FPGA Application Week 11 Exercise

CYEE 10828241 Chen Da-Chuan

May 8, 2023

Contents

List of Figures

List of Tables

1 Exercise 11-1

1.1 Objective

Create a stack circuit, which writes program counter to a designated stack location when enabled. Also export the specified stack location value if its told to write.

1.2 Operation

Table 1: Homework 11-1 Operation detail

type	var	operation
input	PCX	program counter, which instruction is being executed
input	clk	clock signal
input	SP	stack pointer, where to store PCX
input	PUSH	push signal, when high, store PCX to stack
input	STKO	stack output, output the stored PCX specified by SP

1.3 Code

```
1   module stk_v(PCX, clk, SP,PUSH,STKO );
2   input [11:0] PCX;
3   input clk, PUSH;
4   input [2:0] SP;
5   output [11:0] STKO;
6   reg [11:0] Q0,Q1,Q2, Q3, Q4,Q5, Q6, Q7,STKO;
7   always@(posedge clk )
8   begin
9    if (PUSH==1)
10   begin
11    case (SP)
12      3'b000 :   Q0= PCX;
13      3'b001 :    Q1= PCX;
14      3'b010 :    Q2= PCX;
15      3'b011 :    Q3= PCX;
16      3'b100 :    Q4= PCX;
17      3'b101 :    Q5= PCX;
18      3'b110 :    Q6= PCX;
19      3'b111 :    Q7= PCX;
20    endcase
21   end
22  end
23
24  always@(SP)
25  begin
26   case (SP)
27     3'b000 : STKO= Q0;
28     3'b001 :    STKO= Q1;
29     3'b010 :    STKO= Q2;
30     3'b011 :    STKO= Q3;
31     3'b100 :    STKO= Q4;
32     3'b101 :    STKO= Q5;
33     3'b110 :    STKO= Q6;
34     3'b111 :    STKO= Q7;
35     default :
36    begin
37      STKO=0;
38     end
39   endcase
40
41  end
42
43  endmodule
44
```

Figure 1: Main file

```
1   `timescale 1 ns/1 ns
2   module test_stk;
3   reg [11:0] PCX;
4   reg clk, PUSH;
5   reg [2:0] SP;
6   wire [11:0] STKO;
7
8   stk_v DUT( .PCX(PCX) , .clk(clk), .PUSH(PUSH), .SP(SP), .STKO(STKO));
9
10  initial
11  begin
12    PCX = 12'h2fc ;
13    clk = 0;
14    PUSH = 0 ;
15    SP = 0;
16    #400 PUSH = 1;
17  end
18  always #50 clk = ~clk ;
19  always #100 PCX = PCX + 1 ;
20  always #100 SP = SP +1 ;
21
22  endmodule
```

Figure 2: Test file

1. Figure 1 Line1-6: Declare module name, input and output.
2. Figure 1 Line7-22: If triggered by clock signal, and if write enable is true, write PCX to specified stack location.
3. Figure 1 Line24-39: If triggered by pointer, output the specified stack location.
4. Figure 2 Line1-6: Define time unit and precision, module name, registers and wires.
5. Figure 2 Line8-8: Include main file.
6. Figure 2 Line10-17: Set initial value for PCX, clk, PUSH, and SP. At 400ns, set PUSH to 1.
7. Figure 2 Line18-18: Every 50ns, alternate clk between 0 and 1.
8. Figure 2 Line19-19: Every 100ns, add PCX by 1.
9. Figure 2 Line20-20: Every 100ns, add SP by 1.

1.4 Result

Figure 3: Exercise 11-1 Result

Table 2: Exercise 11-1 Truth Table

clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
PCX (12'h)	2fc	2fd	2fe	2ff	300	301	302	303	304	305
PUSH (1'b)	0	0	0	0	1	1	1	1	1	1
SP (4'h)	0	1	2	3	4	5	6	7	0	1
STKO (12'h)	000	000	000	000	300	301	302	303	304	305
description	PUSH=0 no write pass	PUSH=0 no write pass	PUSH=0 no write pass	PUSH=0 no write pass	PUSH=1 write PCX to Q4 pass	PUSH=1 write PCX to Q5 pass	PUSH=1 write PCX to Q6 pass	PUSH=1 write PCX to Q7 pass	PUSH=1 write PCX to Q0 pass	PUSH=1 write PCX to Q1 pass

2 Exercise 11-2

2.1 Objective

Code a stack pointer circuit which changes stack pointer location according to PUSH and POP signal.

