

COMPE 470

STUDENT PROJECT

HOUSE UTILITIES

Final Project Report

AUTHOR:

GIORGI SOLOMNISHVILI

5/05/20

1) Introduction

I wrote Verilog modules that represent house utilities: 1- digital clock displaying time in the following format: hh:mm:ss – MM/DD – MCDY; 2 - door lock which unlocks if correct combination is entered; 3/4 - dishwasher/gas_cooker which can be set to for certain amount of time – countdown is displayed on timer and 1 minute after timer reaches 0 the power is turned off; 5 – garage door which has remote control allowing the user to open and close the door or activate smart mode. In smart mode, when a car approaches the garage, the door opens. The door stays open until car goes into the garage and the driver comes out of it. After the driver leaves the garage, door closes and the smart mode is disabled. I wrote 4 top modules. Top module for clock has 31 instances of submodules instantiated, top module for dishwasher – 10, top module for lock 2. I am going to describe what each of those modules do. I am going to provide block diagram for each of those top modules. I will explain the code in detail. Moreover, I will provide commented source code. I have tested modules and I am going to include resulting waveforms in the project.

2) High level design in block diagram form:

I wrote 18 modules for four house utilities. Those utilities are a digital clock displayed on the wall, 4-digit combination lock, dishwasher and gas cooker, and garage door control.

Digital Clock: The house has a digital clock displayed on the wall of the living room. The clock shows a 24-hour time format. The clock displays hours, minutes, seconds, and date (mm/dd/year). I use the following 7 – segment LED configuration 14 times to display time in the following format – h₁₀ h : m₁₀ m : s₁₀ s; mm/dd/MCDY.

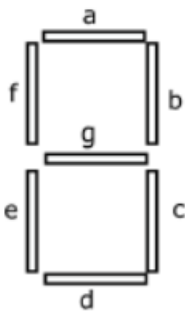


Figure 1

If we represent abcdefg as 7 bit long number, then

126 represents 0, 48 - 1, 109 - 2, 121 - 3, 51 - 4, 91 - 5, 61 - 6, 112 - 7, 127 - 8, 115 – 9.

Moreover, 10 seconds before the New Year, the clock displays count down for 10 seconds.

The clock displays a weekday: M,T,W,TH,F,SAT,SUN. The clock uses the following LED configurations to display weekdays:

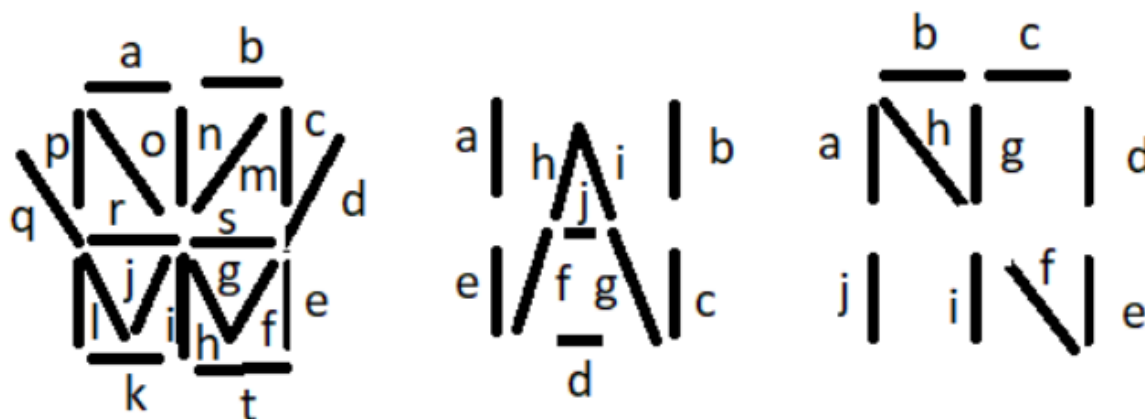


Figure 2

The first one is used to display the first letters: M, T, W, F, S. If we represent abcdefghijklmnopqrst as 20 bit long number, then M represents to 281BO, T – C1040, W – 16C08, F C0116, S – C8417. Numbers are in HEX.

The second one is used to display the second letters: H (TH), A (SAT), U (SUN). If we represent abcdefghij as 10 bit long number, then H represents to 3A1, A – 1F, U – 3E0. Numbers are in HEX.

The third one is used to display the third letters: T (SAT), N (SUN). If we represent abcdefghij as 10 bit long number, then T represents to 18A, N – 275. Numbers are in HEX.

I use Doomsday Algorithm by John Conway to determine weekday. Conway noticed that there are 12 dates in the year that occurs at the same day. For example, 7th November and 11th July are Saturdays in 2020. They will be Sundays in 2021. Every year these two dates will be on the same weekday. Conway called such dates Doomsday. Table 1 shows 12 Doomsdays:

Jan	Feb	Mar	Apr	May	June	July	Aug	Sep	Oct	Nov	Dec
3 (4)	28(29)	14	4	9	6	11	8	5	10	7	12

Table 1

If the year is leap year, Doomsdays in January is 4 and in February – 29. Otherwise, Doomsdays in January is 3 and in February – 28. If we know what weekday the Doomsday of the year is, we have reference point weekday for each month of that particular year. For example, all Doomsdays in 2020 are Saturdays. 10th August of 2020 is Monday, because 8th August (Doomsday) is Saturday.

In order to identify weekday for Doomsdays for a particular year, we need to assign a number to each weekday. See table 2:

Weekday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
---------	--------	---------	-----------	----------	--------	----------	--------

Number	1	2	3	4	5	6	0
--------	---	---	---	---	---	---	---

Table 2

The Gregorian Calendar repeats everything in every 400 years. If Doomsdays in 2020 are Saturday, Doomsdays in 2420 will also be Saturday. Conway called numbers corresponding to the Doomsday weekdays of century year Century Code. Table 3 lists Century Codes for 1500, 1600, 1700 and 1800 years.

Year	1500	1600	1700	1800
Century Code	3	2	0	5

Table 3.

Since Gregorian Calendar repeats in every 400 years, we know Century Codes for every century based on table 3. Now I will demonstrate calculating weekday with an example. 28th April of 2020 is Tuesday. It is also Due Date for draft report. The date is 4/28/2020. The century code – **CE, is 2**, because $2000 - 400 = 1600$ and CE for 1600 is 2. Now, we need to calculate the integer part and remainder of the ratio of the number created by the last two digit of the year and 12. The last two digit of 2020 is 20. **$20/12 = 1$ and the remainder is 8**. Next, we need to determine the integer part of ratio of the remainder and 4. **$8/4 = 2$** . Next, we need to find sum of those four bolded number: **$2 + 1 + 8 + 2 = 13$** . If the sum is more than 6, we need to find the remainder of the ration of that sum and 7. **If we divide 13 by 7, the remainder is 6**. That last remainder is a weekday of every Doomsday in 2020. **6 corresponds to Saturday**. Doomsday in April is 4th April. In 2020, 4th April was Saturday. To find out what day is 28th April, we need to subtract 4 from 28 and find the remainder of the ratio of the difference and 7. **$(28-4)/7 = 3$ and the remainder is 3**. Next, we need to add that remainder to the number corresponding to weekday for the Doomsday, in our case 6. **$3+6=9$** . **The remainder of the ratio of that sum and 7 is weekday for 4/28/2020. $9/7 = 1$ and the remainder is 2. 2 corresponds to Tuesday**. This algorithm is implemented in module clock-week-day.

This features are implemented by the following modules: clock, clock_digit_counter, leap_year_detector, clock_mm_dd_counters, clock_year_YY, clock_weekday, clock_digit_display, clock_digit_display_weekday, clock_division_remainder, clock_set_time. Each of these modules explained in detail below. Figures 3 to 7 represent block diagram for those modules. Figures 3 to 7 together represent block diagram for module called Top_module_for_clock.

Figure 3 shows block diagram for h_10 h:m_10 m:s_10 s counters of digital clock

(Note, it is not shown on block diagrams but every closed loops under time names – minute, second – 10, etc. increments value by 1. Fig 3-4)

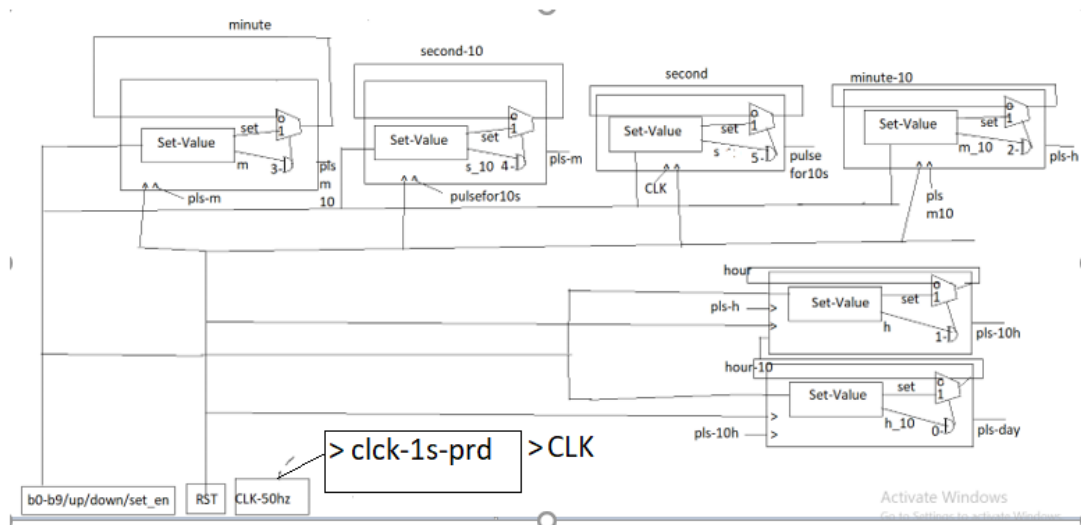


Figure 3

Each instance of time counter (second, second_10, minute and minute_10) takes RST signal and some sort of clock signal. RST resets counters' values to 00:00:00. Counter for second takes signal CLK which has period of 1s. CLK is generated by module clk-1s-prd. When Count value of second comes down from 9 to 0, it generates pulse. This pulse – pulse for 10s, is an input clock for second_10 counter module. When the latter's count value comes down from 5 to 0, it generates pulse pls-m. The latter pulse is an input clock for minute module. This minute module generates pulse – pls_m_10, when its counter value comes down from 9 to 0. That pulse is an input clock for minute 10 module. Minute_10 module generates pulse – pls-h, when its value changes from 5 to 0. This pulse is an input clock for module hour. Whenever its counter value resets to 0, the module hour generates pulse – pls_10h. pls_10h is an input clock for module hour_10. Whenever its counter value resets to 0, the module hour_10 generates pulse – pls_day. As seen from figure 3, each instance of time digit counter module has sub_module Set_value whose output is multiplexed. Each time digit counter is numbered from 0 to 5. 0 corresponds to the first hour counter, ..., 5 corresponds to last second counter. For example, when the user tries to set time, he/she enters a digit that he wants to be assigned to minute digit, Set-Value generates the new value for minute digit and position of minute digit which is 3. This new value and position is sent to all instances of digit counter modules and if position coincides with assigned number of that instance, the counter value will be updated.

Figure 4 shows block diagram for mm/dd/MCDY counters of digital clock:

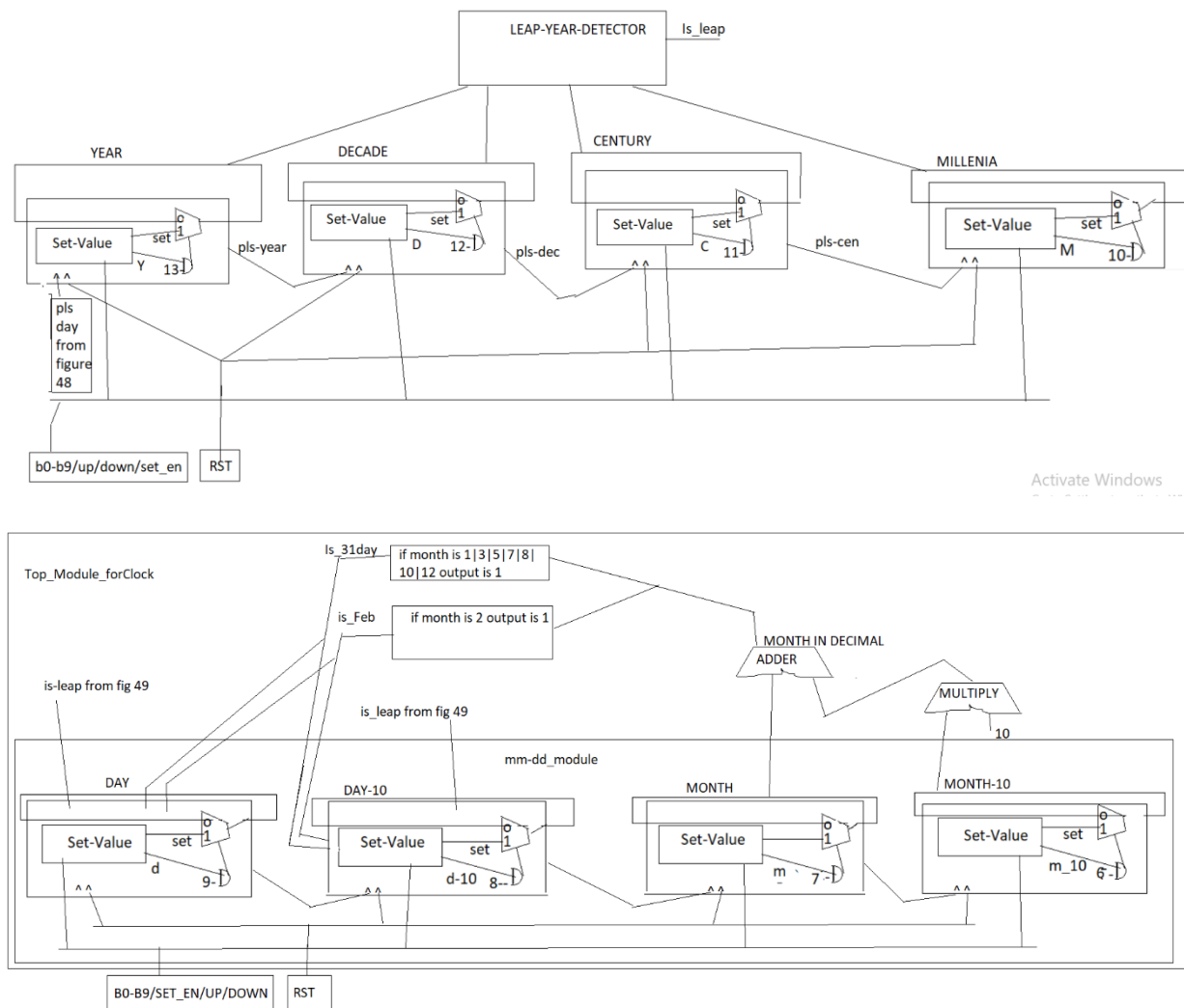


Figure 4

Logic for counting modules – DAY, DAY_10, MONTH, MONTH_10, YEAR, DECADE, CENTURY, MILLENIA, are similar to time digit counting modules presented on figure 3. The difference is that I have 1 more module instantiated on figure 4 – LEAP_YEAR_DETECOT, which detects if the year is leap. Century changing years, like 1900, 1800, 1200, are leap years if they are divisible by 400. The ordinary years, like 2020, 1994, are leap if they are divisible by 4. So the module LEAP_YEAR_DETECOT takes output digitss of YEAR, DECADE, CENTURY, MILLENIA, and determines if the year is leap. Moreover, I also check outputs of modules MONTH and MONTH_10 to see if month has 31 days, 30 days or the month is February. This information is delivered to modules – DAY and DAY_10, so that they can up to correct value based on the month. Figure 3 and 4 together present block diagrams for Top_Module_For_clock.

There is 1 more module - Week_day_detector, instantiated in Top_Module_For_clock. This is displayed on figure 6. Week_day_detector determines weekday (MON, ..., SUN), based on date

– d10/d/m10/m/MCDY. Those 8 digits are outputs of DAY, DAY_10, MONTH, MONTH_10, YEAR, DECADE, CENTURY, MILLENIA. Module Week_day will be explained in details

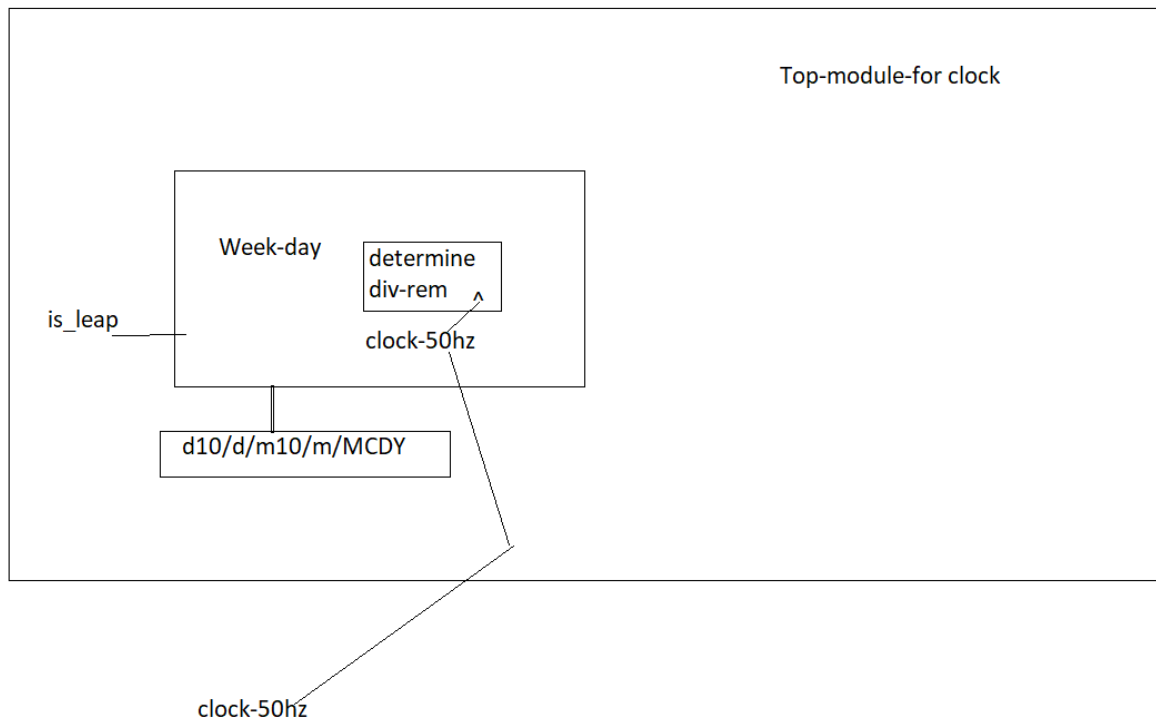


Figure 6 – determining week day/d10/m10/m/d/MCDY from figure 50 and 49/is-leap from figure 5

Top_module_for_clock also has 14 instances of module that implements 7 segment BCD (See figure 7). 14 digits, that represent time and date, are inputs for 14 instances of Digit_to_display module. It is also determined if it is 10s seconds before NEW YEAR. In that case, 7 segment BCD presents countdown from 9 to 0.

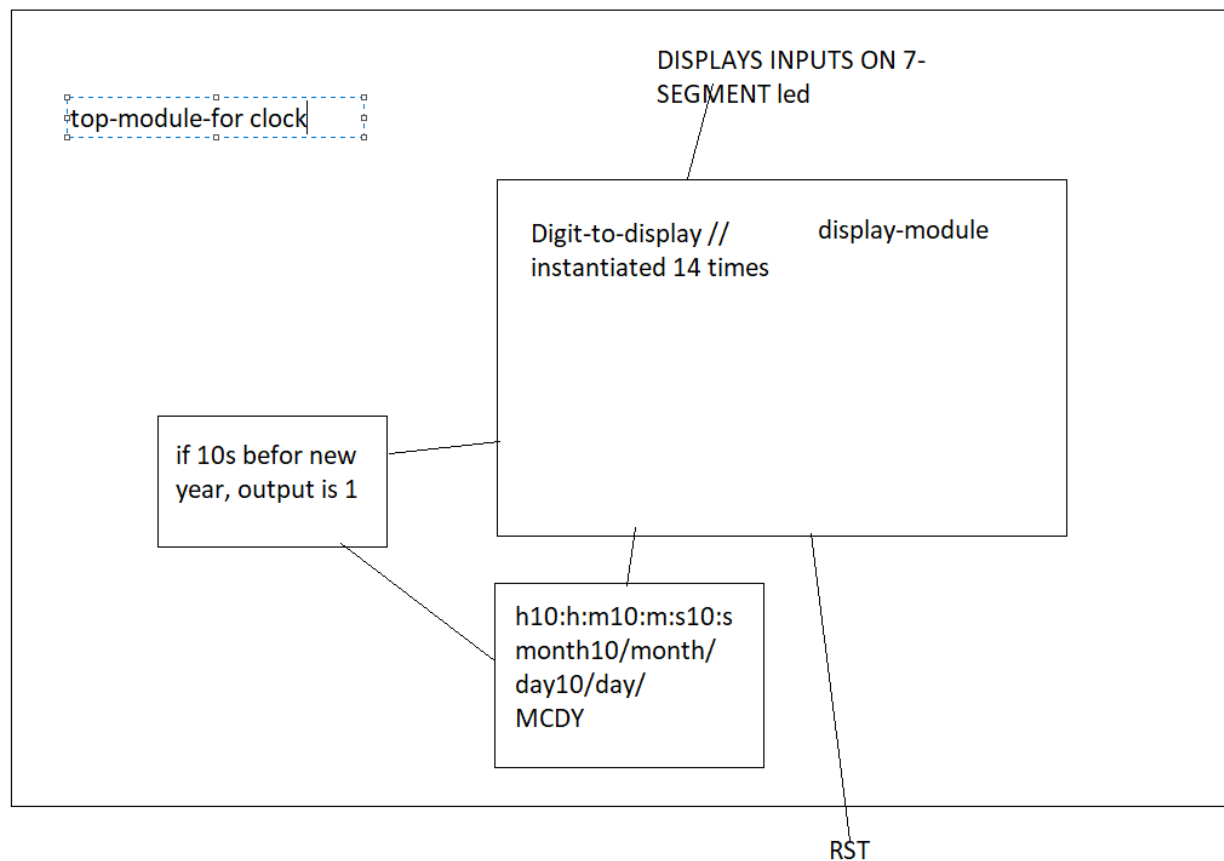


Figure 7// 7-segment LEDS

Figures 48-52 shows only connections of modules. It does not show logic behind those modules. Everything on those pictures are instantiated in top-module-for-clock

Combination lock: I want my house to have a 4-digit lock. When the lock is installed, the owner can choose any combination he wants. I wrote two modules for this particular device. Those modules are Lock-sequence-detector and lock keypad.

