ISTANBUL TECHNICAL UNIVERSITY

ELECTRIC-ELECTRONIC FACULTY

EHB326E
Introduction to Embedded Systems
Master-Slave Picoblaze Configuration

Designer:
M. Akif Özkanoğlu
040150037
Ömer Demirci
040160235

Supervisor:
M. Erhan Yalçın

January 9, 2020

Contents

List of Figures

Introduction

In this project, it is designed master-slave configurated microcontrollers which passes data between microcontrollers and block ram.The microcontroller for this project is choosen as picoblaze 6 which is 8-bit Xilinx soft-processor. Block RAM is used to read 8 bit data by master PicoBlaze. After reading process, master PicoBlaze sends data to slave PicoBlaze through 1 bit width line. For data transmission, A FIFO structure is constructed with a datapath and basic controller blocks.

FIFO Structure

Definition of FIFO

FIFO is the abbreviation for First-In, First-Out data buffer.It is a method for handling data structures where the first element is processed first and the newest element is processed last. In the structure of that project, FIFO is constructed as circular queue since its pointers increments by 1 bit. When the all bits is set high, all of them will be zero, then it repeats again from the beginning of the queue.

In case of a circular array, read pointer will always point to the front of the queue which is firstly sent, and read pointer will always point to the end of the queue. Initially, the read and the write pointers will be pointing to the same location, this would mean that the FIFO is empty or full.

New data is always added to the location pointed by the write pointer, and once the data is added, write pointer is incremented to point to the next available location.

In a circular FIFO, data is not actually removed from the memory array. Only the read pointer is incremented by one position when read strobe is executed. As the array data is only the data between write and read, hence the data left outside is not a part of the memory anymore, hence removed. The write and the read pointer will get reinitialised to 0 every time they reach the end of the bit limit. Circular FIFO write and read reinitialised.

In this design, while input data is 8 bit, output send 1 bit data. Thus, read pointer will need extra bit pointer although one write pointer is sufficient for writing processor.

reference [1]

Example of fifo

Figure 2.1 shows a fifo which is loaded with data

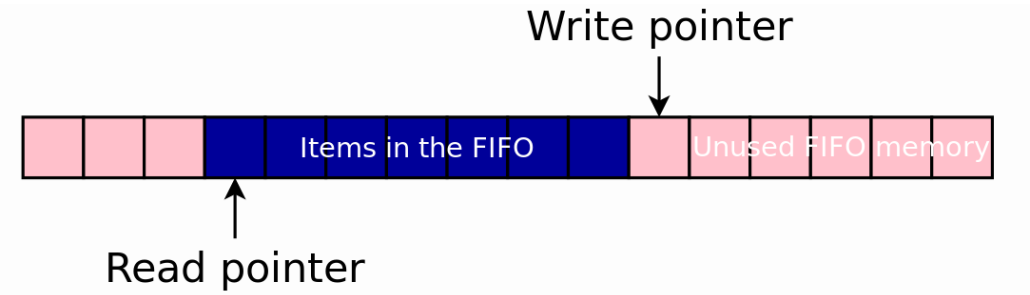


Figure 2.1: Example of FIFO Data.

Figure 2.2 shows a full FIFO

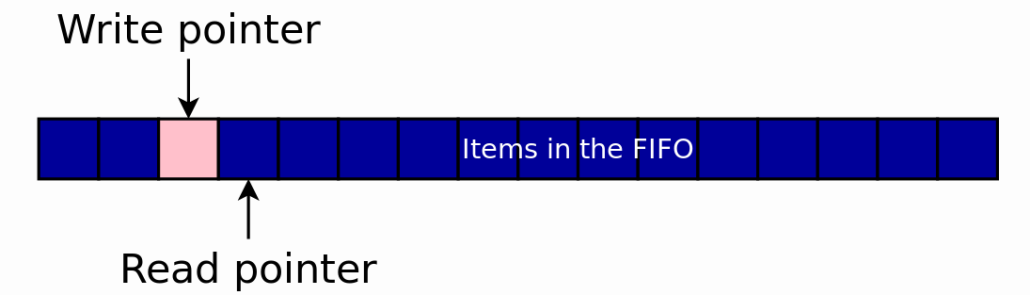


Figure 2.2: Example of full FIFO.

Figure 5.1 shows a empty FIFO

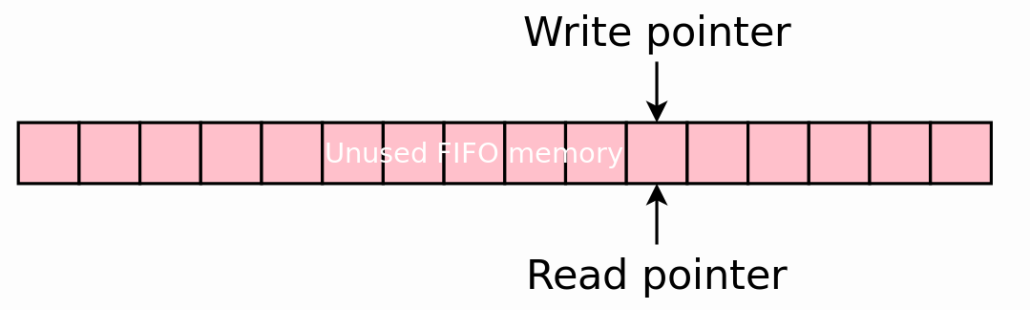


Figure 2.3: Example of empty FIFO.

In a circular FIFO, data is not actually removed from the memory array. Only the read pointer is incremented by one position when read strobe is executed. As the array data is only the data between write and read, hence the data left outside is not a part of the memory anymore, hence removed. The write and the read pointer will get reinitialised to 0 every time they reach the end of the bit limit. Circular FIFO write and read reinitialised.

