논문 2012-49SD-4-3

K-FPGA 패브릭 구조의 평가 툴킷

(Evaluation Toolkit for K-FPGA Fabric Architectures)

김 교 선*

(Kyosun Kim)

요 약

FPGA용 CAD툴에 대한 학계의 연구는 상용 FPGA에 적용하기에는 단순하고 비효율적인 아키텍처를 가정하고 있기 때문에 실용성 측면에서 뒤처져 왔다. 최근 상용 FPGA 아키텍처의 배치 위치 및 배선 그래프 데이터베이스를 구축하고 인터페이스를 제공함으로써 상용 FPGA에 적용할 수 있는 배치 배선 툴의 개발을 가능하게 하려는 시도가 있었다. 본 논문은 신규 FPGA 아키텍처로 개발되고 있는 K-FPGA의 경쟁력을 벤치마킹 할 수 있는 툴킷 개발에 대해 기술한다. 이는 학계 CAD 툴의 실용성 한계를 한층 더 확장하고 있다. 기존 상용 툴과 매핑, 패킹, 배치, 배선 각 단계 별로 데이터를 교환할 수 있어 세부 툴별 비교 평가가 가능하며 이전 단계의 결과물을 기다리거나 결과의 질에 영향을 받지 않으면서 각 단계를 독립적으로 개발할 수 있는 체계를 구축하였다. 또한, 상용 FPGA의 아키텍처를 추출하여 단위 셀 라이브러리를 구축함으로써 FPGA 아키텍처의 신규 개발 시 참조 설계 역할을 할 뿐만 아니라 상시 벤치마킹 환경을 제공하도록 하였다. 특히, 아키텍처 정보를 툴 내에 하드 코딩하지 않고 하드웨어 설계자에게 익숙한 표준 HDL 형식으로 기술하여 읽어 들일 수 있도록 함으로써 아키텍처에 수시로 다양한 변경을 시도하면서 최적화해도 툴이 유연하게 수용할 수 있는 데이터 구동 방식의 툴 개발을 추구하였다. 실험을 통해 단위 셀 라이브러리 및 툴 기능을 검증하였으며 개발 중에 변경되고 있는 FPGA 아키텍처 상에서 임의의 설계를 매핑해 보고 정상 동작할 지 시뮬레이션으로 검증할 수 있음을 확인하였다. 배치 및 배선 툴이 개발 중이며 이들이 완성되면 실용적이고 다양한 신규 FPGA 아키텍처들을 개발하고 그 경쟁력을 평가할 수 있게 될 뿐만 아니라 신규 아키텍처를 위한 최적화 CAD 툴 개발 연구가 활발해지는 시너지 효과도 기대할 수 있다.

Abstract

The research on the FPGA CAD tools in academia has been lacking practicality due to the underlying FPGA fabric architecture which is too simple and inefficient to be applied for commercial FPGAs. Recently, the database of placement positions and routing graphs on commercial FPGA architectures has been built, and provided for enabling the academic development of placement and routing tools. To extend the limit of academic CAD tools even further, we have developed the evaluation toolkit for the K-FPGA architecture which is under development. By providing interface for exchanging data with a commercial FPGA toolkit at every step of mapping, packing, placement and routing in the tool chain, the toolkit enables individual tools to be developed without waiting for the results of the preceding step, and with no dependency on the quality of the results, and compared in detail with commercial tools at any step. Also, the fabric primitive library is developed by extracting the prototype from a reporting file of a commercial FPGA, restructuring it, and modeling the behavior of basic gates. This library can be used as the benchmarking target, and a reference design for new FPGA architectures. Since the architecture is described in a standard HDL which is familiar with hardware designers, and read in the tools rather than hard coded, the tools are “data-driven”, and tolerable with the architectural changes due to the design space exploration. The experiments confirm that the developed library is correct, and the functional correctness of applications implemented on the FPGA fabric can be validated by simulation. The placement and routing tools are under development. The completion of the toolkit will enable the development of practical FPGA architectures which, in return, will synergically animate the research on optimization CAD tools.

Keywords : FPGA, Fabric Architecture, Toolkit, Evaluation, Data-Driven

* 정회원, 인천대학교 전자공학과
(Electronic Engineering, University of Incheon)

※ 본 논문은 지식경제부 기술혁신사업 (산업원천기술개발사업 Configurable 디바이스 및 회로구현기술, KI002168)으로 지원된 연구임.

접수일자: 2012년2월8일, 수정완료일: 2012년3월28일

Ⅰ. 서 론

지금까지 FPGA (Field Programmable Gate Array)에 대한 기존 연구는 IP 검증, 신규 디지털 시스템의 프로토타이핑, 또는 하드웨어를 통한 운영체제 확장 실험 등과 같이 상용 FPGA 소자를 설계 구현에 응용하는 형태, 그리고 단순한 FPGA 가상 구조를 정의하고 그 구조에서 다양한 설계 변수를 최적화하는 CAD툴 연구 형태[1~2]의 극단적인 두 유형으로 양분되어 왔다. 그러나 실용적인 연구가 되려면 상용 FPGA 구조에 직접 적용할 수 있는 합성, 배치, 배선 등의 CAD툴을 개발하여야 한다. 불행하게도 지금까지 상용 FPGA 소자의 데이터베이스는 개발사만의 전유물이었고 CAD툴을 개발하려는 사용자에게 필요한 정보 제공 인터페이스는 충분하지 않았다. 만약 학계에서 CAD 툴 개발 후 그 성능을 실제 FPGA 상에서 실험할 수 있다면 신뢰도 높은 결론을 제시할 수 있을 것이다. 예를 들어 Freuβer와 Spallek은 범용 구조의 캐리 체인을 수용할 수 있도록 상용 FPGA의 패브릭 (Fabric)을 구성하는 슬라이스의 구조를 변경하여 그 개선 효과를 여러 가지 실험으로 예측하였다[3]. 만약 구조 변경에 따라 필요한 툴 변경 요구 사항을 상용 FPGA를 위해 제공되는 CAD툴에서 직접 수정하여 그 개선 효과를 증명할 수 있다면 즉시 제품에 적용할 수 있게 되며 학계의 연구가 업계의 발전을 가속화시키는 바람직한 모델이 구축될 것이다. 최근 이러한 학계의 시도가 USC 그룹의 Torc[4] 및 BYU 그룹의 RapidSmith[5] 툴킷 형태로 나타났다. 두 툴킷 모두 Xilinx 사의 XDL 파일 형식[6]을 근간으로 데이터베이스를 구축하고 C++/Java 형태의 API (Application Programmable Interface)를 제공함으로써 설계 입력, 논리 합성, 패킹, 배치, 배선, 아키텍처 뷰어 등을 플러그-인 할 수 있도록 하고 있다. 특히, 계층적 재구성을 통해 패브릭 배열 구조를 효율적으로 표현함으로써 XDL 기술을 1000배 이상 데이터 압축하는 시범을 보이기도 했다. 하드 매크로를 사용하여 FPGA 컴파일 시간을 단축하는 연구[7]에서 Torc가 응용되었는데 이는 상용 FPGA를 대상으로 하는 학계 최초의 CAD 툴이라 할 수 있다. 한편, 기본 특허의 만료 시기가 다가오면서 Xilinx 및 Altera의 아성이었던 FPGA 산업에 세계적으로 많은 벤처 기업들이 관심을 보이고 있으며 FPGA 툴킷 플랫폼[8]을 공급하는 전문 업체가 등장하기도 했다. 이들도 모두 XDL을 통한 Xilinx FPGA와의 데이터베이스 인터페이스를 기초로 하고 있다. 국내에서도 “Configurable 디바이스 및 회로구현 기술”이라는 지식경제부 기술혁신사업이 2009년부터 진행되어 왔으며 신규 아키텍처인 K-FPGA가 개발되고 있다. 본 논문은 상기 과제에서 K-FPGA 아키텍처를 개발 및 평가하는 패러다임을 소개하고 아키텍처 변화에 대한 툴킷의 유연한 대처 방안을 소개한다. Torc나 RapidSmith 등의 기존 툴킷이 주로 상용 FPGA의 패브릭 배열 구조를 추출하여 C++/Java API로 제공함으로써 그 구조상에서 배치 및 배선 등을 개발할 수 있도록 하는 것에 중점을 두고 있는 반면에 K-FPGA 툴킷은 상용 FPGA 툴킷과의 인터페이스를 긴밀히 구축하여 툴 검증 및 벤치마킹을 용이하게 하면서도 패브릭 및 그 배열 구조를 개선하거나 신규 구조로 교체할 경우 툴에서 변경 사항을 유연하게 수용할 수 있도록 한 점이 다르다. 특히, 프로그래밍 언어 기반의 API 대신에 하드웨어 설계자에게 익숙하며 시뮬레이션 검증이 가능한 표준 하드웨어 기술 언어 (Hardware Description Language, HDL)를 사용하여 패브릭 기본 구조를 기술할 수 있도록 한 것이 특징이다. 뿐만 아니라 매핑된 넷리스트를 패브릭 구조에 패킹할 때의 구성 규칙도 HDL 형식으로 기술할 수 있도록 하였다.

