SoC 설계 공간 탐색을 위한 환경 개발

†안 용 진(安 容 辰), †한 기 성(韓 基 聖), †조 영 철(曺 暎 喆), †유 준 희(柳 俊 熙),
†정 진 용(鄭 鎭 容), †이 강 희(李 康 熙), †최 기 영(崔 起 榮),
††정 의 영(鄭 義 英), ††최 규 명(崔 奎 明)
†서울대학교 전기.컴퓨터공학부, ††삼성 전자 LSI 사업부 CAE센터
전화 : (02) 880-1304-311 / 팩스 : (02) 887-6575
H.P 번호 : 016-211-7705

The Interactive Environment for SoC Design Space Exploration

†Yongjin Ahn, †Keesung Han, †Youngchul Cho, †Junhee Yoo,
†Jinyong Jung, †Ganghee Lee, †Kiyoung Choi,
††Eui-Young Chung, ††Kyu-Myung Choi
†School of Electrical Engineering and Computer Sciences, Seoul National University
††CAE Center, Samsung Electronics Co., Ltd.
E-mail : {ayjin,kshan,rams,ihavnoid,jyjung,berean97}@poppy.snu.ac.kr,
kchoi@azalea.snu.ac.kr,
{euiyoung.chung,kmchoi}@samsung.com

SoC 설계 공간 탐색을 위한 환경 개발

The Interactive Environment for SoC Design Space Exploration

Abstract

Time-to-market pressure and evergrowing design complexity have resulted in the necessity of System-on-Chip demanding an efficient design environment that enables exploring large design space. In this paper, we introduce an interactive environment for fast SoC design space exploration. It takes a Kahn Process Network model in SystemC and transforms it into a graphical model called Hierarchical Module Dependency Graph which is used for estimating HW/SW cost/performance/power. The environment supports simulation-based estimation. For the software, it also supports static estimation. With the environment, we can perform HW/SW partitioning efficiently with accurate estimation, resulting in fast design space exploration, which is our ongoing work.

1. 서론

전자 시스템의 설계에서 SoC (System-on- Chip)는 성능, 전력 소비, 비용, 신뢰도, 크기 등 여러 가지 중요한 관점에서 필수적으로 지향해야 할 것이 되었다. 특히 반도체 공정 기술의 발전으로 그 구현이 가능해짐으로써 많은 사람들이 SoC에 대해 관심을 가지게 되었고, SoC의 효율적인 설계 방법과 이를 뒷받침해주는 tool에 대해서 많은 연구와 개발이 이루어지고 있다. 이러한 방법들은 대체로 프로세서와 기타 HW IP를 포함하는 architecture template에 application을 mapping하는 방법으로 HW/SW 통합설계를 하게 되며, 주로 manual mapping과 evaluation을 통한 system 수준의 최적화를 하고 기존의 synthesis system을 이용해서 SoC를 설계하는 환경을 채택한다. 그러나 이러한 방법은 system designer의 직관에 의존하기 때문에 많은 경험과 숙련도를 요구할 뿐만 아니라 시간도 많이 걸리게 된다. 이러한 문제를 극복하기 위해서는 보다 효율적인 설계 방법론과 tool의 개발이 더욱 중요해지고 있다.

세계 유수의 연구소 및 대학에서 플랫폼 기반 SoC 설계기술에 대한 활발한 연구가 진행되고 있다. 프랑스에 소재하고 있는 TIMA 연구소[1] 내의 SLS (System Level Synthesis) 그룹에서는 다중 프로세서 SoC 플랫폼 상에 주어진 application이 효율적으로 수행될 수 있도록 하는 설계 환경을 개발하고 있다. 이 연구는 SystemC[2]를 입력으로 받아 가상의 architecture를 생성하고 이로부터 prototype SoC를 생성하는 설계 과정을 개발하는 것이 목적이다. 벨기에의 IMEC에서 개발한 OCAPI-xl[3]은 플랫폼을 모델링하고 application을 플랫폼에 mapping하는 도구이다. Architecture는 C++ class library로 구현되어 있으며 functionality를 architecture에 mapping하고 simulation 기반의 성능 분석을 통해 설계공간을 탐색한다.

