Landscape Generator Based on FFT 3D-Spectral Visualization

- CSEE 4840 Group FFT

Yuxiao Qu (yq2381), Ning Xia (nx2173),

Yucong Li (yl5363), Yimin Yang (yy3352)

CONTENTS

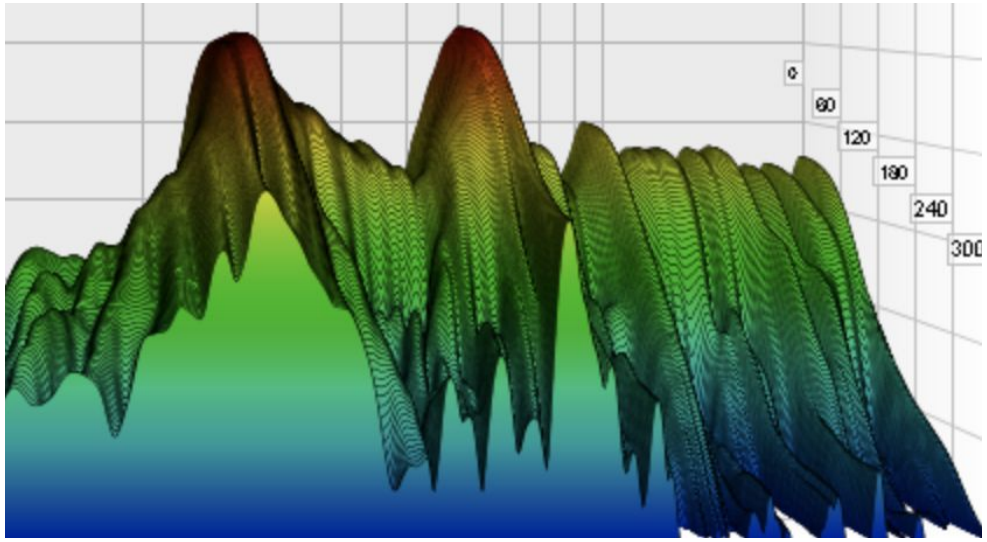
Overview

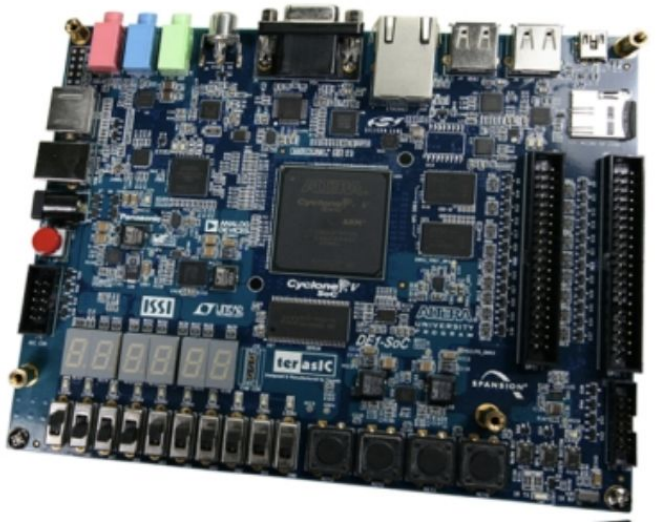
- Overview
- Diagram

Overview

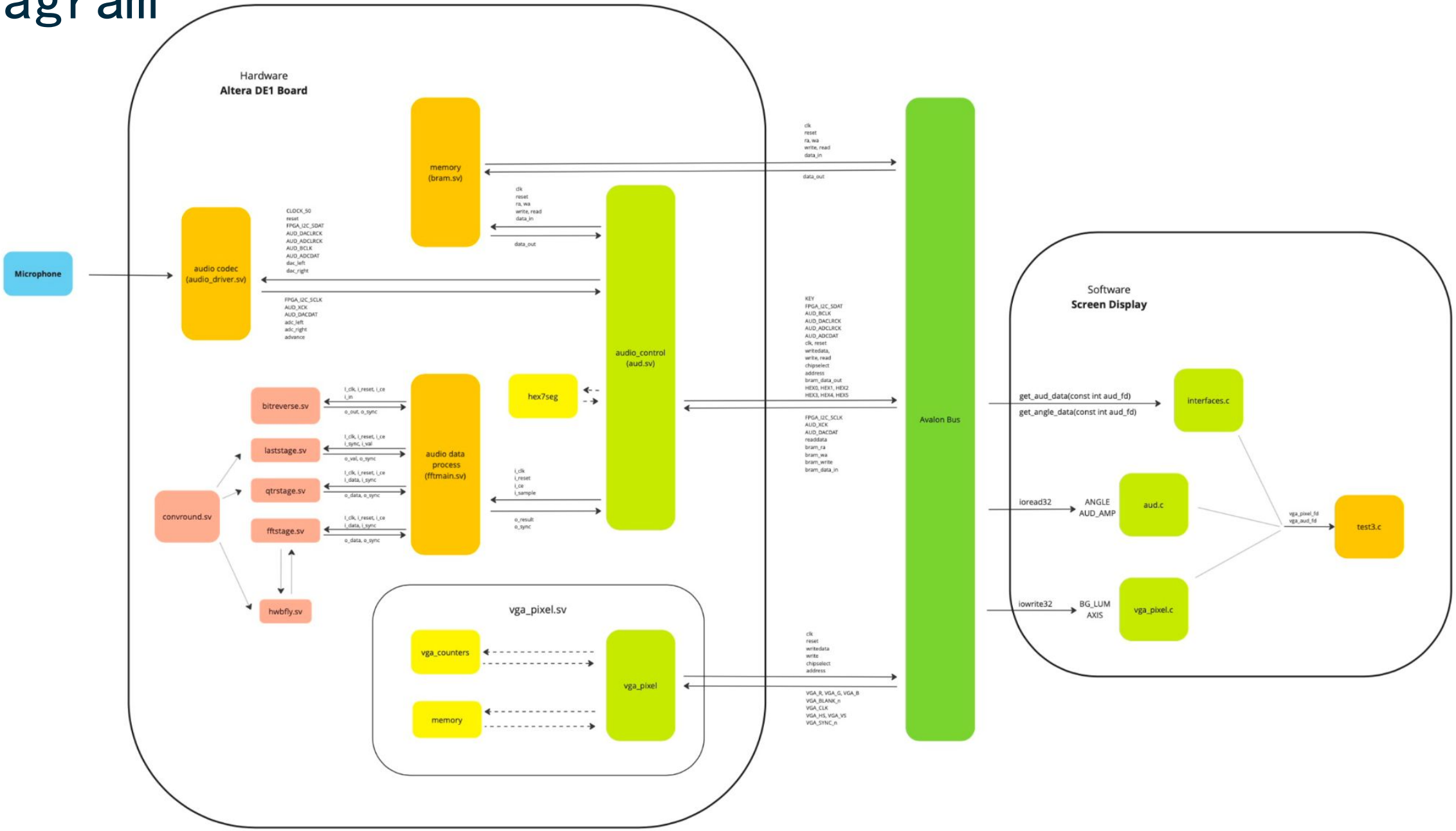
- **What we have…**

1. Audio Data Processing - Obtain audio from microphone
2. FFT(Fast Fourier transform) Calculation - Execute FFT on audio data
3. Data Visualization - Show FFT Results on VGA as dynamic waterfall waveform
4. Button Control - Adjust the shape of the visualization (30 - 60 degree)





Diagram

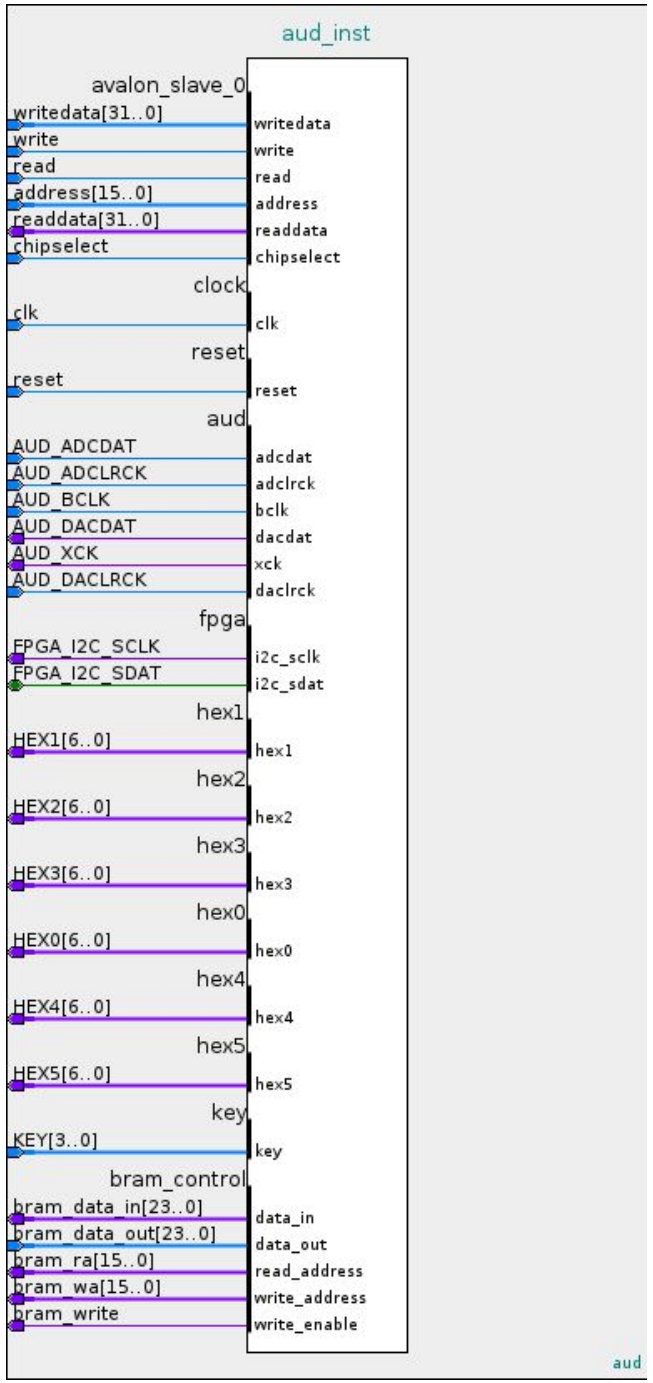

Hardware
Altera DE1 Board
Microphone
audio codec
(audio_driver.sv)
CLOCK_50
reset
FPGA_I2C_SDAT
AUD_DACLRCK
AUD_ADCLRCK
AUD_BCLK
AUD_ADCDAT
dac_left
dac_right
FPGA_I2C_SCLK
AUD_XCK
AUD_DACDAT
adc_left
adc_right
advance
memory
(bram.sv)
clk
reset
ra, wa
write, read
