Team Members

We are submitting our own work, and we understand severe penalties will be assessed if we submit work for credit that is not our own.

Print Name	ID Number
Print Name	ID Number

Estimated Work Hours

1 2 3 4 5 6 7 8 9 10

5%will be deducted from total score for each day late

Point Scale

4: Exemplary
3: Complete
2: Incomplete
1: Minor effort
0: Not submitted

Score = Points awarded (Pts) x Weight (Wt)

GRADER

Deliverables / Team Demo	Wt	Pts	Score	Grader Signature	Date	Days Late
Demo working Project	1.5					
Documentation / Source Code	3.0					
Team Accountability Report	0.5					

Total In-Lab Score

General Statement: Your team (two) must implement a sequential circuit that will address all of the locations in a 256**x**16 RAM module. You are to display both the address inputs (to the RAM) and the data outputs (from the RAM) on the 7-segment displays. Note: top-level module may be either a verilog module or a schematic.

Design Specifications: This design is to be comprised of eight different modules, three of which were already in the previous sequential labs (**clk_500Hz**, **one_shot** and **hex_to_7seg**). The remaining five are listed below:

The first three modules make up the "***Display Controller***" -- see page 2 for more explanation.

1: **pixel_clk** -- a 480 Hz clock used to "time multiplex" the common anode inputs to the 7-segment displays.
2: **pixel_controller** -- generates the signals for the common anode inputs to the 7-segment displays and also generates the multiplexer select signals for multiplexing the address/data nibbles.
3: **ad_mux**: an 8-to-1 MUX for multiplexing address/data information to the **hex_to_7seg** module.
4: **addr_seqr** -- used to output an 8-bit address for each location (sequence through) in the RAM. This sequencer is to be clocked by a debounced push button switch (e.g. address_step).
5: **ram1** -- used to store 256 16-bit values, initially starting with **55FFh** @ address **00h**, **55FEh** @ **01h**, **55FDh** @ **02h**, **55FCh** @ **03h**, … **5501h** @ **FEh**, **5500h** @ **FFh**. We'll use the Xilinx "**Coregen IP**" Core generator to generate a verilog **Instantiation Template File** that will be used in creating this module. Details of using the Xilinx **Coregen IP** tool starts on page 4. **NOTE:** clk of the ram1 is to be clocked by the 50 MHz (Nexys2) or 100 MHz (Nexys3) clock!

All nine modules will be written in verilog. Your "top-level" module may be either a structural verilog module or a schematic diagram. All nine of the modules must be interconnected from this "top level" module via appropriate module instantiations. If you use a schematic for interconnection, all modules must be created as "symbols" in the schematic using the "**Symbol Wizard**" in the Schematic Editor (see page 3 for details). You need to develop each module, and then interconnect them appropriately. Note that for this lab, you'll have one state diagram and table.

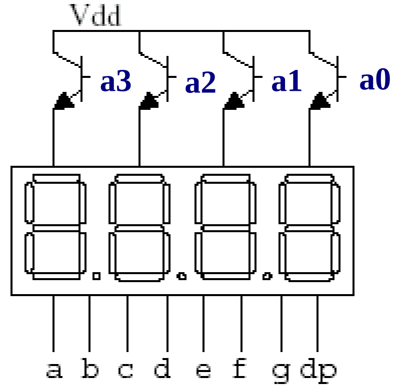
Requirements: Your team must turn in one copy of **(1)** this cover sheet, followed by **(2)** a printout of the "top-level" verilog module **or** schematic, **(3)** a state diagram & table of the **pixel_controller**, **(4)** a "top-down" sequence of **all** the verilog modules, followed by, lastly, **(5)** the completed "**CECS 301 Team Accountability Reporting Sheet**" (p. 8).

