Assignment: DIGITAL CLOCK

Name: Arnav Tuli
Entry Number: 2019CS10424

My Design: CONSTRAINTS

- The design has to work with the following inputs and outputs provided by the BASYS3 FPGA board.
    - 4 seven-segment displays, each one can display a digit, optionally with a decimal point.
    - 5 push-button switches.
    - A 10 MHz clock signal.
- The clock should display the time of the day in hours, minutes and seconds.
- There should be a provision for the user to set the time to a desired value.
- The clock should follow a 24-hour format, i.e., the time should range from 00:00:00 hr:min:sec to 23:59:59 hr:min:sec.

My Design: SPECIFICATIONS: INPUT/OUTPUT

I have designed a digital clock that uses the following inputs and outputs provided by BASYS3 FPGA board.

- **4 seven-segment displays**, each one displays a digit/symbol. However, I am **not** going to use the optional decimal point that is provided in any of the display boards.
- **4 out of the 5** available push-button switches.
- A **10 MHz** clock signal.

My Design: SPECIFICATIONS: FSMs

Before I explain my major design decisions and how they ensure the proper functionality of the clock, I will describe the different FSMs (finite state machines) that my design uses:

Button States: Buttons play a very important role in my design as it is the only way that the user can interact with the **clock**. So, knowing about the states of the button is crucial in understanding how the different changes are brought about in the **clock**. In my design a button can have the following **four** states:

- **off**: If a button is in the **off** state, then it means that it is currently not pressed by the user. However, if a button is not pressed by the user, then it does not necessarily mean that the button is in **off** state. The difference will be made clear when I describe the **invalid** state below.
- **On1**: If a button is in the **On1** state, then it means that it is currently pressed by the user. However, if a button is pressed by the user, then it does not necessarily mean that the button is in **On1** state. The difference will be made clear when I describe the **pressed** and **invalid** state below.
- **pressed**: If a button is in the **pressed** state, then it means that it has been kept pressed by the user for the past **0.5 s** (half of a second), and is also currently pressed. This way, it becomes clear that if a button is pressed, then it may not be in the **On1** state, rather it may be in the **pressed** state depending on how long it has been kept pressed. However, the button may be

pressed for a long time, but it still may not be in the **pressed** state. The difference will be made clear when I describe the **invalid** state below.

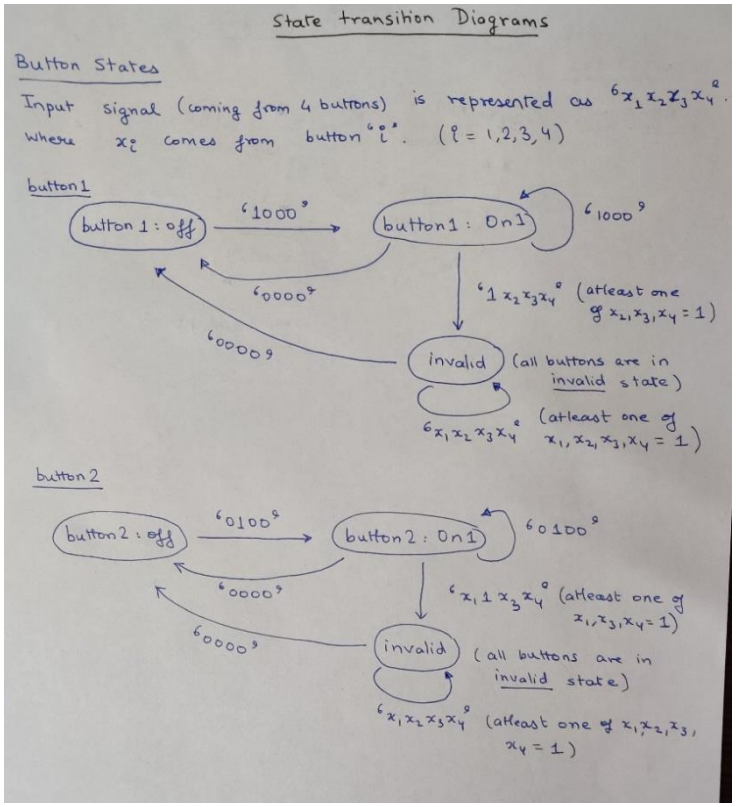
- **invalid**: Suppose a button is in **On1** or **pressed** state. Now, if the user presses any other button (apart from the one that is already pressed by the user), then **all** the buttons go into the **invalid** state, and they **remain** in this state till the user releases **all** of the push-button switches available on the board. Hence, it becomes clear why a button may not be in **off** state even if it is not pressed by the user, as some other two buttons may be pressed, due to which **all** of them go into the **invalid** state. Similar reasoning explains why a button may not be in **On1** or **pressed** state even though it is pressed/kept pressed by the user. It is straight-forward that a state like **invalid** is crucial for my design, as it allows me to resolve conflicts when two or more button-signals are high (or 1) at the same instant. The way this resolution takes place is described later below.

