No Caller ID: Final Report

Because sometimes it's good to be a little incognito.

The secret agents behind the next big thing in spy-tech:

Noah Silverstein (nis2116), Max Acebal (mra2180), Marian Abuhazi (ma4107), and Rebekah Kim (rmk2160).

Abstract

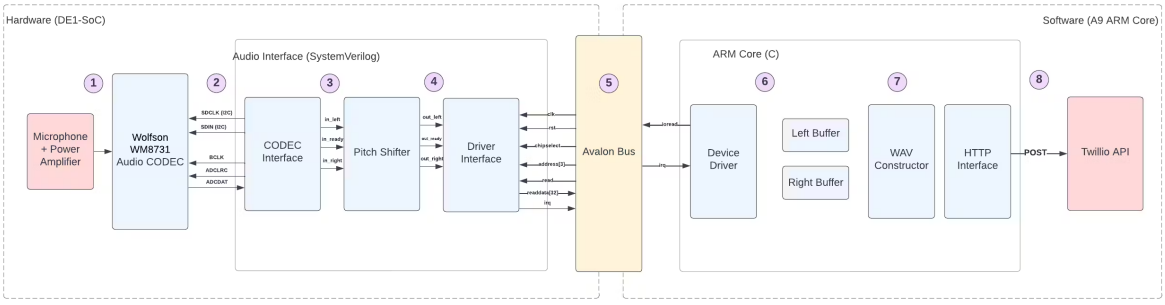
Team No Caller ID is proud to present our design for incognito phone calls. By sampling a microphone input through the *Mic in* port on the *Terasic FPGA* and converting the audio signal to a digital format, we will apply pitch-shift effects to the audio through a custom hardware module. This distorted audio will be sent to software and saved as *.wav* file to be sent as a pre-recorded phone call using the *Twilio* API.

The real-time audio effect will allow you to remain completely anonymous and can be used to throw somebody off your scent, allow for more privacy, or even have some fun.

System Overview

System Block Diagram

The full system from microphone input to API call output is detailed in the following system block diagram. All relevant connections are listed between functional units. The left side of the diagram indicates the FPGA hardware, and the right side indicates the ARM Core software. The Avalon Bus connects between the two in the center. In the following section, we will give an overview of each module and the full data flow through our system and give more details in the next section.



Modules and Data Flow

Below, we will discuss each data transition depicted in the system block diagram. Each number corresponds to the label within the system diagram, and the data flow and modification is described accordingly.

1. Microphone to Board

The DE1-SoC houses the Wolfson WM8731 audio CODEC, which in turn, contains an analog to digital converter (ADC) and an associated mic in port. An external, powered, microphone will be connected to this port as our audio source. Specifically, we will use a Shure SM58 microphone powered via a pre-amplifier connected to the mic in port of the ADC.

2. Audio CODEC to CODEC Interface

The Wolfson audio CODEC is configured via an I2C connection to our Audio Interface. The CODEC Interface, a small controller written in SystemVerilog which wraps various Altera IP blocks to help with configuration, puts the CODEC into slave mode and configures its operations via a series of programmable registers set via the I2C interface (the SDCLCK and SDIN lines). The CODEC contains the following 11 registers mapped to the listed addresses.

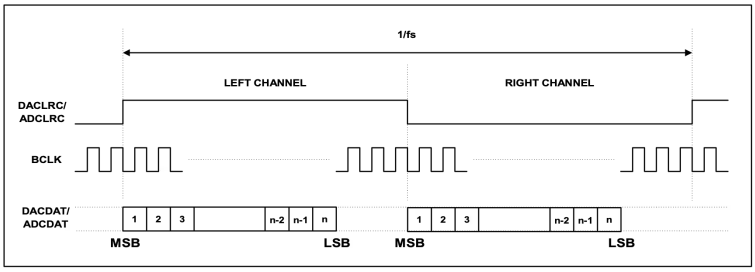
REGISTER	B 15	B 14	B 13	B 12	B 11	B 10	B 9	B8	B7	B6	B5	B4	B3	B2	B1	B0
R0 (00h)	0	0	0	0	0	0	0	LRIN BOTH	LIN MUTE	0	0	LINVOL				
R1 (02h)	0	0	0	0	0	0	1	RLIN BOTH	RIN MUTE	0	0	RINVOL				
R2 (04h)	0	0	0	0	0	1	0	LRHP BOTH	LZCEN	LHPVOL						
R3 (06h)	0	0	0	0	0	1	1	RLHP BOTH	RZCEN	RHPVOL						
R4 (08h)	0	0	0	0	1	0	0	0	SIDEATT		SIDETONE	DAC SEL	BY PASS	INSEL	MUTE MIC	MIC BOOST
R5 (0Ah)	0	0	0	0	1	0	1	0	0	0	0	HPOR	DAC MU	DEEMPH		ADC HPD
R6 (0Ch)	0	0	0	0	1	1	0	0	PWR OFF	CLK OUTPD	OSCPD	OUTPD	DACPD	ADCPD	MICPD	LINEINPD
R7 (0Eh)	0	0	0	0	1	1	1	0	BCLK INV	MS	LR SWAP	LRP	IWL		FORMAT	
R8 (10h)	0	0	0	1	0	0	0	0	CLKO DIV2	CLKI DIV2	SR				BOSR	USB/ NORM
R9 (12h)	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	ACTIVE
R15(1Eh)	0	0	0	1	1	1	1	RESET								
	ADDRESS							DATA								

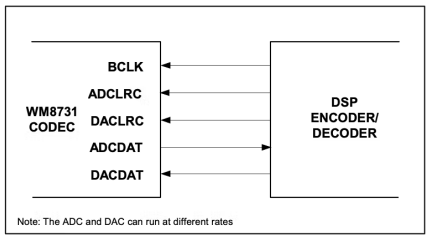
Our CODEC Interface has a simple I2C serial interface which loads the values listed above into the CODEC configuration registers. The relevant registers to set are R4 (bit 2 = 1 to enable mic In) and R7 (bit 1:0 = 01 for MSB-First, left-justified, bit 3:2 = 00 for 24 bits, bit 6 = 0 to enable slave mode). The following values are determined via the CODEC manual, and the relevant configuration options are listed below.

REGISTER ADDRESS	BIT	LABEL	DEFAULT	DESCRIPTION
0000111 Digital Audio Interface Format	1:0	FORMAT[1:0]	10	Audio Data Format Select 11 = DSP Mode, frame sync + 2 data packed words 10 = I^2S Format, MSB-First left-1 justified 01 = MSB-First, left justified 00 = MSB-First, right justified
	3:2	IWL[1:0]	10	Input Audio Data Bit Length Select 11 = 32 bits 10 = 24 bits 01 = 20 bits 00 = 16 bits
	4	LRP	0	DACLRC phase control (in left, right or I^2S modes) 1 = Right Channel DAC data when DACLRC high 0 = Right Channel DAC data when DACLRC low (opposite phasing in I^2S mode) or DSP mode A/B select (in DSP mode only) 1 = MSB is available on 2nd BCLK rising edge after DACLRC rising edge 0 = MSB is available on 1st BCLK rising edge after DACLRC rising edge
	5	LRSWAP	0	DAC Left Right Clock Swap 1 = Right Channel DAC Data Left 0 = Right Channel DAC Data Right
	6	MS	0	Master Slave Mode Control 1 = Enable Master Mode 0 = Enable Slave Mode
	7	BCLKINV	0	Bit Clock Invert 1 = Invert BCLK 0 = Don't invert BCLK

0000100 Analogue Audio Path Control	0	MICBOOST	0	Microphone Input Level Boost 1 = Enable Boost 0 = Disable Boost
	1	MUTEMIC	1	Mic Input Mute to ADC 1 = Enable Mute 0 = Disable Mute
	2	INSEL	0	Microphone/Line Input Select to ADC 1 = Microphone Input Select to ADC 0 = Line Input Select to ADC
	3	BYPASS	1	Bypass Switch 1 = Enable Bypass 0 = Disable Bypass
	4	DACSEL	0	DAC Select 1 =Select DAC 0 = Don't select DAC
	5	SIDETONE	0	Side Tone Switch 1 = Enable Side Tone 0 = Disable Side Tone
	7:6	SIDEATT[1:0]	00	Side Tone Attenuation 11 = -15dB 10 = -12dB 01 = -9dB 00 = -6dB

Once configured, the CODEC reports data, when requested, via a simple serial interface to our CODEC Interface. In short, the CODEC Interface pulls ADCLRCK high or low to receive 32 bits of either right or left channel data (ADCDAT) with the MSB sent first. With a 50MHz master clock, 48kHz sampling rate is trivially created via a clock divisor.





Most of the configuration and interaction with the CODEC is handled via Alterra IP blocks, with the top level *Audio Video Config* and *Audio CODEC* modules providing interfaces for configuration and operation of the CODEC. In addition, we took inspiration from a DE1-SoC based embedded systems class at the University of Toronto who similarly used the Altera IP blocks in addition to a top level wrapper that gives a simple interface for interacting the CODEC as a whole (just as our CODEC Interface module does).

3. CODEC Interface to Pitch Shifter Module

The CODEC Interface has two 24 bit registers to store a single left sample and a single right sample. As it pulses the BCLK signal and receives data from the CODEC, it writes into these registers. Then, the CODEC Interface, passes the left stereo data via in_left and right stereo data via in_right to the Pitch Shifter Module where they are stored in circular buffers. When new samples are ready, having been sampled and fetched from the CODEC, a ready flag is raised and the data is put on the output lines connected to the Pitch Shifter module. On the next cycle, the Pitch Shifter module reads the data from the registers and stores them to it's own registers.

4. Pitch Shifter Module to Driver Interface

The Audio Interface implements the octave shifting algorithm described later in the Algorithm's section, resulting in two 24 bit sample registers (left and right) which are updated at the same rate that new values are read in. They are similarly passed to the driver interface via out_left and out_right wires along with a ready flag. The driver interface follows the same protocol of reading the samples on the cycle after the ready flag goes high, just as the Pitch Shifter module did from the CODEC interface module.

5. Driver Interface to ARM Core

The Driver Interface reads left and right samples from the Pitch Shifter Module. At the same rate as the ADC samples (48kHz), it raises the irq (interrupt request) telling the ARM Core that it has a sample ready to read. Then, the device driver will handle the interrupt via reading from the Driver Interface via the Avalon bus. Specifically, the device driver places two read requests to two consecutive addresses, one for each of the left and right sample. The Driver Interface will send two 32 bit values of data over the readdata line of the Avalon interface, each being a 24 bit sample left padded with 0's. Finally, the device driver confirms that it has received the data by reading from a final address, and the driver interface module will lower the interrupt, allowing the ARM core to move on to other work.

6. Device Driver to User Memory Space

The device driver on the software side will receive ioctl read requests from the user level program. It will pause the user level process until an interrupt is raised from the hardware modules, indicating that new samples are ready. When an interrupt is raised, an interrupt handler is triggered and the device driver places several ioread's via the Avalon bus to fetch two 24 bit samples, which have been octave shifted. Having received these samples, and confirmed back to the hardware that the data has been successfully received, the device driver will unpause the user level program and send the data from kernel space to user space so that it can be processed for transmission.

7. Sample Accumulation and File Formatting

The C program will accumulate audio samples in two arrays (left and right channels). It sits in a loop, placing ioctl's reads to the device driver to fetch samples until a certain number of samples (length of recording) is stored. Once this happens, the data will be formatted as a WAV file as described in the Software section below.

8. API Request

The WAV file will be uploaded to the Github via a bash script and then sent as a phone call using a Twilio POST HTTP request. Hosted on the Twilio site, we have a specially formatted XML file that takes our request, downloads the WAV file from Github, and makes the call with the audio.

Hardware

Sample Protocol

We configured our modules to process 2 (left and right) 24-bit samples simultaneously. The CODEC is processing 48,000 samples per second from the microphone line, and our hardware and software modules are processing these samples at a 50 MHz clock. Clearly, the majority of the time our modules are in wait states waiting for valid data to arrive from the CODEC. We make use of two main signals throughout all of our interfaces to ensure that we only pitch-shift and save data that is valid. These two signals are *in_ready* and *out_ready*. The *in_ready* flag is first raised by the CODEC interface, this flag alerts to the pitch-shifter that it is safe to perform computations on the samples. Then, when the pitch-shifter is done computing, it produces its own *out_ready* signal that alerts to the drive interface that pitch-shifted samples are ready to be passed to software. The pitch-shifter's *out_ready* signal becomes the *in_ready* to the driver interface module. The driver module then raises interrupts upon valid data becoming available for the C program to process.

Audio CODEC Interface

We use Altera's IP block to configure I2C (**Altera_UP_I2C_AV_Auto_Initialize.v**) and slow the 50 MHz master clock to match our sampling rate of 48kHz (**Altera_UP_Slow_Clock_Generator.v**) through **audio_and_video_config.v**. This module handles all of the I2C configuration required to start using the Wolfson Audio CODEC. We simply hard code the parameters that we would like to be written into the configuration registers on the CODEC, and it handles the rest.

The configuration parameters we used are listed below:

```
parameter AUD_LINE_IN_LC = 9'h01A;
parameter AUD_LINE_IN_RC = 9'h01A;
parameter AUD_LINE_OUT_LC = 9'h07B;
parameter AUD_LINE_OUT_RC = 9'h07B;
parameter AUD_ADC_PATH = 9'd149;
parameter AUD_DAC_PATH = 9'h006;
parameter AUD_POWER = 9'h000100000;
parameter AUD_DATA_FORMAT = 9'd73; // corresponds to 1001001 bit 6 = 0 to enable slave    mode, bit 3:2 = 00 for 24 bits, bit 1:0 = 01 for MSB-First, left-justified
parameter AUD_SAMPLE_CTRL = 9'd0;
parameter AUD_SET_ACTIVE = 9'h001;
```

We also use a wrapper around the codec IP block (**audio_codec.v**) to have a cleaner interface for outside modules to interact with in **codec_interface.v.** This module was heavily inspired by a similar wrapper written for a class at the University of Toronto. The following signals are connected to the next module (**pitch_shifter.sv**) to pass the sampled data and the ready flag**.**

```
output signed [23:0] adc_left, adc_right;
output advance; // ready signal
```

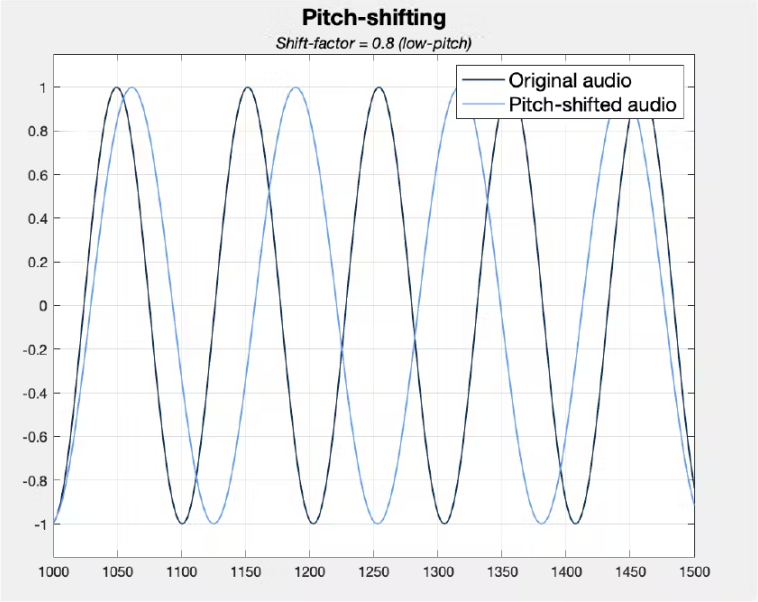
As with the rest of our hardware modules, this wrapper follows a simple protocol of raising the ready flag (called advance) while at the same time putting out new samples on it's output registers. The next module in our processing line will then be expected to read the samples on the following clock cycle.

Pitch Shifter Module

Pitch Shifter

Introduction

Pitch-shifting is the process of altering the pitch of an audio signal without changing its duration. Consider the following visual. After pitch-shifting we can lower the pitch of the "Original audio" without reducing nor increasing the length of the audio signal.



