EE 5143 – Course Project – Dr Rahman

The project described in this document is based on implementing a 12-hour clock, hosted in a Cyclone IV FPGA. This is a more advanced undertaking than all the labs encountered so far in this course. However, it is mostly based on concepts introduced in the lab assignments. Even though this is a somewhat more advanced project than all the lab projects, it still uses only a very small fraction of resources inside a Cyclone IV FPGA. This should give you some appreciation of how much uncommitted logic is packed inside even a relatively low-end FPGA these days. You can use such an FPGA to either build one very complex system or a set of completely separate systems so that almost all of the LEs and pins on the FPGA are utilized.

A complete block schematic diagram of a 12-hour clock is given on the next page. You will need to study it carefully. Each block requires writing its VHDL code, compiling it and generating a block diagram. Once all the blocks have been constructed, a top_level can be implemented to wire up all the component blocks. This is then compiled, pin assignments made and a final compilation performed. For the sake of this project, connect the 50 MHz input to FPGA pin R8, the second_blink output to any of the eight LEDs while all the other outputs ca be assigned to arbitrary pins. Either a .sof or a .jic file can then be downloaded to the FPGA. Note that this clock is not adjustable – it always starts at 12:00 and advances from there (as they say – even a broken clock shows the correct time twice a day). Also note that as the DE0-Nano board does not have an LCD or 7-segment display so it is not possible to verify the functionality of this system completely using this board. However, proper compilation will show that all code has been correctly written (at least syntactically and semantically even if not functionally). No VHDL code is provided. You will need to write all the code yourself. To qualify for marks you will need to show screenshots of successfully compiled code for the various blocks. Writing and compiling code for the top four blocks is essential and 80% marks can be obtained if this is done. To get the other 20% marks you need to code the other four blocks as well. You will get partial marks from the reserved 20% if you complete some of the bonus part.

Here is a description of the complete system. Note that a PLL has not been used but could be used to reduce the 50 MHz clock's frequency. It is up to you to decide whether you want to get the 1 Hz output (second signal) using a PLL or using code. Most professional engineers will do this using a PLL because it saves on the logic resources of the FPGA (though by a tiny bit). The top_level schematic diagram of the system is given on the next page. Here is a description of how it works. This should be read with reference to that diagram.

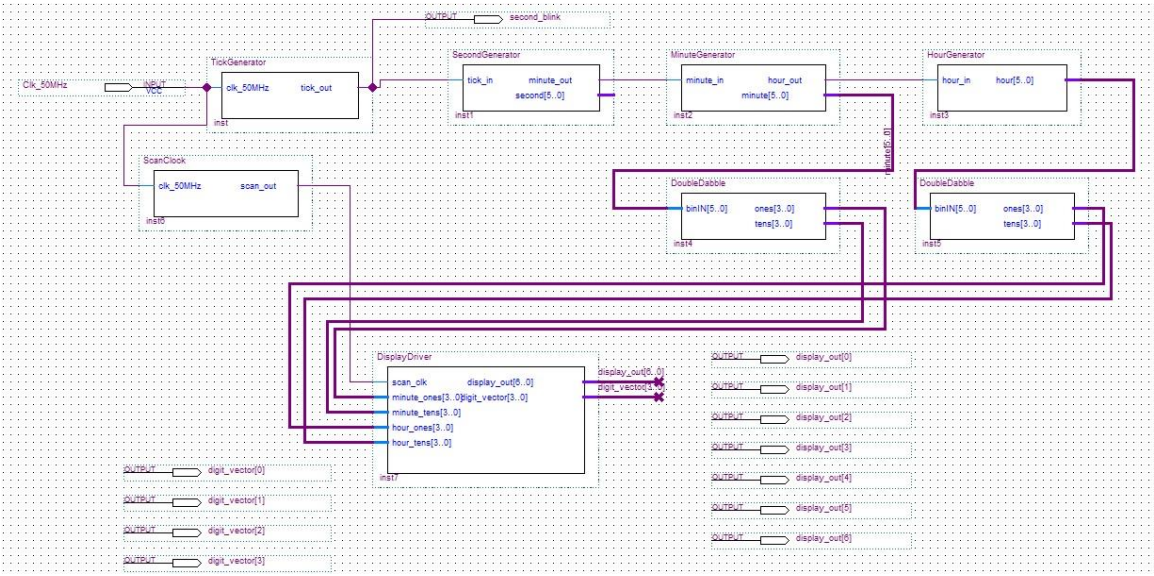
The TickGenerator takes in the 50 MHz clock and generates a 1 Hz output clock – essentially generating second ticks. Its output can drive a second-by-second blinking LED. Additionally, the output tick_out is also sent to the SecondGenerator block.