Due Date: Monday, March 13, 2017 **{Monday of Week 8}**

Display Controller (Four 7-segments)

To cause a cohesive 4-digit number to be displayed on the Digilent display, the anode signals (**a3, a2, a1, a0**) and corresponding cathode patterns must be asserted in a regular, repeating sequence. The Digilent Spartan3 board is configured such that both the anodes (a3..a0) and cathodes (a,b,c,d,e,f,g) are asserted when they are low. For example, consider the example of displaying the number "5F9C" on the displays. First, the 7 cathode signals for a digit "C" must be output to the 7 segments while **a0** is driven low and **a3, a2, and a1** are driven high. Second, the 7 cathode signals for a digit "9" must be output to the 7 segments while **a1** is driven low and **a3, a2, and a0** are driven high. Third, the 7 cathode signals for a digit "F" must be output to the 7 segments while **a2** is driven low and **a3, a1, and a0** are driven high. Fourth, the 7 cathode signals for a digit "5" must be output to the 7 segments while **a3** is driven low and **a2, a1, and a0** are driven high. This "scanning" sequence must go on continuously for each 7-segment display to stay illuminated.

Anodes -- connected to FPGA via transistors for greater current



Cathodes -- connected to FPGA pins via 100Ω resistor

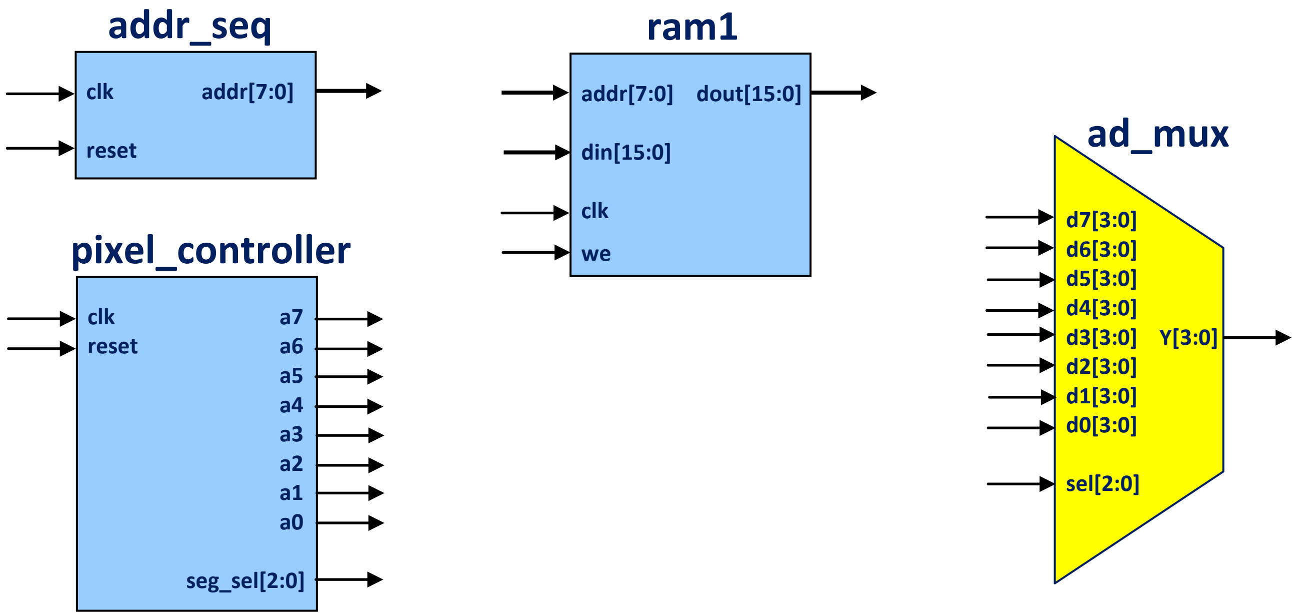
In order that scanning not be noticed by a human observer, the four anode patterns and their corresponding cathode signals should be driven sequentially at a rate of no less than 45Hz, and more preferably at 60Hz. Thus, all four anode signals and cathode patterns should be driven at least once every 16ms -- 4 ms per anode/cathode pattern. A single cycle wherein each display element is driven once in sequence is known as a "refresh" cycle. Thus, we can use a clock with a period of 4 ms, corresponding to a frequency of 250 Hz.

