특집

음원 DSP의 FPGA 구현

朴 柱 成, 張 豪 根
釜山大學校 電子工學科

I. 서 론

1980년대 후반에 나오기 시작한 FPGA는 현재에 이르러 초기의 게이트 수와 속도 상의 단점을 극복하면서 ASIC화가 지니는 여러 가지 단점들을 보완시켜주는 또 하나의 중요한 방법으로 떠오르게 되었다. ASIC은 제작 후 디자인의 수정이 불가능하며 적은 양의 IC를 생산하고자 하는 경우에는 비용이 많이 드는 단점이 있다. 반면 FPGA는 높은 유연성, 빠른 프로토타입핑, 디자인 수정의 용이함, 그리고 reconfigurability 등의 장점을 지니면서 ASIC의 단점을 보완시켜주는 대체 도구로서의 역할 뿐만 아니라, 하드웨어 구조상의 특성을 최대한 살려 특정 연산 작업을 수행시키는 custom computing machine으로서도 사용되고 있다.

반도체 기술이 발달함에 따라 엄청난 수의 트랜지스터가 하나의 칩 내에 집적되고, 칩의 동작은 고도로 복잡해짐에 따라 설계 후 실제 시스템 상에서 칩의 동작여부를 판단하는 것이 중요한 관건이 되고 있다. 특히, ASIC화 하는데 많은 비용 부담을 감수해야 하는 경우 칩의 실제 동작 여부를 미리 아는 것은 매우 중요한 일이 될 것이다. 이에 따라 단순히 설계 툴 상의 시뮬레이션으로만 만족하지 않고, 설계된 로직을 실제로 하드웨어화 시켜 그 동작 여부를 확인하려 하게 되고, 이를 위해 하드웨어 에뮬레이터나 FPGA를 이용하게 된다. 하드웨어 에뮬레이터는 고가의 장비이므로 손쉽게 이용할 수 없지만, FPGA는 적은 비용으로 실험실 수준에서 IC를 검증해 볼 수 있는 가장 유용한 방법이라 할 수 있다. 본 실험실에서 2년동안 연구해 온 음원 DSP(Digital Signal Processor)를 FPGA로 구현한 것도 그러한 취지에서 나온 것으로, 당장 ASIC화 하는데 따른 비용과 위험을 최소화시키기 위해 미리 FPGA로 구현해 보므로써 시스템에서 칩의 동작을 확인하려 한 것이었다.

본 고에서는 ASIC을 목표로 한 음원 칩을 설계하고, 이를 FPGA로 구현해 시스템에서 그 동작을 검증하는 일련의 방법들과 절차에 대해서 이야기 하고자 한다. 이것은 FPGA를 위한 복잡한 테

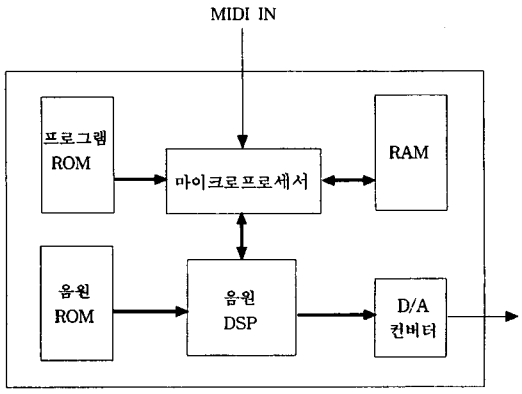
크닉을 사용하지 않고, 설계된 로직을 그대로 FPGA로 가져가는 기본적이고도 일반적인 방법론이 될 것이다. II장에서는 음원 DSP의 동작과 로직 설계에 대해, III장에서는 이를 FPGA로 구현하기 위한 방법과 절차에 대해서 논하도록 한다. 마지막으로 IV장에서는 FPGA로 구현하는데 따른 문제점과 개선 방향을 언급하고 결론을 맺도록 한다.

II. 음원 DSP의 설계

사운드 기능은 멀티미디어 환경의 가장 기본적이고 필수적인 요소라 할 수 있다. 전자 음악, 특히 디지탈 음악 분야는 1980년대 디지탈 음향 악기간 신호 전달의 국제적 공통 규격인 MIDI (Musical Instrument Digital Interface)가 발표된 이후 급속한 성장을 보이고 있으며, 음원 칩은 PC 사운드 카드에서부터 고기능의 신서사이저에 이르기까지 다양한 형태의 사운드와 음악을 손쉽게 우리에게 제공해주는 핵심적인 전자부품이다.

악기음 합성 방식에 있어서는 가산 합성(Additive synthesis), 감산 합성(Subtractive synthesis), Waveshaping 합성, FM 합성(Frequency Modulation synthesis), 그리고 PCM 합성(Pulse Code Modulation synthesis 혹은 Wavetable synthesis) 등이 있으며, 이들 중 현재에는 FM 합성과 PCM 합성이 주로 쓰이고 있다. FM 합성은 여러 개의 정현파 오실레이터를 다양한 형태로 연결하므로써 악기음의 풍부한 동적 스펙트럼을 표현하고, PCM 합성은 실제 악기의 원음을 샘플링해서 메모리에 저장시켜 두고 이를 이용해 음정과 음색을 변화시켜 사용하는 방식이다.