세계 유수의 설계도구 회사에서도 SoC 설계 도구를 상업화하고 있다. Synopsys사의 CoCentric System Studio[4]는 SystemC 디자인 및 검증 도구이다. 시스템 명세로부터 몇 번의 구체화 과정을 거쳐 상위수준의 언어로 시스템을 구체화하고 이를 검증한 후 구체화된 시스템의 각 block을 architecture에 mapping 한다. Coware사의 ConvergenSC[5]는 플랫폼 기반의 SoC 디자인을 목적으로 하고 있고 GUI (Graphical User Interface)를 통해 플랫폼을 만들고 TLM (Transaction Level Modeling)으로 시스템을 기술 할 수 있다. System 설계자를 위해 다양한 analysis 환경을 제공하고 있고, system 검증을 위해 SystemC simulator와 디버깅 환경을 제공하고 있다. Mentor Graphics의 Platform Express[6] 또한 플랫폼 기반의 SoC 디자인 및 검증 도구로서 GUI를 통해 시스템을 block diagram으로 표현하고, AMBA나 VCI와 같은

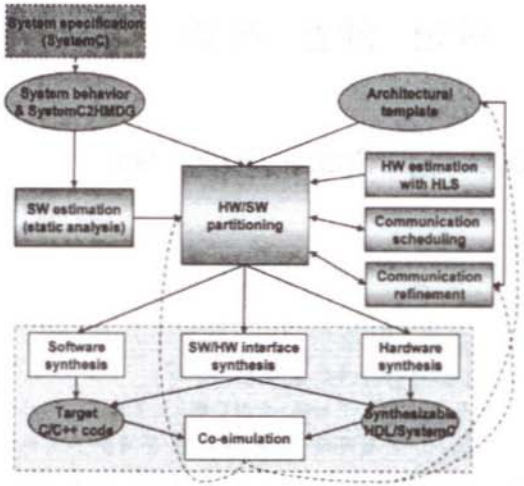

그림 1. 전체 SoC 설계 흐름도.

bus를 사용하여 block들을 연결한다. 검증을 위해 HW/SW 통합검증 도구인 Seamless CVE를 지원한다. 이와 같이, 여러 설계 도구들은 시스템 명세에서부터 SoC까지의 일련의 설계과정 중 전체 혹은 일부분을 지원하고 있다. 그러나 architecture의 선택 및 HW/SW 분할 등은 사용자가 직접 결정해야 하며 도구들은 단지 그 결과의 분석 및 검증에 도움을 준다. 이러한 과정들은 최적화를 위해서 넓은 범위의 설계 공간 탐색이 필요한데, 설계자의 직관에 의존해야 하기 때문에 제대로 최적화하기가 어렵다. 본 논문에서는 기존에 연구되어온 여러 설계 도구들의 단점을 보완하기 위한 SoC설계 방법론을 정립하고, 이를 위해 개발되고 있는 GUI를 통한 설계환경을 제시한다.

제 2절에서 본 논문에서 제안하는 SoC 설계 흐름도에 대해서 언급하고, 제 3절에서 이를 위한 설계 환경 개발에 대해 설명하도록 한다. 제 4절에서 실제 예제를 통한 설계 환경 평가 및 간단한 실험 결과를 보이도록 한다. 마지막으로, 제 5절에서 추후 연구에 대해 논하고 결론을 맺도록 한다.

2. SoC 설계 흐름도

본 논문에서 제안하는 전체적인 SoC 설계 흐름도는 legacy C 코드로부터 시작하는 이전 연구[7]를 보완, 확장한 것이라 말할 수 있다. 그림 1이 그 SoC 설계 흐름도를 보여준다. 먼저, KPN (Kahn Process Network)[8]으로 modeling된 SystemC를 SystemC2HMDG를 이용하여 HMDG (Hierarchical Module Dependency Graph)를 생성하게 된다. 이 과정에서 SystemC code안에 구현이 되어 있는 function들

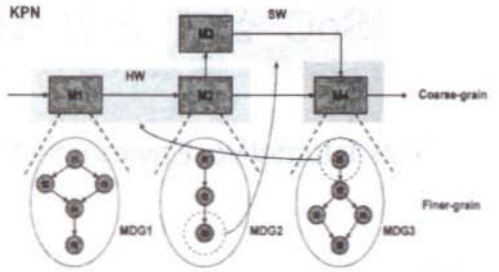

그림 2. HMDG.

