FPGA-Backed Robotic Transcriber

Team π

Joshua Klein

Nevin Zheng

Sami Shahin

Philip Yuan

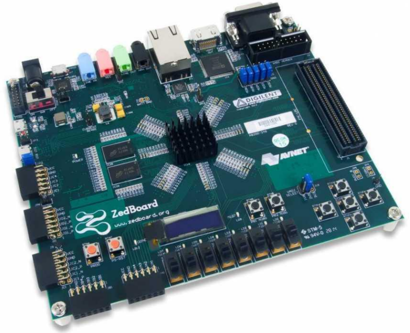
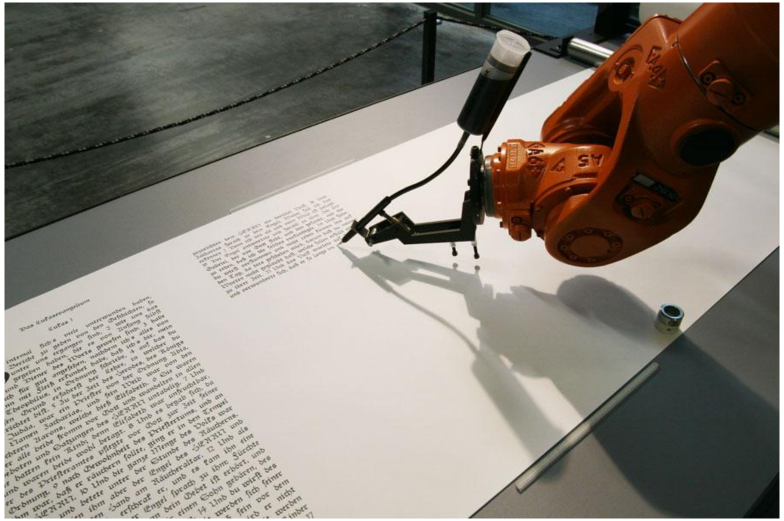


Table of Contents

Overview
Project Motivation

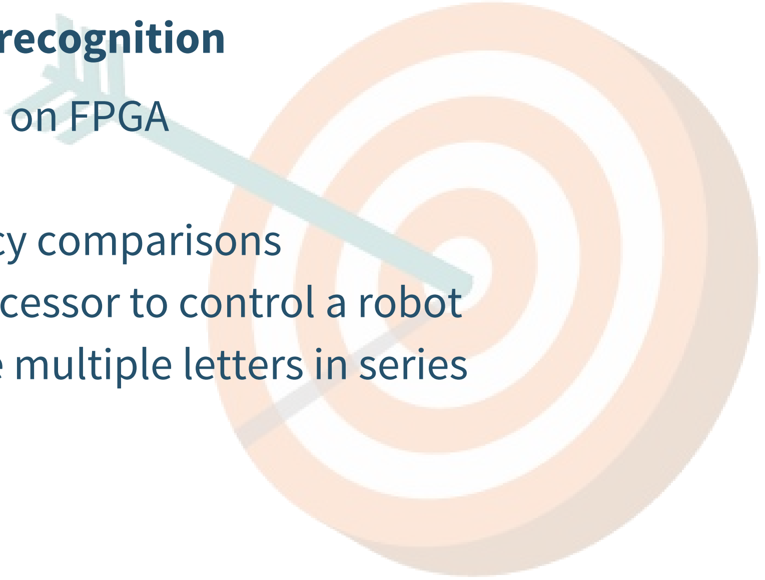
- **We're also learning DSP!:**
  - Combination of concepts
- **Benefits of FPGA**
  - Optimal for specific calculations
- **Popularity:**
  - Speech recognition is trendy
- **Ease of use:**
  - Write simply by speaking
- **Time and energy constraints:**
  - People are busy
- **Artistic presentation:**
  - Improved handwriting



Overview
Project Goals

Goal: Robot transcriber with speech recognition

- Implement audio signal processing on FPGA
  - Getting and storing audio
  - Fourier Transform for frequency comparisons
- Interface the FPGA with a microprocessor to control a robot
  - Be able to have the robot write multiple letters in series



Overview
Spectral Density Comparison

Energy spectral density is a measurement of how a signal's energy is distributed over frequency (rather than time).

1. Compute the DFT of a signal: samples of the Discrete-Time Fourier Transform (DTFT)

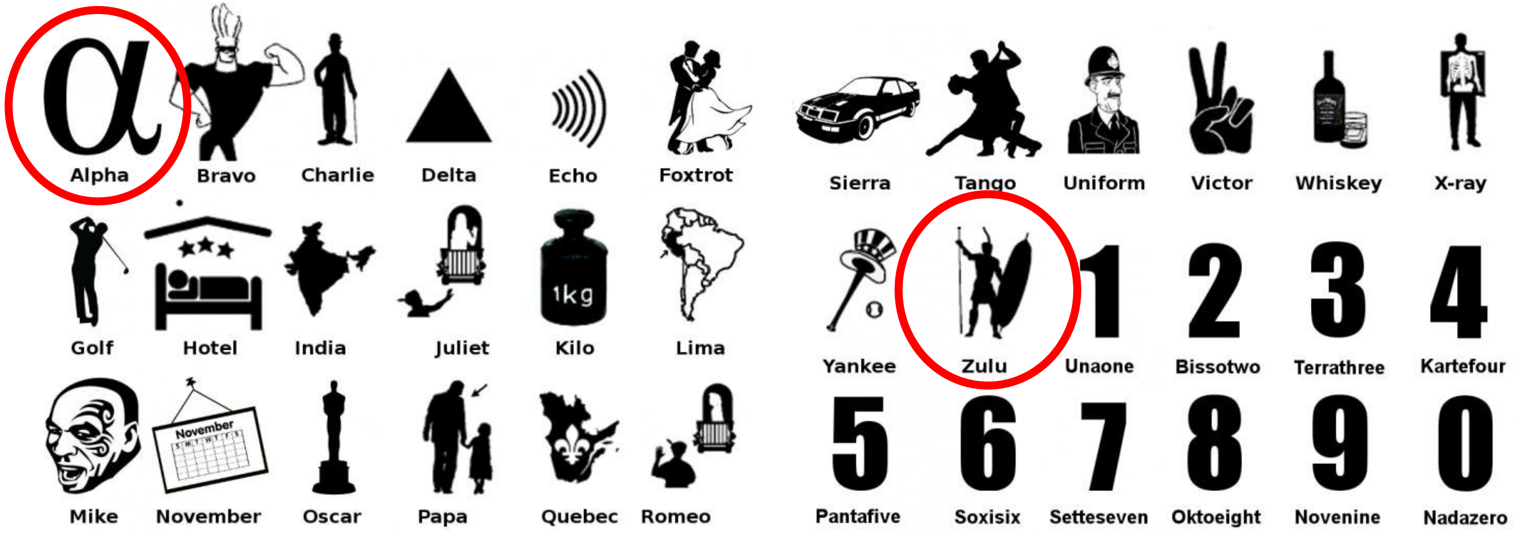
$$X[k]_N = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi k}{N}n} \ ; \ k \in [0, N)$$

2. Compute the energy spectral density for certain frequencies

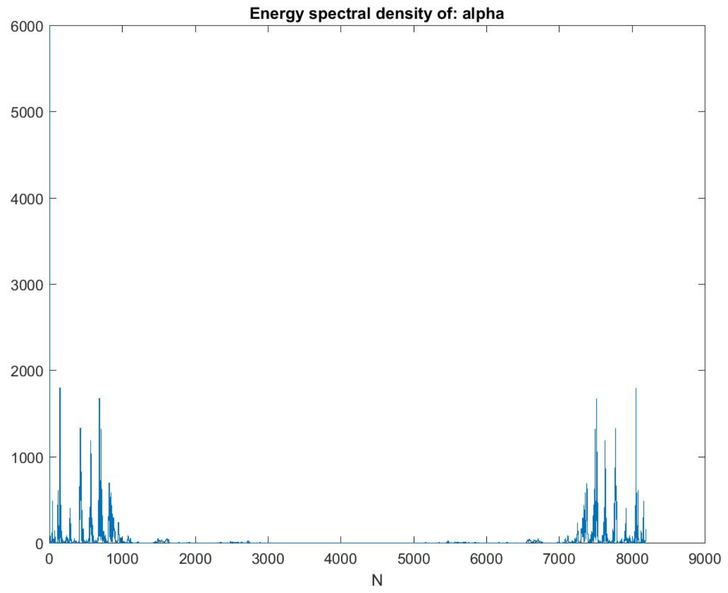
$$\left|X[k]_N\right|^2$$

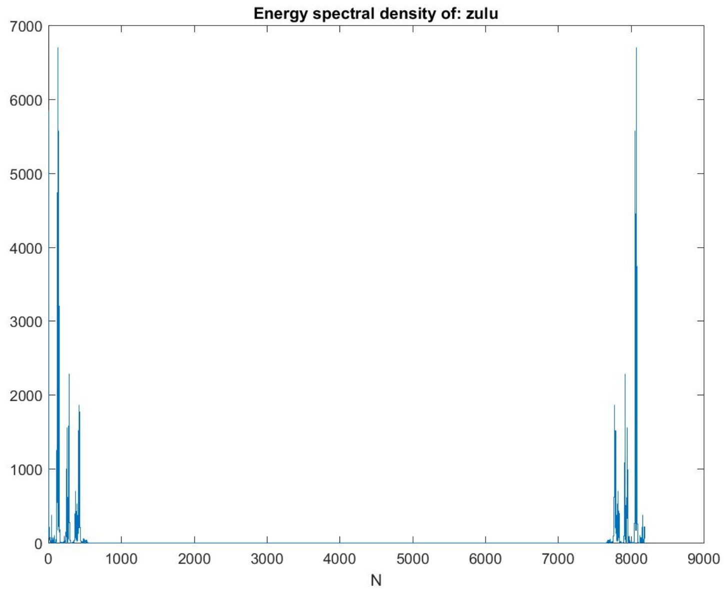
3. Compare with values experimentally computed on MATLAB