data_in
data_out
audio_control
(aud.sv)
KEY
FPGA_I2C_SDAT
AUD_BCLK
AUD_DACLRCK
AUD_ADCLRCK
AUD_ADCDAT
clk, reset
writedata,
write, read
chipselect
address
bram_data_out
HEX0, HEX1, HEX2
HEX3, HEX4, HEX5
FPGA_I2C_SCLK
AUD_XCK
AUD_DACDAT
readdata
bram_ra
bram_wa
bram_write
bram_data_in
Avalon Bus
bitreverse.sv
laststage.sv
qtrstage.sv
fftstage.sv
convround.sv
hwbfly.sv
audio data
process
(fftmain.sv)
hex7seg
i_clk
i_reset
i_ce
i_sample
o_result
o_sync
vga_pixel.sv
vga_counters
memory
vga_pixel
VGA_R, VGA_G, VGA_B
VGA_BLANK_n
VGA_CLK
VGA_HS, VGA_VS
VGA_SYNC_n
Software
Screen Display
get_aud_data(const int aud_fd)
get_angle_data(const int aud_fd)
interfaces.c
ioread32
ANGLE
AUD_AMP
aud.c
iowrite32
BG_LUM
AXIS
vga_pixel.c
vga_pixel_fd
vga_aud_fd
test3.c

Hardware

- Audio
- DIF FFT
- VGA Display

Hardware – Audio

The Audio module synthesizes most of the functions, including sound acquisition, FFT, memory management, key control, and Hex7seg.



