반도체설계교육센터
IC DESIGN EDUCATION CENTER

IDEC newsletter

VOL. 204
JUNE 2014

IDEC Newsletter I **통권** 제204호
⊙ **발행일** 2014년 5월 30일 ⊙ **발행인** 박인철 ⊙ **편집인** 남병규 ⊙ **제 작** 푸울디자인
⊙ **기 획** 전항기 ⊙ **전 화** 042) 350-8535 ⊙ **팩 스** 042) 350-8540 ⊙ **홈페이지** http://idec.or.kr
⊙ **E-mail** jhg0929@idec.or.kr ⊙ **발행처** 반도체설계교육센터(IDEC)

반도체설계교육센터 사업은 미래창조과학부(산업통상자원부), 한국반도체산업협회, 반도체회사(삼성전자, SK하이닉스, 매그나칩반도체, 동부하이텍, 앰코테크놀로지코리아, KEC, 에이티세미콘, TowerJazz)의 지원으로 수행되고 있습니다.

IDEC newsletter

VOL. 204 JUNE 2014

MPW (Multi-Project Wafer) 2014년 MPW 진행 현황

공정	회차구분 (공정_년도순서)	모집팀수 ((mmxmm)x 칩수)/희별	정규모집 신청마감	참여팀수 ((mmxmm))x 칩수)	DB 마감 (Tape-out)	Die-out	비고
삼성 65nm	S65-1401	(4x4)x48	2013.12.09	(4x4)x20	2014.02.17	2014.08.18	제작중
	S65-1402		2014.02.03	(4x4)x33	2014.08.25	2015.02.27	설계중
	S65-1403		2014.06.02	(4x4)x10	2014.12.15	2015.06.12	정규모집중
매그나칩/ SK하이닉스 0.18㎛	MS18-1401	(3.8x3.8)x20	2013.12.09	(3.8x3.8)x20	2014.02.24	2014.07.28	제작완료
	MS18-1402		2014.01.06	(3.8x3.8)x25	2014.05.19	2014.10.20	DB검토중
	MS18-1403		2014.02.03	(3.8x3.8)x25	2014.08.11	2015.01.12	설계중
	MS18-1404		2014.05.05	(3.8x3.8)x24 (3.8x1.9)x2	2014.11.10	2015.04.13	설계중
매그나칩/ SK하이닉스0.35㎛	MS35-1401	(5x4)x20	2014.01.06	(5x4)x20	2014.06.16	2014.10.06	설계중
	MS35-1402		2014.06.02	(5x4)x2	2014.12.01	2015.03.23	정규모집중
동부0.11㎛	D11-1401	(5x2.5)x24	2013.12.09	(5x2.5)x24	2014.04.02	2014.08.06	제작중
	D11-1402		2014.04.07	(5x2.5)x17 (2.35x2.35)x14	2014.10.01	2015.02.04	설계중
동부0.18㎛ BCD	D18-1401	(5x2.5)x4	2013.12.09	(5x2.5)x4	2014.03.27	2014.07.02	제작중
	D18-1402		2013.12.09	(5x2.5)x4	2014.04.30	2014.08.06	설계중
	D18-1403		2014.02.03	(5x2.5)x3 (2.35x2.35)x2	2014.08.13	2014.11.19	설계중
	D18-1404		2014.05.05	(5x2.5)x3 (2.35x2.35)x2	2014.11.05	2015.02.11	설계중
동부0.35㎛ BCD	D35-1401	(5x2.5)x6	2013.12.09	(5x2.5)x6	2014.02.19	2014.05.28	제작완료
	D35-1402		2014.01.06	(5x2.5)x6	2014.05.28	2014.09.03	DB검토중
	D35-1403		2014.03.03	(5x2.5)x4 (2.35x2.35)x2	2014.09.10	2014.12.17	설계중
	D35-1404		2014.05.05	(5x5)x 4 (2.35x2.35)x2	2014.11.19	2015.02.25	설계중
TowerJazz 0.18㎛ BCD	TJB18-1401	(5x2.5)x6	2014.01.06	(5x5)x 3	2014.05.12	2014.09.08	제작중
	TJB18-1402		2014.04.07	(5x5)x 1	2014.10.20	2015.02.16	설계중
TowerJazz 0.18㎛ CIS	TJC18-1401	(2.5x2.5)x4	2014.01.06	(2.5x2.5)x4	2014.05.05	2014.09.01	제작중
	TJC18-1402		2014.04.07	(2.5x2.5)x4	2014.10.13	2015.02.09	설계중
TowerJazz 0.18㎛ CA18HA	TJR18-1401	(2.5x2.5)x4	2014.01.06	(2.5x2.5)x4	2014.06.23	2014.10.20	설계중
	TJR18-1402		2014.04.07	(2.5x2.5)x4	2014.10.20	2015.02.16	설계중
TowerJazz 0.18㎛SiGe	TJS18-1401	(2.5x2.5)x4	2013.12.09	(2.5x2.5)x4	2014.03.11	2014.07.08	제작중

* 일정은 사정에 따라 다소 변경될 수 있음.
* 우선/정규 모집은 마감일 2주전부터 신청 가능함
* 회차 표기 방법 변경 : 공정코드-년도 모집순서 [예시] 삼성65nm 2014년1회차:S65-1401]
* Package 제작은 Die out 이후 1개월 소요됨
* 동부와 TowerJazz 공정은 sub chip[5mmX2.5mm 또는 2.35mmX2.35mm]으로 분리하여 모집
* 선정 결과는 모집 마감 후 15일 이내 개별 통보됨
* 2014년 우선모집은 완료됨에 따라 일정에서 제외함
* 기준일 : 2014. 5. 30.

* 담당 : 이의숙 (042-350-4428, yslee@idec.or.kr)

2014년 6월 교육프로그램 안내

수강을 원하는 분은 IDEC홈페이지(www.idec.or.kr)를 방문하여 신청하시기 바랍니다.

KAIST 개설 강좌 안내

센터명	강의일자	강의제목	분류
본센터	6월 24일	XMODEL을 활용한 디지털 PLL 설계 및 시뮬레이션	설계강좌
	6월 25일	CMOS RF: What's next? Digital calibration of nonlinear memory error in sigma-delta ADC	세미나
	6월 26일 – 27일	Calibre xRC	Tool강좌
	6월 30일 – 7월 2일	고성능 데이터변환기 설계를 위한 이론 및 실습	설계강좌

- 강좌일 : 6월 24일
- 강좌 제목 : XMODEL을 활용한 디지털 PLL 설계 및 시뮬레이션
- 강사 : 김재하 교수 (서울대학교)

강좌개요 XMODEL은 디지털 시뮬레이터인 SystemVerilog상에서 아날로그 회로의 동작 및 특성을 정확하고 빠르게 모델링하고 시뮬레이션할 수 있는 툴 패키지이며, 같은 기능을 가진 Verilog-AMS에 비해 10~100배의 속도 성능을 자랑한다. 본 강좌에서는 XMODEL의 개념과 기본 사용법을 배우고, 이를 디지털 PLL를 구성 설계에 응용하는 법을 실습한다. 예를 들어, 디지털 PLL을 구성하는 회로요소들인 TDC, DCO 등의 특성을 모델링하는 법을 배우고, 디지털 루프필터 설계를 포함한 PLL 시스템을 구성하고, 지터 등 그 PLL 시스템의 동작 특성을 여러 시뮬레이션을 통해 측정하는 실습을 진행한다.
수강대상 대학원생 및 직장인
강의수준 중급
강의형태 이론+실습
사전지식, 선수과목 Verilog등 HDL언어에 대한 기본적인 이해, Phase-Locked Loop의 기능과 동작에 대한 기본적인 이해

- 강좌일 : 6월 25일
- 강좌제목 : CMOS RF: What's next?, Digital calibration of nonlinear memory error in sigma-delta ADC
- 강사 : 조병학 상무, 이승철 수석 (삼성전자)

강좌개요 • CMOS RF: What's next? By Thomas Cho
In the past decade, the consumer market for mobile devices such as notebooks, smart phones, tablets, etc., has grown exponentially, transforming people's everyday lifestyle. One of the key enabling factors is the scaled CMOS technology, which gave birth to a low-cost small-form-factor implementation of highly integrated RF solutions with powerful DSP, often on the same die. As the consumers' insatiate appetite for smaller, cheaper, and now SMARTER devices will continue in the future, a new question arises now for CMOS RF engineers. What would RF solutions look like in the future? Will we simply re-design RF using smaller geometry CMOS process? Will cheaper and yet more powerful digital make CMOS RF slowly disappear, having only antenna and A/D converter remain? If not, what new challenges are waiting for us? In this talk, a brief history of CMOS RF development will be presented along with discussion on the new challenges for CMOS RF in the next decade.

• Digital calibration of nonlinear memory error in sigma-delta ADC
by Seung-Chul Lee
A 1-0 MASH $\Sigma\Delta$ analog-to-digital converter (ADC) demonstrates a digital linearization technique treating integrator distortion with memory and capacitor mismatch errors. A two-tap sequential polynomial derived from an output-referred error analysis accurately models the non-ideality of a first-order modulator. The model parameters are extracted by correlating various moments of the ADC digital output with a one-bit pseudorandom noise (PN) superimposed on the input, largely reducing the circuit overhead associated with the nonlinear calibration. An analog-domain PN removal technique also resolves the loss of conversion dynamic range problem in the multi-bit feedback digital-to-analog converter (DAC) calibration due to PN circulation in the $\Sigma\Delta$ loop.

- 강좌일 : 6월 26일 – 27일
- 강좌 제목 : Calibre xRC
- 강사 : 정인철 과장(한국멘토그래픽스)

강좌개요 Parasitic 저항 및 커패시터를 추출하는 Calibre xRC의 사용법과 Rule File Generation에 대하여 교육 합니다. 다양한 Design Style에 맞는 Extraction 방법에 대하여 실습 위주로 교육합니다.
수강대상 Calibre xRC User
강의수준 초급
강의형태 이론+실습
사전지식, 선수과목 Calibre 경험이 필요하나, Basic 내용이 포함되어 있기 때문에 처음 Tool를 사용하시는 분도 가능

- 강좌일 : 6월 30일 – 7월 2일
- 강좌 제목 : 고성능 데이터변환기 설계를 위한 이론 및 실습
- 강사 : 류승탁 교수 (KAIST)

강좌개요 Nyquist data converter를 중심으로 동작의 기본 원리부터 시작하여 최신 설계 동향까지 다루게 된다. 먼저 ADC/DAC의 동작원리와 성능척도에 대해 소개하고, 여러 형태의 ADC및 DAC에서 성능저하를 일으키는 요인에 대해 고찰하여 실제 설계에서 고민해야 할 점들을 이야기한다.
수강대상 데이터 변환기 설계를 목적으로 하는 대학원생, 직장인
강의수준 중급
강의형태 이론+실습
사전지식, 선수과목 학부 전자회로 지식을 필수로 함

*문의 : KAIST IDEC 구재희 (042-350-8536, kjh9@idec.or.kr)

IDEC 논단

IoT시대, 개방형 SoC 플랫폼의 필요성

이 석 희 원장
SK하이닉스 미래기술연구원

2007년 애플의 아이폰이 등장했을 때 노키아의 최고경영자 올리 페카 칼라스부오는 '우리는 세 살배기용 스마트폰 따위는 만들지 않는다' 라고 일축했다. 당시 노키아는 세계 휴대전화 시장의 40%를 차지하고 핀란드 전체 법인세의 23%를 납부하고 있었다. 고인이 된 스티브 잡스조차 아이폰 출시 당시 시장점유율 1%를 목표로 한다고 말했을 정도로 애플이 휴대전화 시장에서 노키아를 넘어설 것이라고는 아무도 예상하지 못 했을 것이다. 그러나 7년 후 애플의 스마트폰 시장점유율은 13% 이상인 반면 노키아의 시장 점유율은 겨우 3.5%이다. 휴대전화사업의 부진으로 막대한 규모의 적자에 시달리던 노키아는 수많은 직원들을 해고하고 본사 사옥을 매각하는 등 구조조정에 힘을 쏟았으나 결국 신용평가사들로부터 투자부적격으로 분류되는 수모 끝에 휴대전화사업부를 마이크로소프트에 넘기고 말았다. 많은 이들이 노키아가 스마트폰 시장에 대한 대응이 늦었고 그로 인해 몰락했다고 생각하지만 사실 노키아는 아이폰이 등장하기 훨씬 전인 1990년대부터 스마트폰이라고 불릴 만한 제품들을 여럿 출시해왔다. 1996년에 이미 달력, 계산기, 주소록, 전자우편, 인터넷이 가능한 노키아 9000 커뮤니케이터를 출시했고 2000년에는 터치스크린과 심비안 OS를 사용한 에릭슨380를 에릭슨, 모토로라 등과 공동 개발하였다. 더 놀라운 사실은 이미 노키아가 1990년대 말에 노트북을 대체하기 위한 용도로 타블렛 PC를 극비리에 개발했었다는 것이다. 새로운 제품의 시대를 누구보다 빨리 예상했고 그것을 만들어낼 기술도 이미 십 수년 전부터 가지고 있었다. 시장지배력은 그야말로 압도적이었다. 대체 무엇이 문제였을까?

