

하드웨어 소프트웨어 통합설계

Hello SystemC



학 과: 컴퓨터정보공학부

담당교수: 이준환교수님

학 번: 2017202087

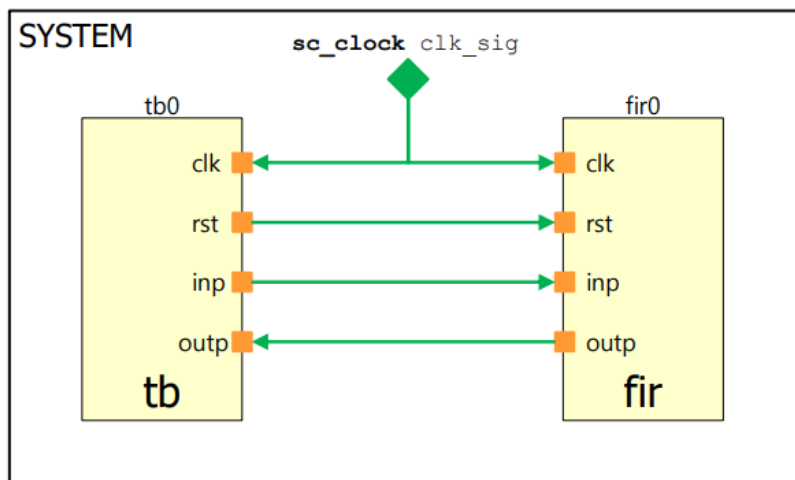
성 명: 홍 세 정

1. Problem statement

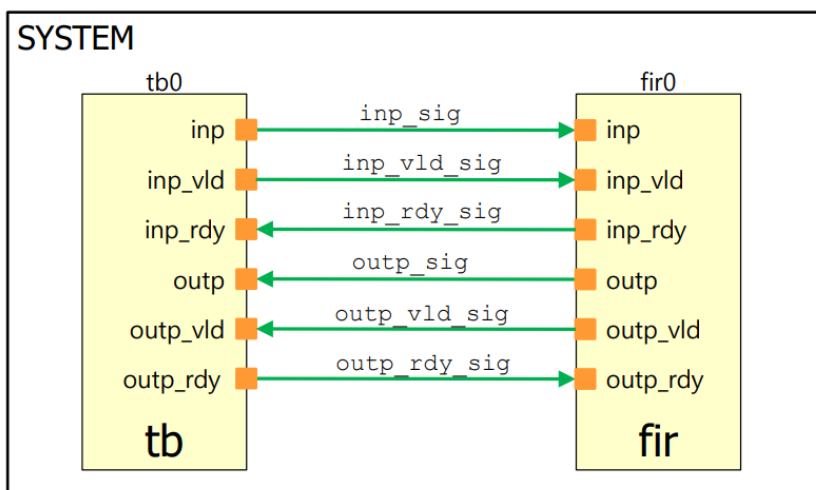
이번 과제는 FIR(finite impulse response)를 설계해보고 확인해본다.

또한 handshaking signal 사용과 testbench module을 설계해본다. SYSTEM_wHS는 handshaking signal을 사용하고, SYSTEM은 사용하지 않고 구현한다. 각 Fir과 tb를 구현한다. 모듈 fir 및 tb는 port와 signal를 사용하여 동시에 통신한다.

SYSTEM module



SYSTEM_sHS module



Test environment은 다음과 같다.

다음과 같이 input과 output을 구분하여 알맞게 port, wire을 연결할 수 있다.

- Role of handshake signal

Input과 output을 주고받을 때 valid한 input과 받을 준비가 되어있다는 signal을 주고 받는다.

Sender와 receiver로 나뉘고 두개의 쌍을 handshake signal이라고 불린다. Sender와 receiver가 서로 소통하며 확인한다.

Sender은 data를 보낼 때 valid 신호를 같이 보낸다. Valid가 1일때만 data를 받아들인다.

Receiver는 valid가 1이면 무조건 받을 수 있는 것이 아니다. 받을 준비가 완료되었을 때만 data를 받아들인다. 데이터의 수신을 확인한다. (Ready signal)

Sender와 receiver가 모두 1일 때 data를 받을 수 있다.