In this design, while input data is 8 bit, output send 1 bit data. Thus, read pointer will need extra bit pointer although one write pointer is sufficient for writing processor.

Figure 2.4 flowchart of read pointer. Through this algorithm, it is ensured that read pointer will increase when the least significant is loaded into slave PicoBlaze. Since, the algorithm executed by slave processor calculates first loaded data into FIFO with shift left operation, finally it sends to output. For instance, it firstly takes most significant bit of the data, accumulate with next most significant bit after shift left operation in two instruction execution.

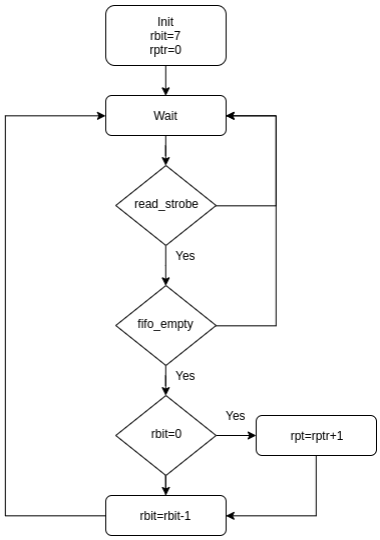


Figure 2.4: Flowchart of read pointer.

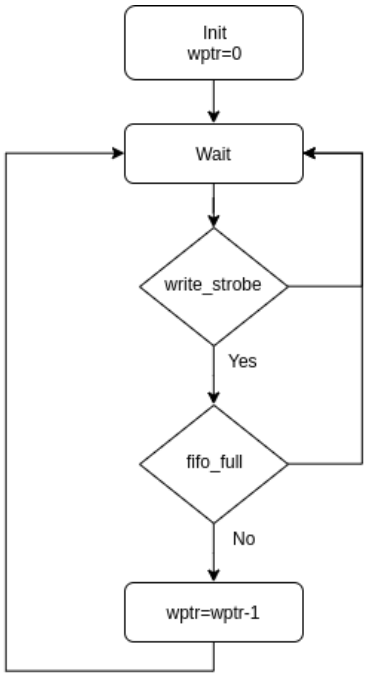


Figure 2.5: Flowchart of write pointer.

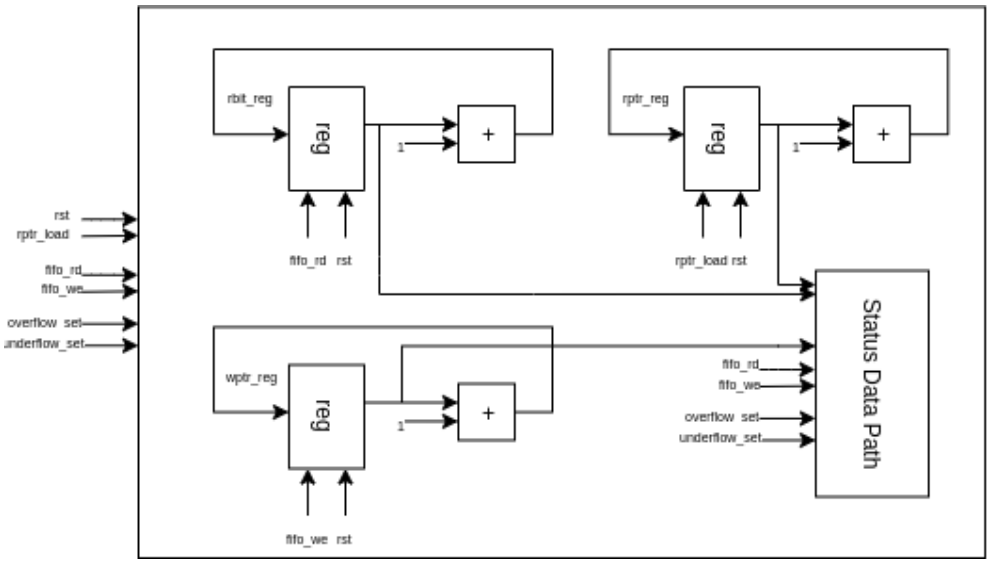


Figure 2.6: Datapath

Figure 2.6 is basic block diagram of data path. It adjusts pointer registers with load signals provided by controller of the FIFO top module. After that, it refresh next memory flags inside status data path module.

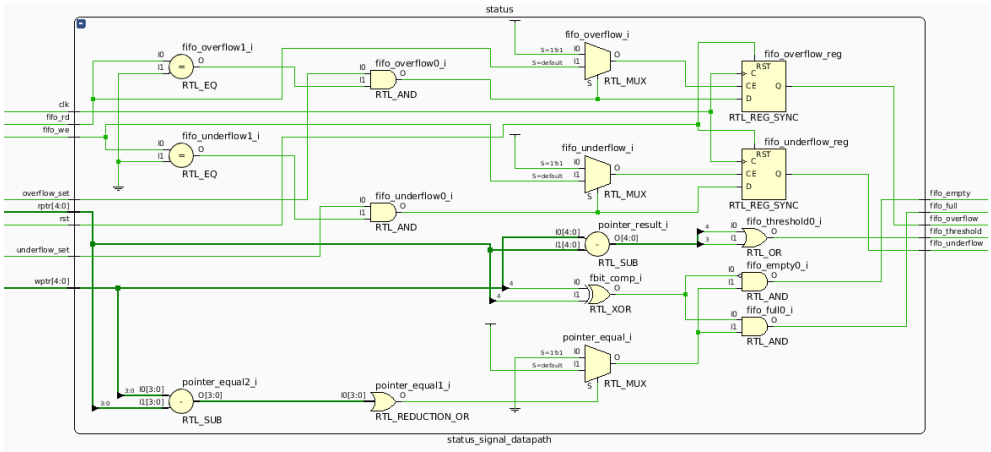


Figure 2.7: Data path for status signals.

