AUTOTUNE

CSEE W4840: Embedded Systems Spring 2025 Report

Charlie Mei (jm5912), Amanda Lee Jenkins(alj2155), Millie Chen(sc5405), Meng Fan Wang(mw3751)

Table of Contents:

Introduction

Our project enables users to experience real-time vocal pitch correction using a digital autotune system built on an FPGA. Users sing into a digital microphone connected directly to the FPGA, which captures their voice input in real time. This audio input is processed on-board, enabling a fully embedded and self-contained autotune experience. The system is operated using a physical keyboard interface: the user initiates recording by pressing a designated key, then begins singing. Releasing the key stops the recording process.

Throughout the session, the hardware system analyzes the audio, detects any pitch inaccuracies, and adjusts the voice to match the closest musical notes using pitch-shifting algorithms implemented on the FPGA. The shifted pitch will be stored in a buffer and the software will generate a .wav file which can be played afterwards.

System Overview

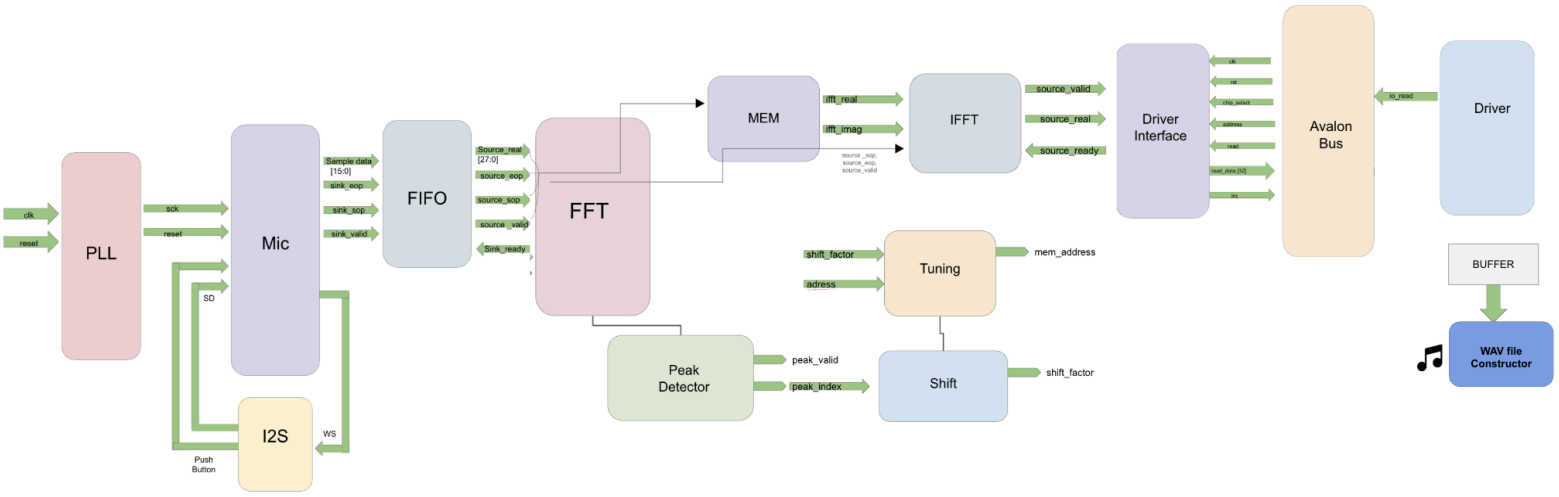


Figure 1: Total system block design showing the how the system is integrated

Our autotune system is implemented entirely on an FPGA and consists of several modular components that work together to capture, process, and playback pitch-corrected audio in real time. The diagram above outlines the end-to-end signal flow, from microphone input to speaker output, integrating digital signal processing, memory interfacing, and user interaction.

The pipeline begins with a Phase-Locked Loop (PLL), which generates the clock signal used for the microphone module and FIFO. The Microphone (Mic) module receives audio input from the user via an I2S interface, synchronized using the word select (WS) and serial clock (SCK) signals. The raw audio data is then serialized and streamed into a FIFO buffer, which ensures smooth data flow into the signal processing stage and allows passing data and outputting data under different clock frequencies. .

Once buffered, the audio data undergoes a Fast Fourier Transform (FFT) to convert the time-domain samples into their frequency-domain representation. This frequency-domain data is stored in on-chip memory (MEM) and simultaneously passed to a Peak Detector, which identifies the dominant frequency (i.e., the user's sung pitch). The peak index is then fed into the Shift module, which calculates the appropriate pitch shift factor to align the input note to the nearest intended musical note.

The calculated shift factor is sent to the Tuning module, which manages memory addressing and ensures the correct shift is applied during the inverse FFT. The processed data then passes through the Inverse FFT (IFFT) to convert it back to the time domain. This corrected signal is streamed to the Driver Interface, where it is made accessible to the HPS through a memory-mapped Avalon interface.

Finally, the Avalon Bus enables communication between the custom hardware and a Linux device driver running on the ARM processor. This driver exposes the processed audio to a userspace application, allowing users to save the output and replay it through speakers. Throughout the entire process, users interact with the system via push buttons and receive guidance from an on-screen display, completing a responsive and integrated autotuning experience.

Hardware

Our hardware part includes three major components: audio in, FFT, and Driver Interface.

The design of audio in implements a complete I2S (Inter-IC Sound) microphone interface that captures audio data, processes it, and transmits it to a FIFO through an Avalon streaming interface, and the FIFO will output audio data through another Avalon streaming interface.

Microphone and I2S Interface

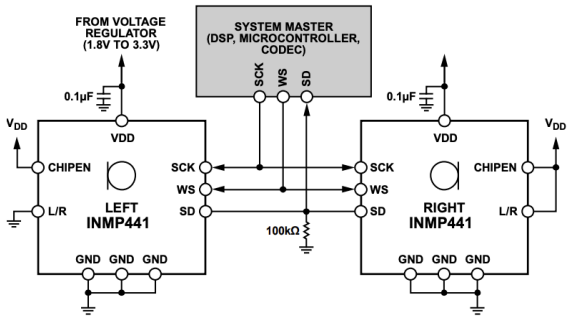


Figure 7. System Block Diagram

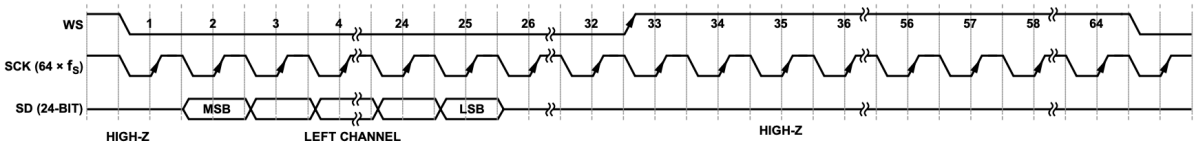


Figure 9. Mono-Output I²S Format Left Channel (L/R = 0)

TABLE 3. SERIAL DATA PORT TIMING SPECIFICATIONS

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
t_{SCH}	SCK high	50			ns	
t_{SCL}	SCK low	50			ns	
t_{SCP}	SCK period	312			ns	
f_{SCK}	SCK frequency	0.5		3.2	MHz	
t_{WSS}	WS setup	0			ns	
t_{WSH}	WS hold	20			ns	
f_{WS}	WS frequency	7.8		50	kHz	

TIMING DIAGRAM

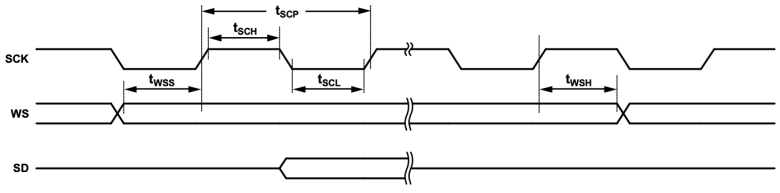


Figure 1. Serial Data Port Timing

Figure 2: I2S Data Stream Requirement and I2S Configuration(source)

We used a digital microphone chip(INMP441) for audio input. The figures above show the requirement for the input requirement for the chip and also the output data format. In order to meet the requirements, we used a PLL generated by Intel Altera Phase-Locked Loop (Altera PLL) IP Core. This PLL will use 50 MHz(system clock frequency) as reference clock(clk in the diagram below) and output a clock frequency that is 2.048 MHz. This clock is the sck clock we used for the microphone's serial data clock input.

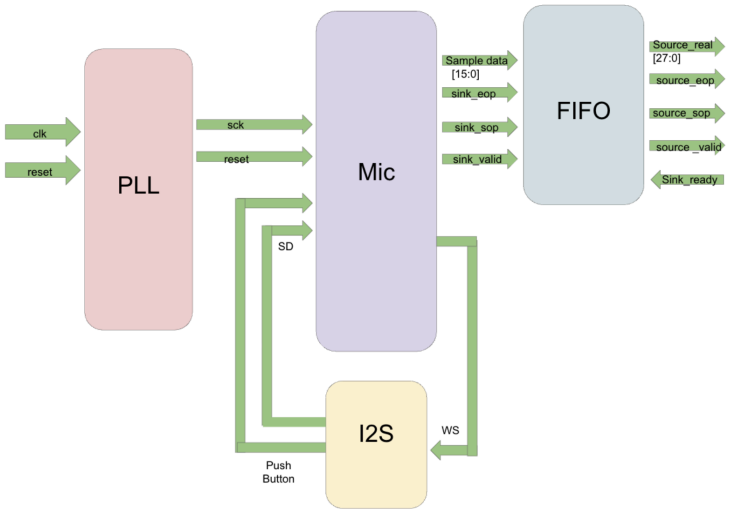


Figure 2: Microphone system block design showing the how the microphone passes data to the system

I2S interface:
The microphone module also handles all aspects of the I2S communication protocol:

1. Word Select (WS) Generation: Divides the 2.048 MHz SCK by 64 to generate a 32 kHz WS signal (alternating between left and right channels)
2. Frame Rate Control: Implements a novel approach of capturing only 1 out of every 4 frames to accommodate slower clock rates while maintaining data integrity
3. Serial Data Capture: Captures 24 bits of serial data from the microphone when WS is low (left channel)
4. Data Validation: Ensures captured data is valid before sending it downstream
5. Sample truncation: Converts 24-bit audio samples to 16-bit by taking the most significant bits

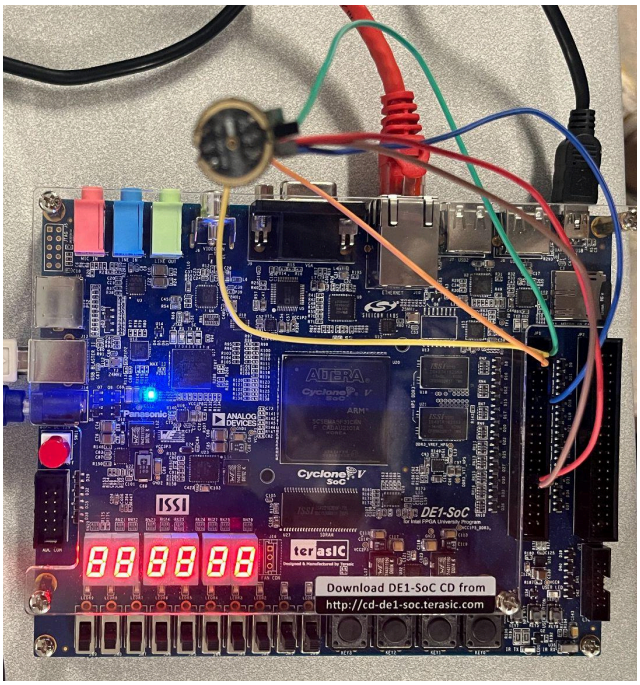


Figure 3: Microphone connected to the FPGA board

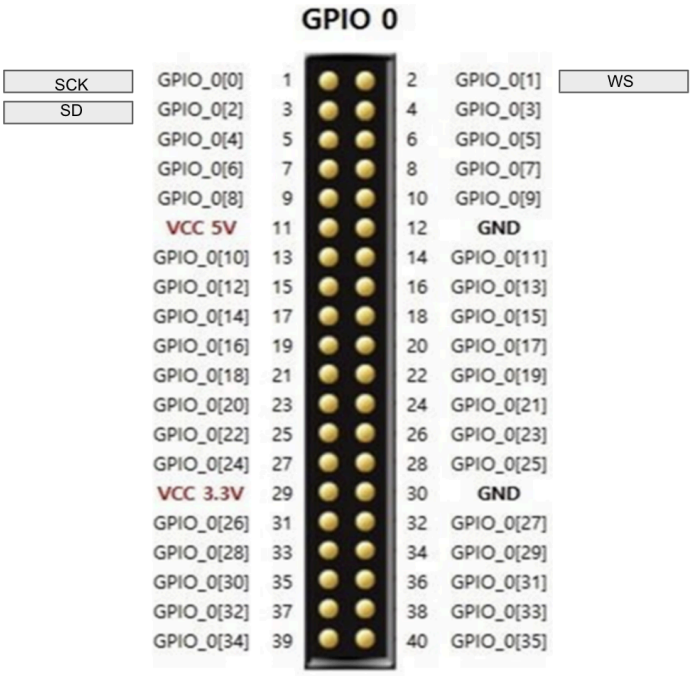


Figure 4: Gpio pin connection to the microphone

The SCK is also routed to GPIO pin 0 for external connection to the microphone, and the Gpio pin connection figure above shows how the microphone connects with the FPGA board and receives these signals. SD and WS are assigned to GPIO_0[2] and GPIO_0[3], respectively. We also have a push button signal that is when the user keeps pressing push button KEY[0], the recording will begin, and when the user releases the button, the recording will stop. We also have a HEX0 connected with i2s_hex which was assigned to the first 7 bits of sample_data.

All I2S operations are synchronized to SCK to maintain proper timing.

