FPGA Alarm Clock Implemented using Verilog

Kyle Acosta
Electrical and Computer Engineering
California State Polytechnic University
Pomona, USA
kaacosta@cpp.edu

Jimmy Luu
Electrical and Computer Engineering
California State Polytechnic University
Pomona, USA
jluu203@gmail.com

Rob Ranit
Electrical and Computer Engineering
California State Polytechnic University
Pomona, USA
rranit@protonmail.com

Jeff Tang
Electrical and Computer Engineering
California State Polytechnic University
Pomona, USA
jefft1840@gmail.com

Abstract— This project showcases the successful development of an advanced FPGA-based smart alarm clock, representing the culmination of collaborative efforts within the ECE 3300 course. Leveraging the capabilities of Xilinx Vivado ML Edition (2023.1) software and implemented on the Nexys A7-100T FPGA board, our project introduces significant advancements in hardware integration, coding proficiency, and user-centric design.

The Clock module, a pivotal component, showcases refined features including a 24-hour display with precise timekeeping capabilities. Iterative enhancements, such as a revamped minute counter, expanded hour display (00-23), and an optimized loading mechanism, contribute to the project's commitment to achieving precision and functionality.

Simultaneously, the completed Alarm module incorporates innovative features, including LEDs and a switch for dynamic input control. Integrated into the comprehensive AlarmClock module, these features provide users with a holistic wake-up solution, emphasizing user customization with load number LEDs and up counter indicators.

Our version-controlled repository ensures efficient tracking of changes across three distinct components: Alarm, Clock, and the Combined AlarmClock. The adoption of Verilog as the coding language underscores our commitment to meticulous hardware-level programming, establishing a versatile foundation for future FPGA applications.

In summary, this FPGA-based Smart Alarm Clock project not only showcases the seamless integration of advanced features using Xilinx Vivado and Nexys A7-100T but also underscores the significance of precise version control and hardware programming methodologies. Our submission contributes to the discourse on embedded systems, highlighting successful innovations in FPGA technology for a more personalized and intelligent wake-up experience.

Keywords—FPGA, Nexys A7, alarm clock, Verilog, 24-hour

I. Introduction

Field-Programmable Gate Arrays (FPGAs) are applied in many different industries [1]. Proficiency in designing Verilog logic, programming, and testing FPGAs is essential for electrical and computer engineers.

The Xilinx Nexys A7-100T, is an Artix-7 FPGA, containing many different inputs and outputs specifically designed for the educational level. By combining different basic Verilog projects, it is possible to design a fully functional alarm clock. Through the design process, students and any engineers are able to test their skills using switches, light emitting diodes (LEDs), 7-segment displays, buttons, video graphics array (VGA), audio output, pulse-width modulation (PWM), and system clocks.

The core of this project is a 24-hour clock, ranging from 00:00 through 23:59. The time is displayed on the four 7-segment displays to the left. The alarm utilizes the remaining displays to the right. Functions such as setting the time or enabling the clock/alarm are applied through switches and 5 buttons. The VGA port is utilized to show the 24-hour clock with seconds on any VGA compatible monitor.

II. 24-Hour Clock

A. Up-Counter

The up-counter module counts from 0 while the *clk* and *en* signals are high. This will continue until the set maximum. Depending on the digit, the max will either be 2, 5, or 9.

TABLE I. Up Counter Inputs/Outputs

Register Size	**Module Directions**				
	Input				***Output***
1-bit	clk	rst	en	load_en	flag
4-bit	load_num				Out

Once the counter *out* reaches the highest value allowed, the next clock cycle will have the counter reset back to 0. When *out* is at the max, the *flag* will output a 1; otherwise, *flag* is 0.

Within the counter module is logic for the *load_en* and *load_num* inputs. While the *en* and *load_en* signals are high, the *out* will be set equivalent to *load_num*. However, if *load_num* is greater than the set maximum, then *out* is capped at the given maximum.

B. Incrementing Clock Digits

The clock uses a total of 8 up-counters. Four are used for the clock, while the remaining counters are used to set the alarm. Each digit is designate a counter, 2 counters for the hour and the other 2 for the minute.

