論文98-35C-11-4

32비트 부동소수점 DSP의 Cycle Based Simulator에 관한 연구

(A Study on Cycle Based Simulator of a 32 bit floating point DSP)

禹 鍾 植 *, 梁 海 龍 **, 安 哲 弘 ***, 朴 柱 成 ****

(Jong Sik Woo, Hae Yong Yang, Cheol Hong An, and Ju Sung Park)

요 약

본 논문에서는 C 언어로 코딩된 32 비트 부동소수점 DSP(Digital Signal Processor)의 CBS(Cycle Base Simulator)의 설계에 관한 내용을 다룬다. 개발된 CBS는 TMS320C30과 호환되는 DSP 설계를 위한 것으로 VHDL로 게이트 레벨 설계에 앞서 DSP의 구조, 기능블록의 동작, 제어신호 등을 확정하는 데 사용된다. CBS는 상용 시뮬레이터에서는 제공되지 않는 각 파이프라인 스텝에서의 제어신호, 주요 기능 기능블록의 값, 버스 및 레지스터의 값을 알려주므로 게이트 레벨 설계시 중요한 레퍼런스가 된다. 이러한 주 기능 외에 CBS의 효율적인 수행과 결과 확인을 위하여 여러 가지 인터페이스 기능이 추가되었다. CBS의 동작의 검증은 여러 알고리즘에 대하여 상용 시뮬레이터의 결과 비교를 통하여 이루어졌으며, 전체 DSP의 시뮬레이션 속도는 VHDL을 통한 로직 시뮬레이션보다 수십 배가 빠른 것을 확인하였다. 본 연구에서 만든 CBS는 특정 DSP를 위한 것이지만 그 개념은 다른 VLSI 설계에 응용될 수 있을 것이다.

Abstract

This paper deals with CBS(Cycle Base Simulator) design of a 32 bit floating point DSP(Digital Signal Processor). The CBS has been developed for TMS320C30 compatible DSP and will be used to confirm the architecture, functions of sub-blocks, and control signals of the chip before the detailed logic design starts with VHDL. The outputs from CBS are used as important references at gate level design step because they give us control signals, output values of important blocks, values from internal buses and registers at each pipeline step, which are not available from the commercial simulator of DSP. In addition to core functions, it has various interfaces for efficient execution and convenient result display. CBS is verified through comparison with results from the commercial simulator for many application algorithms and its simulation speed is as fast as several tenth of that of logic simulation with VHDL. CBS in this work is for a specific DSP, but the concept may be applicable to other VLSI design.

* 正會員, (株) 보이소半導體
(VOISO Semiconductor Co., Ltd)
** 正會員, 現代電子(株)
(Hyundai Electronics Co., Ltd)
*** 正會員, 三星電子(株)
(Samsung Electronics Co., Ltd)
**** 正會員, 釜山大學校 電子工學科
(Dept. of Electronics Engineering, Pusan National University)

※ 본 연구는 산업자원부, 과학기술부, 정보통신부에서 시행하는 주문형반도체 개발사업의 지원을 받아 수행되었습니다.

接受日字:1998年2月9日, 수정완료일:1998年10月26日

Ⅰ. 서 론

반도체 공정과 설계기술이 발전함에 따라 다기능을 가지는 수백만 게이트 급의 VLSI가 구현되고 있으며 이러한 다기능, 대용량의 VLSI를 설계하는 경우 설계기간의 대부분을 기능설계에 보내고 있다. 그 중에서 분석과 합성에 소요되는 설계기간보다는 검증단계에서 아주 많은 시간을 소비하게 된다. Time to Market의 정책을 가장 중요시하는 ASIC (Application Specific Integrated Circuits)에서는 설계시간 단축이 최상의 요구조건이 되며 이는 검증시간의 단축을 통하여 이루어질 수 있다.

수십만 게이트급의 DSP나 ASIC을 설계함에 있어서 검증시간을 줄이기 위해서는 게이트레벨 설계와 같은 구체적인 작업이전에 설계될 VLSI의 구조, 명령어, 주변과의 인터페이스 등을 종합적으로 검토할 수 있는 방안이 필요하다. 본 논문에서는 TMS320C30과 호환되는 DSP를 설계함에 있어 검증시간을 단축시키는데 도움을 준 CBS에 관한 내용을 다룬다. 설계될 칩은 32비트 부동소수점 연산을 취하고 4단계 파이프라인 구조를 가지며, 113개의 명령어 세트를 갖는 DSP이다. 제작된 시뮬레이터는 DSP의 파이프라인 사이클을 기반으로 내부구조가 모델링되었다. 이런 연유로 구현된 시뮬레이터를 Cycle Based Simulator라고 명명한다.