그것은 바로 자신들이 모든 것을 해낼 수 있다고 믿는 오만함, 다시 말해 '자만' 이었다. 성공한 자가 자만을 경계하는 것은 미덕이지만 현대의 첨단 기술 산업 군에서 '자만' 의 의미는 조금 더 의미심장하다. 바로 '폐쇄성' 이다. 실제 노키아를 쓰러뜨린 것은 매력적인 디자인에 부드러운 터치감을 자랑하는 아이폰 그 자체가 아니라 앱스토어였다. 노키아가 실패한 이유는 대부분의 일반 사용자들이 스마트폰으로 무엇을 해야 할지 알 수 없었고 마니아들을 위한 장난감이나 바쁜 비즈니스맨을 위한 복잡한 통신 수단 정도로 인식했기 때문이었다. 애플은 앱스토어의 매출에 대해 30%의 수수료를 받는 조건으로 독립적인 개발자들이 만든 수천 개의 애플리케이션을 유통시키면서 스마트폰의 놀라운 확장성과 다양한 활용 예제를 소비자들에게 확실하게 각인시켰다. 애플은 순식간에 IT 업계의 맹주이자 혁신의 아이콘으로 떠올랐다. 만약 애플이 이 모든 애플리케이션을 단독으로 개발하려 시도했다면 절대 불가능했을 일이었다. 아이폰이라는 하드웨어와 iOS라는 플랫폼을 제공하되 그 안에서 동작하는 소프트웨어는 사용자와 시장에 맡기는 개방성이 애플의 성공 이유였던 것이다. 애플의 움직임을 주시하던 구글은 한 발자국 더 나아갔다. 구글은 모바일 OS 업체인 안드로이드를 인수하고 그 OS를 다수의 휴대폰 제조업체에 공개했다. 애플이 소프트웨어를 개방시켰다면 구글은 OS까지 개방시킨 것이다. 이러한 구글의 전략은 그야말로 전 세계 스마트폰 시장을 폭발시켰다. 애플의 성공을 불안과 시기 속에 지켜보던 수많은 휴대폰 제조업체들이 다양한 사양과 소비자 요구를 반영한 제품을 앞 다투어 쏟아냈다. 현재 전 세계 스마트폰 5대 중 4대는 구글의 안드로이드 OS를 사용한다. 만약 구글이 안드로이드를 독점적으로 사용하며 스마트폰을 직접 생산하여 애플과 경쟁하려 했다면 어떻게 되었을까? 애플을 이기는 것은 가능했을지도 모르겠지만 지금과 같은 거대한 스마트폰 시장과 안드로이드로 대표되는 모바일 생태계는 열리지 않았을 것이다.

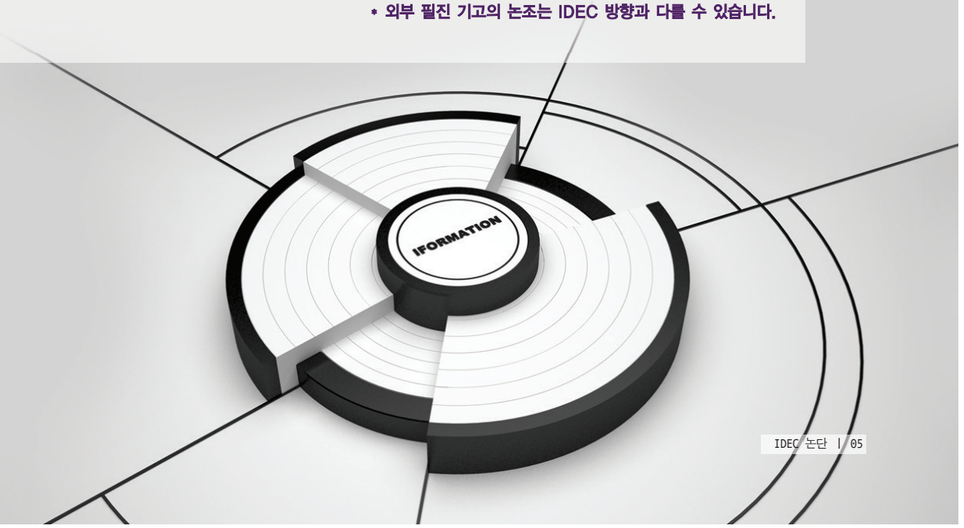
지금 전세계 ICT 업계 종사자들과 전문가들은 지난 약 10년을 지배했던 스마트폰 시장의 성장 둔화에 대한 우려와 새로운 성장동력으로 등장한 IoT (Internet of Things : 사물인터넷)에 대한 기대로 술렁이고 있다. IoT는 인간과 사물, 서비스의 환경요소가 인간의 명시적인 개입 없이 상호 협력적, 지능적 관계를 형성하는 사물 공간 연결 망을 뜻한다. IoT와 유사한 개념은 M2M (Machine to Machine), 유비쿼터스 등의 이름으로 이전부터 존재했고 이미 일부 구현되고 있다. 우리의 스마트폰은 과거 한 나라의 정보기관이나 가지고 있었을 법한 방대한 양의 정보들을 실시간 스트리밍을 통해 언제 어디서든 제공해 준다. 자동차에는 사고를 내기가 더 어려울 정도로 정교한 ADAS가 장착되고 있고 심지어 베개에도 센서가 장착되어 수면 중 흘린 땀과 체온 등을 측정한 후 수면의 질을 분석해 주기도 한다. 하지만 IoT 시대에는 더 나아가 이런 베개로부터 간밤의 수면 상태가 전송되면 스마트폰이 사용자가 감기에 걸릴 가능성을 판단하고 근처 병원에 다시 이 상황을 전달, 사용자가 퇴근 후 집에 오면 조제된 약이 미리 도착해 있는 수준의 상호 협력적이고 지능적인 교류가 이루어지게 될 것이다. 이처럼 사람들이 IoT에 열광하는 것은 단순히 2020년까지 10조 달러 이상의 경제 효과를 창출한다거나 수백억 개의 사물들이 연결된다는 거창한 숫자 때문만은 아니다. IoT는 개방적 생태계의 궁극적인 모습이고 그로 인해 상상할 수 있는 거의 모든 서비스들이 실제로 구현될 수 있을 것이라는 기대가 있기 때문이다. 이미 우리는 스마트폰 시장에서 개방성이 기존의 골리앗들을 쓰러뜨리고 우리의 삶을 송두리째 변화시키는 것을 직접 목격하였다.

그렇다면 IoT 시대의 System IC 반도체는 무엇을 개방해야 하는가? IoT 시대를 준비하는 반도체 제조업체들은 두 가지 딜레마에 빠져있다. 첫째, 새로운 아이디어와 그로 인한 수요, 공급이 매 순간 나타났다 사라지는 복잡하고 거대한 IoT 생태계에서 모든 것을 예측하여 선제적으로 대응하는 것이 불가능하다는 점이다. IoT 디바이스가 요구하는 초저전력, 초저가, 초소형의 특징을 구현하려면 필연적으로 System IC 반도체는 SoC의 형태를 띄게 되는데 그 기능과 형태에 대한 요구가 천차만별이다. 예를 들어 어떤 고객은 MCU와 Bluetooth, Flash memory에 초저전력 특성이 결합된 SoC를, 또 다른 고객은 AP와 Wi-Fi, PMIC에 고성능 특성이 결합된 SoC를 원할 수도 있는데 무엇을 선택하고 집중할 것이냐의 문제는 차라리 예언에 가까울 정도로 어렵다. 다양한 포트폴리오를 갖추는 것이 전통적인 해결책이지만 이러한 방대한 포트폴리오를 개발하고 유지하는데 필요한 resource를 감당할 수 있는 SoC 제조업체는 지금도 극소수이며 이들 업체들도 적절한 제품을 선정하고 시장 진입 시점을 판단하는 것에 애를 먹고 있다. 둘째, IoT 관련 SoC 초기 수요의 상당수는 적은 물량을 주문하는 다수의 소형 고객들이지만 이들이 요구하는 SoC를 구현하기 위한 기술들은 막대한 규모의 투자와 R&D 노력이 수반되어야 확보할 수 있는 것들로 대량 주문, 대량 생산을 통한 원가 절감과 수익성 제고가 전제되지 않는다면 해당 기술을 개발하는데 위험부담이 너무 크다는 점이다. 특히 스마트폰과 같은 단일 form factor가 존재하지 않고 제품 순환 주기도 상대적으로 길 것으로 예상되는 IoT 시장에서 이러한 우려는 더욱 커질 수 밖에 없다.

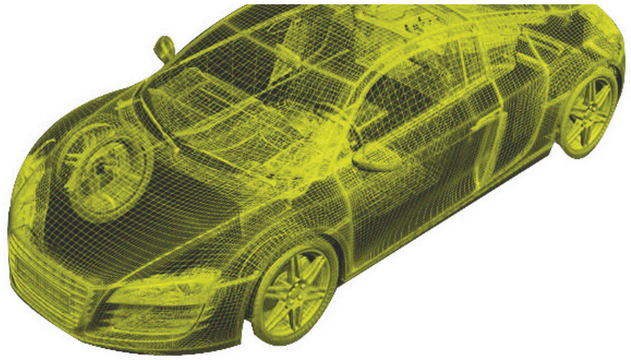
반대로 IoT SoC를 구매할 Start up 업체들 역시 난관에 부딪쳐 있다. 이들 업체의 대부분은 컴퓨터 앞에 앉아 OS와 API를 붙들고 새로운 solution과 application을 개발하는데 몰두하고 싶어하지 이를 실제로 구동시킬 SoC를 설계할 여력과 관심은 없다. 그로 인해 현재의 Foundry + Fabless의 사업 모델 (1987년 모리스 창이 TSMC를 설립하며 제시한 Foundry 사업 모델도 반도체 산업에서 IDM에 종속되어 있던 Design house들에게 Chip 생산을 개방하여 성공한 또 다른 예이다)을 적용할 수 없어서 OEM이나 EMS가 정해주는 SoC와 module 형태를 그대로 받아들이거나 자신들의 제품에 그나마 가까운 SoC를 chip 공급자가 제공하는 포트폴리오에서 고를 수 밖에 없다. 이러한 괴리는 가격과 성능 측면에서 필연적으로 타협을 요구하며 이는 작은 Start up 업체들에게 심각한 경쟁력 저하로 이어질 수 있다. 만약 유사한 형태의 SoC마저 존재하지 않을 경우, 제품 개발 자체가 불가능하거나 여러 chip을 PCB에 SMT하는 방법으로 구현해야 해서 처음 기획했던 제품과는 본질적으로 다른(IoT로 보기 어려운) 형태가 될 수도 있다. 이러한 문제를 해결하기 위해서는 여러 SoC를 개별적으로 개발하여 그 중 몇 가지가 운 좋게 시장에서 많이 쓰이기를 기대하기보다 각각의 사용자들이 자신들의 필요에 따라 SoC를 선택, 조합할 수 있는 개방적이고 유연한 공통 플랫폼의 개발이 우선되어야 한다. IoT SoC는 크게 Processor, Connectivity, Memory, Sensor의 네 가지 기능을 구현해야 하는데 각각의 기능들은 독립적인 sub spec이나 표준을 가지고 있다. 이것이 기존의 SoC가 특정 애플리케이션에만 한정적으로 적용될 수 있고 그 애플리케이션의 종류가 늘어나면 과도하게 방대한 포트폴리오를 확보해야 하는 이유이다. 만약 각 sub part들의 IP를 공통의 인터페이스로 연결하고 필요에 따라 추가, 변경, 대체, 제거할 수 있는 유연한 플랫폼을 제시하면 사용자는 마치 레고 블럭을 조립하듯 필요한 기능들을 조합해 SoC를 구성할 수 있고 IoT 생태계를 현실화하는 일이 보다 용이해질 것이다. 애플리케이션 개발자들은 SoC를 직접 설계하지 않고도 자신들이 원하는 기능을 갖춘 SoC를 확보할 수 있고 Chip maker는 적은 물량의 개별 SoC들을 하나의 플랫폼으로 묶어 생산함으로써 유의미한 규모를 창출할 수 있다. 이러한 선순환의 과정은 어떤 의미에서 라즈베리 파이나 아두이노와 같이 오픈소스를 기반으로 한 개발자 보드 마이크로 컨트롤러와 닮아있다. 즉, 유연하고 확장 및 변경이 용이한 플랫폼을 제공하여 수많은 개발자들을 끌어들이고 사용 사례를 늘려나가며 하나의 생태계를 형성하는 것이다.

IoT 시장에서의 System IC 반도체의 방향성은 여전히 수많은 논의가 존재하며 그 누구도 명확한 비전을 제시하지 못 하고 있다. 혹자는 SoC를, 또 다른 이는 SiP를 말한다. 누군가는 공통 플랫폼을, 다른 누군가는 다양한 포트폴리오를 논한다. 하지만 거꾸로 이러한 현상은 IoT의 무한한 잠재력과 가능성에 대한 암시이기도 하다. 한 가지 확실한 것은 애플이 앱스토어를 만들어 개발자들을 끌어들일 때, 구글이 범 안드로이드 진영을 구축할 때, TSMC가 Foundry 사업모델로 Design house들의 해방자를 자처할 때 자신들이 모든 것을 다 할 수 있다는 자만에 빠져 있던 기업들은 지금 시장에서 사라졌거나 생존의 기로에서 몸부림치고 있다는 것이다. 세계적인 네트워크 장비 회사인 시스코는 IoT를 IoE (Internet of Everything)라고 부른다. 모든 것이 연결되는 세상이라는 뜻이다. 아무리 위대한 기업도 혼자서 모든 것을 잘 할 수는 없다. IoT 시대에는 그 어느 때보다 개방성이 중요한 성공 요소가 될 것이며 그것은 System IC 반도체 분야에서도 마찬가지이다. 멀고 험한 길을 갈 때 친구는 많을수록 좋은 법이다.