The min_0 counter is on a 1-minute clock cycle, so every 60 seconds the output will increment by 1. Once the value of min_0 is 9, the counter will reset to 0 during the next clock cycle, and increment min_1. Min_1's enable is min_0's flag, so only when the flag is high will min_1 be enabled.

For hr_0, it will increase only when both min_1 and min_0 reaches their respective maximums; 5 and 9. Hr_1

uses this same principle, and the counter is enabled only when the other counters have the flags high.

TABLE II. DIGIT LIMITS

Clock Digits	Counters	
	Range of numbers	***Reset Clock***
hr_1	0 – 2	2
hr_0	0 – 9[a]	3
min_1	0 – 5	5
min_0	0 – 9	9

[a] HR_1 has a max of 9, but needs to reset when is equivalent to 3

TABLE III. LOAD LEFT/RIGHT

Case	8-bit Output
0	0000 0001
1	0000 0010
2	0000 0100
3	0000 1000
4	0001 0000
5	0010 0000
6	0100 0000
7	1000 0000

C. 24-Hours Reset

The clock displayed according to the maximum of each counter is 29:59. This does not correspond to the 24-hour clock. So, to ensure that clock returns to 00:00 after 23:59, all counters must reset when it meets this condition: clock enable and flags min_0, min_1, & hr_1 are high; in addition, hr_0 is greater than or equal to 3. The 24-hour reset will clear all counters in the next clock cycle when counters are displaying 23:59.

The other condition to reset all counters occurs while hr_1 flag is active and hr_0's output is greater than 3. This is necessary it is possible to set a time over 23:59, without this limitation. The moment the user disables clock_load and the hour is over 23, all counters are reset.

D. Setting the Clock and Alarm

The Nexys A7 has a limited number of switches, so it is not possible to use the 16 switches to set the value of each digit. Instead, we used two different load modules: load_ud & load_LR.

Load_ud is to increase or decrease a value through the up or down buttons. The up button adds 1, while the down button subtracts 1. If the number increases greater than 9, the number is set to 0. On the other hand, if less than 0 the number is set to 9. The center button is used to reset the value in load_ud.

Load_LR allows the user to use the left and right buttons to change the counter to enable the load function. This utilizes a temporary 3-bit register and a case statement to shift. When the left button is pressed, the register increases by 1 and decreases by 1 when the right button is pressed. While clock load is high, the range for the register is 4 to 7; if out of this range it is set to 4. The alarm load range is 0 to 3; if out, then set to 3. The case statement is utilized only when either the clock or alarm load is enabled, else output 0. The case uses the temporary register to determine which 8-bit output is used. The output only contains one of the bits high. When all counters use one of the 8 bits as the load and up-counter enable, the user can switch between digits and change the value of a single digit.

E. LEDs

Each of the switches has a corresponding LED, e.g., switch is off, then the LED is off. The exception is the LED on/off switch, while the switch is off its LED is active to indicate that all LEDs are active. To disable all LEDs, flip the switch on.

While LEDs are in use, loading the clock or alarm uses LEDs 7 to 4 to signify the digit currently with load enabled. LED 7 represents hr_1 and LED 4 represents min_0. If clock_load is enabled, the LEDs illustrates load_LR's output of 7 to 4; for alarm_load, LEDs display output 3 to 0. LEDs 3 to 0 display the load_ud value in binary. For example, a load_ud value of 7 will be shown by LEDs as: 0111.

F. Timing

The 24-hour clock uses three different frequencies: 2 Hz, 1 Hz, & 60 seconds. The 2 Hz & 1 Hz is based on the Digilent clock divider Verilog code [2].

The 2 Hz clock signal is used for load_ud and load_LR. As 2 Hz is equivalent to 0.5 seconds (secs), it will require the button to be held down for that amount of time. This was determined to be the best timing for the load function as if the clock is too fast, it is unable to be controlled. Conversely, a slower clock would represent a greater delay when increasing/decreasing or changing the digit.

