"내 칩 제작 서비스" 오픈-소스 디자인 킷/C++에서 GDS 까지:

FIR 필터 설계 (2 부: 합성과 배치배선)

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 29 일

ETRI 한국전자통신연구원
www.etri.re.kr

"내 칩 제작 서비스" 오픈-소스 디자인 킷/연구노트10: C++에서 GDS까지

FIR 필터 설계

(2부: RTL 합성과 표준 셀 자동 배치배선)

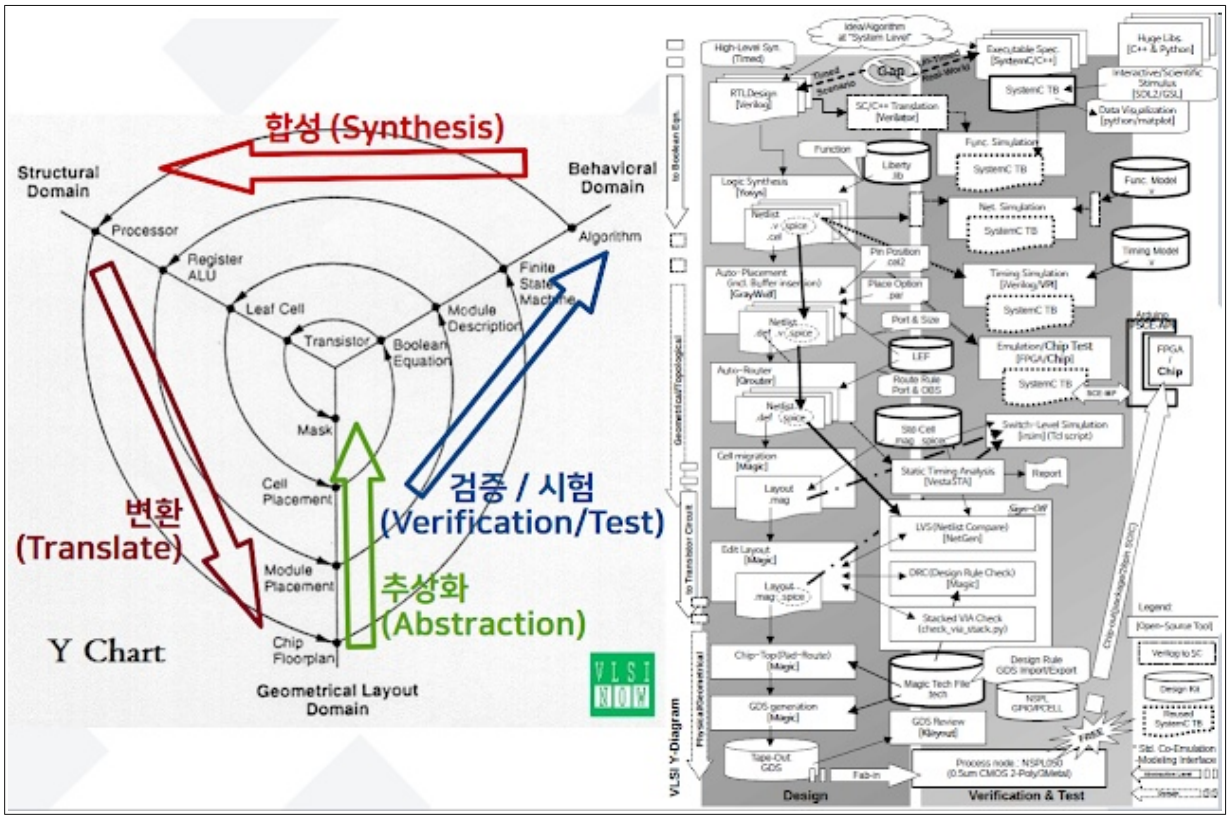
목차:

---

by GoodKook, goodkook@gmail.com

1. 개요

앞서 C++ 언어로 기술한 디지탈 FIR 필터 알고리즘을 시험하고 그로부터 파이프라인 병렬처리 하드웨어 구조를 도출 하여 베릴로그 RTL 까지 설계하고 검증 하는 과정을 다뤘다[기술노트9]. 테스트벤치는 SystemC를 써서 제작하였다. 테스트벤치의 작성에 넓고 높은 추상화 수준의 컴퓨팅 언어를 사용함으로써 언-타임드 알고리즘부터 타임드 RTL까지 재사용이 가능하고 검증의 일관성을 유지할 수 있다. 설계 사양에서 부터 제조 도면을 생성하기까지 추상화 수준의 전환 있지만 그와 무관하게 동일한 테스트벤치를 사용 하여 검증 신뢰성을 높인다.



이번 학습에서는 앞서 알고리즘에서 도출한 RTL 베릴로그를 합성하고 표준 셀 자동 배치와 배선과 레이아웃 생성 과정을 수행한다. 레이아웃은 반도체 제조용 도면으로 설계의 최종 단계에 해당한다. 합성과 자동 배치 배선은 본격적인 설계 자동화 도구들이 동원된다. 자동화 도구는 추상화 수준의 전환에 사용되는 응용 프로그램으로 자체 오류(버그)를 완전히 배제할 수 없다. 게다가 하드웨어 설계의 과정 중 전환되는 추상화 수준의 차이가 매우 크다. 도구를 사용하면서 주어지는 각종 조건(option)에 따라 생성물의 변화 또한 적지 않다. 따라서 자동화 도구의 생성물에 대한 검증이 반드시 수반되어야 한다. 기능의 검증 뿐만 아니라 반도체 공장의 제조 규정을 바르게 준수하고 있는지 검사도 수행되어야 한다. 대표적인 검사로 디자인 룰 체크(DRC, Design Rule Check)가 있으며 이에 더하여 "내 칩 서비스" 공정은 적층 비아 검사(Stacked VIA Check)도 있다. 사인-오프는 제조 공장에 제출하기 전에 검사가 수행되었는지 설계자와 공정 사이의 상호 확인하는 절차다.

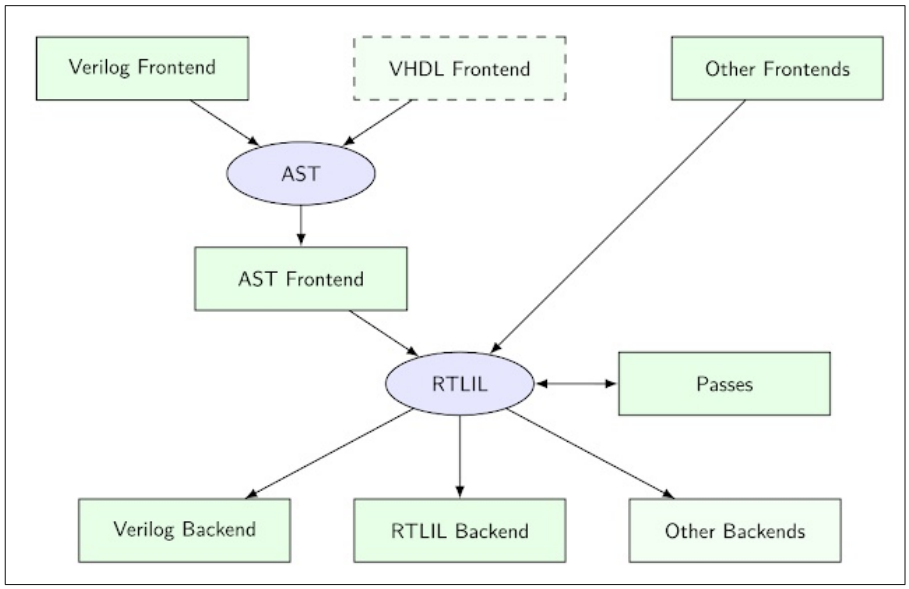
2. 합성

2-1. Yosys 합성기

Yosys는 오픈-소스 RTL 베릴로그 합성기(synthesizer)다. 깃허브 저장소에 다음과 같이 소개하고 있다.

> This is a framework for RTL synthesis tools. It currently has extensive Verilog-2005 support and provides a basic set of synthesis algorithms for various application domains. [출처]
>
> Yosys는 RTL 합성 도구들로 구성한 프레임워크(framework)다. 베릴로그-2005 표준까지 지원하며 합성기로서 다양하게 활용되고 있다.

Yosys 합성기는 베릴로그 언어의 구문을 해석하여 논리식을 만들고 플립플롭을 추론(inference)해낸다. 논리식은 자체 논리 연산자를 사용하여 표현되며, 논리식의 최적화와 목표 공정에서 제공하는 표준 셀로 사상(technology mapping, '테크-매핑')한다. 최종적으로 자동 배치와 배선이 가능한 네트리스트를 생성한다. 위의 소개문에서 Yosys 합성기는 자신을 '소프트웨어 틀(framework)'이라 하고 있다. 언어 구문 해석기(AST), 논리식 표현 방법(RTLIL), 최적화(ABC)와 목표 공정 이전(tech-mapping)에 사용되는 도구들을 한 목적에 맞게 적용 시켜놓았기 때문이다. 사용된 도구들은 Yosys 내부의 것도 있지만 API가 제공되는 오픈-소스 도구들도 많이 활용 되었다. 아래 그림은 Yosys의 프론트엔드(front end) 구성을 간략히 보여준다.



[출처]Yosys Reference Manual[pdf]/Implementation Overview

[1] AST: Abstract Syntax Tree [Wiki]

[2] RTLIL: RTL Intermediate Language [link]

[3] Yosys 빌드에 필요한 패키지 설치,

```
$ sudo apt-get install \
    build-essential clang lld bison flex \
```

```
libreadline-dev gawk tcl-dev libffi-dev git \
graphviz xdot pkg-config python3 libboost-system-dev \
libboost-python-dev libboost-filesystem-dev zlib1g-dev
```

[주] "프레임워크"의 의미는 무엇인가요?

