HW/SW간 인터페이싱을 위한 쓰레드 기반 통신 기법

권성남° 주영표 하순회
서울대학교 전기 컴퓨터 공학부
ksn@iris.snu.ac.kr, joo47@iris.snu.ac.kr, sha@iris.snu.ac.kr

The thread-based communication for HW/SW interfacing

Seongnam Kwon° Youngpyo Joo Soonhoi Ha
School of Electrical Engineering and Computer Science, Seoul National University

알고리즘 설계, 하드웨어 설계, 소프트웨어 설계가 순차적으로 이루어지는 기존 설계 방법으로는 복잡한 내장형 시스템을 타임 투 마켓을 지키며 올바르게 동작하도록 개발하는 것은 갈수록 어려워지고 있다. 이것의 대안으로 하드웨어와 소프트웨어를 동시에 개발하는 하드웨어/소프트웨어(HW/SW) 통합설계 방법이 부각 되고 있고 있으며, 과거 연구에서 HW/SW 통합설계 개발환경[1]을 개발하였다. 개발한 환경에서 시스템의 동작 명세를 위해 데이터 플로우 모델의 일종인 SDF(Synchronous Dataflow)모델[2]을 사용한다. SDF 모델에서 하나의 블록은 coarse grain 기능 블록을 나타내며, 입력 데이터 스트림을 받아서 출력 스트림으로 변환한다. 기능 블록의 내부는 C나 VHDL 같은 상위 수준 언어로 기술된다. 노드가 수행될 때 생성하는 혹은 소모하는 sample의 개수를 노드의 출력 혹은 입력 *sample rate*라고 하며 그림 1(a)의 SDF 그래프 예에서 각 아크에 숫자로 표시되어 있다. SDF 모델에서 sample rate은 정수로 정적으로 결정된다.

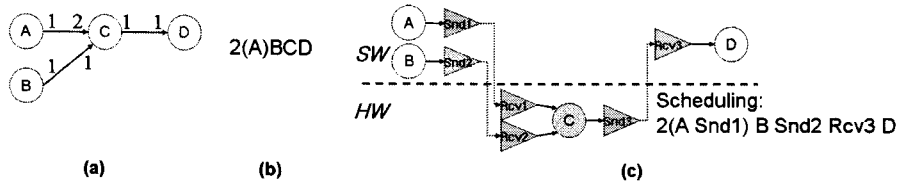

그림 1 (a) SDF 그래프의 예, (b) 스케쥴링 결과, (c) HW/SW 분할된 SDF 그래프와 분할된 SW 스케쥴링의 예

SDF 그래프로부터 코드를 생성하기 위해서 블록의 수행 순서는 컴파일 타임에 미리 결정이 되는데 이를 *스케쥴링*이라고 하며, 그림 1(b)는 스케쥴링 결과를 리스트 형태로 보여주고 있다. 여기서 2(A)는 블록 A가 2번 실행됨을 의미한다. 그림 1(a)의 블록 C가 HW로 분할되면 그림 1(c)와 같이 블록 C와 다른 블록들 사이에 통신 블록(Snd/Rcv)이 삽입되며 소프트웨어 코드 생성을 위해 새로운 스케쥴링을 수행한다. 그 뒤, 블록 C는 VHDL코드로 나머지 블록은 C 코드로 자동 생성하며, 이 때 자동 생성된 HW/SW 간의 통신 코드도 자동 생성한다. 그러나 실제 산업현장에서 내장형 시스템 설계에 많이 사용하고 있는 플랫폼 기반 설계 방법하에서 시스템 성능을 올리기 위한 HW IP가 이미 플랫폼 상에 존재하며, 이것을 이용하려 할 경우 생성된 SW와의 통신은 해당 HW IP와의 통신 방법과 다르기 때문에 문제가 된다. 따라서 데이터 플로우 모델로부터 생성된 SW 코드에서 HW와의 통신 인터페이스를 특정 HW IP에 맞게 생성해줄 필요가 있으며 이것이 이 논문의 초점이다.

