FDE MIDTERM REPORT

Name Display, UFDE & Rabbit, SMIMS Engine, FDE-CLI, I2C

Jimmy Wang (王俊崴) 22302016002

AGENDA

1. Experiment 1: Name Display
2. Experiment 2: Introduction
3. SMIMS VLFD Driver in Rust
4. SMIMS Engine
5. FDE-CLI
6. I2C

NAME DISPLAY

Objective

- Display our name on the 7-segment text LCD

Methodology

- Generate the Verilog code for each sentence to be displayed
- For each sentence
  - Generate a Verilog case block that assigns values to temporary registers (tmp0, tmp1, …)
- A multiplexer selects which sentence (temporary register) to display

SCRIPT

```
def verilog_case_block(sentence, tmp_name):
    lines = []
    lines.append(f"// Case block for {tmp_name} representing \"{sentence}\"")
    lines.append("always @(cnt_lcd)")
    lines.append("    case(cnt_lcd[5:1])")
    for i, ch in enumerate(sentence):
        # Compute mapped value by subtracting 0x20 from the ASCII value.
        mapped_val = ord(ch) - 0x20
        mapped_hex = f"{mapped_val:02X}"
        index_hex = f"{i:X}"  # index in hex (e.g. 0, 1, 2, ... in hex)
        # Use "space" as comment if the character is a space.
        comment = "space" if ch == " " else ch
        lines.append(f"    'h{index_hex} : {tmp_name} = 'h{mapped_hex};    // {comment}")
    lines.append("    default : " + tmp_name + " = 'h00;")
    lines.append("    endcase")
    return "\n".join(lines)
```

Generating the case block for each character in the string

```
# Generate a case block for each sentence.
verilog_sentences = []
for idx, sentence in enumerate(sentences):
    tmp_name = f"tmp{idx}"
    verilog_sentences.append(verilog_case_block(sentence, tmp_name))
    verilog_sentences.append("")  # Blank line for separation

# Generate the multiplexer block that chooses which sentence to output.
mux_lines = []
mux_lines.append("// Multiplexer to select the active sentence output")
mux_lines.append("always @(posedge clk or posedge rst)")
mux_lines.append("    if (rst)")
mux_lines.append("        lcd_db <= 0;")
mux_lines.append("    else if (lcd_en)")
mux_lines.append("        case(sentence_sel)")
for idx in range(len(sentences)):
    mux_lines.append(f"            {idx} : lcd_db <= tmp{idx};")
mux_lines.append("            default : lcd_db <= 0;")
mux_lines.append("        endcase")
mux_lines.append("    else")
mux_lines.append("        lcd_db <= 0;")

# Generate the clocked counter block that increments cnt_lcd and the sentence selector.
counter_lines = []
counter_lines.append("// Counter and sentence selector update block")
counter_lines.append("always @(posedge clk or posedge rst)")
counter_lines.append("    if (rst) begin")
```

Generating the multiplexer for the temporary registers

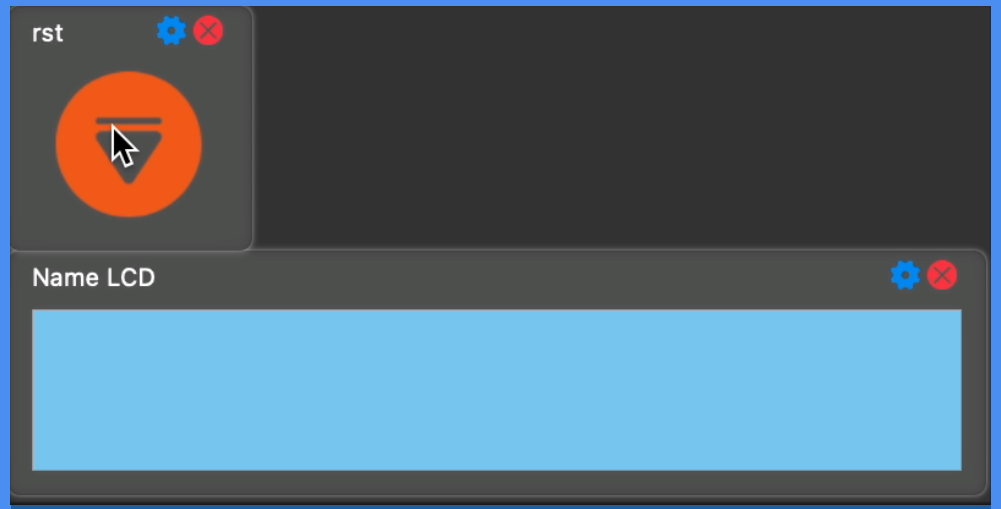
EXPERIMENT RESULTS

Script Input

```
# List of sentences to output.
sentences = [
    "Hello, Fudan!",
    "Jimmy Wang or",
    "Jun Wei Wang",
    "From school of Software!"
]
```

Script input

Rabbit Output

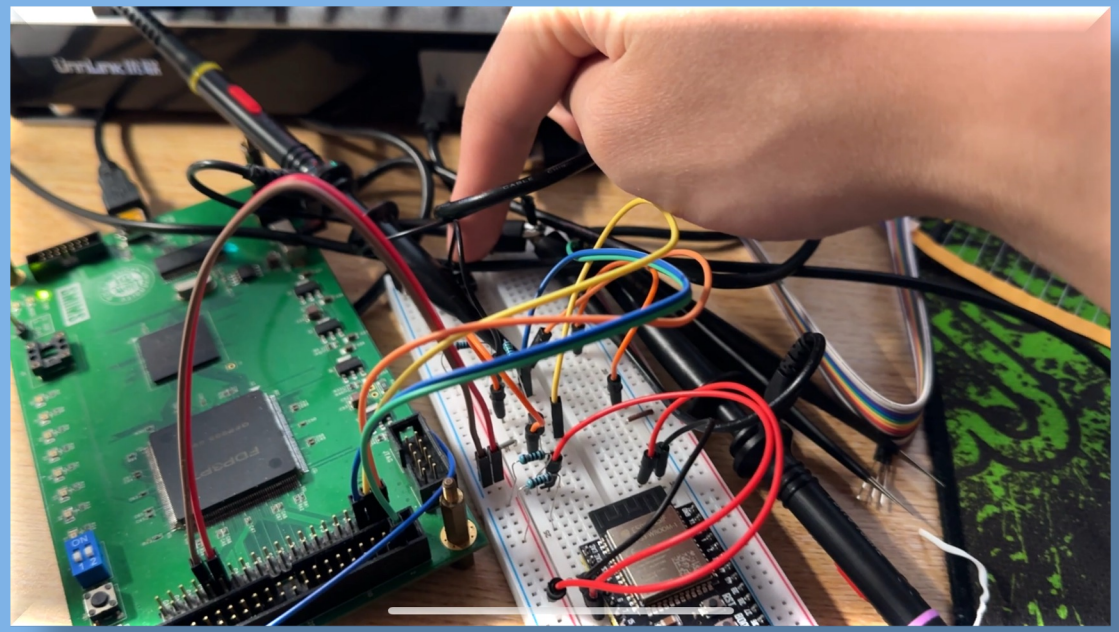


DEMO: Rabbit output results

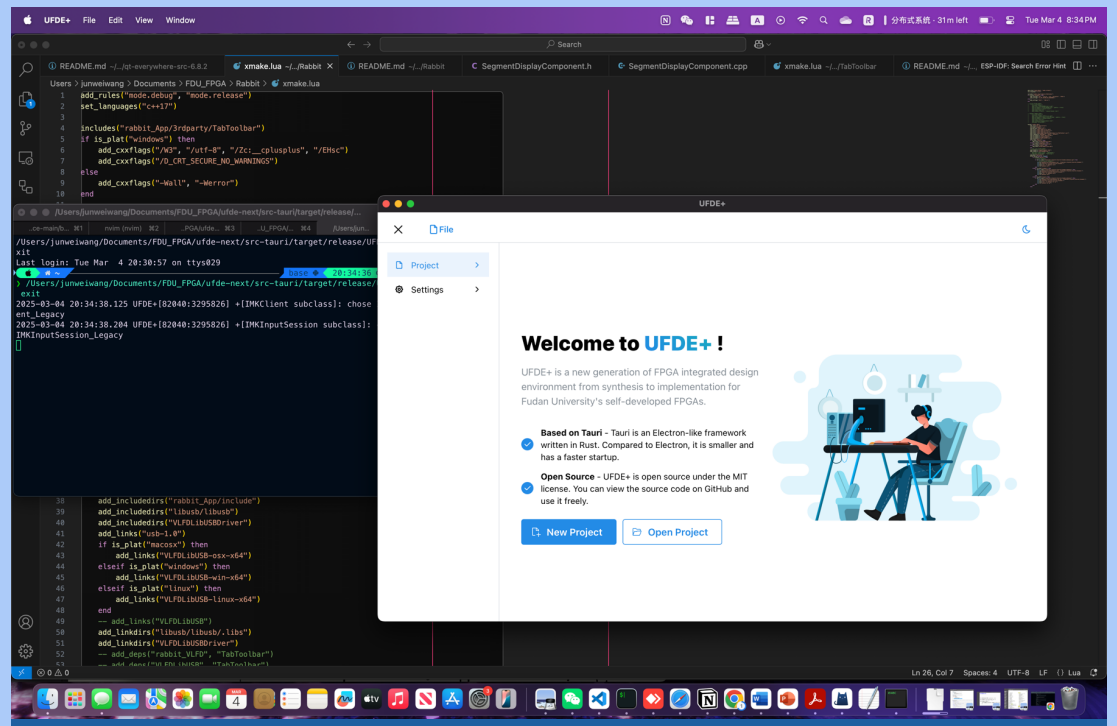
EXPERIMENT 2

Main topics

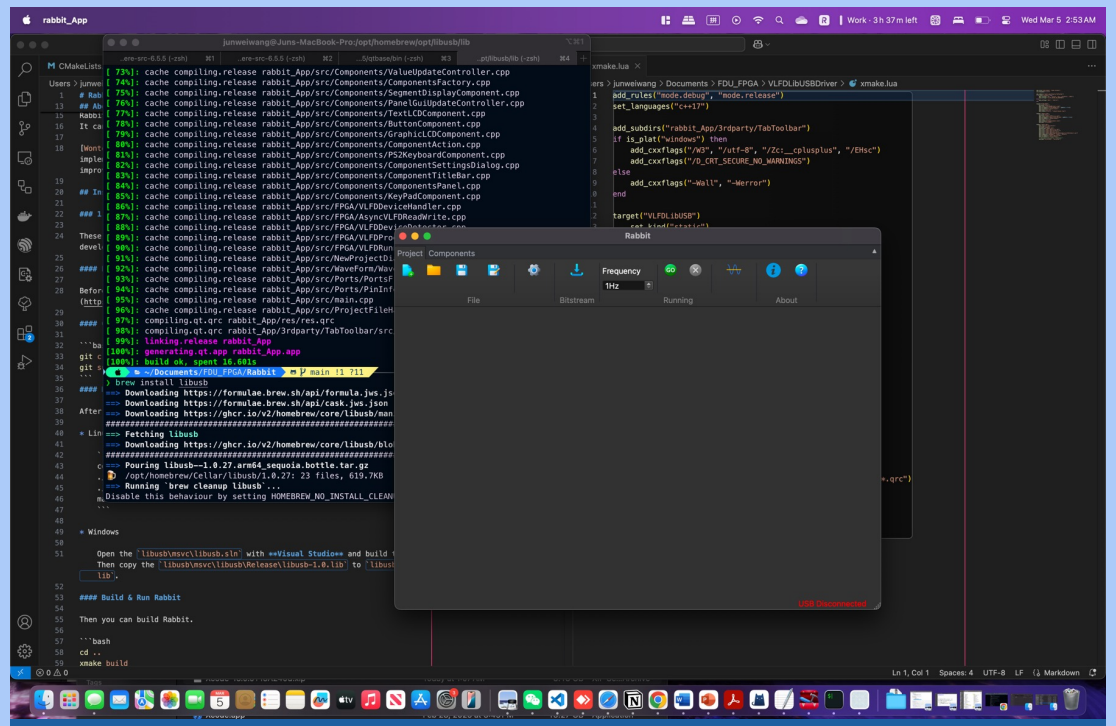
- UDFE & Rabbit compiled on MacOS
- SMIMS Engine
- SMIMS VLFD Driver Reimplemented in Rust
- FDE-CLI
- I2C



UFDE & RABBIT ON MACOS (ARM)

