THE UNIVERSITY OF ALABAMA IN HUNTSVILLE®

Advanced Logic Design Lab

Laboratory Assignment #X

PWM Brightness Control with a HUB75 RGB LED Matrix

(*X% of Final Grade*)

Background

The purpose of this laboratory is to introduce pulse-width modulation (PWM) as a practical hardware technique for controlling LED brightness while simultaneously teaching the timing and scanning fundamentals of HUB75 RGB matrix displays. You will incrementally build a working display pipeline that progresses from a single lit pixel to a full screen animation loaded from a HEX file.

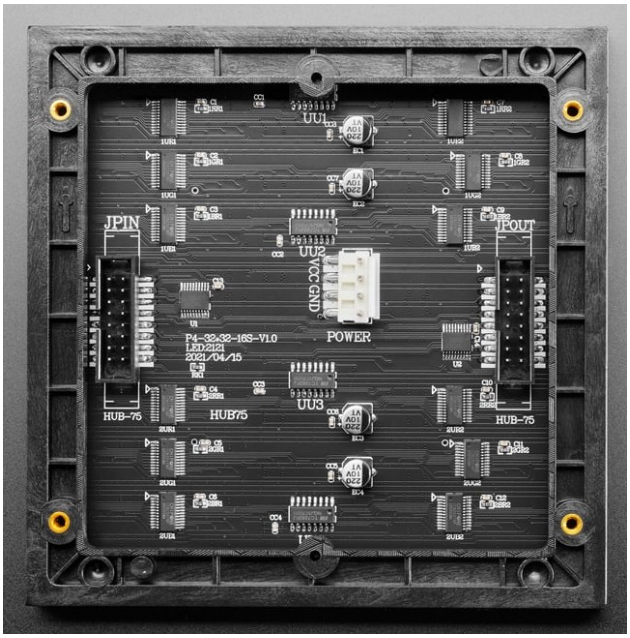

Figure 1: Rear view of a 32×32 HUB75 RGB LED matrix panel.

Panel Scanning

A 32×32 RGB LED matrix contains a total of 1024 pixels, where each pixel is composed of three individual light-emitting diodes (LEDs) corresponding to the red, green, and blue color channels. By varying the intensity of these three LEDs, a wide range of colors can be produced at each pixel location. As illustrated by the dense driver circuitry on the rear of the panel in Figure 1, directly controlling every LED independently would require thousands of dedicated FPGA input/output (I/O) pins and extensive control logic. Such an approach would be impractical in terms of hardware complexity, cost, and signal routing.

To address this limitation, the HUB75 LED panels employ a multiplexed scanning architecture. Rather than driving all rows of the display at the same time, the panel divides the matrix into smaller row groups and activates only one group at a time. Pixel data for the currently active rows is shifted into the panel using high-speed serial data signals, while separate address and control lines select which rows are illuminated. This approach significantly reduces the number of required control signals while still allowing full-color operation across the entire display.