Figure 2.7 shows status signal data path module which determine flag of the register array. To be more specific, by comparing locations of pointers, *"fifo full"* and *"fifo empty"* flag are established and connected as output port into controller of the fifo top module.

Figure 2.8 shows the connection between data path and controller which constitute the FIFO structure. Signals are connected together in the top FIFO module. As a result of these connections, FIFO data structures are successfully realized in order for picoblazes to concurrently work independently from each other.

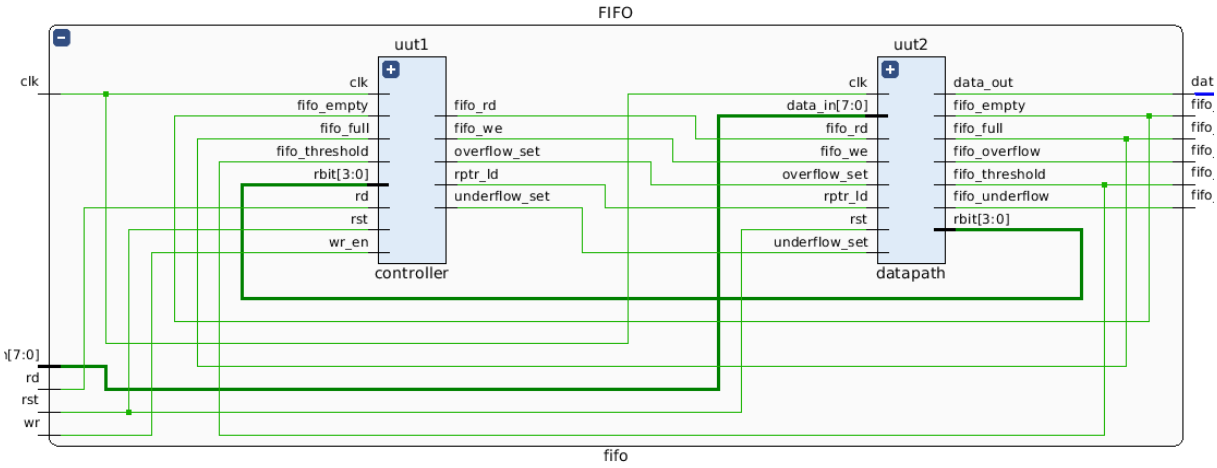


Figure 2.8: Schematic for System.

Design

High Level System Schematic

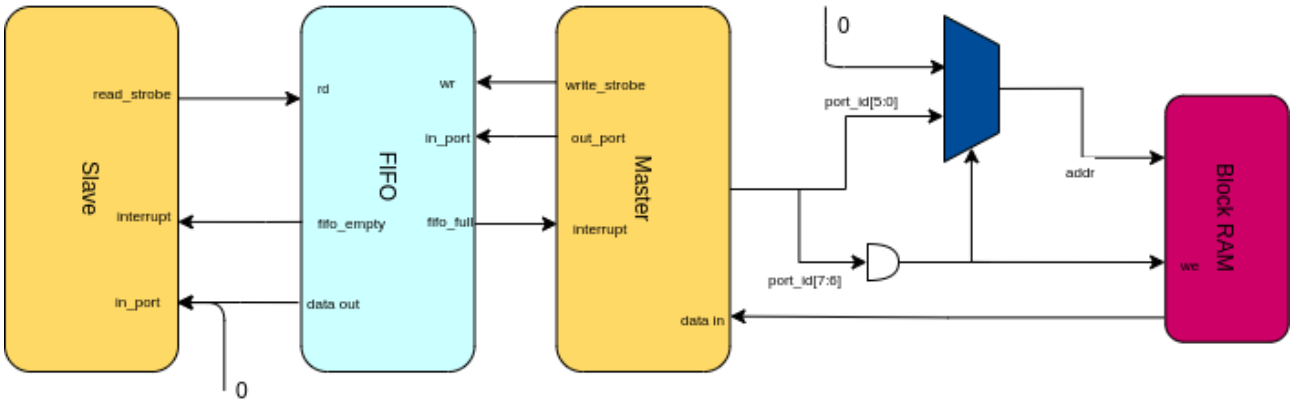


Figure 3.1: High Level Schematic

Figure 3.1 is the high level system connection diagram. A block ram is added to the master which can be read data from arbitrary address from ram. After reading instruction, master transmits data to the FIFO. Slave reads data from FIFO in arbitrary time and writes it to out port.

ASM

Figure 3.2 is the algorithmic state machine for the system. The FIFO signals are state control signals which enables interrupt for both master and slave picoblazes whenever the FIFO is full, master goes into interrupt with FIFO full flag, also slave goes into interrupt when the FIFO empty signal is HIGH.

