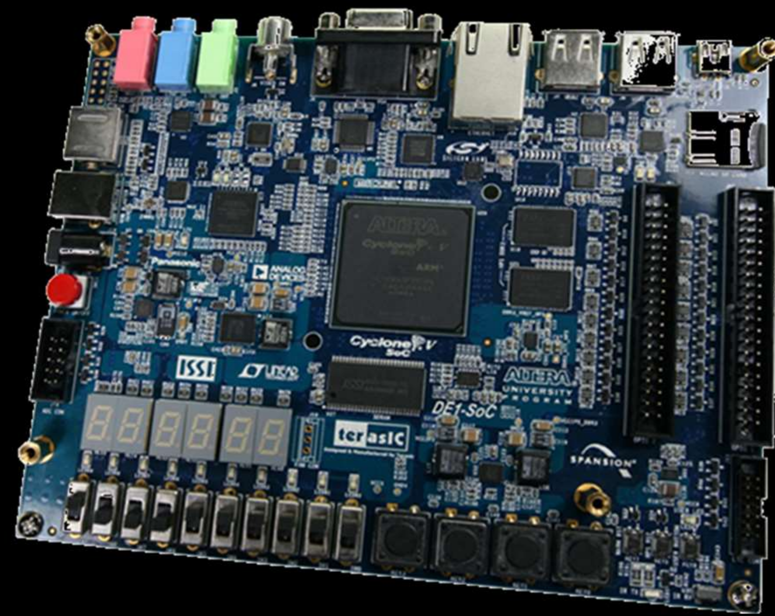
Lab 2 Preparation

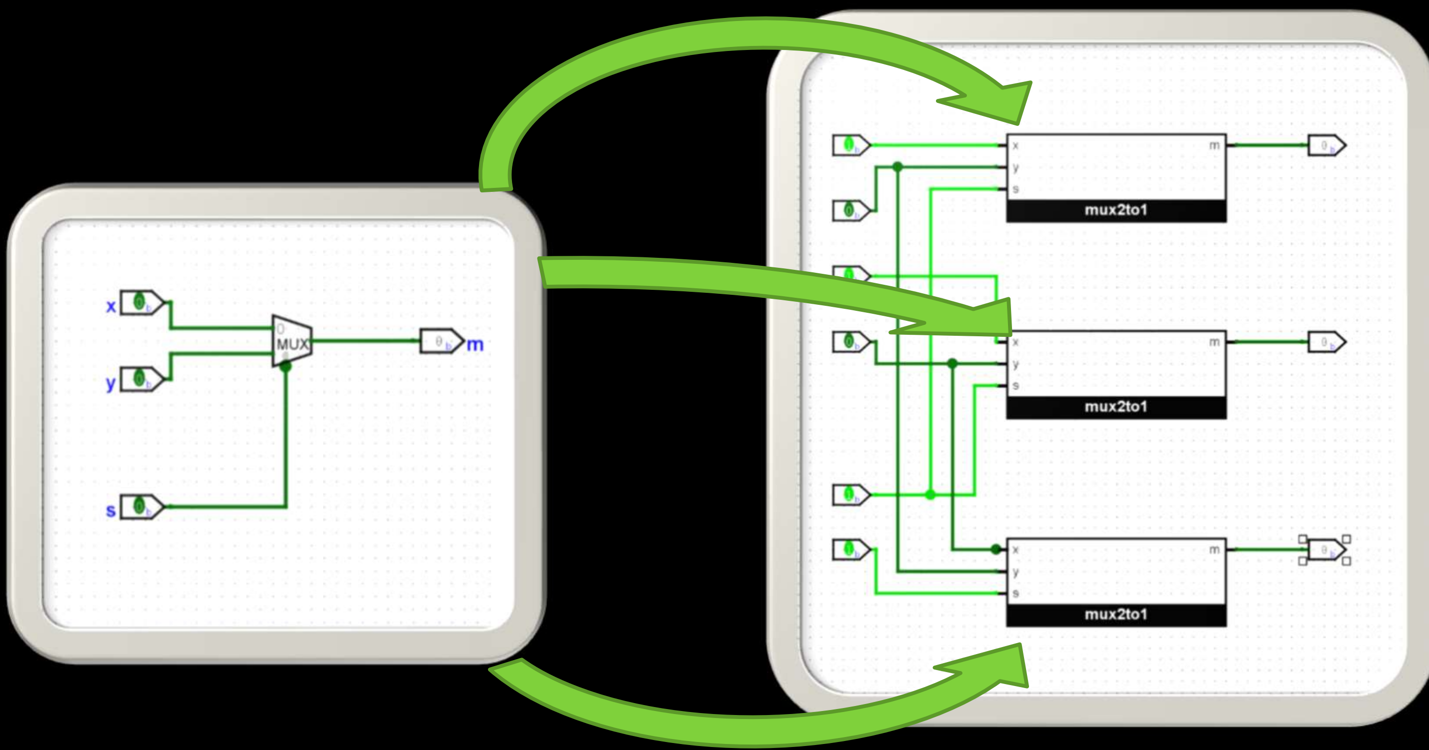
Lab 2

- Lab 2 topics:
  - Multiplexers (cont'd)
  - Design hierarchy
  - Decoders
    - 7-segment displays
- The DE1-SoC
  - Connecting Logisim to DE1-Soc
- Intro to useful components in Logisim.



