Basestation:
FPGA and VGA and DDS, Oh My!

September 27, 2017

By Claire Chen for ECE3400

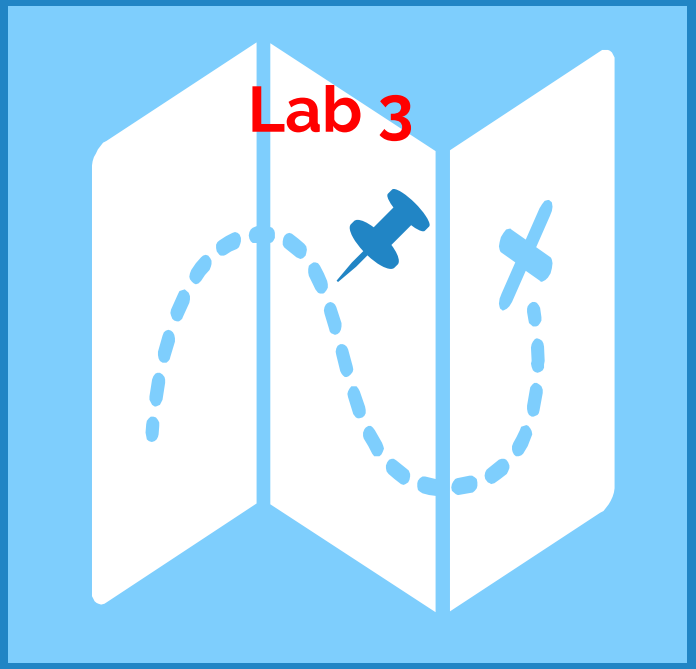
From Monday...

- What does FPGA stand for?
- What are 3 main parts of an FPGA?
- Verilog vs. C

Today:

Lab 3!

Everything you need to know



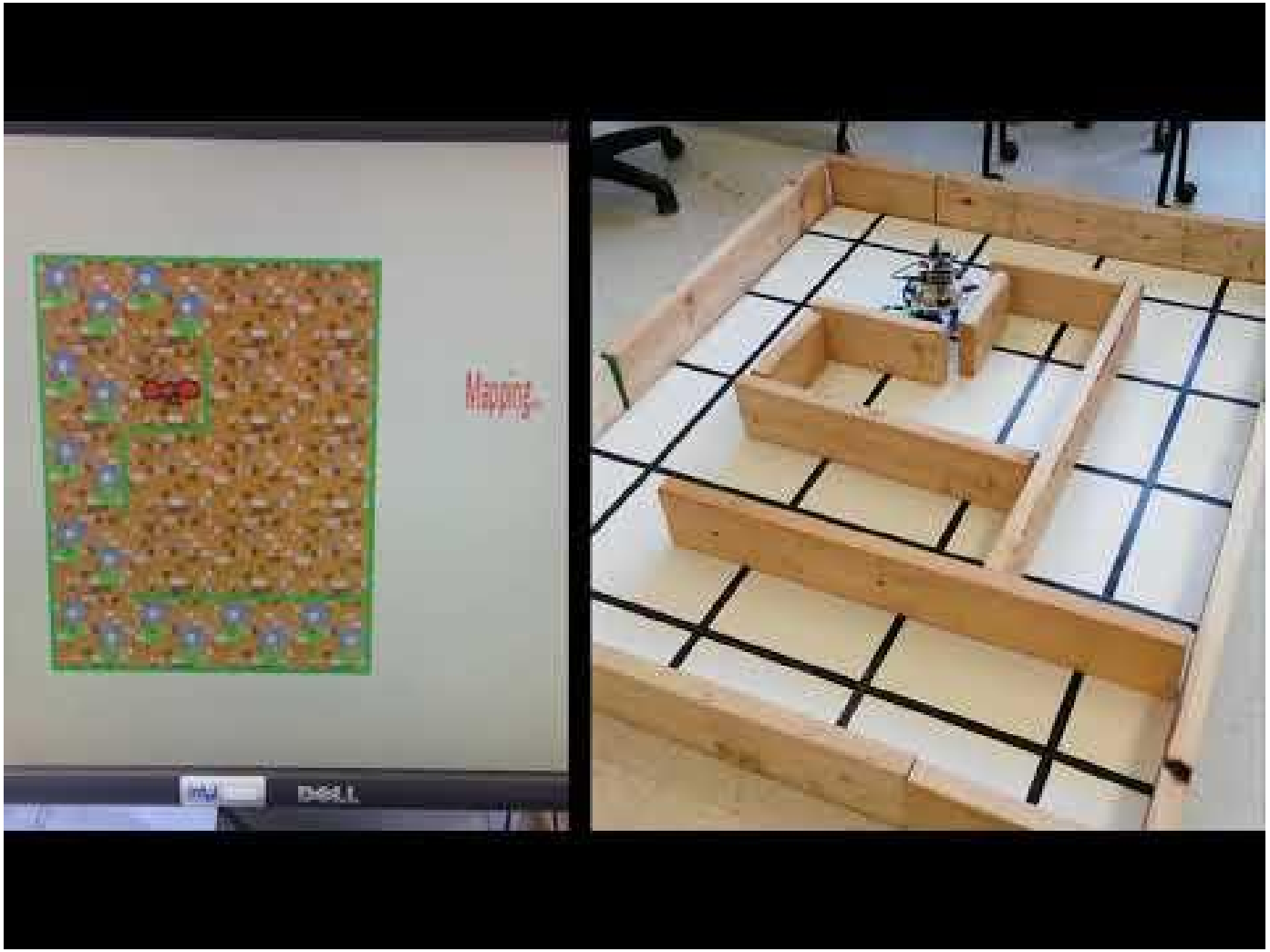
A stepping stone

To the base station

Basestation Requirements

- Arduino receives maze information via radio and transmits this to FPGA
- Display robot's progress as it navigates
  - Wall location, treasure grids
  - Unexplored vs. explored grids
- Display done signal and play 'tune' signaling that robot is done mapping
- Show treasure frequencies
- Show unexplorable areas

An example:



Lab 3 Objectives and Timeline

Graphical

- Input 2 signals into FPGA
- Draw small grid on VGA display
- Change grid state based on inputs

