Xilinx Virtex-II Pro를 이용한 SoC 설계(Ⅱ)

EDK Tutorial 및 데모

지난 시간에 이어 이번 시간에는 GPIO 및 SDRAM Controller를 설계해서 system.mhs 파일에 추가해 hardware design을 마치고, software design을 하기 위한 파일들에 대해 알아본다. 디바이스 드라이버나 boot 코드는 hardware와 매우 밀접한 관계를 갖기 때문에 hardware, software에 대해 잘 알고 있어야 프로그램 할 수 있는 부분이다. 자일링스는 이 부분을 자동화시킴으로써 유저로 하여금 근본적으로 user function에 집중할 수 있도록 배려해 놓았다.

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

Making GPIO

GPIO를 구현하기 위해서는 **그림 1**과 같이 연결할 버스와 포트의 종류가 입력인지, 출력인지를 먼저 정해야 한다. GPIO를 결정하는 요소로는 다음과 같다.

```
GPIO 입출력
GPIO 포트 숫자
Base Address, High Address
```

Base Address와 High Address는 메모리 맵 상에 나와 있는 것처럼 0xF000_0300, 0xF000_03FF로 정의한다.

```
PARAMETER C_BASEADDR = 0xF0000300
PARAMETER C_HIGHADDR = 0xF00003ff
```

GPIO의 입력으로만 사용하는 경우도 있지만 입출력을 user program에 의해서 정할 수도 있다. C_ALL_ INPUTS 값을 '0'으로 정하면 user program에서 입력과 출력을 정할 수 있으며 '1'로 정하는 경우에는 입력으로만 사용가능하다.

```
PARAMETER C_ALL_INPUTS = 0
```

GPIO 포트 숫자를 정해준다.

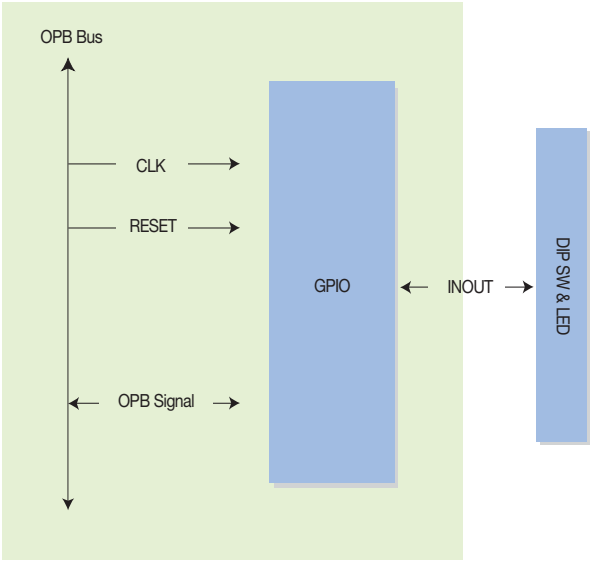


그림 1. GPIO Connection

```
PARAMETER C_GPIO_WIDTH = 4
```

GPIO의 local port 이름은 GPIO_IO로 정해져 있는데, 이 port와 FPGA 핀과 연결해 줘야 한다. 이 local port를 통해 모두 4비트가 액세스되기 때문에 global port인 leds는 VEC=[0:3]으로 정해준다.

```
PORT leds = leds, DIR = INOUT, VEC = [0:3]

BEGIN opb_gpio
  PORT GPIO_IO = leds
END
```

나머지 Parameter는 uart에서 설명한 것과 같은 원리로 생각하면 된다.

```
PARAMETER INSTANCE = mygpio1
PARAMETER HW_VER = 1.00.a
PORT OPB_Rst = peripheral_rst
BUS_INTERFACE SOPB = myopb
```

Making SDRAM Controller

SDRAM Controller를 구현하기 위한 스펙으로는 **그림 2**와 같이 FPGA와 연결된 SDRAM의 SPEC에 따라 정할 PARAMETER 값이 다양하다.