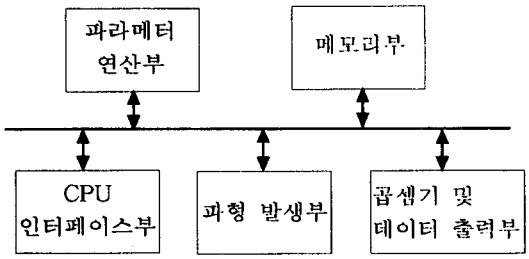
음원 DSP의 동작을 이해하기 위해 그림 1과 같은 간단한 음원 모듈 시스템을 살펴보도록 하자. 그림 1에서 미디 신호의 소스는 여러 가지가 될 수 있다. 특정 전자악기인 경우에는 피아노의 건반이나 기타의 스트링에서 발생되는 연주 정보이며, PC의 경우에는 시퀀스(Sequencer)라고 하는 음악 연주 프로그램으로부터 나오는 연주 정보이다. 마이크로프로세서는 전체 시스템의 동작을 제어하고, 미디 신호를 입력으로 받아 이를 해독해서 해당 악기음을 내기 위한 여러 가지 합성 파라메터를 발생시켜 DSP에 전달하는 역할을 한다. 음원 DSP는 마이크로프로세서로부터 전달되는 합성 파라메터를 받아들여 FM 혹은 PCM 합성 알고리즘을 수행하고, 샘플데이터를 발생시켜 외부 D/A 컨버터로 내보낸다. 이 과정에서 음원 롬에 저장된 원음의 샘플데이터를 이용하기도 한다. 음원 DSP는 합성 알고리즘을 하드웨어로 구현하여 빠르게 수행시킴으로써 더 많은 음을 동시에, 다양한 형태로 내기 위해 필요한 것이다.



〈그림 1〉 음원 모듈 시스템

본 실험실에서 설계한 음원 칩의 특징으로는 FM 합성 방식과 PCM 합성 방식을 이용해 악기음을 합성하며. 44.1Khz의 샘플링 주파수로 32개의 악기음까지 동시에 내는 것이 가능하다. 또한, 2개의 출력 채널을 통해 스테레오 기능을 제공한다. 그림 2는 설계된 음원 칩의 전체적인 기능 블록도를 보여주고 있다.

그림 2에서 CPU 인터페이스부는 마이크로프로세서와 DSP간의 합성 파라메터를 주고 받기 위한 것으로, 마이크로프로세서로부터 메모리에 파라메터를 쓰거나, 메모리에 저장된 파라메터를 읽어내어 마이크로프로세서에 전달하는 역할을 한다. 메모리부는 두 개의 메모리로 구성되는데, 하나는 마이크로프로세서로부터 전달되는 합성 파라메터를



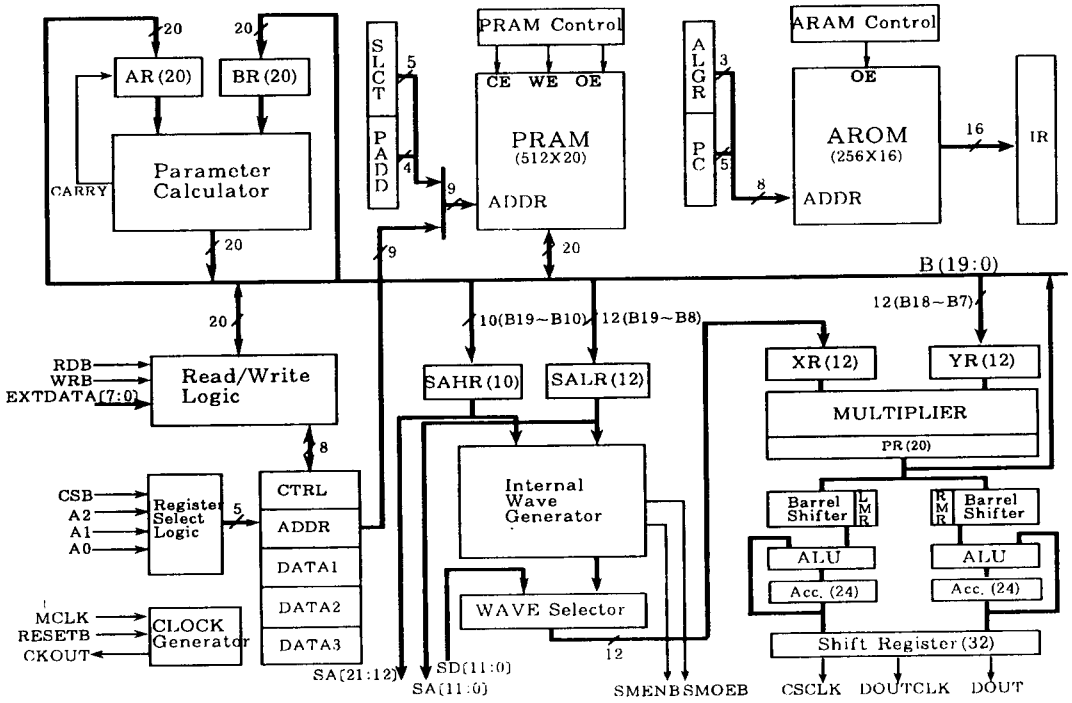
〈그림 2〉 음원 DSP의 기능 블록도

저장하는 램이며, 다른 하나는 파라메터를 이용해 합성 알고리즘을 수행시키는데 필요한 명령어(마이크로인스트럭션)들이 저장되는 롬이다. 파라메터 연산부는 기본적으로 덧셈 연산을 수행하고, 덧셈 결과로부터 나오는 캐리를 통하여 합성 파라메터 값을 계산하는 동작을 수행하게 된다. 파형 발생부는 정현파와 램프파의 샘플데이터를 발생시키고, 외부에 있는 음원 롬의 샘플데이터를 읽어들이는 역할을 한다. 곱셈 및 데이터 출력부는 곱셈을 수행하고, 최종 출력 샘플데이터 값을 만들어 내어 외부 D/A 컨버터로 출력시키는 일을 하게 된다. 음원 DSP의 기능적 분할은 위의 5가지로 구분된다고 볼 수 있다. 따라서 ASIC화 하는 경우는 기능 블록별로 레이아웃을 행하는 것이 일반적인 방법이라 볼 수 있으나, FPGA로 구현할 때 여러 개의 칩을 사용해야 하는 경우에는 반드시 기능 블록별로 분할되는 것은 아니다. 이것은 Ⅲ장에서 다시 다루도록 한다. 그림 3에는 음원 칩의 상세한 하드웨어 구조를 나타내었다. 이를 바탕으로 하여 각 기능 블록별로 좀 더 자세하게 설계된 로직을 살펴보도록 한다.

