

FPGA Hero: Final Report

Fernando Oliver Mediavilla

Dept. of Electrical Engineering & Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
fjoliver@mit.edu

Nicolas Valayannopoulos-Akrivou

Dept. of Electrical Engineering & Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
nva@mit.edu

Abstract—*FPGA Hero* is a single-player rhythm game implemented entirely on a Spartan-7 FPGA. A custom SPI SD-card controller streams audio and chart data into dual-clock FIFOs; an audio pipeline plays back 8-bit PCM via noise-shaped PDM; and a lightweight video engine composites sprites into a palettized framebuffer that is upscaled to 720p60 HDMI. A shared 1 μ s timebase keeps audio, visuals, and input judgment tightly synchronized. Our project achieves stable audio, a 720p display and an end-to-end input→on-screen feedback latency under 10 ms while staying within modest BRAM/DSP budgets.

Index Terms—FPGA, SD card SPI, HDMI/TMDS, PDM, clock-domain crossing.

I. PROBLEM SUMMARY

Music games demand (i) continuous audio, (ii) smooth and correctly timed visuals, and (iii) precise input judgment all at once. A naive, free-running design develops A/V drift and poor hit detection. Our challenge was to realize the entire stack on a Spartan-7 without a CPU or DRAM, keeping subsystems synchronized across multiple clock domains.

Requirements. Table I lists our project checklist and outcomes. Our commitments were met while some stretch goals were realized.

TABLE I
REQUIREMENTS VS. OUTCOMES

Requirement	Status
SD→FIFO streaming	Achieved
Stable audio (no underruns)	Achieved
Correct, on-time note rendering	Achieved
Debounced guitar buttons	Achieved
Two-song playable demo	Achieved
Measured A/V lock ≤ 10 ms	Achieved
Perspective lane effect	Attempted
Guitar + drum hardware	Guitar yes; Drums no
PDM audio output	Achieved

The key to our design was locking all time to a single 32-bit `time_us` counter. Audio sample index, chart timestamps, judge windows, and animations were expressed in microseconds relative to this counter, eliminating long-term phase error by construction.

II. SYSTEM OVERVIEW

The top level (Fig. 1) is partitioned into three domains connected by explicit CDC boundaries:

- **Control/Game** at 100 MHz: timebase, chart ingest, instance list builder, judge/score/HUD, key-scan debounce.
- **SD/SPI** at 25 MHz: sector reads into dual-clock FIFOs for audio and chart streams.
- **Pixel/HDMI**: timing generator, compositor, framebuffer, scaler, palette lookup, TMDS encoder/serializer.

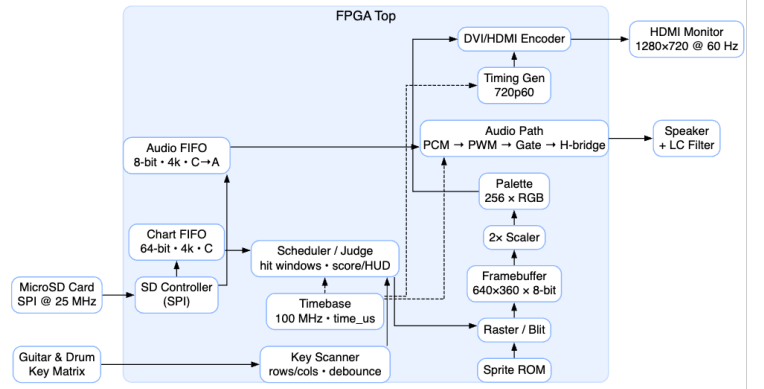


Fig. 1. System architecture with shared timebase and dual-clock FIFOs.

A. Clocking and CDC

Our CDC strategy is summarized in Table II. We treat the domains as asynchronous and pass data through dual-clock FIFOs for single-bit strobes.

TABLE II
CLOCK DOMAINS AND CDC POLICY

Domain	Freq.	To/From	CDC Mechanism
Control	100 MHz	SD	dual-clock FIFOs
SD/SPI	25 MHz	Control	dual-clock FIFOs
Pixel	f_{pix} for 720p60	Control	dual-clock RAM (instance list)
Control	100 MHz	Pixel	RAM read-only in pixel domain

B. Data Flow

Memory. The SD controller reads 512-byte sectors into an 8-bit audio FIFO and a 60-bit chart FIFO. **Audio.** The audio controller drains the FIFO at 44.1 kSa/s and drives a one-bit PDM stream. **Video.** The compositor blits sprites into a 640 $\times$ 360 palettized framebuffer which is upscaled 2 $\times$ to 1280 $\times$ 720 and TMDS-encoded. **Input.** A diode-isolated key matrix from the guitar is debounced and sampled each microsecond tick.

