

Tab 1

Capstone Final Report

TEAM 9

LATTICE FPGA

Author:

Mohammad Alshaiji <alshaiji@pdx.edu>

Riley Cox <rilco@pdx.edu>

Nathaniel Fraly <nfraly@pdx.edu>

Haoyang Han <haoyang@pdx.edu>

Xiang Li <shane29@pdx.edu>

Industry sponsor :

Rahul Koche

Faculty Advisor :

Roy Kravitz

Date: 5/14/2025

Version 0.1

Table of Contents

[Table of Contents](#)

[Executive Summary](#)

[Background](#)

[Research](#)

[Open Source Projects](#)

[Patents, Papers, White Papers, Articles, Conference Proceedings](#)

[Requirements](#)

[Stakeholders](#)

[Requirements](#)

[Objectives and Deliverables](#)

[Approach](#)

[Initial Strategy: Toolchain Selection and Phased Development \(Weeks 1-4\)](#)

[Dual Development Paths\(Weeks 4-9\)](#)

[Radiant Challenges and Yosys Progress\(Weeks 9-12\)](#)

[Strategic Shift: Prioritization of Yosys for Functional System Delivery \(Weeks 12-13\)](#)

[Focused Yosys System and hardware Integration \(Weeks 13-18\)](#)

[Sustained Radiant Investigation](#)

[Final product](#)

[Design](#)

[Hardware](#)

[Bill of materials](#)

[Block Diagram](#)

[Schematic](#)

[Layout](#)

[Enclosure](#)

[Toolchain](#)

[Toolchain Setup and Usage](#)

[Open-Source Toolchain](#)

[Commercial Toolschain](#)

[Hardware](#)

[SPI module](#)

[Open Source RISC-V core](#)

[Buttons](#)

[Firmware](#)

[Buttons and Screen control](#)

[Screen Control Logic](#)

[Test Plan](#)

[Results](#)

[Post Mortem](#)

[Future Work](#)

[Project Resources](#)

[Collaboration sites and repositories](#)

[Tools with version](#)

[Other resources](#)

[Budget](#)

[Conclusion](#)

[Appendices](#)

[Tooling](#)

[Developer's Manual](#)

[1. System Assembly](#)

[2. System Programming](#)

[2.1. FPGA Bitstream Generation \(Hardware Synthesis and Implementation\)](#)

[2.2. C Firmware Compilation and Loading](#)

[3. Testing Subsystems](#)

[User's Manual](#)

[WSL Setup](#)

[Dependencies](#)

[Ubuntu/Debian](#)

[Fedora](#)

[Arch](#)

[RISC-V GNU Toolchain](#)

[Open Source Toolchain Setup](#)

[Ubuntu/Debian](#)

[Arch](#)

[Fedora](#)

[Building on the UPduino board](#)

[Migrating Icicle from UPduino V1 to V2.1](#)

[Test Plan](#)

[1. Test Environment Setup](#)

[2. Test Cases](#)

[2.1. Basic System Power-Up and Initialization Tests](#)

[2.2. Button Control Functionality Tests](#)

[2.3. Power System and Enclosure Tests](#)

[3. Test Procedures](#)

[4. Acceptance Criteria](#)

[Toolchain Comparison: Radiant vs. Yosys](#)

Executive Summary

This project, sponsored by **Lattice Semiconductor**, involved the creation of a portable, handheld electronic device designed to display a vibrant range of colors on an LCD screen. Lattice is a leading company that makes special, low-power, re-programmable computer chips called FPGAs, which served as the "brain" for our device. The goal was to build a finished product that allows a user to change the screen's color and brightness with the press of a button, all neatly packaged in a custom, 3D-printed heart-shaped case and powered by a rechargeable battery.

The most important part of our mission was not just building the device, but *how* we built its "brain." We were tasked with a grand experiment: to build the system in two different ways and compare the experience. Think of it like building the same LEGO model using two different sets of instructions. One set was the official, professional software provided by Lattice (**Lattice Radiant**). The other was a popular, free set of tools created and shared by the open-source community (**Yosys**). Lattice wanted to know which "instruction set" was easier to use and more efficient for creating our final product.

Our team began by trying to build with both toolsets at the same time. We quickly found that the open-source tools were very effective, and we made steady progress, successfully building the device's brain, teaching it to communicate with the screen, and react to button presses. However, we faced significant and persistent challenges with the professional Lattice Radiant software. It proved to be very difficult to get our chosen open-source "brain" (a RISC-V core) to work correctly within the Radiant environment. After many weeks of troubleshooting, and in agreement with our sponsor, we shifted our focus to ensure we could deliver a high-quality, working product.

In the end, this project was a success. We delivered a **fully functional and polished final product** using the open-source tools. The device does everything we set out to do: it displays a vibrant range of colors and text, responds to button presses, and is fully portable thanks to its battery and custom enclosure. While the difficulties with synthesizing the "brain" we used for Yosys with the Radiant software prevented us from completing a direct, side-by-side comparison of power usage, our work provided invaluable real-world feedback to our sponsor about the challenges of integrating popular open-source components with their proprietary tools. Radiant is better suited for structured, high-assurance design environments and offers industry-aligned analysis and enforcement. Yosys is more accessible for exploratory prototyping and open-source flexibility.

Background

Lattice Semiconductor is an FPGA company with headquarters in Hillsboro, Oregon. They market themselves as the low-power programmable logic leader. According to their SEC 10-Q filing on November 04, 2024, their total assets as of that date totaled \$853,661,000. Their total revenues for the quarter ending in September 2024 were \$127,091,000. From their investor relations statements, they describe themselves as “[solving] customer problems across the network, from the Edge to the Cloud, in the growing communications, computing, industrial, automotive, and consumer markets.”(Lattice)

FPGAs serve a particular niche of the market concerned with parallel processing, high-speed processing and the ability to reconfigure hardware. Such properties are leveraged by communication, automotive, aviation, and artificial intelligence systems among others. Traditionally, FPGAs stand as an alternative to the use of ASICs and microcontrollers. For this project specifically, the sponsor was interested in evaluating the trade-offs between the Lattice Radiant toolchain and open-source alternatives through the development of an FPGA-based system that controls an SPI-based LCD screen. The system was designed to feature user interaction via two buttons for adjusting brightness and color, housed in a 3D-printed enclosure and powered by a battery. A series of power profiles, generated during different stages of development using various low-power design techniques, were to be collated to compare the impact of each toolchain on key factors such as synthesis quality, and power consumption.

The team is beginning this project without having previous work to consider within our own designs. We are leveraging a previous team’s toolchain experiences, as provided by Rahul, to select the open-source toolchain that supports the Upduino board, and UltraPlus FPGA we will be using.

Research

Open Source Projects

To simplify the implementation of the project and to avoid reinventing the wheel, Open-source Verilog IPs will be used instead of relying on proprietary Lattice IP. This would also simplify the comparisons between the open source and Lattice toolchain, allowing for the reduction of influences on the final power results. opencores.org has been recommended by our industry sponsor as a suitable place to explore and obtain said IPs.

Patents, Papers, White Papers, Articles, Conference Proceedings

-The IEEE paper, [Development of an FPGA based low power message displaying system using scanning technique](#), is an exploration conducted by Ali et al. on developing techniques for the reduction of power usage in FPGAs that interface with a screen. While the system explored in the paper is more complex than the system we intend to design, it provides inspiration on potential avenues of power reduction, and could inspire the methods we utilize in developing and tracking our power profiles and their results.

-[Pong game on FPGA with CRT or LCD display and push button controls](#), by Szabó et al, provides a useful example on where this project can take us. It provided useful information on how we could implement the push buttons for our device such that it can interface correctly with an LCD. It also provides a suitable benchmark on our progress as we proceed through the project, i.e. developing the base product, implementing low-power techniques, and the toolchain comparison

-[SystemVerilog based design and implementation of LCD Controller IP Core](#), by Chandran O et al.,

provides a suitable source for examining the considerations taken in the design of an LCD controller IP. As it is written in SystemVerilog instead of Verilog, it allows us the opportunity to enhance our understanding of FPGAs by informing us of useful techniques while leaving the burden of implementation on our team. Since the IP being designed is for an LCD controller, this will be useful in determining and visualizing our own mechanism for LCD Interfacing.

Requirements

Stakeholders

1. Industry sponsor : Rahul Koche
2. Faculty Advisor : Roy Kravitz
3. User : Lattice Semiconductor
4. Project team: Nathaniel Fraly, Riley Cox, Mohammad Alshaiji, Xiang Li, Haoyang Han

Requirements

The main priority of this project is to be able to utilize off-shelf products provided by Lattice FPGA to complete a proof of concept design for an FPGA Color-mixing screen. This can be used in conjunction with an LCD screen utilizing SPI to display color/pattern changes on the press of a button, and contrast/brightness adjustments with the press of another. The proposed boards to be used are the UPduino boards using Lattice UltraPlus ICE40UP5K FPGA.

The project will start by developing a functional product without incorporating power-saving techniques, ensuring functionality. Three distinct low-power design implementations will be integrated and tested to assess their impact on performance and energy efficiency. Then a comparison of the Lattice Radiant and open-source toolchains will be conducted, evaluating usability, workflow efficiency, and synthesis quality. After finished, the product will be refined and enhanced with additional features, completing the development process.

1. Must control an SPI-based LCD screen to adjust brightness and color via two buttons.
2. Must use Lattice Radiant FPGA toolchain to develop FPGA bitstream.
3. Must use at least 3 kinds of low-power design techniques.
4. Must have a working final product.
5. Must be in an enclosure.
6. Must have Phase-1 complete by April
7. Should compare the ease of use and efficiency between the open-source toolchain and the Lattice Radiant toolchain.
8. Should collect and visualize power consumption data and compare the results of the open-source toolchain and the Lattice Radiant toolchain.
9. Should be encased in a 3D-printed enclosure.
10. Should be powered by a battery.
11. May be able to show meaningful text on the screen.
12. May be able to show images on the display.
13. May be able to act as a debugging tool to show the debug messages on screen.
14. May be able to interact with other sensors.

Objectives and Deliverables

1. A fully working SPI LCD display which uses FPGA for color-changing, two buttons for functionality and all inside a 3-D printed shell.
2. RTL source code, ideally in SystemVerilog
3. Associated apps or software for LCD display controlling
4. Report comparing the usage of Open Source toolchain and Lattice Radiant toolchain
5. Reports with statistics and graphs on power consumption of different solutions
6. User Manual
7. Bill of Materials
8. Project proposal
9. Weekly Progress Reports
10. Final report
11. ECE Capstone Poster Session Poster
12. Documentations and source code stored in Github repository

During the course of the project, we encountered significant compatibility issues between the Lattice Radiant toolchain and the open-source RISC-V core. As a result, our team spent a considerable amount of time diagnosing the problems and modifying the code to achieve successful synthesis and operation under Radiant. Consequently, the originally planned power-consumption objectives were deprioritized after agreement with our sponsor, and ultimately abandoned due to time constraints. The primary goal of the project remained to deliver a fully integrated product, complete with enclosure and internal PCB, capable of driving the SPI LCD display to show colors and controlled via button inputs.

Approach

The primary objective of this project was to design and build an FPGA-based system capable of controlling an SPI LCD screen, with user interaction facilitated by two buttons for adjusting color and brightness, all housed within a 3D-printed enclosure and powered by a battery. Along with the final product, we also aim to conduct a comparative analysis of the Lattice Radiant toolchain against an open-source alternative, evaluating aspects such as ease of use, workflow efficiency, synthesis quality, and power consumption.

Initial Strategy: Toolchain Selection and Phased Development (Weeks 1-4)

The project started with the selection of appropriate FPGA development toolchains. Lattice Radiant was designated as the proprietary tool for the Lattice iCE40 Ultraplus FPGA chip. For the open-source counterpart, the Yosys synthesis suite, in conjunction with Icestorm (place and route) and NextPNR (bitstream generation), was selected.

This decision was informed by documented successes of previous student teams employing this suite with the target iCE40 Ultraplus FPGA on the Upduino platform. In the early stages of development, both the industry advisor and academic advisor recommended that we begin development work with the

Lattice Radiant toolchain. This guidance was based on the anticipation that professional software tools like Radiant would generally offer greater stability and reliability compared to open-source alternatives, thereby potentially accelerating our initial development progress. The "Icicle" open-source RISC-V core was chosen as the processing unit for the FPGA implementation. The project's execution was initially structured into three distinct phases: first, the development of a functional prototype capable of displaying 7-15 solid colors with two-button control (color selection, brightness/contrast adjustment), enclosed and battery-powered; second, the implementation of text display functionality on the LCD; and third, the addition of image display capabilities.

Dual Development Paths(Weeks 4-9)

The team worked on a dual-path development strategy, with the intent of making concurrent progress in synthesizing the design using both the Yosys toolchain and Lattice Radiant toolchain. The open-source Yosys path yielded consistent advancements: the Icicle RISC-V core was successfully synthesized and verified operational on the UPduino v2.1 hardware platform. Development of the SPI module, essential for LCD communication, was initiated, involving investigation of available SPI IP cores and reference to Adafruit ST7735 libraries. Concurrently, C firmware development for screen initialization and control functions was started. In contrast the Lattice Radiant path encountered immediate technical obstacles. Initial attempts to synthesize the Icicle RISC-V core within Radiant were met with "duplicate module" errors and other synthesis anomalies. Furthermore, complications with FTDI drivers impeded board communication and reliable bitstream programming.