구현된 CBS는 다음과 같은 장점이 있다. 첫째, C 언어로 프로그래밍 되었기 때문에 VHDL 시뮬레이션 시간의 수십 분의 일에 해당하는 시간에 시뮬레이션을 행할 수 있어서, 수천 사이클의 응용프로그램을 10분대에 검증을 완료할 수 있는 장점이 있다. 이는 복잡한 알고리즘을 여러개 실행시켜 호환성을 검증해야하는 경우는 설계오류에 의한 turn around time을 획기적으로 감소시키는 결과를 가져온다. 둘째, 상용시뮬레이터가 제공하는 정보를 포함해서 내부의 여러 버스 값, ALU 및 승산기의 결과, 모든 레지스터 값, 제어에 필요한 마이크로 프로그램의 데이터 값을 제공해주기 때문에 논리회로 설계시 중요한 레퍼런스로 활용될 수 있다. 셋째, VHDL과 C 언어와의 유사성으로 C 언어로 정의되어 있는 CBS를 짧은 시간 내에 VHDL로 변환할 수 있다. 그 외에도 VHDL & 게이트 레벨 시뮬레이션 과정에서 시뮬레이션을 수행하는 툴과의 interface를 통해서 사람의 개입 없이 프로그램의 실행에 의해 결과 값들을 비교하고 오류가 발생한 경우에 한해서 알려주는 방식을 택할 수 있다. 본 논문에서 제작한 CBS는 호환 DSP 설계의 모든 단계에 사용되면서 설계시간을 계속적으로 줄일 수 있는 기능을 가진다.

본 논문은 Ⅱ장에서 CBS를 포함한 호환칩 설계의 전반적인 과정을, Ⅲ장에서 CBS의 코어부를 지원하는 주변 환경을, Ⅳ장에서는 CBS 제작방법을, Ⅴ장에서 CBS의 검증에 관한 내용과 결론을 내린다.

Ⅱ. CBS를 포함한 전체 설계 흐름

CBS의 제작은 전체 DSP 설계의 일부이므로 전체적인 설계흐름을 이해하는 것이 CBS를 이해하는데 도움이 된다.

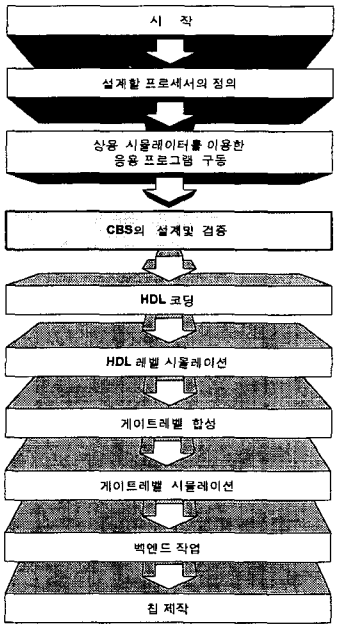

그림 1. 전체 칩 제작의 흐름도
Fig. 1. Flow chart of entire chip fabrication.

이 장에서는 전체적인 설계흐름 중에서 CBS 제작을 위한 준비과정과 각 단계에서 CBS의 역할에 중점을 두어 설명한다. 대상으로 하는 DSP를 설계하고 칩으로 만들기 위해서는 그림 1과 같은 흐름을 따른다. 그 중에서 CBS 제작은 VHDL 코딩작업 앞에 위치함을 볼 수 있다. 목표로 하는 DSP를 정확하게 이해하기 위하여 사용자 매뉴얼을[1] 숙독한 후 상용 시뮬

레이터를 사용해서[2] [3] 여러 응용 프로그램을 수행시켜 본다[4]. 이 결과는 이후 모든 과정에서 검증을 위한 참고자료로 사용된다. 이 과정이 끝나면 DSP를 모델링 하게 되는데 이 과정이 그림 2에 나와있다. 실제 프로세서의 설계와 같이 기능블록의 모델링, 그리고 제어신호의 모델링 순서를 따르게 된다.

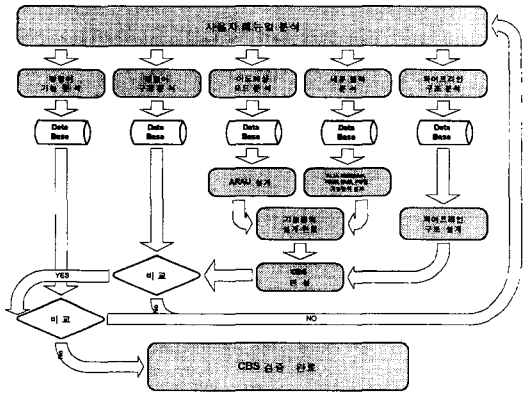

그림 2. CBS의 설계와 검증과정
Fig. 2. The design and verification flow of CBS.