Ⅱ. 본 론

1. 배경 이론

상용 FPGA 툴킷을 설명하려면 먼저 상용 FPGA의 패브릭 구조 이해가 필요하다. 그림 1에 Xilinx Spartan III[9] 슬라이스 구조를 나타내었다. 먼저 F와 G로 표시된 LUT (Look-Up-Table)가 2개, FFX와 FFY로 표시된 플립플롭이 2개 포함되어 있다. 또한, 입력 수가 많은 광폭 입력 함수 구현을 위한 멀티플렉서 (F5MUX 및 F6MUX), 그리고 산술연산 논리 구현을 위한 캐리 멀티플렉서 (CYMUXF, CYMUXG) 및 전용 게이트들 (XORF, XORG, FAND, GAND)이 있다. 이 밖에 연결 선택을 프로그램 할 수 있는 멀티플렉서, 인버터(INV), 스위치 (USED)들이 연결 경로를 재구성하여 구성 요소들을 선택적으로 사용할 수 있도록 하고 있다. 이러한 슬라이스들이 2차원 배열 구조 형태로 FPGA 칩 상에 수천~수백만 개 분포되며 그것들 사이사이에 배치된 스위치 박스들이 슬라이스들 간의 연결을 결정한다. 슬라이스 내부 혹은 스위치 박스 내에서 연결을 재구성

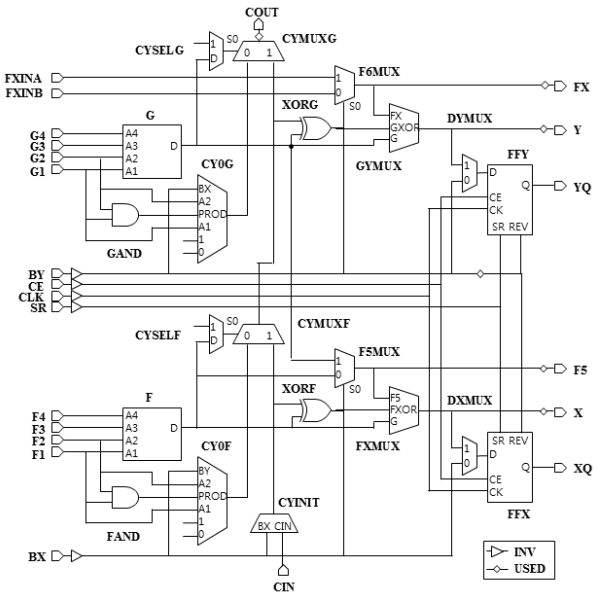

그림 1. Xilinx Spartan III 슬라이스 (SLICEL) 구조
Fig. 1. Slice structure (SLICEL) of the Xilinx Spartan III.

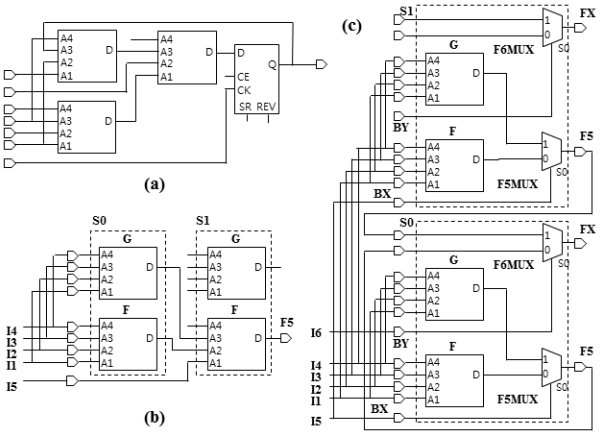

그림 2. 광폭 입력 함수의 FPGA 구현
Fig. 2. FPGA Implementation of a wide fanin function.

하는 멀티플렉서들은 SRAM 내에 저장된 비트 값으로 제어되는데 이 비트들은 외부 비휘발성 메모리에 저장되었다가 초기화 시에 스트림 형태로 FPGA 칩에 로드된다. 결국, FPGA 툴킷의 최종 목표는 소자 기능 및 연결을 재구성하여 FPGA 칩이 원하는 기능을 수행하도록 하는 이 비트 스트림을 만들어 내는 것이다.

최근까지 학계에서는 LUT와 플립플롭만 존재하는 매우 간단한 슬라이스 구조를 가정하여 자동화 툴이 연구되어 왔다. 학계의 대표적인 논리합성 툴은 UC Berkeley의 ABC[1]이고 배치 배선 툴은 캐나다 토론토 대학의 VPR[2]이다. 간단한 슬라이스 구조를 가정하기 때문에 ABC에서 합성되는 회로는 그림 2(a)와 같이 LUT와 플립플롭들만 연결된 형태이다. 사실 4-입력 LUT로 입력이 4개인 모든 조합 논리의 구현이 가능하며 조합 논리의 입력 개수가 늘어나도 LUT들을 트리 형태로 연결하면 되므로 이 형태로 어떤 디지털 회로도 구현할 수 있다. 그림 2(b)에 LUT를 트리 형태로 연결하여 입력이 5개인 조합 논리를 구현한 회로를 보이고 있다. 그림 1의 슬라이스 구조를 사용한다면 2개의 슬라이스 (S0와 S1)가 필요함을 알 수 있다. 만약, 그림 1의 슬라이스에 있는 F5MUX를 사용하여 Shannon의 확장 정리를 적용한다면 그림 2(c)의 S0 슬라이스에서와 같이 2개의 LUT, 즉 하나의 슬라이스만으로 저렴하게 5-입력 조합 논리를 구현할 수 있다. 더 나아가, S1 슬라이스로 또 하나의 5-입력 조합 논리를 구현하고 이 둘을 S0 슬라이스의 F6MUX로 연결하면 6-입력 조합 논리를 구현할 수 있다.

사실, 광폭 입력 함수가 많지 않은 설계의 경우 이 멀티플렉서 사용으로 인한 슬라이스 사용량 감축은 크지 않을 수 있다. 그러나 회로의 속도를 결정하는 슬라이스의 단수가 감소된다는 점이 더 중요하다. 신호가 하나의 슬라이스에서 다른 슬라이스로 전달될 경우 긴 연결선을 지날 뿐만 아니라 스위치 박스를 하나 이상 거치게 되는데 이 배선 지연 시간은 LUT의 게이트 지연 시간보다도 더 우세하게 된다. 따라서 슬라이스 단수가 증가하면 회로의 성능이 저하될 수밖에 없다. 그림 2(c)에서 LUT 출력들은 내부의 전용 배선을 통해 F5MUX로 연결되기 때문에 이 배선에 의한 지연은 거의 무시할 수 있다. 또한, F5MUX 출력을 F6MUX로 연결시킬 경우에도 인접한 슬라이스 간에 전용 지역 배선을 만들면 스위치 박스를 거치지 않고 단거리로 연결될 수 있다. 그림 2(b)의 회로는 2단의 슬라이스 연결로 5-입력 조합 논리를 구현하며 6-입력을 구현하려면 3단 연결이 필요할 것이다. 반면에 그림 2(c)에서 보는 바와 같이 광폭 입력 멀티플렉서와 전용 배선을 이용하면 6-입력이나 그 이상의 입력에도 슬라이스 단수가 늘어나지 않는다.

산술 연산을 수행하는 회로에서는 대개 캐리 체인의 지연 시간이 회로 전체의 성능을 좌우한다. 이 경우에도 슬라이스의 단수를 줄이는 것이 중요하다. 그림 3(a)는 2-비트 리플 캐리 가산기 회로를 나타내었다. 이를 LUT만으로 구현한다면 그림 3(b)와 같이 될 것이며 첫 번째 캐리 출력 C1이 S0 슬라이스에서 S1 슬라이스로 연결되어 2단 회로가 된다. 비트 수가 증가하면 단수는

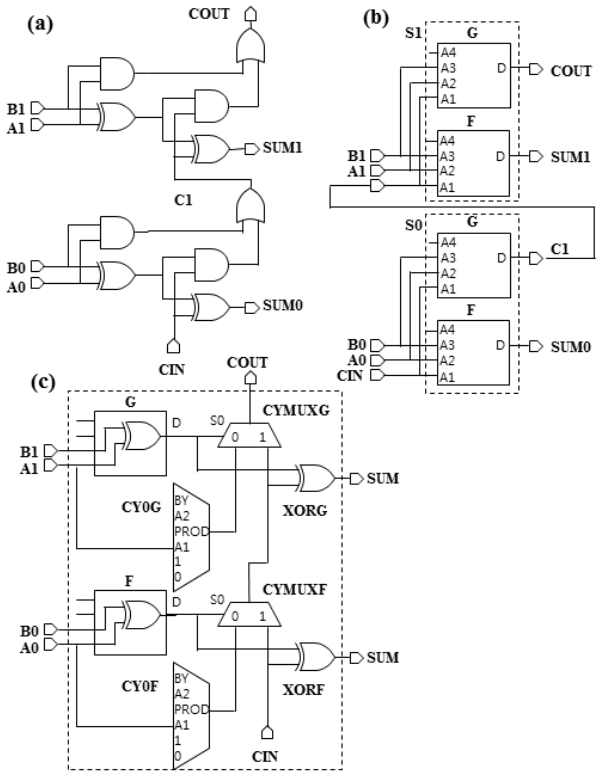

그림 3. 캐리 체인의 FPGA 구현
Fig. 3. FPGA Implementation of a carry chain.

