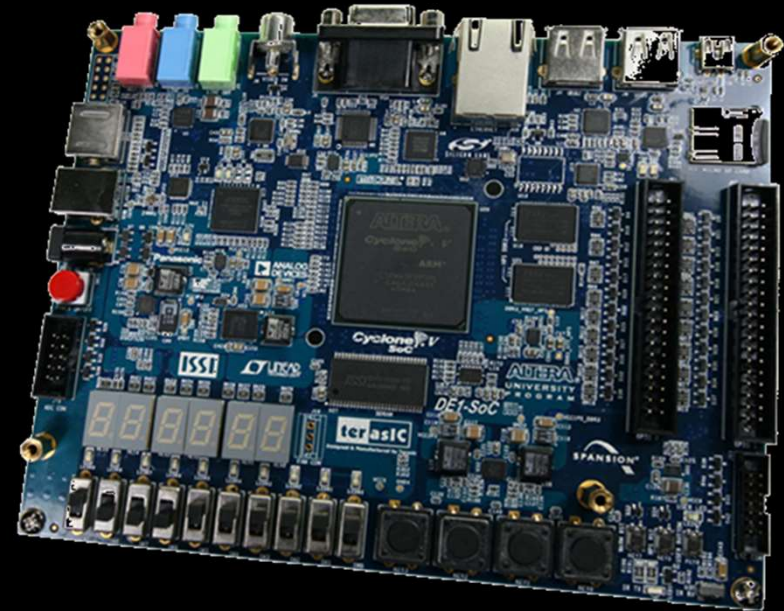


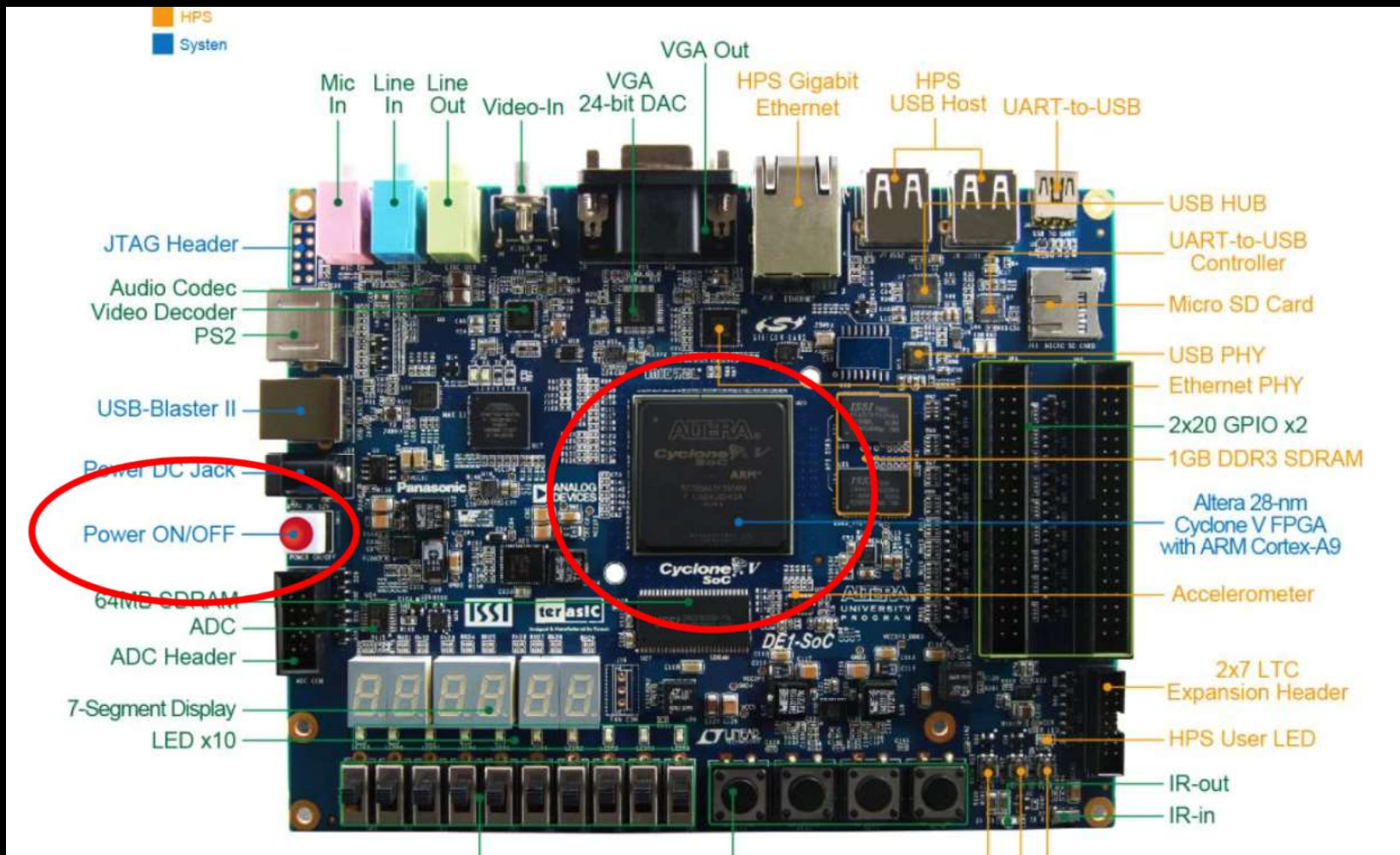
Lab 2 Tutorial

Lab 2

- Using the DE1-SoC
- Creating a project using Quartus Prime.
- Intro to Verilog.
- Lab2 Topics
 - Multiplexers, Hierarchy, Decoders, 7-segment displays



Meet the DE1-SoC board!

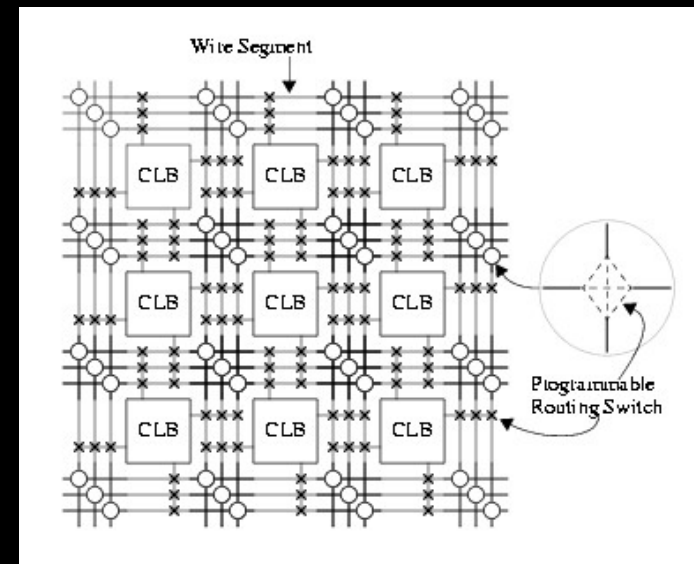


Meet the DE1-SoC board!

- It's a System On a Chip (SoC) w/
 - Altera's Cyclone® V 5CSEMA5F31 FPGA, and
 - a Dual-core ARM Cortex-A9 hard processor (HPS)
- 64 MB SDRAM on FPGA device
- Six 7-segment displays
- 10 toggle switches
- 10 LEDs
- 9 green LEDs
- Four pushbutton switches

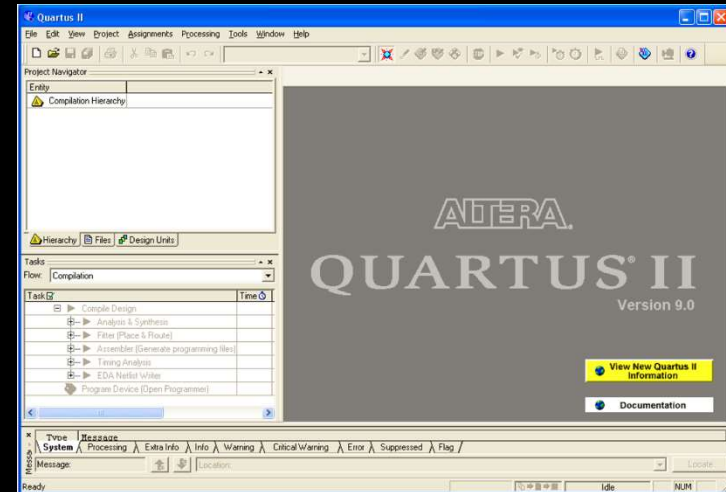
What does that mean?

- Key term: **FPGA**.
 - Stands for Field Programmable Gate Array.
 - A regular network of logic that can be programmed and reprogrammed to implement any circuit.
 - Circuits aren't build by hand from now on; they're programmed using languages like **Verilog** or **VHDL**.



Quartus Prime

- Tool provided by Altera that **compiles Verilog programs, and uploads the result to the FPGA.**
- When you do your lab, login at the lab computers with your UTORid.
 - Make sure you've activated your ECF account.
- Quartus Prime should be available from the start menu. You should use Quartus 16.