Audio

- Input signal into FPGA
- Output signals from FPGA into speakers, via DAC
- Play a three-tone done signal from FPGA

Timeline

This week: Complete milestone 2, continue refining line following, start thinking about maze representation

Next week (Oct 2): Start lab 3

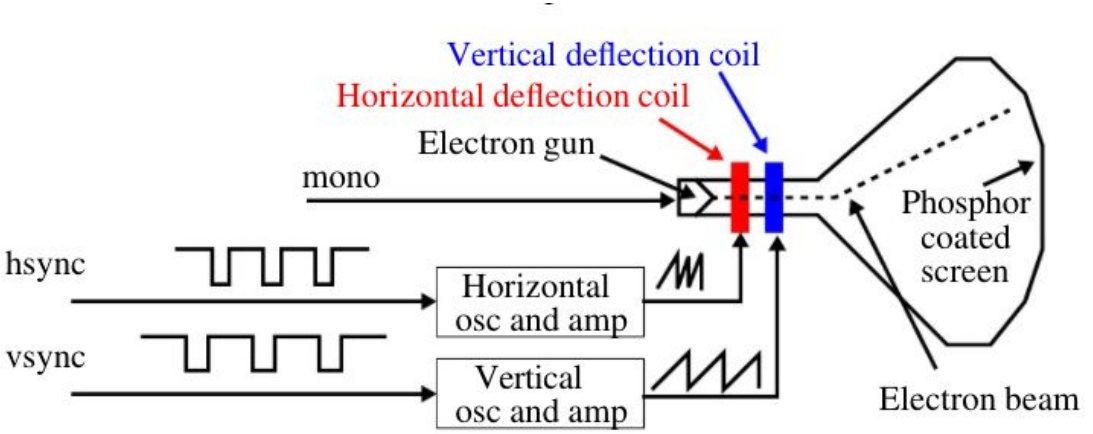
Next, next week (Oct 9): Continue lab 3

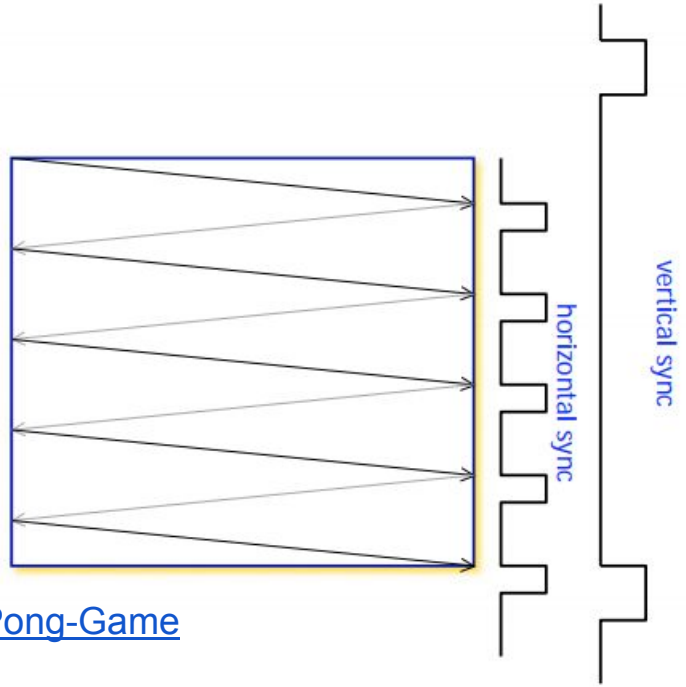
3 weeks from now (Oct 16): Lab 3 due

VGA System

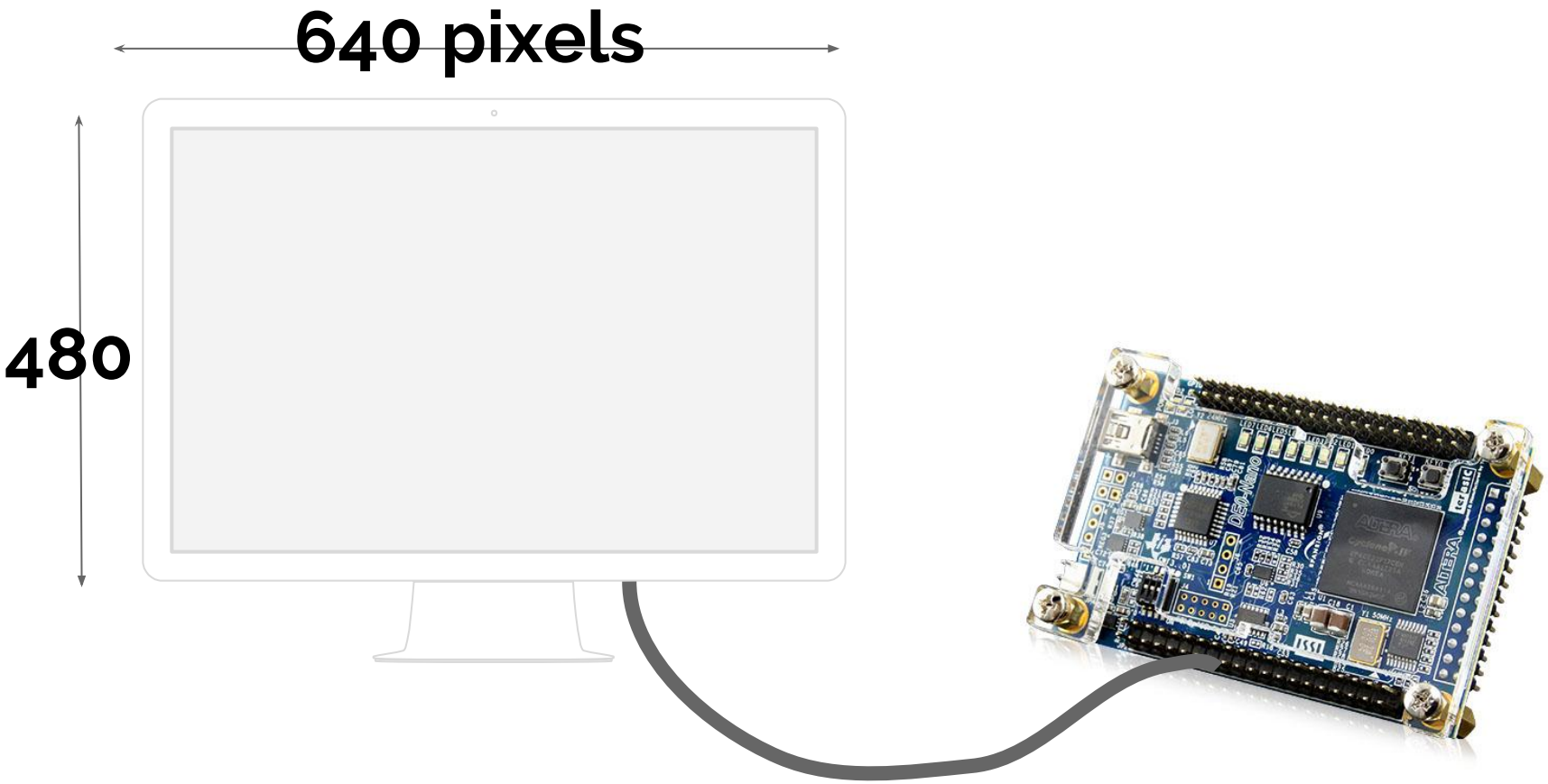
How does VGA work?