앞에 uart나 gpio처럼 Base Address와 High Address를 정해준다. Ram size는 0xF1FF_FFF – 0xF100_0000 + 1 = 0x100000 bits이며, 이 메모리가 32비트 단위로 액세스되기 때문에 C source에서 0x1000000/4 = 0x400000 = 4194304의 index 값을 가지게 된다.

```
PARAMETER C_BASEADDR = 0xF1000000
PARAMETER C_HIGHADDR = 0xF1FFFFFF
```

SDRAM은 CAS, RAS, BANK와 같이 요구되는 많은 PARAMETER가 있다. 이 값에 대한 자세한 내용은 해당

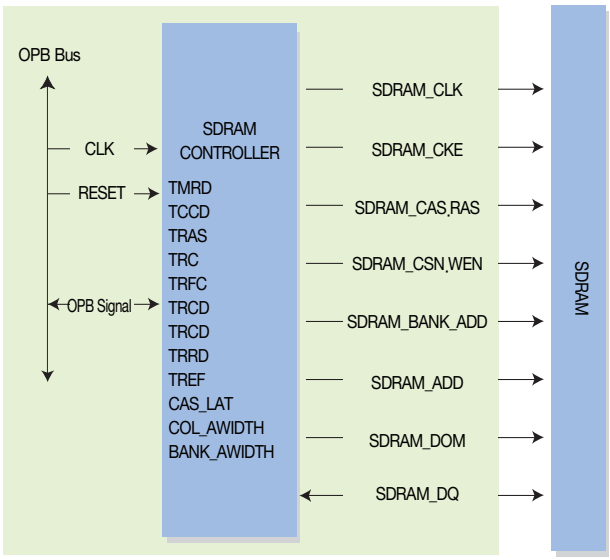


그림 2. SDRAM Controller Connection

SRAM을 참조하기 바라며, 각 파라미터 중 시간을 정의하는 단위는 모두 psec 단위이다.

```
PARAMETER C_SDRAM_TMRD = 15000
PARAMETER C_SDRAM_TCCD = 1
PARAMETER C_SDRAM_TRAS = 48000
PARAMETER C_SDRAM_TRC = 70000
PARAMETER C_SDRAM_TRFC = 75000
PARAMETER C_SDRAM_TRCD = 19000
PARAMETER C_SDRAM_TRRD = 16000
PARAMETER C_SDRAM_TRP = 19000
PARAMETER C_SDRAM_TREF = 64
PARAMETER C_SDRAM_CAS_LAT = 2
PARAMETER C_SDRAM_COL_AWIDTH = 9
PARAMETER C_SDRAM_BANK_AWIDTH = 2
PARAMETER C_SDRAM_AWIDTH = 12
PARAMETER C_SDRAM_DWIDTH = 32
```

Local Port와 global port를 연결해주고 hardware version, instance name, bus 연결에 대해 정의한다.

```
PORT sdram_clk = sys_clk, DIR = output
PORT sdram_cke = sdram_cke, DIR = output
PORT sdram_csn = sdram_csn, DIR = output
```

```
PORT sdram_rasn = sdram_rasn, DIR = output
PORT sdram_casn = sdram_casn, DIR = output
PORT sdram_wen = sdram_wen, DIR = output
PORT dqm = dqm, DIR = output, VEC = [0:3]
PORT ba = ba, DIR = output, VEC = [0:1]
PORT a = a, DIR = output, VEC = [0:11]
PORT dq = dq, DIR = inout, VEC = [0:31]

BEGIN opb_sdram
 PARAMETER INSTANCE = sdram1
 PARAMETER HW_VER = 1.00.c
 PORT opb_clk = sys_clk
 PORT sdram_clk_in = sys_clk
 PORT sdram_cke = sdram_cke
 PORT sdram_csn = sdram_csn
 PORT sdram_rasn = sdram_rasn
 PORT sdram_casn = sdram_casn
 PORT sdram_wen = sdram_wen
 PORT sdram_dqm = dqm
 PORT sdram_bankaddr = ba
 PORT sdram_addr = a
 PORT sdram_dq = dq
 BUS_INTERFACE SOPB = myopb
END
```

Make new XPS project

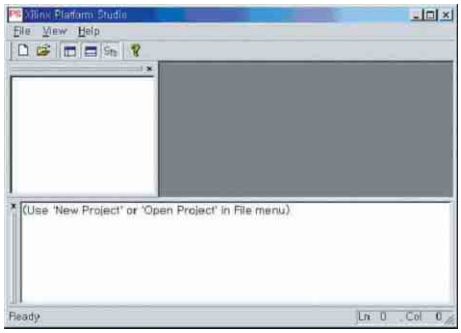
XPS를 시작하려면, 시작〉프로그램〉Xilinx Embedded Development Kit〉Xilinx Platform Studio를 선택하면 XPS가 시작된다(그림 3a). VIRTEX-II Pro의 PPC405를 이용한 project 파일의 확장자는 xmp이고, project를 만들 폴더를 선택한다(그림 3b, 3c). 마지막으로 targetting 될 디바이스를 Virtex2pro, xc2vp4, -fg456, -7로 선택한다 (그림 3d).

