

# Realtime Sound Localization

Matheu Campbell, Elvis Wang, Peiran Wang, Dawn Yoo

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

# Project Objectives

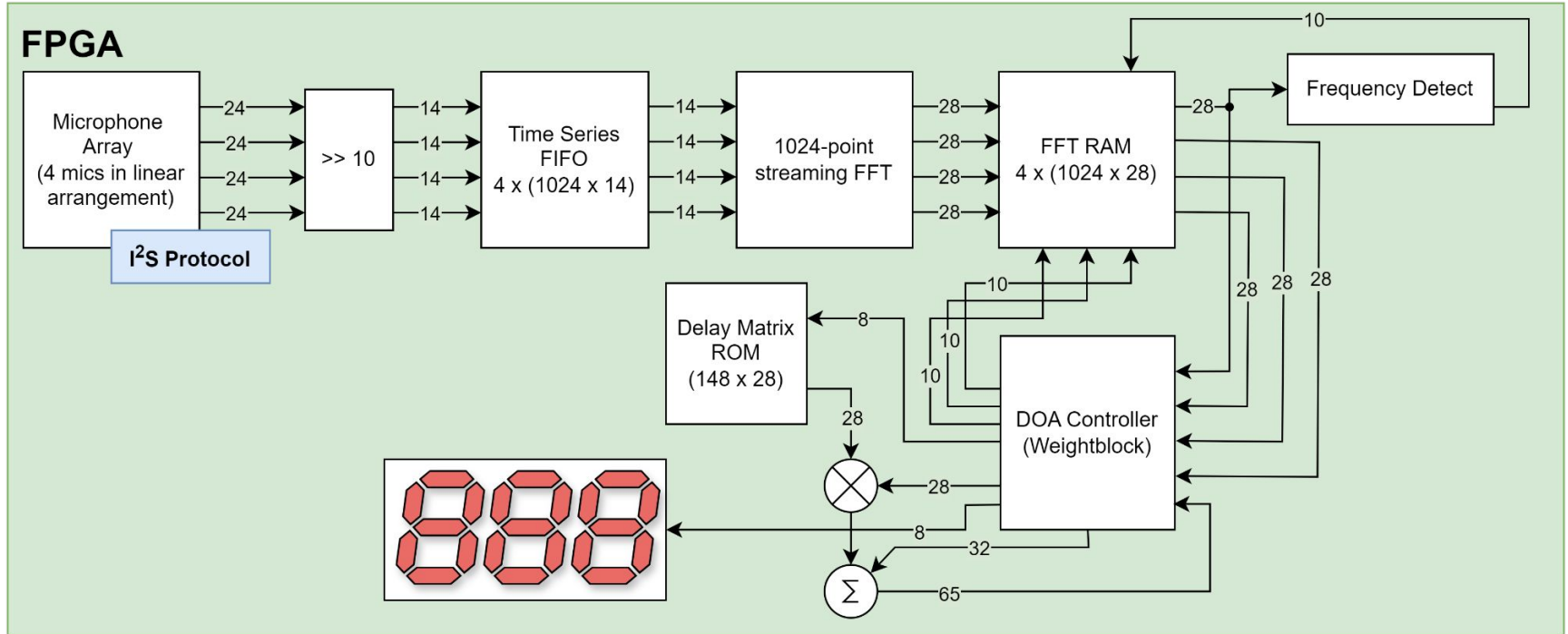
```
graph LR; A(Record Sound Data) --> B(Execute DOA Algorithm); B --> C(Display Direction);
```

