흐름도를 이용한 인터페이스 회로 생성 알고리즘에 관한 연구

A Study on the Interface Circuit Creation Algorithm using the Flow Chart

우경환* , 이천희**
Woo, Kyong Hwan* , Yi, Cheon Hee**

Abstract

In this paper, we describe the generation method of interface logic which replace between IP & IP handshaking signal with asynchronous logic circuit. Especially, we suggest the new asynchronous sequential **"Wave**form **to VHDL"** code creation algorithm by flow chart conversion : **Wave2VHDL** - if only mixed asynchronous timing waveform is presented the level type input and pulse type input for handshaking, we convert waveform to flowchart and then replace with VHDL code according to converted flowchart. Also, we confirmed that asynchronous electronic circuits are created by applying extracted VHDL source code from suggest algorithm to conventional domestic/abroad CAD Tool, Finally, we assured the simulation result and the suggest timing diagram are identical.

Key Words; Interface, Wave2VHDL, Asynchronous Sequential Waveform, VHDL.

* 우송공업대학
** 청주대학교

1. 서론

최근에 SOC(System On a Chip)에 대한 관심이 높아지면서 다양한 기능을 구현할 수 있는 대규모 디지털 시스템의 설계 수요가 요구되고 있다. 이와 같은 대규모 시스템은 고속화되고 고집적화 될수록 동기방식의 전역 클럭(Global Clock)만 사용한 설계로는 클럭 스큐(Clock Skew) 현상을 제거하기가 매우 어려워진다. 따라서 이러한 문제를 해결하기 위한 방법의 하나로 전역 클럭을 사용하지 않는, 비동기 회로에 대한 연구가 활발히 진행되고 있다.[1, 2]

한편, 프로세서가 포함된 시스템을 설계 할 때 프로세서와 주변장치간의 제어신호를 핸드쉐이크 하는 방법은 시스템 운영 소프트웨어(Operating System)내부에서 해당신호들을 정의하고 제어하는 소프트웨어로 처리하는 방법이 있고, 프로세서와 주변장치 사이에 인터페이스용 집적회로를 설계하여 하드웨어로 처리하는 방법이 있다.

소프트웨어로 처리하는 방법을 살펴보면, 프로세서가 포트들의 동작상태를 수시로 감시해야 하기 때문에 소프트웨어의 추가부담이 발생하게되며, 감시 및 제어에 소요되는 시간만큼의 지연시간이 발생하게 된다. 특히 다양한 주변장치가 존재하는 시스템에서는 감시루틴으로 인한 지연이 시스템에 심각한 영향을 줄 수 도 있다. 이와 같은 문제점을 보완하기 위해 하드웨어로 처리하는 방법을 살펴보면 다음과 같다. 주변장치의 입력신호 또는 상태 신호들을 이용하여 실시간으로 프로세서를 직접 인터럽트(Interrupt)시키는 하드웨어에 의한 비동기식 신호처리 방법을 고려 할 수 있다. [3] 이 방법은 주 클럭(메인 클럭 : Main Clock)을 사용하지 않고, 그 대신에 명령이나 데이터의 도착에 근거하여 명령을 실행하는 데이터-흐름 회로 구조가 있으며, 이 구조는 감시루틴을 사용하는 소프트웨어적 신호처리보다 고속으로 동작 할 수 있는 장점이 있다.

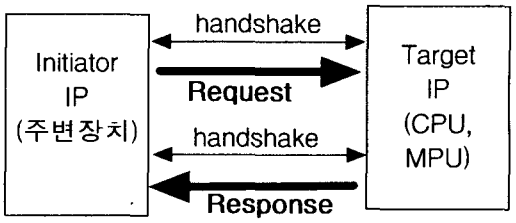
또한 클럭을 사용하지 않으므로 명령이나 데이터 신호가 없을 때에는 IC가 전력을 거의 사용하지 않게 되어 저 전력 소자를 구현할 수 있는 장점이 있다.[4, 5]

<그림 1>의 예를 살펴보기로 하자. <그림 1>은 SOC 구현 시에 프로세서와 IP(Intellectual Property) 또는 IP와 IP 사이의 인터페이스에 필요한 신호들을 나타내고있다. 이 부분은 통상적으로 <그림 2>와 같은 타이밍 파형이 함께 제시된다.