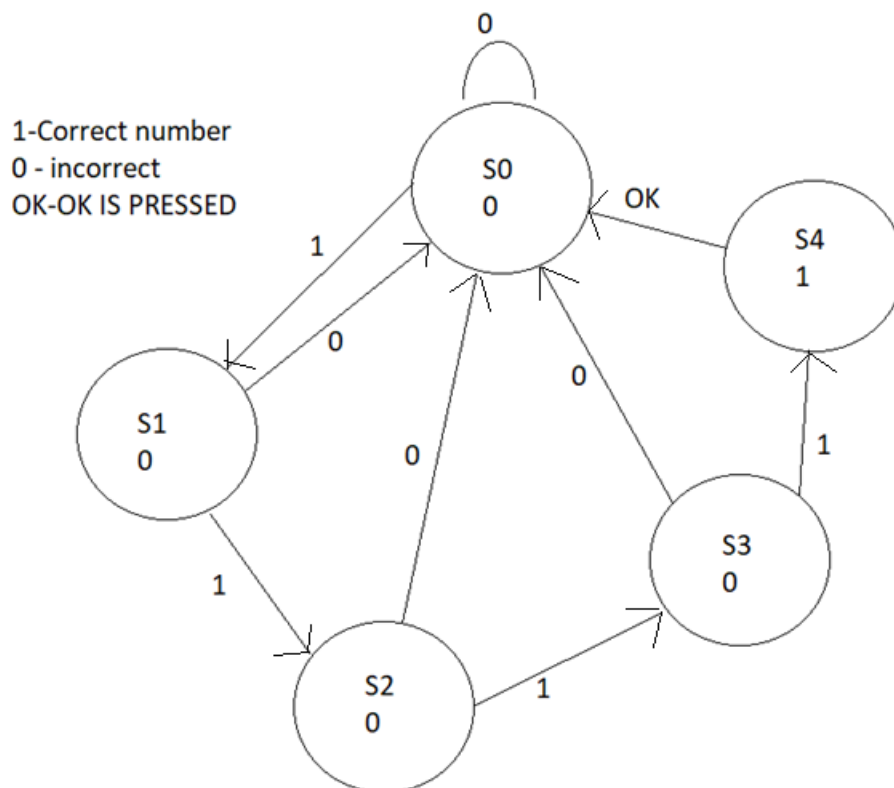
The lock has a sequence detector FSM with 5 states: s0, s1, s2, s3, s4. S0 is a reset state. Lock reset signal resets state to s0. The sequence detector has 4 bit with array variable called combination. The array stores 4 4bit long numbers. Those numbers are the correct combination. The module takes two inputs – input number and number position. Number position selects one of combination array's value. The chosen value is compared to input number. If correct number

is inputted, the state advances, otherwise the state goes to reset state – s0, and stays there till the end. If all input numbers are correct, then the door opens. Door locks itself whenever we physically close it.

The module keypad_for_lock represents 0-9 buttons and OK button. This keypad has two functions. If the door is physically open, pushing buttons on the keypad will set new unlock combination. If the door is closed, the user enters unlock combination with that keypad. The keypad generates a pulse after its button is pressed. Sequence detector uses that pulse as a clock. The keypad also returns position of input number. Based on position of input number, one value from the array – combination, will be selected. Input number will be compared to that value of variable combination at that position. The lock is unlocked after the user enters correct combination and presses OK.

Those two above-mentioned modules are instantiated in top_module_for_lock.

Figure 8 represents bubble diagram and next state table for sequence detector. Figure 9 represents block diagram for top_module_for_lock.



Current state	Next State Input = 0	Next State Input = 1	output
S0	S0	S1	0

S1	S1	S2	0
S2	S2	S3	0
S3	S3	S4	0
S4	S0	S0	1

Figure 8

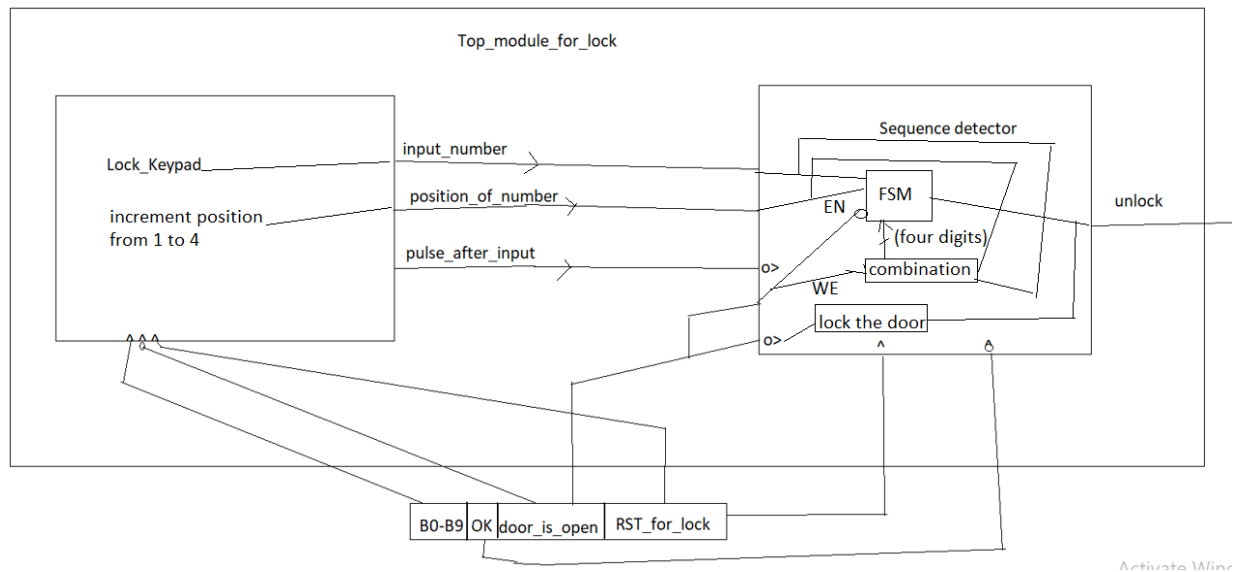


Figure 9

Dishwasher Gas Cooker: I wrote a hardware description code for a dishwasher timer. If the owner set timer at t, the washer will do the dishes in t time. The washer also displays a timer on its screen. The washer displays hours and minutes.

I implemented this device in four modules: clock, dishwasher_cooker_keyboard, clock_digit_display and timer. Clock and digit_display are the similar to ones used in digital clock design. I just instantiated already existing modules. Clock gives us clock signal whose period is 1m or 60s.

The keyboard module has 10 buttons – b0 to b9, representing digits for timer. It also has POWER which turns on/off the machine and START button which starts cooking or washing. The keypad generates pulse after pushing digit buttons – b0 to b9. The keypad also gives us signals specifying if power is on and if the device is doing its job or not. The keypad also gives us the position of entered digit. This position determines entered number is hour or minute. If position is 1, the entered number corresponds the first digit of hour. If position is 4, the entered number corresponds to the last digit of seconds.

Timer takes entered numbers generated by the keyboard and upon pressing start it begins count down. When countdown reaches to 00:00, signal finish is set to 1 and the power is turned off. Timer is instantiated four times to represent h10h:m10m. Those timer digits are displayed on 7-segment BCD which is also instantiated 4 times. If power is off, the LEDs are turned off as well.

Clock module, keyboard, 4 instances of timer and 4 instances of Clock_digit_display are instantiated in top_module_for_dishwasher_gascooker. Figure 10 displays block diagram for top_module_for_dishwasher_gascooker.

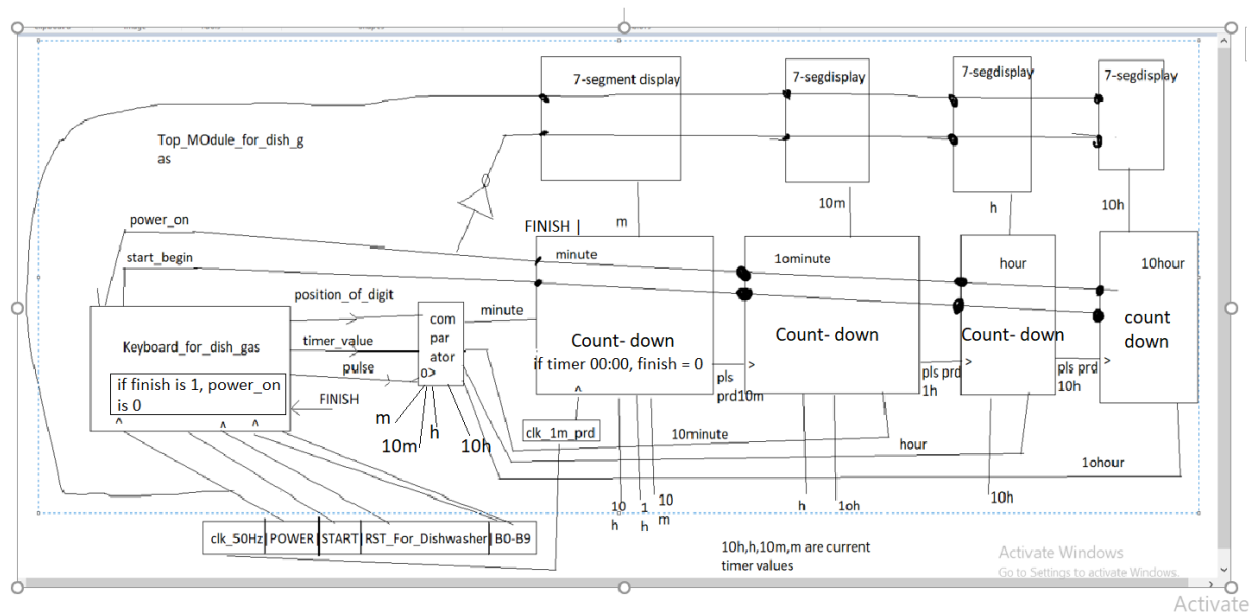


Figure 10

Garage Door: I wrote hardware description code for a garage door. The user of the garage has a remote control with two buttons. One of them simply opens and closes the garage. The module has an output signal called open_the_door. If signal open_the_door, the garage door opens. Otherwise the door closes. Pressing that button generates a pulse. The hardware uses that as clock and at positive edge of the pulse assigns open_the_door to its complement.

I used FSM to implement smart garage mode. The FSM has s0, s1, s2 and s3 states. S0 is a reset state. When smart garage mode is enabled, state becomes s0. At s0 the door is closed. If an object is detected in front of the garage, the state goes to s1 and the door opens. As long as an object is in front of garage, the state does not change. After the object left the front of garage, the state advances to s2, the door is still open. State does not change until an object is detected again. If an object is detected again, state advances to s3, the door is still open. After the object leaves the front of garage, state goes to s0, door closes and smart mode is disabled. The remote control has a button that turns on the smart garage mode. The module has an input signal called is_object. If an object is present in front of the garage door, is_object is 1. When smart mode is

on, if an object is detected in front of the door, the door opens. Figure 11 represents bubble diagram and Figure 12 - state table of FSM. Figure 13 represents block diagram for garage door.

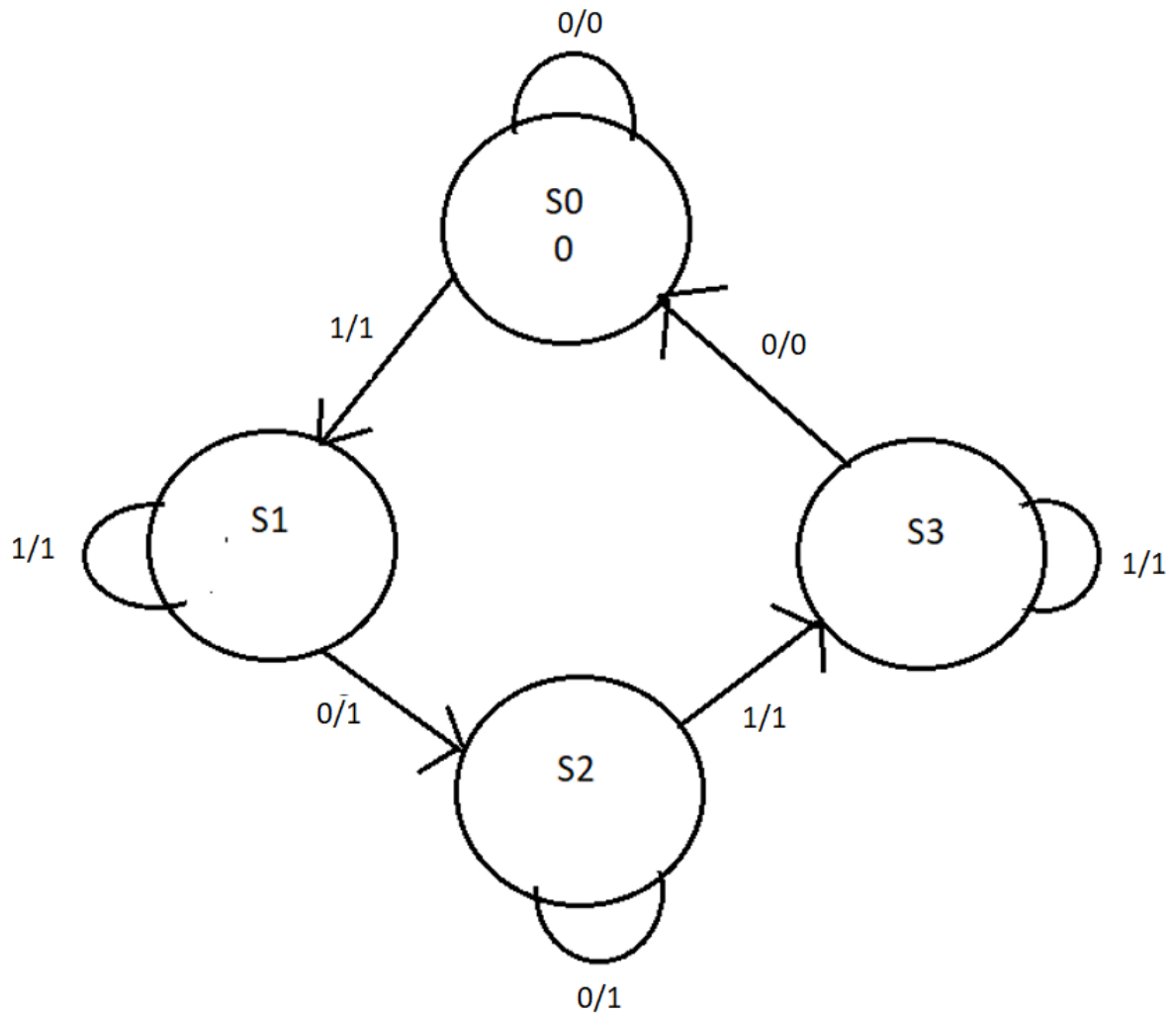


Figure 11

Current State	Next State Input = 0	Next State Input = 1	Output (open_the_door) Input = 0	Output (open_the_door) Input = 1
S0	S0	S1	0	1
S1	S2	S1	1	1
S2	S2	S3	1	1
S3	S0	S3	0	1

Figure 12

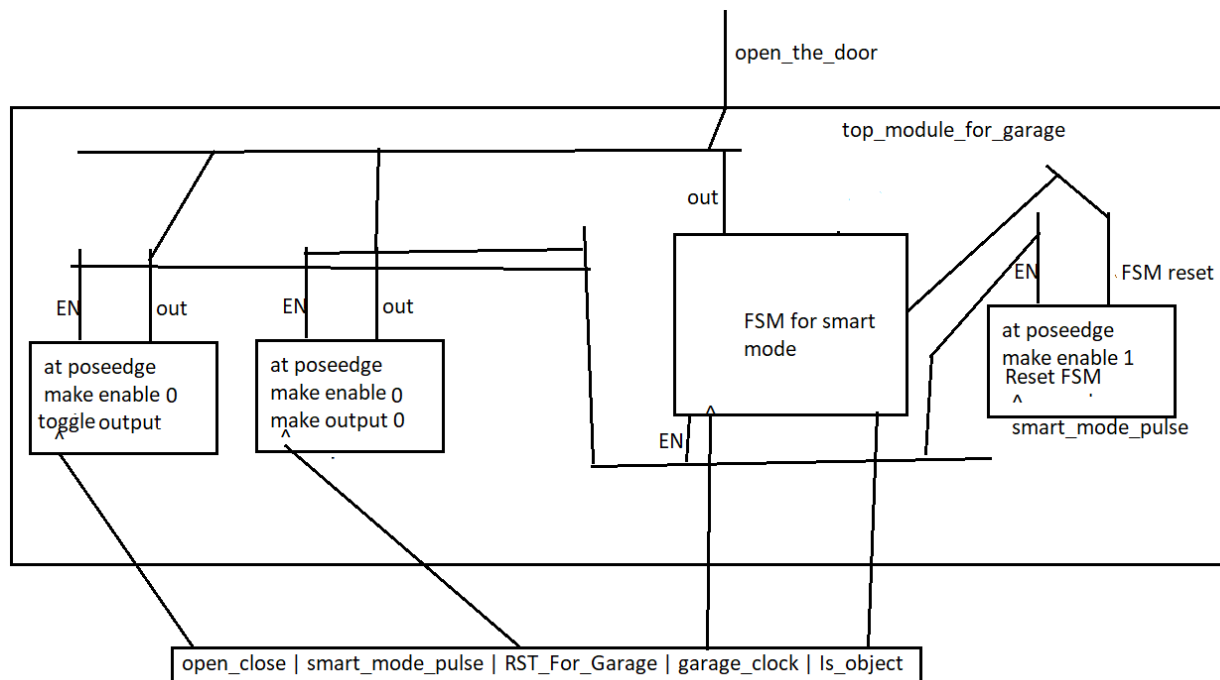


Figure 13

3) A descriptive paragraph for each top level function explaining what's in it and how it works:

Digital Clock

Top_module: This is top module for clock only. Since the digital clock uses 14 digits to display time, date and year, the module has 14 output nodes for values of each digit; 14 outputs nodes representing for 7-segment BCD's active LEDs; one output for weekday, and 3 for weekday LEDs. The module has the following inputs: clock, Reset, set time enable, keyboard – 10 digit buttons, up and down buttons.

There are 31 instances of modules instantiated in top module. I will explain each of them and clarify how they interact in top module. Totally there are 10 different kinds of modules in top module for digital clock.

Module clock: the module has 1 input signal and 1 output. The input signal is a square wave clock with 50 Hz frequency. The output is the same signal with modified period. The module has a parameter. The value of parameter is the period of output clock in seconds. The instance of clock has parameter 1 and gives clock with 1s period. The module has a counter which is initiated at 0. The counter is 11 bit wide, thus the maximum value of counter is 2047. The module also has a reg variable tmp which is instantiated at 1 (this will be output clock). Period of input clock is $1/50=0.02s$. At positive edge of the input clock, counter is incremented. Now, if I want the output clock to have half period 0.5s, I should toggle tmp after positive edge of input clock occurs 25 times, because $0.5/0.02$ is 25. That would give me an output clock whose period

is 1s. If I want an output to have period N seconds of period (N – natural number), I should toggle tmp after posedge of input clock occurs $25 \times N$ times. Counter increases at posedge of input clock. The counter counts from 0 to $25 \times N - 1$. After it reaches $25 \times N - 1$, the counter resets to 0 and tmp is toggled. This gives us a signal tmp with period of Nsecond. N is parameter of the module. **Endmodule clock.**

I use 14 digits to display time. Hence, I need 14 instances of counter. I wrote 3 different kinds of counters :

clock_digit_counters for hours, minutes, second – instantiated 6 times;

clock_mm_dd_counters for DD/MM – instantiated once.

clock_year_YY for MCDY – instantiated four times.

There is a module set_time_enable instantiated inside each of those clock_digit_counters. I will explain them after talking about clock_digit_counters.

I gave each of those 14 digits identifying number. See the following table:

Digit	H_10	H	Min_10	min	S_10	S	M_10	M	D_10	D	Mill	Cen	Dec	Y
Number ID	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Table 1

Module Clock digit counters: The module has a parameter – whichdigit. The number of parameters identifies digit of the clock which is represented by that instance of the module (see table 1). If parameter is 5, the instance is for the last second digits. One of the module's input is an asynchronous reset which resets time digit counter to 0. Hence, if reset is 1, time is 00:00:00.

The module has a 4-bit wide input – counter-in, which represents current value of time digit this of this particular instance, and a 4-bit wide output – counter-out, which represents next value of time digit of this particular instance. In top module, this particular output and input are connected, thus creating the loop between counter_out and counter_in. Usually, they equal to each other and at positive edge of input clock, their value increase by 1 or resets to 0. The module has an output pulse which becomes 1, if the counter_out becomes 0. This means that, for example, if instance of module is for digit representing the last second digit, pulse becomes 1 in every 10s. Thus, the period of the pulse is 10s. This pulse will be an input clock of instance of the module for second_10, the first second digit. Overall, each instance of the module has input clock with period of time digit that instance represent (second, 10second, minute, 10min, hour, 10 or 4 hour) and output pulse with period of the instance of time digit which is positioned left to

it (for 10second, minute, 10min, hour, 10 or 4 hour, a day). For example, output pulse of module for the last second digit is an input clock of module for the first second digit.

I have an always statement for incrementing counter_out and toggling pulse. The always is executed at posedge of input clock or RST. There is an if_else statement in always block.

If reset is 1, counter resets.

Next, if the parameter is 0 (hence counter for the first hour digit), the counter_out increments and pulse is assigned to 0, until it reaches to 2, meaning that time reached 20:00. During the next posedge of input clock, the counter_out goes to 0 and pulse becomes 1, meaning we have a midnight 00:00. Pulse became 1 because digit for day should change.

Next, if the parameter is 1, the instance is for the second hour digit. If the first hour digit is less than 2, pulse is assigned to 0 and the counter_out increments, until it reaches to 9. During the next posedge of input clock, the counter_out goes to 0 and pulse becomes 1, meaning we should increment the first hour digit. At this point hour can change from 00 to 19 hours. The following part makes sure that 20 to 23 hours values are also possible. Else If, the first hour digit is 2, pulse is assigned to 0 and the counter_out increments, until it reaches to 3. During the next posedge of input clock, the counter_out goes to 0 and pulse becomes 1, meaning we set the first hour digit to 0 as well, because we have midnight. At this point hour can change from 00 to 19 hours. At this point hour can change from 00 to 23 hours.

Next, if parameter is 2 or 4, meaning we have the first digit of minute or the first digit of seconds (min_10 or sec_10), the counter increases till it reaches 5. During the next posedge of input clock, the counter resets and pulse is set to 1. Counter increases up to 5, because there are 60 minutes in an hour and 60 seconds in a minute.

Otherwise, if parameter is 3 or 5, meaning we have the last digit of minute or second, the counter increases till it reaches 9. During the next posedge of input clock, the counter resets and pulse is set to 1. This logic is implemented inside the always block with if-else statement. This module is instantiated 6 times. **EndModule_clock_digit_counter.**

Module clock mm dd counters: When I was writing module for mm/dd, I had to take lots of things into account. Also, those four values are closely related and dependent on each other. That

is why I include them in the same module. The module is instantiated once in top module for clock. The module has an asynchronous reset which resets date to 11/07 (birthday of the author). The module has an input clock which is output of instance of previously discussed module for the first hour digit. The input clock has period of 1 day. The module has four input signals to identify if year is leap, month is February, month has 30 or 31 days. The module has output counters for the first and second day, and month digit - counter_for_clock_day, counter_for_clock_10day, counter_for_clock_month, counter_for_clock_10month. The module also has output pulses for counting the first digit of day, the first and the second digit of month and the digit for least significant decimal in year. Those pulses are

pulse_period_10days, pulse_period_month, pulse_period_10month, pulse_period_year.

The module has four always statements. Each always statement is for each of the output time digits.