Radiant Challenges and Yosys Progress(Weeks 9-12)

The period spanning Weeks 9 through 12 represented a critical juncture for the project. On the Lattice Radiant path, persistent efforts continued. After considerable exploration and debugging, issues related to the FTDI driver were largely resolved. But fundamental difficulties in instantiating the Icicle RISC-V core within the Radiant environment still remained. The synthesis process generated an excessive number of errors and integration faults, blocking any further progress on Radiant toolchain development. Concurrently, development within the Yosys toolchain, while generally more progressive, encountered its own unique and time-consuming challenge. Although the Icicle RISC-V core had been successfully instantiated, a critical problem emerged where a necessary clock signal was not being correctly output from the FPGA. After investigation, this fault was traced to an inconsistency between the pin assignments detailed in the Upduino v2.1 board documentation and the actual pinout designated on the physical board's silkscreen. After solving the clock signal issue, we successfully integrated the custom-developed SPI controller with the Icicle RISC-V core. This integration needed modifications to the RISC-V memory map within `icicle.sv` to enable memory-mapped I/O control of SPI transactions directly from C firmware. The functionality of this integrated module was subsequently validated through logic analyzer-based signal verification.

Strategic Shift: Prioritization of Yosys for Functional System Delivery (Weeks 12-13)

The widening gap in developmental progress and the persistent, unresolved compatibility issues between the Lattice Radiant toolchain and the open-source Icicle RISC-V core precipitated a major strategic realignment around Weeks 12-13. Following consultations with the industry sponsor, the team made the decisive pivot to prioritize all primary development efforts on the Yosys toolchain. This shift aimed to ensure the timely delivery of a functional hardware system as per the project's core requirements. A direct consequence of this reprioritization was the deferral, and ultimate abandonment, of the originally planned in-depth power consumption analysis and low power design, due to the resultant time constraints.

Focused Yosys System and hardware Integration (Weeks 13-18)

With a focus on Yosys Toolchain, system integration proceeded at an accelerated pace. The button control module, providing user input for color and brightness adjustments, was successfully integrated and tested. This phase involved debugging issues such as non-deterministic screen behavior upon button module implementation and a hardware redesign of the button interface circuitry upon discovery that the iCE40UP5k FPGA lacks internal pull-down resistors. Firmware development included C functions for comprehensive screen initialization and text rendering primitives like ``draw character``, ``draw text``, and ``fill rectangle``. Parallel hardware development efforts included iterative refinement of the 3D-printed enclosure, incorporating sponsor feedback for a heart-shaped design, and the design of a custom PCB to integrate all electronic components, during which the team transitioned PCB design tools from EZCAD to KiCAD for enhanced capabilities. A functional protoboard was assembled and tested to validate system operation prior to final PCB fabrication and assembly.

Sustained Radiant Investigation

With the primary focus on the Yosys pathway, efforts to diagnose and resolve the complex issues within the Lattice Radiant environment continued, as a secondary objective. These investigations, conducted with guidance from advisors, addressed specific challenges such as anomalies in PLL (Phase-Locked Loop) routing by modifying tool-generated wrapper files, and discrepancies in the handling of initial hardware blocks during synthesis compared to the Yosys flow.

Final product

To integrate a final product, the power supply solution was finalized, successfully integrating a 3.7V LiPo rechargeable battery with a power boost converter circuit to deliver a stable 5V supply to the UPduino board and associated components; this system was validated with the prototyped hardware. The design for the custom Printed Circuit Board (PCB), intended to consolidate all system electronics, reached completion. Simultaneously, the design for the 3D-printed enclosure was being finalized to accommodate two user buttons, a power switch, and a charging port, with previous prototypes having been printed and evaluated.

Alongside these hardware finalization efforts, significant engineering resources remained dedicated to diagnosing and attempting to resolve the persistent issues with the Lattice Radiant toolchain. Radiant team continued in-depth debugging, managing to reduce synthesis warnings, resolve specific clock generation and timing interactions involving the Phase-Locked Loop (PLL), and confirm via synthesis reports that the compiled C application was loading into the target memory. This effort even extended to implementing dual-port Block RAM (BRAM) to meet design requirements. Despite these advances, achieving fully correct program execution on the Radiant-based System-on-Chip (SoC) remained an ongoing challenge.

Design

This section details the hardware and software design of the FPGA-based color-mixing monitor. The system was developed to meet the core requirements of interfacing with an SPI-based LCD, controlled by user-input buttons, and housed within a custom enclosure.

Hardware

The central processing unit of the system is the **Lattice iCE40 UltraPlus ICE40UP5K FPGA**, resident on an **UPduino development board**. While the project proposal initially mentioned the UPduino v3.0, the team primarily utilized the UPduino v2.1 provided by sponsor for development and porting of the Icicle RISC-V core. This FPGA was chosen for its low power consumption and adequate logic resources for implementing the RISC-V soft processor and peripheral controllers.

Visual output is handled by an **Adafruit ST7735R 1.8-inch Color TFT LCD**. This display interfaces with the FPGA via an SPI (Serial Peripheral Interface) bus, allowing for efficient data transfer for screen updates, including solid color displays and text.

User interaction is facilitated through **push buttons**. The core design consistently specified two buttons for primary user functions: color selection and brightness adjustment. The physical enclosure was designed to accommodate these two user buttons alongside a separate power switch.

The system is designed to be portable and is powered by a **battery-based power supply system**. This comprises a 3.7V LiPo rechargeable battery, with its voltage stepped up to the required 5V for the UPduino board and other components using a Adafruit power boost converter board.

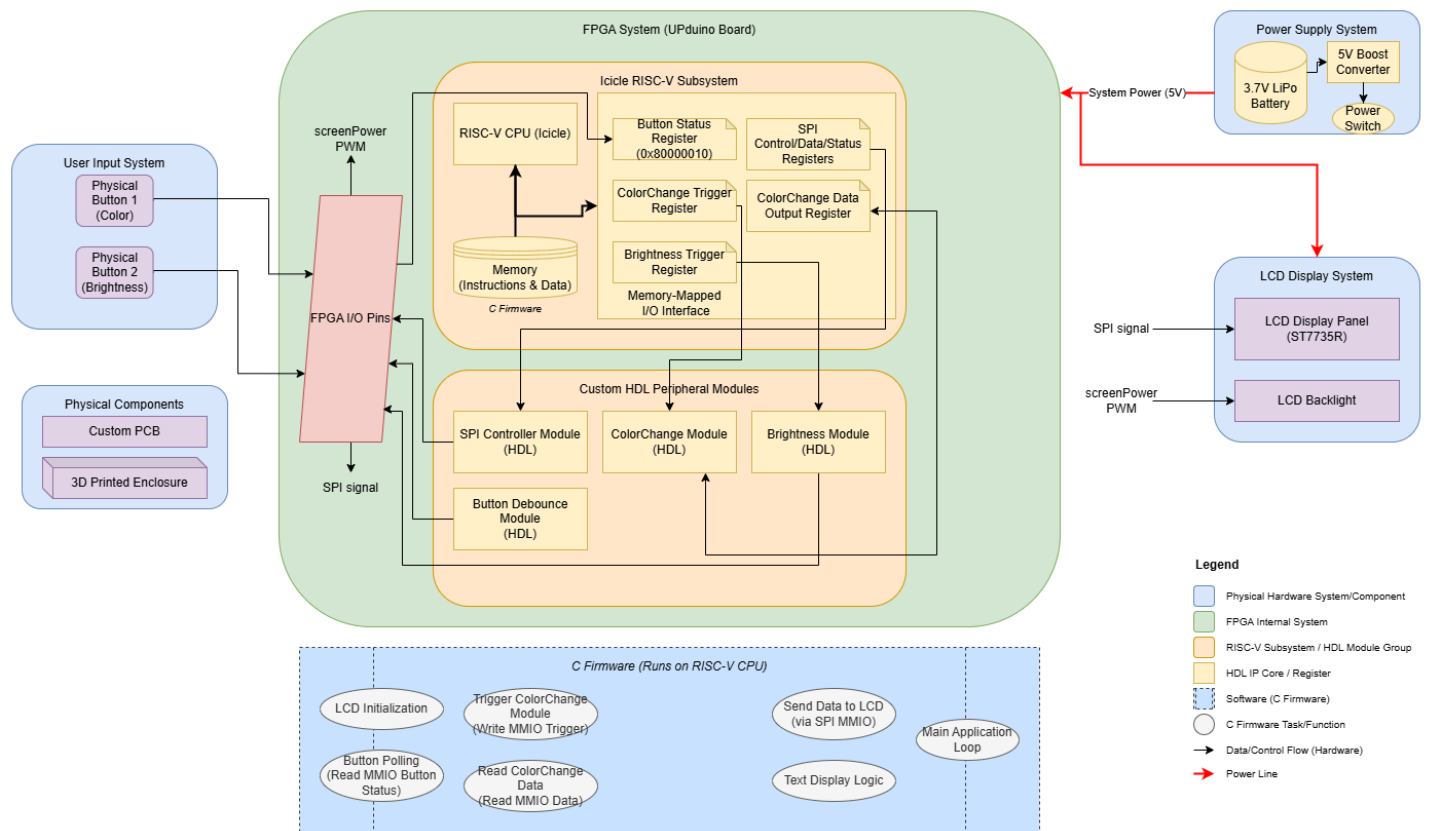
The PCB integrates the UPduino board, power circuitry, buttons, and connectors for the LCD in a compact and robust manner. The 3D-printed enclosure in heart shape, provides protection for the electronics and a user-friendly interface.

The physical assembly consists of the **custom Printed Circuit Board (PCB)** and the **3D-printed enclosure**.

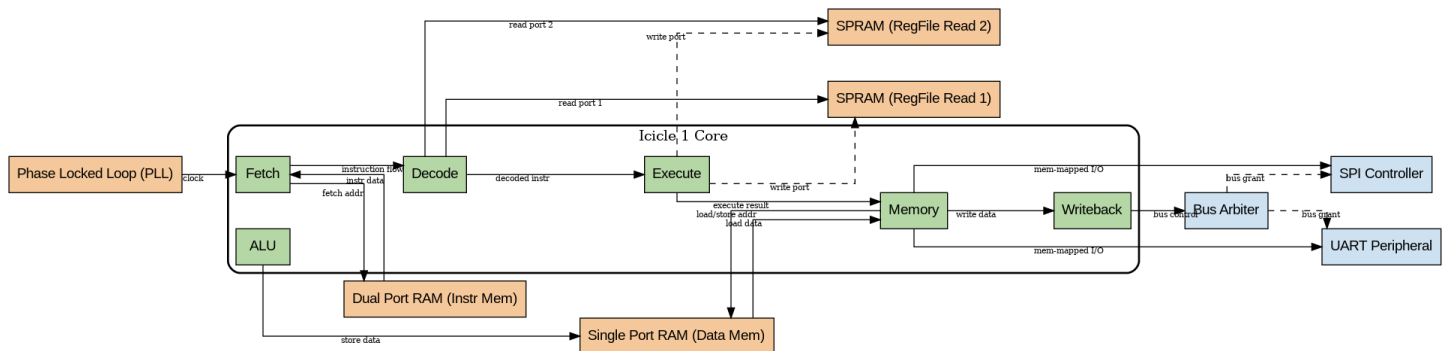
Bill of materials

Reference	Qty	Value	DNP	Exclude from BOM	Exclude from Board	Footprint	Datasheet
1.44"TFT1	1	~				ECE412Library:1.44_ TFT Display	~
C1,C2,C3	3	2.2u	DNP			Capacitor_SMD:C_0805_2012Metric	~
R1,R2,R3	3	15k				Resistor_SMD:R_0805_2012Metric	~
SW1,SW2,SW3	3	~				Button_Switch_THT:SW_PUSH_6mm	~
SW4	1	~				Button_Switch_THT:SW_Slide-03_Wuerth-WS-SLTV_10x2.5x6.4_P2.54mm	~
U1	1	~				ECE412Library:PowerBoost-500	~
Upduino1	1	~				ECE412Library:UPduinoV3.1	~