Pitch-shifting can be performed in the frequency domain using the Fast Fourier Transform (FFT) algorithm, manipulating the frequencies of the signal, and reverting to a time-domain signal using an inverse FFT. However, this method is mathematically intensive and complicated to implement in hardware.

In this project we will apply audio effects on a voice signal captured on the mic line to the DE1-SOC board. By default, we offer one type of audio effect: the ability to shift the signal to a lower pitch by making the user sound like they have a "Darth Vader" voice. However, the lower pitch can easily be altered to a higher pitch by changing the pitch-shifting factor parameter in the Pitch Shifter module.

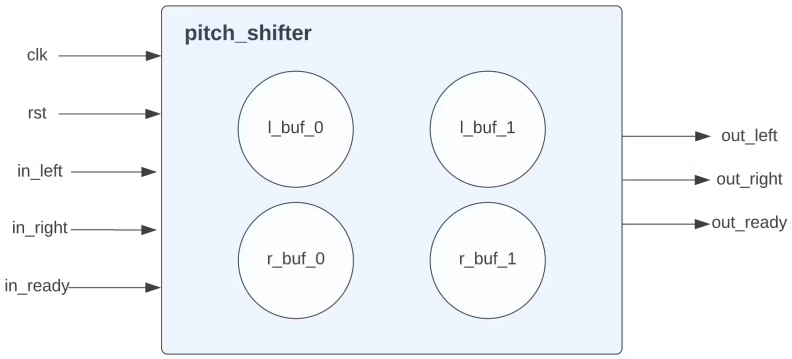
We have implemented the algorithm described below to perform the pitch-shifting entirely in the time domain. The algorithm has been adapted from PitchShifter by Sean Carroll, Gulnar Mirza, and James Talmage from Cornell University.

Overview

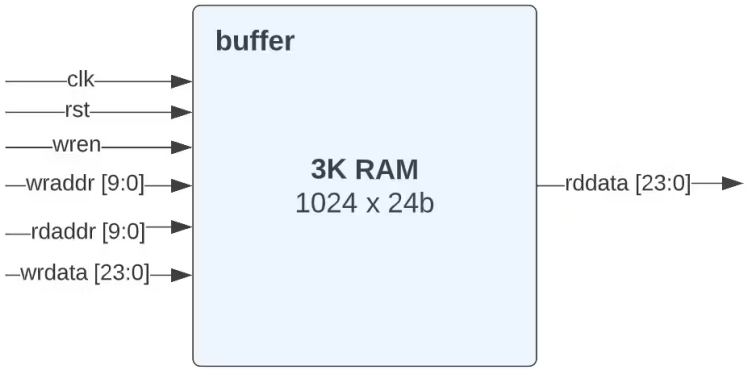
The Pitch Shifter module, *pitch_shifter.sv*, implements the time-domain shifting algorithm by instantiating 4 1024 x 24-bit RAM blocks from *buffer.sv*. The algorithm takes incoming 24-bit samples passed in from the CODEC interface and stores them one at a time into the 1024-RAM blocks. The module will then read back data from the RAM blocks at some *read* offset which is determined by the shift factor parameter (0.5 to 2.5). The Pitch Shifter module shifts the signal to the desired octave by either repeatedly outputting the same sample for multiple cycles, or skipping some cycles altogether. The repetition or skipping of samples simulates the effect of prolonging or shortening a tone, which creates the lower or higher pitch desired. It also has the effect of speeding or slowing the perceived rate of audio playback since samples are either repeated or discarded. Through extensive MATLAB simulations, we have determined that a good shift factor for a lower pitch is around 0.8-0.95, and a good shift factor for a higher pitch is between 1.5 to 2.

Architecture

The Pitch Shifter module takes in the following signals: a clock (*clk*), a reset (*rst*), a 24-bit left speaker data sample (*in_left*), a 24-bit right speaker data sample (*in_right*), and a ready signal raised by the CODEC interface (*in_ready*). The module outputs 3 signals: *out_left* and *out_right*, which correspond to the left and right outputs of the shifter, and *out_ready* which indicates to the driver interface that the data is valid.



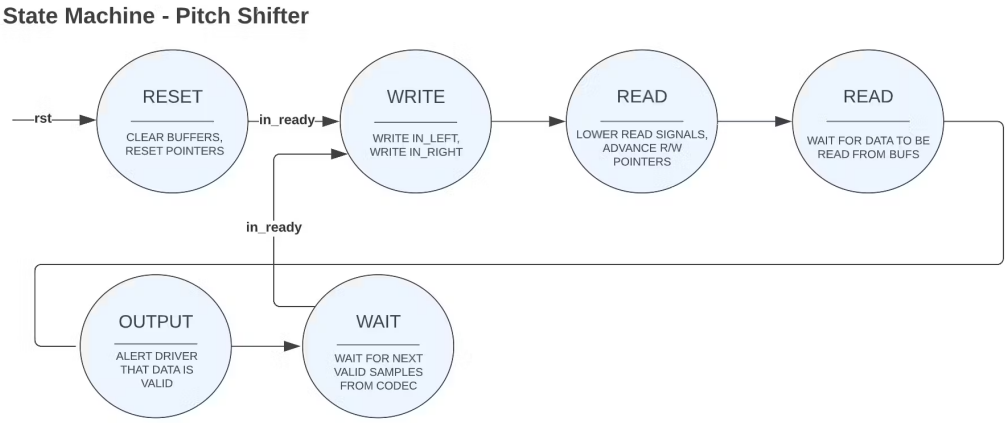
The Pitch Shifter module instantiates 4 of the following 3K RAM blocks. These 1024 x 24-bit memory arrays work as circular buffers, meaning they hold at most 1024 samples at a time, and any new data that comes in will override old data in the buffer.



Algorithm

On every rising edge of *clk*, *pitch_shifter* reads the *in_left* and *in_right* samples and passes the values read to the instantiated buffer modules. In particular, *in_left* is written to *l_buf_0* and *l_buf_1* and *in_right* is written to *r_buf_0* and

r_buf_1 simultaneously. Each buffer has corresponding read and write pointers that indicate at which address in the RAM blocks the data should be either written to or read from. All pointers are initialized to 0 upon *rst*.



Now, let's simulate shifting a pair of samples *in_left* and *in_right* from the moment they are passed into the Pitch Shifter by the CODEC to the moment they are output to the driver.

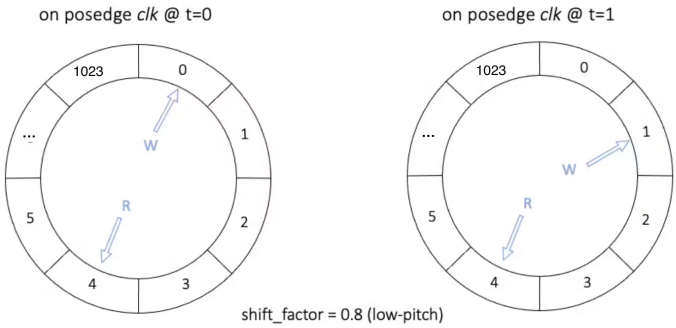
Writing to buffers

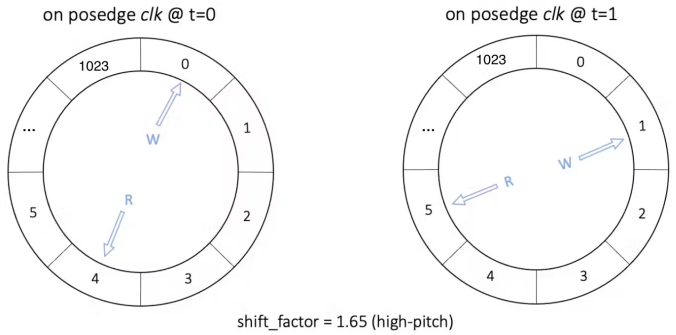
Once the Pitch Shifter receives the *in_ready* flag from the CODEC, it will write *in_left* to the *l_buf_0* and *l_buf_1* buffers at addresses *l_w_0* and *l_w_1* respectively. Simultaneously, it will write *in_right* to the *r_buf_0* and *r_buf_1* buffers at address *r_w_0* and *r_w_1*. It is important to note that upon reset, these pointers were initialized. *l_w_0* and *r_w_0* were initialized to the 0th index of the RAM, and *l_w_1* and *r_w_1* were initialized to the 511th index of the RAM (BUFFER SIZE / 2). The motivation behind this is that we wanted the write pointers of the 2 buffers for the left and for the right to be 180 degrees apart in order to reduce the probability of constantly outputting old samples. You can read more about this motivation here, but in short, the 180 degree writing offset helps to significantly reduce noise in the output audio.

Reading from buffers

Now that the new samples are written to the left and right buffers, they need to be read at the pitch-shifted location. This is done by offsetting the read pointer for each buffer by the integer portion of the shift-factor index. Upon reset, we declared the following four flags: *l_i_o, l_i_1, r_i_0 and r_i_1* and initialized them all to 0. Every time we read in a new pair of samples from the CODEC, we add on the shift factor amount to this index. Hence if the shift factor is set to 0.8, after 6 samples, the index will equal 4.8. However, we cannot read from the buffer at some decimal address, so we take the whole number part of the index and set that to be equal to the each of the read pointers.

Let's simulate this process for some random pair of samples that we get from the CODEC:
After initialization, the shifting index is 0. But once we get new samples (left and right), we add on the shift factor to the index. If the shift-factor is 0.8, the index is increasing by a factor of 0.8, and the read pointer will be the floor of the index. Hence, if our pointer is at address 4, and we add 0.8, then we floor it, the read pointer still remains at address 4 as shown below. However, if the shift-factor is 1.65 and our address pointer points to address 4, we will read from a location 1 cell away. This resembles the essence of pitch-shifting as it "skips" some samples for high-pitch shifting and "repeats" other samples for the low-pitch shifting.






```
// Update fractional indexes
  l_i_0 <= l_i_0 + shift_factor;
  l_i_1 <= l_i_1 + shift_factor;
  r_i_0 <= r_i_0 + shift_factor;
  r_i_1 <= r_i_1 + shift_factor;

// Advance read pointers
  l_r_0 <= l_i_0[$clog2(BUFFER_SIZE)+28:29]; // Take first 10b (int part of i_0)
  r_r_0 <= r_i_0[$clog2(BUFFER_SIZE)+28:29];
  l_r_1 <= l_i_1[$clog2(BUFFER_SIZE)+28:29]; // Take first 10b (int part of i_1)
  r_r_1 <= r_i_1[$clog2(BUFFER_SIZE)+28:29];
```

Averaging values read from buf_1 and buf_2

A dual-buffer design allows us to improve the quality of the output audio signal. Essentially, by having a second buffer whose read pointer is always 180° shifted from the main buffer we minimize the likelihood that the buffer will encounter and output an “older” sample that has already been written to the bus. This eliminates “clicking” sounds and white noise.

After, we have read from the buffers as shown in the previous steps, we will take the average of the two values read from the left and the average of the two values left from the right. This average becomes the out_left and out_right respectively.

```
// Take average of outputs of buffers 0 and 1 for left and right buffers
assign out_left = state == OUTPUT ? (((l_rddata_0 === 24'dx ? 0 : l_rddata_0) >>> 1) + ((l_rddata_1 === 24'dx ? 0 : l_rddata_1) >>> 1)): out_left;

assign out_right = state == OUTPUT ? (((r_rddata_0 === 24'dx ? 0 : r_rddata_0)  >>> 1) + ((r_rddata_1 === 24'dx ? 0 : r_rddata_1) >>> 1)): out_right;
```

Butterworth Filter

At some point during our debugging of the Pitch Shifter module, we recognized that the Pitch Shifter introduced some noise into our signal which often made the words spoken sound muffled and hard to understand. To reduce the high frequency noise we sought to implement a 6th order low-pass Butterworth filter. More background on the selection of Butterworth filter coefficients can be obtained from here.

The SystemVerilog code for butterworth_filter.sv was translated and adapted from the 18-bit IIR6 Verilog filter written by ECE 576 Cornell University. In the process of writing this filter we discovered that the CODEC was transmitting 24-bit signed, two's complement, fixed-point values to the Pitch Shifter module, and hence to this filter. We recognized that because Butterworth filters (and IIR) filters in general are implemented using a MAC (Multiply-and-Accumulate) algorithm, we would need to increase the precision of our samples in order to not lose significant parts of our audio in the filtering.

For this reason, we increased the original 18-bit input to the filter to be a 32-bit input. This meant that we would need to pad our 24-bit samples with 8 sign extended bits in order to preserve sign, but also to increase the accuracy of the sample after calculations.

Then, we would need to cut off these 8 bits before outputting the sample to the driver interface module in order to remain consistent with our 24-bit samples elsewhere.

The butterworth_filter.sv code instantiates two of the butterworth_filter_mono modules which we adapted from Cornell's code in order to filter the *in_left* sample and *in_right* samples separately. Butterworth_filter.sv raises the *out_ready* flag for the driver to read the data, when both mono modules are done filtering their respective samples.

Unfortunately, we did not get this filtering module working properly yet. We believe we are super close but at the moment the filter appears to add more noise to the output audio than assist in removing the high frequencies. We suspect that the issue lies in the way we handle the sign extension when padding our 24-bit samples with 8 extra bits to fit the 32-bit IIR.

Driver Interface

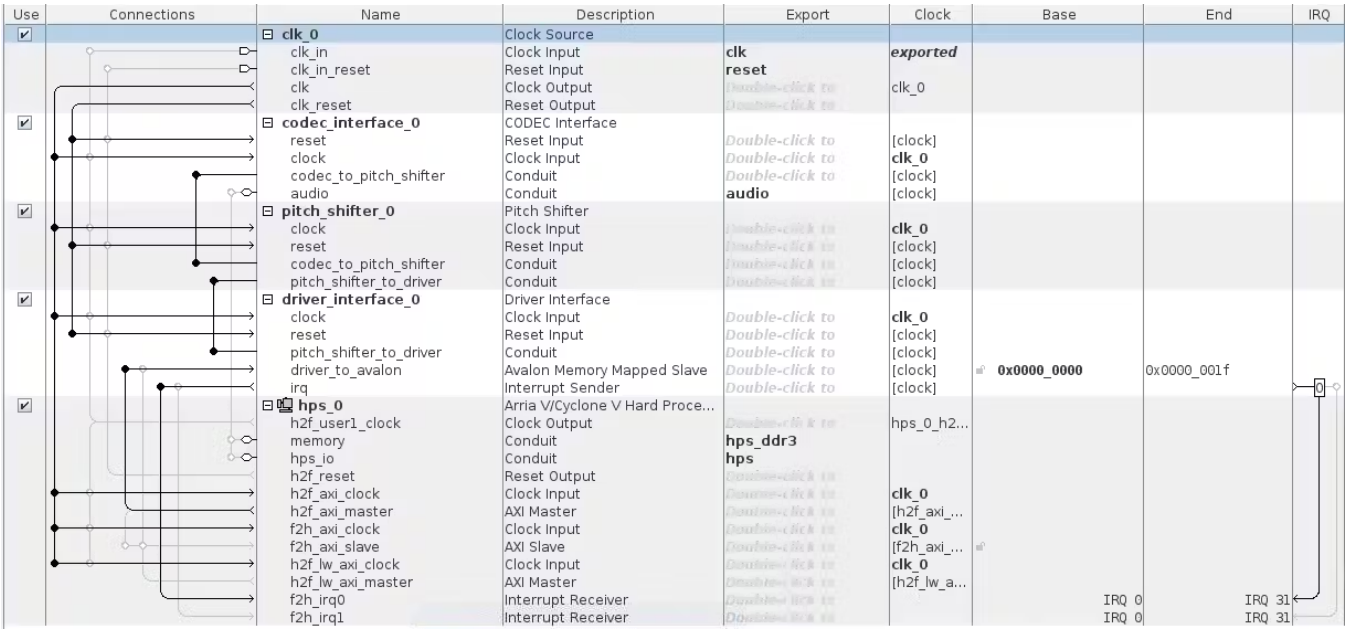
The driver interface module is responsible for taking the final, pitch shifted, samples from the Pitch Shifter module, and passing them across the Avalon bus, to the linux device driver running on the ARM core. Specially, its job is to raise interrupts when new samples arrive, and respond tor read requests when received from the Avalon bus. The operation can be described in two phases (a very simple finite state machine):

1. Wait for new samples to arrive. When the in ready flag goes high, read the incoming samples into local registers on the following clock cycle. Also raise the outgoing interrupt flag which will be transmitted across the Avalon bus. Go to state 2.
2. Wait for read requests from the Avalon bus.
    a. Receive a read request for address zero. Send the left sample across the Avalon bus.
    b. Receive a read request for address one. Send the right sample across the Avalon bus.
    c. Receive a read request for address two. Lower the outgoing interrupt flag and go to state 1.

