ECE-4710 / 5710 · COMPUTER HARDWARE DESIGN · WINTER 2026

# RGB LED Color Control

*Using PWM with an Accelerometer, Temperature Sensor, and Switch Inputs*


**Yousif Fatohi**

**Andrew Kashat**

**Sedim Baffrow**

**Peter Younan**


INSTRUCTOR

**Prof. Daniel Llamocca**

PRESENTATION

April 23, 2026

TARGET

Digilent Nexys-4 DDR · Artix-7 XC7A100T


Electrical and Computer Engineering Department · Oakland University

OVERVIEW

# What the Circuit Does

*Tilt the board, warm the chip — watch the LED change color.*

Three sensor outputs are read from the onboard ADXL362 via SPI, and are mapped to the RGB LED's three color channels. The ADXL362's X-axis output drives the red channel, the Y-axis drives the green channel, and the IC's temperature drives the blue channel. Nine slide switches allow the user to set a minimum brightness floor for each color channel: off, about 25%, about 50%, or about 75%.

**CONTROL**

3-state FSM (S1 → S2 → S3)

**DATAPATH**

3 × 16-bit regs + Map To Max blocks + 3 × PWM

**RED**

X-axis tilt

LD16_R

**GREEN**

Y-axis tilt

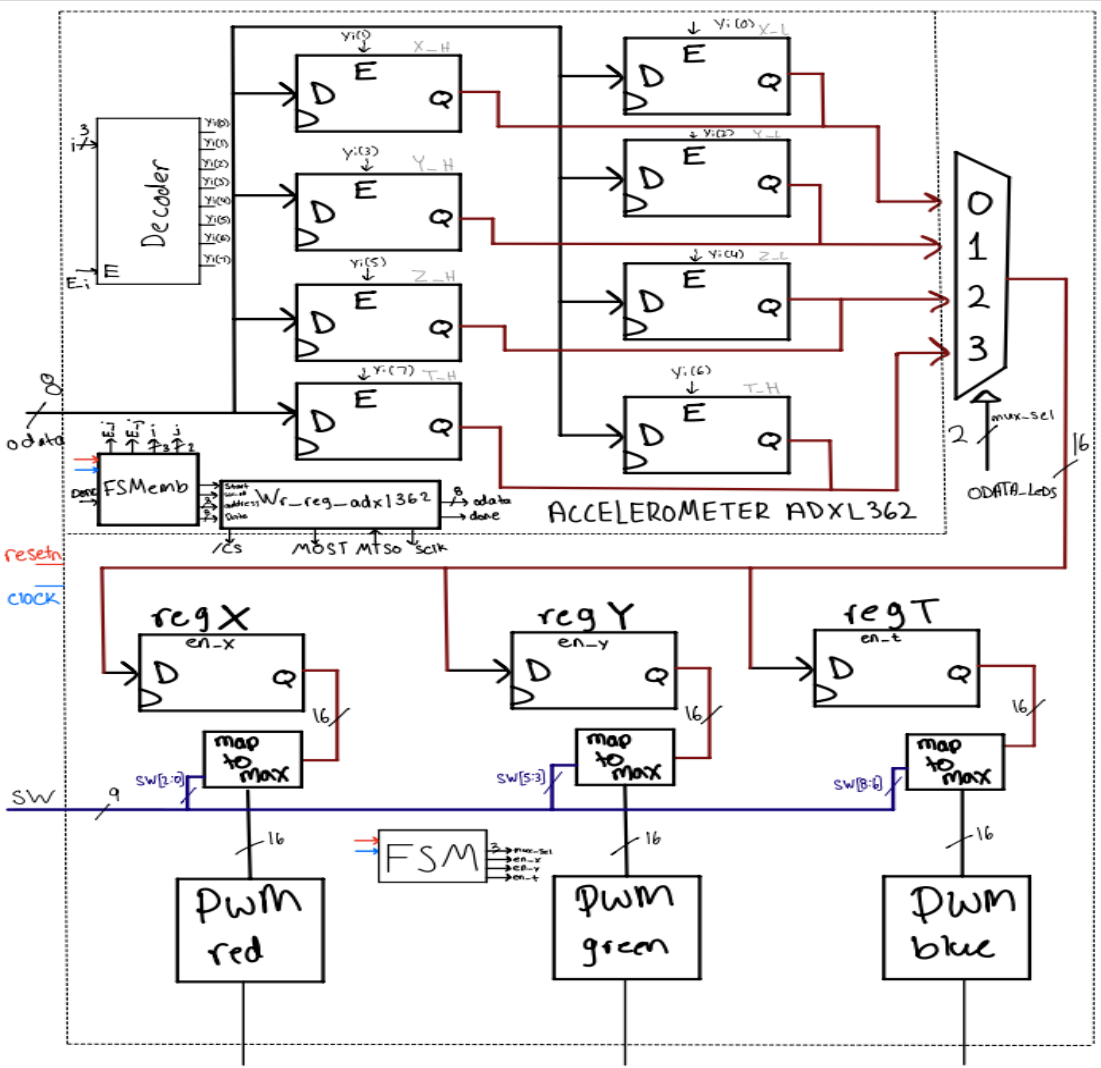
LD16_G

**BLUE**

Chip temperature

LD16_B

# System Block Diagram



HOW IT WORKS

# One Full Cycle Through the Pipeline

| STEP 1 | STEP 2 | STEP 3 | STEP 4 | STEP 5 |
|---|---|---|---|---|
| **ADXL362** | **FSMemb** | **FSM + Regs** | MAP TO MAX | **PWM → LED** |
| Senses tilt & temp | SPI → 16-bit words | Route & latch per axis | \|x\|, ×64 saturate, max() with floor | Duty drives brightness |