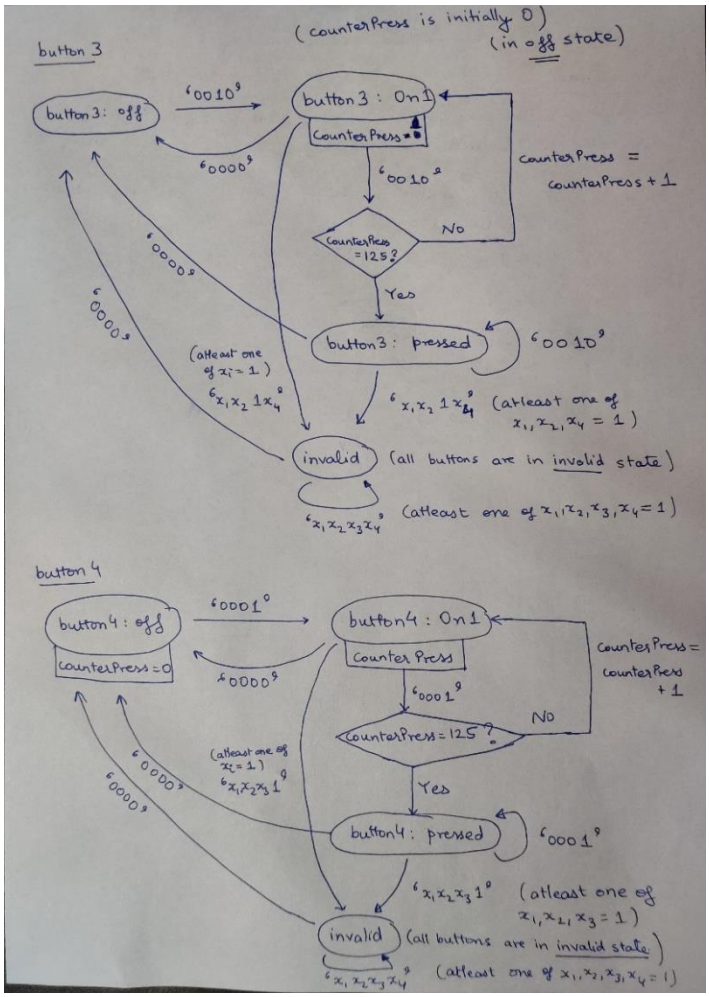
Digit States: As button states control the states of the input provided by the user, digit states control the **state of the output**, i.e., the 4 seven-segment display boards. Since, each display board shares a common cathode for each segment (from A to G), it is not possible to display all the four symbols simultaneously. Therefore, I need to display each symbol for a fraction of duration and then display the next symbol in line. This refresh needs to be done at a suitably high frequency so that the human eye always sees all the four symbols together (persistence of vision). A common **refresh rate** that can be used is **62.5 Hz**, or a refresh every **16 ms**. As each symbol is displayed for an equal time duration, the cathode signals (as well as anode signals) need to be changed every **4 ms** (= 16 / 4). This change in the anode/cathode signals is brought about by the **Digit States**. It is not surprising that there are **four** digit-states in my design:

- **d0**: The right-most seven-segment board is activated **iff** the current digit state is **d0**.
- **d1**: Second seven-segment board from the right is activated **iff** the current digit state is **d1**.
- **d2**: Second seven-segment board from the left is activated **iff** the current digit state is **d2**.
- **d3**: The left-most seven-segment board is activated **iff** the current digit state is **d3**.