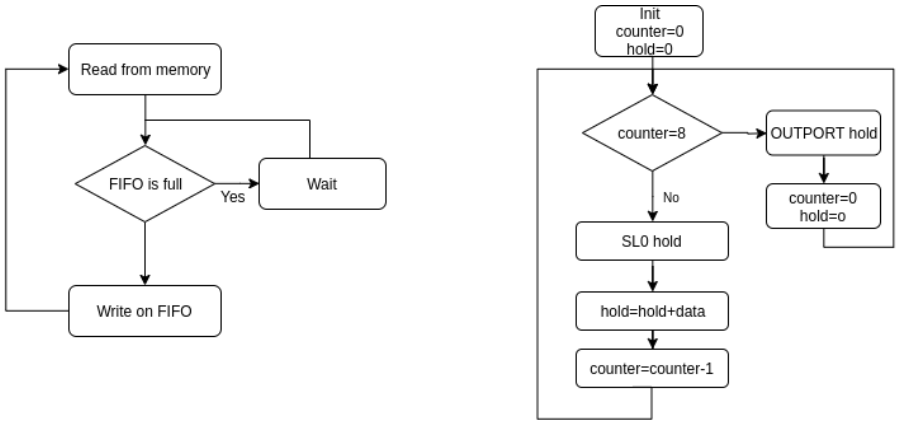
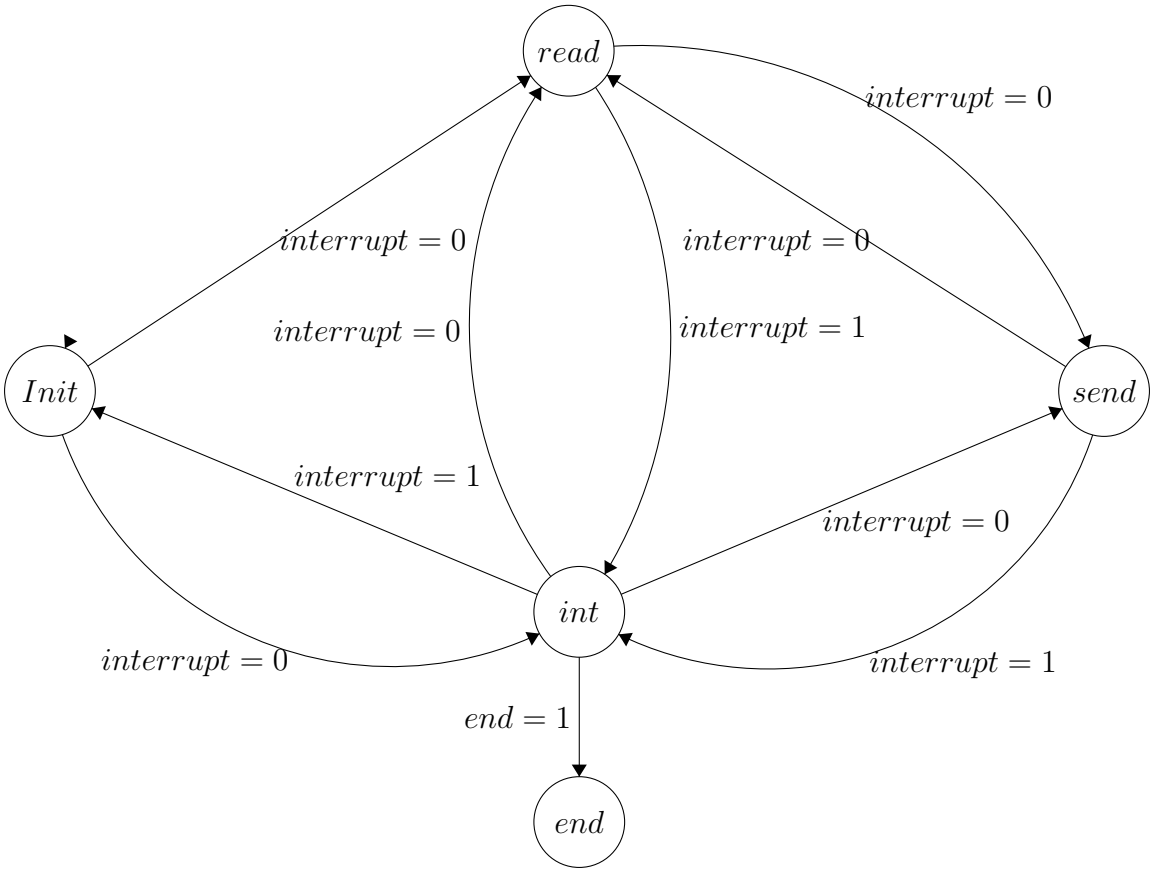
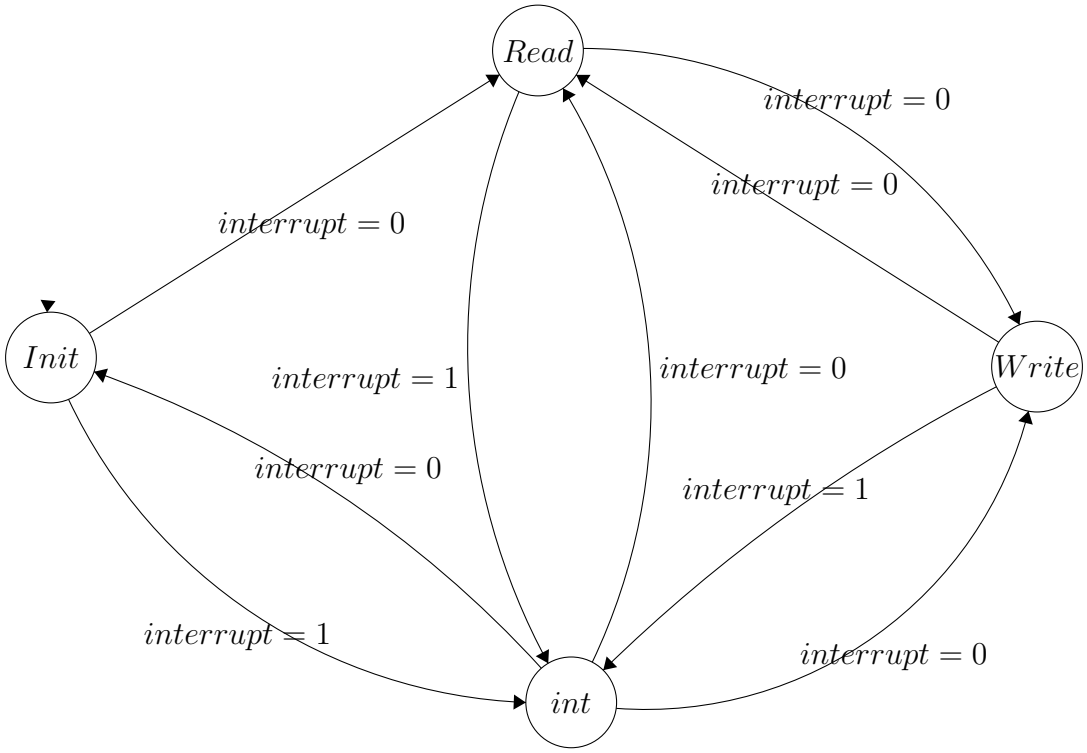