1. CPU 인터페이스부

인터페이스부는 5개의 인터페이스 레지스터와 이와 관련된 로직들로 구성된다. 5개의 레지스터는 다음과 같다.

- 제어 레지스터 : 2비트로 이루어지며, 파라메터를 저장하는 램에 데이터를 쓰거나 읽는 동작을 제어한다. 비트 1은 음원 칩 전체의



〈그림 3〉 음원 DSP의 내부 로직도

동작을 enable 혹은 disable 시키는데 사용되며, 비트 0는 쓰기 혹은 읽기 동작을 구분하는 일을 하게 된다.

- 어드레스 레지스터 : 9비트로 이루어지며, 메모리의 어드레스를 지정한다.
- 데이터 레지스터3 : 20비트의 데이터 중 상위 4비트를 저장한다.
- 데이터 레지스터2 : 20비트의 데이터 중 B15-B8 비트를 저장한다.
- 데이터 레지스터1 : 20비트의 데이터 중 B7-B0 비트를 저장한다.

마이크로프로세서는 5개의 레지스터에 순차적으로 값들을 써넣으므로써 PRAM의 어드레스에 데이터를 쓰거나 읽을 수 있다. 이 블록에서 외부로 나가는 핀은 마이크로프로세서 읽기 신호 RDB, 마이크로프로세서 쓰기 신호 WRB, 8비트의 데이터 I/O 버스, 레지스터 선택 신호 A(2 : 0), 클럭 입력 등이 있다. 파라메터 램과 20비트의 내부 데이터 버스와 연결되며, 9비트의 어드레스 신호가 읽기/쓰기 제어신호와 함께 메모리부에 전달된다.

2. 메모리부

메모리부에는 두 개의 메모리가 있으며, 어드레스 발생 로직, 읽기/쓰기 제어신호 발생 로직 등으로 구성된다. 마이크로프로세서가 미디 신호를 받아들여 알고리즘 수행에 필요한 파라메터를 계산해서 DSP에 전달하게 되는데, 이를 저장하는 것이 파라메터 램(PRAM)이다. 한 악기음 당 16개의 파라메터가 사용되고, 파라메터당 20비트로 이루어지므로 32개의 악기음을 동시에 내기 위해서는 총 512개의 파라메터를 저장할 램이 필요하다. 따라서 PRAM은 512×20비트의 크기를 갖는다. 알고리즘의 수행은 알고리즘 롬(AROM)에 저장된 마이크로인스트럭션의 수행으로 이루어진다. FM 혹은 PCM 합성은 한 알고리즘당 32개의 마이크로인스트럭션으로 구성되며, FM 알고리즘과 PCM 알고리즘을 합쳐 총 8개의 알고리즘에 대한 명령어들이 AROM에 저장되어 있다. 하나의 마이크로인스트럭션은 디코더가 필요없는 직접 제어신호 16비트로 이루어져 있다. 따라서 AROM의 크기는 256×16비트가 된다.

PRAM의 32개의 각 세그먼트는 악기음 하나를 내기 위한 합성 파라메터가 저장되는 곳이다. 악기음 샘플데이터의 출력 시간 간격인 22.7㎲(1/44.1Khz)동안 32개의 세그먼트는 순차적으로 액세스된다. 따라서 한 세그먼트를 액세스하는 시간은 708ns가 되며, 해당 악기음의 샘플데이터를 만들기 위해 AROM에 저장된 한 알고리즘의 32개 명령어가 순차적으로 실행된다. 하나의 명령어는 한 사이클 안에 동작이 끝나므로, 어떤 알고리즘의 수행이 끝나는데는 32개의 클럭 사이클이 필요하다. 알고리즘당 하나의 샘플데이터가 만들어진다고 보면, 22.7㎲동안 32개의 샘플데이터가 만들어져 데이터 출력부의 어큐뮬레이터에 더해져서 저장되며, 새로운 샘플데이터가 만들어지는 다음 주기동안 외부 D/A 컨버터로 내보내진다.

AROM의 어드레스는 명령어 카운터로부터 나오며, 상위 3비트는 현재 액세스되는 세그먼트의 실행 알고리즘 번호이며, 그 알고리즘 수행이 끝날 때까지 유지된다. 나머지 5비트는 5비트 이진 카운터의 출력이다. PRAM의 어드레스는 두 군데에서 발생된다. 하나는 슬롯 카운터로서 상위 5비트는 5비트 이진 카운터의 출력으로서 해당 알고리즘이 수행되는 동안 그 값을 유지한다. 나머지 4비트는 명령어가 실행될 때마다 바뀌며, 명령어 실행에 필요한 파라메터 번지를 가리킨다. 또 하나의 어드레스는 인터페이스부의 어드레스 레지스터로부터 9비트가 나온다.

슬롯 카운터의 출력은 PRAM의 어드레스로서 사용될 뿐만 아니라, 하나의 합성 주기(Synthesis frame)가 시작되고 끝나는 것을 알려준다. 이 정보는 데이터 출력부에 들어가서 어큐뮬레이터에 저장된 샘플데이터 값을 쉬프터 레지스터로 전달하는데 사용된다. 명령어 카운터의 출력은 한 알고리즘의 실행 사이클 수를 나타내며 명령어 실행 사이클과 메모리 인터페이스 사이클을 구분하는 기능을 한다. 한 알고리즘의 32개의 사이클 중 마지막 2 사이클은 PRAM과의 인터페이스를 위해 사용되므로, 명령어 카운터의 출력은 PRAM의 두 개의 어드레스 중 하나를 선택하는데 사용된다.