• 외부 필진 기고의 논조는 IDEC 방향과 다를 수 있습니다.



스마트카를 위한 모바일 기기에서의 영상처리 기술



서론

최근 자동차와 IT 융합기술의 연구 개발이 가속화되며 '스마트카' 시대가 도래하고 있다.아직은 스마트카가 보편화되지 못한 상태에서 국내 자동차 수는 지속적으로 증가 추세에 있고 그와 함께 자동차 교통사고 수와 그로 인한 부상자 및 사망자의 수도 역시 증가하고 있다.이러한 상황에서 운전자의 주행을 보조하여 사고위험으로부터 안전성을 부여하는 주행보조시스템(ADAS: Advanced Driver Assistance System)에 대한 필요성이 더욱 부각되고 있다.

이와 동시에 이동통신 기술을 자동차에 접목하여 주행 중 차량 내 정보를 통합 관리하고,운전자가 차 안에서 오락, 정보 등 다양한 컨텐츠를 즐길 수 있도록 하는 카-엔터테인먼트에 관련된 연구도 많이 진행되고 있다.이 중에서도 모바일과 자동차의 커넥티비티를 이용하여 상기의 기술들을 누릴 수 있도록 하는 연구가 관심을 받고 있다.스마트폰을 이용하는 사람들이 증가하면서 많은 사람들이 보유하고 있는 모바일과 자동차의 커넥티비티를 이용한다면 스마트카에서 필요한 3G, Wi-Fi 등의 네트워크 기능과 그에 따른 기존의 다양한 컨텐츠를 그대로 누릴 수 있기 때문이다.

이러한 모바일 기술은 몇 년 새 비약적인 발전을 이루고 있다.특히, 그림 1에서 제시하고 있는 것처럼,모바일의 두뇌라 할 수 있는 AP(Application Processor)의 계산능력 역시 상당한 발전을 이루었다.AP가 데스크톱컴퓨터의 계산능력을 따라오게 되면서 모바일의 효용성이 커졌고,많은 계산 자원이 필요한 영상처리 및 컴퓨터 비전 기술이 구현 가능하게 되어AR(Augmented Reality) 및 HCI(Human-Computer Interaction) 관련 연구도 많이 진행되고 있다.하지만 모바일에 장착되는 디스플레이 장치 및 카메라,다양한 센서들이 집약된 MEMS(Micro Electro Mechanical System),그리고 통신 모듈이 함께 발전[1]하여 처리해야 할 복잡도와 데이터량이 급수적으로 증가하기 때문에 고해상도 영상에서 컴퓨터비전 알고리즘을 CPU만으로 실시간 처리하는 것은 거의 불가능하다고 할 수 있다. 따라서 모바일 GPU역시GPGPU(General Purpose Graphics Processing Unit)로 이용함으로써 영상처리를 병렬화 및 고속화하여 실시간 영상처리가 가능하도록 하는 연구가 활발히 이루어지고 있다[2,3,4].

본론

모바일에서 영상처리를 할 수 있는 방법은 기본적으로 CPU를 이용한 방법과 GPU를 이용한 방법으로 나뉜다.모바일 AP의 CPU는 GPU에 비하여 클럭 주파수가 높기 때문에 처리속도는 빠르다는 장점이 있지만 전력이 많이 소모된다는 단점이 발생한다.반면, 모바일 GPU는 저전력 구조를 통하여 설계된 더 많은 수의 코어들로 동작할 수 있다는 장점이 존재한다[5].안드로이드OS에서는CPU를 이용한 영상처리를 주로 JavaCV나 OpenCV를 이용하여 수행하게 된다.인하대학교 컴퓨터비전 연구실에서는 영상처리에서 일반적으로 사용하는 OpenCV를 이용하여 GPU를 이용한 병렬영상처리와의 비교에 신뢰도를 높였다. OpenCV는 Open Source Computer Vision Library의 약자로 Intel에서 개발하여 공개한 오픈소스 기반의 컴퓨터비전 라이브러리인데,기본적인 영상처리 함수부터 고등의 영상처리 함수까지 다양한 알고리즘을 구현하여 제공하고 있다.또한,대표적인 OS 환경에 맞추어 미리 컴파일된 라이브러리를 제공할 뿐만 아니라 원본 소스코드 또한 제공하기 때문에 사용자의 환경에 맞추어 빌드하여 사용할 수 있다.

모바일 GPU를 이용한 병렬영상처리를 수행하기 위해서는 GPU에서 지원하는 공용 API를 이용하거나 모바일 GPU 제조사에서 제공하는 라이브러리를 사용해야 한다.현재 많은 개발자들은 하나의 산업표준인 OpenGL ES를 이용하여 GPU 프로그래밍을 하고 있다[6].또한 2013년 상반기부터 모바일 AP에서 OpenCL을 지원하기 시작하면서 이에 대한 연구도 진행되고 있다[7].인하대학교 컴퓨터비전 연구실에서는 OpenGL ES 2.0과 OpenCL 1.1 각각에 대한 구동 프레임워크를 구현하였고 몇몇의 기본적인 함수들과 GPU의 성능평가에 대표적인 SIFT와 SURF 특징점 검출 함수에 대하여 OpenCV를 벤치마크하여 구현 및 결과를 비교해보았다. OpenGL ES는 3D 그래픽 API를 제공하는 오픈 라이브러리인 OpenGL의 임베디드 버전으로, OpenGL이 워크스테이션이나 고성능 PC와 같은 환경에서 3D 영상을 렌더링하는 것을 목표로 하는 것과는 달리 적은 용량의 메모리와 낮은 속도의 CPU를 갖춘 임베디드 환경에 최적화되어 있다.이전의 그래픽 파이프라인은 프로그래머가 변경할 수 없었지만 GLSL(openGL Shading Language)이 출시됨에 따라 프로그래밍이 가능한 GPU를 사용할 수 있게 되었다.GLSL을 이용하면 그래픽 파이프라인 중 정점 셰이더(Vertex shader)와 프래그먼트 셰이더(Fragment shader)를 임의로 수정할 수 있는데, 이를 이용하면각각의 화소 값을 동시에 연산하여 GPU에서의 병렬연산이 가능하다. 그림 2는 OpenGL ES 2.0을 이용한 병렬영상처리를 위한 프레임워크를 도식화하고 있다. 1) 영상(카메라/파일)이 입력되면 2) 입력 영상을 텍스처 메모리에 저장하고,3) 텍스처 메모리를 FBO(Frame Buffer Object)로 지정한 후 4) 정점/프래그먼트 셰이더를 통한 병렬처리를 수행하여 5) 마지막으로 처리 결과가 저장된 텍스처를 화면에 출력하게 된다. OpenGL ES 2.0 파이프라인을 효과적으로 이용하기 위하여 입력영상을 GPU 메모리에 텍스처 형태로 저장하여 GPU에서의 고속 접근을 가능하도록 하였고, 결과가 저장된 FBO의 텍스처를 그대로 디스플레이 장치에 출력하는 구조를 통하여 프로세서간 자료이동을 최소화하였다.또한, FBO를 패스 간에 연속적으로 이용하여 GLSL을 통하여 계산된 결과를 프레임 버퍼에 저장함으로써 GPU와 CPU간 자료이동으로 인한 성능저하를 방지할 수 있다. 하지만 OpenGL ES의 파이프라인 구조는 셰이더 간 독립성으로 인해 동시적으로 발생한 결과를 서로 공유할 수 없고,셰이더 당 32bit(RGBa)결과만 출력할 수 있다는 제약조건을 가지고 있다.이 때문에 다중 텍스처를 이용하고 다중 패스로 셰이더를 설계하여야 한다.따라서 알고리즘에 따라 한 개의 동작을 여러 개의 단계로 나누어 동작하도록 커널을 구현해야 하므로 개발의 비효율성을 갖고 있다.

OpenCL은 개방형 범용 병렬 컴퓨팅 프레임워크로서 CPU, GPU, DSP 등의 다중 프로세서를 동시에 사용하는 heterogeneous 시스템에서 병렬 프로그램 개발을 도와주는 표준 라이브러리이자 일종의 API라 할 수 있다.상용 모바일의 경우, ARM의 Mali-T6xx 시리즈, Qualcomm의 Adreno 3xx 시리즈 등 최근에 다수의 AP 제공자들이 OpenCL 1.1을 지원하는 모바일 GPU를 생산하면서 모바일에서도 OpenCL을 이용한 병렬처리가 이슈화되고 있다. OpenCL은 앞의 OpenGL ES의 텍스처와 FBO 그리고 렌더 버퍼와 같은 데이터들을 공유할 수 있어서, 이를 이용하면 OpenGLES의 제약 사항인 고속 공유 메모리와 외부 메모리를 추가로 사용하는 것이 가능하며,메모리 관리 모델을 제공하여 병렬 컴퓨팅 과정에서의 메모리 사용의 효율성을 높일 수 있다.

OpenCL 1.1을 이용하기 위한 프레임워크는 그림 3에 도식화하였다. OpenCL 1.1 라이브러리는 계산 관련 API로서 실제 영상처리 부분만을 위하여 동작하고,입/출력은 OpenGL ES 2.0의 프레임워크와 동일하게 동작한다. 1) 영상이 입력되면 2) 입력 영상을 OpenGL 타입의 텍스처로 메모리에 저장하고 3) GL_Sharing 관련 함수로 메모리 객체로 지정하여 4) OpenCL 커널에 구현된 일련의 코드로 병렬처리를 수행한다. 5) 마지막으로 결과를 OpenGL 타입의 텍스처로 변환 후 바로 화면에 출력한다.일반적으로 모바일 환경에서는 CPU와 GPU가 동일한 하나의 메모리를 다른 구역으로 구분하여 각각 활용하지만, OpenGL 1.1에서는 위의 메모리 객체를 CPU와 GPU에서 모두 접근이 가능하도록 도와준다. 즉, CPU와 GPU 간 자료이동에 대한 제약에서 벗어날 수 있는 것이다.

Category	Algorithm	OpenCV (ms)	OpenGL (ms)	SpeedUp
Image Filtering	boxFilter	331.5	17.7	×18.7
	buildPyramid	65.6	7.9	×3.5
	Dilate	41.5	16.4	×2.5
	GaussianBlur	219.7	20.8	×10.6
	Laplacian	60.1	18.5	×3.2
	medianBlur	469.1	17.1	×27.5
	Sobel	61.8	15.6	×4.0
Geometry Transform	warpAffine	70.6	9.6	×7.4
	warpPerspective	107.1	10.6	×10.1
Pixel Transform	cvtColor	49.8	8.9	×5.6
	threshold	14.2	8.1	×1.8
Histogram	calcHist	19.4	77.3	×0.3
	calcBackProject	28.7	9.9	×2.9
Feature Detection	Canny	326.4	20.2	×16.2
	cornerHarris	278.0	22.7	×12.3
Robust Feature Detection	SIFT	1147.9	229.6	×5.0
	SURF	965.5	179.1	×5.4

표1. OpenCV 및 OpenGL ES 2.0 구현 함수 속도 비교

표 1은 CPU를 이용하는 OpenCV4Android의 영상처리 함수들의 수행결과와 GPU를 이용하는 OpenGL ES 2.0으로 구현된 병렬영상처리 함수들의 수행속도 결과를 비교해보았다.각 알고리즘에 따라 2~27배 정도의 처리속도 개선을 확인할 수 있었으나, calcHist 함수처럼 전체 화소에 대해 각 화소 값의 빈도수를 누적하는 경우 병렬처리의 결과가 더 느릴 수 밖에 없었다. 하나의 화소를 담당하는 각각의 셰이더가 독립적으로 동작하며 공유 메모리에 값을 저장할 수 없기 때문에 셰이더를 반복하여 동작하도록 구현하였기 때문이다.

Category	Algorithm	OpenGL (ms)	OpenCL (ms)	SpeedUp
Image Filtering	boxFilter	17.7	16.3	×1.1
	buildPyramid	7.9	9.3	×0.9
	Dilate	16.4	17.6	×0.8
	GaussianBlur	20.8	16.5	×1.2
	Laplacian	18.5	17.2	×1.2
	medianBlur	17.1	18.2	×0.9
	Sobel	15.6	14.6	×1.1
Geometry Transform	warpAffine	9.6	21.8	×0.4
	warpPerspective	10.6	22.7	×0.5
Pixel Transform	cvtColor	8.9	8.7	×1.0
	threshold	8.1	8.3	×1.0
Histogram	calcHist	77.3	48.1	×1.6
	calcBackProject	9.9	10.2	×1.0
Feature Detection	Canny	20.2	20.6	×1.0
	cornerHarris	22.7	20.1	×1.1
Robust Feature Detection	SIFT	229.6	212.2	×1.1
	SURF	179.1	214.1	×0.8

표2. OpenGL ES 2.0 및 OpenCL 1.1 구현 함수 속도 비교

다음 표 2에서는 GPU를 이용하는 OpenGL ES 2.0과 OpenGL 1.1로 구현된 병렬영상처리 함수들의 처리 속도를 비교하여 나타내었다.비교적 간단하며 화소들 간의 의존성이 낮은 함수들에 대해서는 동작이 다르지 않아 성능차이를 거의 보이지 않았으나,영상의 기하학적 처리와 관련된 함수에서는 OpenGL ES 2.0을 이용하는 것이 더 효과적인 것을 확인할 수 있었다.이는 OpenGL ES 2.0으로 구현된 함수에 정점 셰이더를 이용한 텍스처 매핑 기법을 사용하였기 때문이다. OpenGL은 3D 렌더링을 목적으로 하기 때문에 영상의 기하학적 처리 부분에서 효과적임을 알 수 있다.한편, OpenCL 1.1을 이용하면 전역 메모리와 워크그룹 내부에서 공용으로 사용 가능한 로컬 메모리를 활용할 수 있고, 각 쓰레드에서 동일 주소의 메모리 값을 참조 시 발생할 수 있는 문제를 해결하는 atomic 함수들을 이용할 수 있다.따라서 각 화소 간의 종속성 문제에 대하여 더 자유로울 수 있기 때문에 관련함수에 대하여 더 구현하기 용이하였고 처리 속도도 빠른 것을 볼 수 있었다.

