# RGB LED Color Control Using PWM

An FPGA-Based Real-Time Color Modulation System Driven by Accelerometer and Switch Inputs on the Nexys-4 DDR

Yousif Fatohi, Andrew Kashat, Sedim Baffrow, Peter Younan
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: yfatohi@oakland.edu, andrewkashat2@oakland.edu, sedimbaffrow@oakland.edu, pyounan@oakland.edu

***Abstract*—This work presents an FPGA-based color-control system, which uses pulse width modulation to control the brightness of an RGB LED based on data collected from an accelerometer via an SPI interface. The system works on the Digilent Nexys-4 DDR board, collecting X-axis, Y-axis and temperature data and mapping them to the red, green and blue color channels accordingly. A Map To Max circuit converts each sensor value into a PWM duty cycle by taking the absolute value, applying a fixed scale factor of sixty-four with saturation and combining the result with a switch-selected brightness floor. The blue temperature channel also subtracts a baseline value before scaling so that normal room-temperature readings do not force the LED to full brightness. The overall system is organized into a separated FSM-based control unit and datapath with the required accelerometer interface, holding registers, Map To Max blocks and PWM generators. The design was verified in simulation and realized on hardware.***

## I. Introduction

This document discusses the design, implementation and verification process of a FPGA-based color control system. This design reads accelerometer data and temperature from an on-board sensor and generates corresponding brightness control signals in the form of PWMs. The scope of the report includes system architecture, internal structures of the control unit and datapath, the external interface with the sensor, user-switches based brightness floor control, simulation strategy employed to verify the design, and hardware results obtained on the Digilent Nexys-4 DDR board [2], [3].

Motivation for the design is based on combining several concepts presented in class into a functional project. The work uses two-process FSM approach, datapath and register-level control design techniques, reusable PWM generators introduced throughout the class. In addition to that, the work required research of ADXL362 accelerometer SPI communication protocol and register map, as well as practical knowledge of the Nexys-4 DDR board [1]. It turned out to be valuable to have something tangible to connect theoretical knowledge of synchronous digital design to - the changing LED color was directly related to changing sensor readings and switch settings.

The rest of the report is structured in accordance with the design methodology employed throughout the work. The methodology section describes the high-level system architecture, internal structure of each block and interaction between the control unit and datapath. The experimental setup section gives information about the tools and hardware used for verification of the design. The results section provides simulation waveforms, results of testing the hardware on the board and details on solving the occurring problems during the process. Discussion and conclusion section concludes the report and describes possible improvement of the design.

## II. Methodology

This section gives complete description of the design methodology and architecture. The design follows the traditional control-and-datapath architecture common for class projects [1]. The control unit is an FSM, coded in the two-process style, with synchronous low-active reset. The datapath consists of accelerometer interface, holding registers, Map To Max blocks and three PWM generators. All newly introduced VHDL sources use IEEE.NUMERIC_STD and do not create inferred latches by default setting their outputs in FSM [1], [5].

To keep the design manageable, the project was broken into smaller functional blocks and verified step by step before full integration. The accelerometer interface was used to obtain the sensor data, the control FSM was used to sequence which value would be stored, and the holding registers were used to preserve stable values for the RGB channels between sensor updates. After that, the Map To Max blocks converted the stored sensor values into duty-cycle values, and the PWM modules drove the LED outputs. Organizing the project in this way made debugging more practical, since each part of the design could be checked separately before being connected into the complete top-level system. This was useful because the project involved both simulated signals and real hardware behavior, which do not always look exactly the same during testing. By separating the design into clear blocks, it was easier to compare the simulation results with the board behavior.

