NUS
National University of Singapore

Department of Electrical and Computer Engineering

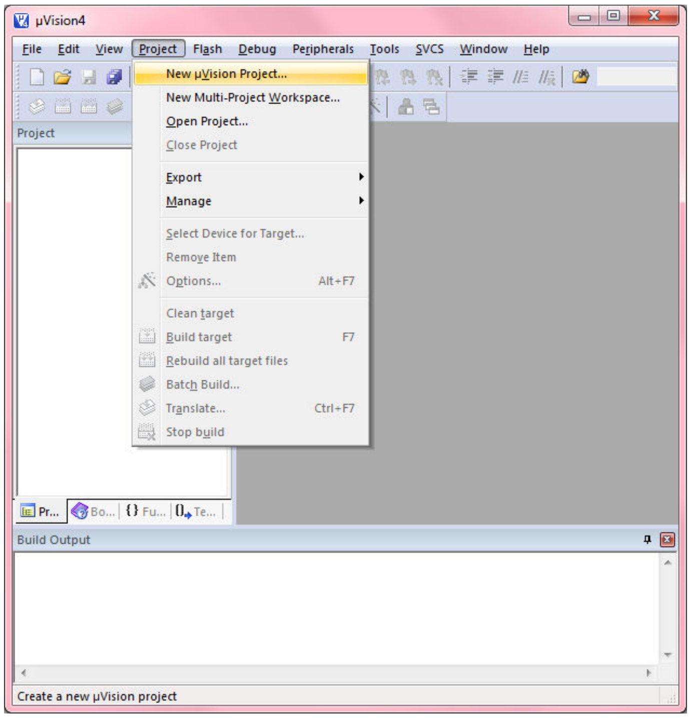
CG3207: COMPUTER ARCHITECTURE

C51 Cross Compiler to VHDL

C51 Cross Compiler:
This manual briefly illustrates how to use a C51 cross compiler to create hexadecimal codes for use in the ROM of the VHDL project. Note that this is a quickstart version of the **C51 Cross Compiler** and it is highly recommended that you understand the latter before reading this manual.

After going through this manual, you should be able to start creating your own code in C and obtain the hexadecimal format code for use in the ROM file. Together with the **CG3207 VHDL Refresher** manual, you should be able to download your program to the FPGA and show its function.

Brief Introduction to C51 Cross Compiler



Create a new µVision project

1. Download the Keil C51 compiler from the following web address:

   https://www.keil.com/download/product/

   The above website requires free registration. Alternatively, a copy (***c51v905***) has been uploaded to IVLE for your convenience.

2. Proceed with the installation of the program.

3. After the installation, launch the program "***Keil uVision4***".

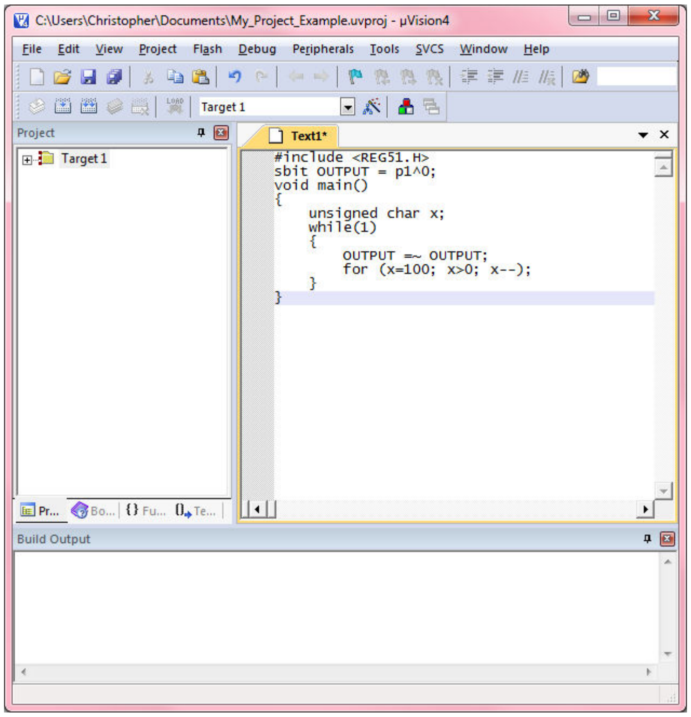
4. Go to Project → New uVision Project and enter your project file name

5. In the select device window, Choose:

   Atmel → T80C51

   Leave the other settings as default. They can be changed depending on your requirement later on.

6. When asked to "Copy Standard 8051 Startup code...", choose ***No***. The other option is also correct, and will lead to a change in the order of instructions created in the hex file later on.