Overview
Vocabulary: ICAO/NATO Alphabet



Overview
Vocabulary: ‘Alpha’ and ‘Zulu’





Overview
High Level Description

1. Sample audio with a button press
2. Store in BRAM
3. Calculate the DFT
4. Energy spectral density
5. Comparison of values to guess
6. Communicate with Arduino
7. Write guess with robot arm

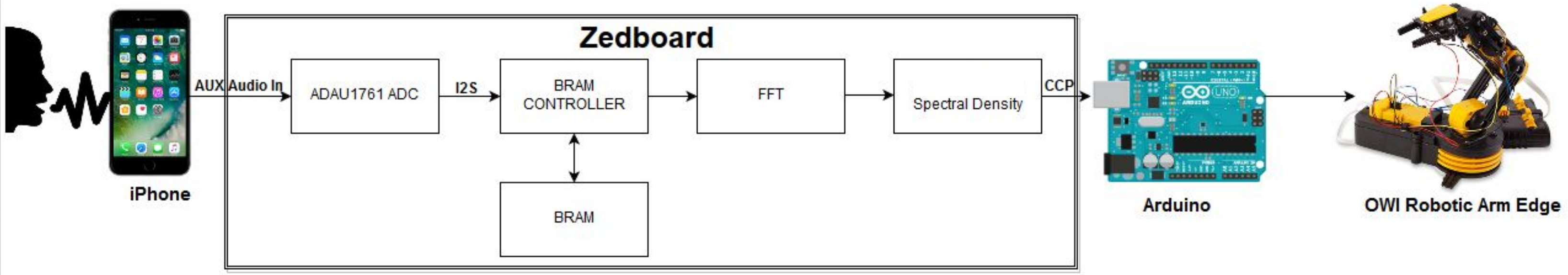
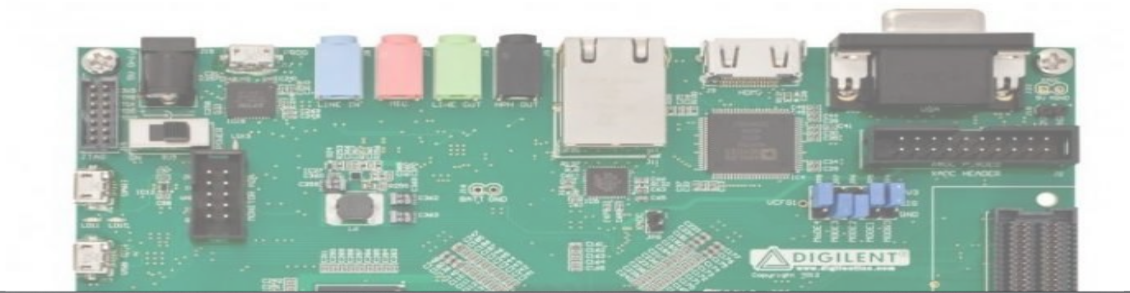


Table of Contents

Design Specification: Meeting the Design Goals
Data Recording And Storage

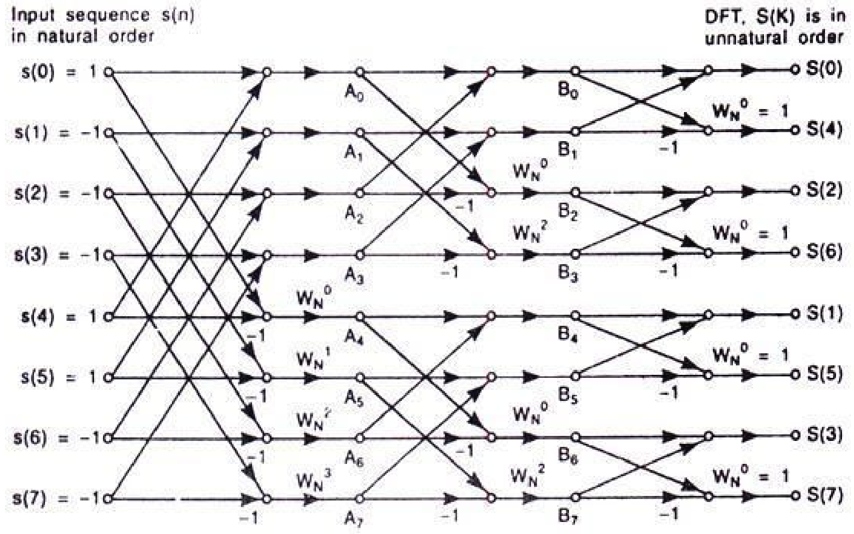
1. **Data Recording:** Sample audio by way of onboard ADAU1761
   a. 24-bit precision
   b. 1.024 seconds at 8 kHz (telephone sample rate) = 8192 samples
      i. ICAO/NATO alphabet!
2. **Data Storage:** Controller
   a. 8192 x 24 BRAM
   b. States to control read/write and burst read/write



Design Specification: Meeting the Design Goals

Frequency Calculations

1. **FFT to calculate DFT:**
   a. N = 8192
   b. Burst Radix-2
   c. Fixed Point
2. **Average Energy Spectral Density:**
   a. Easier calculations with imaginary values
   b. Comparison to values obtained through MATLAB experimentation



Design Specification: Meeting the Design Goals
Robot Arm

1. **CCP:**
   a. 3 wires to interface Arduino and FPGA
   b. No buffer required
2. **Robot Arm:**
   a. 4 12V batteries
   b. 2 movable joints
   c. 4 wires to interface with Arduino Uno: up/down motions of both joints

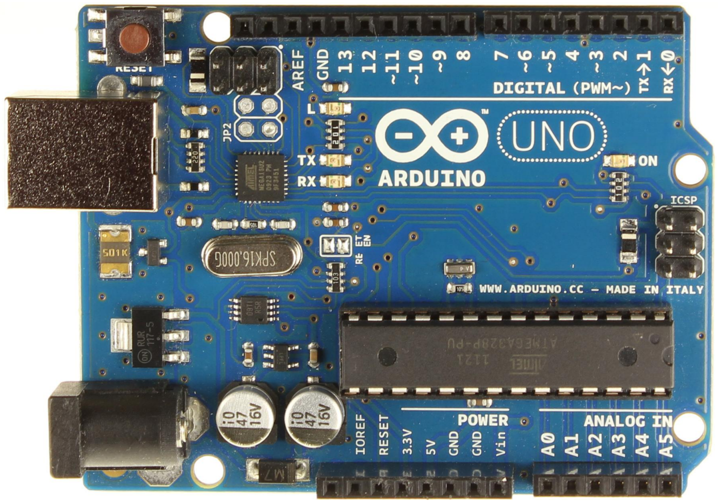


Table of Contents

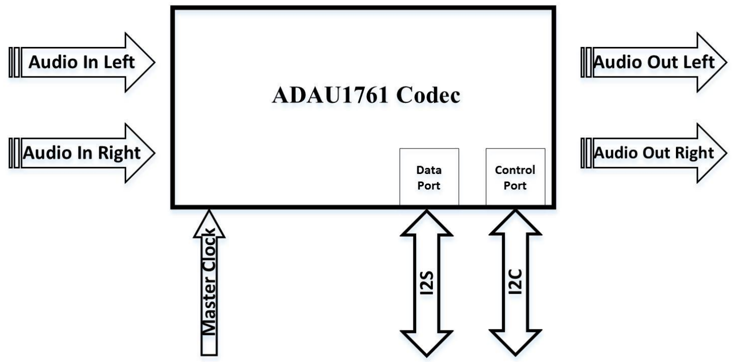
Detailed Functionality
ADAU1761

48kHz sampling x 1024 = 49.152 MHz clock

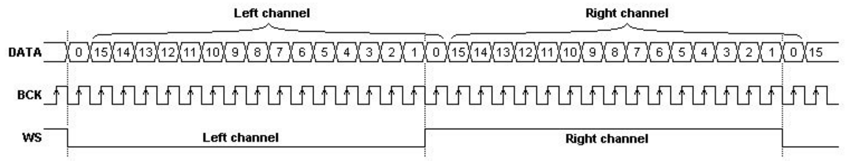
Serial Bus Interfaces

- **I2C:**
  - SDA, SCL
  - Sampling rates, bit width
- **I2S:**
  - Data line: SD
  - Clock lines: SCK, WS

Thanks to Mike Field at Hamsterworks



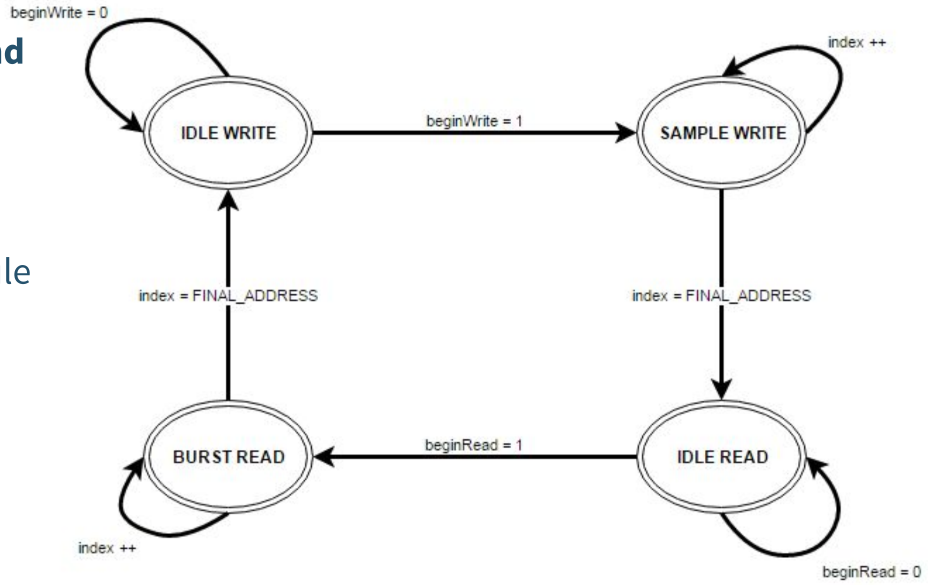
TDA1543 D/A serial bus format (I2S)



