Xilinx Virtex-II Pro를 이용한 SoC설계(Ⅳ)

Simulation Microblaze system

지난 호에서는 user define peripheral을, 프로그램을 통해 제어하는 방법에 대해서 설명하였다. 지난 호에 이어, 이번 호에서는 modelsim을 통해 simulation하는 방법에 대해서 설명하고 본 연재를 마친다.

글: 김 혁/Insight Korea xilinx FAE 과장
hyukkim@memec-korea.com

Define simulation model

XPS에서는 Modemsim behavioral simulation을 할 수 있다. Behavioral simulation은 synthesis 과정이나 implementation 과정이 필요없기 때문에 설계 에러나 기능을 확인하는데 효율적이다. XPS에서 지원하는 Simulation 툴로는 modelsim과 verilog-xl이 있는데, 여기서는 modelsim을 사용하도록 한다.

XPS 〉 Option 〉 Project Options 〉 HDL and Simulator에서 Modelsim을 선택한다. 그리고 Simulation Model은 "Behavioral"로 선택한다(그림 1).

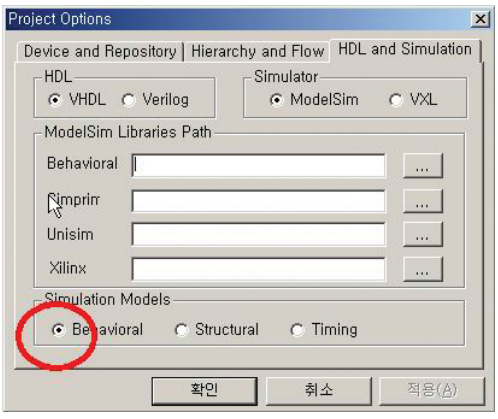


그림 1. setting simulation model type

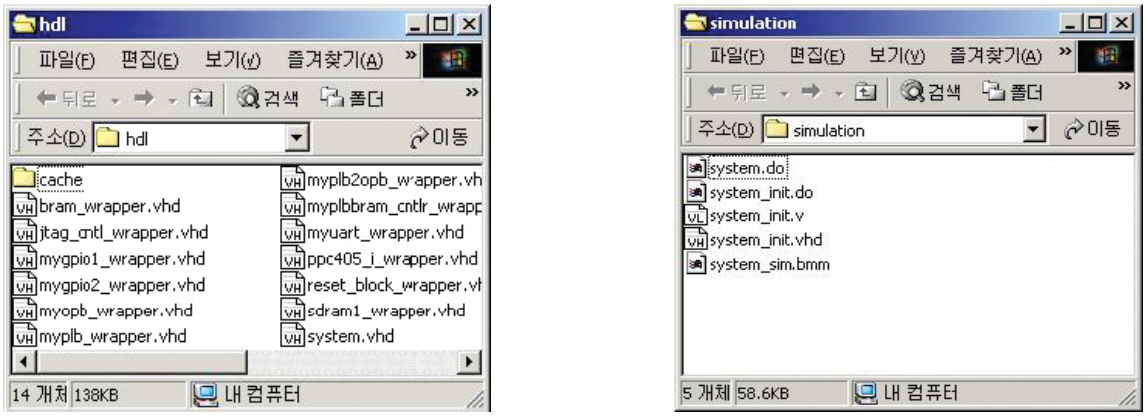
1. File list for simulation

modelsim용 library가 모두 compile되어 있는 경우, behavioral simulation을 하기 위해서는 system.mhs에서 정의한 모든 hardware에 대한 wrapper_file이 만들어져야 한다. 이 file은 XPS 〉 Tools 〉 Sim Model Generation을 선택하면 "simulation"과 "hdl" folder가 만들 수 있다(그림 2(a), 2(b)).

simulation folder에 있는 system_init.v 또는 system_init.vhd은 Microblaze용 code와 data가 bram에 어떤 형식으로 write 되는지 보여준다. System_sim.bmm은 몇 개의 bram이 사용되고 디코딩되는 address 범위를 보여준다(그림 2(b)).

2. Testbench file

그림 3을 보면 user가 만들어야 하는 파일로는 system.mhs, system.c와 testbench임을 알 수 있다. Testbench를 만들기 위해서는 Microblaze로 만든 전체 시스템의 top file이 무엇인지 알아야 한다. 이 top file은 그림 2와 같이 hdl folder에 있는 system.vhd file이다. 이 파일에서 Microblaze의 top entity를 찾을 수 있다(리스트 1).