C:\Users\Christopher\Documents\My_Project_Example.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Target 1
Text1*
Pr... Bo... Fu... Te...
Build Output

```
#include <REG51.H>
sbit OUTPUT = p1^0;
void main()
{
   unsigned char x;
   while(1)
   {
      OUTPUT =~ OUTPUT;
      for (x=100; x>0; x--);
   }
}
```

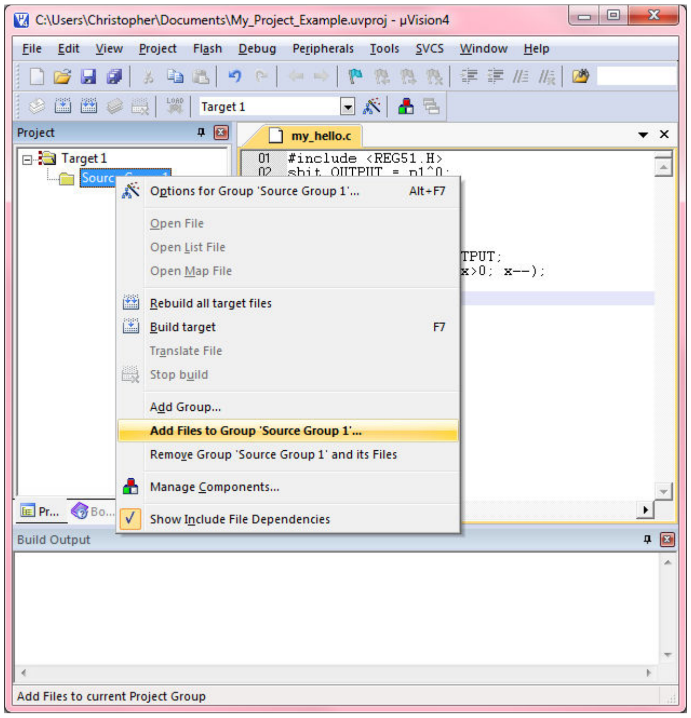
7. Go to File → New

8. Type your desired code. For example:

```
#include <REG51.H>
sbit OUTPUT = P1^0;
void main()
{
      unsigned char x;
      while(1)
      {
                  OUTPUT =~ OUTPUT;
                  for (x=100; x>0; x--);
      }
}
```

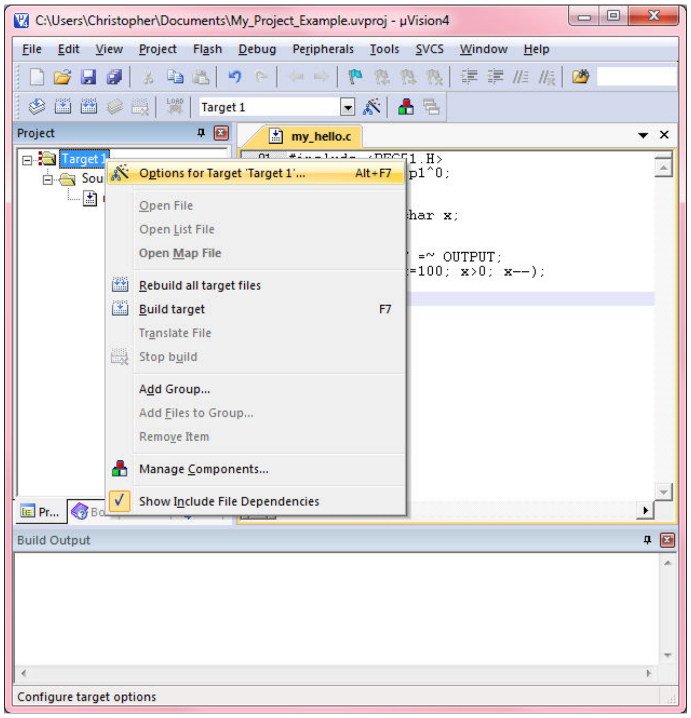
9. Go to File → Save

   For example, save the text as: ***my_hello.c***

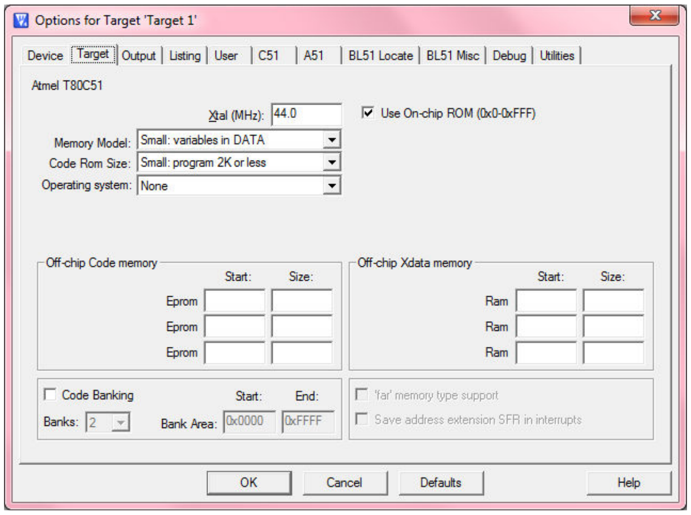

C:\Users\Christopher\Documents\My_Project_Example.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
my_hello.c
Options for Group 'Source Group 1'... Alt+F7
Open File
Open List File
Open Map File
Rebuild all target files
Build target F7
Translate File
Stop build
Add Group...
Add Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files
Manage Components...
Show Include File Dependencies
Build Output
Add Files to current Project Group

