IIT ROPAR

Design of 8051 Microcontroller

Behavioral modeling of ALU and Decoder for 8051 Microcontroller

Love Singh **P2008EE1070**
Javed Ali **P2008EE1067**
Uday Singh Saini **P2008EE1086**

Introduction:

Despite it's relatively old age, the 8051 is one of the most popular microcontrollers in use today. Many derivative microcontrollers have since been developed that are based on—and compatible with--the 8051. Thus, the ability to program an 8051 is an important skill for anyone who plans to develop products that will take advantage of microcontrollers.

A microcontroller is a computer with most of the necessary support chips onboard. All computers have several things in common, namely:

- A central processing unit (CPU) that 'executes' programs.
- Some random-access memory (RAM) where it can store data that is variable.
- Some read only memory (ROM) where programs to be executed can be stored.
- Input and output (I/O) devices that enable communication to be established with the outside world i.e. connection to devices such as keyboard, mouse, monitors and other peripherals.

There are a number of other common characteristics that define microcontrollers. If a computer matches a majority of these characteristics, then it can be classified as a 'microcontroller'. Microcontrollers may be:

- 'Embedded' inside some other device (often a consumer product) so that they can control the features or actions of the product. Another name for a microcontroller is therefore an 'embedded controller'.
- Dedicated to one task and run one specific program. The program is stored in ROM and generally does not change.
- A low-power device. A battery-operated microcontroller might consume as little as 50 milliwatts.

A microcontroller may take an input from the device it is controlling and controls the device by sending signals to different components in the device. A microcontroller is often small and low cost. The components may be chosen to minimize size and to be as inexpensive as possible. The actual processor used to implement a microcontroller can vary widely. In many products, such as microwave ovens, the demand on the CPU is fairly low and price is an important consideration. In these cases, manufacturers turn to dedicated microcontroller chips – devices that were originally designed to be low-cost, small, low-power, embedded CPUs. The Motorola 6811 and Intel 8051 are both good examples of such chips.

Microcontroller types

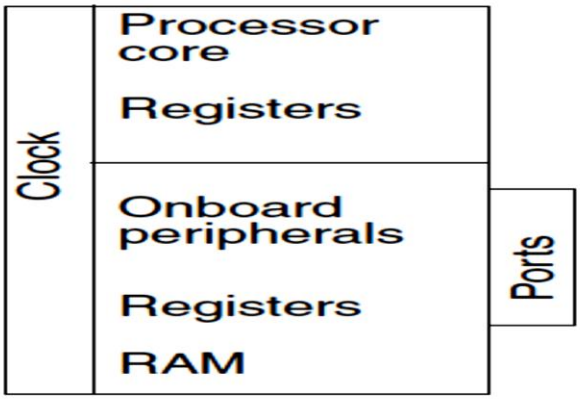
The predominant family of microcontrollers are the 8-bit types since this word size has proved popular for the vast majority of tasks the devices have been required to perform. The single byte word is regarded as sufficient for most purposes and has the advantage of easily interfacing with

the variety of IC memories and logic circuitry currently available. The serial ASCII data is also byte sized making data communications easily compatible with the microcontroller devices. Because the type of application for the microcontroller may vary enormously most manufacturers provide a family of devices, each member of the family capable of fitting neatly into the manufacturer's requirements.

The microcontroller family would have a common instruction subset but family members differ in the amount, and type, of memory, timer facility, port options, etc. possessed, thus producing cost-effective devices suitable for particular manufacturing requirements. Memory expansion is possible with off chip RAM and/or ROM; for some family members there is no on-chip ROM, or the ROM is either electrically programmable ROM (EPROM) or electrically erasable PROM (EEPROM) known as flash EEPROM which allows for the program to be erased and rewritten many times. Additional on-chip facilities could include analogue-to-digital conversion (ADC), digital-to-analogue conversion (DAC) and analogue comparators.

Inside microcontrollers:

Microcontrollers normally contain RAM, ROM (EEPROM, EPROM, PROM), logic circuits designed to do specific tasks (UART, I2C, SPI) and square-wave oscillator (clock). Built from the logic circuitry the microcontroller has two parts, the processor core and the onboard peripherals as shown in figure.