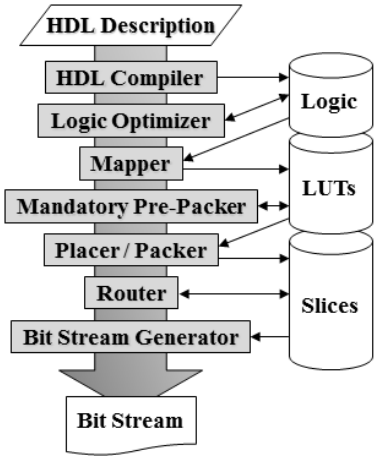

그림 4. FPGA 설계 툴 체인 사용 절차
Fig. 4. FPGA design flow using the tool chain.

비트 수만큼 증가할 것이다. 그러나 그림 3(c)와 같이 캐리 멀티플렉서와 전용 XOR 게이트들을 사용하면 한 슬라이스 안에 2-비트 가산기를 구현할 수 있다. 또한, 인접한 슬라이스와 전용 지역 배선을 통해 연결하면 가산기의 비트 수가 증가하더라도 단수 증가를 피할 수 있다.

그림 4에 FPGA 설계 툴 체인 사용 절차를 나타내고 있다. 먼저 RTL HDL로 기술된 설계가 입력되면 HDL 컴파일러가 하드웨어 추론을 통해 이를 논리 수식(Logic)으로 변환한다. 논리 수식을 최적화한 후 LUT와 플립플롭 등과 같은 논리 단위 셀 (LUTs)에 매핑시킨다. 이 때, 산술 연산 전용 게이트, 광폭 입력 함수 멀티플렉서, 블록 메모리, 블록 승산기 등이 포함된다. 흔히, 배치와 패킹은 동시에 진행되며 패킹 후에는 LUT와 플립플롭 등이 슬라이스 내부로 포함되기 때문에 넷리스트 상에 나타나지 않고, 패브릭 단위 셀, 즉, 슬라이스, I/O 버퍼 (IOB), 블록들로만 연결된 넷 리스트 (Slices)로 변환된다. Xilinx FPGA 툴킷은 이 넷 리스트를 XDL 파일 형식[6]으로 입출력한다. 이때, 슬라이스에 포함되는 논리 단위 셀들 및 그들 간의 연결은 재구성 속성 정보 형태(cfg)로 각 슬라이스에 저장된다. 캐리 체인 및 광폭 입력 함수 구현에 사용되는 논리 단위 셀들은 멀티플렉서 및 전용 게이트들 간의 인접성을 만족시키기 위해 상대적 배치 위치가 미리 결정되며 이를 계속 유지해야 하므로 필수 전처리 과정으로 먼저 패킹한다. 배치에서 LUT 및 플립플롭 등의 위치가 결정되면 그들이 속하게 될 슬라이스도 최종 결정되므로 패킹이 완료된다. 배선은 다양한 길이의 수평, 수직 연결선들 간에 위치한 스위치들의 연결 방향을 선택하여 슬라이스의 핀들을 서로 연결하는 경로를 만들어 내는 과정이다. 배치 배선 후에는 각 스위치들의 상태를 나타내는 비트들을 스트림 형태로 출력하여 FPGA 칩에 로드할 수 있도록 한다.

2. FPGA 패브릭 구조 개발 툴 킷

FPGA의 경쟁력은 칩 상에 주어진 패브릭에 일반적으로 어떤 규모의 회로를 구현할 수 있는지를 나타내는 기능 밀도 (Function Density) 및 구현된 회로의 성능으로 좌우된다. 이는 패브릭 구조, 즉, 슬라이스 및 스위치 박스, 블록의 내부 구조, 배선 형태, 그리고 이들의 조성비 및 칩 상에 배치되는 배열 형태 등 복합적 요인들로 결정된다. 따라서 FPGA 패브릭 구조 개발 툴킷은 새로운 패브릭 구조가 FPGA의 경쟁력을 얼마나 향상시킬 수 있는지 평가할 수 있어야 한다. 예를 들어, 기능 밀도 및 성능 향상을 위해 LUT 입력 수를 늘리거나 산술 연산용 전용 게이트를 추가하거나 혹은 전용 지역 배선을 추가하는 등 패브릭 구조를 변경할 수 있다. 최종적으로는 다양한 회로를 이 새로운 구조상에 구현했을 때 과연 기능 밀도 및 성능이 얼마나 향상되는지 확인해야 한다. 따라서 매핑, 패킹, 배치 및 배선에 이르는

툴 킷 전반에도 구조 변경에 따른 수정이 필요하다. 신규 FPGA 구조를 개발하면서 이러한 수정 요구가 수시로 발생할 수 있기 때문에 툴이 패브릭 구조의 변화에 유연하게 대처할 필요가 있다. Torc[4] 및 RapidSmith[5]는 XDL을 통하여 Xilinx FPGA와 넷 리스트 호환성 등을 먼저 확보하고, 상용 FPGA의 배치 및 배선 구조 데이터베이스 인터페이스를 제공함으로써 학계에서도 상용 FPGA를 위한 배치 배선 툴을 개발 가능하도록 하고 있다. 이들은 C++/Java 오브젝트들을 원시 코드 형태로 제공하여 툴 개발 및 변경 효율성의 향상을 추구하였다. 그러나 FPGA 패브릭 구조 변경 시 이를 탄력적으로 대응하여 검증하고 평가할 수 있는 툴킷 개발에는 적지 않은 추가적 노력이 필요하다. 특히, Xilinx FPGA의 슬라이스 구조가 툴킷 내에 하드 코딩되어 있어 변경에 취약하다. 상용 FPGA를 위한 툴 개발이 일부 가능하게 되었지만 그것이 최종 목표인가? 툴 체인이 완성된 다음 단계는 상용 수준을 능가하는 아키텍처의 개발을 추구하게 될 것이다. 툴킷이 지속적인 아키텍처 변경을 허용하는 높은 범용성을 확보하지 않으면 이 분야의 연구가 또 다른 한계성에 부딪치고 말 것이다. 따라서 툴 체인은 소프트웨어 내에 고정된 모델이 아니라 패브릭 구조를 정의하는 설계 데이터에 의해 제어되는 데이터 구동 형태가 바람직하다. 특히, 설계 데이터는 프로그래밍 언어가 아니라 그동안 하드웨어 설계자가 사용해 왔던 HDL로 기술될 수 있어야 한다. 본 논문에서는 지속적으로 변경되는 아키텍처를 검증 및 평가할 수 있는 설계 방법론을 제안하고 이를 실현할 수 있도록 데이터 구동형으로 개발된 K-FPGA 툴킷을 보고하고자 한다.

가. 단위 셀 라이브러리 개발 절차

그림 5에 단위 셀 라이브러리 개발 절차를 나타내고 있다. 기본적으로 모든 설계 데이터는 SI2 (Silicon Integration Initiative)에서 C++ 원시 코드로 제공하는 oaDB (OpenAccess Database)[10]에 저장된다. 이것은 EDA (Electronic Design Automation) 업계에서 널리 사용되고 있으며 실질적인 표준으로 자리 잡고 있는 공통 데이터베이스이다.

Xilinx 툴킷은 칩 상에 단위 셀의 배치 위치 및 배선 그래프를 XDLRC라는 대용량 리포트 파일로 출력한다. Torc[4] 및 RapidSmith[5]가 제공하는 상용 FPGA 패브릭 정보도 역시 이 파일에서 취한 것이다. 그런데 이 파

