Xilinx Virtex-II Pro를 이용한 SoC설계(III)

Debugging과 Microblaze

지난 시간에 이어 이번 시간에는 GDB를 이용해 user program을 debugging하는 방법을 설명한다. Xilinx는 PPC405와 같은 hardware IP CPU 뿐만 아니라 Microblaze라는 netlist CPU도 제공하고 있다. EDK(Embedded Development Kit)은 PPC405뿐만 아니라 Microblaze를 이용한 시스템도 같이 개발할 수 있다.

글: 김혁/Insight Korea Xilinx FAE 과장

hyukkim@memec-korea.com

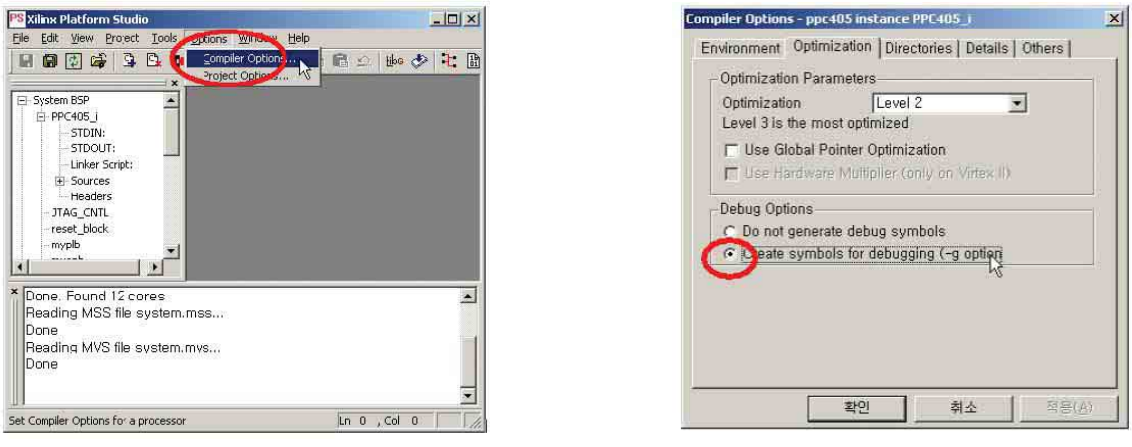
Debugging

Adding Debug Inforamtion

일반적으로 compiler 기본 option은 "Do not generate debug symbols"이다. 이 option을 바꾸기 위해서는 XPS 〉 Options 〉 Compiler Options 〉 Optimization에서 Create symbols for debugging(-g option)을 선택한다(그림 1).

Debugging 정보를 넣어서 컴파일 하도록 옵션을 조정했으므로 Tools〉 Compile Program source를 선택한다. 이때 컴파일되는 것은 하드웨어가 아닌 소프트웨어이므로 하드웨어 netlist를 다시 만들 필요는 없다.

이제 새롭게 elf file이 만들어졌기 때문에 bit 파일에 update시켜야 되므로 Tools 〉 Update Bitstream을 선택한다.



a b

그림 1. Add debugging inforamtion

Invoking XMD

디버깅을 하기 위해서는 XMD라는 프로그램을 실행시켜야 한다. 이 프로그램은 GDB라는 프로그램과 Target Board의 프로그램간의 프로토콜을 해석하는 역할을 한다.

Insight V2Pro Board의 전원을 넣고 XMD%에서 "ppcconnect" 명령을 내리면 현재 JTAG CHAIN으로 연결된 상태를 보여준다. 이때 ppcconnect는 모두 소문자로 입력해야 한다(**그림 2, 3** 참조).

Starting The GDB

Tools 〉 Software Debugger를 실행시키면 GDB 프로그램이 실행된다(**그림 4-a**). 다음으로 File〉Source를 선택한 후 ppc405_i/code/executable.elf 파일을 선택한다.(그림 4-b). 이미 obj에는 debugging 정보가 들어가 있기 때문에 GDB 프로그램은 C source코드를 보면서 deugging 할 수 있도록 해준다(그림 4-c).

만일 debugging정보가 없다면 assemble language형태로 sourc가 보이게 되며 이제 GDB와 XMD사이를 연결해 주기 위해 Run〉Run을 선택하면 debugging 실제 프로그램이 board이 download된다(그림 4-d).

GDB setting

Target Selection에서 Target, Hostname, Port를 **그림 5-a**와 같이 정해준다. GDB는 그림 5-b와 같이 source code와 함께 현재 실행되고 있는 모습을 보여준다. 좌측에 있는 빨간색 점은 현재 35행에서 break point가 설정된 것을 보여주며 38행에 있는 녹색 선은 다음에 실행될 포인트를 의미한다(그림 5-b).

그림 5-c는 Hyper Terminal을 통해 프로그램을 실행결과를 보여주고 있다. 프로그램을 디버깅하다 보면 새롭게 break point를 설정하고, 그 break point까지 프로그램을 강제로 실행시킬 때가 필요하다.

SDRAM TEST는 반복적으로 실행되는 횟수가 많이 있기 때문에 break point는 93행에 setting하고 Continue를 누르면 93행까지 바로 실행되는 것을 볼 수 있다(그림 5-d).

Microblaze

Microblaze는 Software Processor Core로써 netlist 형태로 제공되는 IP이며, Xilinx FPGA중 Virtex 시리즈(Virtex, Spartan-II, Virtex-E, Spartan-IIE, Virtex-II, Spartan-III, Virtex-II Pro)라면 어느 디바이스에도 targetting을 할 수 있다. 이때 Microblaze는 약 450개의 slice를 사용한다.