이 인터페이스 부분을 논리회로로 구현하게 되면 앞에서 살펴본 바와 같은 장점들이 있으나, 클럭 신호를 생략한 상태의 비동기 순차회로로 설계하기가 용이하지는 않다. 또한 본 예에서는 요구와 응답 및 핸드쉐이크 신호만 나타내었으나, 제어 신호들이 많아질수록 설계의 어려움이 가중된다[6].

본 논문에서는 이와 같은 인터페이스 회로들을 용이하게 설계할 수 있는 새로운 개념의 "파형변환 알고리즘"을 제안하고 이 알고리즘을 IP 인터페이스 논리회로의 설계에 적용함으로써, 제안된 알고리즘의 유효성을 입증하고자 하였다.

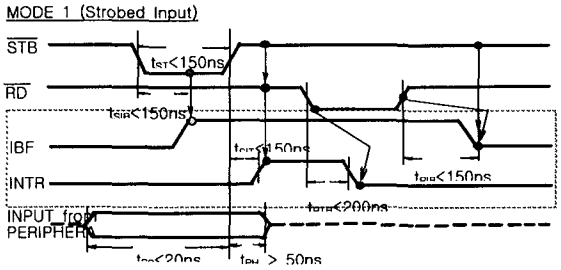
본 논문의 제 2장에서는 기존의 비동기식 펄스형 순차회로 파형 합성 알고리즘들을 고찰하였으며, 제 3장에서는 IP 인터페이스 논리회로를 용이하게 설계할 수 있는 새로운 파형 변환 알고리즘을 제안하고 이의 구현 방법을 상술하였으며, 제 4장에서는 파형변환 알고리즘의 구현 결과를 기존의 CAD용 설계 툴(Tool)에 적용하여 그 유효성을 증명하였으며, 제 5장에서 결론을 기술하였다.[7]



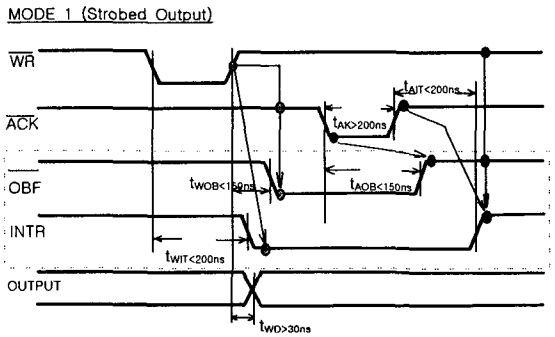
<그림 1> IP 인터페이스 블럭도
Fig 1. Block diagram of IP interface

2. 파형에 의한 비동기 펄스형 순차 회로 합성 알고리즘의 고찰

IP 인터페이스에 있어서 일반적인 요구(Request)와 응답(Response)에 관련된 구체적인 타이밍 파형을 <그림 2>와 <그림 3>에 나타내었다.[3, 6]



<그림 2> IP 인터페이스의 요구 타이밍도
Fig 2. Request timing diagram for IP interface



<그림 3> MPU IP의 응답 타이밍도
Fig 3. Response timing diagram for MPU IP

<그림 2>와 <그림 3>처럼 제시된 타이밍 파형만 가지고, IP 인터페이스 논리회로와 유사한 비동기 펄스형 순차회로(차후 인터페이스 논리회로와 비동기 펄스형 순차회로를 혼용하기로 함)를 합성할 수 있는 알고리즘 중에 현재상태와 다음상태를 상태도로 표현하여, 천이도를 작성한 후 논리방정식을 추출하고, 마스터 플립플롭용 회로를 합성하여 슬레이브 플립플롭용 회로를 추가로 합성하는 기법이 있다. 이 기법은 적용 예정인 마스터용 플립플롭과 슬레이브용 플립플롭을 사전에 설정해서 그에 알맞은 회로를 생성하는 방법으로, 주 클럭을 반드시 인가해야 하며, 두 번의 절차를 거쳐서 합성하는 등, 알고리즘이 매우 복잡하고 구현에 어려움이 많다. 따라서 서론에 언급한 것과 같이 주 클럭을 사용하지 않고, 대신에 명령이나 데이터의 도착에 근거하여 명령을 실행하는 데이터-흐름 회로 구조로 된 IP 인터페이스 논리회로의 설계에는 적용할 수 없다는 문제점이 있다.[8]

