多周期 MIPS CPU 设计
实验报告

蔡子凯 PB15111676

一、 实验目的

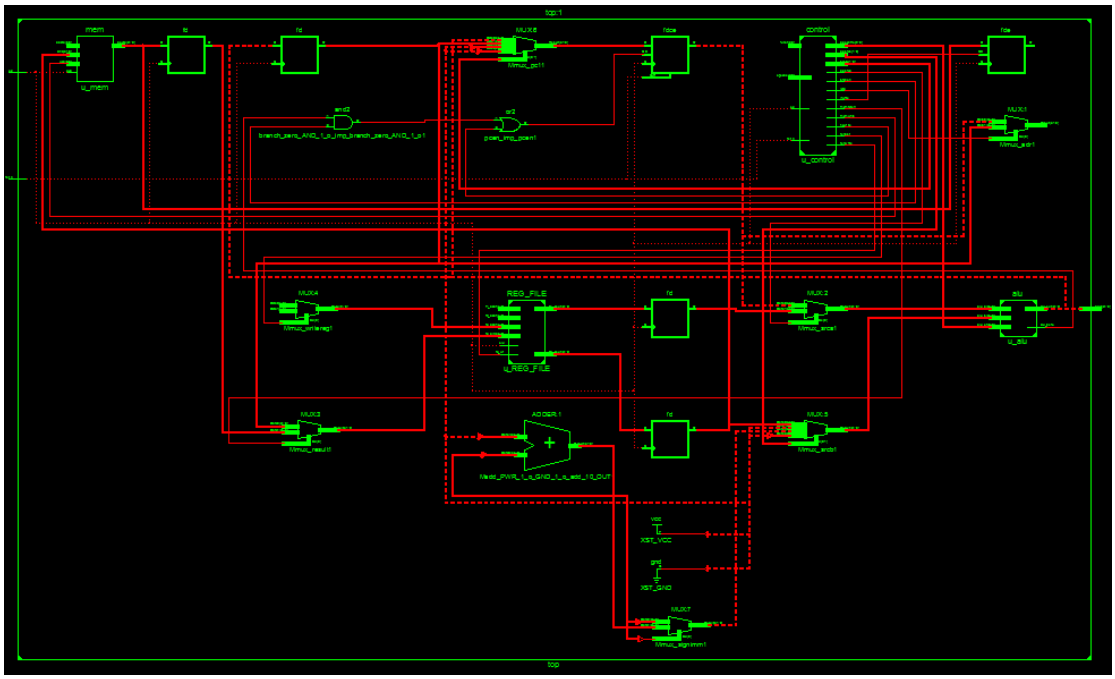
1. 多周期 MIPS CPU 的设计。

二、 实验内容

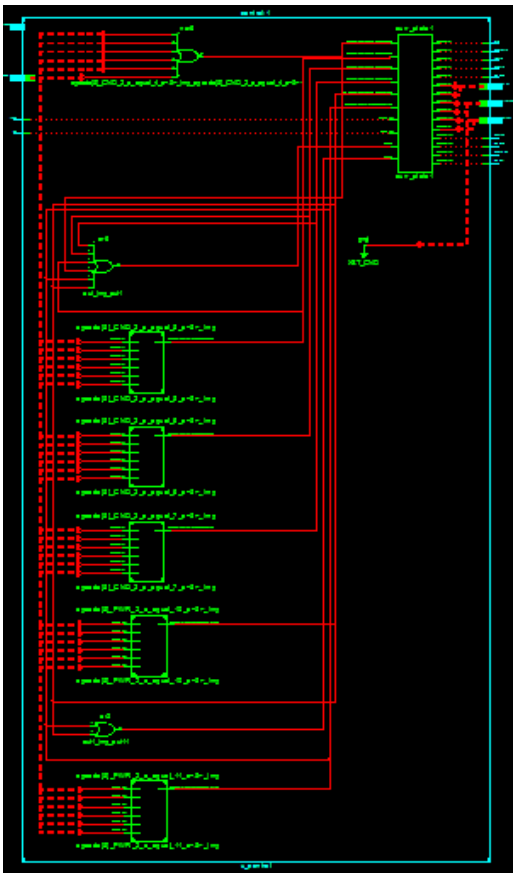
1. 设计 CPU，完成指定程序代码的执行，其功能是起始数为 3 和 3 的斐波拉契数列的计算。只计算 20 个数。
2. 实验设计中可以不使用给定的数据通路和状态机，但仅允许使用一个存储器。
3. 对指令/数据存储器的附加要求：使用同步存储器，对数据通路和状态机进行适当修改。

