Ho Chi Minh University of Technology

Faculty of Computer Science and Engineering
Course: Logic Design with HDL (CO1025)

Logic Design Assignment:

Designing and implementing memory in FIFO and PWM in Verilog

Group 7

Lecturer:	Pham Quoc Cuong	
Student:	Phan Minh Toan	- 1852798
	Tran Nguyen Khoi	- 1952797
	Hua Vu Minh Hieu	- 2052990
	Nguyen Quang Khoi	- 2153485

Ho Chi Minh, May 2022

Contents

List of Figures

Listings

1 Introduction

1.1 FIFO

First In, First Out, commonly known as FIFO, is an asset-management and valuation method in which assets produced or acquired first are sold, used, or disposed of first.

1.2 PWM

In Power Electronics, Pulse-Width Modulation (PWM) is the core for control and has proven effective in driving modern semiconductor power devices. Majority of power electronic circuits are controlled by PWM signals of various forms. Pulse Width Modulation is effective and commonly used as control technique to generate analog signals from a digital device like a micro controller. This post will discuss Pulse Width Modulation, various types of modulation techniques, signal generation, its applications, advantages and disadvantages.

2 Background and Applications

2.1 FIFO

In computing and in systems theory, FIFO an acronym for first in, first out (the first in is the first out) is a method for organizing the manipulation of a data structure (often, specifically a data buffer) where the oldest (first) entry, or ”head” of the queue, is processed first.
A FIFO buffer is a read/write storage array that automatically tracks the order in which data enters the module and reads out the data in the same order. In hardware, FIFO buffers are used for synchronization purposes.

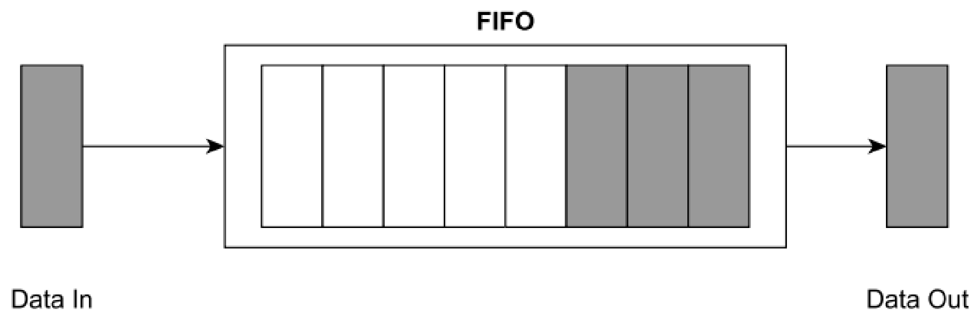

Figure 1: FIFO Demonstation

Application: Inventory is assigned costs as items are prepared for sale. This may occur through the purchase of the inventory or production costs, through the purchase of materials, and utilization of labor. These assigned costs are based on the order in which the product was used, and for FIFO, it is based on what arrived first. For example, if 100 items were purchased for \$10 and 100 more items were purchased next for \$15, FIFO would assign the cost of the first item resold of \$10. After 100 items were sold, the new cost of the item would become \$15, regardless of any additional inventory purchases made.

2.2 PWM

Pulse Width Modulation (PWM) controls analog circuits with a microprocessor's digital outputs. In this technique, Digital-to-Analog conversion is not necessary as the noise effects are minimized by keeping the signal digital. In PWM technique the energy is distributed through a series of pulses rather than a continuously varying (analog) signal. By increasing or decreasing pulse width, the energy flow to the motor shaft can be controlled.
Application:

- PWM Techniques are used in Telecommunications for encoding purposes.
- PWM Techniques are used in Telecommunications for encoding purposes.
- Computer Motherboard requires PWM Signals that controls the heat generated in the board. 4 Pin PWM header is embedded in the fan that helps to dissipate the heat from the motherboard.

- It is also used in Audio/Video Amplifiers.

3 Design

3.1 FIFO

In FIFO, We create 4 specific blocks. They are:

- Read block: For the request to get the oldest data-in out.
- Write block: For the request to push the lasted data-in into the queue.
- Signal block: To control the sequence of signal as well as control the status of memory.
- Memory Block. To store data