Detailed Functionality
BRAM Controller

Interface between BRAM controller and audio stream

- Press button to sample audio
- Reads sample data into FFT module



Detailed Functionality
Xilinx LogiCORE IP: FFT 7.1

“The **Xilinx LogiCORE™ IP Fast Fourier Transform** (FFT) implements the Cooley-Tukey FFT algorithm, a computationally efficient method for calculating the Discrete Fourier Transform (DFT)”

- 8192-point FFT implementation
- Radix-2 Fixed-point
- Treated as black box

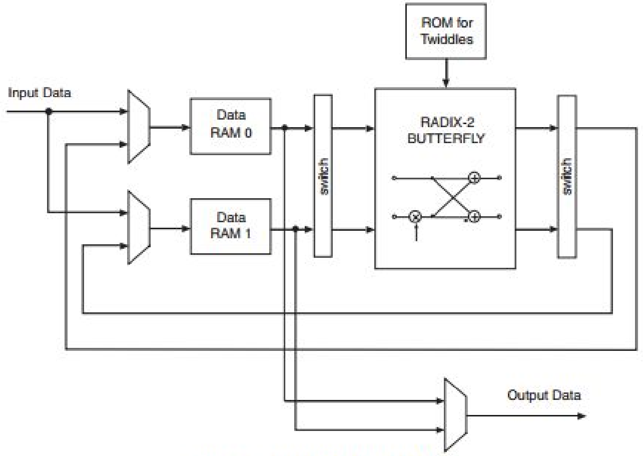


Figure 5: **Radix-2, Burst I/O**

Detailed Functionality
Xilinx LogiCORE IP: FFT 7.1

- **Real and imaginary inputs and outputs**
  - XN_RE, XN_IM, XK_RE, XK_IM
- **Input and output indice**
  - XN_INDEX, XK_INDEX
- **Ready, Busy, Early Done, Done signals**
- **Scaling options**
  - Automatic vs. input array

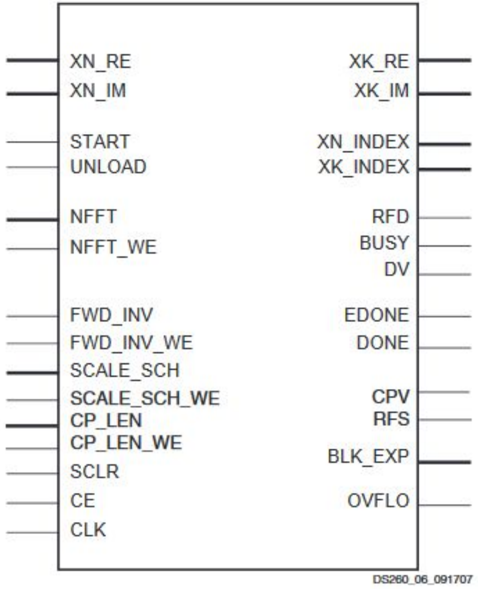
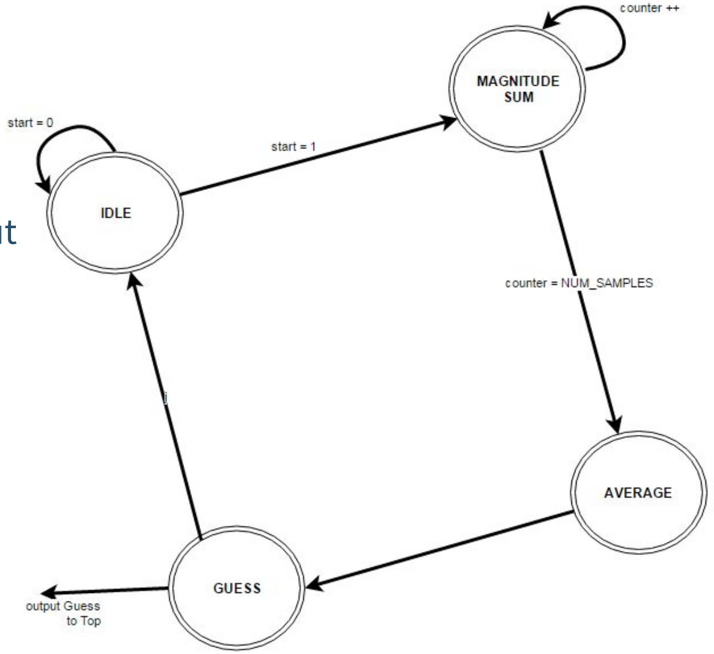


Figure 7: **Core Schematic Symbol (Single Channel)**

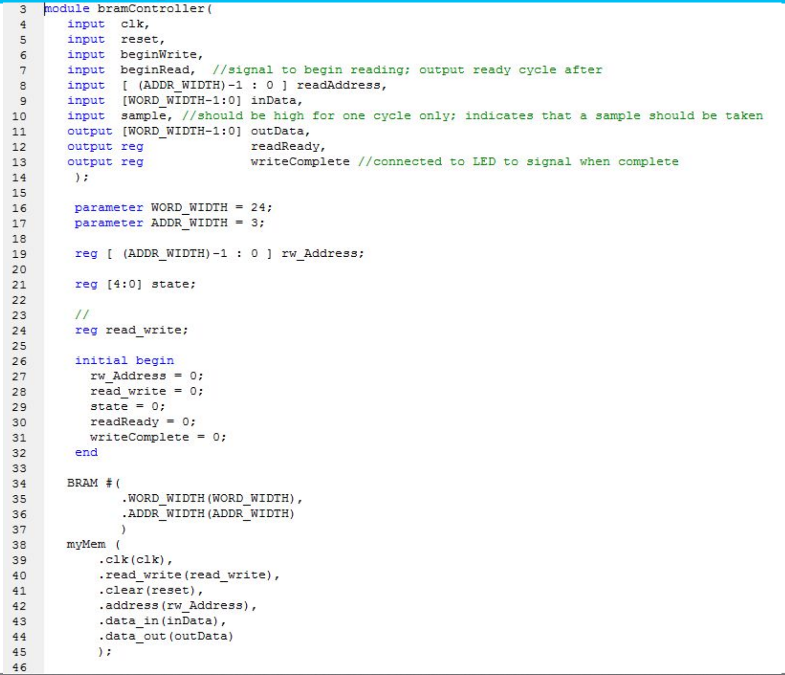
Detailed Functionality
Average Energy Spectral Density

- **Idle**
    - Waits for start signal
- **Magnitude Sum**
    - Sums squared magnitude of FFT output
- **Average**
    - Take average of FFT
- **Compare**
    - Compare with threshold