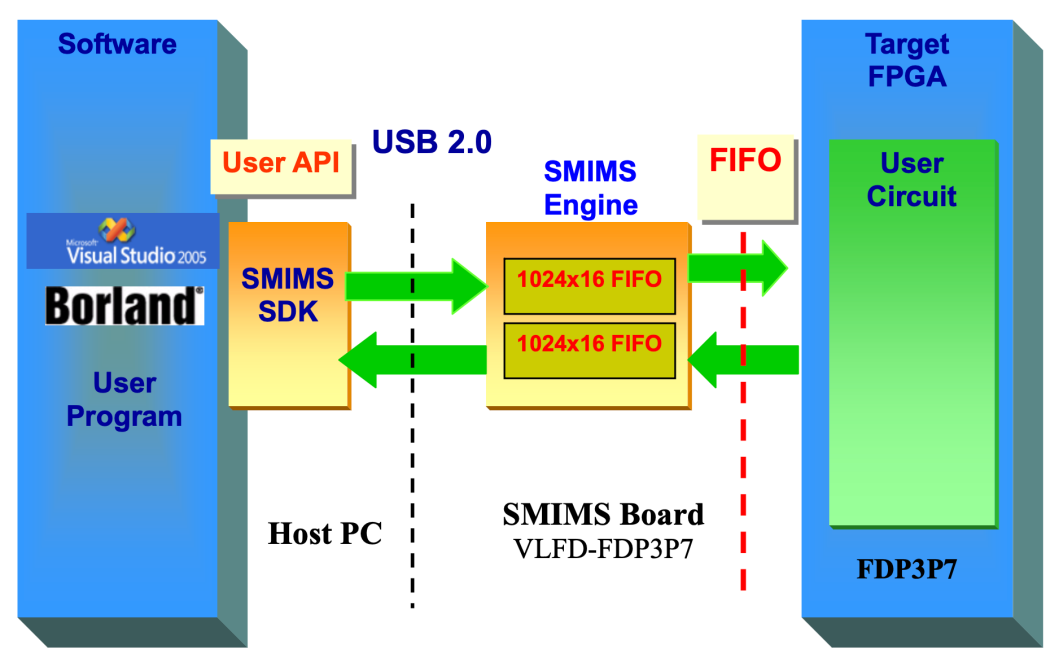


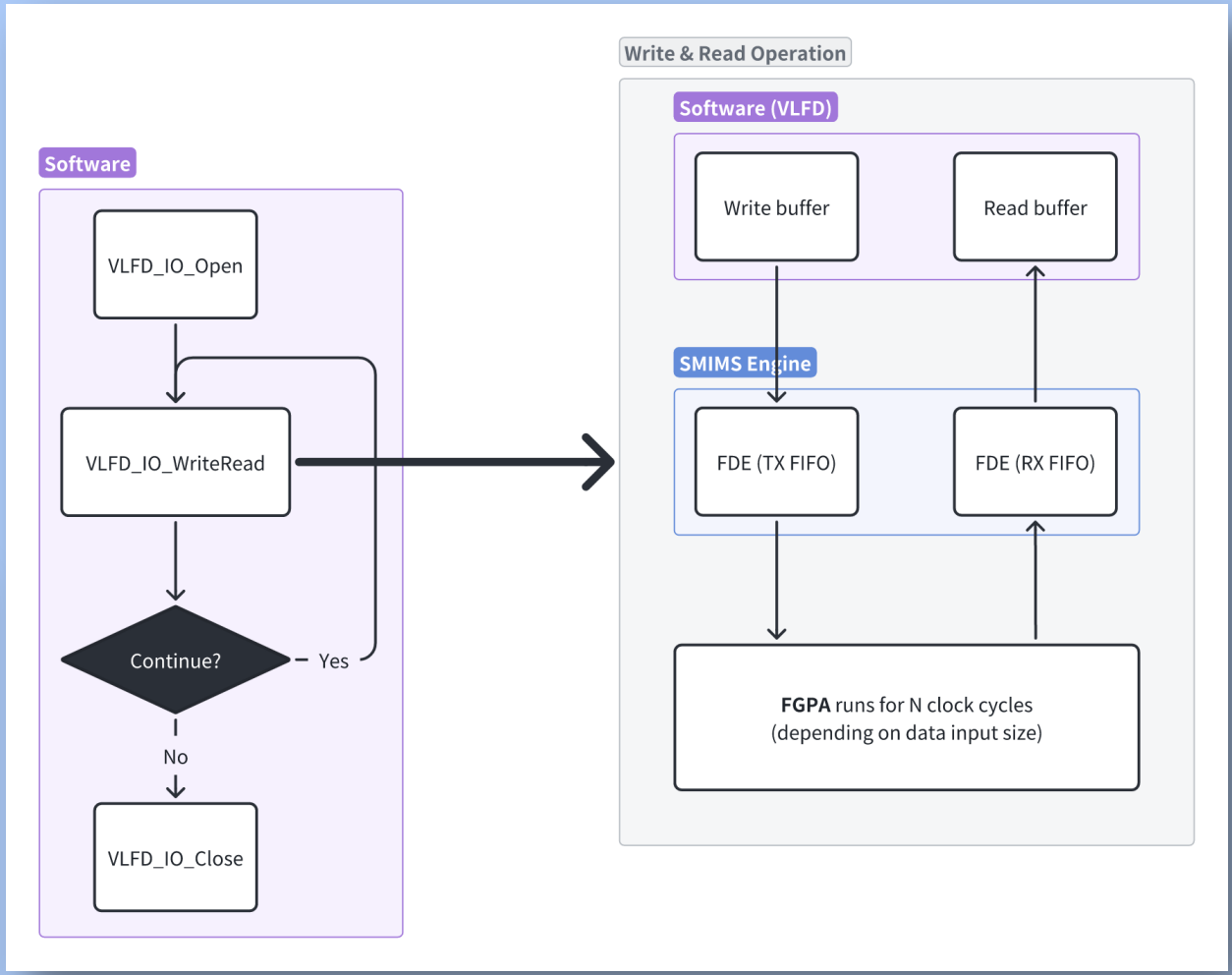
UFDE compiled on MacOS



Rabbit Compiled on MacOS

SMIMS ENGINE





DEMO

```
00000000001111111111111111111111111111111111111111011000101000
00000000001111111111111111111111111111111111111111011100000000
00000000001111111111111111111111111111111111111111011001000101
00000000001111111111111111111111111111111111111111011100000000
...
```

Raw data from RabbitReadData.txt

```
<design name="name_display">
  <port name="lcd_db[0]" position="P7"/>
  <port name="lcd_db[1]" position="P6"/>
  <port name="lcd_db[2]" position="P5"/>
  <port name="lcd_db[3]" position="P4"/>
  <port name="lcd_db[4]" position="P9"/>
  <port name="lcd_db[5]" position="P8"/>
  <port name="lcd_db[6]" position="P16"/>
  <port name="lcd_db[7]" position="P15"/>
  <port name="clk" position="P77"/>
  <port name="rst" position="P151"/>
  <port name="lcd_en" position="P11"/>
  <port name="lcd_rst" position="P18"/>
</design>
```

Constraints .xml file

VeriComm FPGA IO Pins VeriComm FPGA IO Pins

Input		Output	
VeriComm	Physical Pin	VeriComm	Physical Pin
clk	P77		
input 0	P151	output 0	P7
input 1	P148	output 1	P6
input 2	P150	output 2	P5
input 3	P152	output 3	P4
input 4	P160	output 4	P9
input 5	P161	output 5	P8
input 6	P162	output 6	P16
input 7	P163	output 7	P15
input 8	P164	output 8	P11
input 9	P165	output 9	P10

Mapping spreadsheet

indices mapped to **one** bit on the data read from the FIFO (little endian)

- lcd_db[0] ~ lcd_db[7] → output[0] ~ output[7]
- rst → input[0]
- lcd_rst → output[8]

DEMO

```
00000000001111111111111111111111111111111111111111011000101000
00000000001111111111111111111111111111111111111111011100000000
00000000001111111111111111111111111111111111111111011001000101
00000000001111111111111111111111111111111111111111011100000000
...
```

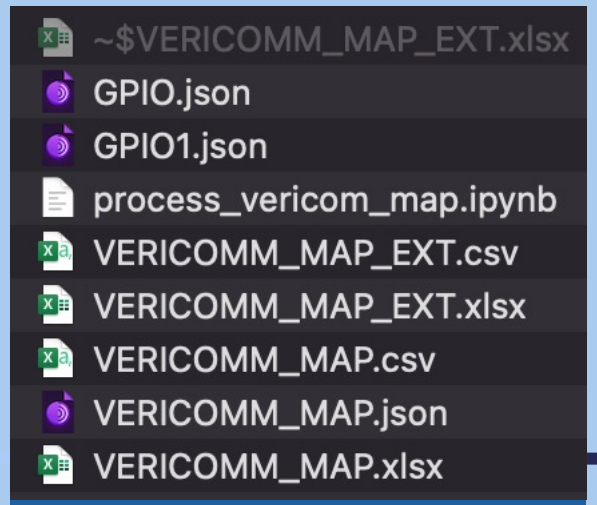
Raw data from RabbitReadData.txt

- lcd_db[0] ~ lcd_db[7] → output[0] ~ output[7]
- rst → input[0]
- lcd_rst → output[8]

4	3	2	1
00000000 00111111	11111111 11111111	11111111 11111111	11110110 **00101000**
00000000 00111111	11111111 11111111	11111111 11111111	11110111 **00000000**
00000000 00111111	11111111 11111111	11111111 11111111	11110110 **01000101**
00000000 00111111	11111111 11111111	11111111 11111111	11110111 **00000000**

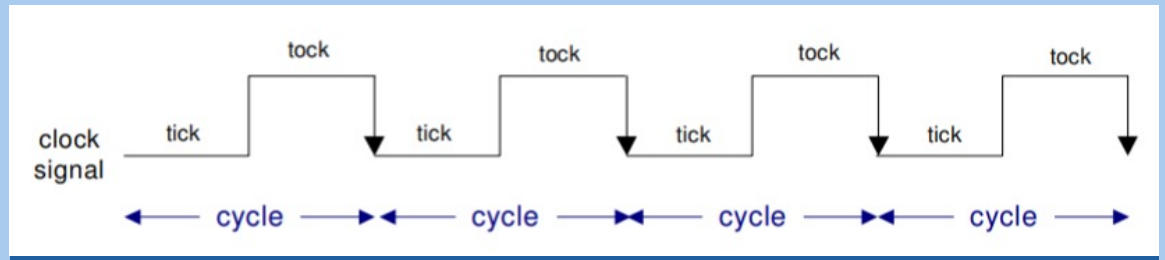
Raw	Hex	
00101000	0x28	H
00000000	0x00	
01000101	0x45	e
00000000	0x00	
		…

SMIMS ENGINE SUMMARY



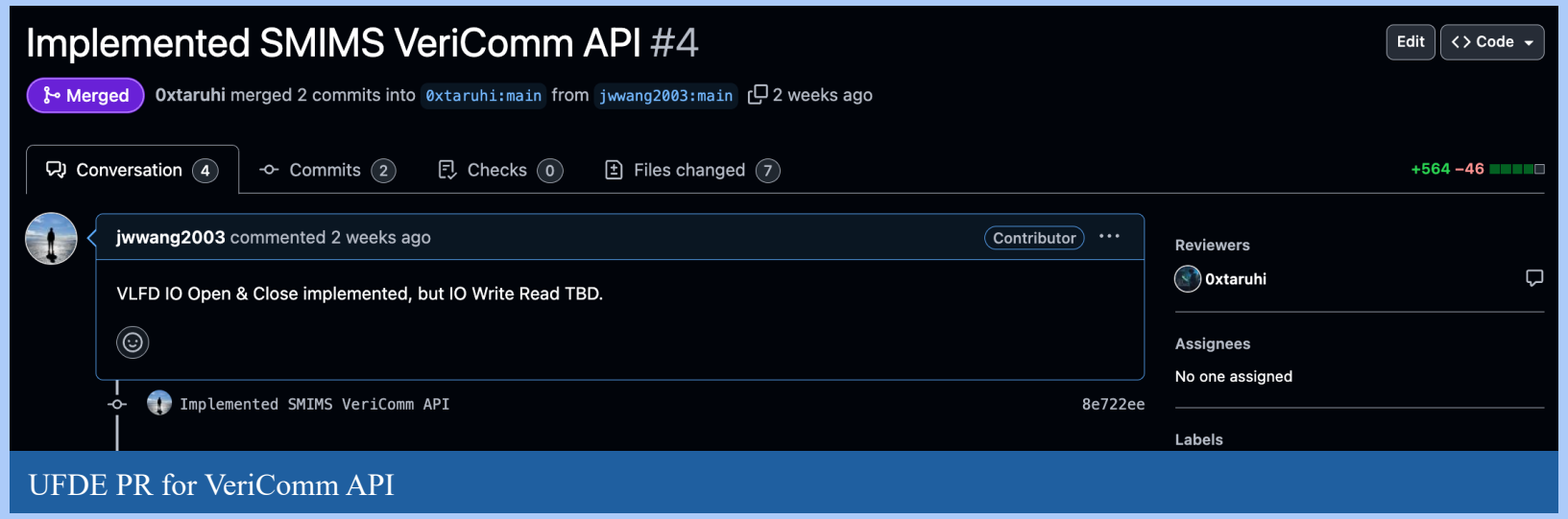
VeriComm ports & GPIO domentation

- Has two FIFOs (TX & RX) of size 1024x16
- Each set of data is 64 bits long so it takes up four slots in each FIFO
- The number of items in the FIFO determines the number of ”tick tocks”
- For each entry in the TX FIFO, expect its response from the RX FIFO
- Data read from the USB driver is in little-endian format (split into byte chucks, read from rightmost chunk)



