EEE 498/591: Project Part 1
Serial "Hello World"

August 21, 2016

[20 Points] Due at start of class Tue (Aug 30th)
Summary: This class will make use of a number of different systems. Most of them should be somewhat familiar to you, especially Verilog. In HW1 we will make sure that everyone has some practice writing and executing code using the class tools. This is an opportunity to make sure that you are comfortable with the course. Also, we will use an auto-grader for the course, and this homework will be an opportunity to practice using it. Note that because this is a simple assignment, there are some tasks that have points but no submission. In order to receive all of the "give-me" points, you must score at least a 3 on the sequential multiplier problem.
Pair Programming: This assignment may be completed in pairs. Our expectation is that you will complete the assignment together: discussing concepts, debugging errors, and learning from each-other. You may submit your assignment individually, though you will be evaluated with the same criteria. Only one assignment needs to be submitted for each programming pair. You may select a new programming partner for each part of the project.

1. **[0 Points] Prepare your submission:** Your should create a directory that you will submit through email. We will refer to that directory as "$DIR". It is up to you where you create your directory, but a good place might be " /eee591/ProjPart1"[1] Your directory should contain a file to help us resolve your identity. Your file should be "$DIR/submit.txt" with a line indicating your name and posting id. It should contain your last name (as it appears in blackboard), followed by your first name, following by your posting id. There should be a single line for each of you. For example:

   ```
   Hamm, John, 1234569
   Douglas, Michael, 135711
   ```

2. **[1 Points] Logging in remotely:** The eecad lab is in high demand, and it will often be difficult to find a seat physically in the lab. Instead you should make use of remote access on the machines. Generally there are two methods to do this:

   - On Linux or MAC: You should use ssh with X forwarding. In this case USERNAME is your ASURITE user-name and N is some number (1-40).

[1] You can use "mkdir -p /eee591/ProjPart1" to create all of the directories in one go

```
ssh -YC $USERNAME@eecad$N.eas.asu.edu
```

- On Windows: There are a number of helpful packages like MobaXterm that can provide x-window emulation on your platform. If you are familiar with other approaches on Windows, feel free to help out your classmates with a Piazza post.

3. [**1 Points**] **HelloWorld.pl** In this assignment we will write some very basic Perl code. In the course we will make use of Perl to generate RTL, generate tests, write reports, analyze large files, etc. Perl is often fundamental to doing logic design, architecture, verification, and physical design in industry. Though, don't worry; we will start simple. In this exercise we require you to write a single Perl script that prints the string "Perly Hello World" with a carriage return[2]. Your file should be named "HelloWorld.pl".

   - All Perl files should contain the following headers to make your life easier.

     ```
     use strict ;                    # Use a strict interpretation
     use warnings FATAL=>qw(all);    # Turn warnings into errors
     use diagnostics ;               # Print helpful info, for errors
     ```

   - A print statement in Perl looks like:

     ```
     print STDOUT "WARNING ... Perl is fun\n";
     ```

   - Your Perl script can be run using the command

     ```
     perl HelloWorld.pl
     ```

4. [**1 Points**] **HelloWorld.c** In this assignment we will write some very simple C code. In the course we will make use of C to write benchmarks for our architectures, build verification code, and build some reference models. Note that there is a little more effort to write basic code relative to Perl, so we will often prefer to use Perl for one-off tools. In this exercise we require you to write a single C program that prints the string "C says Hello World" with a carriage return. Your file should be named "HelloWorld.c".

   - You will likely need to do a little work to get your helloWorld program running. You will need to create a file and import the appropriate libraries and write a main function.