Display Modes/States: Since, I only have 4 seven-segment display boards, I cannot possibly display hours, minutes and seconds simultaneously because that will require 6 display boards. Hence, I need to provide the user with different display modes which in turn allows the user to gather all the relevant information. Also, as per the design constraint, I need to allow the user to change the time according to their need. Therefore, there needs to be some display states that allow users to change the time. In my design, I have used **six** display modes/states:

- **hourMin**: hour:min (hh:mm) is displayed on BASYS3 board **iff** display mode is **hourMin**. This mode does not facilitate changing of time by the user.
- **minSec**: min:sec (mm:ss) is displayed on BASYS3 board **iff** display mode is **minSec**. This mode does not facilitate changing of time by the user.
- **hourSec**: hour:sec (hh:ss) is displayed on BASYS3 board **iff** display mode is **hourSec**. This mode does not facilitate changing of time by the user.
- **setH**: H:hour (H:hh) is displayed on BASYS3 **iff** display mode is **setH**. This mode facilitates changing of the hour count only (increment/decrement).
- **setM**: M:min (M:mm) is displayed on BASYS3 **iff** display mode is **setM**. This mode facilitates changing of the minute count only (increment/decrement).
- **setS**: S:sec (S:ss) is displayed on BASYS3 board **iff** display mode is **setS**. This mode facilitates changing of the second count only (increment/decrement).





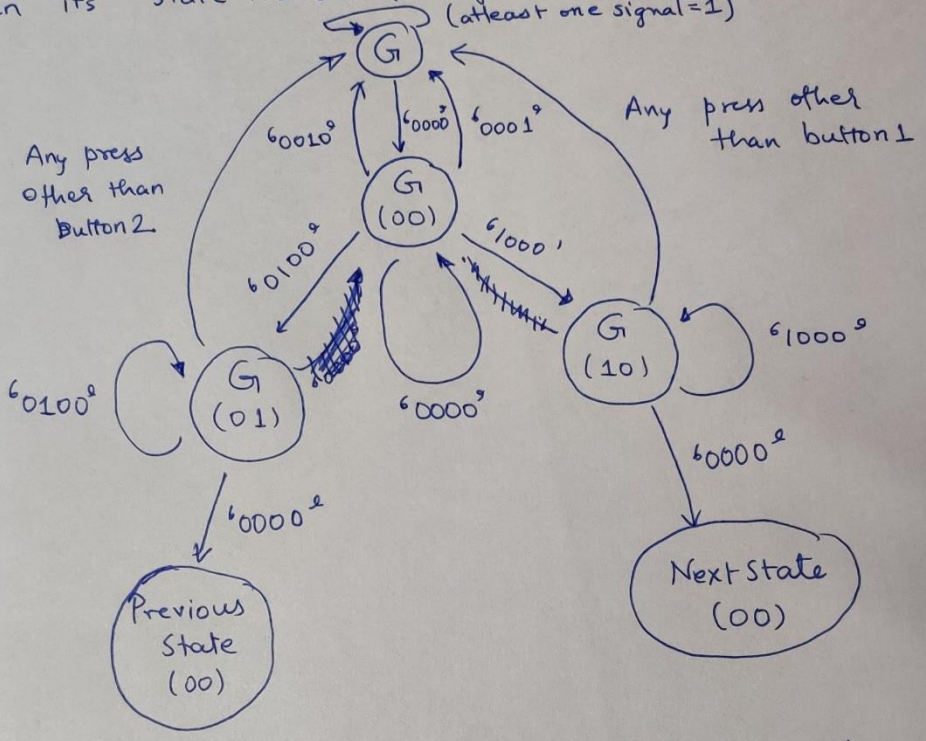
Only showing non-trivial transitions. For e.g., off->off is trivial.