2.2 Operation

Table 3: Homework 11-2 Operation detail

type	var	operation
input	PUSH	push signal, point the stack pointer to the previous location
input	POP	pop signal, point the stack pointer to the next location
input	clk	clock signal
output	SP	stack pointer

2.3 Code

```
1   module sp_v (PUSH, POP,clk,SP );
2   input PUSH,POP,clk;
3   output[2:0] SP;
4   reg [2:0] SP;
5   always @(posedge clk)
6   begin
7     if (POP==1)
8       SP=SP+1;
9     else if (PUSH ==1)
10      SP=SP-1;
11  end
12  endmodule
```

Figure 4: Main file

```
1   `timescale 1 ns/1 ns
2   module test_sp;
3   reg PUSH,POP,clk;
4   wire[2:0] SP;
5
6   sp_v DUT( .PUSH(PUSH) , .POP(POP), .clk(clk), .SP(SP));
7   initial
8   begin
9     PUSH = 0 ;
10    POP = 1;
11    clk = 0 ;
12  end
13
14  always #50 clk = ~clk ;
15  always #100 POP = ~POP ;
16  always #200 PUSH = ~PUSH ;
17
18
19  endmodule
20
```

Figure 5: Test file

1. Figure 4 Line1-4: Declare module name, input and output.
2. Figure 4 Line7-8: If clock is triggered, if POP signal is 1, SP will be added by 1.
3. Figure 4 Line9-10: If clock is triggered, if PUSH signal is 1, SP will be subtracted by 1.
4. Figure 5 Line1-4: Define time scale, module name, registers, and wires.
5. Figure 5 Line6-6: Include main file.
6. Figure 5 Line7-12: Set initial value of PUSH, POP, and clk signals.
7. Figure 5 Line14-14: Every 50ns, alternate clock signal.
8. Figure 5 Line15-15: Every 100ns, alternate POP signal.
9. Figure 5 Line16-16: Every 200ns, alternate PUSH signal.

2.4 Result

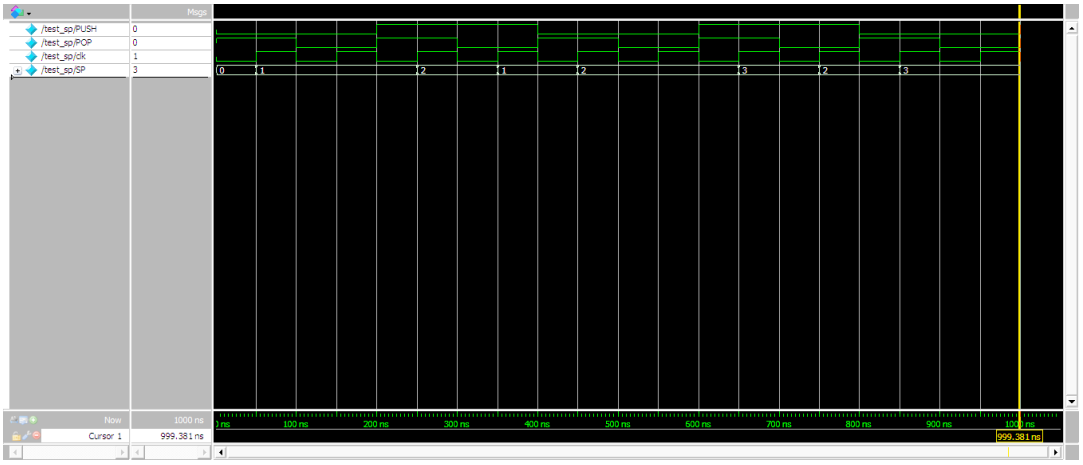

Figure 6: Exercise 11-2 Result

Table 4: Exercise 11-2 Truth Table

clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
PUSH (1'b)	0	0	1	1	0	0	1	1	0	0
POP	1	0	1	0	1	0	1	0	1	0
SP (4'h)	1	1	2	1	2	2	3	2	3	3
description	POP=1 SP+=1 pass	POP=0 PUSH=0 SP=SP pass	POP=1 SP+=1 pass	POP=0 PUSH=1 SP-=1 pass	POP=1 SP+=1 pass	POP=0 PUSH=0 SP=SP pass	POP=1 SP+=1 pass	POP=0 PUSH=1 SP-=1 pass	POP=1 SP+=1 pass	POP=0 PUSH=0 SP=SP pass

3 Exercise 11-3

3.1 Objective

Create a PC counter module which can return to stack pointer location, jump to specified location, or keep pointer at the next location.

