

Lab Assignment 1

UCSD Alarm Clock

Due August 8, 2013, Thursday, 4:00pm

Introduction

The goal of this lab is to get you be familiar with *LogicWorks*. Your task for the lab is to implement an Alarm Clock. This assignment consists of two parts. In the first part, you will implement a basic alarm clock with the functionality described in Part 1. In the second part, you will extend the alarm clock you have built in Part 1 to add a capability of automatically switching between two different alarm times during different weekdays to develop the UCSD Alarm Clock which should be an instant hit with UCSD students whose MW schedule is typically quite different from the one on TuTh.

A library containing several pre-designed components will be provided so that you can build your alarm clock out of these components. It will be helpful for the second part for you to draw a behavioral diagram of how this alarm clock will function. We will provide you the behavioral and structural diagrams for the first part.

Part 1: Basic Alarm Clock

Behavior: At its simplest, an alarm clock has two functionalities – keeping track of the current time, and buzzing when the current time matches the alarm time. While it is possible to think of alarm clocks with preset alarm times, the ensuing market fragmentation makes such a solution uneconomical – one would have to buy a new alarm clock any time they wanted to change their alarm time. This pushes us to add functionality that allows us to program the alarm time of the clock. Furthermore, electronic gadgets are prone to losing power and (unless you invest in nonvolatile memory) lose track of internal storage values in the process. Thus, we should extend our programmability to the current time as well as the alarm time. In any case, as we shall see shortly, setting the current time and setting the alarm time are very similar tasks, with similar implementations in hardware.

Keeping track of current time, the clock's first and most basic functionality, is rather straightforward. At least in the initial formulation, we will content ourselves with keeping track of hours, minutes, and seconds, and leave higher-order signifiers of our temporal existence to more advanced implementations. The relationship between the three can be captured in **Figure 1**, wherein the somewhat eccentric yet subtly elegant babylonian system of 24 hours, 60 minutes and 60 seconds is described. As we will see later on in the implementation, we will spare you the small details by providing you a **ClockDev** component that implements this functionality.

The buzzing part is essentially nothing but an equality check between the minutes and hours of the current time and the alarm time (not too many people seem to care about alarms that buzz on a particular second, so the buzzer will not check for equality of seconds), complete with an **AlarmOn** switch that is used to turn the buzzer on and off, in case the user doesn't want to be rudely awakened on a day of leisure (See switch "S5" in **Figure 2** and **Figure 4**). The setting of alarm time and current time is shown in **Figure 3**. The user will enable the setting of the time with a pair of switches, **SetCurrentTime** and **SetAlarmTime**: If neither switch is on, then the