     ```
     #include<stdio.h>
     main()
     {
         /* ... */
     }
     ```

   - The stdio library will provide you access to the formatted print statement:

     ```
         printf("It could be worse...\n");
     ```

[2] often referred to as a newline

- Many of you have likely used an IDE for building code. However, in this class we will make use of the command line tools that are actually called by IDE in the background.

```
gcc -c -o HelloWorld.o HelloWorld.c  # Compile into objects
gcc -o HelloWorld HelloWorld.o       # Link the objects
```

- This will have created an executable called "HelloWorld". You can run this executable by simply writing the filename into the terminal.

```
./HelloWorld
```

5. [**1 Points**] **Project Directory** For most assignments in the class you will be provided a starter directory that includes a setup script for the environment, a Makefile for the flow, and starting code for the assignment. While the theme of the projects is generally an incrementally more complex design, we will often provide the solutions from the last assignment as the starting point for the current assignment.

   - Get a copy of the directory:

```
mkdir -p eee498/proj1-work         # Make a directory to work in
cd eee498/proj1-work               # Go to that directory
cp /afs/asu.edu/class/e/e/e/eee333/2016_03_Fall-Brun/Proj-01.tar.gz ./
tar -xzvf Proj-01.tar.gz # Decompress the Tarball
```

   - Unzip the tarball:

```
tar -xzvf assignment1.tar.gz
```

6. [**1 Points**] **HelloWorld.vp** In this course we will be using Genesis2. It is a tool that allows for the Perl elaboration of SystemVerilog. You and your partner should spend some time skimming the user guide. This adds an extra step to the process of building hardware:

   - Generate RTL using Gen2
   - Compile RTL into an executable using vcs compiler
   - Run the RTL simulation
   - Synthesize the RTL using design compiler

   Here we will focus on the first step, RTL generation.

   - In genesis2 your are only allowed to have one module per file. So you should create two modules "HelloWorld.vp" and "top_HelloWorld.vp".

```
////////////////////////////////////////////////////////
// HelloWorld.vp
//;
//; # Any Line that starts with a slash slash semi-colon is perl
//; # So we can follow our good habits from before
```

```
//; use strict ;                    # Use a strict interpretation
//; use warnings FATAL=>qw(all);    # Turn warnings into errors
//; use diagnostics ;               # Print helpful info, for errors
//;
//; print STDOUT "This is a print at rtl generation time\n";
//;
//; # Genesis2 Allows us to create parameters
//; my $bW = parameter( name=>"bitWidth", val=>8, doc=>"Width of input");
//;
//; # Any expression contained in back-ticks is evaluated to a string
//; # and printed in-line ... for example :

// The bitwidth of this module is `$bW`

// We use the gen2 builtin mname to indicate the module name
//  This is because module names are derived from the generator
//  file name, where uniquely generated files are given a unique name.
module `mname` (
          input  logic [`$bW-1`:0] a,
          input  logic [`$bW-1`:0] b,

          input logic clk,
          input logic rst,

          output logic [`$bW`:0] z
                );
// Empty module
endmodule: `mname`

////////////////////////////////////////////////////////
// top_HellowWorld.vp

//; use strict ;
//; use warnings FATAL=>qw(all);
//; use diagnostics ;

// A top module has no inputs or outputs
module `mname` ();

//; #Instantiating a module in gen2 requires an extra
//; # step where we generate the module, then instance it
//; my $hw = generate('HelloWorld', 'my_HelloWorld');
//; # We can querry the value of parameters
//; my $bW = $hw->get_param("bitWidth");
```

```
  logic [‘$bW-1‘:0] a;
  logic [‘$bW-1‘:0] b;
  logic [‘$bW‘:0] z;

// # Then we can instantiate our DUT
    ‘$hw->instantiate‘ (.a(a),.b(b),.z(z));

endmodule: ‘mname‘
```

- You can run genesis2 using the Makefile. Because this is a general flow, you should indicate which module is the focus. In this case we are indicating that HelloWorld should be built. The flow will automatically find a module named "top_HelloWorld.vp" and assume that this is the test bench for HelloWorld. The flow also assumes that the instance name of HelloWorld is "my_HelloWorld".

  ```
  source setup-eecad.cshrc # initialize the environment
  make gen MODULE_NAME=HelloWorld
  ```