Figure 3.2: ASM

Master FSM



Slave FSM


Read
Init
Write
int
interrupt = 0
interrupt = 0
interrupt = 0
interrupt = 1
interrupt = 0
interrupt = 1
interrupt = 1
interrupt = 0

Simulation

Master Simulation

Figure 4.1 shows the master picoblaze simulation result. According to the Master simulation signals, it can be seen that master picoblaze reads data from ram and writes data to the FIFO accordingly. It can be seen from the picture that after FIFO is full, interrupt is asserted and read-write process is terminated by interrupt untill the FIFO full flag becomes low

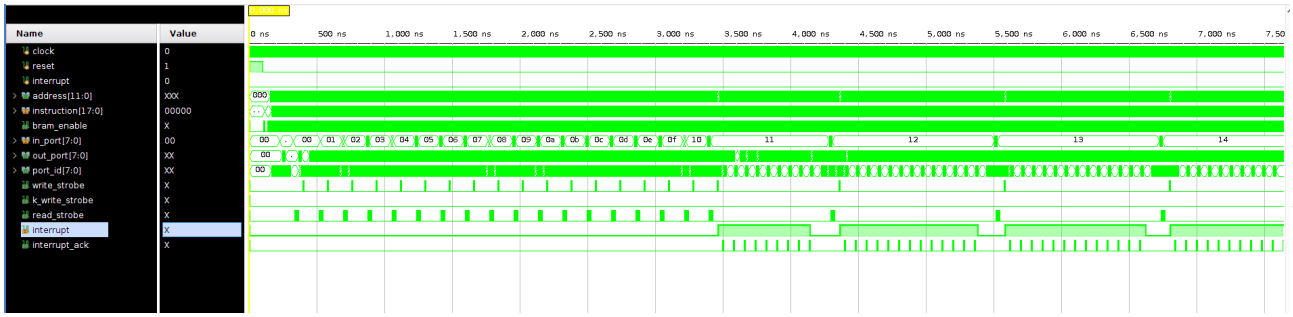


Figure 4.1: Master Picoblaze Simulation.

Slave Simulation

Figure 4.2 shows the master picoblaze simulation result. The slave picoblaze goes into interrupt only at the startup, because it reads 1-bit data from picoblaze while master writes 8-bit at once to the FIFO.

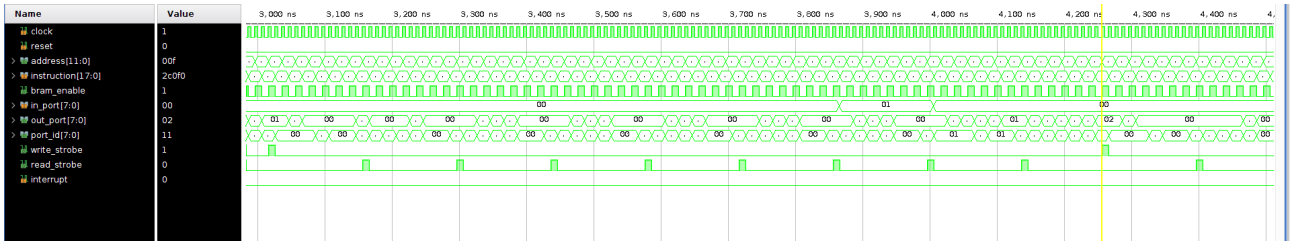


Figure 4.2: Slave Picoblaze Simulation.

FIFO Datapath Simulation

Figure 4.3 shows the FIFO simulation result. FIFO full flag is mostly high because of the write speed is more than read speed 8 times.

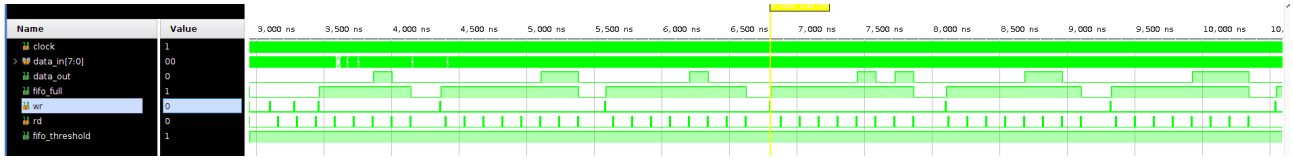


Figure 4.3: FIFO Simulation.

TesBench

TestBench Verilog Code

```
1  `timescale 1ns / 1ps
2
3  module TOP_tb;
4      reg clock,reset,interrupt;
5      TOP uut(clock,reset);
6      initial
7      begin
8          interrupt<=0;
9          reset<=0;
10     end
11
12     always
13     begin
14         clock<=0;
15         #5;
16         clock<=1;
17         #5;
18     end
19     initial
20     begin
21         reset<=1;
22         #100;
23         reset<=0;
24     end
25 endmodule
```

Conclusions

This report summarized the design and simulation of the master-slave picoblaze configuration connected with a block ram. According to the simulations, it is seen that FIFO structure enables asynchronous write-read operation between mater and slave and interrupt signals for controlling write-read operations.