- Signals sent serially
- R, G, B color - analog
- 2 synchronization signals, H-sync and V-sync
- 60Hz refresh rate (54ns per pixel) → Use 25MHz clock (40ns per pixel)



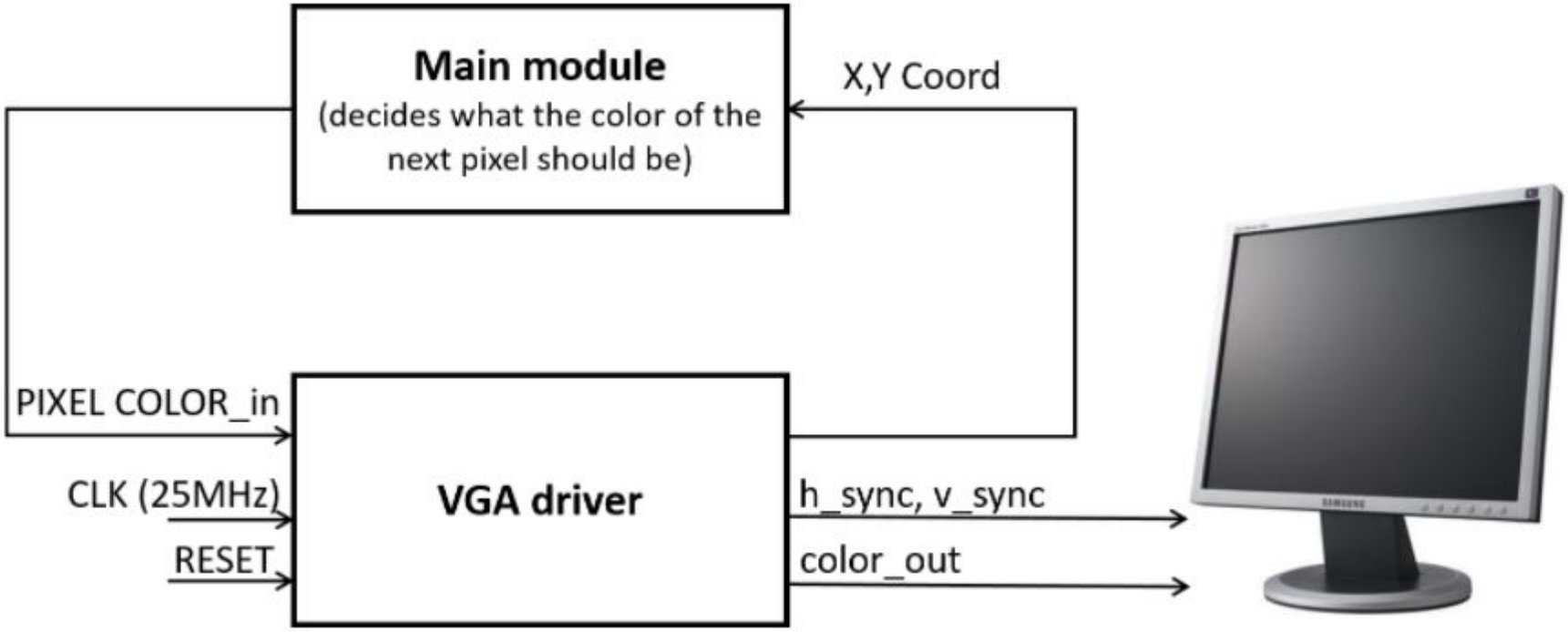


Images from: https://www.scribd.com/document/97633111/Vga-Ping-Pong-Game
http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/VGA.pdf



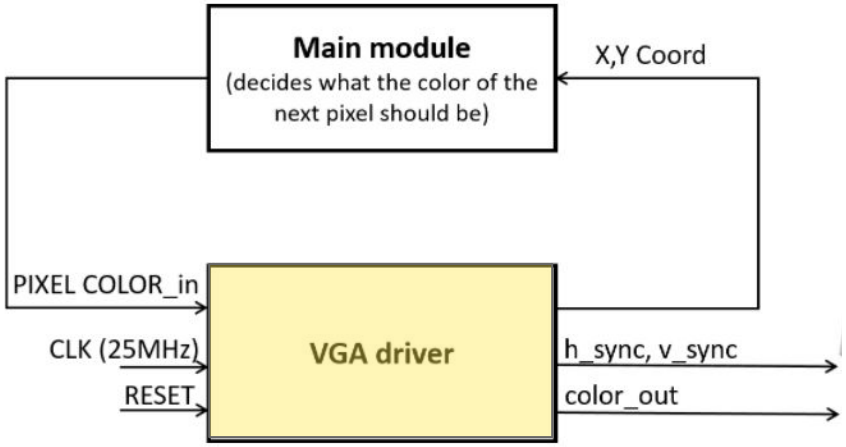
FPGA VGA Controller → VGA Connector → VGA Cable