제시된 타이밍 파형만으로 펄스형 순차회로를 합성할 수 있는 알고리즘으로 Janus 알고리즘이 있다. Janus 알고리즘은 두 IP에 대한 이벤트(Event) 중심으로 기술된 타이밍도를 입력으로 받아들여 이를 이벤트 그래프로 변환시킨 후 SR-래치와 D-FF으로 구성된 세 종류의 템플리트를 동기와 비동기 특성에 맞추어 적용시키는 합성 절차를 거쳐서 논리회로를 생성시키는 방법이다. 이 방법 역시 템플리트용 D-FF의 클럭 입력에 주 클럭을 반드시 인가해야 하는 문제점이 있다.[9]

또한 제시된 펄스형 순차회로 파형을 전자회로로 합성 할 수 있는 "진리치 비교 알고리즘(TRUth values COMparison algorithm : TRUCOM)"이라고 명명한 알고리즘도 제안되어 있다.[10] 이 알고리즘은 제시된 파형을 플립플롭과 같은 저장소자로 구현할 수 있는 펄스형 순차회로 합성 알고리즘으로서, 레벨형 입력과 펄스형 입력이 혼합된 비동기 회로에서, 사용하려는 플립플롭을 설정한 후 파형 동작 순서에 따라서 신호와 매칭 시킬 단자를 가정하여 진리치를 대입한 후, 가정한 단자의 논리 방정식을 구현하여 회로를 생성 할 수 있는 설계 알고리즘이다. 이 방법은 첫 번째로 예를 든 천이도에 의한 방법과는 달리 주 클럭을 사용하지 않으므로, 본 연구에서 구현하려는 비동기식 회로의 합성이 가능하기는 하지만, 단계별로 논리 회로도를 그려나가야 하기 때문에 프로그램화하기가 어렵고, 구현된 회로는 디자인 룰이 변경 될 경우에 시뮬레이션 과정에서 발생될 수 있는 타이밍 문제점들을 해결하기가 어렵다. 따라서 본 논문에서는 새로운 개념의 "파형변환 알고리즘"을 제안하여 이

문제점들을 해결하고자 하였다.

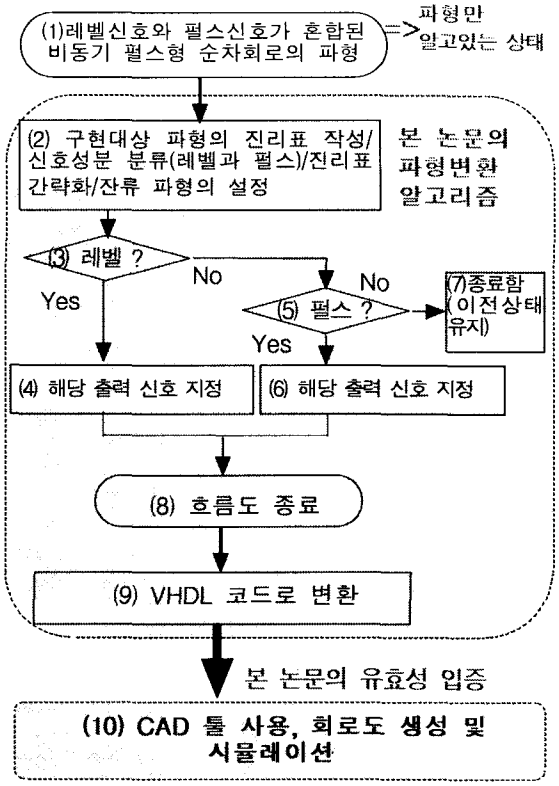
본 논문에서는 <그림 2>와 <그림 3>처럼 제시된 인터럽트 발생기의 타이밍도를 예로 적용 방법을 기술하였다. 이 타이밍도는 주변장치와 IP, 프로세서와 IP 사이를 인터페이스 시키는 일반적인 파형이며, 주 클럭이 없는 비동기식으로 구동되어야한다.[3, 6]

<그림 2>를 간략히 설명하면, 인터페이스 회로는 주변장치 IP로부터 스트로브(STB) 입력을 받아서 그림의 점선 부분과 같이 IBF (Interrupt Buffer Full)를 출력하고, 읽기(RD) 신호의 상승(Rising) 엣지에서 IBF를 종료 해야한다. 그림에서 STB와 RD가 핸드쉐이킹되며, IBF가 요구신호가 된다. 또한 STB 신호는 레벨 입력(Level Input)이고, RD는 엣지입력(Edge Input)이다. 따라서 이 부분은 레벨 신호와 펄스 신호들이 혼합된 순차회로로 설계해야 한다. 또한 그림에서 INTR 출력도 STB와 RD 입력에 의해 출력된다.