- Why they are needed

수행되는 순서가 정해져 있지 않기 때문이다.

세계 simulation processes data들을 독립적으로 하드웨어처럼 생산하고 소비하도록 한다. source와 sink가 동시에 일어나기 때문에 그렇다. simulation때문에 생기는 문제 port로 해결로 하였다.

Fir_main이 입력을 받아들인 후 출력하는 것이 한 cycle에 마다 일어나지 않는다. 매 사이클 output in multiple cycle이 걸리면, 들어온 input을 한 cycle에 제대로 처리하지 못하는 경우가 발생하여서 handshake signal이 필요하다.

- What happen if no handshake signals are available

사용하지 못하면 output을 제대로 반환하지 못한다.

Delay를 사용하지 않게 되면 input값을 받아서 처리하는 일을 하지 못한다. wait하는 동안 input이 변하게 되면 못 받아들이는 상황이 생긴다. Input을 받아들일 준비가 되지 않았는데 input을 받으면 문제를 일으킬 수 있다.

Input이 변하지 않았을 때 output이 변하지 않으므로 input을 여러 번 받았다고 생각하지 못한다.

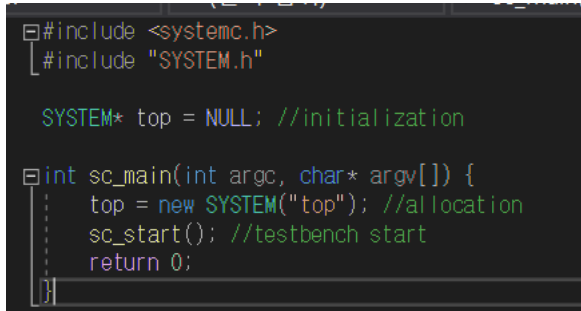
- How this project shows the above

SYSTEM module과 SYSTEM_wHS module을 이용하여 handshake signal을 확인해볼 수 있다. SYSTEM_wHS에는 handshake signal을 만들어서 delay를 주어서 확인해본다. Inp_vld_sig과 inp_rdy_sig, outp_vld_sig, outp_rdy_sig을 만들어서 연결해주고 vld와 rdy 모두 1일 때만 작동하는 것을 확인한다.

2. SystemC code

SYSTEM module

- Main.cpp



```
#include <systemc.h>
#include "SYSTEM.h"

SYSTEM* top = NULL; //initialization

int sc_main(int argc, char* argv[]) {
    top = new SYSTEM("top"); //allocation
    sc_start(); //testbench start
    return 0;
}
```

Main.cpp는 다음과 같다. Top은 다음과 같이 선언하고 할당해준다.

Sc_start로 testbench를 시작한다.

- SYSTEM.h

```
#include <systemc.h>
#include "fir.h"
#include "tb.h"

SC_MODULE(SYSTEM) {
    tb* tb0;
    fir* fir0;
    //module instaciation

    sc_signal<bool> rst_sig;
    sc_signal<sc_int<16>> inp_sig;
    sc_signal<sc_int<16>> outp_sig;
    //wire
    sc_clock clk_sig; //clock signal

    SC_CTOR(SYSTEM) : clk_sig("clk_sig", 10, SC_NS) { // clock period 10ns
        tb0 = new tb("tb0");
        tb0->clk(clk_sig);
        tb0->rst(rst_sig);
        tb0->inp(inp_sig);
        tb0->outp(outp_sig);
        //testbench connect

        fir0 = new fir("fir0");
        fir0->clk(clk_sig);
        fir0->rst(rst_sig);
        fir0->inp(inp_sig);
        fir0->outp(outp_sig);
        //fir connect
    }
    //instanciation & port connection

    ~SYSTEM() {
        delete tb0;
        delete fir0;
    }
    //desructor
};
```

Testbench와 fir을 연결해준다. 위 SYSTEM의 설계처럼 input과 output을 선언해주고 연결해 줄 수 있다. Clock signal은 10ns 주기로 반복되도록 설정하였다.

~SYSTEM()으로 소멸자도 반드시 구현해주어야 leak가 발생하지 않는다.