FPGA Video Controller



VGA Driver Module

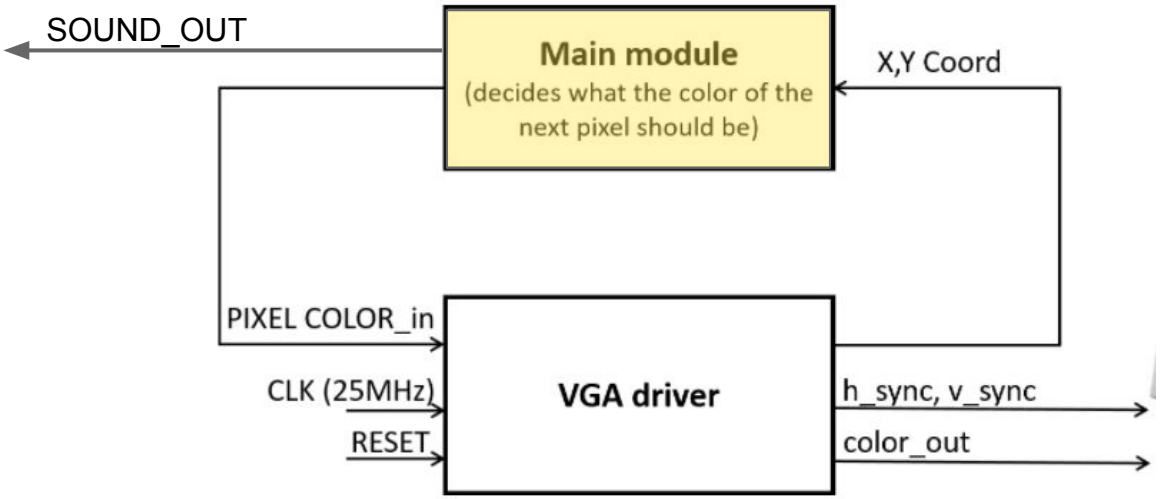
- Inputs?
- Outputs?
- Color format?



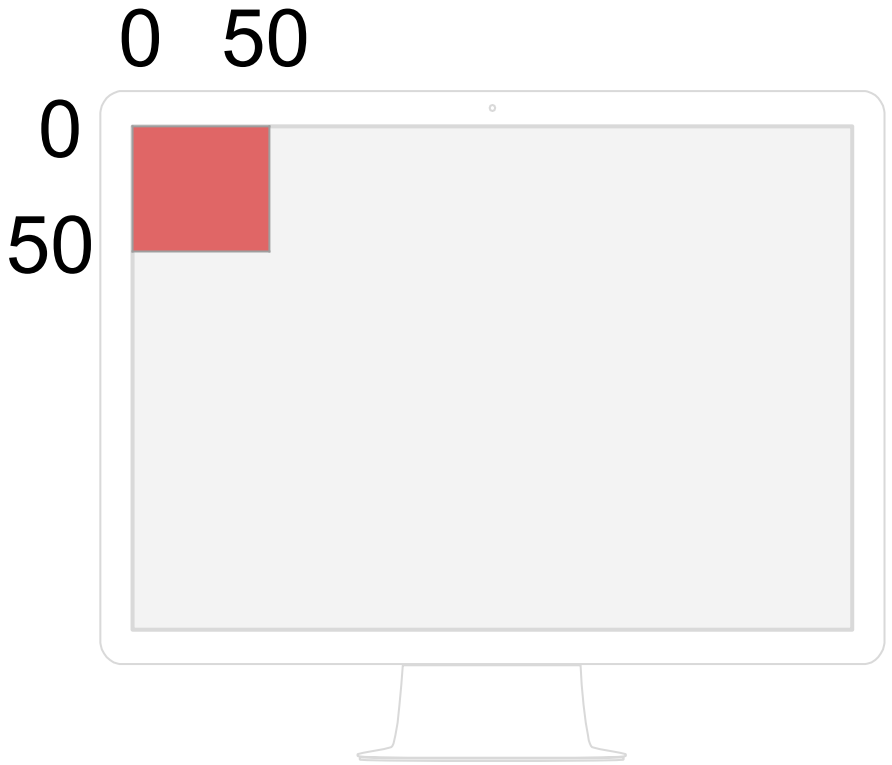

```
25   input CLOCK; //PIXEL CLOCK - DRIVE AT 25MHZ for 60 Hz 640 x 480 VGA
26   input RESET;
27   input  [7:0] PIXEL_COLOR_IN; //COLOR GIVEN TO THE VGA DRIVER
28
29
30   output [9:0] PIXEL_X; //HORIZONTAL POSITION OF THE NEXT PIXEL;
31   output [9:0] PIXEL_Y; //VERTICLE POSITION OF THE NEXT PIXEL;
32   output [7:0] PIXEL_COLOR_OUT; //COLOR TO BE DISPLAYED
33   output       H_SYNC_NEG; //THE REVERSE POLARITY HORIZONTAL SYNC SIGNAL
34   output       V_SYNC_NEG; //THE REVERSE POLARITY VERTICAL SYNC SIGNAL
```

Top Level Module

- Contains logic to
  - Set pixel color
    - Need memory configuration to store pixel data for screen
  - Output sound
    - Need to consider clock period to create sounds of different frequencies



Drawing on VGA Display



Goal: Draw 50p x 50p red square

- Can set pixel color based on coordinates output by VGA Driver Module

```
if x_coord < 50 and y_coord < 50
        pixel_color = red
else
        pixel_color = grey
```

Drawing on VGA Display

How about graphics?!