Import MHS File

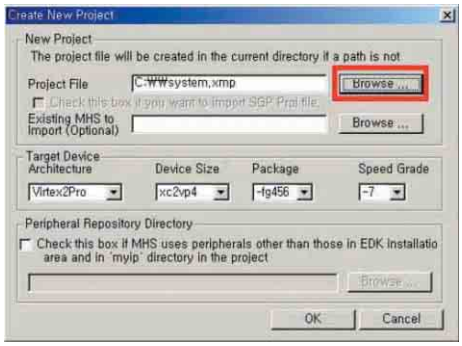
시스템을 구성하는 하드웨어를 정의한 system.mhs 파일을 지정해 준다(그림 4a, 4b). 그리고 나면 XPS는 system.mhs 파일을 읽어 어떤 퍼리퍼럴을 사용하고, 각 instance의 이름이 어떻게 되는지 보여준다. mygpio1, mygpio2, myuart, myplb, myopb가 정의된 것을 볼 수 있다(그림 4c).

Setting Up The Compiler Options and PPC

그림 5와 같이 XPS에서 PPC405_i를 선택하면 PPC405

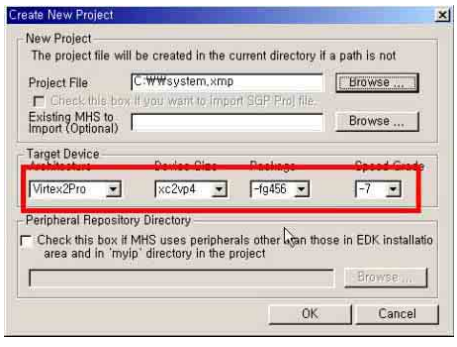


a



b

c



d

그림 3. Make new XPS project

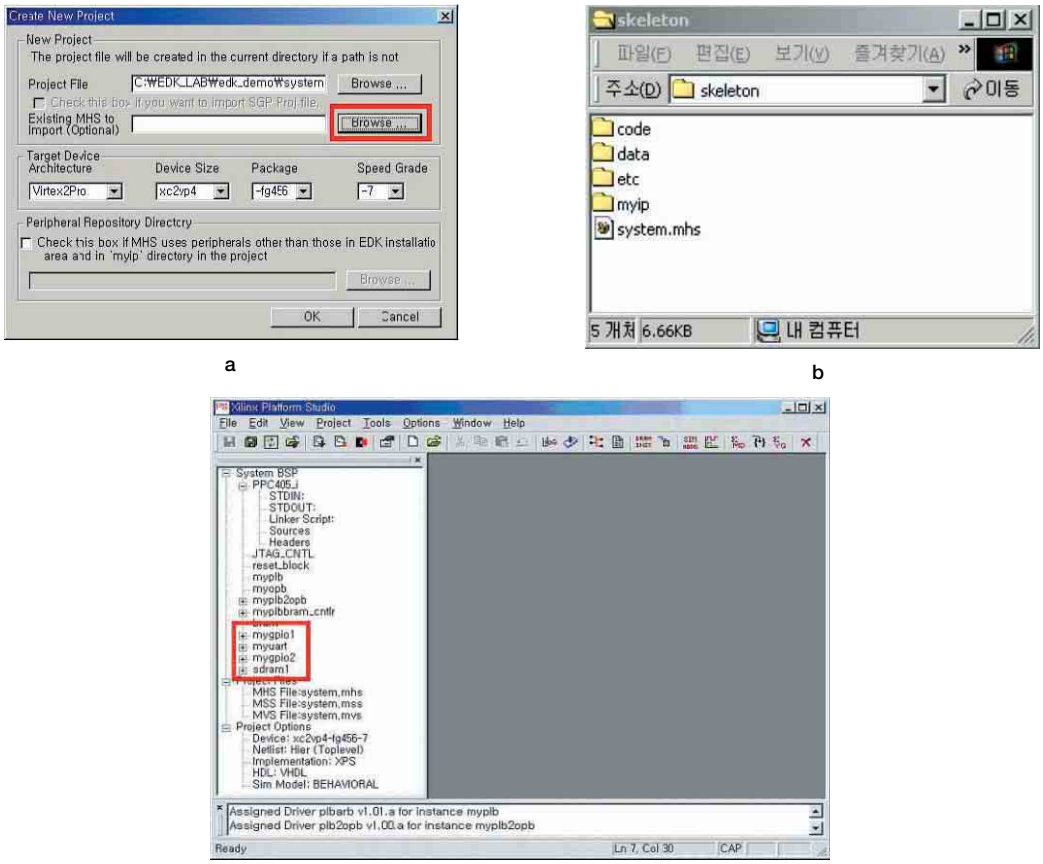


그림 4. Import MHS File

의 STDIN, STDOUT 및 compile option을 정할 수 있다.