Microblaze는 Local Memory Bus(LMB)와 On-chip Peripheral Bus(OPB), 2 종류의 bus를 사용하고 있다. 또한 **그림 6**에 나타난 것처럼 Microblaze는 data bus와 instruction bus가 각각 존재하는 Harvard architecture를 사용하고 있다. 따라서 OPB도 Instruction Bus와 Data Bus가 분리되어 있으며, LMB도 Instruction Bus와 Data Bus가 분리되어 있다.

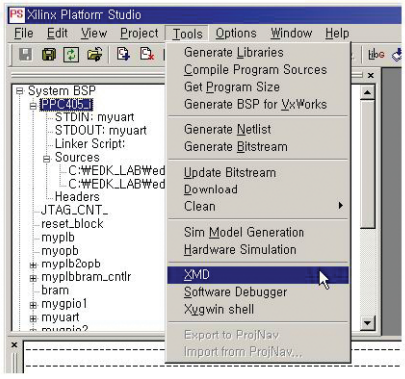


그림 2. XMD Window

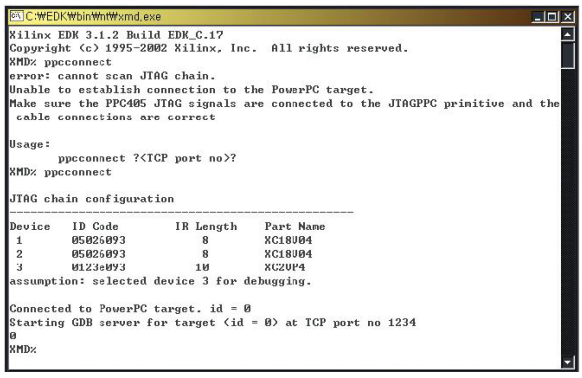
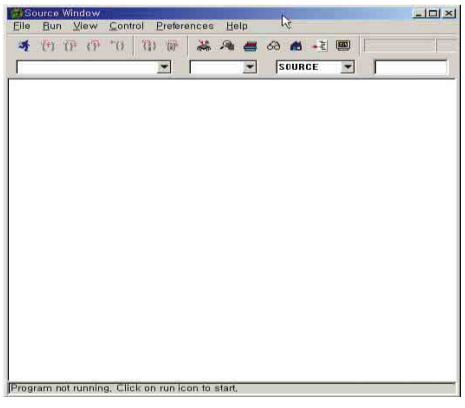
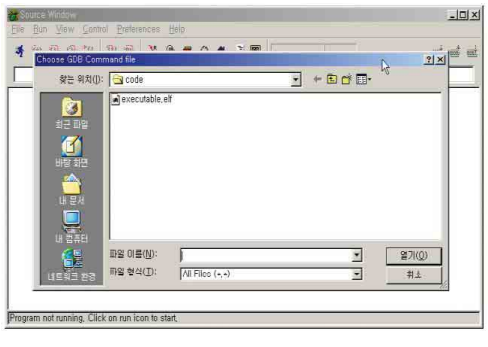


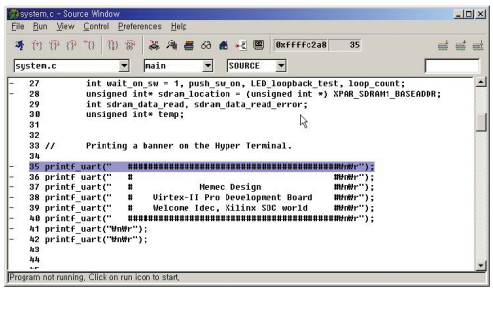
그림 3. XMD Window and PPCConnect



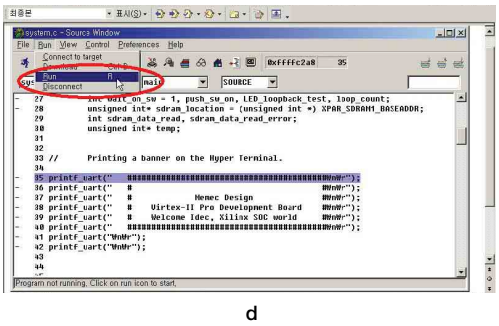
a



b



c



d

그림 4. Running GDB

Design Simple Microblaze system

Microblaze connection mode

Microblaze는 LMB와 OPB를 동시에 사용할 수 있으며 Harvard Architecture를 사용하기 때문에 LMB와 OPB에서 모두 instruction과 data를 fetch할 수 있다. 사용자 입장에서는 어느 버스에서 instruction을 가져오고 어느 버스에서 data를 가지고 오는지 결정을 해야한다. **그림 7**은 Microblaze에서 사용할 수 있는 버스의 조합을 보여준다.

프로그램의 크기가 FPGA 내부 메모리에 넣기 어려울 정도로 큰 경우, 이 프로그램은 OPB를 통해서 fetch해야 한다. 이 경우에 FPGA 내부 메모리에는 전체 프로그램 중에 자주 호출되는 interrupt service routine을 넣는 것이 좋고 나머지는 외부 external memory에 둬, microblaze가 OPB를 통해 코드를 fetch하도록 한다.

하지만 코드와 데이터의 크기가 내부 block memory에 들어갈 만큼 적은 경우에는 그림 7-c와 같은 Bus Connection을 많이 사용한다.

FPGA 내부 Bram에 넣을 수 있는 크기는 클수록 좋겠지만, Block Ram을 많이 연결하면 Ram을 access하기 위해 Mux를 많이 사용하게 된다. 이렇게 되면 access 속도가 빠른 LMB의 특성을 제대로 살릴 수 없게 된다.