- You should edit the files so that the module "HelloWorld" prints the string "Ofer S. says Hello World" with a carriage return.
- You should investigate three new directories to see that Genesis2 is "printing" your RTL:

  ```
  ./genesis_synth # The design files that it expects to use for synthesis
  ./genesis_verif # your test bench files
  ./genesis_work # The perl modules created by the flow
  ```

7. [**1 Points**] **VCS HelloWorld:** In this course we will simulate Verilog designs using Synopsys vcs. This breaks the process of simulating an RTL design into two steps: compilation and running. In the compile step the RTL is translated into an executable binary. In the run step the simulation is completed by executing the binary. This might seem complicated, but the provided Makefile greatly simplifies the process. For this assignment you should extend your initial design so that it prints the following with a carriage return during simulation:

   ```
   [eee498] Agent Hans Handsome Says Hello
   ```

   It might be helpful to review the "initial" procedure block and the task "$display". You can compile and run the simulation with the command:

   ```
   make comp MODULE_NAME=HelloWorld
   make run MODULE_NAME=HelloWorld
   ```

   Note that the Makefile tracks dependencies, so you do not alway need to run the intermediate commands (gen, comp, run). You can always just run the final target (e.g. run). Also, you should expect a good number of lint errors, as our module is empty and top leaves its pins floating. We will resolve most of these issues in the next problem.

8. [**1 Points**] **Waveforms:** In this exercise you should extend your HelloWorld design to include an adder:

```
// Behavioral inference of structural adder
assign z = a + b ;
```

Extend your HelloWorld bench to include some stimulus:

```
//Some stimulus for top_HelloWorld.vp
   initial begin
       for( int i = 0 ; i < 10000 ; i++) begin
       #1000 //1ns delay
          a = $random();
          b = $random();
       end
   end
```

You may wish to extend your bench to include a check on the functionality. Though in this exercise you should include some code to allow for wave capture.

```
//; my $region = "my_HelloWorld";
//Capture the waves in top_HelloWorld.vp
    initial begin
        // if this is a "+wave" run, it must record all signals
        if ( $test$plusargs("wave") ) begin
           $display("%t: Starting Wave Capture",$time);
           //         levels,    instance
           $vcdpluson(0,       `$region` );
          $vcdplusmemon(0,    `$region` );
        end
   end // initial begin
```

Now you can now run the simulation and then view the captured waveforms.

```
make run_wave MODULE_NAME=HelloWorld
make viz MODULE_NAME=HelloWorld
```

In the window that opened you can load the waveform by navigating.

```
File -> Open Database -> vcdplus.vpd
```

You can then use this interface to explore the RTL. If you right click a signal, use “add to waves” to view the waveform for that signal.

9. [**1 Points**] **Synthesis:** We will often run synthesis to evaluate the quality of the physical design. Here we are interested in three direct measurements: power, delay, and area. In this case power is actually the average instantaneous power incorporating switched capacitors, short circuit, and leakage. Delay for clocked designs is a measurement of the designs ability to meet specified clock period. Area is the area consumed by the gates on the silicon. This hello world problem will walk you through running synthesis on your Hello World design and how to analyze the resulting design.

   - **Activity Factor Extraction:** In order to more accurately estimate the power consumption of your design, we will need to calculate the RTL activity factor. The activity factor indicates the number of transitions that occur at a node α per cycle, which when combined with capacitance on the node c, the frequency of the design f, and the design voltage v indicate the total pwoer $P = \alpha c v^2 f$. To incorporate this, we need to add a bit more magic code to your testbench:

```
//; my $region = $self->mname() ;

   logic  triggerStart, triggerStop ;
   logic        foo ;

   initial begin
      foo = 1'b0;
      #1
      //$set_gate_level_monitoring("rtl_on");
      //$set_gate_level_monitoring("mda");
      $set_gate_level_monitoring("rtl_on","sv");
      $set_gate_level_monitoring("rtl_on","mda");
      #1
      $set_toggle_region(`$region`);
      #1
      @( posedge triggerStart );
      $display( "Starting Toggle Collection" );
      $toggle_start();
      #100000000000000000000
      foo = 1'b1;
   end

   initial begin
      #1
      @( posedge triggerStart );
      #1
      @( posedge triggerStop );
      $display( "Stopping Toggle Collection" );
      #1
      $toggle_stop();
      #1
```

```
      $toggle_report( "top.saif" , 1.0e-12 , `$region` );
   end
```