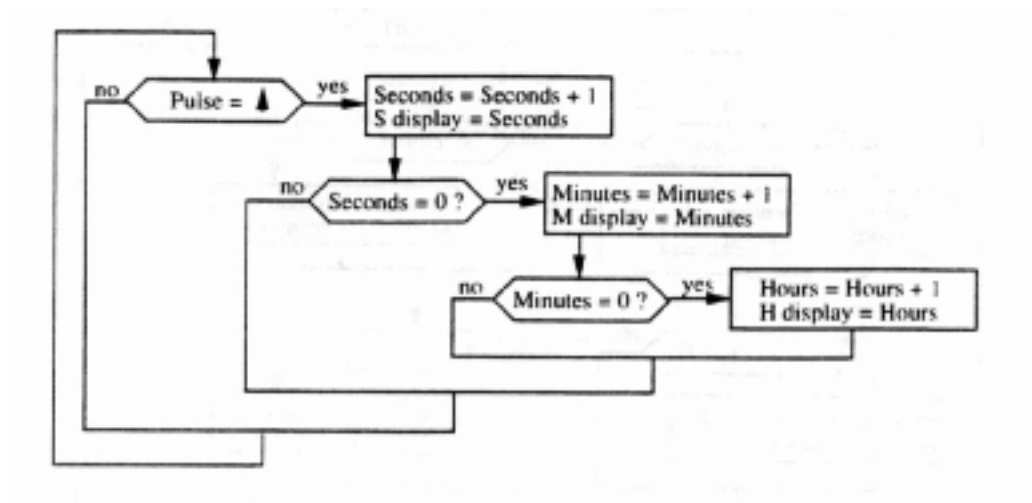


Figure 1: Clock Tick Behavior

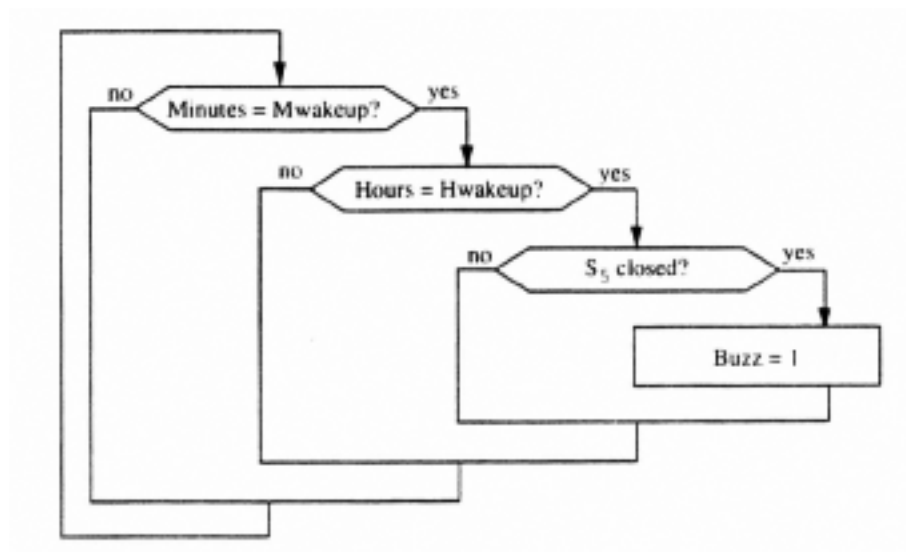


Figure 2: Alarm Buzz Behavior

In addition, the behavioral description in **Figure 2** that illustrates the buzzing functionality can be seen to necessitate an equality check between the current time and alarm time. We will provide you with a **6Comp** circuit that implements this functionality.

We also provide you with a **2x6MUX**. A MUX, or multiplexer, is a circuit component which acts as a “decider” among possible output alternatives. A 2x6 MUX receives 2 inputs, each of which are 6-bits, and outputs one of them based on a third input S, or “Select”. (See the Component Library section of this Lab for a more detailed description of the 2x6MUX.) The reason for including this component is a bit more subtle and perhaps not as evident from the behavioral description that we have provided you. One issue is that while an Alarm Clock normally would display the current time, when you are setting the alarm time, it would be more than desirable to have the clock display the Alarm Time, so that you know how close you are getting to the desired time. So a MUX will be needed to switch between displaying the current time (most of the time) and the alarm time (when the user is setting the alarm time).

A more detailed look at **Figure 3**, shows that switches **SetCurrentTime** and **SetAlarmTime** determine, in combination, the functionality that you are interested in implementing at a particular moment in time. (In **Figure 3**, **SetCurrentTime** and **SetAlarmTime** are modeled by S1 and S2, respectively.) When both **SetCurrentTime** and **SetAlarmTime** are off (we shall use the notation “(S1, S2) = (0, 0)”), the clock should passively tick and display the current time. If (S1, S2) = (1, 0), the user should be able to set the current time (the clock should display the current time). If (S1, S2) = (0, 1), the user should be able to set the alarm time (the clock should display the alarm time, so that the user can see it as it changes.) (For now, (S1, S2) = (1, 1) will have the same functionality as (S1, S2) = (0, 0), but later we will use this fourth condition to give the user control over more functionalities.) The switches are modeled by the **Binary Switch** provided by the component libraries native to LogicWorks, and the selection of functionality can be implemented by using a **decoder**, which can also be found in the LogicWorks libraries.

Now that we took a high level look at the components, it's time to guide you to the interconnect issue in order to complete our structural analysis. One way to think of it is to start at the outputs: What information is the circuit ultimately supposed to compute? The final outputs of the alarm clock are a time (the current time or the alarm time, depending on the user's current needs) and whether or not the buzzer is ringing. For this, you will need the LCD displays and a binary probe. For each of these outputs, we can then ask, where do these displays receive their information from? The LCDs, for example, must output either the current time or the alarm time under different conditions. Thus, the inputs of the LCD must come from a multiplexer, which will choose between alternate inputs based on the system configuration (determined by the switches **SetCurrentTime** and **SetAlarmTime**). And then with the multiplexer, we again ask, where is the information for its inputs coming from? And so on, recursively, until we reach the base case of primary inputs – the clock and the buttons and switches which the user uses to control the clock.