따라서 최대 access 속도를 보장하기 위해 Block Ram의 크기는 FPGA family에 따라 한정되어 있고, 그 크기는 Virtex-II, Virtex-II Pro의 경우 64Kbytes, 나머지 Virtex, Spartan-II, Virtex-E, Spartan-IIE은 16Kbytes이다.

Define memory map of microblaze system

이 시스템은 Microblaze Software CPU, UART, myDpramBport로 구성되어 있으며 메모리 맵은 **그림 8**과 같다.

Making Uart

OPB에 연결되는 UART는 앞에 PPC405에서 설계한 UART를 그대로 사용할 수 있습니다. UART를 설계할 때 필요한 mhs 파일은 지난 호에 연재된 것을 참조하면 된다. 다만 Microblaze는 reset 블록이 따로 없기 때문에

OPB_Rst는 XPS에서 자동으로 연결할 수 있도록 comment 처리해 준다.

```
PORT rx = rx, DIR = INPUT
PORT tx = tx, DIR = OUTPUT

BEGIN opb_uartlite
  PARAMETER INSTANCE = myuart
  PARAMETER HW_VER = 1.00.b
  PARAMETER C_DATA_BITS = 8
  PARAMETER C_CLK_FREQ = 100000000
  PARAMETER C_BAUDRATE = 19200
  PARAMETER C_USE_PARITY = 0
  PARAMETER C_BASEADDR = 0xf0000100
  PARAMETER C_HIGHADDR = 0xf00001ff
# PORT OPB_Rst = peripheral_rst
  PORT RX = rx
  PORT TX = tx
  BUS_INTERFACE SOPB = myopb
END
```

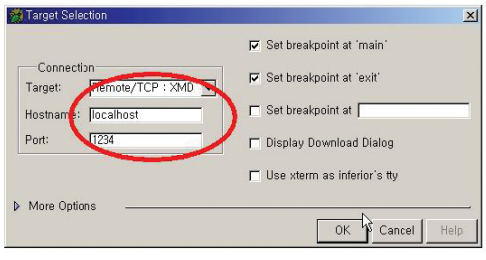
Making myDpramBport

myDpramBport는 Microblaze system에서 OPB를 통해 access되는 4Kbytes의 메모리 공간이다. Xilinx FPGA 내부에 있는 BRam은 모두 True Dual Port Memory이기 때문에 **그림 9**와 같이 A Port는 OPB에 연결해 Microblaze가 사용하도록 하고, B Port는 external cpu와 같은 외부 디바이스에서 사용할 수 있도록 FPGA의 pin으로 나오게 한다.

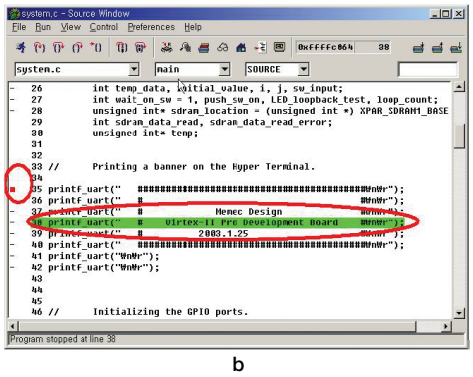
이렇게 하면 FPGA내부의 Microblaze와 외부 디바이스가 myDpromBport를 통해 message를 주고받을 수 있게 된다.

일단 4Kbyte ram block을 만들기 위해 coregen을 통해 mydpram8x4096짜리 memory block을 만든다. 그러면 coregen은 mydpram8x4096이라는 netlist를 만들어 주는데 이 netlist에 OPB와 관련된 신호들을 연결해 주면 myDpramBport라는 peripheral을 만들 수 있다.

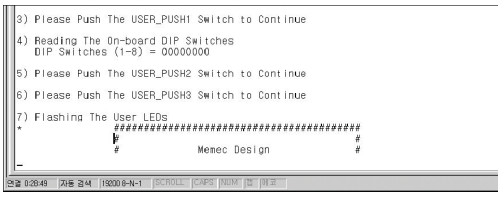
MyDpramBport peripheral을 완성하고 나면 **그림 10**