<분할되기 전 블록의 위치에 인터페이스 코드 삽입>

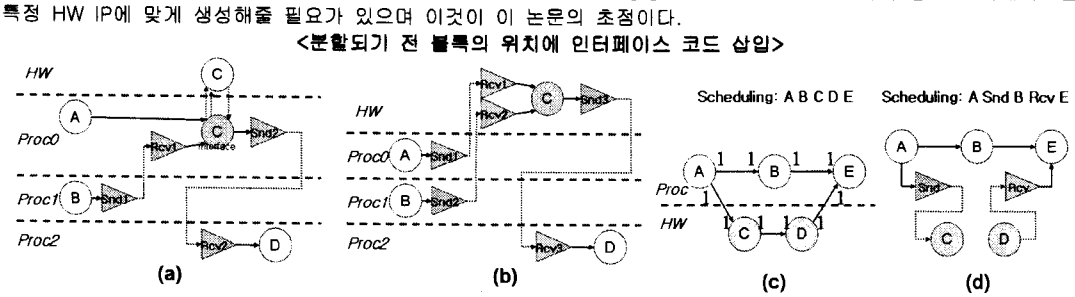

그림 2 (a) 분할되기 전 블록의 위치에 인터페이스 코드를 삽입했을 때의 통신과 (b) 자동 생성된 HW IP와의 통신, (c) SDF 그래프의 예와 (d) 블록 C, D를 HW에 매핑 후 수정한 그래프

SW 코드에서 특정 부분을 HW를 통해 가속하여 SW의 성능을 향상시키고자 할 때, 가장 단순한 방법이 그 부분의 코드 위치에 HW 인터페이스 코드를 삽입하는 방법이다. 예를 들어 그림 1(a)의 예제에서 블록 C가 HW로 분할되어 있다면 HW interfacing 코드는 블록 C 코드의 위치에 삽입이 된다. 이 방법은 매우 간단하지만 성능 면에선 크게 2가지 문제점을 보인다. 첫째로 데이터를 HW에 보내고 완료를 기다리고 결과를 HW에서 읽어오는 동안 SW코드는 기다리고 있어야 한다. 둘째로 멀티프로세서 환경에서 문제가 된다. 그림 1(a)의 예제에서 만약 블록 A, B, D가 서로 다른 프로세서에 매핑 되어 있고 블록 C가 하드웨어에 매핑 되어 있으며, 블록 C에 대한 인터페이스 코드

를 블록 A에 매핑 SW코드에 생성한다고 가정해보자. 이 경우 통신은 그림 2(a)와 같이 이루어진다. 여기서 점선은 프로세서 간 통신을 보여주며, 이것을 위해 통신비용(시간, 메모리 등)이 추가로 필요하다. 이 예제에서 하드웨어와의 통신을 위해 프로세서 1에 매핑 된 블록 B의 데이터를 프로세서 0에 전송하고 하드웨어 C의 결과도 프로세서 0을 통해 프로세서 2로 전송이 되며, 총 5번의 통신이 발생하게 된다. 그림 2(b)는 만약 블록 C가 제안하는 통합설계 프레임웍에서 자동 생성된 HW일 경우 SW/HW간 통신을 보여준다. 그림 2(a)와 달리 데이터를 프로세서 0에 모아줄 필요가 없기 때문에 불필요한 통신이 2번 줄어든다. 그림 2(b)가 그림 2(a)와 통신 측면에서 다른 점은 통신이 기존 블록 C의 위치가 아닌 블록 C의 포트 위치에서 발생한다는 점이다.

<블록의 포트 위치에 인터페이스 코드 삽입>

이 방법은 분할된 블록의 위치에 통신 블록(Send/Receive)을 적절히 삽입하는 방법으로 데이터 플로우 모델을 분할하여 멀티프로세서 코드를 생성하는 연구에서 일반적인 방법이다. 그러나 이 방법은 그림 2(c)와 같이 연결 된 복수의 블록이 동시에 HW로 분할되었을 때 문제가 발생한다. 여기서 블록 C와 D가 HW에 매핑되었다면 블록 A와 C, D와 E 사이에 그림 2(d)와 같이 통신 블록이 삽입될 것이다. 여기서 블록 C의 결과를 블록 D에 전달해야 하는데 SW에 매핑된 블록의 스케쥴링에는 블록 C, D간 통신 정보가 포함되어 있지 않다. 임의로 통신 코드를 삽입할 수도 있지만 복잡한 그래프, 복잡한 매핑을 고려하여 데드락 등의 부가 현상 없이 효율적인 통신코드를 생성하는 것은 매우 어려운 일이다.