## Three latch states, three axes

S1 latches X (Red), S2 latches Y (Green), S3 latches Temp (Blue). On each valid read, the FSM cycles through X, Y, and Temp.

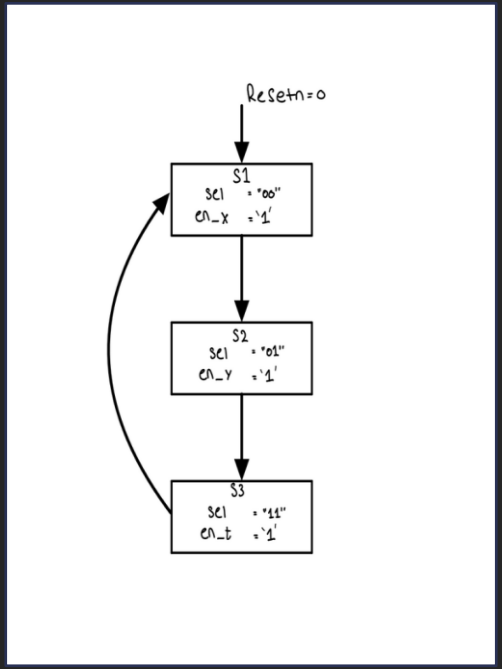
## Registers smooth the output

In between sample sets, each register retains
its value, meaning that there is no flickering due to waiting for new PWM values.

## Switches shape the response

For each channel, three switches decide priority encoding of a minimum brightness setting of either off, 25%, 50%, or 75%. However, the real-time sensor signal (with scaling of ×64 and saturation) can brighten using max().

# FSM — Our 3-State FSM




**FUNCTION**

Cycles through X, Y, and Temp one axis per clock, steering the accelerom mux and pulsing the right register enable so each sample lands in its own latch.

**State outputs**

| | | | |
|---|---|---|---|
| S1 | `sel = "00"` | `en_x = 1` | *latch X → Red* |
| S2 | `sel = "01"` | `en_y = 1` | *latch Y → Green* |
| S3 | `sel = "11"` | `en_t = 1` | *latch T → Blue* |

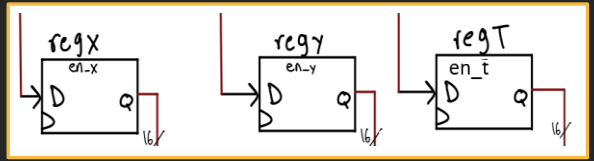
**What's happening:** This is the only FSM designed by our group. It is quite minimal in design: no calculation or any decision making; only sequencing is involved. On every valid pulse on FSMemb, our 3-state FSM goes through transitions of S1 to S2 to S3 and back to S1. In doing so, we enable sel [1:0] and the one-hot registers for each axis.

DATAPATH · STORAGE

# Holding Registers — regX / regY / regT

**FUNCTION**

Three 16-bit clock-enable registers capture each axis's latest sample and hold it steady, so the PWM downstream always has a valid duty.