Hardware – FFT Module

- **DIF FFT**

$$X[2k] = \sum_{n=0}^{\frac{N}{2}-1} x[n]W_{\frac{N}{2}}^{kn} + \sum_{n=\frac{N}{2}}^{\frac{N}{2}-1} x\left[n+\frac{N}{2}\right]W_{\frac{N}{2}}^{k\left(n+\frac{N}{2}\right)}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x[n]W_{\frac{N}{2}}^{kn} + \sum_{n=0}^{\frac{N}{2}-1} x\left[n+\frac{N}{2}\right]W_{\frac{N}{2}}^{kn}$$

$$= DFT_{\frac{N}{2}}\left\{x[n] + x\left[n+\frac{N}{2}\right]\right\}$$

$$X[2k+1] = DFT_{\frac{N}{2}}\left\{W_N^n\left(x[n] - x\left[n+\frac{N}{2}\right]\right)\right\}$$

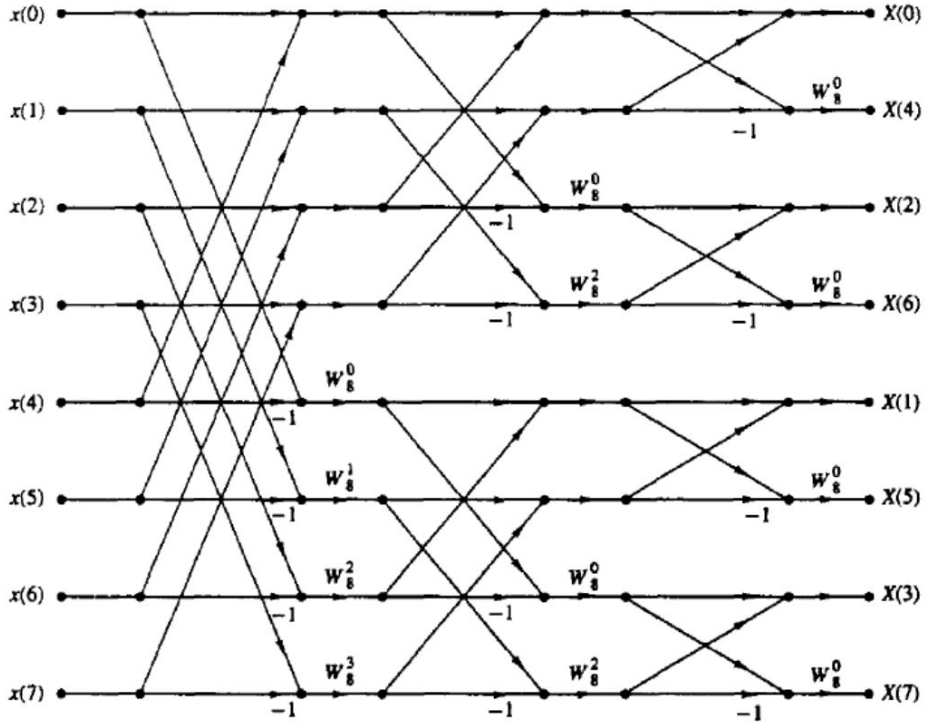
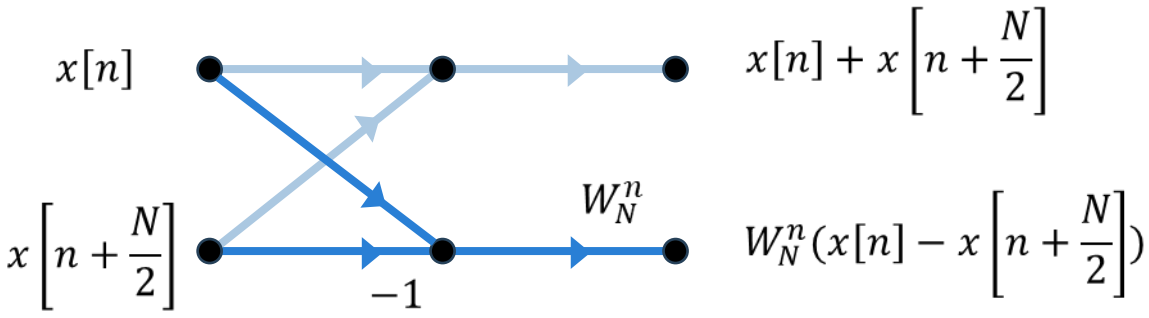


Figure: N=8 point decimation in frequency FFT algorithm

- **Difference between DIF and DIT FFT**
  - The output is bit-reversed
  - Multiplication is done after butterfly