Record Sound Data

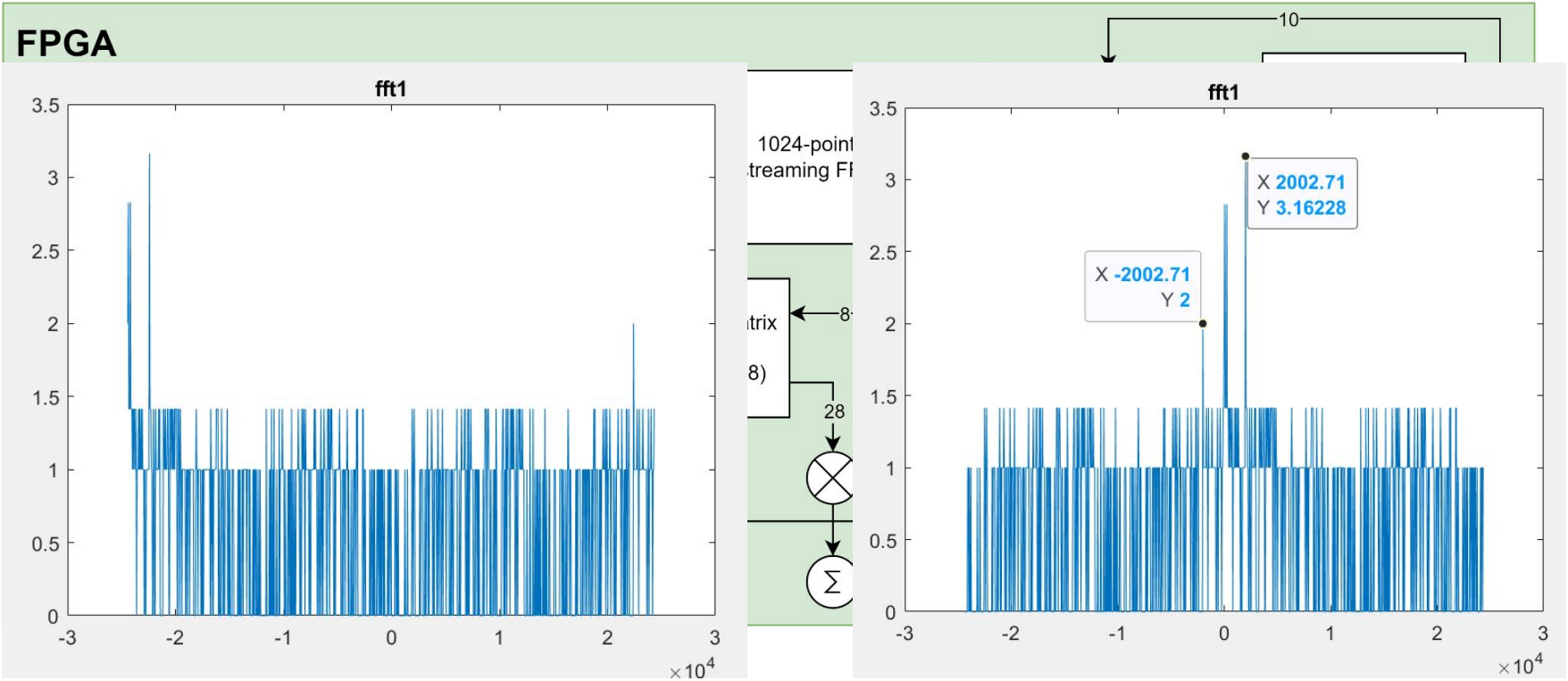
Execute DOA  
Algorithm

Display  
Direction

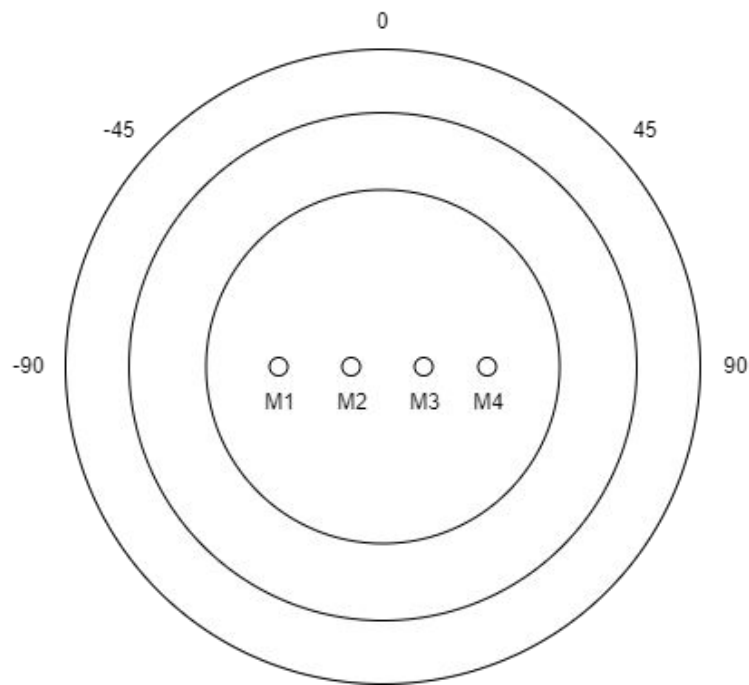
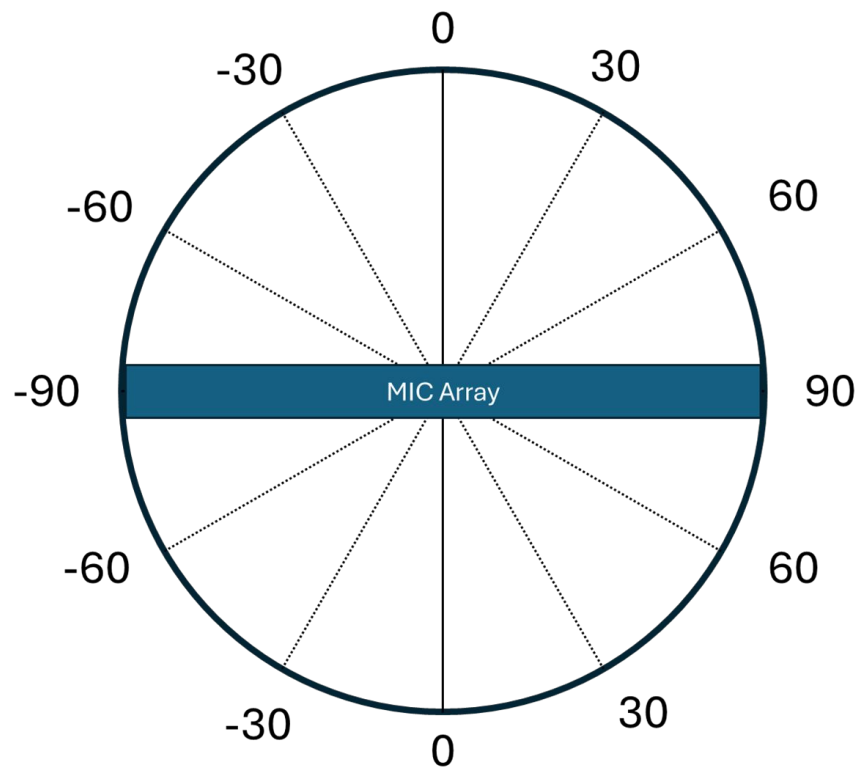
# System Block Diagram (single-axis system, duplicated for dual-axis)