(a) "hdl" folder

(b) "simulation" folder

그림 2. Making Simulation model

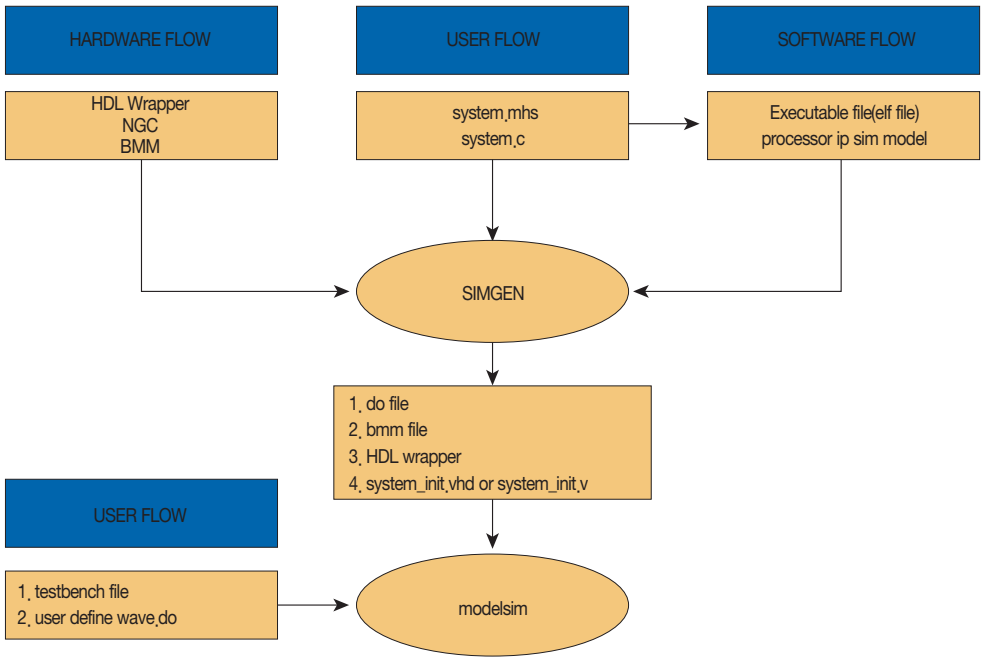


그림 3. modelsim flow

이 top entity를 가지고 testbench를 만들 때, 현재 bram에 있는 데이터를 configuration 시켜야 하기 때문에 testbench에는 **리스트 2**에 있는 configuration을 testbench에 넣어준다.

리스트 3은 완성된 testbench file이다.

3. BRAM initialize

그림 2(b)를 보면 simulation folder에 system_sim.bmm이 있는 것을 알 수 있다. bmm 파일은 Microblaze가 fetch하는 instruction이나 data가 저장되는 Block RAM의 위치를 정의한 파일이다.

```
entity system is
port (
      -- instance global
      sys_clk : in std_logic;
      mydprambport_data : inout std_logic_vector(0 to 7);
      rx : in std_logic;
      mydprambport_web : in std_logic;
      system_reset : in std_logic;
      tx : out std_logic;
      mydprambport_clkb : in std_logic;
      mydprambport_addrb : in std_logic_vector(0 to 9);
      mydprambport_enb : in std_logic;
      rts : out std_logic);
end system;
```

리스트 1. Top entity of system

```
configuration testbench_conf of testbench is
      for behavior
          for uut : system
                for IMP
                      for all : bram_lmb_wrapper use configuration
                      work.bram_lmb_conf;
                      end for;
                end for;
          end for;
      end for;
end testbench_conf;
```

