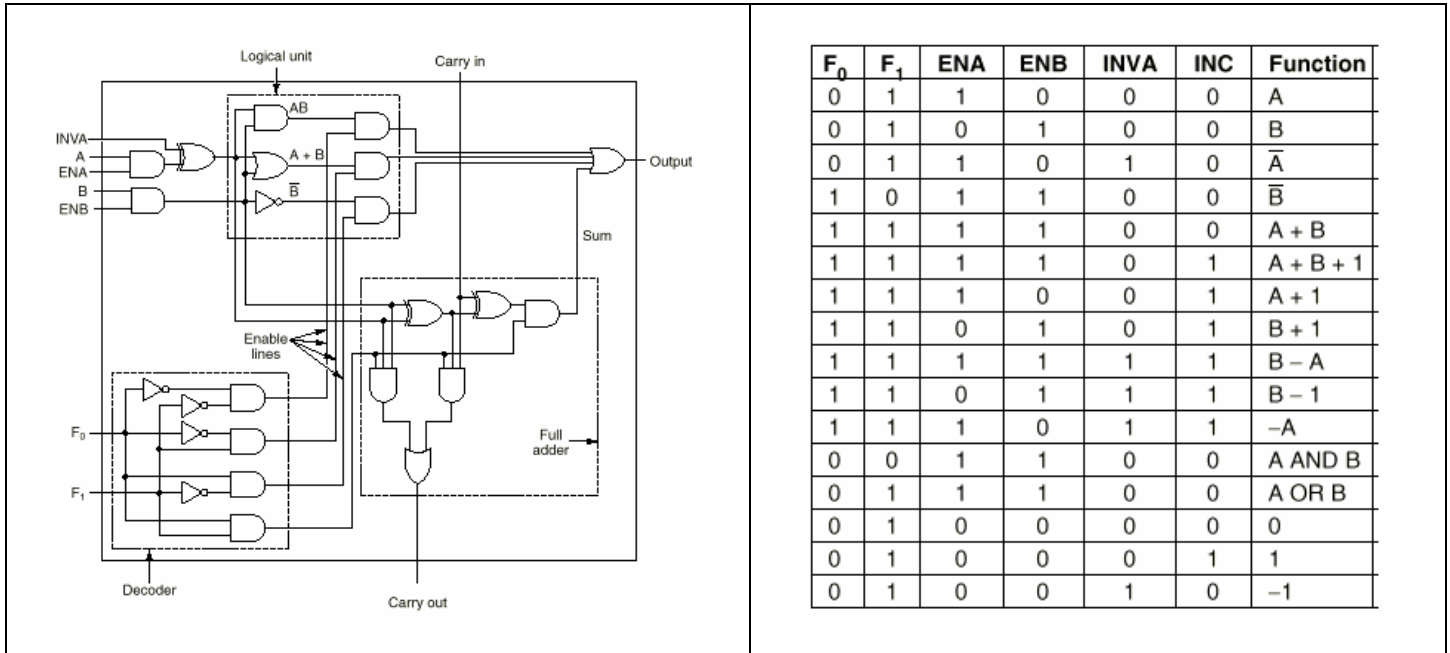


**CSCE 491 – Capstone Computer Engineering Project**  
**Homework Assignments #3 & #4**  
**Creating Extended Models and Test Drivers using ASM Threads**

**HW #3: Extension of the Arithmetic Logic Unit (ALU) Design**

The design, from the Tanenbaum, 4<sup>th</sup> ed., *Computer Architecture*, text, has 4 functions available by use of function code inputs F1, F0, and 2 additional functions available through use of explicit control input signals. The gate-level schematic of the circuit is shown in Figure 1, and the truth table representation of the functionality is shown in Figure 2, as with HWs #1 and 2. The Lecture Notes for *Week #4* give the background on these two assignments.



*Figures 1 & 2. The 1-bit ALU design specification.*

For this assignment, you will take either of the ASM models for the ALU circuit that you created in HWs #1 and #2, and use one of these for extending the capability of the ALU so that it supports the Subtraction operation. We won't modify the basic ALU model, since we want to keep it as a "black box". Instead, we'll write some additional ASM logic, contained in a separate concurrent thread, that will "drive" the original thread to perform the subtract operation using the existing functions of the ALU.

As discussed in the Week #4 Lecture Notes, it may be possible to configure the ALU by its control input signals to perform this subtraction in a single operation, or perhaps in multiple sequenced operations executed on the ALU back-to-back. This is up to you and how you envision using the ALU's basic functions to implement subtraction.