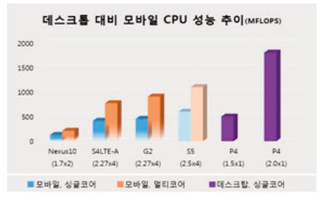


그림1. 모바일CPU의 성능 추이

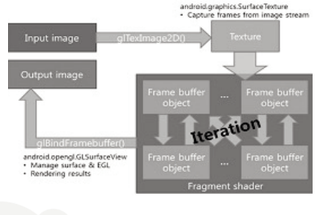


그림2. OpenGL ES 2.0 기반 병렬영상처리 프레임워크

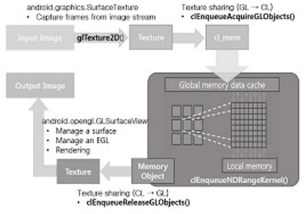


그림3. OpenCL 1.1 기반 병렬영상처리 프레임워크

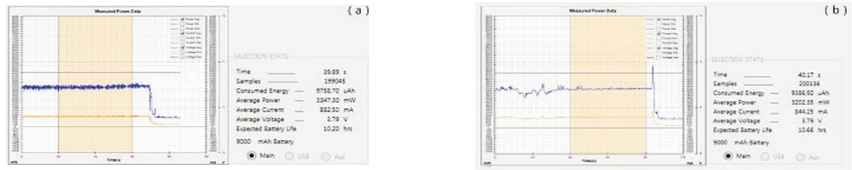
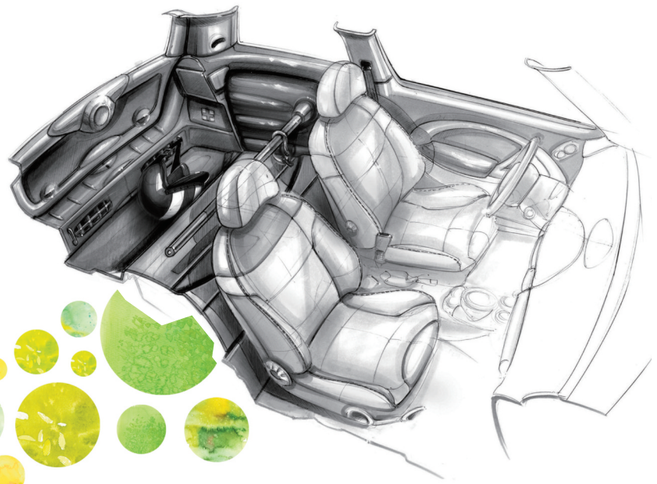


그림4. (a) CPU와 (b) GPU에서 Viola-Jones 차량 검출기 수행 시 소모전력량 측정 결과

이렇게 모바일에서 GPU를 이용한 연산은 CPU에서의연산보다 더 빠른 처리속도를 보인다는 것을 알 수 있다.하지만 한가지 더 짚어보고 가야할 문제가 바로 소모전력 문제이다.그래서 본 연구실에서는 OpenCV의 Viola-Jones 기반 차량 검출 어플리케이션과 직접 구현한 OpenCL 1.1 기반 Viola-Jones 차량 검출 어플리케이션의 소모전력을 비교해보았다.그 그래프가 그림 4에 나타나있고,노란색으로 강조된 안정적인 40초 구간 내에서 소모 전력량을 계산했다. 그 결과 표 3에서 볼 수 있듯이 GPU를 이용한 병렬연산처리 시에 비교적 더 적은 전력을 소모한다는 사실을 확인할 수 있었다.이는 20초 간의 소모 전력으로 매우 낮아 보일 수 있지만 일반적인 운전 시 수십 분에서 수시간 동안 구동을 해야 하므로 소모전력량 차이는 커질 수 밖에 없다.그림 5는CPU와 GPU기반에서의 차량 검출 결과이다.이외에도 차선검출,보행자검출 등 많은 ADAS 관련 어플리케이션을 모바일로 구현할 수 있다.



결론

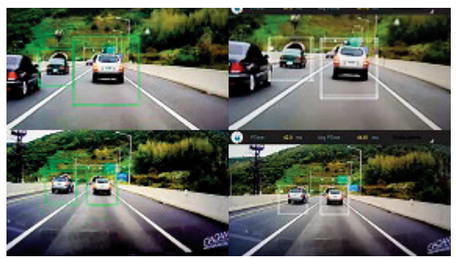
최근 스마트폰의 카메라 성능이 유효 화소 4100(Lumia 1020)만에 이르는 등 고화질의 영상을 취득하는 데는 문제가 없게 되었지만 그만큼 영상처리를 하는 데는 처리 속도 문제가 발생할 수 밖에 없게 되었다.

표3. CPU와 GPU에서 Viola-Jones 차량 검출기 수행 시

Computer	Average Power	Average Current	Consumed Energy
CPU	3347.30mW	882.50mA	9758.70uAh
GPU	3202.35mW	844.25mA	9386.92uAh

소모전력량 측정 결과

그림5. CPU(왼쪽)와 GPU(오른쪽) 기반의 차량 검출 결과



따라서 고속의 병렬 영상처리가 필요하게 되었고, 이제는 모바일에서도 CPU뿐만이 아닌 GPU도 적극적으로 활용해야 할 때가 왔다. 또한,기존의 스마트폰을 이용하여 ADAS를 구축하면 별도의 장비가 필요하지 않아 비용을 최소화할 수 있다는데 큰 장점이 있다.

김 학 일 교수
소속 : 인하대학교 정보통신공학과
연구분야 : 패턴인식, 컴퓨터비전, 바이오인식
E-mail : hikim@inha.ac.kr
Homepage : http://vision.inha.ac.kr

Reference

[1] N. Bhas (2012), MEMS in Mobile: Smartphones, Tablets, eReaders & Ultrabooks,U.K.Hampshire:Juniper Research.

[2] I. Park, M. Lee, Y. Choi, "Trends of Computer Vision on Embedded platform,"Journal of the Institute of Electronics Engineers of Korea, vol.39, pp.157-164, 2012.

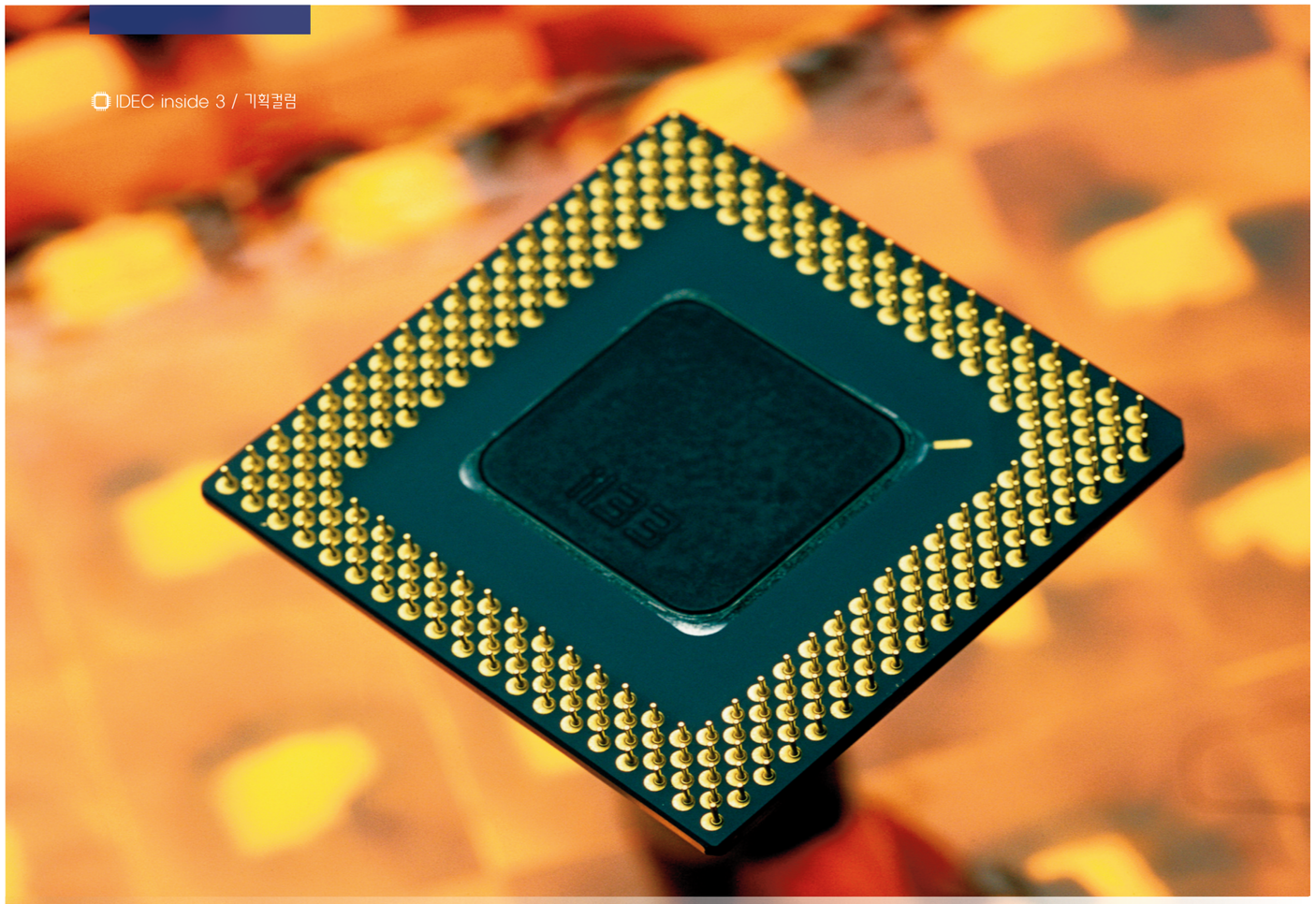
[3] N. Singhal, J. Yoo, H. Choi, I. Park, "Implementation and optimization of imageprocessing algorithms on embedded GPU," IEICE transaction on Information andSystems, vol.95, pp.1475-1484, 2012.

[4] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn, T. Purcell,"A Survey of General-Purpose Computation on Graphics Hardware," Proc. OfComputer Graphics Forum, no.26, pp.80-113, 2007.

[5] 이환용, "핸드폰에서의 그래픽스 및 플랫폼 기술 동향," 정보과학회지, vol.29,pp.55-62, 2011.

[6] A. Munshi, OpenGL ES 2.0 Programming Guide, Addison-Wesley, 2009.

[7]A. Munshi, OpenCL Programming Guide, Addison-Wesley, 2012.



Ansys사 PowerArtist-XP

A. 목적 : 칩 설계 초기 단계에서의 Power Management 및 Estimation

B. 구분 : Ansys의 PowerArtist-XP는 Design Engineer or Verification Engineer에게 Power Management Based Debugging Solution및 Low Power Design 가이드를 제공

C. Supported Platform and O/S System

- Solaris (64bit) 8,9,10
- RedHat 7,8,9
- Red Hat Enterprise (64bit) Linux 3,4,5
- SuSE (SLES 9/10) (64bit) Linux

D. 특성 및 기능

Apache Subsidiary of Ansys사의 PowerArtist는 반도체 설계 Process의 Early Stage인 RTL 설계 단계에서 Chip이 소모할 파워를 예측하는 기능을 가지고 있으며, 예측된 파워를 기반으로 파워 누수 부분과 수정 가능 포인트를 알려줌으로써 Front-End 기반 설계에서의 전력관리 및 저전력 설계가 가능하도록 돕는 획기적인 툴이다. PowerArtist는 RTL 설계 단계에서 실제로 소모될 IC의 전력을 예측할 수 있다. IC 기능 동작 검증을 위하여 만들어진 시나리오를 바탕으로 실제 IC의 동작환경을 통해 소비전력을 유추하고 평균 전력량 및 순시적 전력 변동량을 측정하여 초기 설계단계에서부터 IC 소비 전력을 관리할 수 있도록 도와준다.

PowerArtist 특성 및 성능

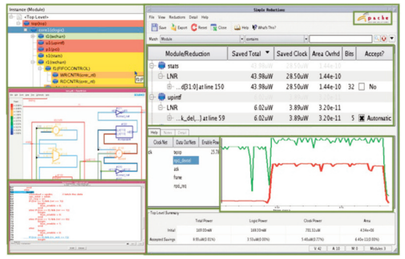


그림1. PowerArtist GUI

평균전력량 측정(Average Power Calculation)

FSDB 나 VCD 등의 Activity Waveform을 기반으로 해당 시간 구간 동안의 평균 전력을 각각의 컴포넌트, 인스턴스 단위로 파워를 계산한다.