이 C code로 생성되며 생성된 C code와 HMDG는 HW/SW estimation을 위해 입력으로 들어가게 된다. SW estimation 과정은 C code를 입력으로 하여 static analysis를 통해 이루어지며 estimation된 결과가 HMDG에 annotation된다. HW estimation 과정은 C code로부터 CDFG를 생성을 하고, 생성된 CDFG를 가지고 최적화과정을 수행하게 된다. 최적화된 CDFG로부터 simulation을 통해 HW의 크기, 성능, 그리고 전력 소모를 estimation하게 된다. SW와 마찬가지로 estimation된 결과가 HMDG에 annotation된다. Annotation된 HMDG를 바탕으로 HW/SW 분할을 수행하게 된다. 이 과정에서 architectural template을 또한 입력으로 받게 되며 communication scheduling[9]을 통해 communication 비용을 구할 수 있다. 이렇게 분할된 결과를 받아 synthesis과정이 이루어지게 되고 simulation을 통해 검증과정이 이루어진다. 본 논문에서는 시스템 명세로부터 효율적이고 정확한 HW/SW 분할을 위한 HW/SW estimation에 대해 주로 다루기로 한다.

2.1. 시스템 명세 및 HMDG

일반적으로 system 모델로서 C를 사용하는 것은 기존의 legacy C 코드를 바로 재사용할 수 있다는 장점이 있지만, system에 존재하는 병렬성을 표현할 수 없다는 큰 단점이 있다. 이러한 문제점에 대한 대안으로 본 논문에서 새로이 도입하려는 것은 SystemC를 이용한 KPN으로 system을 모델링하는 것이다. 또한, SystemC가 표현할 수 있는 여러 계산 모델 중 KPN을 선택한 이유는 KPN이 개념적으로 단순하지만 모델링할 수 있는 범위가 넓으며, 또한 이에 기반을 둔 SoC 설계 환경 구축이 용이하기 때문이다. 또한, KPN의 구현과 관련해서는, KPN의 특성상 각 module들은 수행되는 순서와 무관하게 같은 연산 결과를 만든다는 장점이 있다. KPN은 module들이 크기 제한 없는 FIFO channel을 통해 통신하는 모델이므로 실제 구현

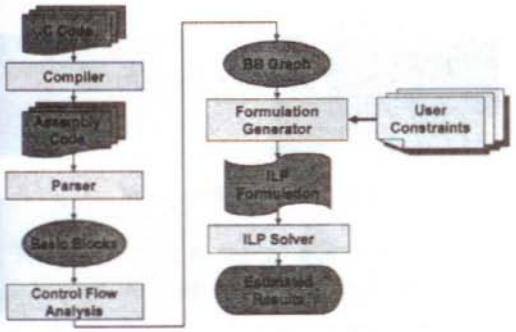

그림 3. SW estimation 흐름도.

과정에서는 FIFO channel의 크기를 제한하는 일이 필요하므로 이를 고려한 communication 구체화 과정이 요구된다.

본 연구에서는 그림 2와 같은 KPN과 이전 연구에서 개발된 MDG (Module Dependency Graph)의 혼합 모델인 HMDG라고 하는 그래프 모델을 제안한다. 그림 2에서 볼 수 있듯이, 기존의 KPN에서는 설계자가 coarse grain의 macro block(process)으로 시스템을 나눠서 명세하게 되므로 이런 block이 HW/SW 분할 단위로 고정되게 된다. 그러나 제안하는 모델을 사용하면 각 block 간의 concurrency를 유지하면서 finer grain의 분할을 할 수가 있다. 예를 들어, 기존의 모델로는 그림 2에서 MDG2의 function f3은 macro block M2가 HW로 분할이 될 때 SW로 분할이 될 여지를 잃게 되는데, HMDG와 같은 모델을 사용하면 f3이 SW로 분할될 가능성을 줌으로써 좀 더 효율적인 설계를 할 수가 있게 된다.

2.2. SW estimation

SW estimation 방법은 simulation 기반의 estimation과 static analysis 기반의 estimation으로 나눌 수 있다. ISS (Instruction Set Simulator)를 사용하는 simulation 기반의 방법은 simulation 시간이 길고, BCET(Best Case Execution Time)와 WCET(Worst Case Execution Time)에 대한 정보를 제공하지 못한다는 단점이 있다. 최근의 embedded system은 real-time system을 요구하는 경우가 많아지고 있기 때문에 이런 real-time 제약을 만족하기 위해서는 BCET과 WCET의 분석이 반드시 필요하다.