10. Expand ***Target 1***, right click on ***Source Group 1*** and select ***Add Files to Group 'Source Group 1'***

11. Add the file that you have saved, for example: ***my_hello.c***



12. Right click on ***Target 1*** and click on ***Options for Target 'Target 1'***



Options for Target 'Target 1'

Device | Target | Output | Listing | User | C51 | A51 | BL51 Locate | BL51 Misc | Debug | Utilities

Atmel T80C51

Xtal (MHz): 44.0

[✓] Use On-chip ROM (0x0-0xFFF)

Memory Model: Small: variables in DATA

Code Rom Size: Small: program 2K or less

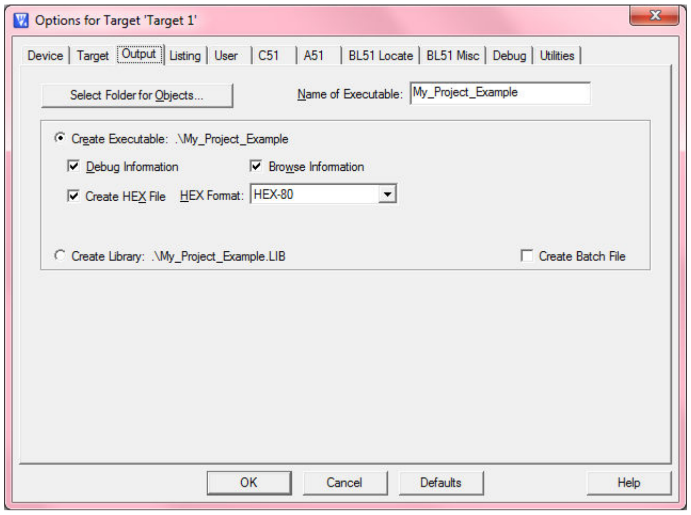
Operating system: None

[ ] Code Banking — Banks: 2 — Bank Area: Start: 0x0000 End: 0xFFFF

[ ] 'far' memory type support

[ ] Save address extension SFR in interrupts

13. Set the settings for ***Target*** as shown in the screenshot. You can change them later on depending on your needs.



Options for Target 'Target 1'

Device | Target | Output | Listing | User | C51 | A51 | BL51 Locate | BL51 Misc | Debug | Utilities

Name of Executable: My_Project_Example

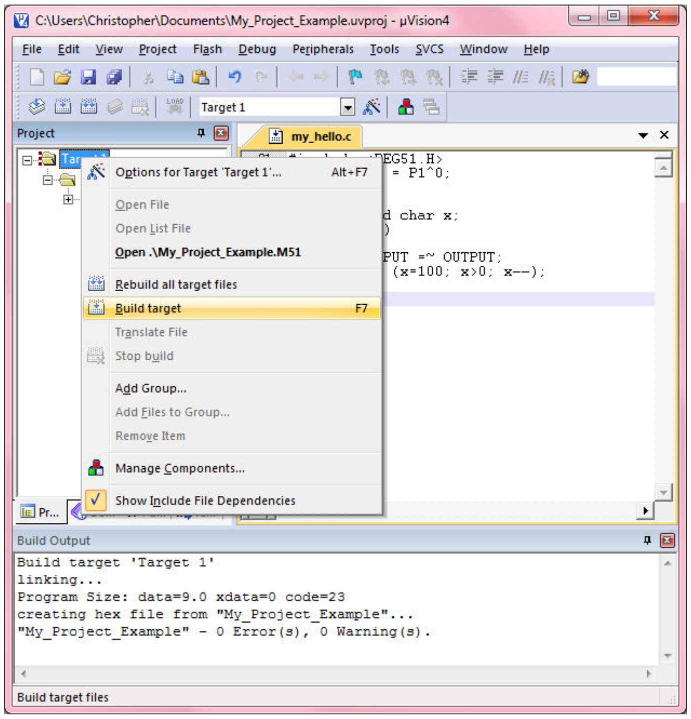
(●) Create Executable: .\My_Project_Example

[✓] Debug Information [✓] Browse Information

[✓] Create HEX File HEX Format: HEX-80

( ) Create Library: .\My_Project_Example.LIB [ ] Create Batch File

14. Set the settings for ***Output*** as shown in the screenshot. You can change them later on depending on your needs.



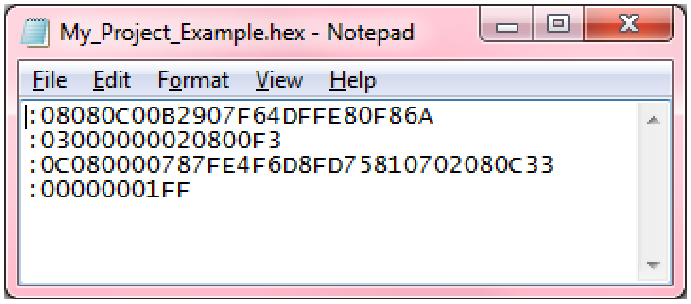
C:\Users\Christopher\Documents\My_Project_Example.uvproj - µVision4