Quartus Prime (cont'd)

- How to create projects for the lab:
 1. Create a new Quartus Prime project for your circuit. Select **Cyclone V 5CSEMA5F31C6** as the target chip, which is the FPGA chip on the Altera DE1-SoC board.
 2. Create a Verilog module for the current part of the lab and include it in your project.
 3. Include in your project the required pin assignments for the DE1-SoC board, as discussed above. Compile the project.
 - The “play” icon in the menu bar.
 4. Download the compiled circuit into the FPGA chip.
 - Click the icon with the ribbon cable on the menu bar.



Intro to Verilog

- Verilog is a hardware description language (HDL) that is used to specify a circuit design.
- Instead of specifying actual gate connections, Verilog takes in more high-level functionality, which is translated into digital logic before implementing it on the hardware.

About pin assignments

- In Quartus, select *Assignments* → *Import Assignments*
 - ▣ The DE1 SoC.qsf file (posted on Portal associates signal names to pins on the chip.
- If you name for example the inputs of your design with *SW[0]*, *SW[1]*, and *SW[2]* and have declared your output as 7-bit port named *HEXo*, everything will work fine!

Learning about Verilog

```
module dn;

  reg a,b,c;

  initial begin : a_b
    $monitor ("%0t %m a: %b b: %b", $time, a, b);
    #100;
    {a,b} = 2'b01;
    c = 1;
    b <= 1'b0;
    #100;
    {a,b} = 2'b10;
    c = 1'b2;
    #100;
    {a,b} = 2'b11;
    c = 0;
    #1000 $finish;
  end // a_b

  initial begin : extra
    $monitor ("%0t %m c: %b", $time, c);
    $monitor ("%0t CVC: c: %b", $time, c);
  end // extra

  always @(*) begin : alw_strobe
    $strobe ("%0t %m STROBE+ALWAYS: a: %b b: %b c: ", $time, a, b, c);
  end //

  always @(*) begin : al
    $display ("%0t %m ALWAYS: a: %b b: %b c: ", $time, a, b, c);
  end // al

endmodule
```

Creating Verilog: XOR gates

- XOR gates have high output when the inputs are different, and low output when the inputs are the same.



| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- Could we make this from AND, OR and NOT gates?

Creating Verilog: XOR gates

```
module xor(Y,A,B);  
    output Y;  
    input A, B;  
    wire not_A, not_B;  
    wire A_and_notB, B_and_notA;  
    not n1(not_A, a);  
    not n2(not_B, b);  
    and a1(A_and_notB, A, not_B);  
    and a2(B_and_notA, B, not_A);  
    or o1(Y, A_and_notB, B_and_notA);  
endmodule
```

A sample Verilog program

```
1 // Simple module that connects the
2 // SW switches to the LEDR lights
3 module part0 (SW, LEDR); LEDR is red LED
4     input [9:0] SW;           // toggle switches
5     output [9:0] LEDR;        // red LEDs
6
7     assign LEDR[9:0] = SW[9:0];
8 endmodule
```

- This is a basic Verilog module that:
 - specifies the ports that will be used (line 1),
 - designates how many will be used for input (line 2) and for output (line 3)
 - Specifies the logic that will take place within the module (line 4)

Things to note

- The name of the module should be the same as the high-level line that says `module XXX (...)`.
- The ports can be `input`, `output` and `inout` (focus on the first two).
- The `assign` keyword.
- The square brackets indicate a vector of values. Individual bits within that vector can be read or assigned using usual array notation
(e.g. `LED[0] = SW[0]`
or `LED[2:0] = SW[2:0]`).
- Don't forget the semicolons after each statement!

Wires in Verilog

- The first declaration is for a single wire, while the second indicates a vector.

```
// The following lines create  
// intermediate wires.  
    wire Sel;  
    wire [7:0] X, Y, M;
```

To Be Continued Next Week

- Next week, we'll write a module in Verilog and do an in-class demo.
- The next slide (operators) is provided for future reference. Each lab handout introduces you to the operators you'll need.

