마이크로프로세서 FBD 시각화

이정원° · 이기호
이화여자대학교 공과대학 컴퓨터학과

Microprocessor FBD Visualization

Jung-Won Lee° · Kiho Lee
Dept. of Computer Science and Engineering Dept, Ewha Womans University

요 약

하드웨어의 제품 사양에는 제품의 특징, FBD(Functional Block Diagram), 핀의 기능 및 배열, 프로그래밍 모드 및 각 블록의 기능 등이 함께 제시된다. 이 때 다른 사양과는 달리 설계 초기 단계부터 설정되는 가장 개념적인 FBD는 마이크로프로세서의 경우 메모리 인터페이스, 레지스터 파일, 데이터패스, 예외 처리기, 각 종 제어기, 타이머 등으로 구성된다. FBD의 각 블록들은 여러 명의 설계자들에게 분할 되고 이 중 마이크로프로세서 설계의 대부분의 시간을 소비하게 되는 각 종 제어기의 설계는 여러 블록이 공동으로 제어 신호를 공유하게 된다. 이 신호에 의해 전체 시스템의 정확성(correctness)이 결정되므로 제어기에서 각 블록에 공급하는 제어 신호는 적절한 타이밍에 정확한 값을 가져야만 한다. 따라서 본 논문은 마이크로프로세서의 FBD를 모델링 할 수 있는 시각 도구를 제안함으로써 제어 신호에 따른 전체 블록의 유기적인 데이터 흐름을 한 눈에 파악할 수 있도록 한다. 이는 설계 초기부터 각 블록들을 설계하는 설계자들간의 공통의 다이어그램인 FBD를 중심으로 설계를 해나감으로써 대화 오류를 감소시키고 제어신호 디버깅을 용이하게 하여 설계 시간을 단축시키는 것을 목표로 한다.

1. 서론

소프트웨어나 하드웨어를 설계하는 경우 설계 초기 단계에서 이미 요구 사항 분석 결과로 도출된 제품의 특징 및 블록 다이어그램이 설정된다. 현재 소프트웨어 설계는 시스템 레벨에서 소프트웨어를 모델링 하는 객체지향 분석 및 설계 방법론으로 UML 이라는 모델링 언어가 제공되고 있다. Use-Case, Class, Sequence, Collaboration 등의 다이어그램을 이용하여 소프트웨어 시스템을 구성하는 객체들 사이의 메시지 흐름이나 기능 등을 정의하고 설계자들간에 이러한 다이어그램을 공유하고 시스템 요구사항에 부합하는지를 확인 할 수 있다. 또한 소프트웨어의 구조(architecture)를 기반으로 하는 개발을 지원하기 위한 ADL(Architecture Description Language)언어들도 등장하고 있으며 이러한 언어들은 특정 모델의 세부 구현이라기 보다는 전체 응용 프로그램의 상위 수준의 구조에 초점을 두고 소프트웨어 시스템의 개념적인 구조를 모델링하는 특성들을 제공한다.

한편, 하드웨어의 설계는 SPW, System Architect, DSP station 등의 도구를 이용하여 시스템 레벨에서 하드웨어를 명세화 한다. 즉 하드웨어 시스템을 가장 상위 수준에서 보았을 때 이미 설계된 라이브러리나 컴포넌트들을 조합하여 전체 시스템을 명세화 하는 도구를 이용하는 것으로 CBSE(Component-Based Software Engineering)의 원리와 유사하다. 그러나 하드웨어 설계의 경우 기존 컴포넌트나 라이브러리를 이용한다 할지라도 이러한 블록들에게 공통적으로 신호를 공급해야 하는 제어기는 별도로 설계해야 한다. 제어기는 시스템을 구성하는 여러 블록들이 제대로 동작하기 위한 제어 신호를 공급해 주는 핵심 블록이 된다. 제어기는 명령어를 분석하고 조건을 검사하여 각 블록으로 적절한 타이밍에 정확한 값을 가진 제어 신호를 보내 주어야 한다.[2]