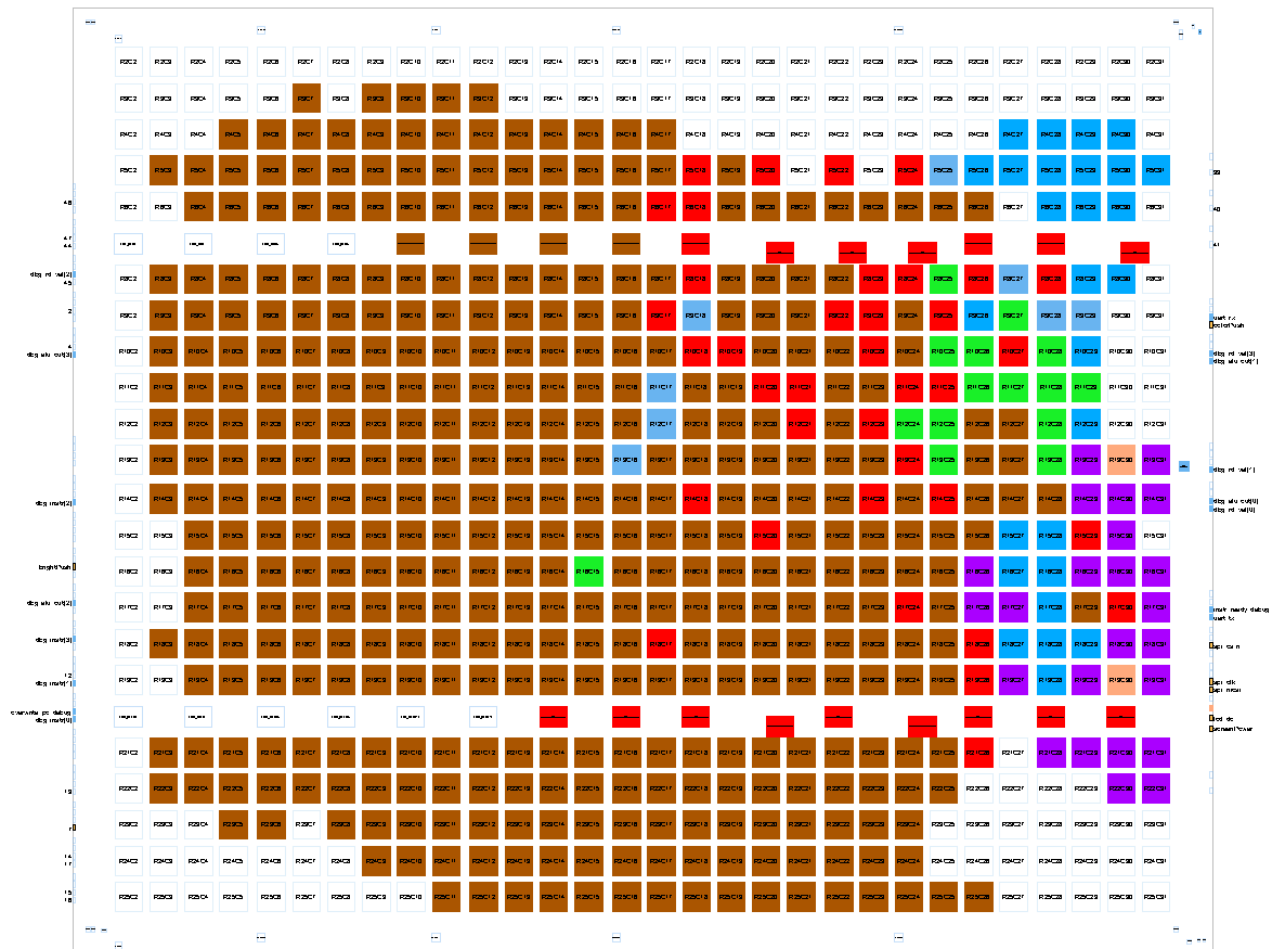
Block Diagram



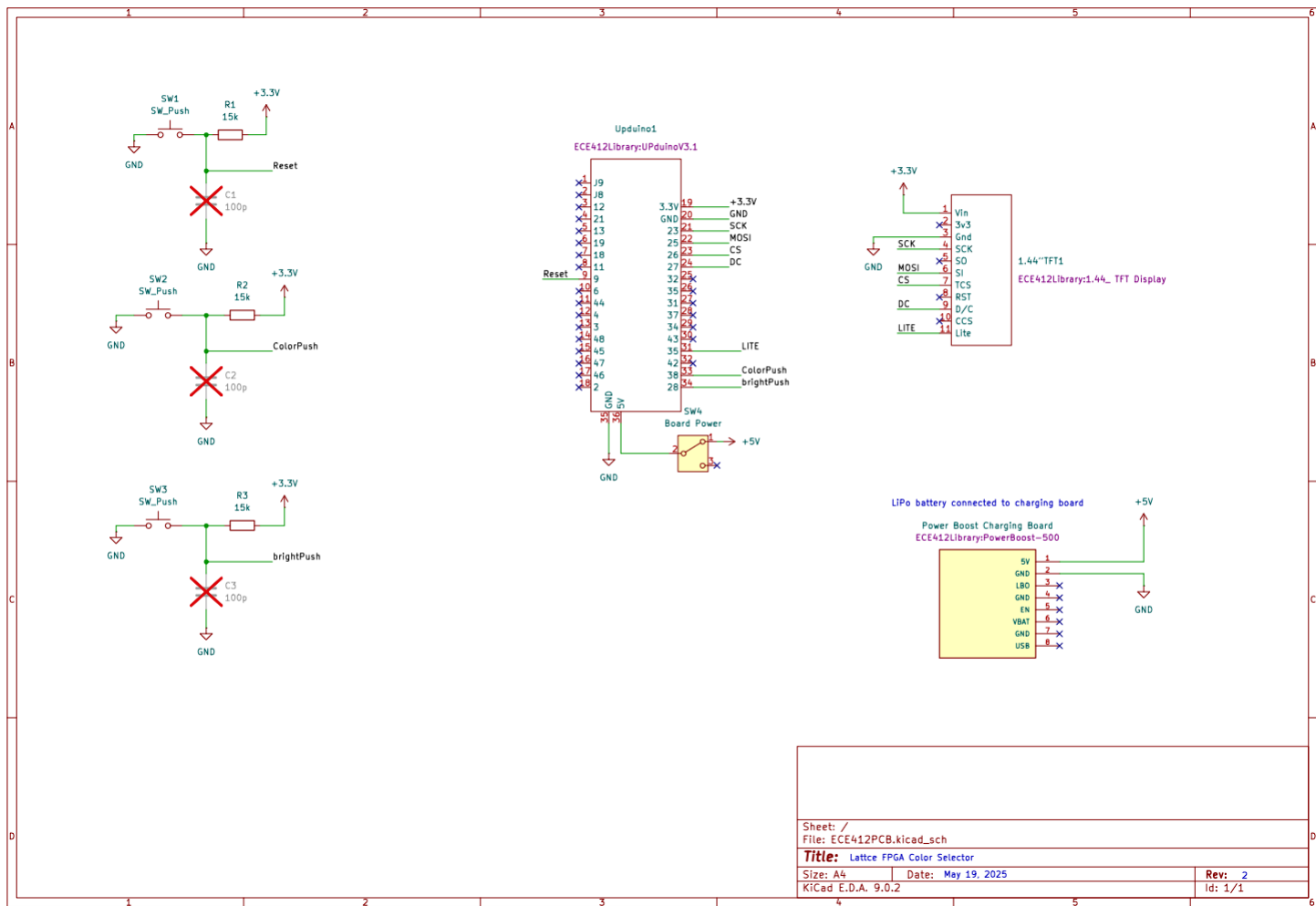
For RISC-V icicle core:

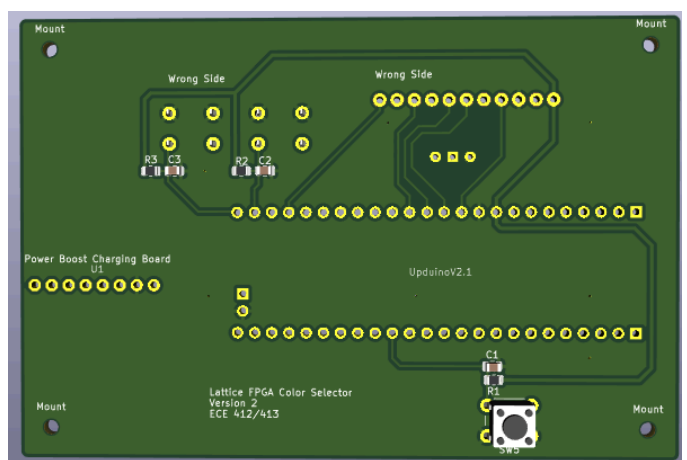


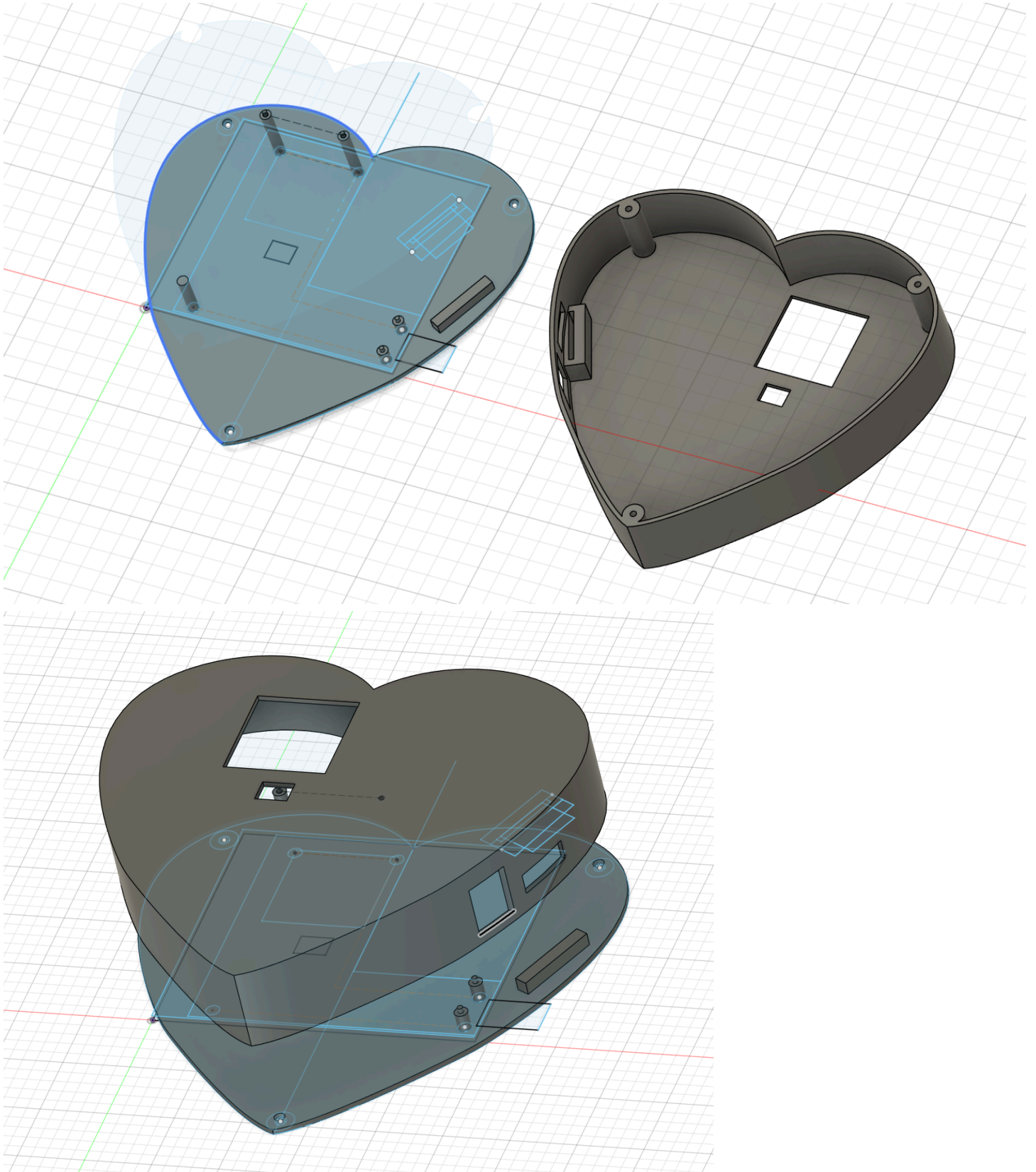
Physical Designer Output



Schematic







Toolchain

Toolchain Setup and Usage

Two distinct toolchains were utilized for this project, each with its own suite of tools for HDL synthesis, implementation, firmware compilation, and board programming. Detailed installation and setup guides for these toolchains are provided in [Appendix User's Manual](#).

Open-Source Toolchain

This toolchain centered around a collection of community-developed tools. Yosys was employed for HDL synthesis from Verilog to an optimized netlist, place-and-route were handled by Icestorm. The final bitstream generation for the iCE40UP5K FPGA was performed using NextPNR (specifically the icepack utility). Firmware for the "Icicle" RISC-V soft processor was developed in C and compiled using a standard RISC-V GCC toolchain. The resultant bitstream was programmed onto the UPduino board using a compatible utility icепrog. The team primarily operated these tools within Linux environments or the Windows Subsystem for Linux (WSL).

Commercial Toolchain

Lattice Radiant is the official integrated development environment (IDE) from Lattice Semiconductor. It provides a comprehensive suite of tools for HDL design entry, synthesis, place-and-route, timing analysis, bitstream generation, and device programming.

