实验项目名称：Function Units Design

一、 实验目的和要求

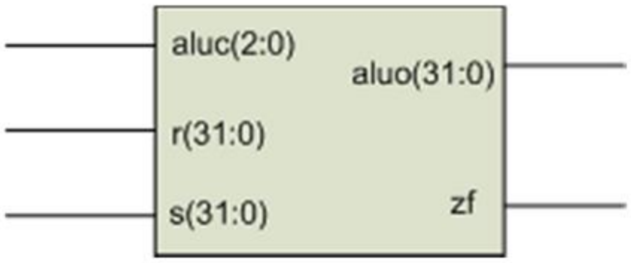
Please do the project step by step according to the project tutorial. Design the basic function components including ALU, ALU Controller and register file using Xilinx.

二、 实验内容和原理

Design the basic function components including ALU, ALU Controller and register file.

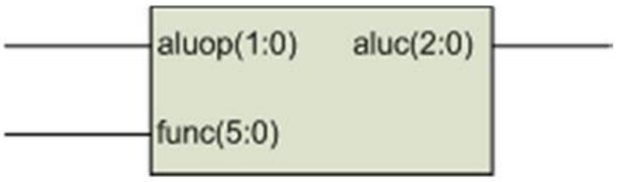
1. ALU

1) Input: Operator r and s are32-bit integers, and aluc is the code which defines ALU Operating.
2) Processing: alul = r op s; zf =(op == sub)&&(r==s);
3) Output: aluo is the result of (r op s), zf is the flag of zero output.



2. ALU Controller

1) Input: 2-bit ALU Operation Code, 6-bit Function Code
2) Processing: mapping according to ALU Operation Code Table
3) Output: 3-bit ALU Controller

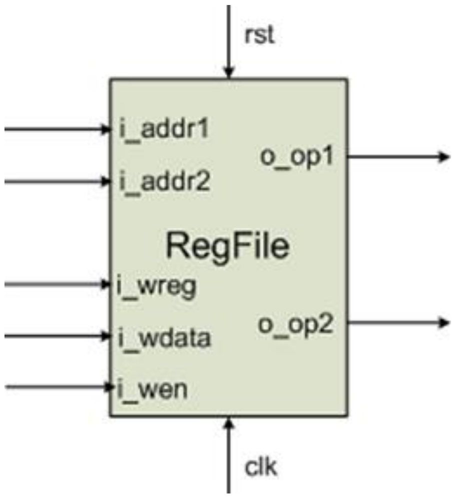


3. Register file

1) Input: Reading Address 1, Reading Address 2, Writing Address, Writing Data, Write Enable.

2) Processing: According read or write instruction, output the operators.

3) Output: Operator 1, Operator 2.



三、 实验过程和数据记录

1. ALU

The code of the file “macro.vh” :

```
//macro.vh
//macro definitions
//ALU CONTROL CODES
`define ALU_CTL_AND  3'b000
`define ALU_CTL_OR   3'b001
`define ALU_CTL_ADD  3'b010
`define ALU_CTL_SUB  3'b110
`define ALU_CTL_SLT  3'b111
```