Appendix

Vivado Schematic

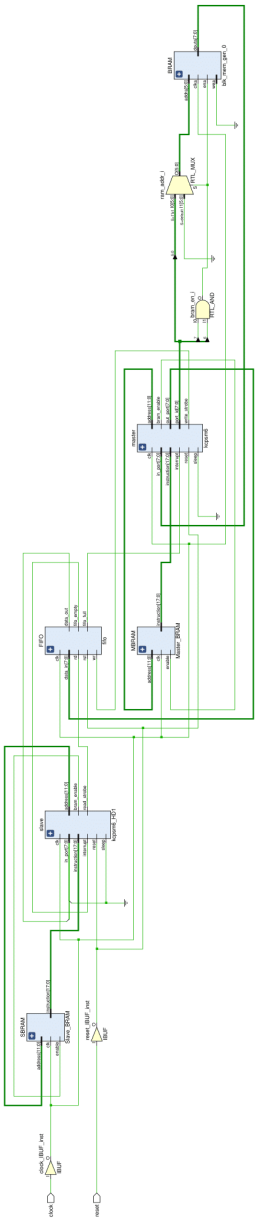


Figure 5.1: Vivado Schematic.

Master Assembly Code

```
#ifDef proc::xPblze6
        #set proc::xPblze6::scrpdSize,                          64
                                        ; [64, 128, 256]
        #set proc::xPblze6::clkFreq,                            100000000
                                ; in Hz

        #set IOdev::Master_BRAM::en,                                    TRUE
        #set IOdev::Master_BRAM::type,                          mem
        #set IOdev::Master_BRAM::size,                          1024

        #set instmem::pageSize,                                 1024
        #set instmem::pageCount,                                1
        #set instmem::sharedMemLocation,                        loMem   ; [ hiMem, loMem ]

        #set IOdev::Master_BRAM::value,                         instMem

        #set IOdev::Master_BRAM::verilogEn,                             TRUE
        #set IOdev::Master_BRAM::verilogEntityName,             "Master_BRAM"
        #set IOdev::Master_BRAM::verilogTmplFile,                       "
            ROM_form_7S_4K_14March13.v"
        #set IOdev::Master_BRAM::verilogTargetFile,             "Master_BRAM.v"
#endIf

#EQU write_address s0
#EQU data s1
#EQU read_adress s4
#EQU interrupt_reg sA
#ORG ADDR, 0
        INT ENABLE
init:
        LOAD write_address, 0x25
        LOAD data, 0x00
        LOAD read_adress, 0xFF
        LOAD interrupt_reg, 0X00
read:
        ADD read_adress, 0x01
        COMP read_adress, 64
        JUMP Z,dis_interrupt
        ADD read_adress, b11000000
        RDPRT data, (read_adress)
        RDPRT data, (read_adress)
        ADD read_adress, b01000000
send:
        WRPRT data,(write_address)
        JUMP read
dis_interrupt:
        INT DISABLE
end:
        JUMP end
isr:
        ADD interrupt_reg , 01
        RETI ENABLE
#ORG ADDR, 1023
    JUMP isr
```

Slave Assembly Code

```
#ifDef proc::xPblze6
        #set proc::xPblze6::scrpdSize,                          64
                                        ; [64, 128, 256]
        #set proc::xPblze6::clkFreq,                            100000000
                                ; in Hz

        #set IOdev::Slave_BRAM::en,                                     TRUE
        #set IOdev::Slave_BRAM::type,                           mem
        #set IOdev::Slave_BRAM::size,                           1024

        #set instmem::pageSize,                                 1024
        #set instmem::pageCount,                                1
        #set instmem::sharedMemLocation,                        loMem   ; [ hiMem, loMem ]

        #set IOdev::Slave_BRAM::value,                          instMem

        #set IOdev::Slave_BRAM::verilogEn,                              TRUE
        #set IOdev::Slave_BRAM::verilogEntityName,              "Slave_BRAM"
        #set IOdev::Slave_BRAM::verilogTmplFile,                        "
            ROM_form_7S_4K_14March13.v"
        #set IOdev::Slave_BRAM::verilogTargetFile,              "Slave_BRAM.v"
#endIf

#EQU hold s0
#EQU data s1
#EQU counter s2
#EQU read_adress s4
#EQU write_address,sF
#EQU interrupt_reg sA
#ORG ADDR, 0
        INT ENABLE
init:
        LOAD data, 0x00
        LOAD read_adress, 0xFF
        LOAD interrupt_reg, 0X00
        LOAD write_address, 0x11
        LOAD hold, 0x00
        LOAD counter,0x00
read:
        COMP counter,8
        JUMP Z, write
        SL0 hold
        RDPRT data, (read_adress)
        ADD hold, data
        ADD counter, 1
        JUMP read
write:
        WRPRT hold, (write_address)
        LOAD hold, 0x00
        LOAD counter,0x00
        JUMP read
isr:
        ADD interrupt_reg , 01
        RETI ENABLE
#ORG ADDR, 1023
    JUMP isr
```