Tasks for Lab 2

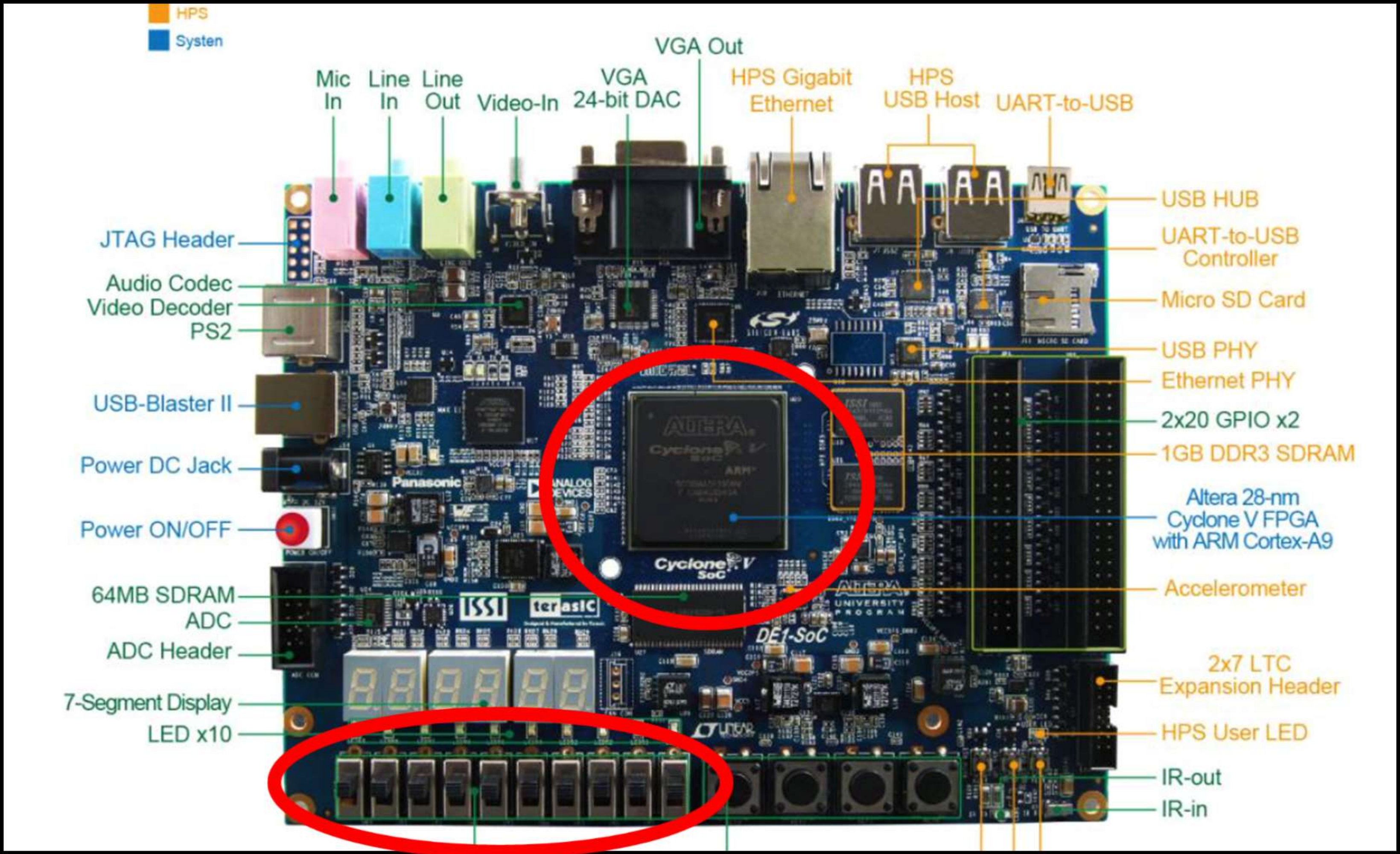
- Part I: Creating modules
  - Once created, a module can be used as a component.



Tasks for Lab 2

- Part I: Creating modules
  - Create a simple module that makes a wrapper for the mux circuit we provide.
  - Set inputs to buttons (labeled `SW0`, `SW1`, `SW2`) and output to LED (labeled `LEDR`)
    - Labels correspond to DE1-SOC inputs and outputs

Meet the DE1-SoC board!