# System Block Diagram (single-axis system, duplicated for dual-axis)



# Visual Diagram



# I2S Interface

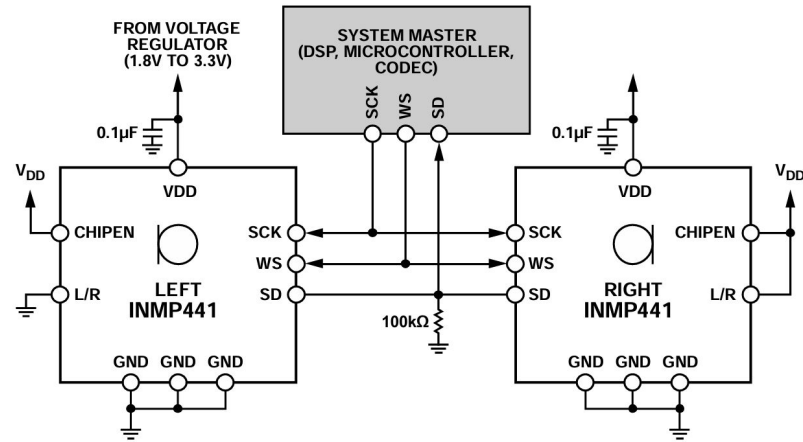
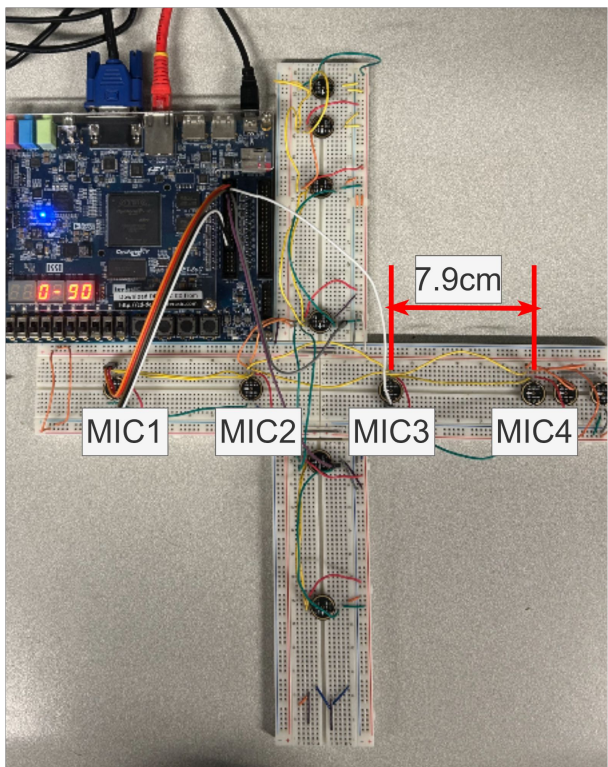
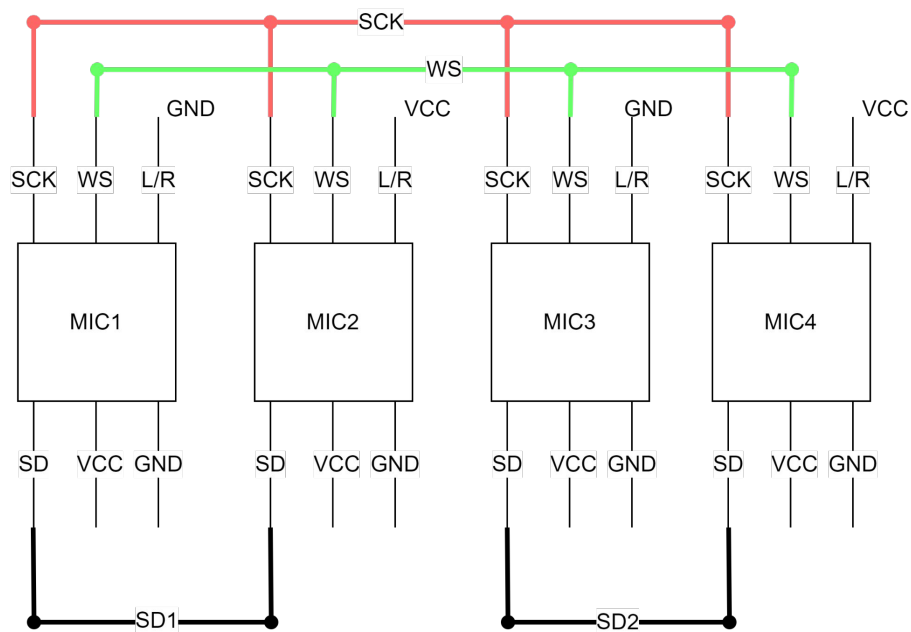