상용 시뮬레이터와 본 연구에서 작성한 CBS의 차이점을 표 1에 요약했다. CBS가 제작되고 검증된 이후에는 모든 설계의 기준으로 사용된다.

표 1. 상용, behavioral, cycle based 시뮬레이터의 차이
Table 1. The difference of commercial, behavioral, cycle based simulator.

	상용 시뮬레이터	CBS
목 적	Software 개발자를 위한 환경 제공	DSP 내부 구조의 정확한 정의
실행단위	명령어 실행	파이프라인 사이클
어셈블러, 링커	외장	내장
제어신호	확인 불가	확인 가능
내부버스	확인 불가	확인 가능
ALU, MUL 입.출력	확인 불가	확인 가능
C 인터페이스	有	無
Memory Map	PC 메모리의 한계로 50K 이상 할당 불가	16M 워드의 메모리 할당 가능
out file gen.	無	有
컴파일 비트수	32비트	64비트

VHDL[5]과 CBS를 작성한 C 언어는 모두 상위 레벨 언어라는 점에서 많은 유사점이 있다. 그러므로 CBS 코딩시 VHDL로의 변환을 미리 고려해두면, C 언어에서 VHDL로 변환에 많은 시간이 걸리지 않는다. 본 논문에서는 이러한 점을 고려하여 다음과 같이 코딩하였다.

- 설계될 하드웨어는 입. 출력을 가지는데 이를 CBS 작성시 입력 변수와 리턴 값으로 모델링 한다. 여러 개의 출력을 가지는 경우는 구조체(structure)로 정의한다.
- 입. 출력 변수의 이름에 하드웨어 설계를 고려한 이름을 붙인다. 신호의 의미 외에 설계시 사용될 비트수, 제어신호와 입. 출력 신호의 구분 등을 한다.
- VHDL로의 변환시 변환이 용이한 조건 문을 사용해서 모델링한다.

C 모델을 기초로 해서 VHDL을 작성한 후, 그리고 이를 게이트 레벨로 합성한 후에 원하는 결과를 가지는지에 대한 검증 과정을 필요로 한다. 이 단계의 게이트 레벨 시뮬레이션은 상용 툴을 이용하였다. 이 때 제작된 CBS를 다시 사용하게 되는데, 상용 툴과 CBS를 인터페이스 하는 프로그램을 작성해서 사람의 개입 없이 두 모델의 결과를 비교할 수 있게하여 편리하게 사용하였다.[6] [7] [8]. 본 연구에서 사용된 게이트 레벨 시뮬레이션 툴로 Synopsys를 사용했다.

Ⅲ. CBS의 주변 인터페이스 제작

본 논문에서 구현한 CBS는 어셈블 코드를 입력으로 하고 파이프라인 사이클마다 DSP의 내.외부 데이타를 출력하는 것을 목표로 했다. 상용 링커와 어셈블러는 PC에서 실행되고 CBS는 워크스테이션에서 실행되기 때문에 서로 다른 환경을 오가는 불편을 없애기 위하여 자체적으로 링커와 어셈블러를 만들었다. CBS 실행 결과의 효율적인 확인을 위해서 GUI (Graphic User Interface)를 제작했다. 그림 3에 CBS를 중심으로 개발된 인터페이스들이 어떻게 연결되는가를 나타내고 있다.

1. 제작된 어셈블러, 링커의 특징

상용 어셈블러와 링커는 DSP를 응용하기 위하여 개발된 프로그램을 기계어로 바꾸는 역할을 한다. 어셈블러는 프로그램을 목적 코드로 변환한 후에 링커가 실행 코드로 변환하는 순서를 따른다. 그러나 제작된 인터페이스환경에서는 링커가 어셈블러보다 먼저 쓰이

면서 프로그래머가 사용한 레이블과 심볼을 어셈블러가 인식할 수 있는 코드로 변환한다. 이러한 순서를 택한 이유는 GUI 환경으로부터 CBS 수행에 필요한 어드레스나 데이터를 직접 받아들이기 위해서 이다. 링커는 상용으로 쓰이는 모든 레이블을 지원하지는 않으며 12가지의 자주 쓰이는 레이블만을 지원한다.

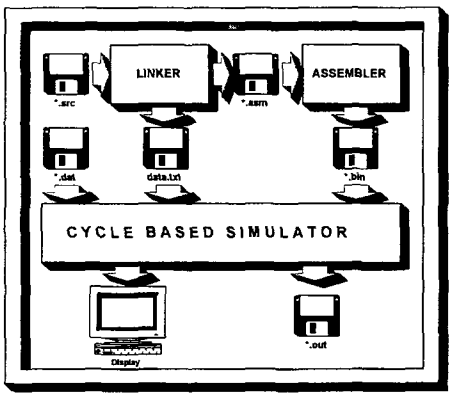

그림 3. CBS의 외부 인터페이스 블록 다이아그램
Fig. 3. Block diagram of CBS external interface.