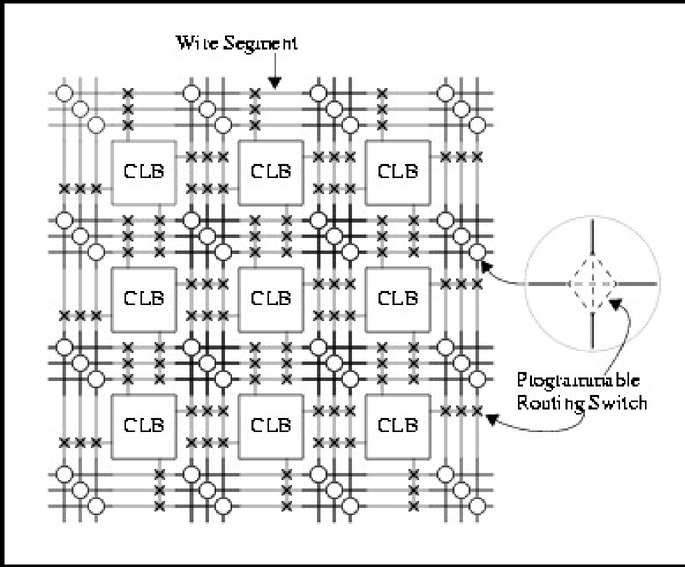


Meet the DE1-SoC board!

- In a non-COVID semester, we would be using Logisim to upload your designs onto the DE1-SOC.
  - Maybe in 2021 ☹
- What is the DE1-SOC?
  - It's a System On a Chip (SoC) with:
    - 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 generally built by hand; they're programmed using languages like Verilog, VHDL or Logisim.



Connecting Logisim to DE1-SoC

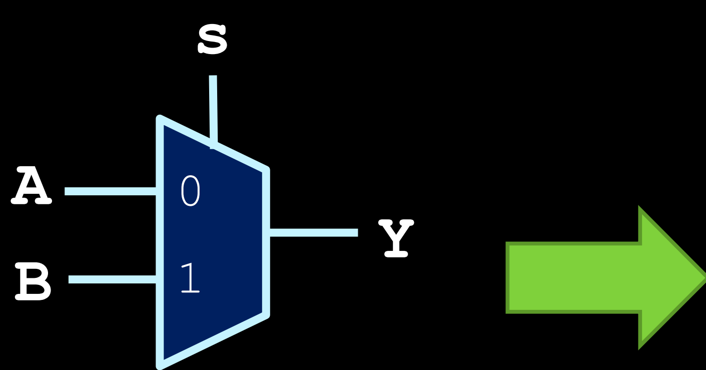
- The circuits we build in Logisim can be downloaded to the DE1-SoC board.
- You can map your inputs/outputs in Logisim to the switches or LEDs on the DE1-SoC board and then test your circuit on the board.
- Even though we currently do not access to the boards in the lab, it is still good to know the process and learn about the board.

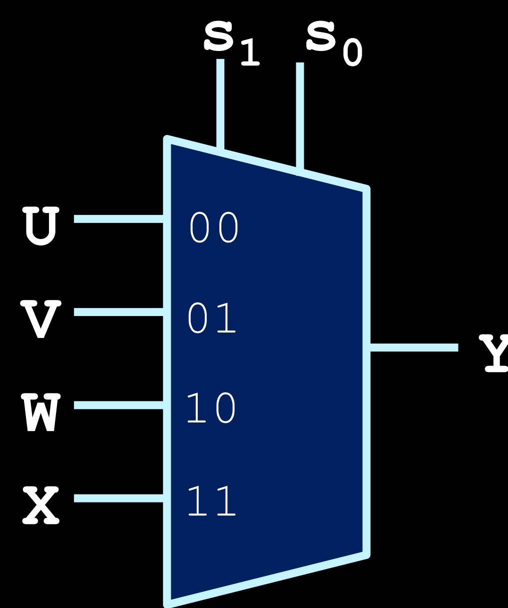
Connecting Logisim to DE1-SoC

- The details about the process can be found in lab2 handout.
- If you want to upload your design:
  - Make sure that you follow the steps carefully and understand how the circuit would have been if it can be downloaded on the board.
  - Note: you will need a tool called Quartus installed to compile in Logisim. Therefore we encourage you to test out the compilation of your design on the teach.cs machines. Quartus has been installed on them.

Tasks for Lab 2

- Part II: Designing with modules.
  - Make a 4-to-1 mux
    out of 2-to-1 muxes.