This module is quite simple in operation, but crucial to connecting our hardware and software. It acts as the bridge between the two halves of our system.

System Connections

All SystemVerilog modules are added and connected in the Qsys platform builder from Altera. The connections are shown below.



Software

Kernel Space Programs

Device Driver & Interrupts

We are using interrupts to send the sample data from hardware to software. We first register the interrupt by getting an irq number assigned from the kernel and our custom handler to populate the data with `request_irq`.

```
int irq = irq_of_parse_and_map(pdev->dev.of_node, 0);
dev.irq_num = irq;

// Request our interrupt line and register our handler
ret = request_irq(irq, (irq_handler_t) irq_handler, 0, "csee4840_audio", NULL);
```

When the user level program places an `ioctl` read to the device driver, it is paused and put in a wait queue. Whenever the interrupt is raised from the hardware, the `irq_handler` is triggered to read in the samples into a buffer and raise the ready signal. Then the sleeping process placed in the wait queue wakes up to actually copy in the data from the refreshed buffer.

```
DECLARE_WAIT_QUEUE_HEAD(wq);

#define L_SAMPLES(x) (x)
#define R_SAMPLES(x) ((x) + 4)
#define RESET_IRQ(x) ((x) + 8)
```

```
static void read_samples(audio_samples_t *samples) {
 samples->l = ioread32(L_SAMPLES(dev.virtbase));
 samples->r = ioread32(R_SAMPLES(dev.virtbase));
 ioread32(RESET_IRQ(dev.virtbase));
 dev.samples = *samples;
}

static long audio_ioctl(struct file *f, unsigned int cmd, unsigned long arg) {
  switch (cmd) {
    case AUDIO_READ_SAMPLES:
      // Sleep the process until woken by the interrupt handler, and the data is ready
      wait_event_interruptible_exclusive(wq, dev.ready.audio_ready);
      // The data is now ready, send them to the user space
      vla.samples = dev.samples;
      audio_ready_t ready = { .audio_ready = 0 };
      dev.ready = ready;

      // Copy the data to the user space
      if (copy_to_user((audio_arg_t *) arg, &vla, sizeof(audio_arg_t)))
          return -EACCES;
      break;
  }
}

irq_handler_t irq_handler(int irq, void *dev_id, struct pt_regs *reg) {
   // Read samples from the hardware device
   audio_samples_t samples;
   read_samples(&samples);

   // Raise the ready flag and wave the user space program
   audio_ready_t ready = { .audio_ready = 1 };
   dev.ready = ready;
   wake_up_interruptible(&wq);

   return IRQ_RETVAL(1);
}
```

User Space Programs

Caller.c

Our user level program `caller.c`, takes the input stream from our audio device and stores them into a long unsigned int buffer array. We do this In our `read_samples()` function, where we store the left and right samples in consecutive positions.

```
void read_samples() {
   audio_arg_t vla;

   if (ioctl(audio_fd, AUDIO_READ_SAMPLES, &vla)) {
```

```
        perror("ioctl(AUDIO_READ_SAMPLES) failed");
        return;
    }

    buffer[idx++] = vla.samples.l;
    buffer[idx++] = vla.samples.r;
}
```

Our main function takes care of the final steps including, defining a string constant for the path of our audio device and opening using the open() function, and reading samples until our buffer is full. To allow for 5 seconds of audio, we instantiate our buffer with a size of S_RATE * 5 * 2. Our 44,100 sample rate is multiplied by five seconds and then again by two to account for both the left and right samples. Finally we write the buffer into a wav file using `write_wav()`.

```
static const char filename[] = "/dev/audio";
printf("Audio Userspace program started\n");
if ( (audio_fd = open(filename, O_RDWR)) == -1) {
  fprintf(stderr, "could not open %s\n", filename);
  return -1;
}
while(idx < BUF_SIZE){
  read_samples();
}
write_wav("./wavfiles/anonymous_audio.wav", BUF_SIZE, buffer, S_RATE);
```

WAV File Construction

To generate a WAV file we must take the streamed input and write it to a file with the correct WAV file headers. From there, we will write a **long unsigned integer** input to the file line by line. The code to do this formatting is adapted from an existing function we found online.

For our purposes our sample rate will be **48 kHz**, we will have **2 channels** (left and right), we will have **3 bytes (24 bits)** per sample**.** This is because we will be using our 24 bit input modified input signal buffer.

Wav files need to be written in a specific format to be interpreted as audio. Several lines in the file above the data stream are first initialized to set the number of channels, sample rate, and byte rate. Then we write the left and right buffer data consecutively to the file in **little endian** format.

```
/* 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);
```

API Access

To make our *anonymous* phone call we used **Twilio**, a service that allows users to make programmable phone calls and text messages by leveraging their API. We created a script that gets run at the end of our Caller.c program that executes the necessary terminal commands to export our saved WAV file to Github and make an HTTP POST request.

```
curl -X POST "https://api.twilio.com/2010-04-01/Accounts/$TWILIO_ACCOUNT_SID/Calls.json" \
--data-urlencode "ApplicationSid=$TWILIO_APPLICATION_SID" \
--data-urlencode "To=+1$TO_PHONE_NUMBER" \
--data-urlencode "From=+1$FROM_PHONE_NUMBER" \
-u $TWILIO_ACCOUNT_SID:$TWILIO_AUTH_TOKEN
```

Our anonymous audio then gets sent as a phone call to the inputted phone number by a toll free number provided by Twilio.

The Twilio website allows us to set up a request URL containing a TwiML file. TwiML is essentially a XML file which Twilio has setup to perform various calling and texting. We use the <Play> feature to download a file at an asset hosting site, in our case Github, and send it as our phone call. This integration of Twilio's powerful features allows engaging and *secretive* Anonymous calls to be made for our large and growing user base.

```
<?xml version="1.0" encoding="UTF-8"?>
<Response>
 <Play>https://github.com/max-acebal/wavfiles/blob/main/anonymous_audio.wav?raw=true</Play>
</Response>
```

Discussion

Challenges and Solutions

Signed and Fixed Width Numbers

Given poor documentation on the output format of samples from the Wolfson Audio CODEC, we initially falsely assumed that our samples were unsigned integers. In reality, the samples are signed, fixed-point numbers representing sign as

well as fractional values. Most importantly, this caused serious errors in our SystemVerilog code as we did not mark our registers as being signed. This meant that specific operations, specifically right shifting, would corrupt our data. We learned, after much reading of cryptic and hard to find documentation, that SystemVerilog only treats the `>>>` operation as a signed right shift if the input is explicitly marked as signed. Otherwise, it treats it as a normal `>>` operation and does not perform sign extension.

Addition and Overflow

Once we learned that our samples were fixed-point, we quickly realized that the ordering of our mathematical operations was quite important. In our Pitch Shifter module, we average two samples together by adding them and dividing by two. However, since we only use two bits to store the integer part of our samples, it is very easy for the addition to cause an overflow. Thus, we needed to divide each sample by two and add them together to avoid the overflow while performing the same operation.

Interrupts

We spent a considerable amount of time trying to get interrupts to work. We knew that having our hardware module generate interrupts when it had new samples was the 'right' solution, but it was not something that any prior groups had done, or that we had learned in class. This meant we were starting largely from scratch on getting it to work. Unfortunately, linux device drivers are often poorly documented, and we faced a variety of issues in this process. Below are a few such examples:

- To add an interrupt that the ARM core could recognize, we needed to properly mark the signal in Qsys when placing and connecting our modules. The output logic signal from our device driver module needed to be marked as an `Interrupt Provider` and connected directly to the `HPS` and given a specific interrupt width and address.
- Once the interrupt was properly added in the system, it appeared in our `.dts` device tree file, but we struggled to reference that interrupt in our device driver initialization. All interrupts from the FPGA fabric are combined and sent via the `GIC` or Generic Interrupt Controller which modifies the interrupt number before reaching the ARM core. This means that we needed to use the right linux calls to properly ordain the interrupt number to reference rather than using any hard coded number (like you can for some interrupts like those provided from the on board buttons).
- Combing the above learnings, we still struggled to request the interrupt and register our interrupt handler with the kernel. We faced an issue repeatedly causing a kernel panic which required us to completely reboot our system every time we wanted to try a change. This made it extremely difficult to debug. Unfortunately, as often is the case, the issue was rather trivial—we were trying to request the interrupt before we had mapped our virtual base address for our device. This meant that the interrupt was trying to dereference a null pointer. Simply re-ordering the requests we made to the kernel fixed the issue.

Taking all this together, we were able to get our interrupt working perfectly! It was not an easy journey, but we were extremely happy to have it work in the end.

Sleeping and Waking the User Level Program

We knew that we wanted our user level program to place blocking read calls to fetch new samples. However, we did not want our entire process/kernel to pause and wait for interrupts. Thus, we needed to learn how to pause/sleep our user level program when it requested a new sample, and to wake it up later when we received an interrupt and fetched new samples. We learned that this is a somewhat regular need, and can be accomplished with Wait Queue's. In short, we needed to tell the kernel to pause the user level process, placing it in a wait queue, and then later wake up the process from the wait queue after we received new samples. This process took a reasonable amount of research, but we were able to get it working well without significant debugging. Sometimes the Linux kernel gives a set of very helpful tools! Without the wait queue, we had unintentionally read in the same samples repeatedly, making pitch lower than we intended and rendering the audible unintelligible.

Microphone Pre-Amp and Clipping

Despite the Mic In Port on the FPGA providing a small level of amplification to the input signal, the only mic we had available required an external power amplifier. This mic and added amplification, caused unexpected noise to our system. To mitigate this issue we tried varying the output levels of the mic to still produce audio without adding too much extra noise, we did this by lowering the volume knobs on the amplifier. However, our best efforts still result in a small amount of clipping even at low volumes. The audio still sounds understandable and has the intended effect, but a low powered mic could have saved us a lot time trying various volume levels.

Valid Pitch Factors

Though we had run an extensive set of tests of our pitch shifter module in a Matlab simulation, we did not expect how much various sources of noise would impact which pitch shift factors would be understandable in practice. While we had a rough sense of ranges that we thought would work based on our simulation, it turned out that a much smaller range was feasible in hardware without more sophisticated filtering or other methods to reduce noise. In the end, we needed to perform extensive manual testing to learn which shift factors worked in practice and not just in theory.

Uploading Assets to Twilio

When it came to uploading assets to Twilio on our machine, we ran into compatibility issues since Twilio can not be installed locally on our 32bit operating systems. We solved this issue by hosting our WAV file asset on Github and calling a TwiML XML by using a Twilio bin. This way, we were able to run all the code by simply using a HTTP POST request.

Avalon Bus Width

We originally had set the Avalon bus width to 64 bits so that we might send two 24 bit samples at once. We quickly realized that this was completely nonsensical. The ARM core on the DE1-SoC is a 32 bit processor. It has no way to reasonably understand or process a 64 bit input. Thus, we switched the bus width to 32 bits and sent our left and right samples separately in consecutive read operations.

Debugging Qsys

We ran into several issues while integrating our components in Qsys to construct the full system. One rather confusing issue was a compilation error stating that several cryptic wires were grounded but not connected to a calibrated low. After wracking our brains for several hours trying to fix this issue, we found a post on a Japanese forum, which after translating, instructed us to run two TCL scripts buried in the Qsys program. This immediately, and permanently fixed our issue. Documentation is often hard to come by, so we learned to be appreciative of the random nuggets of gold found on the internet.

Future Improvements

There are several features we could add to enhance our NoCallerID system. Firstly, introducing a robust noise reduction filter could reduce unwanted background noise for the call. Low pass filters, like a Butterworth, could remove the unwanted boomy low frequencies. Additionally, integrating a range of hardware/software-controlled audio effects, such as distortion or a chorus, would allow for a wider spectrum of calls and give our system much more customizability. These effects could be easily adjusted using a web application to be user friendly. Lastly, we could further improve the user experience by implementing an actual phone and keypad interface for our system. By combining the familiarity of a traditional phone call interface with the exciting possibilities of hardware/software-controlled audio effects, users could have fun injecting their personality or anonymity into their conversations.

Team Roles

Max Acebal (mra2180)

My main role of the team was debugging the HW/SW interface, WAV file construction, and using the Twilio API to make the phone call using the WAV file. Although my primary focus was integrating many of the software pieces of the project, I had a hand in debugging code from every other section, as well as, creating hardware connections on Qsys with other teammates. Some advice I would offer to future teams would be to get started connecting the whole system as fast as possible. While it was relatively straightforward to work on a single isolated part of the project, the real task was getting everything to work together. Try to understand not just your role in the project, but each of your team members roles too. This way you can help each other debug and find solutions much quicker.

Marian Abuhazi (ma4107)

My main role in the project was in relation to the Pitch Shifter module, including:

1. Researching how to effectively pitch-shift an audio signal without extensive memory usage and complex algorithms
2. Writing MATLAB simulations for the algorithm
3. Writing the hardware description files for the pitch-shifting algorithm
4. Writing testbenches to ensure the pitch-shifter performed as expected before implementing it into the larger system
5. Investigating how to implement a Butterworth low-pass filter
6. Extensive debugging in collaboration with other team members to combine all modules into a cohesive end-to-end solution

To future teams, I highly recommend simulating your hardware algorithms in software first using languages like C or MATLAB. It is important to be very aware of what kind of data you are dealing with, and simulate it in software as closely as possible. For example, if you are working with binary numbers that are in a specific notation such as Q-notation, or floating point etc, you want to be mindful of how the arithmetic in hardware is different than in software and edit your software simulations to act this way. I also recommend writing clear testbenches to run hardware simulations and ensuring that your outputs, states and timings match your expectations and the expectations of the modules that will follow.

Rebekah Kim (rmk2160)

I have mainly worked on:

- Device driver
- Interrupts
- General debugging and design

I would suggest carefully thinking about the interfaces between modules, how the wires are connected and what data is expected and when. Integration between modules will be the most time-consuming part, so getting the interface down solid and integrating and testing early will be helpful.

Noah Silverstein (nis2116)

We generally shared work across our team evenly, often working collaboratively to write and debug code. That being said, I did focus my efforts on a few areas in particular:

- Connecting our hardware modules to our linux device driver, writing the last module on the hardware side that communicates with the Avalon bus, and helping write the linux device driver that knows how to handle incoming interrupts and read data from our device.
- Connecting everything in Qsys and debugging our compilation, ensuring we had access to the right libraries and files so that our design could compile.
- Debugging software compilation and kernel issues.

For future teams, I strongly recommend heeding Prof. Edwards advice—know as early as possible how each and every bit is stored, transmitted, and modified. You will likely hear this from many groups, but integration is 90% of the project. Knowing the interfaces and protocols for how each component of your broader system connect and communicate is vitally important. And more than anything, start connecting your components and testing as soon as possible. You will spend significantly more time than you expect debugging, so try to get to the debugging as soon as possible. Finally, debugging in hardware, and low level software, is very challenging. Expect that this will take more time and effort than you are used to if you have done more software work in the past. Run simulations, think carefully about your each and every line beforehand, and so on to reduce the number of iterations of debugging required. And above all else, have fun! This will not be an easy project or a quick project, but if you have a good group, it can be a blast. Enjoy it, and be proud of what you accomplish!