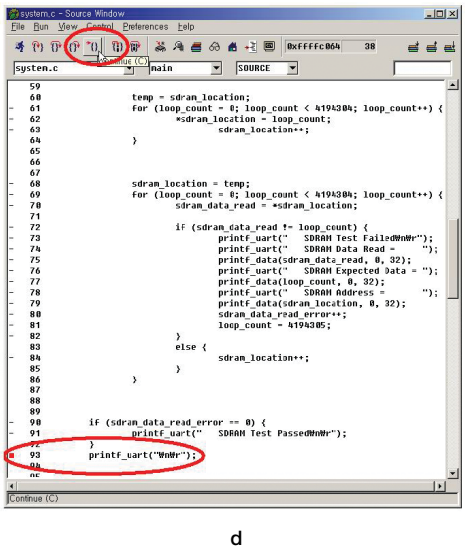
a



b



c



d

그림 5. GDB setting

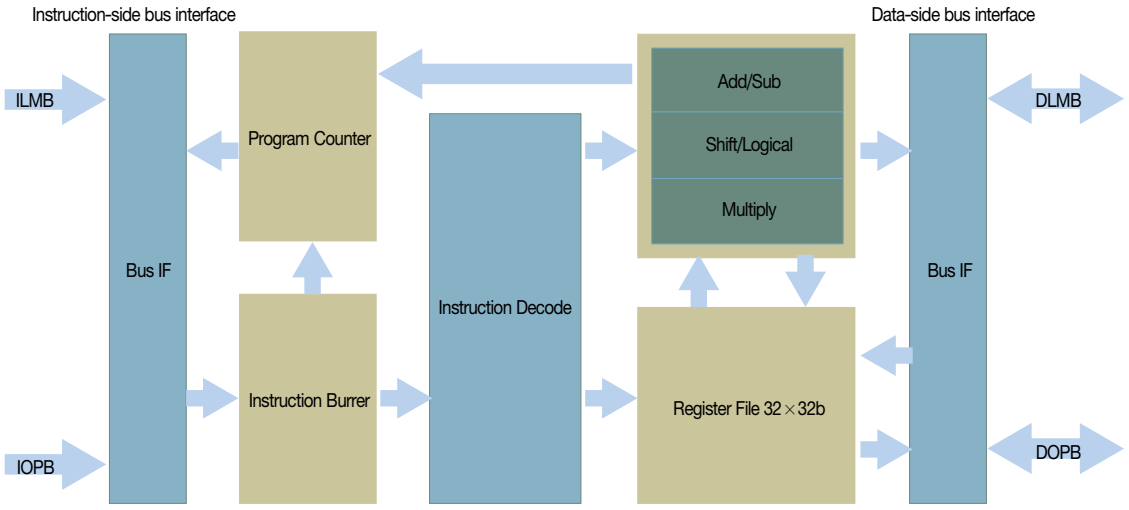


그림 6. Microblaze Core Block Diagram

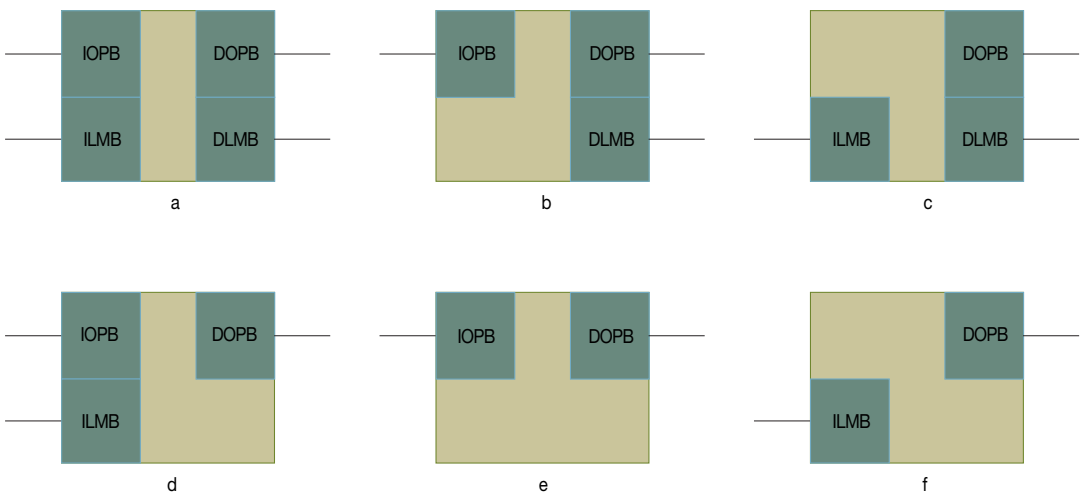


그림 7. MicroBlaze Bus Configurations

과 같이 구성된다.

아직 OPB에 연결되는 모든 신호들에 대해 정의를 하지 않았지만 일단 myDpramBport와 관련된 mhs 파일을 만들면 아래와 같다. 먼저 memory map을 보고, C_Bass Address와 C_High Address를 0xfffe_7000과 과 0xfffe_7fff로 각각 정의한다.

```
PARAMETER C_BASEADDR = 0xFFFE7000
PARAMETER C_HIGHADDR = 0xFFFE7FFF
```

현재 myDpramBport에서 B port만 FPGA 외부 핀과 연결된다. 따라서 local port와 global port를 선언하고 연결해 준다.

```
PORT MYDPRAMBPORT_ADDRB = MYDPRAMBPORT_ADDRB, DIR = IN, VEC = [0:9]
PORT MYDPRAMBPORT_CLKB = MYDPRAMBPORT_CLKB, DIR = IN
```

```
PORT MYDPRAMBPORT_WEB = MYDPRAMBPORT_WEB, DIR = IN
PORT MYDPRAMBPORT_ENB = MYDPRAMBPORT_ENB, DIR = IN
PORT MYDPRAMBPORT_DATA = MYDPRAMBPORT_DATA, DIR = INOUT, VEC = [0:7]
BEGIN opb_mydprambport
...
  PORT MYDPRAMBPORT_ADDRB = MYDPRAMBPORT_ADDRB
  PORT MYDPRAMBPORT_CLKB = MYDPRAMBPORT_CLKB
  PORT MYDPRAMBPORT_WEB = MYDPRAMBPORT_WEB
  PORT MYDPRAMBPORT_ENB = MYDPRAMBPORT_ENB
  PORT MYDPRAMBPORT_DATA = MYDPRAMBPORT_DATA
...
END
```

MyDpramBport을 설계하다 보면, 여러 개의 hardware version이 나올 수 있다. 여기서는 버전을 1.00b를 선택한다.

```
PARAMETER HW_VER = 1.00.b
```

MyDpramBport의 netlist 이름은 OPB_myDpram Bport로 정했다. 이 이름은 HW_VER과 같이 folder 이름으로 사용됩니다. **리스트 1**은 myDpramBport의 완성된 mhs 파일입니다.

Making microblaze

앞으로 설계할 user system은 **그림 11**과 같이 Microblaze core와 Microblaze가 fetch할 코드와 data가 저장된 BRam block, OPB에 연결할 myDpramBport와 UART로 구성되어 있다.