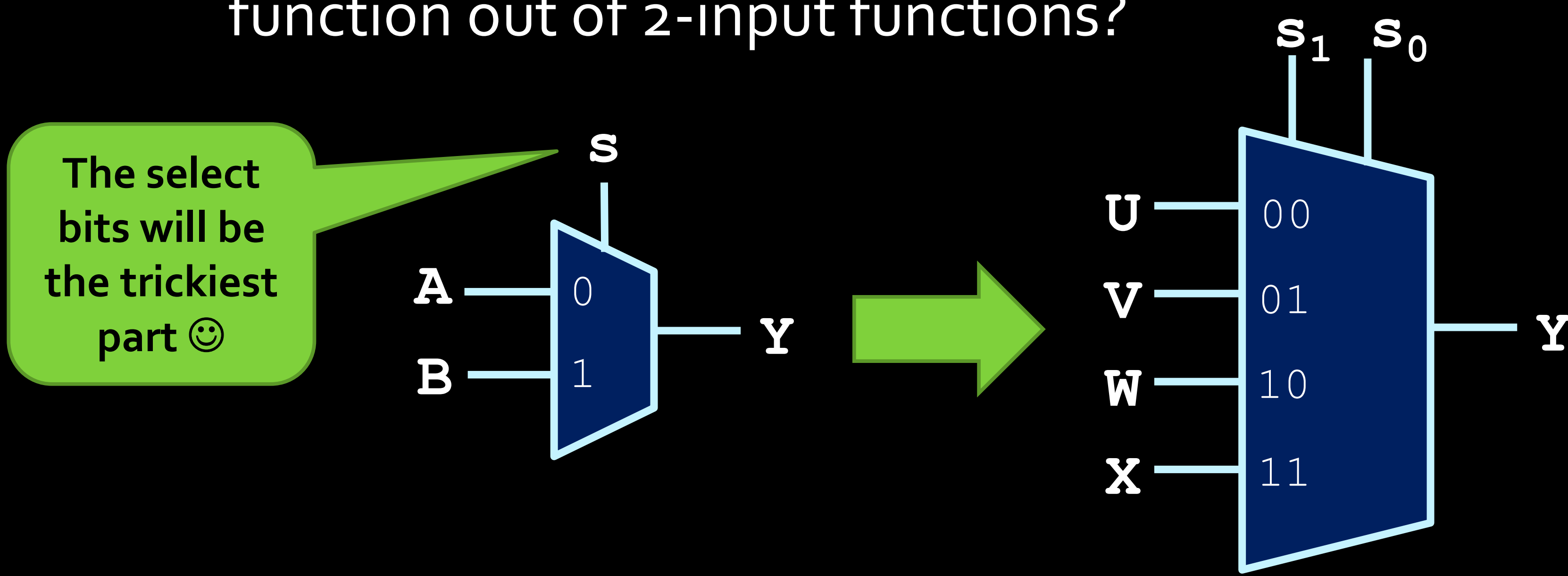


s_0	Y
0	A
1	B

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

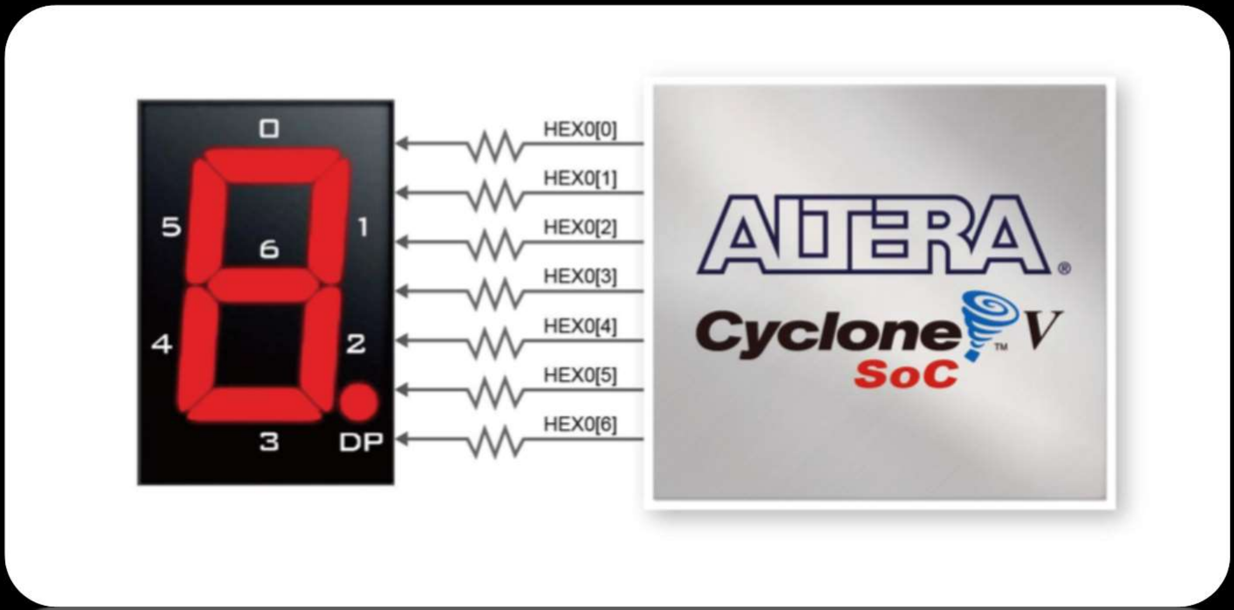
Tasks for Lab 2

- Part II: Designing with modules.
  - If each 2-to-1 mux can handle 2 inputs, how to build something that handles 4?
  - How would you make a 4-input function out of 2-input functions?



