CS 2110 – Fall 2010
Homework 4: State Machines

Due by 11:55pm September 21, 2010

Assignment Overview

This assignment deals primarily with the implementation of state machines in a circuit. We expect you to build a state machine that is capable of driving the stack calculator that is found in the file stackcalc.circ. For full credit, you must turn in a modified version of stackcalc.circ that contains your state machine as well as an image of the state transition diagram (the bubbles and arrows diagrams that we drew examples of in lab) for this circuit. Your state diagram must include the following things:

1. A clearly labeled start state.
2. Which state asserts what output.
3. The transitions between states.

(See the turn-in notes for more information about the format of this file.) Start early – this circuit is more complex than in past exercises, so you will need time to figure out how it works.

Assignment

The purpose of this assignment is to introduce you to how a simple calculating device can move data around a "datapath" under the control of a finite state machine. While not as complicated as the state machine that operates the data path of a modern processor, in many ways the way that a calculator works is fairly similar.

The calculator that you will build operates under the following restrictions:

1. It only handles 4 bit numbers.
2. It only performs addition.
3. You can push and pop to manipulate in memory the operands for addition.

Note that when building your state machine, if you don"t have a state diagram with implicit transitions shown, and if you don"t organize the layout of your circuit or u se clear naming schemes for tunnels, it will be difficult for the TAs to help you. It will be easier on you as well to maintain good organization, so plan before you start building!

The Stack Calculator

We have provided you with a circuit that partially implements a stack calculator. Not only is this a stripped down version of a calculator, but currently it doesn"t really do anything without an excessive amount of user interaction. Your task will eventually be to implement a state machine to operate the internal components of the calculator, but before we get to that, it is vital that you understand how the calculator operates.

Remember that a stack is a data structure where information is stored in a First -In Last- Out manner. The calculator has 3 separate functions: PUSH, POP, and ADD. (Note: The addition function will remove a value from the top of the stack; however, when we refer to the "POP" function we are talking about the operation that occurs only when the user presses the POP button.)

The way a user will interact with this calculator is to push some values onto the stack and then press the add button to pop two values off the stack, add them together, and push the result onto the top of the stack. If the user makes a mistake while pushing values, they can press the pop button which will remove the last number they pushed from the stack.

The calculator must perform a set of steps for each operation, and that"s where the state machine enters in. Right now, there are a bunch of input pins all over the place, but the final user interface will feature only a 4bit input pin, 3 buttons corresponding to **push**, **pop** and **add** as well as a **reset** button for clearing everything. (Hint: all those buttons or switches that aren"t part of the final user interface are probably going to be wires with tunnels that are outputs of the state machine when you"re done.)

The Datapath

Being a stack calculator, the datapath obviously has to contain a **stack**. In the provided file, this is a RAM chip with a 16 word storage capacity. Each „word" or „cell" is 4 bits long, and therefore capable of holding a single number upon which we can operate. To keep track of the top of the stack, there is a register called the **stack pointer** (SP). It will always contain the address of the top of the stack. The SP register is connected to the address pins of the stack (the RAM chip). On the data input to the stack there is multiplexer or mux, called **srcmux** which allows selection between two different input sources for the stack: the 4bit input or the adder. The output of the stack goes to two different registers, **A** and **B** which can be write-enabled independently. The outputs of these two registers serve as the inputs to the adder. The output of the adder goes to the previously mentioned srcmux. The stack pointer register can be incremented or decremented on command. In addition, there is a **reset** pushbutton that is connected to the clear pins of any device which has one that allows the calculator to be cleared.

Note that a number of devices on the datapath currently have a switch connected to a write enable (we) pin. These devices will latch or store what is on their input if the write enable signal is asserted (which means has a value of 1 or true) AND the clock pin sees a rising edge (in other words, they are edge-triggered devices).

Operating the Stack Calculator

When you open the circuit in Logisim the very first thing that you have to do is press the RESET pushbutton at the top of the circuit. This should put the machine in a valid state.

Also, notice in the top left corner that there are two ways to operate the clock. There is a mux with an input pin for the select line which can set the clock either to automatic ("clocked") or to manual ("single step") The default state is manual operation. When you have your state machine working, you will want the clock to oscillate automatically. The manual option is there to help you operate the calculator in the beginning and to help you debug your circuit later on.