The above discussion assumes that we are controlling only four 7-segments displays. As indicated on the previous page, the "***Display-Controller***" developed for this lab will be comprised of four modules. First is the **pixel_clk** module that will "divide" the 100 MHz system clock down to 480 Hz clock for refreshing the eight 7-segment displays. Next is the **pixel_controller**, which generates the **a7**, **a6**, **a5**, **a4**, **a3**, **a2**, **a1**, and **a0** signals for the anode inputs to the 7-segment displays. Simultaneously, this module will also generate the 3-bit multiplexer select signals for the **ad_mux** module. This third, **ad_mux** module, is a simple 8-to-1 MUX for multiplexing the address/data nibbles into the fourth—**hex_to_7seg** module. The "high nibble" of the address (**addr[7:4]**) will be sent to the 7-segment displays in conjunction with anode **a5**. The "low nibble" of the address (**addr[3:0]**) will be sent to the 7-segment displays in conjunction with anode **a4**. The "highest nibble" of the data out of the RAM (**dout[15:12]**) will be sent to the 7-segment displays in conjunction with anode **a3**. The "next nibble" of the data (**dout[11:8]**) will be sent to the 7-segment displays in conjunction with anode **a2**.The "next nibble" of the data (**dout[7:4]**) will be sent to the 7-segment displays in conjunction with anode **a1**. The "lowest nibble" of the data out of the RAM (**dout[3:0]**) will be sent to the 7-segment displays in conjunction with anode **a0**.Thus, the Display Controller must send one of eight hex-nibbles (via the multiplexer) to the **hex_to_7seg** module (to drive the cathode patterns) and assert/deassert the appropriate anode signals ***at the same time.***

Since our **pixel_clk** will provide a 480 Hz frequency (8 "pixels" refreshed at 60 Hz) it will have a period equal to the desired single-digit display time (2.083 ms), the entire display is refreshed once every 16.67ms. This gives us a scanned display of 60 Hz.

Schematic Symbol Creation and Pinouts

Prior to running the Coregen IP module, you have to create all the symbols for the seven other verilog modules in the schematic editor (if you have a top-level schematic). In particular, the ram1 symbol has to have been created prior running Coregen. The graphics below shows the inputs and outputs for the **addr_seqr**, **ram1, pixel_controller** and **ad_mux** modules (symbols):

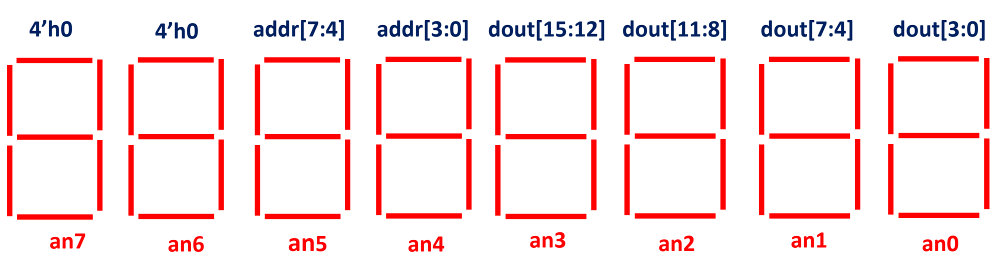


The **clk** input to the **pixel_controller** (e.g. "pixel_clk") is to come from the **pixel_clk** module (e.g. 480 Hz). The **clk** input to the address sequencer (**addr_seqr**), which is merely an eight-bit up counter, is to come from a debounced push-button switch—**Btn_Dwn** (e.g. **addr_step**).
NOTE: The **clk** input to the RAM module (ram1) is to come from the 100Mhz board clock.

The **we** input (i.e. write enable) to the RAM module (**ram1**), is to come from another debounced push-button switch **Btn_Right** (e.g. wr_pulse). The **reset** input for all synchronous modules is to come from push-button switch **Btn_Up**.

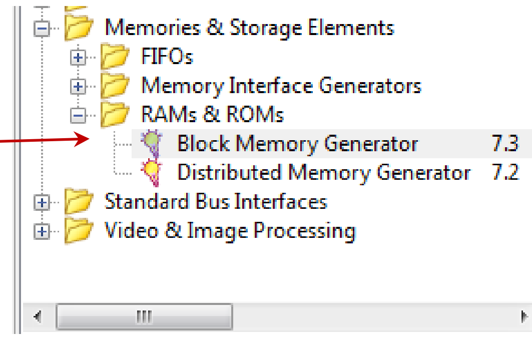
The **din(15:0)** inputs to the RAM (ram1) are to come from the sixteen slide switches (**SW15 din(15)** … **SW0 din(0)**).