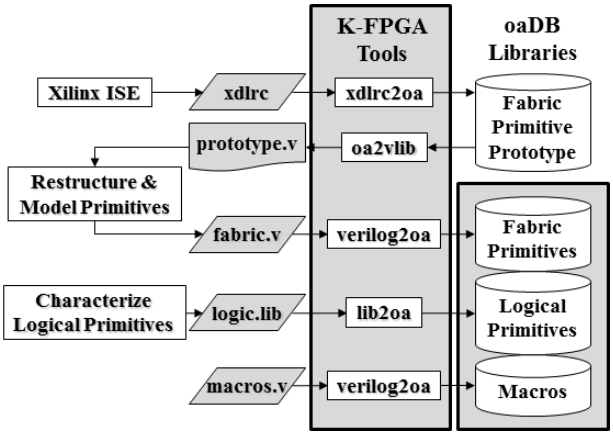

그림 5. 단위 셀 라이브러리 개발 절차
Fig. 5. Design flow of primitive libraries.

```
module pLUT(A1, A2, A3, A4, D);
  input   A1, A2, A3, A4;
  output  D;

  (* values = "`LUT `OFF" *) parameter  cfg = `OFF;
  parameter   init = 16'h0000;
  wire [3:0]  addr, a, wa;
  reg [15:0]  data;

  assign a = {A4, A3, A2, A1};
  assign D = data[a];
  initial begin
    data = init;
  end
endmodule

module pFF(CK, CE, D, Q, SR, REV);
  input   CK, CE, D, SR, REV;
  output  Q;

  (* values = "`FF `LATCH `OFF" *)     parameter cfg=`OFF;
  (* values = "`INIT1 `INIT0 `OFF" *)  parameter init=`INIT1;
  (* values = "`SRHIGH `SRLOW `OFF" *) parameter sr=`OFF;
  (* values = "`SYNC `ASYNC `OFF" *)   parameter sync=`ASYNC;
  wire    ival, syncSR, asyncSR;
  reg     latch, flipflop;

  assign Q=(cfg==`FF)?flipflop:((cfg==`LATCH)?latch:1'b0);
  assign ival=(sr==`SRHIGH)?(REV?1'b0:1'b1):(REV?1'b1:1'b0);
  assign syncSR=(sync==`SYNC)?SR:1'b0;
  assign asyncSR=(sync==`ASYNC)?SR:1'b0;
  initial begin
    latch = (init==`INIT1)?1'b1:1'b0;
    flipflop = (init==`INIT1)?1'b1:1'b0;
  end
  always @(CK or asyncSR) begin
    if(asyncSR)       latch = ival;
    else if(syncSR)   latch = ival;
    else if(CK && CE) latch = D;
  end
  always @(posedge CK or posedge asyncSR) begin
    if(asyncSR)       flipflop = ival;
    else if(syncSR)   flipflop = ival;
    else if(CE)       flipflop = D;
  end
endmodule
```

그림 6. 배치 소자의 Verilog 기술
Fig. 6. Verilog description of two placeable components.

일은 이외에도 패브릭 단위 셀의 게이트 수준 회로도 기술하고 있다. 이를 oaDB로 읽어 들인 후 Verilog 넷 리스트 형태로 출력하면 Xilinx의 패브릭 단위 셀의 HDL 기술 원형(prototype.v)을 얻을 수 있다. 슬라이스, IOB, 블록 메모리, 블록 승산기 등 약 20종의 단위 셀이 구조적으로 기술된다. 단위 셀 내부를 구성하고 있는 LUT나 플립플롭 등 기본 게이트들의 원형도 100여종 기술되어 있으나 이들의 동작 기술은 수작업으로 추가해야 한다. 슬라이스 내부의 LUT는 원래의 기능이외에

분산 ROM/RAM으로도 확장 활용할 수도 있는데 LUT를 분산 ROM으로만 확장 활용할 수 있는 슬라이스를 SLICEL이라 하고 RAM으로도 활용할 수 있는 것은 SLICEM이라 한다. 그림 1에 나타낸 것이 SLICEL의 도면이다. 패브릭 셀 내부의 기본 게이트들은 LUT나 플립플롭 같이 논리 단위 셀과 일대일 대응되는 배치 소자와 멀티플렉서 형태로 배치 소자 간의 연결을 결정하는 특성 소자로 분류할 수 있으며 각각 셀 이름에 'p'와 'c'를 접두사로 붙여 구분한다.

그림 6은 대표적인 배치 소자인 LUT와 플립플롭을

```
module cINV(I1_B, I1, OUT);
  input   I1_B, I1;
  output  OUT;

  (* values = "`I1_B `I1 `OFF" *) parameter cfg = `OFF;
  assign OUT = (cfg==`I1_B)? ~I1_B:((cfg==`OFF)?1'b1:I1);
endmodule

module cMUX(I0, I1, OUT);
  input   I0, I1;
  output  OUT;

  (* values = "`I0 `I1 `OFF" *) parameter cfg = `OFF;
  assign OUT = (cfg==`I0)?I0:((cfg==`I1)?I1:1'b0);
endmodule

module cFXMUX(F5, F, FXOR, OUT);
  input   F5, F, FXOR;
  output  OUT;

  (* values = "`F `F5 `FXOR `OFF" *) parameter cfg = `OFF;
  assign OUT = (cfg==`F5)?F5:
    ((cfg==`F)?F:((cfg==`FXOR)?FXOR:1'b0));
endmodule
```

그림 7. 특성 소자의 Verilog 기술
Fig. 7. Verilog description of configurable components.

```
module SLICEL(BX, BY, CE, CIN, CLK, SR, F1, F2, F3, F4, G1, G2,
  G3, G4, FXINA, FXINB, F5, FX, X, XB, XQ, Y, YB, YQ, COUT);
  input  BX, BY, CE, CIN, CLK, SR, F1, F2, F3, F4, G1, G2,
         G3, G4, FXINA, FXINB;
  output F5, FX, X, XB, XQ, Y, YB, YQ, COUT;

  parameter aBXINV = `OFF, aBYINV = `OFF, aCEINV = `OFF,
    aCLKINV = `OFF, aCOUTUSED = `OFF, aCY0F = `OFF, aCY0G =`OFF,
    aCYINIT = `OFF, aCYSELF = `OFF, aCYSELG = `OFF, aDXMUX =`OFF,
    aDYMUX = `OFF;
  (* aF = "aF_INST:aF:aF_INIT" *) parameter aF = `OFF;
  parameter aF_INST = "", aF_INIT = 16'h0000, a_BEL_PROP_F = "",
    aF5USED = `OFF;
  (* aFFX = "aFFX_INST:aFFX" *) parameter aFFX = `OFF;
  parameter aFFX_INST = "", aFFX_INIT_ATTR = `OFF,
    aFFX_SR_ATTR = `OFF;
  (* aFFY = "aFFY_INST:aFFY" *) parameter aFFY = `OFF;
  parameter aFFY_INST = "", aFFY_INIT_ATTR = `OFF,
    aFFY_SR_ATTR = `OFF, aFXMUX = `OFF, aFXUSED = `OFF;
  (* aG = "aG_INST:aG:aG_INIT" *) parameter aG = `OFF;
  parameter aG_INST = "", aG_INIT = 16'h0000, a_BEL_PROP_G = "",
    aGYMUX = `OFF, aREVUSED = `OFF, aSRINV = `OFF,
    aSYNC_ATTR = `OFF, aXBUSED = `OFF, aXUSED = `OFF,
    aYBUSED = `OFF, aYUSED = `OFF, aCYMUXF = "", aCYMUXG = "",
    aFAND = "", aGAND = "", aXORF = "", aXORG = "", aF5MUX = "",
    aF6MUX = "", aC1VDD = "", aC2VDD = "", aGNDF ="", aGNDG ="";

  pLUT  #(aF,aF_INIT) F (.A1(F1),.A2(F2),.A3(F3),.A4(F4),.D(f));
  pFF   #(aFFX,aFFX_INIT_ATTR,aFFX_SR_ATTR,aSYNC_ATTR) FFX
    (.D(dxmux),.REV(revused),.CE(ceinv),.CK(clkinv),
    .SR(srinv),.Q(XQ));
  pF5MUX F5MUX (.S0(bxinv),.F(f),.G(g),.OUT(f5mux));
  pCYMUX CYMUXF (.I0(cy0f),.I1(cyinit),.S0(cyself),.OUT(cymuxf));
  pXOR   XORF (.I1(cyinit),.I0(f),.O(xorf));
  pAND   FAND (.I0(F1),.I1(F2),.O(fand));
                    :
  cCY0    #(aCY0F) CY0F (.I0(gndf),.I1(c1vdd),.A1(F1),
    .PROD(fand),.A2(F2),.BY(bxinv),.OUT(cy0f));
  cCYINIT #(aCYINIT) CYINIT (.CIN(CIN),.BX(bxinv),.OUT(cyinit));
  cCYSEL  #(aCYSELF) CYSELF (.I1(vddf), .D(f), .OUT(cyself));
  cMUX    #(aDXMUX)  DXMUX (.I1(fxmux),.I0(bxinv),.OUT(dxmux));
  cFXMUX  #(aFXMUX) FXMUX (.FXOR(xorf),.F5(f5mux),.F(f),
    .OUT(fxmux));
  cUSED   #(aXUSED) XUSED (.I0(fxmux),.OUT(X));
  cINV    #(aBXINV) BXINV (.I1_B(BX), .I1(BX), .OUT(bxinv));
                    :
endmodule
```

그림 8. SLICEL의 Verilog 기술 (일부 생략)
Fig. 8. Verilog description of SLICEL.

동작 모델링한 Verilog 기술이며 파라미터를 통해 여러 가지 기능 중 선택하여 재구성할 수 있도록 되어 있다. 그림 7은 3가지 특성 소자를 모델링한 것으로 역시 파라미터를 통해 배치 소자들 간의 연결을 선택적으로 재구성하는 역할을 한다. 각 파라미터에는 지정될 수 있는 값들을 "values"라는 속성으로 열거하고 있는데 이는 XDLRC에서 추출된 정보로서 모델링 시 이들 조합에 대해 동작을 기술하여야 한다. 추출된 회로에는 단위 셀 내 기본 게이트들 각각에 대해 별도의 마스터 모듈이 정의되어 있어 그 수가 지나치게 많은데 이를 종류별로 단일화 하면 계층 구조가 좀 더 간단해 진다.