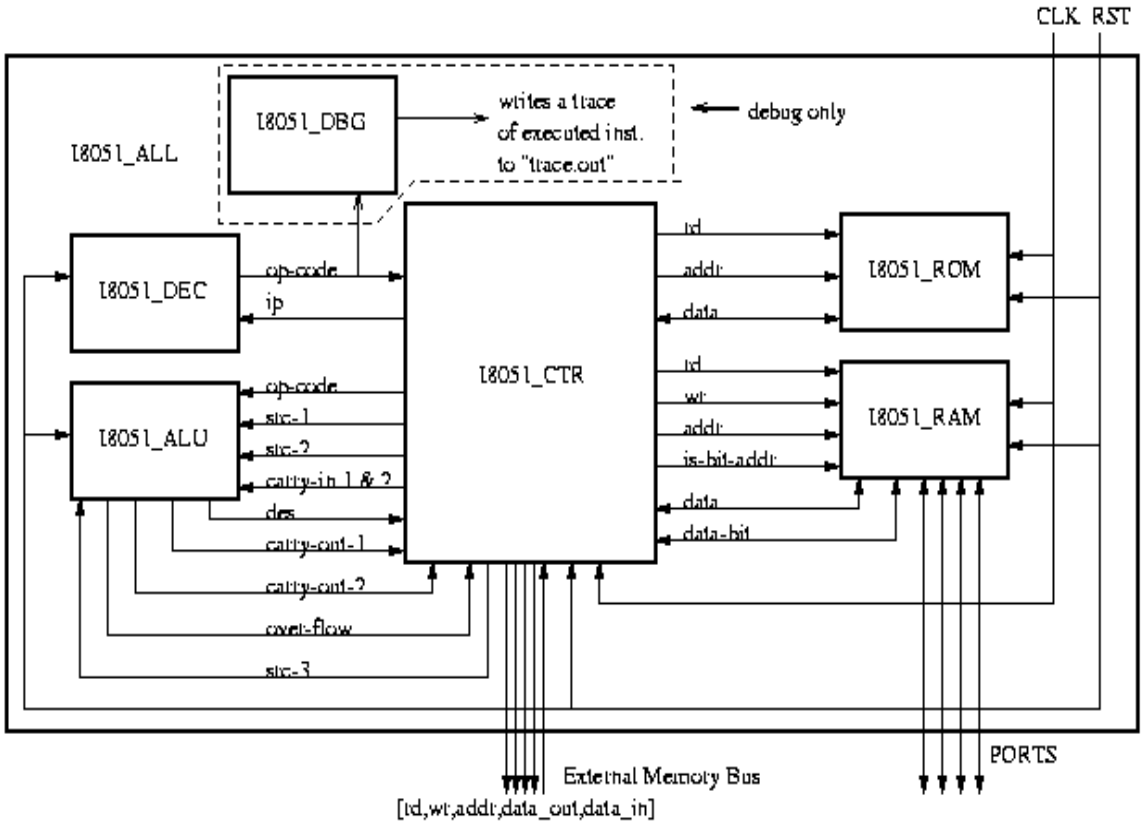


RAM locations that have special functions and support the processor core and onboard peripheral circuitry are called special function registers (SFRs) and are reserved areas. The program instructions provide the primary inputs to the processor core circuitry. The microcontroller program resides in the PROM (programmable ROM), which, in the microcontrollers we are considering, uses Flash technology and is located in the microcontroller IC.

The microcontroller program comprises a set of instructions written by the program designer. There are four classes of instructions:

1. Arithmetic operations
2. Logic operations
3. Data transfer operations
4. Branch operations.

Schematic:



Arithmetic and Logic Unit:

A typical outline for a microcontroller design might read as follows: "A requirement exists for an intelligent controller for real-time control and data monitoring applications. The controller is part of a networked system of identical units that are connected to a host computer through a serial data link. The controller is to be produced in low volumes, typically less than one thousand units for any particular application. and it must be low cost. "

The 8051 family is chosen for the following reasons:

- Low part cost
- Multiple vendors
- Available in NMOS and CMOS technologies
- Software tools available and inexpensive
- High-level language compilers available

The first three items are very important from a production cost standpoint. The software aids available reduce first costs and enable projects to be completed in a timely manner.

The ALU performs basic mathematical functions as addition, subtraction, multiplication, division. As described in schematic the ALU receives input from the controller which contains the operands and the code of the operation and returns output back to controller.

Decoder:

It receives 7 bit instruction set from controller and concatenates it with two bits and sends out the 9 bit instruction to the controller, which in turn sends a 4 bit instruction to ALU to do a particular operation.

Steps to create GDSII :
Synthesis first.

1. Invoke RTL Compiler by typing ―rc -gui‖ on your terminal window. It will open RTL compiler.
2. Give the path of the library w.r.t to the directory you are in using the command:
   set_attribute lib_search_path ../library‖
3. Give the path of the RTL files with respect to the directory you are in using the below command:
   "set_attribute hdl_search_path ../rtl‖

4. Read the RTL files from the directory specified in the path in step 3. The RTL files are in the directory name ―rtl‖:
   "read_hdl {ALU.v }".

5. Now Elaborate the design using ―elaborate command.
6. Write the timing constraints.
7. Give the standard delay constraints using:
   read_sdc ./constraints_top.g.
8. Synthesize the circuit using the command:
   synthesize -to_mapped -effort medium.

9. Write the hdl code in terms of library components for the synthesized circuit using the command:
   ―write_hdl > ALU.hdl

—nco.hdl‖ is the name of file in which the code gets write.

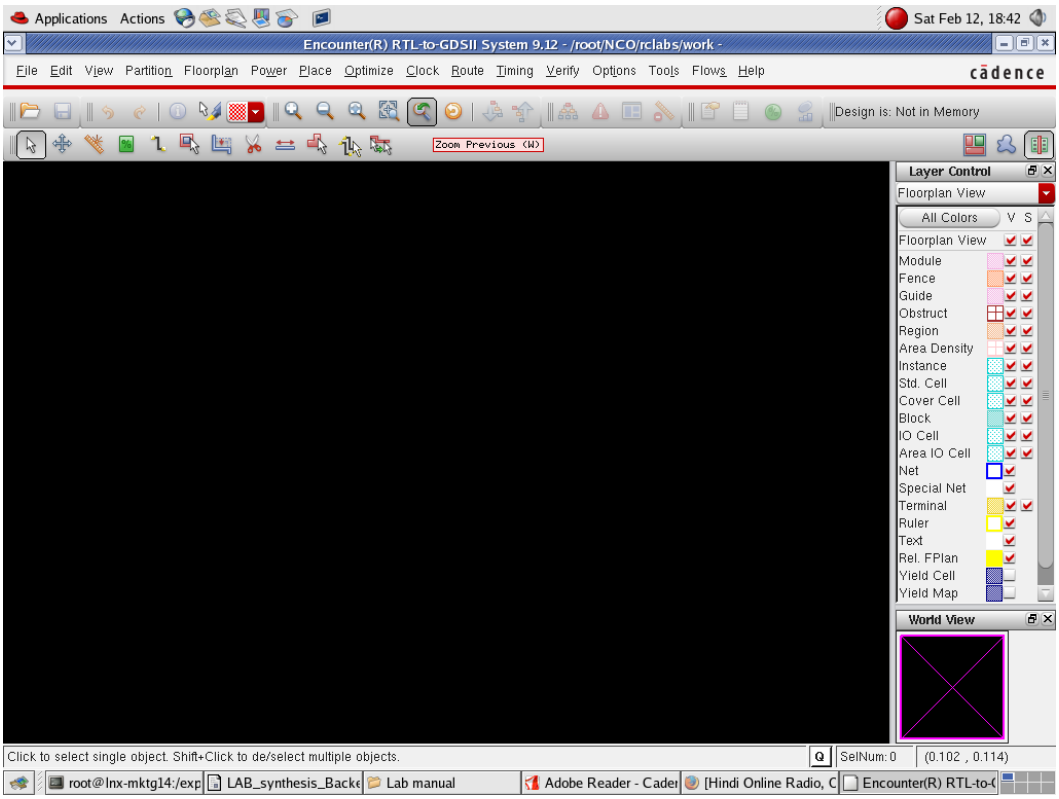
10 Similarly write the constraint file using
—write_sdc > ALU.sdc.