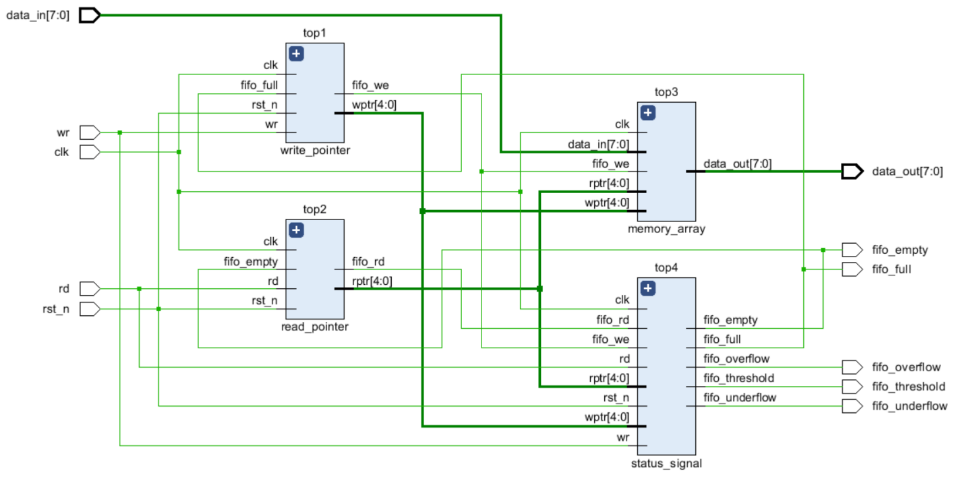

Figure 2: FIFO RTL NETLIST DESIGN

3.2 PWM

IN PWM Design, the inputs will have clock signal, decrease duty signal and increase duty signal. The output will be the Pulse Width.