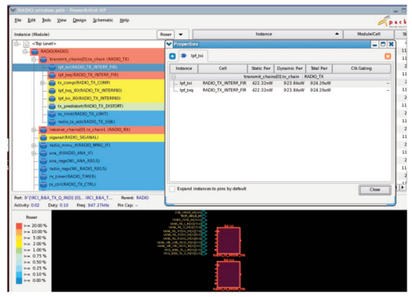


그림2. Average Power Calculation 결과 및 전력소모가 큰 핫스팟 표시

Time Based Calculation

Activity Waveform 파일의 시나리오를 바탕으로 순시적인 전력의 변동량을 측정하여, 최대소비 전력(Peak Power)을 측정하고 기능블록이 시나리오대로 동작하는지를 직관적으로 파악할 수 있도록 도와준다.

그림3. Time Based Calculation 결과

Power Reduction

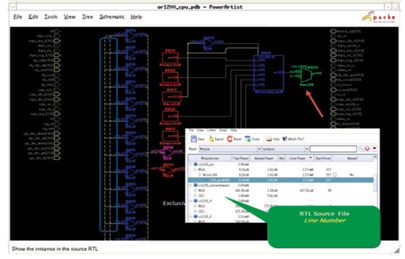


그림4. Power Reduction

GUI를 통해 전력소모량과 전력누수 부분, 그리고 이에 관계된 분석을 위한 여러 가지 파라미터(Activity, Frequency, Capacitance, Clock Gating Efficiency 등)들과 Gate Level의 Schematic을 보여주고, RTL 코드상에서 전력 세이브를 위한 수정 포인트를 짚어주어 디자이너가 직접 저전력설계를 할 수 있도록 안내 해준다.

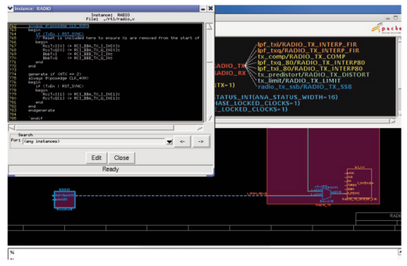


그림5. 저전력 디자인 가이드

ANSYS Apache

회 사 명 : Apache Design Solutions Inc.
(Subsidiary of Ansys Inc.)
웹 주 소 : www.apache-da.com
한국지사 : Ansys Korea Co., Ltd.
전 화 : 02-3441-5000
주 소 : 23F CityAir Tower 159-9 Samsung-dong, Gangnam-gu, Seoul, 135-973, Korea

디지털 시스템의 MPW 설계 방법에 대한 이해 (4)

Auto Place and Route

서론

최근 심각해지는 반도체 업계의 인력난과 어려움에 반하여 대학 및 관련 분야에서는 숙달된 고급 인력의 지속적인 양성이 힘들어 둘 사이에 장벽이 존재하고 있다. 이는 축적된 노하우를 바탕으로 지속적인 칩 제작을 하면서 고급인재를 양성하는 대학 및 연구실이 많지 않다는 것을 뜻하며, 칩을 만들고자 하는 대학은 HOW-TO 문서를 그때그때 검색하고 짧은 기간에 임기응변식의 칩 제작을 한다는 뜻이기도 하다. 실제 MPW 참여자들의 경우 정기적으로 참여하는 팀과는 대조적으로 MPW 참여와 동시에 구글링부터 시작하는 사람들이 많다. 칩 설계 고급 인력 양성을 위해서는 시간과 노력이 많이 들어가야 하므로 직관적이고 정확한 이론 및 실습자료가 매우 필요하다. 지금까지 "SoC 시스템과 IDEC MPW Flow 소개" 와 "FPGA 검증부터 Synopsys Design Compiler를 이용한 합성", 그리고 "Equivalence Check 와 Static Timing Analysis" 에 대해 알아보았다. 이번 호에는 "Auto Place and Route" 에 대해서 소개 하고자 한다.

본론

◆ Auto Place and Route

지난 호에 Design_Compiler를 이용하여 RTL Code를 설계자가 디자인하고자 하는 사이즈, 타이밍 조건에 따라 gate-level netlist로 변환하는 작업을 수행하였고, Formality와 PrimeTime을 통해 검증을 했다(2014년 5월호 참조). 검증된 gate-level netlist와 Design constraint를 이용하여 공정사가 마스크를 제작할 수 있도록 산업계 표준인 GDSII 포맷으로 변환을 해주어야 한다. 이 GDSII 포맷의 내용은 gate-level netlist에 기술되었던 Logic들을 칩 사이즈와 타이밍에 맞도록 위치하고 배선된 실제적인 Physical Data정보가 포함되어 있다. 오늘날에는 수많은 Logic들을 타이밍, 사이즈, 전력, 신호 무결성 등에 따라, 배치 및 배선을 자동으로 수행할 수 있는 Auto P&R Tool들이 개발되어 있다. 따라서 Auto P&R Tool 중 Synopsys사의 IC Compiler를 이용한 배치 및 배선 기법에 대해 살펴보고자 한다.

◆ General IC Compiler Flow

기본적인 IC Compiler Flow에 따라, IDEC MPW의 Auto P&R 방법을 설명하겠다.

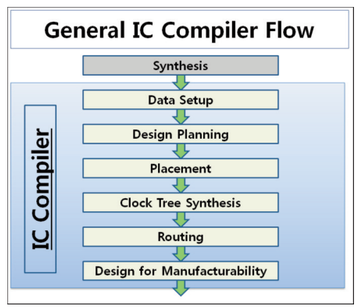


그림1. IC Compiler Flow

1. Data Setup

IC Compiler(이하 ICC)는 초기 입력 데이터로, gate-level netlist, Design constraints, Physical and logical libraries, 공정 프로세스 데이터를 이용하여 GDSII 포맷을 생성하게 된다. 이 때문에 ICC를 실행하기 전에 아래의 데이터 파일을 준비해야만 한다.

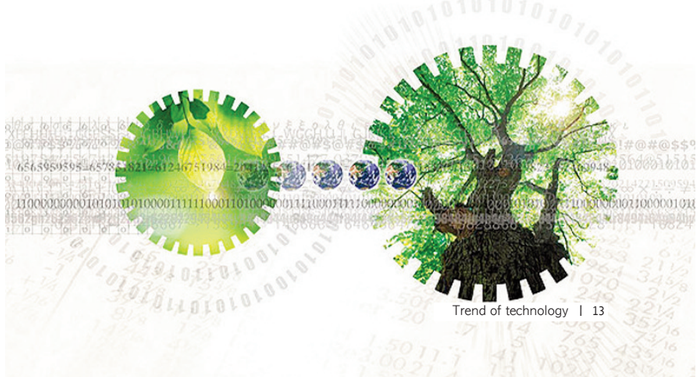
Data	Format	Generator	내용
netlist	.v or .ddc	DC	gate-level netlist
constraint	.sdc or .ddc	DC	Design constraint
logical libraries	.db	공정사	I/O, STD, MEM cell에 대한 logical 정보
Physical libraries	Milkyway	공정사	I/O, STD, MEM cell에 대한 Physical 정보
Process data	.tf, .tluplus, .map	공정사	metal layer에 대한 공정사 Parameters
etc	.tdf	설계자 or IDEC	I/O 또는 Pin 배치 정보

표1. 입력 데이터

표1 입력데이터를 Auto P&R을 실행할 수 있도록 위치 및 파라미터 값을 ICC에 정의해주어야 하는데, 이러한 작업이 Data Setup이다. ICC를 실행하기 전에 위의 입력데이터 및 파라미터를 정의하기 위한 파일을 만들어 두면, 이후 작업을 수월하게 할 수 있을 것이다. 먼저, 공정사에서 제공하는 Physical and logical libraries, 공정 프로세스 데이터의 위치를 지정한다.

```
## Common_Setup.tcl
A. lappend search_path  ~(path)/STD ~(path)/IO ~(path)/MEM
B. set_app_var target_library     " STD_ss.db STD_ff.db "
C. set_app_var link_library " * $target_library IO.db MEM.db"
D. set MW_REFERENCE_LIB_DIR "~(path)/STD ~(path)/IO ~(path)/MEM"
E. TECH_FILE "~(path)/tech_file.tf"
F. set TLUPLUS_MAX_FILE   "~(path)/worst.tluplus"
G. set TLUPLUS_MIN_FILE   "~(path)/best.tluplus"
H. set TLUPLUS_MAP_FILE   "~(path)/tluplus.map"
```

그림2. 공정 데이터 설정

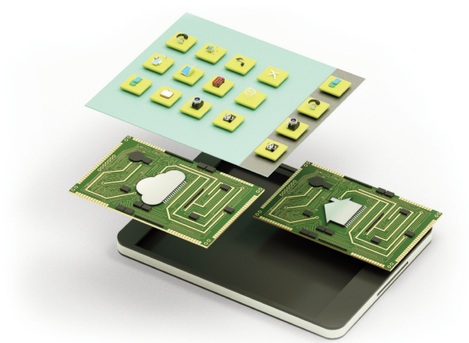


A 행은 target/link library에 사용될 logical library들의 디렉터리를 지정한다. B 행은 P&R 과정에서 timing에 따른 cell들의 변환을 수행하는 라이브러리를 target library로 지정해야 한다. 즉, IO와 MEM가 아닌 STD cell library를 지정한다. C 행은 target library와 함께 Netlist에 기록된 모든 cell의 라이브러리가 포함되도록 한다. D 행은 이미 layout이 된 STD, IO, MEM 등 Physical library들을 지정한다. E~H를 통해 Process data를 지정한다. 그림 2를 통해 공정 데이터 설정이 완료되었으면, 설계 디자인을 위한 데이터 설정을 한다.

```
## User_Design_Setup.tcl
A. set DESIGN_LIB_NAME    "pad_soc_uart_fifo"
B. set NETLIST_FILE       "~(path)/verilog.v or ~(path)/ddc"
C. set NETLIST_FORMAT     verilog | ddc
D. set LOGIC_TOP_MODULE   "TOP"
E. set MW_POWER_NET       "VDD"
F. set MW_GROUND_NET      "VSS"
G. set MW_POWER_PORT      "VDD"
H. set MW_GROUND_PORT     "VSS"
```

그림3. 디자인 데이터 설정

A 행은 설계자가 만들고자 하는 Layout과 Physical Cell 정보를 포함하기 위한 디자인 라이브러리 이름을 선언한다. B~D 행은 합성을 통해 만들어진 netlist의 위치, format, netlist에서 Top module을 지정한다. E~F 행은 Layout에서 그리고자 하는 Power net 이름을 선언하고, G~H 행은 STD, IO 등 각 cell의 power pin을 기입한다. 다음은 Multiple Corners and Multiple Modes (이하 MCMM)를 작성한다. 저 전력 IC에서는 라이브러리 모델, 전압, 인터커넥트 코너 등 잠재적으로 상충하는 전력 및 타이밍 요구사항을 분석하고 최적화하는 게 중요하다. 따라서 단계별 시나리오를 작성하여 MCMM에 최적화한 Layout을 진행할 수 있도록 한다.



```
## MCMM.tcl
A. create_scenario Scenario_name
B. set_app_var auto_link_disable true
C. set_scenario_options -setup true -hold false -leakage_power false
D. read_sdc SDC_file_name.sdc
E. set_tlu_plus_files \
      -max_tluplus $TLUPLUS_MAX_FILE \
      -tech2itf_map $TLUPLUS_MAP_FILE
F. set_operating_conditions \
      -analysis_type on_chip_variation \
      -max slow      -max_lib STD_ss \
G. set_timing_derate -early 0.88
H. set_case_analysis 0 scan_en
I. set_case_analysis 0 test_mode
J. set_app_var auto_link_disable false
```

그림4. 시나리오 설정

A 행은 새로운 시나리오를 만들기 위해 시나리오 이름을 선언한다. B 행과 J 행은 각 시나리오에 대해서 netlist의 연결되는 것을 방지하고, 디자인 최적화를 하기 전에 연결하므로 data setup 시간을 줄일 수 있기 때문에 사용하기를 권장한다. C 행은 시나리오 옵션을 제한하기 위한 명령어이다. 수행하고자 하는 옵션을 true와 false로 제한할 수 있다. D 행에서 SDC 파일 읽어 들인다. 위 예에서, E~G까지는 코너, H~I는 모드에 대한 부분이다. 먼저, TLU+ 파일을 선언한다. 위의 예에서는 max, min의 operating condition이 하나의 모델로 사용되기 때문에 TLU+ 파일도 하나만 사용하였다. 만일, 서로 다른 condition에 이용할 수 있는 TLU+파일이 있다면, -min_tluplus 옵션을 사용하면 된다. F 행은 operation condition을 선언하고, G 행을 통해 timing_derate 값을 준다. H~I 행은 사용하지 않는 모드에 대해서 수행하지 않도록 하였다. 위의 예는 Function Test를 위한 시나리오로 코너 및 모드에 대한 명령어 및 세팅방법은 많다. 자세한 사항은 Synopsys사의 solvnet이나 ICC 매뉴얼을 통해 알 수 있다. CTS, scan Test 등 필요한 시나리오는 위의 예와 solvnet을 참조하기 바란다.