Display Modes

Input signal (coming from 4 buttons) is represented as $'x_1x_2x_3x_4'$,
where x_i comes from button i $(i=1,2,3,4)$

Let $G \in \{hourMin, minSec, hourSec, setH, setM, setS\}$

then its state transition diagram is as follows (atleast one signal=1)



where next state is given by arrow in the cycle

hourMin → minSec → hourSec → setH → setM → setS → hourMin

and previous state is given by arrow in the cycle

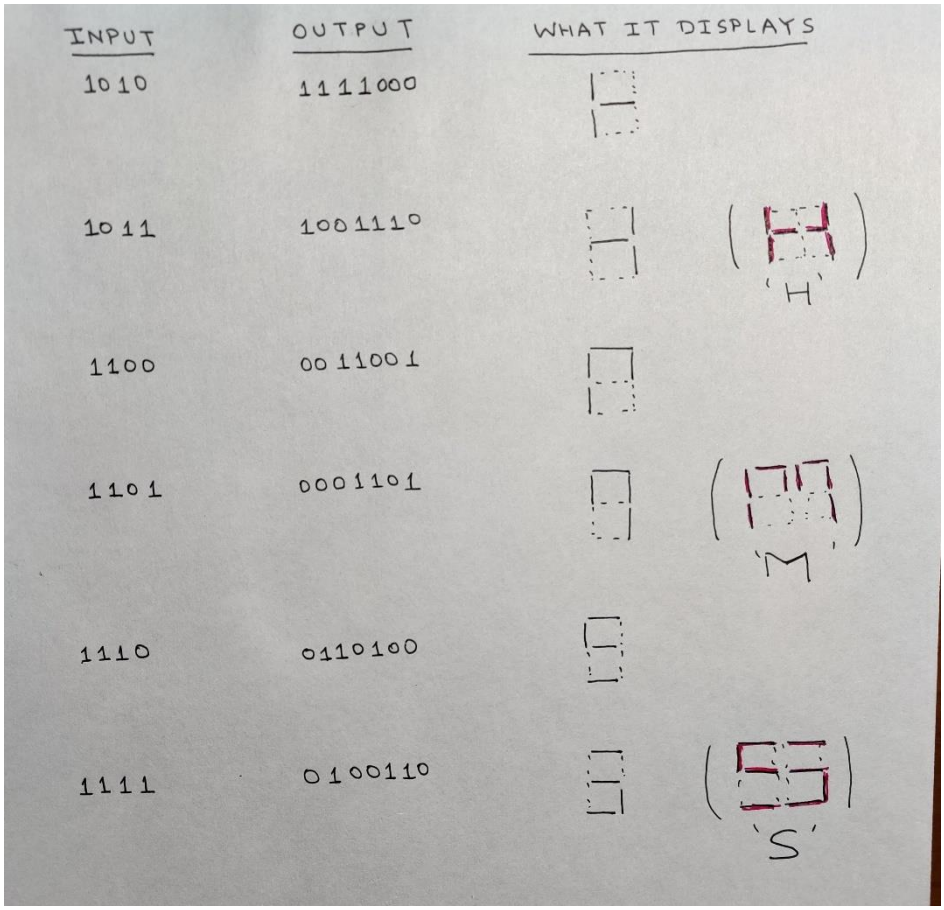
hourMin → setS → setM → setH → hourSec → minSec → hourMin

* Every display mode has 4 associated states as shown above.