Figure 7. System Block Diagram

# I2S Interface



## Single-axis System



# I2S Interface

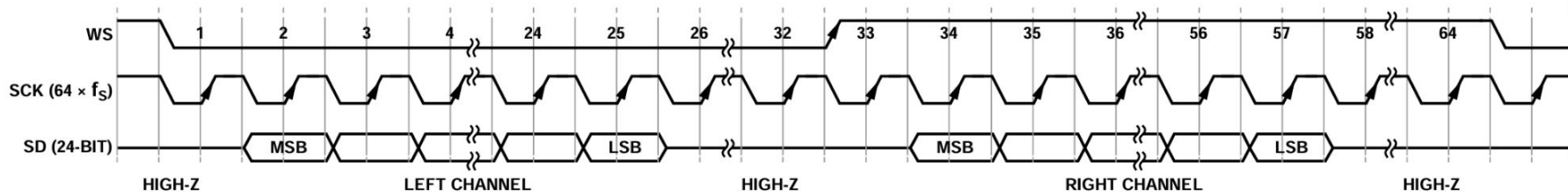


Figure 8. Stereo-Output I<sup>2</sup>S Format

```
if (clk_cnt > 0 && clk_cnt < 25) begin // Left channel, 24-bit dept, MSB first
    left1 <= {left1[22:0], SD1};
    left2 <= {left2[22:0], SD2};
    left3 <= {left3[22:0], SD3};
    left4 <= {left4[22:0], SD4};
end else if (clk_cnt > 32 && clk_cnt < 57) begin // Right channel
    right1 <= {right1[22:0], SD1};
    right2 <= {right2[22:0], SD2};
    right3 <= {right3[22:0], SD3};
    right4 <= {right4[22:0], SD4};
end
```

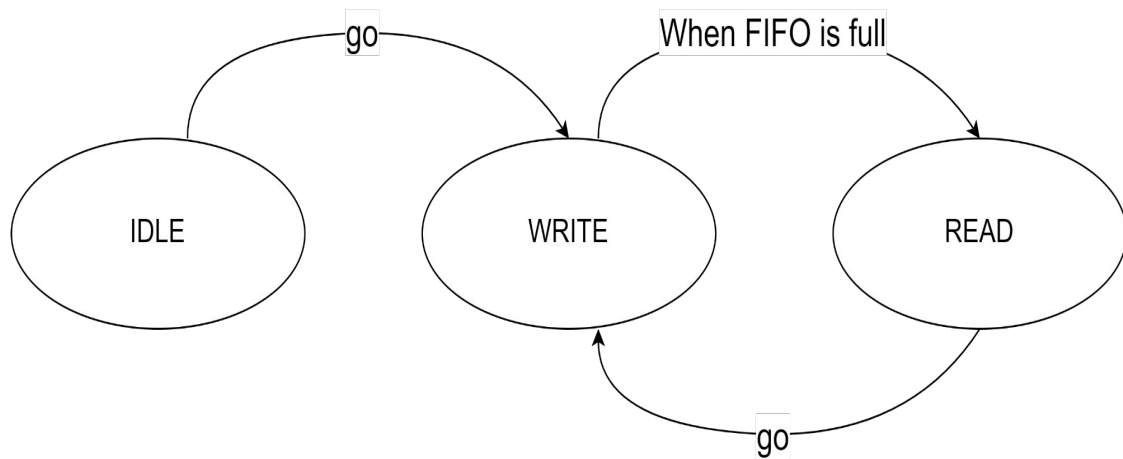


# I2S Interface

```
case (state)
  IDLE: begin
    if (go_SCK)
      state <= WRITE;
    else
      state <= IDLE;
    end
  WRITE: begin
    if (clk_cnt == 57) begin
      ram1_in <= left1[23:8]; // Discard the lesast 8 bits
      ram2_in <= right1[23:8];
      ram3_in <= left2[23:8];
      ram4_in <= right2[23:8];
      ram5_in <= left3[23:8];
      ram6_in <= right3[23:8];
      ram7_in <= left4[23:8];
      ram8_in <= right4[23:8];

      wrreq <= 1'd1;
      wr_addr <= wr_addr + 11'd1; // Start with address 0
    end else if (clk_cnt == 58) begin
      wrreq <= 1'd0;
    end

    if (wr_addr == 10'd1023)
      state <= READ;
    else
      state <= WRITE;
    end
  READ: begin
    wr_addr <= 11'd2047;
    if (go_SCK) begin
      state <= WRITE;
    end else begin
      state <= READ;
    end
  end
end
```



# FFT with RAM– Implementation

**Input:** raw data input, read address of fft RAM

**Output:** FFT RAM results, read request, ready signal

**Quartus IP core:** FFT IP core, RAM IP core

```
module fft_wrapper(  
    input logic clk,  
    input logic rst_n,           // Active low  
    input logic go,             // Reset FSM  
    input logic ready,         // Raw data RAMs ready to be read  
    input logic [13:0]data_in,  // Raw data input  
    input logic [9:0]rd_addr_fft, // Read address of fft RAMs  
  
    output logic fftdone,       // fft wrapper output ready  
    output logic rdreq,        // Read request  
    output logic [27:0]ram_q    // fft results from fft RAM  
);
```