You will need to determine when to trigger the activity factor collection and when to end it by driving triggerStart and triggerStop. For example you might modify your earlier code:

```
// Some stimulus for top_HelloWorld.vp
   initial begin
       triggerStart = 1'b0 ;
       triggerStop = 1'b0 ;
       #10
       triggerStart = 1'b1 ;
       for( int i = 0 ; i < 10000 ; i++) begin
       #1000 //1ns delay
          a = $random();
          b = $random();
       end
       triggerStop = 1'b1 ;
       #10
       $finish();
   end
```

You should now re-run the rtl simulation of your design and take a look at the activity factor file.

```
make clean
make run_wave MODULE_NAME=HelloWorld
cat top.saif | less
```

cat is a program that dumps the text from a file to the terminal. The "—" symbol redirects the standard output to the next command on the line. Less is a program that allows you to scroll up and down with text. In this case you can see that top.saif is a text representation of a data structure indicating the toggles for all of the RTL signals in your design. As a thougt experiment you might consider why triggerStart has a toggle count of 0.

- **Run Synthesis:** You should now run synthesis. It may take a few minutes to complete.

  ```
  make syn MODULE_NAME=HelloWorld COMB_MODE=1
  ```

- **Read the Reports:** You should spend a little time reviewing the various reports:

  ```
  cat syn_HelloWorld.log # A copy of the synthesis logs
  cat phys_HelloWorld/HelloWorld.v # The gate level design
  cat phys_HelloWorld/reports/HelloWorld.mapped.area.rpt # Area report
  cat phys_HelloWorld/reports/HelloWorld.mapped.timing.rpt # Area report
  ```

```
cat phys_HelloWorld/reports/HelloWorld.mapped.power.rpt # Area report
grep Error syn_HelloWorld.log # Find Errors
grep Warning syn_HelloWorld.log # Find Warning
grep LINT syn_HelloWorld.log # Find Lint Issues
```

For example the area in um^2 is reported in the field "Total Area", the power (usually in uW) is reported in the row "Total" under column "Total Power". Timing is a bit more complicated it is most often the difference between data required time and slack. For example $1.00 - 0.0005 = 0.995$ or $1.005GHz$. There should be no errors, a few warnings, and two lint errors related to the clk and rst port.

10. [**12 Points**] **Sequential Multiplier** For the first part of the course project we will implement a sequential multiplier (sometimes referred to as a serial multiplier). A sequential multiplier uses a single adder module to accumulate partial products iteratively. A simple approach would left shift the multiplicand while right shifting the multiplier, using the lsb of multiplier to determine if the current partial product is the shifted multiplicand or zero. You should implemnt your design in "seqMult.vp" with the following interface.

```
// seqMult.vp
//; my $bW = parameter( name=>"bitWidth", val=>16, doc=>"Width of input");
module `mname` (
          input  logic [`$bW-1`:0] a,
          input  logic [`$bW-1`:0] b,
          input  logic             ab_valid,
          output logic             ab_ready,

          input logic              clk,
          input logic              rst,

          output logic             z_valid,
          output logic [`2*$bW`:0] z
                );
// Empty module
endmodule: `mname`
```

As the seqMult is busy, some control flow is need to indicate when a result is available (z_valid) and when new inputs can be provided (ab_ready). It is possible that there are no new inputs for the module, in which case the inputs will not be valid (ab_valid). rst is active low and clk is clock.