A high-level structural schematic that embodies this discussion is provided for you in **Figure 4**.

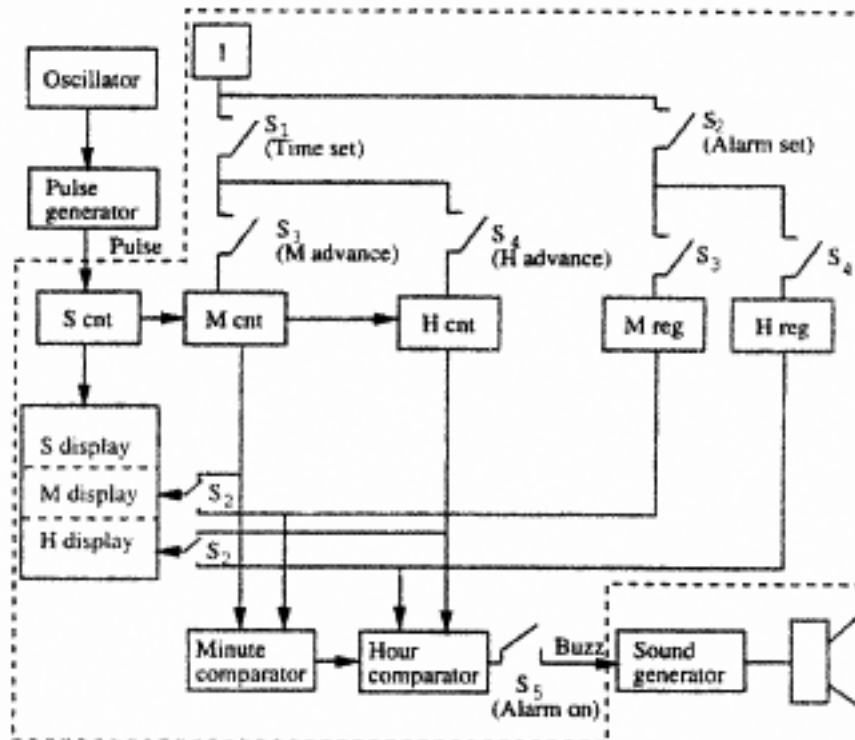


Figure 4: Alarm Clock Structural Schematic

Input/Output: The inputs to this simple alarm clock we're developing are: **Alarm On/Off Switch** (a binary switch), **SetCurrentTime** and **SetAlarmTime** (two binary switches), and the **IncHour** and **IncMinute** buttons. The last two buttons are best implemented with the **SPDT Pushbutton** (provided by LogicWorks native component libraries) connected to constant ground and 5V. (An SPDT Pushbutton has a passive and active input. By default, it outputs the passive input, but as long as the user holds the left mouse click down on the switch, it will output the active input.) The outputs will be the current time on display (which will instead be an alarm time if the alarm is in the process of being set) and the state of the buzzer (a binary probe). No additional inputs or outputs should be necessary for this device, nor should they be included.

Implementation hints:

- You should use a Binary Switch to activate the alarm functionality (Alarm On/Off). (Refer to switch S5 in Figure 2 and Figure 4.)
- You should implement the logic that will enable the user to set the alarm time (in AlarmDev) and the current time (in ClockDev). You should use two switches (Refer to switches S1 and S2 in Figure 3 and Figure 4) and a decoder component, which you can get from the library, to implement the following cases:
 - If S1=1 and S2=0, then the user will set the current time;
 - If S1=0 and S2=1, then the user will set the alarm time;
 - Otherwise, no user modifications to current time or alarm time take place; the clock operates as usual.
- Add a Binary Probe labeled BUZZER that will go high when the alarm is on, and the hours and the minutes of the clock match the alarm setting. If the user turns the alarm off, the BUZZER signal should go down.
- To display the time, you should be using LCD displays (see the **LCD Interface**

component below). Rather than implementing separate displays for **AlarmDev** and **ClockDev**, there should be one single set of LCDs, as in our regular alarm clocks, which only shows the alarm time when we select that switch to set the alarm time. You can use a **MUX** to select between the inputs.