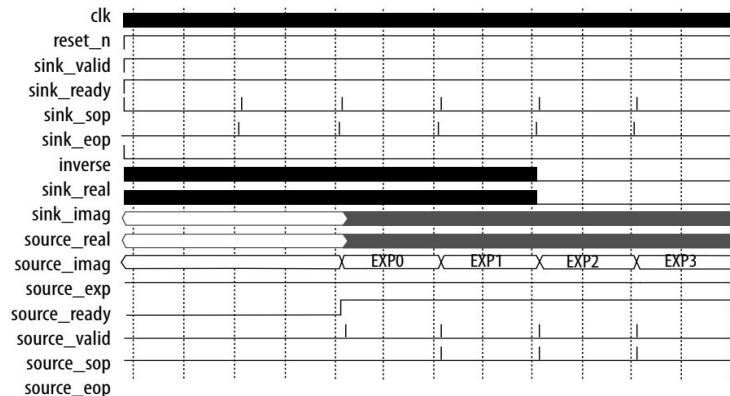
# FFT – Algorithm

```
always @(posedge clk) begin
    if (~rst_n) begin
        count = 10'd1;
        sink_valid <= 1'd0;
        sink_eop <= 1'd0;
        sink_sop <= 1'd0;
        sor <= 1'd0;
    end else begin
        count <= count + 1'b1;
        if (count == 10'd0) begin
            sink_eop <= 0;
            sink_sop <= 1;
            sink_valid <= 1;
        end else if (count == 10'd1) begin
            sink_sop <= 0;
        end else if (count == 10'd1021) begin
            sor <= 1;
        end else if (count == 10'd1022) begin
            sor <= 0;
        end else if (count == 10'd1023) begin
            sink_eop <= 1;
        end
    end
end
end
```

Control signals to ensure we are using 1024-point data streaming FFT

eop : end of packet  
sop : start of packet  
sor : start of raw data

Figure 11. FFT Streaming Data Flow Simulation Waveform



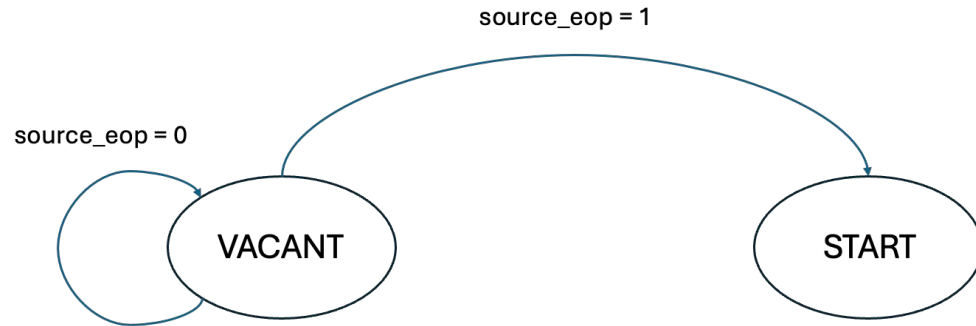
# RAM – Algorithm

```
always @(posedge clk) begin
  if (~rst_n) begin
    wr_addr <= 10'd0;
    state_wr_addr <= VACANT;
  end else begin
    case (state_wr_addr)
      VACANT: begin
        wr_addr <= 10'd0;
        if (source_eop)
          state_wr_addr <= START;
        else
          state_wr_addr <= VACANT;
      end
      START: begin
        wr_addr <= wr_addr + 10'd1;
      end
      default: state_wr_addr <= VACANT;
    endcase
  end
end
```

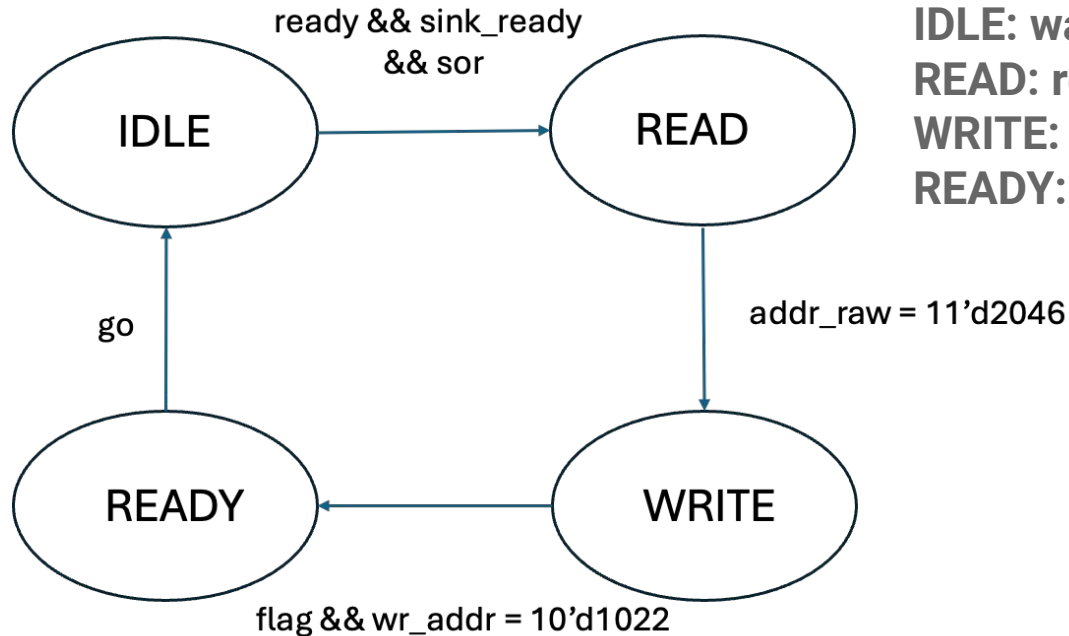
## Control signals for FFT RAM

**VACANT:** wait until source\_eop is high

**START:** wr\_addr signal stream for FFT RAM



# FFT with RAM- Algorithm



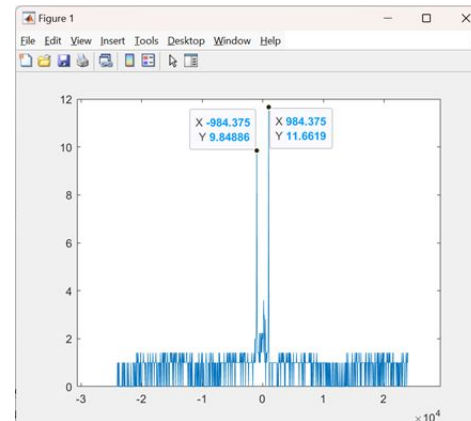
**IDLE:** wait for ready signals (RAM/FFT)