Avalon Streaming Interface

The microphone module outputs a complete Avalon streaming interface to FIFO:

1. sample_data [15:0]: 16-bit audio samples
2. sample_valid: Indicates when sample_data contains valid data
3. out_startofpacket: Marks the beginning of a 2048-sample packet
4. out_endofpacket: Marks the end of a 2048-sample packet

Technical Specifications

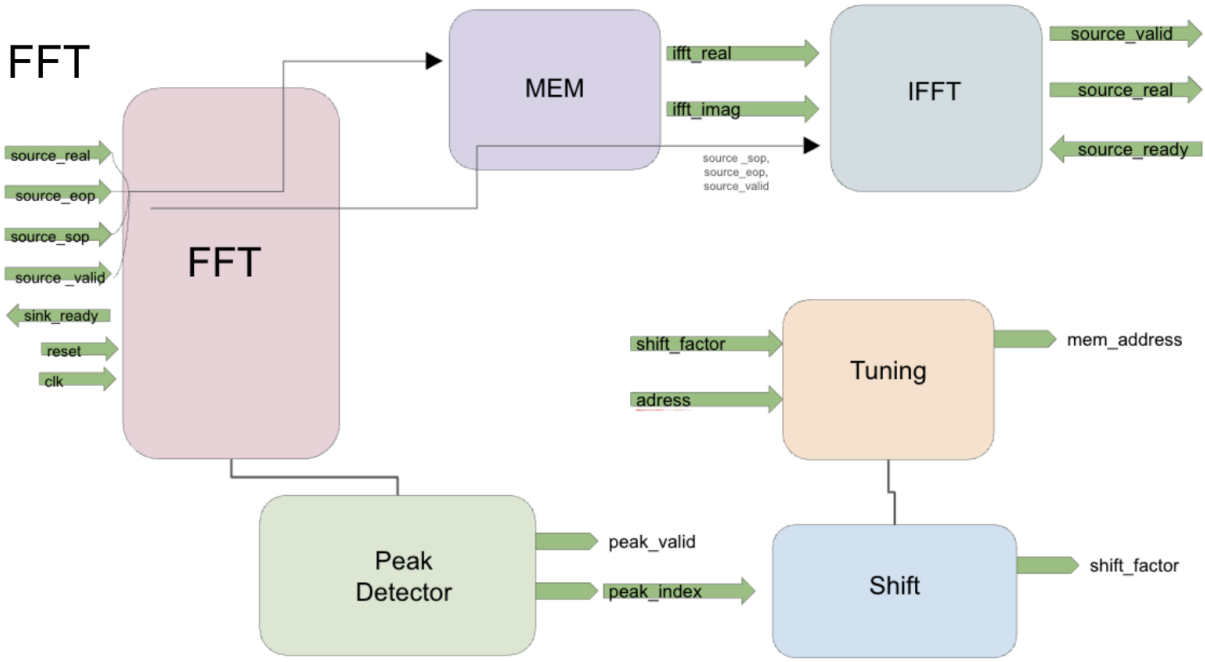
1. SCK Frequency: 2.048 MHz
2. WS Frequency: 32 kHz (SCK/64)
3. Effective Sampling Rate: 8 kHz (sampling every 4th frame)
4. Sample Resolution: 16-bit (converted from 24-bit)
5. Packet Size: 2048 samples

FIFO

We used the Intel Dual-Clock FIFO IP to buffer audio data between modules operating at different clock domains. The FIFO was configured with a 2.048 MHz input clock and a 50 MHz output clock. It receives 16-bit sample_data from the microphone module and outputs 16-bit data to the FFT module for further processing.

FFT

The core part of the project is the FFT block where the transformation of data takes place. We perform Short Time Fourier Transform (STFT) to the data samples by slitting it up into frames of 2048 data samples, compute its Frequency-Domain signals, detect its peak frequency, compute corresponding pitch shifting, and feed the data to Inverse Fast Fourier Transform. The entire diagram looks as below:



Interfaces with microphone:

The entire FFT block interacts with the microphone through Avalon-ST interface while FFT is Avalon-ST Sink and the microphone is Avalon-ST Source. Microphone will determine when to provide FFT Avalon-ST sink information so that FFT can start processing the data.

Interfaces with the driver:

The IFFT block will interact with the driver through Conduit interface The IFFT block will provide driver output data whenever it is ready. IFFT uses source_valid to indicate to the driver whether the current data is valid or not.

Peak detector:

Directed connected to FFT through its source_valid, source_sop, source_eop, will determine the index with highest magnitude. There is a module built inside a peak detector called data2mag which computes the magnitude of current data and updates the peak_index when it is necessary. Since real-time signal is symmetric magnitude in Frequency-domain. Therefore, we only need to go through the first half data output (1024 samples) to determine the peak index. Peak_valid will be high to indicate the current peak_index is valid.

Shift:

Shift module provides a quick O(1) look up table to immediately determine the amount of shifting in the frequency domain. Since every piano key groups up by a factor of 1.059. Each frequency bin has its own shift_factor so that preloading arbitrary table fasten the calculation process.

Tuning:

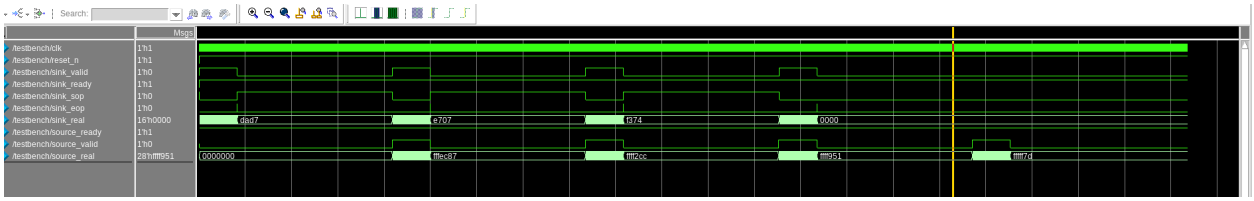
Given the shifting factor, tuning tells us where the data is located in the memory. The key idea is that we are not changing any of the data given by Intel FFT core. Instead, we need to assign them to different positions giving the shifting factor, so they will provided to IFFT in customized order.

Memory Block:

16 * 2048 MEM Block storing data processed in one transformation. Given the address, MEM will feed data back to IFFT in order of the variable “address” that comes in.

Demonstration:

Essentially, the entire FFT module is a wrapper of Intel FFT and other modules help with transforming the data. Theoretically, the latency of the data is 2048 - 4096 cycles for each 2048-point FFT transformation. Given that there are a FFT and IFFT, MEM block and other modules that compute the shifting algorithm. The delay of the output becomes 8192 cycles (4 times of the transformation size). This is also a safe choice since Intel FFT has its own internal FIFO data structures that is fragile with invalid or redundant data. Therefore, we choose to start collecting the next transformation packet when output of the last packet starts to come out.



The simulation above demonstrates. Given 8192 points, which is 4 packets of 2048-point transformation. The delay is approximately 4 times of the transformation length with input taken and output given at some time.

Driver Interface

The driver_interface module is a simple, synchronous hardware interface designed to capture a single 28-bit data value from an input stream and make it available to a memory-mapped bus. It operates on a 50 MHz clock and uses an active-high synchronous reset. The module contains a single 28-bit register (data_reg) and a flag (data_full) to track whether it currently holds valid data. When valid input data is presented (source_valid is high) and the register is not full, the module stores the input in data_reg and marks it as full. This data remains latched until a bus-side read occurs: if chip select, read, and address == 0 are asserted and the register is full, the data is zero-extended to 32 bits and sent out on read_data, and the full flag is cleared to allow new input. The source_ready signal is used to indicate whether the module is ready to accept

new data—it is high only when the register is empty. This design supports a one-sample buffer, acting as a handshake-controlled register bridge between a streaming data source and a processor-readable interface.

Driver between Software and Hardware

Device Driver

This Linux kernel module defines a simple device driver, enabling user-space programs to read audio samples from hardware registers via a memory-mapped interface.

It functions as a platform driver that binds to a compatible device defined in the Device Tree. Upon initialization (audio_init), the driver registers itself and creates a /dev/audio device using the misc_register API. The audio_probe function retrieves the physical memory region of the audio device, requests access to it, and maps it into the kernel's virtual address space using of_iomap.

The driver supports an ioctl command, AUDIO_READ_SAMPLES, which reads a 32-bit value from the audio hardware using ioread32 and transfers it to user space with copy_to_user. Cleanup routines unmap the virtual memory, release the reserved memory region, and deregister the character device. The driver also includes a Device Tree match table and is licensed under the GPL.

Software-Hardware Interface

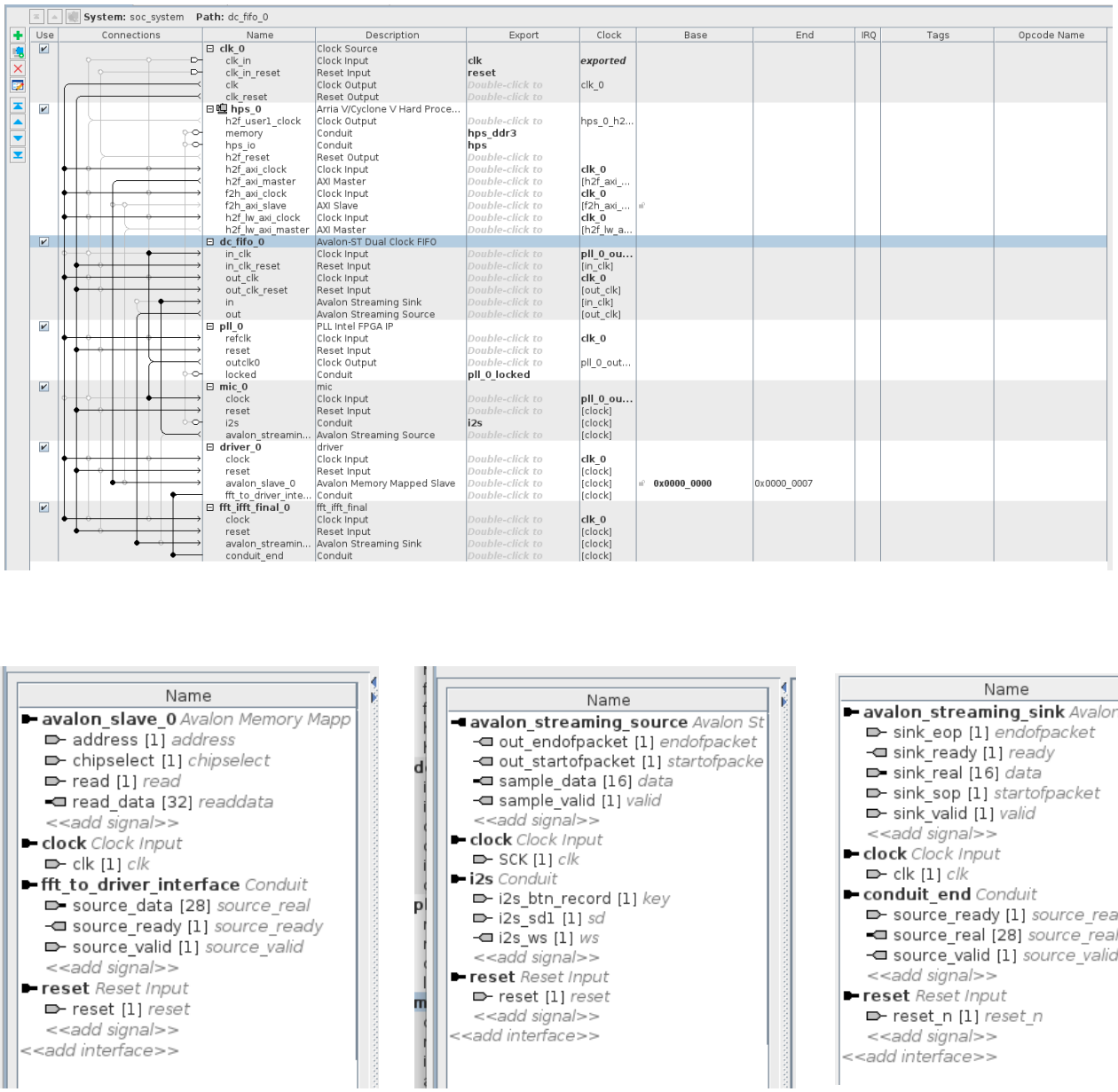


Figure 5: Platform Designer connections and configuration

The figures above show how we configured and created the conduit and interfaces between modules.

Software

User Space Program

Autotune.c

This C userspace program interacts with a kernel audio driver to read raw audio samples from the /dev/audio character device and save them into a WAV file. It first defines a 15-second buffer (BUF_SIZE) for storing 8,000 samples per second. The main function opens the audio device and continuously reads samples into a buffer using an ioctl call with the AUDIO_READ_SAMPLES command, which communicates with the underlying kernel module.

The helper function read_samples performs a single ioctl call, checks for errors, and appends the resulting data to the buffer. After the buffer is filled, the program uses a custom write_wav function (defined in make_wav.h) to write the collected samples to a file named anonymous_audio.wav. This program effectively captures 15 seconds of audio from the kernel driver and saves it in standard WAV format for playback or further processing.

