16-QAM
Demapping Module

Zach Martin
Saunders Riley
EE417 Spring 2022

Description

Objective

- Decode 16-QAM Digital Data
- Store data in a FIFO for access by external device or module
- No error correction or filtering ("Hard Decision demapping")

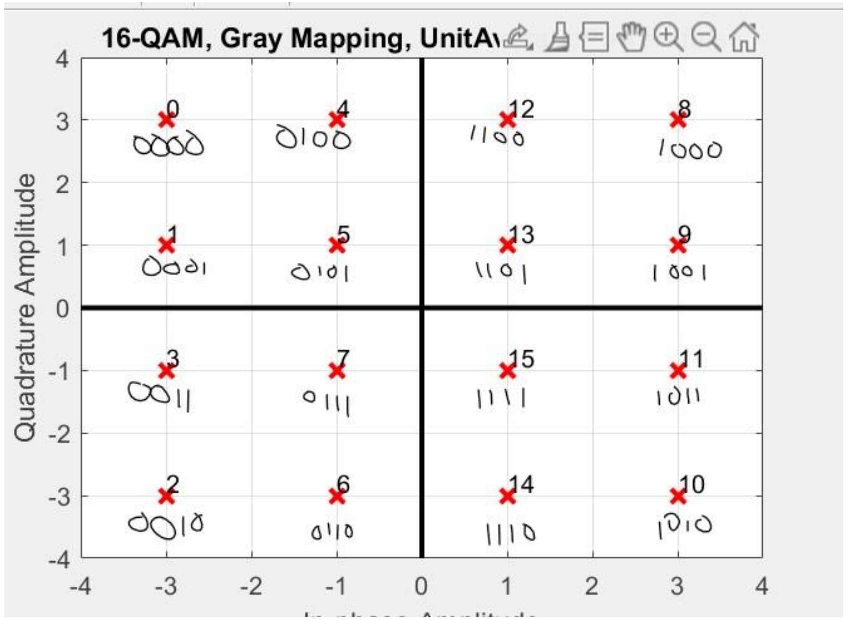
Application

- IEEE 802.11 (WI-FI)
- Mobile Internet (LTE, etc)
- DVB (digital TV)

SR

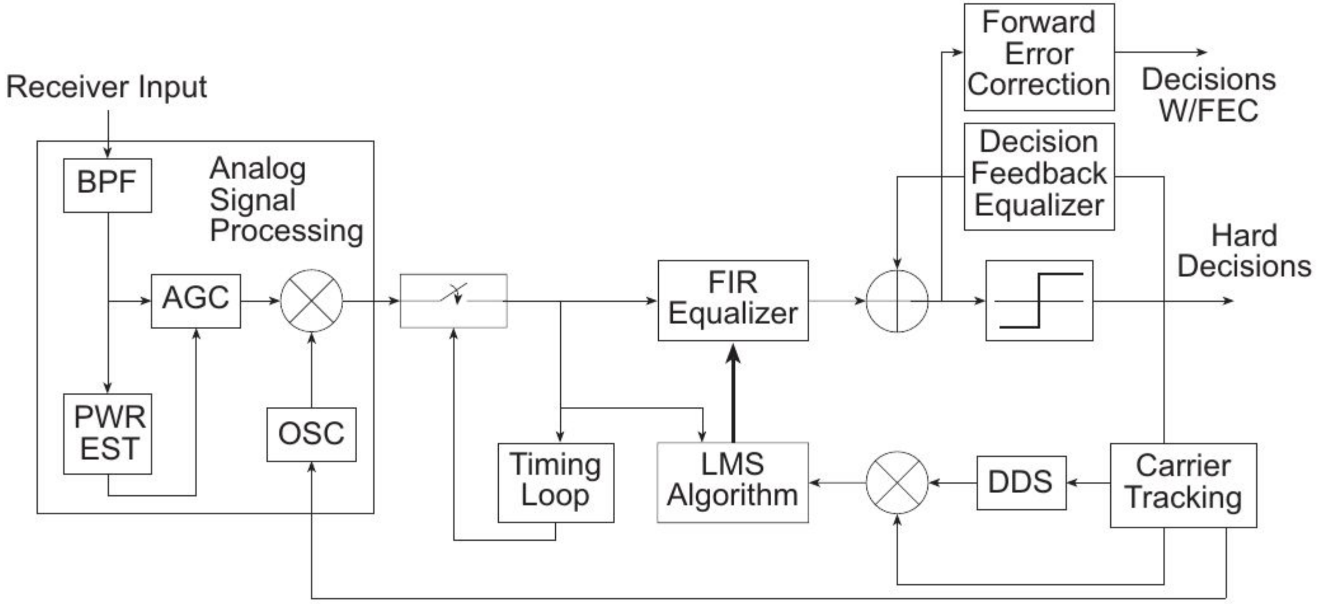
What is QAM?

- Quadrature Amplitude Modulation.
- Uses the amplitude of two orthogonal sinusoids to map to a location on a “constellation diagram”
- Each point on the diagram represents one 4-bit number
- Each point corresponds to two amplitudes: In-phase (horizontal axis) and in-quadrature (vertical axis)



SR

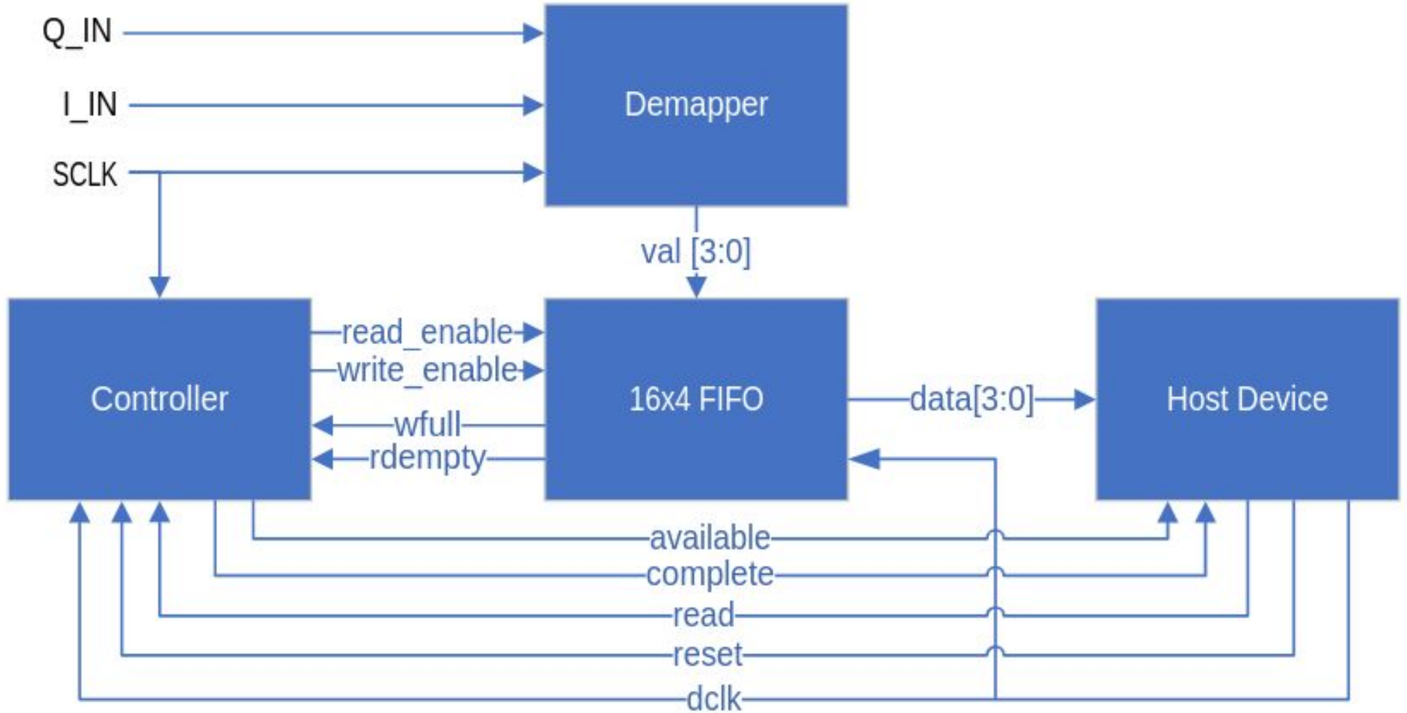
Generic QAM Demodulator



From FPGA QAM Demodulator Design, by Harris and Dick, 2002. Retrieved from https://link.springer.com/chapter/10.1007/3-540-46117-5_13 on 4/27/22

ZM

Hard Decision Demapper Block Diagram



SR

Controller

- Controller operates as a Mealy Machine
- Sequential Logic operates off of the Digital Clock (dclk)
- Four states:
  - Idle - demapper not operating (enable signal is low or reset is high)
  - Write - demapper operating and writing to FIFO
  - Available - signals the host device that a complete data word (64 bits) is available in FIFO
    - Additional symbols received in this state are dropped so as not to overflow FIFO
  - Read - host reading from FIFO
    - Demapper continues to operate in this state