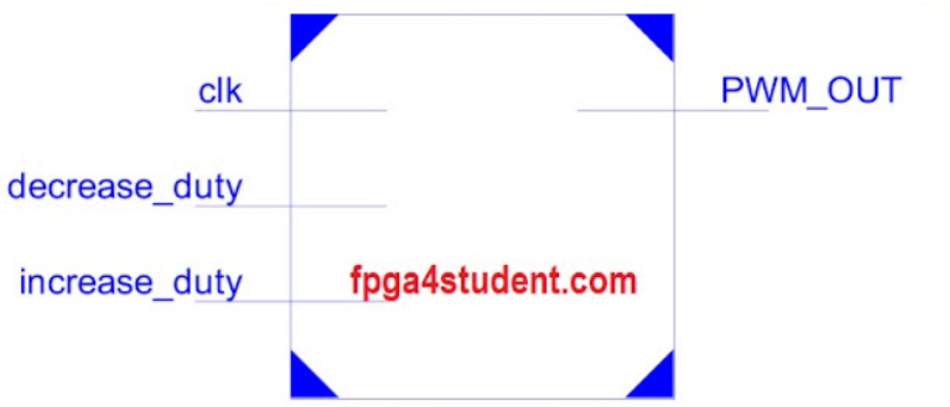

Figure 3: PWM Block

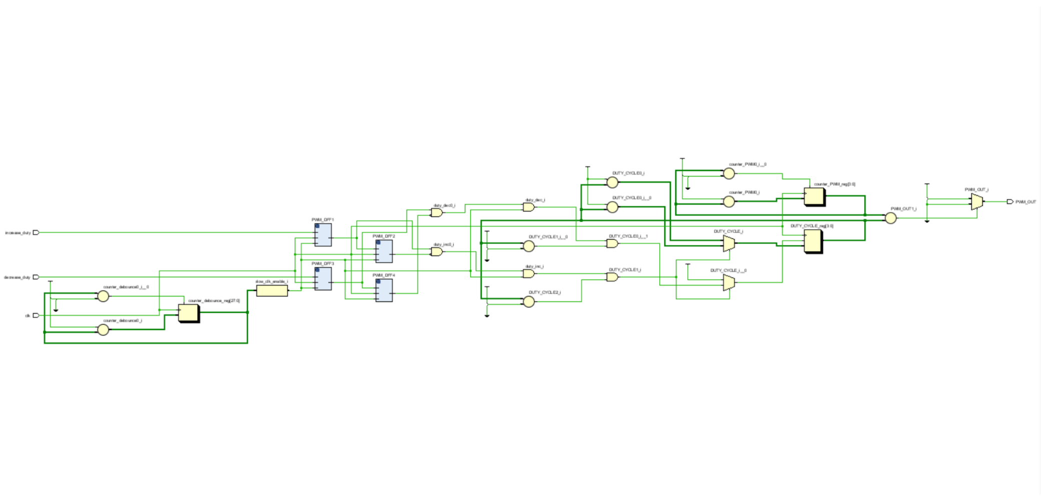

Figure 4: PWM General Netlist 1

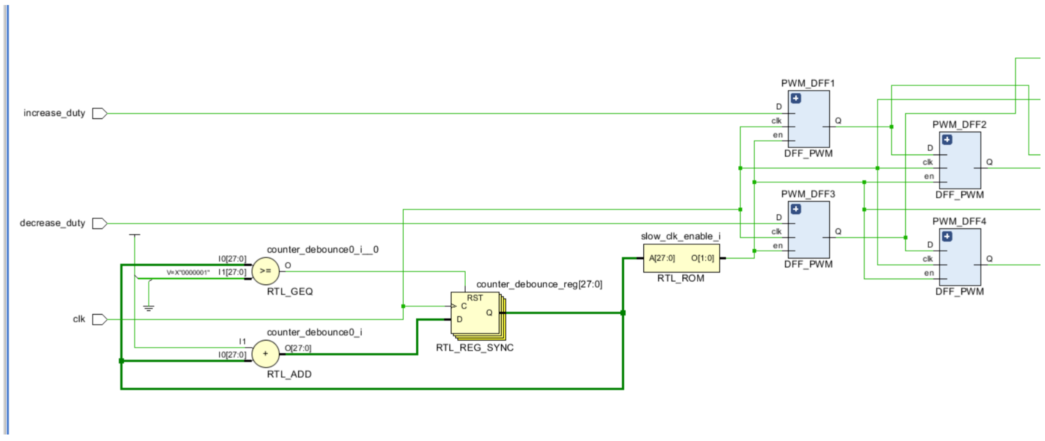

Figure 5: PWM General Netlist 2

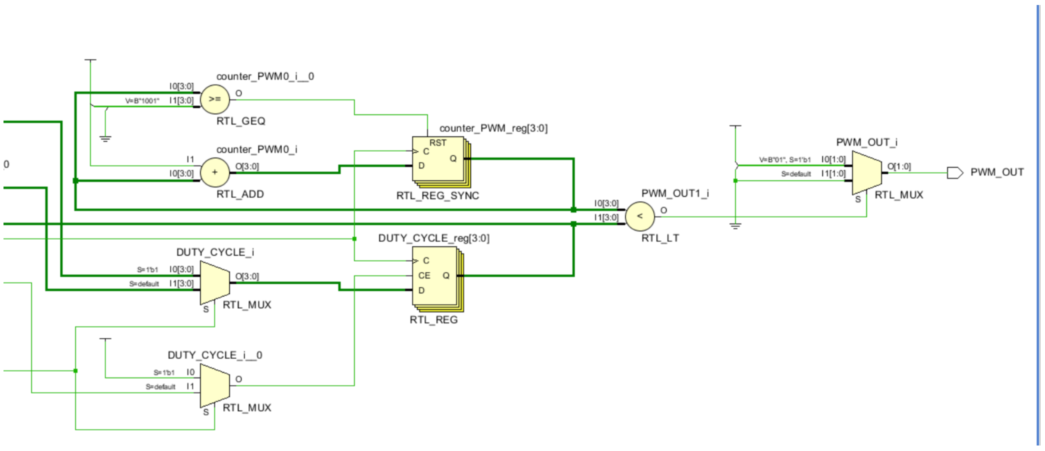

Figure 6: PWM General Netlist 3

4 Implementation and Result

4.1 FIFO

We design our blocks and testbench as follow:

```
module fifo(data_out,fifo_full, fifo_empty, fifo_threshold,
    fifo_overflow, fifo_underflow,clk, rst_n, wr, rd, data_in);
     input wr, rd, clk, rst_n;
     input[7:0] data_in;
     output[7:0] data_out;
     output fifo_full, fifo_empty, fifo_threshold, fifo_overflow,
     fifo_underflow;
```

```verilog
    wire[4:0] wptr,rptr;
    wire fifo_we,fifo_rd;

    write_pointer top1(wptr,fifo_we,wr,fifo_full,clk,rst_n);
    read_pointer top2(rptr,fifo_rd,rd,fifo_empty,clk,rst_n);
    memory_array top3(data_out, data_in, clk,fifo_we, wptr,rptr)
   ;
    status_signal top4(fifo_full, fifo_empty, fifo_threshold,
   fifo_overflow, fifo_underflow, wr, rd, fifo_we, fifo_rd, wptr
   ,rptr,clk,rst_n);
endmodule
```

Listing 1: FIFO Memory

```verilog
module read_pointer(rptr,fifo_rd,rd,fifo_empty,clk,rst_n);
    input rd,fifo_empty,clk,rst_n;
    output[4:0] rptr;
    output fifo_rd;
    reg[4:0] rptr;

    assign fifo_rd = (~fifo_empty)& rd;
    always @(posedge clk or negedge rst_n)
    begin
        if(~rst_n) rptr <= 5'b000000;
        else if(fifo_rd)
            rptr <= rptr + 5'b000001;
        else
            rptr <= rptr;
        end
endmodule
```

Listing 2: Read Block

```verilog
module write_pointer(wptr,fifo_we,wr,fifo_full,clk,rst_n);
    input wr,fifo_full,clk,rst_n;
    output[4:0] wptr;
    output fifo_we;
    reg[4:0] wptr;

    assign fifo_we = (~fifo_full)&wr;
    always @(posedge clk or negedge rst_n)
    begin
        if(~rst_n) wptr <= 5'b000000;
        else if(fifo_we)
            wptr <= wptr + 5'b000001;
        else
            wptr <= wptr;
    end
endmodule
```

Listing 3: Write Block

```
module memory_array(data_out , data_in , clk ,fifo_we , wptr ,rptr);
    input [7:0] data_in;
    input clk ,fifo_we;
    input [4:0] wptr ,rptr;
    output [7:0] data_out;
    reg [7:0] data_out2 [15:0];
    wire [7:0] data_out;
    always @(posedge clk)
    begin
        if(fifo_we)
            data_out2[wptr[3:0]] <=data_in ;
    end
       assign data_out = data_out2[rptr[3:0]];
endmodule
```