본 논문의 SW estimation은 기존의 SYMTA approach[10]를 바탕으로 하는 static analysis 방법을 사용한다. 전체 흐름도는 그림 3과 같다. 입력으로는 C 코드가 주어지며 compiler를 통해 assembly 코드를 생성한다. Parser를 통해 basic block 단위로 코드를 세

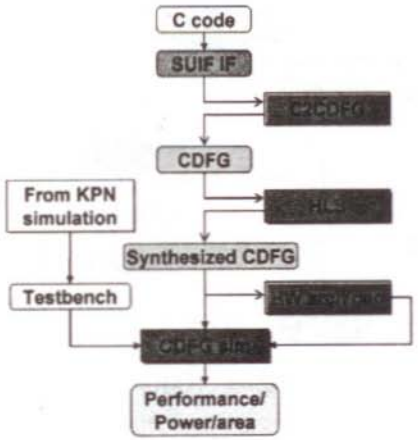

그림 4. HW estimation 흐름도.

분화 한다. Control flow analysis를 통해 basic block 사이의 상관관계를 분석하여 그래프의 형태로 나타내고 data flow analysis를 하여 data의 흐름에 의한 의존관계 및 cache hit 분석을 한다. 이 때 설계자에 의해 일부 변수의 범위 및 loop 횟수 등의 추가적인 정보가 주어지면 보다 정밀한 분석이 가능하다. Data flow analysis까지 마치면 분석 결과는 ILP (Integer Linear Programming) 식으로 표현이 되고 ILP solver를 통해 해를 구하게 된다. 이때, 만약 ILP 식의 목표를 전체 수행시간을 최대화 하는 것으로 두면 WCET가 구해지고, 전체 수행시간을 최소화 하는 것으로 두면 BCET가 구해진다. 전력 소모에 대한 estimation도 수행시간 분석과 비슷하게 최소 전력소모량과 최대 전력소모량을 구한다. 전력소모에 대해서는 평균 전력소모량이 중요하므로 static analysis의 결과는 boundary를 제공하는 정도의 의미를 갖는다.

2.3. HW estimation

Hardware estimation의 flow는 그림 4와 같다. Hardware estimation은 function 단위로 이루어지며, 입력으로 C 코드와 testbench data가 주어지게 된다. 본 연구에서는 SUIF1[11]을 이용해 C 코드로부터 CDFG를 바로 생성하는 툴을 개발하였다. C 코드는 SUIF1을 통해 기본적인 최적화 과정을 거치게 되고 최적화된 SUIF1 intermediate code를 CDFG로 변환하게 된다. 생성된 CDFG는 high-level synthesis[12]과정을 통해 scheduling 및 binding되며, CDFG simulation을 통해서 입력으로 들어온 function의 성능과 크기를 예측하게 된다. 또한, 이를 이용하여 직접 RTL의 HDL 코드를 생성해 내는 것도 가능할 것이다.

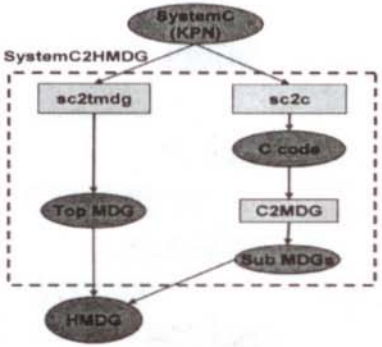

그림 5. SystemC2HMDG 과정.

3. 설계 환경 개발

3.1. HMDG 생성

HMDG는 Top MDG와 Sub MDG들로 구성이 되며 각각 node와 edge들로 구성이 된다. Top MDG의 node는 SystemC code의 SC_MODULE 내부의 SC_THREAD 부분에 해당되며, edge는 그 module에 연결된 port에 해당한다. 특히 각 node는 하위의 function들로 구성된 Sub MDG를 가지게 되며 여기서의 node는 function을 나타내고 edge들은 function들 간의 call 관계를 표현하게 된다. SystemC2HMDG는 SystemC로 구현된 KPN 모델을 입력으로 받아서 HMDG를 생성하기 위한 툴로서 그림 5와 같이 두 단계를 거치게 된다. 첫 번째 단계는 SystemC의 모듈 및 입출력 포트를 lex와 yacc을 사용해서 파싱을 하여 Top MDG를 생성하는 부분으로서 sc2tmdg에 해당한다. 두 번째는 SC_THREAD안의 behavior를 파싱하여 C를 생성하는 부분으로서 sc2c에 해당한다. 여기에서는 SystemC의 내부 표현을 C 언어만 사용하도록 제약을 두고 있기 때문에 sc2c는 어렵지 않게 구현될 수 있다. 이렇게 생성된 C 코드들은 실제로 HW/SW estimation의 입력이 된다. 다시 C 코드들은 C2MDG를 통해 MDG로 변환되어 최종적으로 Top MDG와 합쳐져서 HMDG를 구성하게 된다.