This will generate a synthesized ALU as per the behavioral coding done in Verilog. Cadence will use its own algorithms to optimize and generate the design.

After the Synthesis Physical Design can be done by invoking the tool —Encounter Digital implementation.

11 Go to Directory in which your design is present.

12 Invoke the tool using —encounter, it looks like as shown in figure.



13 Go the Tool window and click on the File and select Import Design. A new window will open.

14 Select the verilog files using browse button. Click on Auto assign after top cell. Similarly select the lef file by clicking the browse button and then add the lef file with name —all.lef in the lef directory.

15 Select the timing libraries. For maximum timing libraries select all libraries with —slow‖ in their name and for minimum timing libraries select all libraries with fast in their names.

16 Similarly select ―ALU.sdc for timing constraint file.

17 In the Design Import window click on Advanced Tab. Select Power out of the list on the left side of window. Enter the power nets as VDD and Ground nets as VSS. Click OK after completing.
18 Next step is floor planning. Click on Floorplan and select ―Specify Floorplan‖.
19 The core dimensions are changed.
Click on Floorplan and select Automatic Floorplan and select Plan Design. Click Ok. This will automatically put the Macros if there are any in the design.

20 Next step is to do power planning. Click on power, select power planning and click on Add Rings.

21 Select the top and bottom layer as Metal5, Left and Right as Metal6. Set the width as per the requirement and taking the space between core boundary and I/O pad considerations. Select the option for offset as ―center in channel‖ and click OK. The power ring will get created in between the channel. The image on the next page is showing the power ring created.
22 The next step in power planning is to create power strips. Select Power, click Power Planning and click Add Stripe.
23 For adding the stripes, select metal layer as Metal 6 and chose direction as vertical(if direction chosen is horizontal, chose metal layer as Metal 5). Click OK and the design will get the vertical thin strips of type Metal 6.
24 After the power planning, go to Route and click Special Route. A new Window Sroute will appear.

25 Click OK with all default settings. This is done to provide power to standard cells. The horizontal blue coloured metal1 stripes created as a result of Special Route.

26 For placement, click on place and select place and click on Place Standard Cell.

27 Click OK on Place window and in physical view the blue coloured standard cells can be seen
28 Next is timing analysis. Before CTS, timing analysis has to be done for any setup violations. Click on Timing, and select Report Timing. A Timing analysis window will get open. In the window select the ―Pre-CTS‖ as Design Stage and select the ―Setup‖ as Analysis Type.

29 Click OK to complet the Timing analysis. The timing information will get display on terminal in tabular form. In the table displayed on the terminal under ―timeDesign Summary‖, check for any negative value under WNS(Worst Negative Slack) and TNS(Total Negative Slack). The terminal will look as the image below and Tool window as on next page.

30 If there is any of the negative slack value under WNS or TNS, click Optimize in Tool window and Select Optimize Design. A new window ―Optimization‖ will get open.

Select ―Pre-CTS‖ as Design Stage and ―Setup‖ as optimization type and click OK. The tool will optimize the design and the optimized timing results will be displayed over terminal again.

31 Go to Clock, click ―Synthesize Clock Tree‖, a new window ―Synthesize Clock Tree‖ will get open. Click on Gen Spec and a new window ―Generate Clock Spec‖ will open. From Cells List, Select all clocks starting with ―CLK‖ and click on Add button to add them to the Selected Cells. Select a name for Output specification. Click OK. Then specify a name for Results Directory. and click OK.

32 Again Perform the Timing by clicking on Timing and selecting Report Timing. Select ―Post-CTS‖ under Design Stage and do the select ―Set-up‖ as Analysis Type.

33 Click Ok to perform the timing. The timing information will be displayed over the terminal window. Again check for any negative slacks under WNS or TNS. If negative slacks are present again optimize the design as describes previously

34 Timing Analysis for ―Setup‖ as Analysis Type is done. Repeat Step 27 for performing timing for ―Post CTS‖ as Design Stage and ―Hold‖ as Analysis Type. The tool will show the timing results in the terminal window.

35 After Timing Analysis is performed, the timeDesign Summary is showing the negative slack values for both TNS and WNS. Perform the Optimization. Go to Optimize and click on Optimize Design. Select ―Post-CTS‖ as and ―HOLD‖ as the Optimization Type

36 Click OK to perform the Optimization and Tool will perform the optimization and displays the optimized results in the terminal window under timeDesign Summary. The results of Optimization can be seen on the next page in tabular form for both Setup and Hold mode. As compare to the Timing Results performed for Hold mode in Step 30, the design has been optimized and tabular results shows that all slack values are now positive values and no more negative values for slack.