2-2. 합성기의 프론트 엔드(구문 해석 및 논리회로 추론)

합성기의 프론트엔드(front end)는 베릴로그를 읽어 논리회로의 네트리스트(게이트와 플립플롭)로 표현해 내기(추론, inference)까지 과정을 말한다. 앞서 작성한 FIR 필터의 RTL 베릴로그를 가지고 합성기 Yosys의 프론트엔드를 살펴보기로 한다.

FIR 필터의 RTL베릴로그 디렉토리로 이동,

```
$ cd ~/ETRI050_DesignKit/devel/Tutorials/2-5_Lab3_FIR8/rtl_verilog
```

Yosys 합성기 실행 후 베릴로그 파일 읽기,

```
$ yosys

yosys> read_verilog fir8.v fir_pe.v
/----------------------------------------------------------------------\
|  yosys -- Yosys Open SYnthesis Suite                                 |
|  Copyright (C) 2012 - 2025  Claire Xenia Wolf <claire@yosyshq.com>|
|  type "license" to see terms                                         |
\----------------------------------------------------------------------/
Yosys 0.54+1 (git sha1 67f8de54d, clang++ 18.1.3 -fPIC -O3)

1. Executing Verilog-2005 frontend: fir8.v

Parsing SystemVerilog input from `fir8.v' to AST representation.
fir8.v:25: ERROR: syntax error, unexpected '[', expecting ',' or '=' or ';'
```

문법 오류가 있다는 보고다. 베릴로그 파일 fir8.v에서 오류가 난 줄을 보면,

```
localparam [7:0] C[`N_PE_ARRAY] = {4, 12, 25, 34, 34, 25, 12, 4 };
```

Yosys 합성기 프론트 엔드의 구문 해석기는 배열형 지역상수(localparam)를 지원하지 않는다. 아래와 같이 바꿔주자.

```
wire [7:0] C[`N_PE_ARRAY];
assign C[0] = 8'd4;
assign C[1] = 8'd12;
assign C[2] = 8'd25;
assign C[3] = 8'd34;
assign C[4] = 8'd34;
assign C[5] = 8'd25;
assign C[6] = 8'd12;
assign C[7] = 8'd4;
```

다시 베릴로그 파일을 읽는다. 구문을 해석한 후 문법 오류가 없으면 AST로 변환하고 RTLIL 형식을 생성한다.

```
yosys> read_verilog -sv fir8.v fir_pe.v
1. Executing Verilog-2005 frontend: fir8.v
    Parsing SystemVerilog input from `fir8.v' to AST representation.
    Generating RTLIL representation for module `\fir8'.
    Successfully finished Verilog frontend.
2. Executing Verilog-2005 frontend: fir_pe.v
    Parsing SystemVerilog input from `fir_pe.v' to AST representation.
    Generating RTLIL representation for module `\fir_pe'.
    Successfully finished Verilog frontend.
```

Yosys의 proc 명령으로 베릴로그 순차구문구역(always 블록)에서 코딩 스타일을 분석하여 조합회로와 플립플롭을 추론해 낸다.

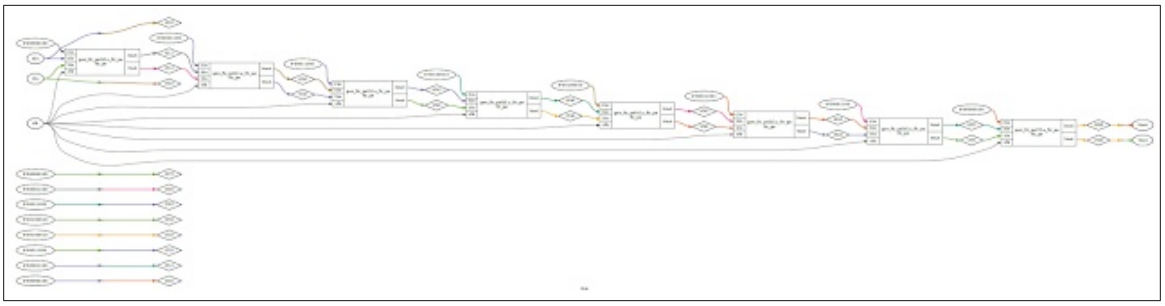
```
yosys> proc
3. Executing PROC pass (convert processes to netlists).
......
3.8. Executing PROC_DLATCH pass (convert process syncs to latches).
    No latch inferred for signal `\fir8.\X[0]'from process
                                          `\fir8.$proc$fir8.v:0$2'.
    No latch inferred for signal `\fir8.\Y[0]' from process
                                          `\fir8.$proc$fir8.v:0$2'.
    No latch inferred for signal `\fir8.\C[0]' from process
                                          `\fir8.$proc$fir8.v:0$2'.
      ......
3.9. Executing PROC_DFF pass (convert process syncs to FFs).
    Creating register for signal `\fir_pe.\Xout' using process
                                       `\fir_pe.$proc$fir_pe.v:34$6'.
        created $dff cell `$procdff$8' with positive edge clock.
    Creating register for signal `\fir_pe.\Yout' using process
                                       `\fir_pe.$proc$fir_pe.v:34$6'.
        created $dff cell `$procdff$9' with positive edge clock.
```

순차구문 구역(베릴로그의 always 블럭)내에 기술된 if 문과 switch 문은 조합회로 MUX 또는 래치(latch)가 될 수 있다. 할당문들을 해석한 결과, 래치는 없으며 상승 엣지 클럭을 갖는 플립플롭들을 추론해 냈다. 이어 최적화를 수행한다.

```
yosys> opt
4. Executing OPT pass (performing simple optimizations).
...
4.7. Executing OPT_CLEAN pass (remove unused cells and wires).
    Finding unused cells or wires in module \fir_pe..
    Finding unused cells or wires in module \fir8..
    Removed 0 unused cells and 14 unused wires.
...
4.16. Finished OPT passes. (There is nothing left to do.)
```

간단한 회로여서 크게 논리 최적화 할 여지는 없었고 몇 개의 네트들을 합쳤다. 이로써 베릴로그 구문으로 부터 네트리스트로 변환하는 과정을 마쳤다. 그림으로 합성해낸 하드웨어 구조를 보자.

```
yosys> show -colors 5 fir8
```

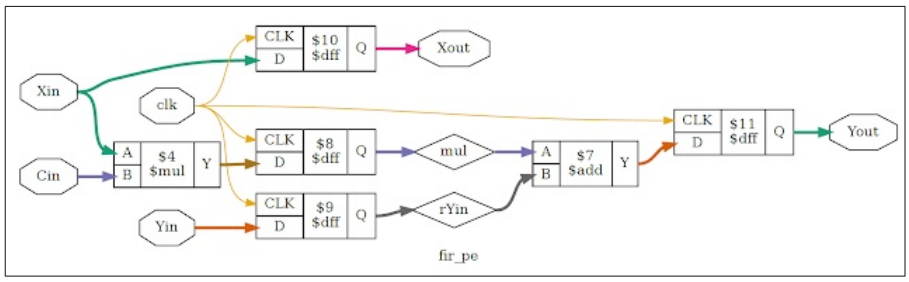


하위 모듈로 8개의 fir_pe가 직렬로 연결된 구조를 보여 준다. Yosys는 변환된 구문을 GraphViz 유틸리티를 활용하여 회로도(schematics)를 그래픽으로 볼 수 있도록 XDOT 형태로 네트리스트를 생성한다. 위의 그림이 전자회로처럼 보이진 않는 것은 표현에 동원된 심볼들의 모양이 낯설 뿐이지 합성결과를 분석하기에 충분히 직관적이다. 이번에는 fir_pe의 내부를 보기로 한다.

[주] Graphviz(Graph Visualization Software)는 DOT 언어 스크립트로 지정된 그래프 그리기를 위해 AT&T 랩스 리서치가 시작한 오픈 소스 도구 패키지이다[WiKi].

fir8을 구성하는 하위 모듈 fir_pe의 회로도를 보면 다음과 같다.

```
yosys> show -colors 5 fir_pe
```



베릴로그의 설계 의도 대로 정확히 합성된 것을 알 수 있다. 베릴로그 언어에서 대수 표현에 사용되었던 산술연산자(*, -, + 등)는 $mul, $add 등 높은 수준의 조합회로 연산기로 대표되었다.

2-3. 합성기의 백엔드(논리회로 추론 및 최적화)

오픈-소스 Yosys 합성기의 명령들을 살펴보면 논리식 최적화, 순차회로(플립플롭)최적화, 유한 상태 머신(FSM) 추론 및 최적화, 메모리(ROM 과 RAM)추론, 테크놀러지 매핑등 상당히 정교한 논리회로 합성을 수행하는 것을 알 수 있다. Yosys 합성기의 다양한 명령들을 예제와 함께 설명한 아래 링크를 참고한다.

Yosys: Synthesis Starter [link]

[주] Yosys: Manual[pdf] Presentation 2022 [pdf] Command Reference [link]

여러 단계의 합성 명령을 묶어 스크립트 synth 로 제공한다. 합성 스크립트의 내용은 아래와 같다.

Yosys: synth - Generic Synthesis Script [link]

프론트엔드를 마친 FIR 필터 베릴로그에 일반 합성 스크립트 synth 를 적용해 보자.

```
yosys> synth -top fir8
```

합성 과정이 정교한 만큼 매우 긴 로그를 출력한다. FIR 필터 설계가 단순하여 처리에 특이 사항은 없지만 합성이 어떤 과정을 거치는지 중요 부분만 간략히 살펴본다.