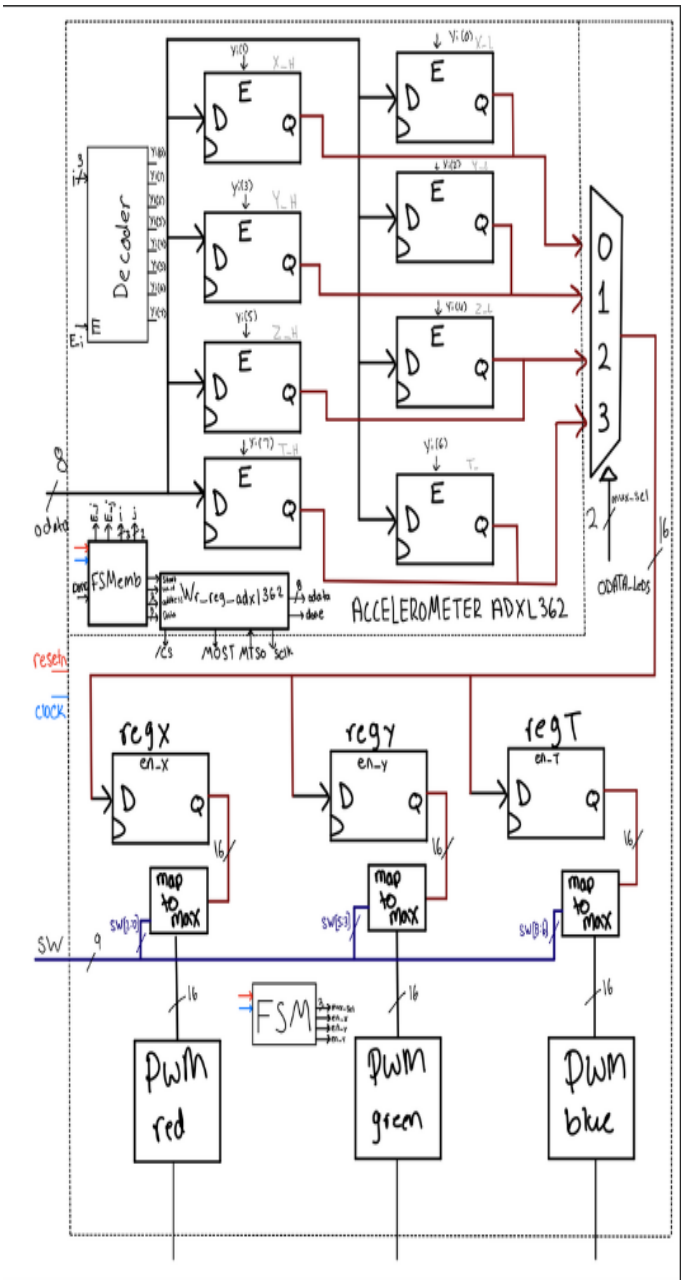


***Fig. 1***. *Top-level block diagram of the RGB LED color control system, showing the ADXL362 accelerometer interface, the control FSM, the three 16-bit holding registers, the three Map To Max blocks, and the three PWM generators driving LD16_R, LD16_G, and LD16_B.*

### *A. System Architecture*

It comprises six blocks: accelerometer interface (accelerom using FSMemb and Wr_reg_adxl362), three 16-bit holding registers (reg_x, reg_y, and reg_T), control FSM (fsm_rgb_ctrl), three Map To Max blocks, three PWM generators (mypwm), and the top-level wrapper module (top_rgb_ctrl). The control FSM cycles through three states to latch X, Y, and temperature words to their respective 16-bit holding registers. State 1 selects and latches the X-axis value, State 2 selects and latches the Y-axis value, and State 3 selects and latches the temperature value. These words then pass through their respective Map To Max block and then PWM generator driving the corresponding RGB LED pin. The X axis reading drives the red channel, Y axis drives the green channel, and the internal temperature drives the blue channel. The whole block diagram of the system can be seen in Figure 1.

### *B. Accelerometer Interface*

The ADXL362 is a three-axis digital low-power accelerometer and the device reports the internal temperature too. Communication with the device is done via SPI protocol, and it measures are made available in byte-wide registers that consist of 16-bit words representing X, Y, Z axes and temperature readings [3]. Accelerometer interface used in the project is accelerom, a course component that consists of two modules. FSMemb controls the order of register address, and Wr_reg_adxl362 is a SPI byte-level communication engine that drives CS, MOSI, MISO and SCLK signals and outputs the read register contents on an 8-bit data bus. Inside accelerom, a decoder enables one of the byte-wise holding registers that enables storing of low and high byte of each reading to the correct place. Four 16-bit words are then connected to a 4-to-1 mux whose selector is driven by the external FSM controlling FSMemb and Wr_reg_adxl362. The re-use of accelerometer interface reduced the amount of SPI level low-level work significantly.