3.2 Operation

Table 5: Homework 11-3 Operation detail

type	var	operation
input	clk	clock signal
input	RET	return signal, return to pointer poped from stack
input	JUMP	jump signal, jump to current pointer add specified number
input	STKO	receive poped stack pointer location
input	NUM	receive number to jump from current location
output	PC	program counter

3.3 Code

```
1   module pccounter_v(clk, RET, JUMP, STKO, NUM, PC);
2   input clk, RET, JUMP;
3   input [11:0] STKO, NUM;
4   output [11:0] PC;
5   reg [11:0] PC;
6
7   always @(posedge clk)
8   begin
9    if (RET==1)
10    PC = STKO;
11   else if (JUMP==1)
12    PC = PC+NUM;
13   else
14    PC=PC+1;
15  end
16
17  endmodule
```

Figure 7: Main file

```
1   `timescale 1 ns/1 ns
2   module test_pccounter;
3   reg clk, RET, JUMP;
4   reg [11:0] STKO, NUM;
5   wire [11:0] PC;
6
7
8
9   pccounter_v DUT( .clk(clk) , .RET(RET), .JUMP(JUMP), .STKO(STKO), .NUM(NUM),
10                  .PC(PC) );
11  initial
12  begin
13   clk = 0 ;
14   RET = 0;
15   JUMP = 1 ;
16   STKO = 12'h168 ;
17   NUM= 12'h486;
18
19  end
20  always #50 clk = ~clk ;
21  always #100 RET = ~RET ;
22  always #200 JUMP = ~JUMP ;
23
24
25  endmodule
```

Figure 8: Test file

1. Figure 7 Line1-5: Declare module name, input, and output.
2. Figure 7 Line9-10: If triggered by clock, if RET is 1, export PC as STKO.
3. Figure 7 Line11-12: If triggered by clock, if JUMP is 1, export PC as PC+NUM.
4. Figure 7 Line13-14: If triggered by clock, if RET and JUMP are not 1, export PC as PC+1.
5. Figure 8 Line1-5: Define time scale, module name, registers and wires.
6. Figure 8 Line9-10: Include main file.
7. Figure 8 Line11-17: Set initial value for clk, RET, JUMP, STKO, and NUM.
8. Figure 8 Line20-20: Every 50ns, alternate clk between 0 and 1.
9. Figure 8 Line21-21: Every 100ns, alternate RET between 0 and 1.
10. Figure 8 Line22-22: Every 200ns, alternate JUMP between 0 and 1.