이러한 제어기를 설계하는 방법에는 PLA(Programmable Logic Array), VHDL(VHsic Description Language), FSM(Finite State Machine)을 이용하여 직접 제어기를 설계하는 방식이 있다. 그러나 이러한 제어기 생성 방법은 전체 블록의 유기적인 데이터 흐름을 한 눈에 파악하기 어렵고 단지 시뮬레이션 후 정적인 웨이브 파일을 통한 사용자의 분석에 의존하고 있다. 더욱이 마이크로프로세서의 명령어 집합의 종류와 수의 다양함에 따른 제어신호의 조합의 수가 방대하므로 각 명령어별 제어신호를 설계자가 일일이 분석하기 어렵다. 따라서 각 블록에서 요구하는 제어 신호에 따라 데이터와 주소 버스의 움직임을 파악하고 만약 오류가 있다면 그 오류를 일으키는 제어 신호를 블록별로 추적하여 디버깅을 할 수 있도록 좀 더 상위 수준에서 FBD를 시각화 할 필요가 있다. 시각화 된 FBD는 설계자들간의 공통의 설계 목표가 되며 또한 공동의 의견 교환의 핵심 매개체가 된다.[1]

본 논문은 마이크로 프로세서의 전체적인 데이터 흐름을 파악할 수 있도록 FBD(Functional Block Diagram)를 모델링 하고 설정된 FBD를 해석(interpretation)하여 FBDDL(Functional Block Diagram Description Language)을 생성한다. 이는 설계자와의 상호작용으로 제어 신호를 입력 받아 PLA 형태의 코드를 생성하고 FBD 모델링을 통해 생성된 FBDDL과 결합하여 각 명령어 주기별 제어 신호의 흐름을 시각적으로 파악할 수 있도록 한다. 최종적으로 마이크로프로세서의 제어기를 자동으로 생성하고 생성된 제어 신호에 따른 각 블록의 동작을 시각적으로 확인하면서 설계 오류를 감소시키고 설계 시간을 단축시키는 것을 목표로 한다.

2. 관련연구

2.1 하드웨어 시스템 명세화

시스템의 복잡도가 급격히 증가함에 따라 표준화된 언어인 VHDL이나 Verilog를 이용한 실행 가능한 시스템 명세를 사용하여 시스템 모델을 구축하는 고 수준의 설계 및 분석 방법론이 대중화 되었다. 이를 지원하는 도구에는 Renoir99(Mentor Graphics), VisualHDL(Summit)등이 있으며 그래픽 표현을 사용하여 방대한 시스템을 분할하고 분할된 각 블록들 중 일부는 이미 검증된 VHDL/Verilog 블록을 재사용 하고 나머지는 설계자가 직접 블록 다이어그램이나 FSM(Finite State Machine)을 이용하여 설계한다. 한 단계 더 나아가 DSP

Station 이나 SPW 와 같은 도구는 신호-흐름(signal-flow) 블록 다이어그램이나 데이터-흐름 언어 등을 이용하여 기존의 범용/특정 목적의 라이브러리들을 재사용하여 시스템 수준에서 시뮬레이션 할 수 있는 환경을 제공한다.

그러나 이러한 도구들을 이용하여 FBD 의 하부 블록들을 설계하고 시뮬레이션하는 것은 가능하나 FBD 자체, 즉 공통의 설계 목표이자 공동의 의견 교환의 핵심 매개체가 되는 FBD 를 모델링 하고 전체 블록의 제어 신호들의 흐름을 데이터/주소 버스 등으로 파악할 수 있는 기법은 제공하지 못하고 있다. 설계 시 가장 오랜 시간을 소요하게 만드는 제어기의 설계는 각 블록마다 원하는 시간에 원하는 값을 정확히 산출 해 내고 또한 산출된 제어 신호들이 데이터/주소 버스의 흐름을 제대로 유도하고 있는지 설계자들이 직접 확인할 수 있는 방법이 요구된다.

2.2 제어기 생성 방법

제어기를 설계하는 방법에는 PLA(Programmable Logic Array), VHDL(VHsic Description Language), FSM(Finite State Machine)을 이용하여 직접 제어기를 설계하는 방식이 있다. [1]

- PLA 를 이용하여 제어기를 구현하는 방식은 가장 일반적인 방법 중의 하나로 이미 정의된 명령어에 대한 출력의 조합으로 제어 신호를 생성해 내게 된다. 입력 부분에 명령어들을 분석한 코드를 기술하고 출력 부분이 제어 신호 스트림이 된다.
- VHDL(Verilog)을 이용한 제어기 설계는 각 명령어에 따른 일련의 제어 신호를 생성할 수 있도록 VHDL 의 IF..THEN..ELSE 또는 CASE 구문을 사용하여 코딩하고 합성(synthesis)을 통해 제어기를 생성해 내는 방법으로 PLA 를 이용하는 것 보다 코드의 길이가 매우 길어 지고 합성 능력에 따라 회로의 크기가 좌우된다.
- FSM 을 이용한 경우는 VHDL 에서 한 단계 더 나아가 시각적인 툴을 이용하여 제어 신호를 생성할 준비가 되어 있는 노드와 각 명령어를 분석하여 입력이 전이를 일으키도록 모델링하고 이를 합성하여 제어기를 생성해 낸다. 제어기 생성하는 방법 중 가장 시각적인 방법으로 이 노드에서 다른 노드로 제어 신호에 따른 전이를 추적하기에는 용이하나 마이크로프로세서와 같이 방대한 제어 신호들의 움직임을 모두 모델링 하면 매우 복잡한 다이어그램이 되어 오히려 판독성(readability)을 감소시킨다.