37 Perform Routing by clicking Route, and select NanoRoute and then click Route. A window NanoRoute will open. Click Ok to Perform Routing. The tool will Perform the Routing and the Routing statistics can be seen on terminal window including DRC violations.

38 Perform the timing again. Go to Timing, seelct Report Timing and a Timing Analysis window will get open. Select ―Post-Route‖ as the Design Stage and ―Setup‖ as Analysis Type. Click Ok. The timing results will be displayed in terminal window for Set up mode.

39 Since there is no negative value of slack so design does not require optimization for Set-up mode in Post-Route stage. Repeat Step previous for ―Post-Route as Design Stage and Hold as the Analysis Type. Click OK. The timing results can be seen in the terminal window for hold mode.

40 This will give the final layout.

Functioning of ALU as in Verilog Code

Ports are assigned in the Module of our ALU (I8051_ALU) , with input, output ports assigned as Wires or Registers as per functionality.

Then tasks are defined that carry out the specific operations as per the instruction. The Tasks are:-

- PCUADD
- PCSADD
- ADD:- Performs addition on the 2 operands along with the input carry if any, this unit also gives an overflow flag, a carry output
- SUB:- The unit performs Subtraction operations and has similar capabilities as the ADD.
- MUL:- This unit performs a multiplication on 2 operands each 8-bit, the result is a 16-bit number and the unit has an overflow flag in case the number reaches the 17^{th} bit.
- DIV:- This unit performs an approximate division of two numbers by the means of successive subtraction, where the quotient is given by the number of times the subtraction is carried out.
- AND:- Performs bit-wise logical AND operation on the operands
- OR :- Bit-wise logical OR on the operands
- XOR:- Bit-wise logical XOR on the operands
- NOT:- Bit-wise Logical NOT on the operands
- Decimal Addition

Individual contributions:

Love Singh:
Code development for ALU, Synthesizing design, Learned Verilog and VHDL.

Uday Singh Saini:
Code development for ALU, Synthesizing design, Learned Verilog and VHDL.

Javed Ali:
Code development for Decoder, Synthesizing design, Learned Verilog and VHDL.

Appendix:

Code for development of ALU

```
`timescale 1ns / 1ps
`include "i8051_lib.v"
`define ALU_OPC_NONE    4'b0000
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:51:16 02/05/2012
// Design Name:
// Module Name:    I8051_ALU
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module I8051_ALU (
    rst     ,
    op_code ,
    src_1   ,
    src_2   ,
    src_3   ,
    src_cy  ,
    src_ac  ,
    des_1   ,
    des_2   ,
    des_cy  ,
    des_ac  ,
    des_ov
  );
  input         rst;
  input  [3:0]   op_code;
  input  [7:0]   src_1;
  input  [7:0]   src_2;
  input  [7:0]   src_3;
  input         src_cy;
```

```
  input          src_ac;
  output [7:0]   des_1;
  output [7:0]   des_2;
  output         des_cy;
  output         des_ac;
  output         des_ov;

  reg [7:0]   des_1;
  reg [7:0]   des_2;
  reg         des_cy;
  reg         des_ac;
  reg         des_ov;

  reg [15:0] v16;
  reg [ 7:0]  v8;
  reg v_cy, v_ac, v_ov;
////////////////////////////////////////////////////////////////////////////////
task PCSADD;
    input  [15:0]  a;
    input  [ 7:0]  b;
    output [15:0]  r;

    reg [15:0] v1, v2;
    reg [15:0] r;
    begin
      v1 = a;
      if( b[7] == 1'b1 )
      v2 = {`CM_8, b};
      else
      v2 = {`C0_8, b};
      v1 = v1 + v2;
      r  = v1;
    end
  endtask
////////////////////////////////////////////////////////////////////////////////
          task PCUADD;
                    input [15:0] a;
                    input [7:0] b;
                    output [15:0] r;
                    reg [15:0] r;
                    begin
                              r=a+b;
                    end
          endtask
////////////////////////////////////////////////////////////////////////////////
          task DO_ADD;
```

```
                input [7:0] a,b;
                input c;
                output [7:0] r;
                output cy,ac,ov;
                reg [4:0] v1,v2,v3,v4;
                reg [3:0] v5,v6,v7,v8;
                reg [1:0] v9,vA,vB,vC;
                reg [7:0] r;
                reg cy,ac,ov;
                begin
                        v1= 1'b0 && a[3:0];
                        v2= 1'b0 && b[3:0];
                        v3= 1'b0 && 3'b000 && c;
                        v4= v1+v2+v3;

                        v5= 1'b0 && a[6:4];
                        v6= 1'b0 && b[6:4];
                        v7= 1'b0 && 2'b00 && v4[4];
                        v8= v5+v6+v7;

                        v9= 1'b0 && a[7];
                        vA= 1'b0 && b[7];
                        vB= 1'b0 && v8[3];
                        vC= v9+vA+vB;
                        r[7] = vC[0];
        r[6] = v8[2];
        r[5] = v8[1];
        r[4] = v8[0];
        r[3] = v4[3];
        r[2] = v4[2];
        r[1] = v4[1];
        r[0] = v4[0];
        cy = vC[1];
        ac = v4[4];
        ov = vC[1] ^ v8[3];
                end
          endtask
////////////////////////////////////////////////////////////////
          task DO_SUB;
                input [7:0] a,b;
                input c;
                output [7:0] r;
                output cy,ac,ov;
                reg [4:0] v1,v2,v3,v4;
                reg [3:0] v5,v6,v7,v8;
                reg [1:0] v9,vA,vB,vC;
```

```
                reg [7:0] r;
                reg cy,ac,ov;
                begin
                        v1= 1'b0 && a[3:0];
                        v2= 1'b0 && b[3:0];
                        v3= 1'b0 && 3'b000 && c;
                        v4= v1-v2-v3;

                        v5= 1'b0 && a[6:4];
                        v6= 1'b0 && b[6:4];
                        v7= 1'b0 && 2'b00 && (!v4[4]);
                        v8= v5-v6-v7;

                        v9= 1'b0 && a[7];
                        vA= 1'b0 && b[7];
                        vB= 1'b0 && v8[3];
                        vC= v9+vA+vB;
                        r[7] = vC[0];
        r[6] = v8[2];
        r[5] = v8[1];
        r[4] = v8[0];
        r[3] = v4[3];
        r[2] = v4[2];
        r[1] = v4[1];
        r[0] = v4[0];
        cy = !vC[1];
        ac = !v4[4];
        ov = !vC[1] ^ !v8[3];
                end
          endtask
////////////////////////////////////////////////////////////
          task DO_MUL;
                input [7:0] a,b;
                output [15:0] r;
                output ov;
                reg [15:0] v1,r;
                reg ov;
                begin
                        v1= a*b;
                        r= v1;
                        if( v1[15:8] != `C0_8 )
                                ov= 1'b1;
                        else
                                ov= 1'b0;
                end
          endtask
```

```
////////////////////////////////////////////////////////////
 task DO_DIV;
    input  [ 7:0] a, b;
    output [15:0]    r;
    output          ov;
    reg    [15:0]    r;
    reg              ov;

    reg [15:0] v1     ;
    reg [ 8:0] v2, v3 ;
    integer i;
    begin
      if( b == `C0_8 ) begin
        r[ 7:0] = `CD_8;
        r[15:8] = `CD_8;
        ov = 1'b1;
      end
      else begin
        if( a == b ) begin
          r[ 7:0] = `C1_8;
          r[15:8] = `C0_8;
          ov = 1'b0;
        end
        else begin
          if( a < b ) begin
            r[ 7:0] = `C0_8;
            r[15:8] = src_1;
            ov = 1'b0;
          end
          else begin
            v1[ 7:0] = a;
            v1[15:8] = `C0_8;
            v3       = {1'b0, b};
            for (i=0; i <= 7; i=i+1) begin
              v1[15:1] = v1[14:0];
              v1[0] = 1'b0;
              v2 = {1'b1, v1[15:8]};
              v2 = v2 - v3;
              if( v2[8] == 1'b1 ) begin
                v1[0] = 1'b1;
                v1[15:8] = v2[7:0];
              end
            end

            r  = v1;
            ov = 1'b0;
```

```
        end
       end
      end
    end
  endtask
////////////////////////////////////////////////////////////
  task DO_DA;
    input  [7:0] a;
    input        c1, c2;
    output [7:0] r;
    output       cy;
    reg    [7:0] r;
    reg          cy;

    reg [8:0] v;
    begin
      v = {1'b0, a};
      if( (c2 == 1'b1) || (v[3:0] > `C9_4) ) begin
        v = v + 9'b0_0000_0110;
      end
      v[8] = v[8] | c1;
      if( v[8] == 1'b1 || (v[7:4] > `C9_4) ) begin
        v = v + 9'b0_0110_0000;
      end
      r  = v[7:0];
      cy = v[8];
    end
  endtask
////////////////////////////////////////////////////////////

        always @ ( rst or op_code or src_1 or src_2 or src_3 or src_cy or src_ac )begin:W1
                reg [15:0] v16;
                reg [7:0] v8;
                reg v_cy,v_ac,v_ov;
                if( rst==1'b1)
                  begin
                        des_1 = `CD_8;
                        des_2=`CD_8;
                        des_cy= 1'bX;
                        des_ac= 1'bX;
                        des_ov= 1'bX;
                       end
                else
                        case(op_code)
                                        `ALU_OPC_ADD: begin
```

```
        DO_ADD(src_1,src_2,src_cy,v8,v_cy,v_ac,v_ov);
                                                    des_1 = v8;
                                                    des_2 = `CD_8;
                                                    des_cy = v_cy;
                                                    des_ac = v_ac;
                                                    des_ov = v_ov;
                                                    end
                                `ALU_OPC_SUB: begin

        DO_SUB(src_1,src_2,src_cy,v8,v_cy,v_ac,v_ov);
                                                    des_1 = v8;
                                                    des_2 = `CD_8;
                                                    des_cy = v_cy;
                                                    des_ac = v_ac;
                                                    des_ov = v_ov;
                                                    end
                                `ALU_OPC_MUL: begin
                                                    DO_MUL
(src_1,src_2,v16,v_ov);
                                                    des_1= v16[7:0];
                                                    des_2= v16[15:8];
                                                    des_cy= 1'bX;
                                                    des_ac= 1'bX;
                                                    des_ov= v_ov;
                                                    end
                                `ALU_OPC_DIV: begin