### *C. Holding Registers*

Three 16-bit holding registers reg_x, reg_y and reg_T are used after the 4-to-1 accelerator mux and before Map To Max blocks. They are D-type flip-flop banks where data input is connected to the 16-bit output of 4-to-1 accelerator mux and enable is connected to respective pulses from the control FSM. As FSM moves through the cycle sequentially through X, Y and temperature, three registers are used to hold the values from one cycle until the next to make sure there is no flickering effect in LED. These registers hold the last read values to ensure constant value of PWM duty cycle between reads.

### *D. Control FSM*

A three-state Moore type of control FSM (fsm_rgb_ctrl) sequences the latching of the three words of interest to the registers. State 1 sets mux_sel = "00" and en_x = 1 to latch the X value in reg_x. State 2 sets mux_sel = "01" and en_y = 1 to latch the Y value in reg_y. State 3 sets mux_sel = "11" and en_t = 1 to latch temperature in reg_T. Selecting "11" value is required since the fourth word represents temperature in 4-to-1 accelerometer multiplexer. From State 3 the FSM returns to State 1 to repeat the latch sequence. Control FSM was implemented in the two-process style: one combination process generates the next state outputs while one synchronous process updates state register on the clock rising edge.

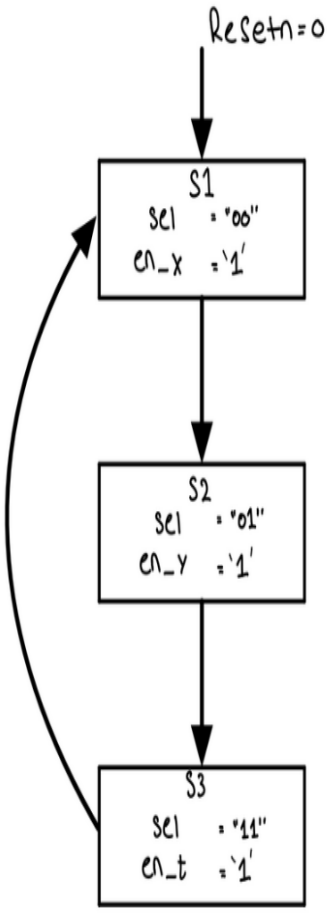


***Fig. 2****. State diagram of the control FSM (fsm_rgb_ctrl), showing the latch sequence, the mux-select value driven in each state, and the enable signal pulsed for the corresponding holding register. In the temperature state, mux_sel must be "11."*

*E. Map To Max Block*

Each color channel has its own Map To Max block. This block is combinational only and performs three tasks. The first task is to subtract the channel baseline and take the absolute value of the signed 16-bit sensor word so that positive and negative changes both increase brightness. For both red and green values, the baseline value is 0. In the case of blue temperature, the base line value of 150 is subtracted such that room temperature output values become zero and only then scaled.

The second step is to left-shift the number by six bits, making it multiply by 64. If the output exceeds the maximum limit of 16-bit duty cycle, then it saturates to 0xFFFF. This prevents wraparound and makes full tilt or high temperature produce predictable maximum brightness.

The third step is comparing the normalized value from the sensor with the threshold set by the switches for the corresponding color channel. The values set by the switches for each channel are SW[2:0], SW[5:3], and SW[8:6] for the red, green, and blue channels respectively. A setting of "000" to any of the channels will result in the output being 0. If the switch group is "001", the floor is 0x1000, which looks about 25% bright. If the switch group is "010" or "011", the floor is 0x4000, which looks about 50% bright. If the switch group is "1xx", the floor is 0x9000, which looks about 75% bright. In this manner, we fix the period of PWM and let the switch floor and the live sensor value determine the final duty cycle through the maximum value.