- Fir.h

```
SC_MODULE(fir) {
    sc_in<bool> clk;
    sc_in<bool> rst;
    sc_in<sc_int<16>> inp;
    sc_out<sc_int<16>> outp;
    //port

    void fir_main();

    SC_CTOR(fir) { //fir execution
        SC_CTHREAD(fir_main, clk.pos()); //positive edge execution
        reset_signal_is(rst, true); //rst == 1, resets
    }
};

const sc_uint<8> coef[5] = { 18,77,107,77,18 }; //not change constant

void fir::fir_main(void) {
    sc_int<16> taps[5] = { 0,0,0,0,0 }; //declare
    outp.write(0);
    wait(); //wait clock edge

    while (true) {
        for (int i = 4; i > 0; i--) {
            taps[i] = taps[i - 1];
        }
        taps[0] = inp.read();
        sc_int<16> val;
        for (int i = 0; i < 5; i++) {
            val += coef[i] * taps[i]; // val is sum of coef * tap
        }
        outp.write(val); //result write
        wait();
    }
}
```

Fir.h는 다음과 같다.

Clk, rst, inp, outp을 다음과 같이 선언해주고 fir_main을 수행한다.

Coef는 변하지 않는 상수로 다음과 같이 선언해주었다.

FIR은 taps 배열을 coef와 곱한 값을 모두 더한 값을 출력해준다.

- Tb.h

```
#pragma once
SC_MODULE(tb) {
    sc_in<bool> clk;
    sc_out<bool> rst;
    sc_in<sc_int<16>> outp;
    sc_out<sc_int<16>> inp;
    //in/out port fir converse

    void source();
    void sink();

    SC_CTOR(tb) { //tb constructor
        SC_CTHREAD(source, clk.pos());
        SC_CTHREAD(sink, clk.pos());
        //positive edge execution
    }
};

void tb::source() {
    inp.write(0);
    rst.write(1); //assert reset
    wait();
    rst.write(0); //deassert reset
    wait();

    sc_int<16> tmp; //fir input

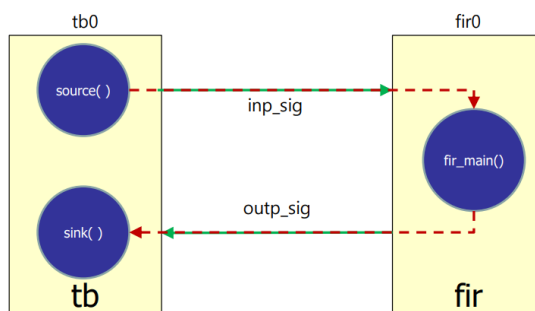
    for (int i = 0; i < 64; i++) {
        if (i > 23 && i < 29) //24~28
            tmp = 256; //input
        else
            tmp = 0;
        inp.write(tmp);
        wait();
    }
}

void tb::sink() {
    sc_int<16> indata;

    for (int i = 0; i < 64; i++) {
        indata = outp.read(); //read
        wait();
        cout << i << " :#t" << indata.to_int() << endl; //print output
    }

    sc_stop();
    //end testbench
}
```

Testbench는 다음과 같다.



source와 sink는 다음과 같이 작동한다.

Source은 inp_sig를 fir에 보내는 역할을 하고 sink은 fir에서 outp_sig를 tb에 보내는 역할을 한다.

Input, output port는 fir port와 반대로 선언해준다. (input, output이 서로 반대)

Source는 input을 보내준다. i가 24~28일때 256을 저장하여 write해준다. 나머지일 경우 0을 저장하여 write

Sink는 fir에서 output을 받아 출력한다. read하여 data를 출력한다.

SYSTEM_wHS module

- Main.cpp

```
#include <systemc.h>
#include "SYSTEM_wHS.h"

SYSTEM* top = NULL; //initialization

int sc_main(int argc, char* argv[]) {
    top = new SYSTEM("top"); //allocation
    sc_start(); //testbench start
    return 0;
}
```

SYSTEM과 main은 같다.

- SYSTEM_wHS.h

```
#include <systemc.h>
#include "fir_whs.h"
#include "tb_whs.h"

SC_MODULE(SYSTEM) {
    tb* tb0;
    fir* fir0;
    //module instaciation

    sc_signal<bool> rst_sig;
    sc_signal<sc_int<16>> inp_sig;
    sc_signal<sc_int<16>> outp_sig;
    //wire
    sc_clock clk_sig; //clock signal

    sc_signal<bool> inp_sig_vld;
    sc_signal<bool> inp_sig_rdy;
    sc_signal<bool> outp_sig_vld;
    sc_signal<bool> outp_sig_rdy;
    //handshaking
}
```

SYSTEM module에 handshake를 추가한 형태이다.