The code of the file “ALU.v” :

```
`include "macro.vh"
module ALU(input CCLK, input [2:0] SW, output LCDRS, LCDRW, LCDE, output
[3:0] LCDDAT, output LED);
   wire [3:0] lcdd;
   wire rslcd, rwlcd, elcd;
   wire o_zf;              //o_zf: zero flag output
   wire [31:0] o_alu;      //o_alu: ALU result
   wire [2:0] i_aluc;      //i_aluc: ALU ctrl
   reg [31:0] o_alu_old;   //o_alu_old: former ALU result
   reg [255:0]strdata;     //strdata: data displayed in the LCD
   reg [31:0] i_r;         //i_r: r input of ALU
   reg [31:0] i_s;         //i_s: s input of ALU
   reg rst;//rst: reset signal indicating whether ALU result is updated

   assign LCDDAT[3]=lcdd[3];
   assign LCDDAT[2]=lcdd[2];
   assign LCDDAT[1]=lcdd[1];
   assign LCDDAT[0]=lcdd[0];
   assign LCDRS=rslcd;
   assign LCDRW=rwlcd;
   assign LCDE=elcd;
   //output zero flag to LED
   assign LED = o_zf;
   //use switches as ALU ctrl input
```

```
    assign i_aluc[0] = SW[0];
    assign i_aluc[1] = SW[1];
    assign i_aluc[2] = SW[2];

    initial begin
        strdata = "1111 2222";   //set display data
        i_r = 32'h1111;          //set r input
        i_s = 32'h2222;          //set s input
        rst = 0;                 //set rst singal
        o_alu_old = 0;           //set initial ALU’s old result
    end

    display M0 (CCLK, rst, strdata, rslcd, rwlcd, elcd, lcdd);
    single_alu M1(i_r, i_s, i_aluc, o_zf, o_alu);

    always @(posedge CCLK) begin
        if (o_alu_old != o_alu) begin
            //display alu result in the LCD using sexadecimal notion
            if (o_alu[15:12] < 10)
                strdata[127:120] = 8'h30 + o_alu[15:12];
            else
                strdata[127:120] = 8'h61 + o_alu[15:12]-10;
            if (o_alu[11:8] < 10)
                strdata[119:112] = 8'h30 + o_alu[11:8];
            else
                strdata[119:112] = 8'h61 + o_alu[11:8]-10;
            if (o_alu[7:4] < 10)
                strdata[111:104] = 8'h30 + o_alu[7:4];
            else
                strdata[111:104] = 8'h61 + o_alu[7:4]-10;
            if (o_alu[3:0] < 10)
                strdata[103:96] = 8'h30 + o_alu[3:0];
            else
                strdata[103:96] = 8'h61 + o_alu[3:0]-10;
            rst = 1;
            o_alu_old = o_alu;          //update alu result
        end
        else
            rst = 0;
    end
endmodule
```

The code of the file “single_alu.v” :

```
`include "macro.vh"
module single_alu(i_r,i_s,i_aluc,o_zf,o_alu);
    input [31:0] i_r;    //i_r: r input
    input [31:0] i_s;    //i_s: s input
    input [2:0] i_aluc;  //i_aluc: ctrl input
    output o_zf;         //o_zf: zero flag output
    output [31:0] o_alu; //o_alu: alu result output
    reg o_zf;
    reg [31:0] o_alu;

    always @(i_aluc or i_r or i_s) begin
        case (i_aluc)
            `ALU_CTL_AND:begin
                o_zf=0;
                o_alu=i_r & i_s;
            end
            `ALU_CTL_OR:begin
                o_zf=0;
```

```
              o_alu=i_r | i_s;
           end
           `ALU_CTL_ADD:begin
              o_zf=0;
              o_alu=i_r + i_s;
           end
           `ALU_CTL_SUB:begin
              o_alu=i_r - i_s;
              o_zf=(o_alu == 0);
           end
           `ALU_CTL_SLT:begin
              o_zf=0;
              if(i_r < i_s)
                 o_alu=1;
              else o_alu=0;
           end
           default:begin
              o_zf=0;
              o_alu=0;
           end
        endcase
    end
endmodule
```