References & Resources

Name	Link
WAV File Construction	
Twilio Voice API	
Wolfson WM8731 Audio Codec	
Cornell ECE 576, DSP Examples	
Cornell Pitch Shifter Project	
FPGA Pitch Shifter	
Univ. of Toronto DE1-SoC Interfaces Tutorial	
Univ. of Washington DE1-SoC Audio Tutorial	
Altera Genertic Interrupt Controller	
Yonsei University ARM Interrupt Handling Slides	
Bootlin Linux Device Driver Slides	
O-Reilly Sleeping and Waking Linux Processes	

Appendix

Hardware Code

hardware/Makefile

```
SYSTEM = soc_system

TCL = $(SYSTEM).tcl
QSYS = $(SYSTEM).qsys
SOPCINFO = $(SYSTEM).sopcinfo
QIP = $(SYSTEM)/synthesis/$(SYSTEM).qip
HPS_PIN_TCL = $(SYSTEM)/synthesis/submodules/hps_sdram_p0_pin_assignments.tcl
HPS_PIN_MAP = hps_sdram_p0_all_pins.txt
QPF = $(SYSTEM).qpf
QSF = $(SYSTEM).qsf
SDC = $(SYSTEM).sdc

BOARD_INFO = $(SYSTEM)_board_info.xml
DTS = $(SYSTEM).dts
DTB = $(SYSTEM).dtb

SOF = output_files/$(SYSTEM).sof
RBF = output_files/$(SYSTEM).rbf

SOFTWARE_DIR = software

BSP_DIR = $(SOFTWARE_DIR)/spl_bsp
BSP_SETTINGS = $(BSP_DIR)/settings.bsp
PRELOADER_SETTINGS_DIR = hps_isw_handoff/soc_system_hps_0
PRELOADER_MAKEFILE = $(BSP_DIR)/Makefile
PRELOADER_MKPIMAGE = $(BSP_DIR)/preloader-mkpimage.bin

UBOOT_IMAGE = $(BSP_DIR)/uboot-socfpga/u-boot.img

KERNEL_REPO = https://github.com/altera-opensource/linux-socfpga.git
KERNEL_BRANCH = socfpga-4.19
DEFAULT_CONFIG = socfpga_defconfig
CROSS = env CROSS_COMPILE=arm-altera-eabi- ARCH=arm

KERNEL_DIR = $(SOFTWARE_DIR)/linux-socfpga
KERNEL_CONFIG = $(KERNEL_DIR)/.config
ZIMAGE = $(KERNEL_DIR)/arch/arm/boot/zImage

TARFILES = Makefile \
 $(TCL) \
 $(QSYS) \
 $(SYSTEM)_top.sv \
 $(BOARD_INFO) \
```

```
 ip/intr_capturer/intr_capturer.v \
 ip/intr_capturer/intr_capturer_hw.tcl \

TARFILE = project-hw.tar.gz

# project
#
# Run the topmost tcl script to generate the initial project files

.PHONY : project
project : $(QPF) $(QSF) $(SDC)

$(QPF) $(QSF) $(SDC) : $(TCL)
 quartus_sh -t $(TCL)

# qsys
#
# From the .qsys file, generate the .sopcinfo, .qip, and directory
# (named according to the system) with all the Verilog files, etc.

.PHONY : qsys
qsys : $(SOPCINFO)

$(SOPCINFO) $(QIP) $(HPS_PIN_TCL) $(SYSTEM)/ $(PRELOADER_SETTINGS_DIR) : $(QSYS)
 rm -rf $(SOPCINFO) $(SYSTEM)/
 qsys-generate $(QSYS) --synthesis=VERILOG

# quartus
#
# Run Quartus on the Qsys-generated files
#
#    Build netlist
# quartus_map soc_system
#
#    Use netlist information to determine HPS stuff
# quartus_sta -t hps_sdram_p0_pin_assignments.tcl soc_system
#
#    Do the rest
#   FIXME: this is wasteful.  Really want not to repeat the "map" step
# quartus_sh --flow compile
#
# quartus_fit
# quartus_asm
# quartus_sta

.PHONY : quartus
quartus : $(SOF)

$(SOF) $(HPS_PIN_MAP) : $(QIP) $(QPF) $(QSF) $(HPS_PIN_TCL)
```

```
 quartus_map $(SYSTEM)
 quartus_sta -t $(HPS_PIN_TCL) $(SYSTEM)
 quartus_fit $(SYSTEM)
 quartus_asm $(SYSTEM)
# quartus_sh --flow compile $(QPF)

# rbf
#
# Convert the .sof file (for programming through the USB blaster)
# to an .rbf file to be placed on an SD card and written by u-boot
.PHONY : rbf
rbf : $(RBF)

$(RBF) : $(SOF)
 quartus_cpf -c $(SOF) $(RBF)

# dtb
#
# Use the .sopcinfo file to generate a device tree blob file
# with information about the memory map of the peripherals
.PHONY : dtb
dtb : $(DTB)

$(DTB) : $(DTS)
 @which dtc || (echo "dtc not found.  Did you run embedded_command_shell.sh?"; exit 1)
 dtc -I dts -O dtb -o $(DTB) $(DTS)

$(DTS) : $(SOPCINFO) $(BOARD_INFO)
 @which sopc2dts || (echo "sopc2dts not found.  Did you run embedded_command_shell.sh?"; exit 1)
 sopc2dts --input $(SOPCINFO) \
  --output $(DTS) \
  --type dts \
  --board $(BOARD_INFO) \
  --clocks

# preloader
#
# Builds the SPL's preloader-mkpimage.bin image file, which should
# be written to the "magic" 3rd parition on the SD card
# in software/spl_bsp
#
# Requires the embedded_command_shell.sh script to have run so the compiler,
# etc is available
#
.PHONY : preloader
preloader : $(PRELOADER_MKPIMAGE)

$(PRELOADER_MKPIMAGE) : $(PRELOADER_MAKEFILE) $(BSP_SETTINGS)
 $(MAKE) -C $(BSP_DIR)
```

```
$(BSP_SETTINGS) $(PRELOADER_MAKEFILE) : $(PRELOADER_SETTINGS_DIR)
 mkdir -p $(BSP_DIR)
 bsp-create-settings \
  --type spl \
  --bsp-dir $(BSP_DIR) \
  --settings $(BSP_SETTINGS) \
  --preloader-settings-dir $(PRELOADER_SETTINGS_DIR) \
  --set spl.boot.FAT_SUPPORT 1

# uboot
#
# Build the bootloader

.PHONY : uboot
uboot : $(UBOOT_IMAGE)

$(UBOOT_IMAGE) : $(PRELOADER_MAKEFILE) $(BSP_SETTINGS)
 $(MAKE) -C $(BSP_DIR) uboot

# kernel-download
#
# Clone the Linux kernel repository
#
# kernel-config
#
#   Set up the default kernel configuration
#
# kernel-menuconfig
#
#   (Optional) Access the kernel configuration menu to make further
#   adjustments about which modules are included
#
# zimage
#
#   Compile the kernel

.PHONY : download-kernel config-kernel zimage
kernel-download : $(KERNEL_DIR)
kernel-config : $(KERNEL_CONFIG)
kernel-menuconfig :
 $(CROSS) $(MAKE) -C $(KERNEL_DIR) menuconfig
zimage : $(ZIMAGE)

$(KERNEL_DIR) :
 mkdir -p $(KERNEL_DIR)
 git clone --branch $(KERNEL_BRANCH) $(KERNEL_REPO) $(KERNEL_DIR)

# Configure the kernel.  Start from a provided default,
```

```
#
# Turn off version checking (makes it easier to compile kernel
# modules and not have them complain about version)
#
# Turn on large file (+2TB) support, which the ext4 filesystem
# requires by default (it will not be able to mount the root
# filesystem read/write otherwise)
$(KERNEL_CONFIG) : $(KERNEL_DIR)
 $(CROSS) $(MAKE) -C $(KERNEL_DIR) $(DEFAULT_CONFIG)
 $(KERNEL_DIR)/scripts/config --file $(KERNEL_CONFIG) \
   --disable CONFIG_LOCALVERSION_AUTO \
   --enable CONFIG_LBDAF \
   --disable CONFIG_XFS_FS \
   --disable CONFIG_GFS2_FS \
   --disable CONFIG_TEST_KMOD

# Compile the kernel

$(ZIMAGE) : $(KERNEL_CONFIG)
 $(CROSS) $(MAKE) -C $(KERNEL_DIR) LOCALVERSION= zImage

# tar
#
# Build soc_system.tar.gz

.phony : tar
tar : $(TARFILE)

$(TARFILE) : $(TARFILES)
 tar zcfC $(TARFILE) .. $(TARFILES:%=project-hw/%)

# clean
#
# Remove all generated files

.PHONY : clean quartus-clean qsys-clean project-clean
clean : quartus-clean qsys-clean project-clean dtb-clean preloader-clean \
 uboot-clean

project-clean :
 rm -rf $(QPF) $(QSF) $(SDC)

qsys-clean :
 rm -rf $(SOPCINFO) $(QIP) $(SYSTEM)/ .qsys_edit \
 hps_isw_handoff/ hps_sdram_p0_summary.csv

quartus-clean :
 rm -rf  $(SOF) output_files db incremental_db $(SYSTEM).qdf \
 c5_pin_model_dump.txt $(HPS_PIN_MAP)
```

```
dtb-clean :
 rm -rf $(DTS) $(DTB)

preloader-clean :
 rm -rf $(BSP_DIR)

uboot-clean :
 rm -rf $(BSP_DIR)/uboot-socfpga

kernel-clean :
 rm -rf $(KERNEL_DIR)

config-clean :
 rm -rf $(KERNEL_CONFIG)
```

hardware/soc_system_top.sv

```
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Originally modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ====================================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
```

```
//                  web: http://www.terasic.com/
//                  email: support@terasic.com
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,


  ///////// VGA /////////
  output [7:0]  VGA_B,
  output        VGA_BLANK_N,
  output        VGA_CLK,
  output [7:0]  VGA_G,
  output        VGA_HS,
  output [7:0]  VGA_R,
  output        VGA_SYNC_N,
  output        VGA_VS
);

   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 ),

.audio_aud_xclk (AUD_XCK),
.audio_aud_bclk (AUD_BCLK),
.audio_adclrck (AUD_ADCLRCK),
.audio_adcdat (AUD_ADCDAT),
.audio_daclrck (AUD_DACLRCK),
```

```
    .audio_dacdat (AUD_DACDAT),
    .audio_sclk (FPGA_I2C_SCLK),
    .audio_sdat (FPGA_I2C_SDAT)
  );

  // 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 DRAM_ADDR = { 13{ SW[0] } };
  assign DRAM_BA = { 2{ SW[0] } };
  assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'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 GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
  assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'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
```