Part 2: Two Alarm Times for the MW versus TuTh schedule

Students at UCSD (and possibly at other colleges that may be less well known) often have different schedules on the set of days MW versus TuTh. This would necessitate a resetting of their alarm times every night before sleep if they were to use the alarm clock we have developed heretofore. Instead a number of 140L students have been clamoring for a chance to expand the alarm clock they have designed heretofore so that it can capture two alarm times, one for their MW schedule and one for the TuTh one. Of course these hardworking 140L students deserve a good sleep over the weekend, so the alarm clock that they are planning to implement should turn itself off automatically for an extended weekend including Friday.

Being a person of ingenuity and opportunity, you've decided to outdo your classmates and actually go ahead to design and implement an alarm clock that delivers such a functionality. By keeping track of the day of the week information, this alarm clock can automatically switch the alarm time during the weekdays according to the schedules of MW versus TuTh, and be quiet and nonintrusive during the extended weekend.

We are going to try to help you a bit to attain this goal of outdoing your classmates by structuring this design task into two subparts. In Part 2a, you will extend the alarm clock you have built in Part 2 by incorporating one additional alarm time. In Part 2b, you are asked to further improve the alarm clock in couple of issues regarding the ability to turn it off for certain sets of days of the week.

2a: Automatically switching between two alarm times

Since the alarm clock needs to provide two different alarm times, of course one additional **AlarmDev** component would be needed to capture the second alarm time. The decision as to which of these two alarm times to utilize for buzzing is automatically controlled by the alarm clock itself. Such a functionality necessitates that the alarm clock be able to keep track of the day of the week information to determine which alarm time is applicable. You would need to use a **DayCounter** component in the design to perform such a task.

In this part of the lab, you will design a **DayCounter** component that will be capable of keeping track of the time. The **DayCounter** will have similar inputs and outputs as the **ClockDev** and **AlarmDev** components. Its inputs will be "CLR", which sets the day to "0" (or Monday); "Set", which allows the user to set the current day when Set = 1; "IncDay", which increments the current day if Set = 1; and "Rollover", which increments the day if the current time becomes 00:00:00. Its outputs will be a 3 bit signal, D0..2, which will give the current day; and a "CurrentAlarm" signal, which indicates which of the two alarms should ring on the current day, according to the following table:

Alarm Schedule Table

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1 st alarm	2 nd alarm	1 st alarm	2 nd alarm	1 st alarm	2 nd alarm	1 st alarm

In order to implement the **DayCounter** component, you will need to use a component called a

Counter-4 UP wo/EN from the LogicWorks native component libraries to keep track of the current day. This component keeps track of a 4-bit integer, and allows it to increment under certain conditions. The counter has two important inputs for the purposes of this lab: CLK and CLR. (You should not need to use the other two sets of inputs, LD and D3..0. It is recommended that you set D3..0 to 0 and LD to 1.) The CLK input increments the integer, and the CLR resets the integer. (Note that while CLR = 1, you will not be able to increment the integer.) Because the counter should reset after day 6 (binary, "110"), and the **Counter-4 UP wo/EN** does not automatically reset that way, you will have to use some glue logic to make the counter reset at the appropriate time. For hints on how to accomplish this, you may look at the inner-workings of the **ClockDev** and **AlarmDev** components to see how the counters for the hours, minutes, and seconds reset on certain conditions.

For a guide to creating custom components in LogicWorks, refer to the LogicWorks tutorial slides on the creation of custom components. Name your custom component **DayCounterA**, and name the library you save it in **Lab1Part2Lib_<your last name here>.clf**.

Now that the alarm clock displays distinct behavior depending on the day of the week, it is important to provide ways to set the day of the week, in case the alarm clock loses power and consequently knowledge of the current day of the week. Therefore, one dedicated **IncDay** button needs to be incorporated into the design, through which the users can directly increment the day of the week information in the **DayCounter** by pushing it. Moreover, a special user mode that enables the functionality of setting days is also needed. The operation of setting days can only be effective when the clock is switched to this mode. Remember that, in the design of Part 1, you are using two switches, S1 and S2, and a 2-to-4 decoder to choose between the set clock (S1=1, S2=0) and set alarm (S1=0, S2=1) modes. The mode S1=1 and S2=1 can perhaps now be utilized this as the control signals to activate the setting of days functionality.