Auto P&R을 위한 입력 파일이 준비되었으면, ICC를 Tool을 실행하기 위한 환경 설정을 해야 한다.

```
setenv ICC /tools/synopsys/ICC/H-2013.03-SP2   -> ICC가 설치된 경로
setenv SNPSLMD_LICENSE_FILE 27000@ls2idec      -> 라이선스 서버의 포트 및 IP주소
set path = ($path $ICC/bin)                    -> ICC bin폴더 위치
```

그림5. IC Compiler 환경 설정

ICC의 환경 설정 파일의 sourcing을 통해 ICC를 구동할 수 있으며 command mode와 GUI mode에서 실행할 수 있다.

```
icc_shell        -> command mode 실행
icc_shell -gui   -> GUI mode 실행
```

그림6. IC Compiler 실행 명령어

ICC 실행 창에 그림2와 그림 30에서 설정한 데이터 파일을 실행한다.

```
icc_shell> source Common_Setup.tcl
icc_shell> source User_Design_Setup.tcl
```

그림7. 설정 데이터 파일을 sourcing

Data-Setup의 첫 번째 단계로, 설정된 데이터 파일들을 layout을 위해 Milkyway 데이터베이스 구조의 design library를 생성한다.

```
icc_shell> create_mw_lib $DESIGN_LIB_NAME -open -technology
$TECH_FILE -mw_reference_library $MW_REFERENCE_LIB_DIR
```

그림8. Design library 생성

생성된 라이브러리에 합성을 통해 만들어진 Netlist를 읽고, Design Cel을 생성한다.

```
icc_shell> import_designs -format $NETLIST_FORMAT $NETLIST_FILE \
           -top $LOGIC_TOP_MODULE
```

그림9. netlist loading 및 design cell 생성

그림 4에서 작성한 시나리오 설정 파일을 읽는다.

```
icc_shell> source MCMM.tcl
```

그림10. 시나리오 sourcing

다음으로 Netlist에는 Power/Ground(이하 P/G) net에 대한 정의가 되어 있지 않다. 따라서 각 cell의 P/G pin을 연결하기 위한 net을 설정한다.

```
icc_shell> derive_pg_connection -power_net $MW_POWER_NET \
      -power_pin $MW_POWER_PORT –ground_net $MW_GROUND_NET \
      -ground_pin $MW_GROUND_PORT
icc_shell> derive_pg_connection -power_net $MW_POWER_NET \
      -ground_net $MW_GROUND_NET -tie
icc_shell> check_mv_design -power_nets
```

그림11. P/G connection 설정

위의 두 번째 명령어는 어떤 input pin이 high 또는 low로 고정된 경우, P/G net에서 input pin에 직접 연결될 수 있도록 해준다. 만일 tie-cell을 사용한다면 이 옵션은 사용하지 마라. P/G 연결이 잘되었는지 check_mv_design 명령을 통해 확인한다. ICC는 많은 변수와 명령어를 통해 timing과 최적화를 할 수 있다. 적용된 변수와 명령어는 ICC를 실행한 시점에만 적용되며, ICC를 재실행 시에는 다시 변수와 명령어를 실행해야만 한다. 따라서 변수와 명령어를 위한 스크립트를 작성한 후 재실행 시 load 하는 방향으로 진행하기를 권장한다.

```
## common_optimization_setting.tcl
A. set_host_options –max_cores 8
B. set_app_var timing_enable_multiple_clocks_per_reg true
C. set_fix_multiple_port_nets –all -buffer_constants
D. set_auto_disable_drc_nets –constant false
E. set_app_var enable_recovery_removal_arcs true
```

그림12. 최적화 설정

A 행의 명령을 통해 설계자 머신의 멀티코어를 사용함으로써, placement와 routing 시 런타임을 줄일 수 있다. B 행은 멀티 클럭을 사용할 경우 타이밍 해석이 가능하도록 한다. 이 경우 false path에 대한 선언도 필요하다. (ex: set_clock_groups -logically_exclusive -group C1 -group C2). 포트별로 독립적인 드라이버를 지정하는 것이 좋은 디자인으로써 C 행을 수행한다. D 행은 placement 단계에선 Clock net과 tie high or tie low net 이외의 모든 net은 버퍼를 삽입한다. tie-cell net에 버퍼를 삽입하지 않으면, crosstalk에 취약하므로 tie-cell net에도 버퍼를 삽입할 수 있도록 한다. E 행을 통해 reset과 같은 asynchronous signal간의 Recovery & Removal timing을 최적화한다. 각 행의 결과는 아래 그림을 참조하기 바라며, 이외의 디자인 최적화를 위한 명령어와 변수는 ICC 매뉴얼을 참고하기 바란다.

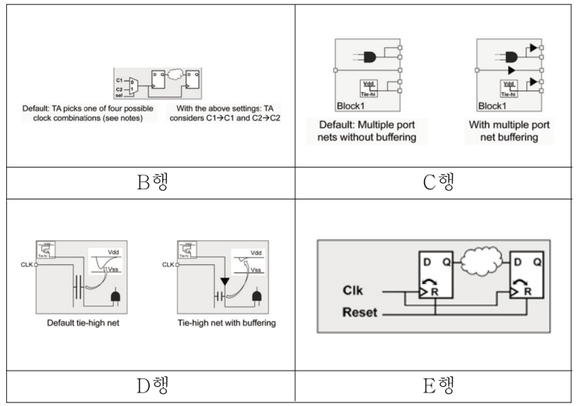


그림13. 최적화

다음으로, 설계자는 Placement를 진행하기 전에 zero-interconnect timing 체크를 통해 디자인이 over-constraint에 대해 체크하는 게 중요하다.

```
icc_shell> set_zero_interconnect_delay_mode true
icc_shell> report_constraint -all
icc_shell> report_timing
icc_shell> set_zero_interconnect_delay_mode false
```

그림14. zero-interconnect timing 체크

report_constraint와 report_timing을 통해 모든 path에서 positive slack이나 일부분 small negative slack 결과가 나오면 다음을 진행해도 된다. 그러나 large negative slack이 발생하면, 이후 ICC에서는 긍정적인 결과를 얻기 어려워서 다시 합성을 진행하는 것이 올바른 선택이다. 그리고 체크를 수행한 후 zero_interconnect_delay_mode를 false로 수정하는 것을 잊지 말아야 한다.

DC에서 reset, enable, select 시그널에 대해선, ideal network를 선언하여 버퍼 생성을 방지하였다. 이러한 시그널에 대해 placement 단계에서 버퍼를 삽입할 수 있도록 ideal network constraint를 제거해야 한다.

```
icc_shell> remove_ideal_network [get_ports "Enable Select Reset"]
```

그림15. ideal net 제거

Date Setup의 마지막 단계로, 지금까지 수행한 내용을 저장한다. 이후 Placement, route 등 단계별 작업을 수행 후 저장하는 습관을 통해 잘못된 단계를 진행 시, 앞선 단계로 돌아갈 수 있으므로, 작업시간을 효과적으로 사용할 수 있을 것이다.

```
icc_shell> save_mw_cel -as Data_Setup
```

그림16. Cell 저장

2. Design Planning

Design Planning은 칩의 사이즈 및 형태 정의, I/O pin과 macro의 위치 및 Power network를 형성하는 단계이다. 칩 사이즈 및 형태를 정의한다.

```
icc_shell> create_floorplan -control_type boundary \
           -start_first_row -flip_first_row -left_io2core 1355 \
           -bottom_io2core 1355 -right_io2core 1355 -top_io2core 1355
```

그림17. Chip 형태 설정

control_type은 세 종류가 있다. Aspect ratio 옵션은 전체 디자인의 width와 height의 비율로 결정된다. 만일 1이면 정사각 형태를 취하게 된다. boundary 옵션은 전체 디자인의 크기를 미리 지정할 경우 사용된다. (ex : set_die_area –coordinate {0 0 4292 3806}) width_and_height 옵션은 core의 width와 height를 각각 지정할 경우 사용된다. 또한, 각 cell의 행을 위치하는 방법 및 core와 io와의 거리를 지정할 수 있다. 위 사항이 모든 지정된 후 실행 결과에서 Core Utilization이 0.6 정도를 권장한다. 향후 optimization을 수행할 경우 buffer 또는 net의 삽입 시 여유를 두기 위함이다.

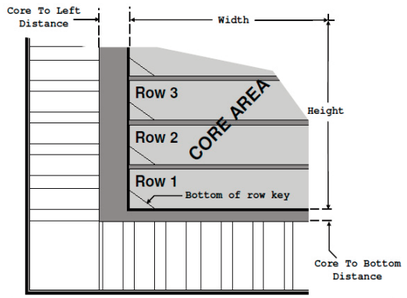


그림18. Parameter 예

공정에서 제공되는 Layer 중 설계자가 routing layer를 지정할 수 있다.

```
icc_shell> set_ignored_layers –max_routing_layer M5
icc_shell> report_ignored_layers
```

그림19. routing layer 지정

위의 예는 M1~M5까지 Layer를 routing layer로 지정하였다.
다음으로 Macro를 위치시키는 방법이다. 설계자가 직접 Macro의 위치를 지정하는 방법과 ICC가 자동으로 위치를 지정하는 방법이 있다. 설계자는 ICC의 Toolbar의 버튼을 이용하여 macro의 위치와 방향을 지정할 수 있다.

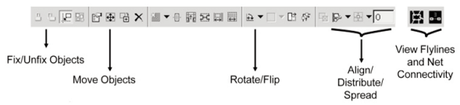


그림20. macro 위치 및 방향 지정

ICC를 이용한 macro의 auto placement를 진행하기 위해서는 ICC에 사전정보를 입력해주어야 한다.

```
icc_shell> set_fp_placement_strategy -min_distance_between_macros 10
icc_shell> set_fp_placement_strategy -sliver_size 12
icc_shell> report_fp_placement_strategy
```

그림21. macro placement를 위한 설정

위의 명령어는 macro 간 최소한의 거리를 지정하고 macro 사이가 좁은 영역일 경우 STD cell이 위치하지 못하도록 설정하므로 써 congestion을 줄인다. 위에서 설정한 정보를 바탕으로 STD cell과 Macro에 대한 Virtual Flat Placement를 수행할 수 있다.

```
icc_shell> create_fp_placement –timing_driven
icc_shell> route_zrt_global -congestion_map_only true -exploration true
```

그림22. Virtual Flat Placement

create_fp_placement 통해 Placement를 할 수 있다.
또한, route_zrt_global을 통해 일정 영역에서 이용 가능한 routing track 수와 global route에 필요한 signal net의 수를 비교할 수 있는 congestion map을 보여줌으로써 route 진행 여부를 판단할 수 있다. 보통 overflow가 10개 이상이거나, GRC overflow가 2% 이상이면, SI, 타이밍 저하, route가 불가능한 결과를 초래할 수 있다.

```
phase1. Routing result:
phase1. Both Dirs: Overflow =    3621  Max = 13 GRCs =   2247 (2.73%)
```

그림23. congestion 결과

위의 예는 route_zrt_global을 실행한 후 log의 결과이다. congestion map에서 "19/6" 이라는 결과를 발생했을 경우, 위의 예처럼 Max overflow가 13이 나올 것이다. 13개는 route가 어려울 수도 있다는 걸을 알려준다. 또한, 2.73%의 결과는 crosstalk 문제나 타이밍 저하의 결과가 발생할 수 있다. 위와 같은 결과 발생 시에는 Chip의 형태나 사이즈의 확인과 수정이 필요할 것이다. 예를 들어 core의 utilization이 너무 높다라면, core 사이즈를 넓혀주고, 세로축에 congestion이 많이 발생하면, 가로축을 core 사이즈를 넓혀주고, 반대로 가로축이 많이 발생하면, 세로축의 core사이즈를 넓혀주는 식으로 chip의 사이즈와 형태를 변경해주어야 할 것이다. 또한, macro와 macro 사이에서 좁아 병목현상이 발생하는 경우 macro 사이를 넓혀주어야 한다.

Virtual Flat Placement 후 congestion에 문제가 없으면, power network를 수행하기 위해 macro를 고정해야 한다. macro의 corner나 edge에는 in/out pin들이 많아 이 부분에서 congestion이 발생할 수 있다. 따라서 사전에 macro 주위에 blockage를 그려줌으로써 문제를 해결할 수 있다. blockage는 Hard blockage와 Soft blockage가 있다. Hard blockage는 standard & macro cell이 배치되는 것을 방지한다. Soft blockage는 standard & macro cell이 배치되는 것을 방지하지만, optimization 과정에서 필요한 buffer 등은 배치될 수 있다.

```
icc_shell> set_keepout_margin -type hard RAM5 -outer {10 10 10 10}
icc_shell> create_placement_blockage –bbox {345 355 392 400} -name A –type hard
icc_shell> set_dont_touch_placement [all_macro_cells]
```

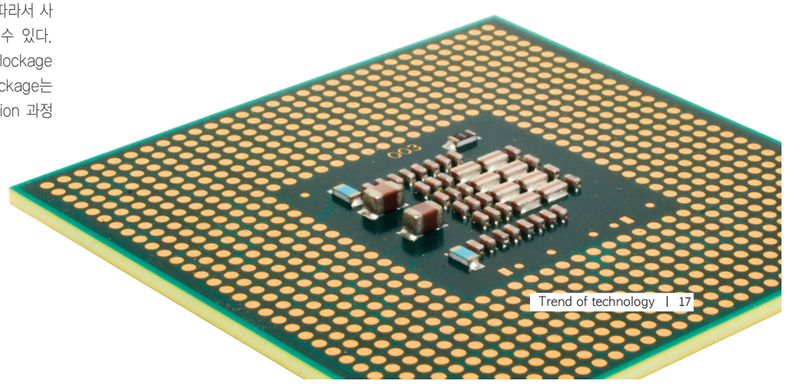
그림24. macro blockage 및 고정

set_keepout_margin의 예는 Hard blockage 형태로 RAM5라는 macro에 대해서 left, bottom, right, top에 대해서 10에 margin으로 모든 cell의 배치를 방지한다. 만일, RAM 5 macro를 움직이면 keepout margin도 같이 움직인다. create_placement_blockage도 hard type의 blockage가 생성되지만, 이것은 –bbox 영역에 대해서 cell 배치를 방지한다. 즉 macro instance가 움직였을 때, –bbox 영역은 움직이지 않는다. 모든 congestion 문제가 해결된 후에는, macro가 움직이는 것을 방지하기 위해 모든 macro를 고정하는 set_dont_touch_placement 명령을 사용한다.

