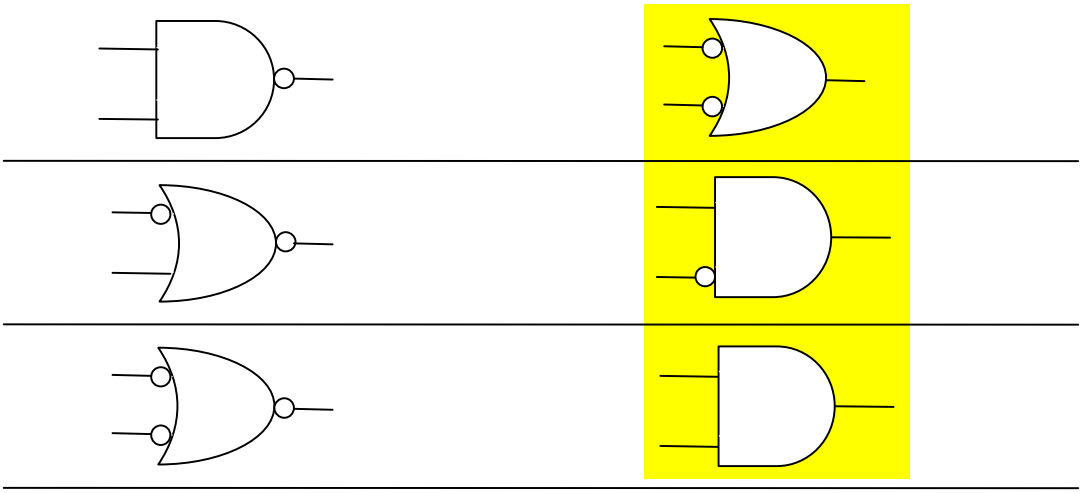
October 11th, 2011 **2612 – Digital Design – Exam 1**

Name_____

1. (6 points) Use DeMorgan's law to draw an equivalent gate representation of each of the following:



2. (6 points) An ASCII code table is provided below. Convert the hexadecimal numbers below, represented in ASCII code to their equivalent alpha- numeric representation.

USASCII code chart

Bits $b_7 b_6 b_5$ →	000	001	010	011	100	101	110	111
b_4 b_3 b_2 b_1 ↓ Row ↓ / Column →	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

47	69	76	65	20	6D	65	20	61	6E	20	41	2E
G	i	v	e	[sp]	m	e	[sp]	a	n	[sp]	A	.

3. An 8 button keypad has a wireless transmitter that generates a 3 bit serial code for each key pressed. An additional parity bit is added before transmission so that the full 4 bit data block has odd parity. [Parity is defined as the number of 1's in a set of bits.]
    a. (4 points) Fill in the parity bit for each set of bits in the table below so that every 4 bit transmission will have odd parity:

a	b	c	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

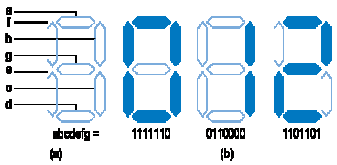
    b. (4 points) Write a Verilog statement using Sum of Products that generates the value of P:

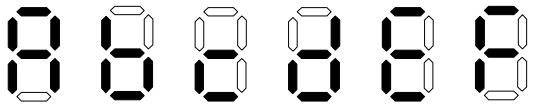
```
P = (~a & ~b & ~c) | (~a & b & c) | (a & ~b & c) | (a & b & ~c);
```

    c. (4 points) Write a Verilog statement using XOR operators that generates the value of P:

```
P = ~(a ^ (b ^ c));
```

4. (12 points) You are asked to expand the design of a 7 segment decoder in order to create a hexadecimal display, i.e., adding the symbols A,B,C,D,E,F to an existing Verilog design. The additional design information is given below (do not confuse the segment driver names (a-g) with the hexadecimal digits (A-F)). Write the Verilog code you would add to the design and indicate with an arrow where you would insert your code in the existing design.





A B C D E F

```
module hex_decoder( input [3:0] hex_in,
        output reg [6:0] seg_out );
        // seg_out maps bit 0 -> a … bit 6 -> g
…
        always @ (hex_in) begin
                case (hex_in)
                        4'b0000: seg_out = 7'b0111111;
                        4'b0001: seg_out = 7'b0000110;
                        4'b0010: seg_out = 7'b1011011;
                        4'b0011: seg_out = 7'b1001111;
                        4'b0100: seg_out = 7'b1100110;
                        4'b0101: seg_out = 7'b1101101;
                        4'b0110: seg_out = 7'b1111101;
                        4'b0111: seg_out = 7'b0000111;
                        4'b1000: seg_out = 7'b1111111;
                        4'b1001: seg_out = 7'b1101111;

                        default: seg_out = 7'b0000000;
                endcase
        end
…
```

```
4'b1010: seg_out = 7'b1110111;
4'b1011: seg_out = 7'b1111100;
4'b1100: seg_out = 7'b1011000;
4'b1101: seg_out = 7'b1011110;
4'b1110: seg_out = 7'b1111001;
4'b1111: seg_out = 7'b1110001;
```