My Design: SPECIFICATIONS: COMPONENTs

I have 3 components in my design. One of them is the **digital clock** itself, which I will describe in detail later. Here I describe the other **two** components that are used within my **digital clock** component:

convert: This component takes a single input (**bin4**) and gives a single output (**bin7**). Its sole purpose is to generate the cathode signals (7-bit signals) corresponding to the input signal (4-bit signal). For example, if **0** (zero) is to be displayed on the seven-segment board, then I want all the segments to light up except **G**. Hence, the input in that case will be **0000** (4-bit binary representation of 0) and the output will be **0000001**, indicating that all the LEDs except **G** will be forward biased, and **G** will be reverse biased (0 on cathode and 1 on anode **iff** forward biased). Apart from the usual input signals (0 – 9) that display the corresponding digit, I have also used **6 special symbols** (10 – 15). Their corresponding input and output are given below, along with what is displayed on the seven-segment board:



stepDown: This component takes the system's clock as an input (**sysClock**, bit, frequency **10 MHz**) and outputs a clock (**downClock**, bit) of frequency **250 Hz**. This stepping down of frequency is achieved by means of a **modulo-40,000** up-counter. Hence, only one additional signal is used in its architecture, i.e., **counter**, which is an unsigned vector of size **16** and is basically a **modulo-40,000** counter. As a modulo-N counter divides the frequency of the clock by N, therefore, the counter has a frequency of **10 MHz / 40,000 = 250 Hz**. The output signal (**downClock**) is high (1) for the first 20,000 counts and is low (0) for the next 20,000 counts, thus, mimicking a real clock of frequency 250 Hz.

Note: The detailed explanation of the VHDL code of these two components is given on a per-line basis as **comments** in the code itself. Kindly refer to those comments for further clarifications.

My Design: SPECIFICATIONS: DIGITAL CLOCK: TYPEs:

I have defined **three** types in my **digital clock** component. These types correspond to the **three** FSMs that I have already described above. These types are:
digitState: Enables **Digit State FSM**. It is an enumeration-type data type and a signal of this type can take any value from the set **{d3**, **d2**, **d1**, **d0}**.
mode: Enables **Display Mode FSM**. It is an enumeration-type data type and a signal of this type can take any value from the set **{hourMin**, **minSec**, **hourSec**, **setH**, **setM**, **setS}**.
buttonPress: Enables **Button State FSM**. It is an enumeration-type data type and a signal of this type can take any value from the set **{off**, **On1**, **pressed**, **invalid}**.

My Design: SPECIFICATIONS: DIGITAL CLOCK: SIGNALs:

In this section I will describe all the signals and input/output port that my **digital clock** entity/architecture uses:
clk: Input Port: bit: It is basically the BASYS3 system's clock signal that is provided to us (10 MHz)
pb1: Input Port: bit: It is the signal coming from push-button 1 (can be low(0) or high(1))
pb2: Input Port: bit: It is the signal coming from push-button 2 (can be low(0) or high(1))
pb3: Input Port: bit: It is the signal coming from push-button 3 (can be low(0) or high(1))
pb4: Input Port: bit: It is the signal coming from push-button 4 (can be low(0) or high(1))
anode: Output Port: unsigned[4]: It is the anode signal for the 4 display boards. Exactly one of the four bits is high(1) at an instant and the rest are low(0).
cathode: Output Port: unsigned[7]: It is the cathode signal for the seven-segment display boards. The signals correspond to the symbol to be displayed on that board and it gets updated every 4 ms. (Decimal point is excluded as it not used. Only segments A to G are used)
digitDisplay: Signal: digitState: Indicates digit state, i.e., which board out of the 4 seven-segment boards is going to be activated at a particular instant. It is initialised with value **d0**, indicating that the rightmost board is activated on start.
displayMode: Signal: mode: Indicates display mode, i.e., what is going to be displayed on the seven-segment boards at a particular instant. It is initialised with value **hourMin**, indicating that hour:min will be displayed on the boards at start (hh:mm).
button1, button2, button3, button4: Signal: buttonPress: Indicates the state of the buttons 1, 2, 3 and 4 respectively at a particular instant. All of them are initialised with **invalid**, so that any button press before starting is not treated as a valid input. After one clock cycle of no input, all the buttons will eventually go into the **off** state.
hour1: Signal: unsigned[4]: It is the 4-bit binary representation of the most significant digit of the hour count (ranges from 0 to 2). It is initialised with **0000** (0).
hour0: Signal: unsigned[4]: It is the 4-bit binary representation of the least significant digit of the hour count (ranges from 0 to 9). It is initialised with **0000** (0).
min1: Signal: unsigned[4]: It is the 4-bit binary representation of the most significant digit of the minute count (ranges from 0 to 5). It is initialised with **0000** (0).
min0: Signal: unsigned[4]: It is the 4-bit binary representation of the least significant digit of the minute count (ranges from 0 to 9). It is initialised with **0000** (0).
sec1: Signal: unsigned[4]: It is the 4-bit binary representation of the most significant digit of the second count (ranges from 0 to 5). It is initialised with **0000** (0).
sec0: Signal: unsigned[4]: It is the 4-bit binary representation of the least significant digit of the second count (ranges from 0 to 9). It is initialised with **0000** (0).