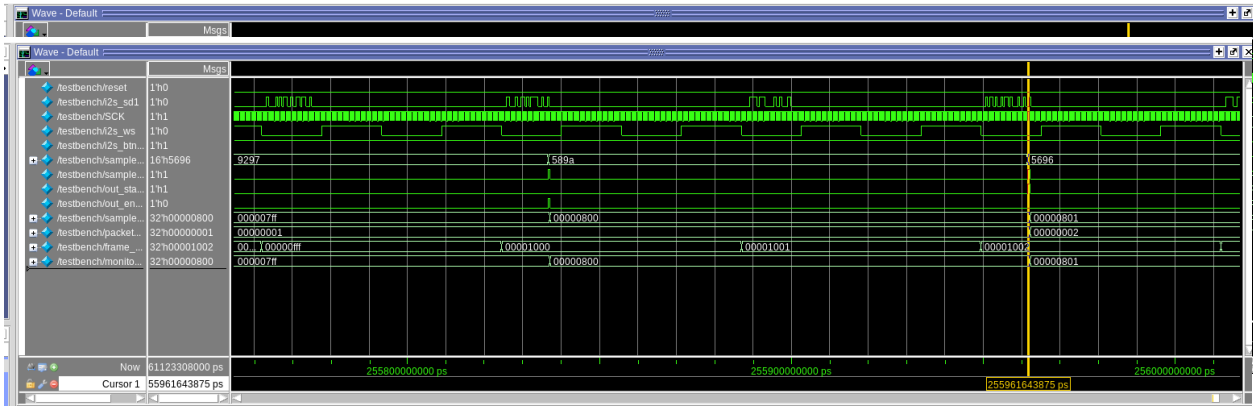
make_wav.c

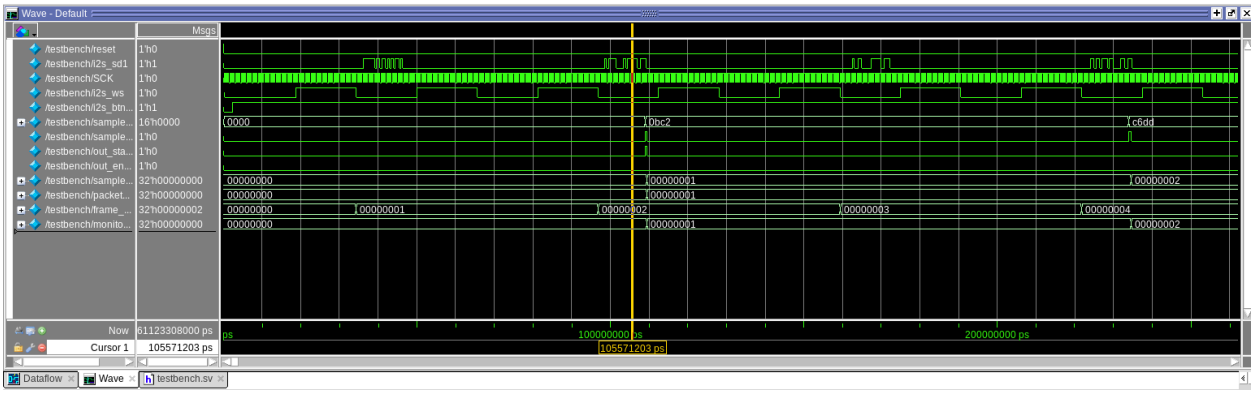
We referenced the previous project, NoCallerID for the make_wav.c. This C program provides a utility to generate a WAV audio file from an array of integers representing audio samples. The output file uses 28-bit signed samples (4 bytes per sample), formatted as mono audio despite a comment mentioning stereo. The key function, write_wav, constructs a valid WAV file conforming to the standard format defined by the RIFF specification. It writes the necessary headers—including “RIFF”, “fmt “, and “data” chunks—with appropriate metadata such as sample rate, byte rate, and block alignment.

The core helper function, write_little_endian, writes multi-byte values in little-endian byte order, which is required by the WAV format. The sample data is written by looping through the array and outputting each sample with 4-byte precision. The program ensures that the file is properly opened and closed, and the use of assert guarantees the output file was created successfully. This utility is useful for converting raw audio data collected in an application into a playable WAV format file.

Testing & Results

Microphone module test and result:





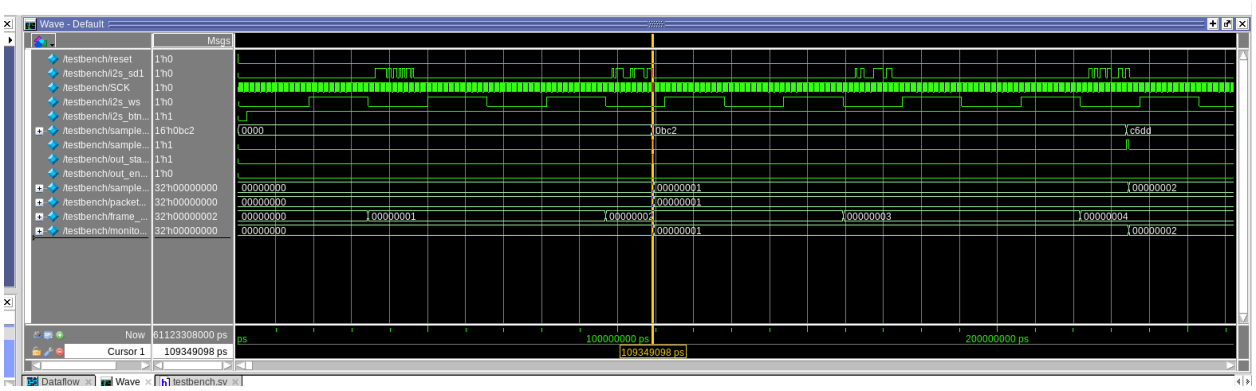


Figure 6: Waveforms from the microphone's testbench

1. Initialization Phase (0-100ns)

- Reset is active (high)
- All signals except SCK are at their default/reset values
- No data processing occurs

2. Reset Release Phase (~100ns)

- Reset goes low
- SCK continues toggling
- Internal counters begin to operate

3. WS Stabilization Phase

- First 10 SCK cycles after recording starts
- i2s_ws begins its proper toggling pattern
- No samples processed yet

4. Active Recording Phase

The waveform shows repeating patterns of:

- Frame Pattern: WS toggles every 32 SCK cycles
- Sampling Pattern: Every 4th frame, you see activity during the low period of WS:
  - 24 bits of serial data on i2s_sd1
  - Followed by a sample_valid pulse
  - sample_data updates with new value

- Capture Windows: During the low period of WS when frame_cnt = 3:
  - 24 SCK cycles of active data shifting
  - 7 SCK cycles of idle time
  - sample_valid goes high briefly after the 24th bit is captured

5. Packet Boundary Events

- At the very first valid sample: out_startofpacket pulses high
- At sample 2048: out_endofpacket pulses high
- At sample 2049: out_startofpacket pulses high again (start of second packet)

Driver Interface

To verify the functionality of our driver interface module, we created a Verilog testbench (driver_interface_tb.sv) that simulates typical read and write interactions between the FPGA hardware and the HPS over a memory-mapped Avalon interface. The test focused on ensuring proper synchronization, handshaking, and data availability during valid transactions.

The testbench instantiates the driver_interface module and drives its inputs with a 50 MHz clock. It initializes the system with a brief reset sequence, then simulates a set of controlled interactions, including:

- Sending a valid 28-bit data sample (0x1234567) using the source_valid signal.
- Reading back the buffered data by asserting chipselect, read, and address.
- Sending a second sample (0xABCDEF0) and verifying proper FIFO behavior.
- Ensuring handshaking logic (source_ready) reacts correctly to internal FIFO full/empty status.

The testbench includes assertions to guarantee the correctness of the source_ready signal based on FIFO state: assert property (source_ready_when_empty) else $error("source_ready should be 1 when data_full is 0"); assert property (source_ready_when_full) else $error("source_ready should be 0 when data_full is 1");

The simulation was run in ModelSim, and the waveform results confirm the correctness of the interface behavior. In Figure X, the source_data signal transitions through values 0x0000000, 0x1234567, and 0xABCDEF0, and the corresponding read_data reflects the correct values after being read. In Figure Y, we verify the proper behavior of control signals: chipselect, read, address, and source_valid all align with the intended timing of the transactions. The

source_ready signal correctly goes low when the FIFO is full and high when ready to accept new data. These results confirm that the driver interface performs as expected under realistic operation, and that it can successfully support read transactions over Avalon-MM with proper data handoff.

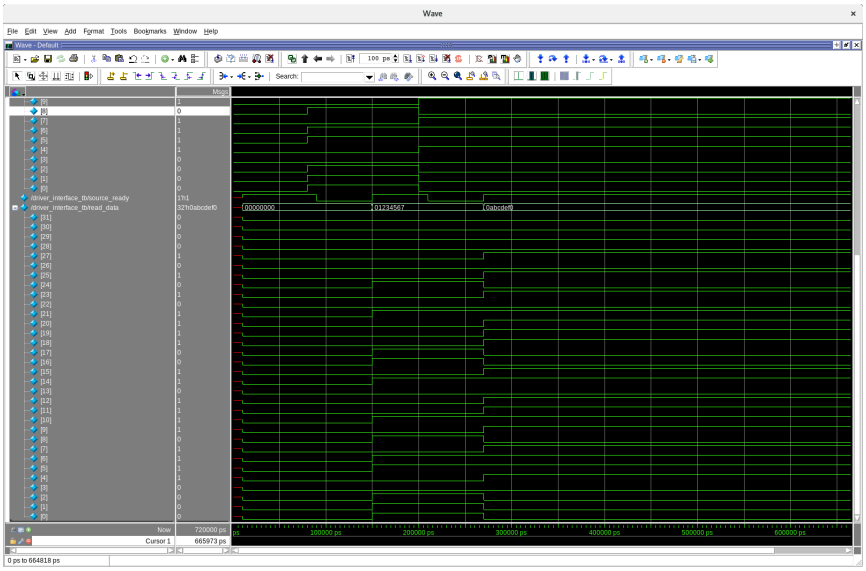


Figure X: Driver Interface TestBench in Modelsim (1)

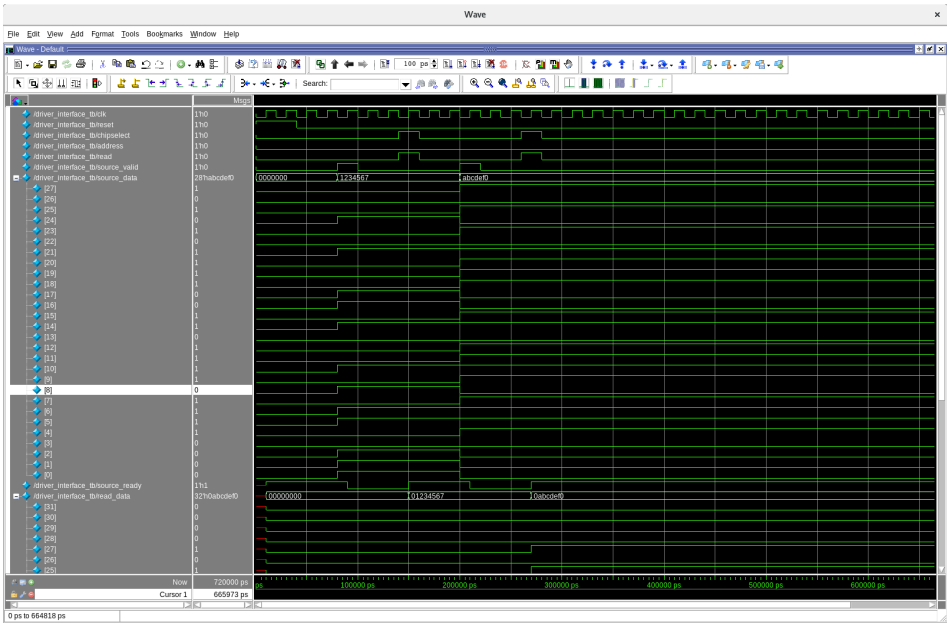


Figure Y: Driver Interface TestBench in Modelsim (2)

Challenges

One of the primary challenges we encountered was successfully integrating all custom and IP blocks within Platform Designer to form a cohesive signal processing pipeline. Each block had specific input/output timing requirements and interface protocols, and ensuring consistent handshaking and data flow across modules required careful attention to signal validity, clock domains, and stream alignment.

A significant hurdle arose in the configuration of the Audio In (Microphone) block. We had to correctly generate and route the serial clock (SCLK) and word select (WS) signals via the I2S interface to ensure they were properly synchronized with the microphone's data stream. This challenge resulted in our software not being able to properly read the data values from our hardware.

Connecting the FFT, peak detection, shift logic, and IFFT modules introduced another layer of complexity. These components use streaming Avalon-ST interfaces with control signals like valid, ready, start-of-packet (SOP), and end-of-packet (EOP). Ensuring these signals were correctly propagated across blocks, and that each module processed complete 2048-sample frames, required iterative simulation and waveform inspection to catch off-by-one errors or incorrect timing. Additionally, coordinating memory access between the tuning logic, FFT output, and inverse FFT input via the shared memory interface introduced synchronization challenges. We had to carefully manage memory addressing and implement valid-ready signaling to avoid race conditions or data corruption.

Also, bridging the custom hardware logic with the HPS (ARM core) via the Avalon-MM bus and a custom Linux device driver required consistent register mapping and memory allocation. Mapping the FPGA memory correctly in the driver and ensuring seamless communication between userspace and kernel space took multiple test iterations and adjustments to the ioread32() logic and ioctl interface.

When we try to compile our hardware files by running *make quartus*, we get these errors, which means our logic is too complicated and the hardware resource is not enough. So the other challenge for our project is it is hard to accurately estimate the hardware budget, and how to simplify our design to meet the limitation.

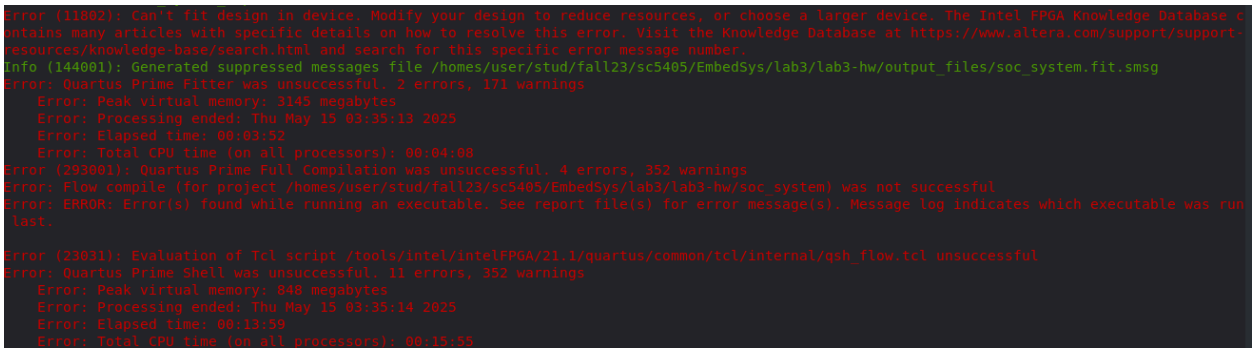


Figure 7: Compiling error

Future Work

One major area for future improvement is real-time autotuning. In our current design, pitch correction is applied only after a complete recording session. Extending the system to perform pitch detection, shifting, and playback in a continuous loop would enable true real-time feedback, which is critical for live performance applications. Achieving this would require tighter synchronization between the FFT, shift logic, and IFFT modules, as well as efficient buffering and latency optimization throughout the pipeline.