그림 8은 그림 1에 나타낸 패브릭 단위 셀 SLICEL의 Verilog 기술이며 지면 제약 상 그림 1의 회로에서 아래쪽 반만 기술하였고 위쪽 반은 생략하였다. 각 파라미터는 이름 사용의 충돌을 막기 위해 'a'를 접두사로 사용한다. 파라미터들은 XDL의 슬라이스 인스턴스에 부여되는 재구성 속성 정보 (cfg)와 그 내용 및 순서가 일치하며 그 값은 기본 게이트에 전달되어 원하는 기능과 연결을 구현한다. 다수의 파라미터를 묶어서 하나의 XDL 재구성 속성으로 지정되는 경우가 있는데 그 조성 규칙을 파라미터의 속성으로 지정한다. 예를 들어, XDL의 "F" 속성은 파라미터 값 세 개가 콜론(:)으로 묶여서 "aF_INST:aF:aF_INIT" 형식으로 지정되도록 aF 파라미터에 속성이 부여되어 있다. 패브릭 단위 셀 라이브러리는 표준 HDL로 기술되어 oaDB에 저장되며 구현될 회로를 실제 패브릭 상에 재구성 해 놓고 원하는 동작을 하는지 시뮬레이션 할 수 있게 된다. 특히 비트 스트림 정보로도 사용되는 슬라이스의 재구성 속성을 모델 파라미터에 대응시켜 툴 체인 전체에서 일관성 있게 관리한다. 따라서 HDL로 기술된 슬라이스 구조가 변경되면 이 파라미터가 바뀌기 때문에 모든 툴에 즉시 영향을 미친다. 논리 단위 셀은 회로 시뮬레이션을 통해 비선형 타이밍 모델을 개발한 후 그림 5에서와 같이 Liberty[11] 표준 형식으로 기술하면 oaDB에 저장할 수 있다. 이 라이브러리는 논리 합성, 타이밍 검증, 배치, 패킹 등에서 사용된다.

나. K-FPGA 패브릭 검증 및 평가 시스템

개발된 FPGA 패브릭에 다양한 설계 예제를 구현해 봄으로써 패브릭을 검증하고 기능 밀도 및 성능을 평가할 수 있다. 따라서 그림 9와 같이 매핑, 패킹, 그리고 배치 및 배선 툴들이 평가 시스템에 모두 포함된다.

평가 시스템을 구축할 때 두 가지 중요한 요소가 있다. 먼저, 평가 시스템 자체가 검증되어야 하기 때문에 각 툴 별로 당대 기술의 기준이라 할 수 있는 상용 FPGA 툴 킷과 벤치마킹 할 수 있어야 한다. 둘째는 각 툴들이 이전 단계의 출력을 기다리거나 결과의 질에 영향을 받지 않으면서 독립적으로 개발되고 검증될 수 있어야 한다. 이를 위해서는 먼저 상용 FPGA의 패브릭 아키텍처를 근간으로 한 단위 셀 라이브러리가 필요한데 앞 절에서 그 개발 방법을 설명한 바 있다. 둘째로 상용 툴과 중간 단계의 결과를 교환할 수 있어야 한다. 따라서 상용 툴의 매핑 결과 (EDIF 형식)를 oaDB로 입력하는 기능 (edf2oa) 및 상용 툴의 패킹 및 배치 결과 (XDL 형식)를 oaDB로 입출력하는 기능 (xdl2oa, oa2xdl)이 개발되었다. 마지막으로 논리 단위 셀 넷 리스트 (LUTs)에 저장된 패킹 정보에 따라 패브릭 단위 셀 넷 리스트 (Slices)를 생성시키는 기능 (log2slice) 및 반대로 Slices에서 패킹 정보를 추출하여 패브릭 단위 셀 넷 리스트 (LUTs)에 저장하는 기능 (slice2log)도 개발되었다. 따라서 매핑 및 배치 전 필수 패킹의 개발 완료를 기다리지 않고 상용 툴의 중간 결과를 사용하여 배치를 개발, 검증 및 벤치마킹을 할 수 있으며, 배치 개발 완료 전에 배선 개발을 시작할 수도 있다.

캐리 체인 및 광폭 입력 함수에 포함되는 멀티플렉서 및 전용 게이트들의 상대적 배치 위치를 결정하는 필수 전처리 패킹도 데이터 구동 형태로 구현되어야 한다. 따라서 그림 10과 같이 기술된 규칙을 읽어 들여 수행된다. 즉, 매핑된 넷 리스트에서 논리 단위 셀들을 그 핀들의 연결 패턴에 따라 슬라이스 반쪽 단위 (HS)로 묶고 이들을 광폭 입력 함수 멀티플렉서 (F5MUX, F6MUX)를 중심으로 묶어 광폭 입력 함수 트리를 추출하거나 캐리 멀티플렉서 (CYMUX)를 중심으로 엮어서 체인을 추출한다. 트리 및 체인에서 슬라이스는 수직으로 적층되는데 최하위 좌표가 0이고 올라가면서 차례로 증가한다. 각 반쪽 슬라이스(HS)가 속할 슬라이스의 좌표를 결정하는 수식이 @s 값으로 주어지고 @h 값으로 주어진 수식은 반쪽 슬라이스가 슬라이스 내에서 아래 (@h = 0)에 혹은 위에 (@h = 1) 위치하는지 결정한다. 따라서 패브릭 구조가 어떻게 바뀌더라도 그에 따라 이 규칙을 수정하면 트리나 체인 구조의 필수 패킹 그룹을 추출하고 그 상대적 위치를 지정할 수 있다. 결과는 모두 oaDB의 Property 형식으로 논리 단위 셀 인스턴스에 저장된다.

논리 단위 셀들을 패브릭 슬라이스 배열 상에 배치할 때는 셀 종류별로 배치 가능한 위치가 별도로 정해져 있으며 필수 패킹 그룹 내의 셀들은 같이 이동시켜야 한다. 배치가 완료되면 해당 위치에 따라 각 셀들이 속할 슬라이스가 자동적으로 결정되므로 패킹도 완료된다. 이 패킹 정보를 이용하여 논리 단위 셀 넷 리스트를 패브릭 단위 셀 넷 리스트로 변환 (slice2log)시켜야 하는데 이때도 데이터 구동 형태로 구현하는 것이 바람직하다. 즉, 변환 규칙을 기술하여 입력할 수 있도록 하여야 한다. 블록 메모리나 블록 승산기는 논리 단위 셀과

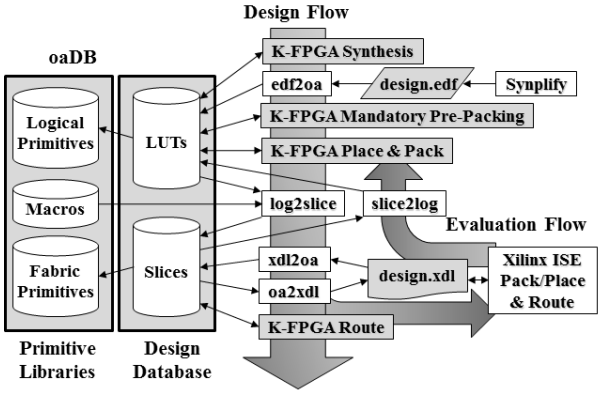

그림 9. K-FPGA 패브릭 검증 및 평가 시스템
Fig. 9. Validation and evaluation system for the K-FPGA fabric.

```
// half slice
%set HS {
  %anchor LUT;
  %add CYMUX @if(CYMUX.S == LUT.O);
  %add XOR @if(XOR.LI == LUT.O);
  %add DFF @if(DFF.D == XOR.O);
  %add AND @if(AND.LO == CYMUX.DI);
  %add DFF @if(DFF.D == LUT.O);
  %add LD @if(LD.D == LUT.O);
}

// wide function mux trees
%tree MT5 {
  %anchor F5MUX;
  %add HS @if(HS.LUT.O == F5MUX.I0);
  %add HS @if(HS.LUT.O == F5MUX.I1), @h(+1);
}
%tree MT6 {
  %anchor F6MUX @h(+1);
  %add MT5 @if(MT5.F5MUX.O == F6MUX.I0);
  %add MT5 @if(MT5.F5MUX.O == F6MUX.I1), @s(+1);
}
            :

// carry chain
%list CCH @s(/2) @h(%2) {
  %anchor HS @with(CYMUX);
  %prepend HS @if(HS.CYMUX.O == $head.CYMUX.CI);
  %append HS @if(HS.CYMUX.CI == $tail.CYMUX.O);
  %append HS @if(HS.XOR.CI == $tail.CYMUX.O);
}
```