입력에 사용되는 vld, rdy와 출력에 사용되는 vld, rdy가 존재한다.

Vld와 rdy가 모두 1이때 data를 받아 들인다.


```

SC_CTOR(SYSTEM) : clk_sig("clk_sig", 10, SC_NS) { // clock period 10ns
    tb0 = new tb("tb0");
    tb0->clk(clk_sig);
    tb0->rst(rst_sig);
    tb0->inp(inp_sig);
    tb0->outp(outp_sig);
    //add handshaking
    tb0->inp_vld(inp_sig_vld);
    tb0->inp_rdy(inp_sig_rdy);
    tb0->outp_vld(outp_sig_vld);
    tb0->outp_rdy(outp_sig_rdy);
    //testbench connect

    fir0 = new fir("fir0");
    fir0->clk(clk_sig);
    fir0->rst(rst_sig);
    fir0->inp(inp_sig);
    fir0->outp(outp_sig);
    //add handshaking
    fir0->inp_vld(inp_sig_vld);
    fir0->inp_rdy(inp_sig_rdy);
    fir0->outp_vld(outp_sig_vld);
    fir0->outp_rdy(outp_sig_rdy);
    //fir connect
}

//instanciation & port connection

~SYSTEM() {
    delete tb0;
    delete fir0;
}

//destructor

```

마찬가지로 handshake signal도 testbench와 fir에 연결할 수 있다.

- Fir_whs.h

```

SC_MODULE(fir) {
    sc_in<bool> clk;
    sc_in<bool> rst;
    sc_in<sc_int<16>> inp;
    sc_out<sc_int<16>> outp;

    //add handshaking
    sc_in<bool> inp_vld;
    sc_out<bool> inp_rdy;
    sc_out<bool> outp_vld;
    sc_in<bool> outp_rdy;

    //port

    void fir_main();

    SC_CTOR(fir) { //fir execution
        SC_CTHREAD(fir_main, clk.pos()); //positive edge execution
        reset_signal_is(rst, true); //rst = 1, resets
    }
};

```

```

#define DELAY 10

const sc_uint<8> coef[5] = { 18,77,107,77,18 }; //not change constant

void fir::fir_main(void) {
    sc_int<16> taps[5] = { 0,0,0,0,0 };

    for (int i = 0; i < 5; i++) {
        taps[i] = 0;
    }

    //initialize handshake
    inp_rdy.write(0);
    outp_vld.write(0);

    outp.write(0);
    wait(); //wait clock edge

    //FIR filter
    while (true) {
        sc_int<16> in_val;
        sc_int<16> out_val;

        inp_rdy.write(1); //write ready == 1 // ready success

        do {
            wait();
        } while (!inp_vld.read()); //when vld == 0
        in_val = inp.read();
        inp_rdy.write(0);
        //tell "tb" that fir is not ready to take another input yet.

        for (int i = 4; i > 0; i--) {
            taps[i] = taps[i - 1];
        }
        taps[0] = in_val;
        sc_int<16> val;
        for (int i = 0; i < 5; i++) {
            val += coef[i] * taps[i];
        }
        for (int i = 0; i < DELAY; i++) wait(); //wait 10time
        outp_vld.write(1); //change 1
        outp.write(val);
        do {
            wait();
        } while (!outp_rdy.read()); //when rdy == 0

        outp_vld.write(0);
    }
}

```

Fir_wHS는 다음과 같이 구현할 수 있다.

여기서 delay는 10으로 설정하였다.

Do while문으로 vld와 rdy가 1일때만 작동하게 하였다.

또한 for문으로 delay만큼의 wait를 하여 기다리게 하였다.