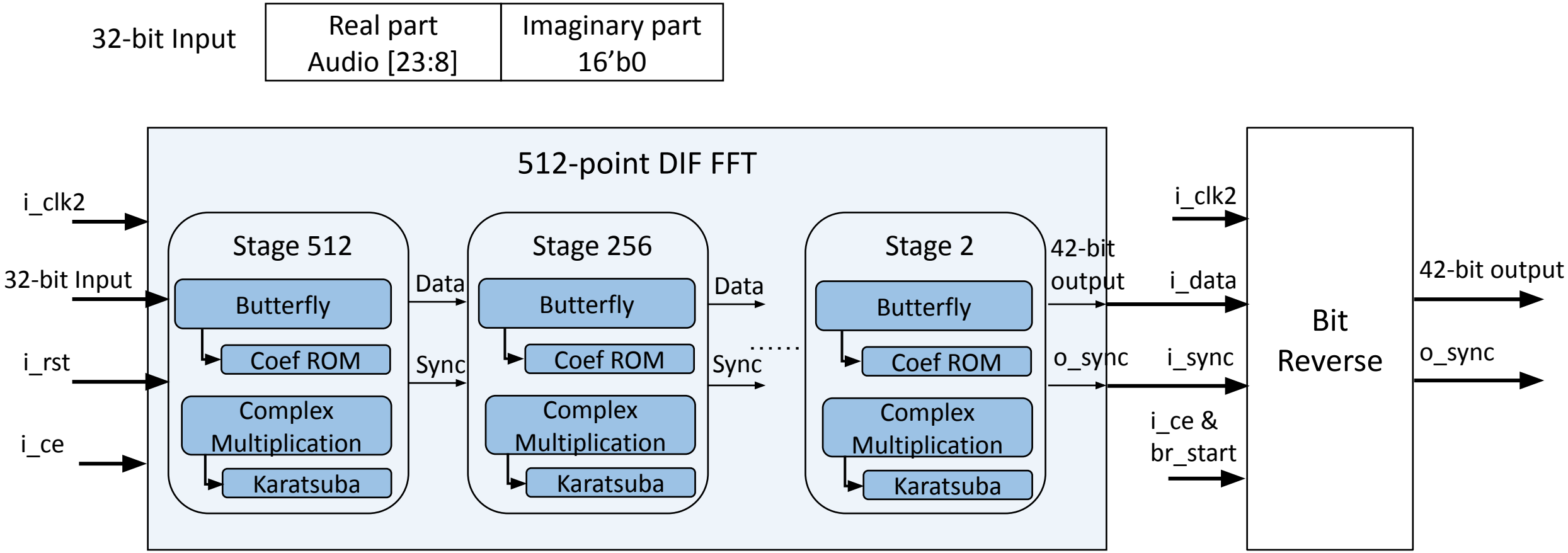


[1] THE FAST FOURIER TRANSFORM (FFT). (n.d.-a). https://eeweb.engineering.nyu.edu/iselesni/EL713/zoom/fft

[2] Proakis, J. G., & Manolakis, D. G. (1996). Digital Signal Processing: Principles, algorithms, and applications. Prentice Hall.

Hardware – FFT Module

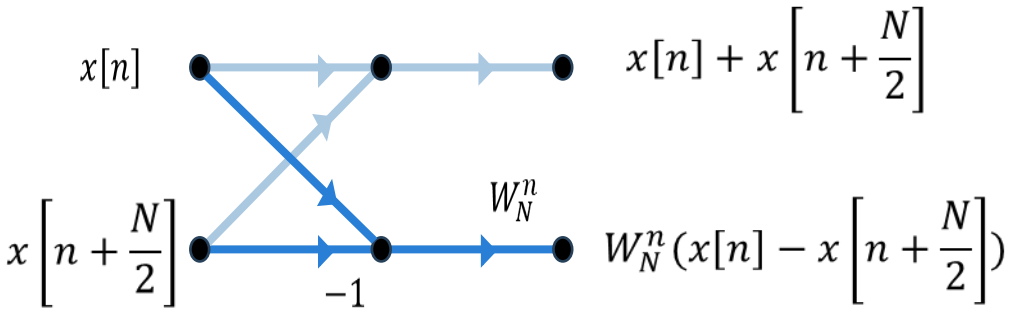
- **FFT Module Architecture**



[1] Dan Gisselquist (2017). A Generic Piplined FFT Core Generator. https://github.com/ZipCPU/dblclockfft/tree/master

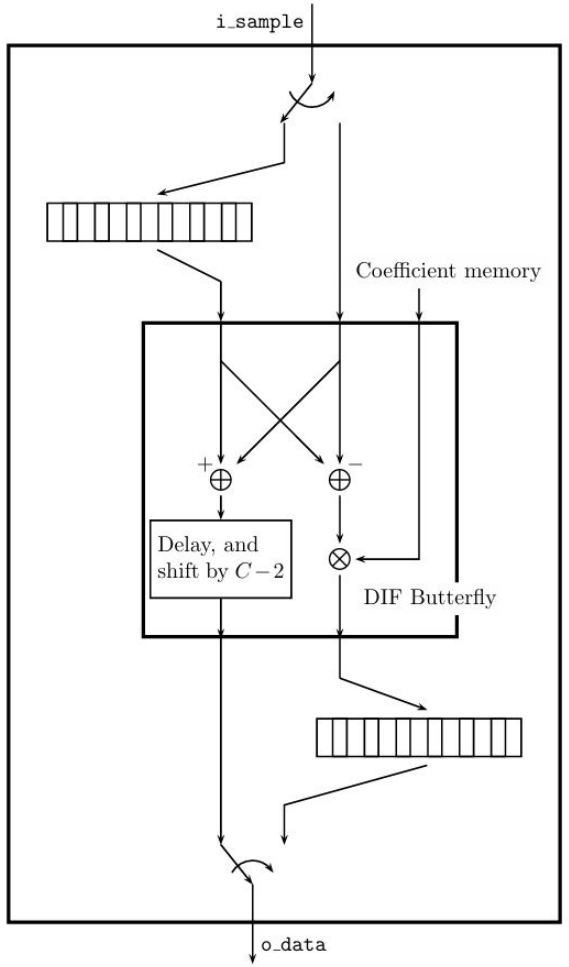
Hardware – FFT Module

- **Internal Stage Architecture**



1. Internal Calculation
   - Butterfly Operation
   - Complex Multiplication

2. Butterfly - Add and Subtraction
   - $O_1 = \boldsymbol{A} + \boldsymbol{B}$
   - $O_2 = (\boldsymbol{A} - \boldsymbol{B}) * C$



