DIGITAL CLOCK

ECE271/CSC222 Final Project Report

Elwin Cabrera
May 11, 2016
Problem Specification:

For our CSC222/ECE271 final, we had to design a digital clock. We had to use Quartus and design a control unit and data unit using a block schematic. We had to get the clock to work like a normal clock counting the hours, minutes, and seconds, as well as keep tracking of AM/PM. We also had to encode a reset into the clock which would reset the clock to 12:00.00 after pressing a key.

Requirement Specification:

We used Quartus as our program holding all our information and what we worked on inside of it. To test our program we would run it in Quartus, then plug in our FPGA’s to see if the clock would visibly run correctly. To program the clock we used a number of different operations. In our data unit, we used: one big LPM counter, six 74163 counters, seven 7447 BCD to 7 segment decoders, one toggle flip flop, and glue logic using an, or, and not gates. In our control unit, we used: nine states, transitions between every state, and inputs and outputs for every state.

Design Spec:

I. Data unit:

For the Data Unit we used one custom counter that was an up counter, with a modulus count of 49,999,999. We did this because our clock is 50Mhz and it takes 50,000,000 clock cycles for a second to pass and we want to know when exactly it reached one second not when one second has passed so we took away one clock cycle to the modulus. The custom counter we made has inputs reset, enable, and clock and only one output. The enable input of our custom counter is used for enabling the counter to start counting, and our reset input of the counter is used to tell the counter to reset itself and start counting again and the output of the counter just tells us when it’s a second (it goes high every second). Both inputs of our custom counter is then fed to an output of our control unit so the control unit can tell when the counter should be enabled or reset.

In addition to a custom counter we also used some other counter components found in the Quartus software (the 74163 counter). For each of the seconds and the minute’s section we had these counters having only three inputs and for outputs. The inputs were the enable, clock, and the reset which was an active low so we had to not them in order to make them function how we want it. The enable is for enabling the counter and telling it if we want it to start counting and the reset is for resetting the counter to start counting again from one clock cycle again and all of the counters including the custom counter were connected by the same clock. Both these inputs were fed to the outputs of our control unit so we can manage the enables and resets of the counter individual counters. The four outputs for each counter are connected to a binary coded decimal to seven segment display decoder that displays what binary number the counter is outputting in to the decoder. We used a total of
six of these counters and six of the decoders two for the seconds two for the minutes and two for the hours.

We used a binary coded decimal to seven segment display decoder (7447 decoder in Quartus). The decoder takes four inputs and takes in binary values and displays it in decimal format, we have these outputs going to each segment of a HEX display on the Altera board we tested our design in. We also decided to make one of the HEX to always display a dash line.

For the AM and PM we used a toggle flip flop. This flip flop had only three inputs and one output. The inputs were the enable, clear, and the clock. The enable enables the output which was outputted to a single segment in a HEX display so it can switch from ‘A’ to ‘P’, and the reset is to reset back to its original state which is ‘A’. Because we only need to enable one HEX for the display to switch from A to P we also had to disable the bottom HEX to always be off.

We put this all these components together to make our data path, we also made separate outputs with glue logic to indicate when the outputs were supposed to reset. For example we had the outputs from our 74164 connected to a two input and gate telling the system that its nine seconds and the same thing for 50 seconds we did this for both our minutes and our seconds so we can send that signal to our control unit so the counter to be reset. However, our hours were different since a regular 12 hour format clock resets after 12 back to one, we had to use glue logic with a four input and gate indicating when it was an individual counter reached 2 and the same thing for when it reached 1. Also, since we wanted our clock to reset after 12 we loaded in binary bits of 1 to one counter and 1 and 2 another so we can specify in our control unit that we want those values loaded in when it’s past 12 and then reset every counter except the counter. Each output of our control unit is connected to an input to our data path and every output of our data path is connected to an input of our control unit, we had this
implementation because we wanted the control unit to have control over when to reset and enable the clock.

Below is a picture of our data path implementation:
(Note: all of these components are connected by one clock)
Our initial state for our control unit is our reset state. When we flip the switch, it turns the clock on to start counting. If we turn the switch off then back on, it resets everything, and enables the clock to load in 12 and reset to AM. The reset transitions to the counting state when the switch is turned on, and no equal to a count of one second.

Boolean for our reset state:
\[-\text{Count\_equal\_1\_sec}\]

The counting state works with the LPM counter, enabling the clock to start counting. It also controls everything, telling the program where, when, and under what circumstances to transition. The counting state can transition to any state, except returning to the reset, which takes precedence.

The first state counting transitions to is Secs09. Secs09 is the ones place for seconds. The output for counting to go to Secs09, is when the counter is equal to one second, but not equal to ten. When in this state, it enables second_one, the ones place for seconds, to count up and resets the count. After one clock cycle, it transitions back to counting.