읽어들인 베릴로그 파일의 계층 구조를 확인하여 빠진 하위 모듈이 있는지 검사한다.

```
5. Executing SYNTH pass.
5.1. Executing HIERARCHY pass (managing design hierarchy).
...
```

병렬 할당 assign 문은 그대로 논리식 이므로 따로 추론할 필요는 없지만 베릴로그 always 순차구문 영역은 코딩 스타일[연구노트3]에 따라 플립플롭, 래치 또는 멀티플렉서로 추론된다.

```
5.2. Executing PROC pass (convert processes to netlists).
      ......
5.2.9. Executing PROC_DFF pass (convert process syncs to FFs).
    Creating register for signal `\fir_pe.\Xout' using process
                                        `\fir_pe.$proc$fir_pe.v:34$6'.
      created $dff cell `$procdff$49' with positive edge clock.
    Creating register for signal `\fir_pe.\Yout' using process
                                        `\fir_pe.$proc$fir_pe.v:34$6'.
      created $dff cell `$procdff$50' with positive edge clock.
    Creating register for signal `\fir_pe.\rYin' using process
                                        `\fir_pe.$proc$fir_pe.v:29$5'.
      created $dff cell `$procdff$51' with positive edge clock.
    Creating register for signal `\fir_pe.\mul' using process
                                        `\fir_pe.$proc$fir_pe.v:24$3'.
      created $dff cell `$procdff$52' with positive edge clock.
      ......
5.6.3. Executing OPT_MUXTREE pass (detect dead branches in mux trees).
    Creating decoders for process `\fir8.$proc$fir8.v:0$48'.
    Creating decoders for process `\fir_pe.$proc$fir_pe.v:34$6'.
    Creating decoders for process `\fir_pe.$proc$fir_pe.v:29$5'.
    Creating decoders for process `\fir_pe.$proc$fir_pe.v:24$3'.
      ......
5.7. Executing FSM pass (extract and optimize FSM).
5.9. Executing WREDUCE pass (reducing word size of cells).
5.12. Executing ALUMACC pass (create $alu and $macc cells).
5.13. Executing SHARE pass (SAT-based resource sharing).
5.15. Executing MEMORY pass.
```

설계 내에 FSM과 메모리, 곱셈-누산기(MACC), 공유 자원(resource sharing)등을 찾아 합성을 시도한다. 예제 FIR 필터 fir8에는 이와 관련하여 추론할 사항이 없었다. 합성이 완료되면 Yosys의 내부 논리 소자들을 목표 공정에서 가능한 표준 셀로 매핑 한다. 테크-매핑 도구는 ABC 다.

ABC: A System for Sequential Synthesis and Verification, Berkeley [link]

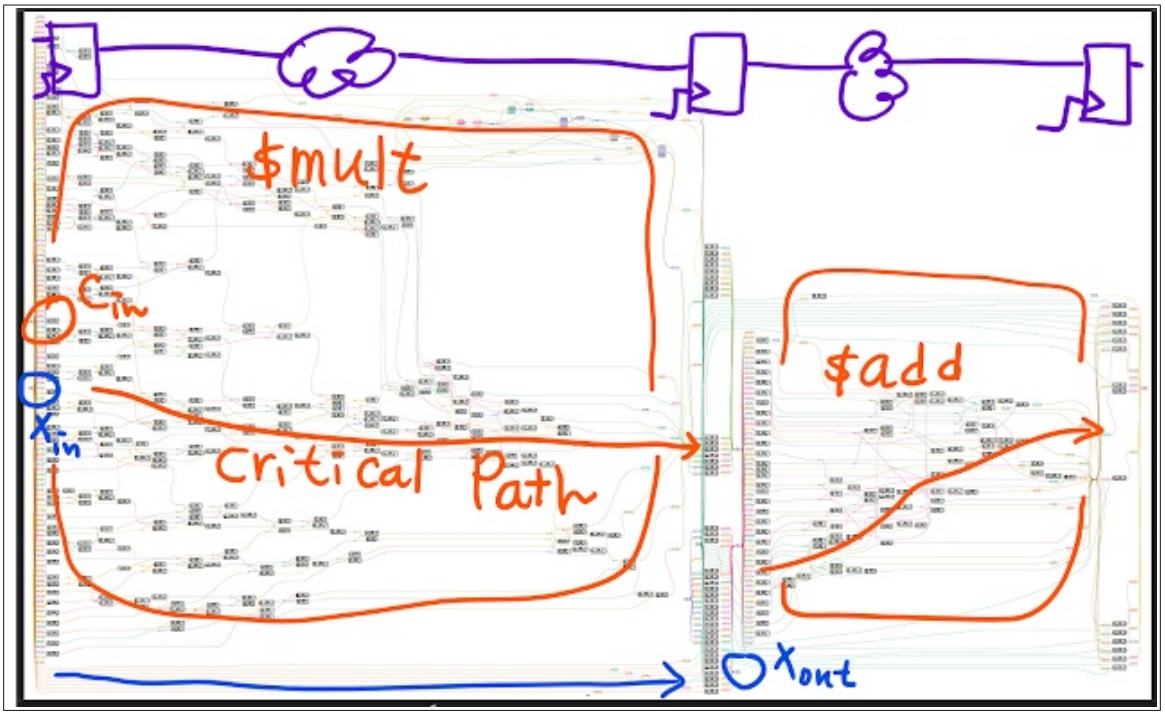
```
5.20. Executing TECHMAP pass (map to technology primitives).
5.22. Executing ABC pass (technology mapping using ABC).
...
5.22.2.2. Re-integrating ABC results.
ABC RESULTS:               AND cells:       60
ABC RESULTS:            ANDNOT cells:      117
ABC RESULTS:              NAND cells:       36
ABC RESULTS:               NOR cells:       38
ABC RESULTS:               NOT cells:        6
ABC RESULTS:                OR cells:       43
ABC RESULTS:             ORNOT cells:        9
ABC RESULTS:              XNOR cells:       38
ABC RESULTS:               XOR cells:      104
ABC RESULTS:        internal signals:      413
ABC RESULTS:           input signals:       48
ABC RESULTS:          output signals:       32
Removing temp directory.
```

목표 공정을 지정하지 않았으므로 내부 논리소자로 매핑 되었다. 합성에 사용된 논리소자들을 보면 다음과 같다.

```
5.25. Printing statistics.
=== fir8 ===
     fir_pe                          8
=== fir_pe ===
   Number of wires:                436
   Number of wire bits:            652
   Number of public wires:           8
   Number of public wire bits:      89
   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:                507
     $_ANDNOT_                     117
     $_AND_                         60
     $_DFF_P_                       56
     $_NAND_                        36
     $_NOR_                         38
     $_NOT_                          6
     $_ORNOT_                        9
     $_OR_                          43
     $_XNOR_                        38
     $_XOR_                        104
```

프론트 엔드에서 $mul, $add 등 높은 수준에서 표현되었던 산술 연산기들이 낮은 수준의 논리 소자들로 변환된 것을 볼 수 있다.

```
yosys> show -colors 3 fir_pe
```



합성된 회로도(그림)를 보며 논리회로를 구체적으로 파악하기는 불가능 하다. 하지만 베릴로그 설계가 RTL의 관점에서 구조적으로 바른지, 의도치 않았던 래치는 없는지, 가장 긴 지연 경로(longest path)는 어느 지점인지 파악하고 대책을 마련할 때 효과적인 분석 자료다. 위의 경우 곱셈기 $mul 이 만들어낸 경로가 가장 길다. 덧셈기 $add 에 비해 균형도 맞지 않는다. 설계 사양에 속도 조건(클럭 주기)이 있다면 이를 맞추기 위해 곱셈기 $mul를 합성기가 제공한 기본 회로를 쓰지 않고 별도의 빠른 곱셈기 또는 다단 곱셈기(multi-stage multiplier)의 설계가 필요해 보인다.

2-4. 디자인 킷 표준 셀로 매핑(technology mapping)

"내 칩 제작 서비스" 표준 셀 디자인 킷에서 제공하는 표준 셀 라이브러리로 매핑 해보자. 표준 셀의 목록은 다음과 같다.

디자인 킷의 표준-셀 종류

표준-셀	기능(리버티 형식[29])
AND2X1, AND2X2	(A B)
AOI21X1	(!((A B)+C))
AOI22X1	(!((A B)+(C D)))
BUFX2,BUFX4	A
CLKBUF1,CLKBUF2	A
DFFNEGX1	ff() { …… }
DFFPOSX1	ff() { …… }

DFFSR	ff (P0002,P0003) {
	next_state : "D";
	clocked_on : "CLK";
	clear : "(!R)";
	preset : "(!S)";
	clear_preset_var1 : L;
	}
	
	"P0002"

INVX1,INVX2,INVX4	(!A)
MUX2X1	(!((S A) + (!S B)))
NAND2X1,NAND2X1	(!(A B)),(!((A B) C))
NOR2X1, NOR2X1	(!(A+B)), (!((A+B)+C))
OAI21X1,OAI22X1	(!((A+B) C)),(!((A+B) (C+D)))
OR2X1, OR2X2	(A+B)

'리버티(Liverty Format)'는 디자인 킷에서 제공하는 표준 셀들의 기능과 특성들을 합성기에게 알려주는 셀 라이브러리 파일 형식으로 Synopsys 사에서 제정되었다[Liberty User Guide]. "내 칩 제작 서비스" 표준 셀 디자인 킷의 리버티 파일과 디자인 파일을 읽어 합성을 수행한다.

```
yosys> read_liberty -lib -ignore_miss_dir -setattr blackbox  \
           /usr/local/share/qflow/tech/etri050/etri05_stdcells.lib
1. Executing Liberty frontend:
    /usr/local/share/qflow/tech/etri050/etri05_stdcells.lib
    Imported 39 cell types from liberty file.
```

디자인 킷은 39종의 표준 셀을 가지고 있다. 이어 디자인 파일을 읽는다.

```
yosys> read_verilog -sv fir8.v fir_pe.v
2. Executing Verilog-2005 frontend: fir8.v
    Parsing SystemVerilog input from `fir8.v' to AST representation.
    Generating RTLIL representation for module `\fir8'.
    Successfully finished Verilog frontend.
3. Executing Verilog-2005 frontend: fir_pe.v
    Parsing SystemVerilog input from `fir_pe.v' to AST representation.
    Generating RTLIL representation for module `\fir_pe'.
    Successfully finished Verilog frontend.
```