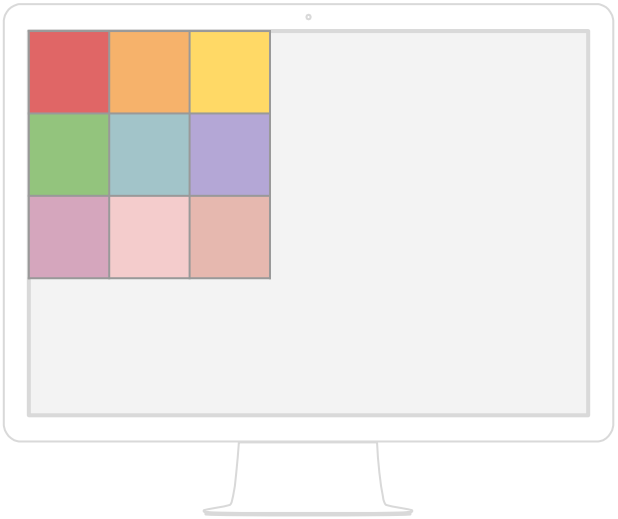
Goal: Draw a 3 x 3 rainbow grid

- Could set pixel color based on coordinates output by VGA Driver Module like before…
- Would be a lot of logic…
- What if we want to change the colors of squares?

```
if x_coord < 50 and y_coord < 50
        pixel_color = red
else if x_coord < 100 and y_coord < 50
        pixel_color = orange
else if x_coord < 150 and y_coord < 50
        pixel_color = yellow
…… many more else-ifs ……
```

Need some way to store pixel data...

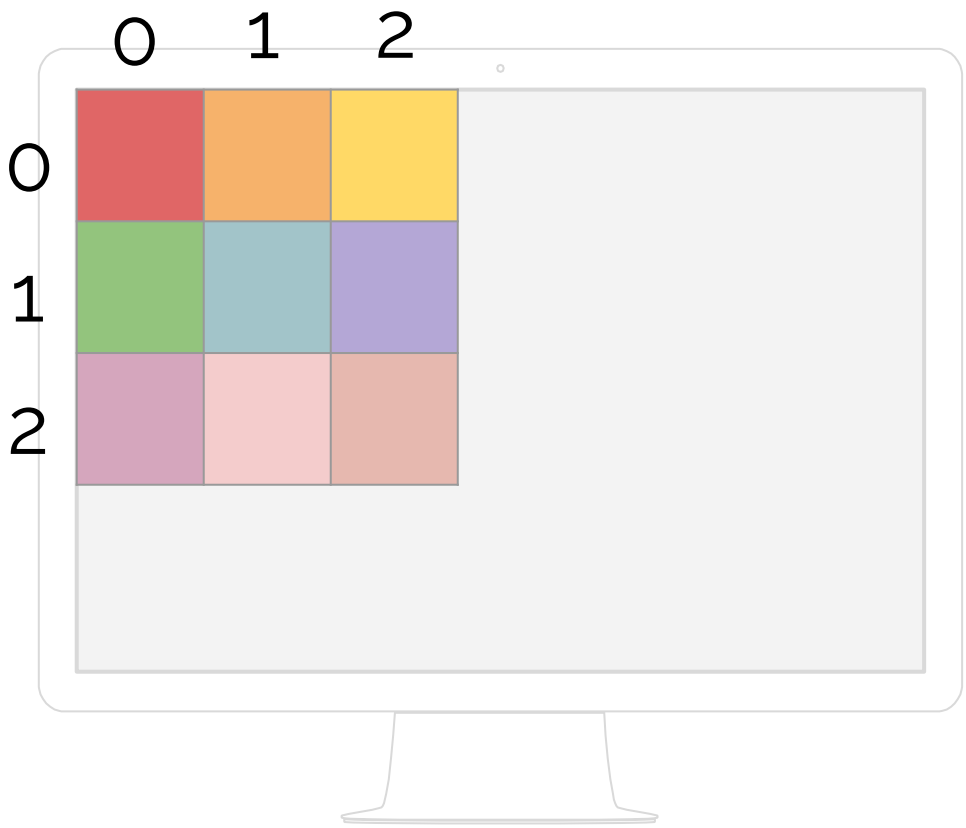
- Make a memory array
    - reg [b-1:0] my_regs [m-1:0][n-1:0]
    - a m x n array of b-bit registers
- Could have array of registers to store color of every pixel in the grid
    - reg [7:0] my_grid [m-1:0][n-1:0]
    - m = ? n = ?



- But that would require a lot of memory, particularly for bigger grids
- Think about memory constraints of FPGA!

Need some way to store pixel data...

- Can take advantage of grid scheme
- Instead of storing every pixel's data, store information about each grid square (such as color)

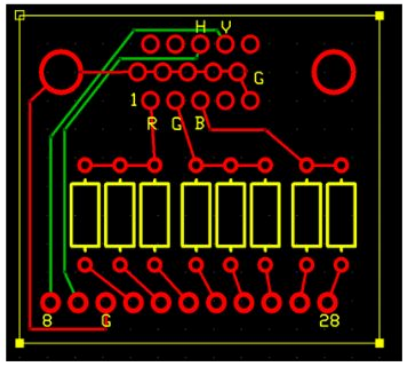



```
// 3x3 grid, 9 color options
reg [3:0] my_grid [2:0][2:0]

// set grid[0][0] to red
my_grid[0][0] = 4'd0; red

if (pixel is in my_grid[0][0] &
my_grid[0][0] == 4'd0;)
        draw as red
```

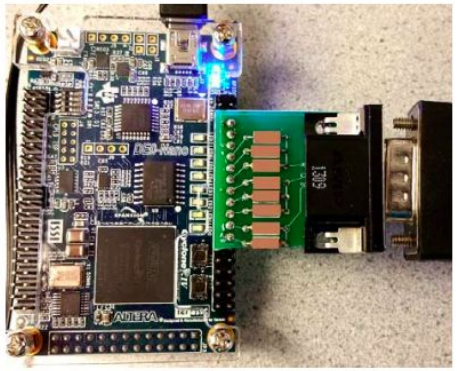
- Just need to determine where pixel is in grid
- Can display much more interesting graphics

VGA Connector

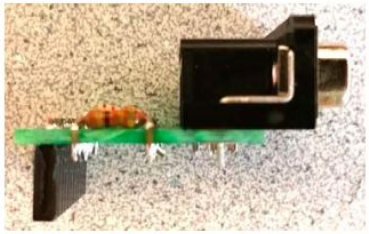
- Resistive 8-bit DAC
- Converts digital signals from FPGA to analog signals between 0-1V required for VGA cable
- How were resistor values chosen (talk about in lab 3 report)?



VGA PCB design



PCB connected to FPGA (resistor values obscured)



Side view of PCB with soldered female header, resistors, and VGA connector

IMPORTANT NOTE!!