3.2 ILP formulation

SW estimation에서는 instruction memory 내용을 parsing하여 각 function 별로 어떠한 instruction들이 수행되는지를 분석하고 이를 basic block 단위로 나눈다. 각 function에 대해 basic block set을 구하고 control analysis를 통해 basic block 사이의 상관관계를 분석한다. 여기에 loop count에 대한 constraints를

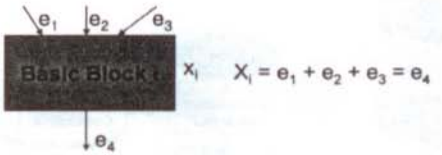

그림 6. ILP formulation의 예.

사용자가 주게 되고 이를 토대로 ILP 식을 생성한다. ILP를 생성하는 기본 개념은 한 basic block의 실행 횟수는 그 basic block의input edge의 수행 횟수를 모두 더한 값과 같으며 이는 output edge의 수행 횟수를 모두 더한 값과 같다는 것을 등식으로 추가하는 것이다. 예를 들어 그림 6과 같이 basic block i의 input edge가 3개 이고 output edge가 하나인 경우를 살펴보자. Basic block i의 수행 횟수를 xi, input edge의 수행 횟수를 각각 e1, e2, e3, output edge의 수행 횟수를 e4라고 하면 xi = e1 + e2 + e3 = e4 라는 식이 유도된다. 이와 같은 방식으로 function의 모든 basic block에 대해 수식을 유도하면 된다. 그림 7은 main function에서 f1 function을 호출하는 프로그램을 그래프로 표현한 것이다. 앞에서 설명한 basic block에 대한 ILP formulation을 적용하면 X1부터 X5까지 5 개의 ILP 식이 나온다. 여기에는 main에서 f1을 호출한 횟수와 f1에서 main으로 돌아가는 횟수가 같다는 조건을 표현하는 식이 없다. e1=e2 라는 수식을 추가함으로써 function call과 return에 대한 조건을 나타낼 수 있다. ILP 식이 모두 생성되면 이를 ILP solver를 이용하여 푼다. ILP의 objective는 BCET의 경우는

$$\min \sum c_i x_i$$

와 같고, WCET의 경우는

$$\max \sum c_i x_i$$

와 같다. ILP solver는 ILOG 사의 CPLEX 8.0[13]을 이용하였다.

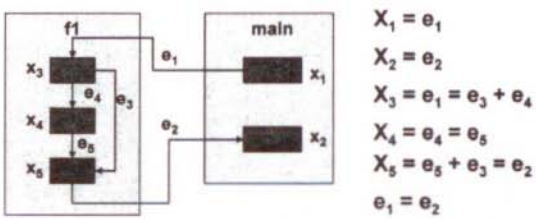

그림 7. Function call의 ILP formulation.

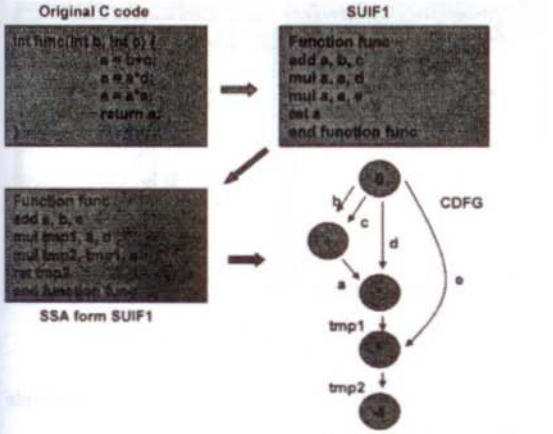

그림 8. C 코드로부터 CDFG로 변환하는 과정.