hardware/soc_system.qsys

```
<?xml version="1.0" encoding="UTF-8"?>
<system name="$${FILENAME}">
 <component
```

```
  name="$${FILENAME}"
  displayName="$${FILENAME}"
  version="1.0"
  description=""
  tags=""
  categories="System" />
 <parameter name="bonusData"><![CDATA[bonusData
{
  element clk_0
  {
    datum _sortIndex
    {
      value = "0";
      type = "int";
    }
  }
  element codec_interface_0
  {
    datum _sortIndex
    {
      value = "1";
      type = "int";
    }
  }
  element driver_interface_0
  {
    datum _sortIndex
    {
      value = "3";
      type = "int";
    }
  }
  element hps_0
  {
    datum _sortIndex
    {
      value = "4";
      type = "int";
    }
  }
  element pitch_shifter_0
  {
    datum _sortIndex
    {
      value = "2";
      type = "int";
    }
  }
  element soc_system
```

```
{
  datum _originalDeviceFamily
  {
    value = "Cyclone V";
    type = "String";
  }
}
element soc_system
{
  datum _originalDeviceFamily
  {
    value = "Cyclone V";
    type = "String";
  }
}
element soc_system
{
  datum _originalDeviceFamily
  {
    value = "Cyclone V";
    type = "String";
  }
}
element soc_system
{
  datum _originalDeviceFamily
  {
    value = "Cyclone V";
    type = "String";
  }
}
element soc_system
{
  datum _originalDeviceFamily
  {
    value = "Cyclone V";
    type = "String";
  }
}
element soc_system
{
  datum _originalDeviceFamily
  {
    value = "Cyclone V";
    type = "String";
  }
}
element soc_system
{
```

```
    datum _originalDeviceFamily
    {
      value = "Cyclone V";
      type = "String";
    }
  }
  element soc_system
  {
    datum _originalDeviceFamily
    {
      value = "Cyclone V";
      type = "String";
    }
  }
  element soc_system
  {
    datum _originalDeviceFamily
    {
      value = "Cyclone V";
      type = "String";
    }
  }
  element soc_system
  {
    datum _originalDeviceFamily
    {
      value = "Cyclone V";
      type = "String";
    }
  }
}
]]></parameter>
 <parameter name="clockCrossingAdapter" value="HANDSHAKE" />
 <parameter name="device" value="5CSEMA5F31C6" />
 <parameter name="deviceFamily" value="Cyclone V" />
 <parameter name="deviceSpeedGrade" value="6" />
 <parameter name="fabricMode" value="QSYS" />
 <parameter name="generateLegacySim" value="false" />
 <parameter name="generationId" value="0" />
 <parameter name="globalResetBus" value="false" />
 <parameter name="hdlLanguage" value="VERILOG" />
 <parameter name="hideFromIPCatalog" value="false" />
 <parameter name="lockedInterfaceDefinition" value="" />
 <parameter name="maxAdditionalLatency" value="1" />
 <parameter name="projectName" value="soc_system.qpf" />
 <parameter name="sopcBorderPoints" value="false" />
 <parameter name="systemHash" value="0" />
 <parameter name="testBenchDutName" value="" />
 <parameter name="timeStamp" value="0" />
```

```
<parameter name="useTestBenchNamingPattern" value="false" />
<instanceScript></instanceScript>
<interface
 name="audio"
 internal="codec_interface_0.audio"
 type="conduit"
 dir="end" />
<interface name="clk" internal="clk_0.clk_in" type="clock" dir="end" />
<interface name="hps" internal="hps_0.hps_io" type="conduit" dir="end" />
<interface name="hps_ddr3" internal="hps_0.memory" type="conduit" dir="end" />
<interface name="reset" internal="clk_0.clk_in_reset" type="reset" dir="end" />
<module name="clk_0" kind="clock_source" version="21.1" enabled="1">
 <parameter name="clockFrequency" value="50000000" />
 <parameter name="clockFrequencyKnown" value="true" />
 <parameter name="inputClockFrequency" value="0" />
 <parameter name="resetSynchronousEdges" value="NONE" />
</module>
<module
 name="codec_interface_0"
 kind="codec_interface"
 version="1.0"
 enabled="1" />
<module
 name="driver_interface_0"
 kind="driver_interface"
 version="1.0"
 enabled="1">
 <parameter name="DATA_SIZE" value="24" />
</module>
<module name="hps_0" kind="altera_hps" version="21.1" enabled="1">
 <parameter name="ABSTRACT_REAL_COMPARE_TEST" value="false" />
 <parameter name="ABS_RAM_MEM_INIT_FILENAME" value="meminit" />
 <parameter name="ACV_PHY_CLK_ADD_FR_PHASE" value="0.0" />
 <parameter name="AC_PACKAGE_DESKEW" value="false" />
 <parameter name="AC_ROM_USER_ADD_0" value="0_0000_0000_0000" />
 <parameter name="AC_ROM_USER_ADD_1" value="0_0000_0000_1000" />
 <parameter name="ADDR_ORDER" value="0" />
 <parameter name="ADD_EFFICIENCY_MONITOR" value="false" />
 <parameter name="ADD_EXTERNAL_SEQ_DEBUG_NIOS" value="false" />
 <parameter name="ADVANCED_CK_PHASES" value="false" />
 <parameter name="ADVERTIZE_SEQUENCER_SW_BUILD_FILES" value="false" />
 <parameter name="AFI_DEBUG_INFO_WIDTH" value="32" />
 <parameter name="ALTMEMPHY_COMPATIBLE_MODE" value="false" />
 <parameter name="AP_MODE" value="false" />
 <parameter name="AP_MODE_EN" value="0" />
 <parameter name="AUTO_DEVICE_SPEEDGRADE" value="6" />
 <parameter name="AUTO_PD_CYCLES" value="0" />
 <parameter name="AUTO_POWERDN_EN" value="false" />
 <parameter name="AVL_DATA_WIDTH_PORT" value="32,32,32,32,32,32" />
```

```
  <parameter name="AVL_MAX_SIZE" value="4" />
  <parameter name="BONDING_OUT_ENABLED" value="false" />
  <parameter name="BOOTFROMFPGA_Enable" value="false" />
  <parameter name="BSEL" value="1" />
  <parameter name="BSEL_EN" value="false" />
  <parameter name="BYTE_ENABLE" value="true" />
  <parameter name="C2P_WRITE_CLOCK_ADD_PHASE" value="0.0" />
  <parameter name="CALIBRATION_MODE" value="Skip" />
  <parameter name="CALIB_REG_WIDTH" value="8" />
  <parameter name="CAN0_Mode" value="N/A" />
  <parameter name="CAN0_PinMuxing" value="Unused" />
  <parameter name="CAN1_Mode" value="N/A" />
  <parameter name="CAN1_PinMuxing" value="Unused" />
  <parameter name="CFG_DATA_REORDERING_TYPE" value="INTER_BANK" />
  <parameter name="CFG_REORDER_DATA" value="true" />
  <parameter name="CFG_TCCD_NS" value="2.5" />
  <parameter name="COMMAND_PHASE" value="0.0" />
  <parameter name="CONTROLLER_LATENCY" value="5" />
  <parameter name="CORE_DEBUG_CONNECTION" value="EXPORT" />
  <parameter
name="CPORT_TYPE_PORT">Bidirectional,Bidirectional,Bidirectional,Bidirectional,Bidirectional,Bidirectional</parameter>
  <parameter name="CSEL" value="0" />
  <parameter name="CSEL_EN" value="false" />
  <parameter name="CTI_Enable" value="false" />
  <parameter name="CTL_AUTOPCH_EN" value="false" />
  <parameter name="CTL_CMD_QUEUE_DEPTH" value="8" />
  <parameter name="CTL_CSR_CONNECTION" value="INTERNAL_JTAG" />
  <parameter name="CTL_CSR_ENABLED" value="false" />
  <parameter name="CTL_CSR_READ_ONLY" value="1" />
  <parameter name="CTL_DEEP_POWERDN_EN" value="false" />
  <parameter name="CTL_DYNAMIC_BANK_ALLOCATION" value="false" />
  <parameter name="CTL_DYNAMIC_BANK_NUM" value="4" />
  <parameter name="CTL_ECC_AUTO_CORRECTION_ENABLED" value="false" />
  <parameter name="CTL_ECC_ENABLED" value="false" />
  <parameter name="CTL_ENABLE_BURST_INTERRUPT" value="false" />
  <parameter name="CTL_ENABLE_BURST_TERMINATE" value="false" />
  <parameter name="CTL_HRB_ENABLED" value="false" />
  <parameter name="CTL_LOOK_AHEAD_DEPTH" value="4" />
  <parameter name="CTL_SELF_REFRESH_EN" value="false" />
  <parameter name="CTL_USR_REFRESH_EN" value="false" />
  <parameter name="CTL_ZQCAL_EN" value="false" />
  <parameter name="CUT_NEW_FAMILY_TIMING" value="true" />
  <parameter name="DAT_DATA_WIDTH" value="32" />
  <parameter name="DEBUGAPB_Enable" value="false" />
  <parameter name="DEBUG_MODE" value="false" />
  <parameter name="DEVICE_DEPTH" value="1" />
  <parameter name="DEVICE_FAMILY_PARAM" value="" />
  <parameter name="DISABLE_CHILD_MESSAGING" value="false" />
  <parameter name="DISCRETE_FLY_BY" value="true" />
```

```
<parameter name="DLL_SHARING_MODE" value="None" />
<parameter name="DMA_Enable">No,No,No,No,No,No,No,No</parameter>
<parameter name="DQS_DQSN_MODE" value="DIFFERENTIAL" />
<parameter name="DQ_INPUT_REG_USE_CLKN" value="false" />
<parameter name="DUPLICATE_AC" value="false" />
<parameter name="ED_EXPORT_SEQ_DEBUG" value="false" />
<parameter name="EMAC0_Mode" value="N/A" />
<parameter name="EMAC0_PTP" value="false" />
<parameter name="EMAC0_PinMuxing" value="Unused" />
<parameter name="EMAC1_Mode" value="RGMII" />
<parameter name="EMAC1_PTP" value="false" />
<parameter name="EMAC1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="ENABLE_ABS_RAM_MEM_INIT" value="false" />
<parameter name="ENABLE_BONDING" value="false" />
<parameter name="ENABLE_BURST_MERGE" value="false" />
<parameter name="ENABLE_CTRL_AVALON_INTERFACE" value="true" />
<parameter name="ENABLE_DELAY_CHAIN_WRITE" value="false" />
<parameter name="ENABLE_EMIT_BFM_MASTER" value="false" />
<parameter name="ENABLE_EXPORT_SEQ_DEBUG_BRIDGE" value="false" />
<parameter name="ENABLE_EXTRA_REPORTING" value="false" />
<parameter name="ENABLE_ISS_PROBES" value="false" />
<parameter name="ENABLE_NON_DESTRUCTIVE_CALIB" value="false" />
<parameter name="ENABLE_NON_DES_CAL" value="false" />
<parameter name="ENABLE_NON_DES_CAL_TEST" value="false" />
<parameter name="ENABLE_SEQUENCER_MARGINING_ON_BY_DEFAULT" value="false" />
<parameter name="ENABLE_USER_ECC" value="false" />
<parameter name="EXPORT_AFI_HALF_CLK" value="false" />
<parameter name="EXTRA_SETTINGS" value="" />
<parameter name="F2H_AXI_CLOCK_FREQ" value="50000000" />
<parameter name="F2H_SDRAM0_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM1_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM2_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM3_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM4_CLOCK_FREQ" value="100" />
<parameter name="F2H_SDRAM5_CLOCK_FREQ" value="100" />
<parameter name="F2SCLK_COLDRST_Enable" value="false" />
<parameter name="F2SCLK_DBGRST_Enable" value="false" />
<parameter name="F2SCLK_PERIPHCLK_Enable" value="false" />
<parameter name="F2SCLK_PERIPHCLK_FREQ" value="0" />
<parameter name="F2SCLK_SDRAMCLK_Enable" value="false" />
<parameter name="F2SCLK_SDRAMCLK_FREQ" value="0" />
<parameter name="F2SCLK_WARMRST_Enable" value="false" />
<parameter name="F2SDRAM_Type" value="" />
<parameter name="F2SDRAM_Width" value="" />
<parameter name="F2SINTERRUPT_Enable" value="true" />
<parameter name="F2S_Width" value="1" />
<parameter name="FIX_READ_LATENCY" value="8" />
<parameter name="FORCED_NON_LDC_ADDR_CMD_MEM_CK_INVERT" value="false" />
<parameter name="FORCED_NUM_WRITE_FR_CYCLE_SHIFTS" value="0" />
```

```
  <parameter name="FORCE_DQS_TRACKING" value="AUTO" />
  <parameter name="FORCE_MAX_LATENCY_COUNT_WIDTH" value="0" />
  <parameter name="FORCE_SEQUENCER_TCL_DEBUG_MODE" value="false" />
  <parameter name="FORCE_SHADOW_REGS" value="AUTO" />
  <parameter name="FORCE_SYNTHESIS_LANGUAGE" value="" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_RX_CLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_TX_CLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_RX_CLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_TX_CLK_IN" value="100" />
  <parameter
     name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC_PTP_REF_CLOCK"
     value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C0_SCL_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C1_SCL_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C2_SCL_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C3_SCL_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS0_SCLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS1_SCLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB0_CLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB1_CLK_IN" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_GTX_CLK" value="125" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_MD_CLK" value="2.5" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_GTX_CLK" value="125" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_MD_CLK" value="2.5" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C0_CLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C1_CLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C2_CLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C3_CLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_QSPI_SCLK_OUT" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SDIO_CCLK" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM0_SCLK_OUT" value="100" />
  <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM1_SCLK_OUT" value="100" />
  <parameter
name="GPIO_Enable">No,No,No,No,No,No,No,No,No,Yes,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
No,No,No,No,Yes,No,No,No,No,Yes,No,No,No,No,No,No,No,Yes,No,No,No,No,Yes,Yes,No,No,No,No,No,No,Yes,No,No,No,No,No,No,
No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No</parameter>
  <parameter name="GP_Enable" value="false" />
  <parameter name="H2F_AXI_CLOCK_FREQ" value="50000000" />
  <parameter name="H2F_CTI_CLOCK_FREQ" value="100" />
  <parameter name="H2F_DEBUG_APB_CLOCK_FREQ" value="100" />
  <parameter name="H2F_LW_AXI_CLOCK_FREQ" value="50000000" />
  <parameter name="H2F_TPIU_CLOCK_IN_FREQ" value="100" />
  <parameter name="HARD_EMIF" value="true" />
  <parameter name="HCX_COMPAT_MODE" value="false" />
  <parameter name="HHP_HPS" value="true" />
  <parameter name="HHP_HPS_SIMULATION" value="false" />
  <parameter name="HHP_HPS_VERIFICATION" value="false" />
  <parameter name="HLGPI_Enable" value="false" />
  <parameter name="HPS_PROTOCOL" value="DDR3" />
```

```
  <parameter name="I2C0_Mode" value="I2C" />
  <parameter name="I2C0_PinMuxing" value="HPS I/O Set 0" />
  <parameter name="I2C1_Mode" value="I2C" />
  <parameter name="I2C1_PinMuxing" value="HPS I/O Set 0" />
  <parameter name="I2C2_Mode" value="N/A" />
  <parameter name="I2C2_PinMuxing" value="Unused" />
  <parameter name="I2C3_Mode" value="N/A" />
  <parameter name="I2C3_PinMuxing" value="Unused" />
  <parameter name="INCLUDE_BOARD_DELAY_MODEL" value="false" />
  <parameter name="INCLUDE_MULTIRANK_BOARD_DELAY_MODEL" value="false" />
  <parameter name="IS_ES_DEVICE" value="false" />
  <parameter
name="LOANIO_Enable">No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,N
o,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,N
o,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No</parameter>
  <parameter name="LOCAL_ID_WIDTH" value="8" />
  <parameter name="LRDIMM_EXTENDED_CONFIG">0x0000000000000000</parameter>
  <parameter name="LWH2F_Enable" value="true" />
  <parameter name="MARGIN_VARIATION_TEST" value="false" />
  <parameter name="MAX_PENDING_RD_CMD" value="32" />
  <parameter name="MAX_PENDING_WR_CMD" value="16" />
  <parameter name="MEM_ASR" value="Manual" />
  <parameter name="MEM_ATCL" value="Disabled" />
  <parameter name="MEM_AUTO_LEVELING_MODE" value="true" />
  <parameter name="MEM_BANKADDR_WIDTH" value="3" />
  <parameter name="MEM_BL" value="OTF" />
  <parameter name="MEM_BT" value="Sequential" />
  <parameter name="MEM_CK_PHASE" value="0.0" />
  <parameter name="MEM_CK_WIDTH" value="1" />
  <parameter name="MEM_CLK_EN_WIDTH" value="1" />
  <parameter name="MEM_CLK_FREQ" value="400.0" />
  <parameter name="MEM_CLK_FREQ_MAX" value="800.0" />
  <parameter name="MEM_COL_ADDR_WIDTH" value="10" />
  <parameter name="MEM_CS_WIDTH" value="1" />
  <parameter name="MEM_DEVICE" value="MISSING_MODEL" />
  <parameter name="MEM_DLL_EN" value="true" />
  <parameter name="MEM_DQ_PER_DQS" value="8" />
  <parameter name="MEM_DQ_WIDTH" value="32" />
  <parameter name="MEM_DRV_STR" value="RZQ/7" />
  <parameter name="MEM_FORMAT" value="DISCRETE" />
  <parameter name="MEM_GUARANTEED_WRITE_INIT" value="false" />
  <parameter name="MEM_IF_BOARD_BASE_DELAY" value="10" />
  <parameter name="MEM_IF_DM_PINS_EN" value="true" />
  <parameter name="MEM_IF_DQSN_EN" value="true" />
  <parameter name="MEM_IF_SIM_VALID_WINDOW" value="0" />
  <parameter name="MEM_INIT_EN" value="false" />
  <parameter name="MEM_INIT_FILE" value="" />
  <parameter name="MEM_MIRROR_ADDRESSING" value="0" />
  <parameter name="MEM_NUMBER_OF_DIMMS" value="1" />
```

```
<parameter name="MEM_NUMBER_OF_RANKS_PER_DEVICE" value="1" />
<parameter name="MEM_NUMBER_OF_RANKS_PER_DIMM" value="1" />
<parameter name="MEM_PD" value="DLL off" />
<parameter name="MEM_RANK_MULTIPLICATION_FACTOR" value="1" />
<parameter name="MEM_ROW_ADDR_WIDTH" value="15" />
<parameter name="MEM_RTT_NOM" value="RZQ/4" />
<parameter name="MEM_RTT_WR" value="RZQ/4" />
<parameter name="MEM_SRT" value="Normal" />
<parameter name="MEM_TCL" value="11" />
<parameter name="MEM_TFAW_NS" value="30.0" />
<parameter name="MEM_TINIT_US" value="500" />
<parameter name="MEM_TMRD_CK" value="4" />
<parameter name="MEM_TRAS_NS" value="35.0" />
<parameter name="MEM_TRCD_NS" value="13.75" />
<parameter name="MEM_TREFI_US" value="7.8" />
<parameter name="MEM_TRFC_NS" value="260.0" />
<parameter name="MEM_TRP_NS" value="13.75" />
<parameter name="MEM_TRRD_NS" value="7.7" />
<parameter name="MEM_TRTP_NS" value="7.5" />
<parameter name="MEM_TWR_NS" value="15.0" />
<parameter name="MEM_TWTR" value="4" />
<parameter name="MEM_USER_LEVELING_MODE" value="Leveling" />
<parameter name="MEM_VENDOR" value="JEDEC" />
<parameter name="MEM_VERBOSE" value="true" />
<parameter name="MEM_VOLTAGE" value="1.5V DDR3" />
<parameter name="MEM_WTCL" value="8" />
<parameter name="MPU_EVENTS_Enable" value="false" />
<parameter name="MRS_MIRROR_PING_PONG_ATSO" value="false" />
<parameter name="MULTICAST_EN" value="false" />
<parameter name="NAND_Mode" value="N/A" />
<parameter name="NAND_PinMuxing" value="Unused" />
<parameter name="NEXTGEN" value="true" />
<parameter name="NIOS_ROM_DATA_WIDTH" value="32" />
<parameter name="NUM_DLL_SHARING_INTERFACES" value="1" />
<parameter name="NUM_EXTRA_REPORT_PATH" value="10" />
<parameter name="NUM_OCT_SHARING_INTERFACES" value="1" />
<parameter name="NUM_OF_PORTS" value="1" />
<parameter name="NUM_PLL_SHARING_INTERFACES" value="1" />
<parameter name="OCT_SHARING_MODE" value="None" />
<parameter name="P2C_READ_CLOCK_ADD_PHASE" value="0.0" />
<parameter name="PACKAGE_DESKEW" value="false" />
<parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM" value="" />
<parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM_VALID" value="false" />
<parameter name="PHY_CSR_CONNECTION" value="INTERNAL_JTAG" />
<parameter name="PHY_CSR_ENABLED" value="false" />
<parameter name="PHY_ONLY" value="false" />
<parameter name="PINGPONGPHY_EN" value="false" />
<parameter name="PLL_ADDR_CMD_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_FREQ_PARAM" value="0.0" />
```

```
<parameter name="PLL_ADDR_CMD_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_ADDR_CMD_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_CLK_PARAM_VALID" value="false" />
<parameter name="PLL_CONFIG_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_CONFIG_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_CONFIG_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_DR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_DR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_HR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_HR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_LOCATION" value="Top_Bottom" />
<parameter name="PLL_MEM_CLK_DIV_PARAM" value="0" />
```

```
<parameter name="PLL_MEM_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_MEM_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_MEM_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_NIOS_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_SHARING_MODE" value="None" />
<parameter name="PLL_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="POWER_OF_TWO_BUS" value="false" />
<parameter name="PRIORITY_PORT" value="1,1,1,1,1,1" />
<parameter name="QSPI_Mode" value="N/A" />
<parameter name="QSPI_PinMuxing" value="Unused" />
<parameter name="RATE" value="Full" />
<parameter name="RDIMM_CONFIG" value="0000000000000000" />
<parameter name="READ_DQ_DQS_CLOCK_SOURCE" value="INVERTED_DQS_BUS" />
<parameter name="READ_FIFO_SIZE" value="8" />
<parameter name="REFRESH_BURST_VALIDATION" value="false" />
<parameter name="REFRESH_INTERVAL" value="15000" />
<parameter name="REF_CLK_FREQ" value="25.0" />
<parameter name="REF_CLK_FREQ_MAX_PARAM" value="0.0" />
<parameter name="REF_CLK_FREQ_MIN_PARAM" value="0.0" />
<parameter name="REF_CLK_FREQ_PARAM_VALID" value="false" />
<parameter name="S2FCLK_COLDRST_Enable" value="false" />
<parameter name="S2FCLK_PENDINGRST_Enable" value="false" />
<parameter name="S2FCLK_USER0CLK_Enable" value="false" />
<parameter name="S2FCLK_USER1CLK_Enable" value="true" />
<parameter name="S2FCLK_USER1CLK_FREQ" value="50.0" />
<parameter name="S2FCLK_USER2CLK" value="5" />
<parameter name="S2FCLK_USER2CLK_Enable" value="false" />
<parameter name="S2FCLK_USER2CLK_FREQ" value="100.0" />
<parameter name="S2FINTERRUPT_CAN_Enable" value="false" />
<parameter name="S2FINTERRUPT_CLOCKPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_CTI_Enable" value="false" />
```

```
<parameter name="S2FINTERRUPT_DMA_Enable" value="false" />
<parameter name="S2FINTERRUPT_EMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_FPGAMANAGER_Enable" value="false" />
<parameter name="S2FINTERRUPT_GPIO_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CEMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_L4TIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_NAND_Enable" value="false" />
<parameter name="S2FINTERRUPT_OSCTIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_QSPI_Enable" value="false" />
<parameter name="S2FINTERRUPT_SDMMC_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPIMASTER_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPISLAVE_Enable" value="false" />
<parameter name="S2FINTERRUPT_UART_Enable" value="false" />
<parameter name="S2FINTERRUPT_USB_Enable" value="false" />
<parameter name="S2FINTERRUPT_WATCHDOG_Enable" value="false" />
<parameter name="S2F_Width" value="1" />
<parameter name="SDIO_Mode" value="4-bit Data" />
<parameter name="SDIO_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SEQUENCER_TYPE" value="NIOS" />
<parameter name="SEQ_MODE" value="0" />
<parameter name="SKIP_MEM_INIT" value="true" />
<parameter name="SOPC_COMPAT_RESET" value="false" />
<parameter name="SPEED_GRADE" value="7" />
<parameter name="SPIM0_Mode" value="N/A" />
<parameter name="SPIM0_PinMuxing" value="Unused" />
<parameter name="SPIM1_Mode" value="Single Slave Select" />
<parameter name="SPIM1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SPIS0_Mode" value="N/A" />
<parameter name="SPIS0_PinMuxing" value="Unused" />
<parameter name="SPIS1_Mode" value="N/A" />
<parameter name="SPIS1_PinMuxing" value="Unused" />
<parameter name="STARVE_LIMIT" value="10" />
<parameter name="STM_Enable" value="false" />
<parameter name="SYS_INFO_DEVICE_FAMILY" value="Cyclone V" />
<parameter name="TEST_Enable" value="false" />
<parameter name="TIMING_BOARD_AC_EYE_REDUCTION_H" value="0.0" />
<parameter name="TIMING_BOARD_AC_EYE_REDUCTION_SU" value="0.0" />
<parameter name="TIMING_BOARD_AC_SKEW" value="0.03" />
<parameter name="TIMING_BOARD_AC_SLEW_RATE" value="1.0" />
<parameter name="TIMING_BOARD_AC_TO_CK_SKEW" value="0.0" />
<parameter name="TIMING_BOARD_CK_CKN_SLEW_RATE" value="2.0" />
<parameter name="TIMING_BOARD_DELTA_DQS_ARRIVAL_TIME" value="0.0" />
<parameter name="TIMING_BOARD_DELTA_READ_DQS_ARRIVAL_TIME" value="0.0" />
<parameter name="TIMING_BOARD_DERATE_METHOD" value="AUTO" />
<parameter name="TIMING_BOARD_DQS_DQSN_SLEW_RATE" value="2.0" />
<parameter name="TIMING_BOARD_DQ_EYE_REDUCTION" value="0.0" />
<parameter name="TIMING_BOARD_DQ_SLEW_RATE" value="1.0" />
<parameter name="TIMING_BOARD_DQ_TO_DQS_SKEW" value="0.0" />
```

```
<parameter name="TIMING_BOARD_ISI_METHOD" value="AUTO" />
<parameter name="TIMING_BOARD_MAX_CK_DELAY" value="0.03" />
<parameter name="TIMING_BOARD_MAX_DQS_DELAY" value="0.02" />
<parameter name="TIMING_BOARD_READ_DQ_EYE_REDUCTION" value="0.0" />
<parameter name="TIMING_BOARD_SKEW_BETWEEN_DIMMS" value="0.05" />
<parameter name="TIMING_BOARD_SKEW_BETWEEN_DQS" value="0.08" />
<parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MAX" value="0.16" />
<parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MIN" value="0.09" />
<parameter name="TIMING_BOARD_SKEW_WITHIN_DQS" value="0.01" />
<parameter name="TIMING_BOARD_TDH" value="0.0" />
<parameter name="TIMING_BOARD_TDS" value="0.0" />
<parameter name="TIMING_BOARD_TIH" value="0.0" />
<parameter name="TIMING_BOARD_TIS" value="0.0" />
<parameter name="TIMING_TDH" value="65" />
<parameter name="TIMING_TDQSCK" value="255" />
<parameter name="TIMING_TDQSCKDL" value="1200" />
<parameter name="TIMING_TDQSCKDM" value="900" />
<parameter name="TIMING_TDQSCKDS" value="450" />
<parameter name="TIMING_TDQSH" value="0.35" />
<parameter name="TIMING_TDQSQ" value="125" />
<parameter name="TIMING_TDQSS" value="0.25" />
<parameter name="TIMING_TDS" value="30" />
<parameter name="TIMING_TDSH" value="0.2" />
<parameter name="TIMING_TDSS" value="0.2" />
<parameter name="TIMING_TIH" value="140" />
<parameter name="TIMING_TIS" value="180" />
<parameter name="TIMING_TQH" value="0.38" />
<parameter name="TIMING_TQHS" value="300" />
<parameter name="TIMING_TQSH" value="0.4" />
<parameter name="TPIUFPGA_Enable" value="false" />
<parameter name="TPIUFPGA_alt" value="false" />
<parameter name="TRACE_Mode" value="N/A" />
<parameter name="TRACE_PinMuxing" value="Unused" />
<parameter name="TRACKING_ERROR_TEST" value="false" />
<parameter name="TRACKING_WATCH_TEST" value="false" />
<parameter name="TREFI" value="35100" />
<parameter name="TRFC" value="350" />
<parameter name="UART0_Mode" value="No Flow Control" />
<parameter name="UART0_PinMuxing" value="HPS I/O Set 0" />
<parameter name="UART1_Mode" value="N/A" />
<parameter name="UART1_PinMuxing" value="Unused" />
<parameter name="USB0_Mode" value="N/A" />
<parameter name="USB0_PinMuxing" value="Unused" />
<parameter name="USB1_Mode" value="SDR" />
<parameter name="USB1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="USER_DEBUG_LEVEL" value="1" />
<parameter name="USE_AXI_ADAPTOR" value="false" />
<parameter name="USE_FAKE_PHY" value="false" />
<parameter name="USE_MEM_CLK_FREQ" value="false" />
```

```
  <parameter name="USE_MM_ADAPTOR" value="true" />
  <parameter name="USE_SEQUENCER_BFM" value="false" />
  <parameter name="WEIGHT_PORT" value="0,0,0,0,0,0" />
  <parameter name="WRBUFFER_ADDR_WIDTH" value="6" />
  <parameter name="can0_clk_div" value="1" />
  <parameter name="can1_clk_div" value="1" />
  <parameter name="configure_advanced_parameters" value="false" />
  <parameter name="customize_device_pll_info" value="false" />
  <parameter name="dbctrl_stayosc1" value="true" />
  <parameter name="dbg_at_clk_div" value="0" />
  <parameter name="dbg_clk_div" value="1" />
  <parameter name="dbg_trace_clk_div" value="0" />
  <parameter name="desired_can0_clk_mhz" value="100.0" />
  <parameter name="desired_can1_clk_mhz" value="100.0" />
  <parameter name="desired_cfg_clk_mhz" value="50.0" />
  <parameter name="desired_emac0_clk_mhz" value="250.0" />
  <parameter name="desired_emac1_clk_mhz" value="250.0" />
  <parameter name="desired_gpio_db_clk_hz" value="32000" />
  <parameter name="desired_l4_mp_clk_mhz" value="100.0" />
  <parameter name="desired_l4_sp_clk_mhz" value="100.0" />
  <parameter name="desired_mpu_clk_mhz" value="800.0" />
  <parameter name="desired_nand_clk_mhz" value="12.5" />
  <parameter name="desired_qspi_clk_mhz" value="400.0" />
  <parameter name="desired_sdmmc_clk_mhz" value="200.0" />
  <parameter name="desired_spi_m_clk_mhz" value="200.0" />
  <parameter name="desired_usb_mp_clk_mhz" value="200.0" />
  <parameter name="device_name" value="5CSEMA5F31C6" />
  <parameter name="device_pll_info_manual">{320000000 1600000000} {320000000 1000000000} {800000000 400000000
400000000}</parameter>
  <parameter name="eosc1_clk_mhz" value="25.0" />
  <parameter name="eosc2_clk_mhz" value="25.0" />
  <parameter name="gpio_db_clk_div" value="6249" />
  <parameter name="l3_mp_clk_div" value="1" />
  <parameter name="l3_sp_clk_div" value="1" />
  <parameter name="l4_mp_clk_div" value="1" />
  <parameter name="l4_mp_clk_source" value="1" />
  <parameter name="l4_sp_clk_div" value="1" />
  <parameter name="l4_sp_clk_source" value="1" />
  <parameter name="main_pll_c3" value="3" />
  <parameter name="main_pll_c4" value="3" />
  <parameter name="main_pll_c5" value="15" />
  <parameter name="main_pll_m" value="63" />
  <parameter name="main_pll_n" value="0" />
  <parameter name="nand_clk_source" value="2" />
  <parameter name="periph_pll_c0" value="3" />
  <parameter name="periph_pll_c1" value="3" />
  <parameter name="periph_pll_c2" value="1" />
  <parameter name="periph_pll_c3" value="19" />
  <parameter name="periph_pll_c4" value="4" />
```

```
 <parameter name="periph_pll_c5" value="9" />
 <parameter name="periph_pll_m" value="79" />
 <parameter name="periph_pll_n" value="1" />
 <parameter name="periph_pll_source" value="0" />
 <parameter name="qspi_clk_source" value="1" />
 <parameter name="quartus_ini_hps_emif_pll" value="false" />
 <parameter
   name="quartus_ini_hps_ip_enable_all_peripheral_fpga_interfaces"
   value="false" />
 <parameter name="quartus_ini_hps_ip_enable_bsel_csel" value="false" />
 <parameter
   name="quartus_ini_hps_ip_enable_emac0_peripheral_fpga_interface"
   value="false" />
 <parameter
   name="quartus_ini_hps_ip_enable_low_speed_serial_fpga_interfaces"
   value="false" />
 <parameter name="quartus_ini_hps_ip_enable_test_interface" value="false" />
 <parameter name="quartus_ini_hps_ip_f2sdram_bonding_out" value="false" />
 <parameter name="quartus_ini_hps_ip_fast_f2sdram_sim_model" value="false" />
 <parameter name="quartus_ini_hps_ip_suppress_sdram_synth" value="false" />
 <parameter name="sdmmc_clk_source" value="2" />
 <parameter name="show_advanced_parameters" value="false" />
 <parameter name="show_debug_info_as_warning_msg" value="false" />
 <parameter name="show_warning_as_error_msg" value="false" />
 <parameter name="spi_m_clk_div" value="0" />
 <parameter name="usb_mp_clk_div" value="0" />
 <parameter name="use_default_mpu_clk" value="true" />
</module>
<module name="pitch_shifter_0" kind="pitch_shifter" version="1.0" enabled="1">
 <parameter name="BUFFER_SIZE" value="1024" />
 <parameter name="DATA_SIZE" value="24" />
</module>
<connection
  kind="avalon"
  version="21.1"
  start="hps_0.h2f_axi_master"
  end="driver_interface_0.driver_to_avalon">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0000" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
  kind="clock"
  version="21.1"
  start="clk_0.clk"
  end="codec_interface_0.clock" />
<connection
  kind="clock"
  version="21.1"
```

```
    start="clk_0.clk"
    end="driver_interface_0.clock" />
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="pitch_shifter_0.clock" />
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="hps_0.f2h_axi_clock" />
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="hps_0.h2f_axi_clock" />
  <connection
    kind="clock"
    version="21.1"
    start="clk_0.clk"
    end="hps_0.h2f_lw_axi_clock" />
  <connection
    kind="conduit"
    version="21.1"
    start="codec_interface_0.codec_to_pitch_shifter"
    end="pitch_shifter_0.codec_to_pitch_shifter">
    <parameter name="endPort" value="" />
    <parameter name="endPortLSB" value="0" />
    <parameter name="startPort" value="" />
    <parameter name="startPortLSB" value="0" />
    <parameter name="width" value="0" />
  </connection>
  <connection
    kind="conduit"
    version="21.1"
    start="pitch_shifter_0.pitch_shifter_to_driver"
    end="driver_interface_0.pitch_shifter_to_driver">
    <parameter name="endPort" value="" />
    <parameter name="endPortLSB" value="0" />
    <parameter name="startPort" value="" />
    <parameter name="startPortLSB" value="0" />
    <parameter name="width" value="0" />
  </connection>
  <connection
    kind="interrupt"
    version="21.1"
    start="hps_0.f2h_irq0"
    end="driver_interface_0.irq">
```

```
  <parameter name="irqNumber" value="0" />
 </connection>
 <connection
  kind="reset"
  version="21.1"
  start="clk_0.clk_reset"
  end="codec_interface_0.reset" />
 <connection
  kind="reset"
  version="21.1"
  start="clk_0.clk_reset"
  end="driver_interface_0.reset" />
 <connection
  kind="reset"
  version="21.1"
  start="clk_0.clk_reset"
  end="pitch_shifter_0.reset" />
 <interconnectRequirement for="$system" name="qsys_mm.clockCrossingAdapter" value="HANDSHAKE" />
 <interconnectRequirement for="$system" name="qsys_mm.enableEccProtection" value="FALSE" />
 <interconnectRequirement for="$system" name="qsys_mm.insertDefaultSlave" value="FALSE" />
 <interconnectRequirement for="$system" name="qsys_mm.maxAdditionalLatency" value="1" />
</system>
```