The FPGA pins are **3.3V**

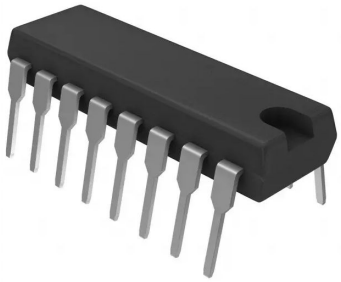
The Arduino pins output **5V**

You *must* build a voltage divider before connecting Arduino pins to FPGA pins

* common ground! *

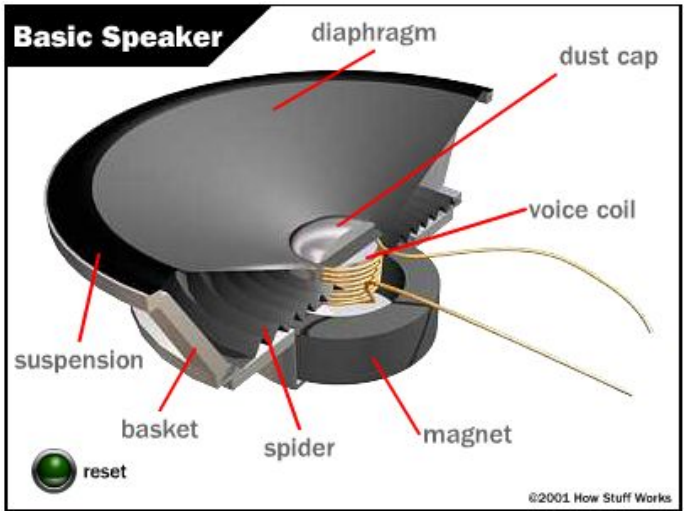
Sound Generation

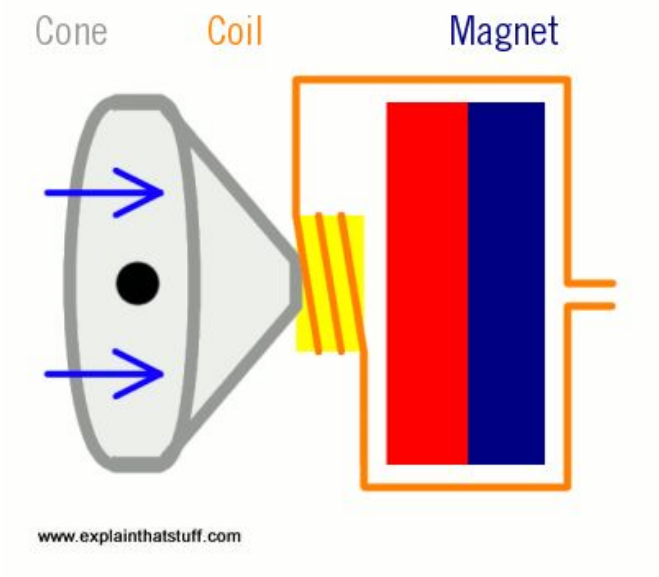
Audio System



FPGA → R2R DAC → Head phone socket → Speakers

How do speakers work?





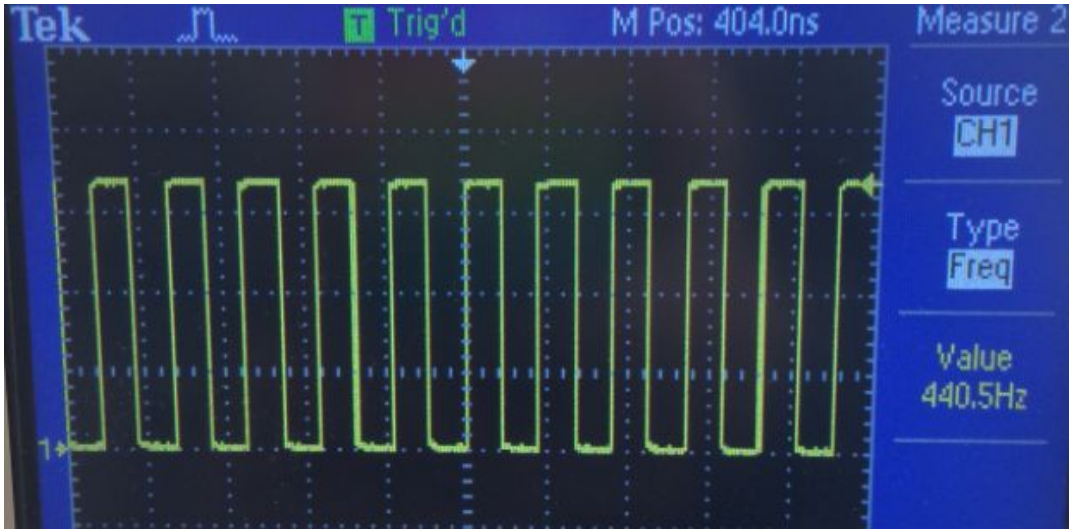
- Essentially the reverse of a microphone
- Electrical signals → physical vibrations
- Lots of parts, including:
  - Diaphragm
  - Voice coil
  - Magnet
- Coil is in constant magnetic field
- Running current through coil creates electromagnet
- Switching current moves EM back and forth, creating sound waves

Sound generation is about timing

- Goal: output wave of a desired frequency
- Can create a 440Hz square wave by toggling one bit at that frequency
- Must consider clock frequency

An exercise:
The system clock runs at 25MHz. To output a 440Hz square wave, every how many clock cycles should I toggle the output bit?