3. 파라메터 연산부

파라메터 연산부에는 20비트의 레지스터 두 개와 20비트 덧셈기가 있다. AR과 BR은 내부 데이터 버스로부터 데이터를 받아들이며, 그 출력은 덧셈기의 입력이 된다. 20비트 덧셈기는 4비트 carry-look-ahead 방식의 덧셈기 5개로 구성되어 있다. 파라메터 연산부에서는 주로 일반적인 덧셈 연산이 이루어지며 보통 프로세서의 ALU기능을 하는 것은 아니다. 그 대신 특정 파라메터에 대한 연산을 위해 특별한 로직이 부가되어 있는데, 샘플데이터의 어드레스와 점진적으로 변하는 악기음의 엔빌로프 등을 계산하는데 사용한다.

4. 파형 발생부

파형 발생부는 샘플데이터의 어드레스를 지정하는 두 개의 레지스터와 내부 파형 발생기, 그리고 내부 파형의 샘플데이터와 외부 음원 롬의 샘플데이터를 선택하기 위한 멀티플렉서 등으로 구성된다. 10비트로 이루어진 SAHR 레지스터와 12비트로 이루어진 SALR 레지스터는 내부 데이터 버스로부터 값을 받아들이며, 외부 음원 롬의 어드레스를 저장하거나, 내부 파형 발생기로부터 파형의 샘플데이터 값을 읽어오는데 사용된다. 이 때 SAHR 레지스터는 파형의 종류를 선택하고, SALR 레지스터는 샘플데이터의 어드레스를 저장한다. 내부 파형 발생기는 정현파형과 램프파형에 대해 샘플데이터를 발생시키며, 12비트의 어드레스로 파형당 4096개의 샘플데이터를 발생시킨다. 정현파형은 look-up 테이블 방식으로 구현되어 한 샘플데이터당 12비트씩 총 4096×12비트의 메모리 용량이 필요하지만, 정현파형의 대칭성을 이용하여 1024개의 샘플데이터만을 저장시켜 놓고, 2의 보수화기와 멀티플렉스 등을 사용하여 전체 파형에 대한 샘플데이터를 얻는다. 램프 파형에 대해서는 12비트의 어드레스로부터 바로 샘플데이터를 만들어내는데, SALR 레지스터 값을 단순히 쉬프트 시키므로써 샘플데이터 값을 얻을 수 있다. 외부와 인터페이스 되는 핀으로는 SAHR 레지스터 값과 SALR 레지스터 값이 음원 롬의 어드레스로 사용하기 위해 출력되고, 음원 롬으로부터 샘플데이터를 받아들이기 위한 12비트의 입력핀이 있다. 내부 혹은 외부에서 읽혀진 샘플데이터는 진폭값과 곱해지기 위해 곱셈부의 XR 레지스터에 전달된다.

5. 곱셈기 및 데이터 출력부

곱셈기 및 데이터 출력부는 파형 발생부로부터 만들어진 샘플데이터를 받아들이는 12비트의 XR 레지스터, 내부 데이터 버스로부터 상위 12비트를 받아들이는 YR 레지스터, 12×12비트의 signed multiplier, 곱셈의 결과값을 저장하는 20비트의 PR 레지스터, 그리고 Barrel shifter, 덧셈기, 어큐뮬레이터, 쉬프터 레지스터로 이루어지는 샘플데이터 출력부로 구성된다. YR에는 XR에 실린 샘플데이터 값과 곱해질 진폭값뿐만 아니라, PCM 합성의 경우 필터의 계수, FM 합성의 경우 변조지수 등 알고리즘에 따라 여러 가지 값이 들어올 수 있다. 12×12비트 signed multiplier는 수정된 Booth 알고리즘에 의해 설계되었다. 24비트의 곱셈 결과 중 상위 20비트만이 PR 레지스터에 저장되며, 특정 명령어에 의해 그 값이 데이터 버스에 실리게 된다. 곱셈의 결과값은 두 개의 출력 채널로 나뉘어 데이터 출력부에 들어가게 된다. Barrel shifter의 역할은 좌우 채널의 출력값을 조정하는 역할을 하며, 여기에 필요한 쉬프트 양은 각각 3비트로 이루어진 LMR 레지스터와 RMR 레지스터에 저장되어 있다. Barrel shifter의 출력은 이전의 알고리즘 수행에서 만들어진 샘플데이터와 덧셈기에서 더해져 24비트의 어큐뮬레이터에 저장된다. 어큐뮬레이터에 저장된 32개의 샘플데이터 값은 하나의 합성주기가 끝날 때마다 쉬프트 레지스터로 전달되는데, 이 과정에서 24비트의 채널별 출력값은 16비트로 조정된다. 쉬프트 레지스터에 전달된 좌우 채널의 32비트 샘플데이터 값은 한 비트씩 DOUT핀을 통해 D/A 컨버터에 전달된다.

이 때 함께 출력되는 클럭으로 채널 선택용 클럭인 CSCLK와 쉬프트 클럭인 DOUTCLK가 있다.

III. 음원 DSP의 FPGA 구현

이 장에서는 설계된 음원 DSP를 Xilinx사의 XC4010 FPGA로 구현하는 것에 대해 설명하고자 한다. Xilinx사의 X4000 시리즈는 SRAM 방식을 채택해 reconfiguration이 용이하고, 내부 3-state 버퍼의 구현이 가능하고, RAM이나 ROM과 같은 메모리를 칩 내부에 포함시킬 수 있으며, 일반 EPROM을 사용해 프로그램을 쉽게 할 수 있는 등의 장점을 지니고 있다. 표 1에는 XC4010의 하드웨어 특성을 나타내고 있다. 여기서 CLB란 Configurable Logic Block을 말하며 FPGA 내에서 하나 혹은 그 이상의 함수가 구현되는 로직 모듈을 일컫는다.