三、 实验结果

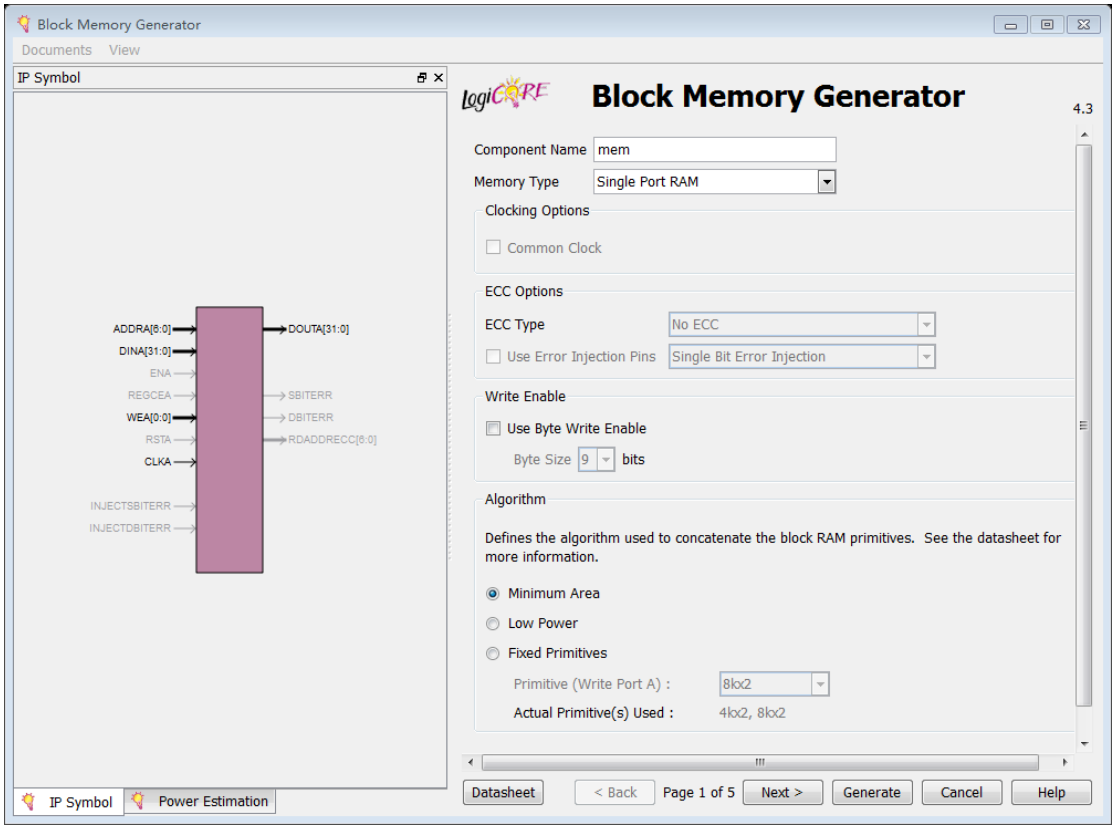
top 模块原理图：

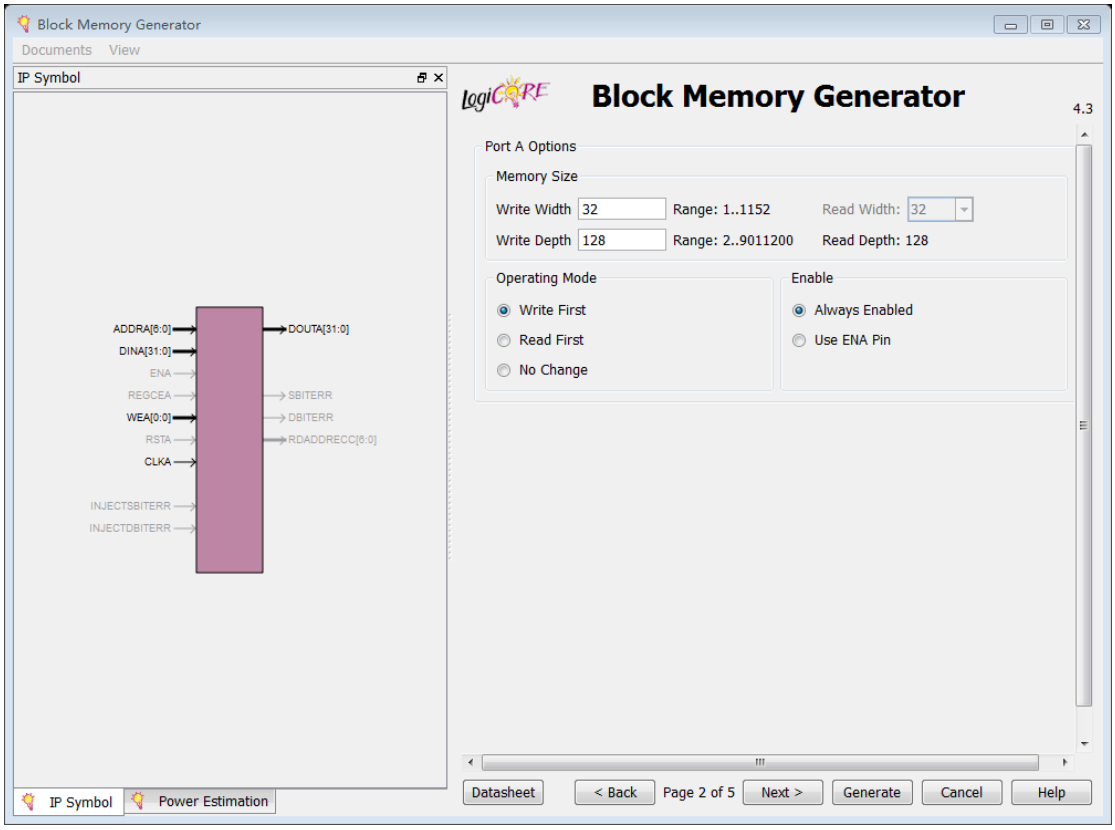


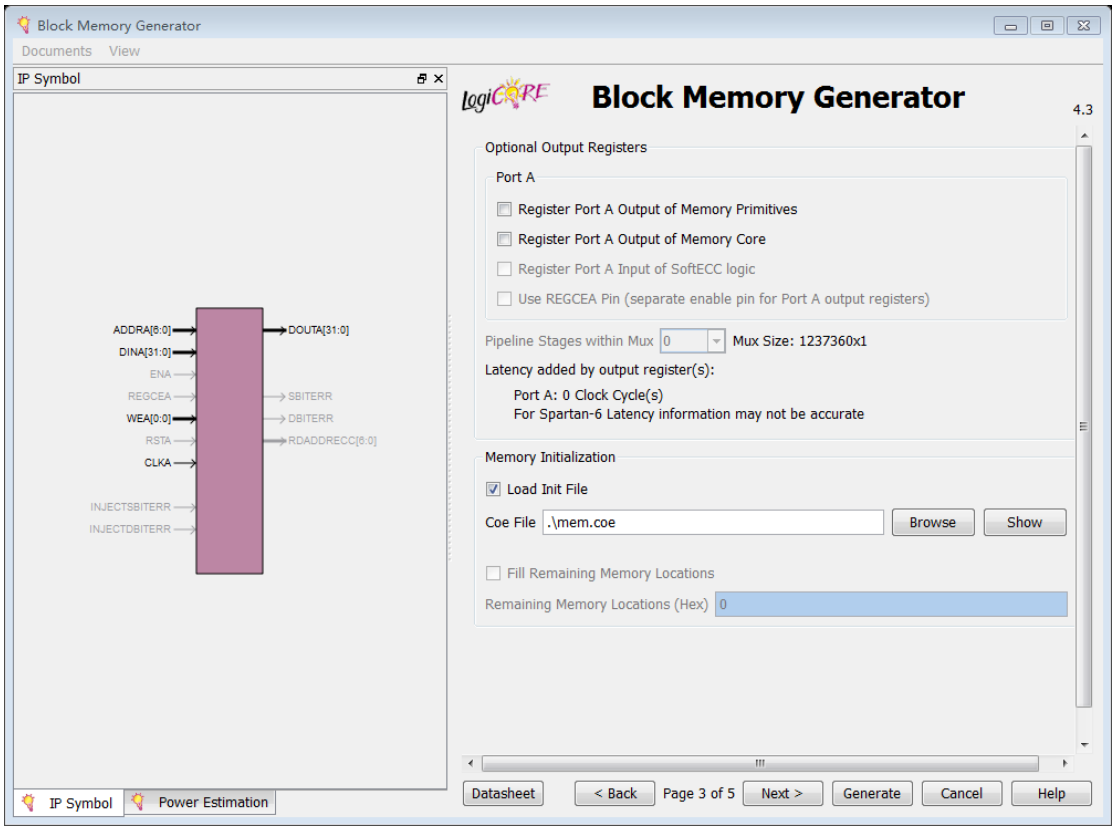
control 模块原理图：

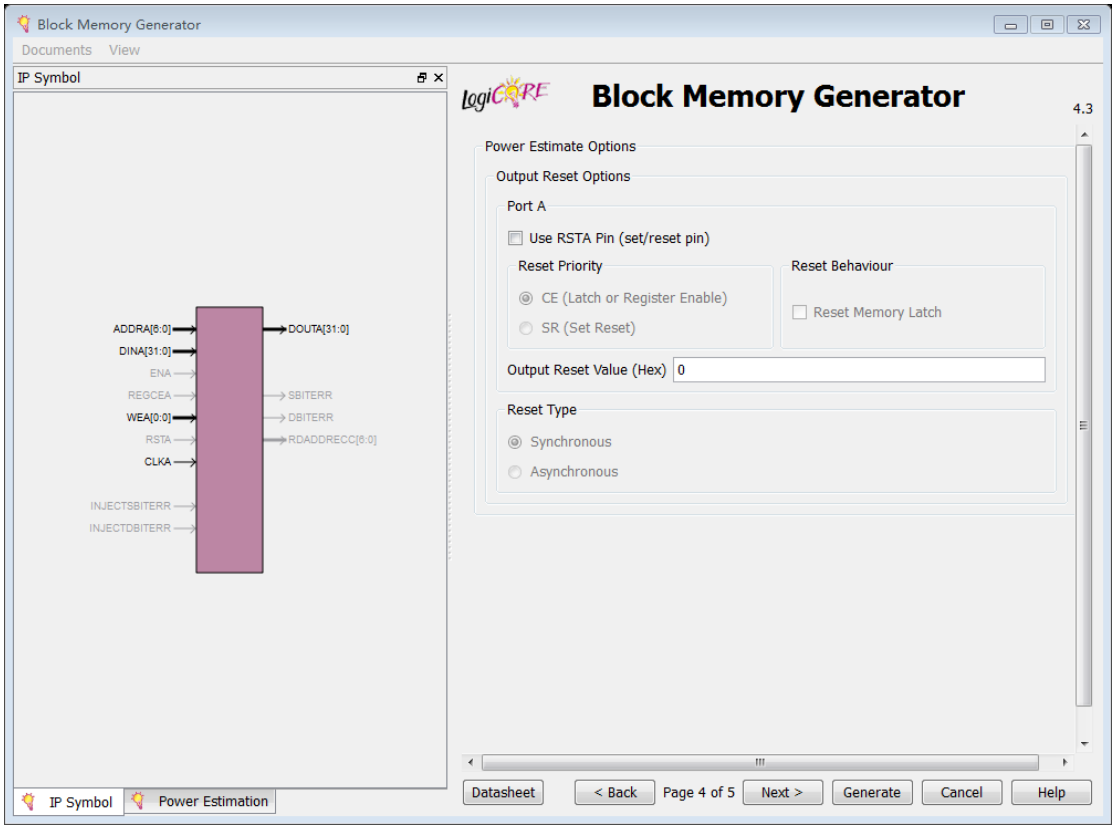


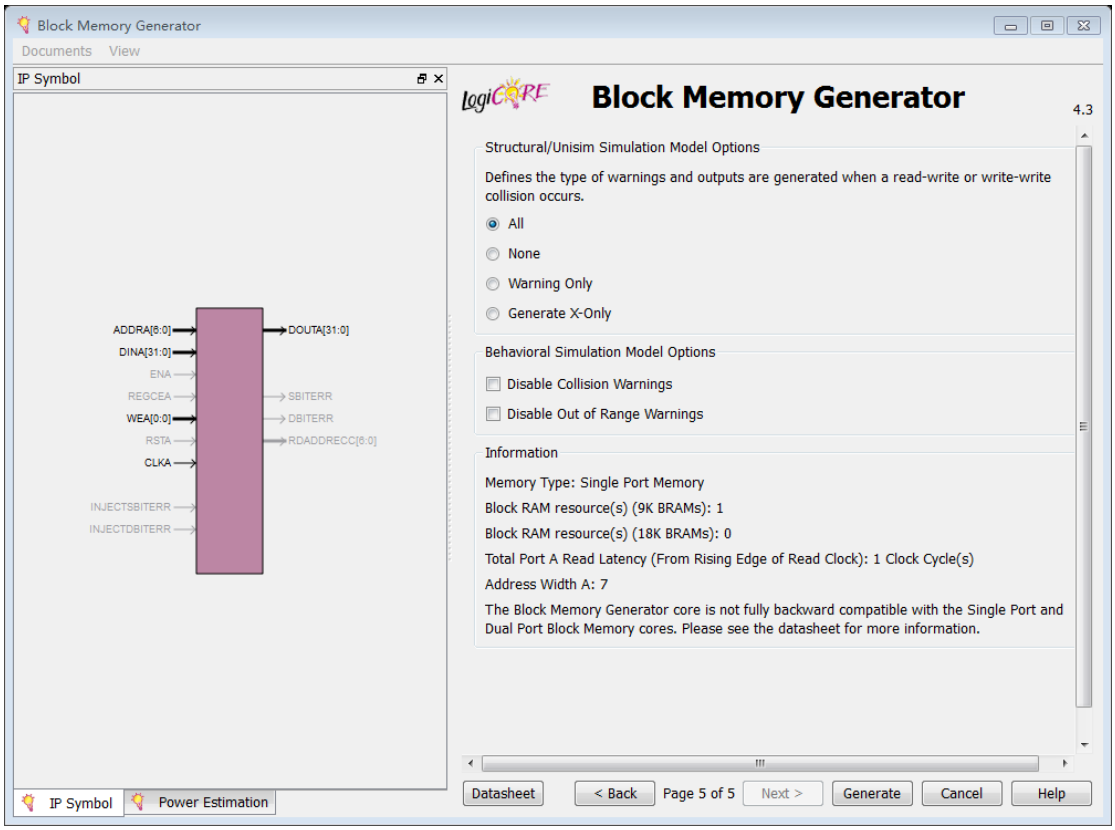
mem 初始化：




Block Memory Generator
Documents View
IP Symbol
ADDRA[6:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[6:0]
IP Symbol
Power Estimation
LogiCORE
Block Memory Generator
4.3
Port A Options
Memory Size
Write Width 32
Range: 1..1152
Read Width: 32
Write Depth 128
Range: 2..9011200
Read Depth: 128
Operating Mode
Write First
Read First
No Change
Enable
Always Enabled
Use ENA Pin
Datasheet
< Back
Page 2 of 5
Next >
Generate
Cancel
Help


Block Memory Generator
Documents View
IP Symbol
ADDRA[6:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[6:0]
IP Symbol
Power Estimation
LogiCORE
Block Memory Generator
4.3
Optional Output Registers
Port A
Register Port A Output of Memory Primitives
Register Port A Output of Memory Core
Register Port A Input of SoftECC logic
Use REGCEA Pin (separate enable pin for Port A output registers)
Pipeline Stages within Mux 0
Mux Size: 1237360x1
Latency added by output register(s):
Port A: 0 Clock Cycle(s)
For Spartan-6 Latency information may not be accurate
Memory Initialization
Load Init File
Coe File .\mem.coe
Browse
Show
Fill Remaining Memory Locations
Remaining Memory Locations (Hex) 0
Datasheet
< Back
Page 3 of 5
Next >
Generate
Cancel
Help


Block Memory Generator
Documents View
IP Symbol
ADDRA[6:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[6:0]
IP Symbol
Power Estimation
LogiCORE
Block Memory Generator
4.3
Power Estimate Options
Output Reset Options
Port A
Use RSTA Pin (set/reset pin)
Reset Priority
CE (Latch or Register Enable)
SR (Set Reset)
Reset Behaviour
Reset Memory Latch
Output Reset Value (Hex)
0
Reset Type
Synchronous
Asynchronous
Datasheet
< Back
