"내칩제작서비스" 오픈-소스 디자인 킷/연구노트 11: MPW 요건 맞추기

FIR 필터의 파이프라인 PE 설계 (1 부: RTL)

연구과제명	반도체 기술 개발 지원 고경력 전문인력 활용 사업(25JB1710)
연구기간	2025 년 6 월~2026 년 12 월
연구책임자	고상춘
기록자	국일호
확인자	
작성일자	2025 년 7 월 19 일

ETRI 한국전자통신연구원
www.etri.re.kr

"내 칩 제작 서비스" 오픈-소스 디자인 킷/연구노트11: MPW 요건 맞추기

FIR 필터 파이프라인 프로세서의 PE 설계 (1부: RTL)

목차

---

by GoodKook, goodkook@gmail.com

I. 개요

반도체 부품의 가격은 웨이퍼 제조 비용을 칩의 갯수로 나눠 계산된다. 게다가 칩이 작을 수록 전력을 덜 소모할 것은 자명하다. 따라서 기능의 사양이 정해지면 가능한 작은 규모의 칩을 설계하라는 압박을 받는다. 칩의 크기는 설계내용(코어)과 입출력 핀의 수로 결정된다. 코어의 크기에 비해 패드의 숫자가 많을 경우 또는 그 반대의 경우가 될 수 있다. 코어의 크기와 패드의 숫자 사이에 균형을 맞추는 일은 반도체 설계 전에 실무적인 고려 사항이라 하겠다. 반도체 부품의 기획 단계에서 부터 설계자의 참여가 필요한 이유 이기도 하다. 다수의 프로젝트를 한 웨이퍼 상에 올려 제작되는 MPW 방식은 비용 경제적 면이 있다. 하지만 규격화를 위해 프로젝트 당 허용되는 칩의 면적 제한이 있다. 칩과 패키지까지 일괄 제공하는 "내 칩 MPW"는 면적뿐만 아니라 입출력 핀의 수도 제한되어 있다. 만일 허용되는 칩의 조건(또는 기획)이 있다면 이에 맞춰 설계가 변경되어야 한다. 설계의 변경에는 반드시 검증이 뒤따라야 한다. 프로세서의 구조가 변경되었더라도 설계 사양(알고리즘)은 불변(Golden Reference)이다. 개발 과정에서 설계 변경은 비일비재하게 발생한다. 그때마다 검증의 틀(테스트벤치)을 매번 재 제작하려면 그 과정에서 실수가 끼어들 위험성이 매우 높다. FIR 필터 알고리즘을 개발하면서 작성된 검증 틀을 재 사용한다. 넓은 추상성을 가진 C++ 테스트벤치 였기 때문에 검증에 최소한의 수고로 최고의 효율을 얻을 수 있다.

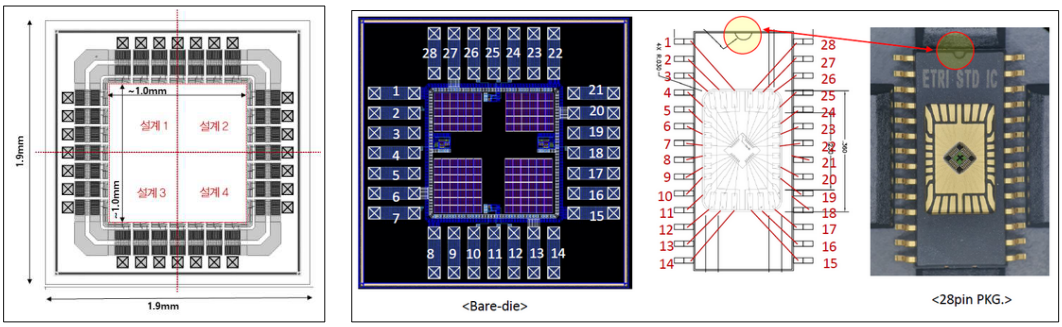
이번 학습은 큰 설계를 칩 제조 요건에 맞춰 분할하고 검증하는 과정을 살펴본다. 설계 구조의 변화로 인해 반도체 부품 사이의 인터페이스 프로토콜(칩 사이의 통신 규약)이 정의될 것이다. 이에 따라 테스트벤치에 최소한의 변경으로 재사용 할 수 있는 방법을 다룬다. SystemC 로 만든 테스트벤치의 효용성을 보게될 것이다.

II. 처리요소(PE)의 입출력 패드 제한(Pad Limit)

II-1. "내 칩 MPW"의 표준 칩 제한

"내 칩 MPW" 서비스의 공고문에 따르면 제작해주는 칩의 면적과 입출력 패드의 제한은 아래와 같다.

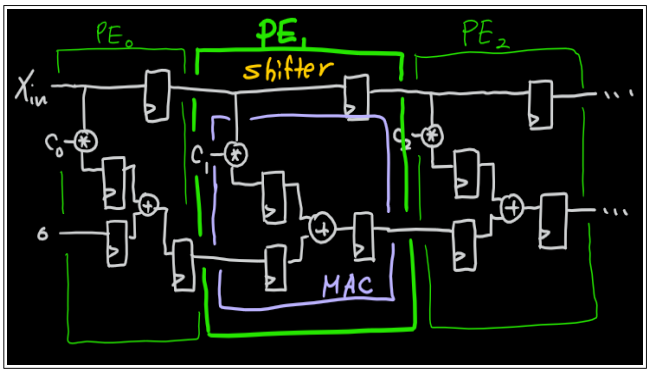
- 칩 크기: 1.9mm x 1.9mm
  * 설계 면적 ~1.5mm x 1.5mm, GPIO 사용시 설계 면적 ~1.0mmx1.0mm
- 패키지 칩을 제공 받을 경우 표준 칩 배치 (PDK 내 "MPW_PAD_28pin") 사용



"내 칩 MPW" 서비스는 공정을 마친 칩을 28핀 SOP 패키지로 제공한다. 약 20여개의 패키지가 제공되는 만큼 다수의 PE들을 조합하여 배열 구조를 구성할 수 있다. 비록 "내 칩 MPW" 서비스의 단일 칩 규모(크기)는 작지만 여러 프로젝트를 조합하면 상당 규모의 알고리즘 구현도 가능할 것이다.