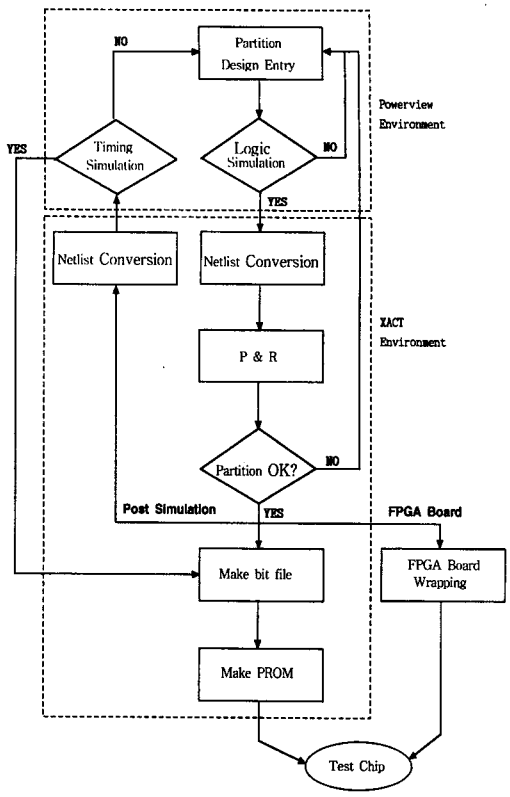
〈표 1〉 XC4010의 하드웨어 특성

Gate	10,000
CLB matrix	20×20
Number of CLBs	400
Number of flip-flops	1,120
Max. RAM bits	12,800
Number of IOBs	160

설계된 음원 DSP는 191핀 PGA 타입인 4010PG191을 사용하여 구현하였다. 여기에 사용된 소프트웨어 툴은 디자인 입력과 시뮬레이션을 위해 Powerview 5.2를 사용하였고, P&R 툴로서 XACT 2.6을 사용하였다. 음원 DSP를 FPGA로 구현하는 순서도를 그림 4에 나타내었다. 이 순서도를 바탕으로 하여 각 단계별로 어떤 일들이 수행되는지를 살펴보도록 한다.

1. 디자인 분할 및 schematic 입력

하나의 FPGA로 원하는 IC를 구현할 수 있다면 문제가 안되겠지만 여러 개의 FPGA 칩을 사용해야 할 경우에, 디자인을 분할하는 일은 회로의 동작 속도와 하드웨어 비용, 그리고 시스템의 전반적인 설계에 중요한 영향을 미치는 요인이 된다. 여기에는 가장 적은 하드웨어를 사용하여 가장 높은



〈그림 4〉 음원 DSP를 FPGA로 구현하기 위한 순서도

동작 속도를 얻는 것이 주된 목표가 될 것이다. 이를 위해 설계된 내용을 잘 분석해서 각 FPGA의 자원들을 효율적으로 사용하는 것이 필요하고, 파이프라인 단계를 잘 설정해서 항상 모든 FPGA가 동작할 수 있도록 해야 한다. 예를 들어 동영상을 압축 코딩하는 하드웨어를 FPGA로 구현한다고 하면, 알고리즘이 수행되는 단계별로 하나의 FPGA를 사용하므로써 연속적인 파이프라인 동작이 이루어지도록 할 수 있다. 디자인 분할 작업을 소프트웨어 툴을 사용하여 자동으로 수행시키지 않고 수작업으로 해야 하는 경우, 그림 4에서 파티션으로부터 P&R까지 몇 번의 반복 작업이 필요할 수도 있다. P&R 후에 나오는 결과와 타이밍 시뮬레이션을 통해 각 칩들의 성능을 확인하므로써 최적의 분할을 얻을 수 있기 때문이다. 설계된 음원 DSP도 몇 번의 반복 작업을 통해 4개의 FPGA 칩으로 분할되었다. 기본적인 원칙은 하나

의 기능 블록을 한 FPGA에 넣는 것이었으며, 게이트 카운트와 입출력 신호의 갯수, 동작 속도 등을 고려하여 최종적으로 다음과 같이 결정되었다.

Chip1 : 메모리부, 인터페이스부
Chip2 : 파라메터 연산부, 파형 발생부
Chip3 : 곱셈부
Chip4 : 데이터 출력부

칩과 칩 사이에서 연결되는 신호는 원칙적으로 레지스터의 출력신호나 입력신호로 설정하므로써 어떤 신호가 combinational 로직을 통해 여러 개의 칩을 거치는 것을 방지할 수 있다. Powerview의 schematic 입력 툴인 Viewdraw를 이용하여 Xilinx의 X4000 라이브러리를 써서 분할된 각 칩에 대한 schematic을 완성하고 이에 대한 netlist를 얻는다.

2. 로직 시뮬레이션

다음으로는 설계된 각 칩에 대한 로직 시뮬레이션을 수행하는 것이다. 분할된 각 칩의 동작을 시뮬레이션을 통해 확인하였고, 4개의 칩을 연결하여 만든 schematic으로부터 전체 칩에 대한 netlist를 얻어서 모든 칩이 연결되었을 때의 동작을 확인하였다.