Another potential enhancement is multi-channel audio support. Our system currently processes mono audio captured from a single microphone. Supporting stereo or multiple input sources would allow more complex use cases such as group singing, vocal layering, or spatial effects. This would require modifying the microphone interface and expanding the internal buffering and processing capabilities to handle multiple audio streams simultaneously.

We also see opportunities to improve frequency resolution and tuning flexibility. Currently, the tuning logic maps detected frequencies to the nearest musical note based on a fixed scale. Future versions could include user-selectable keys or scales (e.g., major, minor, chromatic), enabling a more expressive and customizable pitch correction experience. This could be supported through additional UI elements or software controls exposed via the userspace application.

From a usability standpoint, adding visualizations and waveform feedback on the display would help users better understand how their voice was processed. For instance, a real-time spectrum display or pitch-tracking graph could give users insight into how their vocals were interpreted and tuned.

We will also need to reduce the complexity of our design to meet the hardware requirements.

Individual Contributions & Advice for Future Projects

Meng Fan Wang (mw3751)

I contributed to both the hardware and software interfaces of the project. Specifically, I developed the driver interface and its corresponding testbench, and I also worked extensively on the kernel module and userspace application. While my primary focus was integrating various software components, I was also involved in debugging code across all sections of the project.

One key piece of advice I would offer to future teams is to begin integrating the full system as early as possible. While working on isolated components is manageable, the real challenge lies in ensuring that all parts work together seamlessly. This broader perspective makes it easier to collaborate, debug, and find solutions more efficiently.

Amanda Jenkins (alj2155)

I contributed to the hardware software components of the project. I worked on developing and debugging the driver interface and supported the creation of its testbench. I was also involved in building the kernel module and ensuring that audio data could be successfully passed to the userspace application for processing and playback. My contributions spanned both system-level integration and low-level data handling, with a particular focus on connecting the hardware to the software.

One key piece of advice I would offer to future teams is to prioritize end-to-end testing early on. It's easy to spend time perfecting individual modules, but the real test lies in how these modules interact. By testing full data flow from the microphone to the speakers early in the process, you can uncover timing issues, data formatting mismatches, or integration bugs sooner which saves significant time and frustration down the line.

Millie(sc5405)

I was responsible for the entire audio input pipeline in the design. This included selecting and purchasing the digital microphone chip, as well as soldering it with pin headers for prototyping. I wrote the SystemVerilog code for the microphone module and developed a testbench to verify its functionality in simulation.

I also handled all hardware debugging throughout the project. This involved diagnosing signal issues, verifying timing behavior, and using tools such as Modelsim to troubleshoot data flow between modules.

In Platform Designer, I generated the PLL and Dual-Clock FIFO IPs and managed all interface designs. This included creating custom conduits, configuring Avalon Streaming interfaces, and modifying soc_system_top.sv to correctly pass signals between the microphone and the rest of the system.

For future groups, start early, especially with hardware components. Hardware design comes with many implicit requirements, such as correctly setting up Avalon Streaming sinks and sources, that can be difficult to identify and test. Debugging hardware is more complex than debugging software and often requires long compile times.

Be mindful of your FPGA's resource limits early on. Once modules are integrated, making major structural changes can be very difficult. Planning your design with modularity and resource constraints in mind will help avoid major issues later in the project.

Charlie(jm5912)

I focus on actual data transformation via using Intel FFT packets. A huge part of the task is to understand how Intel FFT behaves by trying different testbenches, figuring out what kind of mistakes will lead to fatal errors (yes! They use this word). I also focus on designing the inner interface between the FFT and IFFT module so that pitch will be shifted accurately and data is carefully transferred in between to avoid any errors. I also work on developing interfaces between microphone and device. We cooperate by telling each other what kind of the data, what format of the data we need to lead to a smooth transition before my FFT receives data or when my IFFT is ready to release data.

For future, I would definitely say to try to start it early. It takes way more than expected time to understand the idea and make it work on the FPGA board. There is just a lot of complication and variation that every tiny bit error would likely going to cause compilation failure. It is good to know that debugging skill is very important but if given enough time I believe would not get enough practice

References

- https://www.cs.columbia.edu/~sedwards/classes/2023/4840-spring/designs/NoCallerID.pdf
- https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/lab3.pdf
- http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/de1-soc_user_manual.pdf
- https://www.intel.com/content/www/us/en/docs/programmable/683374/17-1/dsp-ip-core-features.html

Code Appendix

mic.sv

```
module mic (
    input  logic        reset,        // Reset signal
    input  logic        i2s_sd1,          // Microphone data output
    input  logic        i2s_btn_record,   // Active-high button to control recording
    input  logic        SCK,          // Serial clock for mic (2048kHz)
    output logic     [6:0]   i2s_hex,
    output logic        i2s_ws,           // Word select for mic (512kHz)
    output logic [15:0] sample_data,          // Output to FIFO
    output logic        sample_valid,  // Sample valid signal
    output logic        out_startofpacket, // Start of packet signal
    output logic        out_endofpacket,    // End of packet signal
    output logic        sck_new
);

    // Internal signals
    logic [5:0]   clk_cnt;           // 64 counter to generate i2s_ws signal
    logic [23:0]  shift_reg;         // Shift register for incoming data
    logic [15:0]  sample;            // Truncated sample
    logic         wr_en;             // Write enable (sample ready)
    logic         recording;         // Are we currently recording?
    logic         sck_rst;           // SCK reset signal
    logic         start_capture;     // Signal to start capturing data
    logic [1:0]   frame_cnt;         // Counter for i2s_ws frames
    logic [10:0]  packet_cnt;        // Counter for packet boundaries (0-2047)

    assign sck_new = SCK;
```

```
assign i2s_hex = sample_data [6:0];

 // Control recording state
 always_ff @(posedge SCK or posedge reset) begin
    if (reset) begin
        recording <= 0;
        sck_rst <= 1;
    end else begin
        recording <= i2s_btn_record;  // Direct button control

        // Reset SCK counter when not recording
        if (!recording)
            sck_rst <= 1;
        else
            sck_rst <= 0;
    end
 end

 // i2s_ws clock generator (64 division) and start_capture control
 always_ff @(posedge SCK) begin
    if (sck_rst) begin
        clk_cnt <= 6'd0;
        i2s_ws <= 0;
        start_capture <= 0;
        frame_cnt <= 0;
    end else begin
        clk_cnt <= clk_cnt + 6'd1;

        if (clk_cnt == 31) begin
            i2s_ws <= 1;
        end else if (clk_cnt == 63) begin
            i2s_ws <= 0;
        end

        // At the beginning of each frame (clk_cnt == 0), increment frame counter
        if (clk_cnt == 0) begin
            frame_cnt <= frame_cnt + 2'd1;

            // Only capture once every 4 i2s_ws frames
```

```
            if (frame_cnt == 2'd3) begin
                start_capture <= 1;
            end else begin
                start_capture <= 0;
            end
        end else if (clk_cnt == 24) begin
            start_capture <= 0;
        end
    end
end

// I2S data capture (shift in on rising edge of SCK)
always_ff @(posedge SCK) begin
    if (sck_rst) begin
        shift_reg <= 0;
        wr_en <= 0;
    end else begin
        // Capture data when start_capture is high and i2s_ws is low
        if (start_capture && !i2s_ws) begin
            shift_reg <= {shift_reg[22:0], i2s_sd1};
            if (clk_cnt == 23) begin  // After 24 bits are captured
                // Only set wr_en if we have valid data (check if shift_reg is not all zeros)
                if (shift_reg != 0) begin
                    sample <= shift_reg[23:8];
                    wr_en <= 1;
                end else begin
                    wr_en <= 0;
                end
            end else begin
                wr_en <= 0;
            end
        end else begin
            wr_en <= 0;
        end
    end
end

// Output sample data to FIFO
always_ff @(posedge SCK or posedge reset) begin
```

```
        if (reset) begin
            sample_data <= 0;
            sample_valid <= 0;
            out_startofpacket <= 0;
            out_endofpacket <= 0;
            packet_cnt <= 0;
        end else if (recording) begin
            if (wr_en) begin
                sample_data <= sample;
                sample_valid <= 1;

                // Packet boundary logic
                if (packet_cnt == 0) begin
                    out_startofpacket <= 1;
                    out_endofpacket <= 0;
                    packet_cnt <= packet_cnt + 1;  // Increment counter here
                end else if (packet_cnt == 2047) begin
                    out_startofpacket <= 0;
                    out_endofpacket <= 1;
                    packet_cnt <= 0;  // Reset counter for next period
                end else begin
                    out_startofpacket <= 0;
                    out_endofpacket <= 0;
                    packet_cnt <= packet_cnt + 1;
                end
            end else begin
                sample_valid <= 0;
                out_startofpacket <= 0;
                out_endofpacket <= 0;
            end
        end else begin
            sample_valid <= 0;
            out_startofpacket <= 0;
            out_endofpacket <= 0;
            packet_cnt <= 0;  // Reset counter when not recording
        end
    end

endmodule
```

soc_system_top.sv

```
module soc_system_top(

 ///////// ADC /////////
 inout         ADC_CS_N,
 output        ADC_DIN,
 input         ADC_DOUT,
 output        ADC_SCLK,

 ///////// AUD /////////
 input         AUD_ADCDAT,
 inout         AUD_ADCLRCK,
 inout         AUD_BCLK,
 output        AUD_DACDAT,
 inout         AUD_DACLRCK,
 output        AUD_XCK,

 ///////// CLOCK2 /////////
 input         CLOCK2_50,

 ///////// CLOCK3 /////////
 input         CLOCK3_50,

 ///////// CLOCK4 /////////
 input         CLOCK4_50,
```

```
//////// CLOCK ////////
input       CLOCK_50,

//////// DRAM ////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

//////// FAN ////////
output        FAN_CTRL,

//////// FPGA ////////
output        FPGA_I2C_SCLK,
inout         FPGA_I2C_SDAT,

//////// GPIO ////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

//////// HEX0 ////////
output [6:0]  HEX0,

//////// HEX1 ////////
output [6:0]  HEX1,

//////// HEX2 ////////
output [6:0]  HEX2,

//////// HEX3 ////////
output [6:0]  HEX3,
```

```
//////// HEX4 ////////
output [6:0]  HEX4,

//////// HEX5 ////////
output [6:0]  HEX5,

//////// HPS ////////
inout        HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
output        HPS_DDR3_CAS_N,
output        HPS_DDR3_CKE,
output        HPS_DDR3_CK_N,
output        HPS_DDR3_CK_P,
output        HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output        HPS_DDR3_ODT,
output        HPS_DDR3_RAS_N,
output        HPS_DDR3_RESET_N,
input         HPS_DDR3_RZQ,
output        HPS_DDR3_WE_N,
output        HPS_ENET_GTX_CLK,
inout         HPS_ENET_INT_N,
output        HPS_ENET_MDC,
inout         HPS_ENET_MDIO,
input         HPS_ENET_RX_CLK,
input [3:0]   HPS_ENET_RX_DATA,
input         HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output        HPS_ENET_TX_EN,
inout         HPS_GSENSOR_INT,
inout         HPS_I2C1_SCLK,
inout         HPS_I2C1_SDAT,
inout         HPS_I2C2_SCLK,
inout         HPS_I2C2_SDAT,
inout         HPS_I2C_CONTROL,
```

```
inout        HPS_KEY,
inout        HPS_LED,
inout        HPS_LTC_GPIO,
output       HPS_SD_CLK,
inout        HPS_SD_CMD,
inout [3:0]  HPS_SD_DATA,
output       HPS_SPIM_CLK,
input        HPS_SPIM_MISO,
output       HPS_SPIM_MOSI,
inout        HPS_SPIM_SS,
input        HPS_UART_RX,
output       HPS_UART_TX,
input        HPS_USB_CLKOUT,
inout [7:0]  HPS_USB_DATA,
input        HPS_USB_DIR,
input        HPS_USB_NXT,
output       HPS_USB_STP,

///////// IRDA /////////
input        IRDA_RXD,
output       IRDA_TXD,

///////// KEY /////////
input [3:0]  KEY,

///////// LEDR /////////
output [9:0] LEDR,

///////// PS2 /////////
inout        PS2_CLK,
inout        PS2_CLK2,
inout        PS2_DAT,
inout        PS2_DAT2,

///////// SW /////////
input [9:0]  SW,

///////// TD /////////
input        TD_CLK27,
```

```
 input [7:0]   TD_DATA,
 input         TD_HS,
 output        TD_RESET_N,
 input         TD_VS

);

logic SCK;

   soc_system soc_system0(
     .clk_clk                       ( CLOCK_50 ),
     .reset_reset_n                 ( 1'b1 ),

     .hps_ddr3_mem_a                ( HPS_DDR3_ADDR ),
     .hps_ddr3_mem_ba               ( HPS_DDR3_BA ),
     .hps_ddr3_mem_ck               ( HPS_DDR3_CK_P ),
     .hps_ddr3_mem_ck_n             ( HPS_DDR3_CK_N ),
     .hps_ddr3_mem_cke              ( HPS_DDR3_CKE ),
     .hps_ddr3_mem_cs_n             ( HPS_DDR3_CS_N ),
     .hps_ddr3_mem_ras_n            ( HPS_DDR3_RAS_N ),
     .hps_ddr3_mem_cas_n            ( HPS_DDR3_CAS_N ),
     .hps_ddr3_mem_we_n             ( HPS_DDR3_WE_N ),
     .hps_ddr3_mem_reset_n          ( HPS_DDR3_RESET_N ),
     .hps_ddr3_mem_dq               ( HPS_DDR3_DQ ),
     .hps_ddr3_mem_dqs              ( HPS_DDR3_DQS_P ),
     .hps_ddr3_mem_dqs_n            ( HPS_DDR3_DQS_N ),
     .hps_ddr3_mem_odt              ( HPS_DDR3_ODT ),
     .hps_ddr3_mem_dm               ( HPS_DDR3_DM ),
     .hps_ddr3_oct_rzqin            ( HPS_DDR3_RZQ ),

     .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
     .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
     .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
     .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
     .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
     .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
     .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
     .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
     .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
```

```
.hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
.hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
.hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
.hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
.hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

.hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD         ),
.hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]     ),
.hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]     ),
.hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK         ),
.hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]     ),
.hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]     ),

.hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]    ),
.hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]    ),
.hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]    ),
.hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]    ),
.hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]    ),
.hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]    ),
.hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]    ),
.hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]    ),
.hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT     ),
.hps_hps_io_usb1_inst_STP     ( HPS_USB_STP        ),
.hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR        ),
.hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT        ),

.hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
.hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
.hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
.hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

.hps_hps_io_uart0_inst_RX     ( HPS_UART_RX    ),
.hps_hps_io_uart0_inst_TX     ( HPS_UART_TX    ),

.hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT    ),
.hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK    ),

.hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT    ),
.hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK    ),
```

```
    .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
    .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
    .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

    .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
    .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
    .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
    .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),


    .i2s_ws      (GPIO_0[1]),
    .i2s_sd      (GPIO_0[2]),
    .i2s_key   (KEY[0]),
    .i2s_sck_new (SCK),
    .i2s_hex    (HEX0)

);

  assign GPIO_0[0] = SCK;

  // The following quiet the "no driver" warnings for output
  // pins and should be removed if you use any of these peripherals

  assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
  assign ADC_DIN = SW[0];
  assign ADC_SCLK = SW[0];

  assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
  assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
  assign AUD_DACDAT = SW[0];
  assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
  assign AUD_XCK = SW[0];

  assign DRAM_ADDR = { 13{ SW[0] } };
  assign DRAM_BA = { 2{ SW[0] } };
  assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : { 16{ 1'bZ } };
  assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
```

```
            DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = {
8{SW[0]} };

  assign FAN_CTRL = SW[0];

  assign FPGA_I2C_SCLK = SW[0];
  assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

  assign GPIO_0[35:5] = SW[1] ? { 31{ SW[0] } } : 31'bZ;
  assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : { 36{ 1'bZ } };

  //assign HEX0 = { 7{ SW[1] } };
  assign HEX1 = { 7{ SW[2] } };
  assign HEX2 = { 7{ SW[3] } };
  assign HEX3 = { 7{ SW[4] } };
  assign HEX4 = { 7{ SW[5] } };
  assign HEX5 = { 7{ SW[6] } };

  assign IRDA_TXD = SW[0];

  assign LEDR = { 10{SW[7]} };

  assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
  assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
  assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
  assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

  assign TD_RESET_N = SW[0];


endmodule
```