일반적으로 STDIN, STDOUT은 C 프로그램을 코딩할 때, printf()이나 getch()와 같은 ANSI C의 표준 함수를 사용할 때 어느 퍼리퍼럴을 통해 기본적인 입출력 포트로 사용할 지 결정하게 된다.

프로그램을 compile이나 link할 때 필요한 파일의 경로를 정할 수 있으며, 디폴트 폴더는 system.mhs에 있는 PPC405 instance 이름으로 만들어진 폴더 밑에 code, include, lib, libsrc 폴더가 생긴다. Software flow에서 나온 최종 파일은 일반적으로 bin이나 hex 파일이지만, XPS에서는 ELF(executable and link format) 파일이 최종 파일이다. XPS는 이 파일과 나중에 implementation이 끝나고 나온 bit 파일과 같이 합쳐져 software 정보가 포함된 bit 파일을 만들어준다.

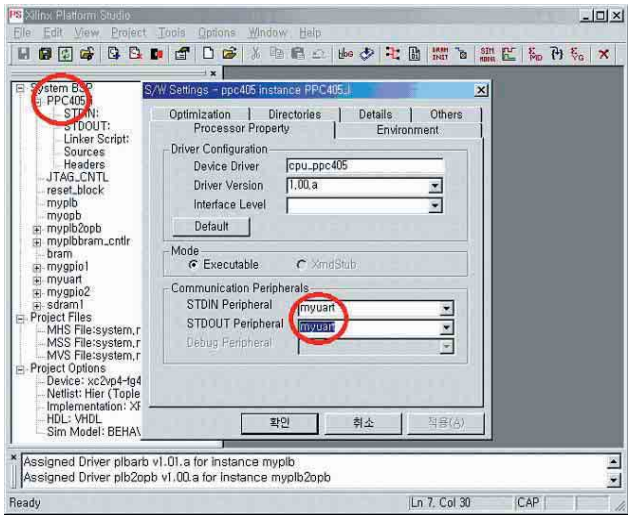


그림 5. Assign STDIN, STDOUT

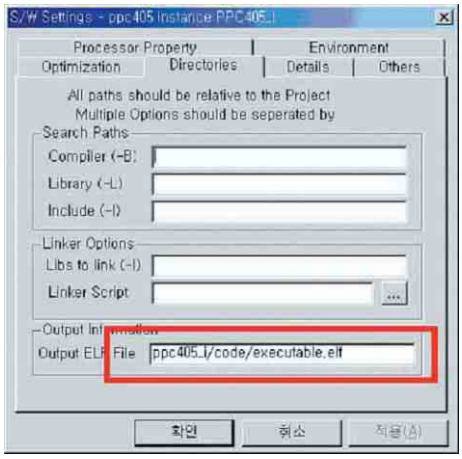
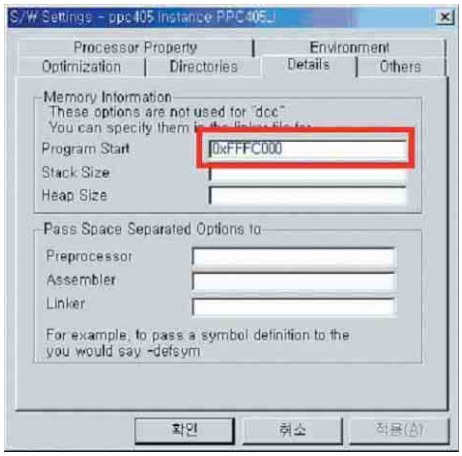
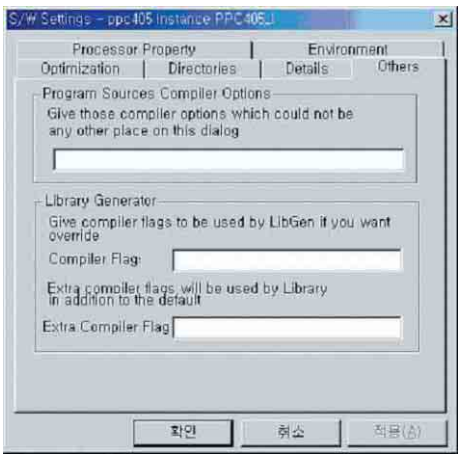