Page 4 of 5
Next >
Generate
Cancel
Help


Block Memory Generator
Documents View
IP Symbol
ADDRA[6:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[6:0]
IP Symbol
Power Estimation
LogiCORE
Block Memory Generator
4.3
Structural/Unisim Simulation Model Options
Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.
All
None
Warning Only
Generate X-Only
Behavioral Simulation Model Options
Disable Collision Warnings
Disable Out of Range Warnings
Information
Memory Type: Single Port Memory
Block RAM resource(s) (9K BRAMs): 1
Block RAM resource(s) (18K BRAMs): 0
Total Port A Read Latency (From Rising Edge of Read Clock): 1 Clock Cycle(s)
Address Width A: 7
The Block Memory Generator core is not fully backward compatible with the Single Port and Dual Port Block Memory cores. Please see the datasheet for more information.
Datasheet
< Back
Page 5 of 5
Next >
Generate
Cancel
Help

	0	1	2	3	4	5	6	7
0x0	20080100	200D0150	8DAD0000	200B0154	8D6B0000	200C0154	8D8C0004	AD0B0000
0x8	AD0C0004	21A9FFFE	8D0B0000	8D0C0004	016C5020	AD0A0008	21080004	2129FFFF
0x10	1D20FFF9	08000011	00000000	00000000	00000000	00000000	00000000	00000000
0x18	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x20	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x28	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x30	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x38	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x40	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x48	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x50	00000000	00000000	00000000	00000000	00000014	00000003	00000003	00000000
0x58	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x60	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x68	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x70	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0x78	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