**READ:** read data out of RAM (FIFO)

**WRITE:** write FFT results to RAM at `source_eop`

**READY:** get ready to read new data

# FFT with RAM- Waveform Simulation



Continuous stream of control signals

FFT data result

FSM states  
WRITE state when wren is high

# Frequency Detection – Implementation

freqdetect.sv

**Input:** Full FFT output for one channel

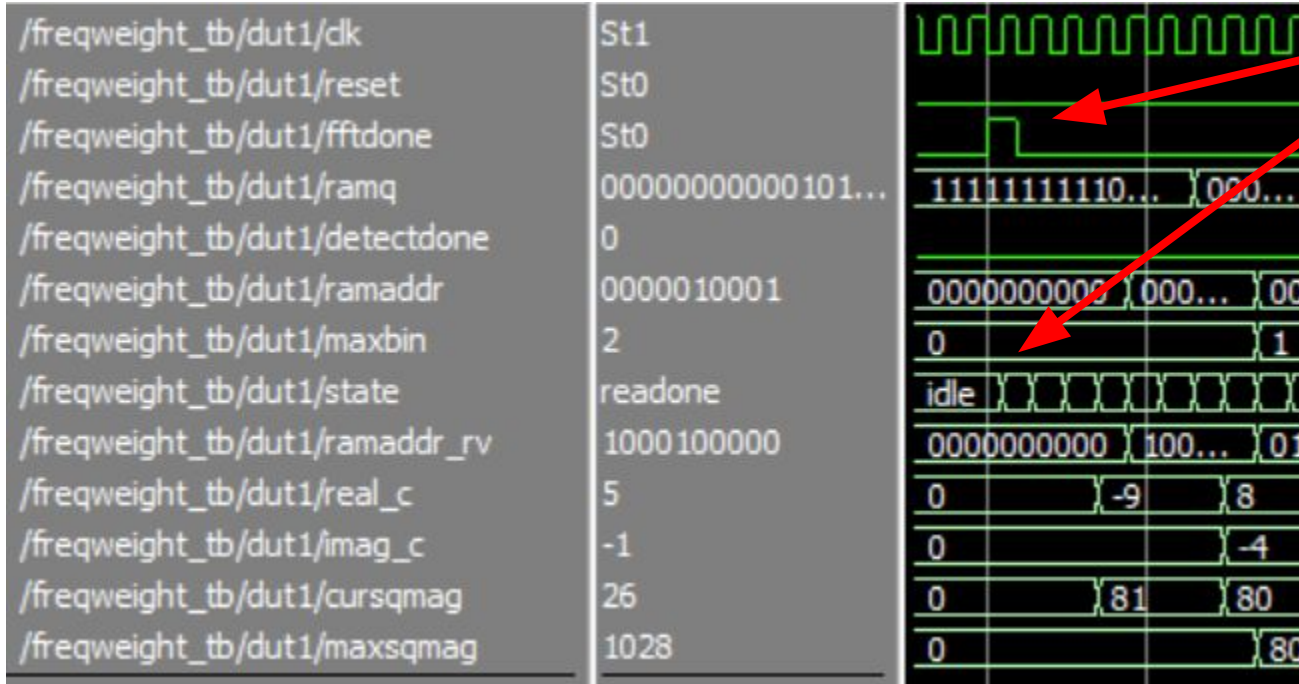
**Output:** Index of the FFT bin corresponding to the frequency of highest magnitude\*

**Algorithm:** Iterate through all bins, updating maxbin when  $|\text{signal}|^2$  exceeds the running max

```
module freqdetect(  
    input logic clk,           // 50 MHz, 20 ns  
    input logic reset,        // Reset key is 0  
    input logic fftdone,      // Set high upon FFT block finishing  
    input logic [27:0] ramq,   // Output port of channel 1 FFT RAM  
  
    output logic detectdone,   // Set high when iteration is complete  
    output logic [9:0] ramaddr, // Address to read from RAM  
    output logic [9:0] maxbin  // Index of max bin  
);
```