driver_interface.sv

```
`timescale 1ns/1ps

module driver_interface #(
   parameter int DATA_SIZE = 28
)(
   // ——— Bus Output
————————————————————————————————————————————————————————————————————————————————
————
   input  logic                clk,       // 50 MHz
   input  logic                reset,     // synchronous active-high
   input  logic                chipselect,
   input  logic                address,   // only bit-0 used
   input  logic                read,
   output logic [31:0]           read_data,

   // ——— FFT input
————————————————————————————————————————————————————————————————————————————————
————————
   input  logic                source_valid,
   input  logic [DATA_SIZE-1:0]   source_data,
   output logic                 source_ready
);

   // Single data register
   logic [DATA_SIZE-1:0] data_reg;
```

```
    logic data_full;

    // ——— Data handling
————————————————————————————————————————————————————————————————————————
————
    always_ff @(posedge clk) begin
        if (reset) begin
            data_reg <= '0;
            data_full <= 1'b0;
            read_data <= '0;
        end else begin
            if (source_valid && !data_full) begin
                data_reg <= source_data;
                data_full <= 1'b1;
            end else if (chipselect && read && data_full && (address == 1'b0)) begin
                read_data <= {{32-DATA_SIZE{1'b0}}, data_reg};
                data_full <= 1'b0;
            end
        end
    end

    // ——— Ready signal to producer
————————————————————————————————————————————————————————————————————
    assign source_ready = !data_full;  // ready when no data in our register

endmodule
```