SR

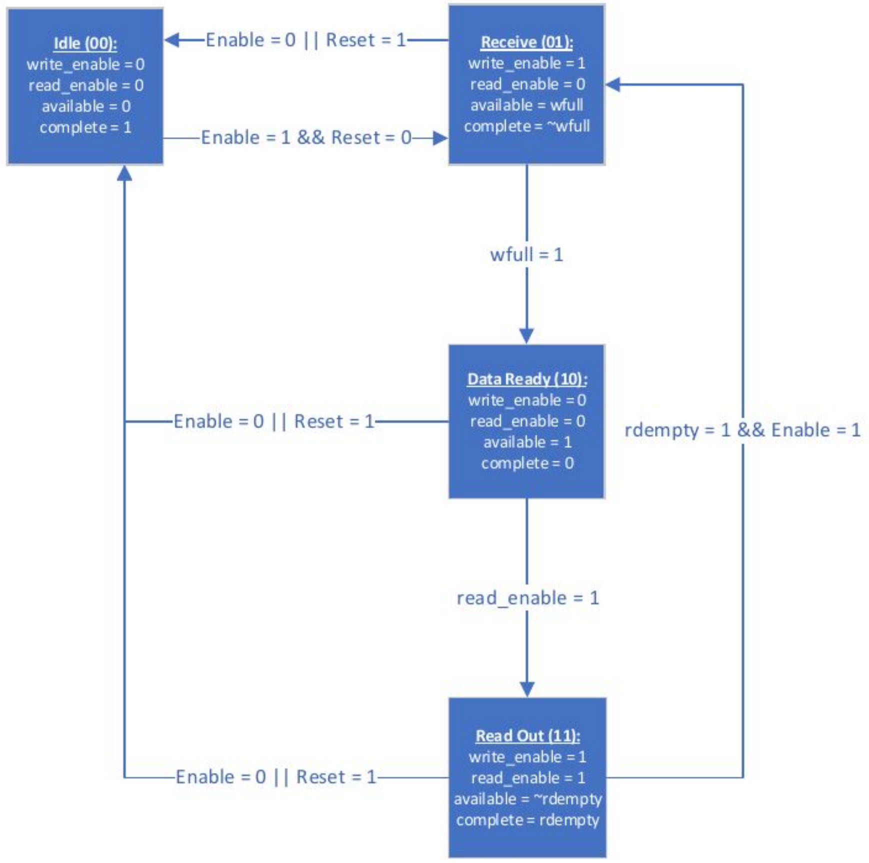
Controller FSM Diagram

Inputs:

- enable = host enable signal
- reset = host reset signal (synchronous)
- sclk = signal (analog) clock
- dclk = host (digital) clock
- read_enable = host read enable signal
- wfull = FIFO full flag
- rdempty = FIFO empty flag

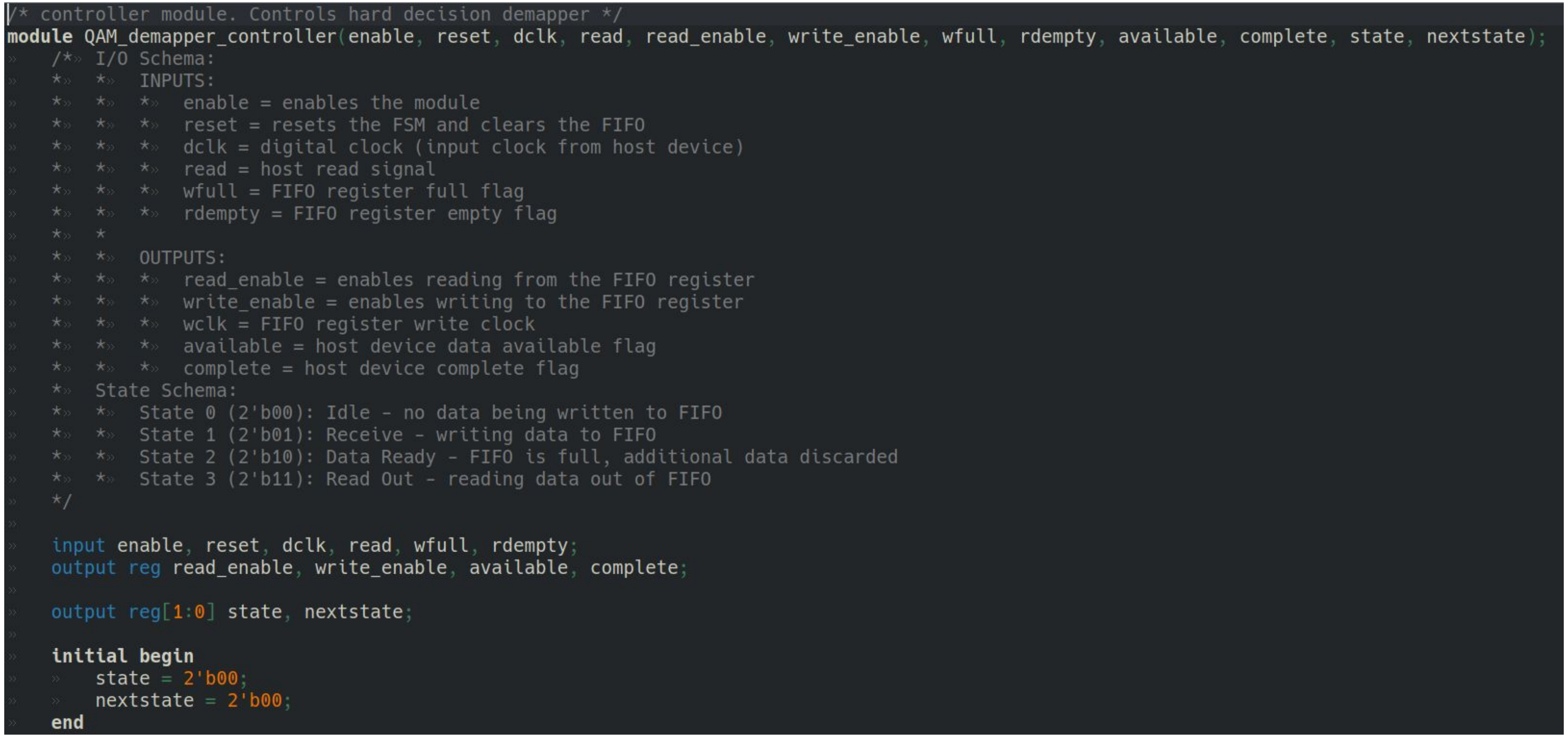
Outputs:

- wclk = FIFO write clock
- rdclk = FIFO read clock
- available = host data available flag
- complete = host read complete flag



SR

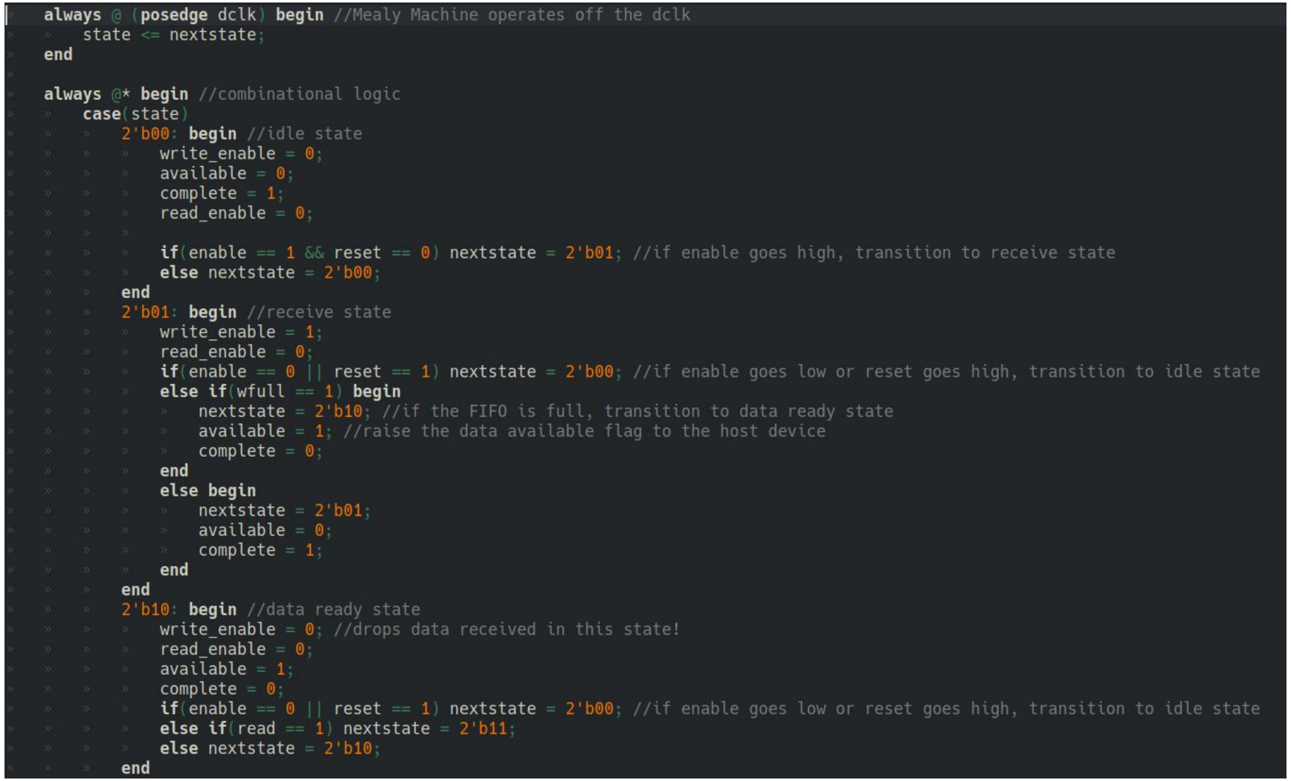
Controller



```verilog
/* controller module. Controls hard decision demapper */
module QAM_demapper_controller(enable, reset, dclk, read, read_enable, write_enable, wfull, rdempty, available, complete, state, nextstate);
	/*	I/O Schema:
	*	*	INPUTS:
	*	*	*	enable = enables the module
	*	*	*	reset = resets the FSM and clears the FIFO
	*	*	*	dclk = digital clock (input clock from host device)
	*	*	*	read = host read signal
	*	*	*	wfull = FIFO register full flag
	*	*	*	rdempty = FIFO register empty flag
	*	*
	*	*	OUTPUTS:
	*	*	*	read_enable = enables reading from the FIFO register
	*	*	*	write_enable = enables writing to the FIFO register
	*	*	*	wclk = FIFO register write clock
	*	*	*	available = host device data available flag
	*	*	*	complete = host device complete flag
	*	State Schema:
	*	*	State 0 (2'b00): Idle - no data being written to FIFO
	*	*	State 1 (2'b01): Receive - writing data to FIFO
	*	*	State 2 (2'b10): Data Ready - FIFO is full, additional data discarded
	*	*	State 3 (2'b11): Read Out - reading data out of FIFO
	*/

	input enable, reset, dclk, read, wfull, rdempty;
	output reg read_enable, write_enable, available, complete;

	output reg[1:0] state, nextstate;

	initial begin
		state = 2'b00;
		nextstate = 2'b00;
	end
```

