MIPS Single-cycle Processor

Assignment Outline

This assignment is to test your understanding of the MIPS-Lite single-cycle processor design presented in the lectures.
A sample Verilog code is available at Appendix A. You will need to add an Instruction memory to the design and initialise it with your assembly code. The procedure for process is given in the Appendix B.
Currently the memory initialization file of the instruction memory ROM is empty. You can write the machine code of your instructions into the memory locations of the initialisation file.
The assignment is to be undertaken in a series of steps which you should complete in order. To ensure that it is your own work that is being assessed, Appendix C shows the specific requirements for each student.

1. Write a programme to: [15 Marks]

a. Load the data stored in the X and Y locations of the data memory into the X and Y registers.

b. Add the X and Y registers and store the result in the Z register.

c. Store the data from the Z register into the Z memory location.

d. Load the data in the Z memory location into the T register.

2. Simulate your program to show the contents of the X, Y, and T registers. [10 Marks]

3. Modify the *program.mif* file to simulate the operation of the **BEQ** instruction. [10 Marks]

4. Modify the design of the processor so that the **Jump** (J) instruction is implemented. [15 Marks] Modify the *program.mif* file to simulate the operation of the **Jump** instruction. [10 Marks]

5. Modify the designs so that the additional instructions given in the column “Design 1” is correctly executed. [15 Marks]

6. Modify the *program.mif* file to simulate the operation of the additional instruction. [10 Marks]

7. Change the processor’s data bus width from 32 to the one indicated in the table. Repeat the simulation of step 2 using your new processor. [15 Marks]

MIPS instructions’ reference data is provided.

Reports
Your report should include the following contents:

1. The MIPS code and how the corresponding machine code is decided.

2. Simulation waveforms for the PC, opcode, ALUResultOut, DReadData and relevant registers (X, Y, T) must be annotated. You should clearly indicate why the simulations show that the operation is correct or incorrect.

2. Description of the modifications made to the Verilog code for implementing the Jump instruction. Highlight the changes made in the code.

3. Description of the modifications made to implement the additional instruction. Highlight the changes made in the code.

4. Description of the modifications made to change the data bus width.

Submission Date

A single PDF file should be submitted to ICE by 1pm on Monday 2nd December 2019 (Week 13). This file should be named as “surname_forename_A2.pdf”, e.g. “Xu_Ming_A2.pdf”. The contents in each submission must be consistent with the tasks assigned to a specific student; otherwise a zero mark will be awarded. There is no demonstration lab session for this assignment.