리스트 2. Configuration for bram

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity testbench is
end testbench;

architecture behavior of testbench is

    component system
    port(
        sys_clk                 : in std_logic;
        system_reset            : in std_logic;
        rx                      : in std_logic;
        tx                      : out std_logic;
        rts                     : out std_logic;
        mydprambport_data       : inout std_logic_vector(0 to 7);
        mydprambport_web        : in std_logic;
        mydprambport_clkb       : in std_logic;
        mydprambport_addrb      : in std_logic_vector(0 to 9);
        mydprambport_enb        : in std_logic
    );
    end component;

    signal sys_clk              : std_logic := '0';
    signal system_reset         : std_logic := '0';
    signal rx                   : std_logic := '0';
    signal tx                   : std_logic;
    signal rts                  : std_logic;
    signal mydprambport_data    : std_logic_vector(0 to 7);
    signal mydprambport_web     : std_logic := '1';
    signal mydprambport_clkb    : std_logic := '1';
    signal mydprambport_addrb   : std_logic_vector(0 to 9) := "0000000000";
    signal mydprambport_enb     : std_logic := '1';

begin

    uut: system port map(
        sys_clk => sys_clk,
        system_reset => system_reset,
        rx => rx,
        tx => tx,
        rts => rts,
        mydprambport_data => mydprambport_data,
        mydprambport_web => mydprambport_web,
        mydprambport_clkb => mydprambport_clkb,
        mydprambport_addrb => mydprambport_addrb,
        mydprambport_enb => mydprambport_enb
    );
    rx <= '0';
    sys_clk <= not sys_clk after 10 ns;

    system_reset <= '1' after 500 ns;
end;

configuration testbench_conf of testbench is
    for behavior
        for uut : system
              for imp
                    for all : bram_lmb_wrapper use configuration
                    work.bram_lmb_conf;
                    end for;
              end for;
        end for;
    end for;
end testbench_conf;
```

리스트 3. testbench for simulaiton

리스트 4에는 이 bmm 파일과 ELF 파일을 가지고 만든 system_init.vhd를 보여준다.

이 파일에서는 모두 4개의 BRAM이 초기화 되고 있는데, 각 RAM의 instance는 ram16_s9_s9_0, ram16_s9_s9_1, ram16_s9_s9_2, ram16_s9_s9_3으로 정의되어 있고, 이 이름은 **리스트 5**와 같이 system_sim.bmm에 정의되어 있다.

이 프로젝트에서 사용되는 microblaze는 reset vector 값이 0이고, code와 데이터는 0x0000_0000에서 0x0000_1fff까지 access하도록 하드웨어를 정했다. 따라서 system.mhs에서 lmb_lmb_bram_if_cntlr는 C_BASEADDR과 C_HIGHADDR은 각각 0x0000_0000, 0x0000_1fff로 정의했고 그 범위는 모두 8k이다.

MicroblazeBlaze Byte Addressable이므로 각 address는 8bit 단위로 최대 32bit까지 한꺼번에 지정할 수 있다. 따라서 8k*8bit=64kbit, Virtex-II Pro는 Bram 한 개의 크기가 16k bit이므로 모두 4 BRAM 블록이 필요하다.

따라서 리스트 5와 같이 bmm 파일도 모두 4개의 bram을 정의한 것을 확인할 수 있다.

4. Simulation do file

Modelsim에서 behavioral simulation하기 위해서는 hdl folder에 있는 각종 파일들을 compile 시켜야 한다. 그림 2(b)를 보면 simulation folder에 system.do file과 system_init.do file이 만들어진 것을 볼 수 있다. System_init.do file에는 hdl folder에 있는 각종 wrapper file과 user가 설계한 file을 compile 시키는 명령어가 포함되어 있다(**리스트 6**).

System.do file에는 system_init.do를 실행하고 behavioral mode로 simulation 하도록 modelsim을 실행시킨다(**리스트 7**).

5. Add your testbench file and wave form to system_init.do file

System.do file에는 system.mhs에 정의된 peripheral의 wrapper file들만 compile 한다. 따라서 testbench까지 추가로 compile 할 수 있도록 system_init.do file을 수