仿真结果：

mem 结果：

	0	1	2	3
0	3	3	6	9
4	15	24	39	63
8	102	165	267	432
12	699	1131	1830	2961
16	4791	7752	12543	20295
20	20	3	3	0

四、 实验分析

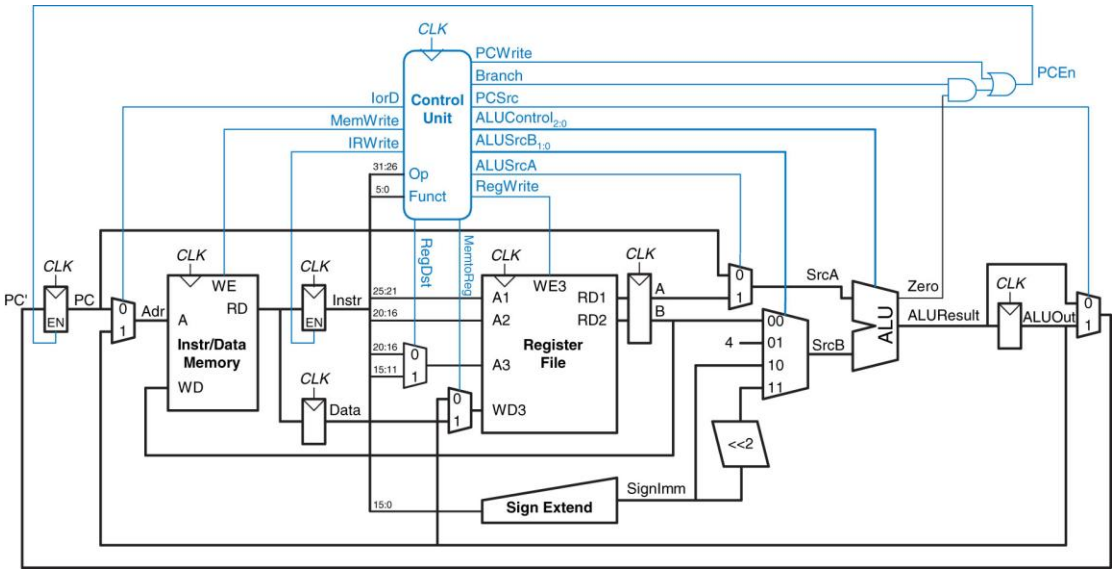
本实验无需下载，只需要进行仿真。

在 alu 模块中，首先输入两个带符号的 32 位（alu_a 与 alu_b）以及一个操作数（alu_op），通过一个 case 语句将两个输入进行对应的运算，然后输出（alu_out）。同时，在本实验中为了实现 bgtz 命令，外加一个 bgtz 接口，链接一组合逻辑电路，在 alu_out 大于 0 时将 bgtz 置为 1，从而实现跳转功能。

在 REG_FILE 模块中，按照提示的接口，先声明 32*32 大小的空间存放数据，然后使用三个 always，第一个用于

写操作，当 r3_wr 有效时将 r3_addr 处的数据更改为 r3_din；第二，三个用于对输出的赋值，分别将 r1_addr，r2_addr 处的数据输出到 r1_dout，r2_dout 处。

在 control 模块中，设计一个简单的状态机，按照下图进行接口的连接，根据 op 和 funct 进行状态的转换，分别给多个输出赋值，以实现各个指令的功能。



注意：此图片省略了 jump 电路，具体参考代码

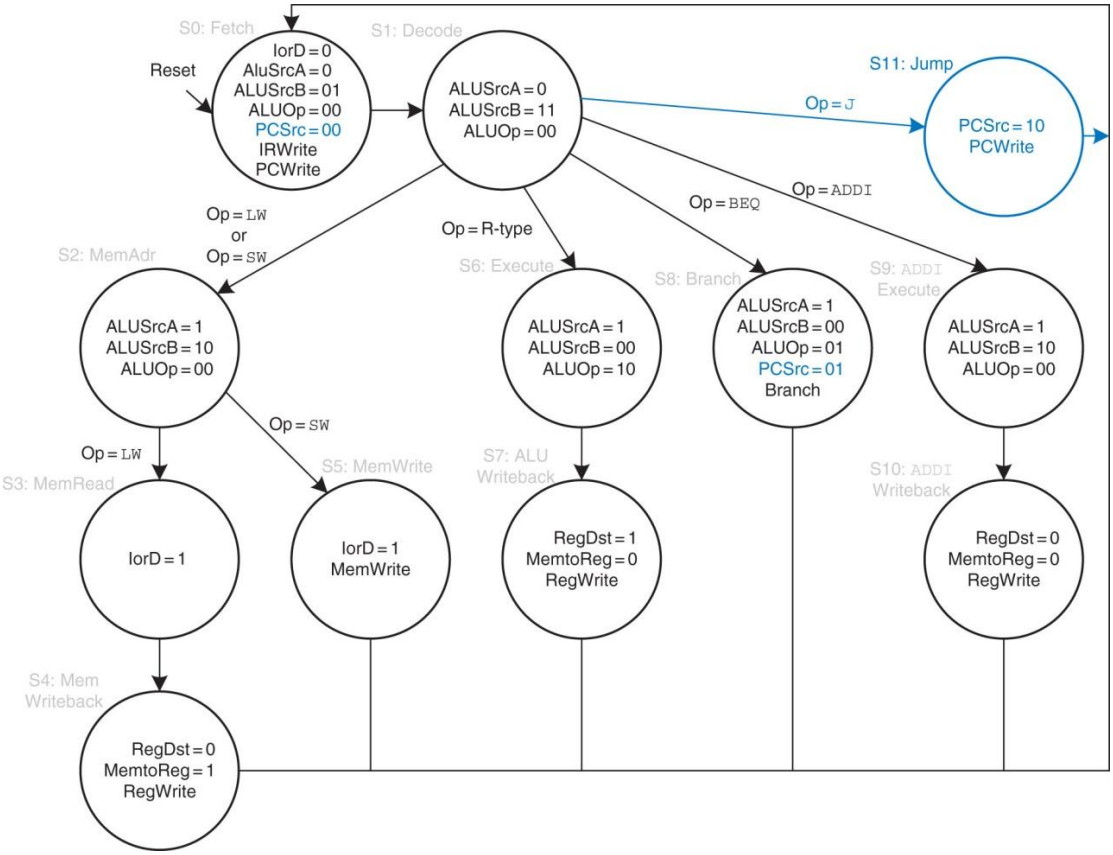
状态机如下图所示，在状态机的设计中，第一段对 rst_n 进行判断，若有效则将当前状态初始化，否则根据 clk 的上升沿跳入下一个状态。第二段设计一组合逻辑电路根据输入与当前状态更改下一状态。第三段设计一个组合逻辑电路根据当前状态给各个输出变量赋值。值得注意的是，由于同步读取的时延关系在多个状态后添加了等待读取状态，其中包括：