2. 링커(Linker)와 어셈블러의 제작

일반적으로 링커의 기능은 어셈블러를 수행해서 나온 목적 코드(Object Code)를 명령파일(Command File)에 정의된 각 코드들의 절대 어드레스를 가지고 있는 실행 코드로 변환하는 것이다. 하지만 CBS에서는 명령 파일의 구조를 택하지 않고 GUI 화면을 통하여 직접 어드레스를 받아서 링커를 통해서 절대 어드레스를 가진 어셈블 코드를 만들어 내게 된다.

CBS에서 어셈블러의 역할은 링커에서 만들어진 어셈블 파일을 실행 가능한 이진 파일로 바꾸는 것이다. 이것이 가능하기 위해서 어셈블 파일을 입력으로 해서 주석을 구분하고 명령어를 비교하며, 어드레싱 모드에 따른 코드를 만들어 내는 일을 반복적으로 수행하도록 했다.

3. GUI 제작 및 기능

CBS로부터 출력되는 데이터로는, 여러 종류의 버스 값, 제어신호와 기능블럭의 입출력 데이터, 레지스터와 메모리의 값 등이다. 이러한 여러 가지 데이터를 텍스터 형태로 출력해서 확인하는데는 많은 어려움이 있다. 이러한 문제점을 극복하고 CBS를 보다 편리하게 수행시키기 위하여 GUI를 제작했다. GUI 제작에 사용된 라이브러리는 Unix용 MOTIF를 사용했으며 X11, Xt, Xm 등의 세부 Library를 link해서 함께 사용하였다[9]. 제작된 CBS의 GUI 화면은 그림 4와 같다.

Source Display Window는 현재 수행중인 source program code를 표시해주는 곳으로, 어떠한 명령어가 Execution, Read, Decode, Fetch 단계에 있는가를 구분해주고 있다. Push Button 영역에는 CBS 실행과정에서 필요한 여러가지 메뉴를 모아두고 있다. 대표적 메뉴로 Load, Assemble, Output File, Memory Display, Break Point 등을 들 수 있다. Register Display 영역은 DSP 내부의 모든 레지스터 및 버스 값들을 나타내준다. 이 영역은 상용시뮬레이터에서 제공되지 않는 많은 정보를 갖고 있기 때문에 설계시에 많은 도움이 된다. Memory Display 영역은 두개로 되어 있으며 나타내고 싶은 영역을 Push Button을 이용하여 지정할 수 있다. Text Editor 영역은 소스코드를 실행하는 동안에 에러가 발생할 경우에 소스코드를 수정하고 저장할 수 있게 해준다.

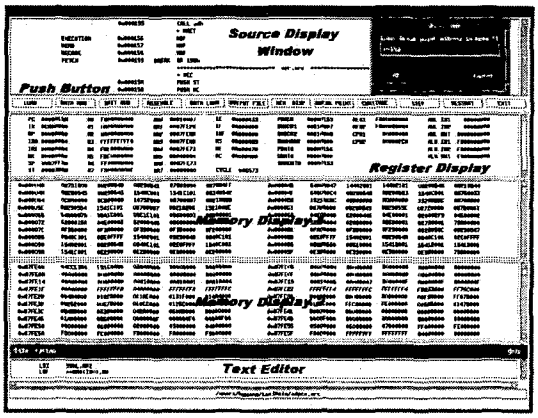

그림 4. CBS의 GUI 화면
Fig. 4. GUI Screen of CBS.

Ⅳ. CBS의 제작

1. 제작 환경

PC의 메모리와 변수 데이터크기의 한계로 인해 Unix를 OS로 하는 workstation에서 프로그래밍 했다. C 언어[10]를 사용했으며 TMS320C30 DSP의 extended register(40비트)를 표현하기 위해 64비트 정수포맷을 제공하는 GCC C 컴파일러를 사용했다.

2. 제작 흐름

그림 5에 CBS 제작과정을 나타냈다. 제작에 앞서 DSP에 관한 여러 가지의 데이터 베이스가 정리되어

있어야 한다. 이 데이터 베이스는 CBS의 제작 중에 계속 갱신된다. 가장 먼저 해야 할 일은 각 기능 블록에 필요한 제어 신호를 정의하는 일이다. 처음에는 생각하지 못했던 제어신호들이 CBS의 제작과 검증을 통해서 많이 생기게 된다. 제어신호가 정의가 되면 기능블록을 상위레벨 언어로 모델링 한다. 각 블록의 내부는 하드웨어의 계층적 설계를 고려해서 다시 여러 단계의 하위 블록으로 구분될 수 있다. 모델링 된 각 블록이 원하는 동작을 하는지를 여러 조건을 인가함으로써 확인한다. 각 블록의 동작 확인이 완료되면 파이프라인 사이클별로 하나의 함수를 만들고 원하는 동작을 수행하는지 확인한다. 그 다음 과정은 파이프라인 4단계를 하나로 붙이고 전체 검증을 한다.