Appendix A

```
// MIPS single Cycle processor originaly developed for simulation by Patterson and Hennesy
// Modified for synthesis using the QuartusII package by Dr. S. Ami-Nejad. Feb. 2009

// Register File
module RegisterFile (Read1,Read2,Writereg,WriteData,RegWrite, Data1, Data2,clock,reset);

    input  [4:0] Read1,Read2,Writereg; // the registers numbers to read or write
    input  [31:0] WriteData;           // data to write
    input  RegWrite;               // The write control
    input  clock, reset;           // The clock to trigger writes
    output [31:0] Data1, Data2;    // the register values read;
    reg    [31:0] RF[31:0];        // 32 registers each 32 bits long
    integer    k;

    // Read from registers independent of clock
    assign Data1 = RF[Read1];
    assign Data2 = RF[Read2];
    // write the register with new value on the falling edge of the clock if RegWrite is high
    always @(posedge clock or posedge reset)
        if (reset) for(k=0;k<32;k=k+1) RF[k]<=32'h00000000;
        // Register 0 is a read only register with the content of 0
        else   if (RegWrite & (Writereg!=0)) RF[Writereg] <= WriteData;
endmodule

//ALU Control
module ALUControl (ALUOp, FuncCode, ALUCtl);

    input  [1:0]  ALUOp;
    input  [5:0]  FuncCode;
    output [3:0]  ALUCtl;
    reg    [3:0]  ALUCtl;

    always@( ALUOp, FuncCode)
    begin
    case(ALUOp)
    2'b00: ALUCtl = 4'b0010;
    2'b01: ALUCtl = 4'b0110;
    2'b10: case(FuncCode)
           6'b 100000: ALUCtl = 4'b 0010;
           6'b 100010: ALUCtl = 4'b 0110;
           6'b 100100: ALUCtl = 4'b 0000;
           6'b 100101: ALUCtl = 4'b 0001;
           6'b 101010: ALUCtl = 4'b 0111;
```

```
            default:   ALUCtl = 4'b xxxx;
            endcase
    default:   ALUCtl = 4'b xxxx;
    endcase
    end
endmodule

//ALU
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
    input  [3:0]  ALUctl;
    input  [31:0] A,B;
    output [31:0] ALUOut;
    output Zero;
    reg    [31:0] ALUOut;

    assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
    always @(ALUctl, A, B) begin //reevaluate if these change
    case (ALUctl)
        0: ALUOut <= A & B;
        1: ALUOut <= A | B;
        2: ALUOut <= A + B;
        6: ALUOut <= A - B;
        7: ALUOut <= A < B ? 1:0;
        // .... Add more ALU operations here
        default: ALUOut <= A;
        endcase
    end
endmodule

// Data Memory
module DataMemory(Address, DWriteData, MemRead, MemWrite, clock, reset, DReadData);
input  [31:0] Address, DWriteData;
input         MemRead, MemWrite, clock, reset;
output [31:0] DReadData;
reg    [31:0] DMem[7:0];

assign  DReadData = DMem[Address[2:0]];
always @(posedge clock or posedge reset)begin
        if (reset) begin
            DMem[0]=32'h00000005;
            DMem[1]=32'h0000000A;
            DMem[2]=32'h00000055;
            DMem[3]=32'h000000AA;
            DMem[4]=32'h00005555;
            DMem[5]=32'h00008888;
            DMem[6]=32'h00550000;
            DMem[7]=32'h00004444;
            end else
```

```
            if (MemWrite) DMem[Address[2:0]] <= DWriteData;
        end
endmodule

// Main Controller
module Control
(opcode,RegDst,Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,RegWrite);

input  [5:0]  opcode;
output [1:0]  ALUOp;
output RegDst,Branch,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite;
reg    [1:0]  ALUOp;
reg    RegDst,Branch,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite;

parameter R_Format = 6'b000000, LW = 6'b100011, SW = 6'b101011, BEQ=6'b000100;
always @(opcode)begin
    case(opcode)
        R_Format:{RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp}=
9'b 100100010;
        LW:     {RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp}=
9'b 011110000;
        SW:     {RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp}=
9'b x1x001000;
        BEQ:    {RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp}=
9'b x0x000101;
        // .... Add more instructions here
        default: {RegDst,ALUSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUOp}=
9'b xxxxxxxxx;
        endcase
    end
endmodule

// Datapath
module DataPath(RegDst, Branch, MemRead, MemtoReg, ALUOp, MemWrite,
ALUSrc, RegWrite, clock, reset, opcode,/* RF1,  RF2, RF3,*/ALUResultOut ,DReadData);

input  RegDst,Branch,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite,clock, reset;
input  [1:0]  ALUOp;
output [5:0]  opcode;
output [31:0] /*RF1,  RF2, RF3,*/ ALUResultOut ,DReadData;

reg    [31:0] PC, IMemory[0:31];
wire   [31:0] SignExtendOffset, PCOffset, PCValue, ALUResultOut,
       IAddress, DAddress, IMemOut, DmemOut, DWriteData, Instruction,
       RWriteData, DReadData, ALUAin, ALUBin;
wire   [3:0] ALUctl;
wire   Zero;
wire   [4:0] WriteReg;
```