II-2. 설계 면적 제한 (Core Limits)

앞서 설계한 8-탭 FIR 필터는 8개의 PE를 모두 넣은 레이아웃의 크기가 무려 2500x2300um 가량이었다 [연구노트10]. "내 칩 MPW" 서비스를 통해 제작하기에 너무 크다. 자동화 도구의 옵션 조정으로 극복할 수 있을 정도를 넘어선다. 큰 설계를 작은 규모로 분할 하여 패키지 배열로 프로세서를 구현 하기로 한다. FIR 필터는 확장 가능한 파이프라인 병렬처리 구조다. 별도의 외부 제어장치 없이 처리요소(PE, Processing-Element)만을 직렬로 연결하여 쉽게 확장 할 수 있는 시스톨릭 어레이(Systolic Array)다. 전체 프로세서에서 PE 만을 분리하여 "내 칩"으로 제작 하기로 한다.



디지털 FIR 필터 fir8를 구성하는 PE, fir_pe의 레이아웃 면적(크기)을 평가해 보자.

```
$ cd ~/ETRI050_DesignKit/Tutorials/2-7_Lab5_FIR8_rtl_ETRI050
```

Makefile은 fir8 모듈에 하위 모듈로 fir_pe를 함께 합성과 레이아웃을 생성하도록 작성되었다. 하위 모듈 fir_pe을 위한 별도의 Makefile 스크립트를 작성할 필요 없이 명령줄 변수를 사용한다.

```
ifeq ($(TOP), fir8)
      TOP_MODULE=fir8
else ifeq ($(TOP), fir_pe)
      TOP_MODULE=fir_pe
else
      TOP_MODULE=fir8
endif
```

환경 변수 TOP에 fir_pe를 지정하여 합성을 수행한다.

```
$ TOP=fir_pe make synthesize
........
12. Printing statistics.
=== fir_pe ===
   Number of wires:                807
   Number of wire bits:           1030
   Number of public wires:         807
   Number of public wire bits:    1030
   Number of ports:                  6
   Number of port bits:             57
   Number of memories:               0
   Number of memory bits:            0
```

```
Number of processes:              0
Number of cells:                901
  AND2X2                         57
  AOI21X1                        88
  AOI22X1                        17
  BUFX2                          24
  DFFPOSX1                       56
  INVX1                         120
  NAND2X1                       162
  NAND3X1                       136
  NOR2X1                         89
  NOR3X1                          5
  OAI21X1                       117
  OAI22X1                        10
  OR2X2                          20
```

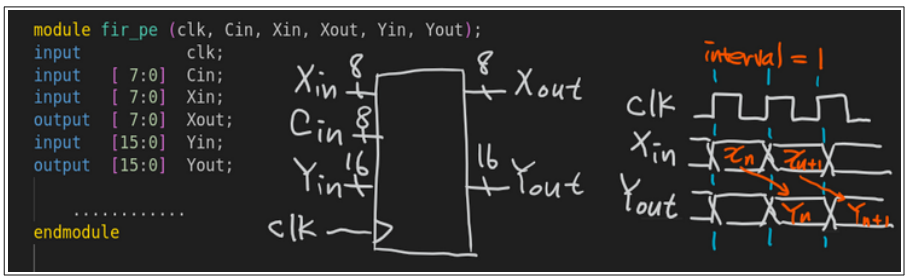
가장 큰 면적을 차지하는 DFFPOSX1 셀이 56개를 차지한다. 이어 배치와 배선 후 레이아웃 크기를 구해본다.

```
$ TOP=fir_pe make route
$ TOP=fir_pe make migrate
$ TOP=fir_pe make size
........
Root cell box:
           width x height     (   llx,   lly   ), (   urx,   ury   )
microns:  936.600 x 913.050   (-9.300, -3.600), ( 927.300,  909.450)
lambda:   3122.00 x 3043.50   (-31.00, -12.00), ( 3091.00,  3031.50)
internal:   6244 x 6087       (   -62, -24    ), (    6182,  6063 )
```

PE 모듈 fir_pe의 레이아웃 크기는 936x913um 가량으로 평가되었다. “내 칩 MPW”의 요건을 충분히 따르는 크기다.

II-3. 입출력 핀 제한 (Pad Limits)

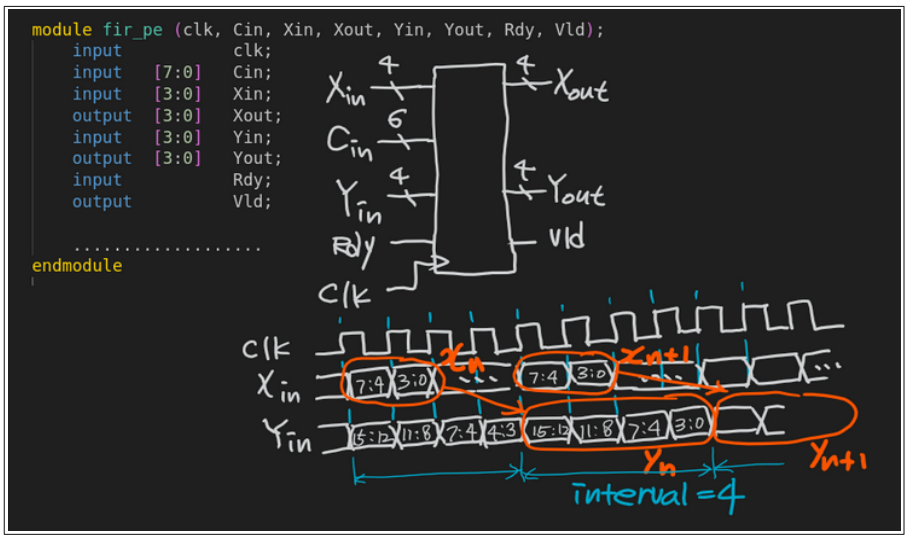
앞서 설계한 FIR 필터 어레이의 PE에 사용된 입출력 포트의 수를 살펴보자. 입력으로 32핀, 출력 24핀 그리고 클럭이 있다. 여기에 전원(VDD)과 접지(GND)까지 합치면 총 59개 패드가 필요하다. “내 칩 MPW”의 28핀 패드 제한을 훌쩍 넘어선다.



처리요소 PE에 필요한 입출력 포트의 종류는 유지한 채 각 포트의 비트 폭을 나눠 패드 제한(pad limit)을 해결하기로 한다. FIR 필터 계수의 범위는 최대 값이 34다. 6비트로 충분히 표현 가능 하다.