3.3 C2CDFG

C 코드는 일반적인 C compiler의 preprocessor를 거친 후, SUIF1의 snoot라는 tool을 통해서 SUIF1 format으로 변환된다. Snoot를 통해서 SUIF1로 변환된 C 코드는 전혀 최적화가 되어있지 않으며, 원래의 C 코드의 구조를 그대로 가지고 있다. 예측을 더 정확하게 하기 위해서는, 실제로 설계자가 하나씩 최적화를 할 경우를 고려하여서, 가능한 한 넓은 범위의 최적화가 이루어져야 한다. 이 단계에서는 target에 관계없는 기본적인 최적화들을 한다. dead code elimination, strength reduction, constant propagation 등이 여기에 포함된다. 최적화가 된 SUIF1 intermediate format은 SSA (static single assignment) 형태로 변환된다. 일반적인 C 코드에서의 각 변수는 어떤 memory location을 상징하는 반면, CDFG에서 각 edge는 어떤 data의 흐름을 표시한다. 그렇기 때문에, 하나의 변수가 여러 개의 edge의 역할을 할 수도 있게 된다. SSA form으로 코드를 변환하게 되면, 이러한 경우들에 대하여 쉽게 분석을 할 수 있다. 그림 8은 C 코드로부터 CDFG를 생성하는 과정을 보인다.

3.4 GUI 환경 개발

주어진 시스템 명세로 부터 HW/SW 분할 과정까지 설계자가 쉽고 빠르게 system을 판단할 수 있도록 GUI를 통한 환경을 개발하였다. SystemC 및 C code를 명시한 file을 입력으로 하여 자동으로 SystemC 코드를 HMDG로 변환하여 주며 HW/SW estimation 및

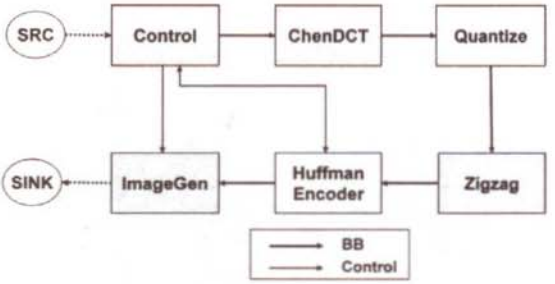

그림 9. JPEG Encoder의 KPN 모델.

분할 과정이 이 환경에서 모두 이루어지고 분할된 결과를 설계자가 쉽게 볼 수 있다.

4. Case study

4.1 JPEG encoder

그림 9는 JPEG encoder에 대한 KPN 모델을 보이고 있다. 이것은 SystemC2HMDG를 통해 HMDG로 바뀌게 되고 C 파일들이 생성된다. 그림 10은 생성된 HMDG의 Top MDG와 Control이라는 이름의 node에 대한 Sub MDG를 보여주고 있다. SW estimation을 위해 우선 생성된 C file들을 armcc를 이용하여 컴파일한다. 그리고 armlink를 이용하여 object file을 만든 후, instruction memory 내용을 parsing하여 각 function 별로 어떠한 instruction 들이 수행되는지를 분석하고 이를 basic block 단위로 나누어 ILP 식들을 생성한다. 한 예로서 HuffmanEncoder의 경우 생성된 변수는 178 개이고 생성된 식은 180개이다. ILP를 푼 결과 WCET은 16,276 cycles, BCET은 1,910 cycles가 나왔다. 실제 armsd에서 여러 다른 input을 가지고 측

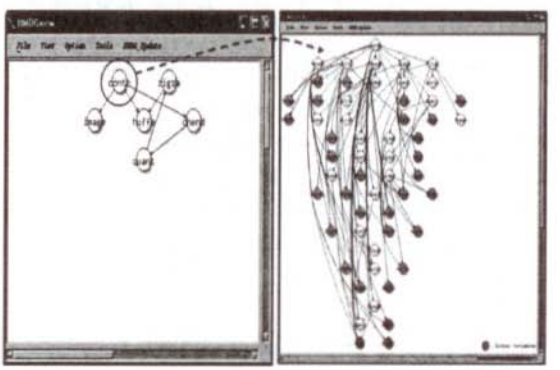

그림 10. JPEG encoder의 Top MDG와 Sub MDG.