For the iCE40UP5K chip, the free license is enough, official site:

<https://www.latticesemi.com/latticeradiant>

Hardware

SPI module

SPI:

The initial SPI controller used a basic four-state machine. It accepted a start pulse, a single data byte, and a flag for command or data. At first, color values were split into two bytes by the software and sent over two back-to-back transactions. Getting data out of the SPI module took more work than expected. After programming the FPGA, there was no activity on the SPI output pins. This problem was traced to the clocking. The iCE40 HSOSC primitive would not drive logic unless the clock was defined in every constraint file: PDC, SDC, and LPF. When any file was missing the definition, the tools either failed timing or produced a bitstream with no toggling.

Once the clock constraints were fixed, SPI signals still did not show up on the expected pins. The Radiant documentation, the GitHub repo, and the Upduino silk screen all had conflicting pin assignments. Logic analyzer testing revealed that outputs were mapped to the wrong pins, not missing. Each signal had to be traced manually before the constraints could be updated. Only then did the SPI lines appear where they should.

As soon as communication worked, timing issues appeared. The chip select and SPI clock were sometimes not aligned with the data, and the LCD would display corrupted or missing pixels. It became clear that the chip select line had to be asserted and deasserted at very specific points. To address this, the SPI controller's state machine was restructured. The output behavior for each state was moved into

combinational logic, while the FSM itself was kept responsible only for transitioning from one state to the next. This change allowed for more precise control of the chip select, SPI clock, and data signals. After making this adjustment, the controller produced reliable output and the LCD began to display correctly.

Initially, the CPU handled all color splitting in software. This approach was slow. To improve speed, color handling was migrated into hardware using a small module. This module managed both bytes of each color transfer, removing the need for the CPU to intervene between bytes. This change reduced transfer latency and made color updates a single hardware-controlled operation, however the improvement was minimal. For future work, it might be beneficial to implement a multi_burst mode as the bytes are still being sent one at a time.

The LCD expected color data in big-endian order. Our design originally produced little-endian color data, which caused color errors on the display. Swapping the byte order in hardware resolved this mismatch.

For the open-source flow with Yosys, some inputs required pullup directives. If these were missing, signals could float, which led to unpredictable behavior that only showed up on hardware, not in simulation.

A register map was used in the final design. Each SPI function, such as data, control, status, DC line, color, and button, had a dedicated register. Address-based control was abandoned since it led to confusion and errors. A status register and handshake logic were added so the CPU could check when the SPI module was busy or ready, replacing the unreliable one-cycle done pulse.

Open Source RISC-V core

The computational core of the system was the "Icicle" 32-bit RISC-V soft processor, an open-source design chosen for its compatibility with the open-source toolchain and its suitability for embedded control applications. The Icicle core, primarily described in Verilog, provided the instruction set architecture (ISA) and execution pipeline. The team successfully synthesized and deployed this core on the UPduino v2.1 board using the Yosys-based open-source flow.

A significant portion of the software design effort involved integrating custom peripherals, such as the SPI controller and the button interface, with the Icicle core. This was achieved by using the core's memory interface module, `rv32_mem.sv`, to define specific address ranges for memory-mapped registers. These registers allowed the C firmware running on the RISC-V CPU to control and exchange data with the hardware peripherals. Writing to a particular address loads data into the SPI transmit buffer, while reading from another address might fetch the current status of the buttons.

Ensuring proper initialization and stable operation of the Icicle core was critical, particularly when attempting to port the design to the Lattice Radiant toolchain, which handled initial block conditions differently than Yosys. To address this, a Single Stage Reset (SSR) mechanism was implemented. This involved forcing the Program Counter (PC) to a known starting address upon reset, ensuring a consistent startup state. The standard RISC-V startup assembly files, such as `start.s` (producing `start.o`), were also essential components for initializing the C runtime environment for the firmware.

While the Icicle core functioned well under the Yosys toolchain after initial clocking and pinout issues were resolved, porting it to Lattice Radiant proved exceptionally challenging. This involved attempts to modify the Icicle source or its instantiation, address PLL and timing issues specific to Radiant's handling of the design, and debug memory loading and program execution anomalies within the Radiant environment. These efforts highlighted the significant differences in how open-source soft cores interact

with proprietary FPGA synthesis and implementation tools compared to open-source flows.

Buttons

The push buttons provide digital inputs to the FPGA. This necessitated a hardware redesign of the button interface circuitry to ensure stable and predictable input signals, preventing floating inputs that could lead to non-deterministic behavior.

The state of the buttons is accessed by the Icicle RISC-V core through a memory-mapped register, SPI_BUTTON_ADDR (address 0x00041014). A '1' in the least significant bit of SPI_BUTTON_ADDR indicates a button press.

The physical debounce of the button was moved to the HDL due to lack of time remaining for PCB revisions. The functionality of the physical debounce was verified on a prototype board by removing the capacitor's direct ground connection.

Color Selection: One button is dedicated to cycling through a predefined set of colors. The current color value is stored in a memory-mapped register, SPI_COLOR_ADDR (address 0x00041010). When the color button is pressed, the register is updated with the next color in the sequence through the HDL.

Brightness Adjustment: The second button is responsible for adjusting screen brightness. The brightness of the screen is determined by a PWM signal that is generated in the HDL. Pressing the button changes the duty cycle of the PWM signal, reducing the brightness of the screen. It is designed to be able to cycle through eight brightness levels.

Firmware

Buttons and Screen control

User interaction with the system is managed through two physical push buttons, providing input for changing the displayed color and adjusting screen brightness. The screen control logic, executed as C firmware on the Icicle RISC-V core, processes these inputs and updates the ST7735R LCD via the SPI module.

Screen Control Logic

The C firmware running on the Icicle RISC-V core continuously monitors the SPI_BUTTON_ADDR for changes. Upon detecting a button press, the firmware initiates the corresponding screen control operation:

SPI Communication for Display Updates

All display updates, including color changes and brightness adjustments, are communicated to the Adafruit ST7735R 1.44-inch Color TFT LCD via the integrated SPI module. The firmware utilizes a set of low-level functions to interact with the display, abstracting the raw SPI transactions. These functions include:

- `spi_send(uint8_t byte, uint8_t dc_flag)`: This function is responsible for transmitting a single byte over the SPI bus, with `dc_flag` indicating whether the byte is a command (0) or data (1).

- `write_command(uint8_t cmd)`: Sends a command byte to the ST7735R display.
- `write_data(uint8_t data)`: Sends a data byte to the ST7735R display.
- `st7735_init()`: Initializes the ST7735R display by sending a predefined sequence of commands and data as specified by the Adafruit ST7735 library initialization routine. This sequence configures parameters such as frame rate control, power control, memory access control (orientation), and color mode.
- `st7735_set_addr_window(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1)`: Defines the rectangular area on the display where subsequent pixel data will be written. This involves sending CASET (Column Address Set) and RASET (Row Address Set) commands followed by the coordinate data.
- `st7735_fill_screen(uint16_t color)`: Fills the entire display with a specified 16-bit RGB565 color. This function internally calls `set_addr_window` for the whole screen and then sends the color data for each pixel.
- `st7735_draw_pixel(uint16_t x, uint16_t y, uint16_t color)`: Draws a single pixel at a given (x,y) coordinate with the specified 16-bit color.

The SPI module also includes a status register, `SPI_STATUS_REG`, which provides feedback on the transfer status, including a busy flag (`SPI_STATUS_BUSY`) and a done flag (`SPI_STATUS_DONE`). The `spi_wait_done()` function polls this register to ensure that a previous SPI transfer is complete before initiating a new one.

For every detailed function and program details, see [Developer's manual](#).

Test Plan

The comprehensive test plan for this project is detailed in Appendix: [Test Plan](#).

This plan outlines a series of tests to verify the functionality, integration, and performance of the FPGA-based SPI LCD display system. It covers module-level verification, system integration testing, and validation against the defined project requirements, including control of the LCD screen via buttons, display of colors, and overall system stability. The test methodology ensures that both hardware and software components are tested.

Results

This project aimed to develop an FPGA-based system to control an SPI LCD screen with user button interaction, housed in a custom enclosure and powered by a battery, while also comparing the efficacy of open-source versus proprietary (Lattice Radiant) FPGA development toolchains. The project culminated in a tangible, functional final product that successfully met several key design and operational objectives, though challenges were encountered with one of the targeted toolchain implementations.

Deliverables Achieved:

- **Functional End Product:** A fully operational prototype was successfully developed and demonstrated. This system utilizes an FPGA (Lattice iCE40UP5K on an UPduino board) to drive an SPI-based LCD screen, responding to user inputs from two physical buttons to change displayed colors and adjust screen brightness.

- **Custom Enclosure:** A 3D-printed enclosure was designed and fabricated, housing all electronic components and providing a user interface with cutouts for the screen and buttons.
- **Battery Operation:** The system was successfully powered by a custom battery solution, incorporating a 3.7V LiPo rechargeable battery and a power boost converter to provide the necessary 5V supply.
- **Open-Source Toolchain Implementation:** The entire functional system, including the Icicle RISC-V soft processor, custom SPI controller, button interface logic, and hardware accelerators for color and brightness, was successfully implemented and validated using the open-source toolchain (Yosys, Icestorm, NextPNR).
- **RTL Source Code and Firmware:** Verilog HDL for the FPGA design (including the modified Icicle core, SPI controller, and hardware effect modules) and C firmware for the RISC-V processor (handling LCD initialization, button polling and text display) were developed and are available in the project repository.
- **Documentation:** Key documentation, including a user guide, progress reports, and contributions to this final report, were completed.
- **PCB Design:** A custom Printed Circuit Board was designed to integrate the system components, reaching a stage ready for faculty review.

Deliverables Not Fully Achieved or Altered:

- **Lattice Radiant Toolchain Implementation:** Despite significant and sustained effort throughout the project duration, a fully functional implementation of the complete system (Icicle RISC-V core with all peripherals) using the Lattice Radiant toolchain was not achieved. While progress was made in resolving specific issues like PLL routing and reducing synthesis warnings, core compatibility and program execution problems with the open-source RISC-V core within Radiant persisted.
- **Direct Toolchain Comparison (Power & Synthesis Quality):** Consequently, the original objective of a detailed, quantitative comparison between the open-source toolchain and Lattice Radiant, particularly concerning power consumption profiles and nuanced synthesis quality metrics, could not be fully realized as initially envisioned. The comparison was thus limited to ease of use and workflow experiences.
- **Low-Power Design Techniques (Explicit Analysis):** While the chosen FPGA (iCE40UP5K) is inherently low-power, the planned detailed exploration and profiling of multiple specific low-power design techniques across both toolchains were deprioritized due to the overriding challenges with the Radiant toolchain implementation.

Tests and Measurements Performed:

- **Functional Testing (Open-Source Flow):**
 - **Power Supply System:** The battery and boost converter system was tested, confirming its ability to power the UPduino board and peripherals.
 - **Button Control:** The two user buttons were tested for correct operation, verifying that they successfully triggered color changes and brightness adjustments on the LCD as intended. This included debugging signal integrity and responsiveness.
 - **Screen Display:** The SPI LCD was tested to confirm correct initialization, display of various colors. Text display functionality was also developed and available.
 - **SPI Communication:** SPI signal integrity and timing were validated using a logic analyzer during the development and debugging of the SPI module to ensure reliable data transfer to the LCD.
- **Radiant Toolchain (Partial Tests):**

- Attempts were made to synthesize and program the FPGA with Radiant, with some success in instantiating modules and resolving PLL issues.
- Memory loading for the C program in Radiant was confirmed via synthesis reports during debugging sessions.
- However, full end-to-end functional testing of the complete system on Radiant was not possible due to unresolved program execution issues.

Meeting Requirements:

The project successfully met the core functional requirements as demonstrated by the Yosys-based implementation:

- **Must control an SPI-based LCD screen to adjust brightness and color via two buttons:** Achieved.
- **Must use Lattice Radiant FPGA toolchain to develop FPGA bitstream:** Partially achieved in terms of effort and some module-level synthesis, but a fully functional bitstream for the complete system was not realized.
- **Must use at least 3 kinds of low-power design techniques:** This requirement was not explicitly tracked with quantitative analysis due to the shift in project focus. The use of the low-power iCE40UP5K FPGA and general good design practices were inherent.
- **Must have a working final product:** Achieved with the Yosys toolchain.
- **Must be in an enclosure:** Achieved.
- **Should compare the ease of use and efficiency between the open-source toolchain and the Lattice Radiant toolchain:** Achieved qualitatively, with significant insights gained into the challenges of using Radiant with a complex open-source soft core.
- **Should collect and visualize power consumption data and compare results:** Not achieved due to Radiant implementation issues and subsequent reprioritization.
- **Should be encased in a 3D-printed enclosure:** Achieved.
- **Should be powered by a battery:** Achieved.
- **May be able to show meaningful text on the screen:** Achieved.