As discussed in the lecture, we will assume Two's Complement arithmetic on unsigned and signed integers. See that set of notes for discussion on 2's Complement, and on the architecture of 2's Comp when using the existing ALU structure and functionality.

We'll follow a similar procedure for creating a design description as with the earlier assignments. This set of steps—along with the variations we'll follow in these assignments—are discussed below:

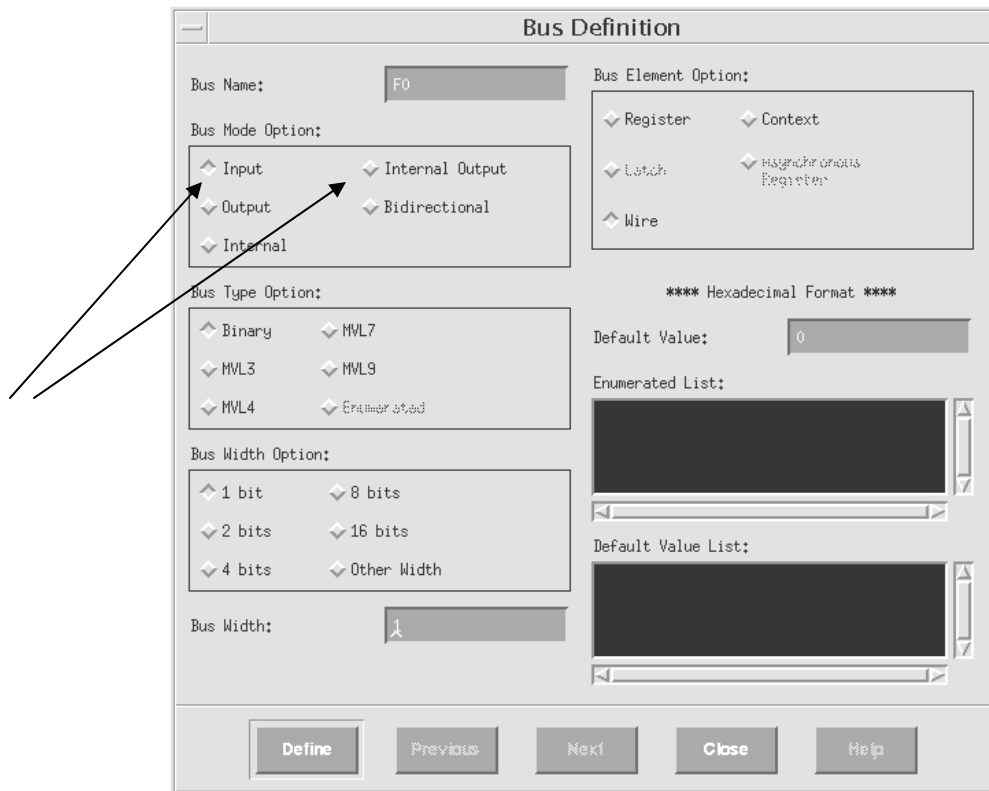


Figure 3. Modifying input buses in the Bus Table signal list.

- (1) We'll start by making one set of modifications to the existing ALU design in order to prepare it to be "enhanced" with a second concurrent thread. Note: you will modify your existing "Input" buses by changing their "Mode" to "Internal". This will allow you to use these signals so that they can be driven by a second ASM thread. This second thread will be the "controller" for the subtraction operation to be added.
- (2) You'll add your new buses to the Bus Table that will be used in the set-up and control of the subtraction operation by this second thread. Since we will basically be using your existing ALU as a "black box", we will make any extensions for supporting subtraction using 2's complement in this second ASM thread. The settings for clocking will be the same. However, in your second thread, you will have to define a "Reset" state for that thread, as you did with the first ASM

thread. You will also define a sequence of states in this thread that will run the test scenarios created for the first ALU, in addition to setting up simulation runs for the subtraction runs.