Students may occasionally need to adjust either of the two alarm times. Therefore, an extra switch, **AlarmSelect**, is needed to signify the alarm time to be adjusted. The **IncHour** and **IncMinute** buttons would modify solely the alarm time signified by the **AlarmSelect** switch. *Please note that the choice of the effective alarm time in each day is purely determined by the day of the week information, and is completely independent of the status of the **AlarmSelect** switch (which only controls which alarm time is to be set).*

Since three sets of time information (two alarm times and one current time) need to share a single set of LCD display, you would need two levels of MUXing to select the appropriate one to display in distinct modes (normal mode of displaying current time, setting Alarm One, and setting Alarm Two). For maintaining the correct alarm semantics, one level of MUXing is also needed to propagate the proper alarm time to the Buzzer block. You are asked to design the glue logic that generates the correct control signals for these MUXing operations. (Alternate implementations of this circuitry require a different form of multiplexing, which requires you to select between two buzzer signals. If you find yourself implementing this version of the circuit, you will need a **MUX-2x1**, which we have provided, but you are on your own when it comes to implementing that circuitry.)

Implementation hints:

- The alarm selection (**AlarmSelect**) for setting/adjusting the alarm time can be implemented using a binary switch.
- The **DayCounter** counts from 0 to 6, with 0 representing Monday, and then counts back to 0 after reaching 6, corresponding to Sunday. Using the binary forms of these numbers to encode Monday through Sunday, you should be able to find a simple way to generate

- the control signal for selecting the effective alarm time of each day.
- The output of the **DayCounter** needs to be displayed on an additional **7-Seg Disp I/O** component.

2b: Disabling extended weekend alarm

Once you have finished the first subpart and are in the process of boasting to your friends regarding your great *UCSD alarm clock*, you notice that while sycophants feign utter delight at your work, some of your truer and trusted friends have two suggestions for you to incorporate before you try to get some venture capital and start off your company to produce and sell these *UCSD alarm clocks*. The major one seems to concern the fact that this alarm clock seems to be buzzing on weekends (unless you force it off which has its own problem of possibly forgetting to turn it back on by Monday). You try to convince your friends that this is a feature and not a bug, and wax philosophical about the benefits of constant work round the week, but your entreaties seem to be falling on deaf ears. Certainly, your friends find it utterly incomprehensible as to why your alarm clock would be buzzing on the weekends (on an alternating MoWe & TuTh schedule to boot), and suggest simply that you turn off the alarm clock functionality for the weekends. Some of the more astute of your friends who have wisely selected classes either on MW or TuTh only, notice that your alarm clock is either on for both sets of days or for neither; they recommend that you improve it to enable either set of days to be independently alarmable. While you are fast despairing of the difficulties of the entrepreneurial path and of the possibility of a quick buck, you realize that if only you read a bit more, a guiding hand will try to help you attain the improved alarm clock, thus retaining your hopes of quick riches.

Regarding the functionality of disabling the alarm on extending weekends, the alarm clock obviously needs to be able to detect the Friday, Saturday and Sunday bit patterns from the DayCounter outputs and use these output patterns to disable the buzzing signal. To do this, you will modify the **DayCounter** component you made in Part 2a so that it outputs a “Weekend” signal, signifying that the current day is a weekend and thus the alarm should be disabled. Name this component **DayCounterB**, and save it in the library you created for the **DayCounterA** from Part A named **Lab1Part2Lib_<your last name here>**.

To provide each alarm a dedicated switch to control them separately, the original **AlarmOn** switch needs to be replaced by two switches, **Alarm1On** and **Alarm2On**. This way, the students with only MW or TuTh schedules need to turn on the alarm that matches their course days and would not worry about being bothered on the other days.

Part 3: Implementation Issues

Testing your implementation: You should make sure your implementations work properly. Firstly, you should test that the switches are implemented correctly. Set the switches for adjusting the current time, and then observe that you can increment hours and minutes. Do the same thing for the alarm time and make sure you can increment hours and minutes for the alarm setting. This step also tests whether you have properly connected the **LCD interfaces** to the outputs.

In Part 1, you should set different clock and alarm times and then wait and observe that the alarm turns on when it is supposed to. Pay attention to how long the BUZZER signal is high. The BUZZER should be high for exactly 1 minute and then should go low. Turning the “Alarm On/Off” switch to Off should immediately set the BUZZER to low; you may want to verify this.

In Part 2a, check if you can set each alarm time individually and properly. Adjust the clock hour to check if the DayCounter can update the day information properly when the clock hour counts

to 00:00 hrs. Turn on the alarm and check if the BUZZER beeps at the expected time. Pay attention to the alarm time alternation between days.