두 디자인 파일은 오류 없이 구문 분석이 완료되어 AST로 표현 되었고 합성을 위해 RTLIL로 변환 되었다. 최상위 모듈을 fir8 로 지정하고 합성을 수행한다.

```
yosys> synth -top fir8
```

설계 fir8을 합성한 회로에는 아래와 같은 셀들이 사용되었다.

```
=== design hierarchy ===
   fir8                            1
     fir_pe                        8
   Number of wires:              3583
   Number of wire bits:          5489
   Number of public wires:         95
   Number of public wire bits:   1041
   Number of ports:                53
   Number of port bits:           505
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:              4128
     $_ANDNOT_                    952
     $_AND_                       456
     $_DFF_P_                     448
     $_NAND_                      344
     $_NOR_                       304
     $_NOT_                        40
     $_ORNOT_                      24
     $_OR_                        384
     $_XNOR_                      384
     $_XOR_                       792
```

코딩 스타일에 따라 클럭의 엣지 방향, 동기 또는 비동기 셋과 리셋등 다양한 플립플롭이 있다. 디자인 킷에서 제공하는 플립플롭의 종류는 많지 않다. 동원 가능한 플립플롭을 알려 주어 적절하게 매핑 되게 한다.

```
yosys> dfflibmap -liberty \
           /usr/local/share/qflow/tech/etri050/etri05_stdcells.lib

5. Executing DFFLIBMAP pass
      (mapping DFF cells to sequential cells from liberty file).
    cell DFFNEGX1 (noninv, pins=3, area=864.00) is a direct match
        for cell type $_DFF_N_.
    cell DFFPOSX1 (noninv, pins=3, area=864.00) is a direct match
        for cell type $_DFF_P_.
    cell DFFSR (noninv, pins=5, area=1584.00) is a direct match
        for cell type $_DFFSR_PNN_.
    final dff cell mappings:
        \DFFNEGX1 _DFF_N_ (.CLK( C), .D( D), .Q( Q));
        \DFFPOSX1 _DFF_P_ (.CLK( C), .D( D), .Q( Q));
        unmapped dff cell: $_DFF_NN0_
        unmapped dff cell: $_DFF_NN1_
          ......
        unmapped dff cell: $_DFFSR_NPN_
        unmapped dff cell: $_DFFSR_NPP_
```

```
    \DFFSR _DFFSR_PNN_ (.CLK( C), .D( D), .Q( Q), .R( R), .S( S));
    unmapped dff cell: $_DFFSR_PNP_
5.1. Executing DFFLEGALIZE pass
        (convert FFs to types supported by the target).
    Mapping DFF cells in module `\fir8':
    Mapping DFF cells in module `\fir_pe':
      mapped 56 $_DFF_P_ cells to \DFFPOSX1 cells.
```

모듈 fir_pe 의 코딩 스타일에서 추론 된 플립플롭 스타일은 $_DFF_P_이 등가인 표준셀 DFFPOSX1로 매핑되었다. 플롭플롭에 이어 조합 논리 회로를 매핑한다.

```
yosys> abc -exe /usr/local/share/qflow/bin/yosys-abc \
   -liberty /usr/local/share/qflow/tech/etri050/etri05_stdcells.lib \
   -script +strash;scorr;ifraig;retime,{D};strash;dch,-f;map,-M,1,{D}
......
7.2.2. Re-integrating ABC results.
ABC RESULTS:            AND2X2 cells:       57
ABC RESULTS:           AOI21X1 cells:       88
ABC RESULTS:           AOI22X1 cells:       17
ABC RESULTS:             INVX1 cells:      120
ABC RESULTS:           NAND2X1 cells:      162
ABC RESULTS:           NAND3X1 cells:      136
ABC RESULTS:            NOR2X1 cells:       89
ABC RESULTS:            NOR3X1 cells:        5
ABC RESULTS:           OAI21X1 cells:      117
ABC RESULTS:           OAI22X1 cells:       10
ABC RESULTS:             OR2X2 cells:       20
ABC RESULTS:        internal signals:      428
ABC RESULTS:           input signals:       48
ABC RESULTS:          output signals:       32
```

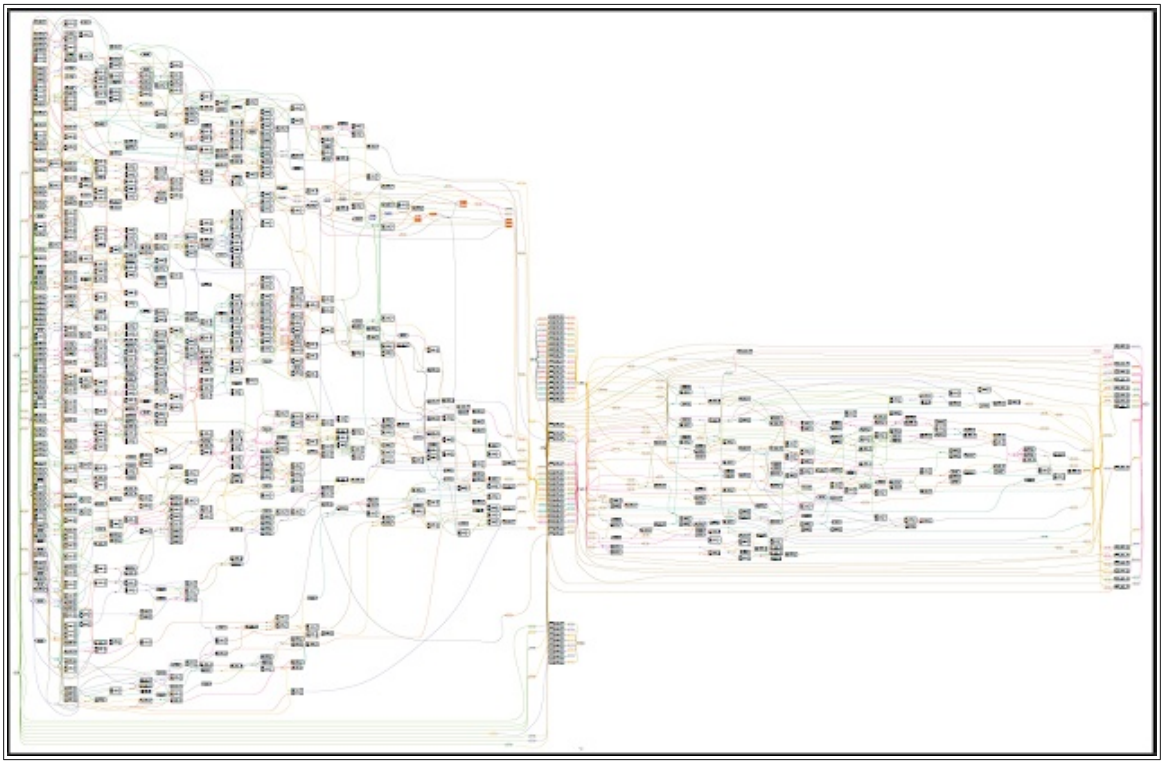
Yosys는 내부적으로 기초 논리 소자들 만 가지고 있다. 너무 작은 단위의 논리 소자들을 사용할 경우 배치가 분산되어 배선이 길어질 수 있다. 자주 사용하는 논리식을 묶어 좀 더 복합적인 기능을 하는 셀을 제공한다. 예를 들어 AOI21X1 셀은 AND와 OR 에 INV를 한데 모은 것으로 구동력이 1인 셀을 의미한다. 디자인 킷에서 제공되는 복합 셀로 매핑한 결과를 보면 다음과 같다.

```
yosys> stat
9. Printing statistics.
=== design hierarchy ===
   fir8                            1
     fir_pe                        8
   Number of wires:            10535
   Number of wire bits:        12441
   Number of public wires:        95
   Number of public wire bits:  1041
   Number of ports:               53
   Number of port bits:          505
   Number of memories:             0
   Number of processes:            0
   Number of cells:             7016
     AND2X2                      456
     AOI21X1                     704
     AOI22X1                     136
     DFFPOSX1                    448
```

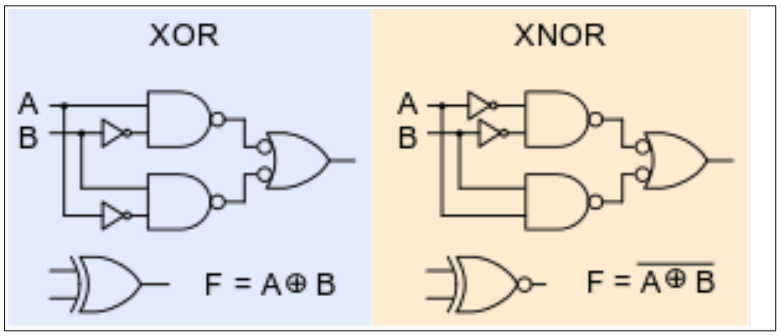
```
INVX1                          960
NAND2X1                       1296
NAND3X1                       1088
NOR2X1                         712
NOR3X1                          40
OAI21X1                        936
OAI22X1                         80
OR2X2                          160
```

디자인 킷에서 제공하는 표준 셀로 테크 매핑된 fir_pe 모듈의 회로도는 아래 그림과 같다.

```
yosys> show -colors 3 fir_pe
```



복합 논리 셀들을 사용했음에도 Yosys 내부 논리 소자들을 사용한 회로도 보다 경로가 더 길어 보인다. 기본 논리소자 $_XOR_ 과 $_XNOR_이 디자인 킷의 표준 셀 목록에 없어 NAND와 NOR를 사용하여 구현 되었기 때문이다. 디자인 킷의 표준 셀 라이브러리 보강이 필요하다.