- (3) You will create the graphical model, as we did for the earlier assignments. Creating this “driver” thread can be done many different ways. I have suggested that you create one additional ASM thread to provide all the control for the subtraction and the various test scenarios. However, you could break it up further, and have one thread that provides all the test cases, and have a third thread that provides the “wrapper” for the ALU for subtraction. We will discuss these options in the lab time.
- (4) Once you have created the model and entered the assignment expressions for the sequence of operations, you will attempt to compile the model, by selecting the “Compile” menu option under the “Tools” menu. Once you compile the model, you should get a dialog box indicating that you have no errors. If you get errors during the compilation, you should go to the state where errors are flagged (with a little “bug” symbol, and click the object while holding down the ‘CTRL’ key (so, CTRL + Left Mouse). This will bring up an explanation of the error. NOTE: make sure the “CAPS LOCK” and the “NUM LOCK” keys are not set. (NOTE: we use different window managers, which have different effects on the windowing of this software package. I can’t keep up with all the specific differences, so we’ll try and resolve these while working in the lab.)
- (5) Once you have compiled the model, you will want to verify its correctness using the *flowHDL* model simulator. In the past, we controlled the simulation by setting stimuli for the design in the Bus Table, and then checked the model’s behavior by looking at the updated bus values in the Bus Table, or signal values in the Waveform Viewer. We will now be using our new ASM thread to provide all the stimulus needed to make the ALU work for our different test runs.

You need to select the “Tools -> Wave Viewer” to open the window for viewing waveforms. You need to select the “Tools -> Simulate” to open the simulator panel. Move the simulator panel to the upper left or right side of the screen.



Figure 4. Simulator cockpit.

To start simulation, you need to set a breakpoint on the reset state of the design. This is done by clicking on this state with the left mouse button. The state will change color (white) indicating that a breakpoint has been set. Then, you will

click the “reset” button on the simulator control panel, as shown in Figure 5. This will start the simulation clock, and pull the reset line, simulating the system reset. Then, you will press the “step” button, advancing the simulation by one clock cycle.

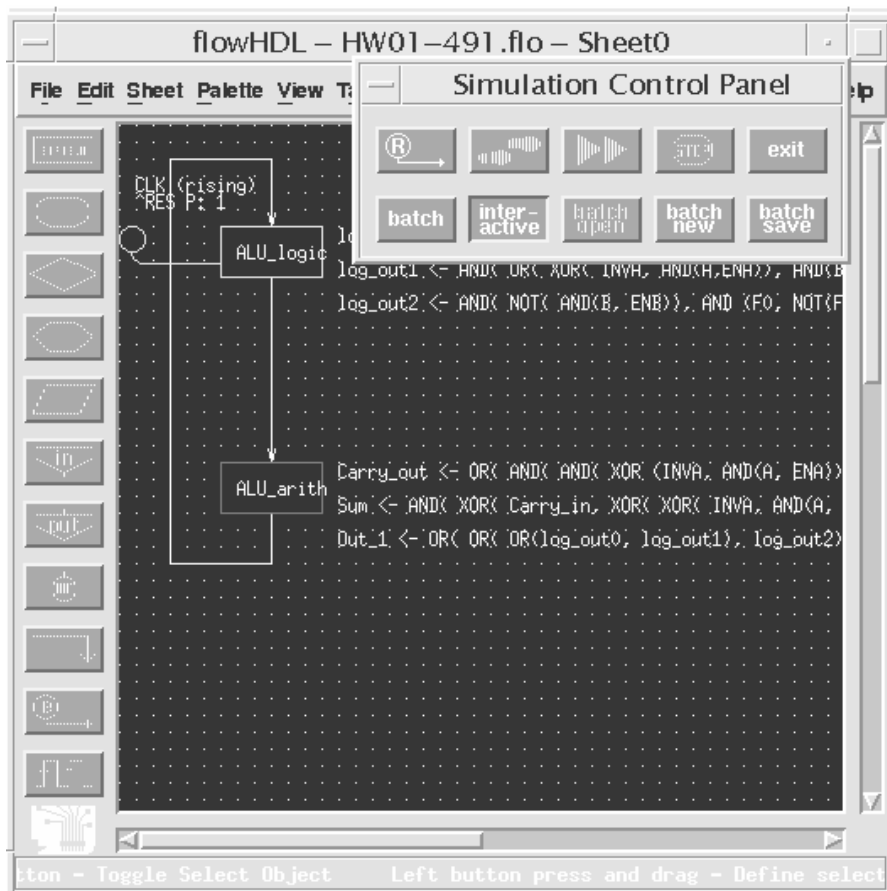


Figure 5. Canvas and Cockpit in Simulator mode.

NOTE: in the lab accompanying HWs #1 and #2, you went through the process of *stimulating* the design by (1) setting a breakpoint, (2) hitting the “Reset”, (3) stepping the simulator once, and, (4) setting the input values to their test values before stepping through the design.