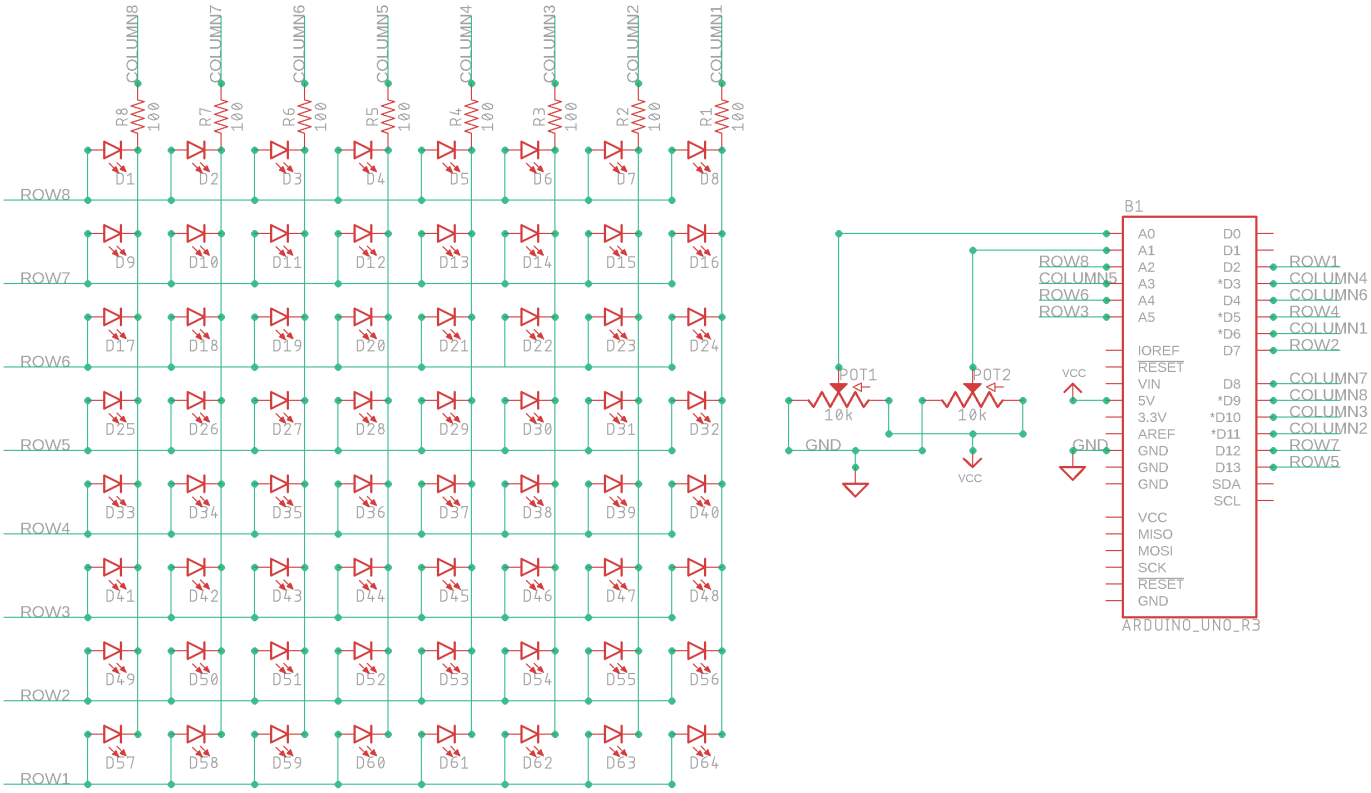

Figure 2: LED matrix row–column diagram illustrating multiplexed row scanning.

Because only a portion of the display is illuminated at any given moment, as shown conceptually in Figure 2, the panel must cycle through all row groups at a high refresh rate. When this scanning process occurs sufficiently fast, the human eye interprets the rapidly changing illuminated rows into a single, stable image. This phenomenon, is known as persistence of vision, which allows the display to appear continuously lit even though each row is only active for a fraction of the total frame time.

For a typical 32×32 HUB75 LED matrix, the panel operates using a 1/16 scan configuration. In this mode, the display is refreshed in 16 discrete row steps per frame. During each step, two rows are driven simultaneously: one row from the top half of the panel and the corresponding row from the bottom half. By rapidly cycling through all 16 row pairs, the controller updates the entire display while maintaining acceptable brightness and visual stability.

Data Signals: Shifting Pixel Columns

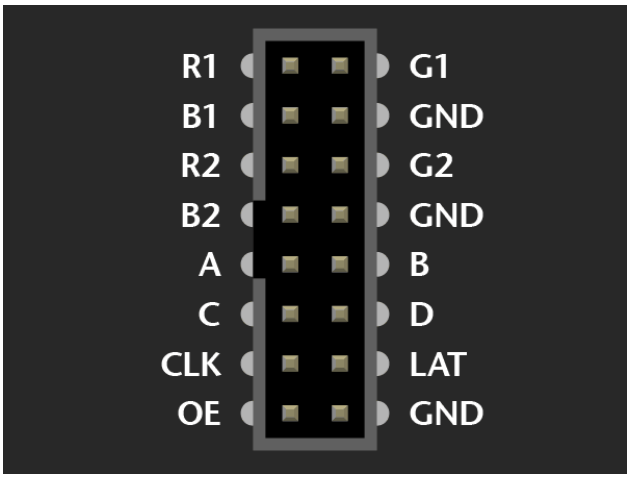

Figure 3: HUB75 connector pinout.

Each row step requires loading color data for 32 pixels into the panel's internal shift registers. The RGB data signals used to provide this pixel information are delivered to the panel through the HUB75 connector, shown in Figure 3. This data corresponds to one complete column sweep across the active row pair and defines the color state of every pixel in that row for the current refresh cycle. Because the HUB75 panel uses a multiplexed scanning approach, the pixel data stored in the shift registers must be refreshed every time the active row address changes. As a result, the controller continuously reloads pixel data for each row step to ensure the displayed image remains correct and visually stable.

Data/Clock Timing Requirements

The HUB75 interface behaves like a synchronous shift register: RGB data must be stable during the active edge of `CLK`. In practice, the controller drives `R1/G1/B1` and `R2/G2/B2` first, then pulses `CLK` to commit that column into the panel. This is why the HDL separates "data setup" and "clock pulse" into two phases (e.g., `pixel_phase`): one phase computes/updates the next pixel values and holds the RGB lines steady, and the next phase asserts `CLK` to shift them into the panel. This structure also aligns naturally with synchronous ROM reads in Phase III, where pixel data becomes valid one clock after an address is applied.