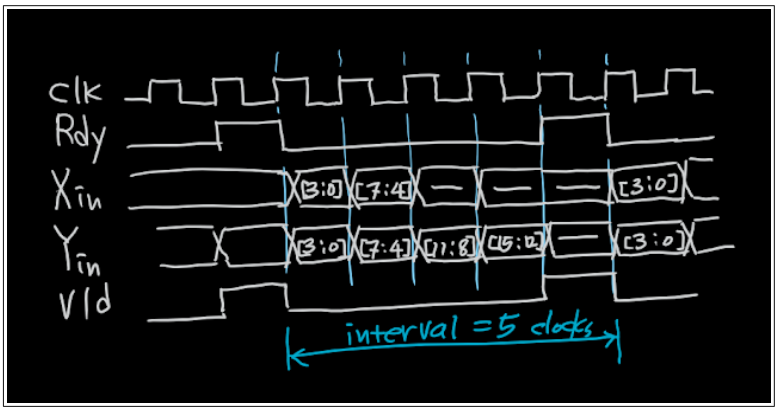
```
localparam [5:0] C[`N_PE_ARRAY] = {4, 12, 25, 34, 34, 25, 12, 4 };
```

입력과 출력을 4비트 씩 분할하였고 필터 계수의 범위를 감안하면 PE에 필요한 입출력 핀의 수는 모두 25개이며 전원과 접지 핀을 포함하더라도 "내 칩 MPW"의 핀 수 요건을 만족한다. 8비트 입출력 Xin과 Xout 을 각각 4비트 씩 분할하면 전송에 2 클럭이 들지만 16비트 Yin과 Yout 의 전송에 4 클럭을 차지한다. 입출력을 분할한 탓에 파이프라인 인터벌(interval)이 1에서 4로 증가할 것으로 예상된다. 4비트 씩 분할한 입출력 포트와 타이밍은 아래와 같다.



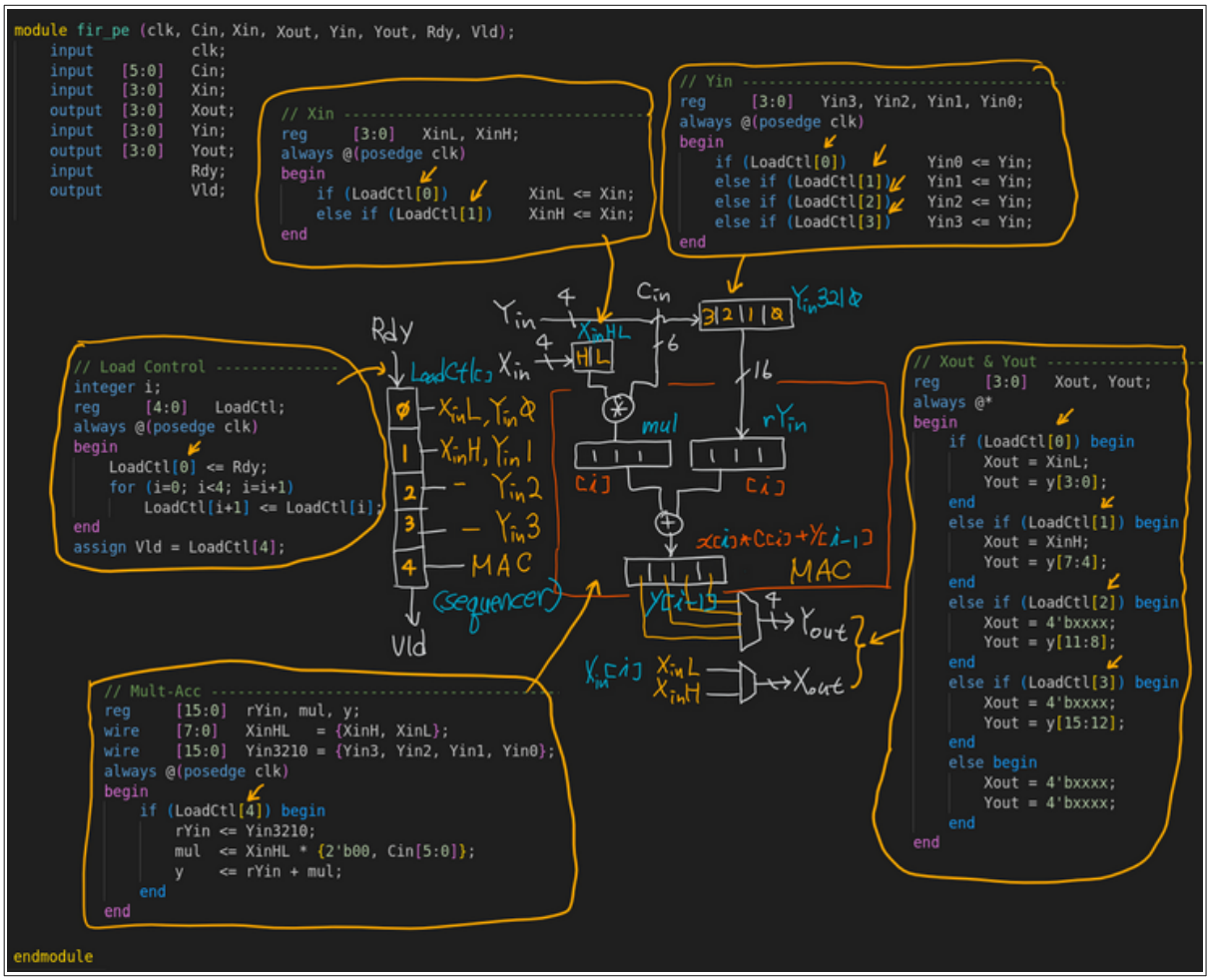
III. 입출력이 분할된 PE의 전송 규약

인터페이스가 변경된 PE의 입출력이 단지 클럭에 동기된 파이프라인이 아니다. 한 데이터를 여러 클럭에 걸쳐 분할 전송하는 방식으로 바뀌었다. 안전한 분할 전송을 위해 첫 4비트 전송이 시작되는 표시가 필요해졌다. 파이프라인 구조 이므로 통신은 항상 일방으로 진행된다. 입출력 인터페이스(In/Out Interface)로 단방향 핸드쉐이크 방식을 채택했다. 전송 규약(protocol)은 아래와 같다. 전송할 데이터의 시작을 이웃 PE에 알리는 Rdy에 이어 연속적으로 Xin 과 Yin가 이어진다. Xin은 8비트 이므로 4개의 전송 클럭 중 첫 2 클럭 동안에서 유효하다. PE들은 파이프라인 배열이므로 Rdy 와 Vld 그리고 Xin과 Xout, Yin 과 Yout의 프로토콜 타이밍은 동일하다.