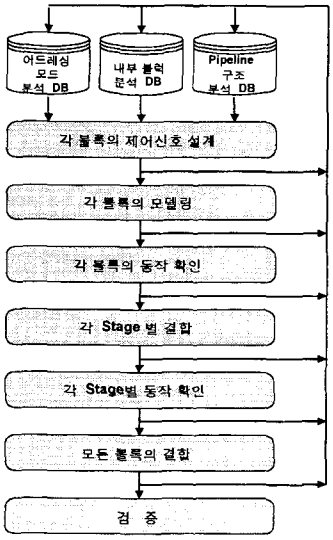

그림 5. 시뮬레이터 내부설계 흐름도
Fig. 5. Flow chart of simulator design.

3. 기능 블록의 C 모델

대상 프로세서에서 쓰일 각 기능 블록들을 모델인하는 단계이다. 본 논문에서 대상으로 삼은 TMS320C30의 블록들을 100여개의 함수로 모델링 했다. 모델링은 다음의 세 단계의 순서를 따랐다.

■ 동작 및 입. 출력 신호의 정의

모델을 생성할 블록의 동작과 그에 따른 입. 출력 신호와 제어 신호를 정의한다.

■ 하드웨어 모델의 설계

이미 정의된 입. 출력을 이용해서 원하는 동작을 수행할 수 있도록 하드웨어 설계를 고려하여 대략적인 모델을 생각한다. 그 예로서 그림 6에 설계될 DSP 내부의 확장 레지스터(Extended Registers) 블록의 하드웨어 모델의 개념도를 나타냈다. 각 레지스터는 게이트 레벨까지 생각한 반면 레지스터의 read/write 제어 신호를 만들어 내는 부분은 블록으로 모델링하고 내부에 들어가야 할 기능들을 대략적으로 생각해둔다.

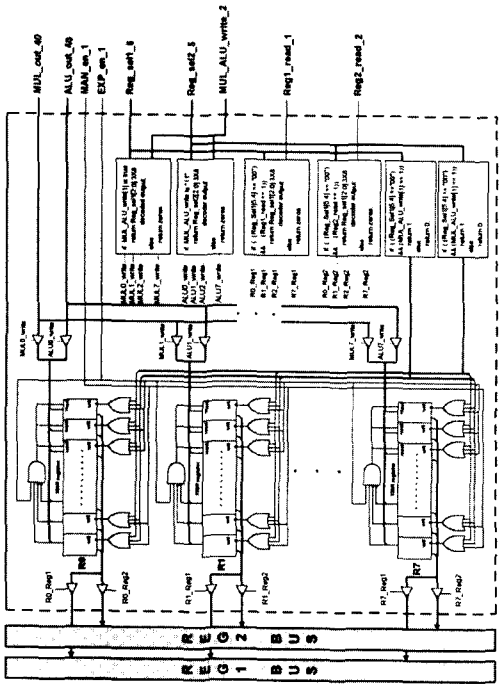

그림 6. 확장 레지스터의 하드웨어 모델의 개념도
Fig. 6. Harware Model Concept of Extended Registers.

```
/* decoder_38 function */
void R_register( unsigned long MUL_out_in40, ... )
{
    unsigned long data0=0, data1=0, ... ;
    ...
switch(in_dec1(Reg_sel1_in5,MUL_ALU_write_in))
{
    case 0x1 : data0 = MUL_out_in40; break;
        ...
}
switch(in_dec2(Reg_sel1_in5, ... ))
        ...
switch(out_dec00(Reg_sel1_in5,MUL_ALU_write_in2))
        ...
switch(out_dec01(Reg_sel2_in5,MUL_ALU_write_in2))
        ...
out0=R_reg(data0,write0,0,EXP_en_in1,MAN_en_in1);
        ...
out7=R_reg(data7,write7,7,EXP_en_in1,MAN_en_in1);
switch(out_dec1(Reg_sel1_in5,Reg1_read_in1))
{
    case 0x1 : REG1_BUS = out0; break;
        ...
}
switch(out_dec2(Reg_sel2_in5,Reg2_read_in1))
        ...
}
```

그림 7. 확장 레지스터의 C 코딩 개념
Fig. 7. C Coding Concept of Extended Registers.

■ C 언어로 코딩

하드웨어 모델이 완성되면 이를 C 언어로 변환한다. 각각의 세부 블록을 C 언어를 사용하여 그 동작을 기술한다. VHDL로의 변환을 고려한 코딩기법은 II장에서 언급한 바 있다. 그림 6에 나타낸 확장레지스터가 CBS에서 어떻게 코딩되었는가를 그림 7에 나타냈다.