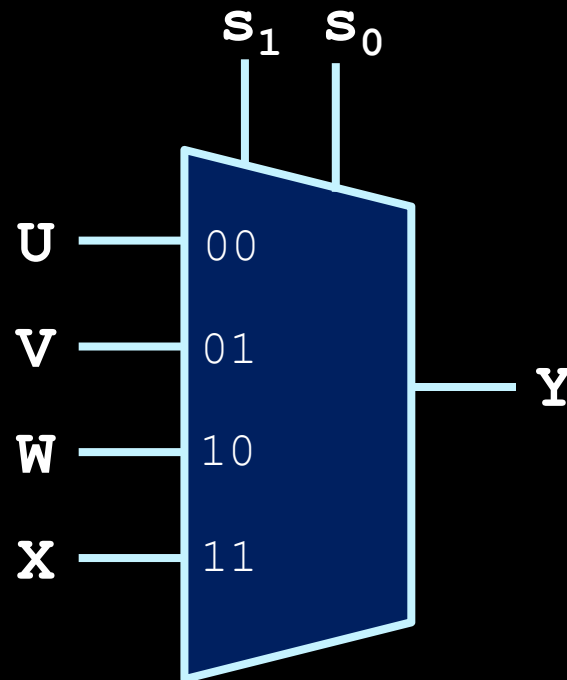
Operators

- Signals can be combined together in all the usual ways, and some new ones that are specific to digital values.

| Operator Type | Operator Symbol | Operation Performed |
|---------------|-----------------|-----------------------|
| Arithmetic | * | Multiply |
| | / | Division |
| | + | Add |
| | - | Subtract |
| | % | Modulus |
| | + | Unary plus |
| | - | Unary minus |
| Logical | ! | Logical negation |
| | && | Logical and |
| | | Logical or |
| Relational | > | Greater than |
| | < | Less than |
| | >= | Greater than or equal |
| | <= | Less than or equal |
| Equality | == | Equality |
| | != | inequality |
| Reduction | ~ | Bitwise negation |
| | ~& | nand |
| | | or |
| | ~ | nor |
| | ^ | xor |
| | ^~ | xnor |
| Shift | >> | Right shift |
| | << | Left shift |
| Concatenation | {} | Concatenation |
| Conditional | ? | conditional |

Multiplexers

- All you need to know:

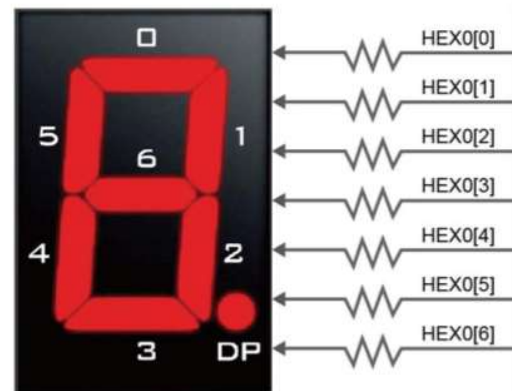


| s_1 | s_0 | Y |
|-------|-------|---|
| 0 | 0 | U |
| 0 | 1 | V |
| 1 | 0 | W |
| 1 | 1 | X |

Using 7-segment displays

The DE1-SoC board has six 7-segment displays. These displays are arranged into three pairs, meant for displaying numbers of various sizes. As indicated in the schematic in **Figure 3-17**, the seven segments (common anode) are connected to pins on Cyclone V SoC FPGA. Applying a low logic level to a segment will light it up and applying a high logic level turns it off.

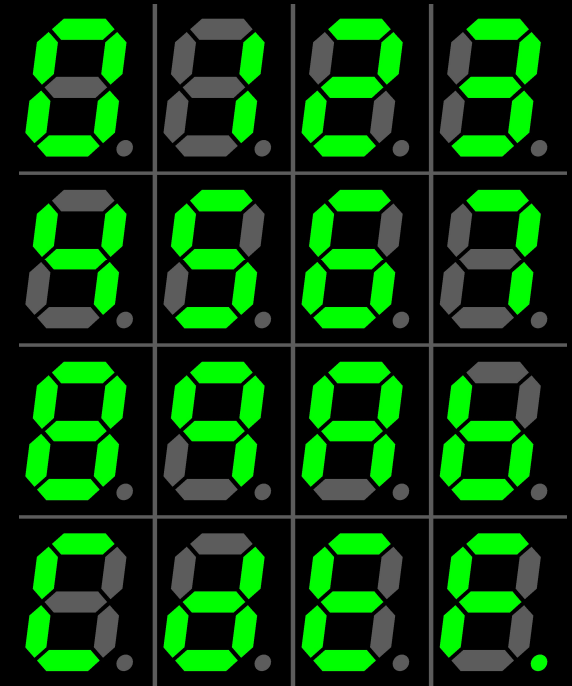
Each segment in a display is identified by an index from 0 to 6, with the positions given in **Figure 3-17**. **Table 3-9** shows the assignments of FPGA pins to the 7-segment displays.