3. 파형변환(Wave2VHDL) 알고리즘의 제안

3-1. 파형변환 알고리즘의 개념

파형변환 알고리즘(Asynchronous sequential **waveform to VHDL** code creation algorithm by the flow chart conversion, 차후 **Wave 2VHDL**로 호칭 함)이란 핸드쉐이킹을 위한 비동기식 순차회로 파형이 제시되었을 때, 이 파형을 흐름도로 변환시키고 변환된 흐름도에 의하여 VHDL 코드로 대체하는 알고리즘으로서, 레벨형 입력과 펄스형 입력이 혼합된 비동기 파형에서, 파형의 진리표를 작성하고 성분을 분류한 후 파형의 동작 순서에 따라서 개별신호들을 흐름도로 바꾸고, 흐름도의 단계별로 VHDL 코드를 대입시킨 후 VHDL 소스를 완성함으로써 CAD 툴을 활용하여 비동기식 전자회로를 생성시킬 수 있는 전자회로 설계에 알맞은 알고리즘이다. 이 알고리즘의 블럭도를 <그림 4>에 나타냈다.



<그림 4> wave2VHDL 알고리즘의 블럭도
Fig 4. Block diagram of wave2VHDL algorithm

그림 내부에 표시된 번호는 제시된 파형에서 출발하여 회로도를 생성하는 것까지의 진행순서를 나타낸 것이다.

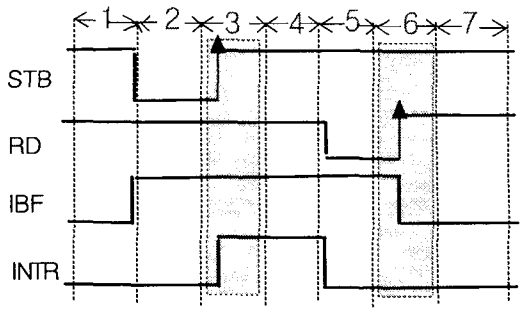
3-2. 파형변환 알고리즘의 절차 및 구현 방법

두 개의 IP 사이에서 데이터를 주고받는 인터페이스 논리회로 설계 방법에 대하여 <그림 2>와 같이 필요한 파형만 제시된 상태를 예로하여 기술하기로 한다.

파형변환(Wave2VHDL) 알고리즘은 5단계의 절차를 거쳐서 구현할 수 있다. 파형변환 알고리즘의 구현 방법을 단계별로 상술하면 다음과 같다.

1) 제 1단계(Step 1) : 구현대상 파형의 상승 엣지 구간을 확대한 진리표 작성

구현대상 파형에서 출력의 변화가 발생하는 구간 및 상승 엣지 구간을 확대하여 구획한 후 파형 동작순서를 정하고 이에 따른 진리표를 작성한다. <그림 2>처럼 제시된 파형을 구현한 예는 아래 <그림 5>와 같다.



		파형 동작순서						
		1	2	3	4	5	6	7
입력	STB	1	0	↑	1	1	1	1
	RD	1	1	1	1	0	↑	1
출력	IBF	0	1	1	1	1	0	0
	INTR	0	0	1	1	0	0	0

<그림 5> 상승 엣지 구간을 확대한 진리표
Fig 5. Extension truth table for rising edge term

2) 제 2단계(Step 2) : 신호 성분 분류(레벨과 펄스)

구현 대상 파형의 진리표에서, 상승 엣지를 기준으로 하여 출력신호의 변화가 발생하였을 경우에는 펄스로, 이전상태를 유지하고 있을 경우에는 레벨로 신호성분을 분류한다. <그림 6>에 <그림 5>의 진리표를 분류한 예를 나타내었다.

<그림 6>에서 파형 동작순서 2와 3을 살펴보면, 출력 IBF는 파형 동작순서 3에서 파형 동작순서 2의 이전상태를 유지하고 있으므로 "레벨"이며, 파형 동작순서 5와 6에서 "1"에서 "0"으로 신호의 변화가 발생하였으므로 "펄스"로 분류한다.

또한, 파형 동작순서 2와 3의 출력 INTR은 "0"에서 "1"로 신호의 변화가 발생하였으므로 "펄스" 이며, 파형 동작순서 6에서 파형 동작순서 5와 같은 이전상태를 유지하고 있으므로 "레벨"로 분류한다. 따라서 파형 동작순서 2와 3 및 파형 동작순서 5와 6에서 레벨과 펄스가 공존하는 것으로 신호성분을 분류한다.