현재 Microblaze는 LMB를 사용해 Bram에서 코드와 데이터를 패치하며 OPB를 통해 외부 peripheral의 데이터를 access한다. 이때 LMB는 instruction side와 data side가 구분되어 있으므로 bus connection은 다음과 같이 정의된다.

```
BUS_INTERFACE DOPB = myopb
BUS_INTERFACE DLMB = d_lmb
BUS_INTERFACE ILMB = i_lmb
```

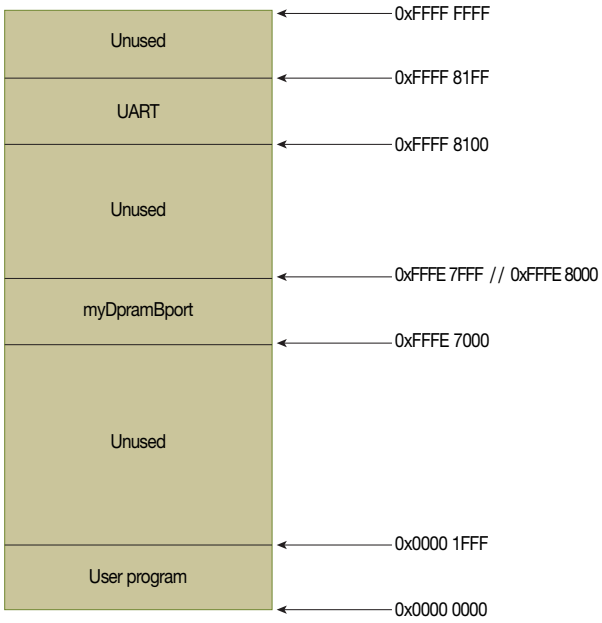


그림 8. Microblaze System Memory Map

```
PORT MYDPRAMBPORT_ADDRB = MYDPRAMBPORT_ADDRB, DIR = IN, VEC = [0:9]
PORT MYDPRAMBPORT_CLKB = MYDPRAMBPORT_CLKB, DIR = IN
PORT MYDPRAMBPORT_WEB = MYDPRAMBPORT_WEB, DIR = IN
PORT MYDPRAMBPORT_ENB = MYDPRAMBPORT_ENB, DIR = IN
PORT MYDPRAMBPORT_DATA = MYDPRAMBPORT_DATA, DIR = INOUT, VEC = [0:7]

BEGIN opb_mydprambport
  PARAMETER INSTANCE = opb_mydpramBPort1
  PARAMETER HW_VER = 1.00.b
  PARAMETER C_BASEADDR = 0xFFFE7000
  PARAMETER C_HIGHADDR = 0xFFFE7FFF
  PORT MYDPRAMBPORT_ADDRB = MYDPRAMBPORT_ADDRB
  PORT MYDPRAMBPORT_CLKB = MYDPRAMBPORT_CLKB
  PORT MYDPRAMBPORT_WEB = MYDPRAMBPORT_WEB
  PORT MYDPRAMBPORT_ENB = MYDPRAMBPORT_ENB
  PORT MYDPRAMBPORT_DATA = MYDPRAMBPORT_DATA
  BUS_INTERFACE SOPB = myopb
END
```

리스트 1. myDpramBport Hardware Spec

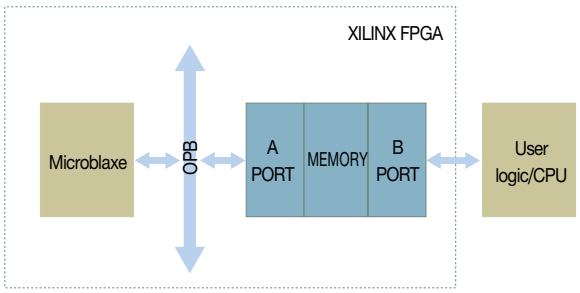


그림 9. myDpramBport block

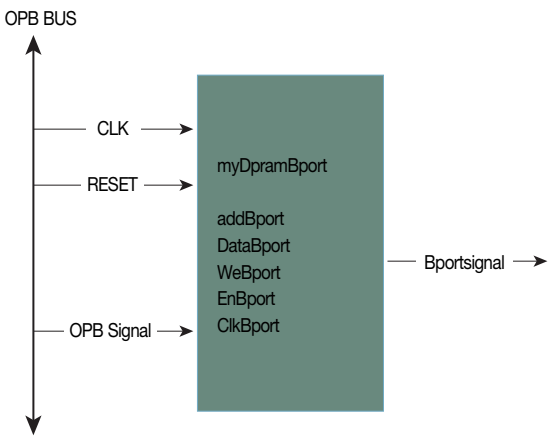


그림 10. myDpramBport interface

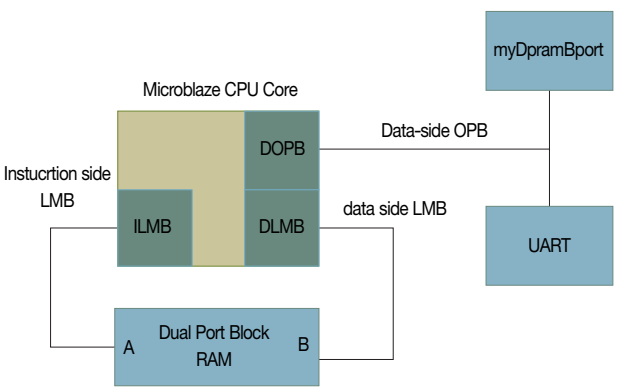


그림 11. Microblaze bus configuration 2

Microblaze는 netlist 형태로 제공되는 CPU이다. 따라서 netlist 버전과 instance 이름을 정해줘야 한다.

```
PARAMETER INSTANCE = mblaze
PARAMETER HW_VER = 1.00.c
```

Microblaze의 데이터 버스 크기를 정해주고 나머지 값들도 아래와 같이 정해준다.

```
PARAMETER C_DATA_SIZE = 32
PARAMETER C_USE_BARREL = 0
PARAMETER C_FSL_LINKS = 0
PARAMETER C_FSL_DATA_SIZE = 32
```

Microblaze의 clk입력포트에 clk신호를 연결해 준다.

```
PORT CLK = sys_clk
```

리스트 2는 Microblaze를 정의한 mhs 파일이다.