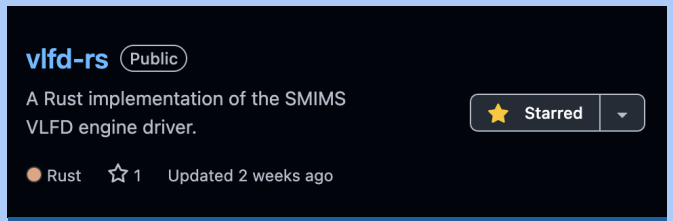
Clock signal reference image

SMIMS VLFD DRIVER IN RUST



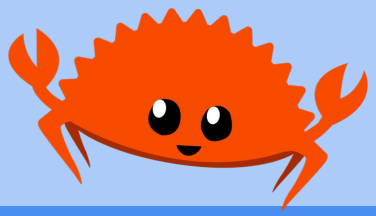
UFDE PR for VeriComm API

https://github.com/0xtaruhi/ufde-next/pull/4



GitHub repo for VLFD driver

https://github.com/jwwang2003/vlfd-rs



FEATURES

Data Trasnfer API

- [x] `SMIMS_FIFO_Write` (device_handler/fifo_write)
- [x] `SMIMS_FIFO_Read` (device_handler/fifo_read)

Implementation Check List (Transfer API)

Command API

- [x] `SMIMS_SyncDelay` (device_handler/sync_delay)
- [x] `SMIMS_CommandActive` (device_handler/command_active)
- [x] `SMIMS_CFGSpaceRead` (device_handler/read_cfg)
- [x] `SMIMS_CFGSpaceWrite` (device_handler/write_cfg)
- [x] `SMIMS_FGPAProgrammerActive` (device_handler/activate_fpga_programmer)
- [x] `SMIMS_VeriCommActive` (device_handler/activate_vericomm)
- [x] `SMIMS_VeriInstrumentActive` (device_handler/activate_veri_instrument)
- [x] `SMIMS_VeriLinkActive` (device_handler/activate_verilink)
- [x] `SMIMS_VeriSoCActive` (device_handler/activate_veri_soc)
- [x] `SMIMS_VeriCommProActive` (device_handler/activate_vericomm_pro)
- [x] `SMIMS_VeriSDKActive` (device_handler/activate_veri_sdk)
- [x] `SMIMS_FlashReadActive` (device_handler/activate_flash_read)
- [x] `SMIMS_FlashWriteActive` (device_handler/activate_flash_write)

Implementation Check List (Command API)

ezIF (Easy Interface)

The EasyInterface (used by Rabbit) is wrapper ontop of the SMIMS FIFO.

- [x] `VLFD_IO_ProrgamFPGA`
- [x] `VLFD_IO_Open` (device_handler/io_open)
- [x] `VLFD_IO_WriteReadData` (device_handler/io_write_read_data)
- [x] `VLFD_IO_Close` (device_handler/io_close)

Implementation Check List (ezIF)

Encrypt API

- [x] `SMIMS_EncryptTableRead` (device_handler/encrypt_table_read)
- [x] `SMIMS_EncryptTableDecode` (device_handler/decoded_encrypt_table)
- [x] `SMIMS_EncryptData` (device_handler/encrypt)
- [x] `SMIMS_DecryptData` (device_handler/decrypt)
- [ ] `SMIMS_EncryptCopy` (Optional)
- [ ] `SMIMS_DecryptCopy` (Optional)
- [ ] `SMIMS_LicenseGen` (Optional)

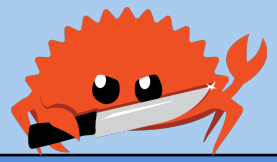
Implementation Check List (Encryption API)

and more…

Future Work

- Finish implementing LicenceGen & serial functions
- Device context (API natively support multi-device)
- Handle asynchronous events (device unplug, etc.)
- Handle errors (prevent segmentation faults)

C++ VS RUST VS JS

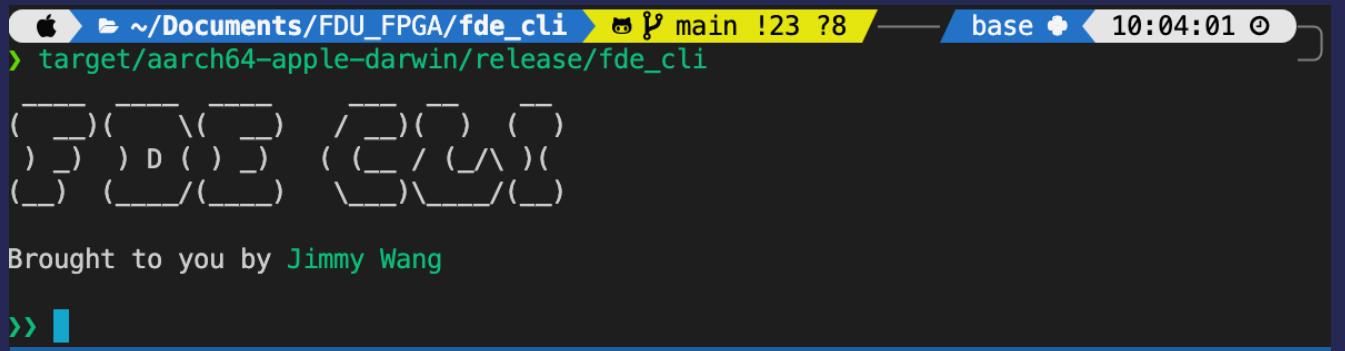


Aspect	C++	Rust	JavaScript (TypeScript)
Type	Compiled, OOP	Compiled, Systems	Interpreted/Compiled, Dynamic
Memory Management	Manual (pointers)	Ownership model, no GC	Automatic Garbage Collection
Performance	Very High (system-level)	High (performance, safety)	Medium (depends on V8 engine)
Concurrency	Manual threads, parallelism	Safe concurrency (async/await)	Async with event loop
Ease of Learning	Hard	Moderate	Easy (TypeScript improves JS)
Tooling	Excellent (IDE support)	Excellent (Cargo, rustfmt)	Excellent (VS Code, NPM)
Ecosystem	Mature	Growing, strong for systems	Huge (Node.js, NPM)
Use Case	High-performance apps	Systems programming	Web development, Async I/O
Integration in Node.js	Native Addons (node-gyp)	FFI, WebAssembly	Native, main language for Node.js
Asynchronous Support	Manual handling (threads)	Advanced async/await	Native async/await, Promises

FDE-CLI

Features

- Aims to offer a “GDB like” debugging experience
- Manage multiple FDE boards
- Debug SMIMS engine
- Manage project files
- Downloading bitstream & fine grain control over SMIMS engine FIFO
- Almost everything Rabbit has to offer, except virtual components & GUI



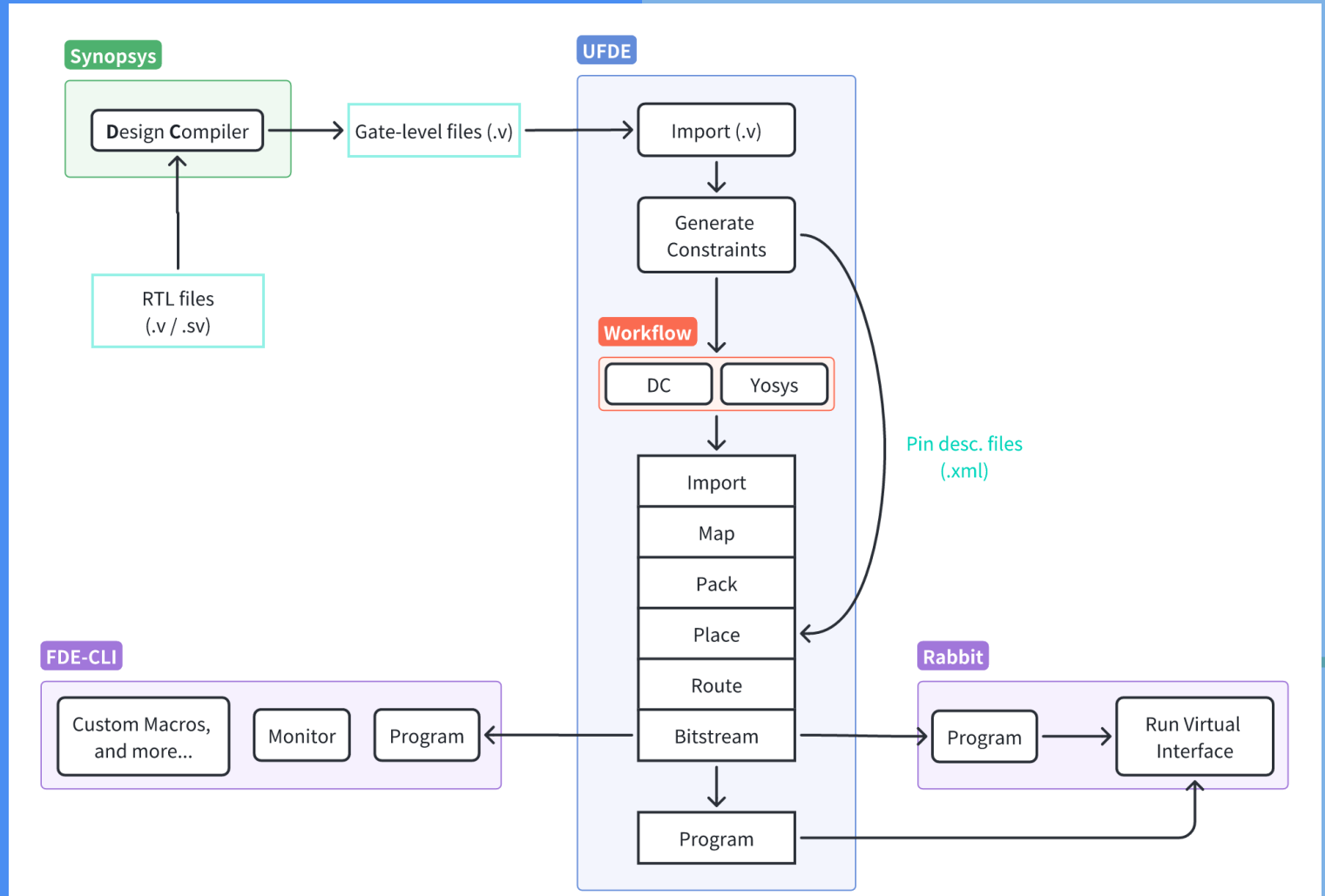
Screenshot of FDE-CLI

Limitations of Rabbit

- Poor cross-platform compatibility
- Virtual components on Rabbit are limited
- No “fine-grain” control over IO on Rabbit
- Not scriptable