Answer: 25MHz/440Hz/2 = ~28409 cycles

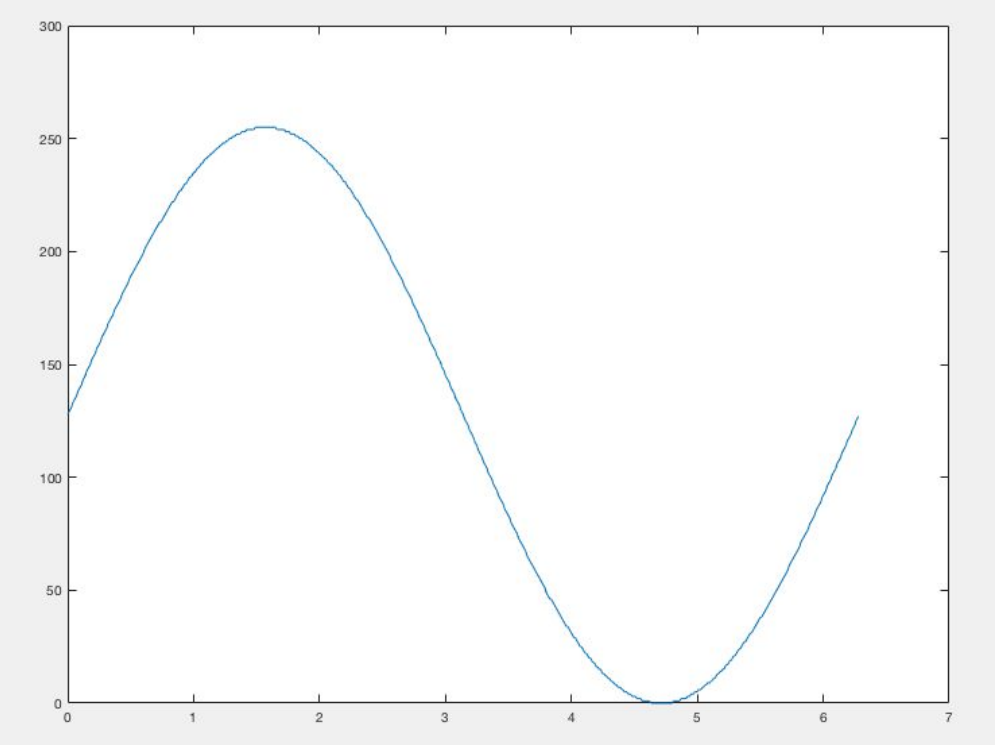


How about more complicated waves?

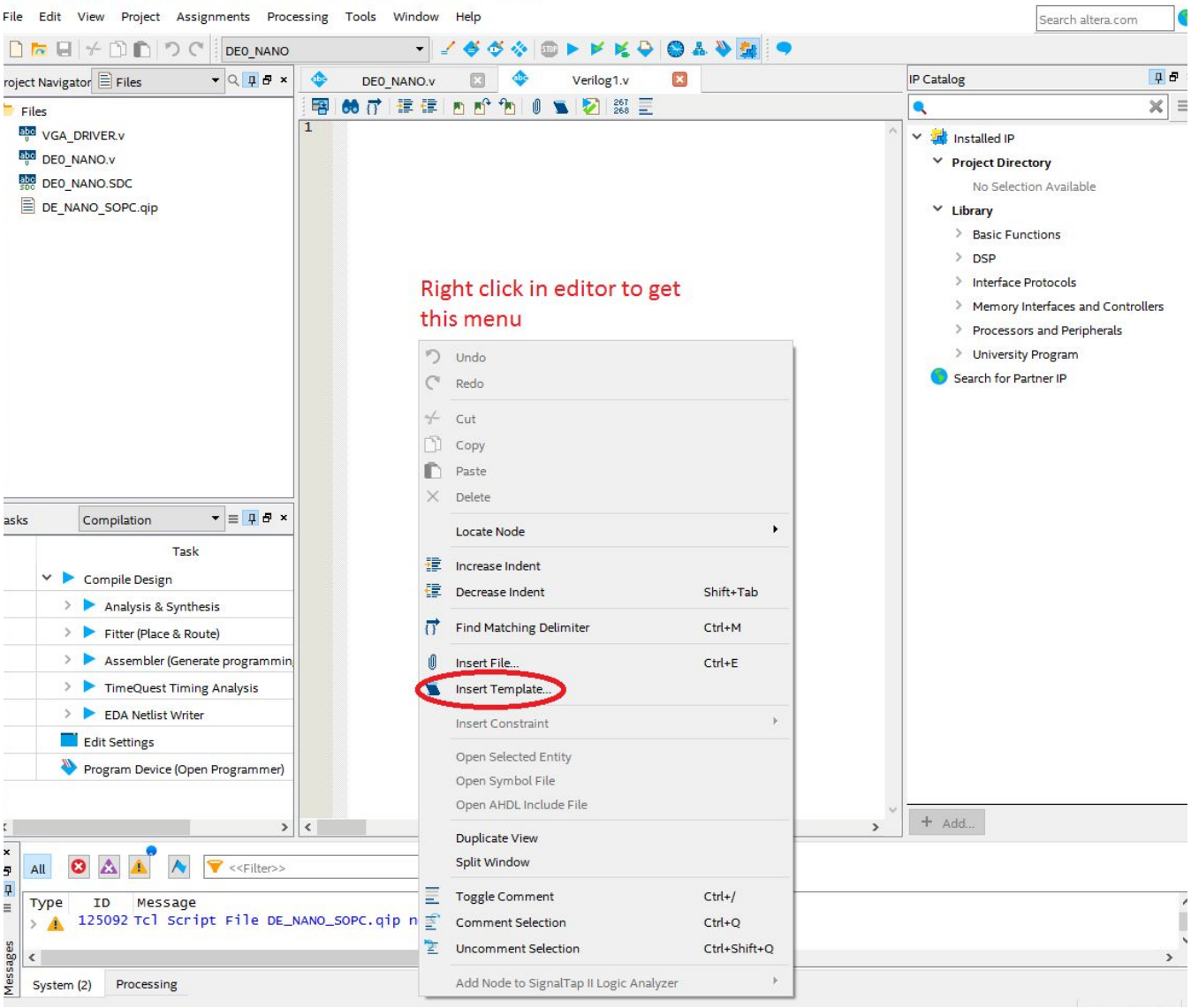
- Use 8-bits to create analog signals via DAC
- Direct digital synthesis from a sine table in ROM
- Create sine table with values from 0 to 255 in Matlab
- Iterate through this sine table at different frequencies to output sine values

An exercise:
The system clock runs at 25MHz and my sine table contains 256 values. Every how many cycles should I iterate to the next element of the sine table to create a 440Hz wave?