그림 6. Specifying The Location Of the Output File



a



b

그림 7. Setting Program start address and other option

system.mhs 파일에 보면, plb_bram_if_cntlr의 PARAMETER 중 C_BASEADDR 값이 0xffff_c000으로 정해진 것을 확인할 수 있다. 이 값은 사용자가 작성한 main() 함수가 0xffff_c000부터 시작한다는 것을 알려준다. 이렇게 되면 boot 코드를 만들 때 main 함수가 0xffff_c000에 있다고 가정하고 boot를 만들어준다. 이 값을 지정하지 않으면 XPS는 0xffff_0000에 main 함수가 있다고 가정한다. Stack Size 기본 값은 256이다(**그림 7a**). 나머지 Other option은 blank로 둔다(**그림 7b**).

Assigning Software Driver to The UART, GPIO

Uart와 GPIO의 Interface level은 0으로 정한다. 이 Interface level은 0, 1로 되어 있는데 일반적으로 0, 1은 standalone으로 사용할 때 정하고, 2는 RTOS를 사용하는 경우 사용한다(**그림 8**).

Generate Library and Header File

그림 9는 library를 만들기 전후의 폴더 구조를 보여준다. XPS 〉 tools 〉 Generate Libraries를 선택하면 library를 만들 수 있다. **그림 9b**를 보면 폴더 PPC405_i가 새로 생긴 것을 볼 수 있다.

그림 10을 보면 PPC405_I folder 밑에 include folder가 만들어진 것을 볼 수 있다(**그림 10a**). Include folder에는 system.mhs 파일에서 정의한 퍼리퍼럴을 액세스하기 위한 디바이스 드라이버가 만들어져 있다. 예를 들어, gpio용 디바이스 드라이버는 xgpio_l.h에 있고 uart용 디바이스 드라이버는 xuartlite_l.h에 있다. 하지만, sdram용 디바이스 드라이버는 따로 있지 않다. 그것은 sdram controller가 있기 때문에 user program에서는 디바이스 드라이버를 따로 호출하지 않아도 SDRAM을 액세스 할 수 있기 때문이다.

Setting mygpio1, mygpio2 as input or output

xparameters.h를 보면 mygpio1, mygpio2를 액세스하기 위한 Address 값을 #define문으로 정의한 것을 볼 수 있다.

```
#define XPAR_MYGPIO1_BASEADDR  0xF0000300
```