3,4 状态间 1 个等待周期。

5 状态后 2 个等待周期。

8 状态后 1 个等待周期。

11 状态后 1 个等待周期。



注意：此图片省略了等待状态，具体参考代码

在 top 模块中，将各个模块以及线网连接，同时使用多个 assign 或 always 语句将外部组合逻辑电路补全。

另外，在对储存器赋值时，将代码段放置在 0 地址开始，数据段从 0x40 开始（mars 中给出的地址过大，不利于仿真），所以对照代码如下。

20082000 20080100

200d2050 200d0150

8dad0000 8dad0000

200b2054 200b0154

8d6b0000 8d6b0000

200c2054 200c0154

8d8c0004 8d8c0004

ad0b0000 ad0b0000

ad0c0004 ad0c0004

21a9fffe 21a9fffe

8d0b0000 8d0b0000

8d0c0004 8d0c0004

016c5020 016c5020

ad0a0008 ad0a0008

21080004 21080004

2129ffff 2129ffff

1d20fff9 1d20fff9

08000011 08000011

仿真时无输出结果，通过直接查看数据观察结果。

五、 意见建议

无。

六、 附录

Verilog 实现代码

top.v

```
module top(
input          clk,
input          rst_n,
```

```
output [31:0]  dout
    );

reg  [31:0]  pc1;
reg  [31:0]  pc;
wire [31:0]  pcjump;
wire [31:0]  adr;
wire [31:0]  rd;
reg  [31:0]  instr;
reg  [31:0]  data;
wire         iord;
wire         memwrite;
wire         irwrite;
wire         pcwrite;
wire         branch;
wire [1:0]   pcsrc;
wire [2:0]   alucontrol;
wire [1:0]   alusrcb;
wire         alusrca;
wire         regwrite;
wire         memtoreg;
wire         regdst;
wire [31:0]  rd1;
```

```
wire [31:0]  rd2;
reg  [31:0]  a;
reg  [31:0]  b;
wire [31:0]  result;
wire [4:0]   writereg;
reg  [31:0]  signimm;
wire [31:0]  signimml2;
wire [31:0]  srca;
reg  [31:0]  srcb;
wire         zero;
wire [31:0]  aluresult;
reg  [31:0]  aluout;

always@(posedge clk or posedge rst_n)
begin
  if(rst_n)
     pc <= 32'b0;
  else if(pcen)
     pc <= pc1;
end

assign adr  =  iord ?  aluout  :  pc;
```

```
always@(posedge clk)
begin
  if(irwrite)
     instr<= rd;
end

always@(posedge clk)
begin
     data <= rd;
end

assign writereg  =  regdst  ?  instr[15:11]
:  instr[20:16];

assign result  =  memtoreg  ?  data :
aluout;

always@(*)
begin
  if(instr[15])
     signimm =  32'hffff0000 +  instr[15:0];
  else
     signimm =  32'h00000000 +  instr[15:0];
```

```
end

assign signimml2[31:2]=  signimm[29:0];
assign signimml2[1:0]  =  2'h0;

always@(posedge clk)
begin
     a  <= rd1;
end

always@(posedge clk)
begin
     b  <= rd2;
end

assign srca =  alusrca ?  a  :  pc;

always@(*)
begin
  case(alusrcb)
     2'b00:     srcb  =  b;
     2'b01:     srcb  =  32'h4;
     2'b10:     srcb  =  signimm;
```

```
    2'b11:     srcb =  signimml2;
  endcase
end

assign pcjump[31:28]=  pc[31:28];
assign pcjump[27:2] =  instr[25:0];
assign pcjump[1:0]     =  2'h0;

assign pcen =  pcwrite |  (branch &  zero);

always@(posedge clk)
begin
    aluout  <= aluresult;
end

always@(*)
begin
  case(pcsrc)
    2'b00:     pc1  =  aluresult;
    2'b01:     pc1  =  aluout;
    2'b10:     pc1  =  pcjump;
    default:     pc1  =  32'h0;
  endcase
```

```
end

assign dout         = aluresult;

control   u_control(
.opcode   (instr[31:26]   ),
.funct    (instr[5:0]       ),
.clk      (clk           ),
.rst_n    (rst_n         ),
.iord        (iord          ),
.memwrite (memwrite      ),
.irwrite     (irwrite          ),
.pcwrite     (pcwrite          ),
.branch   (branch        ),
.pcsrc    (pcsrc         ),
.alucontrol  (alucontrol[2:0]  ),
.alusrcb     (alusrcb[1:0]   ),
.alusrca     (alusrca          ),
.regwrite (regwrite      ),
.memtoreg (memtoreg      ),
.regdst   (regdst        )
);
```

```
mem       u_mem(
.clka        (clk         ),
.wea      (memwrite    ),
.addra    (adr[8:2]    ),
.dina        (b[31:0]        ),
.douta    (rd          )
);

REG_FILE     u_REG_FILE(
.clk      (clk         ),
.r1_addr     (instr[25:21]),
.r2_addr     (instr[20:16]),
.r3_addr     (writereg    ),
.r3_din   (result      ),
.r3_wr    (regwrite    ),
.r1_dout     (rd1         ),
.r2_dout     (rd2         )
);

alu       u_alu(
.alu_a    (srca        ),
.alu_b    (srcb        ),
.alu_op   (alucontrol     ),
```

```
    .alu_out     (aluresult   ),
    .alu_zero (zero        )
    );

endmodule
```

control.v

```
module control(
input         [5:0]   opcode,
input         [5:0]   funct,
input                 clk,
input                 rst_n,
outputreg           iord,
outputreg           memwrite,
outputreg           irwrite,
outputreg           pcwrite,
outputreg           branch,
outputreg  [1:0]    pcsrc,
outputreg  [2:0]    alucontrol,
outputreg  [1:0]    alusrcb,
outputreg           alusrca,
outputreg           regwrite,
outputreg           memtoreg,
```

```
outputreg          regdst
    );

reg [3:0] curr_state;
reg [3:0] next_state;

always@(posedge clk or posedge rst_n)
begin
 if(rst_n)
    curr_state <= 4'h0;
 else
    curr_state <= next_state;
end

always@(*)
begin
 case(curr_state)
    4'd0: next_state =  4'd1;
    4'd1: begin
            case(opcode)
               6'b100011: next_state =  4'd2;
               6'b101011: next_state =  4'd2;
               6'b000000: next_state =  4'd6;
```

```
            6'b000111: next_state =  4'd8;
            6'b001000: next_state =  4'd9;
            6'b000010: next_state =  4'd11;
            default:   next_state =  4'd0;
         endcase
      end
4'd2: begin
         case(opcode)
            6'b100011: next_state =  4'd3;
            6'b101011: next_state =  4'd5;
            default:   next_state =  4'd0;
         endcase
      end
4'd3: next_state =  4'd12;
4'd4: next_state =  4'd0;
4'd5: next_state =  4'd14;
4'd6: next_state =  4'd7;
4'd7: next_state =  4'd0;
4'd8: next_state =  4'd15;
4'd9: next_state =  4'd10;
4'd10:  next_state =  4'd0;
4'd11:  next_state =  4'd15;
4'd12:  next_state =  4'd4;
```

```
    4'd13:  next_state =  4'd14;
    4'd14:  next_state =  4'd15;
    4'd15:  next_state =  4'd0;
    default:   next_state =  4'd0;
 endcase
end

always@(*)
begin
 case(curr_state)
    4'd0:begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b1;
            pcwrite=1'b1;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
            alusrcb=2'b01;
            alusrca=1'b0;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
```

```
        end
    4'd1:begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
            alusrcb=2'b11;
            alusrca=1'b0;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
        end
    4'd2:begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
```

```
            alusrcb=2'b10;
            alusrca=1'b1;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
        end
    4'd3:begin
            iord=1'b1;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
            alusrcb=2'b0;
            alusrca=1'b0;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
        end
    4'd4:begin
            iord=1'b0;
            memwrite=1'b0;
```

```
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
            alusrcb=2'b0;
            alusrca=1'b0;
            regwrite=1'b1;
            memtoreg=1'b1;
            regdst=1'b0;
        end
    4'd5:begin
            iord=1'b1;
            memwrite=1'b1;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
            alusrcb=2'b0;
            alusrca=1'b0;
            regwrite=1'b0;
            memtoreg=1'b0;
```

```
            regdst=1'b0;
        end
    4'd6:begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
            alusrcb=2'b0;
            alusrca=1'b1;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
        end
    4'd7:begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
```

```
            alucontrol=3'b001;
            alusrcb=2'b0;
            alusrca=1'b0;
            regwrite=1'b1;
            memtoreg=1'b0;
            regdst=1'b1;
        end
    4'd8:begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b1;
            pcsrc=2'b1;
            alucontrol=3'b100;
            alusrcb=2'b0;
            alusrca=1'b1;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
        end
    4'd9:begin
            iord=1'b0;
```

```
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
            alusrcb=2'b10;
            alusrca=1'b1;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
        end
    4'd10:  begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
            pcsrc=2'b0;
            alucontrol=3'b001;
            alusrcb=2'b0;
            alusrca=1'b0;
            regwrite=1'b1;
```

```
            memtoreg=1'b0;
            regdst=1'b0;
        end
    4'd11:  begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b1;
            branch=1'b0;
            pcsrc=2'b10;
            alucontrol=3'b001;
            alusrcb=2'b0;
            alusrca=1'b0;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
        end
    default:  begin
            iord=1'b0;
            memwrite=1'b0;
            irwrite=1'b0;
            pcwrite=1'b0;
            branch=1'b0;
```

```
            pcsrc=2'b10;
            alucontrol=3'b0;
            alusrcb=2'b0;
            alusrca=1'b0;
            regwrite=1'b0;
            memtoreg=1'b0;
            regdst=1'b0;
         end
 endcase
end

endmodule
```