hardware/codec_interface/codec_interface.sv

```
/************************************************************************
 * Original file written by Scott Hauck for ECE271 at UW, Winter 2021   *
 ************************************************************************/

/*
This audio driver module provides a simple interface with the DE1 SoC's
audio CODEC. The module is essentially a wrapper for a driver provided
by Altera. The purpose of the wrapper is to make implementation more
straightforward for users of the driver.

CLOCK_50 should be connected to the 50MHz system clock
reset should be connected to the system reset
dac_left and dac_right are 24-bit audio data samples provided by the
 user of the driver which go to the digital-to-analog converter of
 the CODEC.
adc_left and adc_right are 24-bit audio data samples provided by the
 driver which come from the analog-to-digital converter of the CODEC.
advance is used to signal a new sample of audio data. Advance
 transitions on rising edges of CLOCK_50 and will remain low most of
 the time, periodically going high for one clock cycle. Advance
 notifies the user of the driver that there is valid data available
 on adc_left and adc_right until the next rising edge of the clock,
 and that valid data must be provided on dac_left and dac_right by
```

```
 the next rising edge of the clock. Advance pulses will be
 approximately evenly spaced at 48kHz.
FPGA_I2C_SCLK, FPGA_I2C_SDAT, AUD_XCK, AUD_DACLRCK, AUD_ADCLRCK,
 AUD_BCLK, AUD_ADCDAT, and AUD_DACDAT are external connections from
 the FPGA which go to the CODEC. These should be wired directly to
 ports on the top level module.

 Timing diagram:

          __   __   __
CLOCK_50  __|  |__|  |__|  |__
              ___
advance   ____|  |_____
              ___
dac_l/r   --------|__|--------
              ___
adc_l/r   --------|__|--------

 Audio data is represented as PCM samples. Each sample is a 24 bit
 unsigned integer. The sampling rate is 48kHz.
 http://en.wikipedia.org/wiki/Pulse-code_modulation
 */

 `timescale 1 ps / 1 ps

 // synthesis translate_off
 `define DOING_SIMULATION 1
 // synthesis translate_on

 module codec_interface (CLOCK_50, reset, dac_left, dac_right, adc_left, adc_right, advance, FPGA_I2C_SCLK, FPGA_I2C_SDAT,
 AUD_XCK, AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK, AUD_ADCDAT, AUD_DACDAT);
  input CLOCK_50;
  input reset;
  // I2C Audio/Video config interface
  output FPGA_I2C_SCLK;
  inout FPGA_I2C_SDAT;
  // Audio CODEC
  output AUD_XCK;
  input AUD_DACLRCK, AUD_ADCLRCK, AUD_BCLK;
  input AUD_ADCDAT;
  output AUD_DACDAT;
  // Audio data
  input signed [23:0] dac_left, dac_right;
  output signed [23:0] adc_left, adc_right;
  output advance;

  // Local wires.
  wire read_ready, write_ready;
  reg write;
  wire read;
```

```
// Registers are necessary for timing purposes as
//    read_ready and write_ready go high for multiple clock cycles
reg w1, w2;

assign read = write;
assign advance = write;

always @(posedge CLOCK_50) begin
 if (reset) begin
  w1 <= 0;
  w2 <= 0;
  write <= 0;
 end else begin
  w1 <= write_ready & read_ready & !w2;
  w2 <= w1;
  write <= w1 & !w2;
 end
end

wire locked;

// Don't use PLL for simulation
`ifdef DOING_SIMULATION
 reg AUD_XCK;
 initial AUD_XCK = 0;
 always begin
  #(40690);
  AUD_XCK = ~AUD_XCK;
 end
`else
 // PLL produces clock at correct frequency for CODEC
 xck_generator xck_source (.refclk(CLOCK_50),.rst(reset),.outclk_0(AUD_XCK), .locked(locked));
`endif


////////////////////////////////////////////////////////////////////////////////
// Audio CODEC interface.
//
// The interface consists of the following wires:
// read_ready, write_ready - CODEC ready for read/write operation
// readdata_left, readdata_right - left and right channel data from the CODEC
// read - send data from the CODEC (both channels)
// writedata_left, writedata_right - left and right channel data to the CODEC
// write - send data to the CODEC (both channels)
// AUD_* - should connect to top-level entity I/O of the same name.
//        These signals go directly to the Audio CODEC
// I2C_* - should connect to top-level entity I/O of the same name.
//        These signals go directly to the Audio/Video Config module
```

```
////////////////////////////////////////////////////////////////////////////////

 audio_and_video_config cfg(
  // Inputs
  CLOCK_50,
  reset,

  // Bidirectionals
  FPGA_I2C_SDAT,
  FPGA_I2C_SCLK
 );

 audio_codec codec(
  // Inputs
  CLOCK_50,
  reset,

  read, write,
  dac_left, dac_right,

  AUD_ADCDAT,

  // Bidirectionals
  AUD_BCLK,
  AUD_ADCLRCK,
  AUD_DACLRCK,

  // Outputs
  read_ready, write_ready,
  adc_left, adc_right,
  AUD_DACDAT
 );
endmodule
```