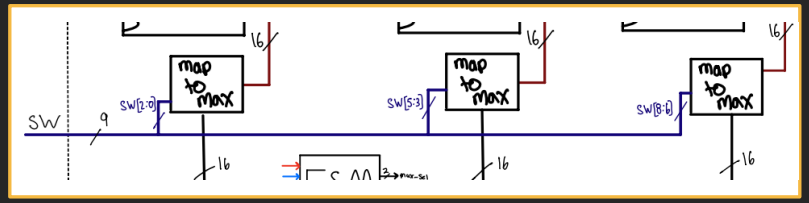

| regX | regY | regT |
|---|---|---|
| CLOCK ENABLE | CLOCK ENABLE | CLOCK ENABLE |
| en_x | en_y | en_t |
| *16-bit* | *16-bit* | *16-bit* |
| → Red PWM | → Green PWM | → Blue PWM |

**Reset behavior:** synchronous, active-low · on resetn = '0', all three registers clear to 0x0000 — LED starts dark.

**What's happening:** The three registers separate the slower sampling process from the faster clock rate for PWM. The PWM module requires a new duty cycle on every system clock pulse, while new samples occur only after several microseconds; the registers compensate for this time discrepancy by keeping the latest data sampled until a new sample occurs.

DATAPATH · USER CONTROL

# Map To Max — Switch Floors + Sensor, Combined




## Three combinational stages per channel

**1 Absolute value − BASELINE**

Pick |value| sufficiently negative for negative tilt to have the same brightness as positive tilt. Now subtract BASELINE = 150 (saturated to zero) to eliminate the DC offset from the temperature sensor value. Red and green operate with BASELINE = 0.

**2 Shift left 6 (×64) with saturation**

Left-shift by 6 (multiply by 64) to convert the value into an 8-bit format. If the operation would cause a 16-bit overflow (top bit set) the result will be saturated to 0xFFFF. It is precisely this which gives us 'full tilt = full brightness'.

**3 max(switch floor, scaled sensor)**

Switches pick a brightness FLOOR (see table). Whichever is larger — the floor or the live sensor — wins. If sw = 000, output is forced to 0 regardless.

## Switch allocation (9 switches · 3 per channel)

| | |
|---|---|
| SW[2:0] | → RED floor |
| SW[5:3] | → GREEN floor |
| SW[8:6] | → BLUE floor |

## Brightness floor per switch code

| SWITCH | DUTY (hex) | LOOKS LIKE |
|---|---|---|
| "000" | 0x0000 | OFF |
| "001" | 0x1000 | ~25% bright |
| "010"/"011" | 0x4000 | ~50% bright |
| "1xx" | 0x9000 | ~75% bright |

**What's happening:** Switches select a minimum brightness value. The live sensor is multiplied by 64 and saturated to 0xFFFF, and whichever is larger determines the final result. That means that the user can force any one channel to be lit with one of four brightnesses (off / 25% / 50% / 75%), but no more than the sensor will let him/her. sw=000 forces hard-off.

DATAPATH · OUTPUT

# PWM — 3-Channel Pulse-Width Modulator

## How one instance works

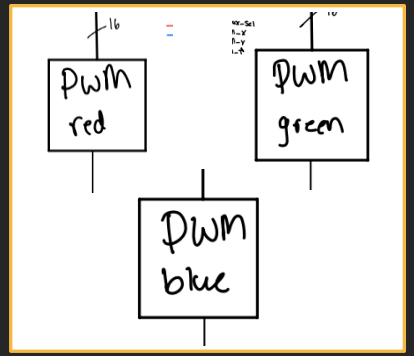
A 16-bit counter runs from 0 up to TPWM − 1, then rolls over.

A comparator checks `counter < duty`. When it is true, the output is '1' — otherwise '0'.

*duty = 0 → LED off · duty = TPWM → LED full on.*

## Project settings

| | |
|---|---|
| `TPWM` | 65535 (fixed) |
| `Counter width` | 16 bits |
| `Duty input` | from Map To Max |
| `PWM freq.` | ~1.53 kHz |
| `Why 1.5 kHz` | well above flicker |




## Three instances drive LD16_R / G / B

**PWM RED** — LD16_R

duty ← max(floorR, |x_reg|×64)

**PWM GREEN** — LD16_G

duty ← max(floorG, |y_reg|×64)

**PWM BLUE** — LD16_B

duty ← max(floorB, |(t_reg−150)|×64)

**What's happening:** In the PWM module section, the duty cycle number that is purely digital becomes light. The human eye will average out the on-off cycle at around 1.5 kHz; therefore, 25% duty will appear as quarter bright, while 75% duty will appear as three-quarters bright. There are three modules using the same clock and TPWM.