REG_FILE.v

```
module REG_FILE(
input                  clk,
input         [4:0]    r1_addr,
input         [4:0]    r2_addr,
input         [4:0]    r3_addr,
input         [31:0]   r3_din,
input                  r3_wr,
outputreg  [31:0]  r1_dout,
outputreg  [31:0]  r2_dout
```

```
    );

reg            [31:0]  data [31:0];

always@(posedge  clk)
begin
 if(r3_wr)
    data[r3_addr]<= r3_din;
end

always@(*)
begin
 if(r1_addr)
    r1_dout =  data[r1_addr];
 else
    r1_dout =  32'h0;
end


always@(*)
begin
 if(r2_addr)
    r2_dout =  data[r2_addr];
```

```
 else
    r2_dout =  32'h0;
end

endmodule
```

alu.v

```
module alu(
input signed  [31:0]  alu_a,
input signed  [31:0]  alu_b,
input         [2:0]   alu_op,
outputreg  [31:0]  alu_out,
outputreg          alu_bgtz
);

always@(*)
begin
 case(alu_op)
    3'h00:     alu_out =  32'h0;
    3'h01:     alu_out =  alu_a+  alu_b;
    3'h02:     alu_out =  alu_a-  alu_b;
    3'h03:     alu_out =  alu_a&  alu_b;
    3'h04:     alu_out =  alu_a|  alu_b;
```

```
    3'h05:     alu_out = alu_a^ alu_b;
    3'h06:     alu_out = ~(alu_a | alu_b);
    default:     alu_out = 32'h0;
 endcase
end

always@(*) //for bgtz
begin
 if(alu_out <= 0)
    alu_bgtz  = 1'h0;
 else
    alu_bgtz  = 1'h1;
end

endmodule
```

mem.coe

```
memory_initialization_radix = 16;
memory_initialization_vector = 20080100
200d0150
8dad0000
200b0154
8d6b0000
```