Build Output

```
Build target 'Target 1'
linking...
Program Size: data=9.0 xdata=0 code=23
creating hex file from "My_Project_Example"...
"My_Project_Example" - 0 Error(s), 0 Warning(s).
```

15. Right click on ***Target 1*** and click on ***Build target***

Note that the compiler is case sensitive. If port 1 is defined in lowercase as ***p1^0***, instead of the uppercase ***P1^0***, error messages will appear.

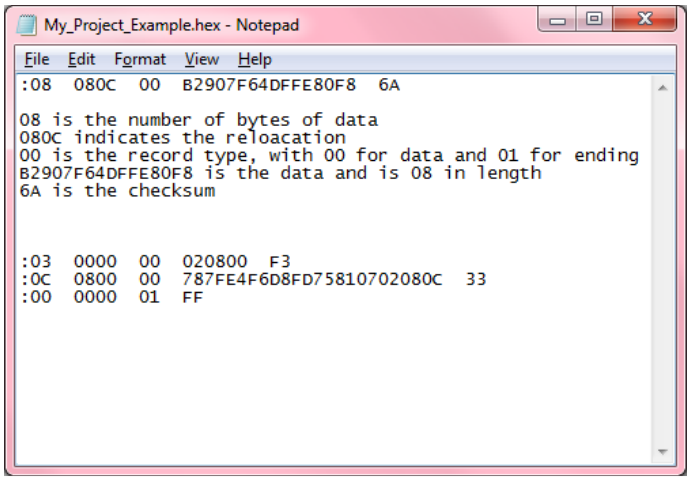


```
:08080C00B2907F64DFFE80F86A
:03000000020800F3
:0C080000787FE4F6D8FD7581070208OC33
:00000001FF
```

16. The hex file will be created in the folder where you created your project. You can use notepad to open it, and a screen similar to the screenshot will be shown.

    The format is that of Intel HEX. Further information is available on:

    http://en.wikipedia.org/wiki/Intel_HEX

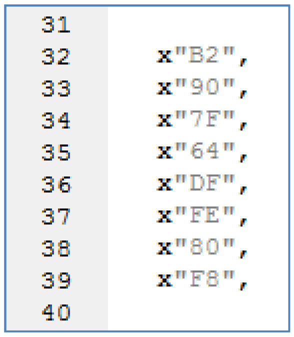


```
:08  080C  00  B2907F64DFFE80F8  6A

08 is the number of bytes of data
080C indicates the reloacation
00 is the record type, with 00 for data and 01 for ending
B2907F64DFFE80F8 is the data and is 08 in length
6A is the checksum


:03  0000  00  020800  F3
:0C  0800  00  787FE4F6D8FD7581070208OC  33
:00  0000  01  FF
```

17. For understanding purposes, some comments have been added as shown in the screenshot.

    The other lines of codes are created depending on the device chosen.

    For this project, it is not required to burn the program directly to the microprocessor. Hence, only the data potion will be used and the hex code related to the initial value of the program counter is not utilised.



```
31
32      x"B2",
33      x"90",
34      x"7F",
35      x"64",
36      x"DF",
37      x"FE",
38      x"80",
39      x"F8",
40
```

18. The data code can be copied directly to the ROM, just as you would for testing your individual instructions in binary format.

    Note the format for hex codes.

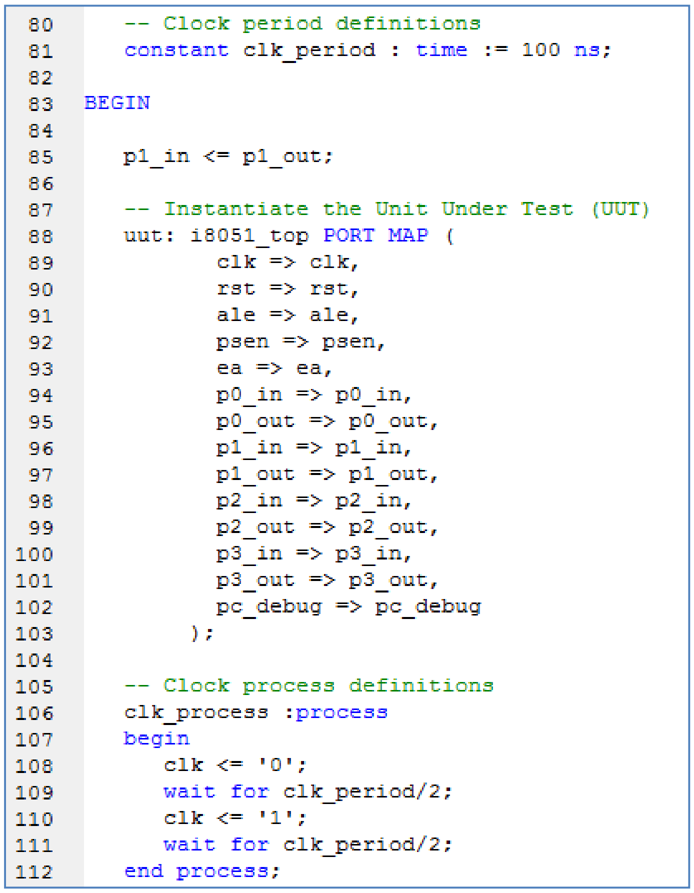
    Details on the actual mnemonic being used is shown on page 26 in the **C51 Cross Compiler.**