*F. PWM Generation*

Three PWM generators employ mypwm.vhd module directly. Every generator runs from a constant period of TPWM=65535 clock cycles and gets its 16-bit duty-cycle input signal from the Map To Max block of the respective color channel. The red PWM duty cycle is generated from max(floorR, |x_reg| x 64). The green PWM duty cycle is generated from max(floorG, |y_reg| x 64). The blue PWM duty cycle is generated from max(floorB, |t_reg - 150| x 64). Each value is saturated to the 16-bit maximum before being sent to the PWM generator. With a fixed PWM period, the switching frequency will stay above the threshold of flickering for the human eye, and the output brightness will be controllable by adjusting the duty cycle. By employing the same module three times, but feeding different input signals, we achieve symmetric design whose channels exhibit similar timing characteristics during simulation.

*G. Top-Level Integration and Constraints*

The top-level module (top_rgb_ctrl) instantiates the accelerometer interface, the control FSM, three 16-bit registers to hold sensor values, three Map To Max blocks, and three PWM generators. The design constraints rgb_ctrl.xdc assigns the 100 MHz system clock, active-low reset button, nine switches, four SPI pins which are connected to ADXL362, and three RGB output signals (LD16_R, LD16_G, and LD16_B). One important note is that switch SW[8] is connected to the Bank 34 of Artix-7 device, which operates at 1.8V voltage on the Nexys-4 DDR board. Its correct IOSTANDARD must be LVCMOS18 and not LVCMOS33, otherwise, synthesis might fail [2], [4]. Unused XDC lines are marked out instead of removed so that they can be used for debugging if needed. Team members decided to reuse accelerom, FSMemb, Wr_reg_adxl362, and mypwm modules from the course library. Control FSM, Map To Max block, top-level integration, constraints file, and testbenches were designed specially for this project.

*H. Timing and Dataflow Considerations*

A critical choice of architecture is related to keeping the process of reading from accelerometer separated from RGB output process. Accelerometer interface generates one 16-bit value at a time, whereas RGB LED needs three values always to be ready. Three holding registers resolve this timing problem in our case. Once X, Y, and temperature

values are captured in registers, PWM generators can operate continuously from stable inputs while the next sensor reading occurs in the background. This approach avoids dependence of LED brightness on exact moment of SPI engine updates of its internal byte registers. In addition, this architecture allows one to easily debug because control outputs do not depend on PWM waveforms and can be examined separately.

The datapath was constructed in such a way that each color channel possesses a repeated structure beyond the registers. Red, green, and blue pass first through their own Map To Max blocks and then through corresponding PWM generators. Repeating of blocks reduces a risk of error caused by specific channel's logic and facilitates data simulation as well. For example, if one PWM channel works properly for a particular duty-cycle input, the other two must behave identically if their inputs differ. Such a structure greatly helped us to debug the design since we could compare results across the channels.

### *I. Switch-Based User Control*

These switches are used to make sure that there are no further communication blocks between the system and the user. These three groups of switches can therefore be viewed as control unit for brightness of each individual color floor. Switches SW[2:0] will control the brightness of red floor, SW[5:3] the brightness of the green floor and SW[8:6] the brightness of blue floor. A lower setting of the switches can make the channel remain off or operate at a low brightness level. The live sensor value is still used because the Map To Max block compares the scaled sensor value with the switch floor and outputs whichever value is larger. This approach is limited as compared to a look-up table, but it works well within this context.

## III. Experimental Setup

The system was verified in two stages. In the first verification stage, the system was simulated using the tb_top_rgb_ctrl.vhd testbench in Vivado. This testbench instantiates the full top_rgb_ctrl module, creates a 100 MHz clock signal, holds the active low synchronous reset signal low for 100 ns, and then releases it so the registers initialize and the FSM begins in State 1. The testbench uses switch stimulus in addition to a repeating MISO pattern of 0x1A2B3C4D so that the SPI read path produces unique and non-zero values. This helps track signals along regX, regY, and regT throughout the whole process.