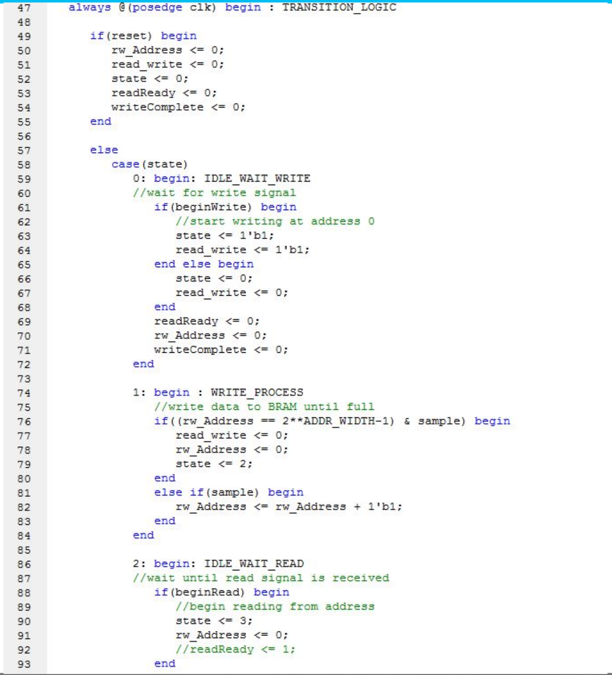


Detailed Functionality

In-Depth: BRAM Controller + Interactions



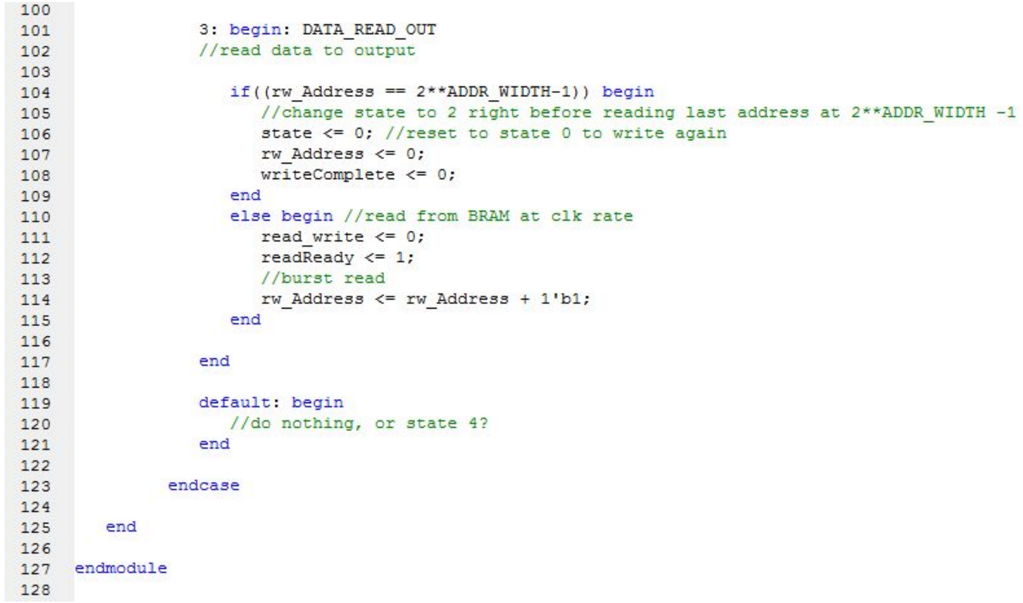
```verilog
module bramController(
   input  clk,
   input  reset,
   input  beginWrite,
   input  beginRead,  //signal to begin reading; output ready cycle after
   input  [ (ADDR_WIDTH)-1 : 0 ] readAddress,
   input  [WORD_WIDTH-1:0] inData,
   input  sample, //should be high for one cycle only; indicates that a sample should be taken
   output [WORD_WIDTH-1:0] outData,
   output reg              readReady,
   output reg              writeComplete //connected to LED to signal when complete
    );

    parameter WORD_WIDTH = 24;
    parameter ADDR_WIDTH = 3;

    reg [ (ADDR_WIDTH)-1 : 0 ] rw_Address;

    reg [4:0] state;

    //
    reg read_write;

    initial begin
      rw_Address = 0;
      read_write = 0;
      state = 0;
      readReady = 0;
      writeComplete = 0;
    end

   BRAM #(
          .WORD_WIDTH(WORD_WIDTH),
          .ADDR_WIDTH(ADDR_WIDTH)
          )
   myMem  (
        .clk(clk),
        .read_write(read_write),
        .clear(reset),
        .address(rw_Address),
        .data_in(inData),
        .data_out(outData)
        );

```



```verilog
   always @(posedge clk) begin : TRANSITION_LOGIC

      if(reset) begin
         rw_Address <= 0;
         read_write <= 0;
         state <= 0;
         readReady <= 0;
         writeComplete <= 0;
      end

      else
         case(state)
            0: begin: IDLE_WAIT_WRITE
            //wait for write signal
               if(beginWrite) begin
                  //start writing at address 0
                  state <= 1'b1;
                  read_write <= 1'b1;
               end else begin
                  state <= 0;
                  read_write <= 0;
               end
               readReady <= 0;
               rw_Address <= 0;
               writeComplete <= 0;
            end

            1: begin : WRITE_PROCESS
               //write data to BRAM until full
               if((rw_Address == 2**ADDR_WIDTH-1) & sample) begin
                  read_write <= 0;
                  rw_Address <= 0;
                  state <= 2;
               end
               else if(sample) begin
                  rw_Address <= rw_Address + 1'b1;
               end
            end

            2: begin: IDLE_WAIT_READ
            //wait until read signal is received
               if(beginRead) begin
                  //begin reading from address
                  state <= 3;
                  rw_Address <= 0;
                  //readReady <= 1;
               end
```

Detailed Functionality

In-Depth: BRAM Controller + Interactions



```

            3: begin: DATA_READ_OUT
            //read data to output

               if((rw_Address == 2**ADDR_WIDTH-1)) begin
                  //change state to 2 right before reading last address at 2**ADDR_WIDTH -1
                  state <= 0; //reset to state 0 to write again
                  rw_Address <= 0;
                  writeComplete <= 0;
               end
               else begin //read from BRAM at clk rate
                  read_write <= 0;
                  readReady <= 1;
                  //burst read
                  rw_Address <= rw_Address + 1'b1;
               end

            end

            default: begin
               //do nothing, or state 4?
            end

         endcase

   end

endmodule

```

Detailed Functionality
Arduino and Robot Arm

- **Robot Arm**
  - Utilizes 2 working joints
  - Each joint has a basic up/down motion
  - Speed of joint motion dependent upon current running through motor
- **Arduino**
  - Controls COM ports for the robot arm using a relay shield
  - 4 relays are used, controlling up or down motion for each motor
  - Dots and dashes and programmed as simple arm motions
    - One joint writes, the other joint determines where to write
  - A reset protocol (carriage return/moving the arm left) is called whenever the robotic arm is approaching the boundary of where it can move
  - The Arduino talks to the FPGA through a custom communication protocol, or CCP

Detailed Functionality
CCP

Arduino:

```
If (P10 == READY && P0 && !P1):
        Write “a”; P10 = LOW;
If (P10 == READY && P1 && !P0):
        Write “b”; P10 = LOW;
Else: Still waiting...
```

FPGA:

```
If (X3 == READY && Buffer has letter):
        Push letter from buffer to X1 (a)
        or X2 (b); Pop front from buffer;
```

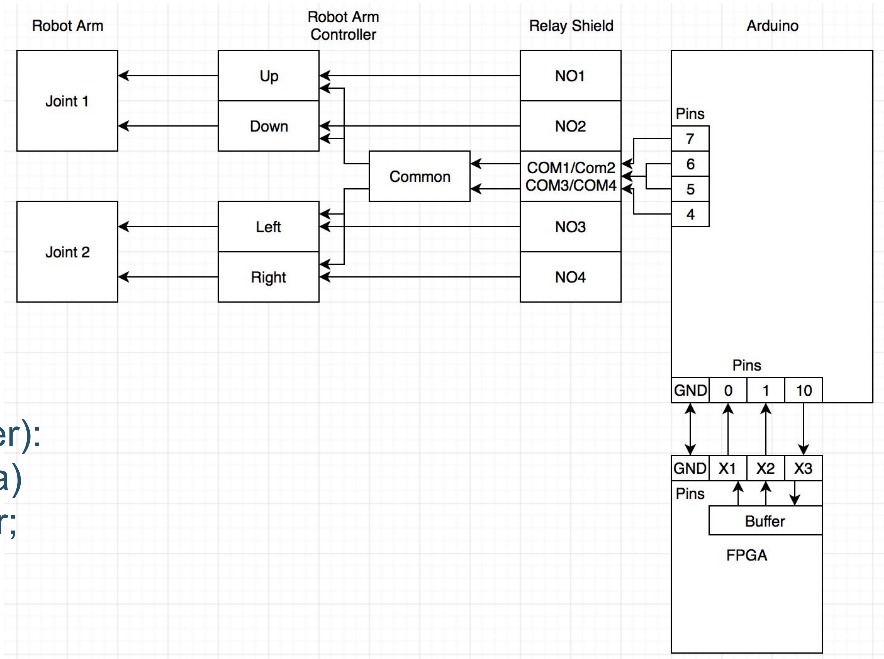
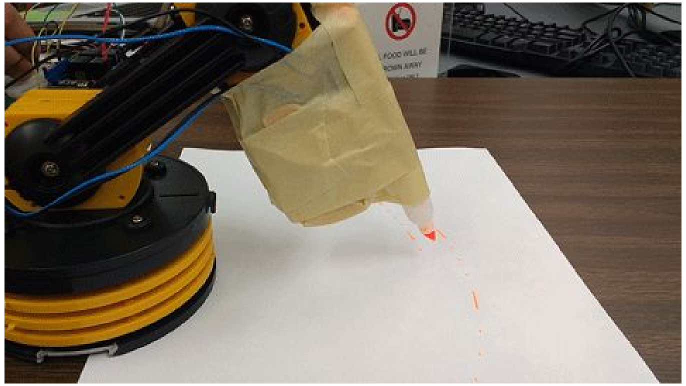


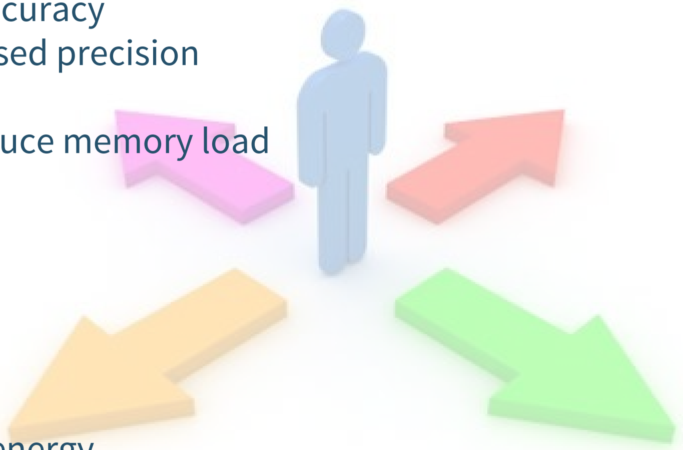
Table of Contents

Analysis
Results... So Far

- **Not completely working, but optimistic**
- **Audio processing**
  - Individual components work through implementation and/or simulation
  - Timing problems need to be resolved
- **Robot arm**
  - Working! Integration mostly works



Analysis
Design Tradeoffs



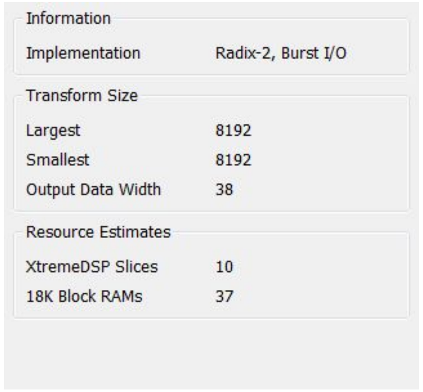
- **Sampling Rate + Bit Depth:**
  - 8 kHz - reduce memory load with decent accuracy
  - 24 bits - increased memory load but increased precision
- **1.024 s Recording Time:**
  - Long enough for word, short enough to reduce memory load
  - FFT vs. DFT LogiCORE IP - N = 2^n samples
- **Price vs. Precision:**
  - Storing audio on iPhone vs. microphone
  - Robot arm vs. 2-D plotter
- **SPI vs. CCP**
  - SPI: more robust
  - CCP: more lightweight, less hardware and energy

Analysis
Design Resource Summary

Our main priority was with implementation.