In this assignment, you will have defined the entire “*stimulus*” for the model in your 2<sup>nd</sup> thread, so there is nothing to set in the Bus Table once simulation is started. Because this new thread “marshals” all of the operation sequencing of the ALU into a sequence of states in your new thread, this second thread should provide all of the input values to your ALU for the entire simulation, for all test cases. So, if you are simulating the Subtraction, along with the original 4 test cases from HWs #1 and #2, then this new thread (or threads) will have all of this set of input assignments, in turn, to “*stimulate*” the design model in order that you can check the results in the resultant waveform.

After setting the initial breakpoint on the “Reset” state, you will “step” through the simulation, as before. OR, you can set the initial breakpoint, as before, and set a second breakpoint in the final state of your “driver” thread. At this point, you can print out the single waveform for the entire simulation run. You’ll have to inspect the entire run to check the results of each test scenario, given your input data. When you find errors in the run, you will need to go back and run them individually to isolate the problem in the model. NOTE: we are running one thread to drive the original ALU thread. This means our new thread could have errors that we’ll need to diagnose and debug. Usually, the problems occur because of the cycle timing constraints between the two threads. You’ll have to be careful with this.

In a later assignment, we will discuss the use of “*handshaking*” using *Poll Loops* as a way to decouple concurrent threads from explicit “cycle counting” as a means of coordinating behaviors between concurrently executing threads. For now, you can insert “wait” states in your “driver” thread as needed to synchronize the timing to control the ALU model.

You will see the waveform display advance along with the clock, showing the updated values of your signals, as before. You will be looking for the specific output from the ALU (1) for the subtraction operation test runs, and (2) for the sequence of test runs you simulated by hand in HWks #1 and #2. You will be interested in checking whether the ALU model is giving the correct results (given the operations and operands you specify in your test cases), and also whether it is giving the result in the time frame you are expecting for it. (Remember, the ALU is supposed to be a combinational logic circuit, so we should see the results on the output within one clock cycle. You will want to take the output of the ALU on each execution cycle, and store its results in Registers you declare in the Bus Table. NOTE: you will need to come up with 4 separate test scenarios for the Subtraction operation, in addition to the original 4 that you defined in the previous assignments, and are now also automating.

- (6) Now, once you get the model completed, you’ll want to do the following:
  - (6a) enter your design information, through the “Options -> Design -> Information” menu dialog. Enter a new design name (so spaces or non-alphanumeric, except “\_” underline character, are not permitted), and specify your name and date and design version (which is v1.0). Use the information as shown in Figure 6.
  - (6b) print your design, using the “File -> Print-> Output” menu dialog. The default printer is the one in 1D39. You should select to print the entire design workspace (which will print the Canvas sheets and the Bus Table). You’ll probably want to set the format to “landscape” instead of portrait for this design, and also reduce the size of the image to be printed. This can be done using the “File -> Print -> Setup” dialog. To print the design, you select the “File -> Print -> Output”, which lets you print to the printer (“139” in

1D39) or to a file (which can be printed from the *Ghostscript* application.  
(6c) print your waveform, using the dialog box accessed from the Wave Viewer (the “piece of paper” icon on the tool panel of the Wave Viewer), as we did in the previous assignment. In these assignments, we are running a single, long simulation run. Therefore, we’ll end up with a single waveform file. You’ll want to keep the “HW03.wav” file created for this single, longer simulation run.