Always statement for the last digit of day has asynchronous reset and clock whose period is a day. If month is February, days change from 01 to 28(9). If the day is before 20 February, the last digit of day is incremented until it reaches 9 and pulse for 10days is set to 0. During the next positive edge of input clock, the counter for the last digit of days becomes 0 and pulse for the first digit of days becomes 1. This pulse will be clock for always statement of the first digit counter. If the day is above 20 February, the last digit is incremented until it reaches 8+Is_leap_year. Is_leap_year is 1 if the year is leap year, otherwise it is 0. During the next posedge of input clock, counter becomes 1, because the month changed and days in new month start at 01. The pulse for 10 days is also set to 1. Else if month has 31 days, days change from 01 to 31. The same logic is repeated. The difference is that when day is above 20, the counter counts up to 9 and then resets to 0. And if counter is above 30, it counts up to 1 and then resets to 1, because new month has started. If month has 30 days, the last digit for day is incremented from 0 to 9, no matter what. This makes sense, because after 29 days of the month, the last digit should become 0. During the next posedge of clock, last digit for day becomes 1 and month changes. I do not need to specify that month changed so set counter to 1. I do not have two consecutive 1s. This works for months with 31 days. If month has 30 days, than 24 hours after the day became 30, pulse for the first day digit is not set to 1. However, it is reset to 0 at this

point. Hence, I have an always statement that corrects this situation and resets the first day digit at negedge of pulse_for_10days.

Next always is for the first digit of day. Its clock is pulse generated by the previous always statement. If month is February, the first digit of day changes from 0 to 2, otherwise it changes from 0 to 3. When the first digit of day resets to 0, pulse for month is generated. This pulse will be clock for always of the last digit of month.

Next always is for the last digit of month. It changes from 1 to 9 (Jan-Sept) and then from 0 to 2 (Oct - December). If the last digit of month resets to 0 or 1, the first digit of month should change so pulse_10month is set to 1. This pulse will be used for the always statement of the first digit of month.

Next always is for the first digit of month. It changes from 0 to 1. When it resets to 0, the year is changed, and a pulse is generated which will be used as a clock for the least significant digit of the year - Y. pulse is called pulse_period_year. **Endmodule clock mm dd counters.**

Module clock year YY: This module is instantiated four times to represent MCDY. The module has parameter which determines which digit of year is that instance. For M parameter is 10, ..., for Y - 13. The module has asynchronous reset which resets the module to value specified by parameter reset. The module counts from 0 to 9 and generates pulse that will be clock for the year digit standing left to that digit. Pulse of module for Y will be clock for D,, pulse of module for C will be clock for M. **Endmodule Clock year yy**

In top module, several logic operations are performed based on outputs of above discussed counters. If output of counter for month outputs 2, the signal Is_February is set. If month is 1,3,5,7,8,10, or 12, signal Is_31day is set. If time is 23:59:50 – 12/31, signal nycountdown is set, which makes & segment BCDs to display countdown for 10 seconds.

Module Leap Year Detector: This module is instantiated in top module and detect if year is leap. The module has four inputs, MCDY. The module constructs two numbers in decimal representation MC and DY. If year is 1234, upper2=two numbers are twelve and low2=thirty-four. If a number is divisible by 4, its last two bits are 00. Hence the module checks the last two bits of low2. If they are 00, year is leap, unless low2 is 0 itself. If low2 is 0, then we check last two bits of upper2. If they are 00, this means that year is divisible by 400 and year is leap. Otherwise year is not leap. **ENDMODULE LEAP YEAR DETECTOR.**

Module Division remainder: This module has an always statement which runs at fundamental clock whose frequency is 50Hz. When CLK changes from 0 to 1 or 1 to 0, the always statement is executed. The module has two inputs – dividend and divisor, and two outputs -quotient and

remainder. If dividend = divisor, the module returns quotient = 1, remainder = 0. If dividend < divisor, quotient is 0, remainder = dividend. Otherwise, divisor is subtracted from dividend until dividend is more than divisor. After each subtraction, the quotient is incremented. When dividend becomes less than divisor, remainder is assigned to dividend. If dividend becomes the same as divisor, 0 is assigned to remainder, quotient is incremented by 1 one more time.

ENDMODULE Division remainder

Module Week Day Detector: This module implements Conway's Doomsday algorithm to determine weekday. Its inputs are Is_leap (discussed above), CLK, DD/MM/MCDY. The module creates 4 decimal values from those inputs and those values correspond to day (28,23, 01...), month (1-12), most_2_sig_digitofyear (MC-0 - 99) and least_2_sig_digitofyear (MC-0 - 99). If look at table representing Century codes, we see that century code of $1500 + 400xZ$ is 3, $1600 + 400xZ$ is 2, $1700 + 400xZ$ is 0, $1800 + 400xZ$ is 5. Z is an integer. If we divide those years by 100, we get that $15 + 4xZ$ is 3, $16 + 4xZ$ is 2, $17 + 4xZ$ is 0, $18 + 4xZ$ is 5. In binary 4 is 100. Thus adding $4xZ$ to any number does not change the last two bits of that number. Hence, century codes of century years whose most_2_sig_digitofyear ends with 2'b11, just like 15, is 3; if it ends with 00, just like 16, - 2; if it ends with 01, just like 17, - 0; if it ends with 10, just like 18, - 5. This is how I found century code of the year.

I also need to find remainder and quotient obtained by dividing least_2_sig_digitofyear by 12. I used module division_remainder for that. To find quotient of remainder by 4, I shifted remainder right two times. Then, I added those four values and found remainder of the sum by 7. That remainder is doomsday of the year.

In the beginning I included table presenting which dates are doomsdays. Those dates are 5/09, 11/07, etc. The following code makes sure that doomsday of the month is assigned to variable Doomsday_of_month:

```
always @(month) begin
```

```
case(month)
```

```
1: Doomsday_of_month <= 3 + Is_Leap_Year;
```

```
2: Doomsday_of_month <= 28 + Is_Leap_Year;
```

```
3: Doomsday_of_month <= 14;
```

```
4: Doomsday_of_month <= 4;
```

```
5: Doomsday_of_month <= 9;
```

```
6: Doomsday_of_month <= 6;
```

```
7: Doomsday_of_month <= 11;
```

```
8: Doomsday_of_month <= 8;
```

```
9: Doomsday_of_month <= 5;
```

```
10: Doomsday_of_month <= 10;
```

```

11: Doomsday_of_month <= 7;
12: Doomsday_of_month <= 12;
default: Doomsday_of_month <= 'bx;
endcase
end

```

Then I find absolute value of difference = Doomsday_of_month-day+Doomsday_of_year. The remainder of that difference by 7 is weekday. I used module clock_division_remainder to calculate quotients and remainders in this module. **ENDMODULE Week Day Detector**

Module Clcok Set Time: The user has access to keyboard for digital clock. There are buttons corresponding to digits from 0 to 9. There are also two buttons: up and down button. The module has two output values – counter and new_value. Each digit on the display has a corresponding number (See table 1). Counter corresponds to number of the time digit from table 1. For example, if counter is 1, the user tries to change the last digit of hour. New_value is value entered by the user. When the set_time_enable is set to 1, counter becomes 0. By pushing up button, the user can increase counter. Max value of counter is 13 because I have only 14 digits on display of digital clock. This module is instantiated in every counter module discussed above. When the new_value changes, based on counter one of the 14 digits will change. **Endmodule set_value.**

Module clock_digit_display: This module simply takes an input digit and displays it on 7 segment BCD. I used case statement to implement this module. The distinguishing characteristic of the module is that it has a parameter- second_digit and input signal nycountdown. If this module is instantiated to display the last second digit, the parameter is 1. As discussed above, nycountdown becomes 1 ten seconds before new year. If nycountdown and second_digit is both 1, the module subtracts input digit to display from 10 and displays that number. Hence, the module displays 9, 8, ..., 1,0. This is 10s countdown before midnight. Else if nycountdown is 1 but second_digit is 0, the module turns off LEDS, because the module is not instantiated to display seconds. Otherwise, the module displays input digit on LEDS. **Endmodule clock_digit_display.**

Module clock digit display weekday: The module has three case statements. The first one displays the first letters of weekdays, the second one displays U if weekday is Sunday, A if weekday is Saturday, H if weekday is Thursday. The last case statement displays N for Sunday and T for Saturday.

The first one is used to display the first letters: M, T, W, F, S. If we represent abcdefghijklmnopqrst as 20 bit width number, then M (input 1) represents to 281B0, T (2 or 4) – C1040, W (3) – 16C08, F (5) C0116, S (0 or 6) – C8417. Numbers are in HEX.

The second one is used to display the second letters: H (TH – input number 4), A (SAT – input number 6), U (SUN input number 0). If we represent abcdefghij as 10 bit long number, then H represents to 3A1, A – 1F, U – 3E0. Numbers are in HEX.

The third one is used to display the third letters: T (SAT – input number 6), N (SUN input number 0). If we represent abcdefghij as 10 bit long number, then T represents to 18A, N – 275. Numbers are in HEX. **Endmodule**

All of those modules discussed above are instantiated in top_module for digital clock. **Endmodule**
Top module.

Dishwasher – Gas cooker

Top module for Dishwasher cooker: This is the top module for dishwasher or gas cooker. The top module has 10 instances of other modules. In total, there are 4 different kinds of modules instantiated in top module for dishwasher/cooker. The device has the following buttons: POWER, START buttons, and buttons representing digits from 0 to 9. The module has reset signal and input clock with 50 Hz frequency. The outputs of the top module are 4 digits displayed on 7 segment BCD. The device has timer display in the following format: hh : mm.

The modules instantiated in this top module are clock, dishwasher_cooker_keyboard, dishwasher_timer and clock_digit_display. The first and the last modules were discussed above. I use module clock to generate 1-minute period clock signal out of 50 Hz input clock. The clock_digit_display just displays timer digits. Since I have 4 digits, clock_digit_display is instantiated 4 times. The rest two modules will be discussed in detail.

Module dishwasher cooker keyboard: The module's inputs are POWER, START buttons, and buttons representing digits from 0 to 9, Reset signal and signal called finish. The outputs are signals like, pulse – generated after a digit is entered by the user, power_on – 1 when power is on, start_begin – becomes 1 after START is pressed; 3 bit long variable called position_of_digit (determining which of those 4 time digits (hh:mm) is entered by the user) and timer_value, which represents entered digit. Whenever the user presses POWER button the power_on signal toggles, meaning that pressing POWER button turns on and off power. Pressing POWER button also sets position_of_digit and timer_value variables to 0. This is implemented in an always statement who is sensitive to change of signal POWER. Whenever any of those buttons representing 0-9 digits are pressed pulse is generated. This pulse is used in top module. Moreover, pressing any particular digit button sets timer_value variable to that value and

increments position_of_digit variable. For example, after turning on the device, if the user presses 3, the position_of_digit becomes 1 which means that the first hour digit should start countdown from 3. Next, if we press 4, position_of_digit becomes 2, meaning the second hour digit should start countdown from 4; .etc. This is implemented by 10 always statements. Each of those always statements has each of those 10 digit buttons in their sensitivity list and they are executed at posedge of a pulse generated by pressing button in their sensitivity list. For example:

Always (posedge button_0)

Timer_value<=0;

The module has an input signal finish. Finish becomes 1 after the timer finishes countdown. At positive edge of finish, power_on becomes 0, and the device turns off. **Endmodule dishwasher cooker keyboard**

The top module for the dishwasher/cooker receives outputs of keyboard. At negative edge of pulse signal, the top module compares position_of_digit to the numbers – 1, 2, 3, and 4. If position is 1, the timer value is sent to an instance of module dishwasher_timer which represents the first hour digit. Else if position is 2, the timer value is sent to an instance of module dishwasher_timer which represents the second hour digit. Else if position is 2, the timer value is sent to an instance of module dishwasher_timer which represents the first minute digit. Else if position is 3, the timer value is sent to an instance of module dishwasher_timer which represents the second minute digit. Otherwise, the top module sends 0 to all instances of dishwasher_timer.

Module dishwasher_timer: the module input signals such as clock, power_on, start_begin. Those signals are generated in module for keyboard and are transferred to module dishwasher via top module for dishwasher. The module for dishwasher timer also has 3 4-bit long input variables previous_digit1, previous_digit2, previous_digit3. Those variables are fed with values of previous three digits. For example, if module is instantiated for the second digit of hour, previous_digit1 is the first digit of hour and when previous digit1 becomes 0, the module counts down to 0 and stops counting. Similarly, if the module dishwasher_timer is an instance of the last minute digit, previous_digit1 is h10, previous_digit2 is h, previous_digit3 is m10. If all three becomes 0, the module counts down to 0 and stops counting. The module also has an input variable timer_start_value, which corresponds to value set by the user. The module has two signal outputs – finish – becomes 1 when the counter of module becomes 0, and pulse which becomes 1 after counter resets from 0 to 9 (from 0 to 5 if the instance of the module is for the first digit of minutes). And finally, the module outputs a variable timer_value_current which holds value stored in the module counter.

Whenever the module receives new timer_start_value, this value is assigned to timer_value_current. This is implemented by an always statement who has timer_start_value in sensitivity list. This always statement assigns 0 to signal finish if timer_start_value is not 0. Otherwise, finish is set to 1. The module is instantiated 4 times and each of them outputs a signal finish. Whenever all four becomes 1, the timer value becomes 00:00 and the power will be turned off, providing that START button was already pressed and the device was doing its job –

cooking or washing. Whenever the user pushes START button, this means that he/she wants to start the machine wash or cook, so finish becomes 0.

The main counter of the module is implemented with an always statement whose sensitivity list contains input clock. The always is sensitive to rising edge of the clock. The module also has a parameter indicating which digit of time does this instance represents (hh:mm – 0123).

If this instance is for last minute digit, output of module clock is input clock signal. The output of module clock has 1m period and module clock was already discussed above. The module counts down from entered value to 0. During the next positive edge of the input clock, the counter resets to 9 and pulse signal is assigned to 1. This pulse signal is going to be an input clock for the first digit for minute. Whenever counter reaches 0, during the next positive edge of the input clock, if the previous 1,2,3 digits are 0s, the counter stays at 0 and finish signal is set to 1.

If the instance is for the first digit of minute, everything is the same, except its input clock is an output signal – pulse from the previous instance, and whenever counter reaches 0, during the next positive edge of the input clock, if the previous 1,2, digits are 0s, the counter stays at 0 and finish signal is set to 1.

If the instance is for the last digit of hour, everything is the same, except its input clock is an output signal – pulse from the previous instance, and whenever counter reaches 0, during the next positive edge of the input clock, if the previous 1, digits are 0s, the counter stays at 0 and finish signal is set to 1.

If the instance is for the first digit of hour, everything is the same, except its input clock is an output signal – pulse from the previous instance, and whenever counter reaches 0, the counter stays at 0 and finish signal is set to 1. **Endmodule dishwasher_timer.**

Output time values from previous four instances of module dishwasher_timer is transferred to clock_digit_display module (discussed above). The module's input value ny_countdown is fed with ~power_on which guarantees that the module displays transferred digit and not 10 – transferred digit whenever the power is on. **Endmodule Top_module dishwasher/cooker.**

Lock with 4 combination

Module top_module_for_lock: The module has 11 input signals representing buttons on keyboard from 0-9 and OK button. The module has an asynchronous reset and an input signal called door_is_open. Whenever the door is physically open door_is_open is 1. The module has 1 output signal called unlock which is 1 if lock is unlocked.

There are two other modules instantiated in top_module_for_lock. Those modules are called Lock_Sequence_Detector and Lock_keypad.

Module Lock_keypad: This module returns position of the number, a number from 0 to 9 and a pulse signal. Position of the number determines what is the position of the input number in combination. Position's value varies from 0 to 4. Reset signal resets position to 0. Whenever a number is entered, position is incremented. Thus, positions of the entered numbers are from 1 to

4. The pressing button is implemented in always statement. Thus, I have 10 always statements which corresponds to 10 digits on keyboard. After each key of the keyboard is pressed a pulse is generated. This pulse will be used in the sequence detector. Whenever, door_open is reset to 0, position resets to 1, meaning that whenever door closes, the lock waits for the first digit of the combination to entered. **Endmodule Lock keypad.**

Module Sequence detector: this module implements FSM that detects correct combination and unlocks the door. FSM's bubble diagram and state table are presented on figure 8. The FSM has an enable signal – which is the same as door_is_open signal. Whenever this enable is 0, the FSM functions. There are three signals in FSM sensitivity list: pulse generated from keyboard, pulse generated by pressing OK and reset signal. Always block representing FSM is executed at posedge of RST or negedge of pulse or negedge of OK. The FSM has 5 states – s0 – s4. S0 is reset state. Whenever FSM is reset, or door physically closes FSM resets. If correct combination is entered, the state advances, otherwise it resets. If FSM reaches state 4, this means that correct combination is entered. After pressing OK the lock unlocks and FSM resets to s0 state. The FSM is implemented with case statements. There is also a small memory element in the module called combination. Combination is an array of 4-bit words. There are four rows in the array. Unlock combinations are stored in the array. Whenever new number is entered and the door is closed, that number is compared to an element of array. If they match, state advances. There is a counter variable that is initially set to 0 and increases up to 3. The counter is incremented after a number is entered. Thus, counter corresponds to position of the entered number in the combination, 0 is first number ... 3 is the fourth.

Moreover, the memory – combination, has a write enable which is the same signal as door_is_open. Whenever the door is physically open, new combination can be written in the module. As discussed above, the keyboard generates a number and position of that number in the combination. If write is enabled, the generated number will be written at generated position. Hence, when door is open, if the user enters a combination like 1234, after the door physically closes 1234 will be unlock combination. When the door closes, the lock locks itself. **End**

Module Sequence detector.

Endmodule top module for lock.

Garage Door

Module Top module for Garage: This module implements the garage door single-handedly.

The module has input signals open_close, smart_mode_pulse, RST_For_Garage, garage_clock. Pushing open_close button opens or closes the door. This is implemented by an always statement which has open_close signal in sensitivity list. The always statement is executed at posedge of opne_close signal. Smartmode_pulse activates smart mode enable. RST is reset signal which disables smart_mode and closes the door if it is open. Input Is_object is 1 if an object is detected in front of the garage. Smart_mode is implemented using FSM which is

implemented with case statement inside the always. This is how smart mode works. Car is not in front of the garage – stay in s0 mode and leave the door closed. Car is detected – go in s1 mode and open the door. Car goes in the garage – go to state s2 and stay open. A person comes out of garage – go to state s3 and stay open. Person leaves the front of the garage – go to state 0, close the door and disable the smart mode. If isobject does not changes, the state does not advance. After the door closes, the smart_mode becomes disable. This is implemented by always statement. The FSM clock has 50 hz frequency.

Endmodule

4) All COMMENTED source code for each module in Verilog

Digital Clock

Top_module:

```
`timescale 1ns / 1ps
```

```
module Top_Module(
```

```
input CLK, RST,
```

```
// digital clock inputs and outputs
```

```
input set_time_enable,b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_0,up,down,
```

```
output [3:0] clock_second, clock_10second, clock_1minute, clock_10minute,  
clock_1hour,clock_10hour,clock_day,clock_10day,clock_1month,clock_10month,clock_year,  
clock_decade, clock_century, clock_millenia,
```

```
output [2:0] weekday, //tells which day it is
```

```
output [6:0] display_clock_second, display_clock_10second, display_clock_1minute,  
display_clock_10minute, display_clock_1hour,display_clock_10hour,
```

```
output [6:0] display_clock_day, display_clock_10day, display_clock_1month,  
display_clock_10month,  
display_clock_millenia,display_clock_century,display_clock_decade,display_clock_year,
```

```
output [19:0] display_I_letter,
```

```
output [9:0] display_II_letter,display_III_letter);
```



```
// instantiating clocks with periods 1s, 10s, 60s

wire clk_out_1; // clock woth period 1s

wire clk_out_10; // clock with period 10s

wire clk_out_60; // clock with period 60s

clock #(1) klok00(.clk_in(CLK),.clk_out(clk_out_1));// 1s

clock #(10) klok01(.clk_in(CLK),.clk_out(clk_out_10)); // 10s

clock #(60) klok10(.clk_in(CLK),.clk_out(clk_out_60)); // 60s


// DIGITAL CLOCK ////////////////////////////////////////

wire [3:0] counter_for_seconds=clk_out_1; // this will count seconds

wire pulse_period_10seconds;

clock_digit_counters #(5)
ut00(.RST(RST),.CLKk(clk_out_1),.counter_in(counter_for_seconds),.counter_out(clock_secon
d),.check(tmp),.pulse(pulse_period_10seconds),

.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),

.b_9(b_9),.b_0(b_0),.up(up),.down(down));


// this will count 10 seconds//////////////////////////////////////////

wire [3:0] counter_for_10seconds = clk_out_10; // this will count 10 seconds

wire pulse_period_1minute;

clock_digit_counters #(4)
ut01(.RST(RST),.CLKk(pulse_period_10seconds),.counter_in(counter_for_10seconds),

.counter_out(clock_10second),.check(tmp),.pulse(pulse_period_1minute),

.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),

.b_9(b_9),.b_0(b_0),.up(up),.down(down));
```

```
// this will count 1 minute//////////////////////////////////////
wire [3:0] counter_for_1minute = clock_1minute; // this will count 1 minute
wire pulse_period_10minute;
clock_digit_counters #(3)
ut11(.RST(RST),.CLKk(pulse_period_1minute),.counter_in(counter_for_1minute),
.counter_out(clock_1minute),.check(tmp),.pulse(pulse_period_10minute),
.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),
.b_9(b_9),.b_0(b_0),.up(up),.down(down));
```

```
// this will count 10 minute//////////////////////////////////////
wire [3:0] counter_for_10minute = clock_10minute; // this will count 10 minute
wire pulse_period_1hour;
clock_digit_counters #(2)
ut100(.RST(RST),.CLKk(pulse_period_10minute),.counter_in(counter_for_10minute),
.counter_out(clock_10minute),.check(tmp),.pulse(pulse_period_1hour),
.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),
.b_9(b_9),.b_0(b_0),.up(up),.down(down));
```

```
// this will count 1 hour//////////////////////////////////////
wire [3:0] counter_for_1hour = clock_1hour; // this will count 1 hour
wire pulse_period_10hour;
clock_digit_counters #(1)
ut101(.RST(RST),.CLKk(pulse_period_1hour),.counter_in(counter_for_1hour),
.counter_out(clock_1hour),.check(tmp),.pulse(pulse_period_10hour),
```

```
.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),
.b_9(b_9),.b_0(b_0),.up(up),.down(down));
wire [3:0] tmp;
assign tmp = (clock_10hour<2);
```

```
// this will count 10 hour//////////
```

```
wire [3:0] counter_for_10hour = clock_10hour; // this will count 10 hour
```

```
wire pulse_period_24hour;
```

```
clock_digit_counters #(0)
```

```
ut110(.RST(RST),.CLKk(pulse_period_10hour),.counter_in(counter_for_10hour),
```

```
.counter_out(clock_10hour),.pulse(pulse_period_24hour),
```

```
.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),
```

```
.b_9(b_9),.b_0(b_0),.up(up),.down(down));
```

```
//////////////////////////////////// you are
HERE
```

```
wire Is_Leap_Year; //this is 1 if it is leap year
```

```
wire Is_February; // is 1 if february
```

```
wire Is_31_day; // 1 if there is 31 days in month
```

```
wire [3:0] month; //month in decimal
```

```
assign month =clock_1month + clock_10month*10;
```

```
assign Is_31_day =
```

```
(month==1)|(month==3)|(month==5)|(month==7)|(month==8)|(month==10)|(month==12); //
```

```
assign Is_February = month==2;
```

```
leap_year_detector
leap_year_detector00(.millenia(clock_millenia),.century(clock_century),.decade(clock_decade),.
year(clock_year),.leap_year(Is_Leap_Year));//determine if we have leap year
```

```
wire pulse_period_10days; // becomes 1 when 10day digit change
```

```
wire pulse_period_month; // becomes 1 when month digit changeonce in a month
```

```
wire pulse_period_10month; // becoms 1 when 10month digit change
```

```
wire pulse_period_year; // becomes 1 at new year
```

```
// counters for mm/dd and four pulses mentine in lines 69-72
```

```
clock_mm_dd_counters
```

```
clock_mm_dd_counters00(.Is_Leap_Year(Is_Leap_Year),.Is_February(Is_February),.Is_31_day
(Is_31_day),.pulse_period_24hour(pulse_period_24hour),
```

```
.RST(RST),.counter_for_clock_day(clock_day),.counter_for_clock_10day(clock_10day),.counte
r_for_clock_month(clock_1month),.counter_for_clock_10month(clock_10month),
```

```
.pulse_period_10days(pulse_period_10days),.pulse_period_month(pulse_period_month),.pulse_
period_10month(pulse_period_10month),.pulse_period_year(pulse_period_year),
```

```
.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),
```

```
.b_9(b_9),.b_0(b_0),.up(up),.down(down));
```

```
wire pulse_period_decade; // becomes 1 once in a decade
```

```
wire pulse_period_century; // becomes 1 once in a century
```

```
wire pulse_period_millenia; // becoms 1 once in a millenia
```

```
// counters for years
```

```
clock_year_YY #(.reset(9),.which_digit(13))
```

```
YEARS(.CLK_for_year(pulse_period_year),.RST(RST),.counter_for_clock_year_or_dec_or_ce
n_mill(clock_year),.pulse_from_year(pulse_period_decade),
```

```
.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),
```

```
.b_9(b_9),.b_0(b_0),.up(up),.down(down));
```

```

clock_year_YY #(.reset(9),.which_digit(12))
DECADE(.CLK_for_year(pulse_period_decade),.RST(RST),.counter_for_clock_year_or_dec_o
r_cen_mill(clock_decade),.pulse_from_year(pulse_period_century),

.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),

.b_9(b_9),.b_0(b_0),.up(up),.down(down));