hardware/driver_interface/driver_interface.sv

```
module driver_interface
    #(parameter DATA_SIZE = 24)
   (input logic clk, // 50 MHz
    input logic rst,
    input logic [DATA_SIZE-1:0] in_left,
    input logic [DATA_SIZE-1:0] in_right,
    input logic in_ready,
    input logic chipselect,
    input logic [2:0] address,
    input logic read,
    output logic [31:0] read_data,
    output logic irq
```

```
    );

    logic [DATA_SIZE-1:0] left_buffer;
    logic [DATA_SIZE-1:0] right_buffer;

    always_ff @(posedge clk) begin
        if (in_ready) begin
            left_buffer <= in_left;
            right_buffer <= in_right;
            irq <= 1;
        end
        if (chipselect && read) begin
  case(address)
   3'h0 : read_data <= {{32-DATA_SIZE{1'b0}}, left_buffer};
   3'h1 : read_data <= {{32-DATA_SIZE{1'b0}}, right_buffer};
   3'h2 : begin
    irq <= 0;
    read_data <= 32'b1;
   end
  endcase
        end
    end
endmodule
```

hardware/pitch_shifter/buffer.sv

```
module buffer
        #(parameter BUFFER_SIZE = 1024,
        parameter DATA_SIZE = 24)

        (input logic clk,
        input logic wren, // 1: write to buffer, 0: idle
        input logic [$clog2(BUFFER_SIZE)-1:0] wraddr, // write address
        input logic [$clog2(BUFFER_SIZE)-1:0] rdaddr, // read address
        input logic signed [DATA_SIZE-1:0] wrdata, // write data
        output logic signed [DATA_SIZE-1:0] rddata); //read data

 logic signed [DATA_SIZE-1:0] mem [BUFFER_SIZE-1:0]; // Memory array: 1024, 24-bit

 always_ff @(posedge clk)
 begin
  rddata <= mem[rdaddr];
  if(wren) mem[wraddr] <= wrdata;
 end
endmodule
```

hardware/pitch_shifter/pitch_shifter.sv

```
module pitch_shifter
      #(BUFFER_SIZE = 1024, parameter DATA_SIZE = 24)
      (input logic clk, // 50 MHz
      input logic rst,
      input logic signed [DATA_SIZE-1:0] in_left,
      input logic signed [DATA_SIZE-1:0] in_right,
      input logic in_ready,
      output logic signed [DATA_SIZE-1:0] out_left,
      output logic signed [DATA_SIZE-1:0] out_right,
      output logic out_ready
      );

 logic [$clog2(BUFFER_SIZE)+28:0] shift_factor = {{$clog2(BUFFER_SIZE){1'b0}}, 4'b1111, 25'b0}; //0.9375 --> Q10.29
 logic l_wren_0, l_wren_1, r_wren_0, r_wren_1; // Write-enable flags
 logic [$clog2(BUFFER_SIZE)-1:0] l_w_0, l_w_1, // Write pointers
    r_w_0, r_w_1;
 logic [$clog2(BUFFER_SIZE)-1:0] l_r_0, l_r_1, // Read pointers
    r_r_0, r_r_1;
 logic [$clog2(BUFFER_SIZE)+28:0] l_i_0, l_i_1, r_i_0, r_i_1; // Fractional indexes; Q BUFFER_SIZE.29
 logic signed [DATA_SIZE-1:0] l_wrdata_0, l_wrdata_1, r_wrdata_0, r_wrdata_1; // Write data
 logic signed [DATA_SIZE-1:0] l_rddata_0, l_rddata_1, r_rddata_0, r_rddata_1; // Read data

 enum logic [2:0] {WAIT = 3'b000, WRITE = 3'b001, READ = 3'b010, OUTPUT =
  3'b011} state; // State encodings

 buffer #(BUFFER_SIZE, DATA_SIZE) l_buf_0(
  .clk(clk),
  .wren(l_wren_0),
  .wraddr(l_w_0),
  .rdaddr(l_r_0),
  .wrdata(l_wrdata_0),
  .rddata(l_rddata_0)
 );
 buffer #(BUFFER_SIZE, DATA_SIZE) l_buf_1(
  .clk(clk),
  .wren(l_wren_1),
  .wraddr(l_w_1),
  .rdaddr(l_r_1),
  .wrdata(l_wrdata_1),
  .rddata(l_rddata_1)
 );

 buffer #(BUFFER_SIZE, DATA_SIZE) r_buf_0(
  .clk(clk),
  .wren(r_wren_0),
  .wraddr(r_w_0),
  .rdaddr(r_r_0),
  .wrdata(r_wrdata_0),
```

```
 .rddata(r_rddata_0)
);

buffer #(BUFFER_SIZE, DATA_SIZE) r_buf_1(
 .clk(clk),
 .wren(r_wren_1),
 .wraddr(r_w_1),
 .rdaddr(r_r_1),
 .wrdata(r_wrdata_1),
 .rddata(r_rddata_1)
);

always_ff @(posedge clk) begin
 if(rst) begin
  state <= WAIT;

  // Initialize left buffer pointers
  // Write
  l_w_0 <= 0;
  l_w_1 <= BUFFER_SIZE/2;

  // Read
  l_r_0 <= 0;
  l_r_1 <= 0;

  // Initialize right buffer pointers
  // Write
  r_w_0 <= 0;
  r_w_1 <= BUFFER_SIZE/2;

  // Read
  r_r_0 <= 0;
  r_r_1 <= 0;

  // Initialize fractional indexes
  l_i_0 <= 0;
  l_i_1 <= 0;
  r_i_0 <= 0;
  r_i_1 <= 0;

  out_ready <= 0;
 end
 else if(state == WAIT && in_ready) begin
  // Write in_left to left buffer
  l_wren_0 <= 1;
  l_wren_1 <= 1;
  l_wrdata_0 <= in_left;
  l_wrdata_1 <= in_left;
```

```
// Write in_right to right buffer
r_wren_0 <= 1;
r_wren_1 <= 1;
r_wrdata_0 <= in_right;
r_wrdata_1 <= in_right;

state <= WRITE;

out_ready <= 0;

end
else if(state == WRITE) begin
// Do not write, just read
l_wren_0 <= 0;
l_wren_1 <= 0;
r_wren_0 <= 0;
r_wren_1 <= 0;

// Advance write pointers
l_w_0 <= l_w_0 + 1;
l_w_1 <= l_w_1 + 1;
r_w_0 <= r_w_0 + 1;
r_w_1 <= r_w_1 + 1;

// Update fractional indexes
l_i_0 <= l_i_0 + shift_factor;
l_i_1 <= l_i_1 + shift_factor;

r_i_0 <= r_i_0 + shift_factor;
r_i_1 <= r_i_1 + shift_factor;

// Advance read pointers
l_r_0 <= l_i_0[$clog2(BUFFER_SIZE)+28:29]; // Take first 10b (int part of i_0)
r_r_0 <= r_i_0[$clog2(BUFFER_SIZE)+28:29];

l_r_1 <= l_i_1[$clog2(BUFFER_SIZE)+28:29]; // Take first 10b (int part of i_1)
r_r_1 <= r_i_1[$clog2(BUFFER_SIZE)+28:29];

out_ready <= 0;
state <= READ;

end
else if(state == READ) begin
// Reading state
state <= OUTPUT;
end
else if(state == OUTPUT) begin
// Alert driver that data is valiD
out_ready <= 1;
```

```
  state <= WAIT;
 end
 else begin // WAIT
  // Do not write
  l_wren_0 <= 0;
  l_wren_1 <= 0;
  r_wren_0 <= 0;
  r_wren_1 <= 0;

  out_ready <= 0;
  state <= WAIT;
 end
end
 // Take average of outputs of buffers 0 and 1 for left and right buffers
 assign out_left = state == OUTPUT ? (((l_rddata_0 === 24'dx ? 0 : l_rddata_0) >>> 1) + ((l_rddata_1 === 24'dx ? 0 : l_rddata_1) >>>
1)): out_left;
 assign out_right = state == OUTPUT ? (((r_rddata_0 === 24'dx ? 0 : r_rddata_0)  >>> 1) + ((r_rddata_1 === 24'dx ? 0 : r_rddata_1)
>>> 1)): out_right;

endmodule
```

hardware/pitch_shifter/tb_pitch_shifter.sv

```
`define OUT_DATA "../../matlab/out_data"
`define OUT_DATA_SIM "../../matlab/out_data_sim"
`define IN_DATA_SIM "../../matlab/in_data_sim"
module tb_pitch_shifter;
 timeunit 1ns;
 timeprecision 10ps;

 localparam BUFFER_SIZE = 10; // Test with a smaller buffer size
 localparam DATA_SIZE = 24;
 logic clk;
 logic rst;
 logic [DATA_SIZE-1:0] in_left;
 logic [DATA_SIZE-1:0] in_right;
 logic in_ready;
 logic [DATA_SIZE-1:0] out_left;
 logic [DATA_SIZE-1:0] out_right;
 logic out_ready;

 integer i;
 integer out_data;
 integer out_data_sim;
 integer in_data_sim;

 logic [DATA_SIZE-1:0] out_left_sim;
 logic [DATA_SIZE-1:0] out_right_sim;
```

```
integer left_error_count = 0;
integer right_error_count = 0;

pitch_shifter #(BUFFER_SIZE, DATA_SIZE) pitch_shifter_0 (
 .clk(clk),
 .rst(rst),
 .in_left(in_left),
 .out_left(out_left),
 .in_right(in_right),
 .out_right(out_right),
 .in_ready(in_ready),
 .out_ready(out_ready)
);

initial begin
 // File IO
 out_data = $fopen(`OUT_DATA,"w");
 if (!out_data) begin
  $display("Couldn't create the output file.");
  $finish;
 end

 out_data_sim = $fopen(`OUT_DATA_SIM,"r");
 if (!out_data_sim) begin
  $display("Couldn't open the Matlab out file.");
  $finish;
 end

 in_data_sim = $fopen(`IN_DATA_SIM,"r");
 if (!in_data_sim) begin
  $display("Couldn't open the Matlab in file.");
  $finish;
 end
 clk <= 1;
 rst <= 0;
end
always #10 clk = ~clk; // 50 MHz

always begin
 // Reset for 1 clk cycle
 rst <=1;
 #20;
 rst <= 0;

 for (i=0 ; i<512; i=i+1) begin
  // Read the binary inputs from Matlab simulation
  $fscanf(in_data_sim, "%b", in_left);
  $fscanf(in_data_sim, "%b", in_right);
```

```
  in_ready <= 1;

  @(posedge clk);
  // compare w/ the results from Matlab sim
  $fscanf(out_data_sim, "%b", out_left_sim);
  $fscanf(out_data_sim, "%b", out_right_sim);

  #20;
  in_ready <=0;

  #60;

  $fwrite(out_data, "%b\n", out_left);
  $fwrite(out_data, "%b\n", out_right);
  if (out_left!== out_left_sim)
  begin

   left_error_count = left_error_count + 1;
  end
  if (out_right!== out_right_sim)
  begin

   right_error_count = right_error_count + 1;
  end
 end

 // Any mismatch between RTL and MatLab simulations?
 if (left_error_count > 0 || right_error_count > 0) begin
  $display("The results DO NOT match with those from Matlab :( ");
 end
 else if (left_error_count < 15 && right_error_count < 15) begin
  $display("The results DO NOT match with those from Matlab, but error is < 15 and due to shift_factor size");
 end
 else begin
  $display("The results DO match with those from Matlab :) ");
 end

 // Close files
 $fclose(in_data_sim);
 $fclose(out_data_sim);
 $fclose(out_data);
 $finish;

end

endmodule
```