2. ALU Controller

The code of the file “ALUC.v” :

```
`include "macro.vh"
module ALUC(input CCLK, input [1:0]BTN, input [3:0] SW, output LCDRS, LCDRW,
LCDE, output [3:0] LCDDAT, output LED);
    wire [3:0]  lcdd;
    wire rslcd, rwlcd, elcd;
    wire         o_zf;           //o_zf: zero flag output
    wire [31:0] o_alu;           //o_alu: ALU result
    wire [2:0]  i_aluc;          //i_aluc: ALU ctrl signal
    wire [1:0]  alu_op;//alu_op: the op field of ALU ctrl input signal
    wire [3:0]  func; //func: the function field of ALU ctrl input signal
    reg [31:0]  o_alu_old;       //o_alu_old: former ALU result
    reg [255:0] strdata;         //strdata: data displayed in the LCD
    reg [31:0]  i_r;             //i_r: r input of ALU
    reg [31:0]  i_s;             //i_s: s input of ALU
    reg rst;//rst: reset signal indicating whether ALU result is updated

    assign LCDDAT[3] = lcdd[3];
    assign LCDDAT[2] = lcdd[2];
    assign LCDDAT[1] = lcdd[1];
    assign LCDDAT[0] = lcdd[0];
    assign LCDRS = rslcd;
    assign LCDRW = rwlcd;
    assign LCDE = elcd;
    //output zero flag to LED
    assign LED = o_zf;
    //use switches as func input
    assign func[0] = SW[0];
    assign func[1] = SW[1];
    assign func[2] = SW[2];
    assign func[3] = SW[3];
    //use buttons as alu_op input
    assign alu_op[0] = BTN[0];
    assign alu_op[1] = BTN[1];

    initial begin
        strdata = "1111 2222";  //set display data
        i_r = 32'h1111;         //set r input
        i_s = 32'h2222;         //set s input
        rst = 0;                //set rst singal
        o_alu_old = 0;          //set initial ALU’s old result
    end

    display M0 (CCLK, rst, strdata, rslcd, rwlcd, elcd, lcdd);
    single_alu M1(i_r, i_s, i_aluc, o_zf, o_alu);
    single_aluc M2(alu_op, func, i_aluc);

    always @(posedge CCLK) begin
        //display alu result in the LCD using sexadecimal notion
        if (o_alu_old != o_alu) begin
            if (o_alu[15:12] < 10)
                strdata[127:120] = 8'h30 + o_alu[15:12];
            else
                strdata[127:120] = 8'h61 + o_alu[15:12]-10;
            if (o_alu[11:8] < 10)
                strdata[119:112] = 8'h30 + o_alu[11:8];
            else
                strdata[119:112] = 8'h61 + o_alu[11:8]-10;
            if (o_alu[7:4] < 10)
                strdata[111:104] = 8'h30 + o_alu[7:4];
            else
                strdata[111:104] = 8'h61 + o_alu[7:4]-10;
            if (o_alu[3:0] < 10)
                strdata[103:96] = 8'h30 + o_alu[3:0];
            else
                strdata[103:96] = 8'h61 + o_alu[3:0]-10;
            rst = 1;
            o_alu_old = o_alu;              //update alu result
        end
        else
            rst = 0;
    end

endmodule
```

The code of the file “ALU.v” :

```
//macro definitions of instruction types
`define ALUC_CTL_AND 3'b000
`define ALUC_CTL_OR  3'b001
`define ALUC_CTL_ADD 3'b010
`define ALUC_CTL_SUB 3'b110
`define ALUC_CTL_SLT 3'b111
`define RTYPE_ADD    4'b0000
`define RTYPE_SUB    4'b0010
`define RTYPE_SLT    4'b1010
`define RTYPE_AND    4'b0100
```

```
`define RTYPE_OR     4'b0101
`define RTYPE_XOR    4'b0110
`define RTYPE_NOR    4'b0111
`define RTYPE_ADDU   4'b0001
`define RTYPE_SLTU   4'b1011
`include "macro.vh"
module single_aluc(aluop, func, aluc);
   input [1:0] aluop; //alu_op: the op field of ALU ctrl input signal
   input [5:0] func; //func: the function field of ALU ctrl input signal
   output [2:0] aluc;   //i_aluc: ALU ctrl signal
   reg [2:0] aluc;

    always @(aluop or func) begin
       case (aluop)
           2'b00:begin
               aluc=`ALU_CTL_ADD;
           end
           2'b01:begin
               aluc=`ALU_CTL_SUB;
           end
           2'b1x:begin
               case(func)
                   `RTYPE_AND:aluc=`ALU_CTL_AND;
                   `RTYPE_OR:aluc=`ALU_CTL_OR;
                   `RTYPE_ADD:aluc=`ALU_CTL_ADD;
                   `RTYPE_SUB:aluc=`ALU_CTL_SUB;
                   `RTYPE_SLT:aluc=`ALU_CTL_SLT;
                  default:aluc=3'b000;
               endcase
           end
           default:aluc=3'b000;
       endcase
   end
endmodule
```