3.4 Result

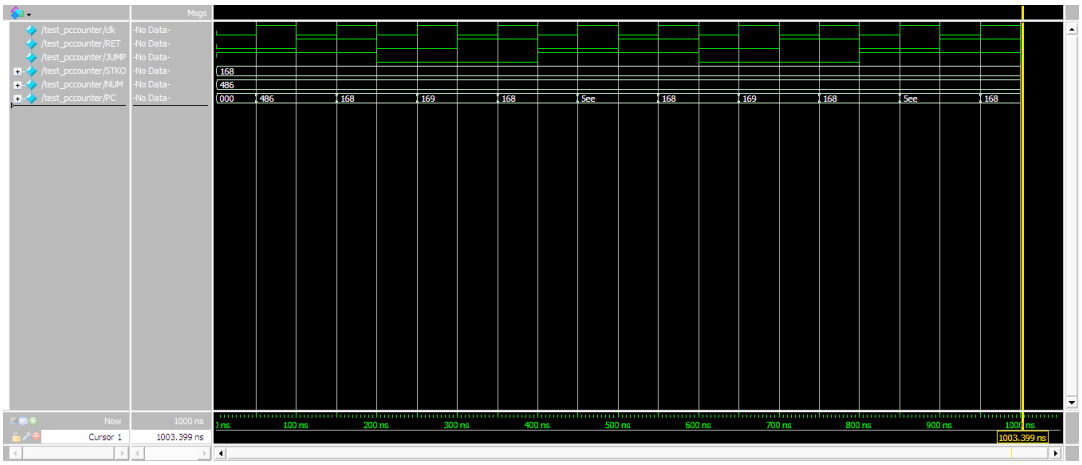

Figure 9: Exercise 11-3 Result

Table 6: Exercise 11-3 Truth Table

clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
RET (1'b)	0	1	0	1	0	1	0	1	0	1
JUMP (1'b)	1	1	0	0	1	1	0	0	1	1
STKO (12'h)	168	168	168	168	168	168	168	168	168	168
NUM (12'h)	486	486	486	486	486	486	486	486	486	486
PC (12'h)	486	168	169	168	5ee	168	169	168	5ee	168
description	RET=0 JUMP=1 PC+=NUM pass	RET=1 PC=STKO pass	RET=0 JUMP=0 PC+=1 pass	RET=1 PC=STKO pass	RET=0 JUMP=1 PC+=NUM pass	RET=1 PC=STKO pass	RET=0 JUMP=0 PC+=1 pass	RET=1 PC=STKO pass	RET=0 JUMP=1 PC+=NUM pass	RET=1 PC=STKO pass

4 Exercise 11-4

4.1 Objective

Create a module that can generate condition code for control system 2.

4.2 Operation

Table 7: Homework 11-4 Operation detail

type	var	operation
input	PROM	program in rom
input	ALU	ALU module output signal
output	tcnd	condition code for control system 2

4.3 Code

```
1   module tcnd_v(PROM, ALU, tcnd);
2   input [3:0] PROM;
3   input [8:0] ALU;
4   output tcnd;
5   reg tcnd;
6   always @(PROM or ALU)
7   begin
8    case (PROM)
9        1 : tcnd = 1'b1;
10       2 : tcnd = 1'b1;
11       3 : tcnd = 1'b1;
12       4 : tcnd = ~|ALU[7:0];
13       5 : tcnd = |ALU[7:0];
14       6 : tcnd = ALU[8];
15       7 : tcnd = !ALU[8];
16     default :
17     begin
18        tcnd = 1'b0;
19     end
20   endcase
21  end
22  endmodule
23
```

Figure 10: Main file

```
1   `timescale 1 ns/1 ns
2   module test_tcnd;
3
4   reg [3:0] PROM;
5   reg [8:0] ALU;
6   wire tcnd;
7
8
9   tcnd_v DUT( .PROM(PROM) , .ALU(ALU), .tcnd(tcnd) );
10  initial
11   begin
12     PROM = 0 ;
13     ALU = 9'h0fa;
14   end
15
16  always #50 PROM = PROM + 1 ;
17  always #50 ALU = ALU +1 ;
18
19  endmodule
20
```

Figure 11: Test file

1. Figure 10 Line1-5: Declare module name, input and output.
2. Figure 10 Line7-20: If either PROM or ALU is triggered, assign tcnd value based on different cases of PROM value.
3. Figure 11 Line1-6: Define time scale, module name, registers and wires.
4. Figure 11 Line9-9: Include main file.
5. Figure 11 Line10-14: Define initial value for PROM and ALU.
6. Figure 11 Line16-16: Every 50ns, add 1 to PROM.
7. Figure 11 Line17-17: Every 50ns, add 1 to ALU.

