DUE DATE: 05 - Dec - 2017

LAB 3:

Alarm Clock Design

Basic Alarm Clock Design
+Buzzer Hold time
+ Two Alarm Clock

NEW YORK UNIVERSITY
ABU DHABI

Lab 3: Alarm Clock Design

Introduction

Your task for the lab is to implement an Alarm Clock and then improve it in couple of ways. This assignment consists of three parts. In the first part, you will implement a basic alarm clock with a slight modification, in that it will keep track of time in 24-hour format instead of the 12-hour format. In the second part, you will upgrade the alarm clock to ensure an appropriate duration for the buzz time. In the third and final part, you will extend the alarm clock you have built in Part 2 to add a capability of automatically switching between two different alarm times during the weekdays. 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 you to draw a behavioral diagram of how this alarm clock will function.

Part 1: Basic Alarm Clock

Behavior: In a typical digital alarm clock, the user can set the current time or set the alarm time. This is accomplished by holding a Time Set or Alarm Set button while pressing a Set Hour or Set Minute button to increment that value by one unit. When the user is setting the alarm, the alarm time is displayed rather than the current time, while in all other cases the current time is displayed. The user also has a switch, which is used to indicate whether the alarm should be active or not. The buzzer sounds when the clock and alarm values match if the alarm switch is set to “On”.

IO: The inputs to this simple alarm clock we’re developing are: Alarm On/Off Switch (a binary switch), Mode Switch (two binary switches), and the Set Hour and Set Minute buttons. The last two buttons are best implemented with the SPDT Pushbutton device connected to constant ground and 5V. 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.

Components: We will provide several components, which you can combine with some simple glue logic to create the alarm clock. The ClockDev component is used to track and output the current time. The AlarmDev component is the equivalent for the current alarm time. You’ll also need Comparators and MUXes, which we will provide, a decoder from the standard Logicworks libraries, and the LCD Interface we will give you to interface to the 7-Seg Disp I/O components for the time output.

Implementation hints:

•You should use a Binary Switch to activate the alarm functionality (Alarm On/Off).

•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 and a decoder component, which you can get from the library, to implement the following cases:

• If S1=0 and S2=0, then the user will set the current time;

• If S1=1 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: Buzzer Hold Time

For the alarm clock you built up in Part 1, unless the user turns the alarm off, the buzzer stays on as long as the clock time and the alarm time match, i.e, for 1 minute duration. For some of us a 1-minute buzz may be too short to ensure that we wake up.

Part 2a: Buzzing Forever

To start you off, we ask you to modify the alarm clock you have implemented in Part 1, so that the buzzer will keep buzzing forever until the user turns it off. This new functionality means that the alarm clock needs to be able to “remember” the on-state of the buzzer. To implement it, we provide you a memory component called a D flip-flop. It has a data input (D), a clock input (CLK) and two asynchronous set & reset inputs (S & R). It also has two output signals, Q & Q’, which are the complements of each other. The Q output always takes on the state of the D input at the rising edge of CLK; in between the rising edges of CLK, Q output retains the last value it latched. The Q output gets set or reset when the D flip-flop S or R inputs are low respectively independent of the clock edges.

You need to use a D flip-flop, together with some simple glue logic, to realize the following functions:

● Whenever the clock time and alarm time match and the alarm is on, the BUZZER signal will stay high forever until the alarm is turned off;

●If the BUZZER signal is high, turning the alarm off will directly turn the BUZZER signal off;

●Once the alarm is turned off, turning it back on will NOT reactivate the BUZZER signal until the alarm time matches again the clock time after an intervening period of mismatch.

Implementation hints:

● Several observations of the alarm clock functionality can be drawn. The buzzer begins to sound at the point when the alarm time and the clock time enter a match situation. The buzzer will be off whenever the Alarm On/Off Switch is off. Try to map these observations to the functionality of the D flip-flop.

● The glue logic can be implemented with no more than one extra logic gate.

Part 2b: One hour buzz scheme

The alarm clock you built up in Part 2a is evidently more effective in accomplishing the task of waking you up. But it can be argued that it is sometimes too effective! If you are not at home while the alarm clock begins to buzz, it will keep buzzing forever, possibly not endearing you to your neighbors.

Therefore, in this part, your task is to further optimize the design, so that the buzzer will keep buzzing only for one hour at most; of course if somebody turns the alarm off during this one-hour period, then the alarm stays shut at that point.