For both the switches and the sensors, a variety of inputs are tested to observe the switch floor settings and to make sure that the absolute value, baseline subtraction, fixed x64 scaling, saturation, and max operation work correctly. Signals for observation include the current state of the FSM, the mux_sel, each enable pulse, the content of the X, Y, and temperature holding registers, the Map To Max duty values, and the three PWMs. The simulation was run long enough for steady state to be confirmed by repeating the FSM cycles several times.

For hardware testing, the following checkpoints were selected as a reflection of what the hardware should do: first, on power-up reset, the FSM should initialize to State 1 without generating any unexpected enable pulse; second, the FSM should cycle through State 1, State 2, and State 3 while generating enable pulses for each holding register; third, the mux_sel should reflect the current word to be latched, particularly in the temperature state, where the mux_sel should equal "11"; and finally, changing switches should result in different brightness floor values. With such checkpoints, we were able to clearly identify issues arising from either the FSM or datapath implementation.

The bitstream was generated only after successful synthesis and implementation without any critical warnings related to clocking or reset and/or I/O standards. Once programming was performed successfully, the switches were set for several combinations of brightness floors, and the board was rotated manually. The LEDs' behavior was observed visually. The hardware demo checked that X-axis tilt changed red intensity, Y-axis tilt changed green intensity, warming the chip changed blue intensity, and all switches up caused saturated maximum brightness.

## IV. Results

During simulation, the FSM passed through State 1, State 2, and State 3 as designed. The enables for the X axis, Y axis, and the temperature sensor were generated once in each cycle, and their holding registers sampled the stimulus inputs. Each Map To Max block was driven by the corresponding switch input value. For a "000" switch configuration, the output of the Map To Max block was forced to 0, meaning that the duty cycle of the pulse-width modulation was off for that channel. For the higher switch settings, the duty cycle output followed the selected brightness floor unless the scaled sensor value was larger. For large scaled sensor values, the duty cycle saturated at 0xFFFF.

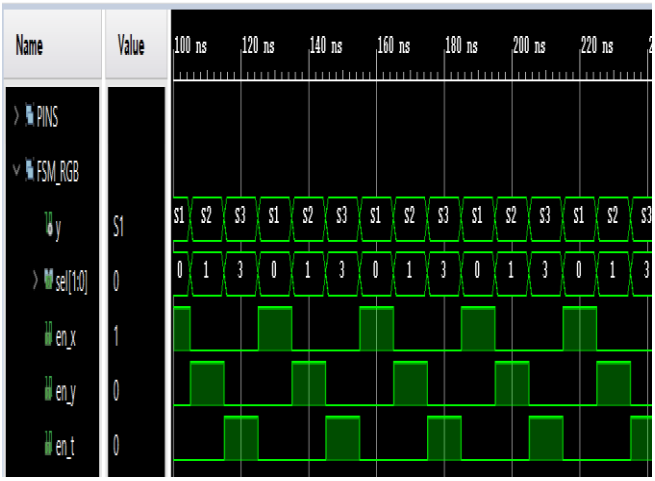


***Fig. 3.*** *Vivado simulation waveform from tb_top_rgb_ctrl.vhd. The trace shows the FSM state, the sel output, the en_x / en_y / en_t enable pulses, the X, Y, and temperature holding registers, and the three PWM outputs over one full control cycle.*

In the hardware, the RGB LED responded according to the intended sensor-to-color mapping. With the board flat, the LED showed a steady baseline color based on the selected switch floors. Tilting the board along the X-axis changed the red channel intensity, while tilting along the Y-axis changed the green channel intensity. Warming the chip by touch caused the blue channel to increase gradually because the temperature reading was mapped to the blue PWM duty cycle. When all switches were placed in the high setting, the corresponding brightness floors caused the LED channels to approach saturated maximum brightness. A picture of the board with implemented design is shown in Fig. 4.

It is noteworthy that the PWM outputs can be used for more than producing fast pulses. More importantly, we can look at the value of the duty cycle of each channel output within one PWM period. When the scaled data gets bigger, the output signal stays high longer and thus becomes brighter. Therefore, in the simulation, enough time should be simulated to get several cycles of the PWM output rather than just a few edges.