VERIFICATION

# Testbench & Hardware Setup

## Testbench — tb_top_rgb_ctrl

- **Clock / reset**

Clock at 100 MHz (10ns period). Keeps resetn low for 100 ns to initialize all the registers and set FSM in S1, then frees it.

- **Switch stimulus**

Runs through 10 SW states – 25%, 50%, 75% floors for each color, off, all channels 25%, all channels 75%, and some combinations. Each state is kept active for 800 µs, so there's always one complete 655 µs PWM cycle on display.

- **MISO pattern driver**

We send a 32-bit repeating pattern (0x1A2B3C4D) through MISO, one bit per clock pulse. So each byte read from SPI by FSMemb will be unique and non-zero. This helps us track signals along regX/Y/T throughout the whole process.

- **What we verify in the waveform**

No assertions here, just visual inspection. We check FSM state transitions (S1 → S2 → S3), one-hot enables, register latch, and that each LED duty is correct for their switch floor for all possible SW states.

## Hardware setup

| | |
|---|---|
| **Board** | Digilent Nexys-4 DDR |
| **FPGA** | Artix-7 XC7A100T |
| **Sensor** | On-board ADXL362 accelerometer + die temp |
| **Output** | RGB LED LD16 (R, G, B pins) |
| **Clock** | 100 MHz on-board oscillator |
| **Switches** | SW[8:0] — 9 slide switches |
| **Toolchain** | Vivado · Vivado Simulator |
| **Language** | VHDL · IEEE.NUMERIC_STD |

**What's happening:** Priot to loading anything on the board, we simulate our design using tb_top_rgb_ctrl. Testbench sets up resetn, SW switches, and generates a cyclic pattern on MISO (with SCLK_T = 16 so SPI operates faster in sim). Then we inspect all internal signals – FSM state, enables, registers, duties.

# Waveform — Three FSM Cycles in Sim


**State order holds**

state cycles S1 → S2 → S3 → S1 … one state per clock, forever.

**Enables fire one-hot**

en_x in S1, en_y in S2, en_t in S3. Never two high at once.

**Registers latch on time**

regx / regy / regT update only when their enable pulses.

**What's happening:** Three complete FSM cycles have been captured in this wave form in the Vivado simulation tool. First observe the state, then the three enables, followed by the three registers; notice that each axis gets loaded into its respective register only when its corresponding enable pulse occurs.

WRAP-UP

# Results & What We Learned

MAIN STRUGGLE

## Calibrating the blue baseline

At room temperature, the ADXL362's onboard temperature sensor output does not read zero; its DC output is a large positive number that changes only very little with an increase in temperature. Without correction, the LED would blink with full intensity at higher temperatures, indicating a change in temperature, which was not the goal here.

**OUR FIX:** Subtract a BASELINE of 150 from the Map To Max block before the shift operation.
Room temperature silicon is interpreted as zero intensity; the LED brightness increases as the silicon heats.

WHAT WE LEARNED

*About our own design decisions*

**Baseline subtraction beats scaling alone**
Offset shifting the DC signal only increases the offset. Removal of the offset is what makes the channel sensitive to levels and not offsets.

**Using a BASELINE generic kept one component**
Rather than develop another function to scale with temperature, Map To Max was parameterized; the same algorithm, three different versions, where blue was set to BASELINE = 150.

Fixed ×64 + max() is cheap and stable
The sensor gain was fixed to ×64, and the priority encoder switch floor were fixed, combined by max() function – no shifting required, no varying TPWM value. The PWM period remained constant for all switch positions, eliminating any LED flickering
during user adjustment.

**Saturation is on purpose**
Capping the value at 0xFFFF in case of an overflow allows the user to have predictable behavior at maximum brightness instead of wraparound behavior to a random low color.

VERIFICATION · HARDWARE

# Live Demo

### What you'll see in the video

1. Flat board, switches mid-range → steady baseline color.
2. Tilt along X → red intensity tracks the tilt angle.
3. Tilt along Y → green intensity tracks independently.
4. Warm the chip by touch → blue channel slowly climbs.
5. All switches up → saturated, maximum brightness.