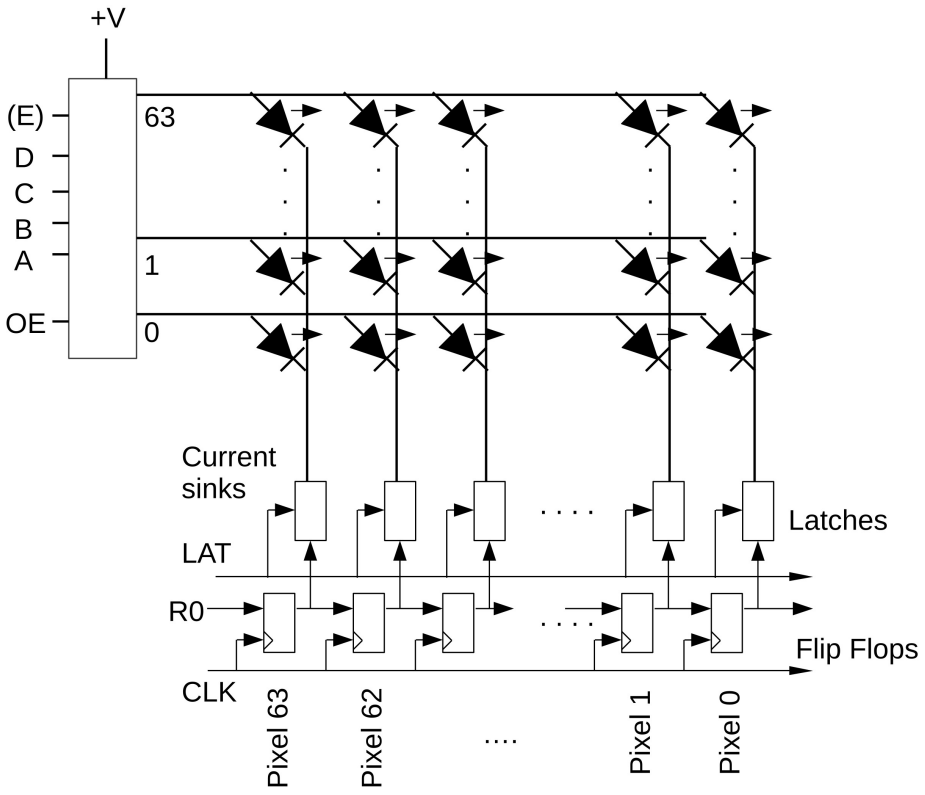

Figure 4: View of the column shifting process.

The HUB75 interface provides separate red, green, and blue data signals for two rows simultaneously, allowing the panel to drive both halves of the display in parallel. The signals `R1`, `G1`, and `B1` carry pixel data for the currently selected row in the top half of the panel, while `R2`, `G2`, and `B2` carry pixel data for the corresponding row in the bottom half. This dual-row data path reduces the number of required row selection steps and improves refresh efficiency, while maintaining full color control for each pixel.

For each column index from 0 to 31, the controller presents the appropriate RGB values for the next pixel on the `R1/G1/B1` and `R2/G2/B2` signals. Once these data lines are stable, the panel clock signal (`CLK`) is pulsed to shift the current pixel data into the panel's internal shift registers, as illustrated conceptually in Figure 4. Each clock pulse advances the shift registers by one position, effectively moving previously loaded pixel data toward the output drivers. This sequence is repeated 32 times per row step, ensuring that all pixel data for the active row pair is fully loaded before the display output is enabled.

In the HDL implementation, this behavior is controlled during a dedicated `SHIFT` phase within the display controller state machine. During this phase, the pixel data outputs are driven based on framebuffer or memory contents, and the `clk_out` signal is asserted in a controlled manner to generate the required shift pulses. By separating the data-shifting operation into a distinct phase, the design ensures proper timing, prevents visual artifacts, and allows for clean synchronization with subsequent latching and output-enable operations.

Row Addressing: `A`, `B`, `C`, and `D` Signals

The HUB75 LED matrix panel includes four row address input signals, `A`, `B`, `C`, and `D`, which are used to select which pair of rows is currently enabled for display. These signals control the internal row-selection circuitry of the panel and determine which LEDs are allowed to emit light during a given scanning interval. At any moment, only the rows specified by the active address are electrically enabled, while all other rows remain disabled.

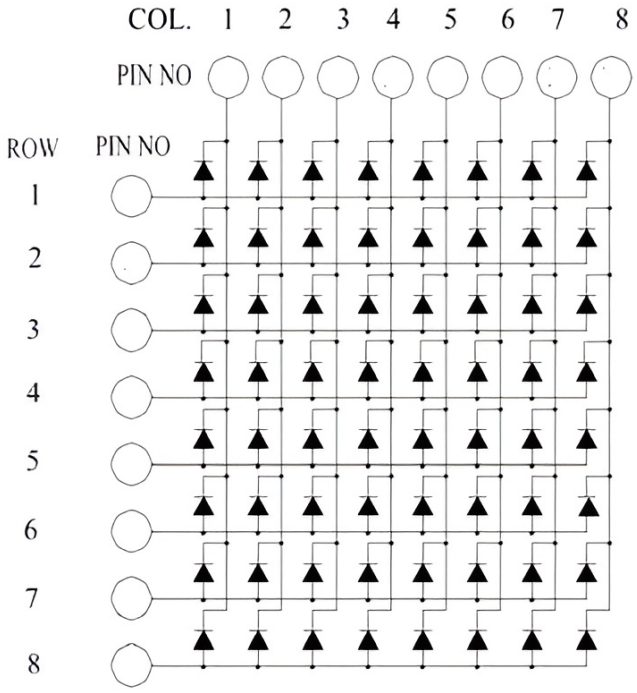

Figure 5: Simplified internal row–column structure of an LED matrix.

Figure 5 illustrates the internal organization of an LED matrix as a grid of rows and columns. In this structure, row lines are used to enable groups of LEDs, while column lines determine which LEDs within the active rows are illuminated. Row addressing signals select which row lines are active, ensuring that only the intended LEDs can conduct current during each scan period.

Because the panel operates using a 1/16 scan configuration, the display refresh process is divided into 16 distinct row steps per frame. During each step, the row address value ranges from 0 to 15 and is represented as a 4-bit binary number across the `A`, `B`, `C`, and `D` inputs. In this encoding, `A` represents the least significant bit, while `D` represents the most significant bit. Together, these signals uniquely identify which row pair is active during the current scan cycle.