The "mapping" of the RAM address inputs **addr [7:0]** and data outputs **dout [15:0]** to the 7-segment display is shown below:



Using Coregen IP

As shown on page 1, **right click** on your "top level" file name in the Sources window in the Project manager and select **New Source → IP (Coregen)**. Type **ram_256x16** for the filename. Click on **Next**, then select "**Block Memory Generator**" as shown in the graphic at the right — click **Next** and then **Finish** to invoke the Xilinx Core Generator for Single Port Block Memory.

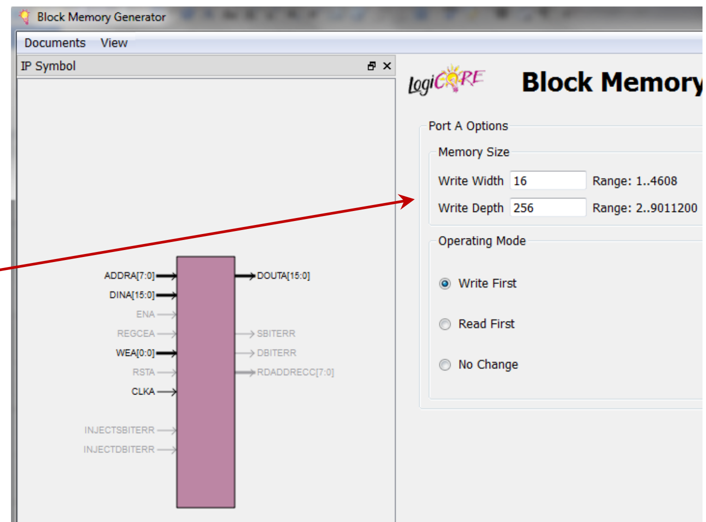


On "Pages 1 to 3" (of 6) of the Coregen dialog boxes, fill in the fields as shown:

Component Name: **ram_256x16**
Interface Type: **Native** — **Page 1**

Memory Type: **Single Port RAM** — **Page 2**

Write and Read Width: **16**
Write and Read Depth: **256** — **Page 3**



All other fields are to be left unchanged (default).

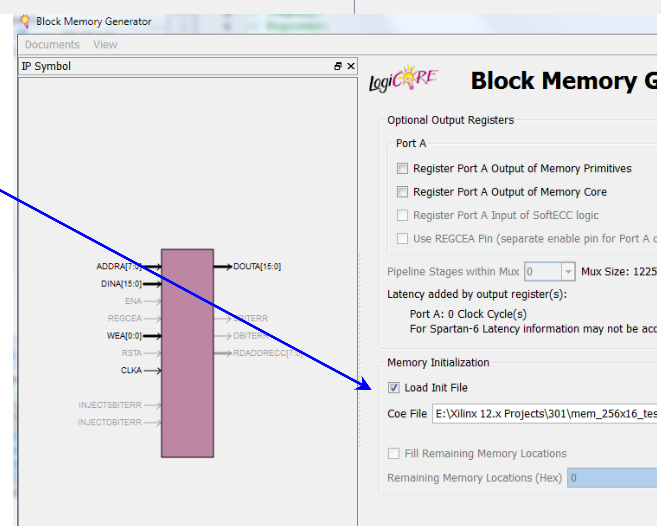
Click "**Next**" to go to page 4

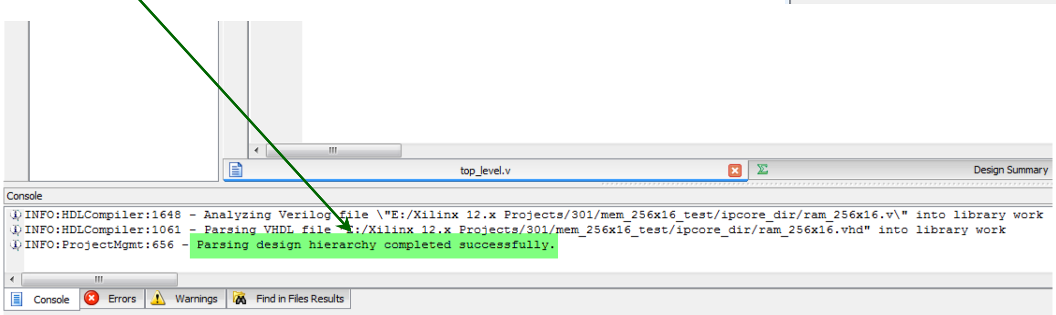
Now, **click** the **Load Init File** check box.