입출력 포트를 4비트로 분할한 PE의 베릴로그 RTL, fir_pe.v 는 아래와 같다. 인접 PE 사이에 전송되는 데이터 중 MAC에 해당하는 Yin과 Yout 을 4 비트로 나눴다. 인터페이스를 위해 인터벌이 4에서 5로 증가했다. 4비트 단위로

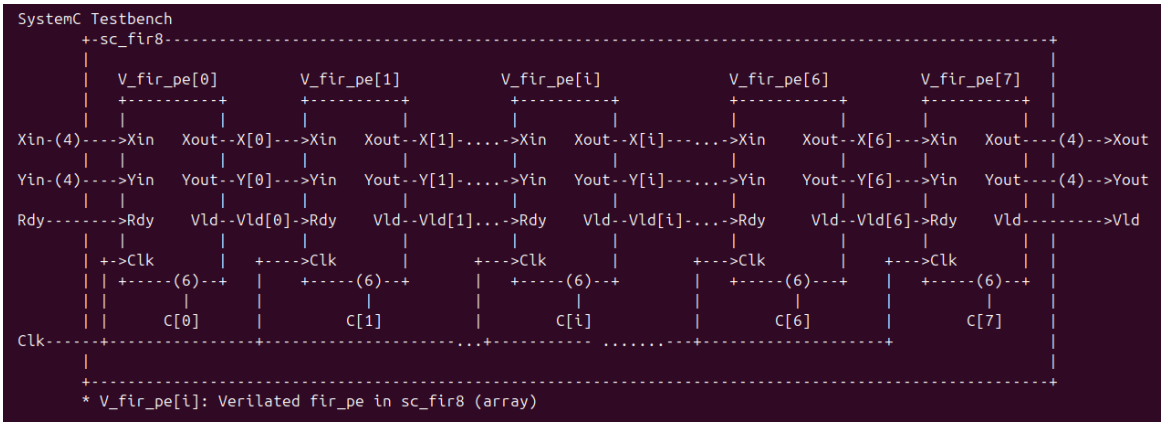
전송받은 Xin과 Yin을 쉬프트 레지스터로 받아 곱셈과 덧셈을 수행한 후 결과를 4비트 씩 출력한다. PE 내에서 별도의 FSM 제어장치(FSM controller) 없이 파이프라인 처리가 이어진다. PE 내부에 명령어 처리는 물론 FSM 제어기를 가지고 있지 않다. 단순한 1 비트 쉬프터가 입출력과 계산 절차를 맞춰 주는 시퀀서(squencer) 역할을 한다.



제작할 칩의 조건(면적과 패드 갯수)에 따라 PE의 내부 구조가 변경되었다. 응용에 따라 변경되는 입출력 방법(인접 모듈과 인터페이스, 프로토콜)은 RTL 설계에 직접적 영향을 준다. 입출력 인터페이스는 알고리즘의 사양(문서 사양과 C++)에서 쉽게 간과되어 선 안된다. 하지만 칩 제작 요건으로 인해 변경이 불가피 하다. FIR 필터 설계 사양 [연구노트9] 중 입출력 인터페이스 변경으로 인해 샘플 당 1클럭에서 5클럭을 사용한다. FIR 필터 하드웨어의 실시간 동작을 위해 5배 빠른 클럭을 적용해야 한다.

IV. 테스트벤치

PE들 사이의 데이터 전송규칙(protocol)이 변경 되었지만 배열 구조는 동일하다. 칩으로 제작할 부분은 PE다. PE를 제외한 FIR 필터의 배열 구조는 SystemC로 기술하기로 한다. 모듈 크래스 sc_fir8은 행위 묘사를 위한 소속 함수와 감응이 없는 구조적 모듈이다. 구성자 내에 Verilator로 변환한 베릴로그 PE의 SystemC 모듈들을 배열로 사례화하고 연결하였다.


SystemC Testbench
sc_fir8
V_fir_pe[0]
V_fir_pe[1]
V_fir_pe[i]
V_fir_pe[6]
V_fir_pe[7]
Xin-(4)
Yin-(4)
Rdy
Clk
Xout
Yout
Vld
C[0]
C[1]
C[i]
C[6]
C[7]
* V_fir_pe[i]: Verilated fir_pe in sc_fir8 (array)


Array untimed model (Algorithm)
Verilated PE module
Instantiate PEs
Structured Array: Binding of PEs
#include <systemc.h>
#include "V_fir_pe.h"
#include "fir8.h" // Filter Tab. Coeff
#define N_PE_ARRAY 8
SC_MODULE(sc_fir8)
sc_in<bool> clk;
sc_in<bool> Rdy;
sc_out<bool> Vld;
sc_in<sc_uint<4> > Xin;
sc_out<sc_uint<4> > Xout;
sc_in<sc_uint<4> > Yin;
sc_out<sc_uint<4> > Yout;
V_fir_pe* u_fir_pe[N_PE_ARRAY];
sc_signal<sc_uint<4> > X[N_PE_ARRAY-1]; // X-input
sc_signal<sc_uint<4> > Y[N_PE_ARRAY-1]; // Accumulated
sc_signal<sc_uint<6> > C[N_PE_ARRAY]; // Filter-Tabs Coeff
sc_signal<bool> Valid[N_PE_ARRAY];
SC_CTOR(sc_fir8): clk("clk")
// Instaltiate PE array
char szPeName[16];
for (int i=0; i<N_PE_ARRAY; i++)
sprintf(szPeName, "u_PE_%d", i);
u_fir_pe[i] = new V_fir_pe(szPeName);
C[i].write(sc_uint<8>(filter_taps[i])); // Filter Coeff.
u_fir_pe[i]->Cin(C[i]);
u_fir_pe[i]->clk(clk);
// 0-th PE
u_fir_pe[0]->Xin(Xin);
u_fir_pe[0]->Xout(X[0]);
u_fir_pe[0]->Yin(Yin);
u_fir_pe[0]->Yout(Y[0]);
u_fir_pe[0]->Rdy(Rdy);
u_fir_pe[0]->Vld(Valid[0]);
// Systolic Array
for (int i=1; i<N_PE_ARRAY-1; i++)
u_fir_pe[i]->Xin(X[i-1]);
u_fir_pe[i]->Xout(X[i]);
u_fir_pe[i]->Yin(Y[i-1]);
u_fir_pe[i]->Yout(Y[i]);
u_fir_pe[i]->Rdy(Valid[i-1]);
u_fir_pe[i]->Vld(Valid[i]);
// Last PE
u_fir_pe[N_PE_ARRAY-1]->Xin(X[N_PE_ARRAY-2]);
u_fir_pe[N_PE_ARRAY-1]->Xout(Xout);
u_fir_pe[N_PE_ARRAY-1]->Yin(Y[N_PE_ARRAY-2]);
u_fir_pe[N_PE_ARRAY-1]->Yout(Yout);
u_fir_pe[N_PE_ARRAY-1]->Rdy(Valid[N_PE_ARRAY-2]);
u_fir_pe[N_PE_ARRAY-1]->Vld(Vld);
};