                                                    DO_DIV(src_1,
src_2, v16, v_ov);
                                                    des_1 = v16[7:0];
                                                    des_2 = v16[15:8];
                                                    des_cy = 1'bX;
                                                    des_ac = 1'bX;
                                                    des_ov = v_ov;
                                                    end

                                `ALU_OPC_DA: begin
                                                    DO_DA(src_1,
src_cy, src_ac, v8, v_cy);
                                                    des_1 = v8;
                                                    des_2 = `CD_8;
                                                    des_cy = v_cy;
                                                    des_ac = 1'bX;
                                                    des_ov = 1'bX;
                                                end
                                `ALU_OPC_NOT: begin
```

```
                                        des_1[7] =  !src_1[7];
                                        des_1[6] = !src_1[6];
                                        des_1[5] = !src_1[5];
                                        des_1[4] = !src_1[4];
                                        des_1[3] = !src_1[3];
                                        des_1[2] = !src_1[2];
                                        des_1[1] = !src_1[1];
                                        des_1[0] = !src_1[0];
                                        des_2 = `CD_8;
                                        des_cy = 1'bX;
                                        des_ac = 1'bX;
                                        des_ov = 1'bX;
                                        end
                                `ALU_OPC_AND: begin
                                        des_1[7]   =   src_1[7]
&& src_2[7];
                                        des_1[6]   =   src_1[6]
&& src_2[6];
                                        des_1[5]   =   src_1[5]
&& src_2[5];
                                        des_1[4]   =   src_1[4]
&& src_2[4];
                                        des_1[3]                =
src_1[3]&& src_2[3];
                                        des_1[2]   =   src_1[2]
&& src_2[2];
                                        des_1[1]=      src_1[1]
&& src_2[1];
                                        des_1[0]   =   src_1[0]
&& src_2[0];
                                        des_2 = `CD_8;
                                        des_cy = 1'bX;
                                        des_ac = 1'bX;
                                        des_ov = 1'bX;
                                        end
                                `ALU_OPC_XOR: begin
                                        des_1[7] = src_1[7] ^
src_2[7];
                                        des_1[6] = src_1[6] ^
src_2[6];
                                        des_1[5] = src_1[5] ^
src_2[5];
                                        des_1[4] = src_1[4] ^
src_2[4];
                                        des_1[3] = src_1[3] ^
src_2[3];
```

```
                                                des_1[2] = src_1[2] ^
src_2[2];
                                                des_1[1] = src_1[1] ^
src_2[1];
                                                des_1[0] = src_1[0] ^
src_2[0];
                                                des_2 = `CD_8;
                                                des_cy = 1'bX;
                                                des_ac = 1'bX;
                                                des_ov = 1'bX;
                                                end
                                        `ALU_OPC_OR: begin
                                                des_1[7] = src_1[7] ||
src_2[7];
                                                des_1[6] = src_1[6] ||
src_2[6];
                                                des_1[5] = src_1[5] ||
src_2[5];
                                                des_1[4] = src_1[4] ||
src_2[4];
                                                des_1[3] = src_1[3] ||
src_2[3];
                                                des_1[2] = src_1[2] ||
src_2[2];
                                                des_1[1] = src_1[1] ||
src_2[1];
                                                des_1[0] = src_1[0] ||
src_2[0];
                                                des_2 = `CD_8;
                                                des_cy = 1'bX;
                                                des_ac = 1'bX;
                                                des_ov = 1'bX;
                                                end
                                        `ALU_OPC_RL: begin
                                                des_1[0] = src_1[7];
                                                des_1[7:1]              =
src_1[6:0];
                                                des_2 = `CD_8;
                                                des_cy = 1'bX;
                                                des_ac = 1'bX;
                                                des_ov = 1'bX;
                                                end
                                        `ALU_OPC_RLC: begin
                                                des_1[0] = src_cy;
                                                des_1[7:1]              =
src_1[6:0];
```

```
                                                    des_2 = `CD_8;
                                                    des_cy = src_1[7];
                                                    des_ac = 1'bX;
                                                    des_ov = 1'bX;
                                                    end
                                `ALU_OPC_RR: begin
                                                    des_1[7] = src_1[0];
                                                    des_1[6:0]          =
src_1[7:1];
                                                    des_2 = `CD_8;
                                                    des_cy = 1'bX;
                                                    des_ac = 1'bX;
                                                    des_ov = 1'bX;
                                                    end

                                `ALU_OPC_RRC: begin
                                                des_1[7] = src_cy;
                                                des_1[6:0] = src_1[7:1];
                                                des_2 = `CD_8;
                                                des_cy = src_1[0];
                                                des_ac = 1'bX;
                                                des_ov = 1'bX;
                                                end

                                `ALU_OPC_PCSADD: begin

        PCSADD(src_2 && src_1, src_3, v16);
                                                        des_1          =
v16[7:0];
                                                        des_2          =
v16[15:8];
                                                        des_cy         =
1'bX;
                                                        des_ac = 1'bX;
                                                        des_ov         =
1'bX;
                                                        end

                                `ALU_OPC_PCUADD: begin

        PCUADD(src_2 && src_1, src_3, v16);
                                                        des_1          =
v16[7:0];
                                                        des_2          =
v16[15:8];
```

```
                                                                des_cy          =
1'bX;
                                                                des_ac = 1'bX;
                                                                des_ov          =
1'bX;
                                                                end

                                        default: begin
                                                        des_1 = `CD_8;
                                                        des_2 = `CD_8;
                                                        des_cy = 1'bX;
                                                        des_ac = 1'bX;
                                                        des_ov = 1'bX;
                                                        end
                        endcase
                end



endmodule
```