Overall, the project delivered a functional hardware product as specified, successfully leveraging the open-source toolchain. The primary deviation from the original plan was the inability to achieve an equivalent functional system using the Lattice Radiant toolchain with the chosen open-source RISC-V core, which subsequently impacted the scope of the comparative analysis. Test results for the Yosys-based system indicate that the power supply, button controls, and screen display functionalities met the project's operational requirements.

Post Mortem

What worked?

- **Open-Source Toolchain (Yosys Flow):** The team was ultimately successful in implementing the full system functionality using the open-source toolchain (Yosys, Icestorm, NextPNR). This allowed for the development of the Icicle RISC-V core, SPI communication, button control, and even

hardware-accelerated color and brightness effects. The availability of examples and a more transparent process eventually led to a working product.

- **Team Collaboration and Adaptability:** Despite significant roadblocks, particularly with the Radiant toolchain, the team adapted its strategy to prioritize a functional deliverable. Members took on different tasks, helped each other troubleshoot, and managed to integrate different hardware and software components. The implementation of GitHub guidelines also aimed to improve workflow.
- **Hardware Development (Enclosure, PCB Design, Power):** The team successfully designed and implemented a 3D-printed custom enclosure for the device. A custom PCB was designed to integrate the components, reaching a review-ready state. A functional battery power system was also implemented.

What was great?

- **Achieving a Functional End Product:** Delivering a working FPGA-controlled LCD system with button interaction and a custom enclosure, despite the toolchain challenges, was a nice accomplishment.
- **Learning Experience with FPGAs and RISC-V:** The project provided a deep, hands-on learning experience with FPGA design, Verilog, RISC-V architecture, embedded C programming, and the intricacies of hardware-software co-design.
- **Support from Advisors:** Guidance from the faculty advisor (Roy Kravitz) and industry sponsor (Rahul Koche) was valuable in navigating technical issues and project decisions.

What didn't work?

Lattice Radiant Toolchain with Open-Source IP:

The primary and most significant challenge was the inability to get the open-source Icicle RISC-V core and the complete system fully operational using the Lattice Radiant toolchain, despite extensive time and effort dedicated to troubleshooting by multiple team members. Issues ranged from initial module duplication errors and clock generation problems to PLL integration, timing closure, and final program execution. This directly impacted a core project goal: the detailed comparison between toolchains.

Many of the integration difficulties encountered in Radiant stemmed not from flaws in the toolchain itself, but from fundamental differences in coding conventions and IP design expectations. For instance, Radiant's synthesis flow handled initial blocks and inferred memory constructs differently than the Yosys-based flow, leading to unexpected errors and runtime issues when instantiating the Icicle core. These challenges highlight the importance of aligning HDL style and module structure with toolchain assumptions, particularly when using third-party open-source IPs within a proprietary environment.

While Radiant provided valuable insights into resource usage and timing behavior, including identification of a critical -28ns setup slack between execute and fetch modules, these analytical strengths could not be fully utilized in the absence of a stable system build. Our team lacked direct access to Radiant-specific support or internal documentation for resolving toolchain-specific edge cases, which likely extended the debugging timeline. In hindsight, having a dedicated technical contact within Lattice or targeted training on integrating external IP would have significantly reduced integration friction and improved our ability to evaluate Radiant's full capabilities in a fair and complete manner.

- **Tool Usability and Documentation (Especially Radiant):**

Working with the Lattice Radiant toolchain introduced challenges not necessarily only due to a lack of documentation, but primarily due to the enforcement of more rigorous design standards compared to what our team encountered in the open-source Yosys flow. The Icicle RISC-V core and surrounding modules were originally developed and validated using open-source tools that allow certain coding constructs, such as C-style initializations and loosely constrained memory definitions, that Radiant correctly flagged or rejected as non-synthesizable under industry-grade design rules. These stricter checks reflect Radiant's professional-level expectations, but also meant that some design elements which passed in Yosys required significant rework or refactoring to be acceptable in Radiant.

This discrepancy contributed to a steeper learning curve, particularly for team members new to hardware design, who were simultaneously learning Verilog, synthesis flows, and platform constraints. While Radiant does offer comprehensive documentation and training resources, the issues we encountered were often subtle and specific to our integration of open-source IP not originally written with Radiant's synthesis requirements in mind. In hindsight, the project could have benefitted from having direct access to Lattice support or an experienced user to help bridge these design differences and guide appropriate remediation strategies early in the process. Such support would likely have allowed us to more efficiently adapt our design and better leverage Radiant's strengths in timing analysis and synthesis optimization.

- **FPGA Development Learning Curve:** The steep learning curve associated with FPGA development, toolchains, and debugging hardware/software interactions was challenging, especially for team members newer to this domain. Unlike some software development fields, the FPGA domain appears to have fewer beginner-friendly, comprehensive tutorials that cover the entire flow from design to hardware with diverse IP.
- **Initial Pinout Discrepancies:** Early in the project, time was lost due to conflicting pin assignments for the UPduino board across different documentation sources, including the board's silkscreen, online references, and various constraint file templates, requiring manual tracing.

What would you do differently?

- **Earlier Risk Assessment of Toolchain/IP Compatibility:** Perhaps a more in-depth, upfront investigation or a smaller-scale proof-of-concept specifically testing the chosen open-source RISC-V core's compatibility with Radiant could have highlighted the potential integration risks much earlier.
- **Allocate More Time for Radiant-Specific Learning/Debugging:** Given the difficulties, dedicating even more scheduled time specifically for understanding Radiant's nuances with custom IP might have been beneficial, though this would have been hard to predict.
- **Consider Simpler Core or Lattice-Provided IP for Radiant Path:** If the primary goal was a toolchain feature comparison (rather than specifically using Icicle on Radiant), evaluating Radiant with a simpler, perhaps Lattice-provided, RISC-V core or even a non-CPU-based design might have yielded a more direct comparison of the toolchain's capabilities for synthesis, place-and-route, and power analysis.
- **More Structured Approach to Pin Verification:** Implement a process to rigorously verify all critical pin assignments against multiple sources (datasheets, schematics, board silkscreen) at the very beginning of hardware interfacing.

What do you wish you would have known back in December?

- **The Extent of FPGA IP Incompatibility Between Toolchains:** We wish we had a clearer understanding of the practical challenges and lower-than-expected "plug-and-play" compatibility when migrating an open-source RISC-V soft core, like Icicle, from an open-source synthesis flow to a proprietary toolchain like Lattice Radiant. Vendor toolchains are optimized for, and work most seamlessly with, their own vendor-specific IP libraries. The expectation that a generic RISC-V core would run smoothly across different toolchains with minimal modification proved overly optimistic.
- **Radiant's Documentation and Community Support Limitations for This Use Case:** A foreknowledge of the limited availability of accessible, in-depth documentation and community troubleshooting resources for using Lattice Radiant with complex, non-Lattice open-source IP would have helped set more realistic timelines and expectations for that part of the project.
- **The True Difficulty and Steepness of the Learning Curve for Unfamiliar FPGA Tools:** While some FPGA experience was present, the true difficulty and time investment required to become proficient with an unfamiliar proprietary toolchain (Radiant) to the point of debugging complex integration issues was underestimated. The FPGA domain, compared to many software fields, seems to have a higher barrier to entry for new users, especially regarding toolchain intricacies and practical debugging on hardware. The general "unfriendliness" of some FPGA tools to newcomers was a significant hurdle.

FPGA Resource Utilization Table

Module	LUTs Used	Other* (Regs)	EBR
Icicle (RV32)	3280	1121	20
Bus Arbiter	120	1	0
Screen & Buttons	11	11	0
SPI	102	52	4
UART	125	83	0
FPGA Utilization: 41.06%			

Future Work

This project established a functional baseline for an FPGA-controlled display system. Future efforts could significantly expand upon this foundation, focusing on completing original objectives and adding new capabilities.

A primary goal would be the **full resolution of the Lattice Radiant toolchain implementation issues** encountered with the open-source Icicle RISC-V core. Achieving a stable Radiant build would enable the

initially planned comprehensive **quantitative comparison with the open-source Yosys flow**, particularly regarding power consumption, resource utilization, and synthesis quality. For a subsequent team, or with extended time, work could focus on:

- **Deepening Radiant Toolchain Expertise and IP Integration:** This could involve mastering Radiant for use with open-source RISC-V cores or, alternatively, exploring Lattice-provided IP cores and standard bus architectures like Wishbone to better leverage the proprietary tool environment. Investigating and implementing additional communication interfaces such as UART would also be beneficial.
- **Expanding System Functionality and Applications:** The system could be enhanced to perform more practical tasks by displaying meaningful text (e.g., sensor data, debug messages), which was an original project phase. Further screen enhancements could include image display capabilities, building upon the existing text display C functions.

Project Resources

Collaboration sites and repositories

Main Github: <https://github.com/Riley-Cox/ECE412-LatticeFPGA>

Reference sites:

Upduino 2.1: <https://github.com/tinyvision-ai-inc/UPduino-v2.1>

Upduino 3.1: <https://github.com/tinyvision-ai-inc/UPduino-v3.0>

32-bit RISC-V system icicle: <https://github.com/grahamedgecombe/icicle/tree/v1>

Adafruit-ST7735-Library: <https://github.com/adafruit/Adafruit-ST7735-Library>

Adafruit-GFX-Library (for text display): <https://github.com/adafruit/Adafruit-GFX-Library>

Yosys Open SYnthesis Suite: <https://github.com/YosysHQ/yosys>

Connect USB in WSL: <https://learn.microsoft.com/en-us/windows/wsl/connect-usb>

PowerBoost 500 Charger - Rechargeable 5V Lipo USB Boost @ 500mA+:

<https://www.adafruit.com/product/1944> (\$14.95)

Adafruit 1.44" Color TFT LCD Display with MicroSD Card breakout - ST7735R:

<https://www.adafruit.com/product/2088> (\$14.95)

Lithium Ion Polymer Battery - 3.7v 1200mAh: <https://www.adafruit.com/product/258> (\$9.95)

Tools with version

For the detailed list, see [Tooling](#).

Detailed setup instructions are in the project [User's Manual](#).

Other resources

File Storage and Version Control: GitHub was the primary platform for all project files, including code, documentation, and tracking issues.

Budget

Key components with known or indicative costs include:

Adafruit 1.44" Color TFT LCD Display with MicroSD Card breakout - ST7735R: \$14.95

Adafruit PowerBoost 500 Charger - Rechargeable 5V Lipo USB Boost @ 500mA+: \$14.95

UPduino Board V3.1 available: \$33

Lithium Ion Polymer Battery - 3.7v 1200mAh:\$9.95

Total **\$72.85**, in this project, the board and display are provided by sponsors.

Other components: PCB printing cost, 3D printing cost, and push buttons, power switch

Appendices

Tooling

1. Hardware Tools

- **FPGA Development Board:**
 - **UPduino v2.1:** This was the primary board used for development and testing of the Icicle RISC-V core and the overall system with the open-source toolchain. Hardware details and schematics are available from the TinyVision AI GitHub.
- **Display:**
 - **Adafruit ST7735R Color TFT LCD:** An SPI-based LCD screen (likely 1.44" or 1.8" version) was used for visual output.
- **User Input:**
 - **Tactile Push Buttons (x2):** Used for user interaction (color change, brightness control).
 - **Power Switch (x1):** For turning the device on/off.
- **Power System:**
 - **3.3V LiPo Rechargeable Battery:** Provided portable power. Specific model/capacity not detailed.
 - **Power Boost Converter Board:** An Adafruit PowerBoost 500 Charger or similar circuit was used to step up the 3.3V from the LiPo battery to 5V for the UPduino and other components.
- **Prototyping & Assembly:**
 - **Protoboard (Breadboard/Perfboard):** Used for initial circuit assembly and testing.
 - **Soldering Iron and related supplies:** For assembling the protoboard and PCB components.
- **Test and Measurement Equipment:**
 - **Logic Analyzer:** Essential for debugging digital signals, particularly SPI communication.
 - **Oscilloscope:** Used for observing signal integrity and timing, especially during clock issue troubleshooting.
- **Physical Construction:**

- **3D Printer:** Utilized to fabricate the custom project enclosure. The specific model of the printer is not detailed in the project documents.
-

2. Software Tools

The software toolchain was a mix of open-source solutions and proprietary vendor software.

- **A. Operating System and Development Environment:**

- **Host OS:** Windows (presumed, as WSL was used).
- **Windows Subsystem for Linux (WSL):** The primary environment for running the open-source FPGA toolchain and RISC-V GCC toolchain.
 - **Linux Distribution:** Debian was specifically used for this project within WSL.
 - **Windows Terminal:** Recommended application for accessing WSL.
- **USBIPD-WIN (for WSL USB Passthrough):**
 - **Purpose:** To connect the UPduino board via USB to the WSL environment for programming.
 - **Source/Installation:** `winget install --interactive --exact dorssel.usbipd-win`.
 - **Configuration:** Detailed steps for binding and attaching the USB device are provided in the project [User's Manual](#).. Reference also <https://learn.microsoft.com/en-us/windows/wsl/connect-usb> .