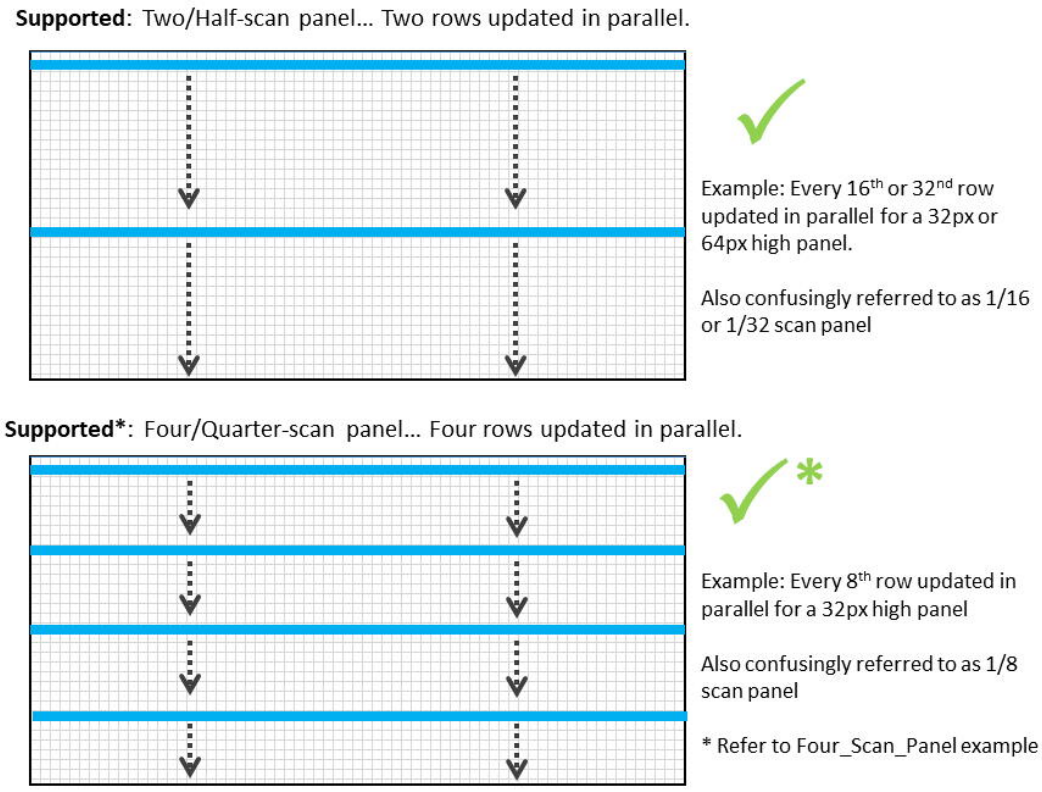

Figure 6: Illustration of multiplexed scanning modes.

As shown in Figure 6, multiplexed LED panels update only a subset of rows at a time, with different scan configurations determining how many rows are active simultaneously. For a 1/16 scan HUB75 panel, two rows are driven in parallel during each scan step: one from the top half of the panel and the corresponding row from the bottom half. This approach reduces the number of required address lines while maintaining full coverage of all display rows.

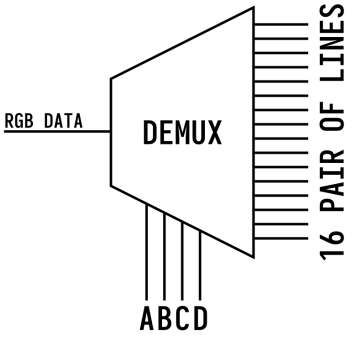

Figure 7: Demultiplexer-based row selection using the `A`, `B`, `C`, and `D` address signals.

Figure 7 conceptually illustrates how the 4-bit row address formed by the `A`, `B`, `C`, and `D` signals is decoded internally to select a specific pair of rows. When the row address is set to 0, row 0 from the top half of the matrix and row 16 from the bottom half are enabled. When the address is set to 1, rows 1 and 17 are selected. This pattern continues sequentially until the maximum row address value of 15 is reached, at which point rows 15 and 31 are displayed.

In the top-level HDL module, the 4-bit row address signal (`row_addr`) is driven by the display controller state machine and updated once per row scan period. This signal is connected directly to the panel row address inputs using the mapping {`PANEL_D`, `PANEL_C`, `PANEL_B`, `PANEL_A`}, ensuring that the binary row address is correctly interpreted by the panel. The row address is held constant while pixel data is shifted and displayed, then updated to the next value to advance the scanning process. This coordinated control of row addressing is essential for achieving uniform brightness, correct image placement, and stable visual output.

Latch and Output Enable: Making Pixels Appear

After shifting 32 columns of pixel data into the panel's internal shift registers, the data must be committed to the LED outputs before it can be displayed. This process ensures that a complete row of pixel data is shown simultaneously, rather than appearing incrementally as the data is shifted.

Two key control signals are responsible for this operation: `LAT` (latch) and `OE` (output enable). The `LAT` signal copies the contents of the shift registers into the panel's output registers, effectively freezing the pixel data for the current row pair. The `OE` signal controls whether the LEDs are allowed to emit light, enabling or disabling the visible output of the panel.

Figure 4 illustrates this latch-and-display behavior at a conceptual level, showing how shifted pixel data is committed and then displayed during a controlled output interval. To further clarify the timing relationships between control and data signals, Figure 8 provides a detailed timing diagram of a typical HUB75 row scan cycle.