The “macro.vh” and “single_alu.v” file is the same as those in the ALU project.

3. Register file

```
//macro definitions of instruction types
`define ALUC_CTL_AND 3'b000
`define ALUC_CTL_OR  3'b001
`define ALUC_CTL_ADD 3'b010
`define ALUC_CTL_SUB 3'b110
`define ALUC_CTL_SLT 3'b111
`define RTYPE_ADD    4'b0000
`define RTYPE_SUB    4'b0010
`define RTYPE_SLT    4'b1010
`define RTYPE_AND    4'b0100
`define RTYPE_OR     4'b0101
`define RTYPE_XOR    4'b0110
`define RTYPE_NOR    4'b0111
```

```
`define RTYPE_ADDU   4'b0001
`define RTYPE_SLTU   4'b1011

module Rtype(clk, rst, instru, Adat, Bdat, result);
   input clk;               //clk: clock signal
   input rst;               //rst: reset signal
   input [31:0] instru;     //instru: instruction input
   //result: ALU result output
   //Adat: data of ALU's A input
   //Bdat: data of ALU's B input
   output [31:0] result, Adat, Bdat;
   wire [31:0]   result;
   wire [31:0]   Wdat, Adat, Bdat, ALUout;
   wire [2:0] ALUoper;

   RegFile x_regfile(clk, instru[25:21], instru[20:16], instru[15:11],
result, Adat, Bdat, 1);
   ALUctr x_aluctr(2'b10, instru[5:0], ALUoper);
   ALUnit x_alunit(Adat, Bdat, ALUoper, result, zero);
endmodule

module ALUnit(Adat, Bdat, ALUoper, Result, zero);
   input [31:0] Adat;       //Adat: data of A input
   input [31:0] Bdat;       //Bdat: data of B input
   input [2:0] ALUoper;     //ALUoper: ALU ctrl signal
   output [31:0] Result;    //Result: ALU result output
   reg [31:0] Result;
   output zero;
   reg zero;

   always @(ALUoper)begin
      case(ALUoper)
         3'b000:   Result=Adat & Bdat;             //And
         3'b001:   Result=Adat | Bdat;             //Or
         3'b010:   Result=Adat + Bdat;             //Add
         3'b110:   Result=Adat - Bdat;             //Sub
         3'b111:   if(Adat < Bdat)   Result=1;     //Slt
                   else   Result=0;
      endcase
      if(Adat == Bdat)zero=1; else zero=0;
   end
endmodule

module ALUctr(ALUop, Func, ALUoper);
   input [1:0] ALUop;//ALUop: the op field of ALU ctrl input signal
   input [5:0] Func; //Func: the function field of ALU ctrl input signal
   output [2:0] ALUoper;    //ALUoper: ALU ctrl signal
   reg [2:0]  ALUoper;

   always @ (ALUop or Func)begin
      case(ALUop)
         2'b00: ALUoper = 3'b010;              //Add
         2'b01: ALUoper = 3'b110;              //Sub
         default:
            case(Func)
               6'b100000: ALUoper = 3'b010;    //Add
               6'b100010: ALUoper = 3'b110;    //Sub
               6'b100100: ALUoper = 3'b000;    //And
               6'b100101: ALUoper = 3'b001;    //Or
               6'b101010: ALUoper = 3'b111;    //Slt
         endcase
```

```
        endcase
    end
endmodule

module RegFile(clk, regA, regB, regW, Wdat, Adat, Bdat, RegWrite);
    input clk;             //clk: clock signal
    input [4:0] regA;      //regA: index of register A
    input [4:0] regB;      //regB: index of register B
    input [4:0] regW;      //regW: index of register to write into
    input [31:0] Wdat;     //Wdat: the data to write into regW
    input RegWrite;        //RegWrite: enable signal of writing
    //Adat: the content of regA
    //Bdat: the content of regB
    output [31:0] Adat, Bdat;
    reg [31:0] Adat, Bdat;
    //register file
    reg[31:0]  Szero, Sat, Sa0, Sa1, Sv0, Sv1, Sv2, Sv3; //0
    reg[31:0]  St0, St1, St2, St3, St4, St5, St6, St7;   //8
    reg[31:0]  Ss0, Ss1, Ss2, Ss3, Ss4, Ss5, Ss6, Ss7;   //16
    reg[31:0]  St8, St9, Sk0, Sk1, Sgp, Sfp, Ssp, Sra;   //24

    initial begin
        Szero = 0; //register file
        Sat = 1;
        Sa0 = 2;
        Sa1 = 3;
        Sv0 = 4;
        Sv1 = 5;
        Sv2 = 6;
        Sv3 = 7;
        St0 = 8;