3. Netlist 변환

로직 시뮬레이션이 통과되면 P&R(Place and Route)을 수행하기 위하여 Powerview에서 나온 각 칩의 netlist 포맷을 XACT 툴에 입력될 수 있는 형태로 바꾸어 주어야 한다. 음원 DSP의 경우 각 칩에 대한 4개의 xnf(Xilinx Netlist Format) 파일이 필요하다. XACT 툴에서 공급하는 xmake 프로그램을 수행시키면 칩 내의 각 디자인 셀에 대한 netlist 변환이 먼저 이루어지고, 최종적으로 이들을 통합시켜 칩에 대한 XNF 파일을 생성시킨다.

명령 : xmake -M -N chip1.1
입력 : 설계된 칩 내부의 모든 디자인 셀에 대한 netlist
출력 : chip1.xnf, chip1.out

4. P&R

P&R은 사용자가 설계한 내용을 FPGA의 로직 블록에 매핑시키고, 각 블록들을 연결하므로써 FPGA 칩에 대한 물리적인 레이아웃을 실행하는 것이다. 각 칩에 대한 xnf 파일을 가지고 X4000 시리즈의 P&R 프로그램인 ppr을 수행시키면 그 결과로서 FPGA 칩의 여러 가지 자원들에 대한 사용정도를 알 수 있다. 또한 I/O 핀들에 대한 매핑 관계가 나오므로 이 결과를 보고 FPGA 보드를 꾸밀 수 있게 된다. 출력 파일인 lca(Logic Cell Array) 파일에는 물리적인 레이아웃에 대한 정보가 담겨있다. 표 2에는 음원 DSP의 4개의 칩에 대한 P&R 결과를 나타내었다.

명령 : ppr chip1.xnf parttype=4010PG191-5
입력 : chip1.xnf
출력 : chip1.lca, chip1.rpt, ppr.log

P&R이 끝나고 나면 3가지로 작업이 나누어질 수 있다. 출력 파일인 lca로부터 타이밍 시뮬레이션을 수행할 수 있고, FPGA의 configuration EPROM을 만드는 작업을 진행할 수 있으며, rpt 파일에 나와 있는 핀들과 I/O 신호에 대한 매핑 관계를 보고 FPGA 보드의 래핑 작업을 행할 수 있다.

〈표 2〉 음원 DSP의 4개의 칩에 대한 최종 P&R 결과

	Chip1	Chip2	Chip3	Chip4
CLBs	101(25%)	212(53%)	252(63%)	247(61%)
Flip-Flops	93(8%)	71(6%)	44(3%)	132(11%)
Equivalant Gates	1832	2861	3021	4917
I/O Pins	92(57%)	104(65%)	53(33%)	34(21%)

5. 타이밍 시뮬레이션

lca 파일에는 FPGA 칩의 물리적인 연결정보가 담겨 있으므로 이를 통해 타이밍 시뮬레이션을 할 수 있다. 타이밍 시뮬레이션을 하기 위해서는 lca 파일로부터 Powerview의 netlist 포맷으로 바꾸어 주어야 하며, 이를 위해서는 다음과 같은 명령어가 실행된다.

① makebits -j -w chip1.lca ;Back annotation timing information
입력 : chip1.lca
출력 : chip1.lca

② lca2xnf -g chip1.lca chip1-post.xnf ; Generate xnf file
입력 : chip1.lca
출력 : chip1-post.xnf

③ xnf2wir chip1-post.xnf
입력 : chip1-post.xnf
출력 : chip1-post.1

6. 비트 파일 생성

lca 파일로부터 configuration 시에 FPGA에 다운로드 될 수 있는 형태의 이진 파일을 생성시킨다. 각 칩에 대한 lca 파일로부터 bit 파일을 만든다. 이 bit 파일은 Xilinx에서 제공되는 XChecker 케이블을 사용할 경우 PC 혹은 워크스테이션의 출력 포트를 통해 바로 FPGA로 다운로드 될 수 있는 형태이다.

명령 : makebits chip1.lca
입력 : chip1.lca
출력 : chip1.bit

7. EPROM 생성

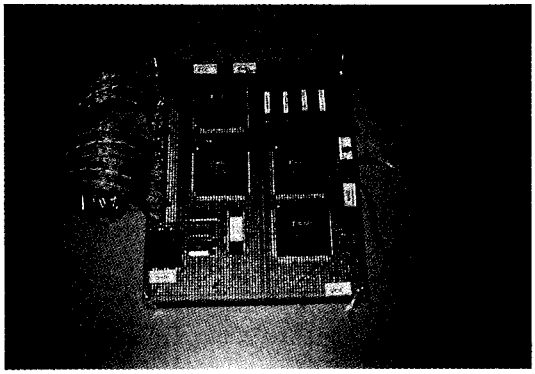
각 칩에 대한 bit 파일로부터 최종적으로 FPGA를 configuration 하는데 사용될 EPROM을 만든다. 일반 8비트 parallel EPROM을 사용할 경우에는 XACT에서 제공하는 makeprom 프로그램을 쓰면 된다. 여러 개의 칩을 사용할 경우에는 EPROM 하나에 모든 칩에 대한 configuration 정보를 넣고 차례로 configuration 시켜줄 수 있다. 이 때 EPROM에 저장되는 bit 파일의 순서는 시작번지부터 configuration 되는 순서대로 저장되어야 한다.

명령 : makeprom -t -s 1024 -f exo -o sound -u 00 chip1 chip2 chip3 chip4
입력 : chip1.bit, chip2.bit, chip3.bit, chip4.bit
출력 : sound.exo

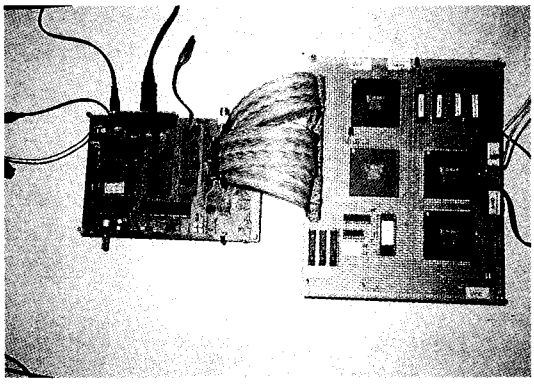
출력되는 파일은 hex 포맷으로 일반 롬 라이터를 사용하여 EPROM에 저장될 수 있다. 음원 DSP의 4개의 칩에 대한 configuration 파일 크기는 이진 형태로 89,053 바이트이다.