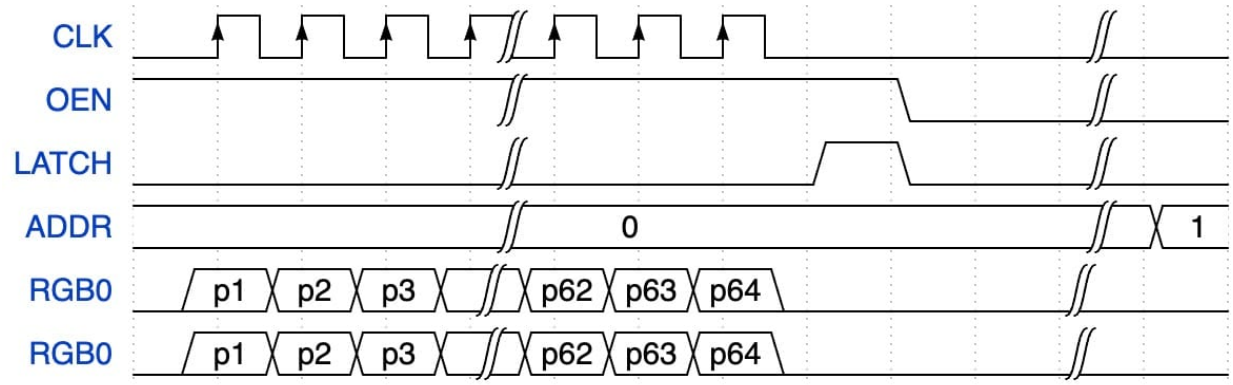

Figure 8: Timing diagram of the HUB75 row scan operation and its relationship between `CLK`, `OE`, `LAT`, row address signals, and RGB data.

As shown in Figure 8, pixel data is first shifted into the panel on successive rising edges of the `CLK` signal while `OE` is held low, ensuring that partially shifted data is not displayed. During this phase, the RGB data signals change on each clock cycle to load one column of pixel data at a time into the internal shift registers.

Once all 32 columns of pixel data have been shifted in, the `LAT` signal is pulsed. This pulse transfers the completed row of pixel data from the shift registers into the panel's output registers. At the same time, the row address signals `A`, `B`, `C`, and `D` are set to select the appropriate row pair for display.

After the data has been latched and the row address is stable, the `OE` signal is asserted for a short display interval. During this interval, the selected row pair becomes visible on the panel. The duration of this interval determines the effective brightness of the row and is controlled by the display controller. When the display interval ends, `OE` is deasserted, the row address is advanced, and the process repeats for the next row pair.

This latch-and-display sequence is repeated for each row pair during a full frame refresh. In the HDL implementation, this behavior is reflected using Finite State Machines (FSM), with distinct states such as `SHIFT`, `LATCH`, and `SHOW`. Separating these operations into dedicated states ensures correct timing, which prevents visual artifacts such as ghosting or tearing, to produces a stable image.

Brightness Control and Pulse-Width Modulation (PWM)

The perceived brightness of an LED matrix is controlled by the amount of time each row remains illuminated during a scan cycle. On the HUB75 panels, this is achieved by adjusting the duration for which the `OE` (output enable) signal is asserted. A longer `OE` on-time allows the LEDs to emit light for a greater portion of the scan period, resulting in a brighter appearance. Conversely, reducing the `OE` on-time decreases the amount of light emitted and produces a dimmer output.

Two common approaches are used to control brightness in LED matrix displays. The simplest method is global brightness control, in which the display controller adjusts the display interval equally for all pixels within a row. In this approach, a single timing parameter (such as `show_cnt`) determines how long `OE` remains asserted for every row, uniformly scaling the brightness of the entire display.

A more advanced method is per-pixel brightness control using pulse-width modulation (PWM). In this approach, each pixel's color intensity is represented using multiple brightness bits, often referred to as bit-planes. Each bit-plane corresponds to a different weight and is displayed for a proportionally scaled duration. By rapidly cycling through all bit-planes within a single frame, the display controller synthesizes a wide range of brightness levels for each individual pixel.

In this laboratory implementation, global brightness control is used as an initial approach due to its simplicity and low hardware overhead. This method provides an effective introduction to brightness control through timing adjustment. The same principles naturally extend to full PWM-based animation, where multiple bit-planes and weighted display intervals are used to achieve fine-grained per-pixel brightness control to produce an image.

Framebuffer Mapping: Relating `(x, y)` Pixel Coordinates to the Scan Process

Although the HUB75 display is refreshed using a row-based scanning process, it is often conceptually simpler to view the image as a two-dimensional framebuffer indexed by pixel coordinates. In this representation, each pixel is addressed using an `(x, y)` coordinate, where the horizontal coordinate `x` corresponds to the column index and the vertical coordinate `y` corresponds to the row index.

For a 32×32 LED matrix, the `x` coordinate ranges from 0 to 31, representing the 32 columns of the display, while the `y` coordinate ranges from 0 to 31, representing the 32 rows. The framebuffer stores the color information associated with each `(x, y)` pixel location, independent of the order in which pixels are physically refreshed on the panel.

During the scanning process, the display controller maps these framebuffer coordinates to the hardware scan indices. The row scan index (`row_idx`) selects which pair of rows is currently being displayed. The top half of the panel corresponds to `y = row_idx`, while the bottom half corresponds to `y = row_idx + 16`. The column index (`col_idx`) determines which column is currently being shifted into the panel and corresponds directly to the `x` coordinate of the framebuffer.

As a result, each pixel shifted into the panel during the scan process is obtained by reading the color data from a specific `(x, y)` location in the framebuffer. For each `col_idx`, the controller outputs the RGB bits for the pixel at `(x = col_idx, y = row_idx)` on the top-half data lines and the pixel at `(x = col_idx, y = row_idx + 16)` on the bottom-half data lines.