TOP Module Verilog Code

```
26
27  `timescale 1ns / 1ps
28
29  module TOP(
30      input clock,reset
31      );
32
33      //////////////////////MASTER PICOBLAZE///////////////////////////////////
34      wire [17:0] instruction_master;
35
36      wire [7:0] in_port_master,
37                 out_port_master,
38                 port_id_master;
39
40      wire [11:0] address_master;
41
42      wire bram_enable_master,
43           write_strobe_master,
44           k_write_strobe_master,
45           read_strobe_master,
46           interrupt_master,
47           interrupt_ack_master;
48
49      Master_BRAM MBRAM(.address(address_master),
50                        .instruction(instruction_master),
51                        .enable(bram_enable_master),
52                        .clk(clock));
53
54      kcpsm6 master(.address(address_master),
55                    .instruction(instruction_master),
56                    .bram_enable(bram_enable_master),
57                    .in_port(in_port_master),
58                    .out_port(out_port_master),
59                    .port_id(port_id_master),
60                    .write_strobe(write_strobe_master),
61                    .k_write_strobe(k_write_strobe_master),
62                    .read_strobe(read_strobe_master),
63                    .interrupt(interrupt_master),
64                    .interrupt_ack(interrupt_ack_master),
65                    .sleep(1'b0),
66                    .reset(reset),
67                    .clk(clock));
68
69
70
71
72      //////////////////////GENERATED BLOCK RAM///////////////////////////////////
73      wire [5:0]ram_addr;
74      wire bram_en=port_id_master[7]&port_id_master[6];
75      assign ram_addr=bram_en?port_id_master[5:0]:0;
76
77      blk_mem_gen_0 BRAM(.clka(clock),
78                         .ena(bram_en),
79                         .wea(1'b0),
80                         .addra(ram_addr),
81                         .dina(8'bZ),
82                         .douta(in_port_master));
83
84      wire [17:0] instruction_slave;
85
86      wire [7:0] in_port_slave,
87                 out_port_slave,
88                 port_id_slave;
89
90      wire [11:0] address_slave;
91
92      wire bram_enable_slave,
93           write_strobe_slave,
94           k_write_strobe_slave,
95           read_strobe_slave,
96           interrupt_slave,
```

```
97          interrupt_ack_slave;
98
99
100     //////////////////////SLAVE PICOBLAZE/////////////////////////////////
101     Slave_BRAM SBRAM(.address(address_slave),
102                      .instruction(instruction_slave),
103                      .enable(bram_enable_slave),
104                      .clk(clock));
105
106     kcpsm6 slave(.address(address_slave),
107                  .instruction(instruction_slave),
108                  .bram_enable(bram_enable_slave),
109                  .in_port(in_port_slave),
110                  .out_port(out_port_slave),
111                  .port_id(port_id_slave),
112                  .write_strobe(write_strobe_slave),
113                  .k_write_strobe(k_write_strobe_slave),
114                  .read_strobe(read_strobe_slave),
115                  .interrupt(interrupt_slave),
116                  .interrupt_ack(interrupt_ack_slave),
117                  .sleep(1'b0),
118                  .reset(reset),
119                  .clk(clock));
120
121
122     //////////////////////FIFO/////////////////////////////////
123     wire [7:0] data_in;
124     assign data_in=out_port_master;
125     wire data_out,
126          fifo_full,
127          fifo_empty,
128          fifo_threshold,
129          fifo_overflow,
130          fifo_underflow;
131
132     assign in_port_slave={7'd0,data_out};
133
134     fifo FIFO(write_strobe_master,
135               read_strobe_slave,
136               clock,
137               reset,
138               data_in,
139               data_out,
140               fifo_full,
141               fifo_empty,
142               fifo_threshold,
143               fifo_overflow,
144               fifo_underflow
145     );
146
147
148     assign interrupt_master=fifo_full;
149     assign interrupt_slave=fifo_empty;
150
151 endmodule
```

FIFO Verilog Code

```
152 `timescale 1ns / 1ps
153
154
155 module fifo(
156     input wr, rd, clk, rst,
157     input [7:0] data_in,
158     output data_out,
159            fifo_full,
160            fifo_empty,
161            fifo_threshold,
162            fifo_overflow,
163            fifo_underflow
164     );
165
166     wire [4:0] wptr, rptr;
167     wire [3:0] rbit;
168     wire rptr_ld,fifo_rd,fifo_we,overflow_set, underflow_set;
169
170     controller uut1(clk,
171                     rst,
172                     wr,
173                     rd,
174                     fifo_full,
175                     fifo_empty,
176                     fifo_threshold,
177                     rbit,
178                     rptr_ld,
179                     fifo_rd,
180                     fifo_we,
181                     overflow_set,
182                     underflow_set);
183
184
185     datapath uut2(clk,
186                   rst,
187                   rptr_ld,
188                   fifo_rd,
189                   fifo_we,
190                   overflow_set,
191                   underflow_set,
192                   data_in,
193                   fifo_overflow,
194                   fifo_underflow,
195                   rbit,
196                   fifo_full,
197                   fifo_empty,
198                   fifo_threshold,
199                   data_out);
200
201 endmodule
```

Controller Verilog Code

```
202 `timescale 1ns / 1ps
203
204 module controller (
205     input clk,rst,wr_en,rd,fifo_full,fifo_empty, fifo_threshold,
206     input [3:0] rbit,
207     output rptr_ld,fifo_rd,fifo_we,overflow_set, underflow_set
208     );
209
210     status_controller status(wr_en,
211                              rd,
212                              fifo_full,
213                              fifo_empty,
214                              overflow_set,
215                              underflow_set);
216
217     assign fifo_rd = (~fifo_empty) & rd;
218     assign fifo_we = (~fifo_full) & wr_en;
219     assign rptr_ld = (rbit[2:0]==3'b000) & fifo_rd;
220
221 endmodule
222
223
224 module status_controller(
225     input wr_en,rd,fifo_full,fifo_empty,
226     output overflow_set,underflow_set);
227
228     assign overflow_set = fifo_full & wr_en;
229     assign underflow_set = fifo_empty&rd;
230 endmodule
```