digit3, digit2, digit1, digit0: Signal: unsigned[4]: These are the 4-bit binary representation of the symbols that have to be displayed on the seven-segment board from left to right respectively. They are initialised with **0000** each (representation of symbol **0**).
c3, c2, c1, c0: Signal: unsigned[7]: These are the corresponding 7-bit cathode signals of the symbols that have to be displayed on the seven-segment board from left to right respectively. They are initialised with **0000001** each (cathode signal for displaying symbol **0**).
masterClock: Signal: bit: It is the **250 Hz** clock that I will be using to trigger various processes in my architecture and is derived from the system's clock (**10 MHz**).
counter1Hz: Signal: unsigned[8]: It is a **modulo-250 up-counter** and it is used to update time count by 1 (only when not in set time state) every second by counting 250 pulses of the master clock. It is initialised with the value **00000000** (0).
counter5Hz: Signal: unsigned[6]: It is a **modulo-50 up-counter** and it is used to update time count by 1 (only when in set time mode and button is **pressed**) every 0.2 s (5 times in a second) by counting 50 pulses of the master clock. It is initialised with the value **000000** (0).
counterPress: Signal: unsigned[7]: It is a **saturating-125 up-counter** and it is used change a button's state from **On1** to **pressed** after 0.5 s (if kept pressed) by counting 125 pulses of the master clock. It is initialised with the value **0000000** (0).

My Design: SPECIFICATIONS: DIGITAL CLOCK: DECISIONs:

In this section I will briefly describe my major design decisions and how the clock will actually work:

- The **masterClock** (250 Hz) signal will be generated from **clk** (10 MHz) signal using component instantiation of **stepDown** component.
- I have created **6** display modes, which are divided into two categories:- **Dynamic** and **Static**. **Dynamic** modes include **hourMin, minSec** and **hourSec** modes. These states display the time in real time and the time is updated every second when in these states. However, user cannot change time according to their needs when in **Dynamic** mode. On the other hand, **Static** modes include **setH, setM** and **setS** modes. These states display a static time, which is not updated every second (counter is stopped). However, user is free to increment/decrement time (using buttons) according to their needs when in **Static** mode (fast increment/decrement is also provided).
- I am allowing the user to change hour, minute and second count **individually**. However, I am not providing them with an option to change each digit separately, rather I am implementing **fast** increment/decrement using which the user can change the count **5 times** in a second just by keeping button 3(or 4) pressed.
- Signals ***hour1, hour0, min1, min0, sec1*** and ***sec0*** are storing the time of the clock in 24-hour format. What they store individually has already been described in the previous section.
- The **digitDisplay** state signal will change every **4 ms** (@ 250 Hz) in a cyclic fashion (**d0 -> d1 -> d2 -> d3 -> d0**). This will ensure that the persistence of vision takes place, and we are able to see the output as we expect it to be.
- The state of the push-buttons is also updated every **4 ms** (@ 250 Hz).
- Push-buttons **1** and **2** are used for changing the display mode. **pb1** changes mode to the next mode, whereas **pb2** changes mode to the previous mode. The **next mode** cycle is as follows: **hourMin -> minSec -> hourSec -> setH -> setM -> setS -> hourMin**. The **previous mode** cycle is simply the reverse of the next cycle.
  Note that there is **no** notion of **pressed** state for **button1** and **button2**, i.e., no matter how long the user keeps the button pressed, it's not going to change its state from **On1** to **pressed**. Also, a change in display mode takes place **iff** the button (1 or 2) is released and the corresponding button

state changes from **On1** to **off**. Hence, state change from **invalid** to **off** will not result in any change in the display mode.