    The ports available for use include P0, P1, P2, P3.

    In this example, port P1^0 was used, and the address is x90 as shown on line 33 and as indicated in the programmer's guide. When running the VHDL simulation test, you should notice that P1^0 is being complemented while in the loop.

    In case any of the instructions being used has logical errors, the end result will not be as expected, and your instructions has to be reverified.

    To make it easier to read the hexadecimal codes, you might want to check page 179-184 of ***The C51 Primer***.



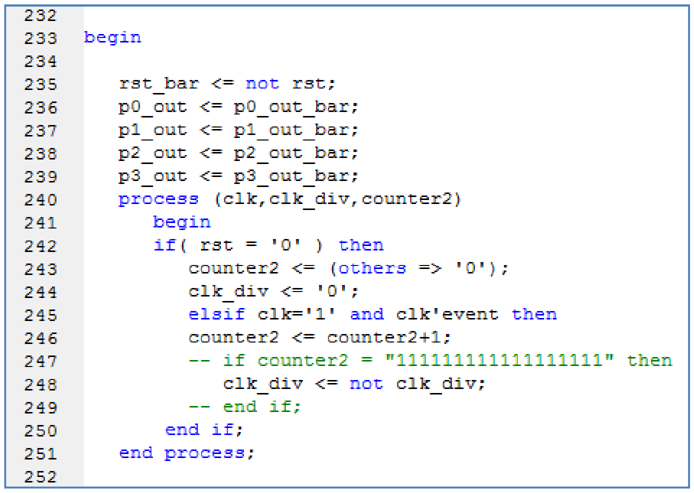
```
   -- Clock period definitions
   constant clk_period : time := 100 ns;

BEGIN

   p1_in <= p1_out;

   -- Instantiate the Unit Under Test (UUT)
   uut: i8051_top PORT MAP (
          clk => clk,
          rst => rst,
          ale => ale,
          psen => psen,
          ea => ea,
          p0_in => p0_in,
          p0_out => p0_out,
          p1_in => p1_in,
          p1_out => p1_out,
          p2_in => p2_in,
          p2_out => p2_out,
          p3_in => p3_in,
          p3_out => p3_out,
          pc_debug => pc_debug
        );

   -- Clock process definitions
   clk_process :process
   begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
      wait for clk_period/2;
   end process;
```

19. The above example makes use of P1 as being both an input and output. In the actual microprocessor hardware, P0, P1, P2 and P3 can act as input ports and output ports, as shown in the microprocessor hardware datasheet 00A.

    However, in the VHDL code given, P1 is defined separately: p1_in as input and p1_out as output. Hence, in the test bench, an additional line of code can be added:

    p1_in <= p1_out;

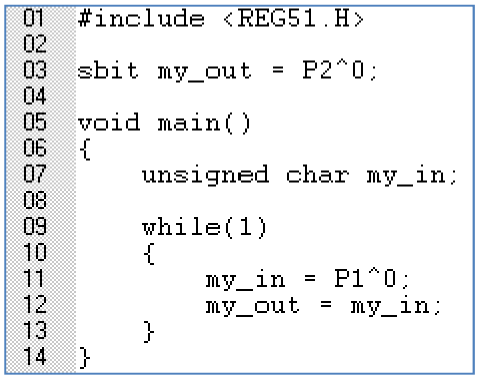
    Modifying the test bench does not affect FPGA programming. The test bench is for simulation only. A pushbutton cannot be used as an output, just as a LED cannot be used as an input.



```
begin

   rst_bar <= not rst;
   p0_out <= p0_out_bar;
   p1_out <= p1_out_bar;
   p2_out <= p2_out_bar;
   p3_out <= p3_out_bar;
   process (clk,clk_div,counter2)
      begin
      if( rst = '0' ) then
         counter2 <= (others => '0');
         clk_div <= '0';
         elsif clk='1' and clk'event then
         counter2 <= counter2+1;
         -- if counter2 = "111111111111111111" then
            clk_div <= not clk_div;
         -- end if;
      end if;
   end process;

```

20. Additionally, these changes are also made for the results to be visible in simulation:

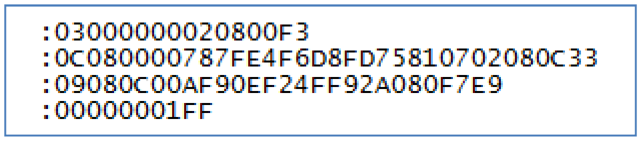
    p0_out <= p0_out_bar;
    p1_out <= p1_out_bar;
    p2_out <= p2_out_bar;
    p3_out <= p3_out_bar;



```c
#include <REG51.H>

sbit my_out = P2^0;

void main()
{
    unsigned char my_in;

    while(1)
    {
        my_in = P1^0;
        my_out = my_in;
    }
}
```

21. These codes show how to make an output follow an input. As an example, it could represent pressing a pushbutton and to have a LED light up.