Listing 4: FIFO Memory Storage

```

 `timescale      10 ps/ 10 ps

 `define            DELAY 10
 module      tb_fifo_32;

 parameter       ENDTIME      = 40000;

 reg      clk;
 reg      rst_n;
 reg      wr;
 reg      rd;
 reg      [7:0] data_in;

 wire      [7:0] data_out;
 wire      fifo_empty;
 wire      fifo_full;
 wire      fifo_threshold;
 wire      fifo_overflow;
 wire      fifo_underflow;
 integer i;
 fifo tb (
   data_out , fifo_full , fifo_empty , fifo_threshold ,
    fifo_overflow ,
   fifo_underflow ,

   clk , rst_n , wr, rd , data_in
   );
```

```

initial
	begin
		clk	= 1'b0;
		rst_n	= 1'b0;
		wr	= 1'b0;
		rd	= 1'b0;
		data_in	= 8'd0;
	end
initial
	begin
		main;
	end
task main;
	fork
		clock_generator;
		reset_generator;
		operation_process;
		debug_fifo;
		endsimulation;
	join
endtask
task clock_generator;
	begin
		forever #DELAY clk = !clk;
	end
endtask
task reset_generator;
	begin
		#(`DELAY*2)
		rst_n = 1'b1;
		# 7.9
		rst_n = 1'b0;
		# 7.09
		rst_n = 1'b1;
	end
endtask
task operation_process;
	begin
		for (i = 0; i < 17; i = i + 1) begin: WRE
			#(`DELAY*5)
			wr = 1'b1;
			data_in = data_in + 8'd1;
			#(`DELAY*2)
			wr = 1'b0;
		end
		#(`DELAY)
		for (i = 0; i < 17; i = i + 1) begin: RDE
			#(`DELAY*2)
```

```
                rd = 1'b1;
                #(`DELAY*2)
                rd = 1'b0;
            end
        end
endtask
// 10. Debug fifo
task debug_fifo;
        begin
            $display("
  ----------------------------------------------------------------");
            $display("------------------------
  --------------------------------");
            $display("--------------- SIMULATION RESULT
  -----------------------");
            $display("-------------------          -------------------------");
            $display("----------------------       -----------------------------"
  );
            $display("
  ----------------------------------------------------------------");
            $monitor("TIME = %d, wr = %b, rd = %b, data_in = %h",
  $time, wr, rd, data_in);
        end
endtask
reg [5:0] waddr, raddr;
reg [7:0] mem[64:0];
always @ (posedge clk) begin
        if (~rst_n) begin
            waddr      <= 6'd0;
        end
        else if (wr) begin
            mem[waddr] <= data_in;
            waddr <= waddr + 1;
        end
        $display("TIME = %d, data_out = %d, mem = %d",$time,
  data_out,mem[raddr]);
        if (~rst_n) raddr      <= 6'd0;
        else if (rd & (~fifo_empty)) raddr <= raddr + 1;
        if (rd & (~fifo_empty)) begin
            if (mem[raddr]
             == data_out) begin
                $display("=== PASS ===== PASS ===== PASS =====
  PASS ===");
                if (raddr == 16) $finish;
            end
            else begin
                $display ("=== FAIL ===== FAIL ===== FAIL =====
  FAIL ===");
                $display("------------------ THE SIMUALTION FINISHED
    ————————————————");
                    $finish;
                end
            end
        end
task endsimulation;
        begin
            #ENDTIME
            $display("————————————— THE SIMUALTION FINISHED
    ——————————————");
            $finish;
        end
endtask
endmodule
```