The 1 Hz signal is used in conjunction with the minute counter (60 secs). 1 Hz corresponds to 1 per second. We can create a minute signal by instantiating a counter increasing every 1 Hz and ranging from 0 to 59. Whenever there is an output signal at 59, this now becomes a clock with a frequency of one minute. The minute counter can be reset be either the complete reset, minute reset switch, or clock load. This ensures that when setting a time, the minute counts from 0.

For all clock digits, the clock it uses depends on the clock load input. While clock load is high, all digit counter modules switch to the system clock of 100 MHz. Otherwise, use the minute clock. The alarm counters use something similar, depending on whether alarm load is active.

The 24-hour reset uses the system clock while clock load or clock counters are greater than 23:59. All other times the reset logic is running at the minute. This ensures that 23:59 will reset to 00:00 after a minute and keeps the reset high only when the clock is not in range.

III. ALARM

A. Sound

The alarm sound module implements a basic segment of music using a case statement. By utilizing the 100 MHz clock frequency and PWM, any musical note and its octave can be played through the mono audio output. For example, the note C4 (middle C) is at the frequency of 261.6 Hz and dividing the clock frequency by the note frequency will result in the correct pitch.

However, the division of the note frequency by the whole clock frequency produced a lower octave of the same note. Therefore, lowering the frequency to 50 MHz resulted in the desired pitch. This occurs because the note's period has been halved, and the duration of each note cycle has been decreased, leading to a higher pitch.

For the duration of the note, values were assigned corresponding with the clock frequency being divided by 8, which was the most optimal value for the note period when testing. For example, quarter notes have a value of 2, half notes have a value of 4, and whole notes have a value of 8. The notes are stored in a 20-bit output register and the duration in a 5-bit output register which is used in the alarm player module.

B. Playing the Music

The alarm player module uses the 100MHz clock (denoted as clk) along with the alarm sound module to produce the music through the mono audio output. The aud_sd output is set to high and acts as an enable for the audio. The audio_out output is responsible for outputting the note frequencies.

The 20-bit counter register is incremented by 1 for every positive edge of clk. When counter value is more than the note frequency's binary representation, counter will reset to 0 and toggle audio_out, resulting in a square wave at the desired frequency.

As stated previously, the duration of the note is passed from the alarm_sound module and multiplied by the clock frequency divided by 8. This value is placed into a 32-bit register denoted as noteTime to account for any duration of time. Similarly, to the counter register, the time1 register increments by 1 for every positive edge of clk. When the time1 value is more than the noteTime value, noteTime will reset to 0 and increment the case statement by 1, moving on to the next note. For example, a quarter note will only play for 2 of the 8 note time period of the clock. This implementation allows for sustaining or stopping notes to mimic tempo and musical notation.

To stop the alarm, the player_en input is set to low which stops the incrementing of counter and time1 as they are set to 0 until the player_en switch is set to high.

C. Alarm LEDs

The alarm clock features two operational RGB LEDs: one indicating whether the alarm is active, and another that flashes when the clock matches the set alarm time. Both LEDs utilize PWM to function.

1. Alarm Enable RGB LED

This LED uses a 20kHz clock divider, implemented using Digilent's clock divider Verilog code [2]. The clock divider needs to be relatively high frequency in order for the LED to remain constantly illuminated. Additionally, the LED features an instantiation of the PWM_Core module, which generates a PWM signal based on the provided clock and duty cycle. The PWM_Core module serves as an LED controller. If all three LEDs, red, green, and blue, need to be used in a project, the module can be instantiated three times and the intensity of each LED can be controlled. For this project, only the green led of LD17 on Nexys A7-100t needs to be activated. The duty cycle is hard coded to be 12.5% so that the intensity of the green LED is not too bright. The load is hard coded as well so that duty cycle gets loaded onto the LED. To activate the LED, the alarm enable switch needs to be switched on. The LED serves as a signal that the alarm is active and will go off when the clock matches the alarm. Switching off the alarm enable switch will turn off the LED.