This coordinate-to-scan mapping is explicitly implemented in the GIF display logic, as illustrated in Listing 1.

Listing 1: Mapping framebuffer `(x,y)` pixels to HUB75 scan outputs

```
/* Scan indices:
   - x is the current column being shifted into the panel (0..31)
   - row_idx selects the active TOP row; bottom row is offset by +16 */
int x = col_idx; // current column
int y_top = row_idx; // active row in the top half
int y_bot = row_idx + 16; // matching row in the bottom half

/* Read framebuffer colors for the active row pair */
Pixel top = framebuffer[y_top][x]; // pixel at (x, y_top)
Pixel bot = framebuffer[y_bot][x]; // pixel at (x, y_bot)

/* Drive HUB75 RGB data lines for this column */
R1 = top.r; G1 = top.g; B1 = top.b; // top-half pixel
R2 = bot.r; G2 = bot.g; B2 = bot.b; // bottom-half pixel

/* Shift this column into the panel */
pulse(CLK); // advances internal shift registers by one column
```

Animated GIF Frames and Memory Mapping

For animated images, the display controller treats a GIF as a sequence of preprocessed image frames stored in FPGA memory. Each frame represents a complete 32×32 pixel image, and animation is achieved by displaying these frames sequentially over time.

In this design, the animation data is stored in a synchronous read-only memory (ROM), referred to as `gif_rom`. Multiple 32×32 frames are arranged contiguously in memory, with each frame occupying 1024 memory locations corresponding to one byte per pixel. The memory address for a given pixel within a specific frame is computed using the following mapping:

$$\texttt{addr} = \texttt{frame\_idx} \times 1024 + y \times 32 + x$$

This addressing scheme allows the controller to select both the current animation frame and the specific pixel location within that frame during the scan process.

Each stored byte contains packed color information for a single pixel. The red, green, and blue color components are encoded as individual bits within the byte, with bit 2 representing the red channel, bit 1 representing the green channel, and bit 0 representing the blue channel. This compact encoding minimizes memory usage while still supporting full RGB output on the LED matrix.

Because `gif_rom` is implemented as a synchronous memory, pixel data appears at the ROM output one clock cycle after the address is applied. To accommodate this latency, the scan finite state machine (FSM) pipelines the address calculation and pixel output stages. This ensures that the correct RGB data is available and stable when it is presented to the HUB75 data lines during each column shift.

The animation playback rate is controlled independently of the display refresh timing. Rather than advancing to a new frame on every refresh, the controller holds each frame on the display for multiple complete refresh cycles using a frame-hold counter. Once the programmed hold duration expires, the controller increments `frame_idx` to advance to the next frame in memory. This approach allows smooth, timing-controlled animation while maintaining a high and consistent display refresh rate.

Figure 9 illustrates the relationship between the display refresh process and the animation frame advancement.

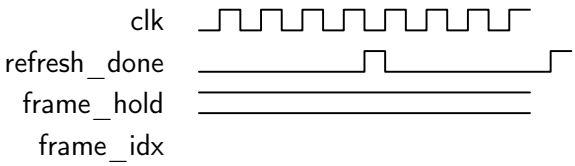

Figure 9: Timing relationship between display refresh cycles and GIF frame advancement.

GIF Playback Theory: ROM-Based Image Streaming

For animated images, the lab can treat a GIF as a sequence of preprocessed frames stored in FPGA memory. Instead of decoding the GIF at runtime, each frame is converted ahead of time into a fixed 32×32 pixel image, and the controller simply streams pixel bytes out of memory while performing the normal HUB75 scan. This approach keeps the HDL simple and makes the animation behavior fully deterministic.

ROM Layout and Address Mapping

The design stores the animation in a synchronous ROM module named `gif_rom`. Frames are placed back-to-back in memory, and each frame uses one byte per pixel (1024 bytes total for a 32×32 image). The address for a pixel at coordinate (x, y) in frame `frame_idx` is:

$$\texttt{addr} = \texttt{frame\_idx} \times 1024 + y \times 32 + x$$

In the HDL, this is implemented by forming a frame base address of `frame_idx * FRAME_PIX` (equivalently `frame_idx` $\ll$ `10` when `FRAME_PIX = 1024`), then adding the row and column offsets.

Row-Pair Mapping (HUB75 1/16 Scan)

The HUB75 panel is refreshed as row pairs. The scan index `row_idx` selects which row pair is active (0–15). For each shifted column `col_idx`, the controller outputs the top-half pixel at ($x =$ `col_idx`, $y =$ `row_idx`) on `R1/G1/B1`, and the bottom-half pixel at ($x =$ `col_idx`, $y =$ `row_idx` $+ 16$) on `R2/G2/B2`.

Packed RGB Format (3-Bit Color)

Each ROM byte contains a compact 3-bit RGB pixel:

- bit 2: Red (`R`)
- bit 1: Green (`G`)
- bit 0: Blue (`B`)

All other bits in the byte are unused for display. This matches the ROM format described in `gif_rom.v` and consumed by `hub75_gif.v`.

Synchronous ROM Timing and Scan Pipelining

Because `gif_rom` is synchronous, the output data `dout` becomes valid one clock cycle after the address is applied. To handle this, the scan FSM pipelines the operation:

- On the "setup" phase of a pixel, the controller computes and registers `addr_top` and `addr_bot` for the *next* pixel.
- In the same cycle, it drives `R1/G1/B1` and `R2/G2/B2` using the ROM outputs from the *previous* cycle.
- On the next phase, it pulses `CLK` high to shift those already-driven RGB bits into the panel shift registers.