The SecondGenerator block generates the second count i.e; it increments an internal variable once every second from 0 to 59 and then rolls back to 0. That (std_logic_vector) variable can be assigned to a (std_logic_vector) signal second[5..0] which comes out of the block. Note that this is a 6-bit std_logic_vector value because counts up to 59 can be accommodated in 6 binary digits (2^6 = 64). As most clocks do not display second counts so this value is not utilized any further here. This block also generates a pulse every time the second counter rolls over i.e. every 1

minute. This comes out as the std_logic signal minute_out which becomes the input minute_in for the next block.

The MinuteGenerator block is very similar to SecondGenerator. It generates the minute count i.e; it increments an internal variable once every minute from 0 to 59 and then rolls back to 0. That variable can be assigned to a signal Minute[5..0] which comes out of the block. Note that this also is a 6-bit std_logic_vector value. This block also generates a pulse every time the minute counter rolls over i.e. every 1 hour. This comes out as the std_logic signal hour_out which becomes the input hour_in for the next block.

Again, the HourGenerator block is very similar to SecondGenerator and MinuteGenerator. It generates the hour count i.e; it increments an internal variable once every hour until it gets to 12 and then rolls back to 1. Note that it will start from 12 and on the next hour tick advance to 1 etc. That variable can be assigned to a signal hour[5..0] which comes out of the block. Note that this also is a 6-bit std_logic_vector value, just to maintain uniformity.



There are two instances of the DoubleDabble block. This block takes in the minute or hour count as a binary number and generates two BCD signals: one holding the one's place digit and the other holding the ten's place digit. Complete code for the double dabble algorithm is provided in the Wikipedia article on Double Dabble. This needs to be appropriately modified. Note that doubledabble will produce (output) a BCD value as two digits (representing decimal values), each requiring 4 bits to hold i.e. a total of 8 bits. The first four bits (3 downto 0) can be assigned to ones[3..0] and the last four bits (7 downto 4) can be assigned to tens[3..0].

The DisplayDriver block takes all the four decimal digits (minute one's place, minute ten's place, hour one's place and hour ten's place), coded as BCD values of 4 bits each, and displays them on four digits of a four-figure 7-segment LED display. Most such displays are multiplexed displays i.e. they take information about only one of the four digits as well as four single bit values that determine which of the four digits will actually glow. In this way, one sends digit values one by one scanning from one end to the other (hour unit's place, hour ten's place, minute unit's place and minute ten's place). If this is done quickly enough, then due to persistence of vision we see

all four digits glowing steadily. Thus, in this block you need to output each digit's value in turn and activate its respective digit_vector bit by making it a logic '1' while all others are kept at logic '0'. The digit's value is sent as a 7-bit vector to drive the seven segments of a 7-segment display. This block also takes a 500 Hz scan clock from the ScanClock block which simply generates a 500 Hz clock. This frequency is not critical. It can be 400 Hz, 600 Hz etc. and there will be no visible difference. Its value needs to be a few hundred Hertz in order to scan quickly so that the display does not seem to flicker. A good thing to do is to display each digit for 4 or 5 periods of the scan clock and then move to displaying the next digit. At each of the four steps issue a value for digit_val as "1000", "0100" etc. to select which digit is activated.

Note that you can verify the functioning of the blocks in the top row by sending their single bit pulse signals to an LED to make sure that their periodicities are indeed 1 second, 1 minute and 1 hour. Those who are more adventurous could try their hand at building their own seven-segment display hardware, as described in the display documentation in the project folder. It can then be interfaced to the FPGA. This is not required and will not get you any additional marks, but it is recommended.

Now it is up to you to get going with this project. I can answer minor queries about this project but will not provide any detailed guidance so as to let you have a go at it yourself. Please submit it by April 22 i.e. about a month from today. I'll provide code for all blocks after the project has been graded. Note that coding problems usually do not have unique solutions so your code does not need to be the same as my code. It only needs to accomplish correctly what each block is supposed to do.