이 중 PLA 를 이용한 제어기 생성이 가장 최적화된 크기의 회로를 생성할 수 있으므로 PLA 를 이용하는 것이 가장 널리 사용되고 있다. 그러나 PLA 를 편집하는 경우에 일반 편집기를 사용하게 되므로 각 열과 행의 구분이 뚜렷하지 않고 일련의 0 과 1 의 조합으로 설계자가 일일이 각 입력에 대한 출력의 조합을 분석하지 않고서는 어느 부분에 어느 신호가 오동작을 일으키는지 한 눈에 파악하기 어렵다는 단점이 있다.

따라서 PLA 를 이용한 제어기 설계 시 마이크로프로세서의 FBD(Functional Block Diagram)를 시각화 하고 각 블록과 데이터 및 주소 버스에 영향을 미치는 제어 신호 사이의 상관 관계를 통합적으로 파악할 수 있도록 제어 신호의 편집 및 디버깅을 용이하게 하는 환경이 요구된다.

2.3 마이크로 프로세서의 FBD

마이크로 프로세서의 중심 블록은 메모리 인터페이스, 레지스터 파일, 데이터패스, 예외 처리기, 각종 제어기, 타이머 등으로 구성되며 이렇게 기능 블록으로 전체를 분할하고 각 블록에 연결되는 버스의 구조를 표현한 것 을 FBD(Functional Block Diagram)라 한다. 다음 그림 1 은 ARM(Advanced Risc Machine) RISC 프로세서의 FBD 를[3], 그림 2 는 Hitachi 사의 SH3 RISC 프로세서의 FBD 를 보여준다.

ARM60 은 코어(core) 부분으로 좀 더 세부적인 기능 블록으로 명시되어 있고 SH3 는 코어와 주변장치(peripheral) 제어기를 포함한 FBD 이다. 두 FBD 모두 기능 블록과 버스의 구조로 구성되어 있다는 것을 알 수 있다.

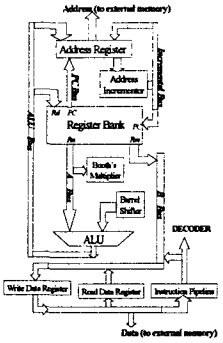


그림 1. ARM60 RISC 의 FBD

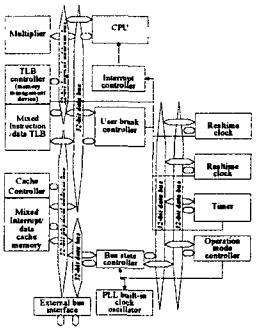


그림 2. SH3 의 FBD

3. FBD 시각화

3.1 그래픽 표기법

앞의 그림 1, 2 에서 공통적으로 보는 바와 같이 FBD 를 구성하는 요소는 크게 특정 역할을 하는 기본 블록과 32 bit 데이터/주소 버스 구조로 이루어진다. FBD 시각화를 위해 FBD 의 구성 요소를 컴포넌트, 버스, 속성으로 분류하고 다음 정의 1, 2, 3 에서 각 요소에 대한 그래픽 표기법을 정의한다.

정의 1 : 컴포넌트 (component)

각 기능 블록(ALU, Register_Bank, 제어기 등)은 컴포넌트(component)라 정의한다. 컴포넌트는 계산 및 데이터 저장, 신호 생성의 단위로 자기 자신만의 데이터와 실행공간을 가지고 정의 2 의 버스와는 포트(port)와 연결된다. 또한 각 블록에 공급되는 제어 신호의 상태를 조건에 따라 ON(값이 '1'인 신호), OFF(값이 '0'인 신호), X(Don't Care 값을 갖는 신호), ALL(모든 신호)로 저장한다. (그림 3)

정의 2 : 버스 (bus)