This is exactly what the `pixel_phase` logic in `hub75_gif` is doing: address setup and RGB driving that happens in one phase, and the clock pulse that commits that pixel into the panel which occurs in the other phase.

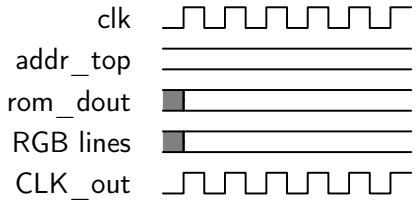

Figure 10: Per-Pixel Pipeline: Relationship between ROM output and `rom_dout`, `CLK_out` in order to shift the driven bits into the panel.

Two-Row Streaming (Top and Bottom Halves)

Since a 1/16-scan HUB75 panel displays two rows at once, the controller streams two pixels per column: one for the top half ($y_{\text{top}} = \texttt{row\_idx}$) and one for the bottom half ($y_{\text{bot}} = \texttt{row\_idx} + 16$). The design instantiates two synchronous ROM reads (one for each half), producing `rom_top_dout` and `rom_bot_dout`. These directly drive:

$$\texttt{R1/G1/B1} \leftarrow \texttt{rom\_top\_dout[2:0]}, \qquad \texttt{R2/G2/B2} \leftarrow \texttt{rom\_bot\_dout[2:0]}$$

This allows the scan logic to stream both halves in parallel during each column shift.

Frame Rate Control

The animation rate is controlled independently from the panel refresh. Rather than switching frames every refresh, the controller holds each frame for multiple full refresh cycles using `frame_hold_cnt`. Only after the scan completes the last row pair of a refresh cycle does the design update the hold counter; when `frame_hold_cnt` reaches `FRAME_HOLD-1`, the controller increments `frame_idx` to advance to the next frame (wrapping back to 0 at the end). This prevents changing frames mid-refresh and produces smooth, stable animation without visible tearing.

Integration note. At the top level, the lab uses `SW[1:0]` to select which phase drives the HUB75 outputs: `00` selects Phase I (moving pixel), `01` selects Phase II (gradient/PWM), and `10` (and `11` by default) selects Phase III (ROM-based animation). The top module multiplexes the HUB75 signals (RGB, `CLK`, `LAT`, `OE`, and row address `A-D`) between the selected phase modules.

Laboratory Assignment

This laboratory assignment is composed of three phases, with each phase culminating with the current state of the design being validated by prototyping it on the Terasic DE2-115 platform. Students must successfully complete and demonstrate the valid functionality of a given phase to the laboratory instructor before proceeding to the next phase. Demonstration will include downloading the design to the FPGA, verifying correct HUB75 timing behavior, and confirming that the specified objectives for each phase are met.

A Quartus project template is provided on Canvas. This template includes the pin assignments, a working top-level module (`de2_115_hub75_lab_top`), and stub module files for each phase. You are responsible for completing the Phase I module as described below.

Phase I: Pixel Motion Bring-Up (RGB Sequencing)

In this phase, you will develop the fundamental HUB75 scan pipeline and verify correct row/column mapping by displaying a single pixel that travels across the panel by completing `hub75_phase1_pixel.v` module. The pixel shall traverse the display from left-to-right across a row, then advance to the next row, and its color shall cycle in the sequence **Red** → **Green** → **Blue** as it moves.

A successful Phase I implementation will demonstrate that your controller correctly performs the HUB75 refresh sequence (`SHIFT` → `LATCH` → `SHOW`), that row addressing (`A-D`) selects the intended row pair, and that the pixel appears stable. Occasional minor timing artifacts may be acceptable at this stage, but the pixel motion and RGB color order must be clearly observable.

A successfully designed module will demonstrate all of the following:

- **HUB75 Scan Timing:**
  - Shifts exactly 32 columns per active row address (0–15)
  - Pulses `LAT` to latch a fully shifted row
  - Uses `OE` to blank the panel during shifting and to enable output during the show interval
- **Pixel Behavior:**
  - Displays exactly one pixel at a time on the 32×32 display
  - Advances the pixel position `(x,y)` at a visible speed and wraps cleanly at the screen edges
- **RGB Color Cycling:**
  - Cycles the pixel color between **Red** → **Green** → **Blue** as it travels across the panel

Checkoff Requirement: Download and run the design on the DE2-115 and demonstrate the moving pixel pattern and RGB sequencing to the lab instructor. Students must obtain the laboratory instructor's approval before proceeding to Phase II.

Phase II: Color Gradient + Brightness Control (PWM)

In this phase, you will extend your HUB75 scan pipeline to generate a full-screen RGB gradient and introduce PWM-based brightness control. Rather than displaying a single pixel, your design must compute color values as a function of pixel position (x, y) and continuously refresh the panel using the same **SHIFT** $\rightarrow$ **LATCH** $\rightarrow$ **SHOW** sequence verified in Phase I.

A successful Phase II implementation will demonstrate that your controller can produce a stable, full-panel image while modulating brightness by controlling OE during the SHOW interval. The gradient must be clearly visible across the entire 32×32 display, with 3 distinct brightness settings that must be observable on the HUB75 display.