```
library unisim;

configuration bram_lmb_conf of bram_lmb_wrapper is
  for IMP
    for bram_block_0_i : bram_lmb_elaborate
      for IMP
        for ramb16_s9_s9_0 : ramb16_s9_s9
          use entity unisim.ramb16_s9_s9(ramb16_s9_s9_v)
          generic map(
            INIT_00 => X"20...B8",
            INIT_3F => X"00...00");
        end for;

        for ramb16_s9_s9_1 : ramb16_s9_s9
          use entity unisim.ramb16_s9_s9(ramb16_s9_s9_v)
          generic map(
          INIT_00 => X"C0...00",
        INIT_3F => X"00...00");
        end for;

        for ramb16_s9_s9_2 : ramb16_s9_s9
          use entity unisim.ramb16_s9_s9(ramb16_s9_s9_v)
          generic map(
            INIT_00 => X"03...00",
            INIT_3F => X"00...00");
        end for;

        for ramb16_s9_s9_3 : ramb16_s9_s9
          use entity unisim.ramb16_s9_s9(ramb16_s9_s9_v)
          generic map(
            INIT_00 => X"78...18",
          INIT_3F => X"00....00");
          end for;

      end for;
    end for;
  end for;
end bram_lmb_conf;

configuration system_conf of system is
  for IMP
    for all : bram_lmb_wrapper use configuration work.bram_lmb_conf;
    end for;
  end for;
end system_conf;
```

리스트 4. system_init.vhd

```
ADDRESS_BLOCK bram_lmb RAMB16 [0x00000000:0x00001FFF]
     BUS_BLOCK
     bram_lmb/bram_block_0_i/ramb16_s9_s9_0 [31:24] ;
     bram_lmb/bram_block_0_i/ramb16_s9_s9_1 [23:16] ;
     bram_lmb/bram_block_0_i/ramb16_s9_s9_2 [15:8] ;
     bram_lmb/bram_block_0_i/ramb16_s9_s9_3 [7:0] ;
     END_BUS_BLOCK;
END_ADDRESS_BLOCK;
```

리스트 5. system_sim.bmm

```
vlib work
vmap work work

vcom -93 -work work
C:/EDK_LAB/demo/edk_myDpramBport_uart/myip/bram_block_v1_00
_a/hdl/vhdl/bram_lmb_elaborate.vhd

vlib opb_mydpramBPort_v1_00_b
vmap opb_mydpramBPort_v1_00_b opb_mydpramBPort_v1_00_b

vcom -93 -work opb_mydpramBPort_v1_00_b
C:/EDK_LAB/demo/edk_myDpramBport_uart/myip/opb_mydpramBPo
rt_v1_00_b/hdl/vhdl/opb_mydpramBPort.vhd
vcom -93 -work work ../hdl/mylmblmb_cntlr_wrapper.vhd
vcom -93 -work work ../hdl/bram_lmb_wrapper.vhd
vcom -93 -work work ../hdl/d_lmb_wrapper.vhd
vcom -93 -work work ../hdl/i_lmb_wrapper.vhd
vcom -93 -work work ../hdl/myopb_wrapper.vhd
vcom -93 -work work ../hdl/mblaze_wrapper.vhd
vcom -93 -work work ../hdl/opb_mydprambport1_wrapper.vhd
vcom -93 -work work ../hdl/myuart_wrapper.vhd
vcom -93 -work work ../hdl/system.vhd
vcom -93 -work work ./system_init.vhd
```

리스트 6. system_init.do file

```
do system_init.do
vsim - c +notimingchecks system_conf
```

리스트 7. system.do file

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic -radix hexadecimal
/testbench/sys_clk
add wave -noupdate -format Logic -radix hexadecimal
/testbench/system_reset
add wave -noupdate -format Logic -radix hexadecimal
/testbench/rx
...
add wave -noupdate -format Literal -radix hexadecimal
/testbench/uut/mblaze/microblaze_0_i/prefetch_addr
add wave -noupdate -format Literal -radix hexadecimal
/testbench/uut/mblaze/microblaze_0_i/reg_addr
add wave -noupdate -format Logic -radix hexadecimal
/testbench/uut/mblaze/microblaze_0_i/reg_write
add wave -noupdate -format Logic -radix hexadecimal
/testbench/uut/mblaze/microblaze_0_i/valid_instr
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {5730182 ps} 0}
WaveRestoreZoom {5510130 ps} {6509871 ps}
configure wave -namecolwidth 435
configure wave -valuecolwidth 73
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
```

리스트 8. Wave.do file

정한다.