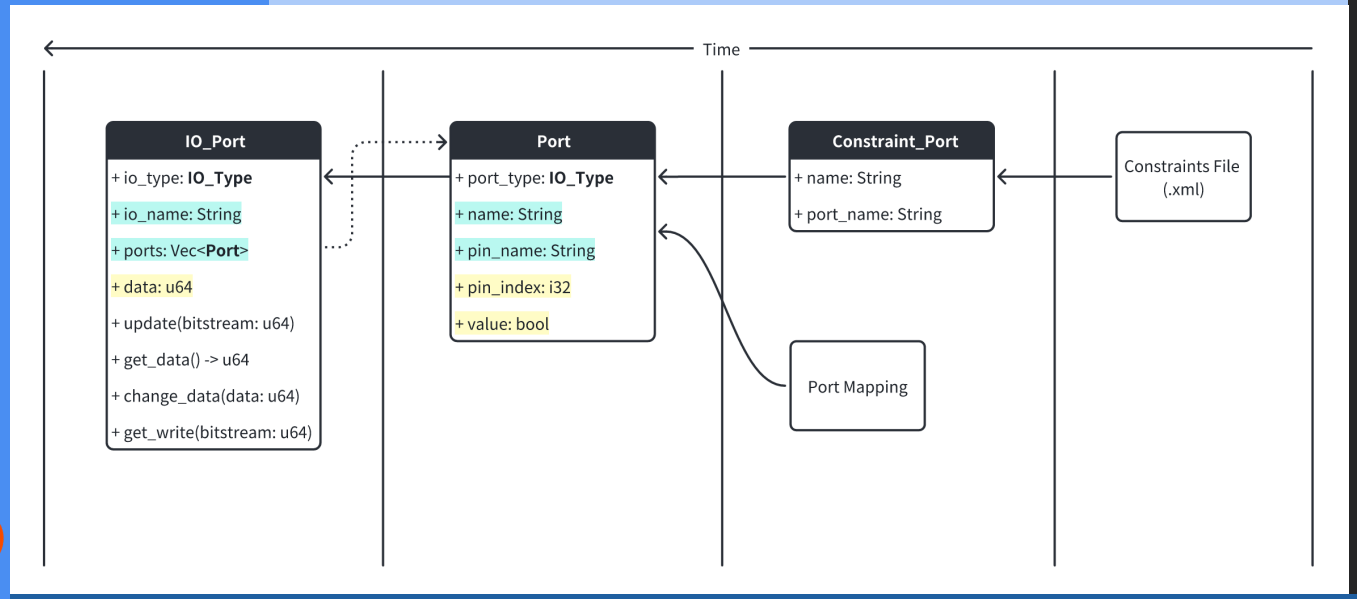
What FDE-CLI offers?

- Better cross-platform compatibility
- SMIMS driver natively in Rust
- More versatility in terms of the processing of IO data from SMIMS’s FIFO
- Fine grain control over the data sent & read
- Scripting



FDE Workflow

FDE-CLI STRUCTURE



IOPort & Port structs & implementations

```
src
  helper
    bitstream.rs 1
    cli_commands.rs
    constraints.rs
    mod.rs
    smims_cfg.rs
  ports
    mod.rs 5
    parse.rs
    table.rs
  utilities
    fifo.rs 6, U
    mod.rs U
  vlfd
    cfg.rs M
    constants.rs 1, M
    device_handler.rs M
    helper.rs U
    mod.rs M
    program_handler.rs 1, M
    README.md M
    serial.rs 9+, M
    structs.rs U
    usb_handler.rs M
  cli.rs 7, M
  constants.rs
  file_parser.rs 2, U
  main.rs M
  manager.rs
```

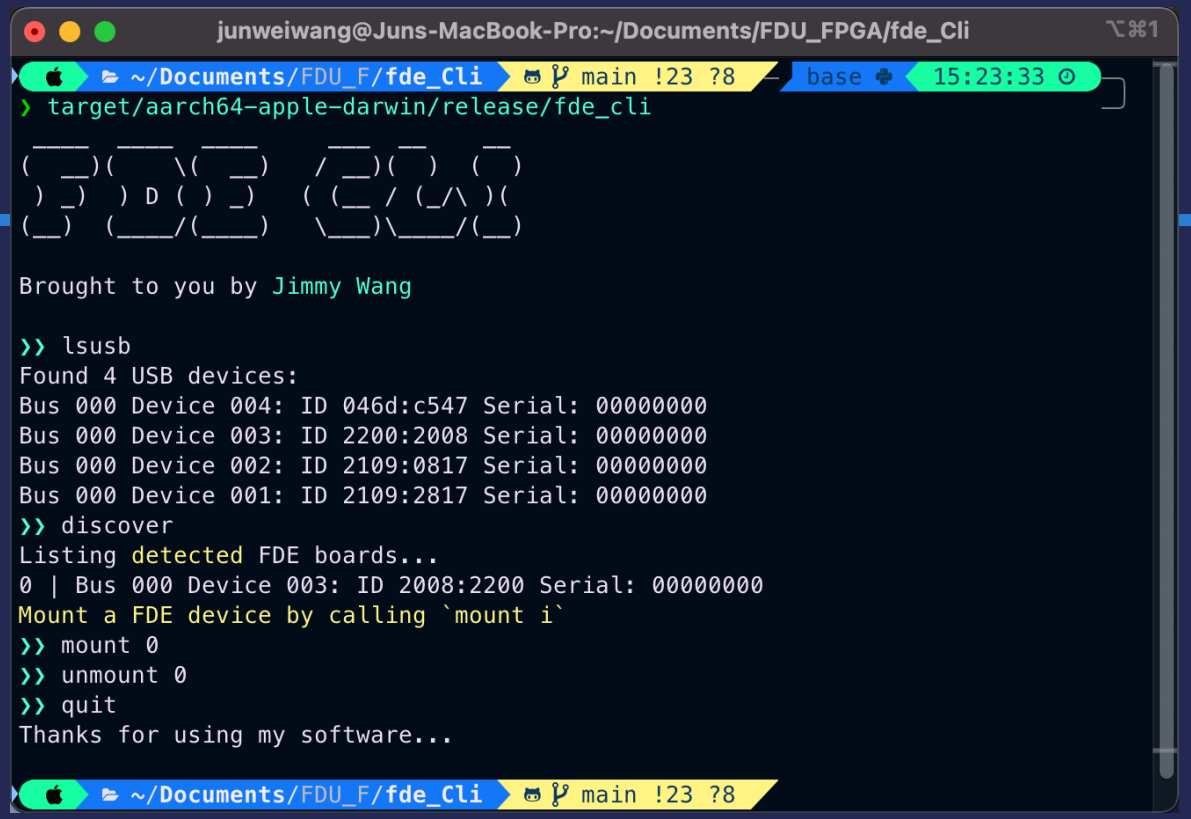
FDE-CLI folder structure

FDE-CLI DEMO

DISCOVERY

FDE board discovery

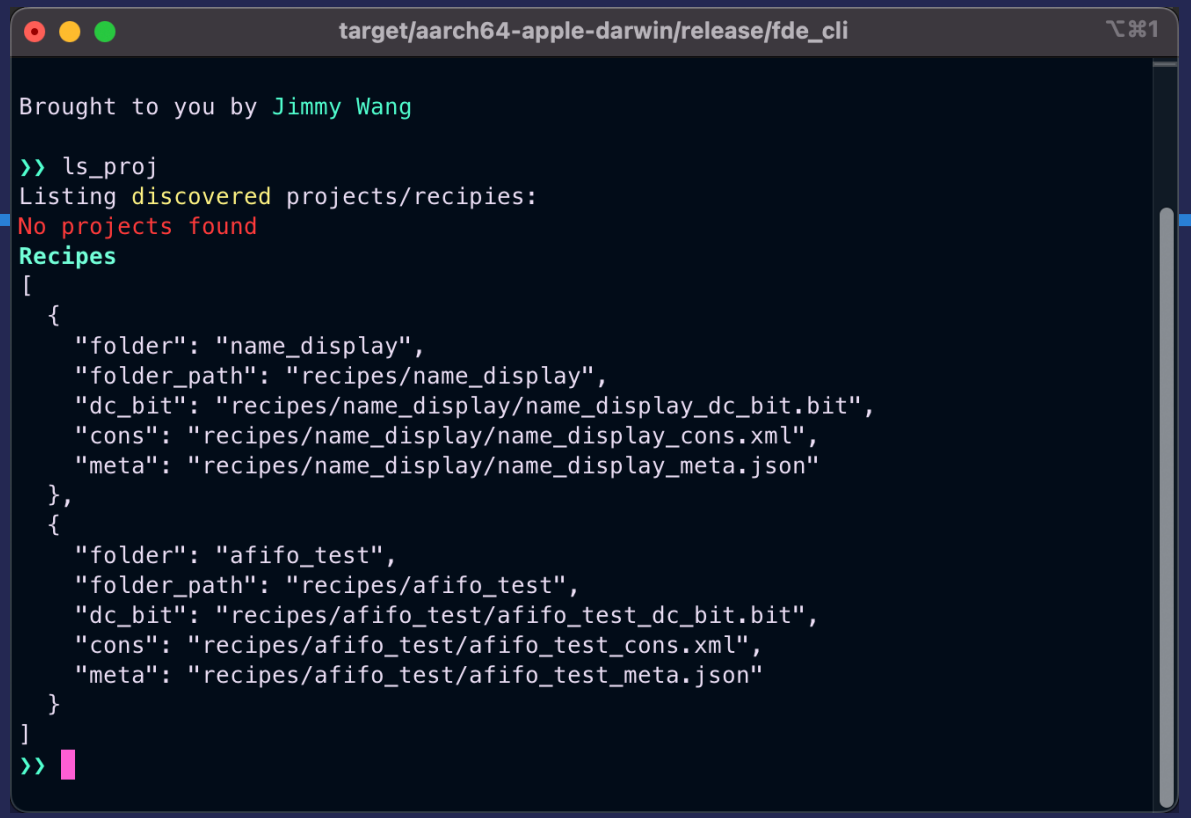
- FDE-CLI can recognize & differentiate multiple FDE boards
- Each detected FDE board can be indexed (0~n boards connected)
- We can “mount” or “unmount” the FDE board(s)



PROJECTS & RECIPES

Similar to UFDE & Rabbit

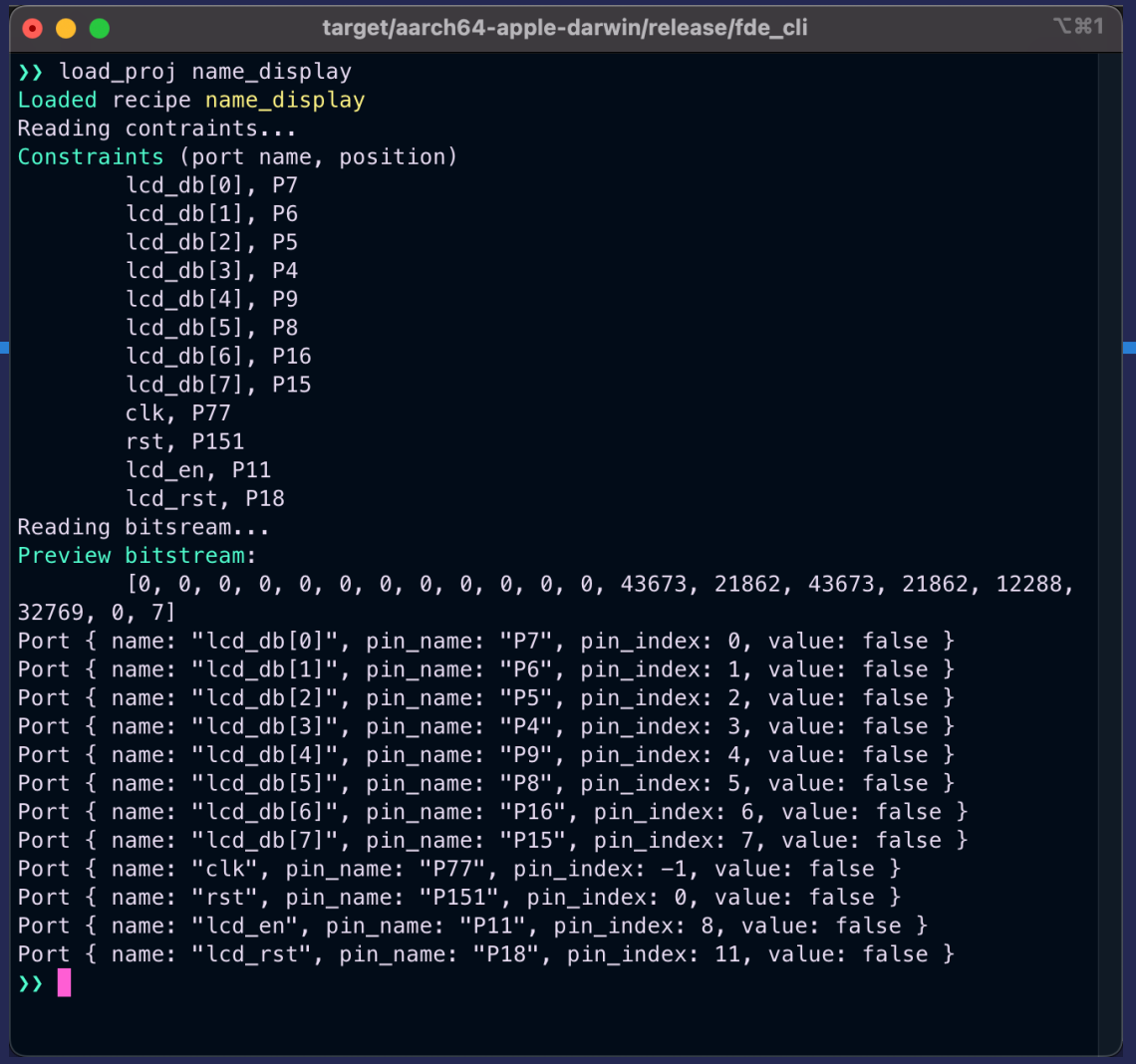
- FDE-CLI has its own project management system
- Projects refer to user imported projects
- Recipes are built in project demos
- In the future, projects/recipes can also have scripts in them