다음으로 각 셀에 Power를 공급하기 위해, 앞선 내용에서 derive_pg_connection 명령어를 통해서 선언된 power net을 생성해야 한다. Power net을 형성하는 방법은 여러 방법이 있으며, 디자인에 따라 생성하는 조건도 많다. 이에 대한 자세한 내용은 solvnet 또는 design documentary를 확인하기 바라며, 한 가지 예를 설명하도록 하겠다.

```
icc_shell> create_rectangular_rings -nets {VDD VSS} -left_offset 1110 \
           -left_segment_layer M4 -left_segment_width 100 –right_offset 1100 \
           -right_segment_layer M4 -right_segment_width 100 –bottom_offset \
           1100 -bottom_segment_layer M3 –bottom_segment_width 100 \
           -top_offset 1100 -top_segment_layer M3 –top_segment_width 100
icc_shell> set_fp_rail_region_constraints -polygon \
  {{415.110 3356.420} {3850.470 3356.420} {3850.470 438.090} {415.110 438.090}}
icc_shell> set_fp_rail_constraints -add_layer -layer M5 –direction horizontal \
           -max_strap 128 -min_strap 16 -max_width 30 –min_width 0.280 \
           -spacing minimum
icc_shell> set_fp_rail_constraints -add_layer -layer M4 –direction vertical \
           -max_strap 128 -min_strap 16 -max_width 30 –min_width 0.280 \
           -spacing minimum
icc_shell> set_fp_rail_constraints -skip_ring -extend_strap core_ring
icc_shell> synthesize_fp_rail -nets {VDD VSS} -voltage_supply 1.98 \
           -synthesize_power_plan -power_budget 350 –pad_masters
           { PVDD_IO PVSS_IO }
icc_shell> commit_fp_rail
```

그림25. Power network 생성



create_rectangular_rings을 통해 Core나 macro 등 특정영역 주위에 Power ring을 생성하도록 net에 대한 layer, width, offset 등을 설정한다. 특정영역에 strap을 생성하기 위해 set_fp_rail_region_constraints 명령을 이용하여 영역을 지정한 후 set_fp_rail_constraints 통해 strap에 대한 조건을 지정한다. synthesize_fp_rail을 통해서 Power Synthesize를 수행한 후 IR-drop heat map을 확인한다. 문제가 없으면, commit_fp_rail 명령어를 실행하여 Power network를 생성한다.

생성된 Power ring과 strap에 Cell들의 Power를 연결한다.

```
icc_shell> preroute_instances
icc_shell> preroute_standard_cells -fill_empty_rows -remove_floating_pieces
```

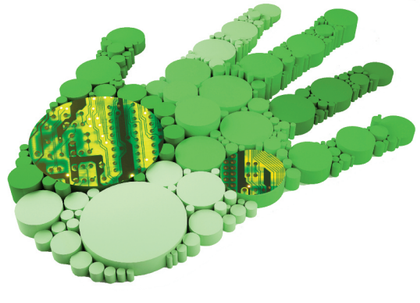
그림26. P/G connect

preroute_instances를 이용하여 macro, PAD 등, P/G pin을 Power ring과 Strap에 연결한다. preroute_standard_cells을 이용하여 STD cell을 연결한다. Power network가 생성된 후, STD cell이 Power net과 DRC가 발생하지 않도록 Power Net Placement Blockage를 한다.

```
icc_shell> set_pnet_options -partial "M2 M3"
icc_shell> set_pnet_options -complete "M2 M3"
icc_shell> legalize_fp_placement
icc_shell> verify_pg_nets
```

그림27. P/G verify

대부분 디자인이 P/G Strap으로 메탈 4 이상을 사용한다. 또한, 대부분의 STD cell의 pin은 메탈 1로 되어 있어서, congestion 문제가 발생하지 않는 선에서 Cell의 밀집도를 높여 chip 사이즈 줄이는 목적으로 Blockage는 수행하지 않는다. 만일 메탈 2와 메탈 3으로 P/G Net을 구성한 경우, DRC violation을 방지하기 위해 P/G Net 아래에 STD Cell을 일부 배치하거나 완전히 차단할 수 있다. 첫 번째는 M2와 M3의 P/G Net 아래에 STD Cell을 배치할 수 있고 두 번째는 STD Cell이 위치되는 것을 방지한다. P/G Net에 대한 Blockage 선언을 한 후 STD Cell을 이동시키기 위해 legalize_fp_placement 명령을 실행한다. verify_pg_nets를 이용하여 Power net의 연결이 잘 되었는지 확인한다.



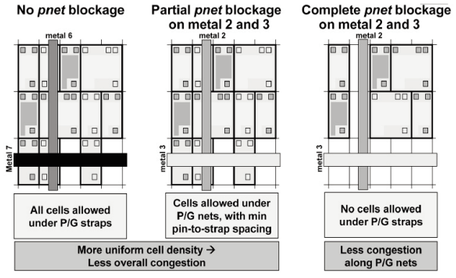


그림28. Power net blockage에 따른 Placement

congestion에 문제가 없는지 확인한다.

```
icc_shell> route_zrt_global -congestion_map_only true -exploration true
```

그림29. Congestion 확인

CTS 이전에는 Clock Signal은 ideal 하게 다루어진다. 그러나 "clock/data" 가 혼용된 Signal은 timing이나 DRC에 맞추기 위해 Placement 단계에서 버퍼가 삽입될 수 있다. CTS 이전에 버퍼삽입을 방지하도록 한다.

```
icc_shell> set_ideal_network [all_fanout -flat -clock_tree]
```

그림30. ideal clock tree 설정

set_ideal_network 명령을 통해 Placement 단계에서 "clock/data" signal에 버퍼 삽입을 방지한다.

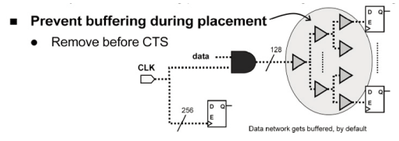


그림31. "clock/data" signal 버퍼 삽입 방지

다음으로 timing 분석을 한다.

```
icc_shell> optimize_fp_timing -hfs_only
icc_shell> report_qor
icc_shell> report_constraint -max_delay -all_violators \
           -scenarios [all_active_scenarios ]
icc_shell> optimize_fp_timing
icc_shell> optimize_fp_timing -effort high
```

그림32. In-Place Optimization

timing 분석하기 전에 먼저, optimize_fp_timing -hfs_only를 통해 High Fanout Synthesis를 수행한다. 그 후 report_qor과 report_constraint 명령을 통해 timing 분석을 한다. timing을 만족시키지 못한다면 optimize_fp_timing, optimize_fp_timing -effort high를 차례로 진행한다.

3. Placement

Placement 단계에서는 Timing과 congestion을 분석하여 Cell 배치를 최적화하는 단계며 Design Planning 단계에서 pre_place된 디자인을 Placement & Optimization을 진행한다.

```
Place_opt
- area_recovery   timing과 congestion에 따라 logic optimization을 수행하는 명령어
                  non-critical 경로에 대해서 buffer removal, cell down-sizing할 수 있는 옵션
- power           leakage & dynamic power를 최적화 하는 옵션
- congestion      추가적인 congestion 알고리즘 실행 옵션
```

그림33. Placement 최적화 옵션

위의 설명에 따라 설계자가 사용하고자 하는 옵션을 실행하면 된다.

```
icc_shell> place_opt -area_recovery -power -congestion
```

그림34. Placement 최적화

place_opt는 기본적으로 setup time 문제만 해결할 수 있고 hold time은 CTS 후에 최적화를 한다. 다음으로 place_opt 수행한 후 congestion과 timing을 분석한다.

```
icc_shell> route_zrt_global -cong true -exploration true   #congestion분석
icc_shell> report_constraint -max_delay -all_violators \
           -scenario [all_active_scenarios]                #timing 분석
```

그림35. congestion 및 timing 분석

place_opt를 수행 후에도 timing이나 congestion 문제가 남았다면 psynopt 명령을 실행한다.

```
icc_shell> psynopt -area_recovery -power -congestion
```

그림36. Incremental Logic Optimization

4. CTS

CTS(Clock Tree Synthesis)는 preroute와 postroute data의 상관관계 및 blockage를 고려하여 clock net을 최적화하는 단계이다. Clock Tree의 최적화는 버퍼크기 조정 및 재배치, 게이트 크기 조정 및 재배치 그리고 delay insertion 등을 수행함으로써 Clock skew와 Clock insertion delay를 향상시킨다.

Clock Tree는 CTS(Clock tree synthesis)와 CTO(Clock tree optimization) 단계로 나뉜다. 먼저, CTS는 설계 디자인의 constraint(transition delay, load capacitance, fanout, buffer level)에 충족하도록, 버퍼 삽입을 통해 clock tree의 균형을 잡도록 한다. CTO는 CTS를 통해 형성된 Clock Tree에 버퍼 추가 삽입, 재배치, 사이즈 조정을 통해 Skew를 최소화한다.

CTS 전에 CTS를 위한 세팅을 수행한다.

```
icc_shell> remove_ideal_network [all_fanout -flat -clock_tree]
icc_shell> set_delay_calculation_options -routed_clock arnoldi
icc_shell> set_app_var cto_enable_drc_fixing true
icc_shell> set_clock_tree_options -target_skew 0.2
```

그림37. CTS 설정

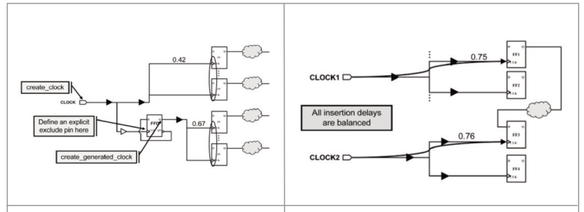
Clock net에 버퍼를 삽입할 수 있도록 ideal network를 제거한다. 또한, CTO 단계까지 완료되면, ICC의 Default delay model인 Elmore 대신에 더욱 정확한 arnoldi model이 사용되도록 세팅한다. CTO 단계에서 tran/cap violation을 수정할 수 있도록 세팅한다. CTS는 default로 skew와 insertion delay를 0ns로 설정되어 있다. runtime과 전체 버퍼수를 줄이기 위해 clock의 skew target을 완화하여 세팅하는 것을 권장한다.

Design에 따라 아래의 옵션을 사용할 수 있다.

```
icc_shell> set_inter_clock_delay_options -honor_sdc true
icc_shell> set_clock_tree_exceptions -exclude_pins [get_pins FFD/CLK]
icc_shell> set_inter_clock_delay_options -balance_group "CLOCK1 CLOCK2"
icc_shell> clock_opt -inter_clock_balance
```

그림38. CTS를 위한 Design 설정

CTS는 기본적으로 SDC에 정의된 network latencies를 준수하지 않는다. 하지만 설계자가 SDC의 network latencies를 매칭하기 위해 필요하다면 set_inter_clock_delay_options을 설정해준다. generated_clock이 Master_clock에 독립적이라면 두 번째 행과 같이 설정해준다. 기본적인 CTS는 inter-clock skew의 균형을 잡아주지 않으므로 세 번째 항과 같이 balance_group을 세팅해준다.



exclude_pin | balance_group

그림39. CTS를 위한 Design 설정 예

CTS를 수행한다.

```
icc_shell> clock_opt -no_clock_route –only_cts -inter_clock_balance
icc_shell> report_clock_tree –summary –setting
icc_shell> report_clock_timing –type skew
icc_shell> foreach $scn [all_scenarios ] {
            current_scenario $scn
            set_fix_hold [all_clocks] }
icc_shell> extract_rc
icc_shell> clock_opt -only_psyn -no_clock_route -power -area_recovery
```

그림40. CTS 실행

clock_opt 명령어를 실행하여 CTS 후 report_clock_tree와 report_clock_timing을 통해 CTS 결과를 확인한다. 그리고 CTS를 실행한 후 hold_time도 체크하기 위해 활성화한다.

또한, timing 최적화를 위해 이전까지는 virtual route extraction을 사용했다면, 다음 스텝 동안에 global route로 실행하기 위해 extract_rc 명령을 실행한다. 그리고 clock_opt -only_psyn 명령을 통해 최적화를 수행한다. CTO를 수행한 후 위의 예제에서 report_clock* 명령을 사용하여 결과를 확인한다.

5. Routing

CTS가 된 후에는 배치된 Cell들을 design의 timing과 공정사의 design rule에 맞게 배선을 하는 작업이다.