System_init.vhd는 simulation folder에 있는 것을 사용해야 한다. System_init.vhd를 컴파일하는 부분을 아래와 같이 수정한다.

```
vcom -93 -work work ../simulation/system_init.vhd
```

나머지 testbench를 컴파일 할 수 있도록 추가한다. 이때 주의할 것은 simulation folder에 testbench.vhd를 만들면 나중에 simulation file을 update 할 때, 이 파일이 없어지게 된다. 그래서 testbench는 simulation folder가 아닌 sim이라는 folder에 저장을 했다.

```
vcom -93 -work work ../sim/testbench.vhd
```

system_init.do 파일도 testbench가 추가되고 system_init.vhd의 위치를 simulation folder로 수정했기 때문에 매번 simulation file을 다시 만들 때마다 계속 수정해줘야 한다.

이것을 방지하기 위해 system_init.do file도 아예 sim folder에 저장을 하고, simulation file을 다시 만들 때마다 sim folder로부터 복사해 simulation folder에 붙여놓

```
vlib work
vmap work work
vcom -93 -work work
C:/EDK_LAB/mblaze/edk_myDpramBport_uart/myip/bram_block_v1_0
0_a/hdl/vhdl/bram_lmb_elaborate.vhd
vlib opb_mydpramBPort_v1_00_b
vmap opb_mydpramBPort_v1_00_b opb_mydpramBPort_v1_00_b
vcom -93 -work opb_mydpramBPort_v1_00_b
C:/EDK_LAB/mblaze/edk_myDpramBport_uart/myip/opb_mydpramB
Port_v1_00_b/hdl/vhdl/opb_mydpramBPort.vhd
vcom -93 -work work ../hdl/mylmblmb_cntlr_wrapper.vhd
vcom -93 -work work ../hdl/bram_lmb_wrapper.vhd
vcom -93 -work work ../hdl/d_lmb_wrapper.vhd
vcom -93 -work work ../hdl/i_lmb_wrapper.vhd
vcom -93 -work work ../hdl/myopb_wrapper.vhd
vcom -93 -work work ../hdl/mblaze_wrapper.vhd
vcom -93 -work work ../hdl/opb_mydprambport1_wrapper.vhd
vcom -93 -work work ../hdl/myuart_wrapper.vhd
vcom -93 -work work ../hdl/system.vhd
vcom -93 -work work ../simulation/system_init.vhd
vcom -93 -work work ../sim/testbench.vhd
vsim -Lf unisim -t ps +notimingchecks work.testbench_conf
view wave
do wave.do
run 16 us
```

리스트 9. edit your system_init.do file

고 사용하는 것이 편리하다. 그리고 system.do file에 있는 simulation을 running 시키는 것까지 미리 포함시킨다.

```
vsim -Lf unisim -t ps +notimingchecks work.testbench_conf
```

microblaze의 각 신호를 살펴보고자 할 때 일정한 규칙을 가지고 보는 것이 중요하다. wave.do file은 그런 역할을 하는 file이다. **리스트 8**은 wave.do file 리스트이다.

Modelsim에서 wave form을 실행시킨다.

```
view wave
```

앞에서 만든 wave.do를 실행시킨다.

그림 4. files on sim folder

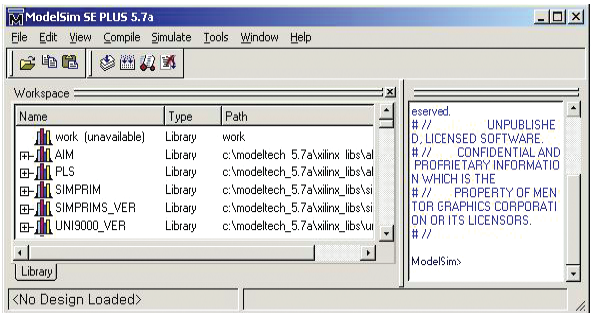


그림 5. Running modelsim

```
do wave.do
```

모두 16usec 동안 running 한다.

```
run 16 us
```

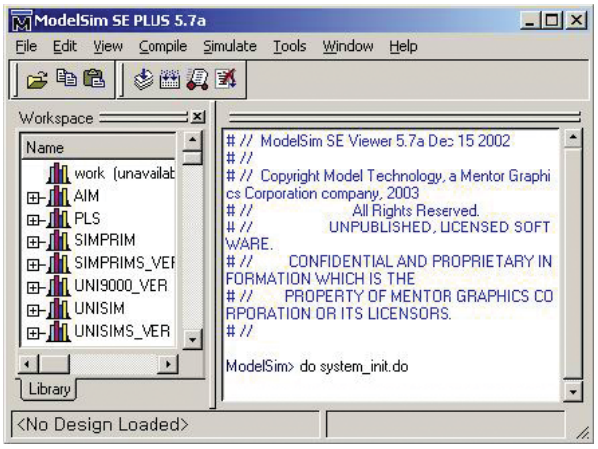
리스트 9는 수정된 system_init.do file이다.