However, there were resource tradeoffs:

- Tradeoffs between FFT types
- Options for resource and performance



Information	
Implementation	Radix-2, Burst I/O
Transform Size	
Largest	8192
Smallest	8192
Output Data Width	38
Resource Estimates	
XtremeDSP Slices	10
18K Block RAMs	37

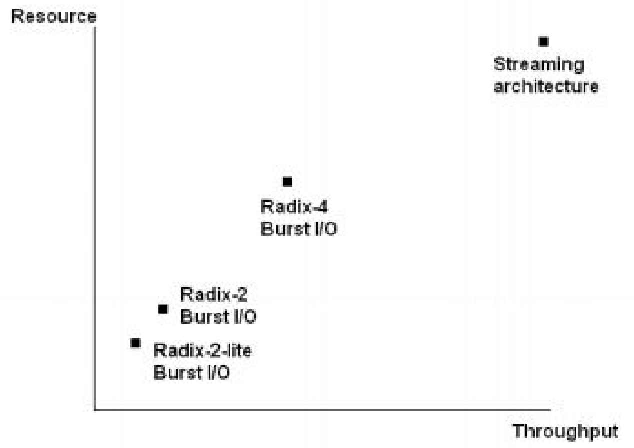
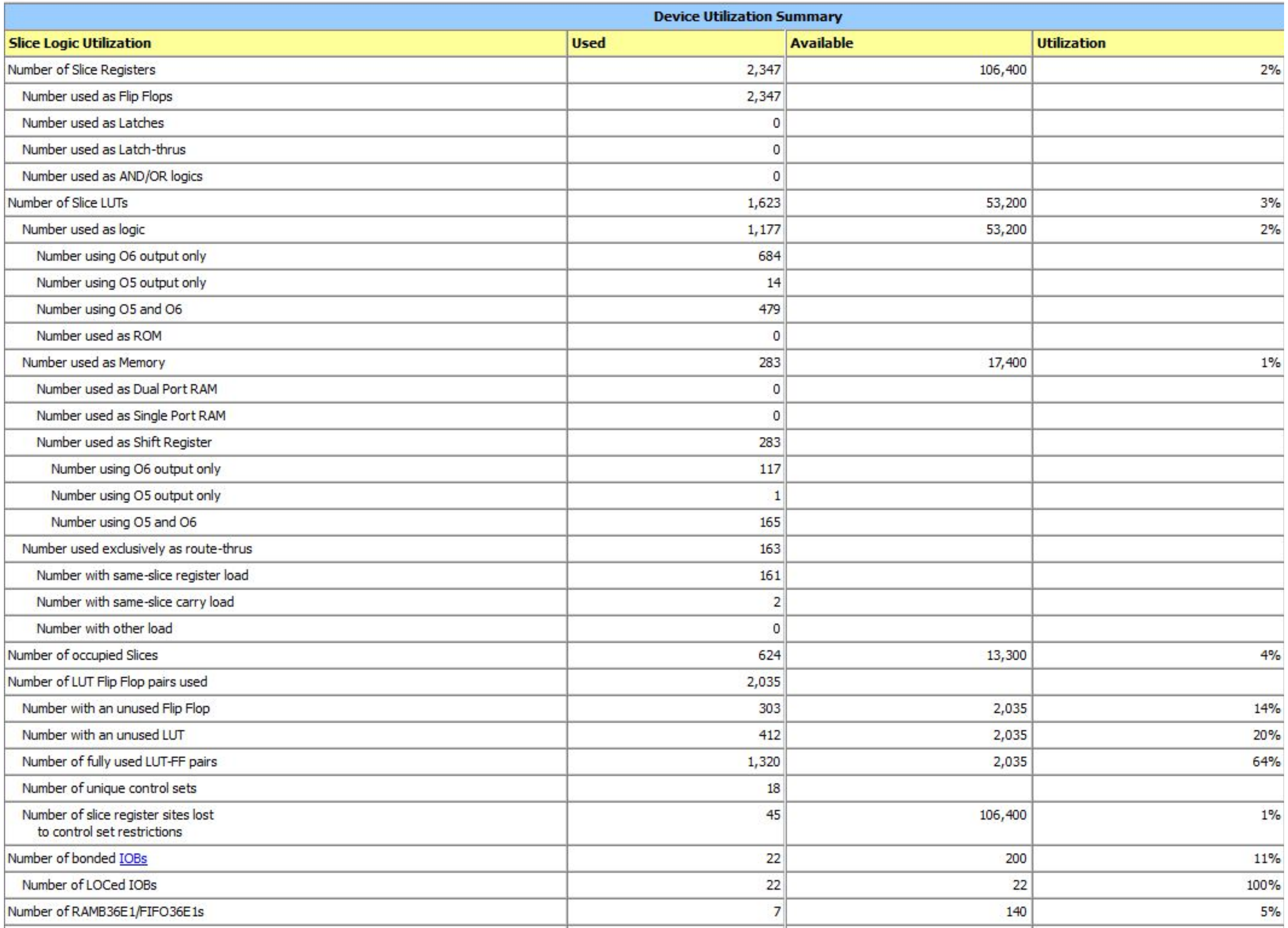


Figure 2: **Resource versus Throughput for Architecture Options**



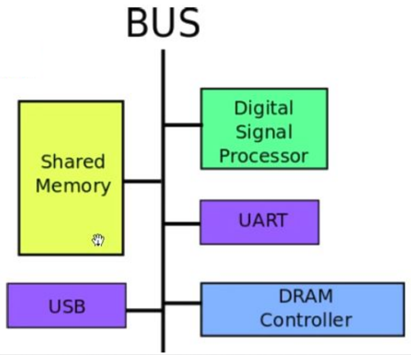
Device Utilization Summary

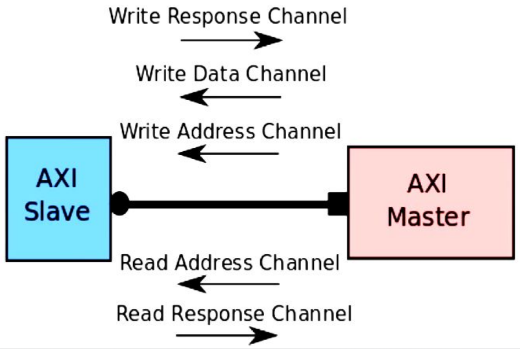
Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	2,347	106,400	2%
Number used as Flip Flops	2,347		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	1,623	53,200	3%
Number used as logic	1,177	53,200	2%
Number using O6 output only	684		
Number using O5 output only	14		
Number using O5 and O6	479		
Number used as ROM	0		
Number used as Memory	283	17,400	1%
Number used as Dual Port RAM	0		
Number used as Single Port RAM	0		
Number used as Shift Register	283		
Number using O6 output only	117		
Number using O5 output only	1		
Number using O5 and O6	165		
Number used exclusively as route-thrus	163		
Number with same-slice register load	161		
Number with same-slice carry load	2		
Number with other load	0		
Number of occupied Slices	624	13,300	4%
Number of LUT Flip Flop pairs used	2,035		
Number with an unused Flip Flop	303	2,035	14%
Number with an unused LUT	412	2,035	20%
Number of fully used LUT-FF pairs	1,320	2,035	64%
Number of unique control sets	18		
Number of slice register sites lost to control set restrictions	45	106,400	1%
Number of bonded IOBs	22	200	11%
Number of LOCed IOBs	22	22	100%
Number of RAMB36E1/FIFO36E1s	7	140	5%

Analysis
Speech Recognition: Failures

Advanced eXtensible Interface (AXI) is a communication standard for connecting various Intellectual Property (IP) cores. A System on Chip (SoC) bus developed by ARM Holdings.