[출처] XOR and XNOR Gates: Truth Tables, Equations, and Logic

2-5. Yosys의 효용성 논의

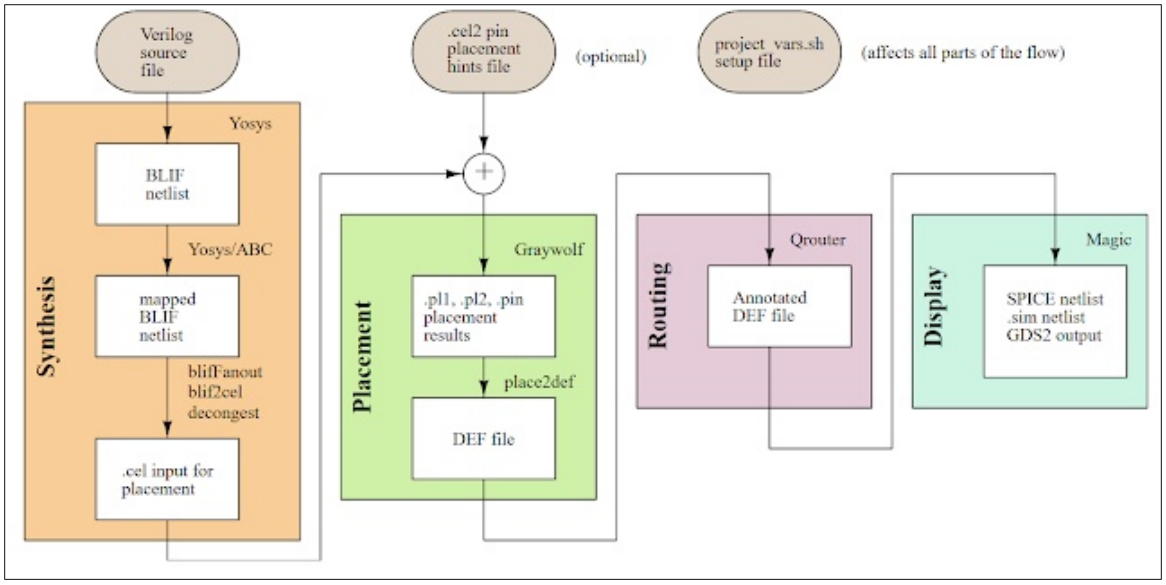
오픈-소스 합성기 Yosys 는 기능적으로 상당히 정교하다. 교육용 뿐만 아니라 상업용으로도 다양한 분야에서 널리 사용되고 있다. 물리적 시간 조건(timing closure) 합성 기능이 없다는 단점이 있다. Yosys에 관한 장단점 논의가 있으니 아래 링크들을 참고 한다.

[1] Yosys: Manual[pdf] Presentation 2022 [pdf] Command Reference [link]
[2] Notes on Verilog Support in Yosys [link]
[3] OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain [pdf]
[4] Roadmap and Recommendations for Open Source EDA in Europe [link]
[5] 시스템 반도체 설계 교육에 "내 칩(My Chip)제작 서비스"와 오픈-소스 EDA도구의 활용 [pdf]
[6] Yosys is toy I don't believe it[reddit][reddit/ko]
[7] Timing Closure [Wiki]

2-6. QFlow: Digital Synthesis Flow

a. QFlow

베릴로그 하드웨어 언어로 기술한 설계를 제조 도면(레이아웃)으로 생성하기까지 합성, 표준 셀 배치와 배선 그리고 레이아웃 생성에 관련된 일련의 추상성 전환이 이뤄진다. 전환의 과정에 자동화 도구들이 동원된다. 오픈-소스 자동화 도구들은 저마다 개발자들이 다르고 고유의 자료 형식을 가지고 있다. QFlow는 이들 도구들이 원할 하게 작동하도록 체계화한 반도체 설계 플로우(프레임워크)다.



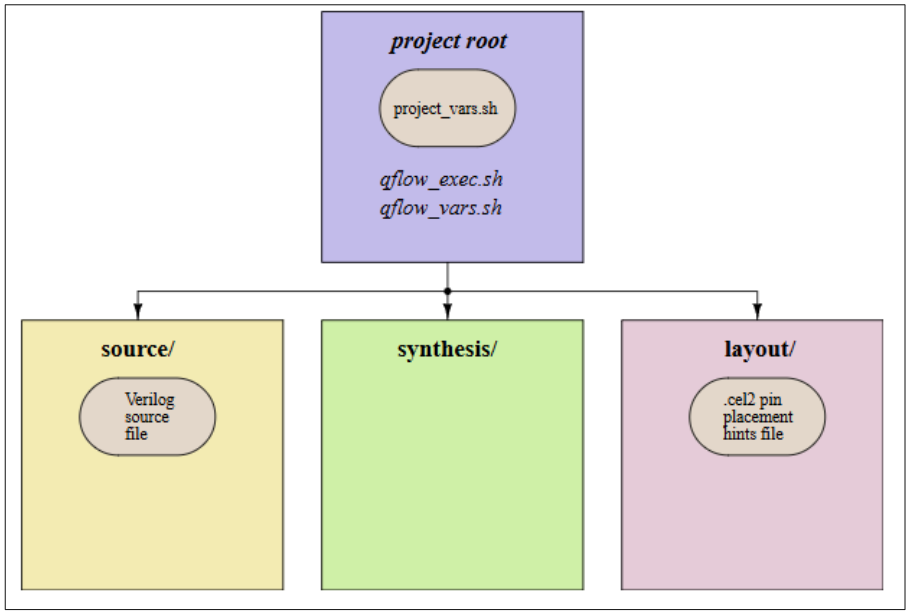
[출처] QFlow Digital Synthesis Flow Reference

"내 칩 제작 서비스"의 0.5um CMOS 공정을 QFlow의 플로우에 이식하고 확장한 "표준 셀 디자인 킷"을 개발하여 공개하였다[바로가기].

ETRI/NSPL 0.5um CMOS MPW Std-Cell Design Kit [link]

b. 디렉토리 구조

QFlow의 프로젝트를 수행 하려면 다음과 같은 디렉토리 구조를 유지해야 한다.



[출처] QFlow Digital Synthesis Flow Reference

프로젝트 최상위 디렉토리에 다음은 파일이 필요하다.

project_vars.sh

프로젝트를 수행하는 동안 각 도구들에게 주어질 옵션들을 설정한 파일로 반드시 필요하다.

qflow_vars.sh

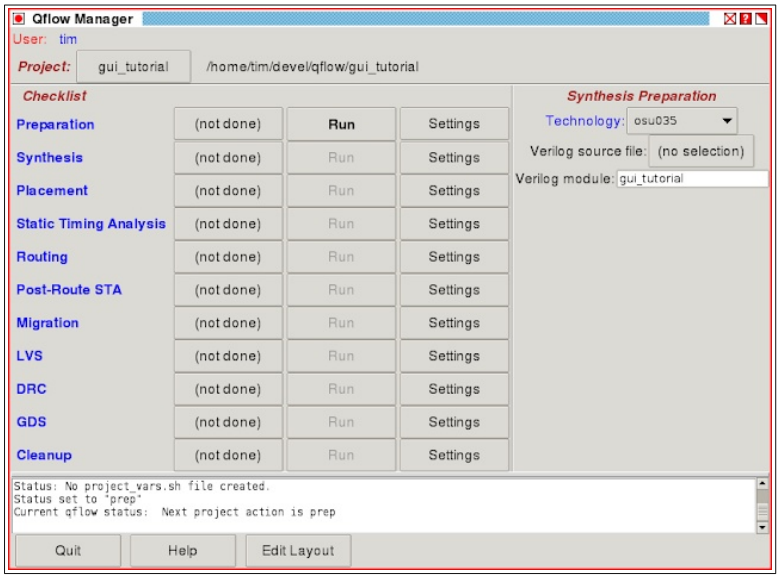
QFlow를 수행하는 동안 각 도구들이 생성하는 중간 파일들이 저장될 디렉토리들을 지정한다.

qflow_exec.sh

QFlow 가 진행되는 단계가 기록된다. 반드시 필요한 파일은 아니다.

2-7. 실습: 디지털 FIR 필터 합성

QFlow는 그래픽 사용자 인터페이스(GUI)를 갖추고 있으나 버튼을 눌러 과정을 진행하는 외에 다른 기능을 가지고 있지 않다. 본 실습은 Makefile을 활용한 명령줄 환경에서 진행한다.



[출처] QFlow GUI Tutorial

a. 실습 디렉토리 구조

반도체 설계 프로젝트의 수행 중 여러 단계의 추상화 수준 전환이 일어난다. 각 단계마다 자동화 도구가 만들어내는 중간 파일과 단계 마다 요구하는 파일들이 있다. 이 파일들을 효과적으로 관리해 주지 않을 경우 큰 낭패를 보게 되므로 각 단계마다 별도의 디렉토리를 두도록 하자.