4.4 Result

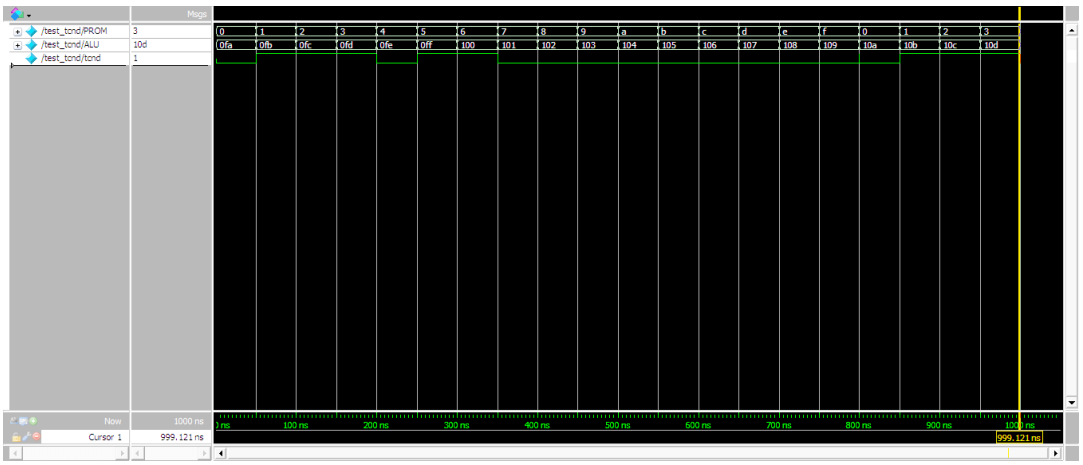

Figure 12: Exercise 11-4 Result

Table 8: Exercise 11-4 Truth Table

clk ↑ (ns)	25	75	125	175	225	275	775	825	875	925
PROM (4'h)	0	1	2	3	4	5	f	0	1	2
ALU (12'h)	0fa	0fb	0fc	0fd	0fe	0ff	109	10a	10b	10c
tcnd (1'b)	0	1	1	1	0	1	0	0	1	1
description	PROM=0 tcnd=0 pass	PROM=1 tcnd=1 pass	PROM=2 tcnd=1 pass	PROM=3 tcnd=1 pass	PROM=4 tcnd=~|ALU[7:0] pass	PROM=5 tcnd=|ALU[7:0] pass	PROM=f tcnd=0 pass	PROM=0 tcnd=0 pass	PROM=1 tcnd=1 pass	PROM=2 tcnd=1 pass

5 Exercise 11-5

5.1 Objective

Code the first part of controller. Ultimatily, the controller can execute base instructions like RET, JUMP, CALL, JZ, JNZ, JC, JNC, Not jump instruction.

5.2 Operation

Table 9: Homework 11-5 Operation detail

type	var	operation
input	clk	clock signal
input	PROM	program in rom
input	JUMP2	
input	JUMP3	
output	JUMP1	
output	RET1	
output	PUSH1	
output	WE1	

5.3 Code

```
1   module controller1_v (clk,PROM,JUMP2,JUMP3,JUMP1,RET1,PUSH1,WE1);
2   input [3:0] PROM;
3   input clk, JUMP2, JUMP3 ;
4   output JUMP1, RET1,PUSH1,WE1;
5   reg JUMP1,RET1,PUSH1,WE1;
6   always @(posedge clk)
7  begin
8       JUMP1=0;
9       RET1=0;
10      PUSH1=0;
11      WE1=0;
12      case (PROM)
13       1 : begin
14             RET1 = ~(JUMP2 | JUMP3 );
15             JUMP1 = ~(JUMP2 | JUMP3 );
16          end
17       2 : begin
18             JUMP1 = ~(JUMP2 | JUMP3 );
19          end
20       3 : begin
21             PUSH1 = ~(JUMP2 | JUMP3 );
22             JUMP1 = ~(JUMP2 | JUMP3 );
23          end
24       4, 5, 6, 7, : JUMP1 = ~( JUMP2 | JUMP3 );
25       8, 9, 10, 11, 12, 13, 14, 15 : WE1 = ~( JUMP2 | JUMP3 );
26       default: begin
27                  JUMP1=0;
28                  RET1=0;
29                  PUSH1=0;
30                  WE1=0;
31                end
32      endcase
33  end
34  endmodule
```

Figure 13: Main file

```
1   `timescale 1 ns/1 ns
2   module test_controller1;
3   reg [3:0] PROM;
4   reg clk, JUMP2, JUMP3 ;
5   wire JUMP1,RET1,PUSH1,WE1;
6
7
8
9   controller1_v DUT( .PROM(PROM) , .clk(clk), .JUMP2(JUMP2), .JUMP3(JUMP3),
10                     .JUMP1(JUMP1),.RET1(RET1), .PUSH1(PUSH1), .WE1(WE1) );
11  initial
12  begin
13     PROM = 0 ;
14     clk = 0;
15     JUMP2 = 0 ;
16     JUMP3 = 0 ;
17  end
18  always #50 clk = ~clk ;
19  always #100 PROM = PROM + 1 ;
20  always #100 JUMP2 = ~JUMP2 ;
21  always #100 JUMP3 = ~JUMP3 ;
22
23  endmodule
```