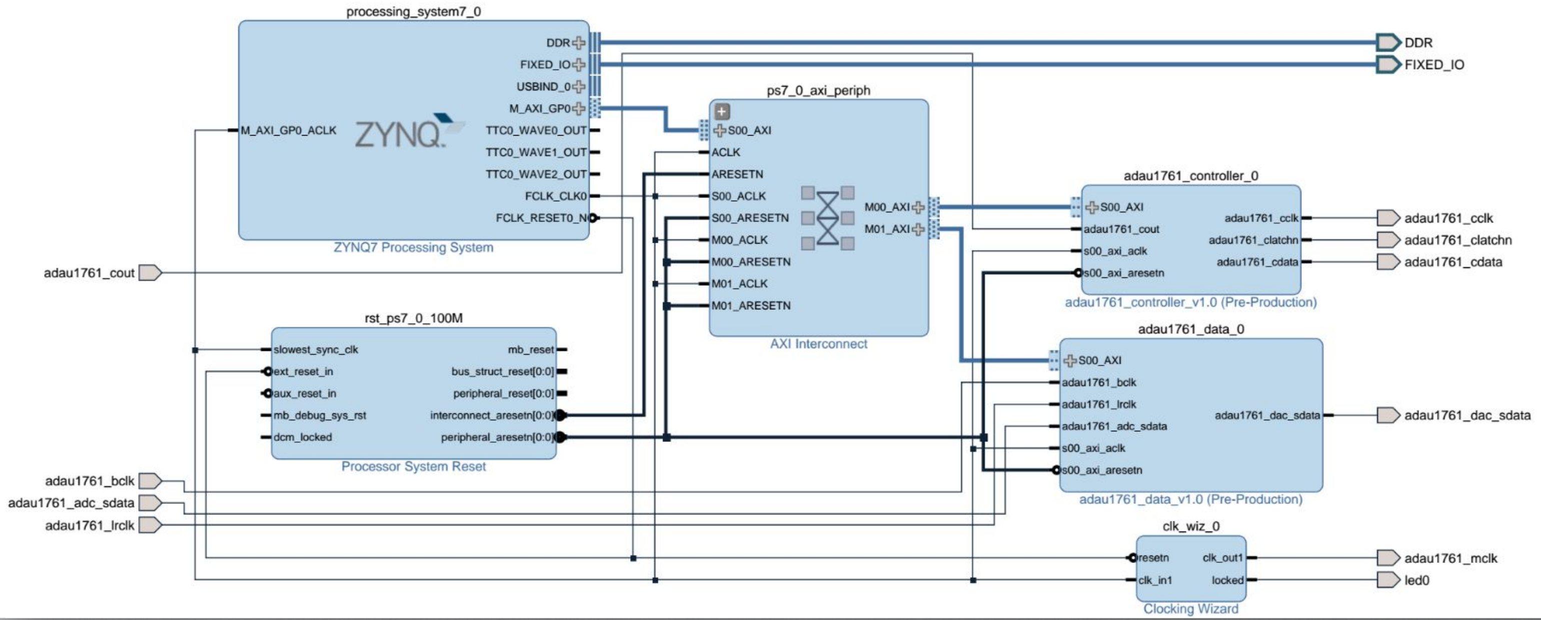
- Standardized signaling rules for communicating
  - Interconnect for multiple slaves and masters
- Initial Vivado project with software ultimately scrapped: had working playback!





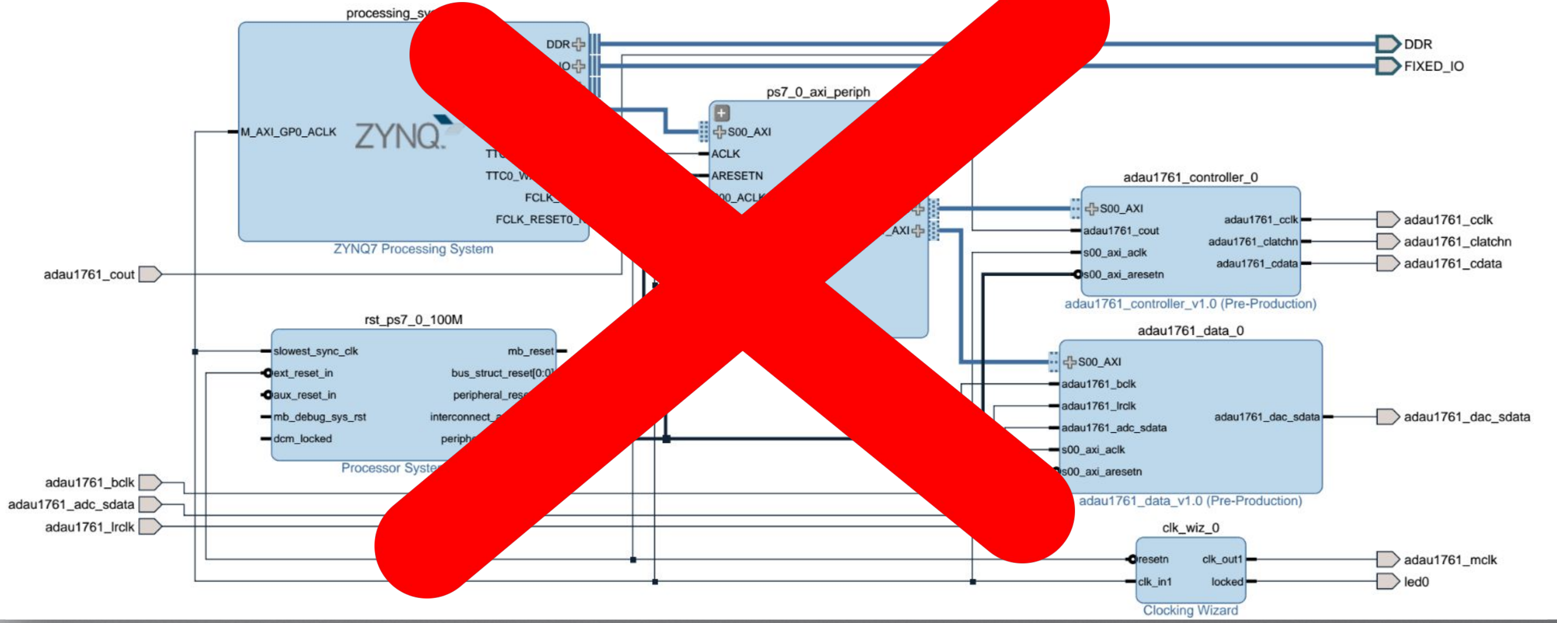
Analysis
Speech Recognition: Failures

Vivado Block Designer: AXI-based design planned for MicroBlaze soft microprocessor



Analysis
Speech Recognition: Failures

Vivado Block Designer: AXI-based design planned for MicroBlaze soft micropro cessor

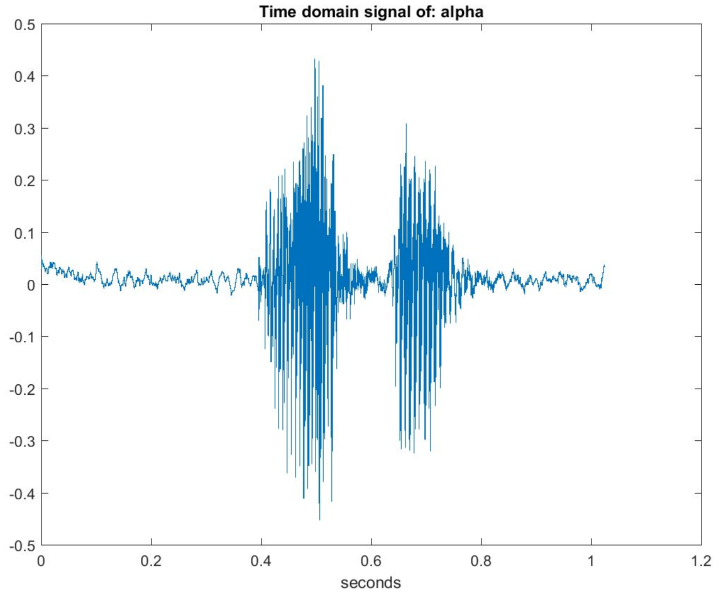


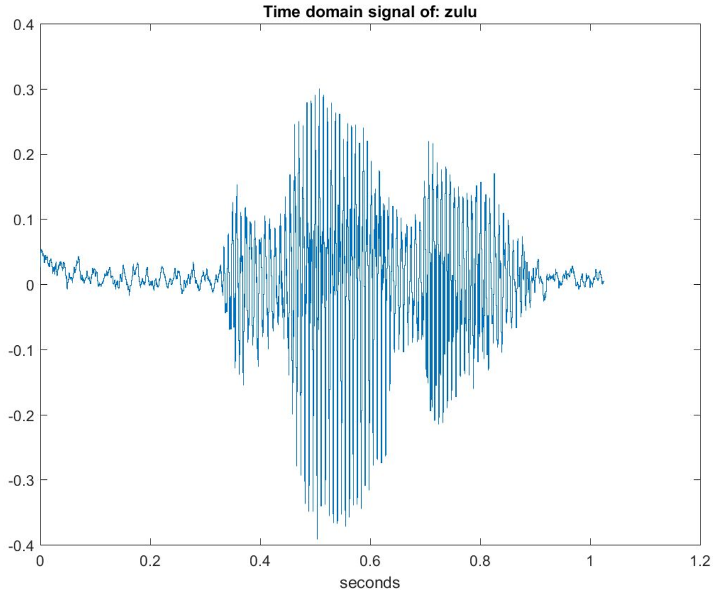
Analysis
Speech Recognition: Successes

- **The algorithm is decent… but nowhere near optimal**
  - Heavily speaker and word dependent
  - Obvious flaws
  - A more ‘established’ (but more complicated) algorithm would be better
    - Mel Cepstrum Coefficients
- **Limitations from hardware-only computations**