LOADING A PROJECT/RECIPE

Interpreting constraints & detecting IO ports

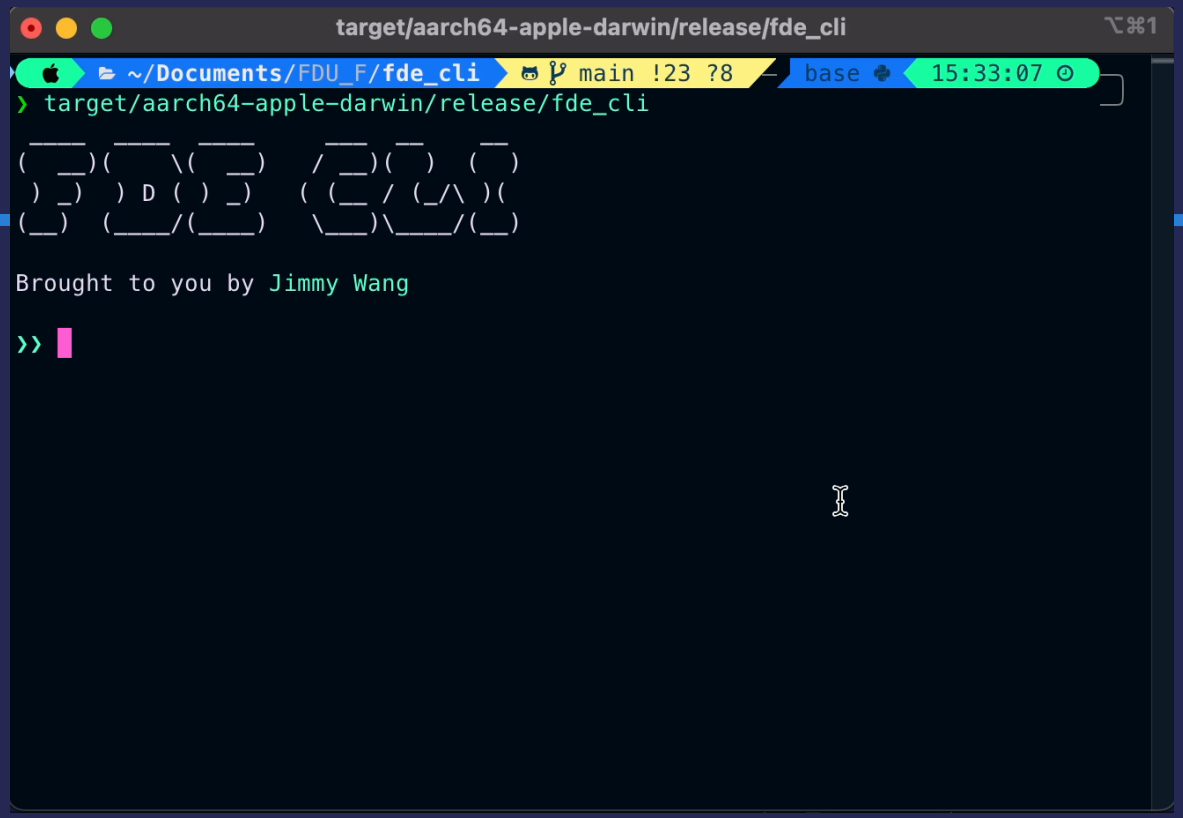
- FDE-CLI can parse and interpret constraint files & bitstreams
- FDE-CLI has a regex mechanism that detects whether an IO port is a single data line or is part of an array
  - Useful later when observing how data changes



FDE-CLI DEMO
NAME DISPLAY

Objective

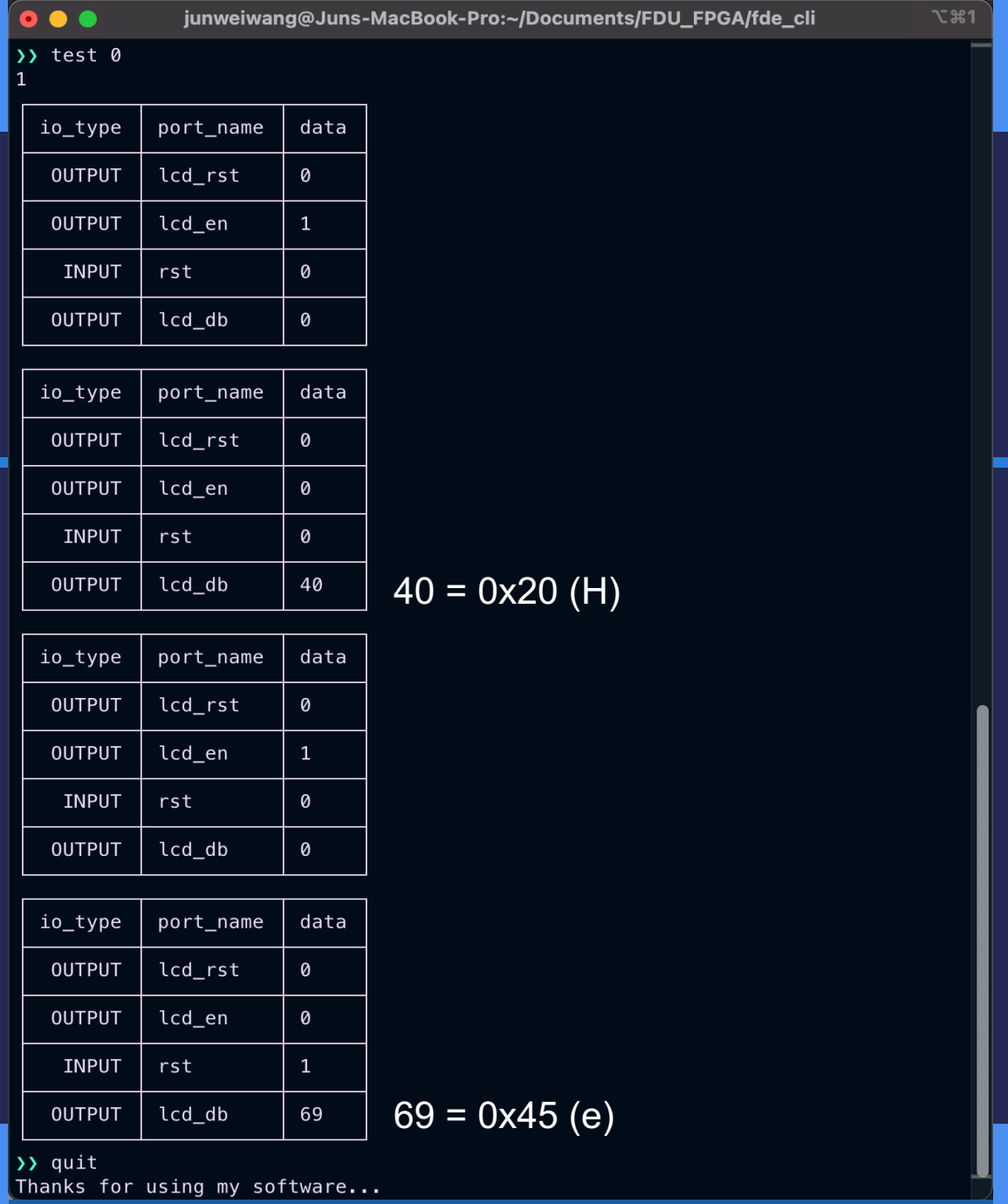
- FDE device discovery
- Loading a project/recipe
- Programming a FDE board
- Running for 4 cycles & printing the output



FDE-CLI DEMO
NAME DISPLAY

Explanation

- Each table is the resulting IO state of each clock cycle
- Notice how in these four clock cycles, we see “H” and ”e” being outputted on the data bus
- We can expect to see “llo” if we run for 6 more cycles

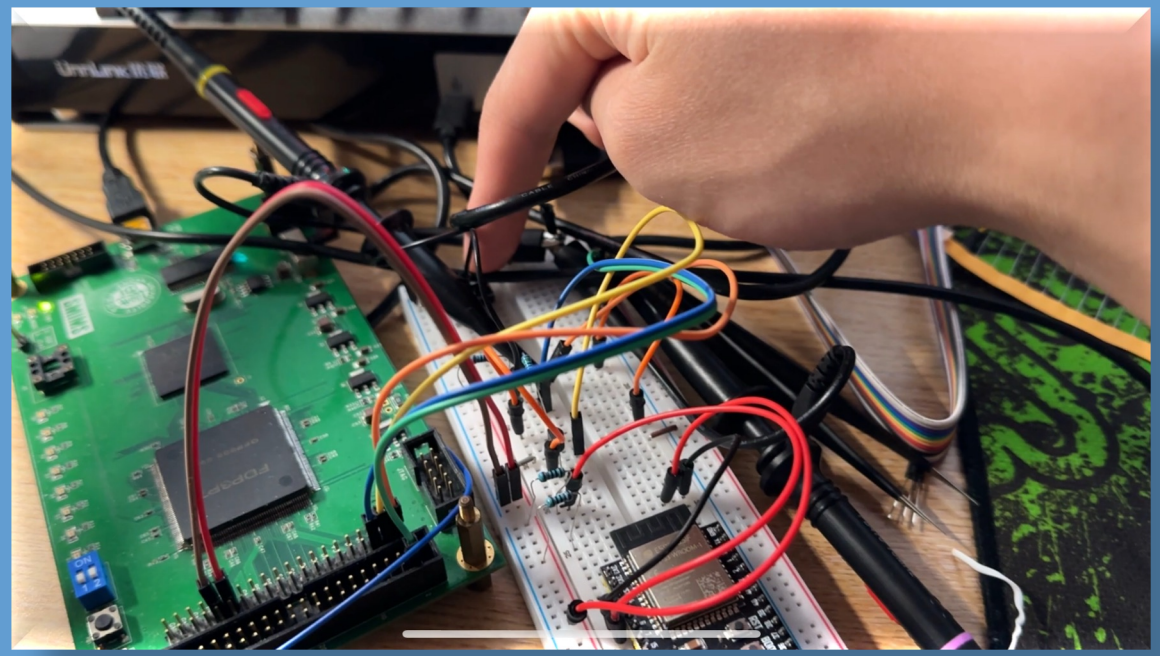


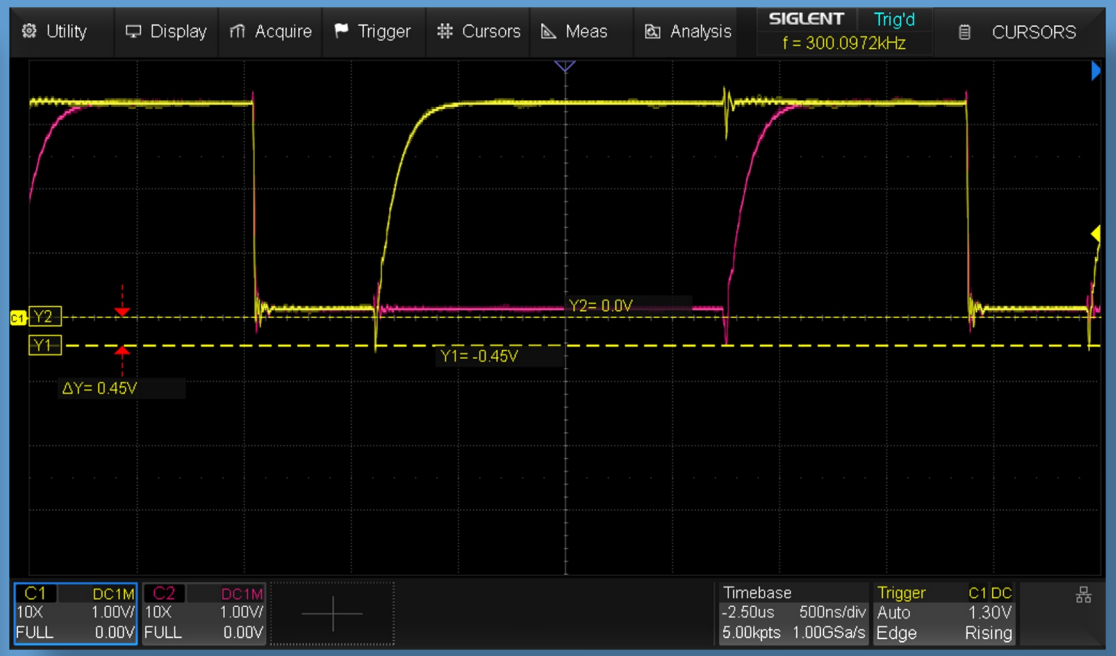
Snapshot of output of the output of name display FDE CLI demo