In Part 2b, check if you can turn on/off the two alarms separately. Turn on the alarms and check the alarm behavior during the extended weekend to see if the alarm is disabled properly.

LogicWorks Hints: In order to implement the alarm clock, you will need to use some logic components, such as comparators, multiplexers, display units, etc. There are various libraries provided by LogicWorks. Each library contains components relevant to the name of library. For example, the library **Simulation IO** contains HexDisplay, BinarySwitch, BinaryProbe, clock, etc. The library **Simulation Logic** contains adders, counters, buffers, registers, decoders, etc. The **Simulation Gates** library contains AND, OR, NOR, XOR, NOT and other gates. We will additionally be adding components relevant to particular labs into the CSE140L library; please check it throughout the quarter for useful components.

In order to be able to start the simulation, you need to put a clock in your design sheet. In this case, you will need to connect the clock to counters (i.e. the **ClockDev** component) to increment time at every clock tick. Also, be sure that your circuit file contains only a single "clock" component. A single clock can control any number of devices.

Be aware of the fact that even if some lines seem to be connected, they may not be actually connected. One way to check is by moving the components to see whether the lines connecting them move together with the component. Another way to connect pins other than putting wires between them is to assign names to the pins, so that the simulator will assume the pins with the same names are connected.

During your simulation, you can use **binary probes** or **hex display** to monitor the lines you want to observe. Sometimes the value of a line can be "unknown" (X or "grayed digits"). If this is observed on the clock/alarm/counter outputs, clicking on the "Clear" binary probe should resolve the problem.

Part 4: Component Libraries

In this lab, you will have to use components from two libraries, and create a third library which contains your implementation of the **DayCounter** components. We will provide you with a library with pre-constructed components, and you will require components from the libraries that already come with LogicWorks.

Components in Provided Library (Lab1Su13)

LCD Interface: This component will be used to connect your design with a set of 7-segment displays so that it will be easier for you to observe the simulation. You will use one "LCD Interface" component for each set of numbers you want to display (i.e. Hours, Minutes, Seconds).

Inputs:

- 0..5 - Input bus that will carry a number between [0,59].

Outputs:

- a1...g1 - Output bus to be connected to the LSD (Least Significant Digit) of your display;
- a2...g2 - Output bus to be connected to the MSD (Most Significant Digit) of your display.

ClockDev: This is the main component through which you will be able to set the current time.

Inputs:

- CLR – Sets current time to 00:00:00;
- CLK – Input from a pulse generator;
- Set – When set, it will allow user to change the current time;
- SH (Set Hours) – When set, this will increment the hours of the clock by 1 if Set = 1;
- SM (Set Minutes) – When set, this will increment the minutes by 1 if Set = 1.

Outputs:

- H4...H0 – The hours represented by a 5-bit integer. H0 is LSB while H4 is MSB;
- M5...M0 – The minutes represented by a 6-bit integer. M0 is LSB while M5 is MSB;
- S5...S0 – The seconds represented by a 6-bit integer. S0 is LSB while S5 is MSB.

AlarmDev: You will be able to set the alarm time with this component. The inputs and outputs are very similar to the **ClockDev** component, except that the **AlarmDev** does not have the CLK input and the seconds output (S5...S0). CLK is not needed here since we do not want to increment the alarm time after we set it. The seconds output is not necessary since the alarm time is typically set in hours and minutes only. Other input and output pins are exactly the same as the **ClockDev** component.

Inputs:

- CLR – Sets alarm time to 0 hrs and 0 minutes;
- Set – When set, it will allow user to change the alarm time;
- SH (Set Hours) – When set, this will increment the hours of the alarm time by 1 if Set = 1;
- SM (Set Minutes) – When set, this will increment the minutes by 1 if Set = 1.

Outputs:

- H4...H0 – The hours represented by a 5-bit integer. H0 is LSB while H4 is MSB;
- M5...M0 – The minutes represented by a 6-bit integer. M0 is LSB while M5 is MSB.

6Comp: This is a 6-bit comparator that you can use to compare two 6-bit signals.

Inputs:

- A5...A0 – One of the two 6-bit inputs;
- B5...B0 – The other 6-bit input.

Output:

- Z – Outputs a '0' if A and B are not equal, and a '1' if they match exactly.

MUX-2x1: This is a 2-to-1 multiplexer which has 1-bit inputs. You will need this component to select between buzzer signals depending on the day of the week.