		파형 동작순서						
		1	2	3	4	5	6	7
입력	STB	1	0	↑	1	1	1	1
	RD	1	1	1	1	0	↑	1
출력	IBF	0	1	1	1	1	0	0
	INTR	0	0	1	1	0	0	0

<그림 6> 출력 신호성분 분류
Fig 6. Classify for Output signal components

3) 제 3단계(Step 3) : 진리표 간략화 및 잔류 파형의 설정

2 단계에서 분류한 레벨과 펄스가 공존하는 파형 동작순서의 우측방향들을 연속적으로 비교하여, 동일한 출력값의 파형 동작순서는 찾아서 생략한다.

<그림 7>에 <그림 5>의 진리표를 비교한 예를 나타내었다. <그림 7>의 파형 동작순서 3과 4를 비교하여 살펴보면, 출력들이 동일하므로 파형 동작순서 4를 생략할 수 있다. 이 때 비교하는 방법은 파형 동작순서 3의 출력들과 파형 동작순서 4의 동일 출력행끼리 익스크루시브오아(XOR)를 하면 되며, 결과값이 "0"이면 동일하고 "1"이면 상이함을 알 수 있다.

		파형 동작순서						
		1	2	3	4	5	6	7
입력	STB	1	0	↑	1	1	1	1
	RD	1	1	1	1	0	↑	1
출력	IBF	0	1	1	1	1	0	0
	INTR	0	0	1	1	0	0	0

<그림 7> 동일 출력치 비교
Fig 7. Comparison with same output

또한, 위와 같은 방법으로 파형 동작순서 4와 5, 5와 6, 6과 7, 7과 1, 1과2의 출력들을 비교하면, 파형 동작순서 7과 1 을 생략할 수 있다. 따라서 파형 동작순서 2, 3, 5, 6이 남게 되며, <그림 8>은 출력신호 IBF에 대하여 남아 있는 파형 동작순서들의 성분을 정리한 것이고(STB가 레벨, RD가 펄스), <그림 9>는 출력신호 INTR에 대하여 남아있는 파형 동작순서들의 성분을 정리한 것이다(RD가 레벨, STB가 펄스).

		파형 동작순서 2 3	5 6	
입력	STB	0 ↑	1 1	(레벨신호)
	RD	1 1	0 ↑	(펄스신호)
출력	IBF	1 1	1 0	

<그림 8> IBF에 대한 입력신호 성분
Fig 8, Input signal component for IBF

		파형 동작순서 2 3	5 6	
입력	STB	0 ↑	1 1	(펄스신호)
	RD	1 1	0 ↑	(레벨신호)
출력	INTR	0 1	0 0	

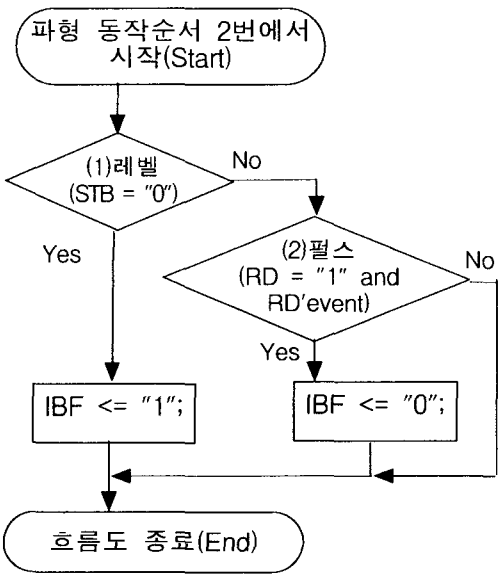
<그림 9> INTR에 대한 입력신호 성분
Fig 9. Input signal component for INTR

4) 제 4단계(Step 4) : 잔류 파형 동작순서 흐름도 작성

3 단계 잔류 파형에 대하여 출력신호당 1개씩의 흐름도를 작성한다. 이 때 레벨 신호를 흐름도의 시작 지점(Start point)으로 하며, 우측방향으로 진행하도록 작성한다. <그림 10>과 <그림 11>에 <그림 8>과 <그림 9>의 잔류 파형 동작순서에 대한 진리표를 흐름도로 작성한 예를 나타내었다.