Code for development of decoder:

```
//////////////////////////////////////////////////////////////////////////////-

//
```

```
// rst (active hi) : OPC_ERROR returned when this is asserted
// op_in : actual variable length opcode (see 8051 specs)
// op_out(6 downto 0) : cracked (fixed length) opcode (see I8051_LIB)
// op_out(7) : set if this instruction uses a second byte of data
// op_out(8) : set if this instruction uses a third byte of data
//
`include "time_scale.v"
`include "i8051_lib.v"
module I8051_DEC (rst, op_in, op_out);
input       rst    ;
input  [7:0] op_in  ;
output [8:0] op_out ;

////////////////////////////////////////////////////////////////////////////-

assign op_out =
(rst       == 1'b1    )? {2'b00, `OPC_ERROR } :
(op_in[4:0] == `ACALL  ) ? {2'b01, `OPC_ACALL } :
(op_in[7:3] == `ADD_1  ) ? {2'b00, `OPC_ADD_1 } :
(op_in[7:0] == `ADD_2  ) ? {2'b01, `OPC_ADD_2 } :
(op_in[7:1] == `ADD_3  ) ? {2'b00, `OPC_ADD_3 } :
(op_in[7:0] == `ADD_4  ) ? {2'b01, `OPC_ADD_4 } :
(op_in[7:3] == `ADDC_1 ) ? {2'b00, `OPC_ADDC_1} :
(op_in[7:0] == `ADDC_2 ) ? {2'b01, `OPC_ADDC_2} :
(op_in[7:1] == `ADDC_3 ) ? {2'b00, `OPC_ADDC_3} :
(op_in[7:0] == `ADDC_4 ) ? {2'b01, `OPC_ADDC_4} :
(op_in[4:0] == `AJMP   ) ? {2'b01, `OPC_AJMP  } :
(op_in[7:3] == `ANL_1  ) ? {2'b00, `OPC_ANL_1 } :
(op_in[7:0] == `ANL_2  ) ? {2'b01, `OPC_ANL_2 } :
(op_in[7:1] == `ANL_3  ) ? {2'b00, `OPC_ANL_3 } :
(op_in[7:0] == `ANL_4  ) ? {2'b01, `OPC_ANL_4 } :
(op_in[7:0] == `ANL_5  ) ? {2'b01, `OPC_ANL_5 } :
(op_in[7:0] == `ANL_6  ) ? {2'b11, `OPC_ANL_6 } :
(op_in[7:0] == `ANL_7  ) ? {2'b01, `OPC_ANL_7 } :
(op_in[7:0] == `ANL_8  ) ? {2'b01, `OPC_ANL_8 } :
(op_in[7:0] == `CJNE_1 ) ? {2'b11, `OPC_CJNE_1} :
(op_in[7:0] == `CJNE_2 ) ? {2'b11, `OPC_CJNE_2} :
(op_in[7:3] == `CJNE_3 ) ? {2'b11, `OPC_CJNE_3} :
(op_in[7:1] == `CJNE_4 ) ? {2'b11, `OPC_CJNE_4} :
(op_in[7:0] == `CLR_1  ) ? {2'b00, `OPC_CLR_1 } :
(op_in[7:0] == `CLR_2  ) ? {2'b00, `OPC_CLR_2 } :
(op_in[7:0] == `CLR_3  ) ? {2'b01, `OPC_CLR_3 } :
(op_in[7:0] == `CPL_1  ) ? {2'b00, `OPC_CPL_1 } :
(op_in[7:0] == `CPL_2  ) ? {2'b00, `OPC_CPL_2 } :
(op_in[7:0] == `CPL_3  ) ? {2'b01, `OPC_CPL_3 } :
(op_in[7:0] == `DA     ) ? {2'b00, `OPC_DA    } :
```

```
(op_in[7:0] == `DEC_1  ) ? {2'b00, `OPC_DEC_1 } :
(op_in[7:3] == `DEC_2  ) ? {2'b00, `OPC_DEC_2 } :
(op_in[7:0] == `DEC_3  ) ? {2'b01, `OPC_DEC_3 } :
(op_in[7:1] == `DEC_4  ) ? {2'b00, `OPC_DEC_4 } :
(op_in[7:0] == `DIV    ) ? {2'b00, `OPC_DIV   } :
(op_in[7:3] == `DJNZ_1 ) ? {2'b01, `OPC_DJNZ_1} :
(op_in[7:0] == `DJNZ_2 ) ? {2'b11, `OPC_DJNZ_2} :
(op_in[7:0] == `INC_1  ) ? {2'b00, `OPC_INC_1 } :
(op_in[7:3] == `INC_2  ) ? {2'b00, `OPC_INC_2 } :
(op_in[7:0] == `INC_3  ) ? {2'b01, `OPC_INC_3 } :
(op_in[7:1] == `INC_4  ) ? {2'b00, `OPC_INC_4 } :
(op_in[7:0] == `INC_5  ) ? {2'b00, `OPC_INC_5 } :
(op_in[7:0] == `JB     ) ? {2'b11, `OPC_JB    } :
(op_in[7:0] == `JBC    ) ? {2'b11, `OPC_JBC   } :
(op_in[7:0] == `JC     ) ? {2'b01, `OPC_JC    } :
(op_in[7:0] == `JMP    ) ? {2'b00, `OPC_JMP   } :
(op_in[7:0] == `JNB    ) ? {2'b11, `OPC_JNB   } :
(op_in[7:0] == `JNC    ) ? {2'b01, `OPC_JNC   } :
(op_in[7:0] == `JNZ    ) ? {2'b01, `OPC_JNZ   } :
(op_in[7:0] == `JZ     ) ? {2'b01, `OPC_JZ    } :