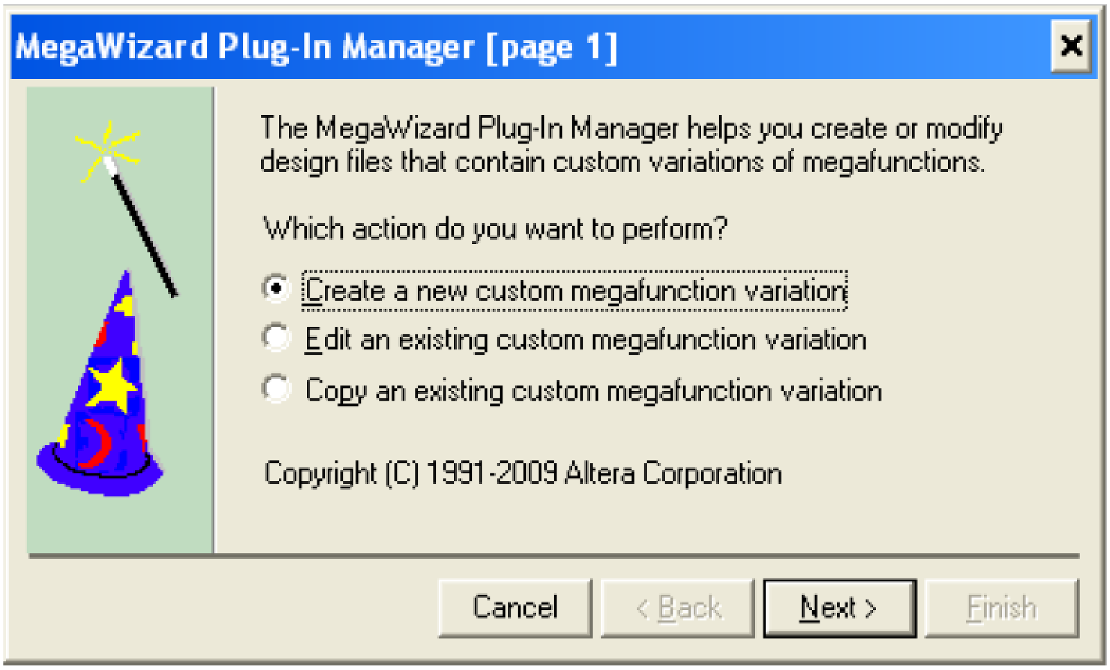
```
//Instruction fields, to improve code readability
wire [5:0]    funct;
wire [4:0]    rs, rt, rd, shamt;
wire [15:0] offset;

//Instantiate local ALU controller
ALUControl alucontroller(ALUOp,funct,ALUctl);

// Instantiate ALU
MIPSALU ALU(ALUctl, ALUAin, ALUBin, ALUResultOut, Zero);

// Instantiate Register File
RegisterFile REG(rs, rt, WriteReg, RWriteData, RegWrite, ALUAin,
DWriteData,clock,reset);

// Instantiate Data Memory
DataMemory datamemory(ALUResultOut, DWriteData, MemRead, MemWrite, clock, reset,
DReadData);

// Instantiate Instruction Memory
IMemory    IMemory_inst (
    .address ( PC[6:2] ),
    .q ( Instruction )
    );

// Synthesize multiplexers
assign WriteReg   = (RegDst)        ? rd              : rt;
assign ALUBin     = (ALUSrc)           ? SignExtendOffset   : DWriteData;
assign PCValue       = (Branch & Zero) ? PC+4+PCOffset   : PC+4;
assign RWriteData    = (MemtoReg)      ? DReadData          : ALUResultOut;

// Acquire the fields of the R_Format Instruction for clarity
assign {opcode, rs, rt, rd, shamt, funct} = Instruction;
// Acquire the immediate field of the I_Format instructions
assign offset = Instruction[15:0];
//sign-extend lower 16 bits
assign SignExtendOffset = { {16{offset[15]}} , offset[15:0]};
// Multiply by 4 the PC offset
assign PCOffset = SignExtendOffset << 2;
// Write the address of the next instruction into the program counter
always @(posedge clock ) begin
if (reset) PC<=32'h00000000; else
    PC <= PCValue;
end
endmodule

module MIPS1CYCLE(clock, reset,opcode, ALUResultOut ,DReadData);
```

```
    input  clock, reset;
    output [5:0]  opcode;
    output [31:0] ALUResultOut ,DReadData; // For simulation purposes

    wire [1:0] ALUOp;
    wire [5:0] opcode;
    wire [31:0] SignExtend,ALUResultOut ,DReadData;
    wire RegDst,Branch,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite;

    // Instantiate the Datapath
    DataPath MIPSDP (RegDst,Branch,MemRead,MemtoReg,ALUOp,
    MemWrite,ALUSrc,RegWrite,clock, reset, opcode, ALUResultOut ,DReadData);

    //Instantiate the combinational control unit
    Control MIPSControl
(opcode,RegDst,Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,RegWrite);
endmodule
```