FDE-CLI FUTURE PLANS

Future to-do:

- Multi-threading (to interact with multiple FDE boards at the **same time**)
- Multi-threading to support polling data from a AFIFO
- Support scripts (similar to DC's .tcl file)
- Better error handling
- Handle asynchronous events (s.a. USB unplug)
- …





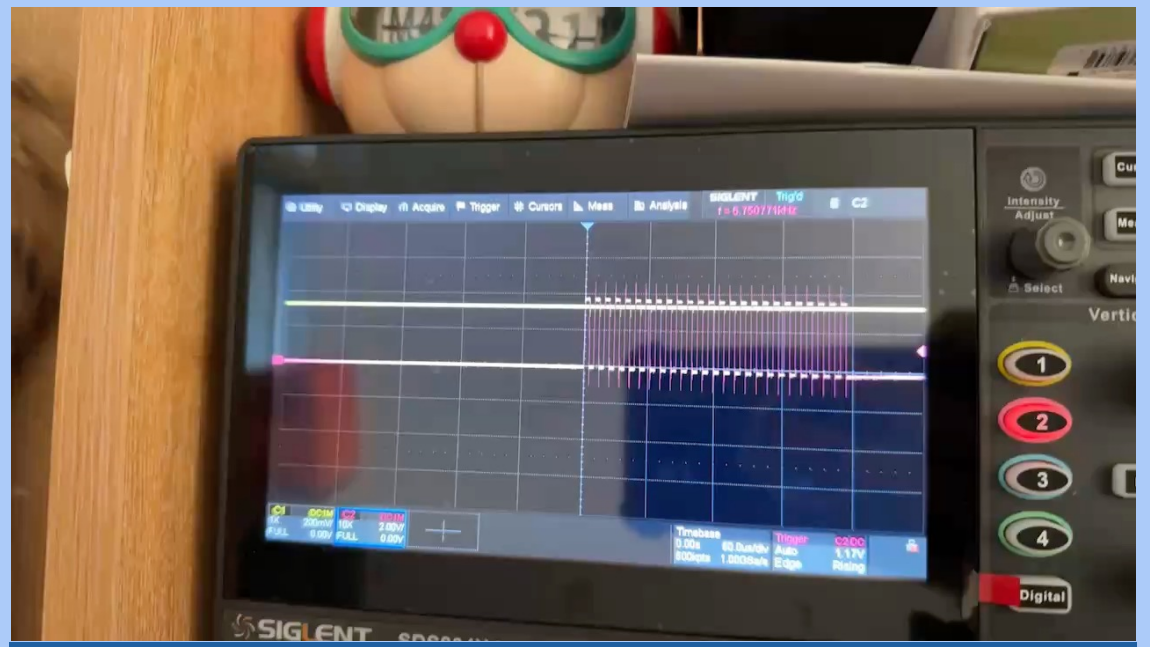
I2C

(***I***nter-***I***ntegrated ***C***ircuit)

GROWING PAINS

Timing Issues

- Qt::PreciseTimer & thread::sleep (Rust) suffer from precision issues
- At high frequencies, we might observer jitters (fluctuations in timing)
- Windows/MacOS is definitely not a **real-time operating** system (**RTOS**), and most Linux machines aren't either



Video of **jitters** observed from the SMIMS clock at high frequencies

Rabbit

- Rabbit does not have any virtual component(s) for reading & writing (hex/binary data) to and from registers
- No fine-control over the processing of data received & transmitted on every clock cycle
- No scripting / automation

SOLUTION

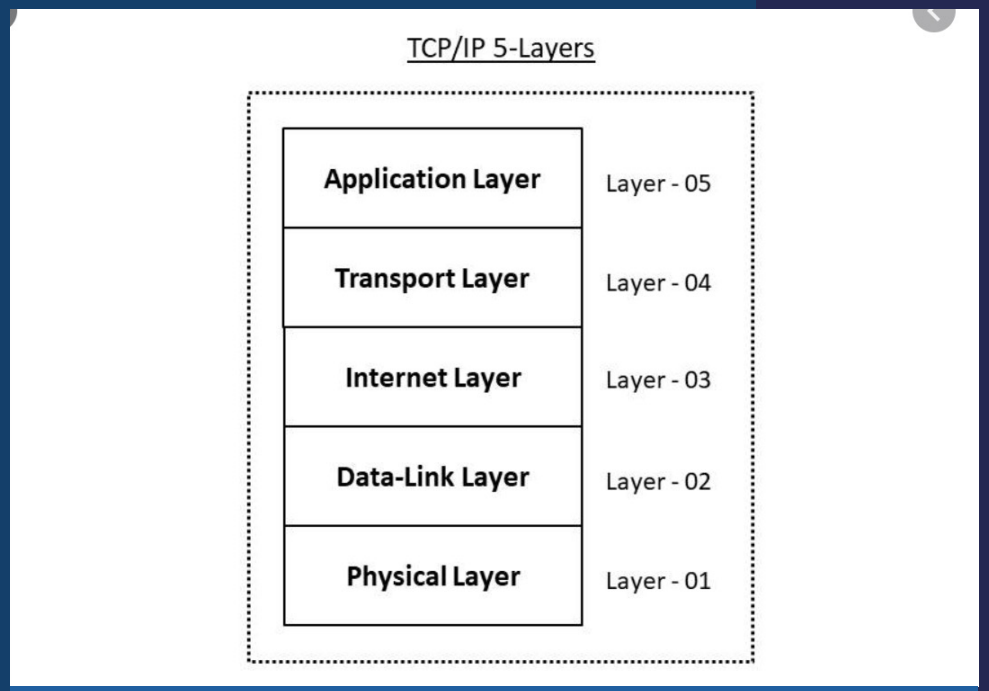
FDE board has two clocks

- SMIMS engine clock (P77)
- Internal 30MHz clock (P185)
- Have I2C running on the 30MHz oscillator
- Use an asynchronous FIFO between I2C module & SMIMS engine

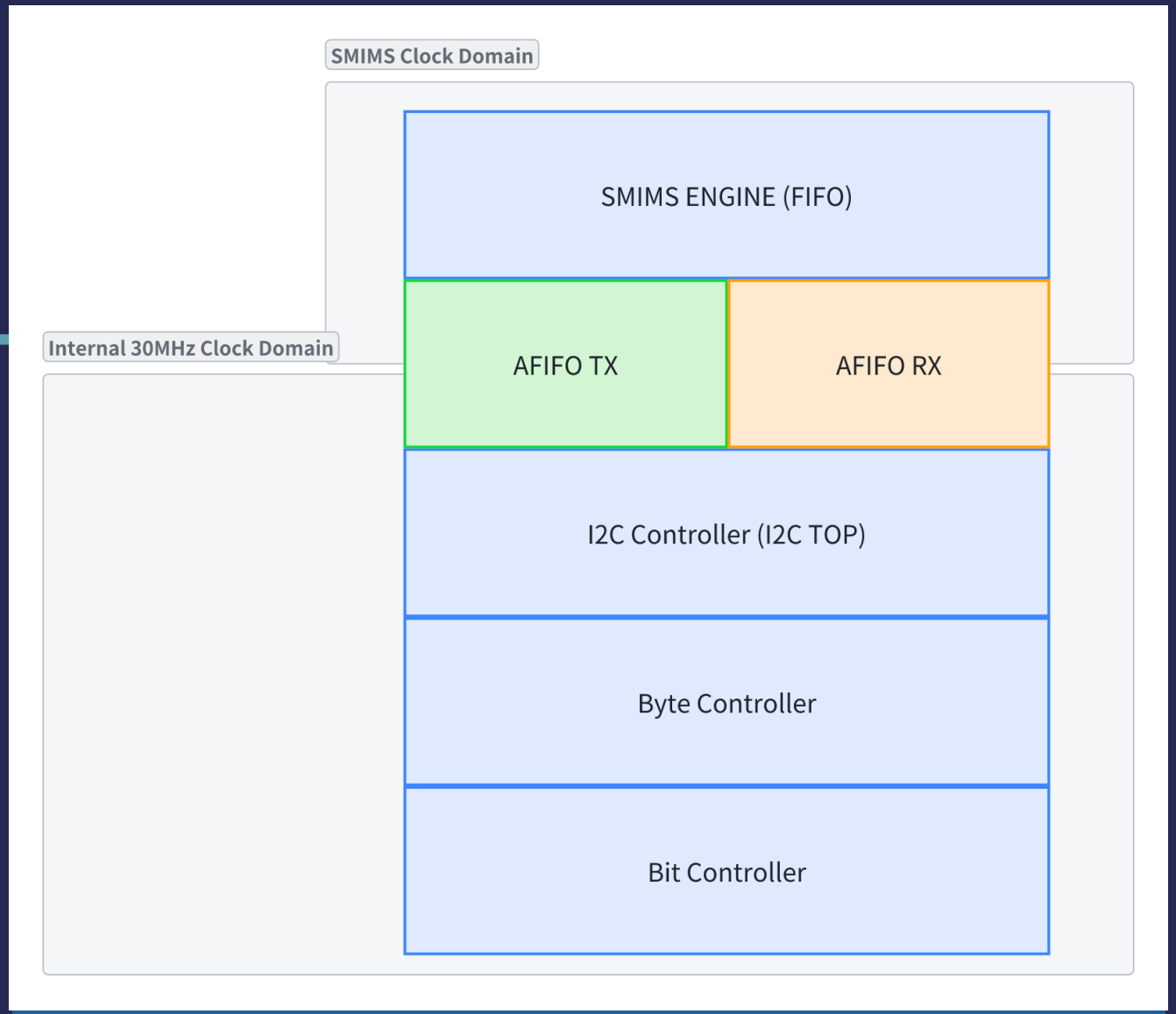
Software

- To accurately interact with my I2C module using AFIFOs, I need to build my own program to write & read from SMIMS engine FIFO
- Hence, the need for FDE-CLI

I2C MODULE TOPOLOGY



The 5-layer networking model



The proposed topology for my I2C module

I2C OPEN-DRAIN

Open-Drain Configuration

- **N-channel MOSFET**: This configuration uses an N-channel MOSFET to drive the SDA/SCL lines.
  - **When the MOSFET is off** the signal line is left "floating" and is pulled **high** by the pull-up resistor, representing **logic HIGH**.
  - **When the MOSFET is on**, the signal is pulled **low** (GND), representing **logic LOW**.
  - Based on 2N7000 N-channel MOSFET