4. 파이프라인 구조 구현

설계할 DSP는 4단계 파이프라인 구조를 가지고 있으므로 각 블록의 코딩과 전체 코딩시 파이프라인 구조에 대하여 충분한 이해가 필요하다. 그림 8에 설계할 DSP의 파이프라인을 고려한 CBS의 파이프라인 구조를 보여주고 있다. 각 단에서는 순차적으로 들어오는 명령어들에 대하여 주어진 기능을 수행하고 그 결과를 버퍼에 저장한다. 그리고 각 단에는 각기 다른 마이크로 ROM을 가지고 제어하도록 한다. 완성된 블록들을 이용하여 DSP 4단계 파이프라인의 기능을 수행하는 fetch(), decode(), read(), execution() 네 함수를 만들었다. 각 함수는 내부 블록과 제어 블록을 모두 포함한 함수로써 블록들을 신호의 흐름을 따라 순차적으로 실행해 나가는 형태로 만들었다. 네 함수 중에서 가장 간단한 Fetch() 함수의 대략적인 구조를 그림 9에 나타냈다.

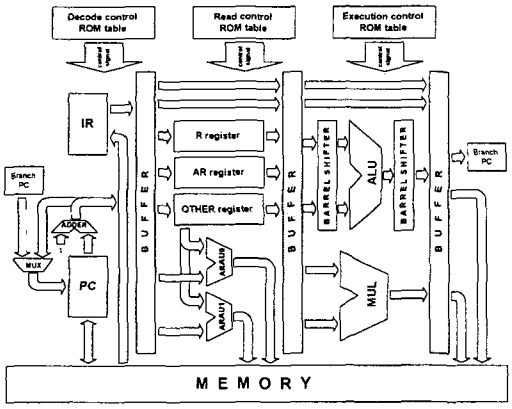

그림 8. CBS 내부의 파이프라인 구조
Fig. 8. Internal pipeline structure of CBS.

하드웨어와 C 언어로 만들어진 프로그램의 가장 큰 차이점은 하드웨어는 병렬로 동작하는 반면 프로그램은 순차적으로 동작한다는 것이다. C언어로 코딩된 CBS가 하드웨어가 동작하듯이 파이프라인 동작을 하기 위해서 네 함수를 execution -> read -> decode -> fetch 순서로 실행시켰다. 이러한 순서로 함수를 실행시킴으로써 각 파이프라인 수행결과를 저장하는 버퍼의 데이터 값의 갱신에 의한 오동작을 막을 수 있다. 함수처리 순서를 바꿈으로써 파이프라인 효과를 가져올 수 있는 것을 표 2를 이용하여 다음과 같이 설명할 수 있다. Add라는 명령어는 n+2 사이클에서 READ된 operand를 n+3 사이클에서 EXE (Add)하므로 n+3 사이클에서 Add 명령어의 execution() 함수를 맨 먼저 수행하고 그 다음에 n+3 사이클의 Sub 명령어의 read() 함수를 수행시키면 read() 함수의 결과가 버퍼에 들어가도 이미 수행된 Add 명령어의 결과에는 영향을 주지 않는다. 따라서 n+3 사이클에서 execution -> read -> decode -> fetch 순서로 함수를 실행시키면 파이프라인 구조를 구현할 수 있다.

```
/* fetch function */
void fetch(unsigned PC_read_in, ...)
{
PC(PC_inc_disable_in, ...)
{
  PADDR_BUS = pc_content;
  if(PC_inc_disable_in == 0 )
    PC_content++;
  else
    PC_content = PADDR_BUS;
  ...
}
MEMORY(PADDR_BUS, ...)
{
  RAM0_BLOCK(PADDR_BUS_in, ...)
  {
    RAM0_address=mem_mux41(PADDR_BUS, ...)
    ...
  }
  RAM1_BLOCK(PADDR_BUS_in, ...)
  INTERRUPT_BLOCK(PADDR_BUS_in, ...)
  EXTERNAL1_BLOCK(PADDR_BUS_in, ...)
  EXPANSION_MSTRB_BLOCK(PADDR_BUS_in, ...)
  EXPANSION_IOSTRB_BLOCK(PADDR_BUS_in, ...)
  PERIPHERAL_BLOCK(PADDR_BUS_in, ...)
  EXTERNAL2_BLOCK(PADDR_BUS_in, ...)
}

IR_content[DECODE]=IR(PDATA_BUS, ...)
}
```

그림 9. fetch () 함수 코드 구조
Fig. 9. Code Structure of fetch() Function.

표 2. 사이클별 명령어 수행 순서
Table 2. Operation Sequence of Instructions at Each cyele.