```

```

clock_year_YY #(.reset(9),.which_digit(11))
CENTURY(.CLK_for_year(pulse_period_century),.RST(RST),.counter_for_clock_year_or_dec
_or_cen_mill(clock_century),.pulse_from_year(pulse_period_millenia),

.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),

.b_9(b_9),.b_0(b_0),.up(up),.down(down));

```

```

clock_year_YY #(.reset(1),.which_digit(10))
MILLENNIA(.CLK_for_year(pulse_period_millenia),.RST(RST),.counter_for_clock_year_or_de
c_or_cen_mill(clock_millenia),

.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),

.b_9(b_9),.b_0(b_0),.up(up),.down(down));

```

//determining weekday

```

clock_week_day clock_week_daymodule00(.Is_Leap_Year(Is_Leap_Year),.CLK(CLK),

.clock_day(clock_day),.clock_10day(clock_10day),.clock_1month(clock_1month),.clock_10mo
nth(clock_10month),

.clock_year(clock_year),.clock_decade(clock_decade),.clock_century(clock_century),.clock_mil
lenia(clock_millenia),

.weekday(weekday));

```

// instantiating 7-segment BCD for time

```

wire ny_countdown;

```

```

assign ny_countdown =
(clock_10hour==2)&(clock_1hour==3)&(clock_10minute==5)&(clock_1minute==9)&(clock_1
0second==5)&(month==12)&(clock_day==1)&(clock_10day==3); // 31 december 23:59:50

//ss

Clock_digit_display #(1)
display_for_1second(.RST(RST),.digit_to_display(clock_second),.ny_countdown(ny_countdown),.display(display_clock_second));

Clock_digit_display #(0)
display_for_10second(.RST(RST),.digit_to_display(clock_10second),.ny_countdown(ny_countdown),.display(display_clock_10second));

//mm

Clock_digit_display #(0)
display_for_1minute(.RST(RST),.digit_to_display(clock_1minute),.ny_countdown(ny_countdown),.display(display_clock_1minute));

Clock_digit_display #(0)
display_for_10minute(.RST(RST),.digit_to_display(clock_10minute),.ny_countdown(ny_countdown),.display(display_clock_10minute));

//hh

Clock_digit_display #(0)
display_for_1hour(.RST(RST),.digit_to_display(clock_1hour),.ny_countdown(ny_countdown),.display(display_clock_1hour));

Clock_digit_display #(0)
display_for_10hour(.RST(RST),.digit_to_display(clock_10hour),.ny_countdown(ny_countdown),.display(display_clock_10hour));


//DAY

Clock_digit_display #(0)
display_for_day(.RST(RST),.digit_to_display(clock_day),.ny_countdown(ny_countdown),.display(display_clock_day));

Clock_digit_display #(0)
display_for_10day(.RST(RST),.digit_to_display(clock_10day),.ny_countdown(ny_countdown),.display(display_clock_10day));

```

```
//MONTH
```

```
Clock_digit_display #(0)  
display_for_1month(.RST(RST),.digit_to_display(clock_1month),.ny_countdown(ny_countdown),.display(display_clock_1month));
```

```
Clock_digit_display #(0)  
display_for_10month(.RST(RST),.digit_to_display(clock_10month),.ny_countdown(ny_countdown),.display(display_clock_10month));
```

```
//YERAR
```

```
Clock_digit_display #(0)  
display_for_millenia(.RST(RST),.digit_to_display(clock_millenia),.ny_countdown(ny_countdown),.display(display_clock_millenia));
```

```
Clock_digit_display #(0)  
display_for_century(.RST(RST),.digit_to_display(clock_century),.ny_countdown(ny_countdown),.display(display_clock_century));
```

```
Clock_digit_display #(0)  
display_for_decade(.RST(RST),.digit_to_display(clock_decade),.ny_countdown(ny_countdown),.display(display_clock_decade));
```

```
Clock_digit_display #(0)  
display_for_year(.RST(RST),.digit_to_display(clock_year),.ny_countdown(ny_countdown),.display(display_clock_year));
```

```
//weekday
```

```
Clock_digit_display_weekday  
Clock_digit_display_weekday00(.day_to_display(weekday),.display_I_letter(display_I_letter),.display_II_letter(display_II_letter),  
.display_III_letter(display_III_letter));
```

```
Endmodule
```

Module clock:

```
`timescale 1ns / 1ps
```

```
// This module takes clock with frequency that equals to the frequency of power supply
```

```

// it has parameter that specifies how many seconds should be the period of new clock
//(counter + 1)*2*clk_in_period this formula calculates period of output clock
//for 1s period, max_val of counter is 25-1
// for 10s period, max_val of counter is 250-1
// counter = 25*period - 1
module clock #(parameter period = 1)(input clk_in,
output clk_out);
reg [10:0] counter = 0; // when counter reaches max value, new clock signal will be toggled
reg tmp = 0; // this will assign to new clock
always @(posedge clk_in) begin
if(counter == (25*period - 1)) begin // when counter reaches that number, half period is over
counter = 0; // reset counter
tmp = ~tmp; // toggle tmp which will be output clock
end
else begin
counter = counter+1; // increment counter
end
end
assign clk_out=tmp; // tmp is output clock

endmodule

```

Module clock_digit_counters

```

`timescale 1ns / 1ps

// parameter which_digit determines whether it is hour, minute or second digit
// h10 - which_digit = 0
// h1 - which_digit = 1

```



```

// m10 - which_digit = 2
// m1 - which_digit = 3
// s10 - which_digit = 4
// s1 - which_digit = 5

module clock_digit_counters #(which_digit = 5)(
input RST, CLKk,
input [3:0] counter_in,check, //counter_in input time digit; check is the first hour digit, I need to
check it before incrementing the second hour digit
input set_time_enable,b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_0,up,down, //this is for changing
value
output reg [3:0] counter_out, // output time digit
output reg pulse); // clock for the time digit whose which_digit is current whichdigit-1

always @(posedge CLKk or posedge RST)begin
if(set_time_enable!=1) // if the user does not try to change time manually
    if(RST) begin // reset clock
        counter_out=0;
    end//if
    else if(which_digit==0) begin // for the first hour digit
        if(counter_in<2) begin // the first hour value changes from 0 to 2, because of 24 hour
format
            pulse=0; // becomes 1 if counter value is reseted to 0, which does not happen here
            counter_out=counter_in+1; // if first hour is 0 or 1, increment it
        end//else if
        else begin//
            pulse=1; // counter is reseted day changed so pulse is 1. This will be clock for day counter
            counter_out=0; //if the first hour is 2 and it should change, it resets to 0, because of 24 hour
format
        end//else
    end
end

```

```

end // else if for the first hour digit

else if(which_digit==1) begin // for the second hour digit

    if((counter_in<9)&check) begin // if the first hour digit <2, snd hour digit increases from 0
to 9.at this point time changes from 00 hours to 19 hours

        pulse=0; //becomes 1 if counter value is reseted to 0, which does not happen here

        counter_out=counter_in+1; // increment second hour digit counter

    end//else if

    else if((counter_in<3)) begin // at this point the first hour digit is 2, time is above 20 hours,
the second hour digit can change from 0 to 3. 24 hour format

        pulse=0; //becomes 1 if counter value is reseted to 0, which does not happen here

        counter_out=counter_in+1; // increment second hour digit counter

    end//else if

    else begin// mid night

        pulse=1; // new day, counter is reseted. this will be clock for the first hour digit

        counter_out=0; // reset counter

    end//else

end // else if for the second hour digit

else begin // for seconds and minutes

    if((counter_in<9)&&((which_digit==3)|(which_digit==5))) begin // if this is for the last
second or last minute digit, counter from 0 to 9

        pulse=0;

        counter_out=counter_in+1;

    end//else if

    else if((counter_in<5)&&((which_digit==2)|(which_digit==4))) begin // if this is for 10
second and 10 minute digit, counter from 0 to 5,

        pulse=0;

        counter_out=counter_in+1;

    end//else if

    else begin// after 10s

        pulse=1;

```

```

        counter_out=0;

        end//else

        end//else
end // always


wire [3:0] counter,new_value;


always @(new_value) begin // user entered new value
if((set_time_enable)&(counter==which_digit)) // the user tries to set time
counter_out=new_value; //output new value
end

// the following module allows user to set time.

clock_set_time
instance00(.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_
5),.b_6(b_6),.b_7(b_7),.b_8(b_8),
.b_9(b_9),.b_0(b_0),.up(up),.down(down),.counter(counter),.new_value(new_value));

Endmodule

```

Module clock_dd_mm

```

`timescale 1ns / 1ps


// this module contains code that counts days and months
// variablewhich_digit determines whether it is month, or day
// m10 - which_digit = 6
// m1 - which_digit = 7
// d10 - which_digit = 8
// d1 - which_digit = 9

```

```

module clock_mm_dd_counters(

input Is_Leap_Year,Is_February,Is_31_day,pulse_period_24hour,RST,// to identify if year is
leap, month is february, month has 30 or 31 days

input set_time_enable,b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_0,up,down,//this is for changing
value

output reg [3:0]
counter_for_clock_day,counter_for_clock_10day,counter_for_clock_month,counter_for_clock_
10month,//output time digit values

output reg pulse_period_10days, pulse_period_month, pulse_period_10month,
pulse_period_year);//output pulses


// this will count days

always @(posedge pulse_period_24hour or posedge RST) begin //
if(RST) begin
counter_for_clock_day=7; //reset to 7
end //if

else if(Is_February) begin // if month is february the second digit for days - d1, days change from
01 to 28 or 29(leap)

    if((counter_for_clock_10day<2)) begin // if the first digit of day is less than 2, it is not 20th
feb yet

        if(counter_for_clock_day<9) begin // d1 increases up to 9

            pulse_period_10days=0; // this pulse is for the first day digit - d10, and becomes 1 when 10
days is passed - d reset. This doesnot happen here

            counter_for_clock_day=counter_for_clock_day+1; // increment day. at this point day
changes from 01 to 19

            end//if

        else begin

            pulse_period_10days=1; // second digit is reset, this pulse will be clock for the first day
digit counter

            counter_for_clock_day=0; //if the first digit for days < 2 and the second digit is 9, reset the
second digit to 0.

```

```

        end//else
    end//if
else begin // if it is february and date is more than 20
    if(counter_for_clock_day<(8+Is_Leap_Year)) begin //d can count up to 8 or 9(leap)
        pulse_period_10days=0;
        counter_for_clock_day=counter_for_clock_day+1;
    end//if
    else begin
        pulse_period_10days=1;// day is reset, so set pulse for 10day digit counter
        counter_for_clock_day=1;// at this point the month changed, so i need to set the second day
        digit to 1. day starts at 01.
    end//else
end//else
end//else if
else if(Is_31_day) begin // month has 31 days
    if((counter_for_clock_10day<3)) begin // day is or is less that 30
        if(counter_for_clock_day<9) begin // the second digit for day increases till 9.
            pulse_period_10days=0; // counter does not reset, so pulse is 0
            counter_for_clock_day=counter_for_clock_day+1; // increase counter. at tis point day
            changes from 01 to 29
        end//if
        else begin// day is above
            pulse_period_10days=1; // //set pulse which will be used for counter of the first digit of day
            - d10
            counter_for_clock_day=0; // reset counter, because the last digit increase from 9 to 0.
        end//else
    end//if
    else begin // day is 30 or 31
        if(counter_for_clock_day<1) begin // if day is 30

```

```

    pulse_period_10days=0; //
    counter_for_clock_day=counter_for_clock_day+1; // next day is 31
end//if

else begin // if day is 31
    pulse_period_10days=1; // the first digit of day d10 should change, so pulse becomes 1
    counter_for_clock_day=1; // we have a new month, that starts at 01. That is why reset to 1,
not to 0.

    end//else

end//else

end//else if

else begin // month has 30 days

    if(counter_for_clock_day<9) begin //counter is less than 9

        pulse_period_10days=0; // no pulse for the first digit of days yet
        counter_for_clock_day=counter_for_clock_day+1; // increment last digit of day
    end//if

    else begin // if day is 09 or 19 or 29

        pulse_period_10days=1; // pulse for the first digit of day

        counter_for_clock_day=0; //set the last digit for day to 0 // I do not have 31 days, which is
followed by 01. I do not have two consecutive 1s

        end//else

    end//else

end //always

// this will count ten days/

always @(posedge pulse_period_10days or posedge RST) begin

    if(RST) begin

        counter_for_clock_10day=0; // reset to 0

    end //if

    else if(Is_February) begin // month is february

```

```

    if(counter_for_clock_10day<2) begin // the first digit for days changes from 0 to 2. Day is
below 20
        pulse_period_month=0; // this will be clock for always of the last digit of month
        counter_for_clock_10day=counter_for_clock_10day+1; //increase 10days - the first digit of
day
    end//if
    else begin // day is 28(9)
        pulse_period_month=1; //month changed. Spring has come
        counter_for_clock_10day=0;//that is why the first digit became 0.
    end//else
end // else if
else begin // not february
    if(counter_for_clock_10day<3) begin // if day is less than 30
        pulse_period_month=0; // month should not change yet
        counter_for_clock_10day=counter_for_clock_10day+1;//increment the first digit of day
    end//if
    else begin // day is 30 or 31
        pulse_period_month=1; // month should change
        counter_for_clock_10day=0; // set the first digit of month to 0. because we have a new month.
    end//else // there is a problem if month has only 30 days. after 30 day, the first digit does not
resets. The following always corrects it
end // else //
end//always

always @(negedge pulse_period_10days) begin // this marks end of month with 30 days.
    if((counter_for_clock_10day==3)&(~Is_31_day)) begin/// day was 30 and needs to change,
month has 30 days
        counter_for_clock_10day=0; // new month, the first digit of day is 0
        pulse_period_month=1; // new month
    end
end

```

end

// this will count month////////

always @(posedge pulse_period_month or posedge RST) begin

if(RST) begin

counter_for_clock_month=1; // reset to 1

end//if

else if(counter_for_clock_10month<1) begin // month is less than 10, it is not october yet

if(counter_for_clock_month<9) begin // it is not september yet

pulse_period_10month=0; // this will be clock for the first digit of month

counter_for_clock_month=counter_for_clock_month+1; // month changes from 01 to 09 (jan-aug)

end//if

else begin

pulse_period_10month=1; // it was september and now it should become october

counter_for_clock_month=0; // october has 0 for the last digit - 10

end//else

end//else if

else begin // month is 10, 11 or 12

if(counter_for_clock_month<2) begin // month is 10 or 11

pulse_period_10month=0;

counter_for_clock_month=counter_for_clock_month+1; // increment the last digit of month

end//if

else begin // it is december

pulse_period_10month=1; // the first digit of month should change

counter_for_clock_month=1; // new year, january is next - 01

end//else

end//else

end // always


```

// this will count 10months
always @(posedge pulse_period_10month or posedge RST) begin
if(RST) begin
counter_for_clock_10month=1;    ////////////reset to 0
end//if

else if(counter_for_clock_10month<1) begin // month is less than 10
counter_for_clock_10month=counter_for_clock_10month+1; // month becomes octomber
pulse_period_year=0; // this will be used as a clock for the least significant digit of year. there is
no new year yet
end//elseif

else begin // new year
counter_for_clock_10month=0;
pulse_period_year=1;
end//else
end//always

```

```

wire [3:0] counter,new_value;

```

```

always @(new_value) begin
if((set_time_enable)&(counter==6)) // month_10
counter_for_clock_10month=new_value;
else if((set_time_enable)&(counter==7)) // month
counter_for_clock_month=new_value;
else if((set_time_enable)&(counter==8)) // day_10
counter_for_clock_10day=new_value;
else if((set_time_enable)&(counter==9)) // day

```

```
counter_for_clock_day=new_value;  
end
```

```
clock_set_time  
instance10(.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_  
5),.b_6(b_6),.b_7(b_7),.b_8(b_8),  
.b_9(b_9),.b_0(b_0),.up(up),.down(down),.counter(counter),.new_value(new_value));
```

```
Endmodule
```

Module clock_year_YY

```
`timescale 1ns / 1ps
```

```
// parameter which_digit determines whether it is month, or day
```

```
// millenia - which_digit = 10
```

```
// century - which_digit = 11
```

```
// decade - which_digit = 12
```

```
// year - which_digit = 13
```

```
module clock_year_YY #(parameter reset=0,which_digit=0)(
```

```
input CLK_for_year, RST,
```

```
input set_time_enable,b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_0,up,down,//this is for changing  
value
```

```
output reg [3:0] counter_for_clock_year_or_dec_or_cen_mill,
```

```
output reg pulse_from_year);
```

```
always @(posedge CLK_for_year or posedge RST) begin
```

```

if(RST) begin
counter_for_clock_year_or_dec_or_cen_mill = reset; //reset to 0
end // if

else if(counter_for_clock_year_or_dec_or_cen_mill<9) begin
counter_for_clock_year_or_dec_or_cen_mill=counter_for_clock_year_or_dec_or_cen_mill+1; //
increment counter

pulse_from_year=0; // no need for pulse
end//else if

else begin
counter_for_clock_year_or_dec_or_cen_mill=0; // ten unit is past
pulse_from_year=1; // give pulse
end//else

end//always

```

```

wire [3:0] counter,new_value;

```

```

always @(new_value) begin // this helps user set new value
if((set_time_enable)&(counter==which_digit))
counter_for_clock_year_or_dec_or_cen_mill=new_value;
end

```

```

clock_set_time
instance01(.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_
5),.b_6(b_6),.b_7(b_7),.b_8(b_8),
.b_9(b_9),.b_0(b_0),.up(up),.down(down),.counter(counter),.new_value(new_value));

```

```

Endmodule

```

Module clock_set_time

```
`timescale 1ns / 1ps

// b_n represents nth button on keyboard
// when set_time_enable is 1, this module will give values to time components
// up increases counter, down decreases

module clock_set_time(

input set_time_enable,b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_0,up,down,// keyboard
output reg [3:0] counter,new_value); // counte for position of input digit; new_value will be
assigned to digit we want to change


always @(set_time_enable) // when we deside to change time, counter becomes 0
counter=0;


always @(negedge up) // if we press up button counter increases
if(counter<13)    // counter cant be more than 13
counter=counter+1;


always @(negedge down) // if we press down button counter decreases
if(counter>0)      // counter cant be less than 0
counter=counter-1;


always @(negedge b_1) // pushing button 1 writes 1 in new value
new_value = 1;
always @(negedge b_2) // pushing button 2 writes 2 in new value
new_value = 2;
always @(negedge b_3) // pushing button 3 writes 3 in new value
new_value = 3;
always @(negedge b_4) // pushing button 4 writes 4 in new value
```

```

new_value = 4;
always @(negedge b_5) // pushing button 5 writes 5 in new value
new_value = 5;
always @(negedge b_6) // pushing button 6 writes 6 in new value
new_value = 6;
always @(negedge b_7) // pushing button 7 writes 7 in new value
new_value = 7;
always @(negedge b_8) // pushing button 8 writes 8 in new value
new_value = 8;
always @(negedge b_9) // pushing button 9 writes 9 in new value
new_value = 9;
always @(negedge b_0) // pushing button 0 writes 0 in new value
new_value = 0;

endmodule

```

module leap_year_detector

```

`timescale 1ns / 1ps

```

```

module leap_year_detector(
input [3:0] millenia, century, decade, year,
output reg leap_year);

```

```

wire [6:0] upper2;

```

```

wire [6:0] lower2;

```

```

assign lower2=decade*10+year; // last two digits of year if year is 1234, lower2 is 23

```

```
assign upper2=millenia*10+century; // first two digts of year. if year is 1234, upper2 is 12
```

```
always @(*) begin // if number is divisible by 4, its last two digits are 00
```

```
if((decade==0)&(year==0)) begin // if the year is century year, 1900,12800...,upper2 should be  
divisible by 4
```

```
    leap_year = upper2[1:0]==(2'b00); // leap year
```

```
end
```

```
else if(lower2[1:0]==(2'b00)) begin // not century year, if low2 is divisible by 4
```

```
    leap_year = 1; // leap year
```

```
end
```

```
else begin
```

```
    leap_year=0; // no leap year
```

```
end
```

```
end
```

```
endmodule
```

Module clock_weekday

```
`timescale 1ns / 1ps
```

```
// Determine weekday using John H Conway's algorithm
```

```
// each day has its own number
```

```
// sun 0 |mon 1| tue 2 |wed 3 |th 4 |fr 5 |sat - 6
```

```
module clock_week_day(
```

```
    input Is_Leap_Year,CLK, // clock signal and signal telling if year is leap
```

```
    input [3:0]
```

```
    clock_day,clock_10day,clock_1month,clock_10month,clock_year,clock_decade,clock_century,  
    clock_millenia,
```

```
    output [2:0] weekday);// output
```

```
wire [4:0] day; // this represents day
```

```
wire [4:0] month; // this represents month
```

```
wire [6:0] year_2_least_signif; // this is last two digits of year 34 out of 1234
```

```
wire [6:0] year_2_most_signif; // this is most two digits of year 12 out of 1234
```

```
assign day = clock_day + clock_10day*10;
```

```
assign month = clock_1month + clock_10month*10;
```

```
assign year_2_least_signif = clock_year + clock_decade*10;
```

```
assign year_2_most_signif = clock_century + clock_millenia*10;
```

```
wire [2:0] century_code; // this is century code
```

```
assign century_code = // choosing doomsdays based on century
```

```
(year_2_most_signif[1:0]==0) ? 2: // if last two bits of year_2_most_signif is 0, century code is  
2
```

```
(year_2_most_signif[1:0]==1) ? 0: // if last two bits of year_2_most_signif is 1, century code is  
0
```

```
(year_2_most_signif[1:0]==2) ? 5: // if last two bits of year_2_most_signif is 2, century code is  
5
```

```
(year_2_most_signif[1:0]==3) ? 3: 2'bx; // if last two bits of year_2_most_signif is 3, century  
code is 3
```

```
wire [6:0] division_of_year_2_least_signif_by_12; // this is us an integer part of ratio of  
year_2_least_signif and 12
```

```
wire [6:0] remainder_of_division_of_year_2_least_signif_by_12; // this is us an remainder of  
ratio of year_2_least_signif and 12
```

```
clock_division_remainder
```

```
clock_division_remainder00(.CLK(CLK),.dividend(year_2_least_signif),.divisor(12),
```

```
.quotient(division_of_year_2_least_signif_by_12),.remainder(remainder_of_division_of_year_2_
_least_signif_by_12)); // year by 12
```

```
//year_2_least_signif
```

```
wire [6:0] qoution_of_remainder_by_4;// this is us an integer part of ratio of
remainder_of_division_of_year_2_least_signif_by_12 and 4
```

```
assign qoution_of_remainder_by_4 = remainder_of_division_of_year_2_least_signif_by_12>>2;
```

```
wire [7:0] sum; // this of all four things calculated above
```

```
assign sum = century_code + division_of_year_2_least_signif_by_12 +
remainder_of_division_of_year_2_least_signif_by_12 + qoution_of_remainder_by_4;
```

```
wire [2:0] Doomsday_of_year; // this is doomsday of the year, remainder of sum/7
```

```
clock_division_remainder clock_division_remainder01(.CLK(CLK),.dividend(sum),.divisor(7),
.remainder(Doomsday_of_year)); // find doomsday of the year
```

```
reg [4:0] Doomsday_of_month; // this table stores which dates are doomsdays in each month
```

```
always @(month) begin
```

```
case(month)
```

```
1: Doomsday_of_month <= 3 + Is_Leap_Year;
```

```
2: Doomsday_of_month <= 28 + Is_Leap_Year;
```

```
3: Doomsday_of_month <= 14;
```

```
4: Doomsday_of_month <= 4;
```

```
5: Doomsday_of_month <= 9;
```

```
6: Doomsday_of_month <= 6;
```

```
7: Doomsday_of_month <= 11;
```

```
8: Doomsday_of_month <= 8;
```

```
9: Doomsday_of_month <= 5;
```



```

10: Doomsday_of_month <= 10;
11: Doomsday_of_month <= 7;
12: Doomsday_of_month <= 12;
default: Doomsday_of_month <= 'bx;
endcase
end