Listing 5: FIFO Memory Testbench

The result is demonstrated as follow, in Log file and Waveform.

```
Vivado Simulator 2018.2
Time resolution is 1 ps
————————————————————————————————————————————————
———————————————————     ————————————————————————
——————————— SIMULATION RESULT ——————————————————
———————————————         ——————————————————————
—————————————————       ————————————————————————
————————————————————————————————————————————————
TIME =                      0, wr = 0, rd = 0, data_in = 00
TIME =                     10, data_out =  x, mem =  x
TIME =                     30, data_out =  x, mem =  x
TIME =                     50, data_out =  x, mem =  x
TIME =                     50, wr = 1, rd = 0, data_in = 01
TIME =                     70, data_out =  1, mem =  1
TIME =                     70, wr = 0, rd = 0, data_in = 01
TIME =                     90, data_out =  1, mem =  1
TIME =                    110, data_out =  1, mem =  1
TIME =                    120, wr = 1, rd = 0, data_in = 02
TIME =                    130, data_out =  1, mem =  1
TIME =                    140, wr = 0, rd = 0, data_in = 02
TIME =                    150, data_out =  1, mem =  1
TIME =                    170, data_out =  1, mem =  1
TIME =                    190, data_out =  1, mem =  1
TIME =                    190, wr = 1, rd = 0, data_in = 03
TIME =                    210, data_out =  1, mem =  1
TIME =                    210, wr = 0, rd = 0, data_in = 03
TIME =                    230, data_out =  1, mem =  1
TIME =                    250, data_out =  1, mem =  1
TIME =                    260, wr = 1, rd = 0, data_in = 04
TIME =                    270, data_out =  1, mem =  1
TIME =                    280, wr = 0, rd = 0, data_in = 04
TIME =                    290, data_out =    1, mem =    1
TIME =                    310, data_out =    1, mem =    1
TIME =                    330, data_out =    1, mem =    1
TIME =                    330, wr = 1, rd = 0, data_in = 05
TIME =                    350, data_out =    1, mem =    1
TIME =                    350, wr = 0, rd = 0, data_in = 05
TIME =                    370, data_out =    1, mem =    1
TIME =                    390, data_out =    1, mem =    1
TIME =                    400, wr = 1, rd = 0, data_in = 06
TIME =                    410, data_out =    1, mem =    1
TIME =                    420, wr = 0, rd = 0, data_in = 06
TIME =                    430, data_out =    1, mem =    1
TIME =                    450, data_out =    1, mem =    1
TIME =                    470, data_out =    1, mem =    1
TIME =                    470, wr = 1, rd = 0, data_in = 07
TIME =                    490, data_out =    1, mem =    1
TIME =                    490, wr = 0, rd = 0, data_in = 07
TIME =                    510, data_out =    1, mem =    1
TIME =                    530, data_out =    1, mem =    1
TIME =                    540, wr = 1, rd = 0, data_in = 08
TIME =                    550, data_out =    1, mem =    1
TIME =                    560, wr = 0, rd = 0, data_in = 08
TIME =                    570, data_out =    1, mem =    1
TIME =                    590, data_out =    1, mem =    1
TIME =                    610, data_out =    1, mem =    1
TIME =                    610, wr = 1, rd = 0, data_in = 09
TIME =                    630, data_out =    1, mem =    1
TIME =                    630, wr = 0, rd = 0, data_in = 09
TIME =                    650, data_out =    1, mem =    1
TIME =                    670, data_out =    1, mem =    1
TIME =                    680, wr = 1, rd = 0, data_in = 0a
TIME =                    690, data_out =    1, mem =    1
TIME =                    700, wr = 0, rd = 0, data_in = 0a
TIME =                    710, data_out =    1, mem =    1
TIME =                    730, data_out =    1, mem =    1
TIME =                    750, data_out =    1, mem =    1
TIME =                    750, wr = 1, rd = 0, data_in = 0b
TIME =                    770, data_out =    1, mem =    1
TIME =                    770, wr = 0, rd = 0, data_in = 0b
TIME =                    790, data_out =    1, mem =    1
TIME =                    810, data_out =    1, mem =    1
TIME =                    820, wr = 1, rd = 0, data_in = 0c
TIME =                    830, data_out =    1, mem =    1
TIME =                    840, wr = 0, rd = 0, data_in = 0c
TIME =                    850, data_out =    1, mem =    1
TIME =                    870, data_out =    1, mem =    1
TIME =                    890, data_out =    1, mem =    1
TIME =                    890, wr = 1, rd = 0, data_in = 0d
TIME =                    910, data_out =    1, mem =    1
TIME =                     910, wr = 0, rd = 0, data_in = 0d
TIME =                     930, data_out =   1, mem =   1
TIME =                     950, data_out =   1, mem =   1
TIME =                     960, wr = 1, rd = 0, data_in = 0e
TIME =                     970, data_out =   1, mem =   1
TIME =                     980, wr = 0, rd = 0, data_in = 0e
TIME =                     990, data_out =   1, mem =   1
TIME =                    1010, data_out =   1, mem =   1