One important issue discovered during testing was the DC offset of the ADXL362 temperature output. At room temperature, the temperature reading was already a positive value, so using it directly would cause the blue LED to appear bright even without heating the chip. To fix this, the Map To Max block subtracts a baseline value of 150 from the temperature reading before scaling. This makes room temperature correspond to low blue intensity, while warming the chip causes the blue channel to increase.

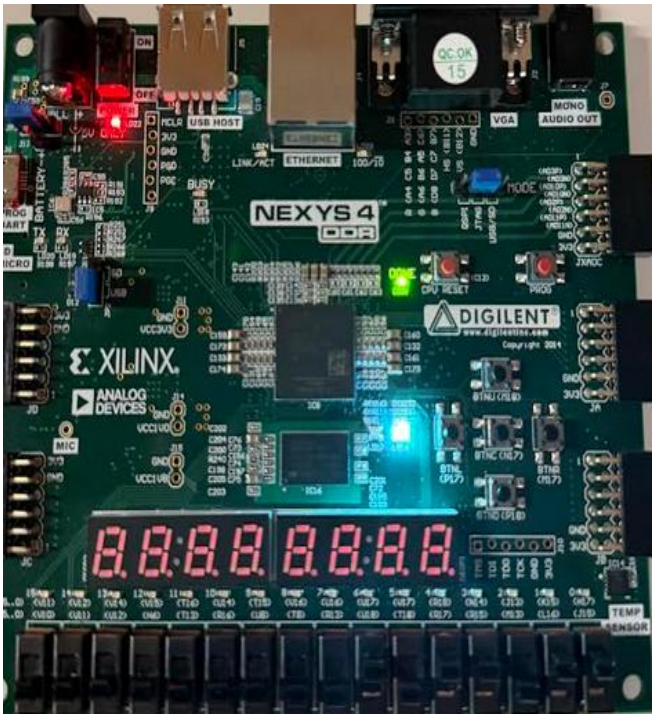


***Fig. 4.*** *The Digilent Nexys-4 DDR board under test. The onboard RGB LED (LD16) reflects the current sensor state as controlled by the brightness-floor switches SW[8:0].*

## V. Discussion and Limitations

Project goals have been accomplished. It has been proven that the RGB system, controlled by multiple sensor inputs through FSM-and-datapath approach, works as expected. The correctness of the control sequence, enable pulses, holding registers, Map To Max conversion, and PWM generation is demonstrated by the results of simulations and verified through a board demo where X-axis tilt controls red intensity, Y-axis tilt controls green intensity, and temperature controls blue intensity.

The main design challenge was calibrating the sensor values so that the LED response was visible and predictable. The X and Y channels could be mapped directly through absolute value, fixed x64 scaling, saturation, and switch-floor comparison. The temperature channel required an additional baseline subtraction because the ADXL362 temperature output does not start near zero at room temperature.

One more possible limitation of the current design is the direct relationship between the channel input and the resulting color of the LED - X axis controls red, Y axis controls green, and the thermometer controls the blue light. Such an approach makes it easy for users to comprehend the system behavior, but it is quite limiting, since the resulting color is highly dependent on the actual range of each of the sensors' outputs. One way to improve the performance of the system is to add normalization for all channels prior to scaling to make the color response more balanced.

This project demonstrates the role that hardware constraints play in FPGA-based designs. Even if the initial design appears to work as expected in simulation, a design will fail to run on hardware if some pins are incorrectly assigned or if the reset signal polarity is wrong or voltages are chosen improperly. That is why checking the board manual, ensuring the compatibility of used I/O standards and trying to verify a few signals on LEDs is quite effective for establishing connections between simulation results and their real-world equivalents.

Another important lesson from the project was that small design choices can have a large effect on the final hardware behavior. For example, keeping the PWM period fixed made the LED output more stable, while using switch-selected brightness floors made the system easier to test because each channel could be forced to a visible level. This helped confirm that the RGB outputs were working even when the live sensor values were changing slowly. Overall, these choices made the design more predictable and easier to demonstrate on the board.