[1] Dan Gisselquist (2017). A Generic Piplined FFT Core Generator. https://github.com/ZipCPU/dblclockfft/tree/master

Hardware – FFT Module

- **Internal Stage Architecture**

3. Coefficient - Twiddle Factor $W_N^k = e^{-j\frac{2\pi k}{N}}$

  - Coefficients for Different Stages are Pre-Stored in Memory $W_N^{N-k} = -W_N^k$
  - The upper bits are the real part of the coefficient;
  - The lower bits are the imaginary part of the coefficient.

cmem_16.hex (N = 16, k : 0 ~ 7)

```
// Each line contains a coefficient.  The real portion
// of the coefficient is in the upper 23 bits, whereas
// the lower 23 bits contain the imaginary portion
//
//
100000000000
0ec83673c10f
0b504f695f62
061f78e26f94
000000600000
39e087e26f94
34afb1695f62
3137ca73c10f
```

Hardware – FFT Module

- **Fast Multiplication Algorithm - Karatsuba Algorithm**

1. Basic Idea

$$\begin{cases} M = m_1 + jm_2 \\ N = n_1 + jn_2 \end{cases}$$

$$M \times N = \begin{matrix} m_1 \cdot n_1 - m_2 \cdot n_2 \\ + m_1 \cdot n_2 j + m_2 \cdot n_1 j \end{matrix}$$

4 Multiplications

$$\begin{cases} P1 = m_1 \cdot n_1 \\ P2 = m_2 \cdot n_2 \\ P3 = (m_1 + m_2) \cdot (n_1 + n_2) \end{cases}$$

3 Multiplications

$$M \times N = (P1 - P2) + j(P3 - P1 - P2)$$

2. Implementation in Butterfly Operation in DIF FFT

$$\begin{matrix} A = a_1 + ja_2 \\ B = b_1 + jb_2 \\ C = c_1 + jc_2 \end{matrix} \rightarrow \begin{matrix} O_1 = A + B \\ \boldsymbol{O_2 = (A - B) * C} \end{matrix}$$

$$\begin{cases} P1 = (a_1 - b_1) \cdot c_1 \\ P2 = (a_2 - b_2) \cdot c_2 \\ P3 = [(a_1 - b_1) + (a_2 - b_2)] \cdot (c_1 + c_2) \end{cases}$$

$$\rightarrow O_2 = (P1 - P2) + j(P3 - P1 - P2)$$

Hardware – FFT Module

- **Fast Multiplication Algorithm - Karatsuba Algorithm**

$$A = a_1 + ja_2$$
$$B = b_1 + jb_2$$
$$C = c_1 + jc_2$$

$$O_1 = A + B$$
$$O_2 = (A - B) * C$$

$$P1 = (a_1 - b_1) \cdot c_1$$
$$P2 = (a_2 - b_2) \cdot c_2$$
$$P3 = [(a_1 - b_1) + (a_2 - b_2)] \cdot (c_1 + c_2)$$

```
always @(posedge i_clk)
if (i_ce)
begin
    // One clock just latches the inputs
    r_left <= i_left;   // No change in # of bits
    r_right <= i_right;
    r_coef  <= i_coef;
    // Next clock adds/subtracts
    r_sum_r <= r_left_r + r_right_r; // Now IWIDTH+1 bits
    r_sum_i <= r_left_i + r_right_i;
    r_dif_r <= r_left_r - r_right_r;
    r_dif_i <= r_left_i - r_right_i;
    // Other inputs are simply delayed on second clock
    ir_coef_r <= r_coef[(2*CWIDTH-1):CWIDTH];
    ir_coef_i <= r_coef[(CWIDTH-1):0];
end
```

$A + B$

$A - B$

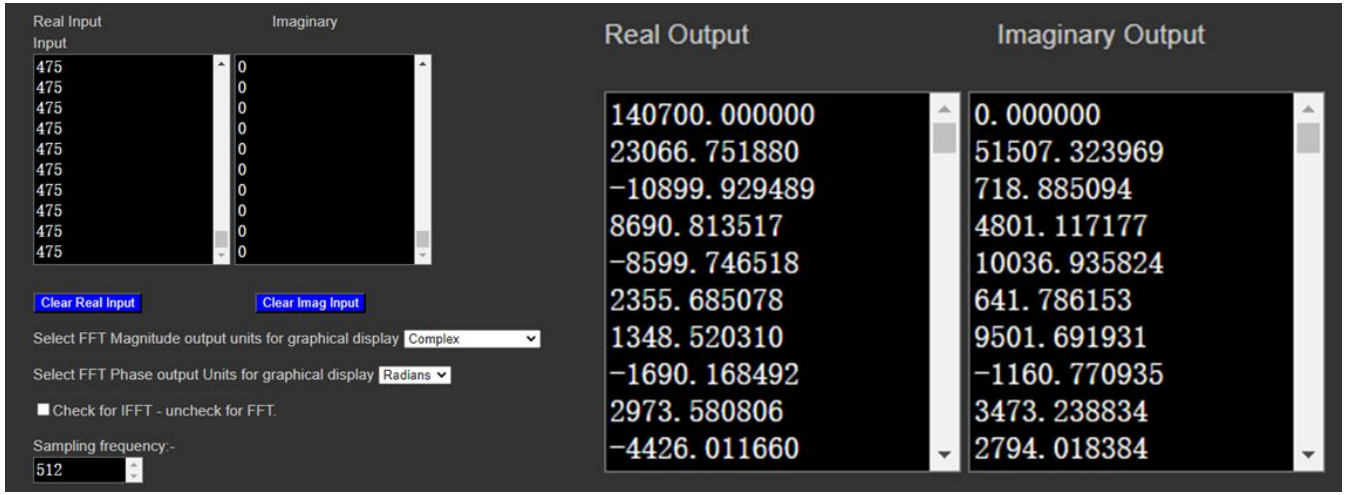
```
always @(posedge i_clk)
if (i_ce)
begin
    // Second clock, pipeline = 1
    p1c_in <= ir_coef_r;  //c1
    p2c_in <= ir_coef_i;  //c2
    p1d_in <= r_dif_r;    //a1-b1
    p2d_in <= r_dif_i;    //a2-b2
    p3c_in <= ir_coef_i + ir_coef_r; // c1+c2
    p3d_in <= r_dif_r + r_dif_i;  //a1-b1+a2-b2
end
// }}}

// Perform our multiplies

always @(posedge i_clk)
if (i_ce)
begin
    // Third clock, pipeline = 3
    //   As desired, each of these lines infers a DSP48
    rp_one   <= p1c_in * p1d_in;
    rp_two   <= p2c_in * p2d_in;
    rp_three <= p3c_in * p3d_in;
end
```