6. Files on sim folder

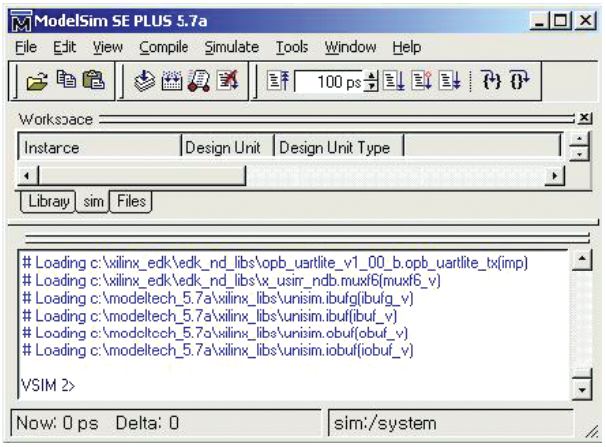
Simulation file을 만들 때마다 새로 update 해야 하는 파일들은 sim folder라는 곳에 저장해서 반복해 사용하는 것이 효율적이다(**그림 4**).

7. Running modelsim

XPS 〉 tools 〉 Hardware Simulation을 선택한다.



(a) system.do file의 실행



(b) vsim0| 실행된 화면

그림 6. runngin system.do file

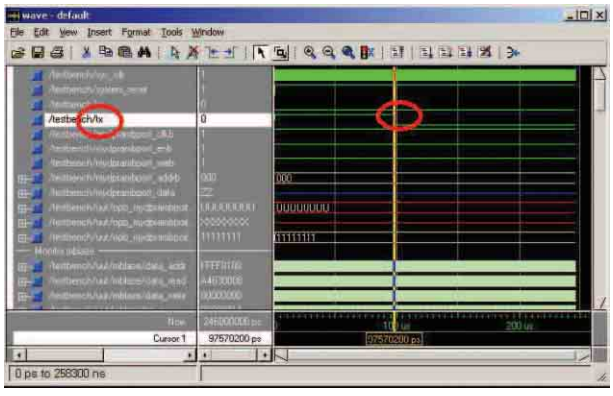


그림 7. Modelsim simulation with uart function

Sim folder에 있는 system_init.do file과 wave.do file을 simulation folder로 복사해 준다(**그림 5**).

8. Running system_init.do file

그림 6과 같이 system.do file을 실행시키면(a) 각종 wrapper file을 compile시키는 메시지가 출력된 후 vsim이 실행된 것을 볼 수 있다(b).

9. Modelsim simulation output

그림 7은 Modelsim으로 simulation한 결과이다. 약 10usec 부분에서 tx가 변하는 것을 볼 수 있다. 이렇게 tx가 변하는 것은 system.c에서 main()에 있는 아래 코드를 실행하고 있기 때문이다.

```
printf_uart("\n\r\n\r 1) Test SDRAM");
```

system.mhs file에서는 uart의 속도가 19200 bps로 정의되어 있는데, 이 경우 uart_clk는 52usec마다 high와 low를 반복한다.

따라서 uart_clk는 104usec의 주기를 가진다. 더구나 각 문자는 8비트로 이루어졌고 stop bit까지 있다면 9비트가 1개 문자를 만들어주기 때문에 ascii code 한 문자를 uart_tx로 보내기 위해서는 9*52=468usec의 시간이 소요된다.

첫 번째 문자열인 "\n\r\n\r 1) Test SDRAM"은 18개 문자로 이루어졌기 때문에 468*18=8424usec의 시간이 소요된다. 이 정도 시뮬레이션 하려면 아마 5, 6시간이 소요되므로 printf_uart()를 실행하지 않도록 프로그램을 다시 코딩해야 한다.

리스트 10과 같이 조건부 compile문을 넣어 simulation이 정의된 경우에는 printf_uart()가 그대로 return 되도록 source code를 수정한다.