Activating HEX displays

- Need to set segment 0 (top segment) **low** in the following input cases:

| | | |
|------|----|-----|
| 0000 | -- | "0" |
| 0010 | -- | "2" |
| 0011 | -- | "3" |
| 0101 | -- | "5" |
| 0110 | -- | "6" |
| 0111 | -- | "7" |
| 1000 | -- | "8" |
| 1001 | -- | "9" |
| 1010 | -- | "A" |
| 1100 | -- | "C" |
| 1110 | -- | "E" |
| 1111 | -- | "F" |



- How do we express this?

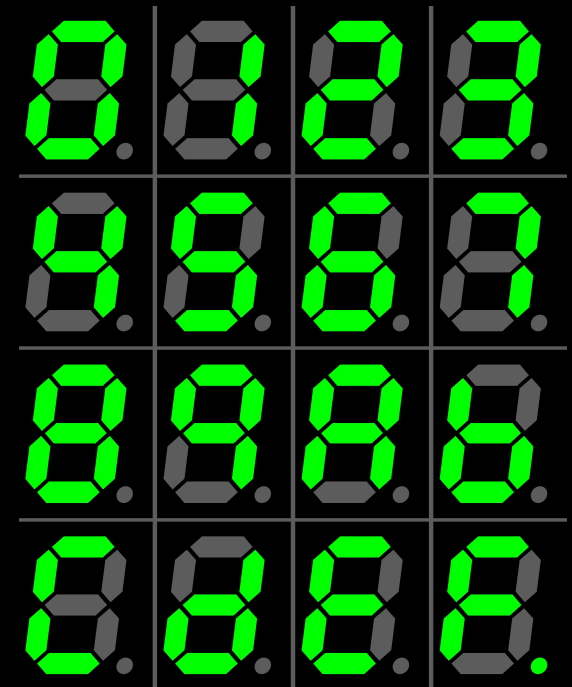
Activating HEX displays

- Could also set segment 0 (top segment) **high** in the other input cases:

- 0001 -- "1"
- 0100 -- "4"
- 1011 -- "B"
- 1101 -- "D"

- Can be expressed as a four-part Boolean expression:

```
HEX[0] = ~SW[3] & ~SW[2] & ~SW[1] & SW[0] |  
         ~SW[3] & SW[2] & ~SW[1] & ~SW[0] |  
         SW[3] & ~SW[2] & SW[1] & SW[0] |  
         SW[3] & SW[2] & ~SW[1] & SW[0];
```



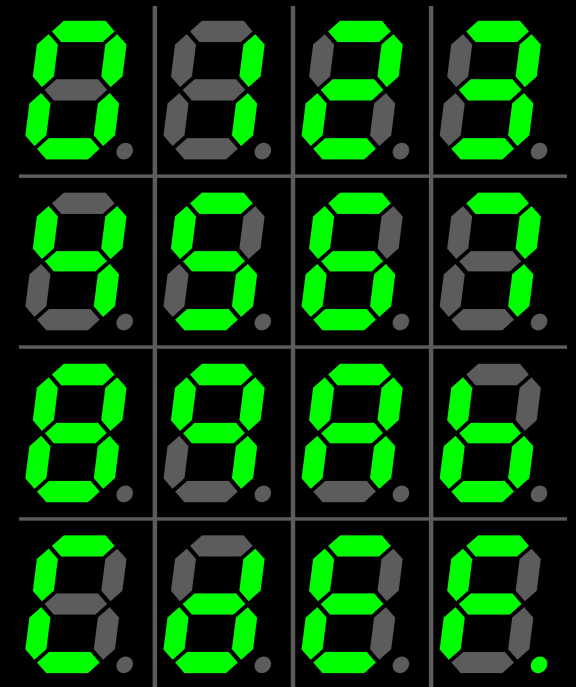
Activating HEX displays

```
HEX[0] = ~SW[3] & ~SW[2] & ~SW[1] & SW[0] |  
         ~SW[3] & SW[2] & ~SW[1] & ~SW[0] |  
         SW[3] & ~SW[2] & SW[1] & SW[0] |  
         SW[3] & SW[2] & ~SW[1] & SW[0];
```

- Can this be reduced any further?
 - ...sadly, no ☹
- How do we know?
 - Karnaugh maps!

Activating HEX displays

- Can you write the expressions for HEX[1] to HEX[6]?
- Can you reduce these expressions to the simplest gate form?



Some Verilog References

- Check out the Verilog Primer and other resources on Blackboard for more information on the Verilog language.
- ECE also provides reference documentation for the lab rooms at:
 - <http://www-ug.eecg.utoronto.ca/desl/>