Analysis

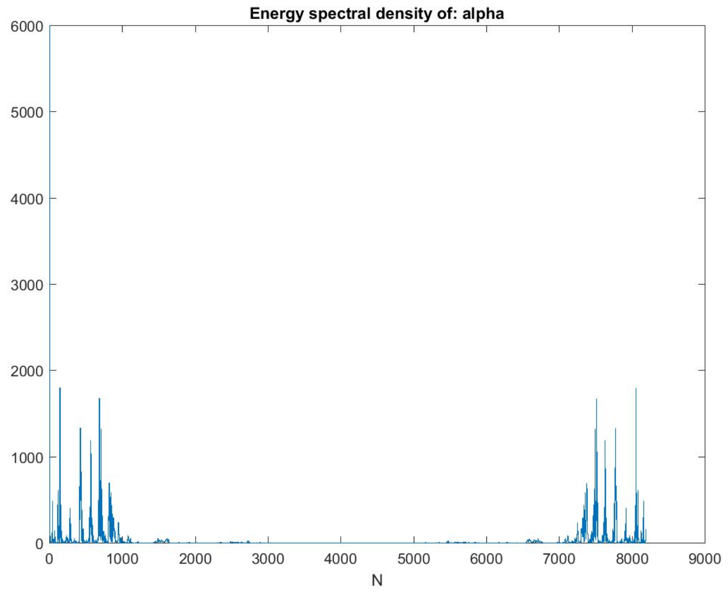
Speech Recognition: Successes

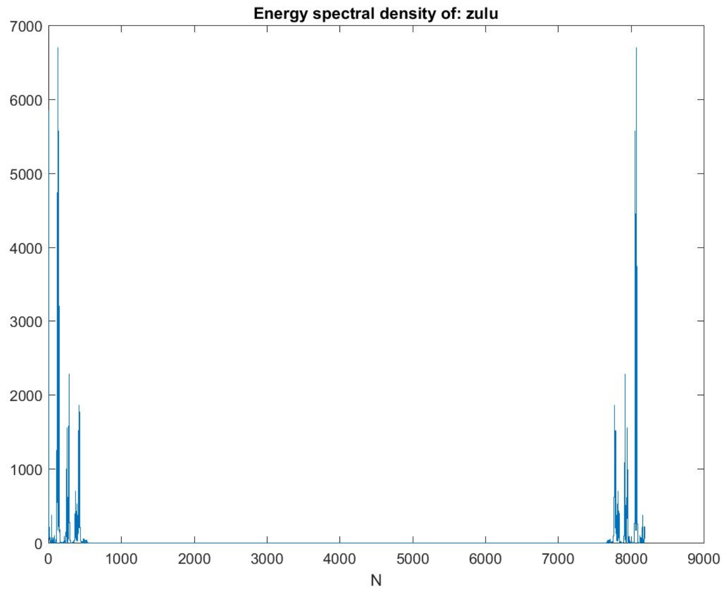




Analysis

Speech Recognition: Successes





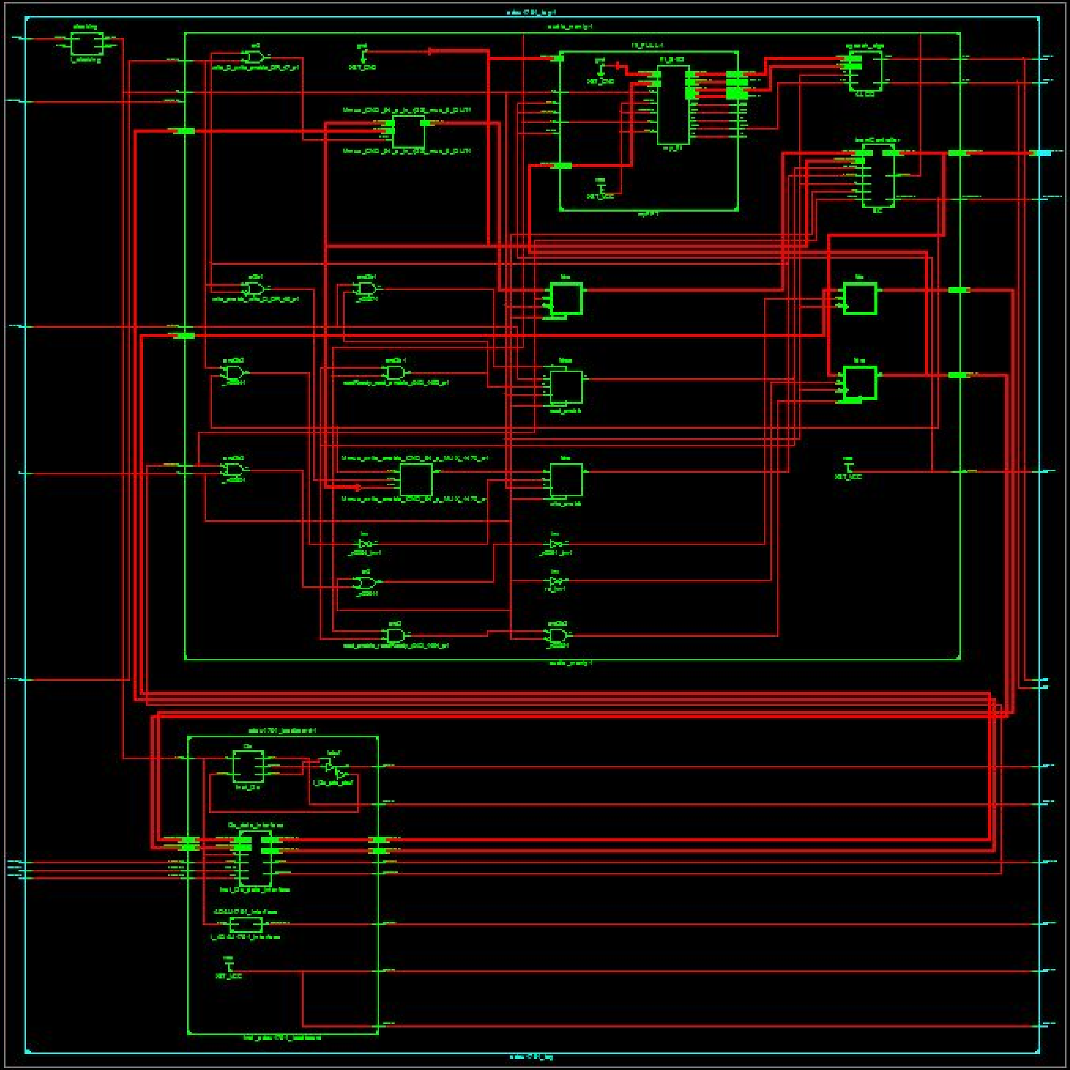
Analysis
Robotic Arm: Failures And Successes

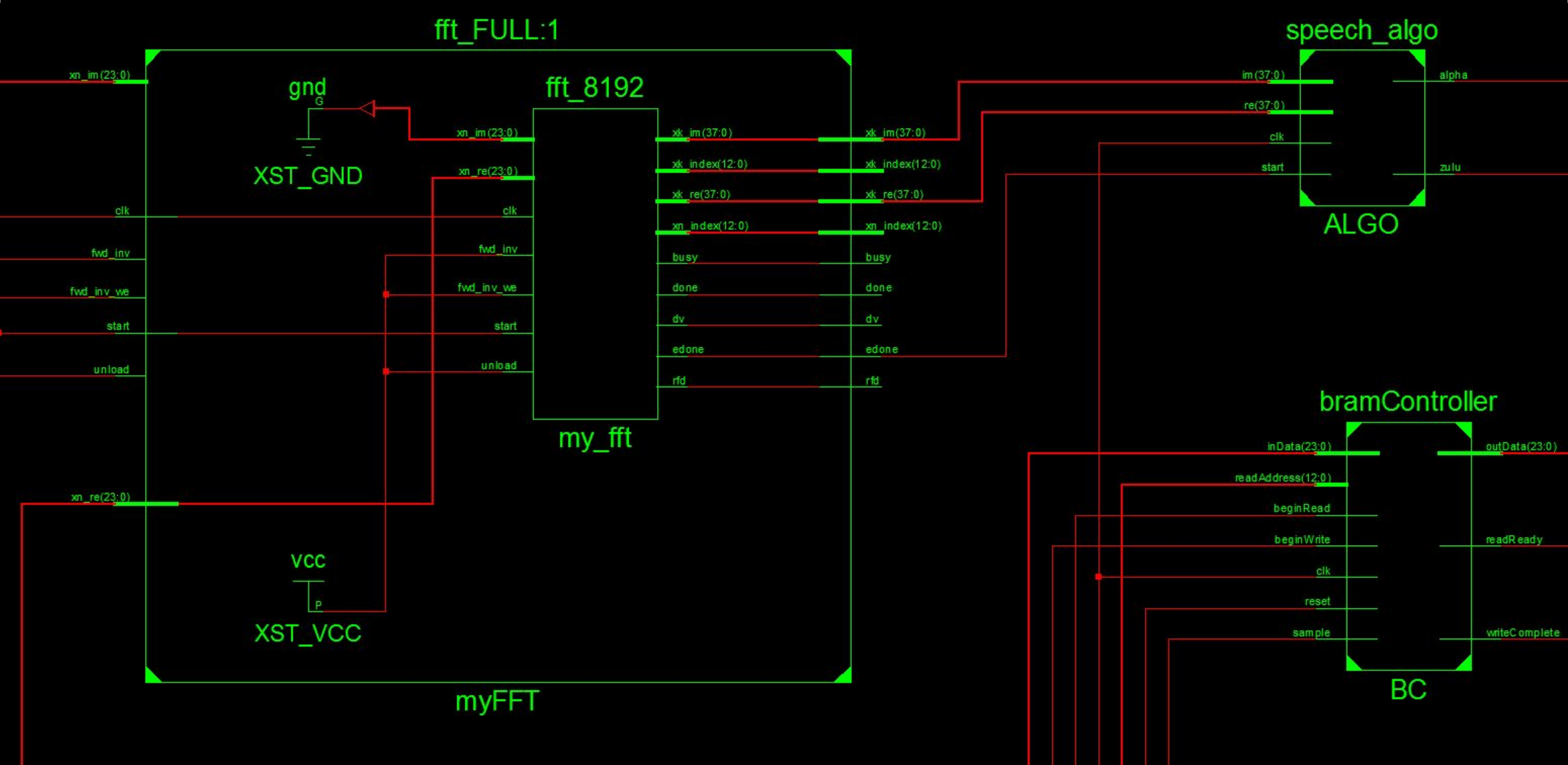
Failures

- Lack of stepper motors and encoders: system is only semi-autonomous
- SPI would be more robust than CCP
- Robot has limited movement
  - Relay shield with 4 relay modules means only 2 joints available