Datapath Verilog Code

```
231
232 `timescale 1ns / 1ps
233
234
235 module datapath(
236     input clk,rst,rptr_ld,fifo_rd,fifo_we,overflow_set, underflow_set,
237     input [7:0] data_in,
238     output fifo_overflow, fifo_underflow,
239     output [3:0] rbit,
240     output fifo_full, fifo_empty, fifo_threshold, data_out);
241
242     wire [4:0] rptr;
243     wire [4:0] wptr;
244
245     rptr_reg reg1(clk,
246                   rst,
247                   rptr_ld,
248                   rptr);
249
250     rbit_reg reg2(clk,
251                   rst,
252                   fifo_rd,
253                   rbit);
254
255     wptr_reg reg3(clk,
256                   rst,
257                   fifo_we,
258                   wptr);
259
260     memory_array_reg memory(data_out,
261                             data_in,
262                             clk,
263                             fifo_we,
264                             wptr,
265                             rptr,
```

```
                                rbit);

    status_signal_datapath  status(clk,
                                   rst,
                                   overflow_set,
                                   underflow_set,
                                   fifo_we,
                                   fifo_rd,
                                   wptr, rptr,
                                   fifo_full,
                                   fifo_empty,
                                   fifo_threshold,
                                   fifo_overflow,
                                   fifo_underflow);
endmodule



module rptr_reg(clk,rst,load,rptr);
    input clk,rst,load;
    output reg [4:0] rptr;

    always @(posedge clk)
    begin
        if(rst)
        begin
            rptr <= 5'b000000;
        end
        else if(load)
        begin
            rptr <= rptr + 5'b000001;
        end
    end
endmodule

module rbit_reg(clk,rst,load,rbit);

    input clk,rst,load;
    output reg [3:0]rbit;

    always @(posedge clk)
    begin
        if(rst)
        begin
            rbit <= 4'b0111;
        end
        else if(load)
        begin
            rbit <= rbit + 4'b1111;
        end
    end

endmodule


module wptr_reg(clk,rst,fifo_we,wptr);

    input clk,rst,fifo_we;
    output [4:0] wptr;

    reg [4:0] wptr;


    always @(posedge clk )
    begin
        if(rst)
            wptr <= 5'b00000;
        else if(fifo_we)
            wptr <= wptr + 5'b00001;
        else
           wptr <= wptr;
    end

endmodule
```

```
340
341
342 module memory_array_reg(data_out,data_in,clk,load,wptr,rptr,rbit);
343
344     output data_out;
345     input[7:0] data_in;
346     input clk,load;
347     input[4:0] wptr,rptr;
348     input [3:0]rbit;
349
350     reg[7:0] data_reg[15:0];
351
352     assign data_out = data_reg[rptr[3:0]][rbit[2:0]];
353
354     always @(posedge clk)
355     begin
356         if(load)
357             data_reg[wptr[3:0]] <= data_in;
358     end
359
360 endmodule
361
362
363 module status_signal_datapath(
364     input clk,
365           rst,
366           overflow_set,
367           underflow_set,
368           fifo_we,
369           fifo_rd,
370
371     input [4:0] wptr, rptr,
372
373     output reg fifo_full,
374                fifo_empty,
375                fifo_threshold,
376                fifo_overflow,
377                fifo_underflow);
378
379
380     wire fbit_comp, overflow_set, underflow_set;
381     wire pointer_equal;
382     wire [4:0] pointer_result;
383
384     assign fbit_comp = wptr[4] ^ rptr[4];
385     assign pointer_equal = (wptr[3:0] - rptr[3:0]) ? 0:1;
386     assign pointer_result = wptr[4:0] - rptr[4:0];
387
388     always @(*)
389     begin
390         fifo_full =fbit_comp & pointer_equal;
391         fifo_empty = (~fbit_comp) & pointer_equal;
392         fifo_threshold = (pointer_result[4]||pointer_result[3]) ? 1:0;
393     end
394
395     always @(posedge clk )
396     begin
397         if(rst)
398             fifo_overflow <=0;
399         else if((overflow_set==1)&&(fifo_rd==0))
400             fifo_overflow <=1;
401         else if(fifo_rd)
402             fifo_overflow <=0;
403         else
404             fifo_overflow <= fifo_overflow;
405     end
406
407     always @(posedge clk )
408     begin
409         if(rst)
410             fifo_underflow <=0;
411         else if((underflow_set==1)&&(fifo_we==0))
412             fifo_underflow <=1;
413         else if(fifo_we)
```

```
414             fifo_underflow <=0;
415         else
416             fifo_underflow <= fifo_underflow;
417     end
418
419  endmodule
```

TestBench Verilog Code

```
420 `timescale 1ns / 1ps
421
422 module TOP_tb;
423     reg clock,reset,interrupt;
424     TOP uut(clock,reset);
425     initial
426     begin
427         interrupt<=0;
428         reset<=0;
429     end
430
431     always
432     begin
433         clock<=0;
434         #5;
435         clock<=1;
436         #5;
437     end
438     initial
439     begin
440         reset<=1;
441         #100;
442         reset<=0;
443     end
444 endmodule
```

Memory Coe File

```
1  MEMORY_INITIALIZATION_RADIX=10;
2  MEMORY_INITIALIZATION_VECTOR=
3  0,
4  1,
5  2,
6  3,
7  4,
8  5,
9  6,
10 7,
11 8,
12 9,
13 10,
14 11,
15 12,
16 13,
17 14,
18 15,
19 16,
20 17,
21 18,
22 19,
23 20,
24 21,
25 22,
26 23,
27 24,
28 25,
29 26,
30 27,
```

```
28,
29,
30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63;
```

References

[1] *FIFO Structure*. URL: https://www.wikizeroo.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEu (accessed: 01.09.2020).