TIME =                    1030, data_out =   1, mem =   1
TIME =                    1030, wr = 1, rd = 0, data_in = 0f
TIME =                    1050, data_out =   1, mem =   1
TIME =                    1050, wr = 0, rd = 0, data_in = 0f
TIME =                    1070, data_out =   1, mem =   1
TIME =                    1090, data_out =   1, mem =   1
TIME =                    1100, wr = 1, rd = 0, data_in = 10
TIME =                    1110, data_out =   1, mem =   1
TIME =                    1120, wr = 0, rd = 0, data_in = 10
TIME =                    1130, data_out =   1, mem =   1
TIME =                    1150, data_out =   1, mem =   1
TIME =                    1170, data_out =   1, mem =   1
TIME =                    1170, wr = 1, rd = 0, data_in = 11
TIME =                    1190, data_out =   1, mem =   1
TIME =                    1190, wr = 0, rd = 0, data_in = 11
TIME =                    1210, data_out =   1, mem =   1
TIME =                    1220, wr = 0, rd = 1, data_in = 11
TIME =                    1230, data_out =   1, mem =   1
==== PASS ====== PASS ====== PASS ====== PASS ====
TIME =                    1240, wr = 0, rd = 0, data_in = 11
TIME =                    1250, data_out =   2, mem =   2
TIME =                    1260, wr = 0, rd = 1, data_in = 11
TIME =                    1270, data_out =   2, mem =   2
==== PASS ====== PASS ====== PASS ====== PASS ====
TIME =                    1280, wr = 0, rd = 0, data_in = 11
TIME =                    1290, data_out =   3, mem =   3
TIME =                    1300, wr = 0, rd = 1, data_in = 11
TIME =                    1310, data_out =   3, mem =   3
==== PASS ====== PASS ====== PASS ====== PASS ====
TIME =                    1320, wr = 0, rd = 0, data_in = 11
TIME =                    1330, data_out =   4, mem =   4
TIME =                    1340, wr = 0, rd = 1, data_in = 11
TIME =                    1350, data_out =   4, mem =   4
==== PASS ====== PASS ====== PASS ====== PASS ====
TIME =                    1360, wr = 0, rd = 0, data_in = 11
TIME =                    1370, data_out =   5, mem =   5
TIME =                    1380, wr = 0, rd = 1, data_in = 11
TIME =                    1390, data_out =   5, mem =   5
==== PASS ====== PASS ====== PASS ====== PASS ====
TIME =                    1400, wr = 0, rd = 0, data_in = 11
TIME =                    1410, data_out =   6, mem =   6
TIME =                  1420, wr = 0, rd = 1, data_in = 11
TIME =                  1430, data_out =  6, mem =  6
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1440, wr = 0, rd = 0, data_in = 11
TIME =                  1450, data_out =  7, mem =  7
TIME =                  1460, wr = 0, rd = 1, data_in = 11
TIME =                  1470, data_out =  7, mem =  7
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1480, wr = 0, rd = 0, data_in = 11
TIME =                  1490, data_out =  8, mem =  8
TIME =                  1500, wr = 0, rd = 1, data_in = 11
TIME =                  1510, data_out =  8, mem =  8
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1520, wr = 0, rd = 0, data_in = 11
TIME =                  1530, data_out =  9, mem =  9
TIME =                  1540, wr = 0, rd = 1, data_in = 11
TIME =                  1550, data_out =  9, mem =  9
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1560, wr = 0, rd = 0, data_in = 11
TIME =                  1570, data_out = 10, mem = 10
TIME =                  1580, wr = 0, rd = 1, data_in = 11
TIME =                  1590, data_out = 10, mem = 10
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1600, wr = 0, rd = 0, data_in = 11
TIME =                  1610, data_out = 11, mem = 11
TIME =                  1620, wr = 0, rd = 1, data_in = 11
TIME =                  1630, data_out = 11, mem = 11
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1640, wr = 0, rd = 0, data_in = 11
TIME =                  1650, data_out = 12, mem = 12
TIME =                  1660, wr = 0, rd = 1, data_in = 11
TIME =                  1670, data_out = 12, mem = 12
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1680, wr = 0, rd = 0, data_in = 11
TIME =                  1690, data_out = 13, mem = 13
TIME =                  1700, wr = 0, rd = 1, data_in = 11
TIME =                  1710, data_out = 13, mem = 13
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1720, wr = 0, rd = 0, data_in = 11
TIME =                  1730, data_out = 14, mem = 14
TIME =                  1740, wr = 0, rd = 1, data_in = 11
TIME =                  1750, data_out = 14, mem = 14
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1760, wr = 0, rd = 0, data_in = 11
TIME =                  1770, data_out = 15, mem = 15
TIME =                  1780, wr = 0, rd = 1, data_in = 11
TIME =                  1790, data_out = 15, mem = 15
== PASS ===== PASS ==== PASS ==== PASS ===
TIME =                  1800, wr = 0, rd = 0, data_in = 11
TIME =                    1810, data_out =  16, mem =  16
TIME =                    1820, wr = 0, rd = 1, data_in = 11
TIME =                    1830, data_out =  16, mem =  16
== PASS ===== PASS ==== PASS ==== PASS ==
```