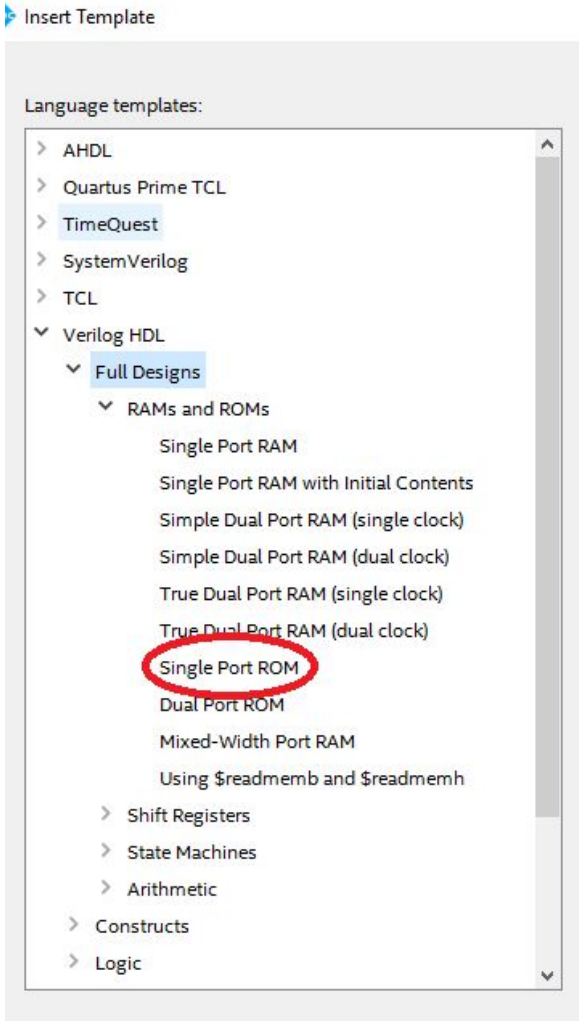
Answer: (1/440Hz) / (40ns) / 256 = ~222 cycles



Inferred Memory Blocks - example



Inferred Memory Blocks - example

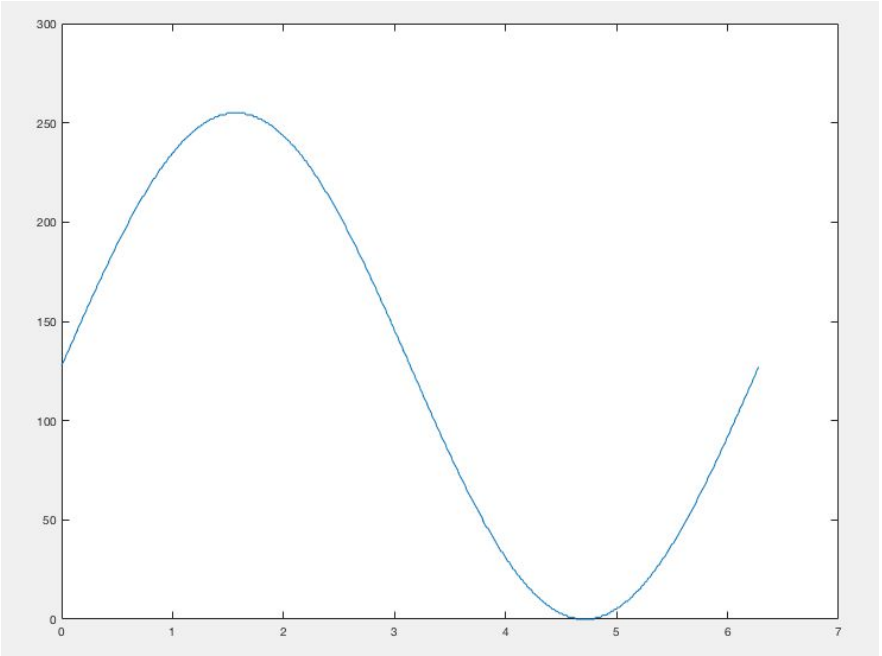



```
// Quartus Prime Verilog Template
// Single Port ROM

module single_port_rom
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=8)
(
    input [(ADDR_WIDTH-1):0] addr,
    input clk,
    output reg [(DATA_WIDTH-1):0] q
);

    // Declare the ROM variable
    reg [DATA_WIDTH-1:0] rom[2**ADDR_WIDTH-1:0];

    // Initialize the ROM with $readmemb.  Put the memory contents
    // in the file single_port_rom_init.txt.  Without this file,
    // this design will not compile.

    // See Verilog LRM 1364-2001 Section 17.2.8 for details on the
    // format of this file, or see the "Using $readmemb and $readmemh"
    // template later in this section.

    initial
    begin
        $readmemb("single_port_rom_init.txt", rom);
    end

    always @ (posedge clk)
    begin
        q <= rom[addr];
    end

endmodule
```

Initialize ROM variable elements here

Inferred Memory Blocks - example

```
module SINE_ROM
(
        input [9:0] addr,
        input clk,
        output reg [7:0] q
);

        // Declare the ROM variable
        reg [7:0] sine[628:0];

        sine[0] <= 8'b10000000;
        sine[1] <= 8'b10000001;
        sine[2] <= 8'b10000010;
        sine[3] <= 8'b10000011;
        sine[4] <= 8'b10000101;
        sine[5] <= 8'b10000110;
          ...
        sine[624] <= 8'b01111010;
        sine[625] <= 8'b01111011;
        sine[626] <= 8'b01111101;
        sine[627] <= 8'b01111110;
        sine[628] <= 8'b10000000;
```



How many bits do you think this ROM will take up memory?

Inferred Memory Blocks - example

- The FPGA chips in lab have 594 kb of embedded memory (608,256 bits)

- Our ROM table uses 5032 bits!

Synthesis Report for sine table ROM

Flow Summary	
Flow Status	Successful - Fri Sep 22 14:51:58 2017
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	DE0_NANO
Top-level Entity Name	DE0_NANO
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	192 / 22,320 (< 1 %)
Total combinational functions	191 / 22,320 (< 1 %)
Dedicated logic registers	100 / 22,320 (< 1 %)
Total registers	100
Total pins	87 / 154 (56 %)
Total virtual pins	0
Total memory bits	5,032 / 608,256 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Thank you!

Any questions?

Lab: Monday Eve

Open Lab: Tuesday Afternoon