```

reg [6:0] difference; // difference of Doomsday_of_year and day. Only value without sign. This is always positive

```

//wire [2:0] remainder;

always @(Doomsday_of_year or day) begin
if(Doomsday_of_month>day)
difference = Doomsday_of_month-day+Doomsday_of_year;
else
difference = day-Doomsday_of_month+Doomsday_of_year;
end

```

```

// remainder of difference / 7 is weekday

clock_division_remainder
clock_division_remainder10(.CLK(CLK),.dividend(difference),.divisor(7),
.remainder(weekday));

```

Endmodule

Module clock_division_remainder

```

`timescale 1ns / 1ps

// dividend is to be divided by divisor

```

```

module clock_division_remainder(
input CLK,
input [6:0] dividend,divisor,
output reg [6:0] quotient, remainder);

reg [6:0] first_term, second_term,result;// dividend, divisor, result of division

always @(dividend or divisor) begin //whenever we have new dividend or divisor
first_term = dividend;
second_term = divisor;
result = 0;
end

always @(CLK) begin
if(dividend<divisor) begin // if dividend<divisor, quotient is 0 and remainder is dividend
quotient = 0;
remainder = dividend;
end//if
else if(dividend==divisor) begin // if they equal, quotient is 1 and remainder 0
quotient = 1;
remainder = 0;
end//els if
else begin
    if(first_term>second_term) begin // subtract divisor from dividend untill dividend becomes less
or equal to dividend
        first_term = first_term - second_term;
        result = result+1; // after each difference, increase result.
    end
end
end

```

```
    else if(first_term==second_term) begin // if dividend becomes the same as divisor, dividend is
divisible by divisor
```

```
        quotient = result+1; // result is result so far + 1
```

```
        remainder = 0; // remainder 0
```

```
    end
```

```
    else begin
```

```
        quotient = result; // output quotient
```

```
        remainder = first_term; // output remainder
```

```
    end
```

```
end
```

```
end
```

```
endmodule
```

Module clock_digit_display

```
`timescale 1ns / 1ps
```

```
module Clock_digit_display #(parameter Second_Digit = 0)(
```

```
input [3:0] digit_to_display,
```

```
input ny_countdown,RST,
```

```
output reg [6:0] display);
```

```
// abcdefg
```

```
always @(*) begin
```

```
if(RST) display=7'b1111110;
```

```
else if(ny_countdown==0) begin // usual day
```

```
case(digit_to_display)
```

```
0: display = 7'b1111110;
```

```
1: display = 7'b0110000;
```

```
2: display = 7'b1101101;
```

```

3: display = 7'b1111001;
4: display = 7'b0110011;
5: display = 7'b1011011;
6: display = 7'b0111101;
7: display = 7'b1110000;
8: display = 7'b1111111;
9: display = 7'b1110011;
endcase
end //else if
else if(ny_countdown&&Second_Digit) begin // ten minutes before NY and second digit
case(10-digit_to_display)
0: display = 7'b1111110;
1: display = 7'b0110000;
2: display = 7'b1101101;
3: display = 7'b1111001;
4: display = 7'b0110011;
5: display = 7'b1011011;
6: display = 7'b0111101;
7: display = 7'b1110000;
8: display = 7'b1111111;
9: display = 7'b1110011;
endcase
end // esle if
else begin // ten minutes before NY and not second digit
display=0;
end//else
end // always
endmodule

```

Module clock_digit_display_weekday

```
`timescale 1ns / 1ps
```

```
// this is module for lighting up LED for weekday
```

```
// SUN - 0; M - 1; T - 2; W - 3; TH - 4; F - 5; SAT - 6
```

```
module Clock_digit_display_weekday(
```

```
input [2:0] day_to_display,
```

```
output reg [19:0] display_I_letter,
```

```
output reg [9:0] display_II_letter,display_III_letter);
```

```
// for the first letter
```

```
always @(*) begin
```

```
case(day_to_display)
```

```
//          abcdefghijklmnopqrst
```

```
0: display_I_letter = 20'b11001000010000010111; // S
```

```
1: display_I_letter = 20'b00101000000110110000; // M
```

```
2: display_I_letter = 20'b11000001000001000000; // T
```

```
3: display_I_letter = 20'b00010110110000001000; // W
```

```
4: display_I_letter = 20'b11000001000001000000; // T
```

```
5: display_I_letter = 20'b11000000000100010110; // F
```

```
6: display_I_letter = 20'b11001000010000010111; // S
```

```
default: display_I_letter=0;
```

```
endcase
```

```
end
```

```
//// for the second letter
```

```
//SUN TH SAT
```

```

always @(*) begin
case(day_to_display)
//          abcdefghij
0: display_II_letter = 10'b1111100000; //U
4: display_II_letter = 10'b1110100001; //H
6: display_II_letter = 10'b00000011111; //A
default: display_II_letter=0;
endcase
end

// SUN - 0; M - 1; T - 2; W - 3; TH - 4; F - 5; SAT - 6

//// for the third letter
//SUN SAT
always @(*) begin
case(day_to_display)
//          abcdefghij
0: display_III_letter = 10'b1001110101; // N
6: display_III_letter = 10'b0110001010; // T
default: display_III_letter=0;
endcase
end
endmodule

```

Dishwasher / Cooker

Top_Module_for_dishwasher_cooker

```

`timescale 1ns / 1ps

// top module for dishwasher

```

```

module Top_Module_For_Dishwasher(

input cloc1_for_dishwasher, POWER, START,RST_For_Dishwasher, // power and start button
generates impulses

input
b_1_for_dishwasher,b_2_for_dishwasher,b_3_for_dishwasher,b_4_for_dishwasher,b_5_for_dish
washer,b_6_for_dishwasher,b_7_for_dishwasher,b_8_for_dishwasher,b_9_for_dishwasher,b_0_
for_dishwasher,

output [3:0] dishwasher_10hours,dishwasher_hours,dishwasher_10minutes,dishwasher_minutes,

output [6:0]
dishwasher_10hours_display,dishwasher_hours_display,dishwasher_10minutes_display,dishwas
her_minutes_display);

wire pulse,power_on,start_begin; // if poweron is 1 - device is turned on, if start_begin is 1, the
device xdoes its job

wire [2:0] position_of_digit; // which time digit os the user input

wire [3:0] timer_value; // what is user input


//keyboard for dishwasher

dishwasher_cooker_keyboard
dishwasher_cooker_keyboard00(.POWER(POWER),.START(START),.RST_For_Dishwasher(
RST_For_Dishwasher),

.b_1_for_dishwasher(b_1_for_dishwasher),.b_2_for_dishwasher(b_2_for_dishwasher),.b_3_for_
dishwasher(b_3_for_dishwasher),

.b_4_for_dishwasher(b_4_for_dishwasher),.b_5_for_dishwasher(b_5_for_dishwasher),

.b_6_for_dishwasher(b_6_for_dishwasher),.b_7_for_dishwasher(b_7_for_dishwasher),

.b_8_for_dishwasher(b_8_for_dishwasher),.b_9_for_dishwasher(b_9_for_dishwasher),.b_0_for_
dishwasher(b_0_for_dishwasher),

.pulse(pulse),.power_on(power_on),.start_begin(start_begin),.position_of_digit(position_of_digit
),.timer_value(timer_value),

.finish(finish));

```

```
wire pulse_1m,pulse_10m,pulse_1h,pulse_10h;// rising edges are 1m, 10m, 1h, 10h away from each other
```

```
clock #(60) clk10(.clk_in(clk_for_dishwasher),.clk_out(pulse_1m)); // 60s
```

```
reg [3:0]
```

```
dishwasher_for_10hours,dishwasher_for_hours,dishwasher_for_10minutes,dishwasher_for_minutes;// set values for timers
```

```
wire finish_10h,finish_h,finish_10m,finish_m; // is 1 if h10, h, 10m, m finishes counting respectively
```

```
wire finish; // this is 1, when timer counts down to 0. this impulse cuts off the power
```

```
assign finish =finish_10h&finish_h&finish_10m&finish_m; // finish is 1, kill the power
```

```
// h10=0; h=1; m10=2; m=3
```

```
dishwasher_timer #(3)
```

```
minute(.clockk(pulse_1m),.power_on(power_on),.start_begin(start_begin),.timer_value_start(dishwasher_for_minutes),
```

```
.pulse(pulse_10m),.timer_value_current(dishwasher_minutes),
```

```
.previous_digit1(dishwasher_10hours),.previous_digit2(dishwasher_hours),.previous_digit3(dishwasher_10minutes),.finish(finish_m));// for minute
```

```
dishwasher_timer #(2)
```

```
minute_10(.clockk(pulse_10m),.power_on(power_on),.start_begin(start_begin),.timer_value_start(dishwasher_for_10minutes),
```

```
.pulse(pulse_1h),.timer_value_current(dishwasher_10minutes),
```

```
.previous_digit1(dishwasher_10hours),.previous_digit2(dishwasher_hours),.finish(finish_10m));// for 10minute
```

```
dishwasher_timer #(1)
```

```
hour(.clockk(pulse_1h),.power_on(power_on),.start_begin(start_begin),.timer_value_start(dishwasher_for_hours),
```

```
.pulse(pulse_10h),.timer_value_current(dishwasher_hours),
```

```
.previous_digit1(dishwasher_10hours),.finish(finish_h)); // for hour
```



```

dishwasher_timer #(0)
hour_10(.clockk(pulse_10h),.power_on(power_on),.start_begin(start_begin),.timer_value_start(d
ishwasher_for_10hours),

.timer_value_current(dishwasher_10hours),.finish(finish_10h)); // for 10 hour

always @(negedge pulse) begin // giving values for timer
    if(position_of_digit==1) // the first hour digit is entered
        dishwasher_for_10hours=timer_value;

    else if(position_of_digit==2) // the second hour digit is entered
        dishwasher_for_hours=timer_value;

    else if(position_of_digit==3) // third hour digit is entered
        dishwasher_for_10minutes=timer_value;

    else if(position_of_digit==4) // fourth hour digit is entered
        dishwasher_for_minutes=timer_value;

    else begin
        dishwasher_for_10hours=0;
        dishwasher_for_hours=0;
        dishwasher_for_10minutes=0;
        dishwasher_for_minutes=0;
    end
end

//display

```

```
Clock_digit_display
Clock_digit_display00(.digit_to_display(dishwasher_10hours),.ny_countdown(~power_on),.RST(Power),
    .display(dishwasher_10hours_display)); // display 10h
```

```
Clock_digit_display
Clock_digit_display01(.digit_to_display(dishwasher_hours),.ny_countdown(~power_on),.RST(Power),
    .display(dishwasher_hours_display)); // display h
```

```
Clock_digit_display
Clock_digit_display10(.digit_to_display(dishwasher_10minutes),.ny_countdown(~power_on),.RST(Power),
    .display(dishwasher_10minutes_display)); // display 10m
```

```
Clock_digit_display
Clock_digit_display11(.digit_to_display(dishwasher_minutes),.ny_countdown(~power_on),.RST(Power),
    .display(dishwasher_minutes_display)); // display m
```

```
Endmodule
```

Module dishwasher_cooker_keyboard

```
`timescale 1ns / 1ps
```

```
// keyboard for power, start and digit buttons.
```

```
module dishwasher_cooker_keyboard(
```

input POWER, START,RST_For_Dishwasher,finish, // power and start button generates impulses, finish impulse kills power

input

b_1_for_dishwasher,b_2_for_dishwasher,b_3_for_dishwasher,b_4_for_dishwasher,b_5_for_dishwasher,b_6_for_dishwasher,b_7_for_dishwasher,b_8_for_dishwasher,b_9_for_dishwasher,b_0_for_dishwasher,

output reg pulse,power_on,start_begin, // if poweron is 1 - device is turned on, if start_begin is 1, the device xdoes its job

output reg [2:0] position_of_digit,

output reg [3:0] timer_value);

always @(posedge finish)// kill the power, timer is done counting

power_on=0;

always @(posedge RST_For_Dishwasher) begin // reset everythin upon installation - MAIN RST

pulse=0;

position_of_digit=0;

timer_value=0;

power_on=0;

start_begin=0;

end

always @(posedge POWER) begin // pressing power button set timer to 00:00.

position_of_digit=0; // wait to enter the first our digit

timer_value=0;

pulse=0; // no pulse

start_begin=0; // do not start untill start is pressed

end

```
always @(posedge POWER) // when power is pressed, poweron toggles
power_on=~power_on;
```

```
always @(negedge power_on) // no power, do not work
start_begin=0;
```

```
always @(START) begin // start button starts machine
start_begin=~start_begin;
position_of_digit=0;
end
```

```
always
@(b_1_for_dishwasher,b_2_for_dishwasher,b_3_for_dishwasher,b_4_for_dishwasher,b_5_for_
dishwasher,b_6_for_dishwasher,b_7_for_dishwasher,b_8_for_dishwasher,b_9_for_dishwasher,b
_0_for_dishwasher)
pulse=~pulse;// negedge of pulse after key is pressed
```

```
always @(posedge b_1_for_dishwasher) begin // 1 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 1;
end
```

```
always @(posedge b_2_for_dishwasher) begin // 2 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 2;
end
```

```
always @(posedge b_3_for_dishwasher) begin // 3 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 3;
end
```

```
always @(posedge b_4_for_dishwasher) begin // 4 is pressed
```

```
position_of_digit = position_of_digit + 1;
timer_value = 4;
end

always @(posedge b_5_for_dishwasher) begin // 5 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 5;
end

always @(posedge b_6_for_dishwasher) begin // 6 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 6;
end

always @(posedge b_7_for_dishwasher) begin // 7 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 7;
end

always @(posedge b_8_for_dishwasher) begin // 8 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 8;
end

always @(posedge b_9_for_dishwasher) begin // 9 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 9;
end

always @(posedge b_0_for_dishwasher) begin // 0 is pressed
position_of_digit = position_of_digit + 1;
timer_value = 0;
end
```

```
endmodule
```

Module dishwasher_timer

```
`timescale 1ns / 1ps
```

```
// h10=0; h=1; m10=2; m=3
```

```
module dishwasher_timer #(parameter whichdigit=0)(
```

```
input clockk,power_on,start_begin,
```

```
input [3:0] previous_digit1,previous_digit2,previous_digit3,
```

```
input [3:0] timer_value_start,
```

```
output reg pulse,finish, // finish pulse tells that this digit finished counting
```

```
output reg [3:0] timer_value_current);
```

```
always @(posedge start_begin) // when we press start, finish signal becomes 0.
```

```
finish=0;
```

```
always @(timer_value_start) begin// when start value is set
```

```
timer_value_current = timer_value_start; // this is the starting point for that particular digit of  
timer
```

```
pulse=0; // initialize pulse at 0, this will be used as a clock for the time digit standing left to this  
particular digit
```

```
if(timer_value_start==0) // if 0 is the starting point for that particular digit
```

```
finish=1;
```

```
else
```

```
finish=0; // when timer counts down to 0, finish is set to 1
```

```
end
```

```
always @(posedge clockk) begin // begin countdown
```

```

if((power_on)) begin
    if(timer_value_current>0) begin
        timer_value_current=timer_value_current-1; // decrease timer_value_current till 0
        pulse=0;
    end
    else if(whichdigit==0) begin // timer-value is 0 , digit is h10
        timer_value_current=0;
        pulse=1;
        finish=1;//finish
    end
    else if((whichdigit==1)&(previous_digit1==0)) begin // timer-value is 0 , h10 is also 0, digit is
h, stop counting
        timer_value_current=0;
        pulse=1;
        finish=1;//finish
    end
    else if((whichdigit==2)&(previous_digit1==0)&(previous_digit2==0)) begin // timer-value is
0 , previous digits are 0s,digit is m10, stop conting
        timer_value_current=0;
        pulse=1;
        finish=1;//finish
    end
    else if((whichdigit==3)&(previous_digit1==0)&(previous_digit2==0)&(previous_digit3==0))
begin // timer-value is 0 , preivous digits are 0s, digit is m, stop counting
        timer_value_current=0;
        pulse=1;
        finish=1;//finish
    end
    else if((whichdigit==2)) begin // should not stop counting, if digit is for 10minutes, 5 is after 0
because there are 60 minutes in an hour

```

```

    timer_value_current=5;
    pulse=1;
    end

    else begin // should not stop counting, the instance is not for 10m, so after 0 the following
number is 9
        timer_value_current=9;
        pulse=1;
        end
    end
end
else
timer_value_current=0; // power is off, reset to 0
end
endmodule

```

Lock

Top module for Lock

```
`timescale 1ns / 1ps
```

```

module Top_Module_For_lock(
//input [3:0] input_number, // this will be changed by the keyboard
input
b_1_for_lock,b_2_for_lock,b_3_for_lock,b_4_for_lock,b_5_for_lock,b_6_for_lock,b_7_for_loc
k,b_8_for_lock,b_9_for_lock,b_0_for_lock,// keyboard
input CLK_For_Lock,RST_For_Lock,OK,door_is_open,// door_is_open is also write enable - it
is 1 when door is physically open
output unlock);// it is 1 when lock is unlocked

wire [3:0] input_number; // this is pressed digit
wire [2:0] position_of_number; // this the position of pressed number in combination

```



```
wire pulse_after_input; //input digit is checked after that pulse.
```

```
Lock_Sequence_Detector
```

```
Lock_Sequence_Detector00(.input_number(input_number),.CLK_For_Lock(CLK_For_Lock),  
.RST_For_Lock(RST_For_Lock),.OK(OK),.unlock(unlock),.door_is_open(door_is_open),  
.pulse_after_input(pulse_after_input),.position_of_number(position_of_number)); // this  
determines if input combination is correct
```

```
//keypad
```

```
Lock_keypad Lock_keypad00(.RST_For_Lock(RST_For_Lock),.door_is_open(door_is_open),  
.b_1_for_lock(b_1_for_lock),.b_2_for_lock(b_2_for_lock),.b_3_for_lock(b_3_for_lock),.b_4_for_lock(b_4_for_lock),.b_5_for_lock(b_5_for_lock),  
.b_6_for_lock(b_6_for_lock),.b_7_for_lock(b_7_for_lock),.b_8_for_lock(b_8_for_lock),.b_9_for_lock(b_9_for_lock),.b_0_for_lock(b_0_for_lock),  
.input_number(input_number),.pulse_after_input(pulse_after_input),.position_of_number(position_of_number));
```

```
Endmodule
```

Module lock_sequence_detector

```
`timescale 1ns / 1ps
```

```
//This gives unlock the lock if correct combination is entered
```

```
// RST resets the state to s0, locks the door and makes counter =0
```

```
// lock is unlocked, unlock = 1
```

```
// door_is_open shows if the door is physically closed or open. when open, the variable is 1.  
whenever door closes negedge, door locks itself
```

```
module Lock_Sequence_Detector(
```

```

input [3:0] input_number, //
input CLK_For_Lock,RST_For_Lock,OK,door_is_open, //
input [2:0] position_of_number,
input pulse_after_input,
output reg unlock);

parameter s0=0,s1=1,s2=2,s3=3,s4=4,s5=5;

reg [2:0] state;
reg [2:0] counter; // this should show the order of input number. if the first number is entered,
counter=0

        // if the second digit is entered counter =1, etc.

always @(negedge pulse_after_input or negedge OK or posedge RST_For_Lock) begin//
whenever input number changes, or a number is typed ERROR use negedge pulse_after_input
if(RST_For_Lock) begin // reset
state<=s0; // go to reset state
counter<=0; // wait for the first digit to enter
unlock<=0; // lock the door
end // if
else if(door_is_open==0) begin
case(state)
s0: if((input_number==combination[1])&(counter==0)) begin // correct value is entered for the
first digit
state<=s1; // the first digit is correct
counter<=counter+1; // increase counter - 1 digit is already entered
unlock<=0; // do not unlock the door
end
else begin // if incorrect number is entered or the previous values were incorrect
state<=s0; // go to reset state

```

```
unlock<=0; // do not unlock the door  
if(counter<4)  
    counter<=counter+1; // increase counter  
else  
    counter<=0; // reset counter  
end
```

s1: if((input_number==combination[2])&(counter==1)) begin // correct value is entered for the second digit

```
    state<=s2; // the second digit is correct  
    counter<=counter+1; // increase counter - 2 digit is already entered  
    unlock<=0; // do not unlock the door  
end  
else begin //if incorrect number is entered  
    state<=s0; // go to reset state  
    counter<=counter+1; // increase counter  
    unlock<=0; // do not unlock the door  
end
```

s2: if((input_number==combination[3])&(counter==2)) begin // correct value is entered for the third digit

```
    state<=s3; // the third digit is correct  
    counter<=counter+1; // increase counter - 3 digit is already entered  
    unlock<=0; // do not unlock the door  
end  
else begin //if incorrect number is entered  
    state<=s0; // go to reset state  
    counter<=counter+1; // increase counter  
    unlock<=0; // do not unlock the door  
end
```

```
s3: if((input_number==combination[4])&(counter==3)) begin // correct value is entered for the fourth digit
```

```
    state<=s4; // the fourth digit is correct
```

```
    counter<=counter+1; // increase counter - 4 digit is already entered
```

```
    unlock<=0; // do not unlock the door
```

```
end
```

```
else begin //if incorrect number is entered
```

```
    state<=s0; // go to reset state
```

```
    counter<=counter+1; // increase counter
```

```
    unlock<=0; // do not unlock the door
```

```
end
```

```
s4: begin
```

```
    state<=s0; // go to reset state
```

```
    unlock<=1; // do unlock the door
```

```
    counter<=0; // reset counter
```

```
end
```

```
default: begin
```

```
    state<=s0; // go to reset state
```

```
    counter<=0; // wait for the first digit to enter
```

```
    unlock<=0; // lock the door
```

```
end
```

```
endcase
```

```
end // else
```

```
end//always
```

```
reg [3:0] combination [1:4]; // this array stores correct combination
```

```
always @(negedge pulse_after_input) begin // updates combination
```

```
if(door_is_open) begin// when we are allowed to change combination
```

```

combination[position_of_number]=input_number;
end
end

always @(negedge door_is_open) begin // whenever door physically closes, lock
if(counter==0) begin
unlock<=0;
end
end
endmodule