Listing 6: FIFO Output Log

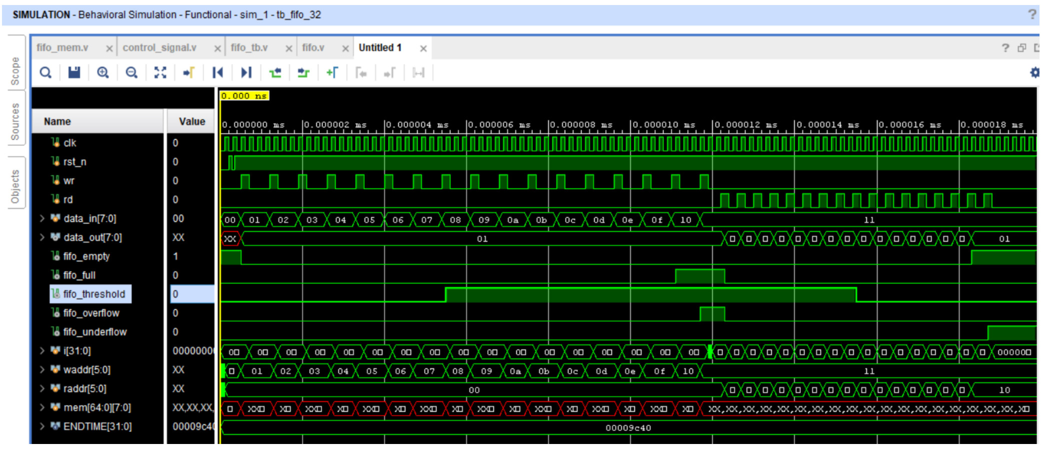

Figure 7: FIFO General Waveform

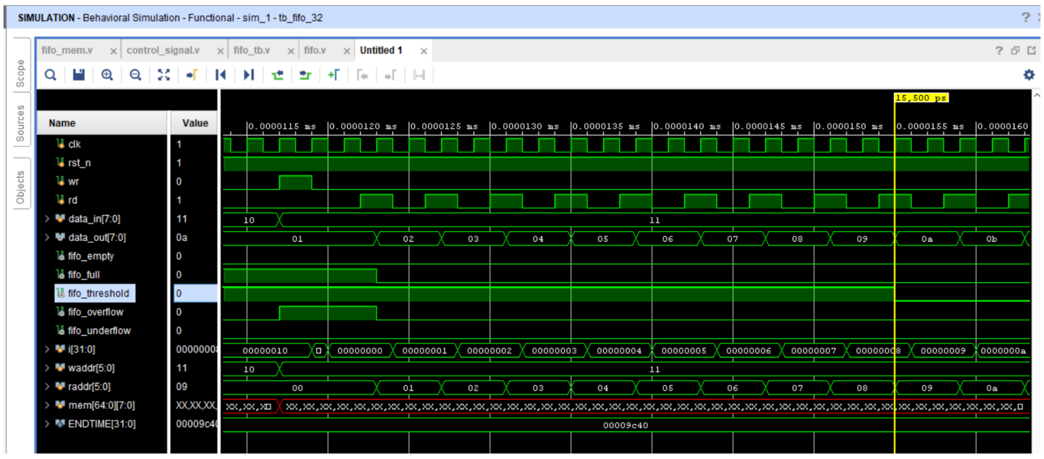

Figure 8: FIFO READ CASE Waveform

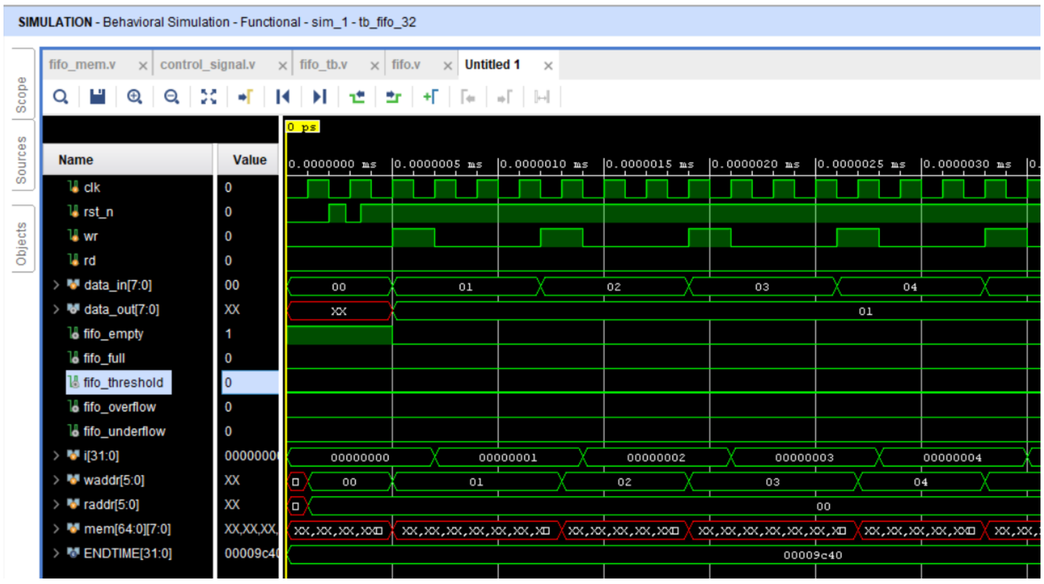

Figure 9: FIFO WRITE CASE Waveform

4.2 PWM

The Design of PWM comes as follow:

```
module PWM_Generator(
    clk , // 100MHz clock input
    increase_duty , // input to increase 10% duty cycle
    decrease_duty , // input to decrease 10% duty cycle
   PWM_OUT // 10MHz PWM output signal
    );
    input clk;
    input increase_duty;
    input decrease_duty;
    output PWM_OUT;
    wire slow_clk_enable; // slow clock enable signal for debouncing FFs
    reg[27:0] counter_debounce=0;// counter for creating slow clock enable signals
    wire tmp1,tmp2,duty_inc;// temporary flip-flop signals for debouncing the increasing button
    wire tmp3,tmp4,duty_dec;// temporary flip-flop signals for debouncing the decreasing button
    reg[3:0] counter_PWM=0;// counter for creating 10Mhz PWM signal
    reg[3:0] DUTY_CYCLE=5; // initial duty cycle is 50%

    always @(posedge clk)
    begin
```

```
        counter_debounce <= counter_debounce + 1;
        if(counter_debounce>=1)
            counter_debounce <= 0;
    end
    assign slow_clk_enable = counter_debounce == 1 ?1:0;

    DFF_PWM PWM_DFF1(clk,slow_clk_enable,increase_duty,tmp1);
    DFF_PWM PWM_DFF2(clk,slow_clk_enable,tmp1, tmp2);

    assign duty_inc =  tmp1 & (~ tmp2) & slow_clk_enable;

    DFF_PWM PWM_DFF3(clk,slow_clk_enable,decrease_duty, tmp3);
    DFF_PWM PWM_DFF4(clk,slow_clk_enable,tmp3, tmp4);

    assign duty_dec =  tmp3 & (~ tmp4) & slow_clk_enable;
    always @(posedge clk)
    begin
        if(duty_inc==1 && DUTY_CYCLE <= 9)
            DUTY_CYCLE <= DUTY_CYCLE + 1;// increase duty cycle
    by 10%
        else if(duty_dec==1 && DUTY_CYCLE>=1)
            DUTY_CYCLE <= DUTY_CYCLE - 1;//decrease duty cycle
    by 10%
    end

    always @(posedge clk)
    begin
        counter_PWM <= counter_PWM + 1;
        if(counter_PWM>=9)
            counter_PWM <= 0;
    end
    assign PWM_OUT = counter_PWM < DUTY_CYCLE ? 1:0;
    endmodule

module DFF_PWM(clk,en,D,Q);
    input clk,en,D;
    output reg Q;
    always @(posedge clk)
    begin
        if(en==1) // slow clock enable signal
            Q <= D;
    end
endmodule
```