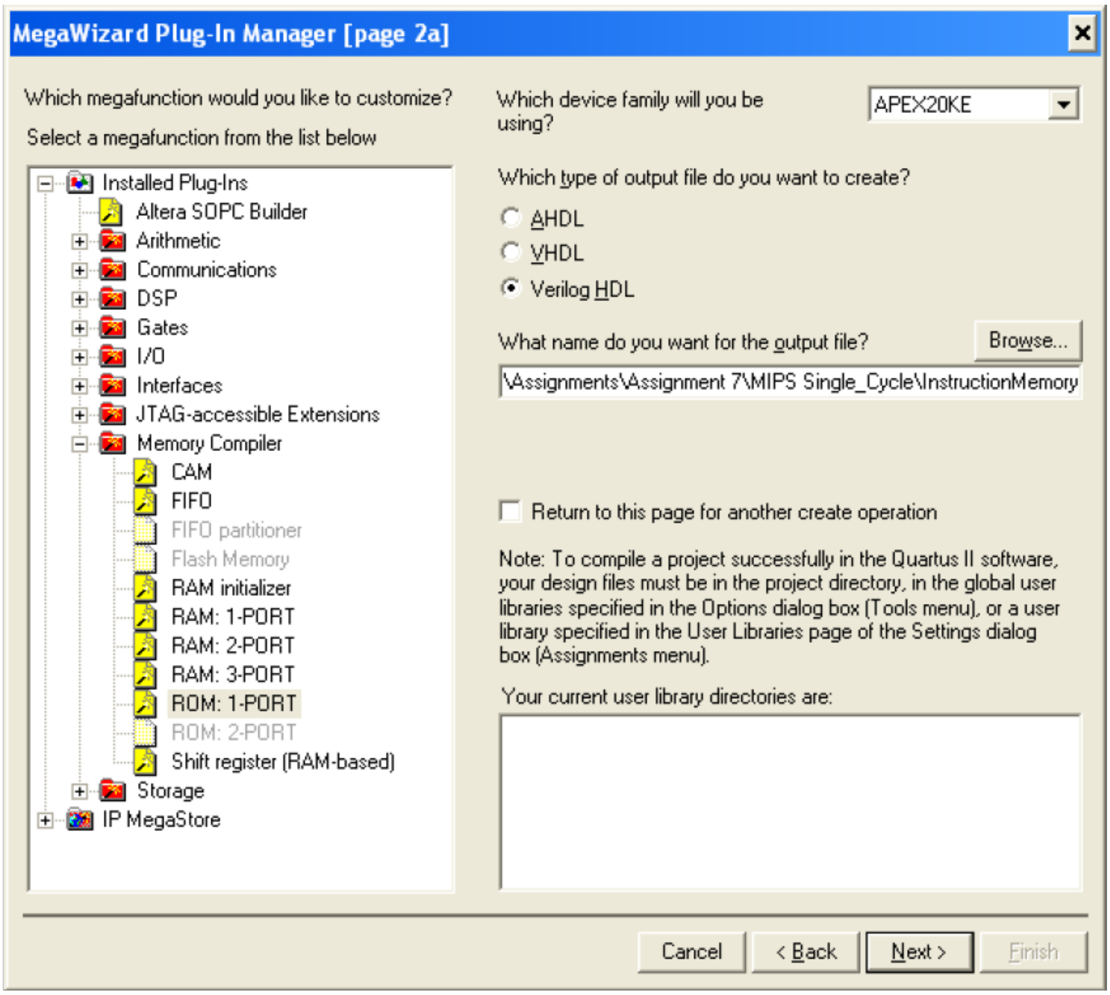
Appdendix B

Inserting LPM ROM

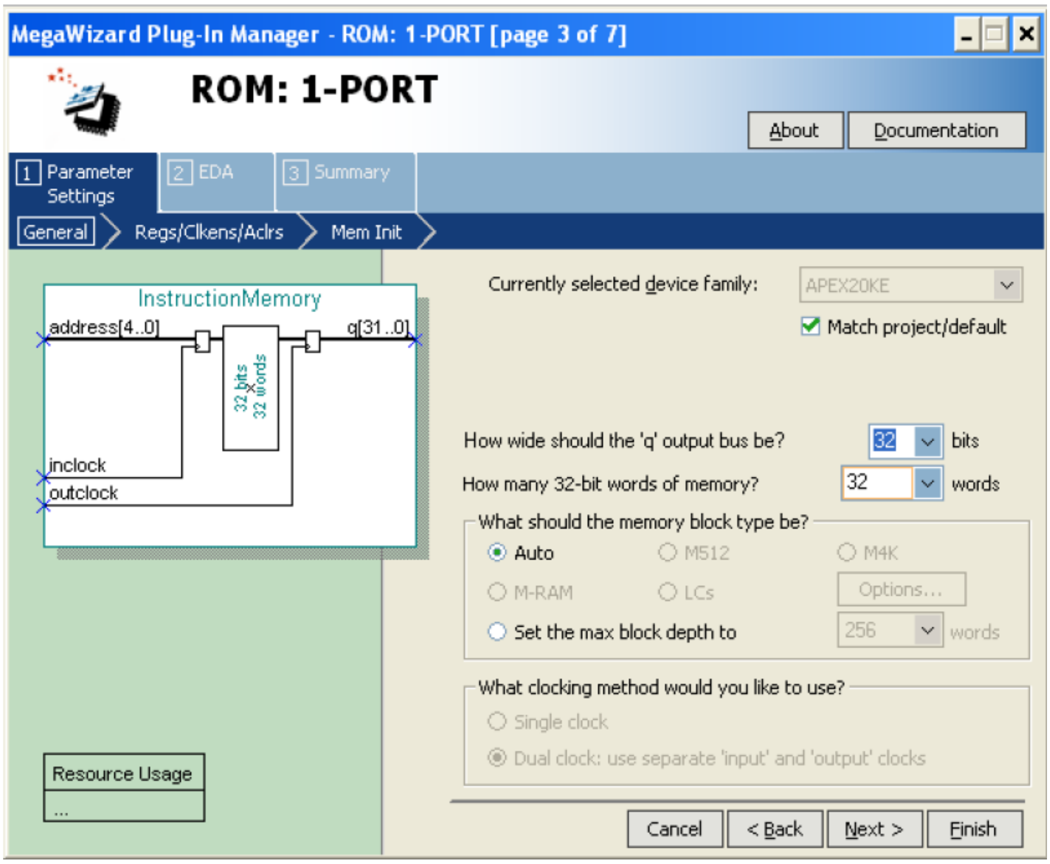
Select MegaWizard Plug-In Manager from the tools menu and click on the Next button.