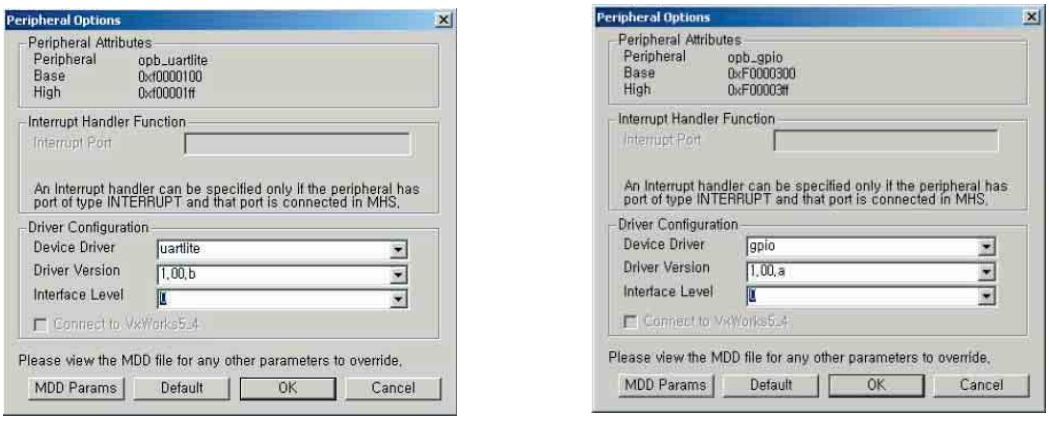


그림 8. UART, GPIO Software Driver Setup

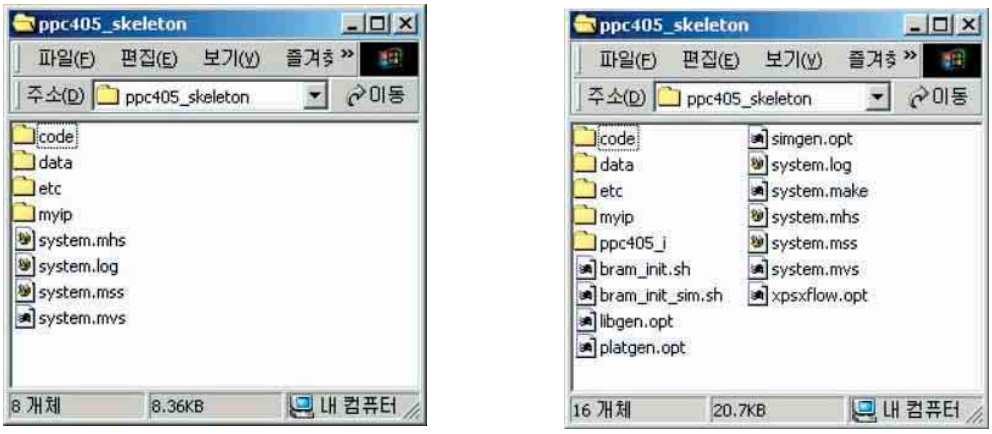


그림 9. Folder Structure before & after Generate Libraries

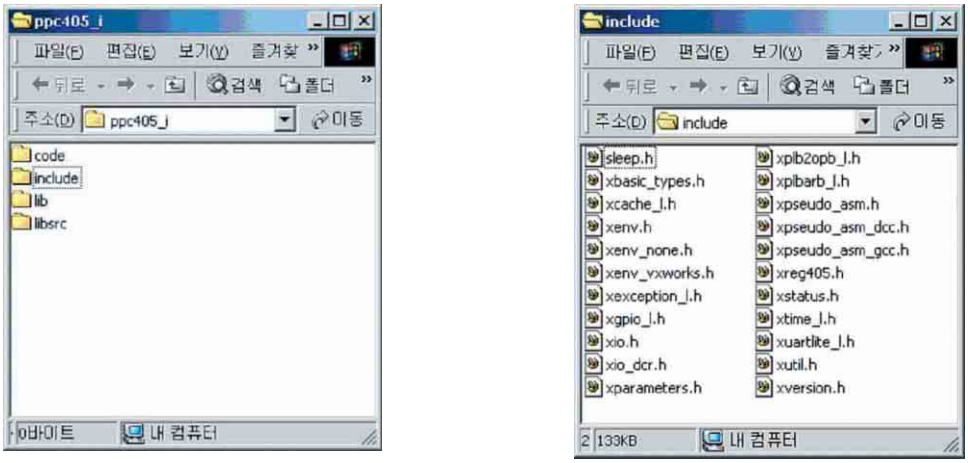
```
#define XPAR_MYGPIO1_HIGHADDR 0xF00003FF
#define XPAR_MYGPIO2_BASEADDR 0xF0000200
#define XPAR_MYGPIO2_HIGHADDR 0xF00002FF
```

gpio 입출력은 하드웨어적으로 정하는 것이 아니라 software로 정할 수 있도록 system.mhs 파일에서 PARAMETER 값을 아래와 같이 정했다.

```
PARAMETER C_ALL_INPUTS = 0
```

따라서 mygpio1은 led 출력으로, mygpio2는 dip sw 및 tact sw 입력으로 정의해야 한다. xgpio_l.h에서는 각 gpio의 입출력을 정할 수 있도록 디바이스 드라이버가 만들어져 있다.

```
#define XGpio_mSetDataDirection(BaseAddress, DirectionMask)₩
    XGpio_mWriteReg((BaseAddress), XGPIO_TRI_OFFSET, (DirectionMask))

XGpio_mSetDataDirection(XPAR_MYGPIO1_BASEADDR, 0x00) // setting as output
```



a b

그림 10. PPC405_I & include folder

```
XGpio_mSetDataDirection(XPAR_MYGPIO2_BASEADDR, 0xfff) // setting as input
```

mygpio1을 output port로 하기 위해서는 먼저 mygpio1의 address를 알아야 한다. 이 어드레스는 xpartameters.h에 XPAR_MYGPIO1_BASEADDR로 정의되어 있다. 따라서 mygpio1을 output port로 하기 위해서는 아래와 같은 함수를 호출하면 된다.

```
XGpio_mSetDataReg(XPAR_MYGPIO1_BASEADDR, 0x00);
```

mygpio2을 input port로 하기 위해서는 mygpio1과 마찬가지로 xparrameters.h에서 어드레스를 참조하고 아래와 같은 함수를 호출한다.

```
XGpio_mSetDataReg(XPAR_MYGPIO2_BASEADDR, 0xff);
```