Making lmb_lmb_bram_if_cntlr and bram

그림 11을 보면 microblaze와 bram이 iLmb, dLmb로 연결된 것을 볼 수 있다. 이 블록을 좀더 자세히 살펴보면, Microblaze의 iLmb, dLmb와 Bram을 연결해주는 lmb_lmb_bram_if_cntlr 이 있는 것을 볼 수 있다.

Lmb_lmb_bram_if_cntlr은 Microblaze와 bram을 연결하는 역할을 하므로 4개의 bus connection이 준비되어야 한다. 이때 bram에는 data와 instruction이 같이 저장되어 있기 때문에 instruction과 data를 access하는 포트를 구별해서 연결한다.

```
BUS_INTERFACE ILMB = i_lmb
BUS_INTERFACE DLMB = d_lmb
BUS_INTERFACE PORTA = lmb_porta
BUS_INTERFACE PORTB = lmb_portb
```

그림 8에 있는 memory map에 의하면 모든 프로그램은 0x0000_0000에서 시작하는 것을 알 수 있다. 따라서

bram을 해당 address에서 디코딩 되도록 Lmb_lmb_bram_if_cntlr의 address를 정해줘야 한다.

```
PARAMETER C_BASEADDR = 0x00000000
PARAMETER C_HIGHADDR = 0x00001fff
```

Bram은 dual port memory이기 때문에 A, B port를 따로 연결할 수 있다.

```
BUS_INTERFACE PORTA = lmb_porta
BUS_INTERFACE PORTB = lmb_portb
```

리스트 3은 Lmb_lmb_bram_if_cntlr와 bram을 정의한 mhs파일이다.

Making LMB bus & OPB bus

LMB는 instruction과 data bus를 연결하기 위해 2개가 필요하고, OPB는 data bus만 연결하면 되므로 1개만 필요하다.

```
BEGIN lmb_v10
  PARAMETER INSTANCE = d_lmb
END

BEGIN lmb_v10
  PARAMETER INSTANCE = i_lmb
END

BEGIN opb_v20
  PARAMETER INSTANCE = myopb
END
```

각 bus는 clk과 reset 신호 및 reset active 조건이 정의되어야 하는데, 이 경우 low active로 정의했다.

```
PORT LMB_Clk = sys_clk
PORT SYS_Rst = system_reset
PARAMETER C_EXT_RESET_HIGH = 0
```

```
BEGIN microblaze
  PARAMETER INSTANCE = mblaze
  PARAMETER HW_VER = 1.00.c
  PARAMETER C_DATA_SIZE = 32
  PARAMETER C_USE_BARREL = 0
  PARAMETER C_FSL_LINKS = 0
  PARAMETER C_FSL_DATA_SIZE = 32
  PORT CLK = sys_clk
  BUS_INTERFACE DOPB = myopb
  BUS_INTERFACE DLMB = d_lmb
  BUS_INTERFACE ILMB = i_lmb
END
```

리스트 2. Microblaze hardware spec

리스트 4는 LMB, OPB를 정의한 mhs 파일이다.

Default folder structure and generate library

앞에서 설명한 것과 마찬가지로 edk는 **그림 12**와 같이 기본적으로 사용하는 folder가 있다(그림 12-a).

XPS 〉 File 〉 new project를 선택한 후 그림 12-b와 같이 system. mhs file과 target device, size, package 및 speed grade를 선택한다.

XPS 〉 Tools 〉 Generate Netlist를 선택하면 그림 12-c와 같이 여러 개의 파일과 mblaze라는 folder가 새로 생성된다. Mblaze라는 folder 이름이 생긴 이유는 Microblaze의 instance이름이 mblaze이기 때문이다. 이 mblaze folder 하위에 있는 include folder를 선택하면 그림 12-d와 같은 header file이 생긴 것을 알 수 있다.

include folder 밑에 있는 파일 중 관심을 가져야 하는 header file은 xpartameters.h와 xuartlite_l.h이다. xparameters.h는 **리스트 5**와 system.mhs 파일에 정의된 peripheral list의 ID와 BASEADDR, HIGHADDR이 #define에 의해서 새롭게 정의된 것을 볼 수 있다.

따라서 software를 coding할 때 #define으로 재정의 된 XPAR_MYUART_BASEADDR를 사용하면 이들 peripheral의 address가 달라져도 source code를 수정할 필요가 없다.

xuart_lite.h 파일에는 system.mhs file에서 정의한 uart를 사용하기 위한 기본적인 입출력 함수들이 정의되어

```
BEGIN lmb_lmb_bram_if_cntlr
  PARAMETER INSTANCE = mylmblmb_cntlr
  PARAMETER HW_VER = 1.00.a
  PARAMETER C_BASEADDR = 0x00000000
  PARAMETER C_HIGHADDR = 0x00001fff
  BUS_INTERFACE ILMB = i_lmb
  BUS_INTERFACE DLMB = d_lmb
  BUS_INTERFACE PORTA = lmb_porta
  BUS_INTERFACE PORTB = lmb_portb
END

BEGIN bram_block
  PARAMETER INSTANCE = bram_lmb
  PARAMETER HW_VER = 1.00.a
  BUS_INTERFACE PORTA = lmb_porta
  BUS_INTERFACE PORTB = lmb_portb
END
```