\*except extremely low frequency/DC components

# Frequency Detection - Simulation

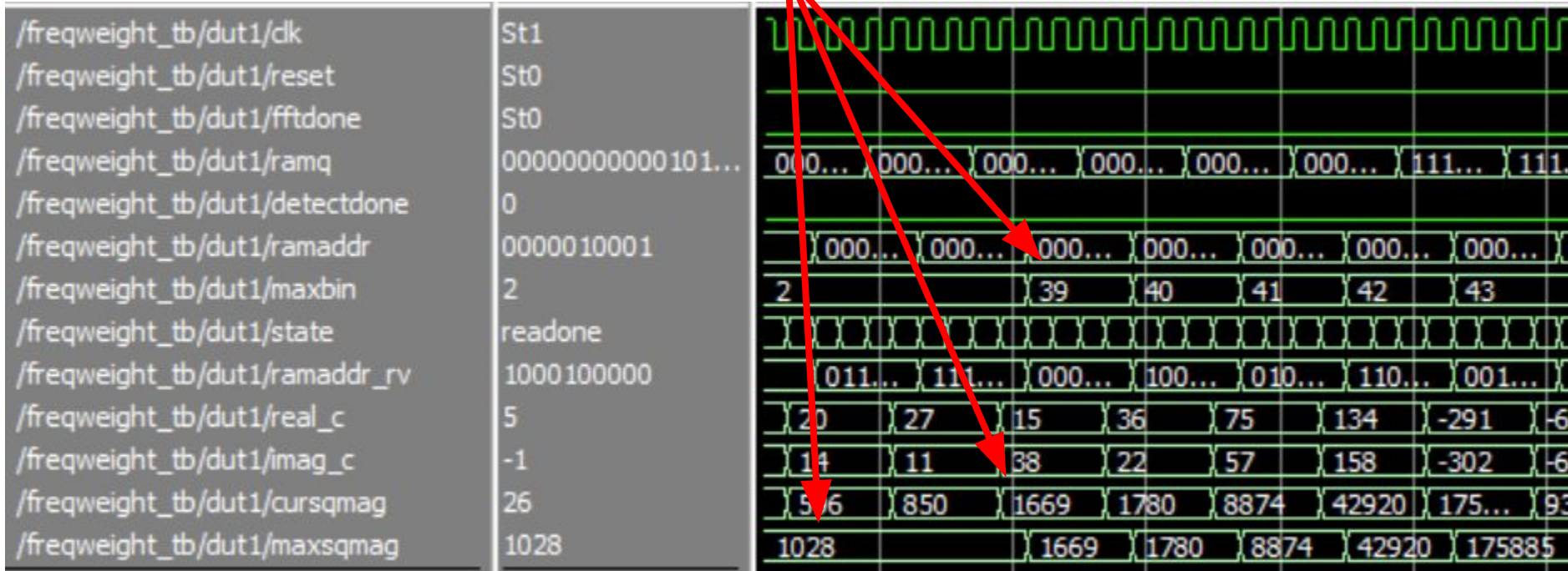


fftdone signal  
triggers beginning  
of iteration

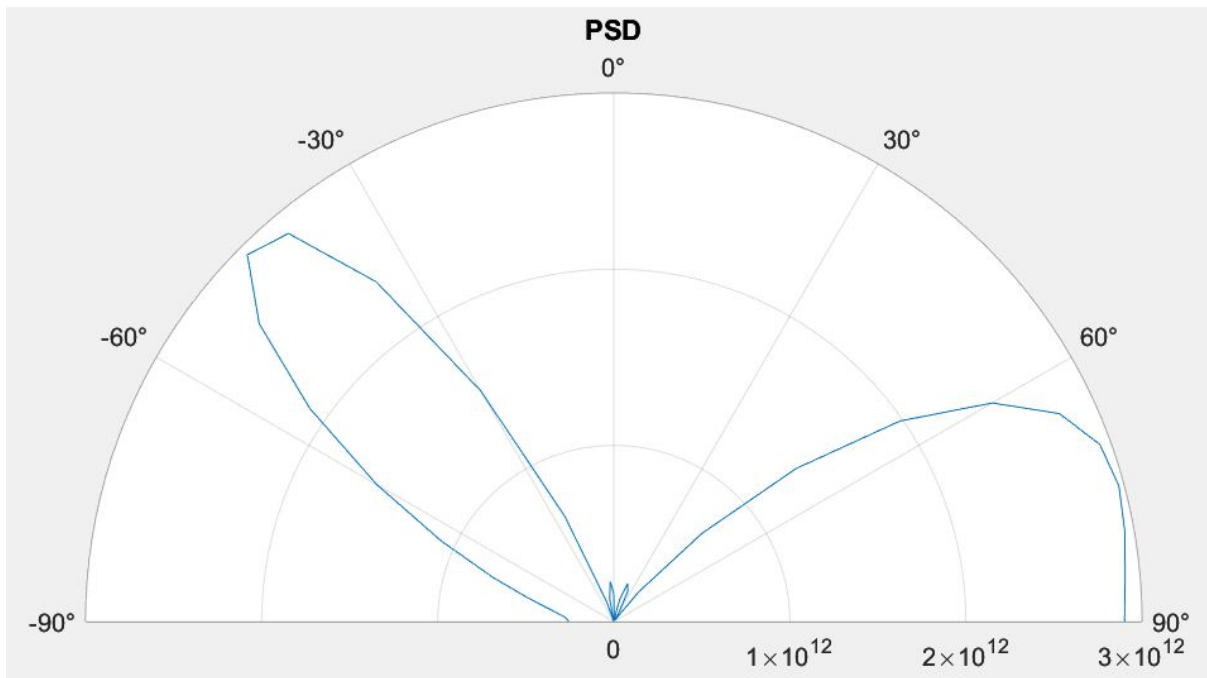


# Frequency Detection - Simulation

Maxbin updated when cursqmag > maxqsqmag



# DOA Estimation



Spectrogram for test data corresponding to 75 degree DOA

## Algorithm

Compute the values in this spectrogram sequentially, storing the direction of peak magnitude. When all directions have been processed, the direction corresponding to the maximum is taken as the DOA.