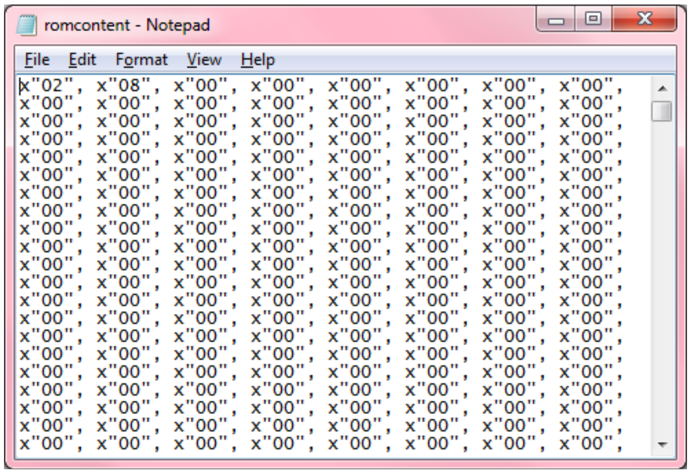
    For this code, the input will be pin 0 of port 1 and the output will be pin 0 of port 2.



```
:03000000020800F3
:0C080000787FE4F6D8FD75810702080C33
:09080C00AF90EF24FF92A080F7E9
:00000001FF
```

22. The hex code is generated. Note that the order of hex code is different from step 16 because the option to include the startup code was chosen in step 6. This does not matter for our case.

    The hex code is further explained in ***Intel Hex Description*** and ***ROM Contents***.

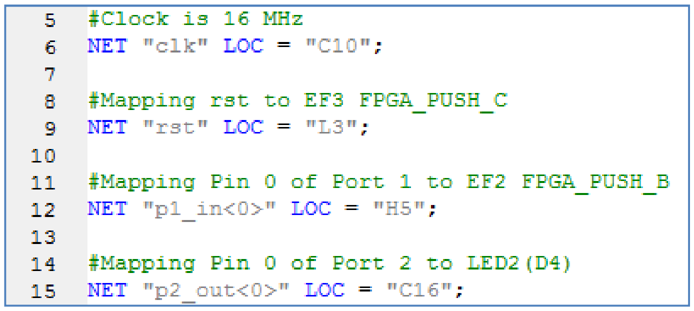


romcontent - Notepad

```
x"02", x"08", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
```

23. Generate the rom content file by following the instructions in ***hex2rom*** (readme.txt)

    The password for the zip file is: **3207**



```
#Clock is 16 MHz
NET "clk" LOC = "C10";

#Mapping rst to EF3 FPGA_PUSH_C
NET "rst" LOC = "L3";

#Mapping Pin 0 of Port 1 to EF2 FPGA_PUSH_B
NET "p1_in<0>" LOC = "H5";

#Mapping Pin 0 of Port 2 to LED2(D4)
NET "p2_out<0>" LOC = "C16";
```

24. After copying the hex code exactly as they are to the int_rom.vhd, create a constraint file with the following:

    NET "clk" LOC = "C10"; #Clock is 16 MHz
    NET "rst" LOC = "L3"; #Mapping rst to EF3 FPGA_PUSH_C
    NET "p1_in<0>" LOC = "H5"; #Mapping Pin 0 of Port 1 to EF2 FPGA_PUSH_B
    NET "p2_out<0>" LOC = "C16"; #Mapping Pin 0 of Port 2 to LED2(D4)

    You can also use square brackets "**[ ]**" instead of "**< >**"

25. There are several ways in which your FPGA can behave now, depending on how the i8051_top is programmed.

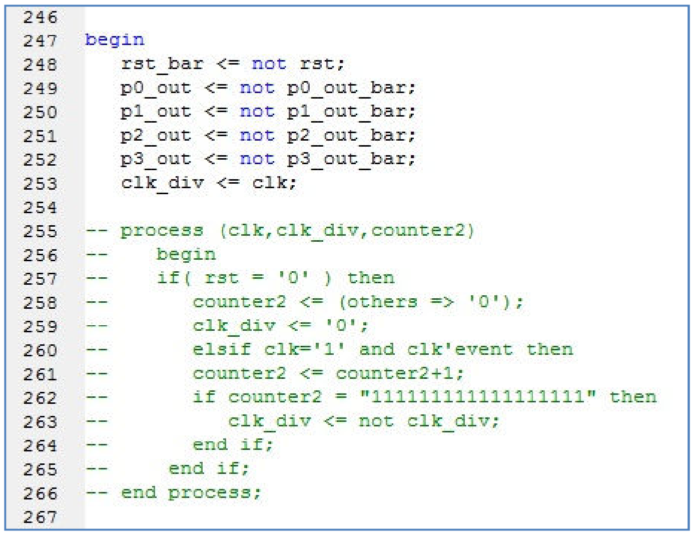
    For this case, the clock "clk_div" will run at a speed of 16 MHz (62.5 ns).