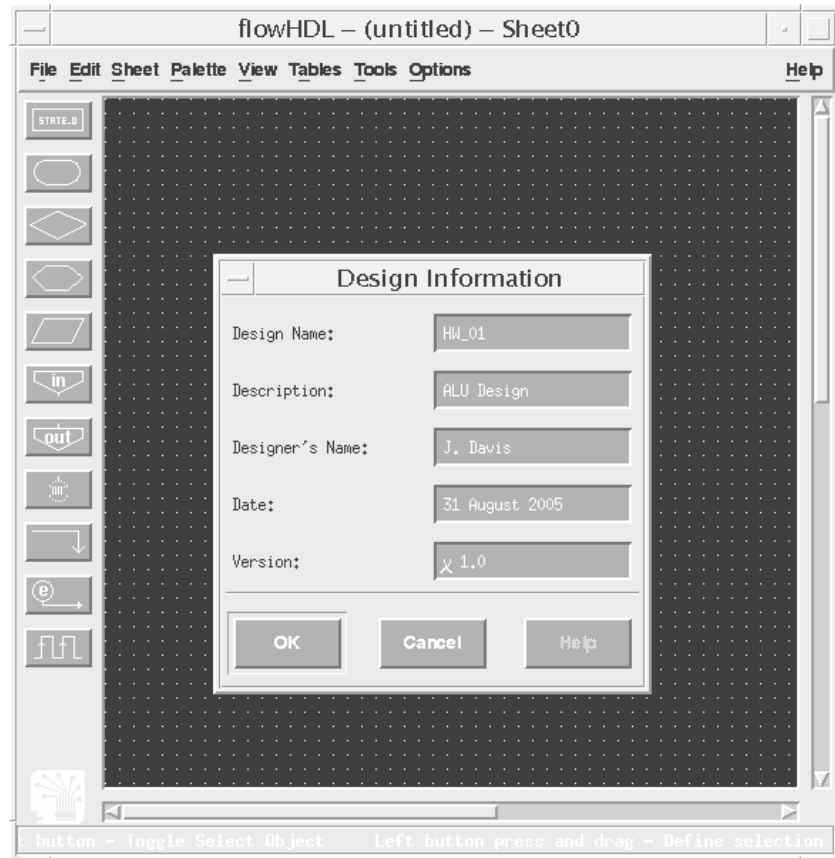


Figure 6. Design Information entry dialog box.

- (7) Finally, you will want to save your model, using the “File -> Save” dialog box. Save the file as **HW03.fl0**, and make sure you are writing it into your “csce491” directory (which you should create). In fact, you will want to periodically save your model after you start creating it in *flowHDL*. Just like most software, it has been known to crash. So, save yourself some aggravation, save your model often during the editing process.
- (8) **Remember:** You also need to preserve the waveform file for the master simulation run and print it in a “compressed” format for inclusion in your assignment submission package. After you simulate each of your 4 test cases, the simulation results will be contained in this one waveform, **HW03.wav**.

Before making your assignment submission, you'll want to check the Assignment Submission Guidelines, located on the course web page. Specifically, you will need to complete a cover page that has the *Effort Distribution* worksheet. I will want you to keep track of the time spent on each activity: (1) analysis & thinking about the problem, (2) editing, and (3) simulation & test planning. The other information we won't use on these assignments. Also, you will need to submit your assignments electronically.

#### **HW #4: The Binary Up/Down Counter**

For this assignment, you will take the specification slides provided in Week's 3 Lecture Notes, and create a model for the counter. Note that we will be using a different style of thread for representing the synchronous sequential FSM machine, using a sequence of states connected with transitions in the ASM notation. The pseudocode for this algorithm is shown in the Week #4 Lecture Notes, along with the block diagram indicating the signal interface for the Counter block.

You'll use the *flowHDL* editor to create the Up/Down Counter model. You'll check the design and compile it, and then you'll simulate it for 4 separate test cases to check for correct functionality and timing of the counter. In the Week #4 Lecture Notes, I give some possible questions to ask about the Counter model that should help in coming up with good test cases.

As you are creating your model (and thinking about it), you'll want to consider the set of "heuristic" questions we discussed in Week #3 Lecture Notes. Specifically, **(1)** when setting up the functions of the Counter, what operations can be done in parallel, and which ones have to be done serial to one another (partially ordered)? **(2)** when managing the counter using the INCRNC() macro-function, at what point can you test the value of the counter for loop termination once you have set the value—remembering that we need a one clock cycle latency when scheduling an assignment operation to a "Register" element.

As you go through your model, and simulate its actions, you may find opportunity to reduce the number of states in the FSM, by moving some operations from a Moore style of execution to a Mealy style. You'll need to balance this opportunity with the correctness of when the operations take place—and on what values they are operating (current or previous). This is the essence of the tradeoff that is part of point 2 above.

You will submit the same set of deliverables as for the other Assignments. You'll need to keep the waveform file, naming it **HW04.wav**. As for HWk #3, we are defining a second thread to contain the entire model stimulus, so that we can run the simulation and obtain all output response in a single waveform.

---

STATEMENT OF PERSONAL RESPONSIBILITY: You can discuss the use of the tool in carrying out this and other assignments, but please do not discuss the solution of the

design problems—unless you are working with your team members—as this will be considered cheating). I need to give everyone opportunity to get comfortable with the tool and the design methods, and this is how we will do it.