Figure 14: Test file

1. Figure 13 Line1-5: Declare module name, input and output.
2. Figure 13 Line8-11: If triggered by clock, reset JUMP1, RET1, PUSH1 and WE1.
3. Figure 13 Line13-25: If triggered by clock, assign RET1, PUSH1, JUMP1, and WE1 with corresponding value in truth table.
4. Figure 14 Line1-5: Define time scale, module name, registers, and wires.
5. Figure 14 Line9-9: Include main file.
6. Figure 14 Line11-17: Set initial value for registers.
7. Figure 14 Line18-18: Every 50ns, alternate clock signal.

8. Figure 14 Line19-19: Every 100ns, add PROM value by 1.
9. Figure 14 Line20-20: Every 100ns, alternate JUMP2 signal.
10. Figure 14 Line21-21: Every 100ns, alternate JUMP3 signal.

5.4 Result

Figure 15: Exercise 11-5 Result

Table 10: Exercise 11-5 Truth Table

clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
PROM (4'b)	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
JUMP2 (1'b)	0	1	0	1	0	1	0	1	0	1
JUMP3 (1'b)	0	1	0	1	0	1	0	1	0	1
JUMP1 (1'b)	0	0	1	0	1	0	1	0	0	0
RET1 (1'b)	0	0	0	0	0	0	0	0	0	0
PUSH1 (1'b)	0	0	0	0	0	0	0	0	0	0
WE1 (1'b)	0	0	0	0	0	0	0	0	1	0
description	PROM=0 set to zero pass	PROM=1 RET1= ~(JUMP2|JUMP3) JUMP1= ~(JUMP2|JUMP3) pass	PROM=2 JUMP1= ~(JUMP2|JUMP3) pass	PROM=3 PUSH1= ~(JUMP2|JUMP3) JUMP1= ~(JUMP2|JUMP3) pass	PROM=4 JUMP1= ~(JUMP2|JUMP3) pass	PROM=5 JUMP1= ~(JUMP2|JUMP3) pass	PROM=6 JUMP1= ~(JUMP2|JUMP3) pass	PROM=7 JUMP1= ~(JUMP2|JUMP3) pass	PROM=8 WE1= ~(JUMP2|JUMP3) pass	PROM=9 WE1= ~(JUMP2|JUMP3) pass

6 Exercise 11-6

6.1 Objective

Create second part of the controller.

6.2 Operation

Table 11: Homework 11-6 Operation detail

type	var	operation
input	clk	clock signal
input	PROM	program in rom
input	tcnd	condition value from ex11-4
output	JUMP1	
output	JUMP2	
output	JUMP3	
output	TEMPRET	
output	RET1	
output	PUSH1	
output	PUSH2	
output	WE1	
output	WE2	

6.3 Code

```
1   module ctr2_v(clk, PROM,tcnd, JUMP1,JUMP2,JUMP3, TEMPRET,RET1, PUSH1,PUSH2,WE1, WE2 );
2   input [3:0] PROM;
3   input clk, tcnd;
4   output JUMP1,JUMP2, JUMP3,TEMPRET,RET1, PUSH1, PUSH2, WE1, WE2;
5
6   reg JUMP2,JUMP3, PUSH2, WE2 ;
7
8   controller1_v inst (.clk(clk), .JUMP2(JUMP2), .JUMP3(JUMP3),
9                       .PROM(PROM), .JUMP1(JUMP1), .RET1(TEMPRET),
10                      .PUSH1(PUSH1),.WE1(WE1));
11
12  assign RET1 = ~JUMP2 & ~JUMP3 & TEMPRET;
13
14
15  always @(posedge clk)
16  begin
17   JUMP2 <= ~JUMP2 & ~JUMP3 & tcnd & JUMP1;
18   JUMP3 <= JUMP2;
19   PUSH2 <= ~JUMP2 & ~JUMP3 & PUSH1;
20   WE2   <= ~JUMP2 & ~JUMP3 & WE1;
21  end
22  endmodule
```