```

Module lock_keypad

```

`timescale 1ns / 1ps

// This gives us an impulse when any key is pressed
// gives us the value of the key which is pressed
// tells us which digit of the combination corresponds to entered value
module Lock_keypad(
input RST_For_Lock,door_is_open,
input
b_1_for_lock,b_2_for_lock,b_3_for_lock,b_4_for_lock,b_5_for_lock,b_6_for_lock,b_7_for_loc
k,b_8_for_lock,b_9_for_lock,b_0_for_lock,
output reg [3:0] input_number, // this is the pressed number
output reg [2:0] position_of_number, // this the position of pressed number in combination
output reg pulse_after_input //input digit is checked after that pulse.
);

always @(posedge RST_For_Lock) begin // reset position and pulse

```

```
position_of_number=0;
pulse_after_input=0;
end
```

```
always @(negedge door_is_open) // when door closes position is reset
position_of_number=0;
```

```
always
@(b_1_for_lock,b_2_for_lock,b_3_for_lock,b_4_for_lock,b_5_for_lock,b_6_for_lock,b_7_for_
lock,b_8_for_lock,b_9_for_lock,b_0_for_lock)

pulse_after_input=~pulse_after_input; // at negedge, sequence detector should be activated
```

```
always @(posedge b_1_for_lock) begin // 1 is pressed
input_number=1;
position_of_number=position_of_number+1;
end
```

```
always @(posedge b_2_for_lock) begin // 2 is pressed
input_number=2;
position_of_number=position_of_number+1;
end
```

```
always @(posedge b_3_for_lock) begin // 3 is pressed
input_number=3;
position_of_number=position_of_number+1;
end
```

```
always @(posedge b_4_for_lock) begin // 4 is pressed
input_number=4;
```

```
position_of_number=position_of_number+1;  
end
```

```
always @(posedge b_5_for_lock) begin // 5 is pressed  
input_number=5;  
position_of_number=position_of_number+1;  
end
```

```
always @(posedge b_6_for_lock) begin // 6 is pressed  
input_number=6;  
position_of_number=position_of_number+1;  
end
```

```
always @(posedge b_7_for_lock) begin // 7 is pressed  
input_number=7;  
position_of_number=position_of_number+1;  
end
```

```
always @(posedge b_8_for_lock) begin // 8 is pressed  
input_number=8;  
position_of_number=position_of_number+1;  
end
```

```
always @(posedge b_9_for_lock) begin // 9 is pressed  
input_number=9;  
position_of_number=position_of_number+1;  
end
```

```

always @(posedge b_0_for_lock) begin // 0 is pressed
input_number=0;
position_of_number=position_of_number+1;
end

endmodule

```

Garage

```

`timescale 1ns / 1ps

// the user has remote control. There are two buttons on the remote control
// one button just opens and closes the door
// the second button activates smart mode.

module Top_Module_Garage_Door(
input open_close,smart_mode_pulse,RST_For_Garage,garage_clock,// open_clos push opens or
closes the door, smartmode_pulse activates smart mode,
input Is_object, // 1 when something is inf front of the door
output open_the_Door); // opens the door

reg enable_smart_mode; // enables smart mode
reg Garage_open_the_Door;
parameter s0=0,s1=1,s2=2,s3=3;
reg [1:0] state;
reg tmp;

always @(posedge RST_For_Garage) begin // reset - close the door, desable smartmode
enable_smart_mode<=0;// disable smart mode
Garage_open_the_Door<=0; // close the door

```



```
//state<=s0;//go to reset state
```

```
tmp<=0; // if it is one and smart mode is enable, smart mode gets disabled
```

```
end
```

```
always @(posedge open_close) begin // after pushing open_close button
```

```
enable_smart_mode<=0;// disable smart mode
```

```
Garage_open_the_Door<=~Garage_open_the_Door; // if garage door is open, close. otherwise  
open it
```

```
//state<=s0;//go to reset state
```

```
end
```

```
always @(posedge smart_mode_pulse) begin// after pushing smart_mode_pulse button
```

```
enable_smart_mode<=1;// enable smart mode
```

```
state<=s0;//go to reset state
```

```
end
```

```
always @(posedge garage_clock) begin
```

```
if(enable_smart_mode) begin
```

```
case(state)
```

```
s0: if(Is_object) begin // object is detected door is closed
```

```
state<=s1; // advance
```

```
Garage_open_the_Door<=1; // open door
```

```
end
```

```
else begin // object is not detected door is closed
```

```
state<=s0; // do not advance
```

```
Garage_open_the_Door<=0; // do not open the door
```

```
end
```

```
s1: if(Is_object) begin // object is in front of the door , door is open
```

```
state<=s1; // do not advance
```

```

Garage_open_the_Door<=1; // leave the door open
end

else begin // object is no longer in front of the garage
state<=s2; // advance

Garage_open_the_Door<=1; // leave door open
end

s2: if(Is_object) begin // object is again in front of the door, door is open
state<=s3; // advance

Garage_open_the_Door<=1; // leave the door open
end

else begin // object still is not in front of the door door is open
state<=s2; // do not advance

Garage_open_the_Door<=1; // leave the door open
end

s3: if(Is_object) begin // object still is in front of the door
state<=s3; // do not advance

Garage_open_the_Door<=1; // leave the door open
end

else begin //the object is no longer detected
state<=s0; // go to reset state

Garage_open_the_Door<=0; // close the door
tmp<=1; // this will make sure that after the door closes, smart mode is disable
end

endcase

end

end

always @(posedge garage_clock) begin
if(tmp&enable_smart_mode) begin

```

```

enable_smart_mode<=0;
tmp<=0;
end
end

assign open_the_Door=Garage_open_the_Door;
endmodule

```

TESTBENCH

```

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////

module TB_Solomnishvili_Giorgi;
// chekc clocks
reg CLK;
wire clk_out_1;
wire clk_out_10;
wire clk_out_60;

clock #(1) klok00(.clk_in(CLK),.clk_out(clk_out_1));// 1s
clock #(10) klok01(.clk_in(CLK),.clk_out(clk_out_10)); // 10s
clock #(60) klok10(.clk_in(CLK),.clk_out(clk_out_60)); // 60s

initial CLK=0;
always #(10000000000*0.5*1/50) CLK=~CLK;// /60 for 60hz 10000000000

```

```

// check 7-segment BCD
reg [3:0] digit_to_display;
reg ny_countdown_check;
wire [6:0] display;
reg [3:0] i;

Clock_digit_display #(1)
digit00(.digit_to_display(digit_to_display),.ny_countdown(ny_countdown_check),.display(display));

initial begin
ny_countdown_check=0;
//digit_to_display=1;
#3; ny_countdown_check=1;
for(i=0;i<10;i=i+1) begin
digit_to_display=i;
#2;
end
end

// check quotient remainder
reg [6:0] dividend,divisor;
wire [6:0] quotient, remainder;

clock_division_remainder
clock_division_remainder00(.CLK(CLK),.dividend(dividend),.divisor(divisor),.quotient(quotient),.remainder(remainder));

initial begin
dividend = 13;
divisor = 23;
end

```

```

// check leap year detector
reg [3:0] milleniacheck, centurycheck, decadecheck, yearcheck;

wire leap_yearcheck;

leap_year_detector
leap_year_detector00(.millenia(milleniacheck),.century(centurycheck),.decade(decadecheck),.year(yearcheck),.leap_year(leap_yearcheck));

initial begin;
milleniacheck = 1;
centurycheck = 9;
decadecheck = 9;
yearcheck = 9;

clock_10monthcheck = 1;
clock_1monthcheck = 1;
clock_10daycheck = 1;
clock_daycheck = 4;
/*
for(m_10=0;m_10<2;m_10=m_10+1) begin: waldo1
    for(m=1;m<10;m=m+1) begin: waldo2
        @(posedge CLK);
        clock_10monthcheck=m_10;
        clock_1monthcheck=m;
        if((clock_1monthcheck==2)&(clock_10monthcheck==1))
            disable waldo2;
        end
        if((clock_1monthcheck==2)&(clock_10monthcheck==1))
            disable waldo1;
        end
end

```

```
*/
```

```
end
```

```
//check clock_week_day
```

```
reg [3:0] clock_daycheck,clock_10daycheck,clock_1monthcheck,clock_10monthcheck;
```

```
wire [2:0] weekday_check;
```

```
wire [4:0] differencecheck=clock_week_day00.difference;
```

```
wire [6:0] second_term=clock_week_day00.clock_divsion_remainder10.divisor;
```

```
wire [6:0] first_term=clock_week_day00.clock_divsion_remainder10.dividend;
```

```
wire [6:0] result=clock_week_day00.clock_divsion_remainder10.quotient;
```

```
wire [4:0] day = clock_week_day00.day; //numbers ind 10base value
```

```
wire [4:0] month = clock_week_day00.month;
```

```
wire [6:0] year_2_least_signif = clock_week_day00.year_2_least_signif;
```

```
wire [6:0] year_2_most_signif = clock_week_day00.year_2_most_signif;
```

```
wire [2:0] century_code = clock_week_day00.century_code;
```

```
wire [2:0] Doomsday_of_year=clock_week_day00.Doomsday_of_year;
```

```
wire [4:0] Doomsday_of_month=clock_week_day00.Doomsday_of_month;
```

```
reg [3:0] m=1;
```

```
reg m_10=0;
```

```
clock_week_day clock_week_day00(.Is_Leap_Year(leap_yearcheck),.CLK(CLK),
```

```
.clock_day(clock_daycheck),.clock_10day(clock_10daycheck),.clock_1month(clock_1monthche  
ck),.clock_10month(clock_10monthcheck),
```

```
.clock_year(yearcheck),.clock_decade(decadecheck),.clock_century(centurycheck),.clock_millen  
ia(milleniacheck),
```

```
.weekday(weekday_check));
```

```
// check clock_set_time
```

```
//check Top module
```

```
reg CLK, RST;
```

```
reg set_time_enable,b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_0,up,down;
```

```
wire [3:0] clock_second;// digit for second
```

```
wire [3:0] clock_10second,clock_1minute,
```

```
clock_10minute,clock_1hour,clock_10hour,clock_day,clock_10day,clock_1month,clock_10mon  
th,year,decade,century,millenia;
```

```
wire [2:0] weekday;
```

```
wire [6:0] display_clock_second, display_clock_10second, display_clock_1minute,  
display_clock_10minute, display_clock_1hour,display_clock_10hour;
```

```
wire [6:0] display_clock_day, display_clock_10day, display_clock_1month,  
display_clock_10month,  
display_clock_millenia,display_clock_century,display_clock_decade,display_clock_year;
```

```
wire [19:0] display_I_letter;
```

```
wire [9:0] display_II_letter,display_III_letter;
```

```
Top_Module
```

```
ut00(.CLK(CLK),.RST(RST),.clock_second(clock_second),.clock_10second(clock_10second),.c  
lock_1minute(clock_1minute),.clock_10minute(clock_10minute),.clock_1hour(clock_1hour),.cl  
ock_10hour(clock_10hour),
```

```
.clock_day(clock_day),.clock_10day(clock_10day),.clock_1month(clock_1month),.clock_10mo  
nth(clock_10month),
```

```
.clock_year(year),.clock_decade(decade),.clock_century(century),.clock_millenia(millenia),.wee  
kday(weekday),
```

```
.display_clock_second(display_clock_second),.display_clock_10second(display_clock_10secon  
d),.display_clock_1minute(display_clock_1minute),.display_clock_10minute(display_clock_10  
minute),.display_clock_1hour(display_clock_1hour),.display_clock_10hour(display_clock_10ho  
ur),
```

```
.display_clock_day(display_clock_day),.display_clock_10day(display_clock_10day),.display_cl  
ock_1month(display_clock_1month),.display_clock_10month(display_clock_10month),
```

```

.display_clock_millenia(display_clock_millenia),.display_clock_century(display_clock_century)
,.display_clock_decade(display_clock_decade),.display_clock_year(display_clock_year),

.display_I_letter(display_I_letter),.display_II_letter(display_II_letter),.display_III_letter(display
_III_letter),

.set_time_enable(set_time_enable),.b_1(b_1),.b_2(b_2),.b_3(b_3),.b_4(b_4),.b_5(b_5),.b_6(b_6)
,.b_7(b_7),.b_8(b_8),

.b_9(b_9),.b_0(b_0),.up(up),.down(down));

```

```

wire tmp = ut00.pulse_period_year;

wire Is_Leap_Year= ut00.Is_Leap_Year; //this is 1 if it is leap year

wire Is_February= ut00.Is_February; // is 1 if february

wire Is_31_day= ut00.Is_31_day;

wire [4:0] difference=ut00.clock_week_daymodule00.difference;

```

```

initial begin

RST=1;set_time_enable=0;

@(posedge clk_out_1);

//@(posedge clk_out_1);

RST=0;set_time_enable=1; // counter = 0


@(posedge clk_out_1);b_1=1; // set h10 to 1

@(posedge clk_out_1);b_1=0;


@(posedge clk_out_1);up = 1; // counter =1

@(posedge clk_out_1);up = 0;


@(posedge clk_out_1);b_2=1; // set h to 2

```



```
@(posedge clk_out_1);b_2=0;
```

```
@(posedge clk_out_1);up = 1; // counter =2
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_3=1; // set m10 to 3
```

```
@(posedge clk_out_1);b_3=0;
```

```
@(posedge clk_out_1);up = 1; // counter =3
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_4=1; // set m to 4
```

```
@(posedge clk_out_1);b_4=0;
```

```
@(posedge clk_out_1);up = 1; // counter =4
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_5=1; // set s10 to 5
```

```
@(posedge clk_out_1);b_5=0;
```

```
@(posedge clk_out_1);up = 1; // counter =5
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_8=1; // set s to 8
```

```
@(posedge clk_out_1);b_8=0;
```

```
@(posedge clk_out_1);up = 1; // counter =6
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_0=1; // set month10 to 0
```

```
@(posedge clk_out_1);b_0=0;
```

```
@(posedge clk_out_1);up = 1; // counter =7
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_7=1; // set month to 7
```

```
@(posedge clk_out_1);b_7=0;
```

```
@(posedge clk_out_1);up = 1; // counter =8
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_2=1; // set d10 to 2
```

```
@(posedge clk_out_1);b_2=0;
```

```
@(posedge clk_out_1);up = 1; // counter =9
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_9=1; // set d to 9
```

```
@(posedge clk_out_1);b_9=0;
```

```
@(posedge clk_out_1);up = 1; // counter =10
```

```
@(posedge clk_out_1);up = 0;
```

```
@(posedge clk_out_1);b_2=1; // set Milen to 2
```

```
@(posedge clk_out_1);b_2=0;
```

```

@(posedge clk_out_1);up = 1; // counter =11
@(posedge clk_out_1);up = 0;

@(posedge clk_out_1);b_0=1; // set century to 0
@(posedge clk_out_1);b_0=0;

@(posedge clk_out_1);up = 1; // counter =12
@(posedge clk_out_1);up = 0;

@(posedge clk_out_1);b_2=1; // set decade to 2
@(posedge clk_out_1);b_2=0;

@(posedge clk_out_1);up = 1; // counter =13
@(posedge clk_out_1);up = 0;

@(posedge clk_out_1);b_0=1; // set year to 0
@(posedge clk_out_1);b_0=0;

@(posedge clk_out_1);set_time_enable=0;
end

// END of DIGITAL CLOCK

// Begin Check of Lokc
wire clock_for_locker;
assign clock_for_locker=CLK;
reg RST_For_Lock;
//reg [3:0] input_number; // this will be changed by the keyboard

```

```

reg
b_1_for_lock,b_2_for_lock,b_3_for_lock,b_4_for_lock,b_5_for_lock,b_6_for_lock,b_7_for_lock,
b_8_for_lock,b_9_for_lock,b_0_for_lock;// keyboard

reg OK;

reg door_is_open;

wire unlock;


wire [2:0] state_for_lock_sequence =
Top_Module_For_lock00.Lock_Sequence_Detector00.state;

wire [2:0] counter_for_combination_digits =
Top_Module_For_lock00.Lock_Sequence_Detector00.counter;

wire [3:0] combination [1:4]
=Top_Module_For_lock00.Lock_Sequence_Detector00.combination; // this array stores correct
combination


Top_Module_For_lock Top_Module_For_lock00(.CLK_For_Lock(CLK),
.RST_For_Lock(RST_For_Lock),.OK(OK),.unlock(unlock),.door_is_open(door_is_open),
.b_1_for_lock(b_1_for_lock),.b_2_for_lock(b_2_for_lock),.b_3_for_lock(b_3_for_lock),.b_4_for_lock(b_4_for_lock),.b_5_for_lock(b_5_for_lock),
.b_6_for_lock(b_6_for_lock),.b_7_for_lock(b_7_for_lock),.b_8_for_lock(b_8_for_lock),.b_9_for_lock(b_9_for_lock),.b_0_for_lock(b_0_for_lock));

initial begin
RST_For_Lock = 0; door_is_open=1; // door is open,
@(posedge CLK); RST_For_Lock=1; // reset clock states
@(posedge CLK); RST_For_Lock=0;


@(posedge CLK); b_7_for_lock=1; // setting combination 1
@(posedge CLK); b_7_for_lock=0;


@(posedge CLK); b_8_for_lock=1; // setting combination 2

```

```
@(posedge CLK); b_8_for_lock=0;
```

```
@(posedge CLK); b_9_for_lock=1; // setting combination 3
```

```
@(posedge CLK); b_9_for_lock=0;
```

```
@(posedge CLK); b_0_for_lock=1; // setting combination 4
```

```
@(posedge CLK); b_0_for_lock=0;
```

```
@(posedge CLK); door_is_open = 0; // close the door
```

```
//entering correct combination to unlock
```

```
@(posedge CLK); b_7_for_lock=1; // setting combination 1
```

```
@(posedge CLK); b_7_for_lock=0;
```

```
@(posedge CLK); b_8_for_lock=1; // setting combination 2
```

```
@(posedge CLK); b_8_for_lock=0;
```

```
@(posedge CLK); b_9_for_lock=1; // setting combination 3
```

```
@(posedge CLK); b_9_for_lock=0;
```

```
@(posedge CLK); b_9_for_lock=1; // setting combination 4
```

```
@(posedge CLK); b_9_for_lock=0;
```

```
@(posedge CLK); OK=1;
```

```
@(posedge CLK); OK=0;
```

```
//entering correct combination to unlock
```

```
@(posedge CLK); b_7_for_lock=1; // setting combination 1
```

```
@(posedge CLK); b_7_for_lock=0;
```

```
@(posedge CLK); b_8_for_lock=1; // setting combination 2
```

```
@(posedge CLK); b_8_for_lock=0;
```

```
@(posedge CLK); b_9_for_lock=1; // setting combination 3
```

```
@(posedge CLK); b_9_for_lock=0;
```

```
@(posedge CLK); b_0_for_lock=1; // setting combination 4
```

```
@(posedge CLK); b_0_for_lock=0;
```

```
@(posedge CLK); OK=1;
```

```
@(posedge CLK); OK=0;
```

```
@(posedge CLK);door_is_open=1;
```

```
@(posedge CLK);door_is_open=0;
```

```
end
```

```
// end of lock
```

```
// begin Dishwasher
```

```
wire clocl_for_dishwasher=Top_Module_For_Dishwasher00.pulse_1m;
```

```
reg POWER, START,RST_For_Dishwasher; // power and start button generates impulses
```

```
reg
```

```
b_1_for_dishwasher,b_2_for_dishwasher,b_3_for_dishwasher,b_4_for_dishwasher,b_5_for_dishwasher,b_6_for_dishwasher,b_7_for_dishwasher,b_8_for_dishwasher,b_9_for_dishwasher,b_0_for_dishwasher;
```

```
wire [3:0] dishwasher_10hours,dishwasher_hours,dishwasher_10minutes,dishwasher_minutes;
```

```
wire [6:0]
dishwasher_10hours_display,dishwasher_hours_display,dishwasher_10minutes_display,dishwasher_minutes_display;
```

```
Top_Module_For_Dishwasher
```

```
Top_Module_For_Dishwasher00(.clock_for_dishwasher(CLK),.POWER(POWER),.START(START),.RST_For_Dishwasher(RST_For_Dishwasher),
```

```
.b_1_for_dishwasher(b_1_for_dishwasher),.b_2_for_dishwasher(b_2_for_dishwasher),.b_3_for_dishwasher(b_3_for_dishwasher),
```

```
.b_4_for_dishwasher(b_4_for_dishwasher),.b_5_for_dishwasher(b_5_for_dishwasher),
```

```
.b_6_for_dishwasher(b_6_for_dishwasher),.b_7_for_dishwasher(b_7_for_dishwasher),
```

```
.b_8_for_dishwasher(b_8_for_dishwasher),.b_9_for_dishwasher(b_9_for_dishwasher),.b_0_for_dishwasher(b_0_for_dishwasher),
```

```
.dishwasher_10hours(dishwasher_10hours),.dishwasher_hours(dishwasher_hours),.dishwasher_10minutes(dishwasher_10minutes),.dishwasher_minutes(dishwasher_minutes),
```

```
.dishwasher_10hours_display(dishwasher_10hours_display),.dishwasher_hours_display(dishwasher_hours_display),
```

```
.dishwasher_10minutes_display(dishwasher_10minutes_display),.dishwasher_minutes_display(dishwasher_minutes_display));
```

```
wire pulse_dish=Top_Module_For_Dishwasher00.pulse;
```

```
wire power_on_dish=Top_Module_For_Dishwasher00.power_on;
```

```
wire start_begin_dish=Top_Module_For_Dishwasher00.start_begin; // if poweron is 1 - device is turned on, if start_begin is 1, the device does its job
```

```
wire [2:0] position_of_digit_dish=Top_Module_For_Dishwasher00.position_of_digit;
```

```
wire [3:0] timer_value_dish=Top_Module_For_Dishwasher00.timer_value;
```

```
wire finish = Top_Module_For_Dishwasher00.finish;
```

```
initial begin
```

```
RST_For_Dishwasher = 1; // reset dishwasher
```

```
@(posedge clock_for_dishwasher);RST_For_Dishwasher=0;
```

```
@(posedge CLK);POWER=1; // press power button
```

```
@(posedge CLK);POWER=0;
```

```
@(posedge CLK);b_1_for_dishwasher=1; // press 1
```

```
@(posedge CLK);b_1_for_dishwasher=0; //
```

```
@(posedge CLK);b_0_for_dishwasher=1; // press 0
```

```
@(posedge CLK);b_0_for_dishwasher=0; //
```

```
@(posedge CLK);b_0_for_dishwasher=1; // press 0
```

```
@(posedge CLK);b_0_for_dishwasher=0; //
```

```
@(posedge CLK);b_0_for_dishwasher=1; // press 0
```

```
@(posedge CLK);b_0_for_dishwasher=0; //
```

```
@(posedge CLK);START=1; // press start
```

```
@(posedge CLK);START=0;
```

```
@(posedge CLK);@(posedge CLK);
```

```
end
```

```
//end dishwasher-gascooker
```

```
//begin garage
```

```
wire garage_clock;
```

```
assign garage_clock=CLK;
```

```
reg open_close,smart_mode_pulse,RST_For_Garage;// open_clos push opens or closes the door,  
smartmode_pulse activates smart mode,
```

```
reg Is_object; // 1 when something is inf front of the door
```

```
wire open_the_Door;
```



```
wire enable_smart_mode=Top_Module_Garage_Door00.enable_smart_mode; // enables smart mode
```

```
//wire Garage_open_the_Door=Top_Module_Garage_Door00.Garage_open_the_Door;
```

```
wire [1:0] state=Top_Module_Garage_Door00.state;
```

```
Top_Module_Garage_Door
```

```
Top_Module_Garage_Door00(.open_close(open_close),.smart_mode_pulse(smart_mode_pulse),  
.RST_For_Garage(RST_For_Garage),.garage_clock(CLK),
```

```
.Is_object(Is_object),.open_the_Door(open_the_Door));
```

```
initial begin
```

```
RST_For_Garage=1;Is_object=0; // reset everythin
```

```
@(posedge CLK);RST_For_Garage=0;
```

```
@(posedge CLK);open_close=1; // press open-close butoon - open the door
```

```
@(posedge CLK);open_close=0;
```

```
@(posedge CLK);open_close=1; // press open-close butoon - close the door
```

```
@(posedge CLK);open_close=0;
```

```
@(posedge CLK);open_close=1; // press open-close butoon - open the door
```

```
@(posedge CLK);open_close=0;
```

```
@(posedge CLK);smart_mode_pulse=1; // activate smart mode
```

```
@(posedge CLK);smart_mode_pulse=0;
```

```
@(posedge CLK);Is_object=0; //testing state 0 - stay
```

```
@(posedge CLK);Is_object=1; //advance the state
```

```

@(posedge CLK);Is_object=1;//testing state 1 - stay
@(posedge CLK);Is_object=0; //advance the state

@(posedge CLK);Is_object=0;//testing state 2 - stay
@(posedge CLK);Is_object=1; //advance the state

@(posedge CLK);Is_object=1;//testing state 3 - stay
@(posedge CLK);Is_object=0;//advance the state


// pressing open-close button deactivates smartmode
@(posedge CLK);@(posedge CLK);@(posedge CLK);
@(posedge CLK);smart_mode_pulse=1; // activate smart mode
@(posedge CLK);smart_mode_pulse=0;

@(posedge CLK);open_close=1; // press open-close butoon - open the door
@(posedge CLK);open_close=0;

end
endmodule

```

5) Specific tool set used

In order to simulate and debug the project, vivado is required. I am providing the testbench that has code to test almost all of my modules. I am providing all the modules as well. It will be sufficient if you run testbench for 4000s by the command run 4000s.