- **B. RISC-V GNU Toolchain (for C Firmware Development):**

- **Purpose:** Compiling C code for the Icicle RISC-V soft processor.
- **Source:** Cloned from the official repository: <https://github.com/riscv/riscv-gnu-toolchain> .
- **Version:** Dependent on the Git repository's state at the time of cloning. No specific version tag was documented as used.
- **Dependencies:** A comprehensive list of dependencies for various Linux distributions (Ubuntu/Debian, Fedora, Arch) is provided in the project [User's Manual](#).
- **Installation & Configuration:** The README.md details the steps involving setting the PATH environment variable, configuring with `./configure --prefix=<writable directory>`, and compiling with `make`.

- **C. Open-Source FPGA Toolchain (Yosys Flow):**

- **Yosys:**
 - **Purpose:** Verilog HDL synthesis.
 - **Source:** <https://github.com/YosysHQ/yosys>. Installed via system package managers (e.g., `sudo apt-get install yosys` for Debian).
 - **Version:** Dependent on the distribution's package repository at the time of installation.
- **Icestorm (Project Icestorm):**
 - **Purpose:** Tools for Lattice iCE40 FPGAs, including place-and-route, bitstream packing (icepack), and analysis.
 - **Installation:** Installed via system package managers (e.g., `sudo apt-get install fpga-icestorm` for Debian).
 - **Version:** Dependent on the distribution's package repository.
- **NextPNR (specifically nextpnr-ice40):**
 - **Purpose:** FPGA place-and-route tool, an alternative or complement within the

- Icestorm ecosystem.
 - **Installation:** Installed via system package managers (e.g., `sudo apt-get install nextpnr-ice40-qt` for Debian).
 - **Version:** Dependent on the distribution's package repository.
- **iceprog:**
 - **Purpose:** Utility for programming the iCE40 FPGA on the UPduino board. Typically part of the Icestorm installation.
- **Project Build Configuration (Yosys Flow):**
 - Navigated to `ECE412-LatticeFPGA/V2.1/src/icicle`.
 - Commands: `make clean`, then `make BOARD=upduino`, then `sudo iceprog top.bin` to program.
 - Specific migrations for UPduino v2.1 (from v1) included modifications to `upduino-defines.sv`, `upduino.mk`, `upduino.pcf`, and `icicle.sv` as detailed in the project `README.md`.
- **D. Proprietary FPGA Toolchain:**
 - **Lattice Radiant Software:**
 - **Purpose:** Integrated Development Environment (IDE) from Lattice Semiconductor for FPGA design, synthesis, place-and-route, timing analysis, bitstream generation, and device programming.
 - Source: Official Lattice Semiconductor website (<https://www.latticesemi.com/latticeradiant>). A free license was sufficient for the iCE40UP5K chip
 - **Installation:** Standard graphical installer provided by Lattice.
 - **Configuration:** Involved creating a project, selecting the iCE40UP5K-SWG36-CXXXX device (exact part number may vary), adding Verilog source files, and managing constraint files (PDC, SDC, LPF). The team encountered significant challenges configuring Radiant for the open-source Icicle core, particularly with clocking, PLLs, and memory initialization.
- **E. Version Control:**
 - **Git:** Distributed version control system. The client version depends on the user's system.
 - **GitHub:** Web-based hosting service for Git repositories, used for project collaboration and code management.
- **F. Hardware Design Software:**
 - **KiCAD:** Used for the final Printed Circuit Board (PCB) design.
 - **3D Modeling Software (for Enclosure):** sharp3D
- **G. General Development & Documentation:**
 - **Text Editors/IDEs:** Standard developer tools for writing Verilog, C, Markdown, etc. (e.g., VS Code, Vim, Notepad++). Not explicitly specified.
 - **Microsoft Office Suite / LibreOffice:** Or similar, for creating presentation materials and formal report sections.

For a step to step install, please refer to the [User's Manual](#).

Developer's Manual

1. System Assembly

The hardware assembly involves connecting the core components: the UPduino v2.1 board (hosting the Lattice iCE40UP5K FPGA), the Adafruit ST7735R LCD, user input buttons, and the battery-powered supply.

Component Integration:

- **UPduino Board:** Serves as the central processing unit, hosting the FPGA.
- **Adafruit ST7735R LCD:** Connects to the FPGA via the Serial Peripheral Interface (SPI) bus. Ensure correct pin-to-pin mapping for SCK, MOSI, CS, and DC lines. Refer to the final pin assignments in the .lpf (or equivalent constraint) files.
- **Push Buttons:** Two push buttons are connected to designated General Purpose I/O (GPIO) pins on the FPGA. Due to the iCE40UP5K FPGA's lack of internal pull-down resistors, external pull-down resistors must be incorporated into the button interface circuitry to ensure stable and unambiguous input signals.
- **Power Supply:** The 3.7V LiPo rechargeable battery is connected to an Adafruit PowerBoost 500C converter (or equivalent), which steps up the voltage to a stable 5V. This 5V output then powers the UPduino board and the LCD module. A physical power switch is integrated into this power path to control system power.
- **Custom PCB:** If the custom PCB is fabricated, all components are soldered onto it, simplifying interconnections and improving system robustness. Prior to PCB fabrication, a functional protoboard assembly is highly recommended for validating all connections and module operations.
- **3D-Printed Enclosure:** The 3D-printed enclosure is designed to house the UPduino board, PCB, battery, and provide secure mounting for the LCD and accessible positions for the two user buttons and the power switch.

2. System Programming

Refer to the [User's Manual](#) for toolchain installation and usage.

2.1. FPGA Bitstream Generation (Hardware Synthesis and Implementation)

The custom hardware components, including the SPI controller and button interface, are integrated with the Icicle RISC-V core at the Register-Transfer Level (RTL), typically described in Verilog.

1. **Hardware Description Language (HDL) Source:** The core logic for the SPI module and button interface resides in .sv (SystemVerilog) files. Key aspects of the SPI module include:
 - A state machine for managing SPI transactions. The design evolved to a restructured state machine with combinational logic for output behavior, enabling precise control over chip select, SPI clock, and data signals to ensure proper LCD communication and prevent timing issues.
 - Hardware-based color handling module to manage both bytes of each color transfer, reducing CPU overhead.
 - Byte order swapping in hardware to ensure big-endian color data as required by the LCD.

- Implementation of a register map for SPI functions (data, control, status, DC line, color, button) and handshake logic (status register with busy/ready flags) for robust CPU-SPI communication.

2. Memory-Mapped I/O Integration: To allow the I/O to trigger a subroutine in C, memory mapped I/O was used. By polling a location in memory that is based on the state of an input (in this case, the button for color changing), we are afforded a responsive and simple method for accepting inputs to trigger the SPI bus to change the screen's color

- The memory location was mapped to be inside the same memory range as the rest of the SPI module's memory.
- Integrated the logic for processing button processes outside the processor hierarchy to prevent critical path lengthening.
- Added debouncing to button processes in the RTL code for simplified circuit implementation.
- Add latching to prevent a button press from creating more than one color transition with logic to clear latch based on existing state machine operation.
- The memory locations for the different I/O's can be found within a case statement in the icle.sv file. Further locations can be added by specifying the location in the case statement.

3. Constraint Files

- Provide the FPGA with instructions on what each pin's configuration is.
- Pullup mode used for input pins for active low button presses.
- Generates system clock using internal oscillator to determine frequency based on critical delay.
- The constraint file for the Yosys development is located in icle/boards/upduino.pcf. To set a pin as an input/output simply assign the signal name a pin on the UPduino board.
- Ex: set_io led[0] 12

FPGA Pin Assignment Table

Signal Name	Pin Number
flash_clk	15
flash_csn	16
flash_io0	17
flash_io1	14
leds[0]	35
leds[1]	10
leds[2]	19
leds[3]	43
leds[4]	34
leds[5]	37
leds[6]	31
leds[7]	32
uart_rx	15
uart_tx	14
spi_mosi	25
spi_clk	23

spi_cs_n	26
lcd_dc	27
brightPush (10K Pull-up)	28
colorPush (10K Pull-up)	38
screenPower	36
r (10K Pull-up)	9

2.2. C Firmware Compilation and Loading

The C language program (firmware) running on the Icicle RISC-V core is responsible for controlling the LCD display and interpreting button inputs.

1. Firmware Source Code: The primary C source files for screen driving are `SPI/code/st7735.c`, `SPI/code/st7735_hw.h`, and `SPI/code/test.c` (or `main.c` in the final application).
 - `st7735_hw.h` defines the memory-mapped register addresses (`SPI_DATA_ADDR`, `SPI_COLOR_ADDR`, `SPI_BUTTON_ADDR` etc.) that the C code uses to communicate with the hardware SPI controller and button input module.
 - `st7735.c` contains the low-level functions for initializing the ST7735R LCD, setting address windows, filling the screen, and drawing pixels, all utilizing the memory-mapped SPI registers.
 - The main application file (`test.c`) contains the high-level logic for reading button inputs, cycling through colors, adjusting brightness, and rendering content to the display. All actions are done by sending messages to the registers in the screen through SPI. For references of the details for controlling the Screen, refer to
 - Adafruit-ST7735-Library: <https://github.com/adafruit/Adafruit-ST7735-Library>
 - Adafruit-GFX-Library (for text display): <https://github.com/adafruit/Adafruit-GFX-Library>
2. Compilation: Use the standard RISC-V GCC toolchain to compile the C source files into an executable format. The Icicle core's startup assembly files (e.g., `start.s`, `start-ram.s`, `start-flash.s`) are essential for initializing the C runtime environment and setting the Program Counter (PC) to a known starting address upon reset (Single Stage Reset - SSR).
3. Loading Firmware: The compiled firmware is typically embedded within the FPGA bitstream or loaded separately if a bootloader is present. In this project, the C application is compiled and loaded into the target memory during the FPGA synthesis process.
4. Bitstream Programming: Use `icprog` (or a similar utility compatible with the UPduino board) to load the generated FPGA bitstream (containing both hardware configuration and embedded firmware) onto the UPduino board's iCE40UP5K FPGA.

3. Testing Subsystems

Details described in the [Test Plan](#).

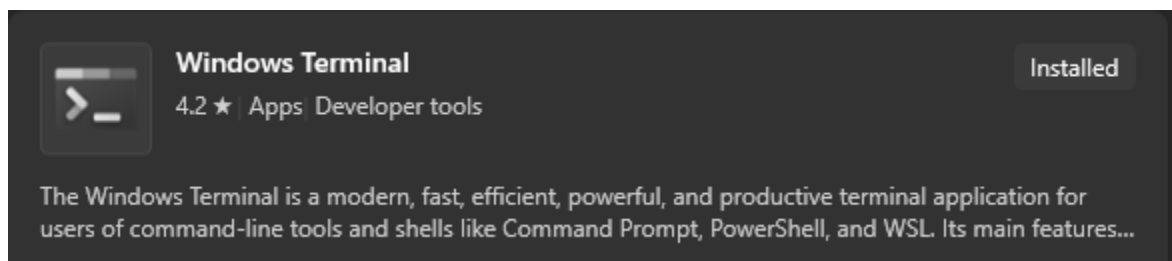
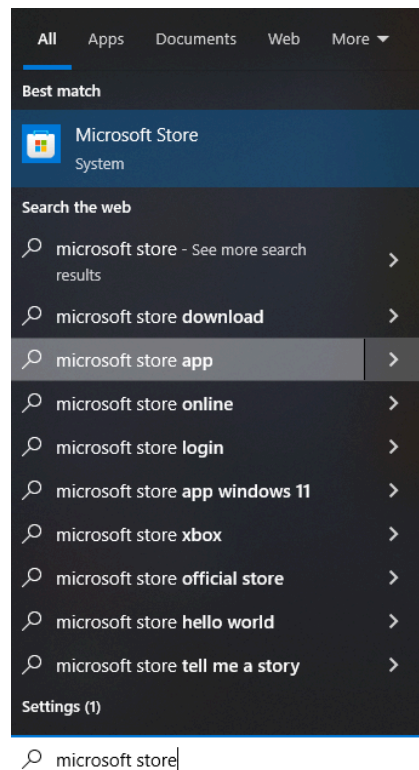
User's Manual

The original User's Manual is located at, <https://github.com/Riley-Cox/ECE412-LatticeFPGA/tree/main/README.md>, below is a Copy.