Figure 16: Main file

```
1   `timescale 1 ns/1 ns
2   module test_ctr2;
3
4   reg [3:0] PROM;
5   reg clk, tcnd;
6   wire JUMP1,JUMP2, JUMP3,TEMPRET,RET1, PUSH1, PUSH2, WE1, WE2;
7
8
9
10  ctr2_v DUT( .clk(clk),.PROM(PROM),.tcnd(tcnd),
11              .JUMP1(JUMP1),.JUMP2(JUMP2),.JUMP3(JUMP3),
12              .TEMPRET(TEMPRET),.RET1(RET1),
13              .PUSH1(PUSH1), .PUSH2(PUSH2),.WE1(WE1), .WE2(WE2));
14  initial
15   begin
16    PROM = 0 ;
17    clk  = 0;
18    tcnd = 1 ;
19   end
20
21  always #50 clk = ~clk ;
22  always #100 PROM = PROM + 1 ;
23
24  endmodule
```

Figure 17: Test file

1. Figure 16 Line1-6: Declare module name, input and output.
2. Figure 16 Line8-10: Include first part of controller.
3. Figure 16 Line15-21: If triggered by clock, assign corresponding value to JUMP2, JUMP3, PUSH2, WE2 from truth table.
4. Figure 17 Line1-6: Define time scale, module name, registers and wires.
5. Figure 17 Line10-13: Include main file.
6. Figure 17 Line14-19: Set initial value of PROM, clk, and tcnd.
7. Figure 17 Line21-21: Every 50ns, alternate clock value.
8. Figure 17 Line22-22: Every 100ns, add 1 to PROM.

6.4 Result

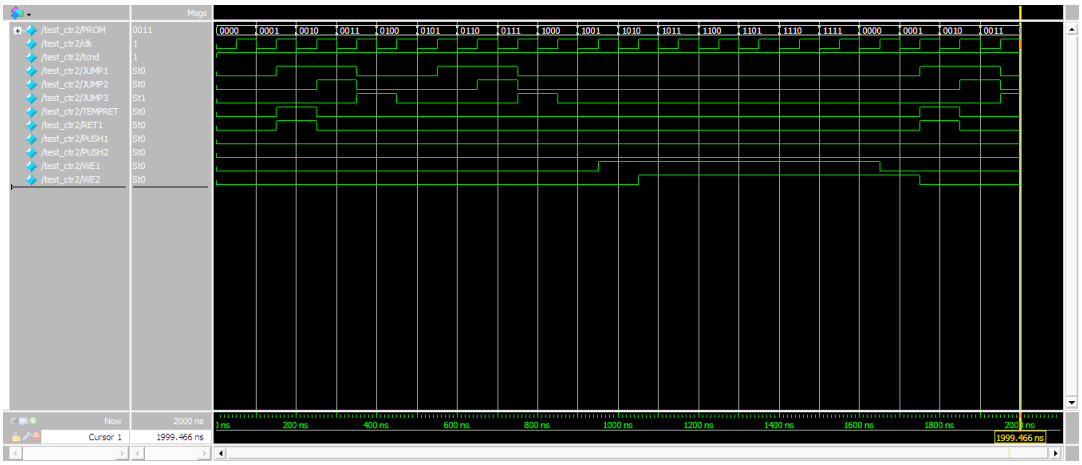

Figure 18: Exercise 11-6 Result

Table 12: Exercise 11-6 Truth Table

clk ↑ (ns)	50	150	250	350	450	550	650	750	850	950
PROM (4'b)	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
tcnd (1'b)	1	1	1	1	1	1	1	1	1	1
JUMP1 (1'b)	0	1	1	0	0	1	1	0	0	0
JUMP2 (1'b)	0	0	1	0	0	0	1	0	0	0
JUMP3 (1'b)	0	0	0	1	0	0	0	1	0	0
TEMPRET (1'b)	0	1	0	0	0	0	0	0	0	0
RET1 (1'b)	0	1	0	0	0	0	0	0	0	0
PUSH1 (1'b)	0	0	0	0	0	0	0	0	0	0
PUSH2 (1'b)	0	0	0	0	0	0	0	0	0	0
WE1 (1'b)	0	0	0	0	0	0	0	0	0	1
WE2 (1'b)	0	0	0	0	0	0	0	0	0	0
description	JUMP2=~JUMP2& ~JUMP3&tcnd&JUMP1 JUMP3=JUMP2 PUSH2=~JUMP2&~JUMP3&PUSH1 WE2=~JUMP2&~JUMP3&WE1									