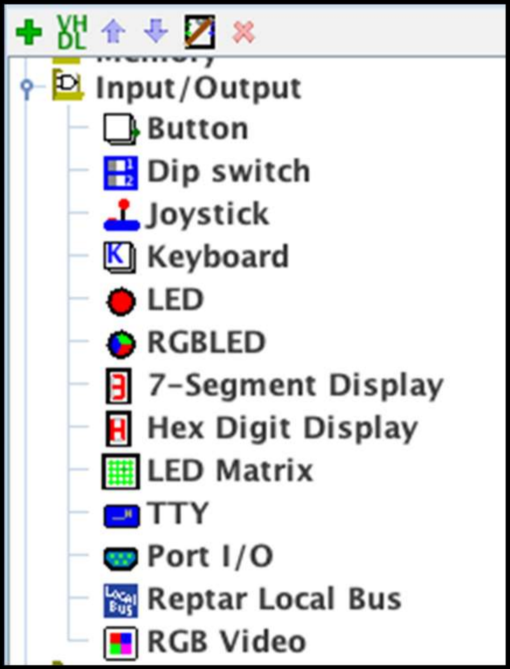
Tasks for Lab 2

- Part III: The 7-segment decoder.
  - This is one of the components in the Logisim toolkit.
  - Also one of the components on the DE1-SOC board!



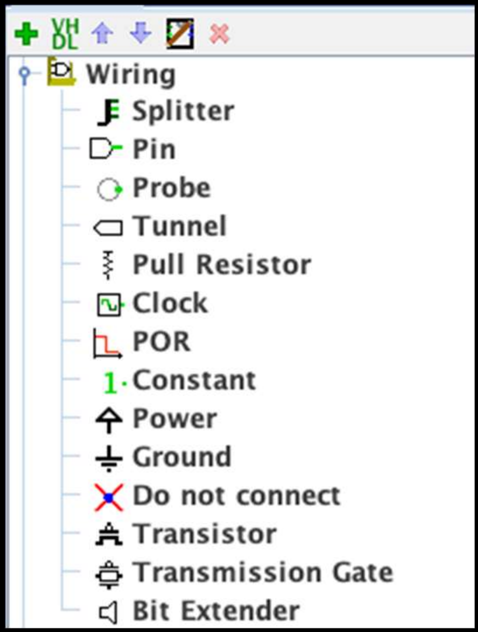
Exploring Logisim Components

- This components and others are listed under Input/Output, for future reference:
  - Button: Can be mapped to the switches and buttons on the DE1 board. Only outputs a 1 when held down with Poke.
  - 7-Segment Display: Can be mapped to the 7-segment display on the DE1 board.
  - LED: Can be mapped to the outputs on the DE1 board.
- Note: always start with the default input/output type from the tool bar and only switch to the above if necessary



Useful Components in Logisim

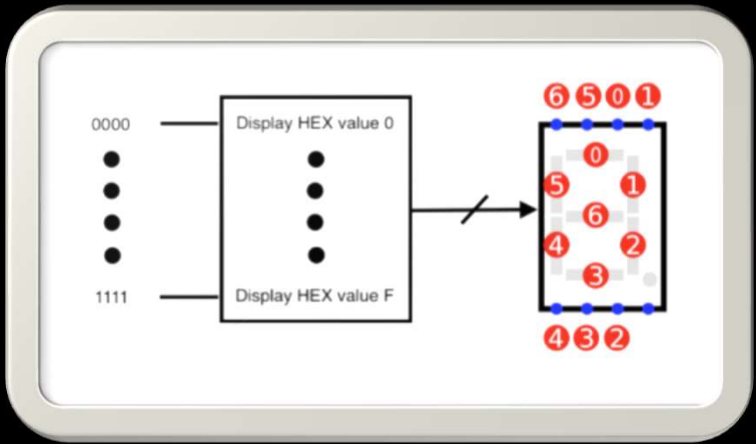
- Wiring:
  - Splitter: Splits buses into individual wires or smaller buses. Works both ways.
  - Clock
  - Constant: Outputs a constant value (can be multiple bits on a bus).
  - Bit Extender: Pads or sign extends bits on a bus.
- You can even make a transistor circuit with the components at the bottom ☺



Useful Components in Logisim

- Arithmetic and memory:
  - Some of the arithmetic components will be useful in later labs. Details about each one can be found i:n http://www.cburch.com/logisim/docs/2.3.0/libs/arith/index.html
    - This doc is for an earlier version, some components may look different now.
  - We will be exploring the memory components later in the course.