XPS 〉 tools 〉 compile program sources를 선택해 프로그램을 컴파일 한다. XPS 〉 tools 〉 clean 〉 simulation을 선택해 기존 simulation 데이터를 모두 지운다. XPS 〉 tools 〉 sim model generation을 선택해 simulation data를 새로 만들어 주고 기존 sim folder에 있는 wave.do file과 system_init.do 파일을 simulation

```
#define SIMULATION

unsigned int max_values[4];
unsigned int duty_values[4];


unsigned int *sdram_location = (unsigned int *) 0xFFFE7000;
unsigned int *temp;
int loop_count, sdram_data_read;

void printf_uart(char *s)
{
#ifndef SIMULATION
    while (*s)
    {
        XUartLite_SendByte(XPAR_MYUART_BASEADDR,*s);
        while (XUartLite_mIsTransmitFull(XPAR_MYUART_BASEADDR)
== true) {
        };
        s++;
    }
#endif
    return;
}
```

리스트 10. add compile codition

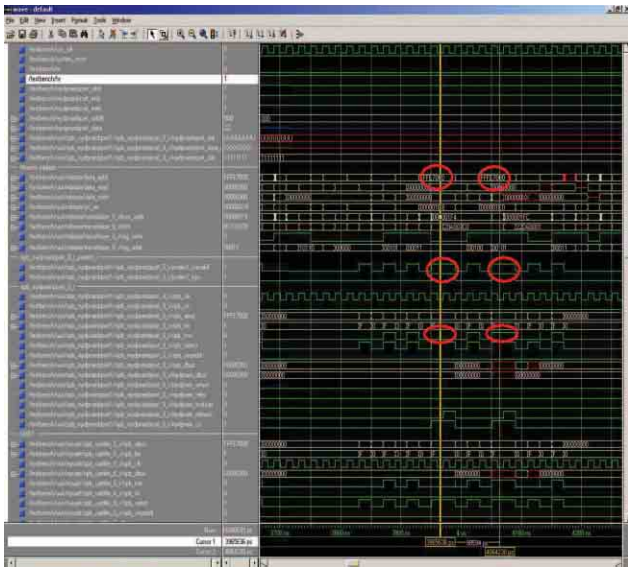


그림 8. simulation access myDpramBport without uart function

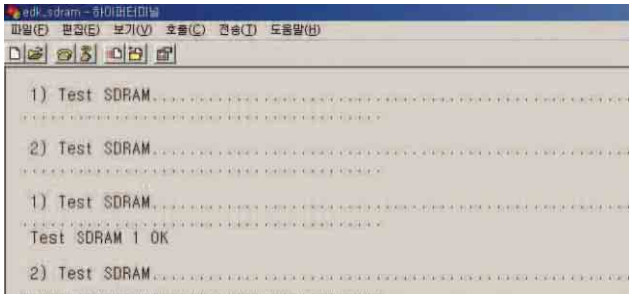


그림 9. myDpramBport test reulst

folder에 복사한다.

XPS 〉 tools 〉 Hardware simualtion을 선택해 modelsim을 실행시킨다. 그림 6과 같이 system_init.do file을 실행시킨다.

그림 8을 보면 myDpramBport address인 0xffff_7000이 OPB Address에 실리고 Availd와 ps 및 rnw 신호가 제대로 나오는 것을 확인할 수 있다.

10. Running on target board

printf_uart()를 실행하도록 프로그램을 다시 uart를 function이 실행될 수 있도록 source code를 아래와 같이 고쳐준다. 이렇게 하면 #ifndef SIMULATION ~ #endif 사이가 다시 compile이 된다.

#define SIMULATION_

XPS 〉 tools 〉 update Bitstream을 선택해 bit 파일에 update된 프로그램 데이터가 write 되도록 해준다.

XPS 〉 tools 〉 downloads를 선택하면 **그림 9**와 같이 hyperTerminal에 실제 running 결과가 나타난다.

이번 호까지 "xilinx SoC solution"이라는 제목으로 모두 4회에 걸쳐 연재를 하였다. 앞으로 자일링스 FPGA인 VIRTEX-II Pro의 응용분야가 어떻게 발전할 지 알 수 없지만, 원가절감과 개발기간 단축이라는 명제 앞에 VIRTEX-II Pro의 PPC405를 사용하는 분야는 점점 더 넓어지고 있다. RTime