SR

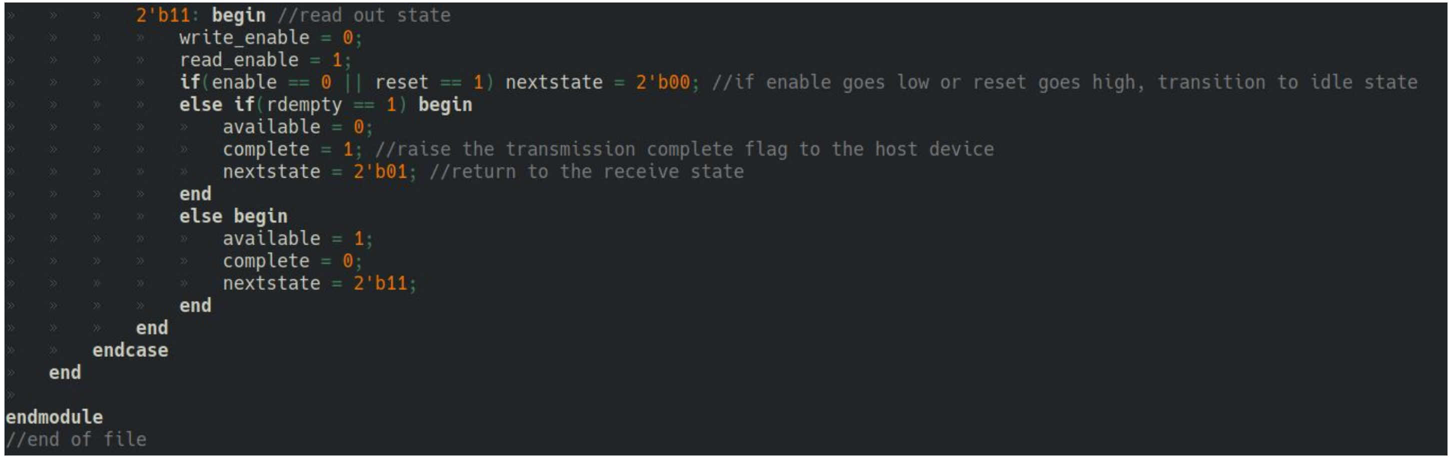
Controller



```verilog
always @ (posedge dclk) begin //Mealy Machine operates off the dclk
    state <= nextstate;
end

always @* begin //combinational logic
    case(state)
        2'b00: begin //idle state
            write_enable = 0;
            available = 0;
            complete = 1;
            read_enable = 0;

            if(enable == 1 && reset == 0) nextstate = 2'b01; //if enable goes high, transition to receive state
            else nextstate = 2'b00;
        end
        2'b01: begin //receive state
            write_enable = 1;
            read_enable = 0;
            if(enable == 0 || reset == 1) nextstate = 2'b00; //if enable goes low or reset goes high, transition to idle state
            else if(wfull == 1) begin
                nextstate = 2'b10; //if the FIFO is full, transition to data ready state
                available = 1; //raise the data available flag to the host device
                complete = 0;
            end
            else begin
                nextstate = 2'b01;
                available = 0;
                complete = 1;
            end
        end
        2'b10: begin //data ready state
            write_enable = 0; //drops data received in this state!
            read_enable = 0;
            available = 1;
            complete = 0;
            if(enable == 0 || reset == 1) nextstate = 2'b00; //if enable goes low or reset goes high, transition to idle state
            else if(read == 1) nextstate = 2'b11;
            else nextstate = 2'b10;
        end
```

SR

Controller



```
            2'b11: begin //read out state
                write_enable = 0;
                read_enable = 1;
                if(enable == 0 || reset == 1) nextstate = 2'b00; //if enable goes low or reset goes high, transition to idle state
                else if(rdempty == 1) begin
                    available = 0;
                    complete = 1; //raise the transmission complete flag to the host device
                    nextstate = 2'b01; //return to the receive state
                end
                else begin
                    available = 1;
                    complete = 0;
                    nextstate = 2'b11;
                end
            end
        endcase
    end

endmodule
//end of file
```

SR

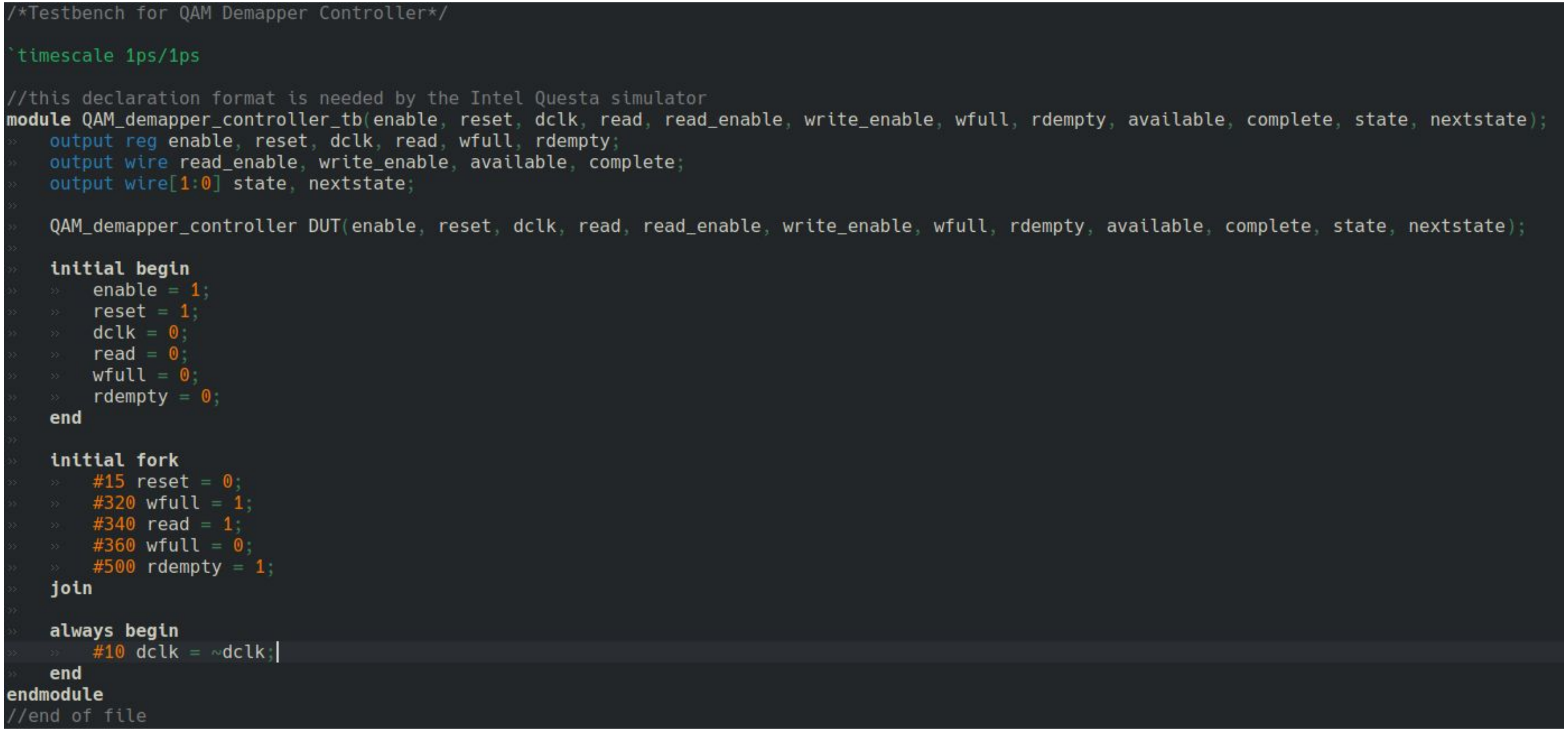
Controller Testbench

Criteria:

- Verify that the controller switches state as expected
- Verify that the controller raises host flags as expected
- Verify that the controller runs the FIFO as expected

SR

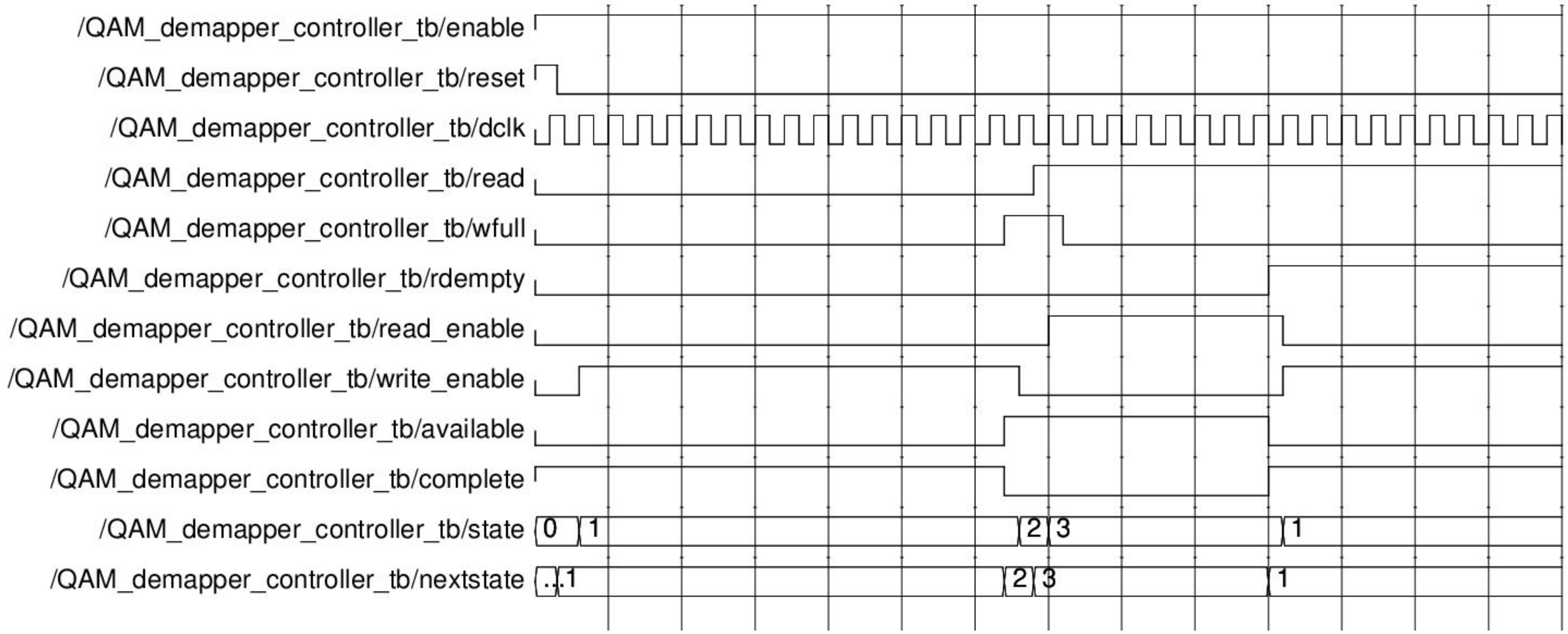
Controller Testbench



```
/*Testbench for QAM Demapper Controller*/

`timescale 1ps/1ps

//this declaration format is needed by the Intel Questa simulator
module QAM_demapper_controller_tb(enable, reset, dclk, read, read_enable, write_enable, wfull, rdempty, available, complete, state, nextstate);
    output reg enable, reset, dclk, read, wfull, rdempty;
    output wire read_enable, write_enable, available, complete;
    output wire[1:0] state, nextstate;

    QAM_demapper_controller DUT(enable, reset, dclk, read, read_enable, write_enable, wfull, rdempty, available, complete, state, nextstate);

    initial begin
        enable = 1;
        reset = 1;
        dclk = 0;
        read = 0;
        wfull = 0;
        rdempty = 0;
    end

    initial fork
        #15 reset = 0;
        #320 wfull = 1;
        #340 read = 1;
        #360 wfull = 0;
        #500 rdempty = 1;
    join

    always begin
        #10 dclk = ~dclk;
    end
endmodule
//end of file
```

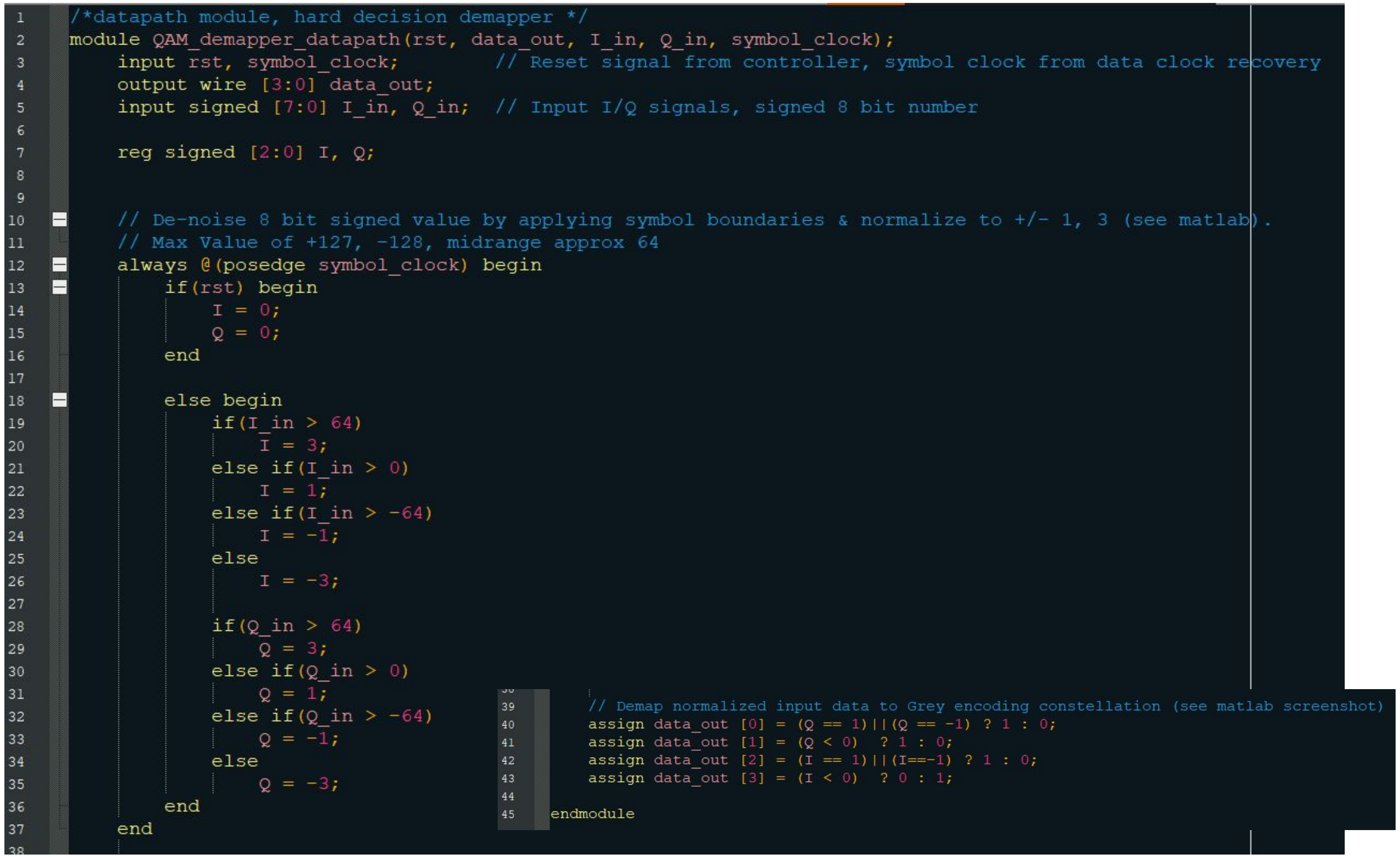
SR

Controller Test Results



SR

Datapath



```
/*datapath module, hard decision demapper */
module QAM_demapper_datapath(rst, data_out, I_in, Q_in, symbol_clock);
    input rst, symbol_clock;        // Reset signal from controller, symbol clock from data clock recovery
    output wire [3:0] data_out;
    input signed [7:0] I_in, Q_in;  // Input I/Q signals, signed 8 bit number

    reg signed [2:0] I, Q;


    // De-noise 8 bit signed value by applying symbol boundaries & normalize to +/- 1, 3 (see matlab).
    // Max Value of +127, -128, midrange approx 64
    always @(posedge symbol_clock) begin
        if(rst) begin
            I = 0;
            Q = 0;
        end

        else begin
            if(I_in > 64)
                I = 3;
            else if(I_in > 0)
                I = 1;
            else if(I_in > -64)
                I = -1;
            else
                I = -3;

            if(Q_in > 64)
                Q = 3;
            else if(Q_in > 0)
                Q = 1;
            else if(Q_in > -64)
                Q = -1;
            else
                Q = -3;
        end
    end
```