버스는 컴포넌트와 다른 버스, 그리고 외부로 연결되는 요소로 데이터/주소의 흐름을 애니메이션 할 수 있는 핵심 요소이다. 해당 버스에 영향을 미치는 제어 신호의 값을 보여 준다. 버스에 데이터/주소의 이동이 있을 때 ON 상태를, 데이터/주소의 흐름이 없을 때 OFF 상태를 보여 준다.(그림 4)

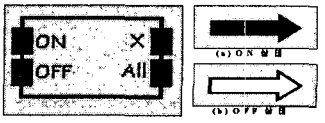
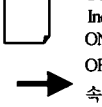


그림 3. 컴포넌트 그림 4. 버스

정의 3 : 속성 (property)

속성은 버스의 상태를 ON/OFF 시키는 조건이다. 즉 해당 버스는 자신에 영향을 미쳐 이벤트를 발생시키는 제어 신호를 속성으로 갖는다. 또한 다수의 제어 신호들이 영향을 미치는 경우 논리적인 연산자 AND, OR 로 결합한다.



CONTROL
Index name ; // signal description
ON_CONDITION: binary value; ← 버스의 ON 조건
OFF_CONDITION: binary_value; ← 버스의 OFF 조건

속성과 버스를 연결

다수의 제어신호를 AND 시켜야만 버스를 ON 시킴.

제어신호 중 하나라도 ON 조건이면 버스를 ON 시킴.

기능 블록 Register_Bank 와 ALU 를 연결하는 A_bus 에 대한 속성을 표현하면 다음 그림 5 와 같다. i_lpm, abus_sel 신호가 모두 ON 조건을 만족해야만 A_Bus 에 데이터가 흐르게 된다.

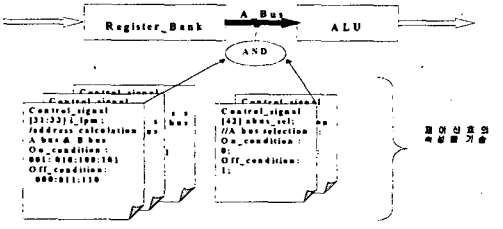


그림 5. FBD 의 시각적 표기

이 때 다수의 설계자들은 FBD 의 애니메이션을 통하여 의도하지 않은 데이터의 흐름을 인지하고 버스의 속성을 확인하면 그 버스의 흐름에 영향을 미친 제어 신호 정보를 볼 수 있다. 따라서 그 시점의 ON/OFF 조건을 시각적으로 확인함으로써 시뮬레이션 결과의 웨이브파형을 통해서나 텍스트 자체를 통한 디버깅 보다 효율적인 방법을 제공한다.

3.2 FBDDL(Functional Block Diagram Description Language) 문법 정의

FBDDL 은 각 요소의 세부 구조를 언어로 기술함으로써 시각적으로 정의한 FBD 와 제어 신호를 연동하여 FBD 의 컴포넌트와 버스를 애니메이션할 수 있는 기반이 된다. 다음 그림 6 은 이러한 FBDDL 의 문법을 BNF 로 정의한다.

```
System_Specification ::= SYSTEM Name = { Component_Dscription  Bus_Description };
Component_Description ::= Component_Description  Component_Statements
Component_Statements := COMPONENT Name = { Port_Description };
Port_Description = Port_Description Port_Statements : Port_Statements
Port_Statements ::= PORT In_Port_Description ;  Out_Port_Description ;
In_Port_Description ::= In_Port_Description In_Port : In_Port
Out_Port_Description ::= Out_Port_Description Out_Port : Out_Port
In_Port ::= IN Port_name; | ε
Out_Port ::= OUT Port_name; | ε
Bus_Description ::= Bus_Description  Bus_Statements : Bus_Statements
Bus_Statements ::= BUS Name = { Connect_Description  Bit_Width  Property_Description }
Connect_Description ::= CONNECT { Name,  Name } ;
Bit_Width ::= BIT_WIDTH  :  Digit
Property_Description ::= PROPERTIES : Relation_Op { Control_Signal_Description } ;
Relation_Op ::= AND | OR | ε
Control_Signal_Description ::= Control_Signal_Description  Control_Signal | Control_Signal
Control_Signal ::= { CONTROL Signal_Description; On_Description; Off_Description; }
Signal_Description ::= Bit_Index  Name ;
Bit_Index ::= [ Bit_Digit ]
Bit_Digit ::= Digit : Digit | Digit
On_Description ::= ON_CONDITION : Signal_Value;
Off_Description :: = OFF_CONDITION : Signal_Value;
Signal_Value ::= Digit : Digit | Digit
Port_Name ::= Name Bit_Index
Name ::= [a-zA-Z]+
Digit ::= [0-9]+
```