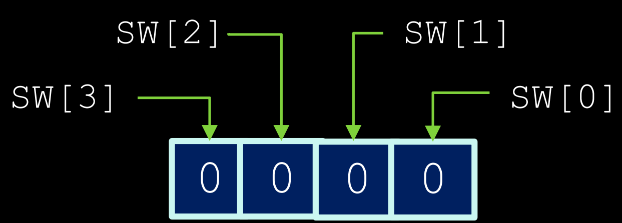
Tasks for Lab 2

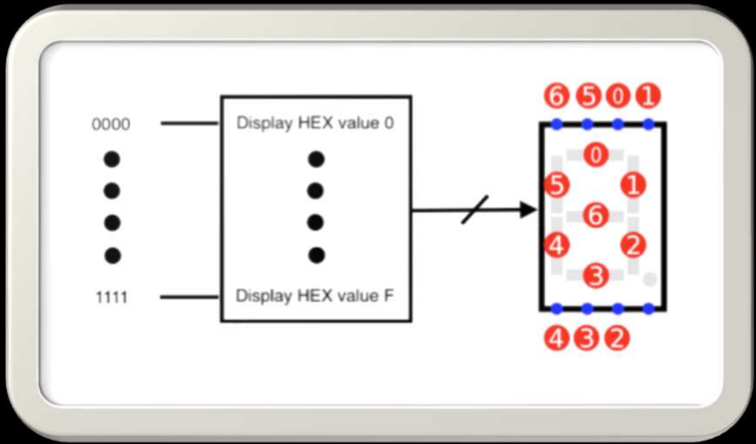


- The diagram on the right illustrates how to use the inputs to the 7-segment decoder (or the HEX decoder) to activate the segments.
- Each segment is active-high, meaning that if you set an input to `1`, the corresponding segment will turn on.
  - The DE1-SOC board is the opposite (i.e. active-low)

Tasks for Lab 2

- Ultimate goal:
  1. Take a 4-bit input coming from the switches on the and interpret those as binary number:

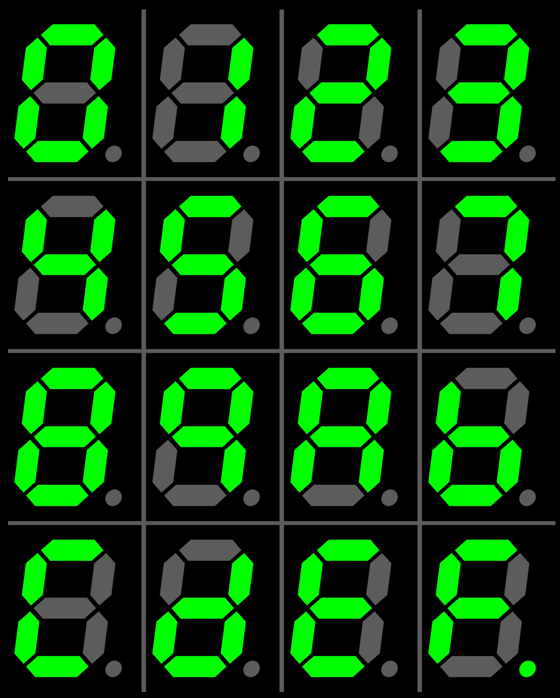




  2. Create seven circuits, one for each segment on the right to activate each segment based on the 4-bit input values.
     - For example: If input is `0000`, display “`0`” on the segments. If input is `1111`, display “`F`” on the segments.

Activating 7-seg displays

- The diagram on the right illustrates the 16 digits we want to show on the 7-segment display.



- How do we make this happen?
  - Consider segment 0 (the top segment in each digit).
  - Need to set it high in the following input cases:

Input		Display
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 7-seg displays

- <u>Answer:</u> Karnaugh Maps!

	$\overline{SW1}*\overline{SW0}$	$\overline{SW1}*SW0$	$SW1*SW0$	$SW1*\overline{SW0}$
$\overline{SW3}*\overline{SW2}$				
$\overline{SW3}*SW2$				
$SW3*SW2$				
$SW3*\overline{SW2}$				

- If we can fill in these table values, we can figure out the circuit's behaviour.

Segment 0 truth table

SW3	SW2	SW1	SW0	HEX[0]
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Segment 0 Karnaugh Map

- Now to fill in the table below….