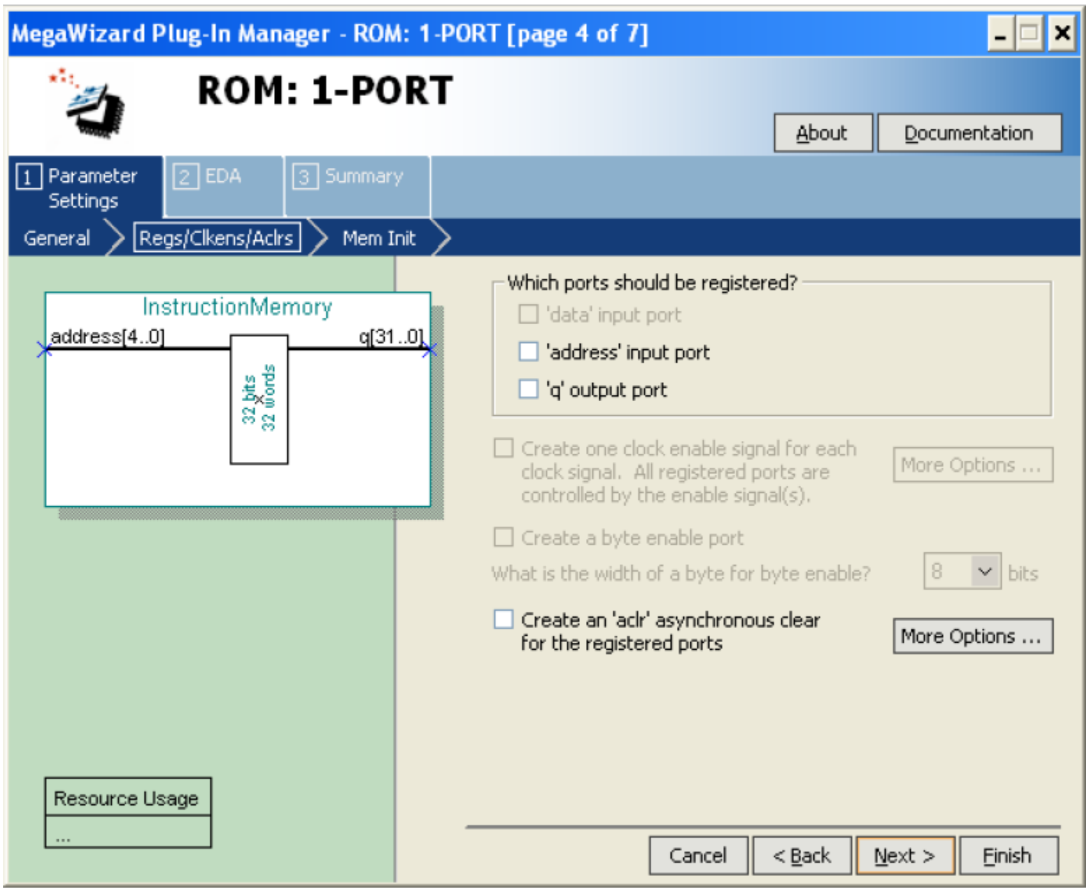
Select the ROM: 1-PORT and give a name to the output file. This name should be the same as the one which has been used for Instruction memory instantiation in your Verilog code.