fft_ifft_final.sv:

```
module fft_ifft_peak(
    input logic clk, // clock
    input logic reset_n, // reset
    input logic sink_valid, // valid
    output logic sink_ready, // ready
    input logic sink_sop, // start of packet
    input logic sink_eop, // end of packet
    input logic [15:0] sink_real, // real part of input
    output logic source_valid, // valid
    input logic source_ready, // ready
    // output logic source_sop, // start of packet
    // output logic source_eop, // end of packet
    output logic [27:0] source_real
    // output logic [9:0] peak_index,
    // output logic peak_valid,
    // output logic [5:0] shift_factor,
    // output logic [15:0] ifft_sink_real,
    // output logic [15:0] ifft_sink_imag
    // output logic [10:0] mem_address,
    // output logic [10:0] shift_address,
    // output logic ifft_sink_valid,
    // output logic ifft_sink_sop,
    // output logic ifft_sink_eop
);

    // constants
    logic [15:0] sink_imag = 0;

    // fft's internal signals
    logic fft_source_sop;
    logic fft_source_eop;
    logic fft_source_valid;
    logic [27:0] fft_source_real;
    logic [27:0] fft_source_imag;
    logic [1:0] fft_source_error;
    logic [11:0] fft_fftpts_out;
```

```
logic fft_source_ready;

// ifft's internal signals
logic [15:0] ifft_sink_real;
logic [15:0] ifft_sink_imag;
logic ifft_sink_valid;
logic ifft_sink_sop;
logic ifft_sink_eop;
logic ifft_sink_ready;
logic [11:0] ifft_fftpts_out;
logic [1:0] ifft_source_error;
logic [27:0] ifft_source_imag;
logic source_sop;
logic source_eop;

// mem_block's internal signals
logic [15:0] mem_block_sink_real;
logic [15:0] mem_block_sink_imag;

logic [10:0] shift_address;
logic [10:0] mem_address;

logic [9:0] peak_index;
logic peak_valid;
logic [5:0] shift_factor;

logic reading;
logic previous_peak_valid = 0;

convert_28_to_16 convert_28_to_16_inst(
    .source_real(fft_source_real),
    .source_imag(fft_source_imag),
    .sink_real(mem_block_sink_real),
    .sink_imag(mem_block_sink_imag)
);

mem_block mem_block_inst(
    .clk(clk),
    .reset_n(reset_n),
```

```
    .source_real(mem_block_sink_real),
    .source_imag(mem_block_sink_imag),
    .address(mem_address),
    .source_valid(fft_source_valid),
    .source_sop(fft_source_sop),
    .source_eop(fft_source_eop),
    .sink_real(ifft_sink_real),
    .sink_imag(ifft_sink_imag)
);

peak_detector peak_detector_inst(
    .clk(clk),
    .reset_n(reset_n),
    .source_real(fft_source_real),
    .source_imag(fft_source_imag),
    .sink_sop(fft_source_sop),
    .sink_eop(fft_source_eop),
    .sink_valid(fft_source_valid),
    .peak_index(peak_index),
    .source_valid(peak_valid)
);

shift shift_inst(
    .peak_index(peak_index),
    .shift_factor(shift_factor)
);

tuning tuning_inst(
    .shift_address(shift_address),
    .shift_factor(shift_factor),
    .mem_address(mem_address)
);

// Instantiate the FFT module
my_fft fft_inst(
    .clk(clk),
    .reset_n(reset_n),
    .sink_valid(sink_valid),
    .sink_ready(sink_ready), //
```

```
    .sink_error(2'b0), // maybe just set it to 0
    .sink_sop(sink_sop),
    .sink_eop(sink_eop),
    .sink_real(sink_real),
    .sink_imag(sink_imag),
    .fftpts_in(12'd2048),
    .inverse(1'b0),
    .source_valid(fft_source_valid), // wire into mem_block's source_valid
    .source_ready(ifft_sink_ready),
    .source_error(fft_source_error),
    .source_sop(fft_source_sop), // wire to mem_block's sink_sop
    .source_eop(fft_source_eop), // wire to mem_block's sink_eop
    .source_real(fft_source_real), // wire to 28_to_16's source_real
    .source_imag(fft_source_imag), // wire to 28_to_16's source_imag
    .fftpts_out(fft_fftpts_out) // wire to ifft's fftpts_in
);

// Instantiate the IFFT module
my_fft ifft_inst(
    .clk(clk),
    .reset_n(reset_n),
    .sink_valid(ifft_sink_valid), // wire from fft's source_valid
    .sink_ready(ifft_sink_ready), // wire to fft's source_ready
    .sink_error(fft_source_error), // wire from fft's source_error
    .sink_sop(ifft_sink_sop), // wire from fft's source_sop
    .sink_eop(ifft_sink_eop), // wire from fft's source_eop
    .sink_real(ifft_sink_real), // wire from 28_to_16's sink_real
    .sink_imag(ifft_sink_imag), // wire from 28_to_16's sink_imag
    .fftpts_in(12'd2048), // wire from fft's fftpts_out
    .inverse(1'b1),
    .source_valid(source_valid),
    .source_ready(source_ready), // can be assumed to be 1 testbench
    .source_error(ifft_source_error),
    .source_sop(source_sop),
    .source_eop(source_eop),
    .source_real(source_real),
    .source_imag(ifft_source_imag),
    .fftpts_out(ifft_fftpts_out)
);
```

```
always_ff @(posedge clk) begin
    if (!reset_n) begin
        reading <= 0;
        previous_peak_valid <= 0;
        shift_address <= 0;
        ifft_sink_valid <= 0;
        ifft_sink_sop <= 1;
        ifft_sink_eop <= 0;
    end
    else begin
        if (previous_peak_valid != peak_valid && peak_valid) begin
            reading <= 1;
        end
        else if (previous_peak_valid != peak_valid) begin
            reading <= 0;
            previous_peak_valid <= peak_valid;
        end
        if (reading && ifft_sink_ready && peak_valid) begin
            if (shift_address == 0) begin
                ifft_sink_valid <= 1;
                shift_address <= shift_address + 1;
            end
            else if (shift_address == 1) begin
                ifft_sink_sop <= 0;
                shift_address <= shift_address + 1;
            end
            else if (ifft_sink_eop) begin
                reading <= 0;
                previous_peak_valid <= peak_valid;
                shift_address <= 0;
                ifft_sink_valid <= 0;
                ifft_sink_eop <= 0;
                ifft_sink_sop <= 1;
            end
            else begin
                shift_address <= shift_address + 1;
                if (shift_address == 2047) begin
                    ifft_sink_eop <= 1;
```

```
                shift_address <= 2047;
            end
            else begin
                ifft_sink_sop <= 0;
                ifft_sink_eop <= 0;
            end
        end
    end
  end

endmodule

module convert_28_to_16(
  input logic [27:0] source_real,
  input logic [27:0] source_imag,
  output logic [15:0] sink_real,
  output logic [15:0] sink_imag
);

  assign sink_real = source_real[27:12];
  assign sink_imag = source_imag[27:12];

endmodule

module mem_block(
  input logic clk,
  input logic reset_n,
  input logic [15:0] source_real,
  input logic [15:0] source_imag,
  input logic [10:0] address,
  input logic source_valid, // wire to FFT's source_valid
  input logic source_sop, // wire to FFT's source_sop
  input logic source_eop, // wire to FFT's source_eop
  output logic [15:0] sink_real,
  output logic [15:0] sink_imag
);

  logic [15:0] mem_real [0:2047];
```

```
    logic [15:0] mem_imag [0:2047];
    logic write_en;

    logic [10:0] write_address;

    assign sink_real = mem_real[address];
    assign sink_imag = mem_imag[address];
    // assign ready = ~write_en;

    always_ff @(posedge clk) begin
        if (!reset_n) begin
            for (int i = 0; i < 2048; i++) begin
                mem_real[i] <= 0;
                mem_imag[i] <= 0;
            end
            write_address <= 0;
        end
        else if (source_valid) begin
            write_en <= 1;
            if (source_sop) begin
                mem_real[0] <= source_real;
                mem_imag[0] <= source_imag;
                write_address <= 1;
            end
            else if (source_eop) begin
                mem_real[write_address] <= source_real;
                mem_imag[write_address] <= source_imag;
                write_address <= 0;
                write_en <= 0;
            end
            else begin
                mem_real[write_address] <= source_real;
                mem_imag[write_address] <= source_imag;
                write_address <= write_address + 1;
            end
        end
    end

endmodule
```

```
module peak_detector(
    input logic clk,
    input logic reset_n,
    input logic [15:0] source_real,
    input logic [15:0] source_imag,
    input logic sink_sop,
    input logic sink_eop,
    input logic sink_valid,
    output logic source_valid,
    output logic [9:0] peak_index
);

    logic [10:0] counter;
    logic [31:0] current_mag;
    logic [31:0] max_mag;

    data2mag data2mag_inst(
        .source_real(source_real),
        .source_imag(source_imag),
        .magnitude(current_mag)
    );

    always_ff @(posedge clk) begin
        if (!reset_n) begin
            peak_index <= 0;
            source_valid <= 0;
            counter <= 0;
        end
        else if (sink_valid) begin
            counter <= counter + 1;
            if (sink_sop) begin
                peak_index <= 0;
                max_mag <= current_mag;
                source_valid <= 0;
            end
            if (counter < 1024) begin
                if (current_mag > max_mag) begin
                    max_mag <= current_mag;
```

```
                peak_index <= counter;
            end
        end
        if (sink_eop) begin
            counter <= 0;
            source_valid <= 1;
        end
    end
  end
endmodule

module data2mag(
  input logic [15:0] source_real,
  input logic [15:0] source_imag,
  output logic [31:0] magnitude
);

  // we are doing 2's complement multiplication
  logic [31:0] real_mult, imag_mult;
  logic real_sign, imag_sign;
  logic [31:0] real_abs, imag_abs;
  logic [31:0] real_mult_abs, imag_mult_abs;

  assign real_sign = source_real[15];
  assign imag_sign = source_imag[15];

  assign real_abs = real_sign ? ~source_real + 1'b1 : source_real;
  assign imag_abs = imag_sign ? ~source_imag + 1'b1 : source_imag;

  assign real_mult_abs = real_abs * real_abs;
  assign imag_mult_abs = imag_abs * imag_abs;


  always_comb begin
    real_mult = real_sign ? ~real_mult_abs + 1'b1 : real_mult_abs;
    imag_mult = imag_sign ? ~imag_mult_abs + 1'b1 : imag_mult_abs;
    magnitude = real_mult + imag_mult;
  end
```

```
endmodule

module tuning(
    input logic[10:0] shift_address,
    input logic [5:0] shift_factor,
    output logic [10:0] mem_address
);

    assign mem_address[10] = shift_address[10];

    always_comb begin
        if (shift_factor == 0) begin
            mem_address[9:0] = shift_address[9:0];
        end
        else begin
            if (shift_address < 1024) begin
                mem_address[9:0] = shift_address[9:0] - shift_factor;
            end
            else begin
                mem_address[9:0] = shift_address[9:0] + shift_factor;
            end
        end
    end

endmodule

module shift(
    input logic [9:0] peak_index,
    output logic [5:0] shift_factor
);

    assign shift_factor = shift_table[peak_index];

    logic [5:0] shift_table [0:1023];
    initial begin
        shift_table[0] = 0;
        shift_table[1] = 1;
        shift_table[2] = 1;
        shift_table[3] = 1;
```

```
shift_table[4] = 1;
shift_table[5] = 1;
shift_table[6] = 1;
shift_table[7] = 1;
shift_table[8] = 1;
shift_table[9] = 1;
shift_table[10] = 1;
shift_table[11] = 1;
shift_table[12] = 1;
shift_table[13] = 1;
shift_table[14] = 1;
shift_table[15] = 1;
shift_table[16] = 1;
shift_table[17] = 2;
shift_table[18] = 2;
shift_table[19] = 2;
shift_table[20] = 2;
shift_table[21] = 2;
shift_table[22] = 2;
shift_table[23] = 2;
shift_table[24] = 2;
shift_table[25] = 2;
shift_table[26] = 2;
shift_table[27] = 2;
shift_table[28] = 2;
shift_table[29] = 2;
shift_table[30] = 2;
shift_table[31] = 2;
shift_table[32] = 2;
shift_table[33] = 2;
shift_table[34] = 3;
shift_table[35] = 3;
shift_table[36] = 3;
shift_table[37] = 3;
shift_table[38] = 3;
shift_table[39] = 3;
shift_table[40] = 3;
shift_table[41] = 3;
shift_table[42] = 3;
```

```
shift_table[43] = 3;
shift_table[44] = 3;
shift_table[45] = 3;
shift_table[46] = 3;
shift_table[47] = 3;
shift_table[48] = 3;
shift_table[49] = 3;
shift_table[50] = 3;
shift_table[51] = 4;
shift_table[52] = 4;
shift_table[53] = 4;
shift_table[54] = 4;
shift_table[55] = 4;
shift_table[56] = 4;
shift_table[57] = 4;
shift_table[58] = 4;
shift_table[59] = 4;
shift_table[60] = 4;
shift_table[61] = 4;
shift_table[62] = 4;
shift_table[63] = 4;
shift_table[64] = 4;
shift_table[65] = 4;
shift_table[66] = 4;
shift_table[67] = 4;
shift_table[68] = 5;
shift_table[69] = 5;
shift_table[70] = 5;
shift_table[71] = 5;
shift_table[72] = 5;
shift_table[73] = 5;
shift_table[74] = 5;
shift_table[75] = 5;
shift_table[76] = 5;
shift_table[77] = 5;
shift_table[78] = 5;
shift_table[79] = 5;
shift_table[80] = 5;
shift_table[81] = 5;
```

```
shift_table[82] = 5;
shift_table[83] = 5;
shift_table[84] = 5;
shift_table[85] = 6;
shift_table[86] = 6;
shift_table[87] = 6;
shift_table[88] = 6;
shift_table[89] = 6;
shift_table[90] = 6;
shift_table[91] = 6;
shift_table[92] = 6;
shift_table[93] = 6;
shift_table[94] = 6;
shift_table[95] = 6;
shift_table[96] = 6;
shift_table[97] = 6;
shift_table[98] = 6;
shift_table[99] = 6;
shift_table[100] = 6;
shift_table[101] = 7;
shift_table[102] = 7;
shift_table[103] = 7;
shift_table[104] = 7;
shift_table[105] = 7;
shift_table[106] = 7;
shift_table[107] = 7;
shift_table[108] = 7;
shift_table[109] = 7;
shift_table[110] = 7;
shift_table[111] = 7;
shift_table[112] = 7;
shift_table[113] = 7;
shift_table[114] = 7;
shift_table[115] = 7;
shift_table[116] = 7;
shift_table[117] = 7;
shift_table[118] = 8;
shift_table[119] = 8;
shift_table[120] = 8;
```

```
shift_table[121] = 8;
shift_table[122] = 8;
shift_table[123] = 8;
shift_table[124] = 8;
shift_table[125] = 8;
shift_table[126] = 8;
shift_table[127] = 8;
shift_table[128] = 8;
shift_table[129] = 8;
shift_table[130] = 8;
shift_table[131] = 8;
shift_table[132] = 8;
shift_table[133] = 8;
shift_table[134] = 8;
shift_table[135] = 9;
shift_table[136] = 9;
shift_table[137] = 9;
shift_table[138] = 9;
shift_table[139] = 9;
shift_table[140] = 9;
shift_table[141] = 9;
shift_table[142] = 9;
shift_table[143] = 9;
shift_table[144] = 9;
shift_table[145] = 9;
shift_table[146] = 9;
shift_table[147] = 9;
shift_table[148] = 9;
shift_table[149] = 9;
shift_table[150] = 9;
shift_table[151] = 9;
shift_table[152] = 10;
shift_table[153] = 10;
shift_table[154] = 10;
shift_table[155] = 10;
shift_table[156] = 10;
shift_table[157] = 10;
shift_table[158] = 10;
shift_table[159] = 10;
```

```
shift_table[160] = 10;
shift_table[161] = 10;
shift_table[162] = 10;
shift_table[163] = 10;
shift_table[164] = 10;
shift_table[165] = 10;
shift_table[166] = 10;
shift_table[167] = 10;
shift_table[168] = 10;
shift_table[169] = 11;
shift_table[170] = 11;
shift_table[171] = 11;
shift_table[172] = 11;
shift_table[173] = 11;
shift_table[174] = 11;
shift_table[175] = 11;
shift_table[176] = 11;
shift_table[177] = 11;
shift_table[178] = 11;
shift_table[179] = 11;
shift_table[180] = 11;
shift_table[181] = 11;
shift_table[182] = 11;
shift_table[183] = 11;
shift_table[184] = 11;
shift_table[185] = 12;
shift_table[186] = 12;
shift_table[187] = 12;
shift_table[188] = 12;
shift_table[189] = 12;
shift_table[190] = 12;
shift_table[191] = 12;
shift_table[192] = 12;
shift_table[193] = 12;
shift_table[194] = 12;
shift_table[195] = 12;
shift_table[196] = 12;
shift_table[197] = 12;
shift_table[198] = 12;
```

```
shift_table[199] = 12;
shift_table[200] = 12;
shift_table[201] = 12;
shift_table[202] = 13;
shift_table[203] = 13;
shift_table[204] = 13;
shift_table[205] = 13;
shift_table[206] = 13;
shift_table[207] = 13;
shift_table[208] = 13;
shift_table[209] = 13;
shift_table[210] = 13;
shift_table[211] = 13;
shift_table[212] = 13;
shift_table[213] = 13;
shift_table[214] = 13;
shift_table[215] = 13;
shift_table[216] = 13;
shift_table[217] = 13;
shift_table[218] = 13;
shift_table[219] = 14;
shift_table[220] = 14;
shift_table[221] = 14;
shift_table[222] = 14;
shift_table[223] = 14;
shift_table[224] = 14;
shift_table[225] = 14;
shift_table[226] = 14;
shift_table[227] = 14;
shift_table[228] = 14;
shift_table[229] = 14;
shift_table[230] = 14;
shift_table[231] = 14;
shift_table[232] = 14;
shift_table[233] = 14;
shift_table[234] = 14;
shift_table[235] = 14;
shift_table[236] = 15;
shift_table[237] = 15;
```

```
shift_table[238] = 15;
shift_table[239] = 15;
shift_table[240] = 15;
shift_table[241] = 15;
shift_table[242] = 15;
shift_table[243] = 15;
shift_table[244] = 15;
shift_table[245] = 15;
shift_table[246] = 15;
shift_table[247] = 15;
shift_table[248] = 15;
shift_table[249] = 15;
shift_table[250] = 15;
shift_table[251] = 15;
shift_table[252] = 15;
shift_table[253] = 16;
shift_table[254] = 16;
shift_table[255] = 16;
shift_table[256] = 16;
shift_table[257] = 16;
shift_table[258] = 16;
shift_table[259] = 16;
shift_table[260] = 16;
shift_table[261] = 16;
shift_table[262] = 16;
shift_table[263] = 16;
shift_table[264] = 16;
shift_table[265] = 16;
shift_table[266] = 16;
shift_table[267] = 16;
shift_table[268] = 16;
shift_table[269] = 16;
shift_table[270] = 17;
shift_table[271] = 17;
shift_table[272] = 17;
shift_table[273] = 17;
shift_table[274] = 17;
shift_table[275] = 17;
shift_table[276] = 17;
```

```
shift_table[277] = 17;
shift_table[278] = 17;
shift_table[279] = 17;
shift_table[280] = 17;
shift_table[281] = 17;
shift_table[282] = 17;
shift_table[283] = 17;
shift_table[284] = 17;
shift_table[285] = 17;
shift_table[286] = 18;
shift_table[287] = 18;
shift_table[288] = 18;
shift_table[289] = 18;
shift_table[290] = 18;
shift_table[291] = 18;
shift_table[292] = 18;
shift_table[293] = 18;
shift_table[294] = 18;
shift_table[295] = 18;
shift_table[296] = 18;
shift_table[297] = 18;
shift_table[298] = 18;
shift_table[299] = 18;
shift_table[300] = 18;
shift_table[301] = 18;
shift_table[302] = 18;
shift_table[303] = 19;
shift_table[304] = 19;
shift_table[305] = 19;
shift_table[306] = 19;
shift_table[307] = 19;
shift_table[308] = 19;
shift_table[309] = 19;
shift_table[310] = 19;
shift_table[311] = 19;
shift_table[312] = 19;
shift_table[313] = 19;
shift_table[314] = 19;
shift_table[315] = 19;
```

```
shift_table[316] = 19;
shift_table[317] = 19;
shift_table[318] = 19;
shift_table[319] = 19;
shift_table[320] = 20;
shift_table[321] = 20;
shift_table[322] = 20;
shift_table[323] = 20;
shift_table[324] = 20;
shift_table[325] = 20;
shift_table[326] = 20;
shift_table[327] = 20;
shift_table[328] = 20;
shift_table[329] = 20;
shift_table[330] = 20;
shift_table[331] = 20;
shift_table[332] = 20;
shift_table[333] = 20;
shift_table[334] = 20;
shift_table[335] = 20;
shift_table[336] = 20;
shift_table[337] = 21;
shift_table[338] = 21;
shift_table[339] = 21;
shift_table[340] = 21;
shift_table[341] = 21;
shift_table[342] = 21;
shift_table[343] = 21;
shift_table[344] = 21;
shift_table[345] = 21;
shift_table[346] = 21;
shift_table[347] = 21;
shift_table[348] = 21;
shift_table[349] = 21;
shift_table[350] = 21;
shift_table[351] = 21;
shift_table[352] = 21;
shift_table[353] = 21;
shift_table[354] = 22;
```

```
shift_table[355] = 22;
shift_table[356] = 22;
shift_table[357] = 22;
shift_table[358] = 22;
shift_table[359] = 22;
shift_table[360] = 22;
shift_table[361] = 22;
shift_table[362] = 22;
shift_table[363] = 22;
shift_table[364] = 22;
shift_table[365] = 22;
shift_table[366] = 22;
shift_table[367] = 22;
shift_table[368] = 22;
shift_table[369] = 22;
shift_table[370] = 23;
shift_table[371] = 23;
shift_table[372] = 23;
shift_table[373] = 23;
shift_table[374] = 23;
shift_table[375] = 23;
shift_table[376] = 23;
shift_table[377] = 23;
shift_table[378] = 23;
shift_table[379] = 23;
shift_table[380] = 23;
shift_table[381] = 23;
shift_table[382] = 23;
shift_table[383] = 23;
shift_table[384] = 23;
shift_table[385] = 23;
shift_table[386] = 23;
shift_table[387] = 24;
shift_table[388] = 24;
shift_table[389] = 24;
shift_table[390] = 24;
shift_table[391] = 24;
shift_table[392] = 24;
shift_table[393] = 24;
```

```
shift_table[394] = 24;
shift_table[395] = 24;
shift_table[396] = 24;
shift_table[397] = 24;
shift_table[398] = 24;
shift_table[399] = 24;
shift_table[400] = 24;
shift_table[401] = 24;
shift_table[402] = 24;
shift_table[403] = 24;
shift_table[404] = 25;
shift_table[405] = 25;
shift_table[406] = 25;
shift_table[407] = 25;
shift_table[408] = 25;
shift_table[409] = 25;
shift_table[410] = 25;
shift_table[411] = 25;
shift_table[412] = 25;
shift_table[413] = 25;
shift_table[414] = 25;
shift_table[415] = 25;
shift_table[416] = 25;
shift_table[417] = 25;
shift_table[418] = 25;
shift_table[419] = 25;
shift_table[420] = 25;
shift_table[421] = 26;
shift_table[422] = 26;
shift_table[423] = 26;
shift_table[424] = 26;
shift_table[425] = 26;
shift_table[426] = 26;
shift_table[427] = 26;
shift_table[428] = 26;
shift_table[429] = 26;
shift_table[430] = 26;
shift_table[431] = 26;
shift_table[432] = 26;
```

```
shift_table[433] = 26;
shift_table[434] = 26;
shift_table[435] = 26;
shift_table[436] = 26;
shift_table[437] = 26;
shift_table[438] = 27;
shift_table[439] = 27;
shift_table[440] = 27;
shift_table[441] = 27;
shift_table[442] = 27;
shift_table[443] = 27;
shift_table[444] = 27;
shift_table[445] = 27;
shift_table[446] = 27;
shift_table[447] = 27;
shift_table[448] = 27;
shift_table[449] = 27;
shift_table[450] = 27;
shift_table[451] = 27;
shift_table[452] = 27;
shift_table[453] = 27;
shift_table[454] = 27;
shift_table[455] = 28;
shift_table[456] = 28;
shift_table[457] = 28;
shift_table[458] = 28;
shift_table[459] = 28;
shift_table[460] = 28;
shift_table[461] = 28;
shift_table[462] = 28;
shift_table[463] = 28;
shift_table[464] = 28;
shift_table[465] = 28;
shift_table[466] = 28;
shift_table[467] = 28;
shift_table[468] = 28;
shift_table[469] = 28;
shift_table[470] = 28;
shift_table[471] = 29;
```

```
shift_table[472] = 29;
shift_table[473] = 29;
shift_table[474] = 29;
shift_table[475] = 29;
shift_table[476] = 29;
shift_table[477] = 29;
shift_table[478] = 29;
shift_table[479] = 29;
shift_table[480] = 29;
shift_table[481] = 29;
shift_table[482] = 29;
shift_table[483] = 29;
shift_table[484] = 29;
shift_table[485] = 29;
shift_table[486] = 29;
shift_table[487] = 29;
shift_table[488] = 30;
shift_table[489] = 30;
shift_table[490] = 30;
shift_table[491] = 30;
shift_table[492] = 30;
shift_table[493] = 30;
shift_table[494] = 30;
shift_table[495] = 30;
shift_table[496] = 30;
shift_table[497] = 30;
shift_table[498] = 30;
shift_table[499] = 30;
shift_table[500] = 30;
shift_table[501] = 30;
shift_table[502] = 30;
shift_table[503] = 30;
shift_table[504] = 30;
shift_table[505] = 31;
shift_table[506] = 31;
shift_table[507] = 31;
shift_table[508] = 31;
shift_table[509] = 31;
shift_table[510] = 31;
```

```
shift_table[511] = 31;
shift_table[512] = 31;
shift_table[513] = 31;
shift_table[514] = 31;
shift_table[515] = 31;
shift_table[516] = 31;
shift_table[517] = 31;
shift_table[518] = 31;
shift_table[519] = 31;
shift_table[520] = 31;
shift_table[521] = 31;
shift_table[522] = 32;
shift_table[523] = 32;
shift_table[524] = 32;
shift_table[525] = 32;
shift_table[526] = 32;
shift_table[527] = 32;
shift_table[528] = 32;
shift_table[529] = 32;
shift_table[530] = 32;
shift_table[531] = 32;
shift_table[532] = 32;
shift_table[533] = 32;
shift_table[534] = 32;
shift_table[535] = 32;
shift_table[536] = 32;
shift_table[537] = 32;
shift_table[538] = 32;
shift_table[539] = 33;
shift_table[540] = 33;
shift_table[541] = 33;
shift_table[542] = 33;
shift_table[543] = 33;
shift_table[544] = 33;
shift_table[545] = 33;
shift_table[546] = 33;
shift_table[547] = 33;
shift_table[548] = 33;
shift_table[549] = 33;
```

```
shift_table[550] = 33;
shift_table[551] = 33;
shift_table[552] = 33;
shift_table[553] = 33;
shift_table[554] = 33;
shift_table[555] = 34;
shift_table[556] = 34;
shift_table[557] = 34;
shift_table[558] = 34;
shift_table[559] = 34;
shift_table[560] = 34;
shift_table[561] = 34;
shift_table[562] = 34;
shift_table[563] = 34;
shift_table[564] = 34;
shift_table[565] = 34;
shift_table[566] = 34;
shift_table[567] = 34;
shift_table[568] = 34;
shift_table[569] = 34;
shift_table[570] = 34;
shift_table[571] = 34;
shift_table[572] = 35;
shift_table[573] = 35;
shift_table[574] = 35;
shift_table[575] = 35;
shift_table[576] = 35;
shift_table[577] = 35;
shift_table[578] = 35;
shift_table[579] = 35;
shift_table[580] = 35;
shift_table[581] = 35;
shift_table[582] = 35;
shift_table[583] = 35;
shift_table[584] = 35;
shift_table[585] = 35;
shift_table[586] = 35;
shift_table[587] = 35;
shift_table[588] = 35;
```

```
shift_table[589] = 36;
shift_table[590] = 36;
shift_table[591] = 36;
shift_table[592] = 36;
shift_table[593] = 36;
shift_table[594] = 36;
shift_table[595] = 36;
shift_table[596] = 36;
shift_table[597] = 36;
shift_table[598] = 36;
shift_table[599] = 36;
shift_table[600] = 36;
shift_table[601] = 36;
shift_table[602] = 36;
shift_table[603] = 36;
shift_table[604] = 36;
shift_table[605] = 36;
shift_table[606] = 37;
shift_table[607] = 37;
shift_table[608] = 37;
shift_table[609] = 37;
shift_table[610] = 37;
shift_table[611] = 37;
shift_table[612] = 37;
shift_table[613] = 37;
shift_table[614] = 37;
shift_table[615] = 37;
shift_table[616] = 37;
shift_table[617] = 37;
shift_table[618] = 37;
shift_table[619] = 37;
shift_table[620] = 37;
shift_table[621] = 37;
shift_table[622] = 37;
shift_table[623] = 38;
shift_table[624] = 38;
shift_table[625] = 38;
shift_table[626] = 38;
shift_table[627] = 38;
```

```
shift_table[628] = 38;
shift_table[629] = 38;
shift_table[630] = 38;
shift_table[631] = 38;
shift_table[632] = 38;
shift_table[633] = 38;
shift_table[634] = 38;
shift_table[635] = 38;
shift_table[636] = 38;
shift_table[637] = 38;
shift_table[638] = 38;
shift_table[639] = 38;
shift_table[640] = 39;
shift_table[641] = 39;
shift_table[642] = 39;
shift_table[643] = 39;
shift_table[644] = 39;
shift_table[645] = 39;
shift_table[646] = 39;
shift_table[647] = 39;
shift_table[648] = 39;
shift_table[649] = 39;
shift_table[650] = 39;
shift_table[651] = 39;
shift_table[652] = 39;
shift_table[653] = 39;
shift_table[654] = 39;
shift_table[655] = 39;
shift_table[656] = 40;
shift_table[657] = 40;
shift_table[658] = 40;
shift_table[659] = 40;
shift_table[660] = 40;
shift_table[661] = 40;
shift_table[662] = 40;
shift_table[663] = 40;
shift_table[664] = 40;
shift_table[665] = 40;
shift_table[666] = 40;
```

```
shift_table[667] = 40;
shift_table[668] = 40;
shift_table[669] = 40;
shift_table[670] = 40;
shift_table[671] = 40;
shift_table[672] = 40;
shift_table[673] = 41;
shift_table[674] = 41;
shift_table[675] = 41;
shift_table[676] = 41;
shift_table[677] = 41;
shift_table[678] = 41;
shift_table[679] = 41;
shift_table[680] = 41;
shift_table[681] = 41;
shift_table[682] = 41;
shift_table[683] = 41;
shift_table[684] = 41;
shift_table[685] = 41;
shift_table[686] = 41;
shift_table[687] = 41;
shift_table[688] = 41;
shift_table[689] = 41;
shift_table[690] = 42;
shift_table[691] = 42;
shift_table[692] = 42;
shift_table[693] = 42;
shift_table[694] = 42;
shift_table[695] = 42;
shift_table[696] = 42;
shift_table[697] = 42;
shift_table[698] = 42;
shift_table[699] = 42;
shift_table[700] = 42;
shift_table[701] = 42;
shift_table[702] = 42;
shift_table[703] = 42;
shift_table[704] = 42;
shift_table[705] = 42;
```

```
shift_table[706] = 42;
shift_table[707] = 43;
shift_table[708] = 43;
shift_table[709] = 43;
shift_table[710] = 43;
shift_table[711] = 43;
shift_table[712] = 43;
shift_table[713] = 43;
shift_table[714] = 43;
shift_table[715] = 43;
shift_table[716] = 43;
shift_table[717] = 43;
shift_table[718] = 43;
shift_table[719] = 43;
shift_table[720] = 43;
shift_table[721] = 43;
shift_table[722] = 43;
shift_table[723] = 43;
shift_table[724] = 44;
shift_table[725] = 44;
shift_table[726] = 44;
shift_table[727] = 44;
shift_table[728] = 44;
shift_table[729] = 44;
shift_table[730] = 44;
shift_table[731] = 44;
shift_table[732] = 44;
shift_table[733] = 44;
shift_table[734] = 44;
shift_table[735] = 44;
shift_table[736] = 44;
shift_table[737] = 44;
shift_table[738] = 44;
shift_table[739] = 44;
shift_table[740] = 45;
shift_table[741] = 45;
shift_table[742] = 45;
shift_table[743] = 45;
shift_table[744] = 45;
```

```
shift_table[745] = 45;
shift_table[746] = 45;
shift_table[747] = 45;
shift_table[748] = 45;
shift_table[749] = 45;
shift_table[750] = 45;
shift_table[751] = 45;
shift_table[752] = 45;
shift_table[753] = 45;
shift_table[754] = 45;
shift_table[755] = 45;
shift_table[756] = 45;
shift_table[757] = 46;
shift_table[758] = 46;
shift_table[759] = 46;
shift_table[760] = 46;
shift_table[761] = 46;
shift_table[762] = 46;
shift_table[763] = 46;
shift_table[764] = 46;
shift_table[765] = 46;
shift_table[766] = 46;
shift_table[767] = 46;
shift_table[768] = 46;
shift_table[769] = 46;
shift_table[770] = 46;
shift_table[771] = 46;
shift_table[772] = 46;
shift_table[773] = 46;
shift_table[774] = 47;
shift_table[775] = 47;
shift_table[776] = 47;
shift_table[777] = 47;
shift_table[778] = 47;
shift_table[779] = 47;
shift_table[780] = 47;
shift_table[781] = 47;
shift_table[782] = 47;
shift_table[783] = 47;
```

```
shift_table[784] = 47;
shift_table[785] = 47;
shift_table[786] = 47;
shift_table[787] = 47;
shift_table[788] = 47;
shift_table[789] = 47;
shift_table[790] = 47;
shift_table[791] = 48;
shift_table[792] = 48;
shift_table[793] = 48;
shift_table[794] = 48;
shift_table[795] = 48;
shift_table[796] = 48;
shift_table[797] = 48;
shift_table[798] = 48;
shift_table[799] = 48;
shift_table[800] = 48;
shift_table[801] = 48;
shift_table[802] = 48;
shift_table[803] = 48;
shift_table[804] = 48;
shift_table[805] = 48;
shift_table[806] = 48;
shift_table[807] = 48;
shift_table[808] = 49;
shift_table[809] = 49;
shift_table[810] = 49;
shift_table[811] = 49;
shift_table[812] = 49;
shift_table[813] = 49;
shift_table[814] = 49;
shift_table[815] = 49;
shift_table[816] = 49;
shift_table[817] = 49;
shift_table[818] = 49;
shift_table[819] = 49;
shift_table[820] = 49;
shift_table[821] = 49;
shift_table[822] = 49;
```

```
shift_table[823] = 49;
shift_table[824] = 49;
shift_table[825] = 50;
shift_table[826] = 50;
shift_table[827] = 50;
shift_table[828] = 50;
shift_table[829] = 50;
shift_table[830] = 50;
shift_table[831] = 50;
shift_table[832] = 50;
shift_table[833] = 50;
shift_table[834] = 50;
shift_table[835] = 50;
shift_table[836] = 50;
shift_table[837] = 50;
shift_table[838] = 50;
shift_table[839] = 50;
shift_table[840] = 50;
shift_table[841] = 51;
shift_table[842] = 51;
shift_table[843] = 51;
shift_table[844] = 51;
shift_table[845] = 51;
shift_table[846] = 51;
shift_table[847] = 51;
shift_table[848] = 51;
shift_table[849] = 51;
shift_table[850] = 51;
shift_table[851] = 51;
shift_table[852] = 51;
shift_table[853] = 51;
shift_table[854] = 51;
shift_table[855] = 51;
shift_table[856] = 51;
shift_table[857] = 51;
shift_table[858] = 52;
shift_table[859] = 52;
shift_table[860] = 52;
shift_table[861] = 52;
```

```
shift_table[862] = 52;
shift_table[863] = 52;
shift_table[864] = 52;
shift_table[865] = 52;
shift_table[866] = 52;
shift_table[867] = 52;
shift_table[868] = 52;
shift_table[869] = 52;
shift_table[870] = 52;
shift_table[871] = 52;
shift_table[872] = 52;
shift_table[873] = 52;
shift_table[874] = 52;
shift_table[875] = 53;
shift_table[876] = 53;
shift_table[877] = 53;
shift_table[878] = 53;
shift_table[879] = 53;
shift_table[880] = 53;
shift_table[881] = 53;
shift_table[882] = 53;
shift_table[883] = 53;
shift_table[884] = 53;
shift_table[885] = 53;
shift_table[886] = 53;
shift_table[887] = 53;
shift_table[888] = 53;
shift_table[889] = 53;
shift_table[890] = 53;
shift_table[891] = 53;
shift_table[892] = 54;
shift_table[893] = 54;
shift_table[894] = 54;
shift_table[895] = 54;
shift_table[896] = 54;
shift_table[897] = 54;
shift_table[898] = 54;
shift_table[899] = 54;
shift_table[900] = 54;
```

```
shift_table[901] = 54;
shift_table[902] = 54;
shift_table[903] = 54;
shift_table[904] = 54;
shift_table[905] = 54;
shift_table[906] = 54;
shift_table[907] = 54;
shift_table[908] = 54;
shift_table[909] = 55;
shift_table[910] = 55;
shift_table[911] = 55;
shift_table[912] = 55;
shift_table[913] = 55;
shift_table[914] = 55;
shift_table[915] = 55;
shift_table[916] = 55;
shift_table[917] = 55;
shift_table[918] = 55;
shift_table[919] = 55;
shift_table[920] = 55;
shift_table[921] = 55;
shift_table[922] = 55;
shift_table[923] = 55;
shift_table[924] = 55;
shift_table[925] = 56;
shift_table[926] = 56;
shift_table[927] = 56;
shift_table[928] = 56;
shift_table[929] = 56;
shift_table[930] = 56;
shift_table[931] = 56;
shift_table[932] = 56;
shift_table[933] = 56;
shift_table[934] = 56;
shift_table[935] = 56;
shift_table[936] = 56;
shift_table[937] = 56;
shift_table[938] = 56;
shift_table[939] = 56;
```

```
shift_table[940] = 56;
shift_table[941] = 56;
shift_table[942] = 57;
shift_table[943] = 57;
shift_table[944] = 57;
shift_table[945] = 57;
shift_table[946] = 57;
shift_table[947] = 57;
shift_table[948] = 57;
shift_table[949] = 57;
shift_table[950] = 57;
shift_table[951] = 57;
shift_table[952] = 57;
shift_table[953] = 57;
shift_table[954] = 57;
shift_table[955] = 57;
shift_table[956] = 57;
shift_table[957] = 57;
shift_table[958] = 57;
shift_table[959] = 58;
shift_table[960] = 58;
shift_table[961] = 58;
shift_table[962] = 58;
shift_table[963] = 58;
shift_table[964] = 58;
shift_table[965] = 58;
shift_table[966] = 57;
shift_table[967] = 56;
shift_table[968] = 55;
shift_table[969] = 54;
shift_table[970] = 53;
shift_table[971] = 52;
shift_table[972] = 51;
shift_table[973] = 50;
shift_table[974] = 49;
shift_table[975] = 48;
shift_table[976] = 47;
shift_table[977] = 46;
shift_table[978] = 45;
```

```
shift_table[979] = 44;
shift_table[980] = 43;
shift_table[981] = 42;
shift_table[982] = 41;
shift_table[983] = 40;
shift_table[984] = 39;
shift_table[985] = 38;
shift_table[986] = 37;
shift_table[987] = 36;
shift_table[988] = 35;
shift_table[989] = 34;
shift_table[990] = 33;
shift_table[991] = 32;
shift_table[992] = 31;
shift_table[993] = 30;
shift_table[994] = 29;
shift_table[995] = 28;
shift_table[996] = 27;
shift_table[997] = 26;
shift_table[998] = 25;
shift_table[999] = 24;
shift_table[1000] = 23;
shift_table[1001] = 22;
shift_table[1002] = 21;
shift_table[1003] = 20;
shift_table[1004] = 19;
shift_table[1005] = 18;
shift_table[1006] = 17;
shift_table[1007] = 16;
shift_table[1008] = 15;
shift_table[1009] = 14;
shift_table[1010] = 13;
shift_table[1011] = 12;
shift_table[1012] = 11;
shift_table[1013] = 10;
shift_table[1014] = 9;
shift_table[1015] = 8;
shift_table[1016] = 7;
shift_table[1017] = 6;
```

```
        shift_table[1018] = 5;
        shift_table[1019] = 4;
        shift_table[1020] = 3;
        shift_table[1021] = 2;
        shift_table[1022] = 1;
        shift_table[1023] = 0;

    end

endmodule
```