200c0154

8d8c0004

ad0b0000

ad0c0004

21a9fffe

8d0b0000

8d0c0004

016c5020

ad0a0008

21080004

2129ffff

1d20fff9

08000011

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

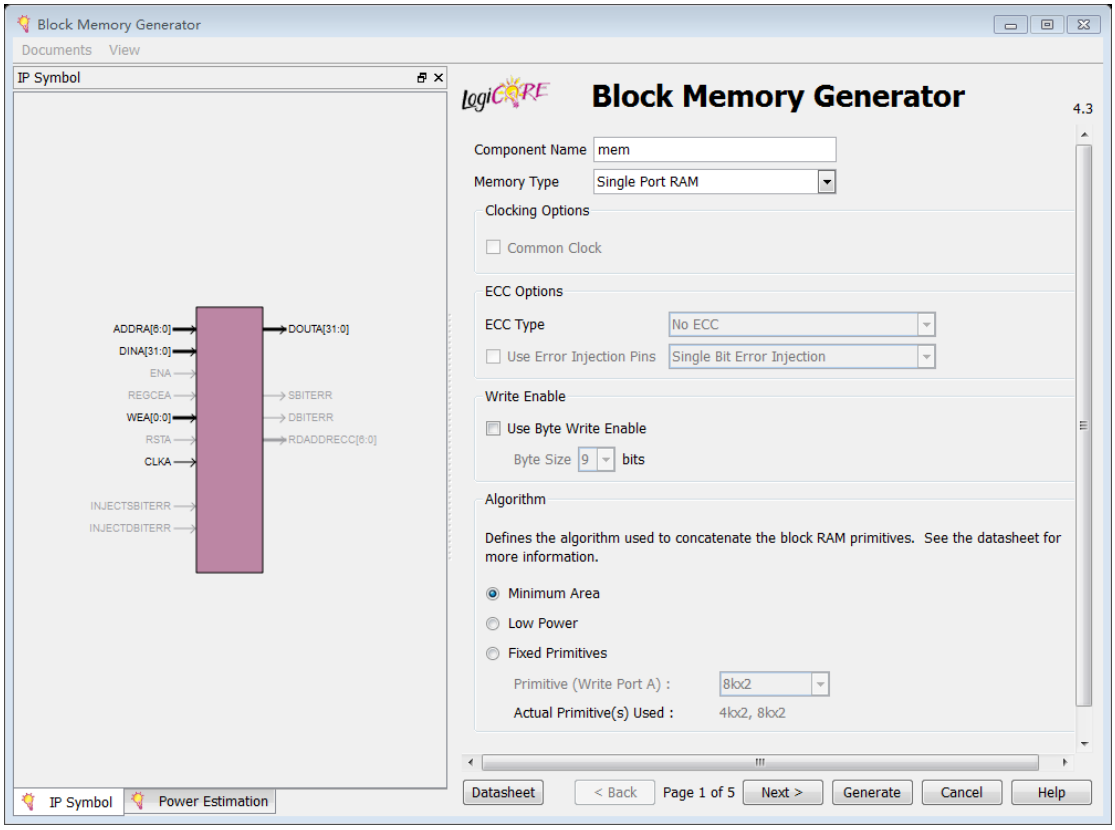
00000014

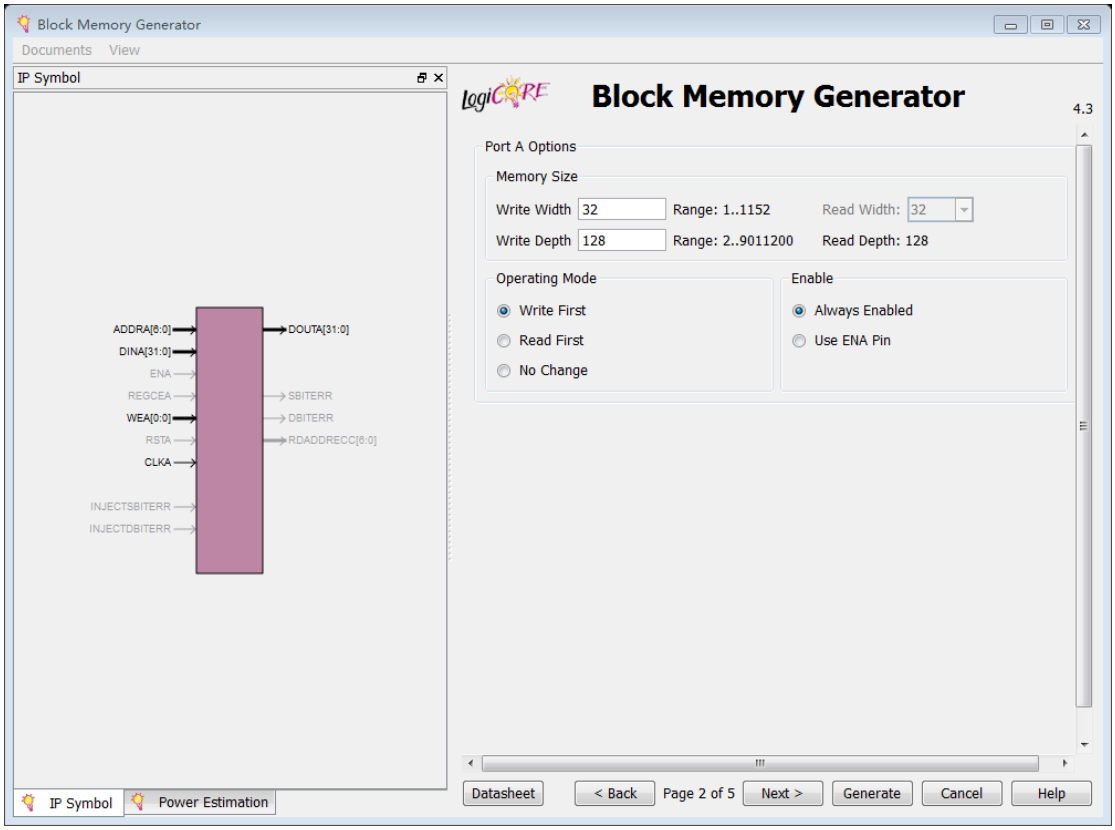
00000003

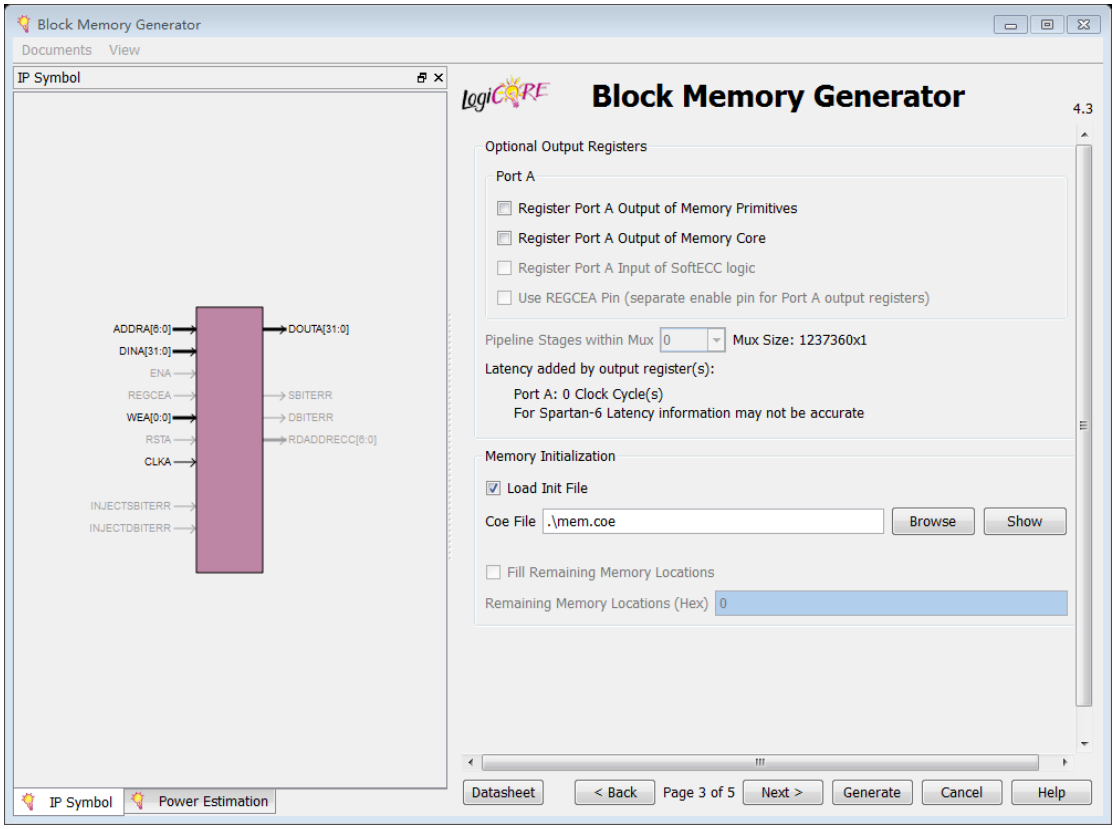
00000003

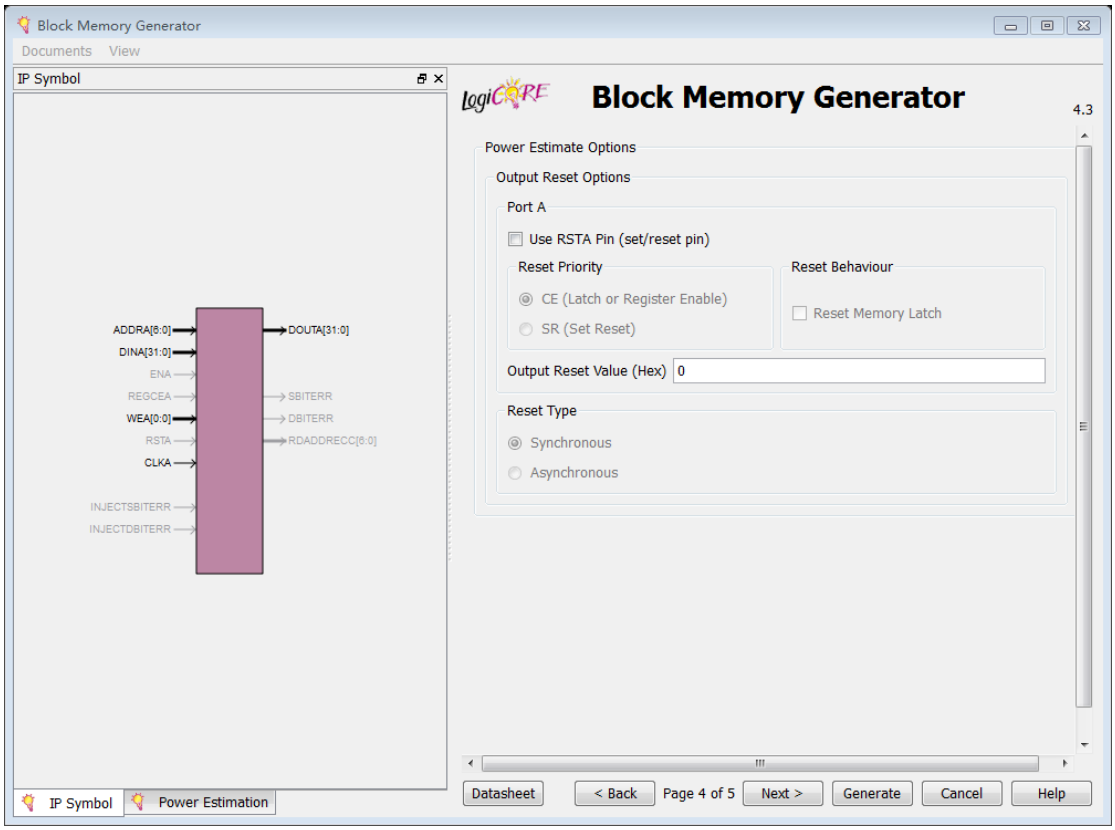
00000000;