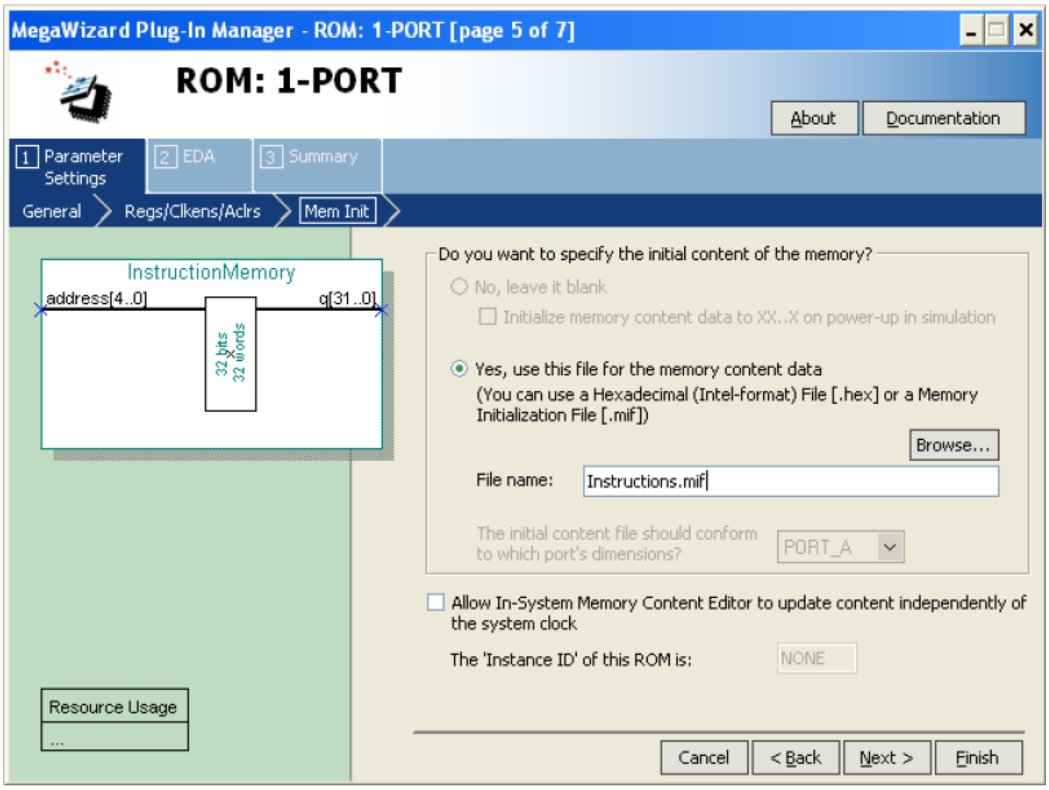
Select the width and depth of the memory.



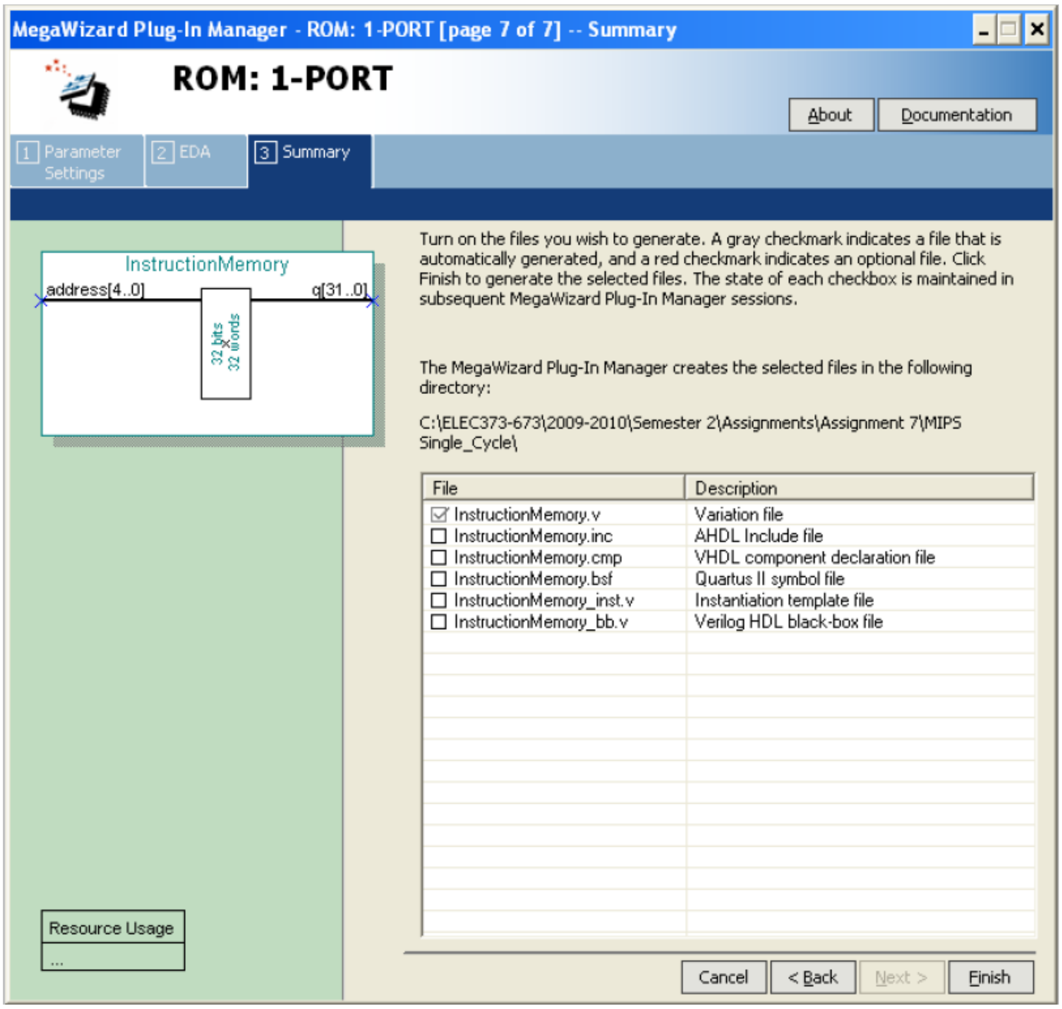
Deselect the input and output registers.



