Gamma-Code Encoder on FPGA Using a Moore Finite State Machine

Hilal Gure
Email: hilal_shide@live.com

***Abstract*—We implement a modular *Gamma-code encoder* in VHDL on an FPGA. A Moore finite state machine (FSM) interprets 2-bit tokens (dot, dash, bar) from a shift register and drives an LED blink pattern with a 0.25 s base tick, while the selected symbol is shown on a seven-segment display. The design includes a lookup table (LUT), shift register, quarter-second clock divider, and output display logic. We provide unit testbenches for each module in ModelSim and validate correct timing and sequencing on hardware. The approach demonstrates clear module boundaries, robust timing based on a generic counter, and practical debugging strategies for FSM-based designs.**

***Index Terms*—VHDL, FPGA, Finite State Machine, Timing, Gamma Code, Seven-Segment Display.**

I. Introduction

The purpose of this project is to design and implement a Gamma-code encoder using a Finite State Machine (FSM) on an FPGA board. The Gamma code uses a pattern of pulses with different durations to represent symbols such as letters, numbers, and special characters. Each symbol is encoded as a sequence of short (dot), medium (dash), and long (bar) pulses.

A dot is defined as four consecutive 0.25-second pulses, a dash lasts 0.75 seconds, and a bar 1.5 seconds. The system uses the switches `SW[3:0]` and push-buttons `KEY[1:0]` as inputs. The switches select which symbol to encode, while `KEY1` starts the display of the Gamma code on `LEDR0`. The button `KEY0` functions as an active-high reset and lights up `LEDR9`.

The goal is to design a digital system capable of translating a selected symbol into LED blinks with correct timing durations. The solution is implemented in VHDL and verified through both simulation and hardware testing on an FPGA.

II. System Design

The design is divided into several submodules, each with a distinct function. This modular structure improves readability, simplifies debugging, and allows isolated testing of each part. Some modules, such as the seven-segment display driver, shift register, and counter, were reused and adapted from earlier laboratory exercises.

The top-level entity (`gamma_code_encoder`) connects all submodules and handles input and output between switches, buttons, LEDs, and the display.

A. Gamma Look-Up Table (LUT)

The LUT takes a 4-bit value from the switches and converts it to an 8-bit Gamma code. Each code represents a symbol or character and consists of four 2-bit sequences where:

- `00` = short pulse (dot)
- `01` = medium pulse (dash)
- `10` = long pulse (bar)
- `11` = null (no pulse)

The mapping is hardcoded using a `case` statement.

B. Shift Register

The shift register stores the 8-bit Gamma code and outputs 2 bits at a time to the FSM. After each shift, the lower bits are filled with `00` to maintain length. The FSM uses the received 2-bit sequence to determine the duration of each LED blink.

C. Quarter-Second Clock Generator

This module divides the FPGA's 50 MHz input clock by a constant value ($k = 12{,}500{,}000$) to generate a tick every 0.25 s. The tick signal (`tick_qsec`) serves as the base timing for the FSM.

D. Finite State Machine (FSM)

The FSM acts as the control unit and determines how the Gamma code is processed and displayed through LED blinks. Implemented as a Moore machine, the LED output depends only on the current state, ensuring predictable timing behavior. The FSM consists of five main states:

- `S1_LOAD`: Load new Gamma code from LUT into the shift register.
- `S2_CHECK`: Check if more data remains.
- `S3_BLINK`: Turn LED on for the duration defined by the 2-bit code.
- `S4_PAUSE`: Short pause between symbols.
- `S5_DONE`: Wait for a new start signal.

III. Verification

Each submodule was verified individually using dedicated testbenches in ModelSim. The testbenches simulate input stimuli and observe outputs to confirm correct functionality.

A. Gamma LUT

The LUT testbench cycles through all valid 4-bit inputs and checks that the corresponding 8-bit Gamma codes match the specification. Invalid inputs such as `1111` are handled correctly.

B. Shift Register

This testbench validates that each shift operation correctly outputs the upper 2 bits and appends `00` at the least significant end. The `finished` signal is asserted after four shifts, confirming that all data have been sent.

C. Counter and FSM

The counter is verified to produce a rollover signal exactly every 0.25 seconds (simulated with a lower value of k for speed). The FSM testbench confirms proper state transitions, LED timing, and correct behavior upon reset and start conditions.

D. Seven-Segment Display

Tests confirm that the 4-bit input value is translated to the correct segment pattern for each symbol. Invalid inputs result in a blank display.

IV. Results and Discussion

The system performs as expected. When `KEY1` is pressed, `LEDR0` blinks according to the Gamma code of the selected symbol. The LUT generates the correct 8-bit sequence, the shift register outputs the correct 2-bit tokens, and the FSM interprets them accurately, producing timed LED blinks.

The seven-segment display correctly shows the chosen symbol, and the quarter-second counter provides a stable timing base. Debugging LEDs were used to visualize internal signals such as `tick_qsec`, `shift_en`, `load_reg`, and `finished`, helping to identify timing issues.

Common challenges included incorrect tick generation and FSM synchronization issues. Replacing an earlier custom counter with the reusable generic `counter_slow` module resolved instability. Testing with ModelSim significantly improved debugging efficiency.

Three key lessons were learned:

- Start simulation early in ModelSim to identify design issues before hardware synthesis.
- Keep FSM logic simple and deterministic; unnecessary states complicate debugging.
- Use visual debug signals (LEDs) to monitor internal states on hardware.

V. Conclusion

A functional Gamma-code encoder controlled by a Moore FSM was successfully designed and implemented in VHDL. The system correctly interprets user input from switches and buttons and outputs the corresponding timed LED blink sequence.

This project provided hands-on experience in modular VHDL design, FSM control, and simulation-based verification. The importance of early simulation and incremental testing was highlighted throughout development. Future improvements could include adding sound output, supporting more symbols, or providing visual feedback through an LCD or VGA interface.

Figures and waveforms

All diagrams, RTL views, and simulation waveforms are available in: [‘docs/latex/figures/‘](docs/latex/figures)

- ** Mamma FSM.png** - FSM architecture showing timing and control of ‘led$_out$‘, ‘$shift_en$‘, and‘$load_reg$‘. − * * $GammaLUT.png$ * * − $LUTmappingthe4$ − $bitswitchinputtoan8$ − $bitgammacode.$ − * * $Skiftregister.png$**−$Shiftregisterusedforsequential2$− $bittokenoutput.$ − * * $Teller.png$ * *|$Quarter$ − $secondcounterdividingthe50MHzclock.$ − * * $RTLVIEWER.png$ * * − $RTLschematicofthetop$ − $levelencoder.$ − * * $Technologymapviewerpostmapping.png$ * *|$FPGAresourcemappingview.$ − * * $Statemachineviewer.png$ * * − $FSMstatediagram.$ − * * $Statetable.png$ * * − $FSMtransitiontable.$ − * * $wavechartcounter_slow.png$ * *|$Counter$ − $waveformverification.$−**$gammaLUTwaveform.png$* *|$LUTwaveform.$ − * * $wavechartgammashiftreg.png$ * * − $shiftregisterwaveform.$ − * * $wavechartfsm.png$ * *|$FSMwaveform.$ − * * $wavechartsevensegment.png$ * *|$Seven-segmentdisplaywaveform.$