INSTR \ Cycle	n	n + 1	n + 2	n + 3
Add	FETCH	DEC	READ	EXE
Sub	X	FETCH	DEC	READ
OR	X	X	FETCH	DEC
AND	X	X	X	FETCH

V. CBS의 검증 및 결론

1. 검증

CBS를 제작함에 있어서 오류를 줄이기 위하여 제작단계마다 철저한 검증을 했다. 각 블록을 설계한 후에 거치는 블록 검증, CBS 완성 후 거치는 명령어 검증, 그리고 여러 명령어를 묶어서 프로그램하는 흐름 검증, 그리고 이 모두가 통합된 여러 응용분야의 프로그램을 수행을 통한 검증의 단계가 있다. 검증의 비교 기준은 상용 시뮬레이터이다. 실제로 본 논문에서는 이 네 단계를 통해서 약 2만 사이클의 검증을 마쳤다.

그림 10과 11은 상용시뮬레이터와 제작된 CBS에 ADPCM(Adaptive Delta Pulse Code Modulation) [11] [12] [13] 디코딩 프로그램을 실행 한 결과를 나타내고 있는데 두 결과가 정확하게 일치하는 것을 확인 할 수 있다. ADPCM 프로그램은 약 7000 사이클을 수행하며 사용된 명령어는 94종으로 전체 명령어의 83%를 차지한다. ADPCM 프로그램에서 사용되지 않은 명령어들은 명령어 검증 단계에서 명령어의 동작을 확인하는 과정을 이미 마쳤다. 그리고 추가적으로 μ-law, A-law, FIR, IIR, Viterbi Encoder/Decoder 알고리즘을 수행시켜 두 시뮬레이터 결과가 정확하게 일치되는 것을 확인하였다. 알고리즘을 C-Compiler를 이용하여 만들었기 때문에 모든 명령어들의 조합을 검증해 볼 수는 없었지만, 위의 검증 단계를 통하여 상당부분 검증이 되었으리라 본다.

2. 결론

본 논문에서는 고성능의 마이크로 프로세서를 설계하기 위한 효율적인 방식인 Cycle Based Simulator를 제작하고 검증하였다. 고수준 언어인 C 언어를 이용하여 약 1만 5천 라인에 걸쳐 각 기능 블록을 계층적으로 모델링하고 이를 이용해서 시뮬레이션을 했다. 이러한 과정을 통하여 파이프라인 단계별로 기능블록에 필요한 제어신호 및 기능블록의 동작이 정확하게 정의되었다. VHDL이나 게이트 레벨로 목표한 DSP를 구현하는 과정에서는 본 논문에서 개발한 CBS에서 생성되는 제어신호와 데이터를 기준으로 사용함로써 설계상에서 생길 수 있는 많은 시행착오를 줄일 수 있다. CBS가 고수준 C언어로 모델링 되었기 때문에 DSP를 VHDL로 합성하여 시뮬레이션한 경우에 비해 시뮬레이션 시간을 많이 줄일 수 있다.

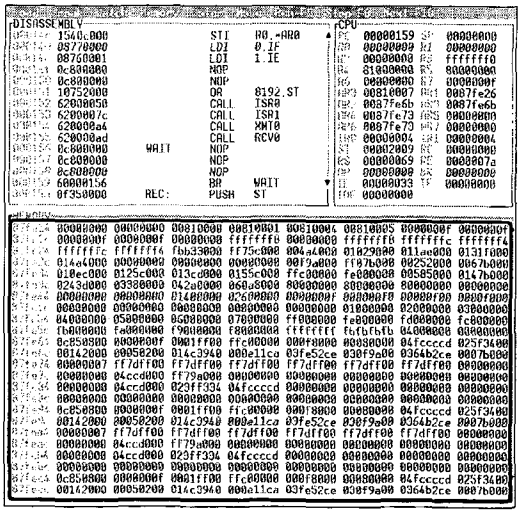

그림 10. 상용 시뮬레이터에서 ADPCM 실행결과
Fig. 10. Result of ADPCM decoding in commercial simulator.

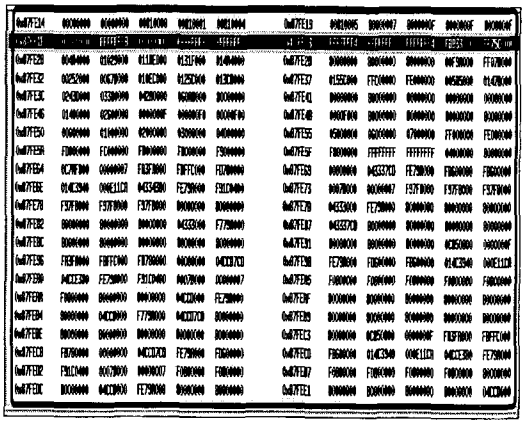

그림 11. CBS의 ADPCM 실행결과
Fig. 11. Result of ADPCM Decoding in CBS.