Inputs:

- I0: The input which is output when S=0
- I1: The input which is output when S=1
- S: The select signal

Outputs:

- O: The output

MUX-2x6: This is a 2-to-1 multiplexer which has 6-bit inputs. You will need this component to select either the current time or the alarm time to display.

Inputs:

- A5...A0 – One of the two 6-bit inputs;
- B5...B0 – The other 6-bit input;
- S – Select signal. When S=0, A5...A0 will be fed to the outputs, whereas when S=1, B5...B0 will be fed to the outputs.

Output:

- Q5...Q0 – 6-bit output bus.

LogicWorks Native Libraries:

Binary Switch: Toggles its output when the user clicks it. The starting output is usually 0.

SPDT Pushbutton: Outputs the passive input unless the user is holding the left mouse click over the button, in which case it outputs the active output.

Counter-4 UP w/EN: Keeps track of a 4-bit integer value, and allows it to increment in certain conditions.

Inputs:

- CLK – An input which determines when the counter increments. Specifically, if CLK transitions from 0 to 1 (and the CLR input is 0), the current value of Q will increment
- CLR – If CLR = 1, the value of Q will be set to 0, and Q can not increment until CLR = 0
- LD – Load from D. (You shouldn't need this for your implementation.)
- D3..0 – The value to be loaded if LD = 1 and CLK transitions from 0 to 1. (You shouldn't need this for your implementation.)

Outputs:

- Q3...0 – The current value of the integer being kept track of by the counter.
- C0 - "Carry-out", or rollover, which turns to 1 briefly when Q transitions from 1111 to 0000. (You shouldn't need this for your implementation)

Student Implemented Library (Lab1Part2Lib)

DayCounterA: Keeps track of the current day of the week, and allows you to set and reset the current day as needed. This is the component used in Part 2A.

Inputs:

- CLR – Sets the day to "0", or Monday. While CLR = 1, no outside signals can change the value of the current day.
- Set – Allows the user to increment the current day using the "IncDay" input.
- IncDay – If Set = 1, the day will increment once when IncDay transitions from 0 to 1.
- Rollover – Indicates to the DayCounter that the hours have rolled over from 23:59:59 to 00:00:00, in which case the day should increment.

Outputs:

- D2..0 – The current day of the week, in the form of a 3-digit binary number.
- CurrentAlarm – If the day is Monday, Wednesday, Friday, or Sunday, outputs a 0.

Otherwise, outputs a 1.

DayCounterB: Nearly identical to the DayCounterA, except that it has an additional output, “Weekend”, which indicates whether the current day is Friday, Saturday, or Sunday.

Inputs:

- Same as DayCounterA

Outputs:

- D2..0 – The Current day of the week, in the form of a 3-digit binary number.
- CurrentAlarm – If the current day is Monday or Wednesday, outputs a 0. If the current day is Tuesday or Thursday, outputs a 1. Behavior is unspecified on any other day. (It is acceptable that it behaves the same as “CurrentAlarm” for the DayCounterA.)
- Weekend – If the current day is Friday, Saturday, or Sunday, outputs a 1. Otherwise, outputs a 0.

Part 5: Deliverables

- **LogicWorks Files:** Turn in a separate LogicWorks circuit file (.cct) for parts 1, 2a, and 2b (three .cct files in total), as well as a library file (.clf) with your **DayCounterA** and **DayCounterB** components. Name your .cct files **Lab1Part1_<your last name here>.cct**, **Lab1Part2A_<last name>.cct**, and **Lab1Part2B_<last name>.cct**. Electronic submission details are on the moodle page.
- **Report:** Turn in a report that explains how you implemented each part. Specifically you should explain the behavioral design of parts 2A and 2B **and provide behavioral diagrams**. You can submit the report in .doc, .txt .pdf or .html format. Model your diagrams after the behavioral diagrams for the basic alarm clock in Figure 1, Figure 2, and Figure 3. You should also discuss implementation considerations in your report. For example, in your implementation you need to compare the hours of the alarm and the clock. How can you compare these two 5-bit signals using the 6-bit comparator we provide? For the DayCounter components you built in parts 2a and 2b, you should explain the logic you needed to add to the counter from the LogicWorks library that allowed it to reset to Monday after Sunday. Also explain the logic for the “CurrentAlarm” output, as well as the “Weekend” output for part 2b.