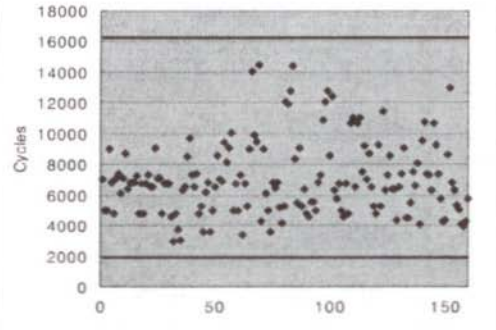

그림 11. HuffmanEncoder의 simulation 결과 분포.

정한 결과는 그림 11에서 볼 수 있다. 그림에서 모든 경우에 대하여 수행 시간이 WCET/BCET의 bound안에 들어옴을 알 수 있다. 따라서 비교적 정확한 estimation이라 할 수 있고, HW/SW 분할을 위한 유용한 정보가 될 것이다.

4.2 H.263 encoder

그림 12는 H.263에 대한 KPN 모델을 만들어 HMDG의 Top MDG와 Sub MDG를 생성한 것을 보여주고 있다. H.263의 경우도 JPEG과 마찬가지로 생성된 HMDG와 C 파일로부터의 HW/SW estimation이 가능하며 현재 실험 중이다.

5. 결론 및 추후 연구

본 논문에서는 효율적인 SoC 설계 방법론과 이를 뒷받침하는 환경 개발에 대한 중요성을 언급하였다. KPN 계산 모델로부터 HMDG라는 그래프 모델을 새롭게 정의하고 자동으로 생성하는 환경을 개발하였고, 또한 SW의 static analysis를 통한 estimation을 통해 WECT/BECT를 구할 수 있었고 C 코드로부터 CDFG를 자동으로 생성하여 high-level synthesis와 simulation을 통해 HW를 estimation하는 환경을 구축하였다. 비록, 현재 HW/SW 분할과 communication 구체화에 대한 연구가 더 진행되어야 하는 상태지만, 빠른 설계 공간 탐색을 위한 전체적인 SoC설계 환경을 마련하는 것은 중요한 일이라 할 수 있을 것이다.

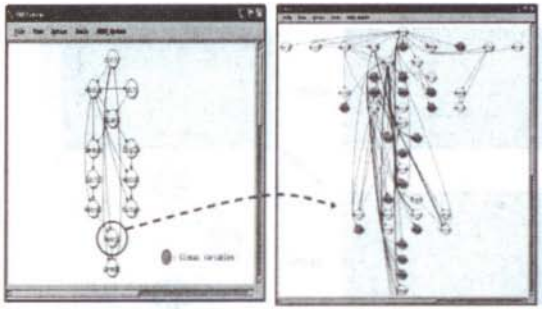

그림 12. H.263의 Top MDG와 Sub MDG.

Reference

[1] TIMA http://tima.imag.fr.

[2] Synopsys, Inc., "SystemC, version 2.0," available at http://www.systemc.org.

[3] OCAPI-xl http://www.imec.be/design/ocapi.

[4] CoCentric data sheet http://www.synopsys.com/products/cocentric_studio/cocentric_studio_ds.pdf.

[5] ConvergenSC™ products http://www.coware.com/portal/page?_pageid=167,123855&_dad=cust_portal&_schema=STAGE.

[6] Platform Express™ http://www.mentor.com/platform_ex.

[7] 한기성, 안용진, 이제형, 곽재화, 유준희, 최기영, "자동화 툴을 이용한 HW/SW 분할 및 사례 연구", CAD 및 VLSI 설계 연구회 학술발표회 논문집, 2003.

[8] G. Kahn, "The semantics of a simple language for parallel programming," in *Proc. IFIP Congress*, 1974.

[9] Y. Cho, G. Lee, S. Yoo, K. Choi, and N. Zergainoh, "Scheduling and timing analysis of HW/SW on-chip communication in MP SoC design", in *Proc. Embedded Software Forum, Design Automation and Test in Europe (DATE)*, 2003.

[10] R. Ernst and W. Ye, "Embedded program timing analysis based on path clustering and architecture classification," in *Proc. ICCAD*, 1997.

[11] SUIF1 homepage http://suif.stanford.edu/suif/suif1/index.html.

[12] J. Jeon, D. Kim, D. Shin, K. Choi. High-level Synthesis under multi-cycle interconnect delay" in *Proc. ASP-DAC*, 2001.

[13] CPLEX 8.0 http://www.ilog.com/products/cplex.