예제 프로젝트 디렉토리로 이동,

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

프로젝트 디렉토리 구조를 보면 추상화 단계별 별도의 디렉토리를 두고 있는 것을 알 수 있다. 프로젝트 디렉토리에 QFlow에 필요한 파일들과 Makefile 이 있다.

```
$ tree
.
├── /layout
│   ├── ETRI050_CMOS.lyp : KLayout 도구용 레이어 정의 파일
│   ├── fir8.cel2 : 핀 배치
│   ├── fir8.par : 자동 배선 옵션
│   └── Makefile
├── /log : 자동화 도구의 처리내역(로그)
├── /simulation : 시뮬레이션
│   ├── Makefile
│   ├── sc_fir8.gtkw
│   └── Vfir8.gtkw
├── /source : 베릴로그 HDL 소스
│   └── fir8.ys : Yosys 합성 스크립트
├── /synthesis
├── Makefile
├── project_vars.sh
├── qflow_exec.sh
└── qflow_vars.sh
```

명령줄에서 make 를 실행하면 다음과 같은 안내문을 보게된다. Makefile의 타겟 all 은 안내문 출력이다.

```
$ make
Makefile for "fir8" QFlow RTL-to-Layout using ETRI 0.5um CMOS Technology
Usage:
    make [option]
         Use one of following options
             synthesize
             place
             sta
             route
             migrate
             lvs
             size
             clean
```

b. 합성

타겟 synthesisze를 주고 make 실행하여 베릴로그 설계를 합성한다.

```
$ make synthesize
```

Makefile의 타겟 synthesisze 가 하는 일은 다음과 같다. 베릴로그 소스 파일은 FIR 필터 알고리즘을 RTL로 설계할 때 작성된 것을 그대로 가져온다. 베릴로그 소스 파일이 디렉토리 ./source에 있어야 하므로 QFlow 합성을 하기 전에 심볼(소프트)링크로 연결하였다. QFlow를 명령줄에서 합성을 실행하면서 공정을 etri050으로 지정했다.

```
#Makefile
VERILOG_SRCS = \
../2-5_Lab3_FIR8/rtl_verilog/fir_pe.v \
../2-6_Lab4_FIR8_rtl_Emulation/source/fir8.v
TOP_MODULE = fir8
# ------------------------------------------------------
synthesize : ./log/synth.log
./log/synth.log : $(VERILOG_SRCS)
    @if ! [ -L ./source/fir_pe.v ]; then \
        ln -s ../../2-5_Lab3_FIR8/rtl_verilog/fir_pe.v ./source/fir_pe.v; \
    fi
    @if ! [ -L ./source/fir8.v ]; then \
        ln -s ../../2-6_Lab4_FIR8_rtl_Emulation/source/fir8.v \
                                                ./source/fir8.v; \
    fi
    qflow synthesize -T etri050 $(TOP_MODULE)
```

디렉토리 ./log 에 합성 기록 파일 synth.log를 확인 할 수 있다. 다음은 FIR 필터에 동원된 표준 셀의 목록이다.

```
13. Printing statistics.
=== fir8 ===
```

```
   Number of wires:                6413
   Number of wire bits:            7833
   Number of public wires:         6413
   Number of public wire bits:     7833
   Number of ports:                   5
   Number of port bits:              49
   Number of memories:                0
   Number of memory bits:             0
   Number of processes:               0
   Number of cells:                7048
     $scopeinfo                       8
     AND2X2                         456
     AOI21X1                        704
     AOI22X1                        136
     BUFX2                           24
     DFFPOSX1                       448
     INVX1                          960
     NAND2X1                       1296
     NAND3X1                       1088
     NOR2X1                         712
     NOR3X1                          40
     OAI21X1                        936
     OAI22X1                         80
     OR2X2                          160
```

QFlow는 디렉토리 ./source에 Yosys 합성 스크립트 fir8.ys 를 생성하게 되는데 내용은 다음과 같다. 합성 테크놀로지 매핑은 etri050 다. 리버티 파일 etri05_stdcells.lib은 "내 칩 제작 서비스"의 디자인 킷에 표준 셀 레이아웃과 함께 제공되었다.

```
# Synthesis script for yosys created by qflow
read_liberty -lib -ignore_miss_dir -setattr blackbox \
             /usr/local/share/qflow/tech/etri050/etri05_stdcells.lib
read_verilog fir8.v
read_verilog fir_pe.v
# High-level synthesis
synth -top fir8
# Map register flops
dfflibmap -liberty /usr/local/share/qflow/tech/etri050/etri05_stdcells.lib
opt
# Map combinatorial cells, standard script
abc -exe /usr/local/share/qflow/bin/yosys-abc \
    -liberty /usr/local/share/qflow/tech/etri050/ etri05_stdcells.lib \
    -script +strash;scorr;ifraig;retime,{D};strash;dch,-f;map,-M,1,{D}
flatten
setundef -zero
clean -purge
# Output buffering
iopadmap -outpad BUFX2 A:Y -bits
# Cleanup
opt
clean
rename -enumerate
write_verilog  fir8_mapped.v
stat
```

합성이 성공적으로 수행되면 시뮬레이션과 LVS 그리고 표준 셀 자동 배치에 사용될 네트리스트를 생성한다. 디렉토리 ./synthesis 에 생성되는 형식별 네트리스트는 다음과 같다.

./synthesis/fir8.v : 합성 후 시뮬레이션 용 네트리스트
./synthesis/fir8.spc : LVS 용 네트리스트
./synthesis/fir8_mapped.v : 표준 셀 자동 배치용 네트리스트

c. 합성 후 네트 시뮬레이션

합성의 결과를 검증 하기 위해 시뮬레이션을 실시한다. 시뮬레이션 디렉토리로 이동,

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

준비된 Makefile로 네트리스트 베릴로그 무결성 검사(linting),

```
$ make lint
verilator --sc -Wall \
          -Wno-UNUSEDSIGNAL \
          --top-module fir8 \
      ~/ETRI050_DesignKit/digital_ETRI/khu_etri05_stdcells_func.v  \
          ../synthesis/fir8.v
```

네트리스트 ./synthesis/fir8.v 는 표준 셀들 사이의 연결을 묘사한 네트리스트다. 표준 셀들을 하위 모듈로 두고 있다. 표준 셀의 시뮬레이션 모델 khu_etri05_stdcells_func.v 을 함께 컴파일 한다. 네트리스트 fir8.v의 무결성을 검사해보니 오류가 발견되었다. 베릴로그 언어의 규정에 모듈 입출력 포트를 와이어로 재선언이 금지되어 있다.

```
%Error: ../synthesis/fir8.v:1154:6: Duplicate declaration of signal: 'clk'
   : ... note: ANSI ports must have type declared with the I/O (IEEE 1800-
2023 23.2.2.2)
 1154 | wire clk ;
      |      ^~~
        ../synthesis/fir8.v:8:11: ... Location of original declaration
    8 |     input clk
      |           ^~~
        ... See the manual at https://verilator.org/verilator_doc.html?
v=5.039 for more assistance.
......
make: *** [Makefile:100: lint] Error 1
```

네트리스트 ./synthesis/fir8.v 에서 모듈 포트 부분을 다음과 같이 수정한다.

```
  module fir8(Xin, Xout, Yin, Yout, clk);
      input [7:0] Xin;
      output [7:0] Xout;
      input [15:0] Yin;
      output [15:0] Yout;
      input clk;
```

수정된 네트리스트 ./synthesis/fir8.v 를 DUT로 한 시뮬레이터를 빌드 한다. 타임드 모델의 검증용 테스트 벤치를 재사용 한다.

```
$ make build
verilator --sc -Wall --trace --top-module fir8 --exe --build \
          -Wno-UNUSEDSIGNAL \
          -CFLAGS -I../sc_timed \
          -CFLAGS -DVCD_TRACE_FIR8 \
          -CFLAGS -DVERILATED_CO_SIM \
          -CFLAGS -DFIR_MAC_VERSION \
          -LDFLAGS -lgsl \
          ~/ETRI050_DesignKit/digital_ETRI/khu_etri05_stdcells_func.v \
          ../synthesis/fir8.v \
          ../../2-5_Lab3_FIR8/sc_timed/sc_main.cpp \
          ../../2-5_Lab3_FIR8/sc_timed/sc_fir8_tb.cpp \
          ../../2-5_Lab3_FIR8/c_untimed/fir8.cpp \
          ../../2-5_Lab3_FIR8/c_untimed/cnoise.cpp
```

타임드 테스트벤치 sc_fir8_tb.h 에 SystemC 모델 sc_fir8 과 변환된 베릴로그 모델 Vfir8 을 동시에 사례화 되도록 조건부 컴파일 매크로 VERILATED_CO_SIM을 적용했다.

```
SC_MODULE(sc_fir8_tb)
{
    ......
    sc_fir8*                u_sc_fir8;     // Timed SystemC model
#ifdef VERILATED_CO_SIM
    sc_signal<uint32_t>     V_Xin;
    sc_signal<uint32_t>     V_Xout;
    sc_signal<uint32_t>     V_Yin;
    sc_signal<uint32_t>     V_Yout;
    Vfir8*  u_Vfir8;    // Verilated SystemC Model
#endif
SC_CTOR(sc_fir8_tb): clk("clk", 100, SC_NS, 0.5, 0.0, SC_NS, false)
    {
        SC_THREAD(Test_Gen);
        sensitive << clk;
        SC_THREAD(Test_Mon);
        sensitive << clk;
        // Instaltiate FIR8
        u_sc_fir8 = new sc_fir8("u_sc_fir8");
        u_sc_fir8->clk(clk);
        u_sc_fir8->Xin(Xin);
        u_sc_fir8->Xout(Xout);
        u_sc_fir8->Yin(Yin);
        u_sc_fir8->Yout(Yout);
#ifdef VERILATED_CO_SIM
        u_Vfir8 = new Vfir8("u_Vfir8");
        u_Vfir8->clk(clk);
        u_Vfir8->Xin(V_Xin);
        u_Vfir8->Xout(V_Xout);
        u_Vfir8->Yin(V_Yin);
        u_Vfir8->Yout(V_Yout);
#endif
};
```

테스트벤치의 콜백 함수 Test_Mon()에 두 DUT의 출력을 받아 비교하는 구문이 있다.

```
/*****************************************************************
```

```
Filename: sc_fir8_tb.cpp
*****************************************************************/
......
void sc_fir8_tb::Test_Mon()
{
    ......
    while(true)
    {
        wait(clk.posedge_event());
        yout = (uint16_t)Yout.read();
#ifdef VERILATED_CO_SIM
        V_yout = (uint16_t)V_Yout.read();
#endif
        ......
        if (y[n]!=yout)
            printf("Error:");
#ifdef VERILATED_CO_SIM
        if (y[n]!=V_yout)
        {
            printf("V_Err:");
            yout = V_yout;
        }
#endif
    ......
    }
}
```

시뮬레이터를 실행한다.

```
$ make run
```

타임드 모델과 합성으로 얻은 네트리스트가 동일한 출력을 냈다. 이로써 합성 도구에 의해 추상화 수준 변경이 무사히 완료되었다.