2. Alarm Match RGB LED

The LED in this module is managed by the alarm_match_rgb_led module, which controls the LED's flashing behavior when the clock matches the set alarm time. The module utilizes the default 100MHz clock, reset, and enable inputs from the alarm_player module. Within this module, two counters are implemented: a PWM counter and a regular count counter. The count counter increments based on the 100MHz clock and toggles the state of each color LED. The red LED is assigned a predefined duty cycle of 99% through the PWM counter, which, in turn, is incremented by the count counter. When the clock is reset or the alarm enable is turned off, all LEDs are turned off as well. As the count counter increments with the enable turned on, the green and blue LEDs toggle between on and off. Simultaneously, the red LED toggles every clock cycle. Due to the specified duty cycle, the red LED remains on, creating the appearance of the LED flashing between red and white.

IV. VGA

A. Displaying the Clock on a Monitor Through VGA

The VGA controller module sets up and calculates the necessary values for the board to display the clock values to an analog monitor. This project utilizes a 640 x 480 resolution. It partitions a screen size of 800 x 525 pixels into sections, ultimately ending up with a 640 x 480 resolution. This module also assumes a 60 Hz refresh rate, and the values for each porch and sync pulse are in Table IV below. The module ensures that the horizontal and vertical signals are synchronized with the system clock. From this system clock, a 25MHz signal is generated to increase counters for the horizontal and vertical signals. Whenever the horizontal or vertical counts are in their respective retrace areas, the *hsync* and *vsync* outputs will be set to 1, accordingly. There is also a *video_on* signal that is 1 only when the pixel counts (the counters for horizontal and vertical signals) are within the display area (resolution + back porch values).

TABLE IV. PIXEL VALUES, 640 X 480 RESOLUTION, 60 HZ REFRESH

Type of Value	Pixel Value
Horizontal Front Porch	48
Horizontal Back Porch	16
Horizontal Retrace / Sync Pulse	96
Vertical Front Porch	10
Vertical Back Porch	33
Vertical Retrace / Sync Pulse	2

The clock digit ROM module contains all the ASCII codes for the digits 0 – 9 and the colon symbol, utilizing a BRAM to store this. Each symbol takes up 32 x 64 pixels worth of space on the screen. This module takes in an 11-bit address signal that is a concatenation of two registers that store the location of the corresponding clock digit or symbol to display onto the VGA, scaled appropriately. For example, if the clock is currently showing 1:27, each individual BCD value from the clock (and the colon) is put into this module to choose 1, :, 2, and 7 in the ASCII format to display in pixels on the monitor, scaled to 32 x 64 pixels for each digit.

The pixel clock generator handles all operations dealing with specific pixel values, including pixel location, and pixel color. Each pixel of each digit is displayed on the screen when the horizontal and vertical counters are in the range of each digit's location of pixels on the screen. By default, these clock digits are shown in red. If the alarm is on, both the clock and the alarm time will turn green until the alarm is turned off. In Table V below, each digit's pixel range is shown. This module will update the clock value display on the VGA output as it instantiates the clock digit ROM module mentioned earlier to output the ASCII characters of the clock values.

TABLE V. PIXEL RANGES FOR EACH DIGIT

Value to Display	X, left	X, right	Y, top	Y, bottom
Clock, Hours 10s digit	224	255	192	256
Clock, Hours 1s digit	256	287	192	256
Clock, Colon	288	319	192	256
Clock, Minutes 10s digit	320	351	192	256
Clock, Minutes 1s digit	352	383	192	256
Alarm, Hours 10s digit	224	255	320	384
Alarm, Hours 1s digit	256	287	320	384
Alarm, Colon	288	319	320	384
Alarm, Minutes 10s digit	320	351	320	384
Alarm, Minutes 1s digit	352	383	320	384

V. CONCLUSION

In conclusion, this project capitalizes on the versatility of the Nexys A7-100T. Utilizing clock division, PWM, LEDs, seven-segment displays, and VGA, this 24-hour alarm clock is a great example of what can be created through the usage of FPGA boards and Verilog.

REFERENCES

[1] Xilinx, "What is an FPGA? Field Programmable Gate Array," AMD, https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html.

[2] Digilent, "Counter and Clock Divider," Digilent Reference, https://digilent.com/reference/learn/programmable-logic/tutorials/counter-and-clock-divider/start.