hardware/pitch_shifter/tb_driver_interface.sv

```
`define OUT_DATA "out_data"
`define OUT_DATA_SIM "../../matlab/out_data_sim"
`define IN_DATA_SIM "../../matlab/in_data_sim"

module tb_driver_interface;
 timeunit 1ns;
 timeprecision 10ps;

 // params
 localparam BUFFER_SIZE = 1024;
 localparam DATA_SIZE = 24;

 // shifter flags
 logic clk;
 logic rst;
 logic [DATA_SIZE-1:0] in_left;
 logic [DATA_SIZE-1:0] in_right;
 logic in_ready;
 logic [DATA_SIZE-1:0] out_left;
 logic [DATA_SIZE-1:0] out_right;
 logic out_ready;

 // driver flags
 logic chipselect;
 logic [2:0] address;
 logic read;
 logic [63:0] read_data;
 logic irq;

 integer i;
 integer out_data;
 integer out_data_sim;
 integer in_data_sim;

 logic [DATA_SIZE-1:0] out_left_sim;
 logic [DATA_SIZE-1:0] out_right_sim;

 integer left_error_count = 0;
 integer right_error_count = 0;

 pitch_shifter #(BUFFER_SIZE, DATA_SIZE) pitch_shifter_0 (
  .clk(clk),
  .rst(rst),
  .in_left(in_left),
  .out_left(out_left),
  .in_right(in_right),
  .out_right(out_right),
```

```
 .in_ready(in_ready),
 .out_ready(out_ready)
);

driver_interface #(DATA_SIZE) driver_interface_0 (
 .clk(clk),
 .rst(rst),
 .in_left(out_left),
 .in_right(out_right),
 .in_ready(out_ready),
 .chipselect(chipselect),
 .address(address),
 .read(read),
 .read_data(read_data),
 .irq(irq)
);

initial begin
 // File IO
 out_data = $fopen(`OUT_DATA,"w");
 if (!out_data) begin
  $display("Couldn't create the output file.");
  $finish;
 end

 out_data_sim = $fopen(`OUT_DATA_SIM,"r");
 if (!out_data_sim) begin
  $display("Couldn't open the Matlab out file.");
  $finish;
 end

 in_data_sim = $fopen(`IN_DATA_SIM,"r");
 if (!in_data_sim) begin
  $display("Couldn't open the Matlab in file.");
  $finish;
 end
 clk <= 1;
 rst <= 0;
end

always #10 clk = ~clk; // 50 MHz

always begin
 // Reset for 1 clk cycle
 rst <=1;
 #20;
 rst <= 0;

 for (i=0 ; i<50; i=i+1) begin
```

```
  // Read the binary inputs from Matlab simulation
  $fscanf(in_data_sim, "%b", in_left);
  $fscanf(in_data_sim, "%b", in_right);
  in_ready <= 1;

  @(posedge clk);
  #20;
  in_ready <=0;

  #60;
  // compare w/ the results from Matlab sim
  $fscanf(out_data_sim, "%b", out_left_sim);
  $fscanf(out_data_sim, "%b", out_right_sim);

  #20;
  chipselect <= 1;
  read <= 1;
  address <= 0;

  #20;
  chipselect <= 0;
  read <= 0;
  address <= 0;

 end

 // Close files
 $fclose(in_data_sim);
 $fclose(out_data);
 $finish;
end
endmodule
```

Software Code

software/Makefile

```
ifneq (${KERNELRELEASE},)

# KERNELRELEASE defined: we are being compiled as part of the Kernel
obj-m := audio.o

else

# We are being compiled as a module: use the Kernel build system
KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
PWD := $(shell pwd)/driver

default: module caller
```

```
module:
 ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

make_wav.o:
 cc -c wavfile_construction/make_wav.c wavfile_construction/make_wav.h

caller.o:
 cc -c caller.c

caller: make_wav.o caller.o
 cc -o caller caller.o make_wav.o

clean:
 ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
 ${RM} caller caller.o make_wav.o

endif
```

software/caller.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 "driver/audio.h"

#define S_RATE  (44100)
#define REAL_S_RATE (44100)
#define BUF_SIZE (S_RATE*5*2) /* 5 second buffer for L/R */

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");
      return;
   }

   buffer[idx++] = vla.samples.l;
```

```
    buffer[idx++] = vla.samples.r;
}

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

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

    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){
        read_samples();
  }

  printf("sample read done, before write_wav");
  for (int i = 100; i < 150; i++)
   printf("samp: %lu\n", buffer[i]);
    write_wav("./wavfiles/anonymous_audio.wav", BUF_SIZE, buffer, S_RATE);

    system("./twilio/place_call.sh");

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

software/driver/Makefile

```
ifneq (${KERNELRELEASE},)

# KERNELRELEASE defined: we are being compiled as part of the Kernel
obj-m := audio.o

else

audio.o:
  cc -c audio.c audio.h

endif
```

software/driver/audio.h

```
#ifndef _AUDIO_H
#define _AUDIO_H

#include <linux/ioctl.h>

typedef struct {
  unsigned int l, r;
} audio_samples_t;

typedef struct {
  int audio_ready;
} audio_ready_t;

typedef struct {
  audio_samples_t samples;
  audio_ready_t ready;
} audio_arg_t;

#define AUDIO_MAGIC 'q'

/* ioctls and their arguments */
#define AUDIO_READ_SAMPLES _IOR(AUDIO_MAGIC, 1, audio_arg_t *)

#endif
```

software/driver/audio.c

```
/* * Device driver for the Audio device
 *
 * A Platform device implemented using the misc subsystem
 *
 * Based on code from Stephen A. Edwards, Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod audio.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree audio.c
 */

#include <linux/module.h>
#include <linux/init.h>
```

```c
#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 <linux/sched.h>
#include <linux/interrupt.h>
#include <linux/of_irq.h>
#include "audio.h"

#define DRIVER_NAME "audio"

// Initialize a wait queue to sleep user level process until irq raised
DECLARE_WAIT_QUEUE_HEAD(wq);

/* Device registers */
#define L_SAMPLES(x) (x)
#define R_SAMPLES(x) ((x) + 4)
#define RESET_IRQ(x) ((x) + 8)

/*
 * Information about our device
 */
struct audio_dev {
 struct resource res; /* Resource: our registers */
 void __iomem *virtbase; /* Where registers can be accessed in memory */
 audio_samples_t samples;
 audio_ready_t ready;
 int irq_num;
} dev;

/*
 * Read left and right samples in from the device
 * Assumes the device information has been set up
 */
static void read_samples(audio_samples_t *samples)
{
 samples->l = ioread32(L_SAMPLES(dev.virtbase));
 samples->r = ioread32(R_SAMPLES(dev.virtbase));
 ioread32(RESET_IRQ(dev.virtbase));
 dev.samples = *samples;
}
```

```c
/*
 * Handle interrupts raised by our device. Read samples,
 * clear the interrupt, and wake the user level program.
 */
irq_handler_t irq_handler(int irq, void *dev_id, struct pt_regs *reg)
{
 // Read samples from the device
 audio_samples_t samples;
 read_samples(&samples);

 // Wake the user level process
 audio_ready_t ready = { .audio_ready = 1 };
 dev.ready = ready;
 wake_up_interruptible(&wq);

 return IRQ_RETVAL(1);
}


/*
 * Handle ioctl() calls from userspace:
 * Read left and right audio samples from the device.
 * Note extensive error checking of arguments
 */
static long audio_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
 audio_arg_t vla;

 switch (cmd) {
  case AUDIO_READ_SAMPLES:
   // Sleep the process until woken by the interrupt handler, and the data is ready
   wait_event_interruptible_exclusive(wq, dev.ready.audio_ready);

   // The data is now ready, send them to the user space
   vla.samples = dev.samples;
   audio_ready_t ready = { .audio_ready = 0 };
   dev.ready = ready;

   // Copy the data to the user space
   if (copy_to_user((audio_arg_t *) arg, &vla,
     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;
 }

 /* Determine the interrupt number associated with our device */
```

```
 int irq = irq_of_parse_and_map(pdev->dev.of_node, 0);
 dev.irq_num = irq;

 /* Request our interrupt line and register our handler */
 ret = request_irq(irq, (irq_handler_t) irq_handler, 0, "csee4840_audio", NULL);

 if (ret) {
  printk("request_irq err: %d", ret);
  ret = -ENOENT;
  goto out_deregister;
 }

 return 0;

out_release_mem_region:
 release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
 free_irq(dev.irq_num, NULL);
 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));
 free_irq(dev.irq_num, NULL);
 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 = "csee4840,audio-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("Team No Caller ID, Columbia University");
MODULE_DESCRIPTION("Audio driver");
```

software/twilio/place_call.sh

```
#!/bin/bash

# Go to the wavfiles director (a different repo) and upload the new audio file
cd /root/final_proj/csee4840/project/software/wavfiles
git add anonymous_audio.wav; git commit -m "new file"; git push;
cd ../twilio

curl -X POST "https://api.twilio.com/2010-04-01/Accounts/$TWILIO_ACCOUNT_SID/Calls.json" \
--data-urlencode "ApplicationSid=$TWILIO_APPLICATION_SID" \
--data-urlencode "To=+1$TO_PHONE_NUMBER" \
--data-urlencode "From=+1$FROM_PHONE_NUMBER" \
-u $TWILIO_ACCOUNT_SID:$TWILIO_AUTH_TOKEN
```

software/wavfile_construction/Makefile

```
CC  = gcc
CXX = g++

# Compilation options:
# -g for debugging info and -Wall enables all warnings

CFLAGS   = -g -Wall $(INCLUDES)
CXXFLAGS = -g -Wall $(INCLUDES)

# Linking options:
# -g for debugging info

LDFLAGS = -g
```

```
test: test.o make_wav.o

test.o: test.c make_wav.h

make_wav.o: make_wav.c make_wav.h

.PHONY: clean
clean:
  rm -f *.o make_wav test

.PHONY: all
all: clean make_wav.o
```

software/wavfile_construction/make_wav.h

```
/* 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/
 */

#ifndef MAKE_WAV_H
#define MAKE_WAV_H

void write_wav(char * filename, unsigned long num_samples, long unsigned int * data, int s_rate);

#endif
```

software/wavfile_construction/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 = 2;   /* 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);
}
```

software/wavfile_construction/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;
}
```

Matlab Code

Generates 100 24-bit random numbers (50 pairs of *in_left* + *in_right* samples) and simulates the pitch-shifting algorithm on them. Outputs 50 pairs to file *out_data*.

```
% <------------ NUMBER GENERATION ------------>
% Script for 24-bit random number generator
% Generates 100 24-bit random binary numbers
% Writes randomly-generated numbers to file in_data
```

```
in_data = fopen('./in_data_sim', 'w');
for i = 1:1:100
   random= randi([0,1],1, 24);
   print_format = [repmat('%g', 1, numel(random)-1), '%g\n'];
   fprintf(in_data, print_format, random);
end
fclose(in_data);
% <------------ SETUP ------------>
% Circular buffer length
l = 1024;
% Shifting factor
% 0.8 for low pitch
% 1 for normal voice
% 1.65 for high pitch
shift_factor = 0.79;
% Sample number
i = 1;
% ------------- BUFFERS AND POINTERS ------------- %
% Create 2 buffers for left data, and two buffers for right data
% Initialize buffers to zero
l_buf_0 = strings([1, l]);
l_buf_1 = strings([1, l]);
r_buf_0 = strings([1, l]);
r_buf_1 = strings([1, l]);
% Read and write pointers for l_buf_0, l_buf_1, r_buf_0, and r_buf_1
% Initialize w_0 to 1. MATLAB arrays start at 1.
% Initialize w_1 to 180 deg shifted cell.
% l_buf write
l_w_0 = 1;
l_w_1 = floor(l/2);
% l_buf read
l_r_0 = 1;
l_r_1 = 1;
% r_buf write
r_w_0 = 1;
r_w_1 = floor(l/2);
% r_buf read
r_r_0 = 1;
r_r_1 = 1;
% Indexing for read pointers
l_i_0= 1;
l_i_1= 1;
r_i_0= 1;
r_i_1= 1;
% Reads 2 binary numbers from in_data file
% First binary number: in_left
% Second binary number: in_right
% "Streams" them into the pitch_shifter function
in_data = fopen('./in_data_sim', 'r');
```

```
out_data = fopen('./out_data_sim','w');
in_left = fgetl(in_data); % Read the left data
in_right = fgetl(in_data); % Read the right data
while ~isequal(in_left, -1) && ~isequal(in_right, -1)
   % < ------- ALGORITHM --------->
   % Read two samples at a time
   % Add to buffers
   % Read from buffers
   % Take average
   % Output to file
   % LEFT BUFFER
   % Add the sample to l_buf_0 at address l_w_0
   % Advance l_w_0
   l_buf_0(l_w_0) = in_left;
   l_w_0 = mod(i, l) + 1;
   % Add the sample to l_buf_1
   % Advance l_w_1
   l_buf_1(l_w_1) = in_left;
   l_w_1 = mod(i+floor(l/2)-1, l) + 1;
   % Read from l_buf_0 at address l_r_0
   l_i_0= l_i_0 + shift_factor;
   % Read from l_buf_2 at address l_r_1
   l_i_1= l_i_1 + shift_factor;
   % RIGHT BUFFER
   % Add the sample to l_buf_1 at address l_w_0
   % Advance l_w_0
   r_buf_0(r_w_0) = in_right;
   r_w_0 = mod(i, l) + 1;
   % Add the sample to l_buf_1
   % Advance l_w_1
   r_buf_1(r_w_1) = in_right;
   r_w_1 = mod(i + floor(l/2)-1, l) + 1;
   % Adjust left values
   if (l_buf_0(l_r_0)=="")
      l_r_0_out = "00000000000000000000000000";
   else
      l_r_0_out = l_buf_0(l_r_0);
   end
   if (l_buf_1(l_r_1)=="")
      l_r_1_out = "00000000000000000000000000";
   else
      l_r_1_out = l_buf_1(l_r_1);
   end
   % Adjust right values
   if (r_buf_0(r_r_0)=="")
      r_r_0_out = "00000000000000000000000000";
   else
      r_r_0_out = r_buf_0(r_r_0);
   end
```

```
    if (r_buf_1(r_r_1)=="")
        r_r_1_out = "000000000000000000000000";
    else
        r_r_1_out = r_buf_1(r_r_1);
    end
    out_left = add(to_array(l_r_0_out), to_array(l_r_1_out));
    out_left = shift(out_left);
    out_right = add(to_array(r_r_0_out), to_array(r_r_1_out));
    out_right = shift(out_right);
    % Read from l_buf_1 at address r_r_0
    r_i_0= r_i_0 + shift_factor;
    % Read from l_buf_2 at address r_r_1
    r_i_1= r_i_1 + shift_factor;
    % Update read pointers
    l_r_0 = mod(floor(l_i_0), l)+1;
    l_r_1 = mod(floor(l_i_1), l)+1;
    r_r_0 = mod(floor(r_i_0), l)+1;
    r_r_1 = mod(floor(r_i_1), l)+1;
    print_format = [repmat('%g', 1, numel(out_left)-1), '%g\n'];
    fprintf(out_data, print_format, out_left, out_right);
    in_left = fgetl(in_data);
    in_right = fgetl(in_data);
    i=i+1;
end
fclose(in_data);
fclose(out_data);
% Converts binary number to a matlab array
function[in_arr] = to_array(in)
    in_arr = char(num2cell(convertStringsToChars(in)));
    in_arr = reshape(str2num(in_arr),1,[]); %#ok<ST2NM>
end
function[out] = add(in_0, in_1)
    % Function for 24-bit adder
    % Takes 2 24-bit numbers
    % Computes bit-wise addition
    if (isvector(in_0) && length(in_0)==24) && (isvector(in_1) && length(in_1)==24)
        out = zeros(1,24);
        carry_out = 0;

        for i = 24:-1:1
            out(i) = in_0(i) + in_1(i) + carry_out;
            if out(i) == 2
                out(i) = 0;
                carry_out = 1;
            elseif out(i) == 3
                out(i) = 1;
                carry_out = 1;
            else
                carry_out = 0;
```

```
        end
      end
    else
      error("The given inputs are not 24-bit vectors");
    end
end
function[out] = shift(in)
    out = zeros(size(in));
    out(2:24) = in(1:23);
end
```