$(A - B) * C$

Hardware – FFT Module

- **FFT Result Checking**

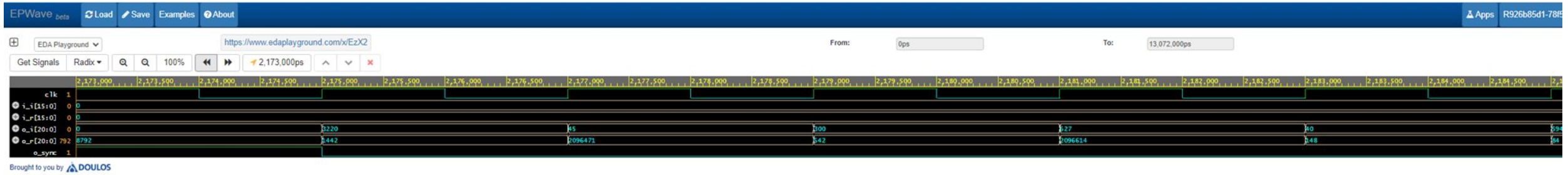
Mathematical FFT Output



Testbench – Simulation of audio input data

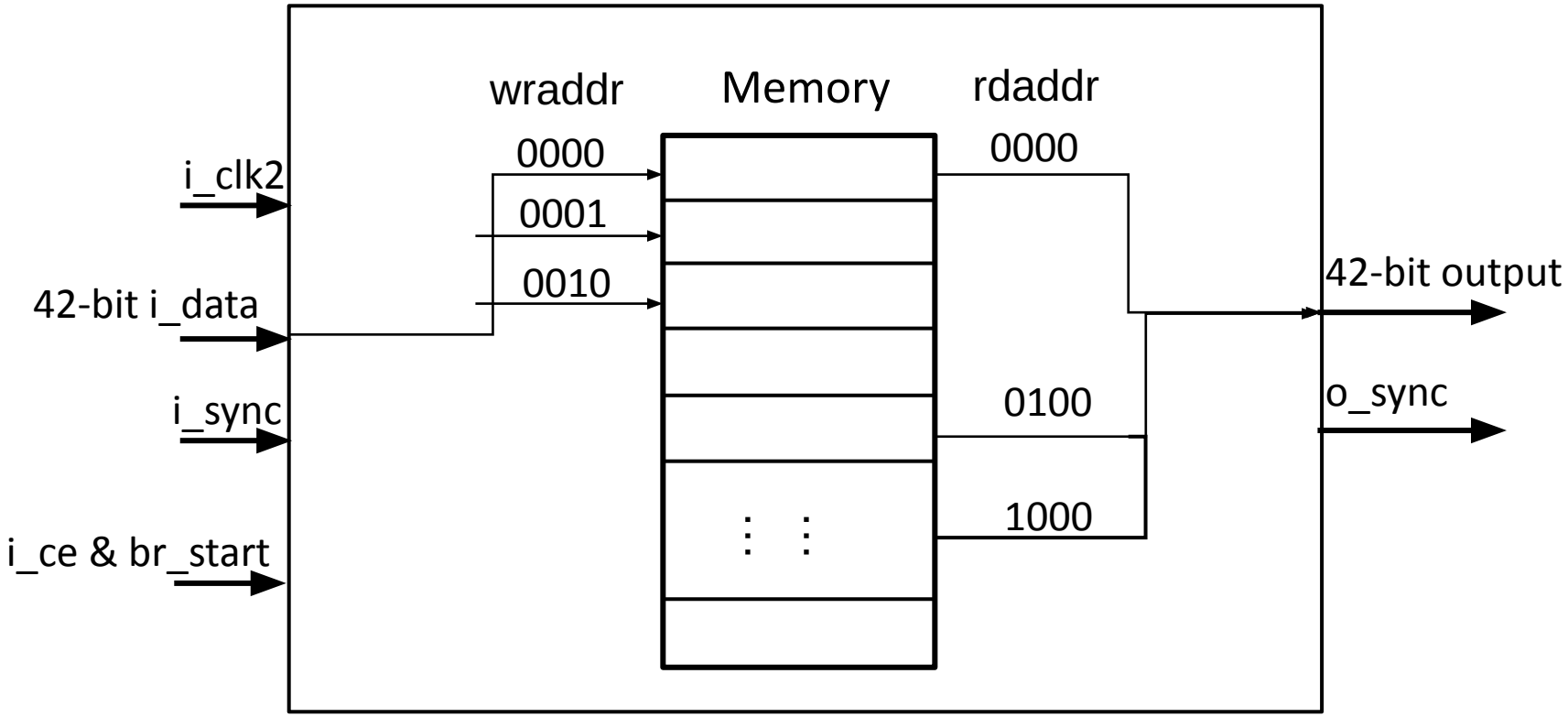
```
//Test stimulus
initial begin

    // Test Case 1
    i_sample = {16'd200, 16'd0}; #200
    i_sample = {16'd100, 16'd0}; #400
    i_sample = {16'd475, 16'd0}; #424
```

Output Waveform



Hardware – FFT Module

- **Bit Reverse Module**



- Memory size: 2^9=512 42-bit data
- The `rdaddr` is computed as the bit-reversal of `wraddr`
- o_sync is set 1 when the bit-reversal process is done and the data output is valid.