8. 음원 DSP의 FPGA 구현 결과

위와 같은 방식으로 음원 DSP에 대한 FPGA를 configuration 하는데 사용될 EPROM을 만들고 FPGA 보드를 완성하였다. 그림 5에 음원 DSP의 FPGA 보드를 보였고, 그림 6에는 FPGA 보드가



〈그림 5〉 음원 DSP를 구현한 FPGA 보드



〈그림 6〉 FPGA 보드를 포함한 음원 모듈 시스템

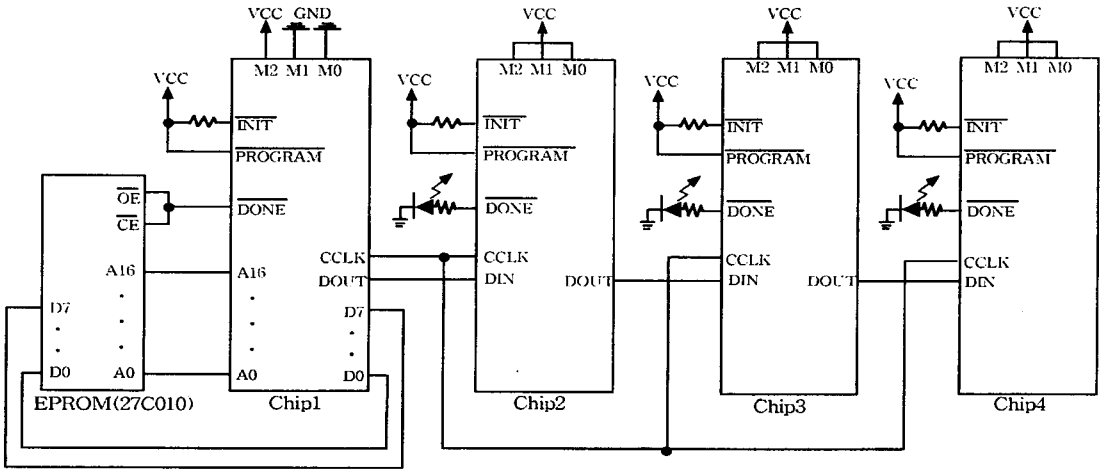
〈표 3〉 Xilinx FPGA의 configuration 방식

Configuration Mode	M2:M1:M0	Configuration scheme
Slave Serial	1:1:1	외부 디바이스로부터 시리얼 비트 스트림과 클럭을 제공받음
Mater Serial	0:0:0	Xilinx Serial PROM을 사용
Synchronous Peripheral	0:1:1	외부 디바이스로부터 바이트 단위의 데이터와 클럭을 제공받음
Asynchronous Peripheral	1:0:1	마이크로프로세서와 handshaking 방식으로 바이트 단위의 데이터를 제공받음
Master Parallel-High	1:1:0	자체적으로 어드레스를 발생시켜 EPROM으로부터 바이트 단위의 데이터를 읽어옴 (어드레스 증가)
Master Parallel-low	1:0:0	자체적으로 어드레스를 발생시켜 EPROM으로부터 바이트 단위의 데이터를 읽어옴 (어드레스 감소)

포함된 음원 모듈 시스템을 보였다.

Xilinx FPGA의 configuration 방식은 6가지 모드로 나뉘어진다. 이 모드를 구분시켜 주는 핀은 M2,M1,M0 이고, 이 핀들은 configuration을 위한 전용 핀들이다.

음원 DSP를 구현한 FPGA 보드는 4개의 FPGA를 Daisy chain 형태로 연결하여 하나의 EPROM으로 모든 칩에 대한 configuraion이 가능하도록 하였다. 가장 먼저 configuration 되는 FPGA는 master parallel 모드이며 나머지 3개는 serial slave 모드를 사용한다. 파워 온시 마스터 FPGA는 자체적으로 어드레스를 발생시켜 EPROM에 있는 configuration 정보를 읽어 들이며, 자신의 configuration이 끝나면 나머지 데이터를 슬레이브 단의 FPGA에 전달하므로써 순차적으로 모든 FPGA의 configuration이 이루어지고, 전체 칩에 대한 configuration이 끝나면 사용자의 I/O 핀들이 액티브 상태가 된다. 그림 7에 음원



〈그림 7〉 음원 DSP 보드의 configuration 연결 형태

DSP의 FPGA 보드에 대한 configuration 연결 형태를 보였다.

4개의 FPGA로 구현된 음원 DSP의 실제 동작을 확인하기 위해 그림 6과 같은 음원 모듈 시스템을 사용하였다. 미디 신호는 PC의 Cakewalk라는 시퀀스 프로그램에 의해 발생되어 전용 케이블을 통해 음원 모듈 시스템으로 전달된다. 음원 모듈 시스템은 미디 신호에 따라 악기음을 합성하여 스피커를 통해 음을 들려준다. 여러 개의 악기음을 연주하는 음악 프로그램을 이용하여 음원 DSP가 제대로 악기음을 합성해 내는지를 들어 보므로써 동작 여부를 확인하였다.