Similar to Part 2a, the alarm clock still needs to "remember" its state once it is turned on, implying the need for a flip-flop in the implementation. But the difference is that this time the onstate should be maintained only for one hour rather than forever even if the Alarm Switch is on. You need to think about how to design the glue logic to implement this "one hour" constraint.

The new version of the alarm clock should work in the following manner:

● Whenever the clock time and alarm time match and the alarm is on, the BUZZER signal will stay high for one hour if nobody turns the alarm off;

● If the BUZZER signal is high, turning the alarm off will directly turn the BUZZER signal off;

● Once the alarm is turned off, turning it back on will NOT reactivate the BUZZER signal until the alarm time matches again the clock time after a period of mismatch.

Implementation hints:

● The buzzer begins to sound at the point when both the hours and minutes of the alarm time and the clock time match. It stops buzzing when the hours differ but the minutes match again (unless of course somebody turns it off in the meantime). Try to map this behavior to the functionality of the D flip-flop using appropriate logic.

●The glue logic can be implemented with no more than one extra logic gate.

Part 3: Two Alarm Times for the SunMonTue versus WedThu schedule

Students often have different schedules on the set of days SunMonTue versus WedThu. 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 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 SunMonTue schedule and one for the WedThu one. Of course these hardworking students deserve a good sleep over the weekend, so the alarm clock that they are planning to implement should turn itself off automatically for weekend. 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 SunMonTue versus WedThu, and be quiet and nonintrusive during the weekend.

We are going to try to help you a bit to attain this goal by structuring this design task in two subparts. In Part 3a, you will extend the alarm clock you have built in Part 2 by incorporating one additional alarm time. In Part 3b, 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.

Part 3a: 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 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. The DayCounter should provide the current day information and update it when the current time rolls over to 00:00hrs.

The output signals of the DayCounter would decide which alarm time to use in each day, in the following way:

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

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 Set Day 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=0, S2=0) and set alarm (S1=1, S2=1)

modes. The mode S1=0,S2=1 and S1=1,S2=0 was left unexploited in your original design as it was declared to be an unused combination; perhaps you can now utilize this combination 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 Set Hour and Set Minute 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.

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 Sunday, and then counts back to 0 after reaching 6, corresponding to Saturday. Using the binary forms of these numbers to encode Sunday through Saturday, 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.

Part 3b: Disabling Weekend alarm

Once you have finished the first subpart and are in the process of boasting to your friends regarding your great 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 alarm clocks. The more 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 Sunday). 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 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 SunMonTue or WedThu 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 and Saturday bit patterns from the DayCounter outputs and use these output patterns to disable the buzzing signal. You are asked to design the glue logic that performs such a task.

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 SunMonTue or WedThu schedules need to turn on the alarm that matches their course days and would not worry about being bothered on the other days.

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 2, you should first check whether the BUZZER stays high for longer than 1 minute. Then adjust the clock time to several other times when the BUZZER is high, and check if the BUZZER is still in the high state. Also try to turn the “Alarm On/Off” back and forth to make sure that the buzzer turns off immediately and does not turn back until the next time clock time and alarm time match.

In Part 3a, 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 00:00hrs. Turn on the alarm and check if the BUZZER beeps at the expected time. Pay attention to the alarm time changes between weekdays.

In Part 3b, check if you can turn on/off the two alarms separately. Turn on the alarms and check the alarm behavior during the weekend to see if the alarm is disabled properly.

Component Library

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 00:00:00;
- 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.

DayCounter: This is the component that keeps track of the day information during the week. It is a modulo-7 counter, which counts from 0 to 6, representing Sunday through Saturday.

Inputs:

- CLR – Sets the day to Sunday;
- INC – The rising edge of this signal triggers the update of the day information by incrementing the counter by 1 (and of course rolls back over to 0 if the value is at 6 when INC's rising edge occurs).

Outputs:

- D2…D0 – The days represented by a 3-bit integer. D0 is LSB while D2 is MSB;

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.

Deliverables

1. LogicWorks Files: Turn in a separate LogicWorks file for each of the five parts (five files in total); please note that Parts 2 & 3 have two subparts each resulting in two separate submissions for each part.
2. Report: Turn in a report that explains how you implemented each part. Specifically you should explain the behavioral design of each part and provide behavioral diagrams. You can submit the report in .doc, .txt .pdf or .html format. 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 4-bit signals using the 6-bit comparator we provide?

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.

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.