        St1 = 9;
        St2 = 10;
        St3 = 11;
        St4 = 12;
        St5 = 13;
        St6 = 14;
        St7 = 15;
        Ss0 = 16;
        Ss1 = 17;
        Ss2 = 18;
        Ss3 = 19;
        Ss4 = 20;
        Ss5 = 21;
        Ss6 = 22;
        Ss7 = 23;
        St8 = 24;
        St9 = 25;
        Sk0 = 26;
        Sk1 = 27;
        Sgp = 28;
        Sfp = 29;
        Ssp = 30;
        Sra = 31;
    end
    always @ (posedge clk)begin
        case(regA)
        //read the content of regA
            0:     Adat = 0;
            1:     Adat = Sat ;
            2:     Adat = Sa0 ;
```

```
           3:     Adat = Sa1 ;
           4:     Adat = Sv0 ;
           5:     Adat = Sv1 ;
           6:     Adat = Sv2 ;
           7:     Adat = Sv3 ;
           8:     Adat = St0 ;
           9:     Adat = St1 ;
           10:    Adat = St2 ;
           11:    Adat = St3 ;
           12:    Adat = St4 ;
           13:    Adat = St5 ;
           14:    Adat = St6 ;
           15:    Adat = St7 ;
           16:    Adat = Ss0 ;
           17:    Adat = Ss1 ;
           18:    Adat = Ss2 ;
           19:    Adat = Ss3 ;
           20:    Adat = Ss4 ;
           21:    Adat = Ss5 ;
           22:    Adat = Ss6 ;
           23:    Adat = Ss7 ;
           24:    Adat = St8 ;
           25:    Adat = St9 ;
           26:    Adat = Sk0 ;
           27:    Adat = Sk1 ;
           28:    Adat = Sgp ;
           29:    Adat = Sfp ;
           30:    Adat = Ssp ;
           31:    Adat = Sra ;
       endcase
       case(regB)
       //read the content of regB
           0:     Bdat = 0;
           1:     Bdat = Sat ;
           2:     Bdat = Sa0 ;
           3:     Bdat = Sa1 ;
           4:     Bdat = Sv0 ;
           5:     Bdat = Sv1 ;
           6:     Bdat = Sv2 ;
           7:     Bdat = Sv3 ;
           8:     Bdat = St0 ;
           9:     Bdat = St1 ;
           10:    Bdat = St2 ;
           11:    Bdat = St3 ;
           12:    Bdat = St4 ;
           13:    Bdat = St5 ;
           14:    Bdat = St6 ;
           15:    Bdat = St7 ;
           16:    Bdat = Ss0 ;
           17:    Bdat = Ss1 ;
           18:    Bdat = Ss2 ;
           19:    Bdat = Ss3 ;
           20:    Bdat = Ss4 ;
           21:    Bdat = Ss5 ;
           22:    Bdat = Ss6 ;
           23:    Bdat = Ss7 ;
           24:    Bdat = St8 ;
           25:    Bdat = St9 ;
           26:    Bdat = Sk0 ;
           27:    Bdat = Sk1 ;
           28:    Bdat = Sgp ;
           29:     Bdat = Sfp ;
           30:     Bdat = Ssp ;
           31:     Bdat = Sra ;
       endcase
    end
    always @ (negedge clk)begin
       if(RegWrite==1)
       case(Wdat)
       //write Wdat into the register file
           0:      Szero = 0;
           1:      Sat = Wdat;
           2:      Sa0 = Wdat;
           3:      Sa1 = Wdat;
           4:      Sv0 = Wdat;
           5:      Sv1 = Wdat;
           6:      Sv2 = Wdat;
           7:      Sv3 = Wdat;
           8:      St0 = Wdat;
           9:      St1 = Wdat;
           10:     St2 = Wdat;
           11:     St3 = Wdat;
           12:     St4 = Wdat;
           13:     St5 = Wdat;
           14:     St6 = Wdat;
           15:     St7 = Wdat;
           16:     Ss0 = Wdat;
           17:     Ss1 = Wdat;
           18:     Ss2 = Wdat;
           19:     Ss3 = Wdat;
           20:     Ss4 = Wdat;
           21:     Ss5 = Wdat;
           22:     Ss6 = Wdat;
           23:     Ss7 = Wdat;
           24:     St8 = Wdat;
           25:     St9 = Wdat;
           26:     Sk0 = Wdat;
           27:     Sk1 = Wdat;
           28:     Sgp = Wdat;
           29:     Sfp = Wdat;
           30:     Ssp = Wdat;
           31:     Sra = Wdat;
       endcase
    end
endmodule
```