III. STORAGE

A. Initialization and Reads (SPI Mode)

We operate the SD card in SPI mode at 25 MHz. The controller issues a standard sequence: CMD0 (idle), CMD8 (voltage check), repeated ACMD41 (initialize), and CMD16 to fix block length at 512 B, followed by CMD17 single-block reads. Bytes stream out on MISO and are pushed into the appropriate FIFO. An MPEG-2 CRC32 module is included for added integrity checks on payloads.

B. Throughput Headroom

SPI at 25 MHz provides ≈ 3 MB/s. Audio consumes ≈ 43 kB/s ($8\text{-bit} \times 44.1$ kSa/s). Chart traffic is negligible (tens of kB per song, prefetched). We maintain generous FIFO space so a long-latency sector does not cause underrun.

IV. AUDIO SUBSYSTEM

A. Runtime FSM and Buffering

Figure 2 shows the FSM: RESET $\rightarrow$ IDLE $\rightarrow$ RETRIEVE $\leftrightarrow$ PLAYING. The controller refills when FIFO level crosses a low watermark and returns to steady drain at a high watermark. The read pointer advances at exactly 44.100 Hz derived from `time_us`.

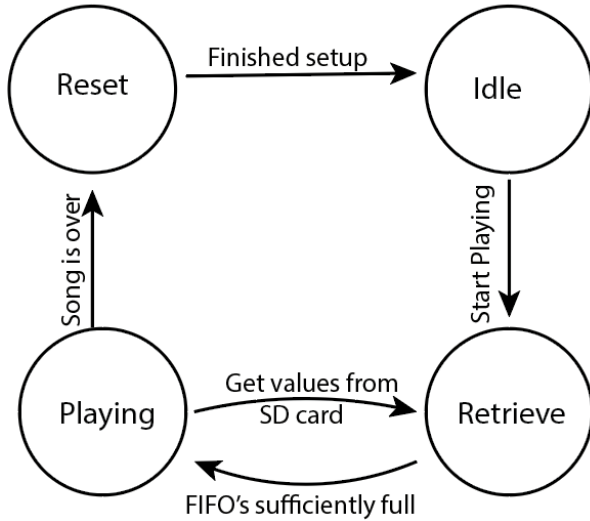


Fig. 2. Audio controller FSM (fill/drain around FIFO watermarks).

B. PDM Output

We implemented both PWM and first-order PDM. PDM integrates the error and emits a one-bit stream whose average equals the input sample. In practice, PDM provided cleaner perceived audio for the same board and speaker.

C. Module Connectivity

Figure 3 shows the SD controller writing into the 8-bit FIFO in the SPI domain while the audio controller reads in the 100 MHz domain. Ready/valid strobes plus watermarks enforce backpressure-free streaming.

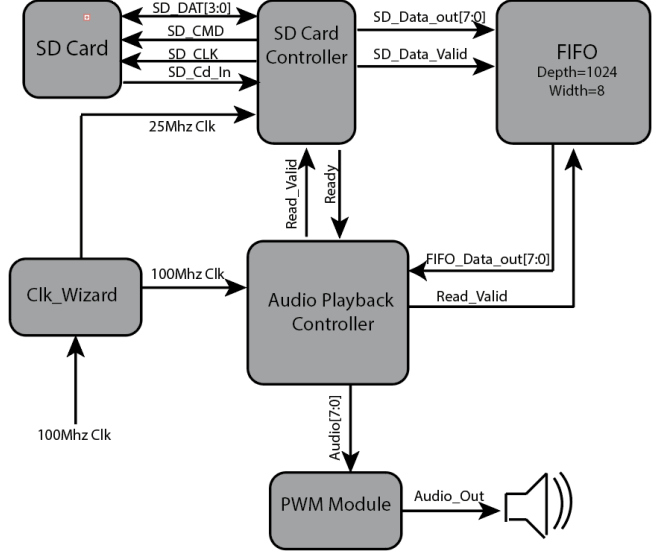


Fig. 3. Audio path: SD $\rightarrow$ FIFO $\rightarrow$ audio controller $\rightarrow$ PDM.

V. VIDEO SUBSYSTEM

A. Sprite/Instance Compositor

The control domain writes an *instance list* (`{sprite_id, x, y, frame, flags}`) into a dual-clock RAM. At pixel rate the compositor fetches glyphs from a sprite ROM and writes 8-bit palette indices into a 640×360 framebuffer.

B. Scaling, Palette, and TMDS

The scaler performs $2\times$ replication to 1280×720 . Palette lookup converts indices to RGB_{8:8:8}. A TMDS encoder and OSERDES serialize the data and clock to drive an HDMI monitor at 720p60 (Fig. 4).