그림 6. FBDDL 의 문법

앞의 그림 5 의 예를 FBDDL 로 번역하면 다음과 같다.

```
SYSTEM  ARM_RISC  =  {
      COMPONENT  Register_Bank  = {
            PORT  IN     ALU_port[32];
                  OUT  A_port[32];  } ;
      COMPONENT  ALU  = {
            PORT  IN     A_port[32];
                  OUT  ALU_port[32];  } ;
      BUS  A_Bus  = {
            CONNECT { Register_Bank.A_port, ALU. A_port }
            PROPERTIES : AND
                  { { CONTROL
                        [31:33] i_lpm;
                        //address calculation   A bus & B bus
                        ON_CONDITION :
                        001:010:100:101
                        OFF_CONDITION:
                        000:011:110   }
                  { CONTROL_SIGNAL
                        [42] abus_sel;
                        //A bus selection
                        ON_CONDITION :
                        0;
                        OFF_CONDITION:
                        1   }
      } }   } ;
```

4. 구현

순수 자바코드만을 이용하여 FBD 의 각 블록과 데이터 버스를 객체화 하여 시각환경을 설계한다. 컴포넌트, 데이터/주소 버스, 제어신호와 같은 구성요소를 가지고 시스템의 구조를 시각적으로 구축한다. 다음 그림 7 은 FBD 모델링을 위한 시각 환경으로 좌측에는 컴포넌트, 버스, 속성, FBDDL 번역 아이콘 및 조작 명령 패널이, 우측에는 아이콘을 이용하여 드로잉 되는 FBD 패널이 보여 진다. 또한 각 컴포넌트마다 제어신호를 ON(blue), OFF(red), X(gray), ALL(green)로 분류하여 정보를 유지한다. 그림 8 은 모델링된 FBD 를 입력으로 생성된 ARM RISC 의 FBDDL 표현을 보여 준다.

구현 환경은 Solaris2.5 운영체제하에서 JDK1.2를 이용하여 구현하였고 구현 대상 마이크로프로세서는 ARM60 RISC 이다.(그림 1의 FBD와 비교)

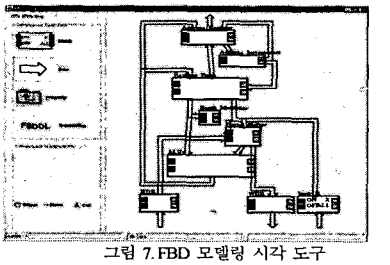


그림 7. FBD 모델링 시각 도구

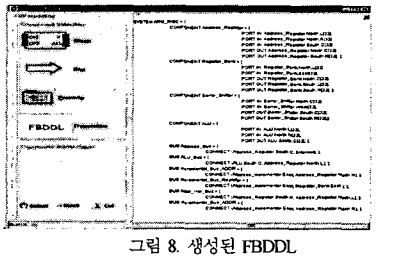


그림 8. 생성된 FBDDL

5. 결론 및 향후 연구과제

본 논문은 마이크로프로세서의 FBD(Functional Block Diagram)를 모델링 할 수 있는 시각 도구를 제안함으로써 제어 신호에 따른 전체 블록의 유기적인 데이터 흐름을 한 눈에 파악할 수 있도록 하였다. 이는 설계 초기부터 각 블록들을 설계하는 설계자들간의 공통의 다이어그램인 FBD 를 중심으로 설계를 해나감으로써 대화 오류를 감소시키고 제어신호 디버깅을 용이하게 하여 설계 시간을 단축시킬 수 있는 환경을 제공하였다. 향후 연구는 다수의 설계자들이 지역적으로 분산되어 있는 경우 공유작업공간을 통해 협업(Collaborative Work)할 수 있는 ECAD(Electronic-CAD)로 확장시키는 것이다.

6. 참고 문헌

[1] 이정원, 이기호, "FBD 시각화를 통한 마이크로프로세서 제어기 자동 생성", 1999년 한국정보과학회 춘계학술 발표회 논문집(A), Vol.26, No.1, 27p ~ 29p
[2] John L. Hennessy, David A.Patterson, "Computer Organization and Design", Morgan Kaufmann Publishers, 1994
[3] "ARM60 Data Sheet", ARM, July, 1993
[4] D.S.Harrison, R.A.Newton, R.L.Spickelmier, T.J.Barness,"Electronic CAD frameworks", Proc. of IEEE 78(2):1062 - 1081, Feb. 1990