    To activate the microprocessor, push button C, as defined in step 24, is pressed and held. When the microprocessor is deactivated, LED2 will be off. When the microprocessor is activated, LED 2 will be on. This is because of the way the C program was written.

    When push button B, as defined in step 24, is pressed and held, LED2 will turn off.

    This is similar to an active low circuit.

    (The bit file for this step is ***Bit - 01***)



```

begin
   rst_bar <= not rst;
   p0_out <= not p0_out_bar;
   p1_out <= not p1_out_bar;
   p2_out <= not p2_out_bar;
   p3_out <= not p3_out_bar;
   clk_div <= clk;

-- process (clk,clk_div,counter2)
--    begin
--    if( rst = '0' ) then
--       counter2 <= (others => '0');
--       clk_div <= '0';
--       elsif clk='1' and clk'event then
--       counter2 <= counter2+1;
--       if counter2 = "111111111111111111" then
--          clk_div <= not clk_div;
--       end if;
--     end if;
-- end process;

```

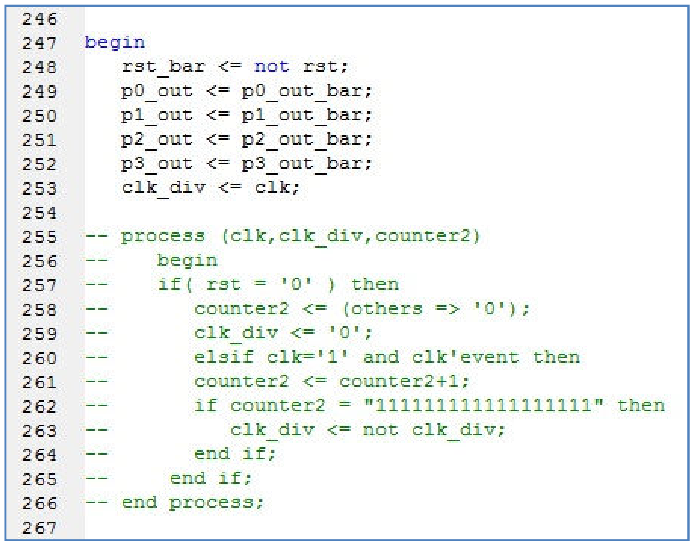
26. For this case, the clock "clk_div" will run at a speed of 16 MHz (62.5 ns).

    To activate the microprocessor, push button C, as defined in step 24, is pressed and held. When the microprocessor is deactivated, LED2 will be on. When the microprocessor is activated, LED 2 will be off. This is because of the way the C program was written.

    When push button B, as defined in step 24, is pressed and held, LED2 will turn on.

    This is similar to an active high circuit and is more intuitive for us.

    (The bit file for this step is ***Bit - 02***)



```

begin
   rst_bar <= not rst;
   p0_out <= p0_out_bar;
   p1_out <= p1_out_bar;
   p2_out <= p2_out_bar;
   p3_out <= p3_out_bar;
   clk_div <= clk;

-- process (clk,clk_div,counter2)
--    begin
--    if( rst = '0' ) then
--       counter2 <= (others => '0');
--       clk_div <= '0';
--       elsif clk='1' and clk'event then
--       counter2 <= counter2+1;
--       if counter2 = "111111111111111111" then
--          clk_div <= not clk_div;
--       end if;
--     end if;
-- end process;

```

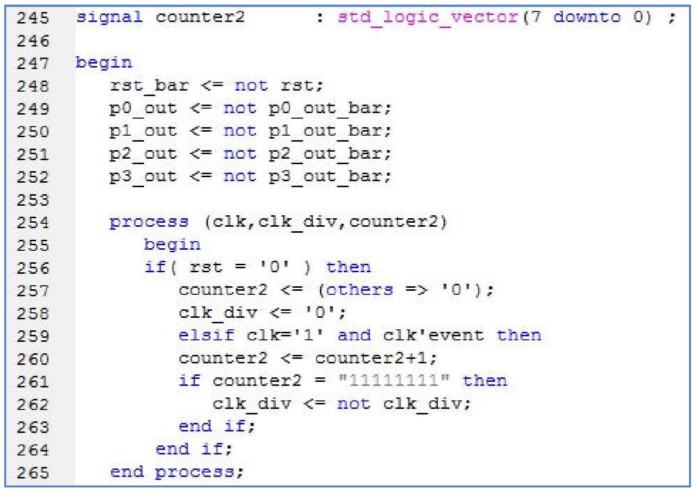
27. For this case, the clock "clk_div" will run at a speed of 2 * 62.5 Khz (2 * 16 us).