6) Notes on anything else we need to know to synthesize and simulate using your test bench(es).

My testbench contains codes for almost all test bench I ran. Lines from 100 – 217 are dedicated to testing module for digital clock. Inside of the initial statement for digital clock, there are 14 pairs of the following assignment statements:

```
@(posedge clk_out_1);b_1=1; // set h10 to 1
```

```
@(posedge clk_out_1);b_1=0;
```

They represent pressing the button to set each of 14 digits of time. This pair sets the first digit of hours to 1. If we change b_1 by b_2, 2 will be assigned to h10 initially. Waveforms from 36th waveform to 86th waveform belongs to digital clock part. The 36th waveform is for RST and 86th is for difference – a variable in module division-remainder. Those waveforms are for RST, set-time-enable signals, buttons (b_0-b9 and up, down button), values for time digit counters (14 in total), values for time digit display (14 in total), value for weekday, values for weekday display (3 in total), tmp signal representing pulse with period 1 year, Isfebruary, isLeap, is31days signal and difference signal.

In the testbench, lines from 220-278 belong to lock. Pairs of assignments like presented above represent pushing button to set combination.

Waveforms from 86th waveform to 104th waveform belongs to lock. Waveforms belong to clock Signal, reset signal, buttons (b0-b9), OK button. There are also signals like door_is_open and Unlock. There are also waveforms representing combination, state and counter. Some of the Signal are going to be x because they represent buttons and the have not been pressed in the Testbench.

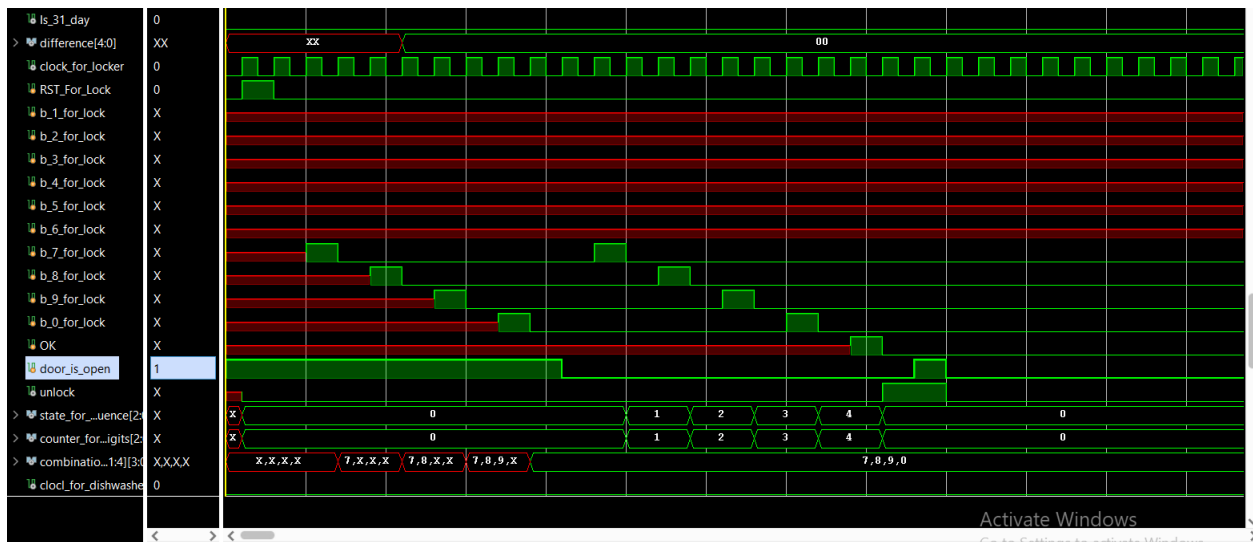


Figure 14.

In testbench, lines from 282 to 325 belong to dishwasher/cooker. The same kind of pairs represent setting timer. On the waveform graph, waveforms starting from clock_for_dishwasher till garage_clock belong to dishwasher/cooker.

The last 8 waveforms belong to part of testbench dedicated to testing garage module (lines 329 till the end). Figure 15 shows waveforms. Open_close pulse is generated 3 times to test if open/close button works. Next smart mode is enabled, and it is tested for all possible combinations of input.

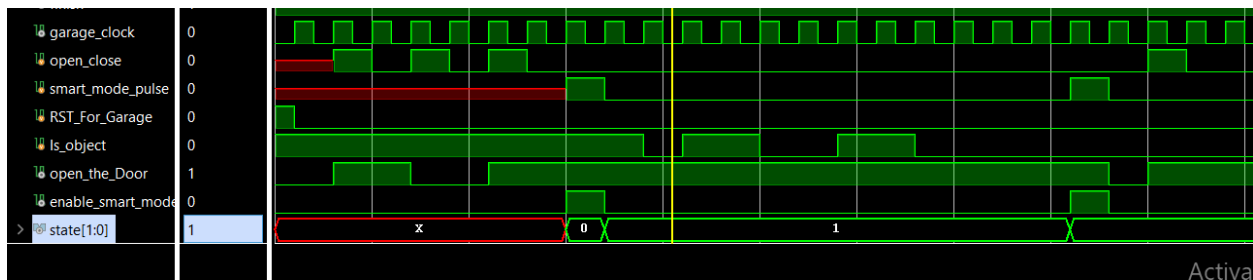
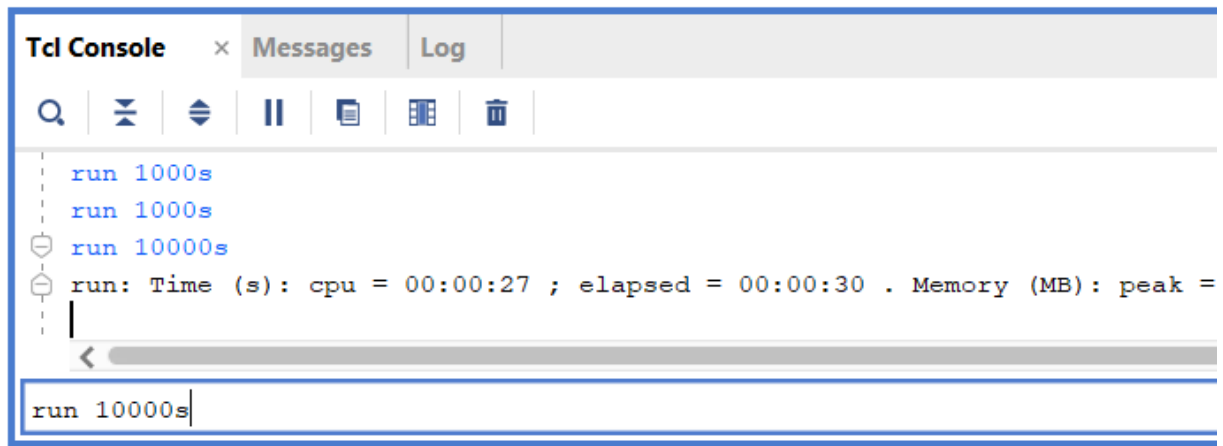


Figure 15

Since my project is highly dependent on time, running it for 10000s should give you enough waveform data to analyze.



The screenshot shows a Tcl Console window with three tabs: 'Tcl Console' (active), 'Messages', and 'Log'. The console contains the following text:

```
run 1000s  
run 1000s  
run 10000s  
run: Time (s): cpu = 00:00:27 ; elapsed = 00:00:30 . Memory (MB): peak =  
|  
<
```

At the bottom, there is an input line with the text 'run 10000s' and a cursor at the end.

It takes 30 seconds to run the simulation. For some reason, when I run simulation for n seconds, the waveforms are generated for 10 times n seconds.

7) Test results:

While working on the project, I wrote several versions of testbenches to check module clock, 7-segment BCD, the module that generates quotient and remainder of a ratio. I also wrote testbenches for modules such as leap_year_detector, clock_week_day. Besides those testbenches, I wrote testbenches that tests top_module_for_clock; top_module_for_lock; top_module_for_dishwasher/cooker; top_module_for_garage door.

Resulting waveforms

Modules associated with Digital clock:

It is assumed that the device is powered up by 50 hz power supply. Module clock uses this fundamental signal as reference points and generates a signal whose period is one second. See figure 16:

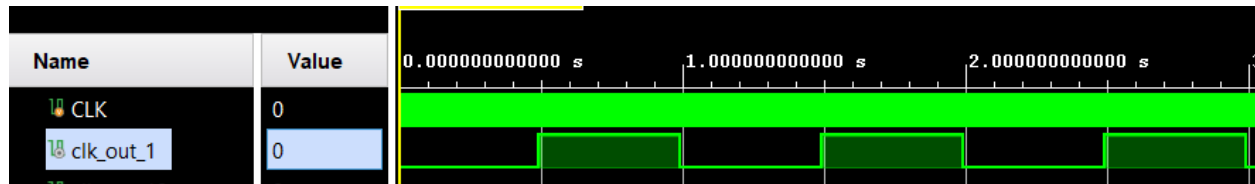


Figure 16

Figure 17 displays signal for the last second. The signal counts from 0 to 9 and then goes to 0. Clk_out has period 1 s.



Figure 17

Figure 18 displays signal for the first second digit. The signal changes when the last second goes from 9 to 0. It counts from 0 to 5 and then goes to 0:



Figure 18

Figure 19 displays signal for the last Minute digit. The signal changes when the first second digit goes from 5 to 0. It counts from 0 to 9 and then goes to 0:



Figure 19

Figure 20 and 21 display signal for the first Minute digit. The signal changes when the last minute digit goes from 9 to 0. It counts from 0 to 5 and then goes to 0:



Figure 20



Figure 21

Figure 22 shows that the last digit of hour changes when the first digit of minutes goes from 5 to 0. Figure 22 also shows that if the first digit of hour is 2, the last digit of hour changes from 0 to 3. Moreover, the first digit of hour changes from 2 to 0 when the last digit of hour changes from 3 to 0.



Figure 22

Figure 23 shows that when the first digit of hour is not 2, it changes when the last digit of hour changes from 0 to 9.



Figure 23

I checked digits for hour and minutes by running the simulation for 100 000s using run 100000s. The vivado simulate 1 million seconds. It took 3-4 minutes.

The digital clock uses 4 digits to display date in the following format: MM/DD. If MM is 01, 03, 05, 07, 08, 10, and 12, DD changes from 01 to 31. In Top module for clock, I created signal Is_31_day which equals is 1 if month is 1 or 3 or 5 or 7 or 8 or 10 or 12. In testbench I created a variable tmp which was assigned with Is_31_day. Figure 24 shows that when tmp is 1, DD maximum value is 31:



Figure 24

Figure 24' shows that Is_31_day is 0 for November and 1 for May.

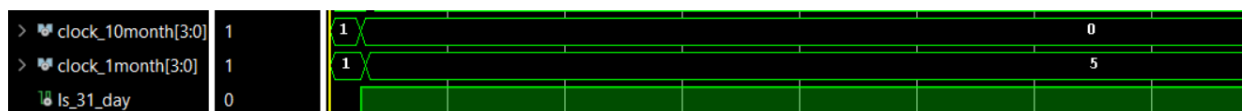


Figure 25 and 26 shows that DD changes from 01 to 31

>  clock_10hour[3:0] 0 

> clock_day[3:0] 1

>  clock_10day[3:0] 0 

[illegible]

	1	9	9	9	0
> millenia[3:0]	1	9	9	9	0
> century[3:0]	9	9	9	9	0
> decade[3:0]	9	9	9	9	0
> year[3:0]	9	9	9	9	0
Is_Leap_Year	0				

[illegible]

> clock_day[3:0]	1	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9							
> clock_10day[3:0]	0										1									2									0								

[illegible][illegible]

Figure 31

The first month digit changes from 0 to 1. The second month digit changes from 1 to 9 if the first month digit is 0, otherwise from 0 to 2. Overall, MM changes from 01 to 12. Figure 32 represents transition from 9 to 0 transition when the first digit of month is 0.



Figure 32

Figure 33 represents transition from 2 to 1 transition when the first digit of month is 1.

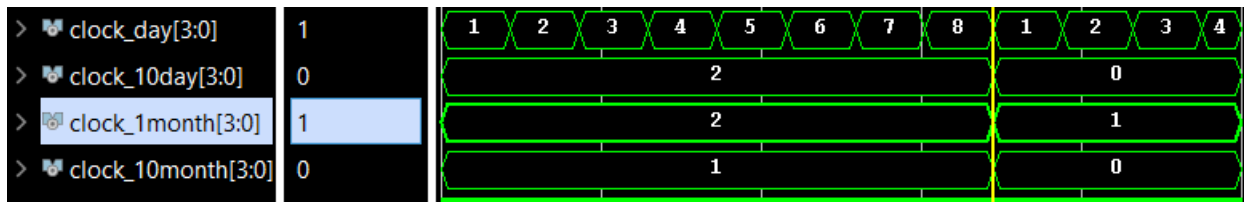


Figure 33

Figure 32 and 33 only proves that month changes from 9 to 0 when 10month is 0 and from 2 to 1 when 10month is 1. It does not show correct days.

The module clock_mm_dd_counters has counters for MM and DD. The instance of module. The module clock_mm_dd_counters outputs a pulse whose period is 1 year. That signal goes to one when the first digit of month goes from 1 to 0. At this point tmp is assigned to that pulse. Figure 34

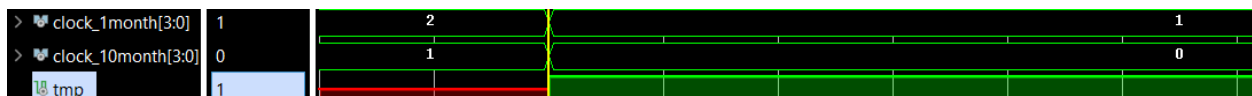


Figure 34

The digital clock uses 4 digits to display date in the following format: Milenia Century Decade Year or MCDY. Each digit changes from 0 to 9. Overall, the digital clock can display years from 000 to 9999. The clock can work for 10000 years. The module clock_year_YY is instantiated 4 times to represent each digit of the year. The module takes input clock signal and increments the counter at positive edge of that signal. Each instance of the module also outputs signal with period of 1, 10, 100 or 1000 year. The instance for year takes output pulse of the module that counts the first digit of month. This signal is represented on Figure 34 as tmp. However, it is not very practical to simulate modules for a year, century or millennia. That is

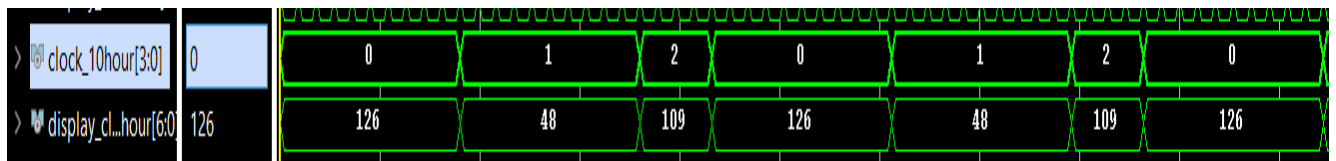


Figure 39

LEDS for the second digit of hour:



Figure 40

LEDS for the first digit of minute:



Figure 41

LEDS for the second digit of minute:



Figure 42

LEDS for the first digit of seconds:

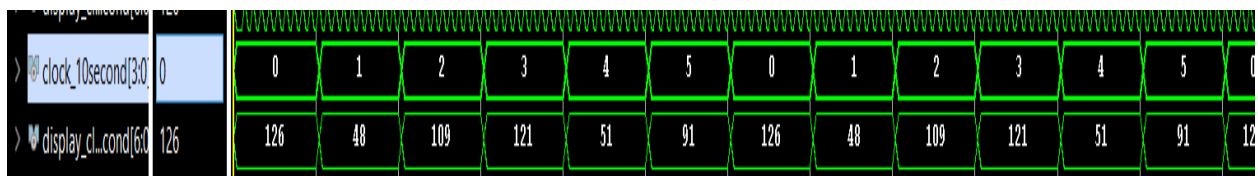


Figure 43

LEDS for the second digit of seconds:



Figure 44

LEDS for the first digit of months:

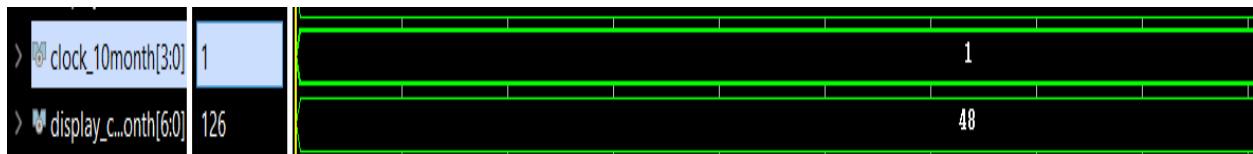


Figure 45

LEDS for the second digit of months:



Figure 46

LED for the first digit of day

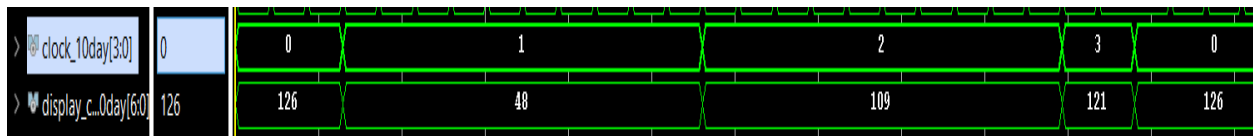


Figure 47

LED for the second digit of day



Figure 48

For year

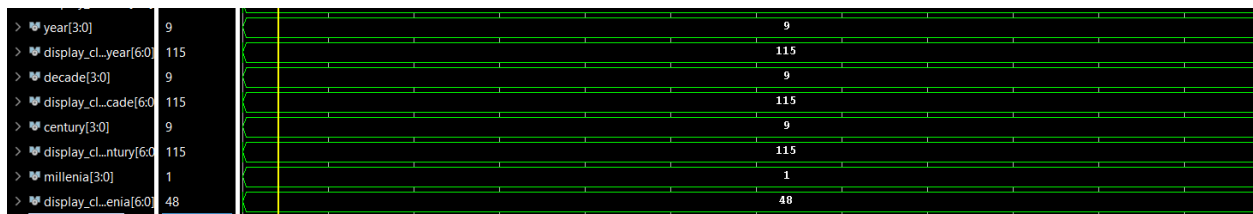


Figure 49

Figure 50 shows that when ny_countdown is 1, the module for 7-segment LED counts down from 9 to 1.

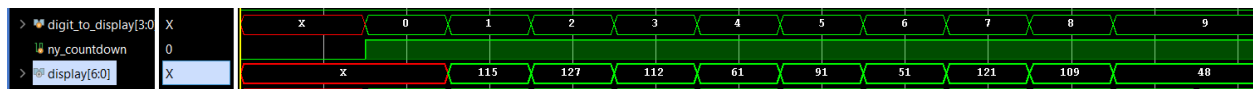


Figure 50

The user can change values for time, date and year manually. The module set time takes 12 inputs which represent set_time_enable signal, 0-9 number buttons, up button, down button. The user has a remote control of the digital clock. Those buttons are on that control. When the user presses set_tim_enable button, digital clock stops counting and waits input values. Pressing buttons B1,B2.....B9,B0, generates values 1,2....9,0 respectively. Figure 51 proves that point. Pressing a button is represented by a pulse:

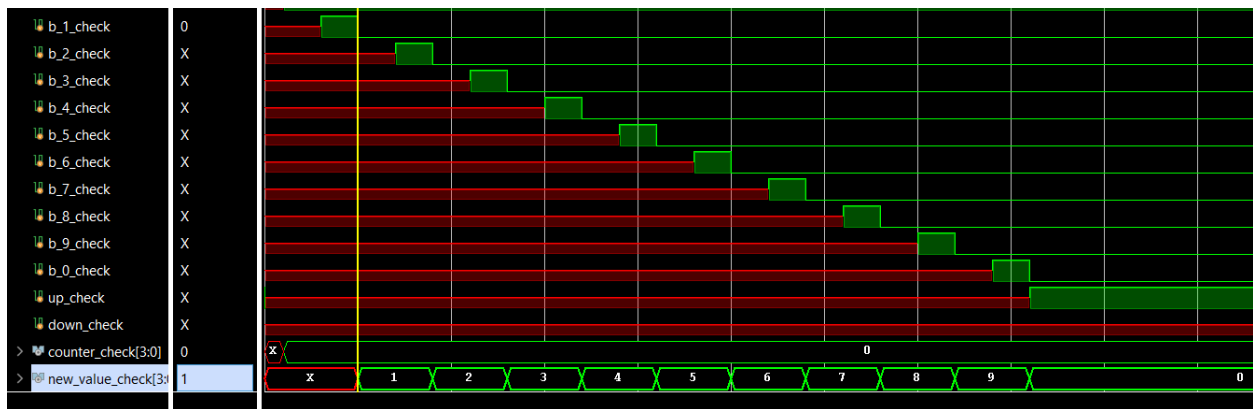


Figure 51

By up and down button, the user can choose a digit of clock that is going to be set. The module has counter which can have values within range 0-13 which corresponds to clock digits in the order hh-mm-ss MM/DD/MCDY. Pressing up increases counter, pressing down decreases counter. This is shown on figure 52 and 53.