	$\overline{SW1} * \overline{SW0}$	$\overline{SW1} * SW0$	$SW1 * SW0$	$SW1 * \overline{SW0}$
$\overline{SW3} * \overline{SW2}$	1	0	1	1
$\overline{SW3} * SW2$	0	1	1	1
$SW3 * SW2$	1	0	1	1
$SW3 * \overline{SW2}$	1	1	0	1

- What are the groupings that you see here?
  - Yes, overlapping is allowed ☺

Segment 0 Karnaugh Map

- What are the equations for these groups?

	$\overline{SW1}*\overline{SW0}$	$\overline{SW1}*SW0$	$SW1*SW0$	$SW1*\overline{SW0}$
$\overline{SW3}*\overline{SW2}$	1	0	1	1
$\overline{SW3}*SW2$	0	1	1	1
$SW3*SW2$	1	0	1	1
$SW3*\overline{SW2}$	1	1	0	1

SW2*SW1*SW0

SW2*SW1

SW3*SW1*SW0

SW3*SW1

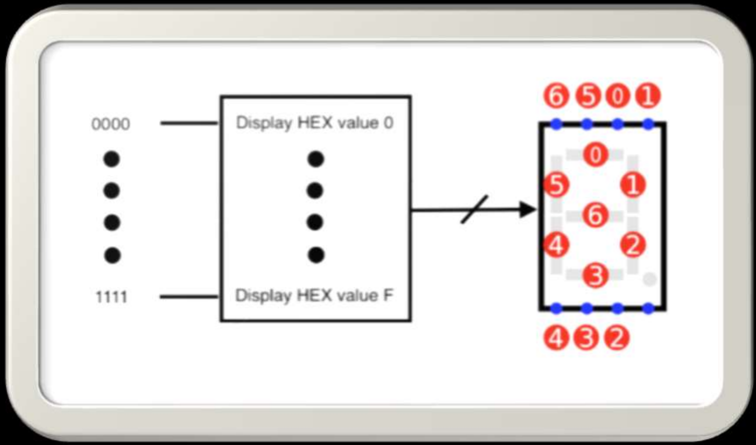
SW2*SW1*SW0

SW3*SW0

Can you figure out which terms are inverted to make these groups work?

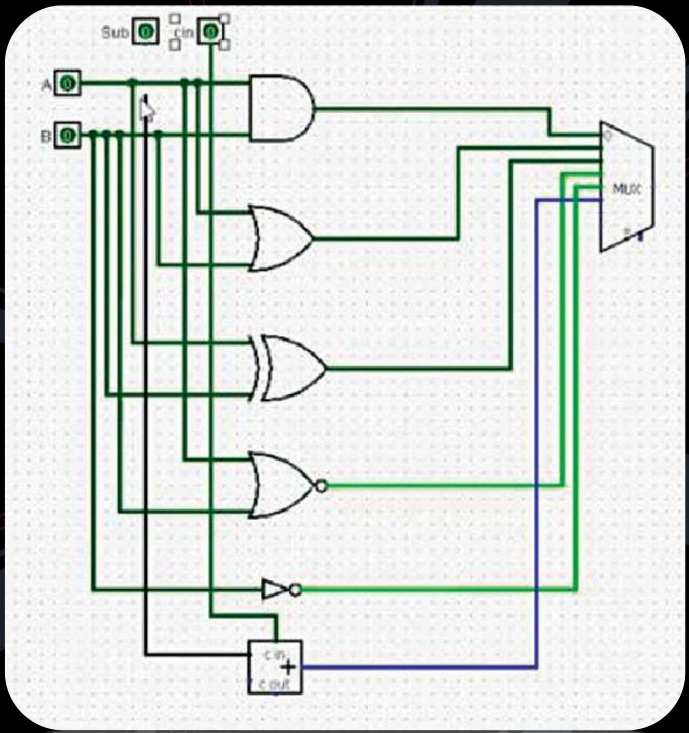
Tasks for Lab 2

- Repeat this process seven times to implement the behaviour for each of the seven segments in the HEX display.
  - Try to get the minimal circuit for each!