테스트벤치(sc_fir8_tb.h 와 sc_fir8_tb.cpp)에서 테스트 신호 생성 쓰레드 함수 Test_Gen()은 다음과 같다. 알고리즘 시험을 위해 작성해둔 언-타임드 fir()로 표준 참고 데이터를 생성하는 것은 앞의 경우(인터벌이 1인 파이프라인 배열 구조)와 동일하다. 인터벌이 늘었고 인터페이스 타이밍이 변경되었다. 테스트용 신호를 새롭게 설계한 PE의 전송 프로토콜에 맞춰 생성한다.

```
#define NOISE_RANGE     AMPLITUDE/2.0
#include "cnoise.h"     // noise generator
#include "fir8.h"       // un-timed fir() model

void sc_fir8_tb::Test_Gen()
{
    double      X_in[F_SAMPLE]; // Noise
    uint16_t    yn;
    int         t = 0;

    // Generate tests & reference from C-Model
    srand(time(NULL));
    cnoise_generate_colored_noise_uniform( X_in, F_SAMPLE, 0, NOISE_RANGE );
    // Alpha=0(White Noise), range=+/-NOISE_RANGE

    for (t=0; t<F_SAMPLE; t++)
    {
        x[t] =  ..... ;

        fir(&yn, x[t]); // C-Model FIR Filter

        y[t] = yn;
    }

    Yin.write(0);
    t = 0;

    while(true)
    {
        wait(clk.posedge_event());
        Rdy.write(true);
        Xin.write(0);
        // Least nibble
        wait(clk.posedge_event());
        Rdy.write(false);
        Xin.write(sc_uint<4>(x[t]));
        // Most nibble
        wait(clk.posedge_event());
        Rdy.write(false);
        Xin.write(sc_uint<4>(x[t]>>4));
        // Skip 2-clocks
        wait(clk.posedge_event());
        Rdy.write(false);
        Xin.write(0);
        wait(clk.posedge_event());
        Rdy.write(false);
        Xin.write(0);

        t = ((++t) % F_SAMPLE);
    }
}
```

검증용 쓰레드 함수 Test_ Mon()은 다음과 같다. FIR 배열의 마지막 PE 출력을 클럭 동기화 프로토콜에 맞춰 읽어 표준 값과 비교한다.


```
void sc_fir8_tb::Test_Mon()
{
    int         n = 0;
    uint16_t    yout, E_yout;

    FILE *fp = fopen ( "sc_fir8_tb_out.txt", "w" );

    while(true)
    {
        wait(clk.posedge_event());
        if (Vld.read()==false) continue;
        // Yout[3:0] ----------
        wait(clk.posedge_event());
        yout = (uint16_t)Yout.read();
        // Yout[7:4] ----------
        wait(clk.posedge_event());
        yout |= ((uint16_t)Yout.read())<<4;
        // Yout[12:8] ----------
        wait(clk.posedge_event());
        yout |= ((uint16_t)Yout.read())<<8;
        // Yout[15:13] ----------
        wait(clk.posedge_event());
        yout |= ((uint16_t)Yout.read())<<12;

        if (yout==0)    continue;

        if (y[n]!=yout)
            printf("Error:");
        printf("[%4d] y=%d / Yout=%d\n", n, (uint16_t)y[n], yout);
        n++;
        if (n==F_SAMPLE)
        {
            fflush(fp);
            fclose(fp);
            sc_stop();
        }
    }
}
```

V. 실습. FIR 필터의 PE 시뮬레이션

"내 칩 MPW"의 요건에 맞춰 재 설계된 FIR 필터를 시뮬레이션을 수행해보자. 예제의 소스 파일들은 디자인 킷의 튜토리얼에 포함되어 있다. 실습 예제의 디렉토리 구조는 다음과 같다.

```
$ cd ~/ETRI050_DesignKit/Tutorials/2-9_Lab7_FIR_PE
$ tree
.
```

```
├── /layout
├── /log
├── /simulation
│   ├── fir_pe_TB.gtkw
│   ├── fir_pe_TB.v
│   ├── Makefile
│   ├── V_fir_pe.h      : 베릴로그 RTL fir_pe.v 에서 변환된 SystemC 모델 싸개(wrapper)
│   ├── sc_fir8.h       : 8개 PE로 구성된 FIR 필터 어레이
│   ├── sc_fir8_tb.h    : FIR 필터 어레이 프로세서 테스트벤치(SC 모듈 크래스)
│   ├── sc_fir8_tb.cpp: FIR 필터 어레이 프로세서 테스트벤치(콜백 함수)
│   ├── sc_main.cpp
│   └── /sc_vpi_co-sim
├── /source
│   ├── fir_pe.v : 입출력 인터페이스가 재 설계된 FIR8 프로세서의 PE
│   └── fir_pe.ys
└── /synthesis
```

V-1. 테스트벤치 빌드(Build Testbench)

RTL 베릴로그 PE를 SystemC로 변환한 후 테스트벤치를 컴파일하여 시뮬레이터를 빌드한다. 예제에 Makefile을 준비해 두었다. 예제의 시뮬레이션 디렉토리로 이동,

```
$ cd ~/ETRI050_DesignKit/Tutorials/2-6_Lab4_FIR_PE/simulation
```