Hardware – FFT Module

- **Square Root Calculator Function**

Step	A	X	T	Q	Description
	00000000	01111001	00000000	0000	Starting values.
1	00000001	11100100			Left shift X by two places into A.
			00000000		Set T = A - {Q,01}: 01 - 01.
				0000	Left shift Q.
	00000000			0001	Is T≥0? Yes. Set A=T and Q[0]=1.
2	00000011	10010000			Left shift X by two places into A.
			11111100		Set T = A - {Q,01}: 11 - 101.
				0010	Left shift Q.
					Is T≥0? No. Move to next step.
3	00001110	01000000			Left shift X by two places into A.
			00000101		Set T = A - {Q,01}: 1110 - 1001
				0100	Left shift Q.
	00000101			0101	Is T≥0? Yes. Set A=T and Q[0]=1.
4	00010101	00000000			Left shift X by two places into A.
			00000000		Set T = A - {Q,01}: 10101 - 10101.
				1010	Left shift Q.
	00000000			1011	Is T≥0? Yes. Set A=T and Q[0]=1.

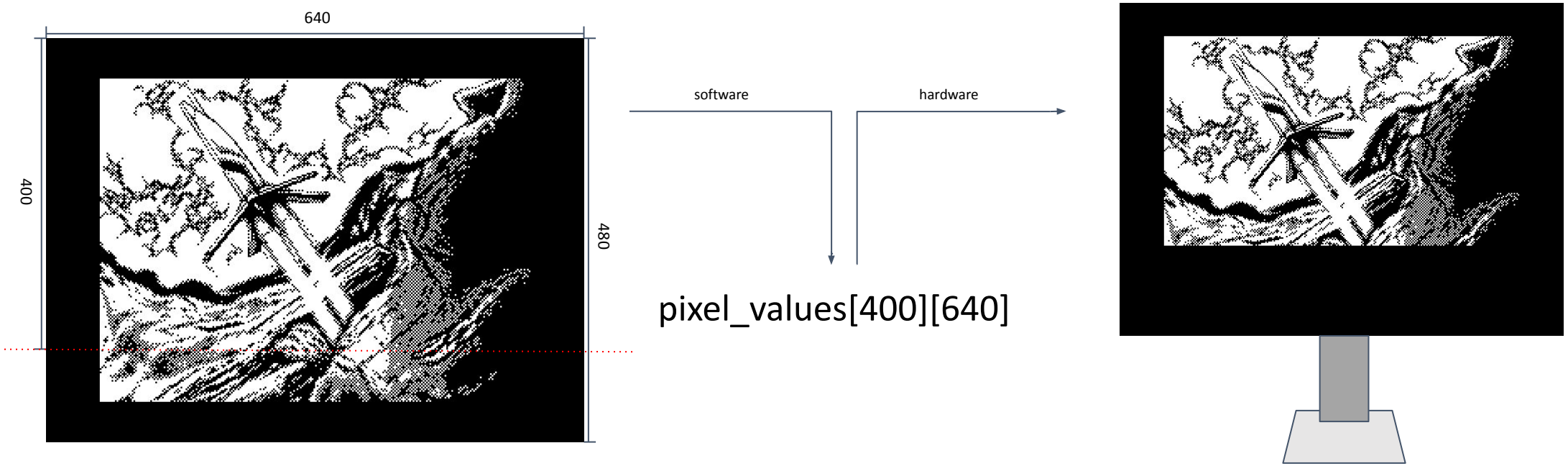
1. Left shift the largest 2 bit of number and add with remain.