Successes

- Writes characters in morse code
- Easily controlled by the arduino
- Arduino microcontroller can interact with FPGA through CCP
- Robotic arm has a carriage return!




fft_FULL:1
xn_im(23:0)
gnd
G
XST_GND
fft_8192
xn_im(23:0)
xn_re(23:0)
clk
fwd_inv
fwd_inv_we
start
unload
xk_im(37:0)
xk_index(12:0)
xk_re(37:0)
xn_index(12:0)
busy
done
dv
edone
rfd
my_fft
clk
fwd_inv
fwd_inv_we
start
unload
xn_re(23:0)
vcc
P
XST_VCC
myFFT
xk_im(37:0)
xk_index(12:0)
xk_re(37:0)
xn_index(12:0)
busy
done
dv
edone
rfd
speech_algo
im(37:0)
re(37:0)
clk
start
alpha
zulu
ALGO
bramController
inData(23:0)
readAddress(12:0)
beginRead
beginWrite
clk
reset
sample
outData(23:0)
readReady
writeComplete
BC

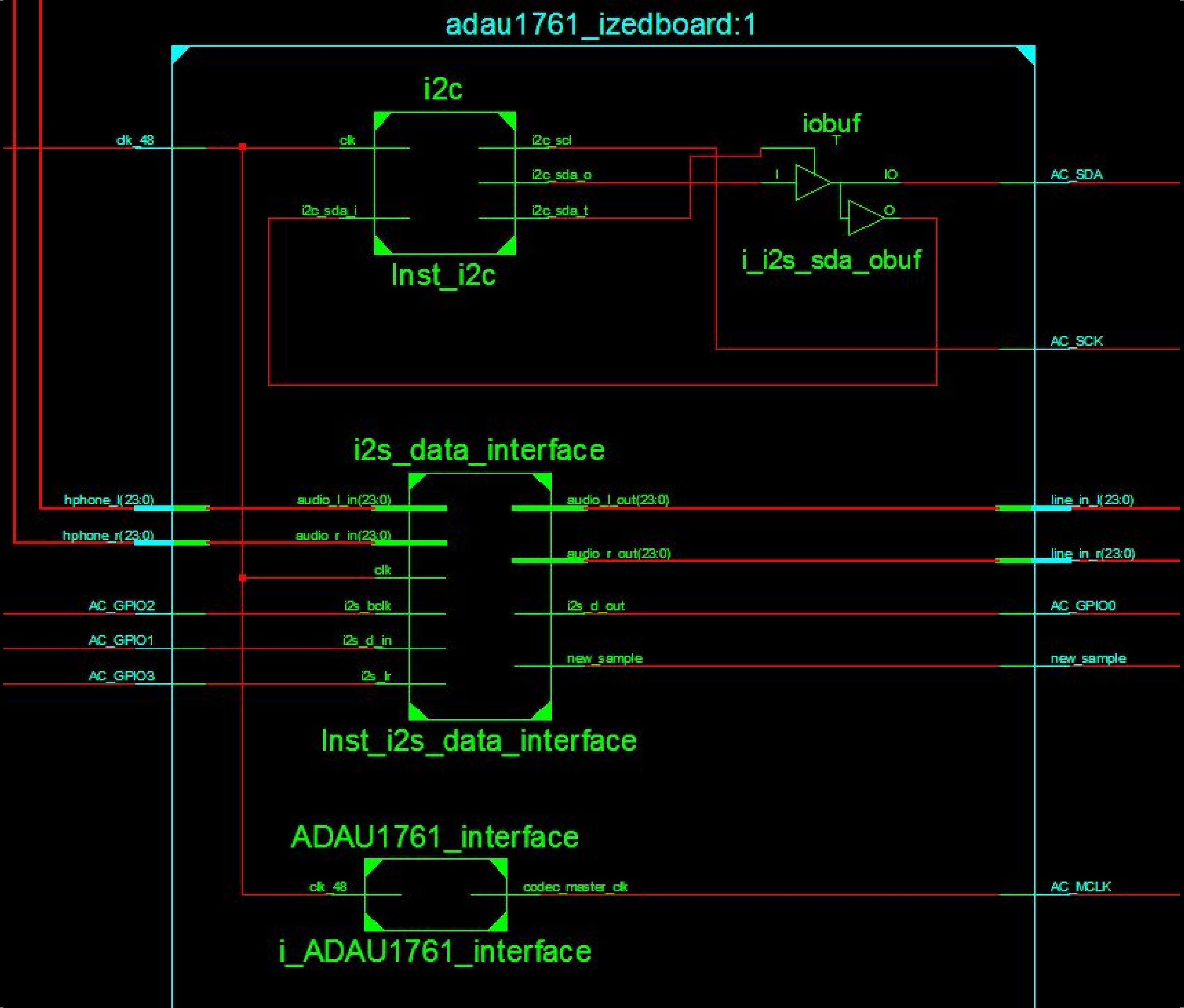

adau1761_izedboard:1
i2c
clk_48
clk
i2c_scl
i2c_sda_o
i2c_sda_i
i2c_sda_t
Inst_i2c
iobuf
T
I
IO
O
i_i2s_sda_obuf
AC_SDA
AC_SCK
i2s_data_interface
hphone_l(23:0)
audio_l_in(23:0)
audio_l_out(23:0)
line_in_l(23:0)
hphone_r(23:0)
audio_r_in(23:0)
audio_r_out(23:0)
line_in_r(23:0)
clk
AC_GPIO2
i2s_bclk
i2s_d_out
AC_GPIO0
AC_GPIO1
i2s_d_in
new_sample
new_sample
AC_GPIO3
i2s_lr
Inst_i2s_data_interface
ADAU1761_interface
clk_48
codec_master_clk
AC_MCLK
i_ADAU1761_interface

Table of Contents

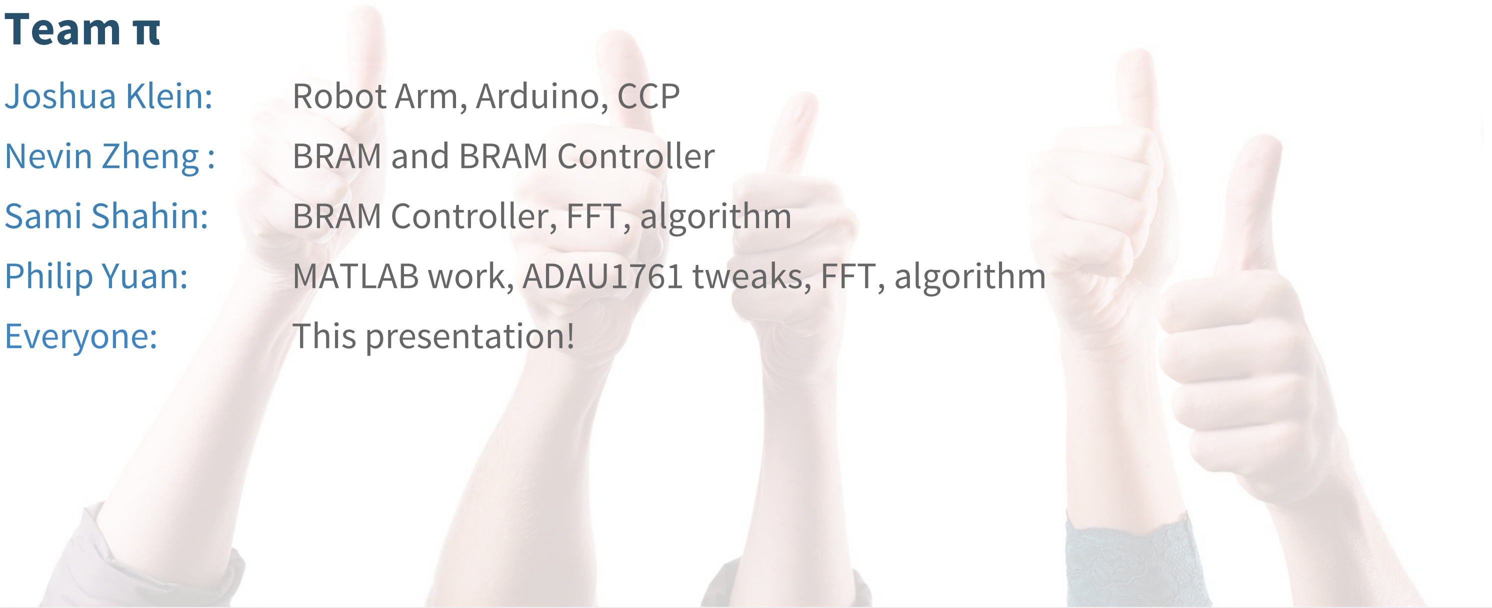
Conclusion
What We Learned

- **Specifics of DSP implemented on FPGA:**
  - Putting theory to practice
- **Verilog, VHDL, … and above:**
  - Limitations and benefits of FPGA’s
  - AXI-based block diagram, benefits of HLS
- **Communicating with other devices**

Conclusion
Future Work

- **Improve algorithm complexity**
  - Expanded system vocabulary
  - Mel Cepstrum
  - Hidden Markov Models
  - Speech coprocessor for mass processing
- **Vivado block design tools**
- **Robotic Arm**
  - Improved control

Conclusion
Attribution of Work



Team π

Joshua Klein:	Robot Arm, Arduino, CCP
Nevin Zheng :	BRAM and BRAM Controller
Sami Shahin:	BRAM Controller, FFT, algorithm
Philip Yuan:	MATLAB work, ADAU1761 tweaks, FFT, algorithm
Everyone:	This presentation!

Thank you.
Questions?