- Tb_whs.h

```
#pragma once
#include <systemc.h>

SC_MODULE(tb) {
    sc_in<bool> clk;
    sc_out<bool> rst;
    sc_in<sc_int<16>> outp;
    sc_out<sc_int<16>> inp;
    //add handshaking
    sc_in<bool> outp_vld;
    sc_out<bool> outp_rdy;
    sc_out<bool> inp_vld;
    sc_in<bool> inp_rdy;
    //in/out port fir converse

    void source();
    void sink();

    SC_CTOR(tb) { //tb constructor
        SC_CTHREAD(source, clk.pos());
        SC_CTHREAD(sink, clk.pos());
        //positive edge execution
    }
};
```

```
void tb::source() {
    inp.write(0);
    inp_vld.write(0);
    rst.write(1); //assert reset
    wait();
    rst.write(0); //deassert reset
    wait();

    sc_int<16> tmp; //fir input

    for (int i = 0; i < 64; i++) {
        if (i > 23 && i < 29) //24~28
            tmp = 256; //input
        else
            tmp = 0;
        inp_vld.write(1); // write
        inp.write(tmp);

        do {
            wait();
        } while (!inp_rdy.read());
        inp_vld.write(0);
    }
}
```

```
void tb::sink() {
    sc_int<16> indata;

    char output_file[256];
    sprintf(output_file, "./output.data");
    FILE* outfp = fopen(output_file, "wb");
    if (outfp == NULL) {
        printf("Couldn't open output.dat for writing.\n");
        exit(0);
    }

    //initialize port
    outp_rdy.write(0);

    for (int i = 0; i < 64; i++) {
        outp_rdy.write(1);

        do {
            wait();
        } while (!outp_vld.read());
        indata = outp.read();
        outp_rdy.write(0);

        indata = outp.read(); //read

        cout << i << " : 暫" << indata.to_int() << endl; //print output, check
        fprintf(outfp, "%g\n", indata.to_double());
    }

    sc_stop();
    //end testbench
}
```

Testbench를 다음과 같이 구현하였다.

Handshake signal인 val와 rdy를 추가하였다.

SYSTEM module과 비교할 수 있도록 print를 해볼 수 있었다.

3. Comparison of SYSTEM and SYSTEM_wHS

- Show results of both systems

SYSTEM module

0 :	0	28 :	4608
1 :	0	29 :	24320
2 :	0	30 :	-13824
3 :	0	31 :	5888
4 :	0	32 :	10496
5 :	0	33 :	5888
6 :	0	34 :	-13824
7 :	0	35 :	24320
8 :	0	36 :	4608
9 :	0		
10 :	0		
11 :	0		

SYSTEM_wHS module

0 :	0	24 :	4608
1 :	0	25 :	24320
2 :	0	26 :	-13824
3 :	0	27 :	5888
4 :	0	28 :	10496
5 :	0	29 :	5888
6 :	0	30 :	-13824
7 :	0	31 :	24320
8 :	0	32 :	4608
9 :	0		
10 :	0		
11 :	0		

결과는 다음과 같이 출력되었다.

- Compare the results with logical explanations

Tmp에 256으로 값을 주었으므로 차례대로 18, 77, 107, 77, 18이 곱해졌다. 이번 과제에서는 unsignbit로 선언하지 않았으므로 음수의 값도 출력된다.

4. Conclusion

- Summarize what you learn from this project

이번 프로젝트에서 FIR이 무엇이고 구현하는 방법을 알 수 있었다. 또한 handshake signal이 무엇인지 알 수 있었다.

전까지 했던 과제는 흐름을 많이 생각하지 않고도 구현가능 하였지만 이번 과제는 testbench와 module의 흐름을 많이 생각하면서 구현하였다. 예전 디지털논리회로의 기억을 상기하면서 구현할 수 있었다.

처음에는 handshake signal의 사용하는 이유를 알지 못하였다. SYSTEM과 SYSTEM_wHS의 결과가 똑같이 나왔기 때문에 굳이 사용하지 않아도 되지 않을까 했지만, 주기를 10ns보다 훨씬 짧게 주면 결과가 다르게 나왔다. 그 이유는 한 cycle에서 작동하지 않기 때문에 결과 값이 다르게 나온 것을 보았다. 리스크를 최대한 많이 줄여주기 위해서 handshake signal을 사용한다는 것을 알 수 있었다.

이번 프로젝트로 다음 프로젝트인 adder구현하기에 더 수월해졌다.