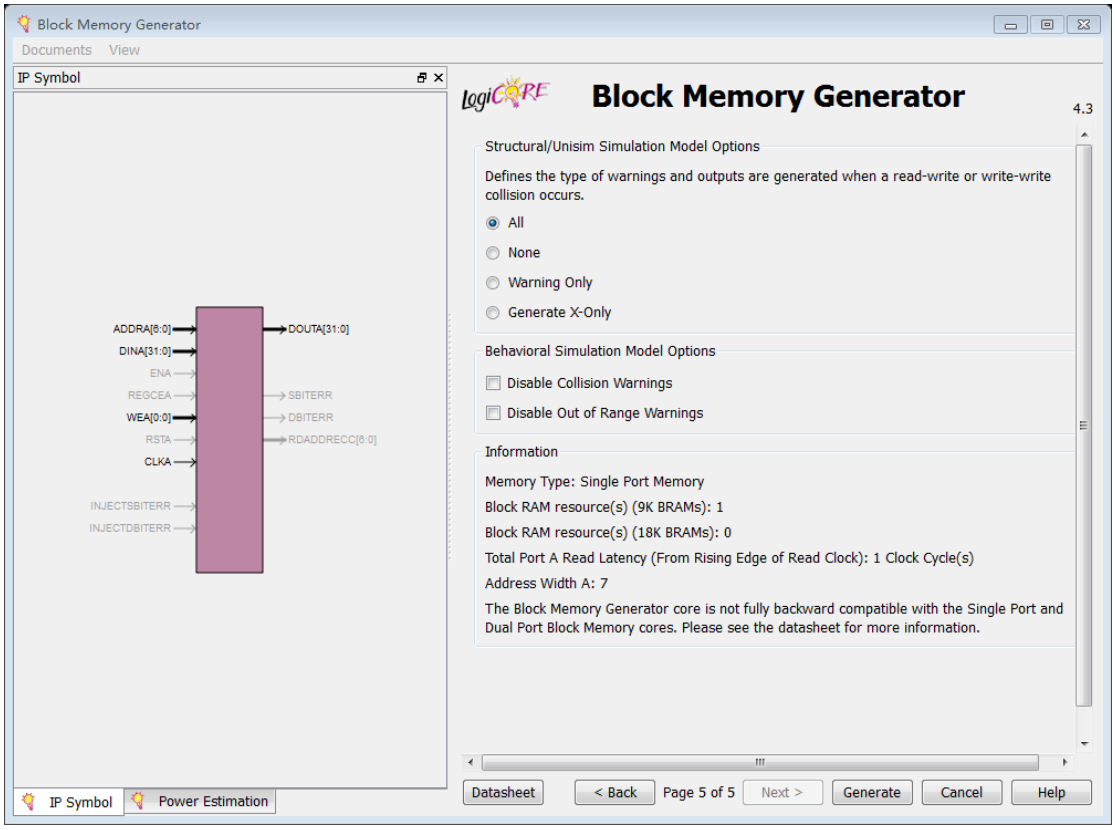
mem


Block Memory Generator
Documents View
IP Symbol
ADDRA[8:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[8:0]
IP Symbol
Power Estimation
Block Memory Generator
4.3
Component Name mem
Memory Type Single Port RAM
Clocking Options
Common Clock
ECC Options
ECC Type No ECC
Use Error Injection Pins Single Bit Error Injection
Write Enable
Use Byte Write Enable
Byte Size 9 bits
Algorithm
Defines the algorithm used to concatenate the block RAM primitives. See the datasheet for more information.
Minimum Area
Low Power
Fixed Primitives
Primitive (Write Port A) : 8kx2
Actual Primitive(s) Used : 4kx2, 8kx2
Datasheet
< Back
Page 1 of 5
Next >
Generate
Cancel
Help


Block Memory Generator
Documents View
IP Symbol
ADDRA[8:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[8:0]
IP Symbol
Power Estimation
Block Memory Generator
4.3
Port A Options
Memory Size
Write Width 32 Range: 1..1152 Read Width: 32
Write Depth 128 Range: 2..9011200 Read Depth: 128
Operating Mode
Write First
Read First
No Change
Enable
Always Enabled
Use ENA Pin
Datasheet
< Back
Page 2 of 5
Next >
Generate
Cancel
Help


Block Memory Generator
Documents View
IP Symbol
ADDRA[8:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[8:0]
IP Symbol
Power Estimation
Block Memory Generator
4.3
Optional Output Registers
Port A
Register Port A Output of Memory Primitives
Register Port A Output of Memory Core
Register Port A Input of SoftECC logic
Use REGCEA Pin (separate enable pin for Port A output registers)
Pipeline Stages within Mux 0
Mux Size: 1237360x1
Latency added by output register(s):
Port A: 0 Clock Cycle(s)
For Spartan-6 Latency information may not be accurate
Memory Initialization
Load Init File
Coe File .\mem.coe
Browse
Show
Fill Remaining Memory Locations
Remaining Memory Locations (Hex) 0
Datasheet
< Back
Page 3 of 5
Next >
Generate
Cancel
Help


Block Memory Generator
Documents View
IP Symbol
ADDRA[8:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[8:0]
IP Symbol
Power Estimation
Block Memory Generator
4.3
Power Estimate Options
Output Reset Options
Port A
Use RSTA Pin (set/reset pin)
Reset Priority
CE (Latch or Register Enable)
SR (Set Reset)
Reset Behaviour
Reset Memory Latch
Output Reset Value (Hex) 0
Reset Type
Synchronous
Asynchronous
Datasheet
< Back
Page 4 of 5
Next >
Generate
Cancel
Help


Block Memory Generator
Documents
View
IP Symbol
ADDRA[6:0]
DINA[31:0]
ENA
REGCEA
WEA[0:0]
RSTA
CLKA
INJECTSBITERR
INJECTDBITERR
DOUTA[31:0]
SBITERR
DBITERR
RDADDRECC[6:0]
IP Symbol
Power Estimation
LogiCORE
Block Memory Generator
4.3
Structural/Unisim Simulation Model Options
Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.
All
None
Warning Only
Generate X-Only
Behavioral Simulation Model Options
Disable Collision Warnings
Disable Out of Range Warnings
Information
Memory Type: Single Port Memory
Block RAM resource(s) (9K BRAMs): 1
Block RAM resource(s) (18K BRAMs): 0
Total Port A Read Latency (From Rising Edge of Read Clock): 1 Clock Cycle(s)
Address Width A: 7
The Block Memory Generator core is not fully backward compatible with the Single Port and Dual Port Block Memory cores. Please see the datasheet for more information.
Datasheet
< Back
Page 5 of 5
Next >
Generate
Cancel
Help