리스트 3. Define lmb_lmb_bram_if_cntlr and BRAM Hardware Spec

```
BEGIN lmb_v10
  PARAMETER INSTANCE = d_lmb
  PARAMETER HW_VER = 1.00.a
  PARAMETER C_EXT_RESET_HIGH = 0
  PORT LMB_Clk = sys_clk
  PORT SYS_Rst = system_reset
END

BEGIN lmb_v10
  PARAMETER INSTANCE = i_lmb
  PARAMETER HW_VER = 1.00.a
  PARAMETER C_EXT_RESET_HIGH = 0
  PORT LMB_Clk = sys_clk
  PORT SYS_Rst = system_reset
END

BEGIN opb_v20
  PARAMETER INSTANCE = myopb
  PARAMETER HW_VER = 1.10.a
  PARAMETER C_EXT_RESET_HIGH = 0
  PORT SYS_Rst = system_reset
  PORT OPB_Clk = sys_clk
END
```

리스트 4. Define LMB and OPB Hardware Spec

있다. **리스트 6**을 보면 UART를 Tx, Rx Port를 통해 데이터를 주고받을 수 있는 함수 가 정의되어 있다.

```
XUartLite_SendByte(Xuint32 BaseAddress, Xuint8 Data) ;
Xuinit8 XuartLite_RecvByte (Xuint32 BaseAddress);
```

XuartLite_SendByte()에서 첫 번째 argument가 BaseAddress인데, 이 값은 앞에 xparameters.h 파일에 있는 XPAR_MYUART_BASEADDR로 정의해서 사용하면 된다.

Making Source program

Source Code는 기본적으로 myDpramBport에 데이터를 write하고 다시 read해 write한 값과 read한 값이 서로 같은지 비교한다. 따라서 myDpramBport를 access하기 위한 변수와 UART를 access하기 위한 함수가 필요하다.

UART는 기본적으로 1byte를 읽어 serial로 바꿔 전송하므로 문자열을 전송하기 위해서는 **리스트 7**과 같은 함수를 정의한다.

myDpramBport를 access하기 위해서는 포인터형 변수가 필요하다. **리스트 8**에 sdram_location 이름으로 pointer형 변수가 선언되었고, 그 값은 0xfffe_7000으로 초기화 됐다. 이 값은 system.mhs에서 myDpramBport의 C_BASEADDR이며, xparameters.h에서 보면 XPAR_OPB_MYDPRAMBPORT1_BASEADDR이 0xfffe_7000으로 정의된 것을 볼 수 있다.

myDpramBport를 access하기 위한 변수는 리스트 8을 참조하면 되고, 이런 변수를 이용해 myDpramBport를 검사하는 함수는 아래와 같이 정의할 수 있다.

첫 번째 SDRAM Test는 myDpramBport에 write/read를 반복해서 검사한다.

```
printf_uart("\n\r\n\r 1) Test SDRAM");
    for (loop_count = 0; loop_count < 1024; loop_count++) {
        *sdram_location = loop_count;
        sdram_data_read = *sdram_location;
        if(sdram_data_read != loop_count) {
                printf_uart("\n\r");
```