Listing 7: PWM Module Design

```
module PWM_Generator_tb;
// Inputs
    reg clk;
    reg increase_duty;
    reg decrease_duty;
    // Outputs
    wire PWM_OUT;
// Instantiate the PWM Generator with variable duty cycle in
   Verilog
    PWM_Generator PWM_Generator_Unit(
        .clk(clk),
        .increase_duty(increase_duty),
        .decrease_duty(decrease_duty),
        .PWM_OUT(PWM_OUT)
        );
// Create 100Mhz clock
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        increase_duty = 0;
        decrease_duty = 0;
        #50;
        increase_duty = 1;
        #50;// increase duty cycle by 10%
        increase_duty = 0;
        #50;
        increase_duty = 1;
        #50;// increase duty cycle by 10%
        increase_duty = 0;
        #50;
        increase_duty = 1;
        #50;// increase duty cycle by 10%
        increase_duty = 0;
        #50;
        decrease_duty = 1;
        #50;//decrease duty cycle by 10%
        decrease_duty = 0;
        #50;
        decrease_duty = 1;
        #50;//decrease duty cycle by 10%
        decrease_duty = 0;
        #50;
        decrease_duty = 1;
        #50;//decrease duty cycle by 10%
        decrease_duty = 0;
        #50;
        increase_duty = 1;
        #50;// increase duty cycle by 10%
        increase_duty = 0;
        #50;
        increase_duty = 1;
        #50;// increase duty cycle by 10%
        increase_duty = 0;
        #50;
        increase_duty = 1;
        #50;// increase duty cycle by 10%
        increase_duty = 0;
        #50
        increase_duty = 1;
        #50;// increase duty cycle by 10%
        increase_duty = 0;
        #50;
        increase_duty = 1;
        #50;// increase duty cycle by 10%
        increase_duty = 0;
        #200;
        $stop;
    end
endmodule
```

Listing 8: PWM Module Design Testbench

And the result is:

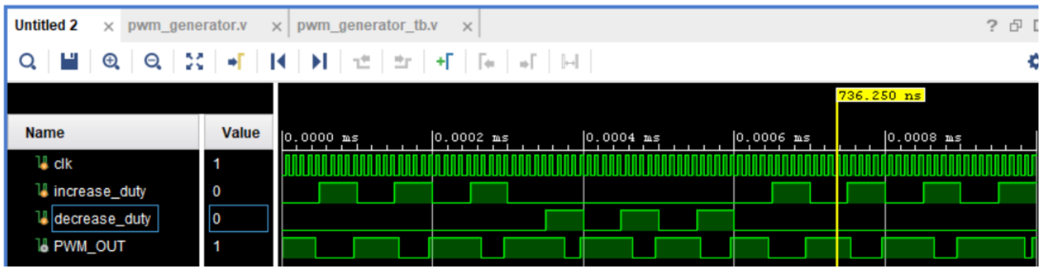

Figure 10: PWM Waveform 1

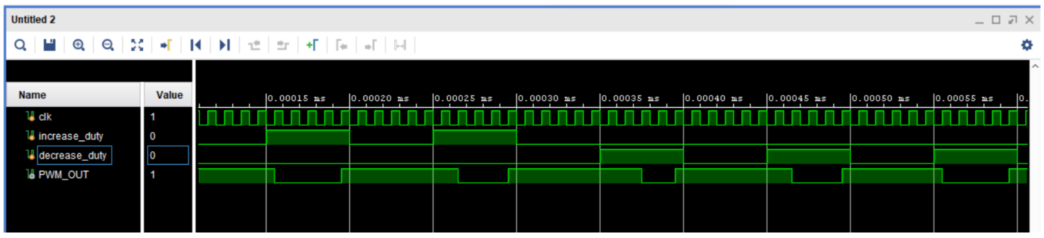

Figure 11: PWM Waveform 2

5 Conclusion

FIFOs are a storage mechanism that operates on a first-in, first-out basis. They are useful for managing the arrival of asynchronous data. Integrating a FIFO with an ISR, such as the UART ISR, can make processing the incoming data much easier for the application developer.
Pulse width modulation is a great method of controlling the amount of power delivered to a load without dissipating any wasted power.