5. (15 points) Design a digital circuit (a Verilog module) that provides a thermometer like readout using a string of 7 adjacent LED's, from a 3 bit binary input (unsigned integer). For example, if data_in is equal to 0, all LED's are off; if data_in is equal to 7, all 7 LED's are on; if data_in is equal to 3, 3 adjacent LED's are on; … so that the LED string behaves like viewing a thermometer. Assume that a logic one will turn on an LED. A control signal, ena is included. If ena is 0, the LED's are off; if ena is 1, the LED's are on. Start with the top module definition below. (I recommend using "case" and/or "if … else" statements.)

```
module therm_code (input [2:0] data_in, input ena, output reg [6:0] led_array);


        always @ (ena, data_in) begin
                case (data_in)
                        3'b000: led_array = 7'b0000000;
                        3'b001: led_array = 7'b0000001;
                        3'b010: led_array = 7'b0000011;
                        3'b011: led_array = 7'b0000111;
                        3'b100: led_array = 7'b0001111;
                        3'b101: led_array = 7'b0011111;
                        3'b110: led_array = 7'b0111111;
                        3'b111: led_array = 7'b1111111;
                        default: led_array = 7'b0000000;
                endcase
                if (ena === 0) led_array = 7'b0000000;
        end




endmodule
```

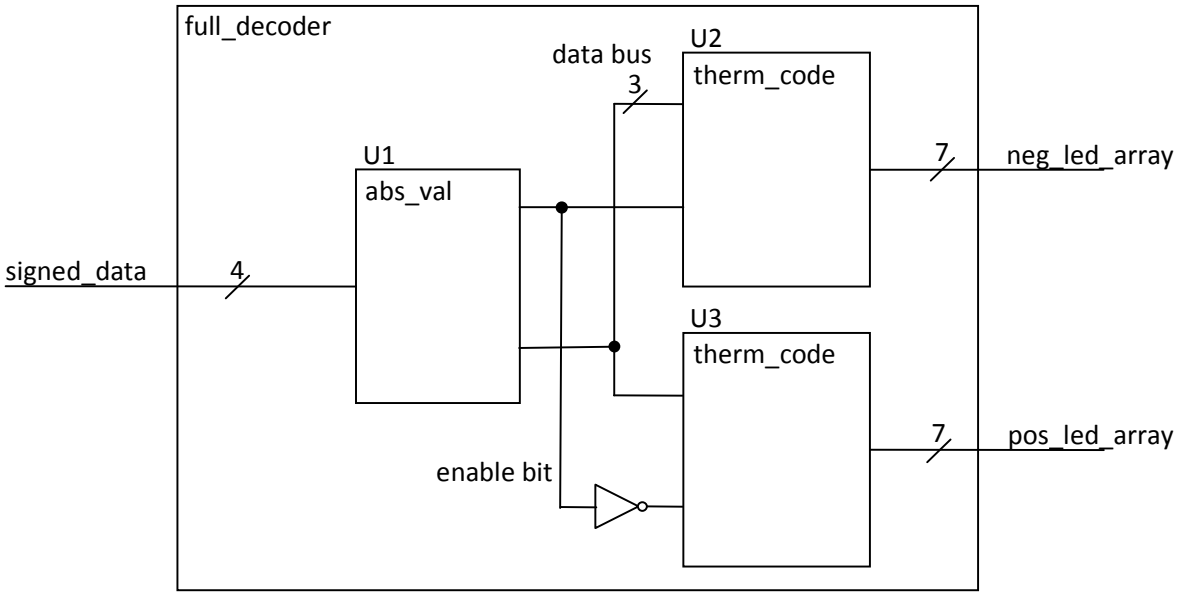
6. (12 points) Design a digital circuit (a Verilog module) that converts a 4 bit signed twos complement number into a 3 bit unsigned binary number (absolute value) with a sign bit (0 positive, 1 negative). (If you can't do this directly to Verilog, you should start with a truth table.)

```
module abs_val (input [3:0] d_2s_cmp, output reg [2:0] data_av, output reg sign);

        always @ (d_2s_cmp) begin
                case (d_2s_cmp)
                        4'b1111: data_av = 1; // minus 1
                        4'b1110: data_av = 2; // minus 2
                        4'b1101: data_av = 3; // …
                        4'b1100: data_av = 4;
                        4'b1011: data_av = 5;
                        4'b1010: data_av = 6;
                        4'b1001: data_av = 7; // minus 7
                        default: data_av = d_2s_cmp[2:0];    // take care of positive
                endcase
                sign = d_2s_cmp[3];    // move the sign bit to sign
        end

endmodule
```

7. (12 points) Using the modules defined in problems 5 & 6 (therm_code, abs_val), design the next level hierarchy illustrated in the block diagram below in a Verilog module. This module will now drive two sets of LED's, one set illuminated for positive numbers and the other for negative numbers.




```
module full_decoder (input [3:0] signed_data, output [6:0] pos_led_array,
        output [6:0] neg_led_array);

        // define wire and reg variables
        wire [2:0] internal_d;
        wire internal_ena;
        wire internal_ena_bar;
        // use an assign statement to take care of inverted enable
        assign internal_ena_bar = ~internal_ena;


        // instantiate the three modules – I'll do the skeleton of the first one
        abs_val u1 (  signed_data, internal_d, internal_ena   );

        therm_code u2 (internal_d, internal_ena, neg_led_array);

        therm_code u3 (internal_d, internal_ena_bar, pos_led_array);


endmodule
```