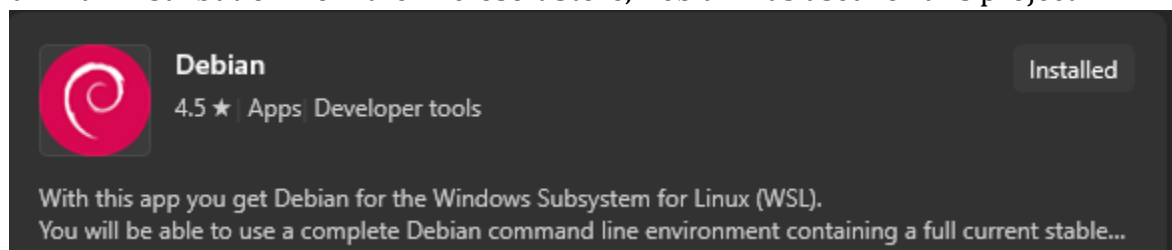
ECE412-LatticeFPGA

WSL Setup

- Install the Terminal app on the Microsoft Store



- Install a Linux Distribution from the Microsoft Store, Debian was used for this project



- Install usbipd through PowerShell by using the following command

Shell

```
winget install --interactive --exact dorssel.usbipd-win
```

- To ensure that it is installed run `usbipd list`, you should see your usb device with its BusID (the UPduino should show up as "serial converter")
- A USB bind is needed. Open Powershell as Admin, then run `usbipd bind --busid <busid>`
- To attach a USB to WSL run `usbipd attach --wsl --busid <busid>`

```
PS C:\Users\riley> usbipd list
Connected:
BUSID  VID:PID    DEVICE                                STATE
1-4    3434:0330  USB Input Device                     Not shared
1-5    046d:c088  USB Input Device                     Not shared
1-6    0951:16df  HyperX Quadcast, USB Input Device    Not shared

Persisted:
GUID                                DEVICE
9959b1d0-57b8-449d-a1a7-7d2184b2e03  USB Serial Converter
62d611c5-658e-4480-ac7b-682365b61c12  USB Serial Converter

PS C:\Users\riley> usbipd bind --busid 1-4
PS C:\Users\riley> usbipd attach --wsl --busid 1-4
usbipd: info: Using WSL distribution 'Debian' to attach; the device will be available in all WSL 2 distributions.
usbipd: info: Detected networking mode 'nat'.
usbipd: info: Using IP address 172.25.192.1 to reach the host.
PS C:\Users\riley>
```

- In WSL, the device should show up when `lsusb` is ran, if command is not found run `sudo apt-get install usbutils`

```
(venv) → ~ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
(venv) → ~
```

- To disconnect the device run `usbipd detach --busid <busid>`

Dependencies

Ubuntu/Debian

Shell

```
sudo apt-get install autoconf automake autotools-dev curl python3 python3-pip python3-tomli
libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool
patchutils bc zlib1g-dev libexpat-dev ninja-build git cmake libglib2.0-dev libslirp-dev xxd
make
```

Fedora

Shell

```
sudo yum install autoconf automake python3 libmpc-devel mpfr-devel gmp-devel gawk bison flex  
texinfo patchutils gcc gcc-c++ zlib-devel expat-devel libslirp-devel xxd make
```

Arch

Shell

```
sudo pacman -Syu autoconf automake curl python3 libmpc mpfr gmp gawk base-devel bison flex  
texinfo gperf libtool patchutils bc zlib expat libslirp xxd make
```

RISC-V GNU Toolchain

- From your home directory, clone the RISC-V GNU toolchain from the following repo

Shell

```
git clone https://github.com/riscv/riscv-gnu-toolchain
```

- From your home directory, open your shell rc file to edit (**vim .zshrc** or **vim .bashrc**) and add

Shell

```
export PATH="<writeable directory>:$PATH"
```

Note that the path you choose for the **<writeable directory>** should end with /bin

Ex: **export PATH="\$HOME/Documents/riscv-toolchain/bin:\$PATH"**

- Then run **source .zshrc** if using zsh or **source .bashrc** if using bash
- **cd** into the riscv-gnu-toolchain directory, run

Shell

```
./configure --prefix=<writeable directory>
```

Note that this **<writeable directory>** does not need to include /bin

Ex: **./configure --prefix=\$HOME/Documents/riscv-toolchain**

- then run **make**

Open Source Toolchain Setup

Ubuntu/Debian

Shell

```
sudo apt-get install yosys fpga-icestorm nextpnr-ice40-qt
```

Arch

Shell

```
yay -S yosys-git icestorm-git nextpnr-git
```

Fedora

Shell

```
sudo dnf install yosys icestorm nextpnr
```

Building on the UPduino board

- Clone this repo

Shell

```
git clone https://github.com/Riley-Cox/ECE412-LatticeFPGA.git
```

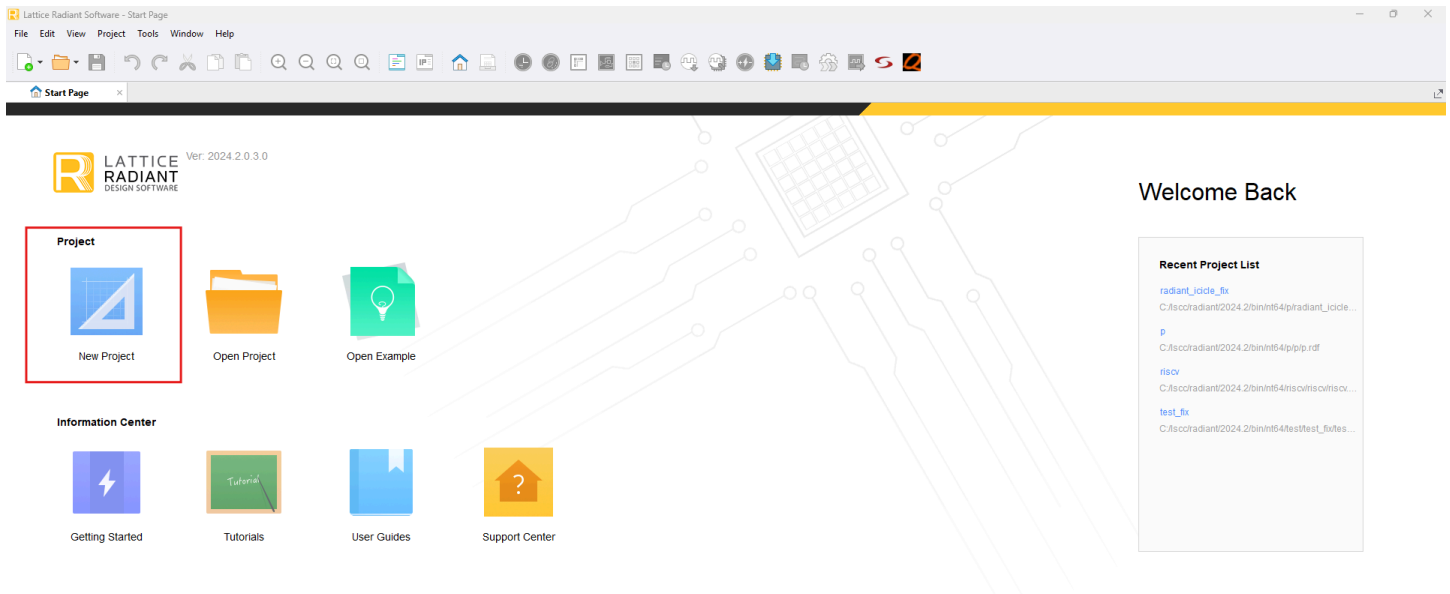
- Navigate to `ECE412-LatticeFPGA/V2.1/src/icicle`
- Run `make clean`
- Run `make BOARD=upduino`
- Run `sudo iceprog top.bin`

Migrating Icicle from UPduino V1 to V2.1

- Removed SPI_FLASH define from the upduino-defines.sv file located in `/icicle/boards`
- Changed PROGMEM value from flash to ram in the upduino.mk file located in `/icicle/boards`
- Changed UART RX/TX pins to 15 and 14 respectively in the upduino.pcf file located in `/icicle/boards`
- Created a flash_csn output in icicle.sv and assign it 1 (`assign flash_csn = 1`)

Setting-up the Project in Radiant

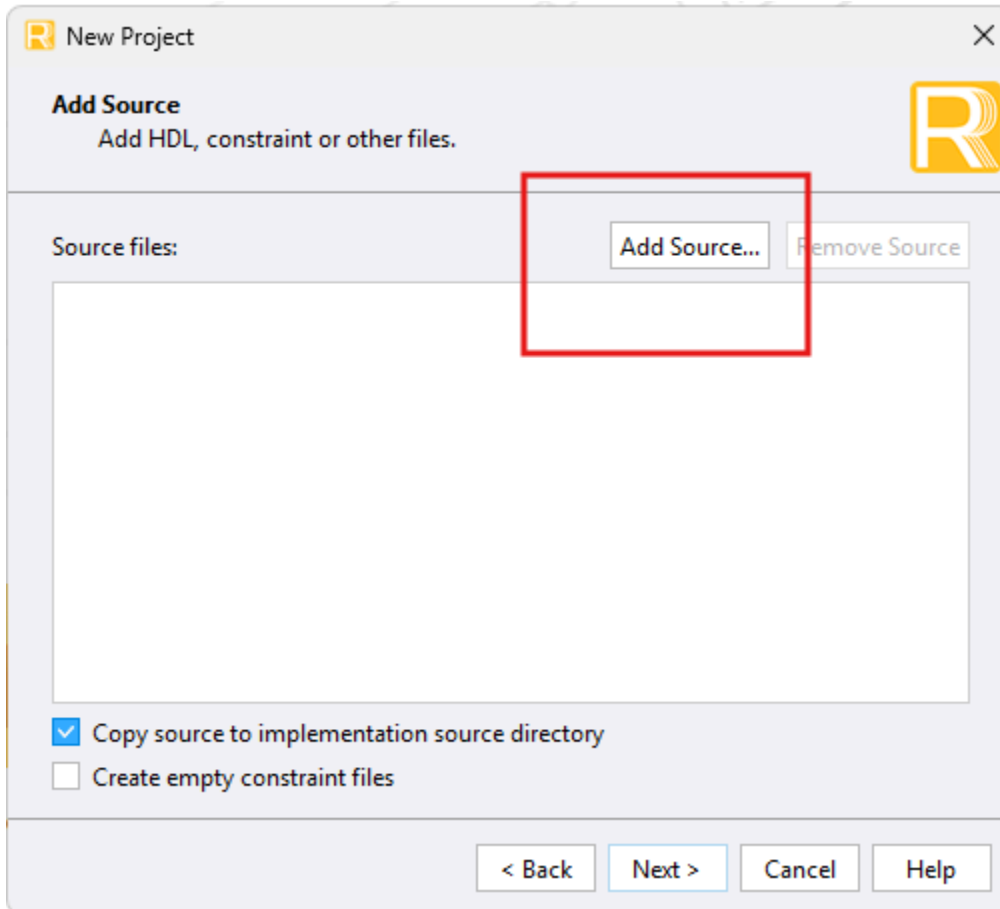
- First Download the FEAT/radiant_button branch from the Github Repository and unzip the folder
- Next create a new project in Radiant




LATTICE


- Next choose a name for the project


- Next add all the .sv files in (V2.1/src/icicle) from the unzipped folder except RVFI_wrapper.sv and Dummy_pll.sv



- Choose the device options depicted in the photo below

 New Project





Select Device

Specify a target device for the project.

Select Device:

Family:

iCE40UP (iCE40 UltraPlus)

Device:

iCE40UP3K

iCE40UP5K

Operating Condition:

Industrial

Performance Grade:

High-Performance_1.2V

Part Number:

iCE40UP5K-SG48I

Package:

SG48

Device Information:

Core Voltage:

1.20 V

LUTs:

5280

Registers:

5280

EBR Blocks:

30

DSPs:

8

PLLs:

1

DLLs:

0

PCSs:

0

PIO Cells:

56

PIO Pins:

39

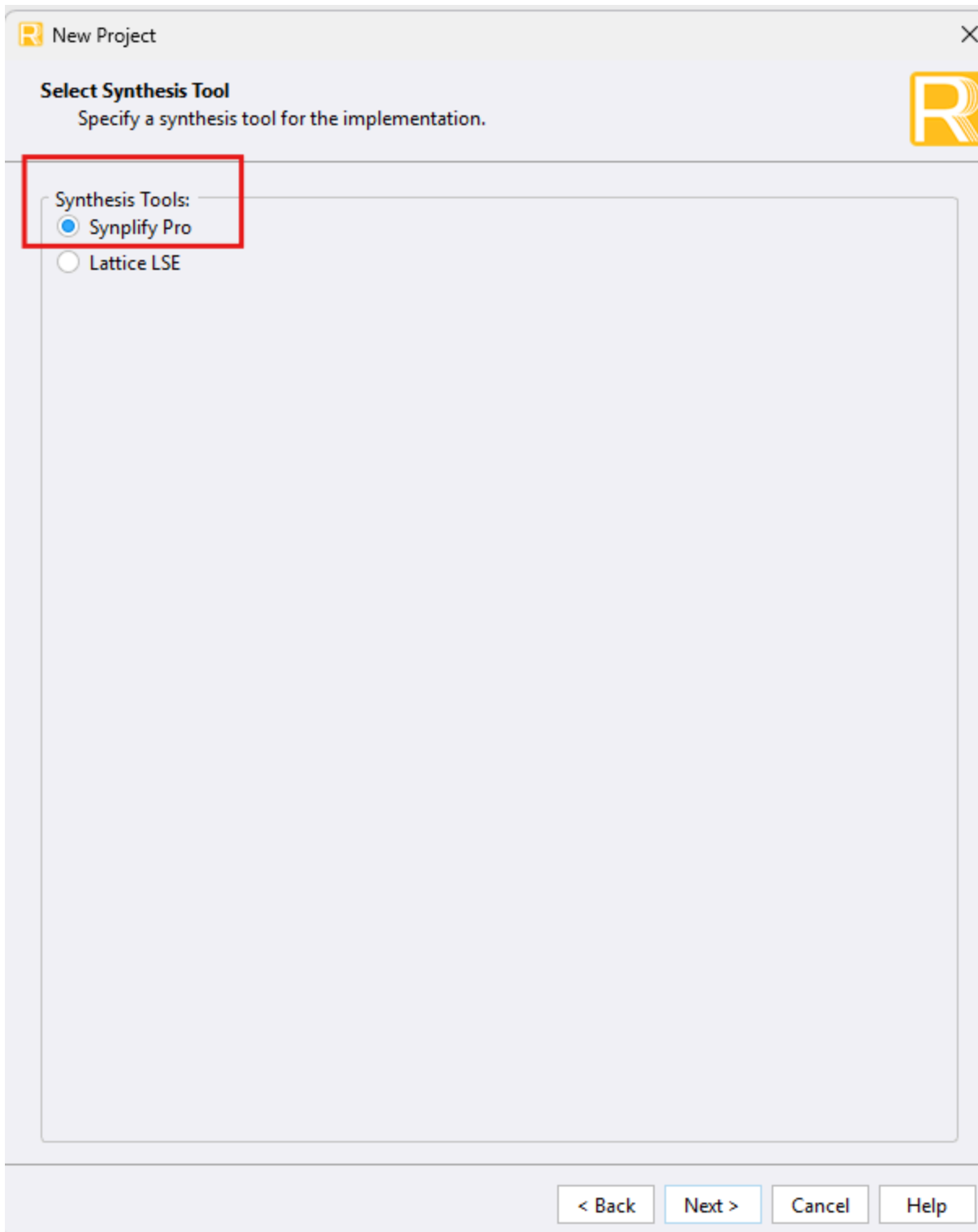
[Online Data Sheet for Device](#)