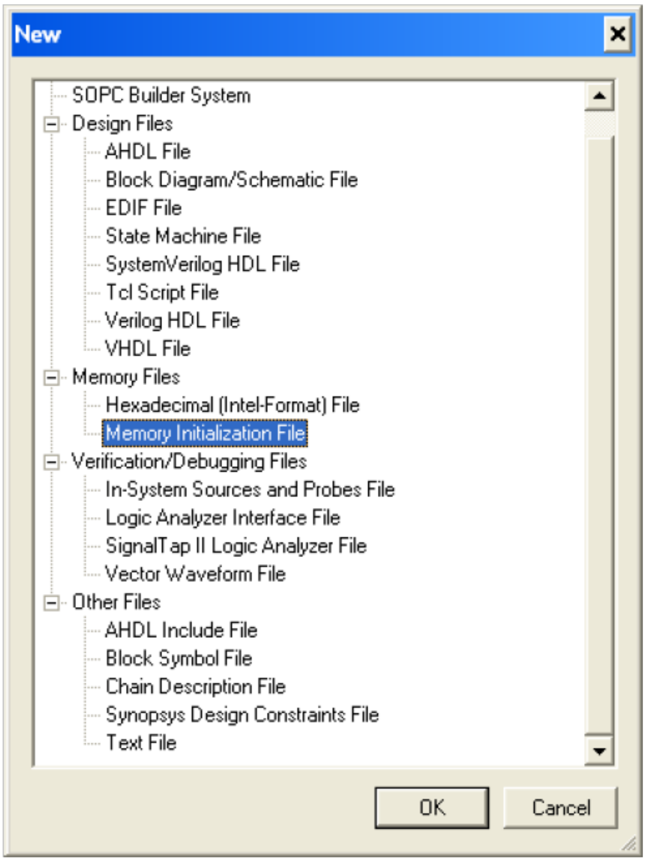
Give a name to the memory initialisation file, e.g. Instruction.mif. You can use your name as the file name. In this case you should edit the Verilog module name in the provided code.



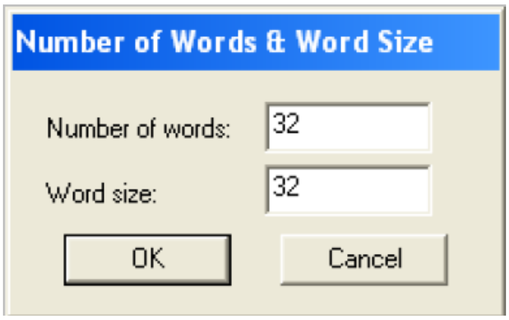
Deselect the creation of the optional files and click on Finish button.



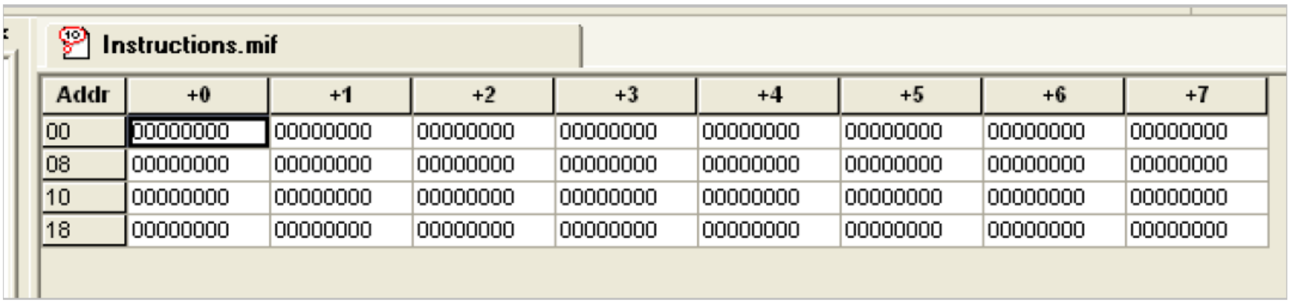
Click on the New icon and select the Memory Initialization File.



Input 32 for the Number of words and 32 for the Word size.



Now you can insert your machine code into the Instruction memory locations. You should be noted that this memory is not byte oriented and addresses are incremented by one not by 4.



Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
08	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
18	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

Appendix C: The tasks for each student

ID	English Name	Memory locations (X,Y,Z,T)	Data bus size	Design
1507241	Wei Dai	3,4,5,6	24	Ori
1613432	Dong Li	7,1,2,3	28	Andi
1614161	Yang Liu	4,5,6,7	20	Ori
1613475	Meichen Bu	1,2,3,4	24	Andi
1510415	Ting-Wei Chang	5,6,7,1	28	Ori
1611435	Bingqing Chen	2,3,4,5	20	Andi
1612920	Jiacheng Cheng	6,7,1,2	24	Ori
1613791	Haoran Ding	3,4,5,6	28	Andi
1405860	Jinpeng Ding	7,1,2,3	20	Ori
1405334	Yimin Du	4,5,6,7	24	Andi
1613589	Cheng Fang	1,2,3,4	28	Ori
1611465	Yuanhao Gong	5,6,7,1	20	Andi
1510104	Zhen Gong	2,3,4,5	24	Ori
1719239	Naomi Kimberly Grant	6,7,1,2	28	Andi
1200021	Chenyang Hu	3,4,5,6	20	Ori
1612924	Minling Huang	7,1,2,3	24	Andi
1612926	Yanbo Huang	4,5,6,7	28	Ori
1612517	Zelin Jiang	1,2,3,4	20	Andi
1612315	Jinlu Li	5,6,7,1	24	Ori
1302823	Ruoxi Li	2,3,4,5	28	Andi
1611136	Weiyi Li	6,7,1,2	20	Ori
1613610	Runze Liu	3,4,5,6	24	Andi
1405431	Xinyu Liu	7,1,2,3	28	Ori
1509026	Xianwang Liu	4,5,6,7	20	Andi
1614274	Yuzhe Liu	1,2,3,4	24	Ori
1611394	Zhen Liu	5,6,7,1	28	Andi
1614649	Kai-Yu Lu	2,3,4,5	20	Ori
1301931	Qian Lu	6,7,1,2	24	Andi
1613209	Ningyu Luo	3,4,5,6	28	Ori
1405104	Jianxiao Lyu	7,1,2,3	20	Andi
1613681	Jinwei Lyu	4,5,6,7	24	Ori
1612320	Jiaxin Ma	1,2,3,4	28	Andi
1612363	Chengwei Ouyang	5,6,7,1	20	Ori
1611493	Jimin Pan	2,3,4,5	24	Andi
1405437	Wenrui Peng	6,7,1,2	28	Ori
1612528	Enze Pu	3,4,5,6	20	Andi
1612564	Chenghu Qiu	7,1,2,3	24	Ori
1612324	Tianyao Ren	4,5,6,7	28	Andi
1614650	Sahand Sabour	1,2,3,4	20	Ori
1613622	Heng Shi	5,6,7,1	24	Andi

1509645	Ziyu Song	2,3,4,5	28	Ori
1509255	Qianyifan Tang	6,7,1,2	20	Andi
1509256	Chengyu Wan	3,4,5,6	24	Ori
1611928	Han Wang	7,1,2,3	28	Andi
1612625	Jialin Wang	4,5,6,7	20	Ori
1509008	Mingnan Wang	1,2,3,4	24	Andi
1613938	Qingyao Wang	5,6,7,1	28	Ori
1613646	Xiaodan Wang	2,3,4,5	20	Andi
1614316	Yuwen Wang	6,7,1,2	24	Ori
1509795	Zonghui Wang	3,4,5,6	28	Andi
1613256	Ziran Wang	7,1,2,3	20	Ori
1612957	Chaoyang Wei	4,5,6,7	24	Andi
1613489	Mingnan Wei	1,2,3,4	28	Ori
1405447	Sichen Wei	5,6,7,1	20	Andi
1611516	Jiacheng Wen	2,3,4,5	24	Ori
1612639	Kunyang Wu	6,7,1,2	28	Andi
1614325	Yuheng Xie	3,4,5,6	20	Ori
1611410	Zikang Xu	7,1,2,3	24	Andi
1611277	Qifan Yan	4,5,6,7	28	Ori
1613799	Jie Yang	1,2,3,4	20	Andi
1614332	Sen Yang	5,6,7,1	24	Ori
1613667	Zhibin Yu	2,3,4,5	28	Andi
1614453	Hengjia Zhang	6,7,1,2	20	Ori
1509811	Jiacheng Zhang	3,4,5,6	24	Andi
1612117	Minxing Zhang	7,1,2,3	28	Ori
1300796	Ning Zhang	4,5,6,7	20	Andi
1507303	Xiangben Zhao	1,2,3,4	24	Ori
1612654	Zibo Zhao	5,6,7,1	28	Andi
1612657	Mingkai Zheng	2,3,4,5	20	Ori
1612912	Dianjun Zhou	6,7,1,2	24	Andi