A successfully designed module will demonstrate all of the following:

- **Full-Screen Gradient Generation:**
  - Computes RGB values based on pixel coordinates (x, y)
  - Produces a continuous gradient across the entire 32×32 display
  - Correct mapping for both the top (rows 0–15) and bottom (rows 16–31) halves of the panel
- **HUB75 Scan Timing:**
  - Shifts exactly 32 columns per active row address (0–15)
  - Pulses **LAT** to latch a fully shifted row
  - Uses **OE** to blank the panel during shifting and to enable output during the show interval
- **Global Brightness Control (PWM using OE):**
  - Implements brightness control by gating OE during SHOW
  - Brightness is adjustable via SW[4:2] and results in multiple discrete brightness levels

Checkoff Requirement: Download and run the design on the DE2-115 and demonstrate a full-screen RGB gradient along with at least three distinct brightness levels controlled by your brightness input (e.g., SW[4:2]). Students must obtain the laboratory instructor's approval before proceeding to Phase III.

Phase III: Animation Playback from a HEX File

In this phase, you will add memory-backed image playback by loading animation frame data from a **HEX file** into FPGA memory (ROM/BRAM) using `$readmemh`. The animation shall consist of multiple frames, where each frame represents a complete 32×32 image. Frames must be stored contiguously in memory, and your controller shall display frames sequentially to create a looping animation.

Because on-chip ROM is typically synchronous, you must account for the one-clock read latency when mapping `(x,y)` coordinates to a memory address. The animation playback rate shall be controlled independently from the scan refresh timing using a frame-hold counter (i.e., each frame is displayed for multiple full refresh cycles before advancing to the next frame).

A successfully designed module will demonstrate all of the following:

- **HEX-Backed Frame Storage:**
  - Loads frame data from a HEX file into on-chip memory using `$readmemh`
  - Stores frames contiguously with a fixed mapping (1024 pixels per 32×32 frame)
  - Uses a consistent pixel byte format (e.g., `bit2=R, bit1=G, bit0=B`)
- **Synchronous ROM Addressing (1-cycle latency):**
  - Accounts for the one-clock read delay when converting `(x,y)` into a ROM address
  - Pipelines ROM addressing during SHIFT such that the displayed pixel corresponds to the previously-read ROM output
- **Animation Playback Control:**
  - Displays frames sequentially and wraps back to frame 0 for continuous looping playback
  - Uses a frame-hold counter so the animation rate is independent of the HUB75 scan refresh
- **HUB75 Scan Timing:**
  - Shifts exactly 32 columns per active row address (0–15)
  - Pulses **LAT** to latch a fully shifted row
  - Uses **OE** to blank the panel during shifting/latching and enable output during SHOW

Checkoff Requirement: Demonstrate the HEX-loaded animation running continuously on the DE2-115 at a stable refresh rate with correct color output and no visible tearing.

Post-Lab Questions

1. **HUB75 Scan Sequencing:** Describe the purpose of each stage in the `SHIFT` → `LATCH` → `SHOW` pipeline. For each stage, state what `CLK`, `LAT`, and `OE` should be doing and why.

2. **Row Address Mapping (1/16 Scan):** For a 32×32 HUB75 panel with 1/16 scan, explain how `row_idx` selects two rows at once. Explicitly relate `row_idx` to y_{top} and y_{bot}.

3. **"Primed" Read at Row Boundaries:** In the LATCH stage, the design preloads the first pixel address of the next row pair. Why is this priming step necessary for a synchronous ROM? What would you expect to see on the display if it were removed?

4. **PWM Brightness Using `OE`:** Describe how global brightness is implemented by gating `OE` during the SHOW interval. If `SHOW_TICKS` is fixed, how does changing the `OE` on-time change perceived brightness?

5. **Refresh Rate vs. PWM Resolution Tradeoff:** Discuss the tradeoff between scan refresh rate, PWM resolution (number of brightness steps), and visible flicker. If you increase PWM resolution without increasing clock speed, what typically happens to flicker and why?

References

[1] Adafruit Industries, "32x16 and 32x32 RGB LED Matrix Guide," Adafruit Learning System. [Online]. Available: `https://learn.adafruit.com/32x16-32x32-rgb-led-matrix`.

[2] SparkFun Electronics, "RGB Panel Hookup Guide," SparkFun Learn. [Online]. Available: `https://learn.sparkfun.com/tutorials/rgb-panel-hookup-guide`.

[3] Chipone Technology, "ICN2038S: 16-Channel Constant Current LED Sink Driver with Dual Latch," Datasheet (PDF), v1.1, Mar. 2017. [Online]. Available: `https://olympianled.com/wp-content/uploads/2019/06/icn2038s.pdf`.

[4] Intel Corporation, "Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide," Intel FPGA Documentation. [Online]. Available: `https://docs.altera.com/r/docs/683240/current`.

[5] Heathen_Hacks-v2, "Row Column Scanning DIY 8x8 LED Matrix," Arduino Project Hub. [Online]. Available: `https://projecthub.arduino.cc/Heathen_Hacks-v2/row-column-scanning-diy-8x8-led-matrix-9233ce`.

[6] Lushay Labs, "LED Panel HUB75," Lushay Labs Documentation. [Online]. Available: `https://learn.lushaylabs.com/led-panel-hub75/`.

[7] mrcodetastic, "ESP32 HUB75 MatrixPanel DMA," GitHub repository. [Online]. Available: `https://github.com/mrcodetastic/ESP32-HUB75-MatrixPanel-DMA`.

[8] wcrsyyc, "About LED Matrix," DIY Blinky Bot. [Online]. Available: `https://wcrsyyc.github.io/diy-blinky-bot/about-matrix.html`.