Boolean for our Secs09 state:
\[\text{Count\_equal\_1\_sec} \& \neg \text{equals\_09\_secs}\]

Secs50 is the tens place for seconds. The output required to transition here is when the count is equal to nine seconds in the one place and one second resulting in ten seconds. Here, the ones place for seconds is reset back to zero, as well as the count, and it enables second_tens, the tens place for seconds, to count up one. After one clock cycle, it transitions back to counting.

Boolean for Secs 50 state:
\[\text{Count\_equal\_1\_sec} \& \text{equals\_09\_secs} \& \neg \text{equals\_50\_secs}\]

Mins09 is the ones place for minutes. The output for counting to go to Mins09 is when the clock is equal to fifty seconds, nine seconds, and a count of one second, but not equal to ten minutes. In this state, both the ones and tens place for seconds are reset, as well as the count. In this state it enables the minutes_one, the ones places for minutes, to count up one. After one clock cycle, it transitions back to
counting.

Boolean for Mins09 state:

\[
\text{Count} \text{\_equal\_1\_sec \& \text{equals\_09\_secs} \& \text{equals\_50\_secs} \& \text{\neg\text{equals\_09\_mins}}}
\]

The next state counting transitions to is Mins50. It transitions when the clock is equal to nine minutes, fifty seconds, nine seconds and a count of one second, but not equal to one hour. In this state, the ones place for minutes is reset, both the tens and ones place for seconds are reset, and the count is reset. Minutes_ten, the tens place for minutes, is enabled to count up one. After one clock cycle, it transitions back to counting.

Boolean for our Mins50 state:

\[
\text{Count} \text{\_equal\_1\_sec \& \text{equals\_09\_secs} \& \text{equals\_50\_secs} \& \text{\neg\text{equals\_09\_mins}} \& \text{\neg\text{equals\_50\_mins}}}
\]

Hours09 is the ones place for hours. The output for counting to go to Hours09, is when the counter is equal to fifty minutes, nine minutes, fifty seconds, nine seconds, and a count of one second, equaling one hour, but not equal to ten hours. When in this state, both the ones and tens place for minutes are reset, both the ones and tens place for seconds are reset. The hours_one is enabled, the ones place for hours, counting up one. After one clock cycle, it transitions back to counting.

Boolean for our Hours09 state:

\[
\text{Count} \text{\_equal\_1\_sec \& \text{equals\_09\_secs} \& \text{equals\_50\_secs} \& \text{\neg\text{equals\_09\_mins}} \& \text{\neg\text{equals\_50\_mins}} \& \text{\neg\text{equals\_09\_hrs}} \& \text{\neg\text{equals\_12\_hrs}}}
\]

The output for counting going to go to Hours10, is when the counter is equal to nine hours, fifty minutes, nine minutes, fifty seconds, nine seconds, and a count of one second, equaling ten hours, but not equal to twelve hours. When in this state, the ones place in hours is reset, both the ones and tens place for minutes are reset, both the ones and tens place for seconds are reset. The hours_ten is enabled, the tens place for hours, counting up one. After one clock cycle, it transitions back to counting.

Boolean for our Hours10 state:

\[
\text{Count} \text{\_equal\_1\_sec \& \text{equals\_09\_secs} \& \text{equals\_50\_secs} \& \text{\neg\text{equals\_09\_mins}} \& \text{\neg\text{equals\_50\_mins}} \& \text{\neg\text{equals\_09\_hrs}} \& \text{\neg\text{equals\_10\_hrs}} \& \text{\neg\text{equals\_12\_hrs}}}
\]

Our last state is Hours12. Hours12 is when hours is equal to twelve. The output for counting to go to Hours12, is when the counter is equal to twelve minutes, fifty minutes, nine minutes, fifty seconds, nine seconds, and a count of one second, equaling thirteen hours. When in this state, the ones place for hours is reset, both the ones and tens place for minutes are reset, both the ones and tens place for seconds are reset. It then resets back down to one hour with zero minutes and seconds, loading a one into the hours place. After one clock cycle, it transitions back to counting.

Boolean for our Hours12 state:

\[
\text{Count} \text{\_equal\_1\_sec \& \text{equals\_09\_secs} \& \text{equals\_50\_secs} \& \text{\neg\text{equals\_09\_mins}} \& \text{\neg\text{equals\_50\_mins}} \& \text{\neg\text{equals\_09\_hrs}} \& \text{\neg\text{equals\_10\_hrs}} \& \text{\neg\text{equals\_12\_hrs}}}
\]

Below is a picture of our input table
We decided to use counter components found in Quartus, because they were easier to implement. The hardest part to figure out for us was to get the clock to reset back down to 1:00.00 when the count reached 12:59.59. We overcame this problem by employing a load to our hours counter and setting it to load to 1 hour when it reached 12:59:59. Working on designing a clock has been an interesting, fun, learning experience. While it has been difficult, we are glad that we got to go through the experience and see our work in action. We both put a lot of work into the entirety of the project. For the most part we both worked on everything together.