초기에는 설계된 음원 DSP 내부의 모든 메모리를 FPGA 내부에서 구현하였다. 메모리부의 AROM과 파형 발생부의 정현파 롬은 그 크기대로 구현되어야 하지만, PRAM은 동작 주파수에 따라 그 크기가 달라질 수 있다. 원래 설계된 음원 DSP는 44.1Khz의 샘플링 주파수로 32개의 악기음을 동시에 내는 것이었다. 한 악기음 합성을 위해 16개의 20비트 파라메터가 필요하므로 32개에 대해서는 512 워드 용량의 PRAM을 필요로 한다. 그리고 여기에 필요한 동작 주파수는 44Mhz이다. 그러나 구현된 FPGA 보드로써 44Mhz의 동작 속도를 낼 수 없으므로, 처음에는 8개 정도의 악기음만을 합성하는 것을 목표로 하였다. 이것을 위해서는 11Mhz로 FPGA가 동작되고, PRAM의 용량은 128 20 비트가 필요하다. 그러나 이렇게 구현된 음원 DSP는 전혀 악기음을 합성해내지 못하였다. 4개의 FPGA 칩 중 어느 것이 잘못 동작되는지를 알아내는 것은 매우 어려운 일이었으나, 마이크로프로세서와 음원 DSP의 동작을 제어하는 시스템 프로그램을 수정하여 음원 DSP의 각 기능 블록들을 테스트해 보므로써 각 칩에 대한 동작 여부를 어느 정도 확인할 수 있었다. 그 결과 메모리와 인터페이스부가 들어간 Chip1이 제대로 동작하고 있지 않다는 것을 알았다. 그래서 동작 주파수를 낮추고, PRAM의 크기도 훨씬 줄이기로 하였다. 2개의 악기음을 동시에 합성시키는 정도로 하기로 하고, 3Mhz의 동작 주파수에 32 20비트로 PRAM을 사용하였다. 이렇게 하므로써 악기음이 정상적으로 합성되는 것을 들을 수 있었다. XC4010이 메모리와 랜덤 로직을 동시에 구현할 수 있지만, 초기에는 큰 용량의 메모리와 랜덤 게이트가 사용되어 원하는 주파수에서 제대로 동작하지 않은 것이라 여겨졌다. 그래서 그림 5의 FPGA 보드에 나와 있는 것처럼 PRAM은 일반 SRAM을 사용하여 외부에 두고, AROM과 정현파 롬은 빠른 EPROM으로 구현하여 칩 외부에 두었다. 이렇게 하므로써 6Mhz 정도의 주파수까지 FPGA 보드가 제대로 동작하는 것을 확인할 수 있었다.

IV. 결 론

지금까지 본 실험실에서 설계된 음원 DSP를 Xilinx FPGA로 구현하는 것에 대하여 설명하였다. 초기의 ASIC을 목표로 설계된 로직도 FPGA로 구현하므로써 실험실 수준에서 간단하게 실제 동작을 확인할 수 있고, 빠르게 프로토타입 IC를 만들 수 있다는 것을 알 수 있었다.

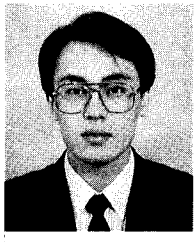
음원 DSP를 FPGA로 만드는 과정에서 느꼈던 것은 사용하는 FPGA의 내부 구조를 잘 이해하고, 여기에 잘 매핑될 수 있는 형태의 로직을 구현하게 되면 적은 하드웨어를 사용하면서 ASIC에 버금가는 성능의 IC를 만들 수 있다는 것이다. 최근에는 FPGA의 테크놀로지가 매우 발전해 수용할 수 있는 게이트 수가 10만 게이트 정도인 것도 있으며, 동작 주파수도 수십 Mhz까지 동작할 수 있는 FPGA가 나와 있는 실정이다. Custom computing machine으로서 FPGA를 사용하고 있는 것도 여러 군데서 볼 수 있다. 다품종 소량의 ASIC은 FPGA로 대체될 날이 곧 올 것이라 여겨진다.

참 고 문 헌

[1] "The Programmable Logic Data Book",

Xilinx, 1994.
[2] "Viewlogic Interface User Guide", Xilinx, 1993.
[3] John W. Gordon, "System Architecture for Computer Music", Computing Surveys, Vol. 17, No.2, June 1985.
[4] Richard F. Moore, "Elements of Computer Music", Prentice Hall, 1990.
[5] Charles Dodge, Thomas A. Jerse, "Computer Music", Schirmer Books, 1985.
[6] Hal Chamberlin, "Musical Applications of Microprocessors", Hayden Book Company, 1985.
[7] 박주성, 김형순 외, "전자악기용 디지탈 신호처리 VLSI 개발", 과학기술처, 1993

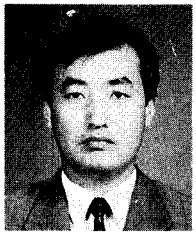
저 자 소 개



張 豪 根

1968年 1月 15日生
1993年 2月 부산대학교 전자공학과 졸업(학사)
1995年 2月 부산대학교 대학원 전자공학과(석사)
1995年~현재 부산대학교 전자공학과 졸업(박사)

관심 분야 : DSP 설계 및 응용, 컴퓨터 구조, Sound 합성



朴 柱 成

1953年 12月 19日生
1976年 2月 부산대학교 전자공학과 졸업(학사)
1978年 2月 한국과학기술원 전기 및 전자공학과(석사)
1989年 8月 University of Florida 전자공학과(박사)

1978年 3月~1985年 8月 한국전자통신연구소 연구원, 선임연구원, 반도체 설계 개발실장 역임
1985年 8月~1989年 8月 University of Florida 연구조교
1989年 8月~1991年 2月 한국전자통신연구소 집적회로 연구부 연구위원
1991年 3月~현재 부산대학교 공과대학 전자공학과 교수

주관심 분야 : DSP 설계 및 응용, Sound합성, 반도체 소자 모델링