<통신 엔진을 이용한 HW/SW 통신>

이전 방법에서 발생하는 문제의 원인은 다양한 인터페이싱 프로토콜을 갖고 있는 HW IP와의 인터페이스를 무리하게 SW 내부에서 처리하려고 했기 때문이다. 따라서 자동 생성된 SW 외부에 통신을 처리하는 프로세스(통신 엔진)를 추가하여 통신만을 담당하도록 하는 방법을 고안하였다. SW와 HW는 직접 통신을 하는 것이 아니라 통신 엔진을 통해서 서로에게 데이터를 전송한다. 하드웨어와의 통신은 통신 엔진이 담당을 하기 때문에 SW내부에서는 통신 코드의 위치와 순서를 고민하지 않고, 데이터가 생성될 때 혹은 필요할 때마다 통신 엔진에 접근하면 된다.
통신 엔진은 별도의 하드웨어로 구현되거나 별도의 프로세서에 전담시킬 수 있다. 또한 프로세서 내부에서 하나의 프로세스 혹은 쓰레드의 형태로 구현될 수 있다. 이 연구에서는 그림 3(a)와 같이 쓰레드 형태로 구현을 하였다. 통신 엔진은 크게 2개의 중첩된 while문으로 구성이 되어 있다. SW쪽에서 데이터가 전송되어야(2번 줄) 안쪽 while 문으로 들어가게 되며, 그 전에는 통신엔진을 재워놓음으로 부담을 최소화 했다. 안쪽 while문은 크게 두 부분(4-12줄, 13-18줄)으로 구성된다. 앞부분은 현재 동작하고 있는 HW IP가 있는 경우 각 HW 수행이 종료되었는지 체크하는 역할(***waitCode()***: 6번 줄)을 수행하며, 뒷부분은 각 HW IP를 동작시키는데 필요한 데이터가 SW로부터 통신엔진에 다 전송되었는지 체크하고(***checkFiringCondition()***: 14번 줄), 그렇다면 데이터를 전송하고 동작시키는(***firingCode()***: 15번 줄) 코드로 구성이 되어 있다. 이 세 함수는 각 HW IP의 특성에 맞게 사용자가 기술해 주어야 한다. 그림 3(b)와 (c)는 HW Motion Compensation(MC)와 IDCT를 위한 통신 코드의 예를 보여준다. 이 코드는 각각 VERSATILE[3] 보드와 SoCBase[4] 가상 프로토타입 환경에서 올바르게 동작함을 확인하였다.

```
1.  while(1) {
2.    wait for "produceSampleSW()"
3.    while(1) {
4.      for each block i {
5.        if (block i is running) {
6.          while((block i).waitCode()==false) {
7.            sleep
8.          }
9.          block i <= wait
10.         break;
11.       }
12.     } // end of for
13.     for each block i {
14.       if ((block i).checkFiringCondition())==true) {
15.         (block i).firingCode()
16.         block i <= running
17.       }
18.     } // end of for
19.     if (there is no fired block) break;
20.   } // end of inner while
21.} // end of outer while
```

(a)

```
1.  waitCode {
2.    static int count = 0;
3.    unsigned char * frame;
4.    if (count!=98) { count++; return 1; }
5.    else count = 0;
6.    frame = hw_ptr_read();
7.    read data from frame;
8.  }
```

```
1.  checkFiringCondition {
2.    if (all data is prepared for HW MC)
3.      return 1;
4.    else return 0;
5.  }
```

```
1.  firingCode {
2.    hw_write();
3.  }
```

(b)

```
1.  waitCode {
2.    return 1;
3.  }
```

```
1.  checkFiringCondition {
2.    if (all data is prepared for IDCT)
3.      return 1;
4.    else return 0;
5.  }
```

```
1.  firingCode {
2.    for Y, U, and V component of frame {
3.      Send Data
4.      Send Control Signal
5.      wait the completion of IDCT
6.      Receive Data
7.    }
8.  }
```

(c)

그림 3 (a) 쓰레드 형태의 통신엔진 의사코드, (b) HW MC를 위한 인터페이스 의사 코드의 라이브러리 파일, (c) HW IDCT를 위한 인터페이스 의사 코드의 라이브러리 파일

감사의 글 - 본 연구는 BK21 프로젝트, 과학기술부 도약연구지원사업(R17-2007-086-01001-0)에 의해 지원되었다. 또한 서울대학교 컴퓨터신기술연구소와 IDEC은 본 연구에 필요한 기자재들을 지원해 주었다. 본 연구는 또한 한국전자통신연구원의 SoC 핵심설계인력양성사업에 의해 부분적으로 지원되었다.

[1] PeaCE official homepage, http://peace.snu.ac.kr
[2] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous dataflow programs for digital signal processing," IEEE Trans. Computer, vol. C-36, pp. 24-35, Jan. 1987
[3] Versatile platform baseboard for ARM926EJ-S product site, http://www.arm.com/products/DevTools/VPB926EJ-S.html
[4] S. Park, S. Chae, "SoCBase: An integrated solution for platform based design," ISOCC'04, pp. 329-332, Oct. 2004.