타겟 build 를 주고 시뮬레이터 빌드,

```
$ make build
verilator --sc -Wall --trace --top-module fir_pe  --exe --build \
    -CFLAGS -std=c++17 -CFLAGS -g -CFLAGS \
    -I../../../2-5_Lab3_FIR8/c_untimed \
    -CFLAGS -DVCD_TRACE_FIR8 -CFLAGS -DVCD_TRACE_FIR8_TB \
    -LDFLAGS -lm -LDFLAGS -lgsl \
    ../source/fir_pe.v ./sc_main.cpp ./sc_fir8_tb.cpp \
    ../../2-5_Lab3_FIR8/c_untimed/fir8.cpp \
    ../../2-5_Lab3_FIR8/c_untimed/cnoise.cpp
......
```

PE의 인터페이스가 변경 되었으므로 테스트벤치는 타임드 sc_fir8_tb.cpp를 수정한 sc_fir8_tb.cpp 를 사용한다. 검증의 기준은 여전히 언타임드 fir8.cpp 다.

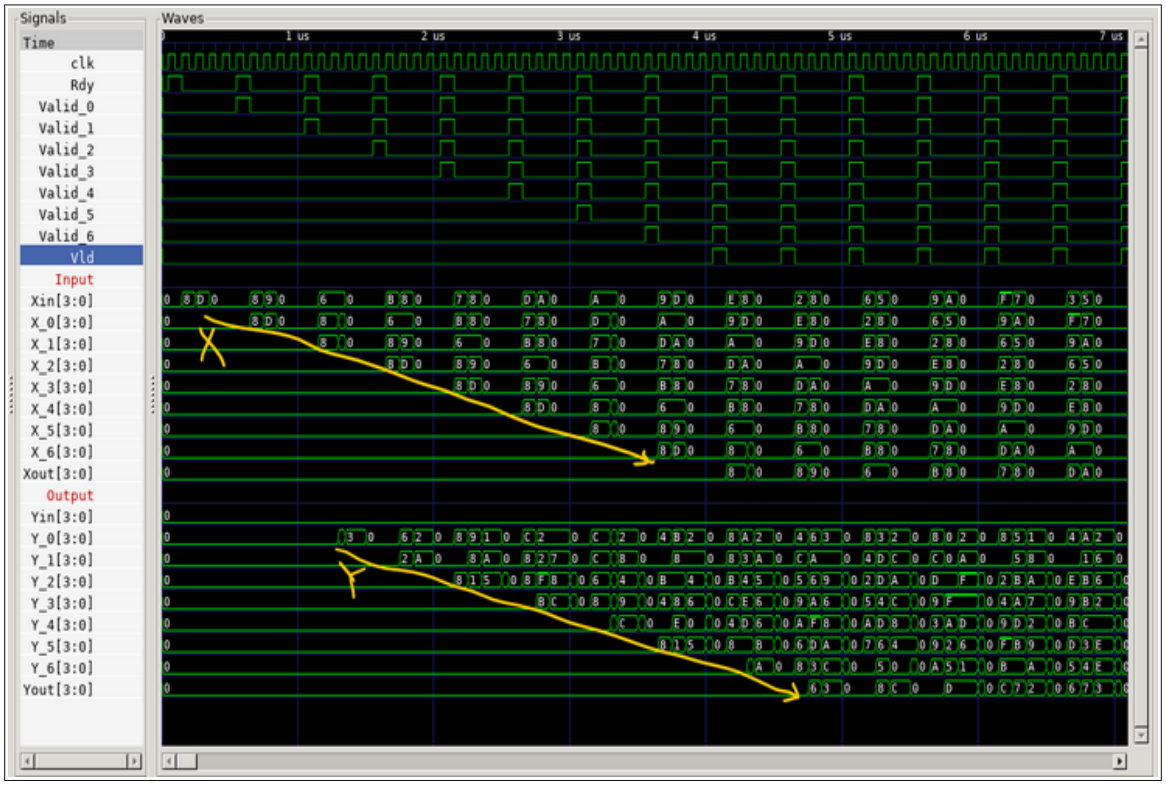
V-2. 시뮬레이터 실행

타겟 run 를 주고 시뮬레이터 실행,

```
$ make run
./obj_dir/Vfir_pe

        SystemC 3.0.2-Accellera --- Jun 13 2025 17:49:45
        Copyright (c) 1996-2025 by all Contributors,
```

```
         ALL RIGHTS RESERVED
[    0] y=664 / Yout=664
[    1] y=2200 / Yout=2200
[    2] y=5518 / Yout=5518
[    3] y=9748 / Yout=9748
 ......
[4797] y=16276 / Yout=16276
[4798] y=15492 / Yout=15492
[4799] y=15477 / Yout=15477

Info: /OSCI/SystemC: Simulation stopped by user.
```

FIR 필터 알고리즘 fir()의 출력과 인터페이스 재 설계 RTL 모델의 출력을 비교하여 오류가 없음을 확인 하였다. 테스트 입력 Xin 과 계수곱과 누산 값 Yin 이 이웃한 PE 사이에 인터벌 5클럭을 두고 전송되는 것을 확인할 수 있다. 특히 Yin은 이전 PE의 출력을 누적해야 하므로 Xin의 전송에 비하여 1 클럭 지연되고 있다.