1. Setting UART as stdin, stdout

Xuartlite_h를 보면 uart를 read, write 할 때 필요한 함수들이 정의된 것을 알 수 있다.

```
#define XUartLite_mIsTransmitFull(BaseAddress)₩
        (XUartLite_mGetStatusReg((BaseAddress)) &
        XUL_SR_TX_FIFO_FULL)

void XUartLite_SendByte(Xuint32 BaseAddress, Xuint8 Data);
Xuint8 XUartLite_RecvByte(Xuint32 BaseAddress);
```

Uart를 액세스하려면 uart의 address를 알아야 하는데, 이 address는 xparameters.h에서 찾을 수 있다.

```
#define XPAR_MYUART_BASEADDR 0xF0000100
#define XPAR_MYUART_HIGHADDR 0xF00001FF
```

이 함수와 address가 define 된 것을 사용하면 **리스트 1**과 같이 print_uart() 함수를 만들 수 있다.

2. Setting SDRAM Access Point

xparameter.h에는 SDRAM을 액세스하기 위한 Address가 정의되어 있다.

```
#define XPAR_SDRAM1_BASEADDR 0xF1000000
```

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

리스트 1. function print_uart

이 값을 pionter로 취해서 한 개 변수로 할당하면 memory를 read, write하는 함수를 만들 수 있다(**리스트 2**).

3. Making Source Code

Source code에는 xparameters.h, xgpio_l.h 그리고 xuartlite_l.h가 반드시 포함되어야 하며 나머지 function.h는 유저가 어떻게 프로그램을 코딩하느냐에 따라 결정된다.

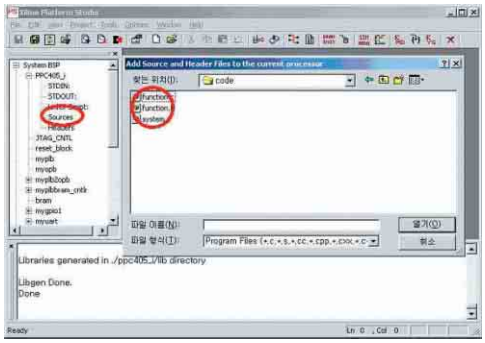
```
#include "xuartlite_l.h"
#include "xgpio_l.h"
#include "xparameters.h"
#include "function.h"

#define true 1
#define false 0

main() {
....
.....
}
```

4. Adding Source File to The Project

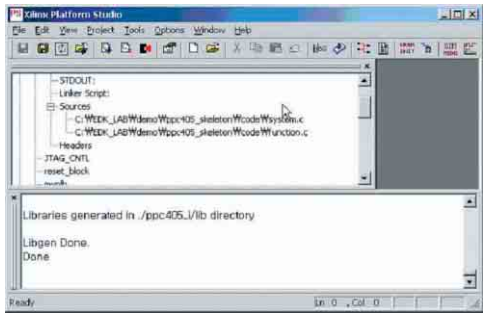
이제 source code를 XPS 등록해야 한다. XPS GUI에서 Source를 선택한 후 code 폴더로부터 system.c와 function.c를 지정한다(**그림 11a**). 그러면 **그림 11b**와 같이 source code 2개가 프로젝트에 올라가 있는 것을 확인할 수 있다.

```
unsigned int * sdram_location = (unsigned int * )XPAR_SDRAM1_BASEADDR;

sdram_data_read_error = 0;

temp = sdram_location;
for (loop_count = 0; loop_count < 0x400000; loop_count++) {
    * sdram_location = loop_count;
    sdram_location++;
}

sdram_location = temp;
for (loop_count = 0; loop_count < 0x400000; loop_count++) {
    sdram_data_read = * sdram_location;
    if (sdram_data_read != loop_count) {
        sdram_data_read_error++;
    }
    else {
        sdram_location++;
    }
}
```