3. 표준 셀 자동 배치와 배선

작은 디지털 설계라도 수많은 표준 셀이 동원된다. 합성 이후 배치와 배선을 수동으로 하는 경우는 없다. 표준 셀의 배치와 배선은 자동화 도구를 사용하여 수행된다. 단지 몇 가지 옵션 수정이 가능할 뿐이다. 자동화 도구를 사용 할수록 반드시 검증의 과정이 수반되어야 한다. "내 칩 제작 서비스" 표준 셀 디자인 킷이 채택하고 있는 설계 플로우 QFlow의 자동 배치배선 도구는 GrayWolf와 QRouter 다. 두 도구의 사용시 약간의 옵션을 줄 수 있다[참조: QFlow 1.4 Reference].

3-1. GrayWolf 표준셀 자동 배치 도구

GrayWolf는 표준 셀 자동 배치를 수행하는 오픈-소스 도구다. 합성으로 얻은 네트리스트 fir8_mapped.v 를 읽어 자동 배치를 수행한다. 이 도구는 예일 대학교의 TimberWolf 프로젝트가 상용화 되기 직전 버젼이 공개되어 발전했다. 매크로 셀 자동 배치와 배선의 기능을 가지고 있었으나 GrayWolf는 표준 셀 자동 배치 기능으로 사용한다 [참조: GrayWolf].

a. 배치 옵션

자동 배치도구의 옵션은 ./layout/fir8.par 의 내용중 아래 사항을 수정 할 수 있다.

```
TWMC*chip.aspect.ratio : 1.0
```
코어의 가로세로 비율을 정한다.

```
GENR*numrows : 6
```
표준 셀 배치할 표준 셀의 행 갯수 제한. 대부분 이 옵션은 사용하지 않는다.

```
GENR*flip_alternate_rows : 1
```
표준 셀을 상하로 뒤집어 배치할 수 있다.

아울러 project_vars.sh 에 셀 배치 밀집도를 조절 한다. 표준 셀 배치 밀집도가 크면 배선공간이 줄어든다. 자동 배선을 실패할 경우 밀집도를 낮춰주고 재배치 해야 한다.

```
set initial_density = 0.6
```

b. 핀 배치 옵션

입출력 핀의 위치를 코어의 상(T)하(B)좌(L)우(R) 변에 배치할 수 있다. 핀배치 파일명은 ./layout/fir8.cel2 다.

```
padgroup group_name [permute| nopermute]
twpin_pinname [fixed|nonfixed ]
......
[restrict side sides]
```

아래의 예는 fir8의 출력 포트 clk, Xin, Yin을 완쪽(L)과 윗면(T)에 배치, 출력 포트 Xout 과 Yout을 오른쪽(R)과 아랫면(B)에 배치는 경우다.

```
padgroup Input_Group permute
twpin_clk nonfixed
twpin_Xin nonfixed
twpin_Yin nonfixed
restrict side LT
padgroup Output_Group permute
twpin_Xout nonfixed
twpin_Yout nonfixed
restrict side RB
```

c. 자동 배치 실행

Makefile의 타겟 place로 자동 배치를 실행하는 명령은 다음과 같다.

```
$ make place
```

```
qflow place -T etri050 fir8
Qflow placement logfile created on Mon Jul 28 07:16:36 PM KST 2025
Running vlog2Cel to generate input files for graywolf
```

```
vlog2Cel  -l .../etri050/etri050_stdcells.lef -u 100
          -o ./layout/fir8.cel ./synthesis/fir8.rtlnopwr.v
      ......
Preparing pin placement hints from fir8.cel2
Running GrayWolf placement
graywolf  fir8
twflow version:2.1 date:Mon May 25 21:15:08 EDT 1992
Authors: Bill Swartz, Carl Sechen
         Yale University
syntax version:v1.1 date:Mon May 25 21:11:10 EDT 1992
TimberWolf System Syntax Checker
Authors: Carl Sechen, Kai-Win Lee, Bill Swartz,
         Dahe Chen, and Jimmy Lam
         Yale University
Read   50 objects so far...
Read  100 objects so far...
Read  150 objects so far...
Read  200 objects so far...
......
Reading info file fir8.info. . .
Reading DEF file fir8.def. . .
Recalculating pin positions
Writing DEF file fir8_mod.def. . .
Done with arrangepins.tcl
DEF2Verilog -v ./synthesis/fir8.rtlnopwr.v
            -o ./synthesis/fir8_anno.v -p vdd -g gnd
            -l .../etri050/etri050_stdcells.lef fir8.def
Generating RTL verilog and SPICE netlist file in directory
Files:
   Verilog: ./synthesis/fir8.rtl.v
   Verilog: ./synthesis/fir8.rtlnopwr.v
   Verilog: ./synthesis/fir8.rtlbb.v
   Spice:   ./synthesis/fir8.spc
......
LEF Read: encountered 0 errors and 8 warnings total.
Running vlog2Cel to generate input files for graywolf
vlog2Cel  -l  .../etri050_stdcells.lef -u 100
          -o ./layout/fir8.cel ./synthesis/fir8.rtlnopwr.v
Running vlog2Spice.
vlog2Spice -i -l .../etri050_stdcells.sp -o fir8.spc fir8.rtl.v
```

합성으로 얻은 네트리스트 ./synthesis/fir8.rtlnopower.v 를 GrayWolf 입력 파일 형식 .cel 로 변환 한다. 이때 디자인 킷의 표준 셀 배치와 배선 규칙을 정의한 etri050_stdcells.lef 를 참조한다. 배치과정에서 네트리스트가 변경(필러 셀과 팬인-아웃을 감안한 버퍼 셀의 삽입)될 수 있다. LVS 를 대비하여 네트리스트를 재생성한다. 자동 배치와 배선도구 사이에 주고받는 파일 양식은 DEF 다. DEF는 네트리스트에 더하여 표준 셀의 배치 좌표, 핀 좌표 그리고 배치 방향 정보를 담고 있다. 수천개에 이르는 표준 셀을 옵션에 맞춰 배치하기는 수동 작업이 불가능 하다. 전적으로 자동화 도구를 사용한다.

3-2. QRouter: 자동 배선 도구

QRouter 는 표준 셀의 배치 정보 DEF 를 받아 세부 배선을 수행한다. 배선은 금속 층에서 이뤄진다. 금속 층의 배선 규칙(두께, 이격거리, 비아 컷의 기하학적 모양 등)은 공정의 디자인 룰을 따른다. 배치와 배선 규칙은 etri050_stdcells.lef 에 정의되었다. 수천 개에 이르는 표준 셀을 수동 작업으로 배선하기는 불가능 하다. 전적으로 자동화 도구를 사용한다. 자동배선 도구에 주어질 옵션은 거의 없다.

```
$ make route

Qflow route logfile created on Mon Jul 28 07:20:11 PM KST 2025
qrouter -noc  -s fir8.cfg
Qrouter detail maze router version 1.4.88.T
Reading LEF data from file
        /usr/local/share/qflow/tech/etri050/etri050_stdcells.lef.
......
Reading DEF data from file fir8.def.
Diagnostic: Design name: "fir8"
  Processed 7164 nets total (0 fixed).
  Processed 2 special nets total (2 fixed).
DEF read: Processed 62028 lines.
......
There are 7164 nets in this design.
*** Running stage1 routing with defaults
Finished routing net gnd
Nets remaining: 7163
Finished routing net vdd
Nets remaining: 7162
Finished routing net \X[4]\[6]
Nets remaining: 7161
......
-----------------------------------------------
Progress: Stage 1 total routes completed: 16156
Failed net routes: 214
-----------------------------------------------
*** Running stage2 routing with options mask 10, effort 10
Nets remaining: 214
Best route of _3261_ collides with net: \X[4]\[2]
Ripping up blocking net \X[4]\[2]
Nets remaining: 214
Best route of _3121_ collides with net: \u_fir_pe2.rYin\[0]
Ripping up blocking net \u_fir_pe2.rYin\[0]
Nets remaining: 214
......
-----------------------------------------------
Progress: Stage 3 total routes completed: 49776
No failed routes!
-----------------------------------------------
*** Writing DEF file fir8_route.def
emit_routes():  DEF file has 7164 nets and 2 specialnets.
but qrouter wants to write 7164 nets and specialnets.
-----------------------------------------------
Final: No failed routes!
-----------------------------------------------
```

배선에 실패할 경우 배치 밀도를 낮춰 재배치 후 다시 배선을 시도한다.