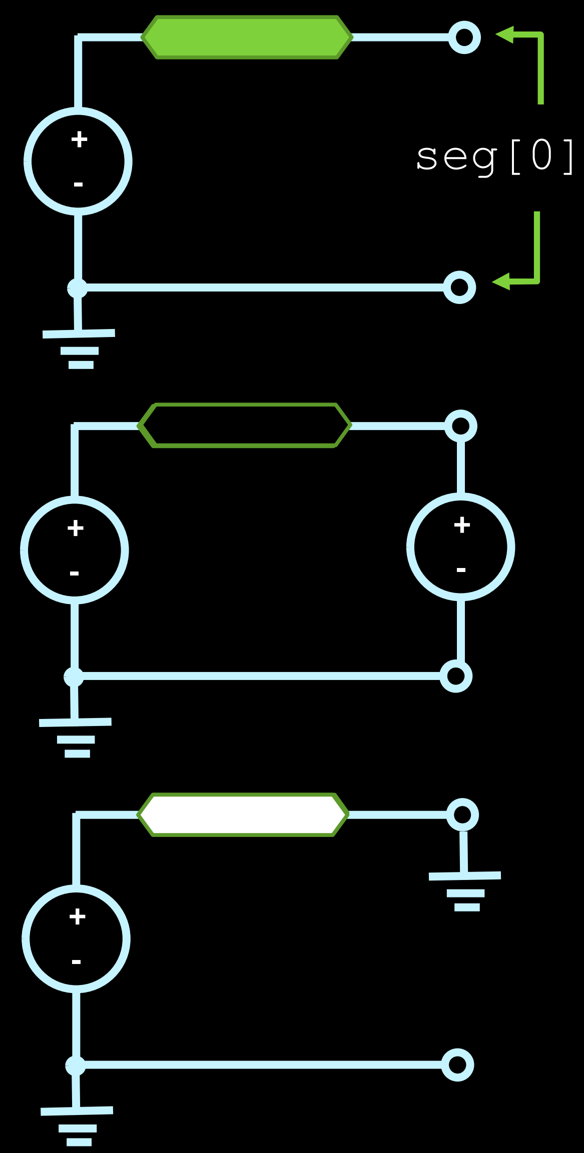
- Once you're done, your seven circuits go into the decoder module in the middle of the diagram above.
  - Make sure to test each segment as you go!

Using Logisim with DE1-SOC



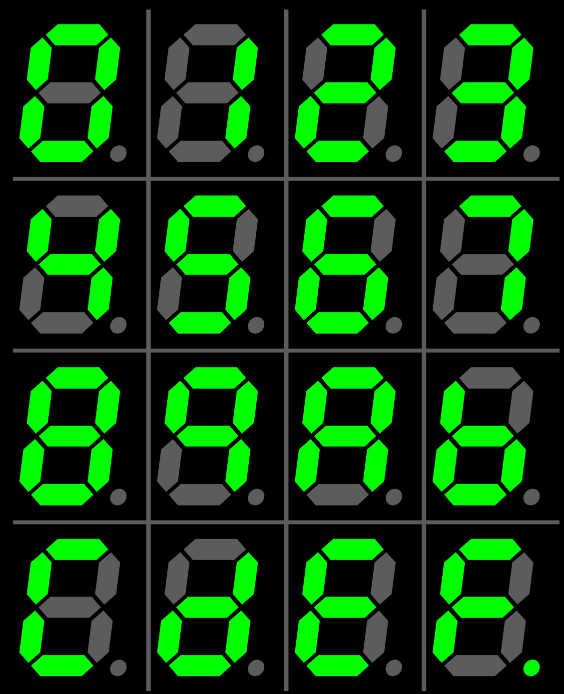
How 7-segments work on DE1

- The default for 7-segments on the DE1 boards are "active low", meaning that they turn on when their input signal is `0`, not `1`.
  - If you set segment `0` high, there is no voltage drop across the segment, so it doesn't turn on.
  - If you set segment `0` low, the voltage drop across the segment makes current flow, causing it to turn on.
- The default for 7-segments in Logisim are "active high" therefore you need to change this in properties.



Activating 7-seg displays

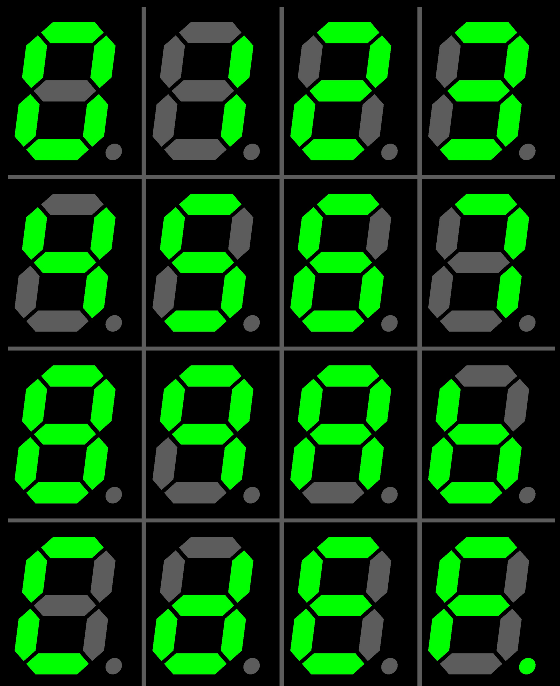
- Need to set segment 0 (top segment) **low** in these input cases instead:
  - 0000 -- "0"
  - 0010 -- "2"
  - 0011 -- "3"
  - 0101 -- "5"
  - 0110 -- "6"
  - 0111 -- "7"
  - 1000 -- "8"
  - 1001 -- "9"
  - 1010 -- "A"
  - 1100 -- "C"
  - 1110 -- "E"
  - 1111 -- "F"



- 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?