# Computing the spatial PSD

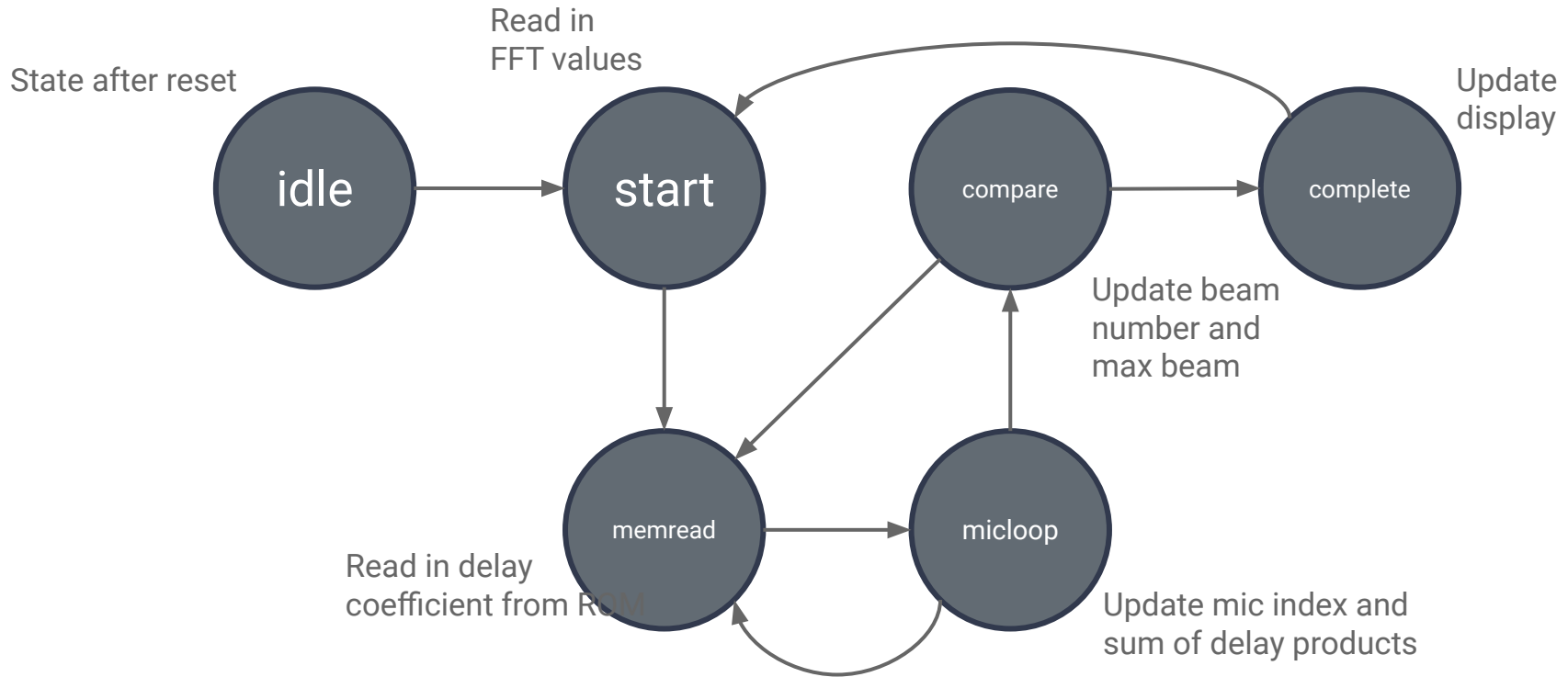
$$|\text{array output}_j|^2 = \left| \sum_{i=0}^3 \text{FFT}_i \cdot D_{ij} \right|^2 \quad \text{for the } j^{\text{th}} \text{ direction, mics labeled 0 through 3}$$

Delay matrix **D** is stored in ROM which is preloaded at compile-time. Multiplications are performed by dedicated IP blocks.





When complete, the 7-segment displays are updated with the newly estimated DOA.

**Relevant Files:** weightblock.sv, angdisplay.sv, realmult.v, compmult.v, delay\_ROM.v













# Weightblock FSM



# Weightblock Simulation

  /freqweight_tb/dut2/bnum	17	0								
  /freqweight_tb/dut2/doa	-45	-90								
			1	2	3	4	5	6	7	
				-85	-80	-75	-70	-65	-60	

Module iterates through all directions (bnum), updating doa as larger array outputs are calculated

 /freqweight_tb/dut2/done	0									
  /freqweight_tb/dut2/bnum	17									
  /freqweight_tb/dut2/doa	-45									
  /freqweight_tb/dut2/disp2	0111111									1111111
  /freqweight_tb/dut2/disp1	0111111									1111000
  /freqweight_tb/dut2/disp0	0111111									0010010
 /freqweight_tb/dut2/state	memread									complete
			31	32	33	34	35	36		
		-45				75				

When done iterating, the state becomes complete, done signal goes high, and the displays are set to the correct values (75 for this set of test data)

[FPGA Implementation of a Bartlett Direction of Arrival Algorithm for a 5.8GHz Circular Antenna Array](#)

[A new direction-of-arrival estimation method using automotive radar sensor arrays](#)

[Minimum Variance Distortionless Response](#)