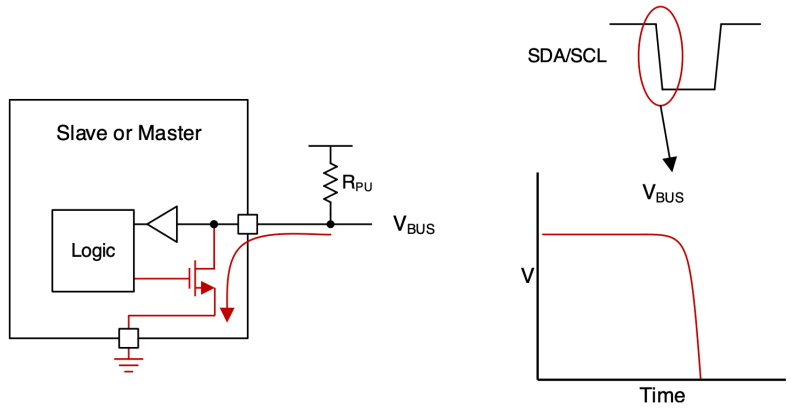
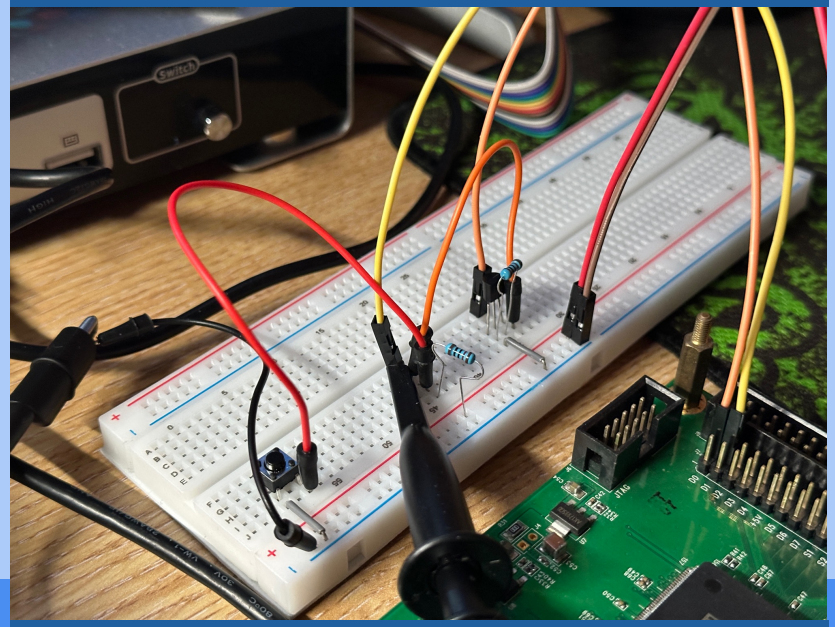


Figure 3. Pulling the Bus Low With An Open-Drain Interface

TI documentation of open-drain I2C configuration

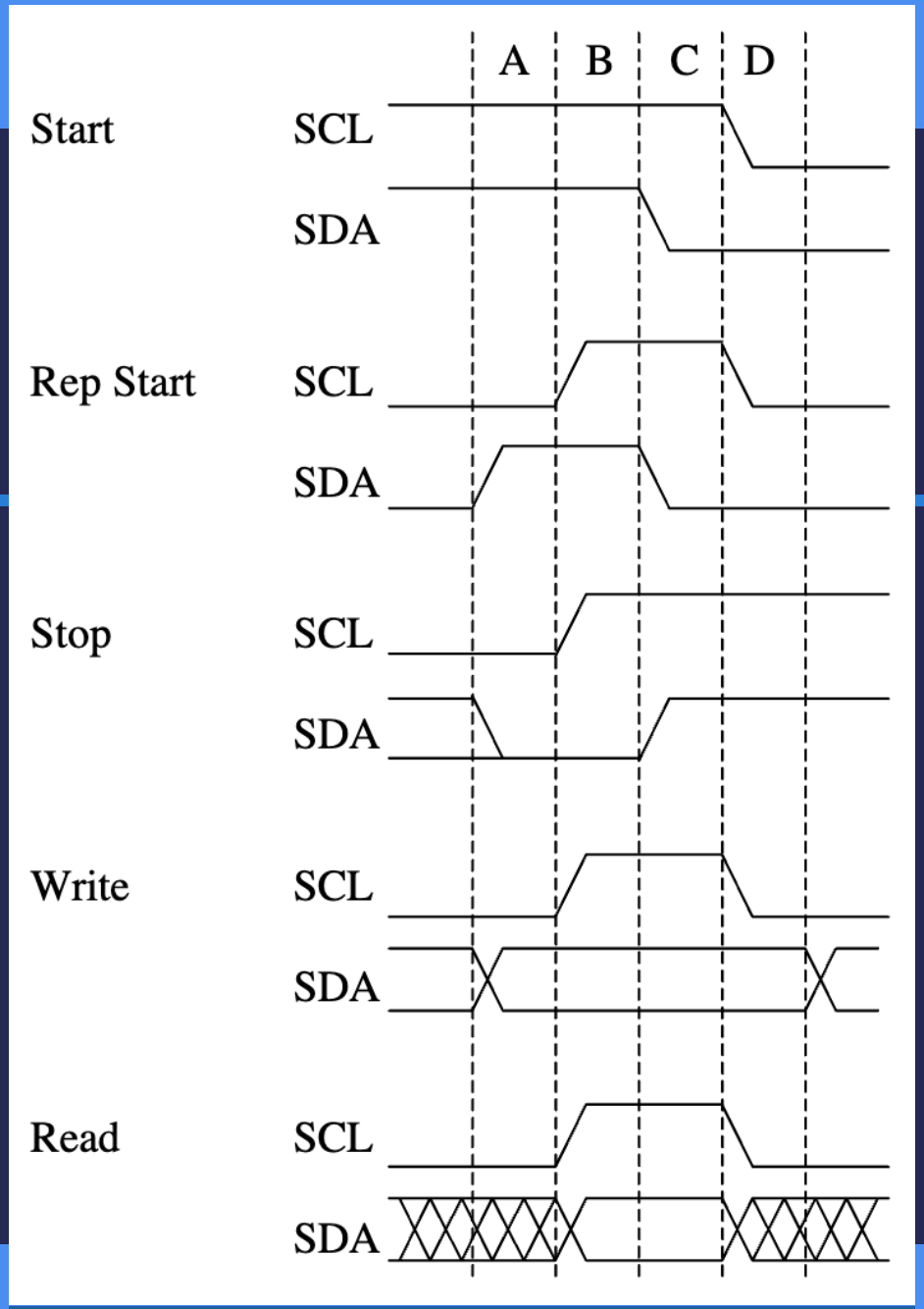


An open-drain configuration implemented on a breadboard

BIT CONTROLLER

Bit Controller Responsibilities

- Handles raw signal interpretation and bus control
- Deals with Start/Stop conditions, ACK/NACK signals
- **Filters**, manages **clock skew**, and handles **arbitration**
- Comprises of the bit controller module itself & two tri-state buffer modules (for SDA & SCL)



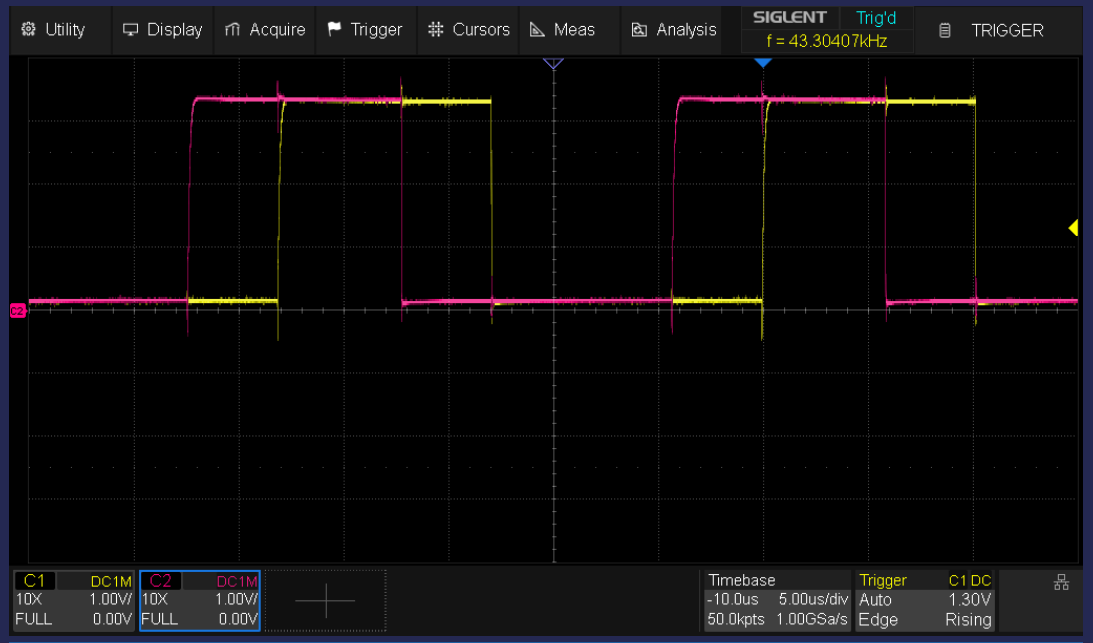
SDA & SCL signal transitions for the 5 conditions

BIT CONTROLLER

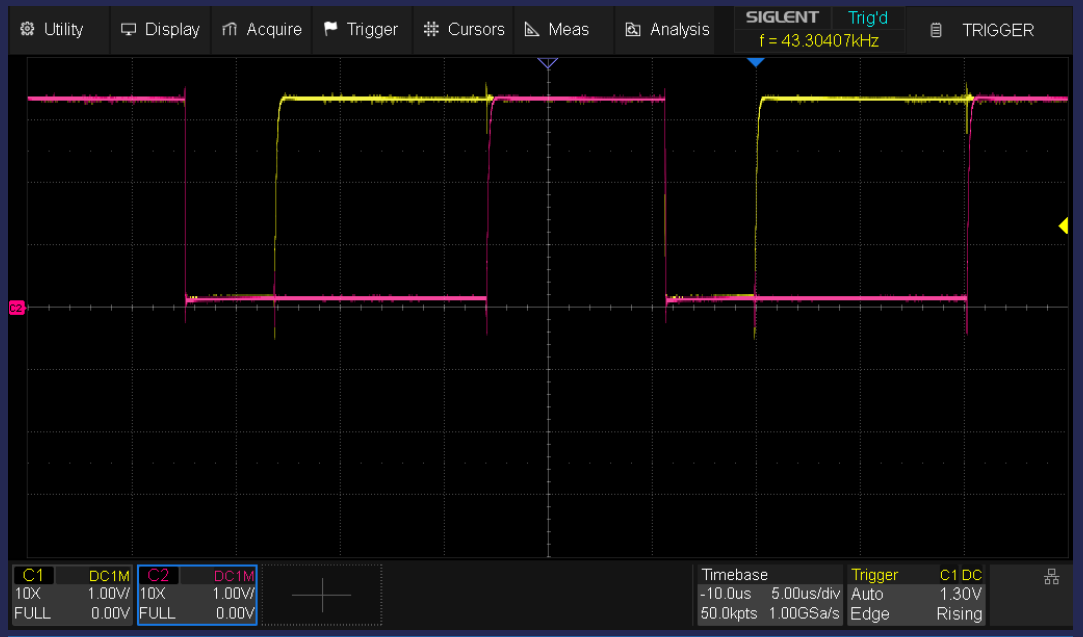
Filters, Clock Skew, and Arbitration in the I2C Bit Command Controller

- **Filters**: Ensures clean, noise-free signal transitions
- **Clock Skew Management**: Synchronizes clock signals between masters to ensure accurate data transfer
- **Arbitration Handling**: Resolves bus conflicts in multi-master setups, preventing data corruption