Functions:	**Inputs to the FSM**:	**Signals to assert**: (Outputs)
• Push • Pop • Add • • •	• Reset button • Clock • Push button Pop button Add button	• INC/DEC (0 = inc, 1 = dec) • SPWE (stack pointer write-enable) • SRCMUX (0 = inputpin, 1 = adder) • STACKWE (stack write-enable) • REGAwe (register A write-enable) REGBwe (register B write-enable)

Operation sequence

There is no specific order in which the PUSH, POP, and ADD functions must be enacted. The user should be able to do any of these operations whenever they want. For example, the user could push ADD immediately after resetting the circuit, and the calculator should just look in the current "top two" cells of the stack and add their contents, whatever they may be, together. It is logical for a user to first push values onto the stack then add them, but the stack calculator should be able to perform any operation at any time. **Note when the user presses the buttons down it should only perform the operation once.**

This requirement, then, lends the solution of having three cycles of multiple states – one cycle for PUSH, one for POP, and one for ADD – that each branch from the same idle, or start, state. The cycles of signals you need to assert to perform each operation are described in the sections below.

After reading these sections, draw a state transition diagram for the entire solution. You may draw this using any means you like (preferably those that do not involve bodily fluids), but make sure it is scanned into a image file (preferably PNG) if on paper or exported to a image file if developed in image editing software on your computer. You will need to submit this diagram for full credit on this assignment. **If you do not submit a state transition diagram, then you will be heavily penalized.** We really want you to plan before you build!

How to perform the push operation:

When you push a number on the stack, the stack pointer will be incremented and then the stack will be write enabled. This will take whatever value is in the input and move it into the stack at the position determined by the value of the stack pointer. To do this manually you will have to perform two steps. The steps are described by what signals should be asserted and what changes have to happen before you pulse the clock (change it from 0 to 1). This push operation can be performed any number of times and in any sequence with
the two other operations.

Any signals that are not asserted should be set to 0

Clock Cycle:	Steps:
1	Set INC/DEC to increment Write enable the STACK POINTER
2	Set SOURCEMUX to keypad Write enable the STACK

How to perform the pop operation:

When you pop a number off the stack, you don"t have to actually clear that piece of memory. All you have to do is decrement the stack pointer so that the next time a number
is pushed onto the stack, it will overwrite the number that you popped. This means that you need one clock cycle of work to accomplish this function. Here are the signals that you must assert:

Any signals that are not asserted should be set to 0.

Clock Cycle:	Steps:
1	Set INC/DEC to decrement Write enable the STACK POINTER

How to perform the add operation:

Now, to perform an addition we actually have to perform three steps. The general idea is that we pop two numbers off the stack, add them together, and push the result back onto the stack. An immediate solution concept would be to decrement the stack pointer twice (to perform the two „pop" operations) and increment it once to perform the „push" operation. However this is somewhat inefficient, in that performing two subtractions and one addition is equivalent to perform a single subtraction. With that in mind, we have the following list of steps to add two numbers:

Any signals that are not asserted should be set to 0.

Clock Cycle:	Steps:
1	Write enable the A register Set INC/DEC to decrement Write enable the STACK POINTER
2	Write enable the B register
3	Set SRCMUX to adder Write enable the STACK

What you have to do

Now that you have experienced firsthand the steps necessary to drive the stack calculator, it is your job to make it easier for the busy TAs to operate. It takes too long to perform all of those steps, so develop a state machine to drive the various components of the datapath and make the stack calculator operate. If your state transition diagram is correct, it should be pretty straightforward to implement.

Remember that after pressing a button, if I continue to hold the button down the operation should still only be performed once. Implementing this will probably require a couple of states in addition to the work cycles described above. If you think you might have a problem with your state diagram, bring it to a TA during office hours (or e -mail it to them) and they will help you.

Package all your state transition logic and your state register as a subcircuit part, with inputs supplied from the buttons on the diagram and outputs that will control the control signals on the diagram. Bring your calculator to life by connecting the input/output pins to the various buttons and pins on the datapath. To do this, you will have to delete many of the input pins that appear in the datapath. (For example, delete the INC/DEC input pin and add a tunnel component with the label "dec" then add another tunnel with the appropriate output pin from your state machine "dec" to connect them.) When you are done, the only buttons that remain should be the push, pop, and add buttons, and they should be the inputs to your state machine. The clock will also be an input to your state machine because you will have to connect it to the register inside. And the reset input should connect to the „0" input of the register you will use inside the state machine.

Submission

When you are finished, submit your modified stackcalc.circ file and an image of your state transition diagram on T-Square.

For the state transition diagram, you may submit either a scanned image of a diagram you drew by hand or an image you generated on the computer. Please submit in PNG or JPG format; if you have any trouble producing an image file from your diagram, let us know and we will help you. **Remember failure to submit this will result in a heavy penalty for this assignment**

And lastly this assignment will be demoed demo times will be up on the wiki soon! Failure to demo this assignment will result in an automatic zero regardless of whether the circuit works or not!