C. Why Palettize?

A full $1280 \times 720 \times 24$ -bit buffer would exceed on-chip BRAM. Palettization slashes storage by $3\times$ and the mid-res buffer reduces pixel traffic by $4\times$, yet the final output remains 720p60 after upscaling.

VI. CONTROL, CHARTS, AND JUDGMENT

A. Global Timebase and Chart Format

A 32-bit microsecond counter, `time_us`, increments at 100 MHz. Each chart entry is a 60-bit packed record:

```
time_us[31:0] | lane[3:0] || dur[15:0] | vel[7:0].
```

Entries stream from SD into a chart FIFO; logic always has the next event available and never blocks on I/O.

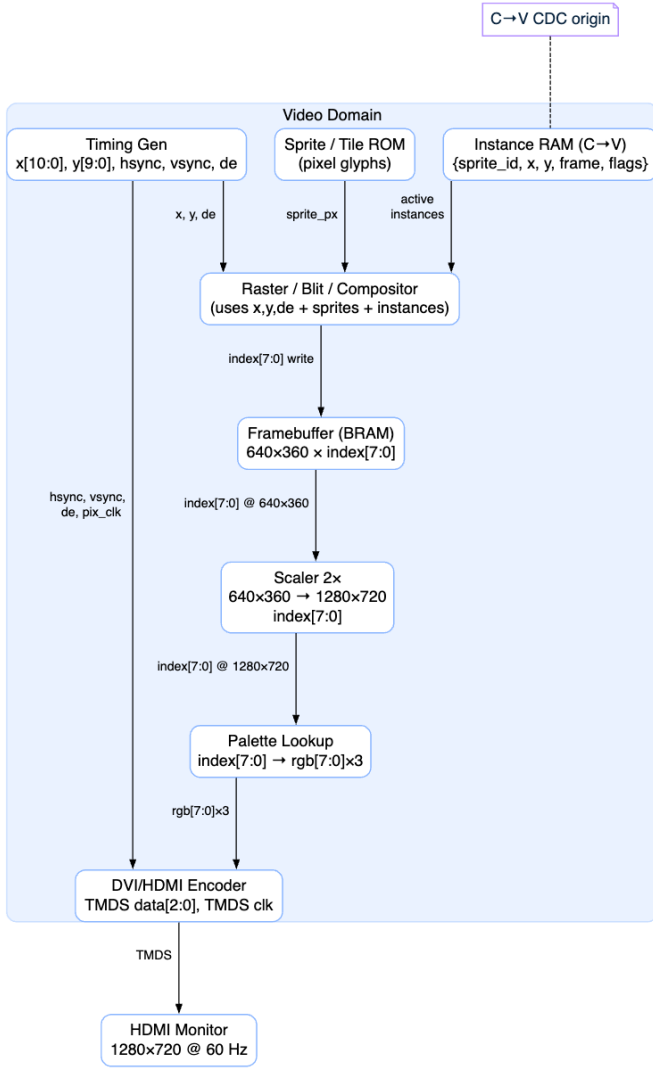


Fig. 4. Video pipeline: timing gen, compositor, palettized framebuffer, scaler, palette, TMDS.

B. Judge FSM and Hit Windows

Figure 5 illustrates the judge. ARM when $\text{now} \geq t_{\text{event}} - \text{preroll}$; Awaiting opens a window $[\text{early}, \text{late}]$; an edge on the correct lane produces SCORE, otherwise MISS on timeout. Sustains require the input to remain asserted until $\text{now} \geq t_{\text{event}} + \text{dur}$.

C. MIDI-to-Chart Toolchain

A Python tool converts MIDI note-on/off to chart records in microseconds: it (i) reads tempo/map, (ii) assigns lanes by pitch-range heuristics, (iii) computes absolute μs using cumulative tempo, and (iv) emits packed 60-bit words consumed directly by the FPGA.

VII. EVALUATION

A. Latency and Throughput Budget

Table III summarizes the dominant contributors. Visual updates are written into the active buffer, so HUD effects typ-

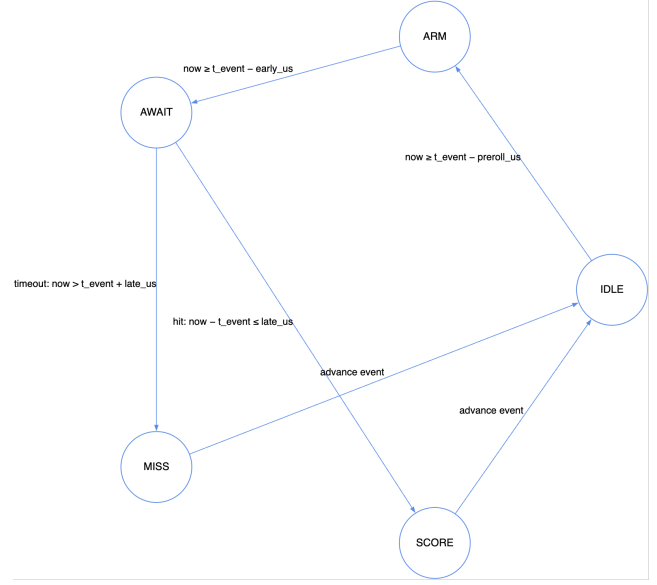


Fig. 5. Judge FSM with configurable early/late windows and sustains.

ically appear within the same frame; measured button→HUD feedback remained under 10 ms.