V-3. 베릴로그 모델의 내부 신호 VCD 추적: PE 동작 확인

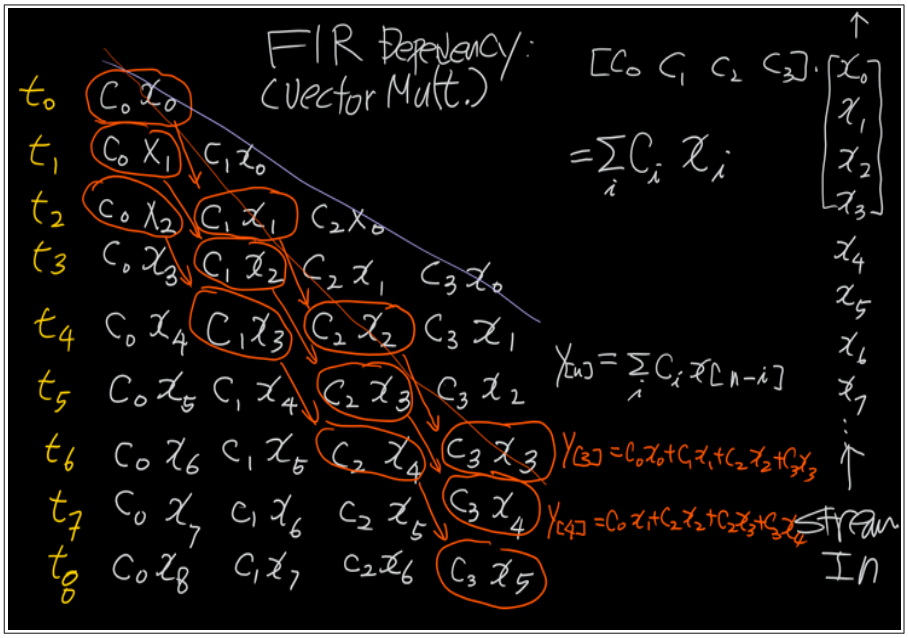
RTL 설계 중 디버깅을 하려면 베릴로그의 내부 신호를 들여다 봐야 하지만 변환된 모델을 SystemC에서 제공하는 방법으로 VCD 기록은 불가능 하다. 하지만 변환 도구 Verilator는 SystemC 모델로 변환하기 전의 베릴로그 모델의 신호를 VCD 덤프 할 수 있는 방법을 제공한다. 테스트벤치 sc_fir8_tb.h 에 추척할 베릴로그 베릴로그 모델을 지정한다. SystemC 테스트 벤치에서 베릴로그 VCD 추적을 지원하는 함수들을 사용하기 위해 verilated_vcd_sc.h 를 들여온다. 테스트벤치 모듈 크래스의 구성자에서 VCD 기능을 켜고 추적할 베릴로그 모듈의 하위 계층 깊이를 지정한다.

```
#include <systemc.h>
#include "sc_fir8.h"

#include <verilated_vcd_sc.h>

SC_MODULE(sc_fir8_tb)
{
    ......
    SC_CTOR(sc_fir8_tb): clk("clk", 100, SC_NS, 0.5, 0.0, SC_NS, false)
    {
        ......
        // WAVE
        fp = sc_create_vcd_trace_file("sc_fir8_tb");
        fp->set_time_unit(100, SC_PS);  // resolution (trace) ps
        sc_trace(fp, clk, "clk");
        sc_trace(fp, Xin,  "Xin");
        sc_trace(fp, Yin,  "Yin");
        sc_trace(fp, Rdy,  "Rdy");

        // Trace Verilated Verilog internals
        Verilated::traceEverOn(true);

        tfp = new VerilatedVcdSc;
        sc_start(SC_ZERO_TIME);
        u_sc_fir8->
            u_fir_pe[N_PE_ARRAY-1]->
                u_Vfir_pe->trace(tfp, 1);  // Trace levels of hierarchy
        tfp->open("Vfir_pe.vcd");
    }

};
```

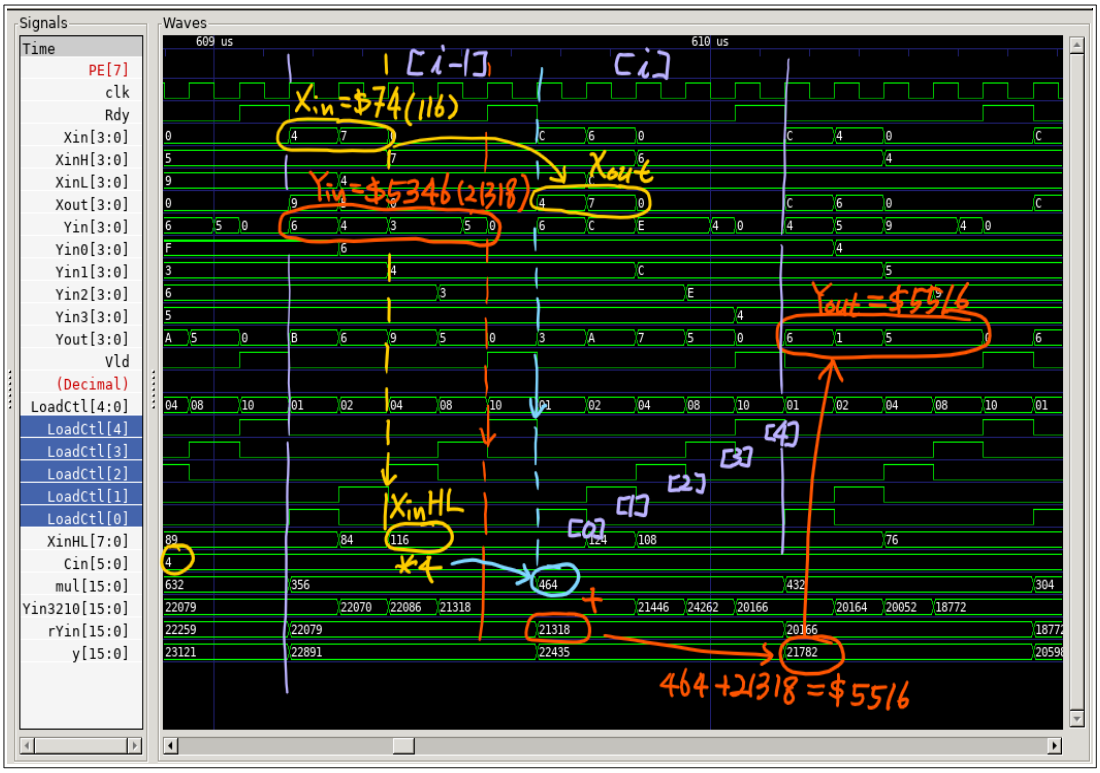
SystemC Trace

Verilator Trace

어레이 구조 FIR 필터를 클럭의 시간순으로 살펴보면 아래와 같다.