- **Pre-design:** You should spend some time thinking about your design. You should draw a block diagram of your design including flip flops, shifters, adders, and state machine module. It is also a very good habit to label the wires in your block diagram with their signal names.

- **Though-experiment:** Once you have a block diagram you should think about how it might operate. You might write out a table of all of the wire values for a number of cycles to get a sense of what you think should happen. This is an opportunity to realize you have missed an important component or missed something critical about the functionality of your design.
- **Write HDL:** If you are happy with your design you should write out the HDL. A good strategy is to focus on implementing the driver for each wire.
- **Write a Testbench:** Once you are happy with the design you should write a test bench. In this case you will need to make sure that you are dealing with control flow correctly and comparing the correct output. Your test bench should measure the average number of cycles that it takes to complete a multiplication. A simple method would be to divide the number of cycles by the count of cycles that the output is valid. Your test bench should be "top_seqMult.vp" and the instance of seqMult should be named "my_seqMult"
- **Debug:** Your design will likely be incorrect in the first pass. You should spend some time using a combination of print statements and waveform viewer to identify how the behavior of your hardware has diverged from its expected behavior.
- **Synthesize:** Once your design is functional, you should run synthesis on it. You should read the logs to make sure that your design completed synthesis correctly. For example, it would be a good idea to discuss the various warnings and errors you see on Piazza to determine which are benign and which are not.

  ```
  make syn MODULE_NAME=seqMult CLK_PERIOD=2.0
  ```

- **Submit:**
  - $DIR/sub1/rtl/seqMult.vp
  - $DIR/sub1/rtl/*.vp – if you created additional verilog files for your hardware design you should include those here so that we can evaluate them as well. Note that you should not include your verification code, we will use our own testbench.
  - $DIR/sub1/char/char.csv which should contain the characterization of your design as a comma delimited file, for example:

    ```
    multRate_cyc, 16.0, # Average Cycles per multiplication
    area_umsq, 10000.0, # Cell Area of your design
    power_mw, .001, # Power of the design
    delay_ns, .5ns, # The minimum achievable timing in the design
    target_ns, .4ns, # The desired clock frequency
    ```

- Rubric: We will run your seqMult.vp and its children using our testbench, Makefile, and syn.tcl. If you found that you needed to make changes to the Makefile or synthesis script, these will not be propogated over. We will procedurally parse the reports to determine the quality of your submission.

Submitted a tarball with all required files	0.6
Genesis2 runs to completion	0.6
VCS compilation runs to completion	0.6
VCS simulation runs to completion	0.6
Synthesis runs to completion	0.6
Design passes 90% verification tests	3.0
Design passes all verification tests	3.0
Synthesis errors in threshold	0.75
Synthesis warnings in threshold	0.75
Synthesis lints in threshold	0.75
Synthesis area in threshold	0.75
Total	12.0

11. [**Bonus - 4 Point**] **Pareto Optimal Design** Designs which are Pareto optimal will recieve four additional bonus points. The two metrics on which you will be evaluated is energy per multiplication (pj/op) and area per multiplication throughput (mm-sq/(op/ns)). You may find that you can significantly increase your energy and area efficiency by reducing the number of partial products accumulated. A simple optimization would skip attempting to accumulate zeros by shifting the multiplier value so that its lsb is always one. A more complicated approach would use Booth encoding to translate strings of ones into two partial products. You can indicate the target clock period you intend for your design using the submitted characterization file.

12. **Submission** Once you have completed your submission directory you should create a "tarball" and email it to the submission email. You should create your tarball with the following command:

    ```
    tar -czvf ProjPart1.tar.gz ProjPart1
    ```

    You should email your tarball to:

    ```
    asu.eee.498.591.fall16@gmail.com
    ```

    If everythign is working well you should recieve a response within a few hours regarding your score. Feel free to submit as many times as you would like. Please note that this will be the first time we are using this email grading system, so we would appreciate your help debugging issues as they come up.