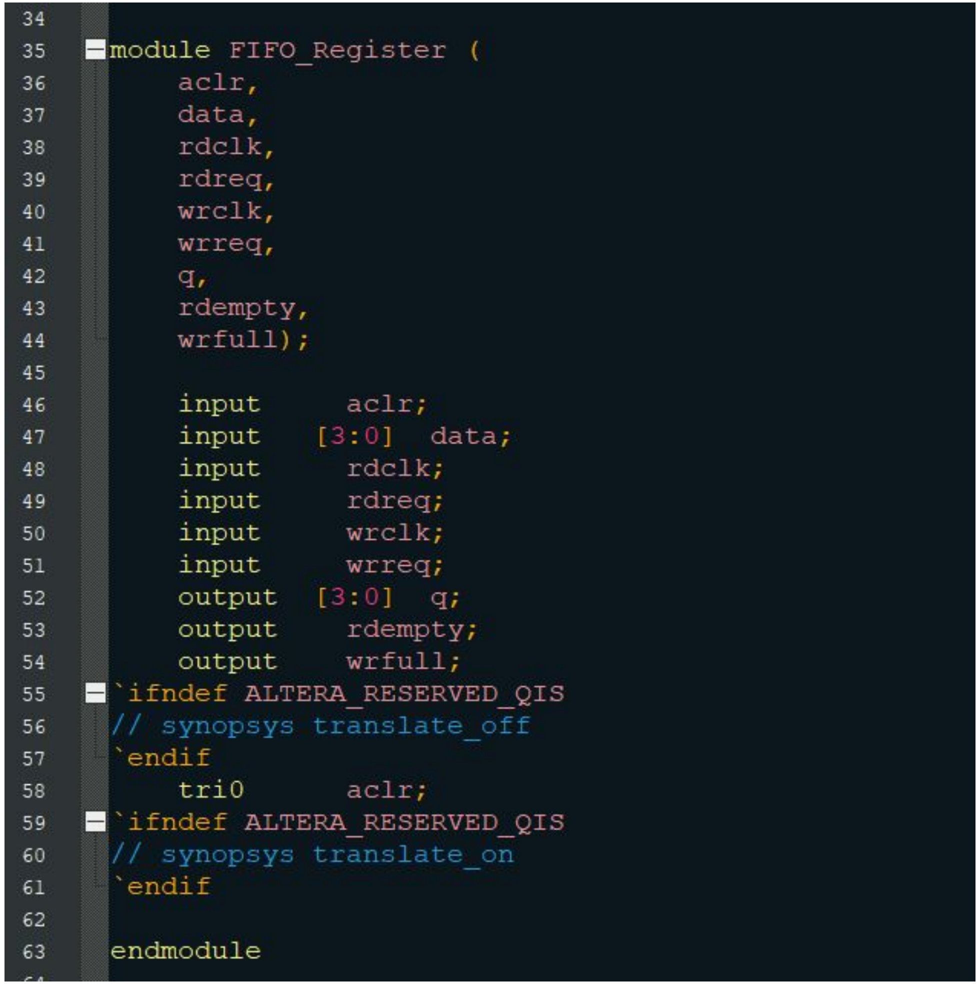
```
    // Demap normalized input data to Grey encoding constellation (see matlab screenshot)
    assign data_out [0] = (Q == 1)||(Q == -1) ? 1 : 0;
    assign data_out [1] = (Q < 0)  ? 1 : 0;
    assign data_out [2] = (I == 1)||(I==-1) ? 1 : 0;
    assign data_out [3] = (I < 0)  ? 0 : 1;

endmodule
```

ZM

Datapath - FIFO

The FIFO module itself is provided by Altera/Intel as an IP module in Quartus, and utilizes the block RAM available on the Cyclone V

The FIFO is set up as 4 bits wide, 16 bits deep, which gives a data word length of 64 bits for the host



```
module FIFO_Register (
    aclr,
    data,
    rdclk,
    rdreq,
    wrclk,
    wrreq,
    q,
    rdempty,
    wrfull);

    input     aclr;
    input   [3:0]  data;
    input     rdclk;
    input     rdreq;
    input     wrclk;
    input     wrreq;
    output  [3:0]  q;
    output    rdempty;
    output    wrfull;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri0      aclr;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

endmodule
```

SR

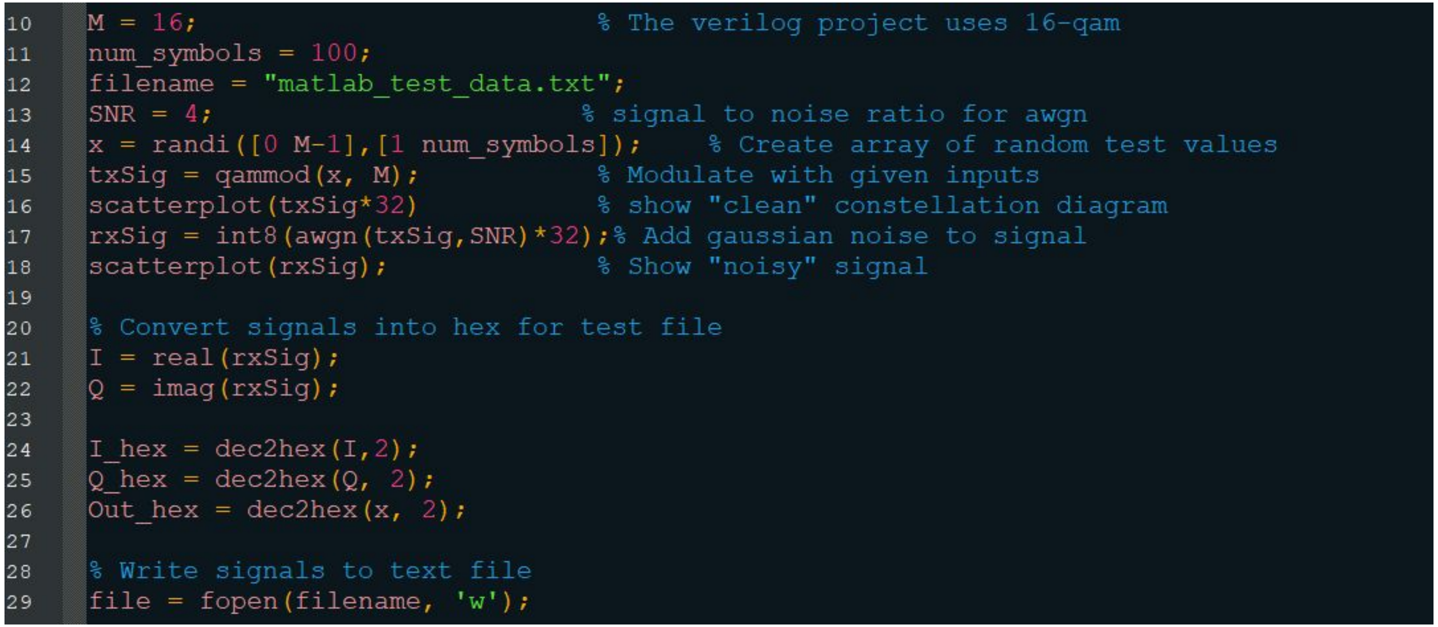
Testing Demapper Datapath

Criteria:

- Test basic symbol decoding
- Test noisy symbol decoding
- Show each error automatically
- Generate noisy symbol automatically