< Back

Next >

Cancel

Help



- After this complete the project setup
- In the new project window that opens, go to FILE —> ADD —> EXISTING FILE... in the toolbar
- In the pop-up window, add all the .ipx files in /V2.1/src/icicle
- Once done, right click on one of the .ipx files and click REGENERATE ALL IPs.....
- Once done, you can continue the synthesis toolchain

Test Plan

This document outlines the test procedures and expected outcomes for the FPGA-based SPI LCD display system. The primary objective of this test plan is to ensure that the final product meets all specified requirements, including functional correctness, user interaction, and system stability.

1. Test Environment Setup

- **Hardware:**
 - UPduino v2.1/v3.1 board with Lattice iCE40UP5K FPGA
 - Adafruit ST7735R 1.44-inch Color TFT LCD
 - Custom PCB
 - Two push buttons
 - 3.7V LiPo rechargeable battery
 - Adafruit PowerBoost 500C (or equivalent)
 - Logic Analyzer
 - Multimeter
- **Software/Toolchain (Open-Source Focus):**
 - Yosys (for HDL synthesis) - *Version to be specified in Tooling Appendix*
 - Icestorm / NextPNR (for place-and-route, bitstream generation) - *Version to be specified in Tooling Appendix*
 - RISC-V GCC Toolchain (for C firmware compilation) - *Version to be specified in Tooling Appendix*
 - iceprog (for bitstream programming) - *Version to be specified in Tooling Appendix*
 - Terminal emulator (e.g., PuTTY, minicom) for serial output.

2. Test Cases

The test cases are categorized by system functionality. Each test case includes a unique ID, description, preconditions, steps, expected results, and criteria for pass/fail.

2.1. Basic System Power-Up and Initialization Tests

Test ID	Description	Preconditions	Steps	Expected Results	Pass/Fail Criteria
TP-001	Power-up and basic FPGA configuration.	UPduino board powered via USB or battery.	1. Apply power to the UPduino board.	The FPGA successfully powers up. The ST7735R LCD shows initial signs of life (backlight on).	LCD backlight illuminates; no smoke or error indicators.
TP-002	RISC-V Core Boot and Firmware Loading.	TP-001 passed. FPGA programmed with bitstream containing Icicle core and SPI module.	1. Observe serial output (if any). 2. Observe LCD display.	The Icicle core executes the initial firmware. LCD transitions from initial state to a programmed display	LCD displays programmed output (e.g., initial solid color) without corruption.
TP-003	SPI Module Initialization.	TP-002 passed. Logic Analyzer connected to SPI lines (SCK, MOSI, CS, DC).	1. Power cycle or reset the system. 2. Observe SPI signals on logic analyzer during LCD initialization.	Correct SPI command sequences and data are observed for ST7735R initialization.	Logic analyzer captures correct SPI waveforms according to the test program.

2.2. Button Control Functionality Tests

Test ID	Description	Preconditions	Steps	Expected Results	Pass/Fail Criteria
TP-004	Color Cycle Button - Forward.	TP-003 passed. LCD displaying an initial color.	1. Press and release "Color" button once. 2. Repeat step 1 multiple times (e.g., 5 times).	The LCD displays the next predefined color in the sequence with each press. The sequence wraps around correctly.	LCD color changes consistently with each button press. Colors are distinct and match predefined sequences.
TP-005	Color Cycle Button - Wrap-around.	TP-004 passed. LCD displaying the last color in the sequence.	1. Press and release "Color" button once.	The LCD displays the first color in the predefined sequence.	The color correctly transitions from the last to the first color.
TP-006	Brightness Adjustment Button - Increase.	TP-003 passed. LCD displaying a stable image.	1. Press and release "Brightness" button once. 2. Repeat step 1 multiple times.	The LCD brightness increases incrementally with each press, up to a maximum level.	Observable increase in screen brightness with each button press until maximum is reached.
TP-007	Brightness Adjustment Button - Decrease.	TP-006 passed. LCD at or near maximum brightness.	1. Press and release "Brightness" button once. 2. Repeat step 1 multiple times.	The LCD brightness decreases incrementally with each press, down to a minimum level.	Observable decrease in screen brightness with each button press until minimum is reached.

TP-008	Button Debouncing/Reliability.	TP-003 passed.	1. Rapidly press and release either button multiple times. 2. Press and hold a button for an extended period.	The system responds reliably to button presses. Holding a button should only trigger a single event.	Button presses are registered accurately; no unexpected state changes.
--------	-----------------------------------	-------------------	---	---	--

2.3. Power System and Enclosure Tests

Test ID	Description	Preconditions	Steps	Expected Results	Pass/Fail Criteria
TP-009	Power Supply Voltage Boost Verification.	All components assembled but not yet powered. DMM connected to the output of the PowerBoost converter and to the UPduino board's 5V input.	<ol style="list-style-type: none"> 1. Connect the 3.7V LiPo battery to the PowerBoost converter input. 2. Turn on the main power switch. 3. Measure the voltage at the output of the PowerBoost converter. 4. Measure the voltage supplied to the UPduino board. 	The PowerBoost converter outputs a stable 5V ($\pm 0.2V$). The UPduino board receives a stable 5V ($\pm 0.2V$).	Measured voltages are within the specified tolerance of 5V.
TP-010	Main Power Switch Functionality.	TP-009 passed. System fully assembled with battery.	<ol style="list-style-type: none"> 1. Ensure the system is off. 2. Flip the power switch to the 'ON' position. 3. Observe system power-up indicators: LCD backlight on. 4. Flip the power switch to the 'OFF' position. 5. Observe system power-down. 	The system powers on reliably when the switch is set to 'ON'. The system powers off completely when the switch is set to 'OFF'.	System power state precisely matches the power switch position.
TP-011	Battery Powered Operation and Stability.	TP-010 passed. System fully assembled with battery.	<ol style="list-style-type: none"> 1. Disconnect any external USB power source. 2. Turn on the system using the power switch. 3. Operate the system (e.g., cycle colors, adjust brightness) for an extended period (e.g., 30 minutes). 	The system powers on and functions correctly using battery power. The battery provides stable power without noticeable performance degradation throughout the test duration.	System operates identically on battery as on external power; no unexpected resets or flickering.
TP-012	Battery Charging Functionality.	System powered by battery. Charger connected.	<ol style="list-style-type: none"> 1. Connect the USB charging cable to the PowerBoost converter. 2. Observe charging indicator (PowerBoost board led) 	The battery charges correctly, and any charging indicator (LED) behaves as expected, showing an active charging state.	Battery voltage increases over time when charging; charging indicator functions as designed.

TP-013	Enclosure Physical Integration and Ergonomics.	All components assembled inside the 3D-printed enclosure.	1. Visually inspect the enclosure for fit and finish. 2. Test tactile feel and response of integrated buttons and power switch. 3. Ensure all ports (charging, programming) are accessible.	All components fit securely within the enclosure without stress or misalignment. Buttons and power switch are easily actuated and provide satisfactory tactile feedback.	Enclosure provides robust protection; all user interfaces are functional and accessible.
--------	---	--	---	---	--

3. Test Procedures

- **Execution:** Tests will be performed manually by the team members. Each test will be documented with the date, tester's name, observed results, and a pass/fail status.
- **Defect Reporting:** Any discrepancies between expected and observed results will be logged as defects, including a detailed description, steps to reproduce, and severity.
- **Regression Testing:** After bug fixes or significant code changes, a subset of critical tests will be re-executed to ensure no new regressions have been introduced.

4. Acceptance Criteria

The project will be considered complete and successful if:

- All "Must" requirements (as defined in the project proposal) are fully met and verified by passing the corresponding test cases.
- A significant majority of "Should" requirements are met.
- The system demonstrates stable and reliable operation during extended use.
- All critical defects are resolved.

Toolchain Comparison: Radiant vs. Yosys

This project explored a dual-toolchain development strategy, evaluating both the Lattice Radiant proprietary toolchain and the open-source Yosys/NextPNR suite for FPGA synthesis and implementation. While both tools ultimately serve the same end goal, compiling RTL into a working system, they differ significantly in expectations, enforcement rigor, and workflow structure.

Radiant, designed for professional and industrial FPGA development, enforces stricter coding standards and provides detailed timing analysis, resource reports, and robust constraint handling. During our attempt to integrate the Icicle RISC-V core into Radiant, we encountered synthesis failures stemming from constructs that were permissible under Yosys but flagged or disallowed in Radiant. These included C-style initializations in SystemVerilog and inferred memory declarations without explicit constraints. Radiant's behavior in these cases is consistent with industry-grade synthesis tools, which emphasize hardware determinism and formal correctness. In contrast, Yosys, while less strict, proved more forgiving with an open-source IP. Its relaxed acceptance of informal initialization practices and lighter checks enabled the Icicle core, which frequently uses HDL coding practices that are typically prohibited, to synthesize and execute early in the project. This flexibility was helpful during rapid prototyping and testing, especially as the team iterated through multiple integration and debugging cycles. However, the trade-off was reduced insight into timing closure and more limited resource optimization feedback compared to Radiant.

One of the more challenging aspects of using Radiant was diagnosing and resolving synthesis warnings, which often required a more advanced understanding of RTL design principles and hardware mapping. Unlike Yosys, which tends to accept loosely defined constructs and defer problems until runtime, Radiant performs stricter static analysis and halts synthesis when critical implementation assumptions are violated. A recurring example involved partial routing issues, where signals, especially those related to control lines like `lc_dc`, fetch control signals, and other signals, were not fully driven across all logic paths. Radiant flagged these conditions with precise but complex warnings, often referring to internal nets or inferred logic blocks without clear reference to the user-written RTL. Debugging these required tracing signal propagation, reviewing how conditional assignments were encoded, and understanding how synthesis tools interpret uninitialized or undriven wires. In several cases, problems traced back to modules that synthesized without complaint in Yosys, but failed in Radiant due to stricter enforcement of deterministic signal flow. While this initially slowed development, it ultimately improved the reliability of our design and reinforced best practices in hardware-safe Verilog coding. Sometimes these partial routing issues were due to a missing clock constraint causing the synthesizer to fail to propagate the correct clock forward, causing clock domain issues that didn't actually exist in design, or a lack of a default cause in some case statements or don't cares.

From a usability perspective, Yosys offered faster compile-test cycles and easier debugging when working with community-developed cores and minimal constraints. Radiant required deeper familiarity with IP constraints, timing models, and synthesis policies, but once understood, it gave more detailed visibility into design behavior, including critical path analysis (e.g., -28ns slack between execute and fetch stages, information pertaining to clock domain issues, better visualizations of device netlist and lut utilization, more comprehensive reports etc.).

While the final system was successfully implemented using the Yosys toolchain, this was not due to shortcomings in Radiant, but rather to compatibility misalignments between our open-source codebase and Radiant's synthesis model. In retrospect, adapting the IP to be Radiant-compliant or starting with Lattice-native cores might have allowed for a more direct and productive evaluation.

Key Takeaway:

Radiant is better suited for structured, high-assurance design environments and offers industry-aligned analysis and enforcement. Yosys is more accessible for exploratory prototyping and open-source flexibility. Each toolchain has strengths that are best leveraged with a development flow tailored to its expectations.