Route를 진행하기 위한 Route option을 세팅해준다.

```
icc_shell> set_delay_calculation_options -postroute arnoldi
icc_shell> set_route_zrt_common_options \
            -post_detail_route_redundant_via_insertion medium
icc_shell> set_route_zrt_global_options \
            -timing_driven true –crosstalk_driven true
icc_shell> set_route_zrt_track_options -timing_driven true –crosstalk_driven true
icc_shell> set_route_zrt_detail_options –optimize_wire_via_effort_level medium
icc_shell> source ~(path)/antenna_rules.tcl
icc_shell > set_route_zrt_detail_options –antenna true
icc_shell > set_si_options –delta_delay true –static_noise true \
            -static_noise_threshold_above_low 0.25 \
            -static_noise_threshold_below_high 0.25 \
            -max_transition_mode total_slew –timing_window true
icc_shell > set_si_options –route_xtalk_prevention true \
        -route_xtalk_prevention_threshold 0.25
icc_shell > set_route_opt_zrt_crosstalk_options -hold true -transition true
```

그림41. Route 설정

각각의 route option 다양하다. 본고에서는 기본적으로 많이 사용하는 option만 기록되어 있음을 참고하기 바란다. postroute delay mode도 CTO와 마찬가지로 arnoldi로 설정한다. contact code를 좀 더 좋게 하기 위해 single via를 multiple via로 변경할 수 있도록 해준다. global and track route option은 timing과 crosstalk을 고려하도록 설정한다. route common options로 redundant via 삽입을 설정했으면 detail option에 via와 wire를 최적화할 수 있도록 설정한다. 또한, 공정 진행 과정에서 배선의 길이가 긴 경우에는 gate oxide가 손상되는 경우가 발생한다. 따라서 공정사에서는 이러한 문제가 발생하지 않도록, antenna rule을 배포한다. antenna 문제가 되는 wire의 길이를 조정 또는 diode 삽입을 할 수 있도록 antenna rule을 ICC로 읽어 들인 후 route 시 antenna rule을 고려할 수 있도록 세팅한다. 또한 crosstalk을 방지 또는 감소시키기 위해 설정을 한다.

설정이 완료된 후 route를 시작한다.

```
icc_shell> route_zrt_group -all_clock_nets –reuse_existing_global_route true
icc_shell> route_opt -initial_route_only
icc_shell> route_opt -skip_initial_route -xtalk_reduction -power
```

그림42. initial route

route는 signal net을 실행 전에 clock net을 최초에 route 한다. clock net에 대한 route가 완료되었으면, signal net에 대해서, global, track assignment, detail route를 하기 위해 route_opt -initial_route_only 명령을 사용한다. initial route를 실행 후 timing, tran/cap, Physical DRC violation이 발생했다면, 3행처럼 post-route optimization을 수행한다. post_route 시 hold나 cell size 등 부분적인 최적화가 필요하다면 route_opt -help 명령을 통해 부족한 부분의 최적화 명령을 찾아 수행하면 된다.

post-route가 완료된 후 Physical DRC 체크 및 수정을 한다.

```
icc_shell> verify_zrt_route
icc_shell> route_zrt_detail –incremental true
```

그림43. Physical DRC 체크

verify_zrt_route를 통해 DRC의 이상 유무 체크를 한 후, violation이 발생했다면 2행을 통해 수정할 수 있다.

6. DFM

DFM은 Chip을 제작과정 중에 발생할 수 있는 issue를 Design 단계에서 고려하여 미연에 방지하는 목적이 있다.

Antenna rule 위반에 대해서 수정한다.

```
icc_shell> set_route_zrt_detail_options –antenna true \
            -insert_diodes_during_routing true –diode_libcell_names \
            diode_cell_name
icc_shell> route_zrt_detail –incremental true
```

그림44. Antenna diode 삽입

Detail route 단계에서 metal jumping을 통해 antenna rule 위반을 수정하였지만, 여전히 수정되지 않은 부분이 있다면, diode를 삽입하여 수정할 수 있다. 먼저 1행과 같이 diode cell을 삽입할 수 있도록 설정한 후, 2행과 같이 명령을 실행하면 된다.

Cell들의 빈 공간에 Filler Cell을 삽입한다.

```
icc_shell> insert_stdcell_filler –cell_with_metal "fillCap64 fillCap32" \
        -respect_keepout –connect_to_power VDD –connect_to_ground VSS
icc_shell> insert_stdcell_filler –cell_without_metal "fill64 fill32" \
        -respect_keepout –connect_to_power VDD –connect_to_ground VSS
icc_shell> insert_pad_filler –cell "IOFILLER60 IOFILLER10"
```

그림45. Filler Cell 삽입

Design에서 cell들이 위치하지 않은 공간은 filler cell 삽입을 통해, 칩의 density, DRC 해결 및 완벽한 P/G net을 형성할 수 있다. 먼저 core에 1행과 같이 metal이 포함된 filler cell을 삽입한 후 남아 있는 공간에 2행과 같이 metal이 포함되지 않은 filler cell을 삽입한다. 다음으로 pad filler도 삽입하여 I/O의 P/G ring이 형성될 수 있도록 한다.

Timing 최적화를 수행한다.

```
icc_shell> report_constraint –all_violators –scenarios [all_active_scenarios]
icc_shell> route_opt –incremental –size_only
```

그림46. Timing 최적화

diode와 filler cell 삽입 중에 timing과 DRC violation이 발생할 수 있다. 따라서 cell sizing을 통해 timing을 최적화한다. 먼저 violation을 확인 후에 2행과 같이 최적화를 수행한다.

DRC 및 LVS를 체크한다.

```
icc_shell> verify_zrt_route
icc_shell> verify_lvs
```

그림47. DRC 및 LVS 체크

netlist와 GDS 파일을 생성한다.

```
icc_shell> change_names -rule verilog -hier
icc_shell> write_verilog -no_unconnected_cells –no_flip_chip_bump_cells \
            -no_corner_pad_cells –no_pad_filler_cells –no_tap_cells \
            -no_core_filler_cells Verilog_out.v
icc_shell > write_stream -format gds -lib_name LIB –cells {CHIP_FINISH_cell }
            CHIP_FINISH.gds
```

그림48. netlist 및 GDS 추출

Layout에 이상이 없으면 STA 및 simulation을 위한 netlist와 공정사에 제출할 GDS 파일을 생성해야 한다. netlist를 생성하기 전에 change_name 명령을 통해 설계자가 define 한 rule 또는 ICC의 default_rule에 맞추어 netlist를 생성하도록 설정한다. write_verilog 명령을 통해 netlist를 생성하되 filler나 spare cell 등 timing이 없는 Physical cell 등은 생성되지 않도록 한다. 그 후 write_stream 명령을 통해 gds를 추출한다. ICC에 사용된 I/O, STD, MACRO cell들은 auto P&R을 수행하는데 필요한 기본적인 Layer 정보만 포함되어 있을 수 있다. 따라서 real layer가 전부 포함된 Physical cell로 변환 과정이 필요할 수도 있다.

Physical cell 변환은 일반적으로 virtuosos와 같은 custom layout tool 등을 많이 사용한다. 또한, ICC에서 추출된 netlist를 통해 지난 호에 다루었던, PrimeTime, Formality를 통해 STA나 Equivalence check를 수행해야 한다.

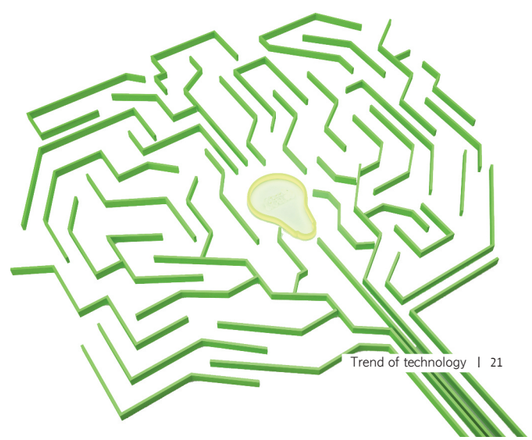
결론

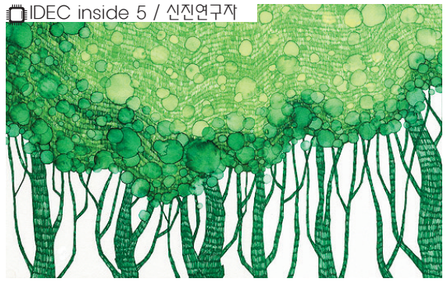
이상으로 Auto Place and Route에 대해 간략히 알아보았다. 본 고에서 기재된 내용은 일반적인 ICC Flow로, 위의 내용에 더불어 ICC에 대한 option등을 추가하여 더 좋은 Design을 설계할 수 있을 것이다.

유 은 광 선임연구원
소속 : 반도체설계교육센터
E-mail : yuk0428@idec.or.kr
http://www.idec.or.kr

참고 문헌

IC Compiler 1 Workshop Student Guide
Solvnet.synopsys.com





나무를 보지 말고 '숲'을 보라

선택이라는 문제에 접근하는 경로는 다양할 것이다. 새로운 시각의 접근도 중요하지만 내가 관심을 두고 있는 분야라면 더욱 다각적이고 깊이 파악할 수 있을 것이다. 전자공학과를 다니던 학부 시절 휴대전화가 보편화하는 모습 속에 통신 시스템에 관심을 두고 Microwave IC를 전공하게 된 민병욱 교수를 만나 보았다.

"우리 생활 곳곳에 자리 잡을 연구분야"

민병욱 교수는 현재, 연세대학교 전기전자공학부 조교수로 재직 중이다. 그는 microwave 회로를 Si chip 위에서 설계하여 하나의 통신/레이더 시스템을 만드는 Microwave integrated circuit 연구를 하고 있다. 현재, 그가 연구하고 있는 Microwave IC는 학위 중에 하던 연구의 연장선상으로 지금은 passive microwave circuit과 active millimeter-wave IC 두 가지 주제로 연구가 진행 중이다고 한다. 차세대 통신을 위한 밀리미터파(Millimeter) IC가 요즘 많은 관심을 받고 있는 실정이라, 상용화가 언제쯤 이뤄질지는 모르지만, Microwave IC는 기술이 발전하면 우리 생활 곳곳에 자리 잡을 분야 중 하나일 것이다.

"관심으로 부터의 출발."

그는 전자공학을 전공하던 학부 시절 휴대전화 보급의 활성화를 보고 통신 시스템에 관심을 두게 되었다고 한다. "전자공학을 전공하면서 electronic circuit에 관심을 두게 되었고 통신 시스템과 함께 공부하다 보니 자연스럽게 Microwave IC를 전공하게 되었습니다." 민병욱 교수는 서울대에서 학사를 마치고 미시간대학교 (University of Michigan)에서 석/박사 학위를 받았다. 석/박사 과정 중에는 〈Si based millimeter-wave phased array system〉이란 연구주제로 CMOS switch 기반의 phase shifter, attenuator 등과 SiGe BJT 기반의 amplifier들을 설계하여 Si Chip 위에서 millimeter-wave phased array system을 개발했다고 한다. "밤새며 tape-out을 준비했던 그 시간이 고생이었던 같아요. 그래도 첫 IC가 내 생각대로 동작했을 때의 짜릿함은 잊을 수 없습니다." 석/박사를 마친 그는 퀄컴(Qualcomm)에서 3년 정도를 근무하다가 지금의 연세대학교로 오게 되었다.

"상호협력은 연구자가 가져야할 중용한 자세"

공부에 지치거나 연구를 진행하면서 만나는 어려움은 누구나 다를 것이다. 이에 민병욱 교수는 시간과의 싸움을 꼽았다. "tape-out이라는 시간 제약 아래서 연구가 진행되다 보니까 더욱 그런 거 같습니다. 완벽한 설계보다는 시간에 맞춘 설계가 많은 것이 안타까워요." 라면서 "아마 Engineering이라는 학문 자체가 trade-off가 기본 바탕이라 생각이 됩니다. 결국, 개인시간을 줄여 설계에 투자할수록 더 좋은 결과가 나오는 거 같아요." 민병욱 교수는 복잡한 Engineering 세계에서 연구자가 가져야 할 자세로 상호협력을 강조했다. "맡은 임무를 성실히 수행해 내는 것이 가장 중요합니다. 그건 꼭 Engineer가 아니더라도 맡은 임무를 성실히 수행하는 것은 사회 구성원의 중요한 덕목이라고 생각하니까요."

"우리 생활의 일부분이 된 반도체"

반도체는 이제 우리 생활의 일부분이라면서 앞으로는 다품종 소량생산의 반도체 시대 즉, 소비자의 다양한 취향을 만족시키는 반도체가 필요한 시대가 도래할 것으로 전망한 민병욱 교수는 우리의 발전방향도 이런 흐름에 맞춰야 한다고 언급했다. 또한, 전공에 대한 완벽한 이해와 함께 더 큰 그림을 그릴 수 있는 식견을 후배 연구원들이나 학생들이 갖기를 바랐다. "저를 포함해서, 반도체 회로 설계를 하다 보면 큰 그림을 잊어버리기 작은 일에 매몰될 수 있는 거 같아요." 라면서 "좀 더 큰 시스템, 나아가 사회를 보는 식견도 필요한 거 같습니다." 지금까지 회로의 설계기술을 높이는 것에 중점을 두었지만, 이제는 전체 시스템을 새롭게 꾸며 좋은 결과를 내고 싶고 Emerging device를 이용한 Microwave circuit를 만들고 싶다는 그의 모습에서 자신의 분야를 서서히 개척해 나가는 열정이 고스란히 전해져왔다.

민 병 욱 교수
연세대학교 전기전자공학부

문의 연세대학교 전기전자공학부 MICS연구실 민병욱 교수
전화 02-2123-5880 E-mail bmin@yonsei.ac.kr
Homepage web.yonsei.ac.kr/mics