그림 10. 배치 전 필수 패킹 규칙 기술
Fig. 10. Description of mandatory pre-packing rules.

```
module mROM64X1(\.A0 , \.A1 , \.A2 , \.A3 , \.A4 , \.A5 , \.O );
  input  \.A0 , \.A1 , \.A2 , \.A3 , \.A4 , \.A5 ;
  output \.O ;

  (* mppg = "{MT }p{:0:0:}i{/F6.S0 S0,S2 F6}" *)
  (* tile = "t" *) (* site = "s" *)
  SLICEL #(.aBXINV("`I1"), .aBYINV("`I1"),
    .aF5USED("`I0"), .aYUSED("`I0"), .aGYMUX("`FX"),
    .aF("`LUT"),.aF_INST("i{/F.S0}"),.aF_INIT("{16'h}v[0:3]"),
    .aG("`LUT"),.aG_INST("i{/G.S0}"),.aG_INIT("{16'h}v[4:7]"),
    .aF5MUX("i{/F5.S0}"), .aF6MUX("i{/F6.S0}"))
   \.O (.F1(\.A0 ), .F2(\.A1 ), .F3(\.A2 ), .F4(\.A3 ),
    .G1(\.A0 ), .G2(\.A1 ), .G3(\.A2 ), .G4(\.A3 ),
    .BX(\.A4 ), .BY(\.A5 ), .F5(\i{/F5.S0} ),
    .FXINA(\i{/F5.S0} ), .FXINB(\i{/F5.S1} ), .Y(\.O ));
  (* mppg = "{MB }p{:0:1}" *)
  (* tile = "t" *) (* site = "s+y1" *)
  SLICEL #(.aBXINV("`I1"), .aF5USED("`I0"),
    .aF("`LUT"),.aF_INST("i{/F.S1}"),.aF_INIT("{16'h}v[8:11]"),
    .aG("`LUT"),.aG_INST("i{/G.S1}"),.aG_INIT("{16'h}v[12:15]"),
    .aF5MUX("i{/F5.S1}"))
   \i{/F5.S1} (.F1(\.A0 ), .F2(\.A1 ), .F3(\.A2 ), .F4(\.A3 ),
    .G1(\.A0 ), .G2(\.A1 ), .G3(\.A2 ), .G4(\.A3 ),
    .BX(\.A4 ), .F5(\i{/F5.S1} ));
endmodule
```

그림 11. 64-비트 분산 ROM의 CPIN 매크로 기술

Fig. 11. CPIN macros description of the 64-bit distributed ROM.

패브릭 단위 셀 간에 핀 및 재구성 속성의 대응 관계만 간단히 정의하면 되나, 분산 메모리의 경우는 다수의 슬라이스에 매핑될 뿐만 아니라 광폭 입력 함수 멀티플렉서를 사용하기 때문에 슬라이스들 간의 연결 및 슬라이스 내의 재구성 속성이 복잡해진다. 그러나 사실, 다수의 논리 단위 셀들이 하나의 패브릭 슬라이스에 패킹되는 일반적인 경우가 훨씬 더 복잡하다. 이는 그 패킹되는 패턴이 수백 가지 조합으로 발생할 수 있기 때문이다. 이들을 모두 규칙으로 기술할 수 있도록 하는 것은 막대한 기술적 도전이라 할 수 있다.

첫째, 조건에 따라 연결 및 재구성 속성을 다르게 지정할 수 있는 언어가 필요하다. 먼저 하드웨어 설계자에게 친숙한 Verilog 넷 리스트 형식을 사용하여 패브릭 단위 셀 인스턴스들로 이 규칙들을 기술하도록 결정하였다. 둘째, 연결을 조건에 따라 선택적으로 해야 하기 때문에 각 핀에 연결된 넷을 기술할 때 단순한 이름이 아닌 조건문을 지정하도록 하였다. 이를 CPIN (Conditional Property, Instance, and Net Names) 문법이라 하며 C 언어 및 Verilog HDL의 조건문과 같이 물음표(?)와 콜론(:)을 사용하여 <조건>?<참-이름>:<거짓-이름>과 같이 기술된다. <조건>에는 &(AND), |(OR), !(NOT)을 포함하는 논리 연산자 및 등가 연산자 (=, !=), 괄호 등이 사용될 수 있다. 간략한 표현을 위해 <조건>과 <참-이름>은 생략될 수 있다. <조건>이 생략되면 <참-이름>의 존재 여부가 조건이 된다. <조건>이 거짓일 때 <거짓-이름>이 생략되면 핀은 미연결 상태가 되며 속성은 생성되지 않는다. <참-이름> 및 <거짓-이름>에는 또 다른 CPIN 구문이 올 수 있어 복잡한 조건을 표현할 수 있다. 핀 이름은 “.”이 접두사로 붙으며 “\”가 접두사로 붙은 이름(예: \I0)은 파라미터에 지정될 수 있는 상수 값이다. 문자열은 중괄호를 사용하여 끼워 넣을 수 있다. CPIN에는 기호가 포함되기 때문에 넷이나 인스턴스 이름으로 사용할 때는 백슬래시로 시작하고 여백 문자로 끝나는 확장 식별자(Escaped Identifier)를 이용한다. 파라미터 값들도 역시 CPIN으로 지정하는데 따옴표를 앞뒤로 사용하는 문자열 형식을 취한다. 확장 식별자와 문자열은 여러 가지 기호를 포함한 조건문 형식의 이름을 Verilog 문법에 위배되지 않게 지정할 수 있는 방법이다.

```
inst "N_6251_a" "SLICEL", unplaced,
  cfg "BXINV::BX BYINV::BY CEINV::#OFF CLKINV::#OFF
    COUTUSED::#OFF CY0F::#OFF CY0G::#OFF CYINIT::#OFF
    CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::#OFF
    F:adr_reg_dest_120_11/F.S0:#LUT:D=0x2000 F5USED::0
    FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
    FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF
    FXMUX::#OFF FXUSED::#OFF
    G:adr_reg_dest_120_11/G.S0:#LUT:D=0x0000
    GYMUX::FX REVUSED::#OFF SRINV::#OFF SYNC_ATTR::#OFF
    XBUSED::#OFF XUSED::#OFF YBUSED::#OFF YUSED::0
    F5MUX:adr_reg_dest_120_11/F5.S0:
    F6MUX:adr_reg_dest_120_11/F6.S0:
    _INST_PROP::XDL_SHAPE_DESC:Shape_12:WIDEFUNC,
      A\ wide\ function\ starting\ with\ F6\ mux\
      \"adr_reg_dest_120_11/F6.S0\"
    _INST_PROP::XDL_SHAPE_ALIGN:Shape_12:S0,S2,
    _INST_PROP::XDL_SHAPE_MEMBER:Shape_12:0,0 "
  ;
inst "adr_reg_dest_120_11/F5.S1" "SLICEL", unplaced,
  cfg "BXINV::BX BYINV::#OFF CEINV::#OFF CLKINV::#OFF
    COUTUSED::#OFF CY0F::#OFF CY0G::#OFF CYINIT::#OFF
    CYSELF::#OFF CYSELG::#OFF DXMUX::#OFF DYMUX::#OFF
    F:adr_reg_dest_120_11/F.S1:#LUT:D=0x1020 F5USED::0
    FFX::#OFF FFX_INIT_ATTR::#OFF FFX_SR_ATTR::#OFF
    FFY::#OFF FFY_INIT_ATTR::#OFF FFY_SR_ATTR::#OFF
    FXMUX::#OFF FXUSED::#OFF
    G:adr_reg_dest_120_11/G.S1:#LUT:D=0xC050
    GYMUX::#OFF REVUSED::#OFF SRINV::#OFF
    SYNC_ATTR::#OFF
    XBUSED::#OFF XUSED::#OFF YBUSED::#OFF YUSED::#OFF
    F5MUX:adr_reg_dest_120_11/F5.S1:
    _INST_PROP::XDL_SHAPE_MEMBER:Shape_12:0,1 "
  ;
```

그림 12. CPIN 매크로 기술에 의해 변환된 64-비트 분산 ROM의 XDL 기술

Fig. 12. XDL description of a 64-bit distributed ROM translated by the CPIN macro description.

그림 13은 그림 1에 나타낸 SLICEL의 CPIN 매크로 기술을 보이고 있으며 역시 지면 제약 상 아래쪽 반만 나타내고 위쪽 반은 생략하였다. 슬라이스 내의 논리 단위 셀 인스턴스들 중 아래쪽에 있는 F, FFX, FAND, CYMUXF, XORF, F5MUX는 각각 L1, F1, A1, C1, X1, F5로 표기하고 위쪽에 있는 것은 각각 L2, F2, A2, C2, X2, F6로 표기한다.