tb_driver_interface:

```
`timescale 1ns/1ps

module driver_interface_tb();
    // Parameters
    parameter DATA_SIZE = 28;
    parameter CLK_PERIOD = 20; // 50MHz clock

    // Test signals
    logic clk;
    logic reset;
    logic chipselect;
    logic address;
    logic read;
    logic source_valid;
    logic [DATA_SIZE-1:0] source_data;
    logic source_ready;
    logic [31:0] read_data;

    // Instantiate the driver interface
    driver_interface #(
```

```
    .DATA_SIZE(DATA_SIZE)
) dut (
    .clk(clk),
    .reset(reset),
    .chipselect(chipselect),
    .address(address),
    .read(read),
    .source_valid(source_valid),
    .source_data(source_data),
    .source_ready(source_ready),
    .read_data(read_data)
);

// Clock generation
initial begin
    clk = 0;
    forever #(CLK_PERIOD/2) clk = ~clk;
end

// Test stimulus
initial begin
    // Initialize signals
    reset = 1;
    chipselect = 0;
    address = 0;
    read = 0;
    source_valid = 0;
    source_data = 0;

    // Reset sequence
    #(CLK_PERIOD*2);
    reset = 0;
    #(CLK_PERIOD*2);

    // Test Case 1: Basic data transfer and read
    // Send data
    source_valid = 1;
    source_data = 28'h1234567;
    #(CLK_PERIOD);
```

```
        source_valid = 0;
        #(CLK_PERIOD*2);

        // Read the data
        chipselect = 1;
        read = 1;
        address = 0;
        #(CLK_PERIOD);
        chipselect = 0;
        read = 0;
        #(CLK_PERIOD*2);

        // Test Case 2: Try to write when data_full
        source_valid = 1;
        source_data = 28'hABCDEF0;
        #(CLK_PERIOD);
        source_valid = 0;
        #(CLK_PERIOD*2);

        // Read first data
        chipselect = 1;
        read = 1;
        address = 0;
        #(CLK_PERIOD);
        chipselect = 0;
        read = 0;
        #(CLK_PERIOD*2);

        // End simulation
        #(CLK_PERIOD*20);
        $finish;
    end

    // Monitor and display results
    initial begin
        $monitor("Time=%0t reset=%b chipselect=%b address=%b read=%b valid=%b
data=%h ready=%b read_data=%h",
                 $time, reset, chipselect, address, read, source_valid, source_data, source_ready,
read_data);
```

```
    end

    // Add waveform dumping
    initial begin
        $dumpfile("driver_interface_tb.vcd");
        $dumpvars(0, driver_interface_tb);
    end

    // Assertions
    property source_ready_when_empty;
        @(posedge clk) !dut.data_full |-> source_ready;
    endproperty
    assert property (source_ready_when_empty) else $error("source_ready should be 1 when
data_full is 0");

    property source_ready_when_full;
        @(posedge clk) dut.data_full |-> !source_ready;
    endproperty
    assert property (source_ready_when_full) else $error("source_ready should be 0 when
data_full is 1");

endmodule
```