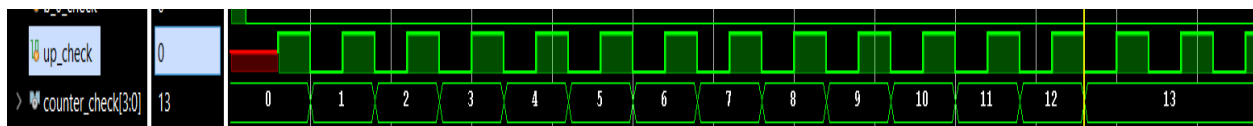


Figure 52



Figure 53

the new values are written in time counters and counting continues. Figure 54 displays setting date to 04/25/2020

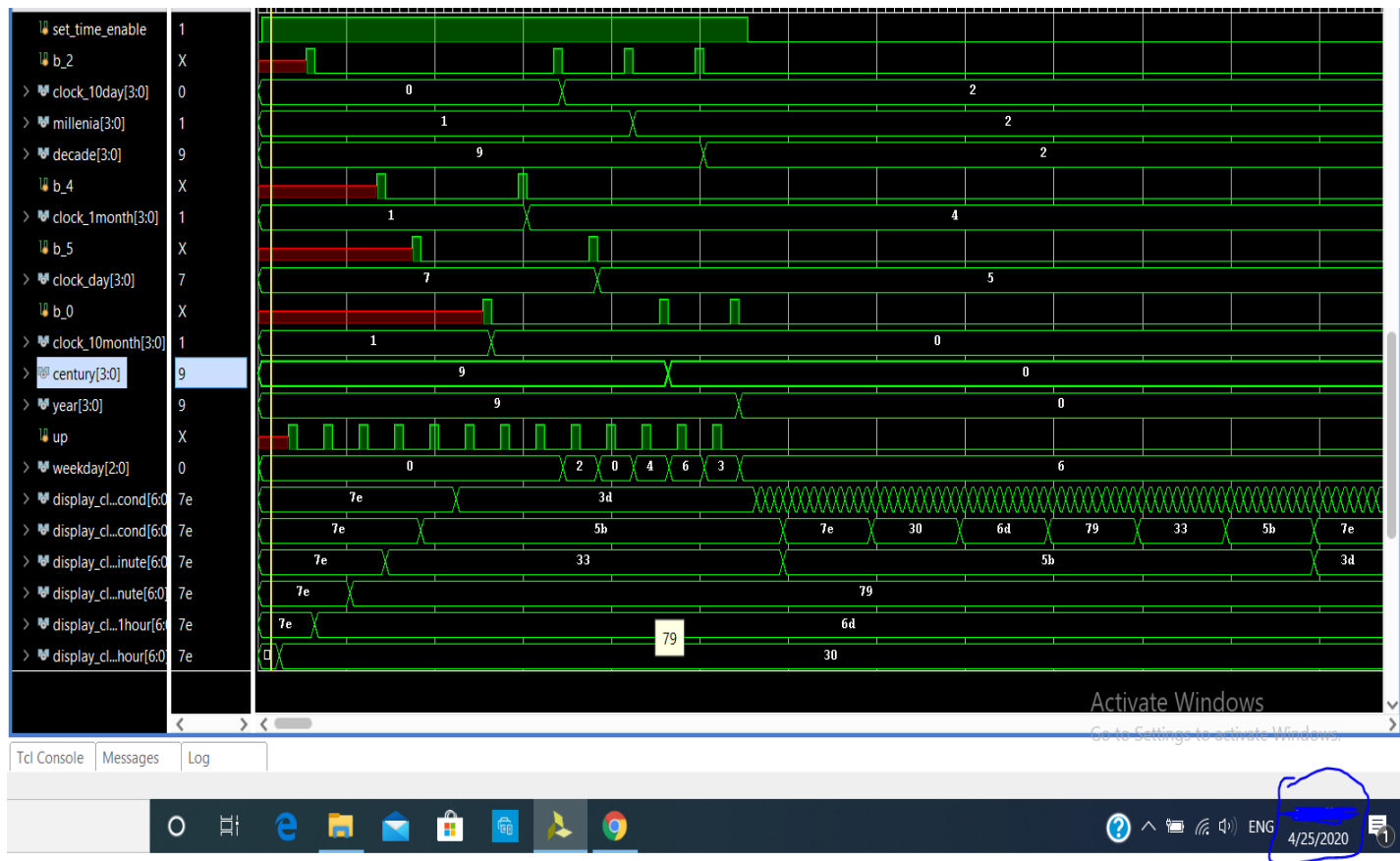


Figure 54

Figure 55 shows that the module `week_day_detector` knows Doomsday dates for non-leap year:



Figure 55

Figure 56 shows that the module `week_day_detector` knows Doomsday dates for leap year:

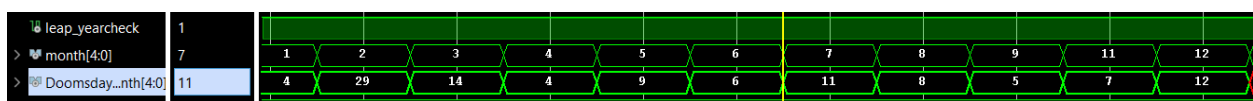


Figure 56

Figures 57 shows that module `week_day_detector` can identify Century Codes for 2700, 2800, 2900, 3000:

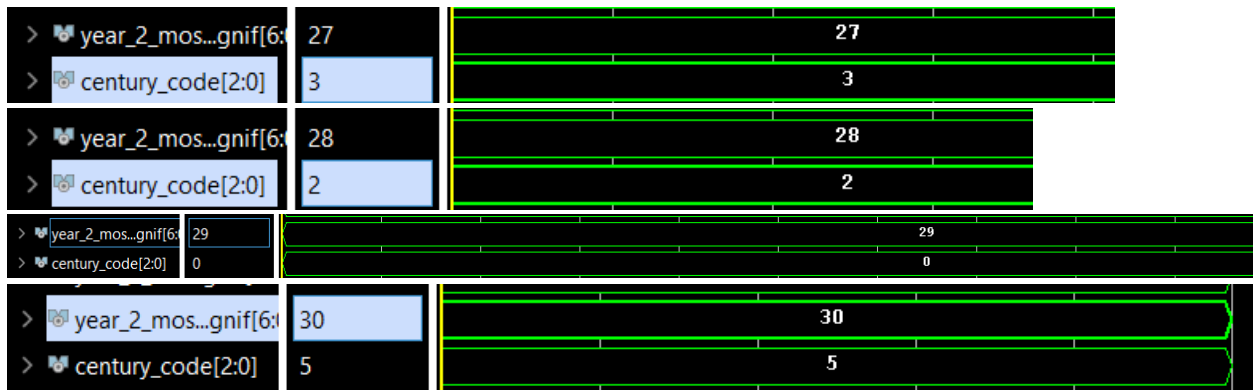


Figure 57

Figure 58 shows that the module week_day_detector can correctly determine weekdays.

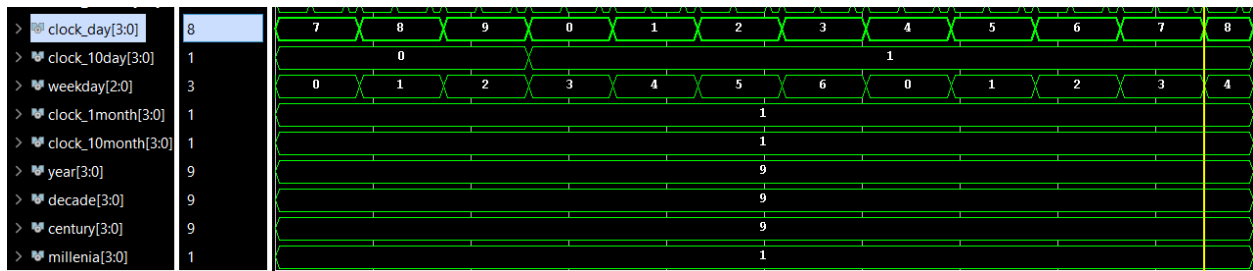


Figure 58

Weekday LEDS work

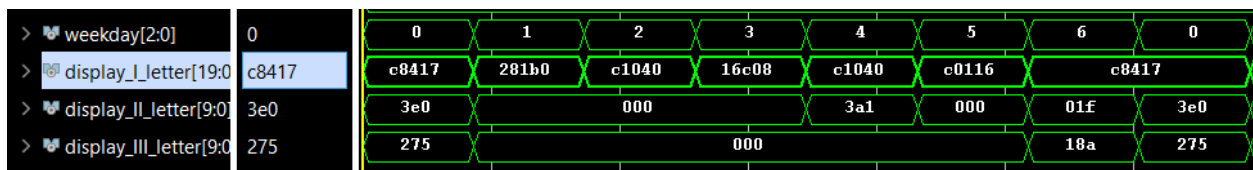


Figure 59

All of those modules are instantiated in Top_module for digital clock. In top module, I have a signal nycountdown which becomes 1 10 seconds before midnight. If nycountdown is 1, 7-segment BCD counts down from 9 to 0. Meaning when input is 1, module shows 9, when input is 9 the module shows 1. See figure 60:

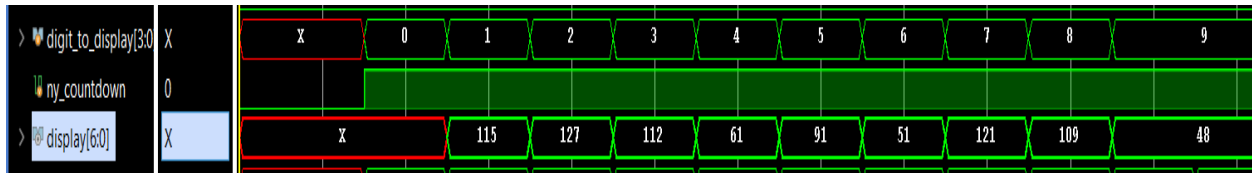


Figure 60

Otherwise, the module counts up:

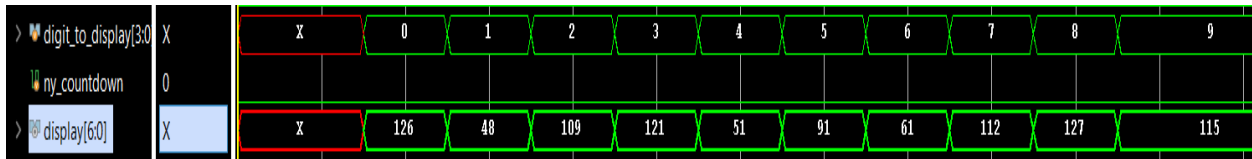


Figure 61

Modules associated with Combination lock:

The lock has a sequence detector with 5 states: s0, s1, s2, s3, s4. S0 is a reset state. Lock reset signal resets state to s0. The sequence detector has 4 bit with array variable combination. The array has 4 4bit long numbers. Those numbers are the correct combination. The module takes two inputs – input number and number position. Number position selects one of combination array's value. The chosen value is compared to input number. If correct number is inputted, the state advances, otherwise the state goes to reset state – s0, and stays there till the end. If all input numbers are correct, then the door opens. Figure 62 shows that correct combination, 0123 in our case, unlocks the door:

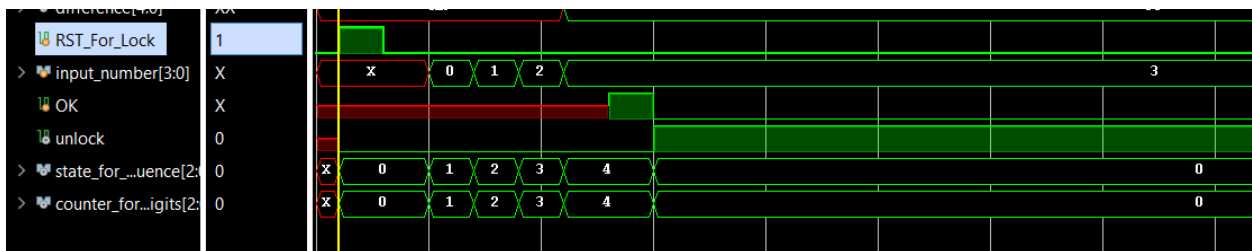


Figure 62

Door locks itself whenever we physically lock it. See Figure 63

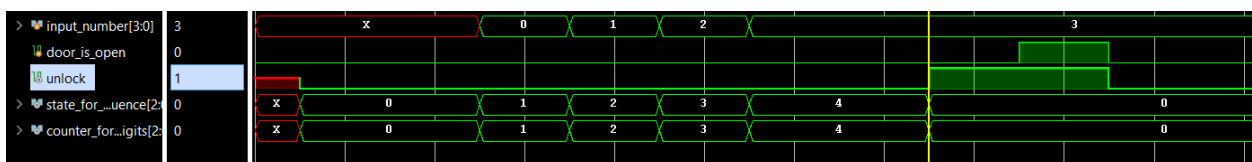


Figure 63

Figure 65 shows that lock does not unlock if wrong combination is entered:

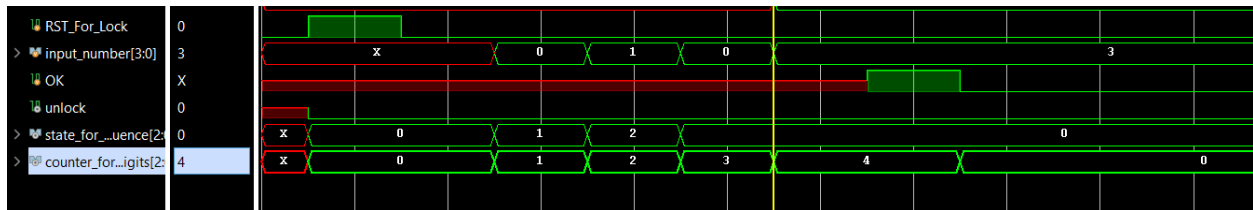


Figure 65

The module keypad_for_lock represents 0-9 buttons and OK button. This keypad has two functions. If the door is physically open, pushing buttons on the keypad will set new unlock combination. If the door is closed, the user enters unlock combination with that keypad. The keypad generates a pulse after its button is pressed. Sequence detector uses that pulse as a clock. The keypad also returns position of input number. Input number will be compared to value of variable combination at that position. Figure 66 displays the process of setting the combination and unlocking the door. The lock is unlocked after the user enters correct combination and presses OK.

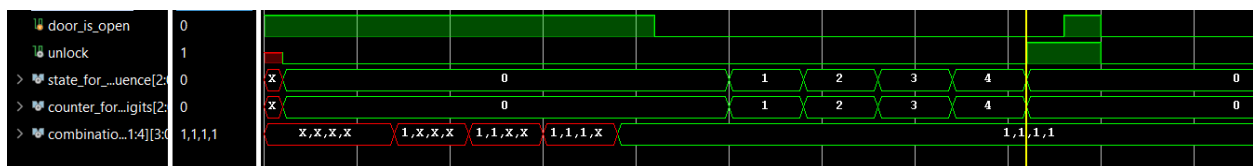


Figure 66

Modules associated with Dishwasher Gas Cooker timer and start:

Timer takes entered numbers generated by the keyboard and upon pressing start it begins count down. When countdown reaches to 00:00, signal finish is set to 1 and the power is turned off. Timer is instantiated four times to represent h10h:m10m. Those timer digits are displayed on 7-segment BCD which is also instantiated 4 times. If power is off, the LEDS are turned off as well. Figure 67, 68 and 69 represents timer values counting down from 10:00 to 00:00



Figure 67

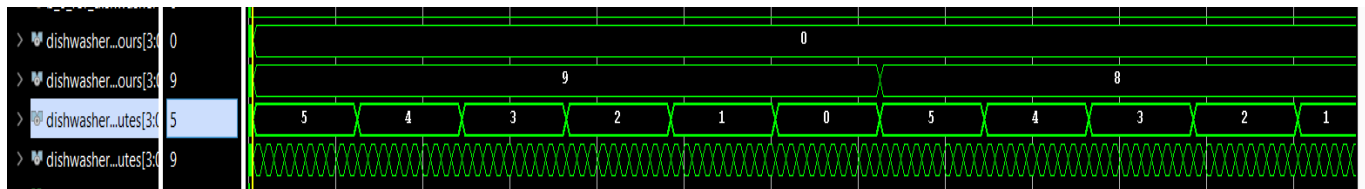


Figure 68

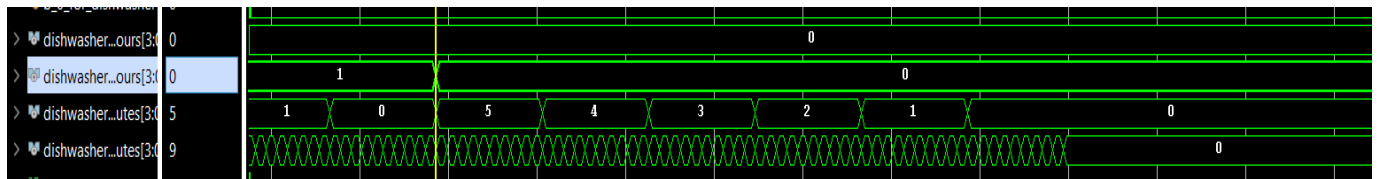


Figure 69

Figure 70 shows that LEDS are of 0, before power is on:

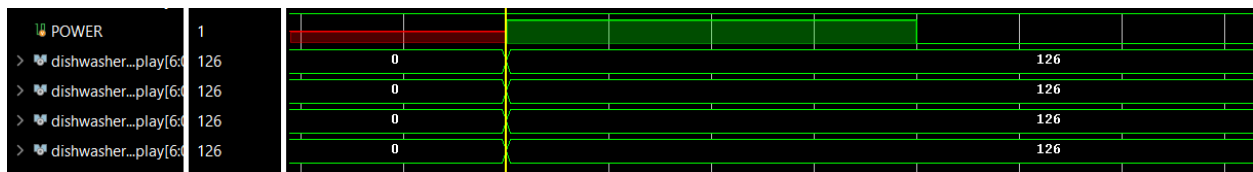


Figure 70

Figure 71 proves that LEDs show timer values correctly

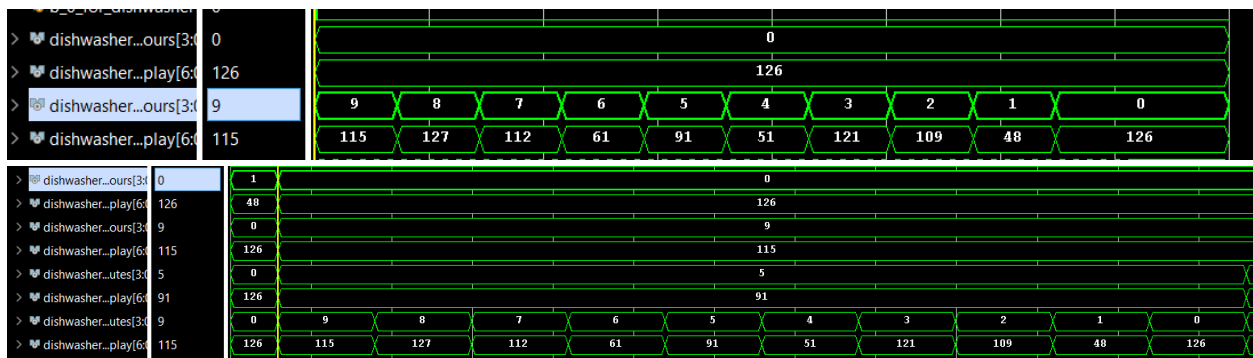


Figure 71

Figure 72 shows that when finish becomes 1, power is turned off.

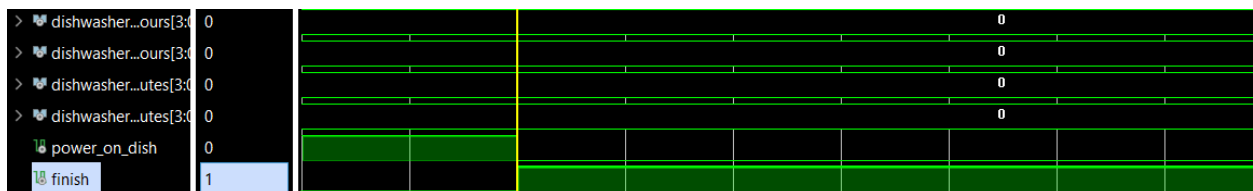


Figure 72

Modules associated with garage door:

Figure 73 shows that garage opens or closes at every pulse generated by pressing open_close button. Figure 73 also shows that FSM works correctly if initially the value of input is_object is 0.

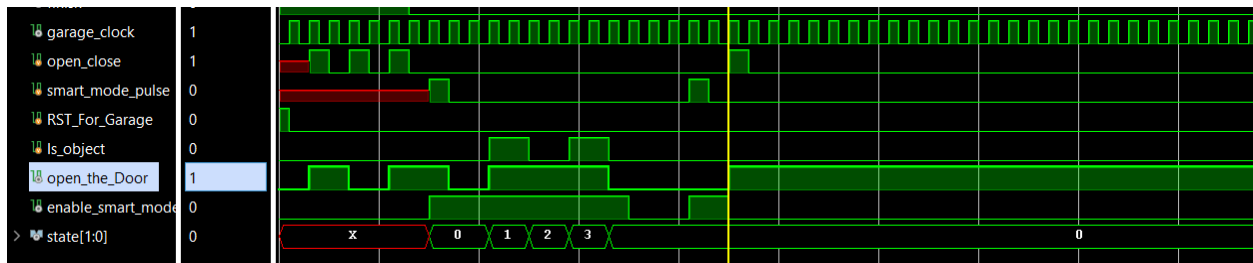
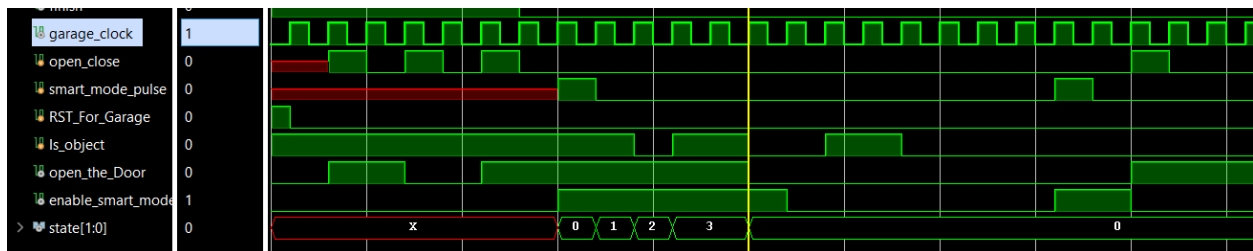


Figure 73

Figure 74 shows that FSM works correctly if initially the value of input is_object is 1.



. Figure 74

8 – 9) See the attachments.

10) Special Items used in project:

In my design I used counters, FSM, one memory element, and combinational logic circuits. The specific devices that are described by my project are digital clock, timer, device that detects if the year is leap, device that can identify weekday based on the year, device that gives us quotient and remainder of a ratio. Review the second part of the report to see schematics of my project.

11) My Design was not implemented on Hardware.

12) What would I do differently if I had time

First of all, I would add some more features to my house. I would make lock combination change in every 24 hours. I would use LFSR to generate 4 new values for combination. After writing this project, my ability to divide the problem into chunks of sub problems improved. If I had time, I would reduce size of the code by modifying them so that I could implement the same module several times.

13) Conclusion:

The report explained what my Verilog modules do and how they do it. Overall, I wrote 4 top modules and 16 modules for them. I provided block diagram for each of those top modules and explained the code in detail. Moreover, I provided commented source code. I have tested modules and I included resulting waveforms in the project.

14) Hours spend on the project.

This project was massive undertaking and it required a lot of thinking and testing of modules. The project is time sensitive, I had to come up with ways to test modules that are supposed to have the same output for years. That is why I spend approximately 168 hours on this project. Almost 30% of that time was spent on testing modules.