Ultra Sparc 기종의 workstation에서 VHDL로 구현된 DSP의 논리시뮬레이션을 통하여 ADPCM을 디코딩 알고리즘을 수행시키는 경우에 하나의 샘플 포인터를 얻는데 6시간이 걸렸으나, CBS에서는 13분만에 결과가 나왔다. CBS를 이용하여 "안녕"이라는 단어를 디코딩하여 소리를 들었으나, 논리 시뮬레이션을 통하여 그 결과를 얻으려면 3 개월 정도 걸리게되어 결과를 확인하는 것이 사실상 불가능하다. CBS를 이용하여 여러 가지 응용 프로그램을 수행시켜 그 결과에 대한 충분한 검토를 거침으로써 실제로 논리회로를 설계할 때 설계오류에의한 시간을 대폭적으로 줄일 수 있다. VLSI급 칩을 설계할 때 본 연구에서 수행한

CBS와 같은 모델의 제작하고 이를 이용하여 칩의 구조, 제어신호 등을 설계하는 것이 설계시간을 단축하는데 큰 도움이 될 것으로 생각된다. 개발된 CBS는 VHDL로의 변환을 고려해서 코딩함으로써 CBS를 참고로 VHDL을 기술할 수 있고, 이는 결국 VHDL을 기술하는데 드는 시간을 많이 단축할 수 있다. 그리고 제작된 CBS를 상용 VHDL 시뮬레이터와의 인터페이스를 통해서 VHDL & 게이트 레벨 시뮬레이션 검증 시간을 대폭 줄일 수 있었다.

본 연구에서 만든 CBS는 특정 DSP를 위한 것이지만, 여기서 사용된 개념은 다른 VLSI를 설계하는데 유용하게 응용될 수 있을 것이다.

참 고 문 헌

[1] "*TMS320C3X User's Guide*", Texas Instruments, 1994.

[2] "*TMS320C3XC Source Debugger User's Guide*", Texas Instruments, 1994.

[3] "*TMS320 Floating-Point DSP Assembly Language Tools User's Guide*", Texas Instruments, 1994.

[4] Rulph Chassaing, "*Digital Signal Processing with C and the TMS320C30*", JOHN WILEY & SONS, 1992.

[5] Steve Carlson, "*Introduction to hdl-based design using VHDL*", Synopsys, 1990.

[6] 안철홍, "*32비트 부동소수점 DSP 컨트롤러 설계*", 부산대학교 전자공학과 석사학위 논문, 1998. 2

[7] "*UNIX 시스템 프로그래밍*", 홍릉과학출판사, 1991

[8] Rozenblit, "*Codesign: Computer-Aided Software/ Hardware Engineering*", IEEE press, 1995.

[9] 샘틀지기, "*C언어 사용자를 위한 MOTIF PROGRAMMING*", 에스컴, 1994

[10] Brian W. Kernighan, "*The C Programming Language*", Prentice-Hall, 1988.

[11] Sklar, "*Digital Communications fundamentals and applications*", Prentice-hall, 1988.

[12] Nikitas Alexandridis, "*Microprocessor Based Systems*", 1993.

[13] Oppenheim, "*Discrete-Time signal processing*", Prentice-Hall, 1989.

저 자 소 개

禹 鍾 植(正會員)

1968년 10월 21일 부산 출생. 1993년 2월 부산대학교 전자공학과 졸업(학사). 1995년 2월 부산대학교 전자공학과 졸업(석사). 1997년 3월 ~ 현재 부산대학교 전자공학과 박사과정. 1997년 10월 ~ 현재 (주)보이소반도체 대표이사. 주관심분야는 DSP 설계, DSP Application, ASIC 설계, 영상 신호처리 및 구현

安 哲 弘(正會員)

1970년 7월 20일 부산출생. 1996년 2월 부산대학교 전자공학과 졸업(공학사). 1998년 2월 부산대학교 전자공학과 졸업(공학석사). 1998년 3월 ~ 현재 (주)삼성전자 멀티미디어 연구소 연구원. 주관심분야는 VLSI 설계, DSP Application

梁 海 龍(正會員)

1973년 8월 10일 부산출생. 1996년 2월 부산대학교 전자공학과 졸업(공학사). 1998년 2월 부산대학교 전자공학과 졸업(공학석사). 1998년 1월 ~ 현재 (주)현대전자 통신시스템 연구소 연구원. 주관심분야는 VLSI 설계, DSP Application, Vocoder, System Software

朴 柱 成(正會員)

1953년 12월 19일 경남 진주 출생. 1976년 부산대학교 전자공학과 졸업(학사). 1978년 KAIST 전기 및 전자공학과 졸업(석사). 1989년 Univ. of Florida EE (Ph. D.). 1978년 3월 ~ 1991년 2월 ETRI 반도체연구