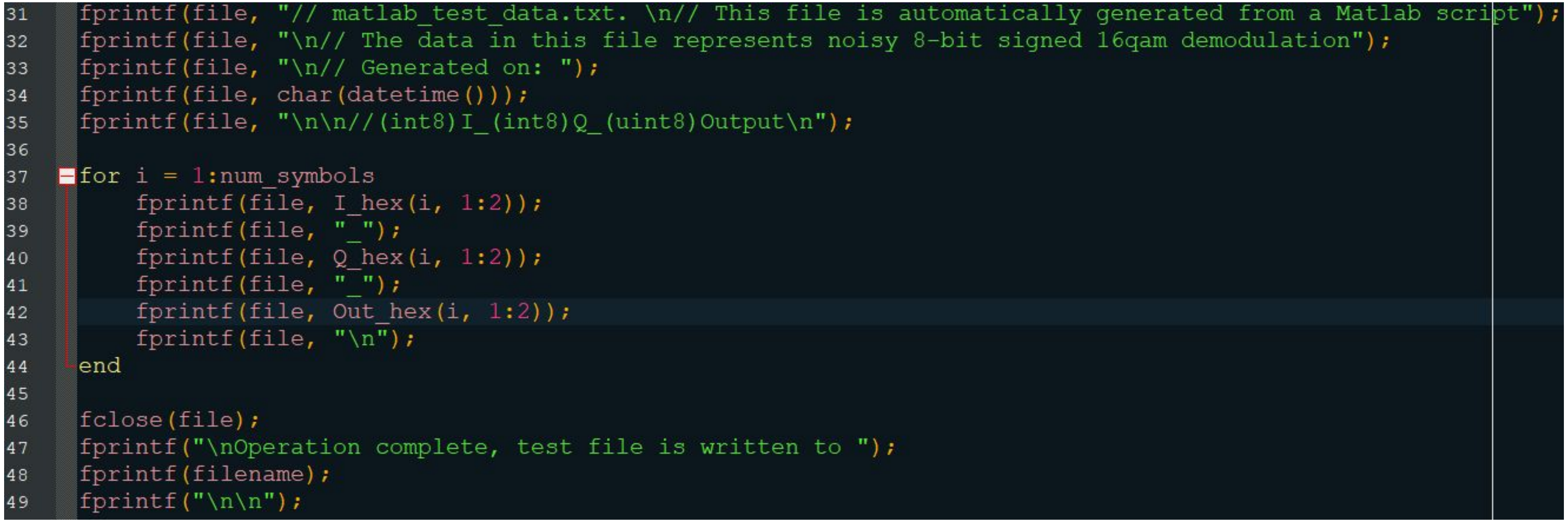
ZM

Matlab Test Bench Script (1/2)


```
M = 16;                             % The verilog project uses 16-qam
num_symbols = 100;
filename = "matlab_test_data.txt";
SNR = 4;                           % signal to noise ratio for awgn
x = randi([0 M-1],[1 num_symbols]);     % Create array of random test values
txSig = qammod(x, M);               % Modulate with given inputs
scatterplot(txSig*32)               % show "clean" constellation diagram
rxSig = int8(awgn(txSig,SNR)*32);% Add gaussian noise to signal
scatterplot(rxSig);                 % Show "noisy" signal

% Convert signals into hex for test file
I = real(rxSig);
Q = imag(rxSig);

I_hex = dec2hex(I,2);
Q_hex = dec2hex(Q, 2);
Out_hex = dec2hex(x, 2);

% Write signals to text file
file = fopen(filename, 'w');
```

ZM

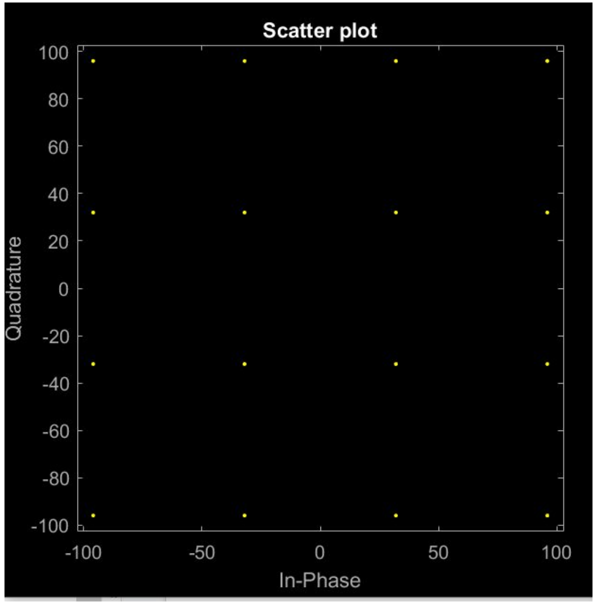
Matlab Test Bench Script (2/2)

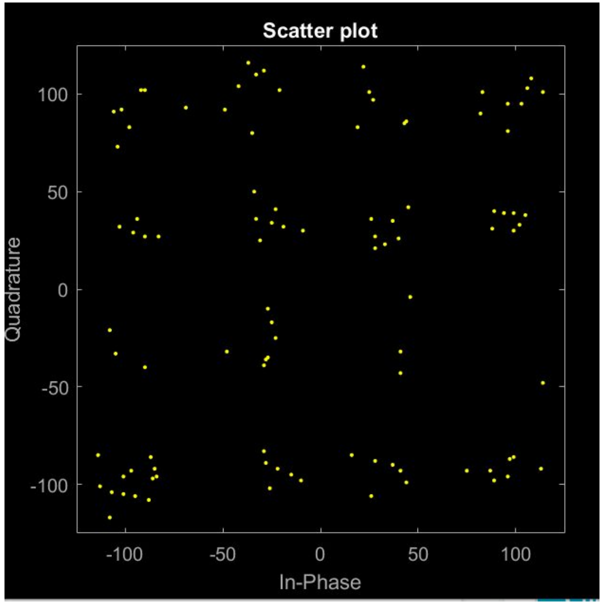


```
fprintf(file, "// matlab_test_data.txt. \n// This file is automatically generated from a Matlab script");
fprintf(file, "\n// The data in this file represents noisy 8-bit signed 16qam demodulation");
fprintf(file, "\n// Generated on: ");
fprintf(file, char(datetime()));
fprintf(file, "\n\n//(int8)I_(int8)Q_(uint8)Output\n");

for i = 1:num_symbols
    fprintf(file, I_hex(i, 1:2));
    fprintf(file, "_");
    fprintf(file, Q_hex(i, 1:2));
    fprintf(file, "_");
    fprintf(file, Out_hex(i, 1:2));
    fprintf(file, "\n");
end

fclose(file);
fprintf("\nOperation complete, test file is written to ");
fprintf(filename);
fprintf("\n\n");
```

ZM

Matlab Plots

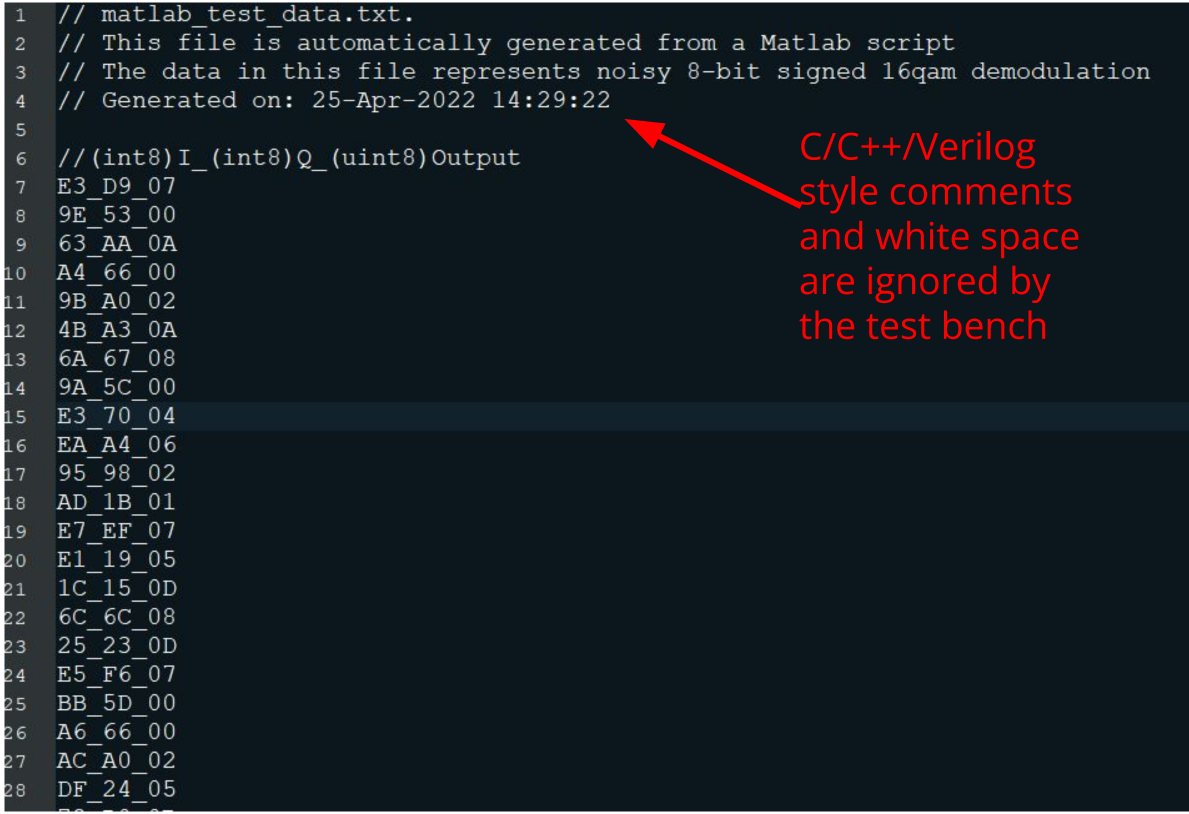




100 “clean” symbols (left) passed through additive gaussian white noise channel produce the image on the right.