TABLE III
THROUGHPUT/LATENCY BUDGET (TYPICAL)

Component	Latency (ms)
Key-scan + debounce (configurable)	0.5
Judge decision (combinational + FSM step)	0.05
Instance write → compositor pipeline	0.5
Scan-out to visible pixel (intra-frame)	8.3
End-to-end (typical)	3–9

B. Resource Utilization (post-synth)

Vivado reported:

- **Block RAM.** RAMB36E1 = 66 total. Principal arrays: $56k \times 24$ (48 BRAM36), $56k \times 8$ (16 BRAM36), and a small 1014×60 (2 BRAM36). These account for framebuffer, glyphs/palette/metadata.
- **DSP.** $1 \times \text{DSP48E1}$ (sprite math), with scaler/palette in LUTs.
- **LUT/FF.** $\text{FDRE} \approx 2678$, $\text{LUT6} \approx 2932$ (others proportional). SerDes: $6 \times \text{OSERDESE2}$, TMDS drivers: $4 \times \text{OBUFDS}$.

C. Memory/Area Overheads and Tradeoffs

Palettization and a mid-res buffer push nearly all frame storage into on-chip BRAM, trading a palette stage for a $3\times$ memory reduction. Dual-clock FIFOs add μs -scale latency but completely decouple bursty SD access from steady audio/video consumption.

D. Use Cases and Goal Coverage

Delivered features: two-song menu, guitar input, chart rendering, judge/score/HUD, stable PDM audio, and tight A/V lock. Minimum and ideal goals were met; stretch goals were partially realized. With more time, we would have implemented:

- **Stereo audio** (duplicate PDM),
- **Second player** (additional key-matrix rows/cols and more instance capacity),
- **Richer effects** (affine sprite transforms or scanline warps with modest extra BRAM/DSP to give illusion of perspective).

VIII. MEASUREMENT AND VALIDATION

A. A/V Lock Method

We instrumented three signals: (1) `key_edge` after de-bounce, (2) a `judge_score` pulse, and (3) a `hud_marker` write-enable into the compositor. A logic analyzer captured these along with `de` and the HDMI pixel clock. The time between (1) and the first visible on-screen HUD pixel bounded input→feedback latency, which measured < 10 ms across trials.

B. Module Tests

Cocotb tests cover UART, command FIFO, clock generators, video timing, line buffer and convolution units, TMDS encode/serialize, the judge FSM, and the audio player. Integration smoke builds are scripted via `build.tcl`.

IX. IMPLEMENTATION/REPLICABILITY POINTERS

Key RTL modules (`hdl/`):

- *Storage/IF*: `sd_controller.sv`, `RAM/FIFO` wrappers.
- *Audio*: `play_song.sv` (FSM), `audio_pdm.sv`.
- *Video*: `video_sig_gen.sv`, `image_sprite*.sv`, `compositor`, `framebuffer`, `tmbs_encoder.sv`, `tmbs_serializer.sv`.
- *Game*: `timebase.sv`, `chart_fifo.sv`, `judge.sv`, `top_hero_control.sv`, `top_level.sv`.

X. CONCLUSION

FPGA Hero demonstrates that a small Spartan-7 can host a full rhythm game: SD-streamed audio, a compact yet expressive 720p video engine, and a judgment pipeline that stays phase-locked by design. Future work includes stereo audio, perspective/affine visual effects, multi-player support, and refined constraints/pipelining to formally close pixel-domain timing.

CONTRIBUTIONS

Fernando focused on SD+audio (SPI controller, audio FIFO/flow, PWM/PDM) and co-designed the judge/game and guitar hardware. Nicolas focused on video+charts (sprite/instance pipeline, framebuffer/scaler/palette, MIDI toolchain) and co-designed the judge/game and guitar; he also created all figures. We jointly defined the top-level architecture and split the writing evenly. No additional credits.

CODE

We do not include source in this paper. The complete repository (RTL, charts, tests, build scripts) is available at <https://github.mit.edu/6205F25/fa25-6205-team11.git>.

REFERENCES

- [1] AMD Xilinx, “7 Series FPGAs SelectIO Resources,” and application notes for TMDS/HDMI, 2023.