四、 实验结果分析

1. ALU:

The following table is the lab result (r input of ALU is 1111, s input of ALU is 2222) :

the status of switches	ALU operation	ALU result displayed in the LCD
000	AND	0000
001	OR	3333

010	ADD	3333
110	SUB	eeef
111	SLT	0001

The result indicates that the code functions correx

2. ALU Controller:

The following table is the lab result (r input of ALU is 1111, s input of ALU is 2222) :

the status of buttons	the status of switches	ALU operation	ALU result
00	xxxx	ADD	3333
01	xxxx	SUB	eeef
1x	0000	ADD	3333
1x	0010	SUB	eeef
1x	0100	AND	0000
1x	0101	OR	3333
1x	1010	SLT	0001

The result indicates that the code functions correctly.

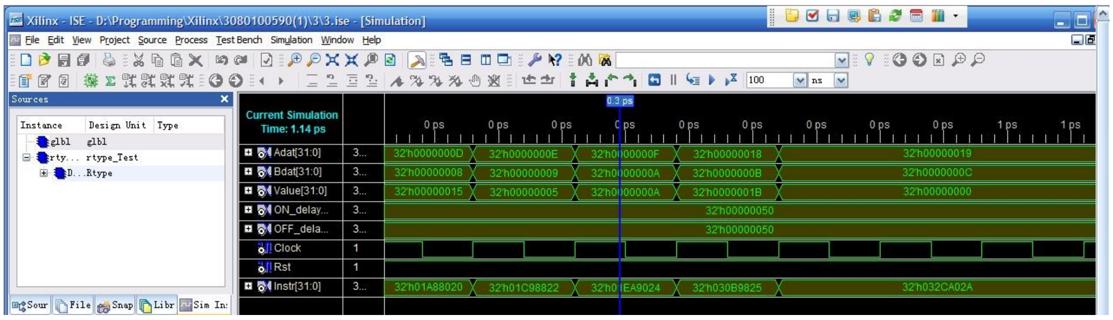
3. Register file:

The code of the test fixture file:

```
module test;
   // Inputs
   reg clk;
   reg rst;
   reg [31:0] instru;
   // Outputs
   wire [31:0] Adat;
   wire [31:0] Bdat;
   wire [31:0] result;
   parameter   ON_delay=80, OFF_delay=80;
   // Instantiate the Unit Under Test (UUT)
   Rtype uut (
      .clk(clk),
      .rst(rst),
      .instru(instru),
      .Adat(Adat),
      .Bdat(Bdat),
      .result(result)
   );

   always begin
      clk =1;
```

```
        #ON_delay;
        clk =0;
        #OFF_delay;
    end

    initial begin
        rst=1;
        instru=32'h01A88020;            //ADD  $S0  , $T5  , $T0
        #160   instru=32'h01C98822;     //SUB  $S1  , $T6  , $T1
        #160   instru=32'h01EA9024;     //AND  $S2  , $T7  , $T2
        #160   instru=32'h030B9825;     //OR   $S3  , $T8  , $T3
        #160   instru=32'h032CA02A;     //SLT  $S4  , $T9  , $T4
        #500   $dumpflush;
        $stop;
    end

endmodule
```

The following picture is the result of simulation:


Xilinx - ISE - D:\Programming\Xilinx\3080100590(1)\3\3.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Sources
Instance Design Unit Type
glbl glbl
rty... rtype_Test
D... Rtype
Current Simulation Time: 1.14 ps
0.3 ps
Adat[31:0] 3... 32'h0000000D 32'h0000000E 32'h0000000F 32'h00000018 32'h00000019
Bdat[31:0] 3... 32'h00000008 32'h00000009 32'h0000000A 32'h0000000B 32'h0000000C
Value[31:0] 3... 32'h00000015 32'h00000005 32'h0000000A 32'h0000001B 32'h00000000
ON_delay... 3... 32'h00000050
OFF_dela... 3... 32'h00000050
Clock 1
Rst 1
Instr[31:0] 3... 32'h01A88020 32'h01C98822 32'h01EA9024 32'h030B9825 32'h032CA02A