Next, **click** the **Browse File …** button, and specify the **ram_256x16_lab4.coe** file (provided by instructor) in the **Coe File** location.

After specifying the "Init File," **click** on the "**Generate**" button to generate the RAM module with the initial values.

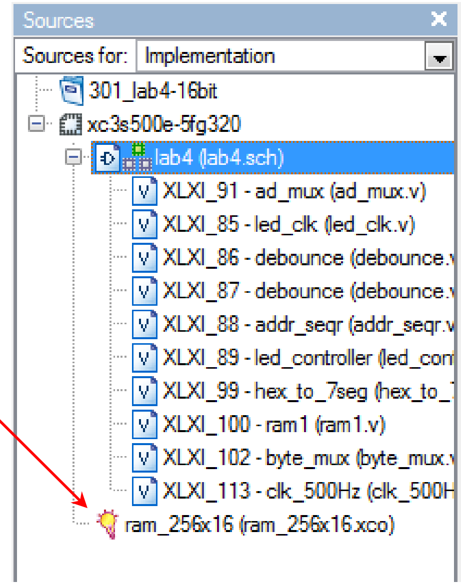
After clicking "Generate," CoreGen will generate the RAM core. **Note: this takes some time…be patient!** Successful generation is indicated in Xilinx console as shown in the graphic below:

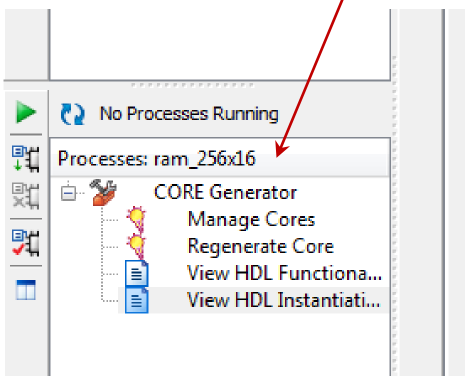




Using Coregen IP (cont.)

You should now have a "blank" RAM core in your "**Hierarchy**" window pane in Project Navigator, as shown in the graphic at the right. When you select the **ram_256x16.xco** "core," you will see "Core Generator" tools in the "**Processes**" window pane of Project Navigator.