이 셀들의 핀과 연결된 넷은 “.” 연산자를 사용하여 얻을 수 있는데 예를 들어 .CLK(\F1.C:F2.C)는 FFX의 C핀에 연결된 넷이 있으면 그것을 슬라이스의 CLK 핀에 연결하고 그렇지 않으며 FFY의 C핀에 연결된 넷을 연결하라는 의미이며 만약 FFX 및 FFY 모두 사용되지 않으면 슬라이스의 CLK 핀은 미연결 상태로 남겨 둔

```
module mSLICEL();
  (* mppg = "g" *) (* tile = "t" *) (* site = "s" *)
  SLICEL  #(
    .aBXINV("C1.CI@pi?`I1_B:F1@xi?(F1.D@pi?`I1_B:`I1):
      F5|C1@ui|!C1@ci&C1?~I1"),
    .aCEINV("F1.CE@pi|F2.CE@pi?`I1_B:F1.CE|F2.CE?`I1"),
    .aCLKINV("F1|F2?`I1"), .aCOUTUSED("C2?`IO"),
    .aCY0F("A1?`PROD:C1.DI={GND}?`I0:C1.DI={VCC}?`I1:
      C1.DI=L1.IO?`A1:C1.DI=L1.I1?`A2:C1@ui?`BY"),
    .aCYINIT("C1@ci|X1@ci?`CIN:C1?`BX"), .aCYSELF("C1?`D"),
    .aDXMUX("F1&!F1@xi?`I1:F1?`IO"),.aF("L1?`LUT"),
    .aF_INST("L1"), .aF_INIT("{16'h}L1@v"),
    .aF5USED("F5&F5!=a?~IO"),.aFFX("F1?`FF"),
    .aFFX_INST("F1"), .aFFX_INIT_ATTR(
      "F1.R|F1.CLR?`INIT0:F1.S|F1.PRE?`INIT1:F1?`INIT0"),
    .aFFX_SR_ATTR(
      "F1.R|F1.CLR?`SRLOW:F1.S|F1.PRE?`SRHIGH:F1?`SRLOW"),
    .aFXMUX("X1?`FXOR:F5=a|F5.O=F1.D|F5&!g?`F5:L1@xo|
      L1.O=F1.D|L1.LO=F1.D?`F"), .aFXUSED("F6&F6!=a?`IO"),
    .aREVUSED("F1.R&F1.S|F1.CLR&F1.PRE|F2.R&F2.S|
      F2.CLR&F2.PRE?`IO"),
    .aSRINV("F1.R@pi|F1.S@pi|F1.CLR@pi|F1.PRE@pi
      |F2.R@pi|F2.S@pi|F2.CLR@pi|F2.PRE@pi?`I1_B:F1.R
      |F1.S|F1.CLR|F1.PRE|F2.R|F2.S|F2.CLR|F2.PRE?`I1"),
    .aSYNC_ATTR("F1.R|F2.R|F1.S|F2.S?`SYNC:F1|F2?`ASYNC"),
    .aXBUSED("C1@xo?`IO"),
    .aXUSED("L1@xo|F5=a&F5@xo|F5@xo&!g|X1@xo?`IO"),
    .aF5MUX("F5"),.aCYMUXF("C1"),.aFAND("A1"),.aXORF("X1"),
                          :
    )
  \F6=a?(F6.O=F2.D?F2.Q:F6.O):F1.Q:F5.O:F2.Q:X1.O:
      L1@xo&L1.LO?L1+{/O}:L1@xo?L1.O:X2.O:L2@xo&L2.LO?
      L2+{/O}:L2@xo?L2.O:C2.LO?C2+{/O}:C2.O:L1@rt?C1.O:
      L1.O:L1.LO?L1+{/O}:L2.O:C1.O (
    .CLK(\F1.C:F2.C ), .F5(\F5@lo?F5.O ),.F1(\L1.IO ),
    .F2(\L1.I1 ),.F3(\L1.I2 ),.F4(\L1.I3 ),.XQ(\F1.Q ),
    .BX(\C1.CI@pi:F5.S:C1@ui?C1.DI:!C1@ci?C1.CI:
      F1@xi?(F1.D@pi:F1.D) ),
    .CIN(\C1@ci?C1.CI:X1@ci?X1.CI ), .XB(\C1@xo?C1.O ),
    .X(\L1@xo?L1.O:F5=a&F5@xo|F5@xo&!g?F5.O:X1@xo?X1.O:
      L1@xo?L1.LO ),
    .CE(\F1.CE@pi:F1.CE@pb:F1.CE:F2.CE@pi:F2.CE@pb:F2.CE ),
    .SR(\F1.R@pi:F2.R@pi:F1.S@pi:F2.S@pi:F1.CLR@pi:
      F2.CLR@pi:F1.PRE@pi:F2.PRE@pi:F1.R:F2.R:F1.S:F2.S:
      F1.CLR:F2.CLR:F1.PRE:F2.PRE )
                          :
    );
endmodule
```

그림 13. SLICEL의 CPIN 매크로 기술 (일부)
Fig. 13. CPIN macro description of SLICEL.

다. “@” 연산자는 단위 셀의 특성을 확인할 때 사용한다. 예를 들어 C1.CI@pi는 CYMUXF의 CI핀에 푸쉬-인 인버터가 연결되어 있는지 확인하는 조건이다. 슬라이스 내의 인버터는 입력을 반전시키거나 비반전 시키도록 재구성할 수 있는데 푸쉬-인 인버터들이 넷 리스트 상에 선택적으로 삽입되어 이들의 상태를 결정한다. 또한, C1@xo는 CYMUXF 의 출력이 슬라이스 외부와 연결되었는지 확인한다. 이밖에 F5=a는 F5MUX가 필수 패킹 그룹의 앵커인지 확인한다.

CPIN 매크로 기술에 의해 논리 단위 셀 넷 리스트가 패브릭 단위 셀 넷 리스트로 변환되면 그 결과가 oaDB 내에 저장되므로 여러 가지 다른 형식으로 출력될 수 있다. 그림 14는 CPIN 매크로 기술에 의해 변환된 SLICEL 인스턴스를 Verilog로 출력한 것이다. 패킹 정보를 근간으로 CPIN 매크로 기술에 따라 핀 연결 및 파라미터 값 지정이 정확히 기술되기 때문에 입력 벡터만 기술하면 별도의 다른 준비 없이 즉시 시뮬레이션을 통해 패킹된 회로 및 패브릭 단위 셀 라이브러리를 검증할 수 있다. Default 값과 동일한 값을 가지는 파라미터는 Verilog 넷 리스트에 출력되지 않도록 하였다.

```
(* mppg = "CC 1:0:0:IFinst.NPC_6_cry_2  " *)
SLICEL #(.aCLKINV(`I1), .aFFX_INIT_ATTR(`INIT0),
    .aFFY_INIT_ATTR(`INIT0), .aFFX_SR_ATTR(`SRLOW),
    .aFFY_SR_ATTR(`SRLOW), .aSYNC_ATTR(`SYNC), .aFXMUX(`F),
    .aDXMUX(`I1), .aF(`LUT), .aF_INST("IFinst.NPC_6_axb_2"),
    .aF_INIT(16'h5f0a),.aFFX(`FF),.aFFX_INST("IFinst.NPC[2]"),
    .aGYMUX(`GXOR), .aDYMUX(`I1), .aG(`LUT),
    .aG_INST("IFinst.NPC_6_axb_3"), .aG_INIT(16'h7b48),
    .aFFY(`FF), .aFFY_INST("IFinst.NPC[3]"),
    .aXORG("IFinst.NPC_6_s_3"), .aBXINV(`I1), .aCYINIT(`BX),
    .aCOUTUSED(`IO),.aCY0F(`PROD),.aCY0G(`PROD),.aCYSELF(`D),
    .aCYSELG(`D), .aCYMUXF("IFinst.NPC_6_cry_2"),
    .aCYMUXG("IFinst.NPC_6_cry_3_0"), .aSRINV(`I1),
    .aFAND("IFinst.NPC_6_cry_2_ma"),
    .aGAND("IFinst.NPC_6_cry_3"))
  \$NPC_c[2] (.CLK(clk_c),.F1(branch_sig_i),.F2(branch_sig_i),
    .F3(NPC_c[2]), .F4(branch_address[2]), .XQ(NPC_c[2]),
    .G1(\IFinst.NPC15 ), .G2(branch_sig_i), .G3(NPC_c[3]),
    .G4(branch_address[3]), .YQ(NPC_c[3]), .BX(GLOBAL_LOGIC0),
    .COUT(\IFinst.NPC_6_cry_3 ), .SR(reset_c));
```

그림 14. SLICEL의 CPIN 매크로 기술에 의해 변환된 넷 리스트의 Verilog 출력
Fig. 14. Verilog description of a slice translated by the CPIN macro description.

Ⅲ. 실 험

개발된 단위 셀 라이브러리와 K-FPGA 패브릭 검증 및 평가 시스템의 검증 과정을 설명하기 위하여 그림 15와 같은 간단한 예제를 선택하였다. 이것은 8-비트의 두 입력 a와 b중 하나를 멀티플렉서로 선택하여 또 다른 8-비트 입력 c와 더하는 MUXADD 회로로서 입출력이 모두 레지스터를 통해 동기화되어 있으며 캐리 체인을 포함하고 있다. 먼저, 이 회로의 RTL 기술을 상용 논리 합성 시스템인 Synplify에 입력하여 논리 단위 셀 넷 리스트를 EDIF 형식으로 출력하고 이를 다시 Xilinx ISE P&R 툴에 입력하여 XDL을 생성시켰다. 이로써 검증 각 단계에서 비교할 수 있는 기준 데이터를 모두 확보한 것이다.