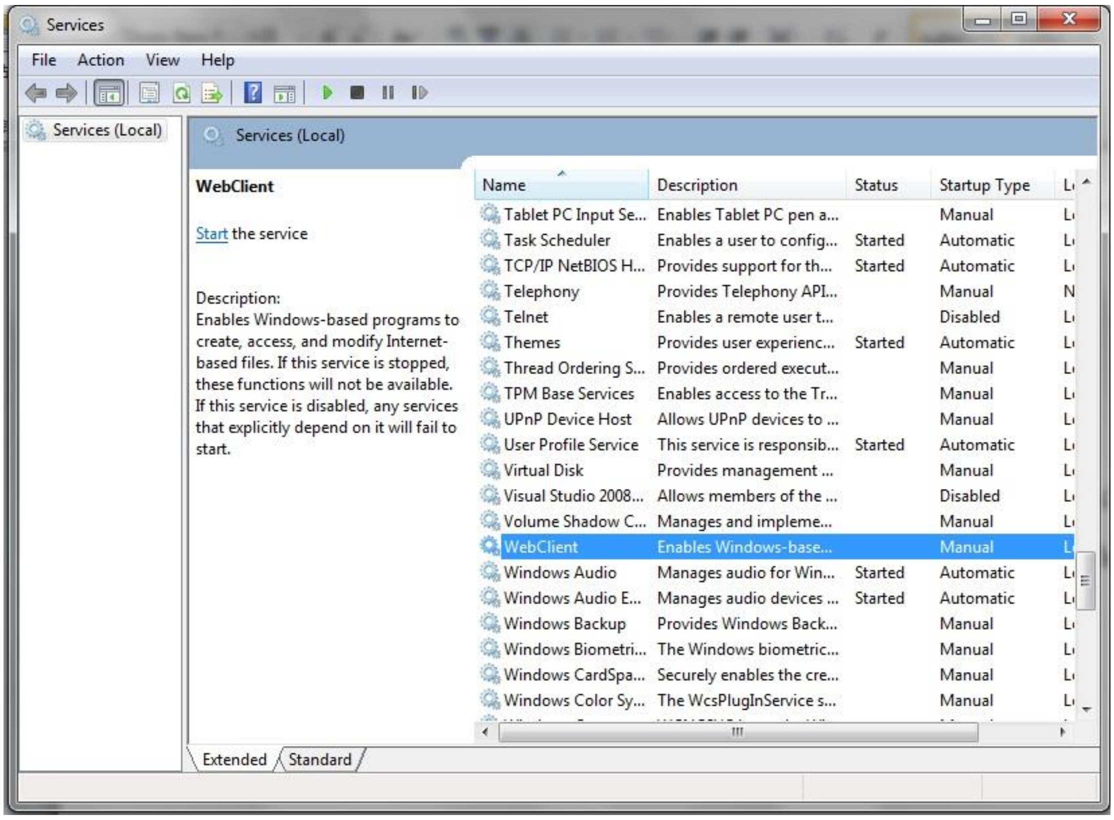
The result indicates that the code functions correctly.

五、 讨论与心得

The task of this lab is not hard and I've finished coding early, but I've met some other troubles.

At first I couldn't pass the "Map" step. Then I checked my error and found that I've set a wrong device which led to the definition in the UCF file is not correspondent with the actual hardware port. So I corrected the device to XC3S500E and got the correct result.

Another trouble is the error I met when running simulation. I've checked my codes many times and made sure there was nothing wrong in it, so I searched the error in the Internet. The solution is to open the WebClient services:


Services

File Action View Help

Services (Local)

Services (Local)

WebClient

Start the service

Description:
Enables Windows-based programs to create, access, and modify Internet-based files. If this service is stopped, these functions will not be available. If this service is disabled, any services that explicitly depend on it will fail to start.

Name	Description	Status	Startup Type	L
Tablet PC Input Se...	Enables Tablet PC pen a...		Manual	L
Task Scheduler	Enables a user to config...	Started	Automatic	L
TCP/IP NetBIOS H...	Provides support for th...	Started	Automatic	L
Telephony	Provides Telephony API...		Manual	N
Telnet	Enables a remote user t...		Disabled	L
Themes	Provides user experienc...	Started	Automatic	L
Thread Ordering S...	Provides ordered execut...		Manual	L
TPM Base Services	Enables access to the Tr...		Manual	L
UPnP Device Host	Allows UPnP devices to ...		Manual	L
User Profile Service	This service is responsib...	Started	Automatic	L
Virtual Disk	Provides management ...		Manual	L
Visual Studio 2008...	Allows members of the ...		Disabled	L
Volume Shadow C...	Manages and impleme...		Manual	L
WebClient	Enables Windows-base...		Manual	L
Windows Audio	Manages audio for Win...	Started	Automatic	L
Windows Audio E...	Manages audio devices ...	Started	Automatic	L
Windows Backup	Provides Windows Back...		Manual	L
Windows Biometri...	The Windows biometric...		Manual	L
Windows CardSpa...	Securely enables the cre...		Manual	L
Windows Color Sy...	The WcsPlugInService s...		Manual	L

Extended / Standard

Because I have no root authority to open it, so I have to finished my simulation in another PC.