리스트 2. Memory access program

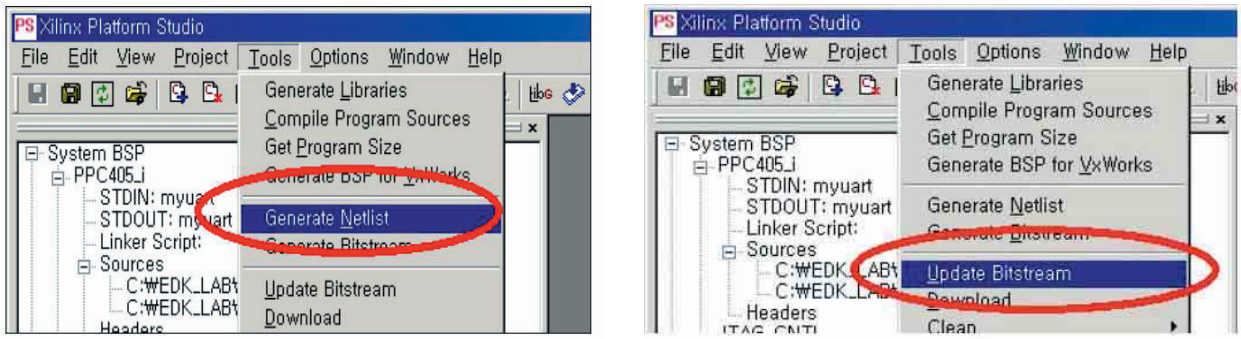


a



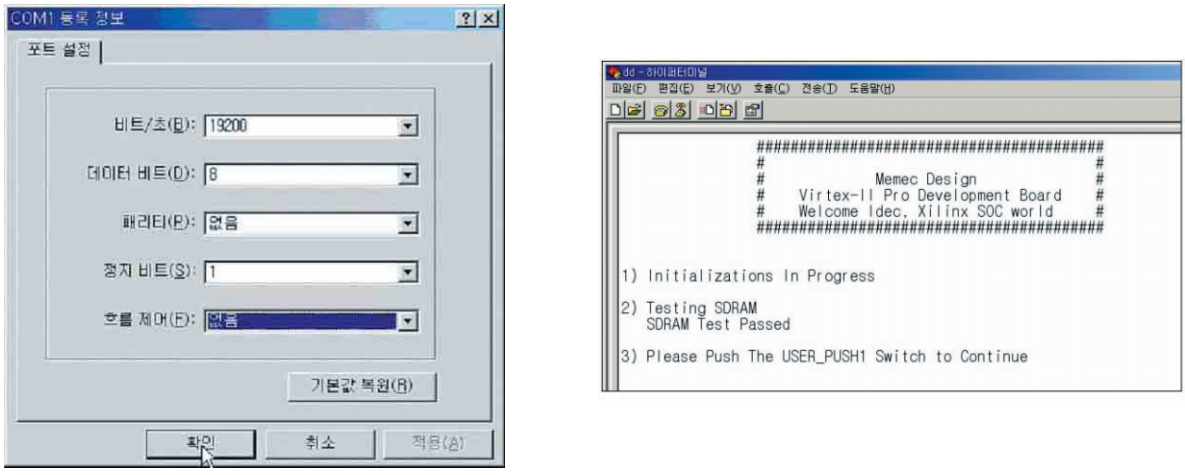
b

그림 11. Adding Source File to The Project



a b

그림 12. Generate Netlist(a)와 Update Bitstream(b)



a b

그림 13. Hyper Terminal Port Settings and output

Implementing The Design

1. User Code Compilation & netlist, bit file generation

Tools 〉 Compile Program Sources를 선택하면 현재 source로 정의된 system.c와 function.c가 컴파일 된다. Netlist를 만들기 위해 Tools 〉 Generate Netlist를 선택한다. 이 과정은 synthesis 과정과 같다(**그림 12a**). Bit file을 만들기 위해서 Tools 〉 Update Bitstream을 선택한다. 이 과정은 implementation 과정과 같다(**그림 12b**).

2. Running The Application Program

Hyper Terminal의 정보를 **그림 13a**와 같이 할당한 후, CPU RESET 스위치로 CPU를 RESET시키면 **그림 13b**와 같은 메시지가 출력된다.

비록 simple process block 내부에 대해서 자세히 다루지 않았지만 uart, gpio, sdram controller를 text base로 설계하는 방법에 대해서 설명했다.

프로그래머를 괴롭히는 상황은 여러 가지가 있지만, 함수 이름정하는 것이나 변수 이름 정하는 것도 고생 아닌 고생이다. 자일링스에서 나오는 device driver program을 이용하면 이런 염려는 하지 않아도 된다. 모든 것이 자동이므로 유저는 user function에만 집중할 수 있다.

다음호에서는 C code를 디버깅하는 방법과 hardware와 software를 같이 modelsim에서 simulation하는 방법에 대해 설명하겠다. RTime