- Push-buttons **3** and **4** are used for changing time (hours, minutes or seconds) only when in **setH**, **setM** or **setS** display mode. In other display modes, they will have no effect whatsoever. Button 3 is for incrementing the time count, whereas button 4 is for decrementing the time count. Similar to buttons 1 and 2 above, time count is incremented/decremented by 1 when the button (3 or 4) is released and corresponding button state changes from **On1** to **off**. Hence, state change from **invalid** to **off** will not result in any change in the time count. However, unlike buttons 1 and 2, that is not the only way in which the time can be updated.
  Note that there is **a** notion of **pressed** state for **button3** and **button4**, i.e., if the button is kept pressed for more than 0.5 s (assuming that no other button is pressed during that duration), then the state of the button changes from **On1** to **pressed** (**counterPress** is used to implement this). When in **pressed** state, the time count is incremented/decremented every **0.2 s** (@ 5 Hz) depending on which button is pressed (3 or 4). Also **note** that there is no change in time when button state changes from **pressed** to **off**.
- In case two or more buttons signals are high(1) at an instant, all the buttons go into **invalid** state, and they remain in this state till **all** the button signals are low(0). This way no action is performed (like mode change, slow/fast increment/decrement) when the buttons go into **invalid** state and the conflict is resolved. Hence, the user must ensure that **only one** button signal is high at a time.
- **Time** is incremented by 1 every second **if** the display mode is **hourMin, minSec** or **hourSec**. This is achieved by using a modulo-250 up-counter (**counter1Hz**) which is of frequency **1 Hz**.
- **Time** is incremented/decremented by the user **if** the display mode is **setH, setM** or **setS**. This is done by using buttons 3 and 4. Fast increment/decrement is made possible by using a modulo-50 counter (**counter5Hz**) which is of frequency **5 Hz**.
- **Display Mode** is changed whenever there is an appropriate change in the states of button 1 and button 2. Since, buttons are checked every 4 ms, the display mode cannot change at a rate faster than 250 Hz.
- The four symbols/digits which have to be displayed on the seven-segment boards are updated whenever there is a change in **Display Mode** or any of the following signals: ***hour1***, ***hour0***, ***min1***, ***min0***, ***sec1*** or ***sec0***. This ensures that my output is in accordance with the time and display mode set by the user.
- The symbols/digits to be displayed are converted into the corresponding cathode signals by using component instantiation of **convert** component. (4 instantiations)
- The **anode** and **cathode** signal are updated concurrently with the change in **digitDisplay** signal, so that the output actually changes **@ 250 Hz** (Refresh period = 16 ms).
- The corresponding output in each **display mode** is shown through an example below:

These are all the major design decisions that I took. The VHDL code that I have written is just a simple encoding of these ideas involving a bunch of concurrent and sequential assignments. Hence, these points also explain the big chunks of my code and what they intend to do. Also, **note** that the detailed explanation of the VHDL code of **digital clock** component is given on a per-line basis as **comments** in the code itself. Kindly refer to those comments for further clarifications.

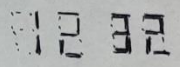
OUTPUT (FPGA)

(Based on Current Display Mode)

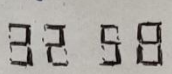
For example, current time is = 12 : 32 : 58

↑ hour count ↑ minute count ↑ second count

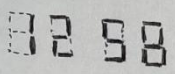
Output for hourMin state



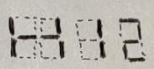
Output for minSec state



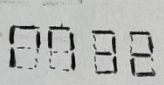
Output for hourSec state



Output for setH state



Output for setM state



Output for setS state