Audio.c:

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "audio.h"

#define DRIVER_NAME "unknown"

/* Device registers */
#define AUDIO_SAMPLES(x) (x)

struct audio_dev {
   struct resource res; /* Resource: our registers */
         void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev;

static long audio_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
   audio_arg_t ada;

   switch (cmd) {

   case AUDIO_READ_SAMPLES:
      // Repeated calls to ioread32
      ada.data = ioread32(AUDIO_SAMPLES(dev.virtbase));
      pr_info("ada.data: %d\n", ada.data);

      // Copy data to user space
      if (copy_to_user((audio_arg_t *) arg, &ada, sizeof(audio_arg_t)))
```

```
            return -EACCES;
        break;

    default:
        return -EINVAL;
    }

    return 0;
}

/* The operations our device knows how to do */
static const struct file_operations audio_fops = {
        .owner                  = THIS_MODULE,
        .unlocked_ioctl = audio_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice audio_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &audio_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
 static int __init audio_probe(struct platform_device *pdev)
 {
    int ret;

    /* Register ourselves as a misc device: creates /dev/audio */
    ret = misc_register(&audio_misc_device);

    /* Get the address of our registers from the device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);

    if (ret) {
        ret = -ENOENT;
```

```
        goto out_deregister;
    }

    /* Make sure we can use these registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.res),
                DRIVER_NAME) == NULL) {
        ret = -EBUSY;
        goto out_deregister;
    }

    /* Arrange access to our registers */
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (dev.virtbase == NULL) {
        ret = -ENOMEM;
        goto out_release_mem_region;
    }

    return 0;

 out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
 out_deregister:
    misc_deregister(&audio_misc_device);
    return ret;
 }

 /* Clean-up code: release resources */
static int audio_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&audio_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id audio_of_match[] = {
        { .compatible = "unknown,unknown-1.0" },
```

```
        {},
};
MODULE_DEVICE_TABLE(of, audio_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver audio_driver = {
        .driver = {
                .name                   = DRIVER_NAME,
                .owner                  = THIS_MODULE,
                .of_match_table = of_match_ptr(audio_of_match),
        },
        .remove         = __exit_p(audio_remove),
};

/* Called when the module is loaded: set things up */
static int __init audio_init(void)
{
        pr_info(DRIVER_NAME ": init\n");
        return platform_driver_probe(&audio_driver, audio_probe);
}
/* Calball when the module is unloaded: release resources */ static void __exit
audio_exit(void) {
        platform_driver_unregister(&audio_driver);
        pr_info(DRIVER_NAME ": exit\n");
} module_init(audio_init);
module_exit(audio_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Autotune");
MODULE_DESCRIPTION("Audio driver");
```

auntotune.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "wavfile_construction/make_wav.h"
#include "audio.h"

#define S_RATE  (8000)
#define BUF_SIZE (S_RATE*15) /* 15 second buffer */

long unsigned int buffer[BUF_SIZE];
int idx;
int audio_fd;

/* Read and save samples from the device to our buffer */
void read_samples() {
   audio_arg_t vla;

   if (ioctl(audio_fd, AUDIO_READ_SAMPLES, &vla)) {
      perror("ioctl(AUDIO_READ_SAMPLES) failed");
      printf("vla.data = %d\n", vla.data);
      return;
   }
   buffer[idx++] = vla.data;
}

int main(int argc, char ** argv)
{
   idx = 0;

   static const char filename[] = "/dev/unknown";

   printf("Audio Userspace program started\n");

   if ( (audio_fd = open(filename, O_RDWR)) == -1) {
      fprintf(stderr, "could not open %s\n", filename);
      return -1;
   }

   printf("buf size: %d\n", BUF_SIZE);
```

```
    while(idx < BUF_SIZE){
        // printf("idx: %d\n", idx);
        read_samples();
    }

    printf("sample read done");

    write_wav("./wavfiles/anonymous_audio.wav", BUF_SIZE, (long unsigned int *)buffer,
S_RATE);

    printf("Audio Userspace program terminating\n");
    return 0;
}
```

Make_wav.c:

```
/* Creates a WAV file from an array of ints.
 * Output is stereo, signed 32-bit samples
 *
 * Based on code from Kevin Karplus, licensed under Creative Commons
 * https://karplus4arduino.wordpress.com/2011/10/08/making-wav-files-from-c-programs/
 */

#include <stdio.h>
#include <assert.h>
#include "make_wav.h"


void write_little_endian(long unsigned int word, int num_bytes, FILE *wav_file)
{
    unsigned buf;
    while(num_bytes>0)
    {   buf = word & 0xff;
```

```
        fwrite(&buf, 1,1, wav_file);
        num_bytes--;
        word >>= 8;
    }
}

/* information about the WAV file format from
   http://ccrma.stanford.edu/courses/422/projects/WaveFormat/
 */

void write_wav(char * filename, unsigned long num_samples, long unsigned int * data, int
s_rate)
{
    FILE* wav_file;
    unsigned int sample_rate;
    unsigned int num_channels;
    unsigned int bytes_per_sample;
    unsigned int byte_rate;
    unsigned long i, j;    /* counters for samples */

    num_channels = 1;   /* stereo */
    bytes_per_sample = 3; /* 24 bit samples */

    sample_rate = (unsigned int) s_rate;

    byte_rate = sample_rate*num_channels*bytes_per_sample;

    wav_file = fopen(filename, "w");
    assert(wav_file);   /* make sure it opened */

    /* write RIFF header */
    fwrite("RIFF", 1, 4, wav_file);
    write_little_endian(36 + bytes_per_sample* num_samples*num_channels, 4, wav_file);
    fwrite("WAVE", 1, 4, wav_file);

    /* write fmt  subchunk */
    fwrite("fmt ", 1, 4, wav_file);
    write_little_endian(16, 4, wav_file);   /* SubChunk1Size is 16 */
    write_little_endian(1, 2, wav_file);    /* PCM is format 1 */
```

```
    write_little_endian(num_channels, 2, wav_file);
    write_little_endian(sample_rate, 4, wav_file);
    write_little_endian(byte_rate, 4, wav_file);
    write_little_endian(num_channels*bytes_per_sample, 2, wav_file);  /* block align */
    write_little_endian(8*bytes_per_sample, 2, wav_file);  /* bits/sample */

    /* write data subchunk */
    fwrite("data", 1, 4, wav_file);
    write_little_endian(bytes_per_sample*num_samples*num_channels, 4, wav_file);
    for (i=0; i<num_samples; i+=num_channels)
        for(j=0; j<num_channels; j++)
            write_little_endian((unsigned int)(data[i+j]), bytes_per_sample, wav_file);

    /* close the file */
    fclose(wav_file);
}
```

Test.c:

```
#include <math.h>
#include "make_wav.h"

#define S_RATE  (44100)
#define BUF_SIZE (S_RATE*2) /* 2 second buffer */

int buffer[BUF_SIZE];

int main(int argc, char ** argv)
{
    int i;
    float t;
    float amplitude = 32000;
    float freq_Hz = 440;
    float phase=0;

    float freq_radians_per_sample = freq_Hz*2*M_PI/S_RATE;

    /* fill buffer with a sine wave */
    for (i=0; i<BUF_SIZE; i++)
```

```
    {
        phase += freq_radians_per_sample;
        buffer[i] = (int)(amplitude * sin(phase));
    }

    write_wav("test.wav", BUF_SIZE, buffer, S_RATE);

    return 0;
}
```