먼저 EDIF (edf2oa) 및 XDL (xdl2oa)을 oaDB에 읽어 들였다. 다음, XDL을 통해 읽어 들인 패브릭 단위 셀 넷 리스트로부터 배치 및 패킹 정보를 추출하여 EDIF를 통해 읽어 들인 논리 단위 셀 넷 리스트에 저

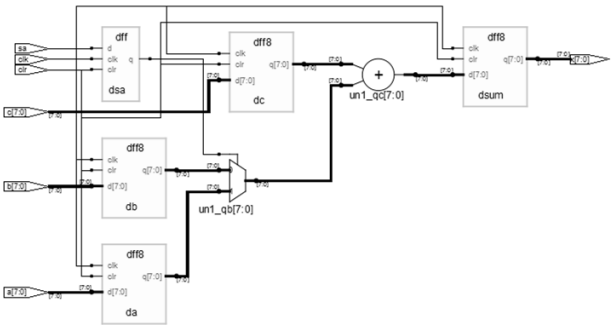

그림 15. 검증 예제 - MUXADD
Fig. 15. Test case - MUXADD.

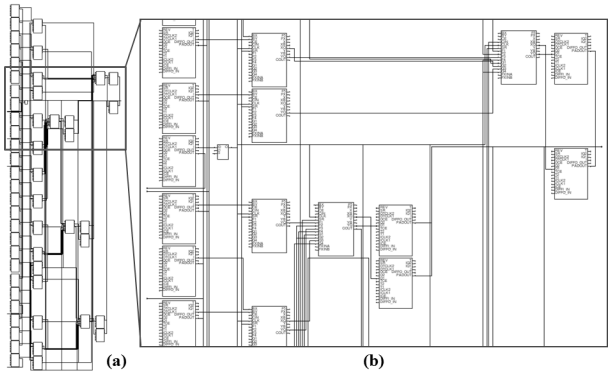

그림 16. 패브릭 단위 셀 넷 리스트로 변환된 MUXADD의 회로도면

Fig. 16. Schematic diagram of MUXADD translated into a fabric unit cell net list.

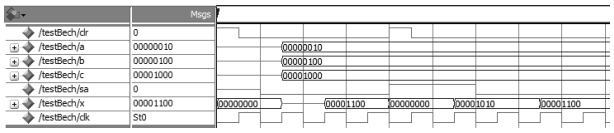

그림 17. 패브릭 단위 셀 넷 리스트로 변환된 MUXADD의 시뮬레이션 결과

Fig. 17. Simulation results of MUXADD translated into a fabric unit cell net list.

장(slice2log)하였다. 저장된 패킹 정보를 근간으로 CPIN 매크로에 따라 논리 단위 셀 넷 리스트로부터 패브릭 단위 셀 넷 리스트를 새롭게 생성(log2slice)시켰다. 그림 16은 이 넷 리스트로부터 자동 생성된 도면이며 일부를 확대하여 보이고 있다. 이것을 XDL로 출력(oa2xdl)하여 Xilinx ISE에서 배선을 오류 없이 수행하는지 확인하였다. 또한, Verilog 형식으로 출력(oa2vlib)하여 시뮬레이션을 수행하였다. 그림 17은 시뮬레이션 결과를 나타내며 두 입력 a=00000010 및 b=00000100를 교대로 c=00001000와 더하는 과정을 보이고 있다. CPIN 매크로에 따라 핀 연결과 파라미터 값 지정이 정확히 수행되며 지정된 파라미터가 패브릭 단위 셀 및 그 하위의 기본 게이트에 전달되어 원하는 동작이 수행됨을 확인할 수 있다.

이 실험은 툴킷(edf2oa, xdl2oa, slice2log, log2slice, oa2xdl, oa2vlib)을 모두 검증할 뿐만 아니라 패브릭 단위 셀 라이브러리도 검증한다. 이 밖에 광폭 입력 함수 멀티플렉서, 분산 메모리, 블록 메모리, 블록 승산기 등이 포함된 다른 대규모 예제도 사용되었으며 모두 성공적으로 수행되어 데이터 구동 방식 툴의 범용성을 확인하였다. 이 변환 툴들은 설계 데이터의 크기에 따라 일차원적인 복잡도를 가지는 문제를 다루기 때문에 예외 없이 모든 경우를 만족시킬 수 있는 기능을 확보하는 것이 중요하며 성능이 크게 문제되지 않는다. 추후 배치 배선 모듈이 완성되면 성능에 대한 비교가 가능하게 될 것이다.

Ⅳ. 결론 및 향후 계획

신규 FPGA 아키텍처로 개발되고 있는 K-FPGA의 경쟁력을 벤치마킹 할 수 있는 툴킷을 개발하였다. 기존 상용 툴과 매핑, 패킹, 배치, 배선 각 단계 별로 데이터를 교환할 수 있어 세부 툴별 비교 평가가 가능하며 이전 단계의 결과물을 기다리거나 결과의 질에 영향을 받지 않으면서 각 단계를 독립적으로 개발할 수 있는 체계를 구축하였다. 또한, 상용 FPGA의 아키텍처를 추출하여 배치 위치 및 배선 그래프 데이터베이스, 그리고 단위 셀 라이브러리를 구축함으로써 FPGA 아키텍처의 신규 개발 시 참조 설계 및 상시 벤치마킹 환경을 제공하도록 하였다. 특히, 아키텍처 정보를 툴 내에 하드 코딩하지 않고 하드웨어 설계자에게 익숙한 표준 HDL 형식으로 기술하여 읽어 들일 수 있도록 함으로써 아키텍처에 수시로 다양한 변경을 시도하면서 최적화해도 툴이 유연하게 수용할 수 있는 데이터 구동 방식의 툴 개발을 추구하였다. 실험을 통해 단위 셀 라이브러리 및 툴 기능을 검증하였으며 임의의 설계가 개발되고 있는 FPGA 아키텍처 상에 탑재되어 정상적으로 동작할지 시뮬레이션 할 수 있음을 확인하였다.

아직 배치 및 배선 툴이 개발 중이다. 역시 칩 아키텍처 기술에 따라 알고리즘이 수행되는 데이터 구동 형식을 취하고 있으며 독립적으로 개발되고 있다. 이들의 결과를 평가에서 공정한 척도로 사용하려면 현재 상용 FPGA 툴킷과 유사하거나 더 높은 성능이 필요하다. 이들이 완성되면 실용적이고 다양한 신규 FPGA 아키텍처들을 개발하고 평가할 수 있게 될 뿐만 아니라 신규 아키텍처에 최적화된 CAD 툴 개발 연구가 활발해지는 시너지 효과를 기대할 수 있을 것이다.

참 고 문 헌

[1] *ABC: A System for Sequential Synthesis and Verification.* Berkeley Logic Synthesis and Verification Group, http://www.eecs.berkeley.edu/~alanmi/abc/abc.html, October, 2007.

[2] V. Betz and J. Rose, “VPR: A New Packing, Placement And Routing Tool For FPGA Research,” in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*. pp.213 - 222, 1997.
[3] T.B. Breuβer and P.G. Spallek, “Enhancing FPGA Device Capabilities by the Automatic Logic Mapping to Additive Carry Chains,” in *Proceedings of the 20th International Workshop on Field-Programmable Logic and Applications*, pp.318-325, September, 2010.
[4] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, M. French, “Torc: Towards Open-Source Tool Flow,” in *Procee0dings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp.41-44, February, 2010.
[5] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, “RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs” in *Proceedings of the 21st International Workshop on Field-Programmable Logic and Applications*, pp.349-355, September, 2011.
[6] *Xilinx Design Language Version 1.6*, Xilinx, Inc., Xilinx ISE 6.1i Documentation in ise6.1i/help/data /xdl, July 2000.
[7] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, “HMFlow: Accelerating FPGA Compilation with Hard Macros for Rapid Prototyping“ in *Proceedings of the 2011 IEEE 19th Annual International Symposium on Field- Programmable Custom Computing Machines*, pp.117-124, May, 2011.
[8] “HqFpga: Unified Platform Solution for FPGA Development Software,” *User Manual for HqFpga 1.5*, Uptops Design Technologies, http://www.uptops-dt.com, Beijing, China, January, 2012.
[9] *Spartan-3 Generation FPGA User Guide*, UG331, v1.6, Xilinx Inc., December 3, 2009.
[10] M. Guiney, E. Leavitt, “An Introduction to OpenAccess: an Open Source Data Model and API for IC Design,” in *Proceedings of Asia and South Pacific Conference on Design Automation*, pp. 434-436, January, 2006.
[11] *Liberty User Guides and Reference Manual Suite Version 2011.09*, Open Source Liberty, http://www.opensourceliberty.org, September, 2011.

저 자 소 개

김 교 선(정회원)
1986년 연세대학교 전자공학과 학사 졸업.
1988년 연세대학교 전자공학과 석사 졸업.
1998년 Ph.D. Department of Electrical & Computer Engineering, University of Massachusetts, Amherst, U.S.A.
1988년～2003년 삼성전자 CAE Center 주임, 선임, 책임, 수석연구원.
2003년～현재 인천대학교 공과대학 전자공학과 부교수
<주관심분야 : 상위수준합성, Reconfigurable Computation, Fault-Tolerance, Embedded Systems, Low-Power Design, Nanoelectronic Architectures, Reconfigurable Architectures>