I2C DEMO
BIT CONTROLLER

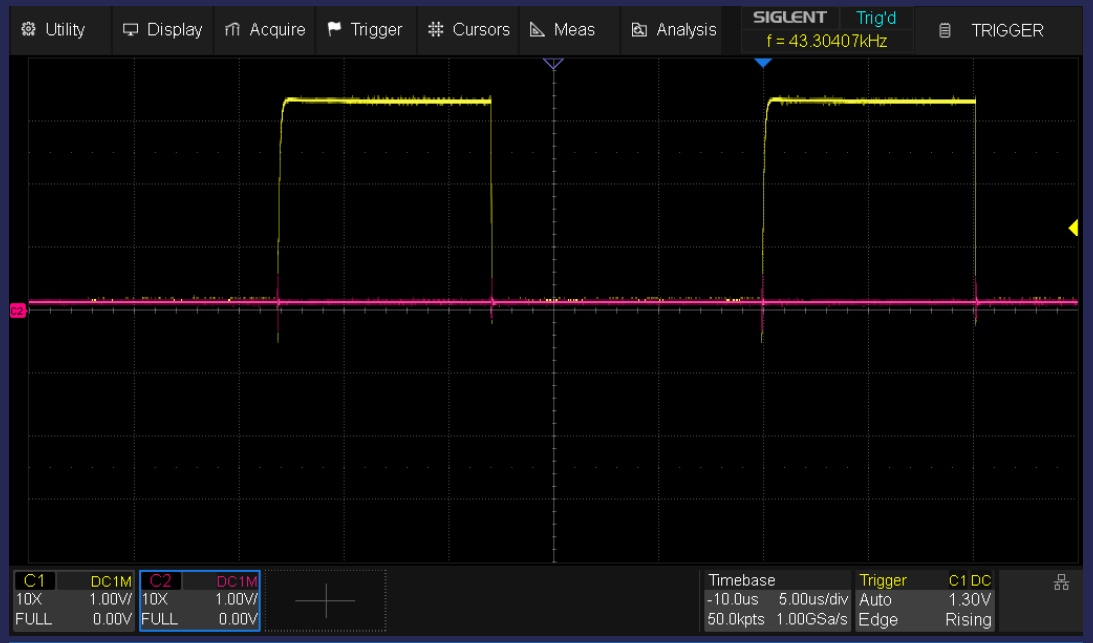


Bit controller: start command

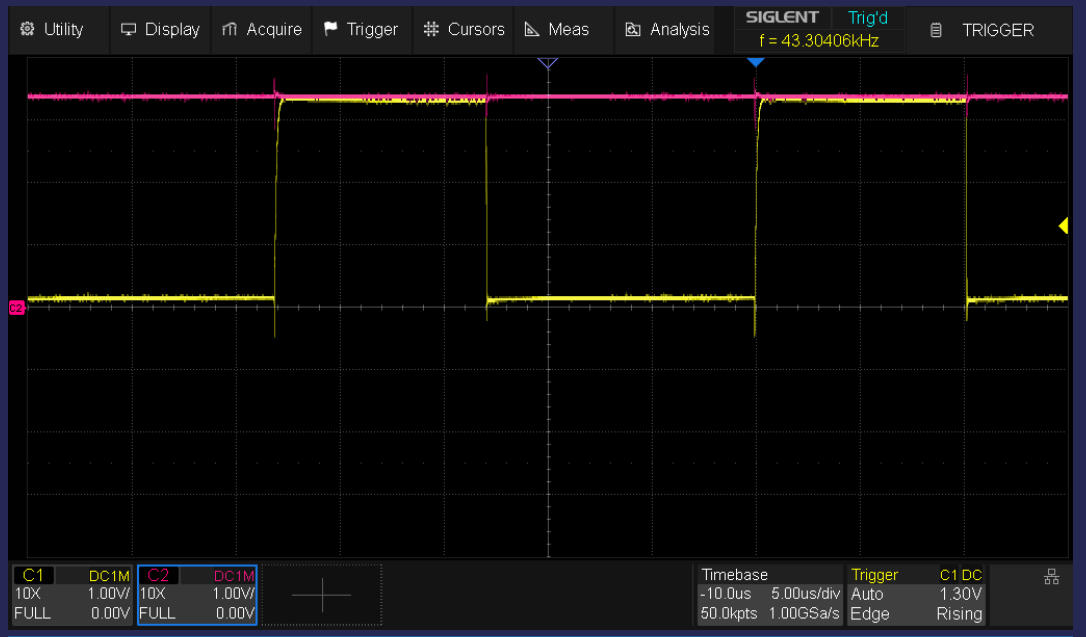


Bit controller: stop command

I2C DEMO
BIT CONTROLLER



Bit controller: write “0”

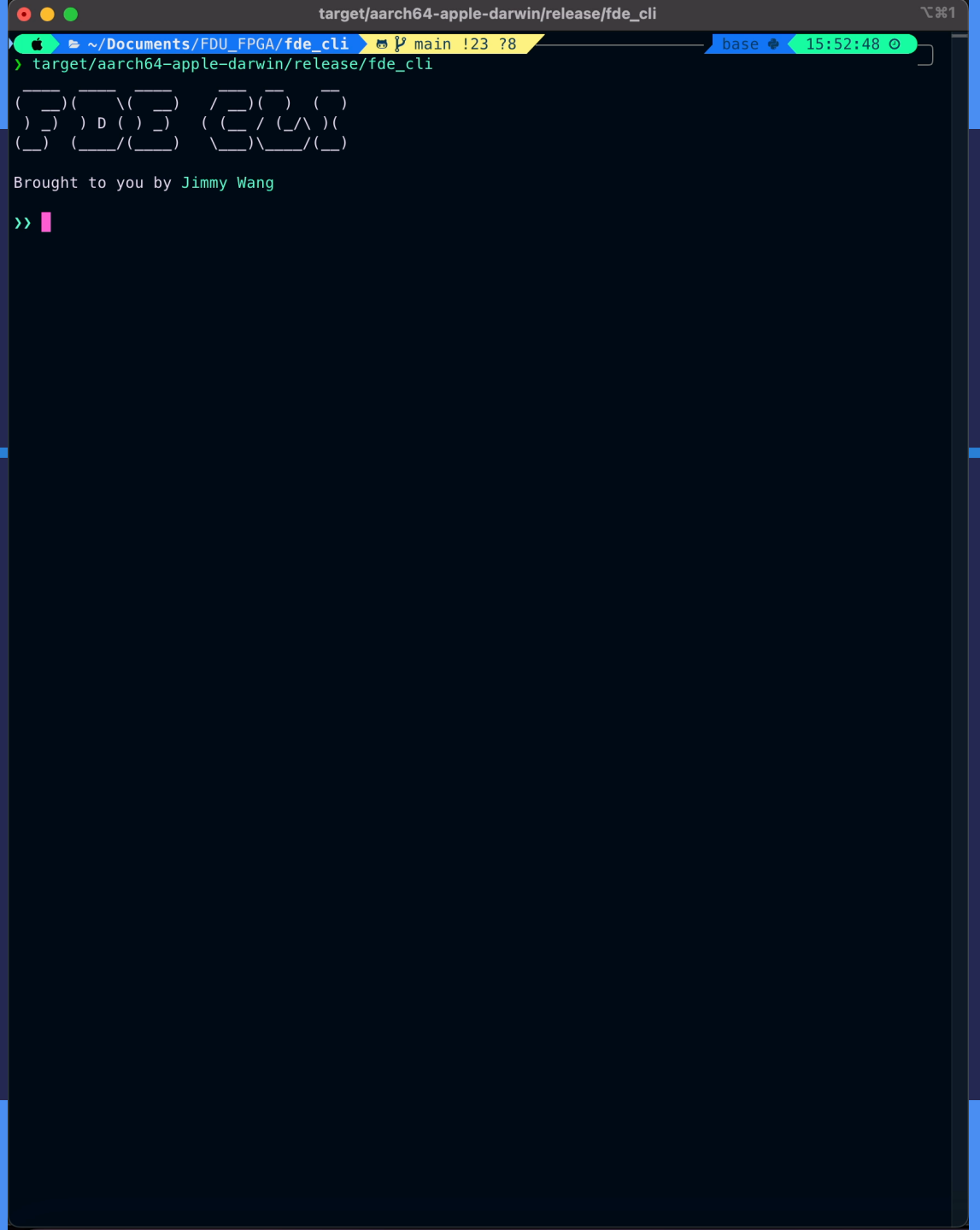


Bit controller: write “1”

FDE-CLI DEMO **AFIFO**

Objective

- Demo an asynchronous FIFO implemented on FDE
  - Two AFIFOs (TX & RX) linked to each other
  - Foundation for I2C module in the future
- **Clock-domain FIFO**
  - SMIMS engine clock based on FDE-CLI
  - Internal 30MHz clock driving the I2C module
- Write & read “1”, “2”, “3” to & from the AFIFO



```
16    # Pause for one clock cycle
17    0x0400
18    0x0000
19    0x0000
20    0x0000
21
22    # Send '1' to FIFO
23    0x0C01
24    0x0000
25    0x0000
26    0x0000
27
28    # Send '2' to FIFO
29    0x0C02
30    0x0000
31    0x0000
32    0x0000
33
34    # Send '3' to FIFO
35    0x0C03
36    0x0000
37    0x0000
38    0x0000
```

Data sent to SMIMS TX FIFO

io_type	port_name	data
INPUT	rst	0
INPUT	i_wdata	1
INPUT	i_rd	1
OUTPUT	fifo_1_w_full	0
OUTPUT	o_rempty	0
OUTPUT	o_wfull	0
INPUT	nReset	0
INPUT	i_wr	1
OUTPUT	o_rdata	1

io_type	port_name	data
INPUT	rst	0
INPUT	i_wdata	0
INPUT	i_rd	1
OUTPUT	fifo_1_w_full	0
OUTPUT	o_rempty	0
OUTPUT	o_wfull	0
INPUT	nReset	0
INPUT	i_wr	1
OUTPUT	o_rdata	0

io_type	port_name	data
INPUT	rst	0
INPUT	i_wdata	0
INPUT	i_rd	1
OUTPUT	fifo_1_w_full	0
OUTPUT	o_rempty	0
OUTPUT	o_wfull	0
INPUT	nReset	0
INPUT	i_wr	1
OUTPUT	o_rdata	0

io_type	port_name	data
INPUT	rst	0
INPUT	i_wdata	1
INPUT	i_rd	1
OUTPUT	fifo_1_w_full	0
OUTPUT	o_rempty	0
OUTPUT	o_wfull	0
INPUT	nReset	0
INPUT	i_wr	1
OUTPUT	o_rdata	1

io_type	port_name	data
INPUT	rst	0
INPUT	i_wdata	2
INPUT	i_rd	1
OUTPUT	fifo_1_w_full	0
OUTPUT	o_rempty	0
OUTPUT	o_wfull	0
INPUT	nReset	0
INPUT	i_wr	1
OUTPUT	o_rdata	2

io_type	port_name	data
INPUT	rst	0
INPUT	i_wdata	3
INPUT	i_rd	1
OUTPUT	fifo_1_w_full	0
OUTPUT	o_rempty	0
OUTPUT	o_wfull	0
INPUT	nReset	0
INPUT	i_wr	1
OUTPUT	o_rdata	3

FUTURE WORK

Hardware wise

- Finish the I2C controller & byte controller modules
- Solder an open-drain circuit for the two signal lines (SDA & SCL)

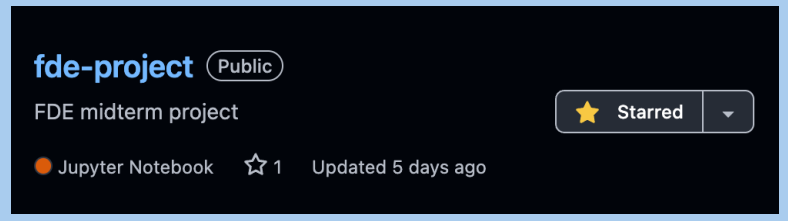
Software wise

- Finish implementing basic features for FDE-CLI
- Integrate with FDE board with a AFIFO for cross-clock domain data sharing
- Finish implementing my I2C IP with FDE-CLI
- Experiment with communicating with some real I2C modules (temperature sensors, etc.)

OPEN SOURCE

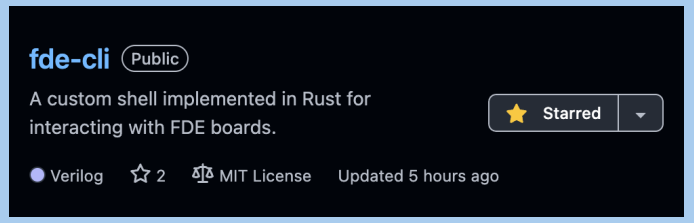
FDE Project

- https://github.com/jwwang2003/fde-project



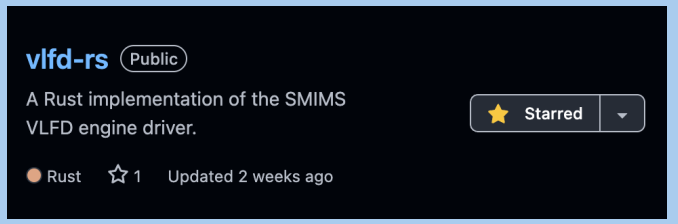
FDE-CLI

- https://github.com/jwwang2003/fde-cli



VLFD-RS

- https://github.com/jwwang2003/vlfd-rs



GitBook Blog

- https://junweiwangs-organization.gitbook.io/fpga-stuff/i2c-protocol/i2c

THANKS FOR LISTENING