Test Bench Data

Noisy I/Q signals stored as a signed 8-bit hex integer for test bench simulation, along with the input symbol to validate decoding

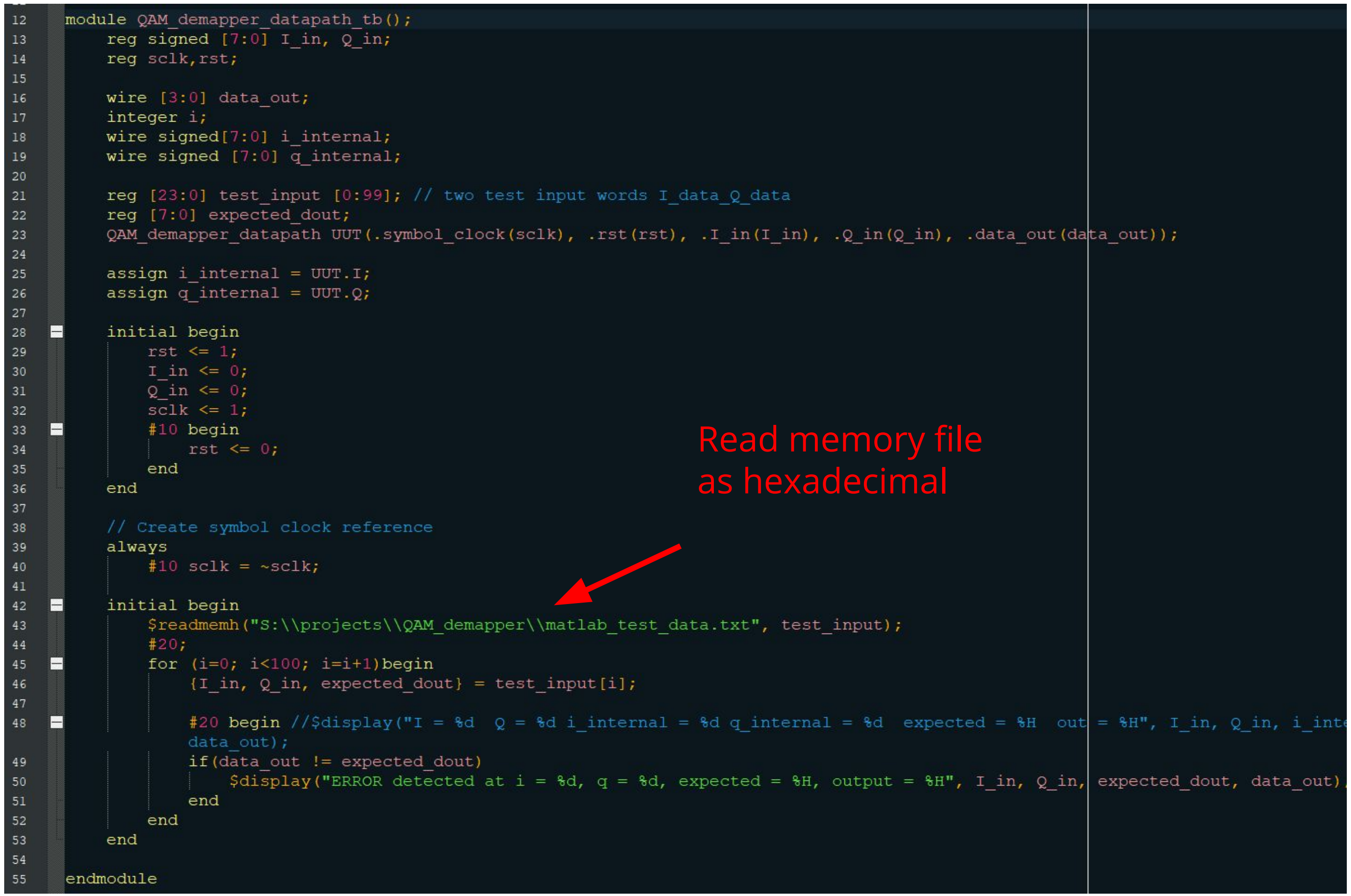


```
// matlab_test_data.txt.
// This file is automatically generated from a Matlab script
// The data in this file represents noisy 8-bit signed 16qam demodulation
// Generated on: 25-Apr-2022 14:29:22

//(int8)I_(int8)Q_(uint8)Output
E3_D9_07
9E_53_00
63_AA_0A
A4_66_00
9B_A0_02
4B_A3_0A
6A_67_08
9A_5C_00
E3_70_04
EA_A4_06
95_98_02
AD_1B_01
E7_EF_07
E1_19_05
1C_15_0D
6C_6C_08
25_23_0D
E5_F6_07
BB_5D_00
A6_66_00
AC_A0_02
DF_24_05
```

C/C++/Verilog style comments and white space are ignored by the test bench

ZM

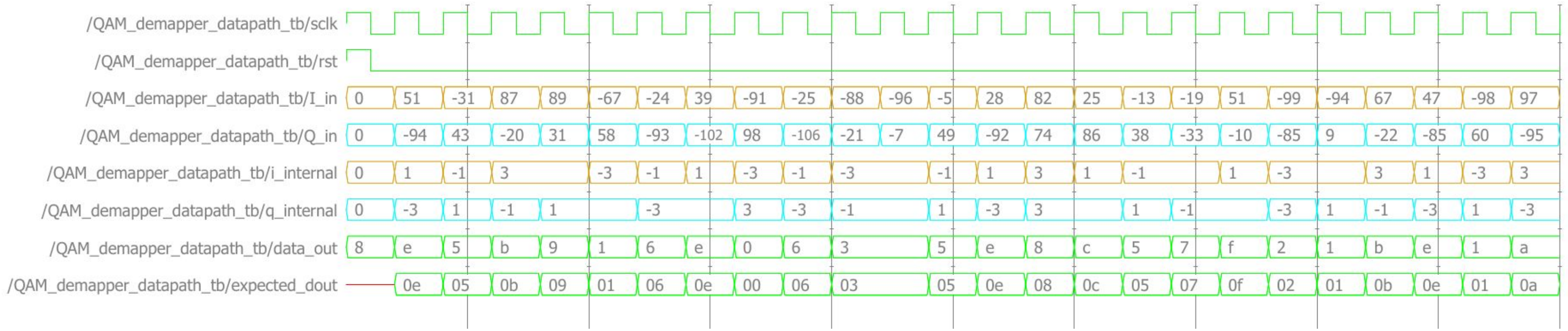
Datapath Test bench



```verilog
module QAM_demapper_datapath_tb();
    reg signed [7:0] I_in, Q_in;
    reg sclk,rst;

    wire [3:0] data_out;
    integer i;
    wire signed[7:0] i_internal;
    wire signed [7:0] q_internal;

    reg [23:0] test_input [0:99]; // two test input words I_data_Q_data
    reg [7:0] expected_dout;
    QAM_demapper_datapath UUT(.symbol_clock(sclk), .rst(rst), .I_in(I_in), .Q_in(Q_in), .data_out(data_out));

    assign i_internal = UUT.I;
    assign q_internal = UUT.Q;

    initial begin
        rst <= 1;
        I_in <= 0;
        Q_in <= 0;
        sclk <= 1;
        #10 begin
            rst <= 0;
        end
    end

    // Create symbol clock reference
    always
        #10 sclk = ~sclk;

    initial begin
        $readmemh("S:\\projects\\QAM_demapper\\matlab_test_data.txt", test_input);
        #20;
        for (i=0; i<100; i=i+1)begin
            {I_in, Q_in, expected_dout} = test_input[i];

            #20 begin //$display("I = %d  Q = %d i_internal = %d q_internal = %d  expected = %H  out = %H", I_in, Q_in, i_inte
            data_out);
            if(data_out != expected_dout)
                $display("ERROR detected at i = %d, q = %d, expected = %H, output = %H", I_in, Q_in, expected_dout, data_out)
            end
        end
    end

endmodule
```