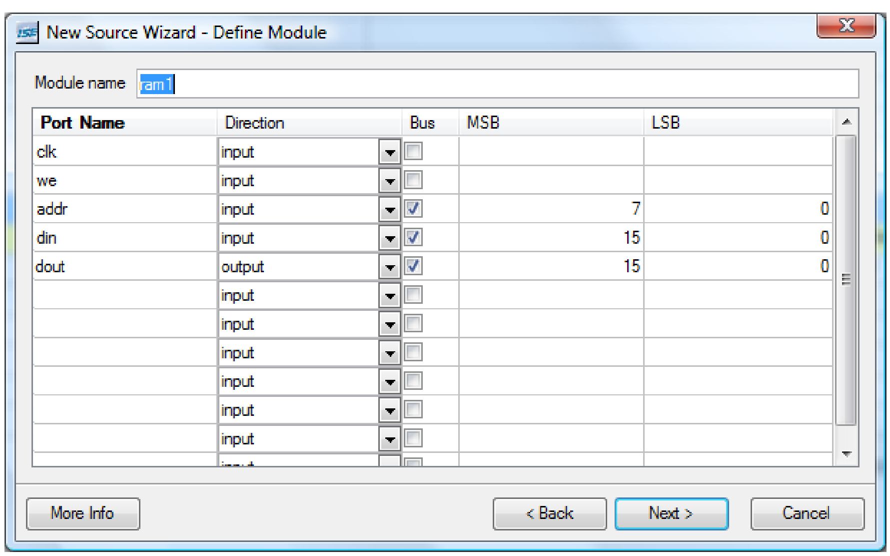




The **ram_256x16.xco** file is a source module for the project. However, it is not yet <u>associated</u> with the **ram1** verilog module. This is done by "instantiating" the ram_256x16 module inside of the **ram1** module using the **Instantiation Template File** created by the CORE Generator (**ram_256x16.veo**). Instantiation template files are files containing code that can be used to instantiate a CORE Generator module into your Verilog design. Verilog instantiation template files have a "**.veo**". The **ram_256x16.veo** file will be in the **ipcore_dir** directory for you current Xilinx Project.

If you have not already done so, create the verilog file **ram1** module. Right click on the "top level" module in the **Hierarchy** window pane and choosing "**New Source**" and "**Verilog Module**" in the dialog windows. Click on the **Next>** button to open the **New Source Wizard** dialog box.

Fill in the **Port Name**, **Direction**, and **MSB** fields as shown in the graphic below. Click on the **Next>** and **Finish** buttons to open the text editor with the **ram1** verilog skeleton in it.



Using Coregen IP (cont.)

Now is the time to copy and paste the **Instantiation Template** section of the **ram_256x16.veo** file (in the **ipcore_dir** directory) into the **ram1.v** source code. Open the file by (1) selecting the **ram_256x16.xco** "core" in the **Hierarchy** pane and (2) clicking on the "**View HDL Instantiation Template**" in the **Processes** pane.

Copy (or cut) according to the comments in the **.veo** file, and then paste into the **ram1.v** source code. **Next, change the instance name and port connections (in parentheses) to your own signal names, as shown below:**

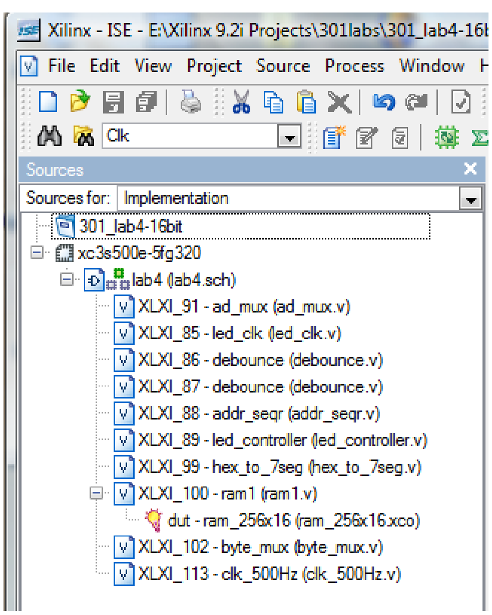
```
module ram1(
    input          clk,
    input          we,
    input    [7:0] addr,
    input   [15:0] din,
    output  [15:0] dout
    );

   //------ Begin Cut here for INSTANTIATION Template ---// //INST_TAG
   ram_256x16 dut (
     .clka(clk),
     .wea(we),
     .addra(addr),    // Bus   [7 : 0]
     .dina(din),      // Bus  [15 : 0]
    .douta(dout));   // Bus  [15 : 0]

   // INST_TAG_END ----- End INSTANTIATION Template --------

endmodule
```

Copy and Pasted from the ram_256x16.veo file, which was created by the CORE Generator.

NOTE: the instance name dut and .clka,, .dina, .addra, .wea, and .douta parameters were changed according to the definitions made for the ram1 module.

Save the **ram1.v** source code by closing the text editor and saving the file.

The Project Navigator will now automatically associate the **ram_256x16.xco** module with the **ram1.v** source module as shown in the Sources window in the graphic below.



Note: as mentioned earlier, the clock input to the **ram1** module is to come from the 100 Mhz board clock.

CECS 301 Team Accountability Reporting Sheet

Team Member #1: ______________________________ **Signature:** ________________________________

Modules/dates you worked on:____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________

Modules/dates you debugged: ____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________

Team Member #2: ______________________________ **Signature:** ________________________________

Modules/dates you worked on:____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________

Modules/dates you debugged: ____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________
____ **Hours:** ________

Percentage of Work Summary: **Team Member#1** __________ **Team Member#2** __________