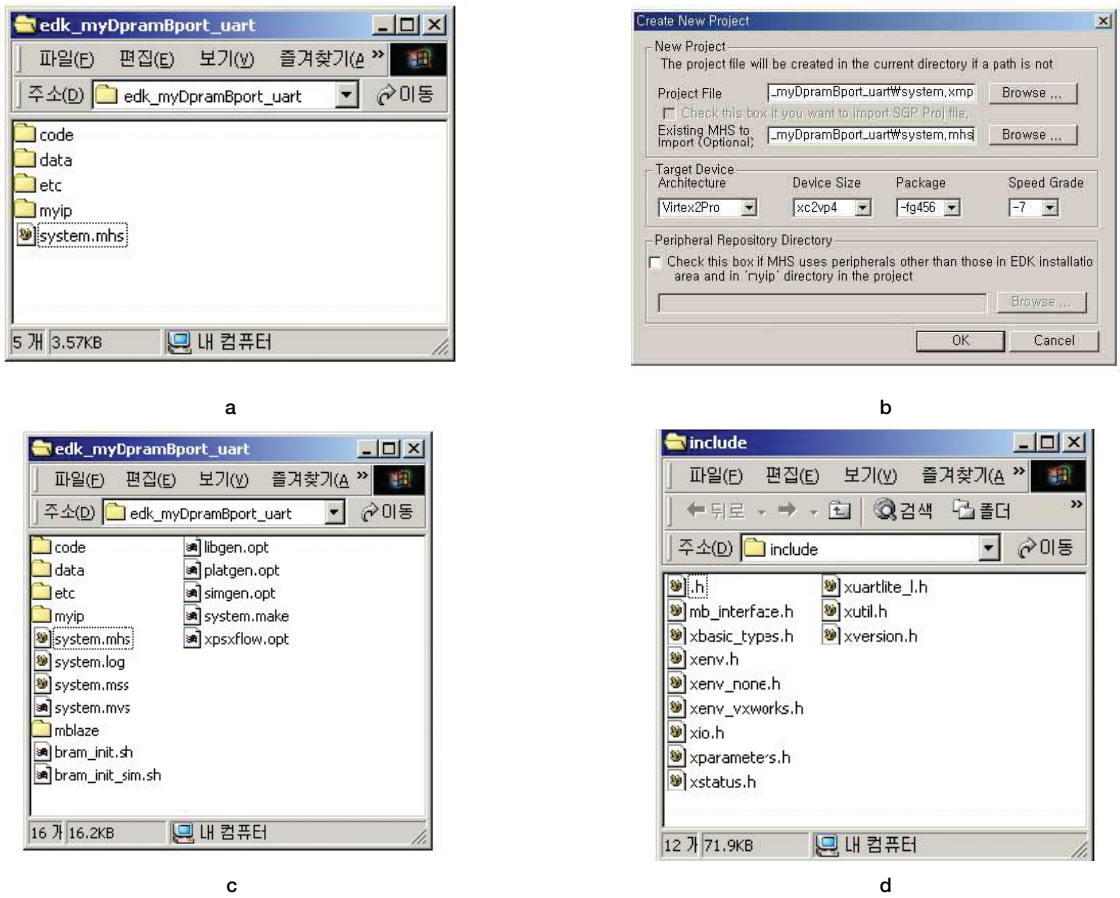


그림 12. Making project and generate library

```
            printf_uart("   SDRAM Address = ");
            printf_data(sdram_location, 0, 32);
            printf_uart("   SDRAM Expected Data = ");
            printf_data(loop_count, 0, 32);
            printf_uart("   SDRAM Data Read =  ");
            printf_data(sdram_data_read, 0, 32);
            error++;
    }
    else
            printf_uart(".");
    sdram_location++;
}
if(error == 0)
printf_uart("\n\r Test SDRAM 1 OK");
```

두 번째 SDRAM Test는 myDpramBport에 전체를 다 write하고 다시 처음부터 읽으면서 검사를 한다.

```
error = 0;
    printf_uart("\n\r\n\r 2) Test SDRAM");
    sdram_location = (unsigned int *)0xFFFE7000 ;
    for (loop_count = 0; loop_count < 1024; loop_count++) {
        sdram_data_read = *sdram_location;
        if(sdram_data_read != loop_count) {
            printf_uart("\n\r");
            printf_uart("   SDRAM Address =  ");
            printf_data(sdram_location, 0, 32);
            printf_uart("   SDRAM Expected Data = ");
            printf_data(loop_count, 0, 32);
```

```
#ifndef XPARAMETERS_H   /* prevent circular inclusions */
#define XPARAMETERS_H   /* by using protection macros */
#define XPAR_MYUART_DEVICE_ID 0

#define XPAR_XUARTLITE_NUM_INSTANCES 1

/* Base addresses of peripherals */
/* _BASEADDRESS means they are either stdin, stdout
or the interrupt controller */
#define XPAR_OPB_MYDPRAMBPORT1_BASEADDR 0xFFFE7000
#define XPAR_OPB_MYDPRAMBPORT1_HIGHADDR 0xFFFE7FFF
#define XPAR_MYUART_BASEADDR 0xFFFF8100
#define XPAR_MYUART_HIGHADDR 0xFFFF81FF
#define XPAR_MYLMBLMB_CNTLR_BASEADDR 0x00000000
#define XPAR_MYLMBLMB_CNTLR_HIGHADDR 0x00001FFF
#endif
```

리스트 5. xparamters.h

```
#define XUartLite_mDisableIntr(BaseAddress) \
        XUartLite_mSetControlReg((BaseAddress), \
        XUartLite_mGetControlReg((BaseAddress)) &
~XUL_CR_ENABLE_INTR)
void XUartLite_SendByte(Xuint32 BaseAddress, Xuint8 Data);
Xuint8 XUartLite_RecvByte(Xuint32 BaseAddress);
```

리스트 6. xuart_lite.h

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

리스트 7. printf_uart

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

리스트 8. variable of point

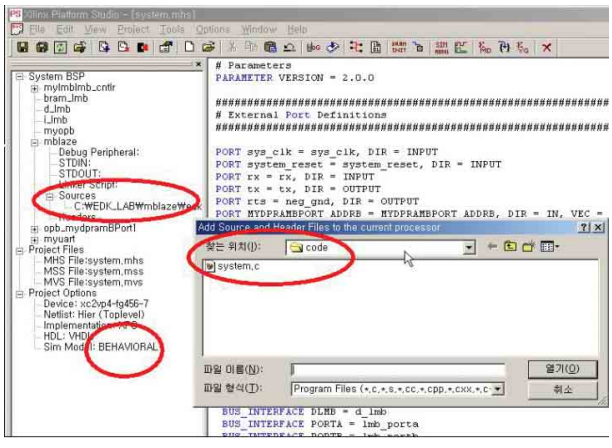


그림 13. Registor source code

```
            printf_uart("   SDRAM Data Read =  ");
            printf_data(sdram_data_read, 0, 32);
            error++;
    }
    else
            printf_uart(".");
```

xparameters.h 파일 내에는 C_BASEADDR = 0xffffe_7000, C_HIGHADDR = 0xffffe_7fff로 정의되어 있기 때문에 myDpramBport의 크기는 모두 4Kbyte이다. 하지만, source에서는 for loop에서는 모두 1024, 즉 1K 만큼만 access하는데, 그 이유는 int라는 변수가 microblaze에는 모두 32bit의 크기를 가지게 되어 한번 access할 때마다 4byte씩 access하기 때문이다.

이 source code를 code folder 밑에 system.c로 저장한 후 XPS에 Source를 등록시켜야 한다.

그림 13과 같이 mblaze 하위 메뉴인 "Source"를 선택하면 source code 입력창이 나타나며, system.c를 source code로 선택한다.

이번 호에는 user define peripheral을 프로그램을 통해 제어하는 방법에 대해서 설명하였다. 다음 호에는 user define peripheral을 modelsim을 통해 시뮬레이션 하는 방법에 대해서 설명하겠다.