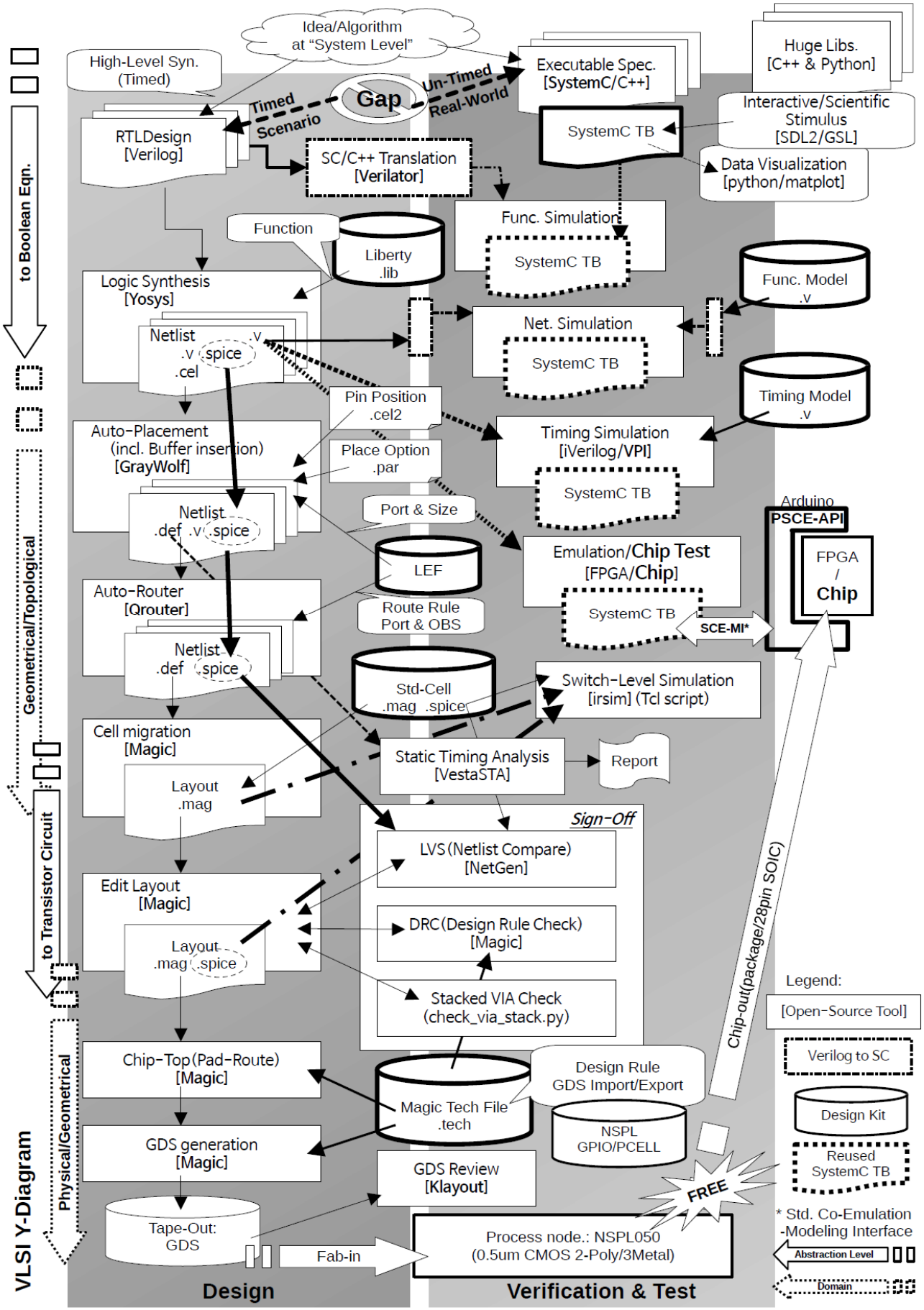


Verilator의 VCD 기록을 보면서 PE 내부의 동작을 살펴보자. 아래 VCD는 FIR 배열에서 마지막 PE의 내부 동작을 보여준다. Xin은 인터벌 마다 이웃 PE에 전송되고 계수 Cin과 곱해진다. Yin 는 이전 PE의 출력과 누산 되어야 하므로 한 클럭 뒤이어 전송되고 있다.



VI. 맺음말

"내 칩 제작 서비스"의 MPW 요건을 만족시키기 위한 FIR 필터의 PE 설계와 검증이 완료 되었다. 실제 칩 제작을 위해 반도체 제조 공장에 제출할 도면(레이아웃)을 만드는 실무 작업이 남았다.


Idea/Algorithm at "System Level"
High-Level Syn. (Timed)
Executable Spec. [SystemC/C++]
Huge Libs. [C++ & Python]
Timed Scenario
Gap
Un-Timed Real-World
RTLDesign [Verilog]
SystemC TB
Interactive/Scientific Stimulus [SDL2/GSL]
SC/C++ Translation [Verilator]
Data Visualization [python/matplot]
to Boolean Eqn.
Function
Liberty .lib
Func. Simulation
SystemC TB
Logic Synthesis [Yosys]
Func. Model .v
Netlist .v .spice .cel
Net. Simulation
SystemC TB
Timing Model .v
Pin Position .cel2
Auto-Placement (incl. Buffer insertion) [GrayWolf]
Timing Simulation [iVerilog/VPI]
SystemC TB
Place Option .par
Netlist .def .v .spice
Port & Size
Arduino PSCE-API
FPGA / Chip
LEF
Emulation/Chip Test [FPGA/Chip]
SystemC TB
SCE-MI*
Auto-Router [Qrouter]
Route Rule Port & OBS
Netlist .def .spice
Std-Cell .mag .spice
Switch-Level Simulation [irsim] (Tcl script)
Geometrical/Topological
Cell migration [Magic]
Static Timing Analysis [VestaSTA]
Report
Layout .mag
Sign-Off
LVS (Netlist Compare) [NetGen]
Edit Layout [Magic]
DRC (Design Rule Check) [Magic]
Layout .mag .spice
Stacked VIA Check (check_via_stack.py)
to Transistor Circuit
Chip-out(package/28pin SOIC)
Legend:
[Open-Source Tool]
Chip-Top (Pad-Route) [Magic]
Design Rule GDS Import/Export
Verilog to SC
Magic Tech File .tech
NSPL GPIO/PCELL
Design Kit
GDS generation [Magic]
Reused SystemC TB
GDS Review [Klayout]
FREE
* Std. Co-Emulation -Modeling Interface
Physical/Geometrical
Tape-Out: GDS
Fab-in
Process node.: NSPL050 (0.5um CMOS 2-Poly/3Metal)
Abstraction Level
Domain
VLSI Y-Diagram
Design
Verification & Test