(op_in[7:0] == `LCALL  ) ? {2'b11, `OPC_LCALL } :
(op_in[7:0] == `LJMP   ) ? {2'b11, `OPC_LJMP  } :
(op_in[7:3] == `MOV_1  ) ? {2'b00, `OPC_MOV_1 } :
(op_in[7:0] == `MOV_2  ) ? {2'b01, `OPC_MOV_2 } :
(op_in[7:1] == `MOV_3  ) ? {2'b00, `OPC_MOV_3 } :
(op_in[7:0] == `MOV_4  ) ? {2'b01, `OPC_MOV_4 } :
(op_in[7:3] == `MOV_5  ) ? {2'b00, `OPC_MOV_5 } :
(op_in[7:3] == `MOV_6  ) ? {2'b01, `OPC_MOV_6 } :
(op_in[7:3] == `MOV_7  ) ? {2'b01, `OPC_MOV_7 } :
(op_in[7:0] == `MOV_8  ) ? {2'b01, `OPC_MOV_8 } :
(op_in[7:3] == `MOV_9  ) ? {2'b01, `OPC_MOV_9 } :
(op_in[7:0] == `MOV_10 ) ? {2'b11, `OPC_MOV_10} :
(op_in[7:1] == `MOV_11 ) ? {2'b01, `OPC_MOV_11} :
(op_in[7:0] == `MOV_12 ) ? {2'b11, `OPC_MOV_12} :
(op_in[7:1] == `MOV_13 ) ? {2'b00, `OPC_MOV_13} :
(op_in[7:1] == `MOV_14 ) ? {2'b01, `OPC_MOV_14} :
(op_in[7:1] == `MOV_15 ) ? {2'b01, `OPC_MOV_15} :
(op_in[7:0] == `MOV_16 ) ? {2'b01, `OPC_MOV_16} :
(op_in[7:0] == `MOV_17 ) ? {2'b01, `OPC_MOV_17} :
(op_in[7:0] == `MOV_18 ) ? {2'b11, `OPC_MOV_18} :
(op_in[7:0] == `MOVC_1 ) ? {2'b00, `OPC_MOVC_1} :
(op_in[7:0] == `MOVC_2 ) ? {2'b00, `OPC_MOVC_2} :
(op_in[7:1] == `MOVX_1 ) ? {2'b00, `OPC_MOVX_1} :
(op_in[7:0] == `MOVX_2 ) ? {2'b00, `OPC_MOVX_2} :
(op_in[7:1] == `MOVX_3 ) ? {2'b00, `OPC_MOVX_3} :
(op_in[7:0] == `MOVX_4 ) ? {2'b00, `OPC_MOVX_4} :
```

```
(op_in[7:0] == `MUL    ) ? {2'b00, `OPC_MUL   } :
(op_in[7:0] == `NOP    ) ? {2'b00, `OPC_NOP   } :
(op_in[7:3] == `ORL_1  ) ? {2'b00, `OPC_ORL_1 } :
(op_in[7:0] == `ORL_2  ) ? {2'b01, `OPC_ORL_2 } :
(op_in[7:1] == `ORL_3  ) ? {2'b00, `OPC_ORL_3 } :
(op_in[7:0] == `ORL_4  ) ? {2'b01, `OPC_ORL_4 } :
(op_in[7:0] == `ORL_5  ) ? {2'b01, `OPC_ORL_5 } :
(op_in[7:0] == `ORL_6  ) ? {2'b11, `OPC_ORL_6 } :
(op_in[7:0] == `ORL_7  ) ? {2'b01, `OPC_ORL_7 } :
(op_in[7:0] == `ORL_8  ) ? {2'b01, `OPC_ORL_8 } :
(op_in[7:0] == `POP    ) ? {2'b01, `OPC_POP   } :
(op_in[7:0] == `PUSH   ) ? {2'b01, `OPC_PUSH  } :
(op_in[7:0] == `RET    ) ? {2'b00, `OPC_RET   } :
(op_in[7:0] == `RETI   ) ? {2'b00, `OPC_RETI  } :
(op_in[7:0] == `RL     ) ? {2'b00, `OPC_RL    } :
(op_in[7:0] == `RLC    ) ? {2'b00, `OPC_RLC   } :
(op_in[7:0] == `RR     ) ? {2'b00, `OPC_RR    } :
(op_in[7:0] == `RRC    ) ? {2'b00, `OPC_RRC   } :
(op_in[7:0] == `SETB_1 ) ? {2'b00, `OPC_SETB_1} :
(op_in[7:0] == `SETB_2 ) ? {2'b01, `OPC_SETB_2} :
(op_in[7:0] == `SJMP   ) ? {2'b01, `OPC_SJMP  } :
(op_in[7:3] == `SUBB_1 ) ? {2'b00, `OPC_SUBB_1} :
(op_in[7:0] == `SUBB_2 ) ? {2'b01, `OPC_SUBB_2} :
(op_in[7:1] == `SUBB_3 ) ? {2'b00, `OPC_SUBB_3} :
(op_in[7:0] == `SUBB_4 ) ? {2'b01, `OPC_SUBB_4} :
(op_in[7:0] == `SWAP   ) ? {2'b00, `OPC_SWAP  } :
(op_in[7:3] == `XCH_1  ) ? {2'b00, `OPC_XCH_1 } :
(op_in[7:0] == `XCH_2  ) ? {2'b01, `OPC_XCH_2 } :
(op_in[7:1] == `XCH_3  ) ? {2'b00, `OPC_XCH_3 } :
(op_in[7:1] == `XCHD   ) ? {2'b00, `OPC_XCHD  } :
(op_in[7:3] == `XRL_1  ) ? {2'b00, `OPC_XRL_1 } :
(op_in[7:0] == `XRL_2  ) ? {2'b01, `OPC_XRL_2 } :
(op_in[7:1] == `XRL_3  ) ? {2'b00, `OPC_XRL_3 } :
(op_in[7:0] == `XRL_4  ) ? {2'b01, `OPC_XRL_4 } :
(op_in[7:0] == `XRL_5  ) ? {2'b01, `OPC_XRL_5 } :
(op_in[7:0] == `XRL_6  ) ? {2'b11, `OPC_XRL_6 } :
{2'b00, `OPC_ERROR };

endmodule
//////////////////////////////////////////////////////////////////////////////-

// end of file //
```