3-3. 표준 셀 병합

배치와 배선은 표준 셀을 부품으로 삼아 금속층에서 이뤄 진다. 부품의 내부 모습은 노출될 필요없이 배선을 위한 포트를 참조한다. 내부가 노출되지 않고 외형만 나타낸 것을 블랙박스(blackbox)라 한다. 배선까지 마친 후 LVS를 수행하려면 표준 셀을 드러내야 한다. 배치와 배선을 마친 레이아웃에 라이브러리로 제공된 표준 셀을 합치는 과정이 셀 병합(migration)이다. 셀 별합은 다음과 같다.

```
$ make migrate

qflow migrate -T etri050 fir8
Technology set to etri050
......
Running magic 8.3.529
magic -dnull -noconsole  migrate_fir8.tcl
...
Reading LEF data from file
         /usr/local/share/qflow/tech/etri050/etri050_stdcells.lef.
......
DEF read: Processed 105730 lines.
......
Extracting FILL into FILL.ext:
Extracting AOI21X1 into AOI21X1.ext:
Extracting NAND3X1 into NAND3X1.ext:
Extracting OAI21X1 into OAI21X1.ext:
Extracting INVX1 into INVX1.ext:
......
Extracting BUFX2 into BUFX2.ext:
Extracting CLKBUF1 into CLKBUF1.ext:
Extracting INVX2 into INVX2.ext:
Extracting fir8 into fir8.ext:
exttospice finished.
```

3-4. LVS

일단 검증된 네트리스트가 준비되면 이후 배치와 배선은 자동화 도구에 의해 레이아웃이 만들어 진다. 합성으로 얻은 네트리스트를 언어로 표현한 회로도라 할 수 있다. 이를 배치와 배선 후 표준 셀 병합까지 마치면 ./layout 디렉토리 레이아웃이 생성된다. LVS는 Layout versus Schematics 의 약자다. 말 그대로 자동화 도구가 생성한 레이아웃에서 네트리스트를 추출하여 합성으로 얻은 네트리스트와 비교한다.

```
$ make lvs

Qflow LVS logfile created on Mon Jul 28 07:24:53 PM KST 2025
netgen -batch lvs "fir8.spice fir8" "./synthesis/fir8.spc fir8"
Netgen 1.5.295 compiled on Fri Jun 13 05:44:58 PM KST 2025
......
Contents of circuit 1:  Circuit: 'OR2X2'
Circuit OR2X2 contains 0 device instances.
Circuit contains 0 nets.
```

```
Contents of circuit 2:  Circuit: 'OR2X2'
Circuit OR2X2 contains 6 device instances.
  Class: pfet                 instances:   3
  Class: nfet                 instances:   3
Circuit contains 7 nets.
Circuit OR2X2 contains no devices.
......
Contents of circuit 1:  Circuit: 'fir8'
Circuit fir8 contains 7137 device instances.
  Class: NOR3X1               instances:  40
  Class: OR2X2                instances: 160
  Class: AOI22X1              instances: 136
  Class: NOR2X1               instances: 712
......
  Class: CLKBUF1              instances:  65
  Class: INVX1                instances: 912
  Class: INVX2                instances:  48
Circuit contains 7164 nets.
Contents of circuit 2:  Circuit: 'fir8'
Circuit fir8 contains 7137 device instances.
  Class: NOR3X1               instances:  40
  Class: OR2X2                instances: 160
  Class: AOI22X1              instances: 136
  Class: NOR2X1               instances: 712
......
  Class: CLKBUF1              instances:  65
  Class: INVX1                instances: 912
  Class: INVX2                instances:  48
Circuit contains 7164 nets.
Circuit 1 contains 7137 devices, Circuit 2 contains 7137 devices.
Circuit 1 contains 7164 nets,    Circuit 2 contains 7164 nets.
Final result:
Circuits match uniquely.
LVS Done.
LVS reports no net, device, pin, or property mismatches.
```

3-5. 코어 크기

자동화 도구에 의해 생성된 레이아웃의 평면 크기를 확인해보자. FIR 필터의 코어 크기를 구하는 명령은 다음과 같다. Makefile의 타겟 size은 디자인 킷의 쉘 스크립트 ./scripts/size_core.sh 를 실행 한다.

> [주] 설계 회사(팀)은 저마다 기본 도구에 더하여 스크립트와 별도의 도구들을 구축하여 설계 효율을 높인다. 이런 도구들은 사내-도구(In-House Tools) 라고 한다. "내 칩 제작 서비스" 다지인 킷의 ./scripts 디렉토리에 유용한 스크립트(쉘 스크립트와 파이썬 코드)들을 볼 수 있다.

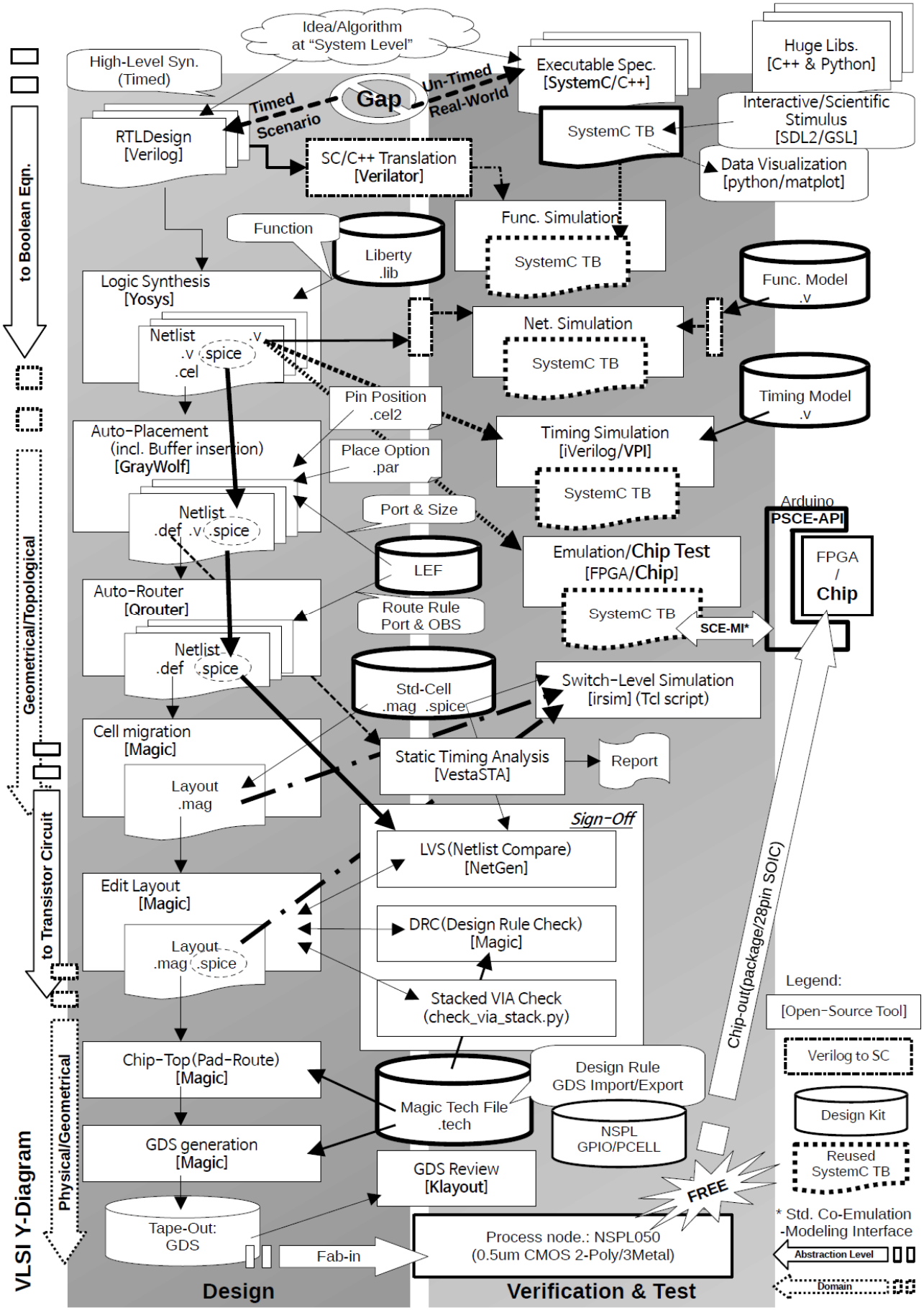
```
$ make size

~/ETRI050_DesignKit/scripts/size_core.sh fir8
*********************************************************************
* Measure the size of the Core
*********************************************************************
```

```
Magic 8.3 revision 529 - Compiled on Fri Jun 13 05:45:16 PM KST 2025.
Starting magic under Tcl interpreter
Using the terminal as the console.
......
Input style lambda=0.30(p): scaleFactor=30, multiplier=1
Root cell box:
             width x height     (   llx,   lly   ), (   urx,    ury   )
microns:   2568.600 x 2533.050  (-9.300, -3.600), ( 2559.300, 2529.450)
lambda:    8562.00 x 8443.50    (-31.00, -12.00), ( 8531.00,  8431.50)
internal:   17124 x 16887       (   -62, -24    ), ( 17062,    16863)
```

4. 맺음말

디지털 FIR 필터의 베릴로그 설계를 합성하여 자동 배치 배선을 거쳐 레이아웃까지 생성하는 과정을 살펴봤다. 소프트웨어 개발은 높은 수준의 프로그램 코드에서 낮은 수준의 기계어 코드로 변환하는 과정이다. 이에 비해 하드웨어 개발은 추상화 수준의 차이는 매우 넓고 깊다. 추상화 수준의 변경에 더하여 기술 방식(파일 양식)의 변화도 동반된다. 소프트웨어 개발에서 컴파일러 사용법에 그리 주목하지 않는 반면 하드웨어 개발에서는 도구를 다루는 엔지니어가 별도로 있다. 하드웨어, 특히 반도체 개발에 자동화 도구가 동원되었더라도 "손이 매우 많이 가는" 작업이다. 그에 따른 검증의 절차 또한 빼놓을 수 없다. 설계 생산성을 높이려면 설계 자동화 도구의 사용자로서 각 도구들이 작동하는 방식과 도구들 사이의 입출력 파일 양식을 이해할 필요가 있다. 개발 도구의 **이용**자에 그치지 않고 도구의 **활용**자가 되어야 한다 [이용과 활용의 차이].

칩의 평면적과 입출력 패드의 수는 상업적 면에서 매우 중요하다. 칩의 크기가 바로 제조 비용이다. 따라서 설계된 레이아웃(코어)의 평면 크기는 칩 제작 요건에 제약 받는다. "내 칩 제작 서비스"의 칩 크기는 1900x1900로 제한된다. 입출력 패드를 제외하면 코어의 크기는 1000x1000um 이내다. 디지털 FIR 필터 fir8의 레이아웃 크기가 2568.600 x 2533.050um 나 된다. "내 칩 제작 서비스"의 MPW 요건보다 크다. 이 크기는 설계 도구의 옵션 조절로 극복할 수 없는 수준이다. 이제 실무적인 문제를 해결해야 할 단계가 되었다.


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