On the whole, this project can be considered quite successful as far as the development process itself goes. From the technical point of view, it integrates a sensor interface, a state machine, holding registers, combinational logic for scaling purposes, pulse width modulation generation, constraint file usage, simulation, and board tests. One of the most natural upgrades of this design will involve adding a capability to observe raw values of each sensor through either a UART interface, a seven segment display or an LED debug mode.

## V. Future Work and Design Extensions

The most beneficial improvement would have been an addition of an LED display of the raw sensor values. While testing the hardware, it was useful to observe the RGB LED response, but this does not fully show how the data changes inside the system. It would help to determine whether the behavior is coming from the SPI transfer, register selection, sign handling, baseline subtraction, scaling stages, or switch floor comparison if the actual values of the X, Y, and temperature words were accessible.

A raw-value display would also make the project easier to present because the audience could see how the sensor readings change in real time. Instead of only watching the final LED color, the user could compare the displayed numbers with the red, green, and blue brightness changes. This would make the connection between the accelerometer data and the PWM output more clear.

It would also be nice to design multiple display modes of the LED display. In the current project, there is only one mapping between sensors and RGB values; however, adding some sort of a mode switch could allow the display to perform different effects using the same hardware. For instance, there can be a mode where the intensity of colors depends on the tilt (X/Y), another mode with only the temperature effect (temperature -> intensity), and also a special diagnostic mode when each LED displays specific bits from the selected sensor word. All of this could be achieved with just a few modifications of the FSM logic.

The way how testbenches were used to verify designs could be improved by incorporating additional checks into the testbenches. For example, in order to test if the register values are changed properly based on the specified register selection and enable values, it could be useful to check if the register is updated or not updated depending on the specified conditions. Additionally, using specific boundary values, such as zero, maximum and minimum integer number, would have helped to verify the correctness of absolute-value calculation, baseline subtraction, saturation, and Map To Max modules.

Lastly, modularity can be achieved by distinguishing between normal and debug modes. Normal mode will involve the use of all board LEDs and switches to conduct the RGB experiment. Debug mode will involve directing chosen signals into the LED row and disabling the RGB output. The minor change will ensure that testing in the future is much safer since debug wires will no longer have to be manually installed and disconnected from the top-level file whenever an issue arises.

## Conclusions

In summary, the main takeaways from this project were the value of clean separation between control and datapath for ease of design verification and debugging, FSM being responsible for sequencing, and datapath taking care of storage and scaling, as well as PWM generation. The use of pre-existing components from courses like FSMemb, Wr_reg_adxl362 and mypwm helped the group to concentrate on new code only, thus minimizing errors. As for Map To Max based duty-cycle control, this was a smart move due to its ability to keep the switching period of PWM output fixed while using sensor data and switch-selected floors on the input side of PWM block. Finally, the fact that the project had some mixed-voltage I/O on SW[8] made me realize the importance of double-checking any constraint prior to implementation. As for future work, it would be necessary to add either UART or display readout for sensor readings, and implement a mode-switch which would enable several color-mapping modes using the same circuitry.

From what we have seen during simulation and testing stages, it becomes clear that simulation and actual hardware testing answer different kinds of questions. On the one hand, simulation showed correctness of sequencing done by FSM and operation of our datapath under controlled conditions, i.e., certain sequences of input data values. On the other hand, testing allowed us to find out where the real board specifics, such as voltage requirements, affected our design and how we needed to communicate with external sensors. Therefore, our final solution can be regarded as completed framework that can still be improved with extra display and diagnostic features.

## References


[1] D. Llamocca, ECE-4710/5710: Computer Hardware Design — Course Materials and VHDL Component Library, Oakland University, Winter 2026.

[2] Digilent Inc., Nexys-4 DDR FPGA Board Reference Manual, Pullman, WA: Digilent, 2016.

[3] Analog Devices, ADXL362: Micropower, 3-Axis, ±2 g /±4 g /±8 g Digital Output MEMS Accelerometer — Data Sheet, Rev. E. Norwood, MA: Analog Devices, 2019.

[4] Xilinx Inc., 7 Series FPGAs Data Sheet: Overview (DS180), v2.6.1. San Jose, CA: Xilinx, 2020.

[5] IEEE, IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-2008. New York, NY: IEEE, 2009.