Read memory file
as hexadecimal

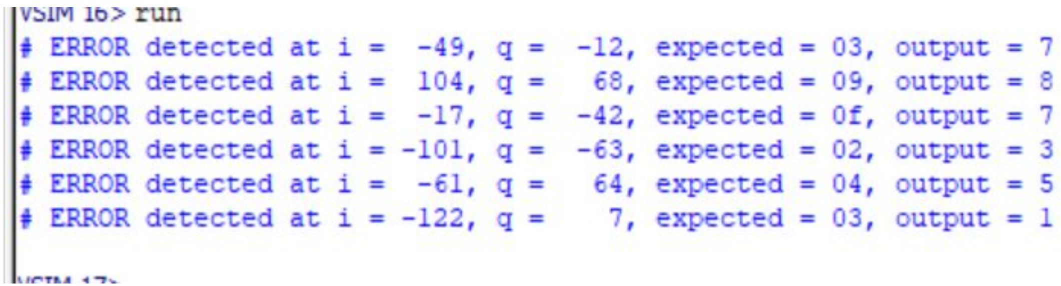
Datapath Test Bench Timing Diagram



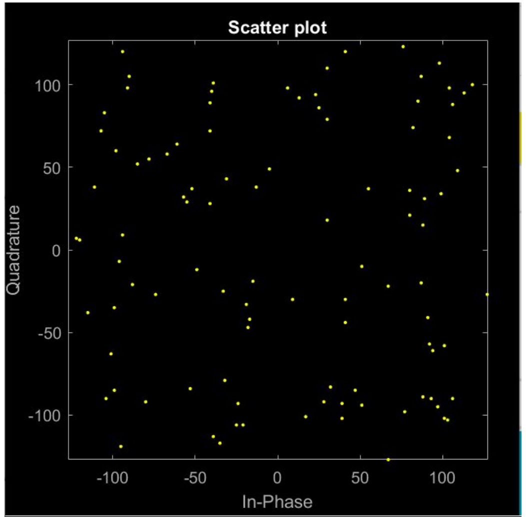
ZM

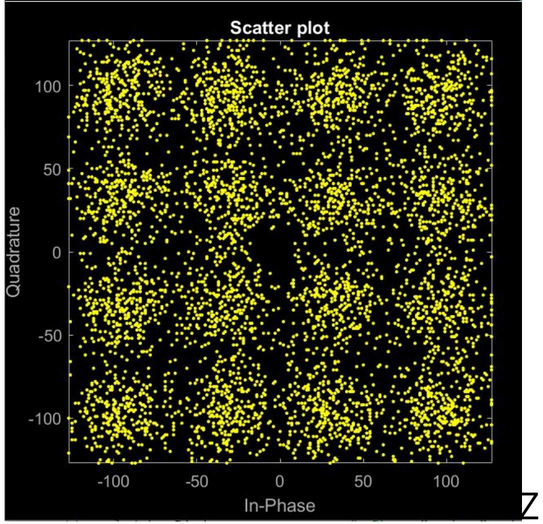
Symbol Errors

- awgn() function allows for varying signal to noise ratio (SNR)
- Example: Test with 3dB SNR.
- Constellation of 100 symbols (top), reference with 5,000 symbols (bottom)
- Console report shows 6 symbol errors out of 100 symbols, for an approximate 6% error rate.



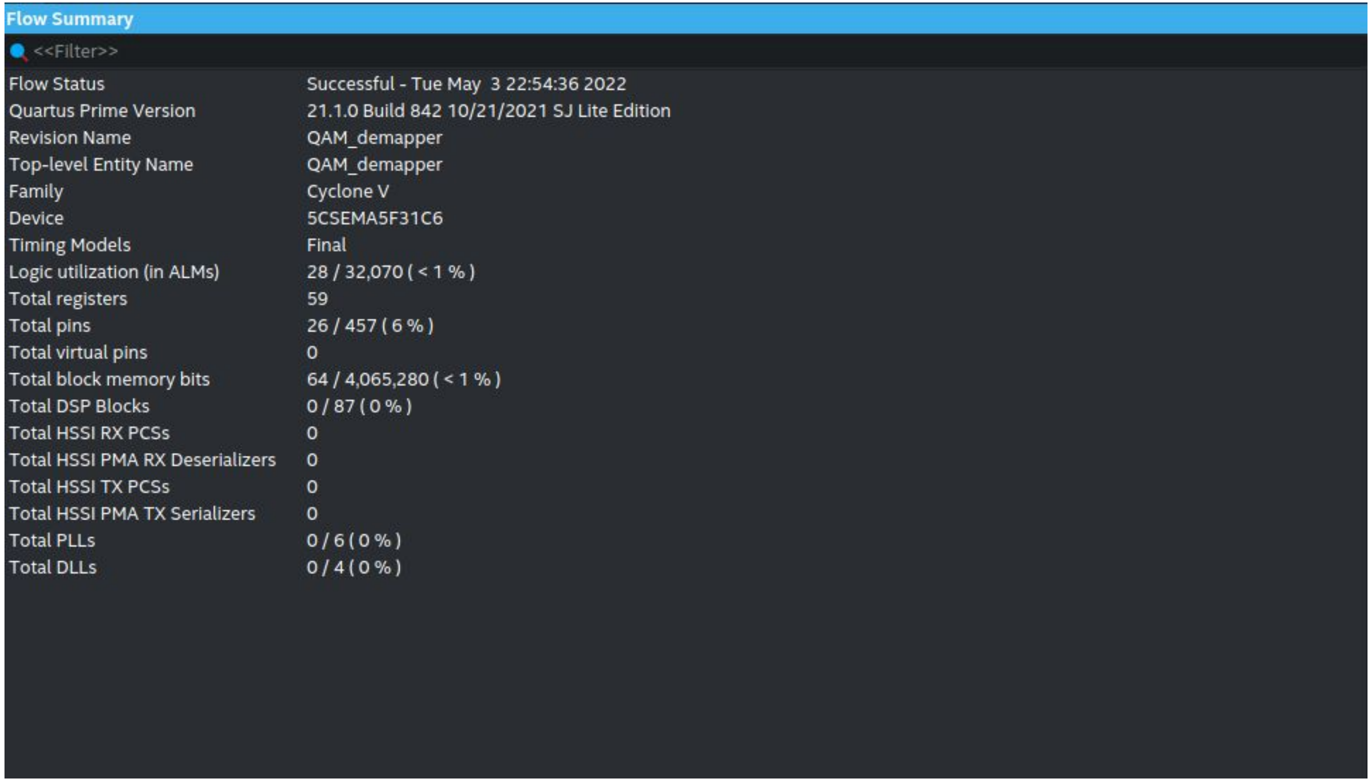
```
VSIM 16> run
# ERROR detected at i =  -49, q =  -12, expected = 03, output = 7
# ERROR detected at i =  104, q =   68, expected = 09, output = 8
# ERROR detected at i =  -17, q =  -42, expected = 0f, output = 7
# ERROR detected at i = -101, q =  -63, expected = 02, output = 3
# ERROR detected at i =  -61, q =   64, expected = 04, output = 5
# ERROR detected at i = -122, q =    7, expected = 03, output = 1

VSIM 17>
```





ZM

Compilation Results



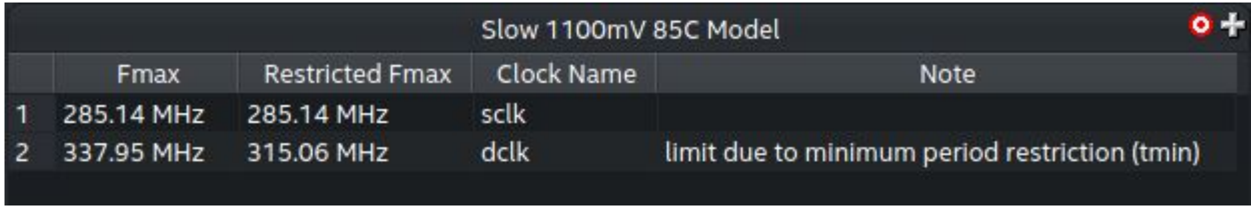
Flow Summary

<<Filter>>

Flow Status	Successful - Tue May 3 22:54:36 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	QAM_demapper
Top-level Entity Name	QAM_demapper
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	28 / 32,070 (< 1 %)
Total registers	59
Total pins	26 / 457 (6 %)
Total virtual pins	0
Total block memory bits	64 / 4,065,280 (< 1 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

SR

Timing Analysis



Slow 1100mV 85C Model

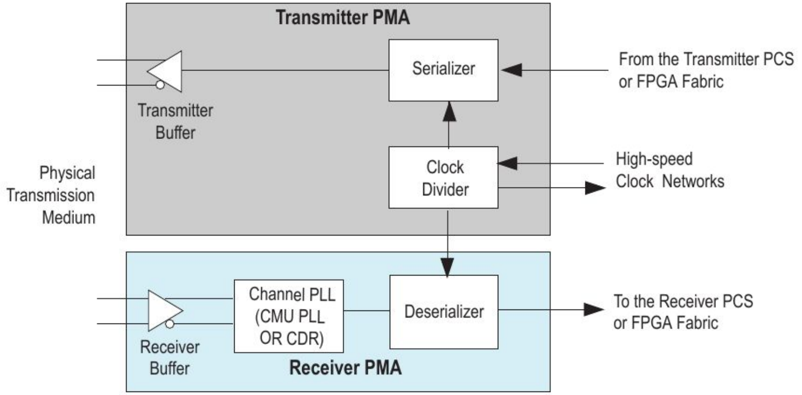
	Fmax	Restricted Fmax	Clock Name	Note
1	285.14 MHz	285.14 MHz	sclk	
2	337.95 MHz	315.06 MHz	dclk	limit due to minimum period restriction (tmin)

- Maximum Symbol Clock = 285MHz (3.509ns)
- Maximum Digital Clock = 315MHz (3.175ns)
- Timing Slack at 200MHz (5ns) Digital and Symbol clock:
    - Setup slack:
        - Dclk = 0.874ns
        - Sclk = 1.495ns
    - Hold slack:
        - Dclk = 0.268ns
        - Sclk = 0.315ns
    - Minimum pulse width:
        - Dclk = 1.366ns
        - Sclk = 1.361ns

SR

Improvements

- Filtering (root raised cosine, etc)
- Error correction
  - Forward Error Correction (simplex)
  - Parity (duplex or half-duplex)
- Interface with transceiver IP blocks built in to the Cyclone V
- Split the digital clock from the FIFO read out clock


Transmitter PMA
Serializer
From the Transmitter PCS or FPGA Fabric
Transmitter Buffer
Clock Divider
High-speed Clock Networks
Physical Transmission Medium
Channel PLL (CMU PLL OR CDR)
Deserializer
To the Receiver PCS or FPGA Fabric
Receiver Buffer
Receiver PMA

Transceiver IP (from the 2012 Cyclone V device Handbook vol 2: transceivers section 1-2)