    FPGA clock is 16 MHz or (1 / 16x10^6) seconds
    The counter2 is 8 bits or 256 possible values

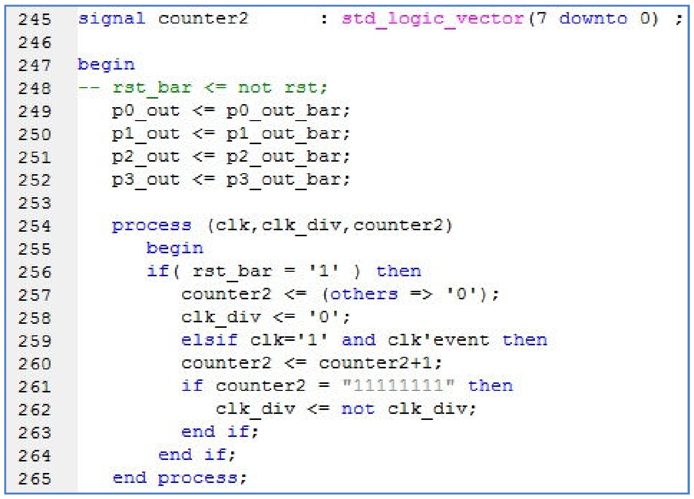
    The clk_div <= clk_div code create an increasing edge after (1 / 16x10^6) * (2^8) = 16 us, and a decreasing edge after another 16 us. Hence, one whole cycle is 32 us.

    This is similar to step 25, except that push button B needs to be held for some seconds before visible changes on the LED occur. Push button C is not affected because it does not make use of clk_div.

    (The bit file for this step is ***Bit - 03***)



```
signal counter2      : std_logic_vector(7 downto 0) ;

begin
   rst_bar <= not rst;
   p0_out <= not p0_out_bar;
   p1_out <= not p1_out_bar;
   p2_out <= not p2_out_bar;
   p3_out <= not p3_out_bar;

   process (clk,clk_div,counter2)
      begin
      if( rst = '0' ) then
         counter2 <= (others => '0');
         clk_div <= '0';
         elsif clk='1' and clk'event then
         counter2 <= counter2+1;
         if counter2 = "11111111" then
            clk_div <= not clk_div;
         end if;
       end if;
   end process;
```



```
signal counter2      : std_logic_vector(7 downto 0) ;

begin
-- rst_bar <= not rst;
   p0_out <= p0_out_bar;
   p1_out <= p1_out_bar;
   p2_out <= p2_out_bar;
   p3_out <= p3_out_bar;

   process (clk,clk_div,counter2)
      begin
      if( rst_bar = '1' ) then
         counter2 <= (others => '0');
         clk_div <= '0';
         elsif clk='1' and clk'event then
         counter2 <= counter2+1;
         if counter2 = "11111111" then
            clk_div <= not clk_div;
         end if;
        end if;
   end process;
```

28. For this case, the clock "clk_div" will run at a speed of 2 * 62.5 Khz (2 * 16 us).

    FPGA clock is 16 MHz or (1 / 16x10^6) seconds
    The counter2 is 8 bits or 256 possible values

    The clk_div <= clk_div code create an increasing edge after (1 / 16x10^6) * (2^8) = 16 us, and a decreasing edge after another 16 us. Hence, one whole cycle is 32 us.

    This is similar to step 26, except that push button B needs to be held for some seconds before visible changes on the LED occur. Push button C is not affected because it does not make use of clk_div.

    (The bit file for this step is ***Bit - 04***)

Create your "simple" program: one which uses "as much instructions as possible". Additional hardware can be used to connect to the FPGA for increased user friendliness, understanding and friendliness.

You can also refer to manual ***C51 Cross Compiler, The C51 Primer***, ***uVision 2*** and website links such as:
http://www.keil.com/support/man_c51.htm
These might help you in case you need to know what C instructions converts to what mnemonics.

The P0, P1, P2 and P3 can be mapped to the FPGA pins, touchpads or LEDs, as described in manual ***Xilinx Spartan-3A Evaluation Kit User Guide***, and using instructions from manual ***CG3207 VHDL Refresher***.

Take note of the SFR Memory Map and bit addressable when addressing the different ports. Also, the I/O pins for the GPIO Header are CMOS that use 3.3 V.

You are encouraged to understand ***Intel Hex Description*** and ***ROM Contents*** to make your debugging easier. Analysis of errors and results can be done in simulation, as creating the bit file for the FPGA each time is time consuming and does not provide details for the individual registers.

In case the above example does not work, you can test just the main hex code **AF90EF24FF92A080F7** and analyse your PC, IR, states and other registers you need. If they work as intended, then you should check the other instructions, such as DJNZ, LMJP and so on.