2. Compare the value to {answer,01}

3. Update lowest bit of answer and remain value.

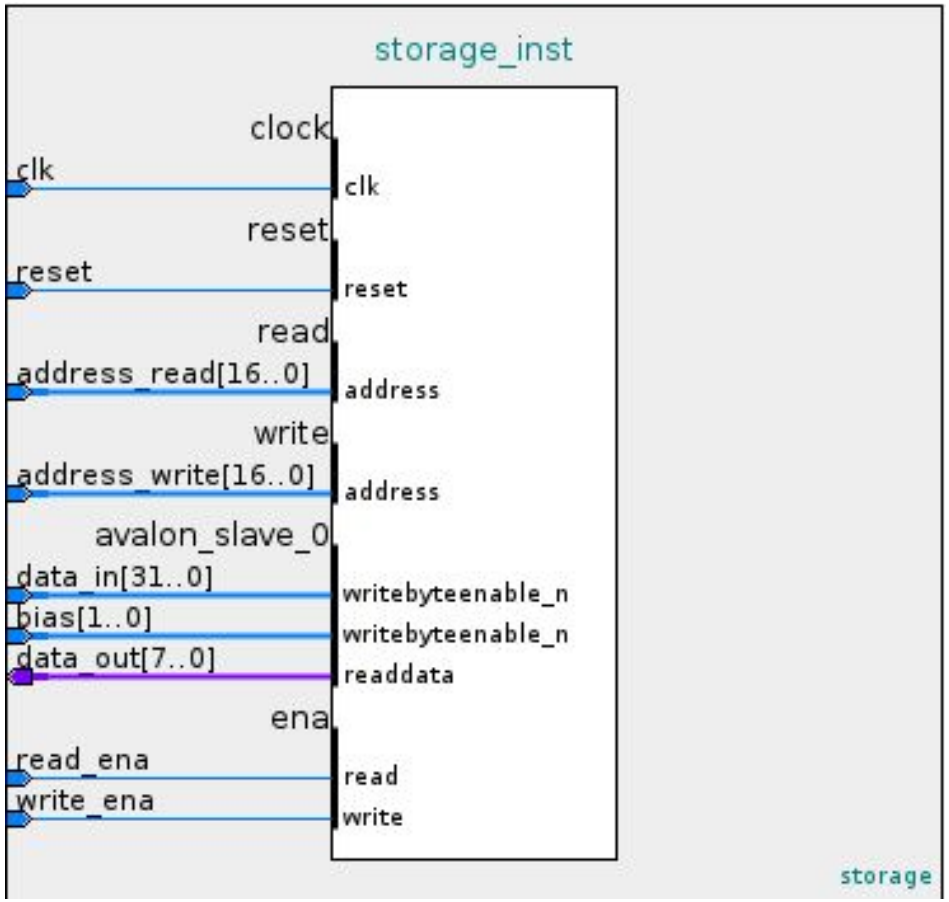
Hardware – VGA

- **Design**
  - 256000(640 * 400) bit grayscale output



Hardware – VGA

- **Frame Storage**
  - Use memory to store the frame
  - Read memory through VGA counter cycle



Hardware-Software Interface

Register Map

		Bits																													
	Address	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
vga_pixel	0	8-bit pixel value								8-bit pixel value								8-bit pixel value								8-bit pixel value						
	1	16-bit x-axis																16-bit y-axis															
Aud	2																											6-bit angle					
	3	32-bit aud data																															

- **Memory-Mapped Registers:**
  - vga_pixel: Address 0 for pixel values.
    Address 1 for pixel positions.
  - aud: Address 2 for angle data.
    Address 3 for audio data.

- **Data Flow:**
  - Software to Hardware:
    Writes pixel colors and positions to registers.
  - Hardware to Software:
    Reads processed audio data and angle data to visualize waveforms.

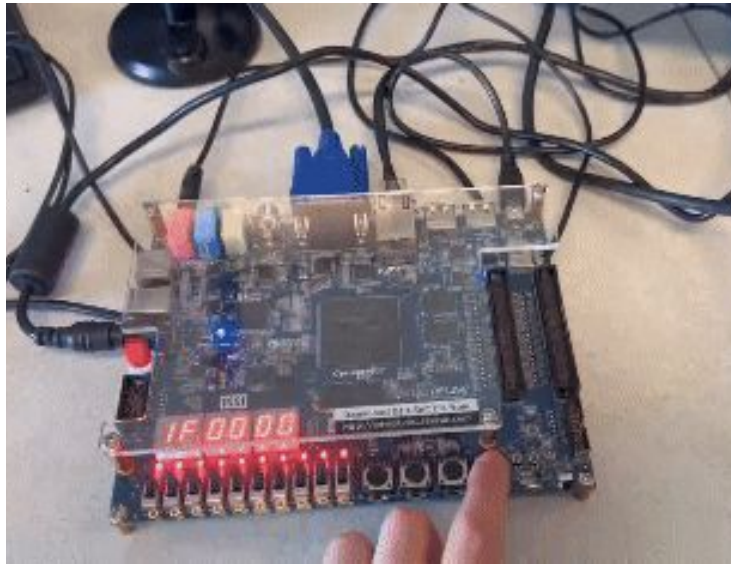

Software

Software

- **Data Capture**
  - Audio Data Acquisition

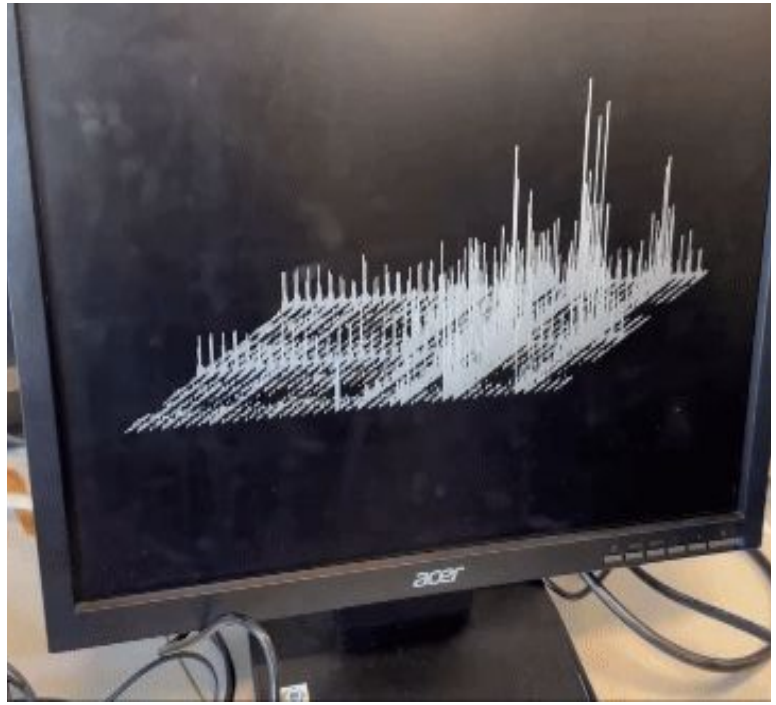
    get_aud_data(aud_fd)
  - Angle Data Acquisition

    get_angle_data(aud_fd)



- **Data Processing and Waveform Generation**
  - Frames and CUR Arrays
    - Frames: Stores historical audio data over multiple frames.
    - CUR: Holds the current state of the waveform to be displayed.
  - Waveform Calculation
    - Function f(CUR, Frames, ang) processes audio data and applies a rotational angle to generate the waveform.

Software



- **Displaying the Waveform**
  - Clearing the Screen
    - Initial screen clear to reset previous data
  - Setting Pixels:
    - Iteratively place pixels on the screen according to waveform data.
    - Use set_pixel_axis to position pixels and set_background_color to set intensity.
  - Continuous Refresh
    - The screen is updated continuously as new audio data is processed.

Demo