<그림 8>에서 레벨 신호는 파형 동작순서 2이며, 여기서부터 흐름도가 시작된다. 따라서 STB가 "0"이면 IBF는 "1"이 되는 것을 <그림 10> 흐름도의 (1)번째 비교문으로 작성한다. 이 후 STB가 "1" 일 경우를 살펴보면, 파형 동작순서 6에서 RD가 라이징될 때만 IBF가 "0"이 되며, 이것을 (2)번째 비교문으로 작성한다. 이 이외의 경우는 파형 동작순서 3과 5가 되며 이 때는 이전 상태를 유지하며 종료하게 된다.

또한, <그림 9>에서 레벨 신호는 파형 동작순서 5이며, 여기서부터 흐름도가 시작된다. 따라서 RD가 "0" 이면 INTR이 "0"이 되는 것을 <그림 11> 흐름도의 (1)번째 비교문으로 작성한다. 이 후 RD가 "1" 일 경우를 살펴보면, 파형 동작순서 3에서 STB가 라이징 될 때만 IBF가 "1"이 되며, 이것을 (2)번째 비교문으로 작성한다. 이 이외의 경우는 파형 동작순서 6과 2가 되며 이 때는 이전 상태를 유지하며 종료하게 된다.

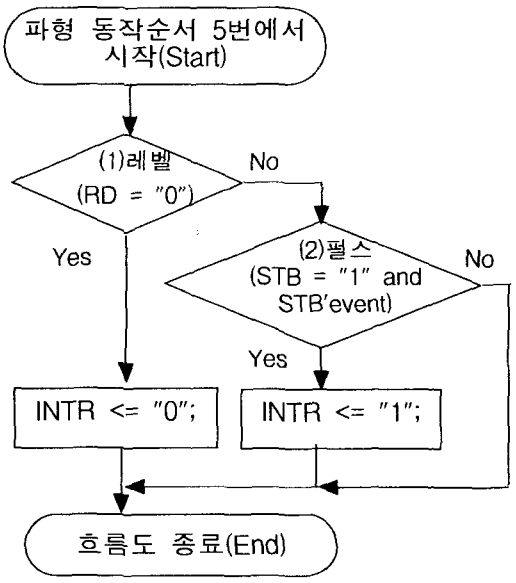


<그림 10> IBF 진리표를 흐름도로 작성한 예
Fig 10. The example flowchart for IBF truth table

5) 제 5단계(Step 5) : 흐름도로부터 VHDL 코드 작성

제 4 단계의 흐름도로부터 VHDL 파일을 작성한다. 이 때 각 흐름도당 1개씩의 Process문을

작성하되, 파형 동작순서가 우선하는 레벨신호가 있는 흐름도부터 작성한다.



<그림 11> INTR 진리표를 흐름도로 작성한 예
Fig 11. The example flowchart for INTR truth table

<그림 12>에 VHDL 파일로 작성한 예를 나타내었다. 그림에서 흐름도 #1(#1 flow)으로 표시한 부분이 IBF에 해당하는 부분이고, 흐름도 #2(#2 flow)으로 표시한 부분이 INTR에 해당하는 부분이다.

이상으로써, 파형변환(Wave2VHDL) 알고리즘의 제안을 상술하였고, 이 알고리즘의 유효성을 살펴보면 다음과 같다.

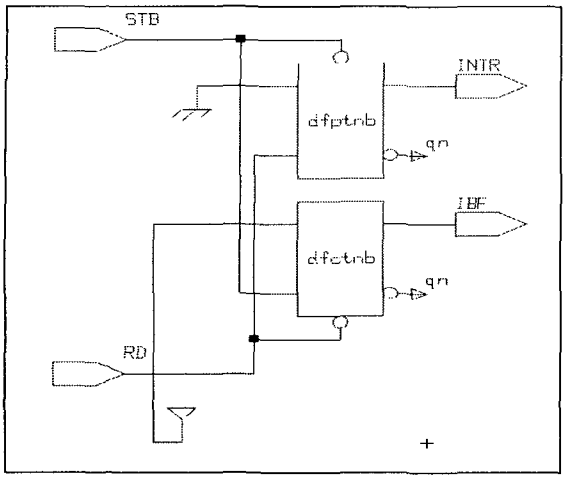
4. 파형변환(Wave2VHDL) 알고리즘의 시뮬레이션

본 논문의 파형변환 알고리즘은 멘토(Mentor)사의 르노아르99(Renoir99)에서 <그림 10>과 <그림 11>처럼 제안된 흐름도를 입력하여 VHDL로 합성 후 모델심(ModelSim)으로 시뮬레이션을 수행하였으며, 시놉시스(Synopsys)사의 브이에스에스(VSS)로도 합성을 수행하였다.

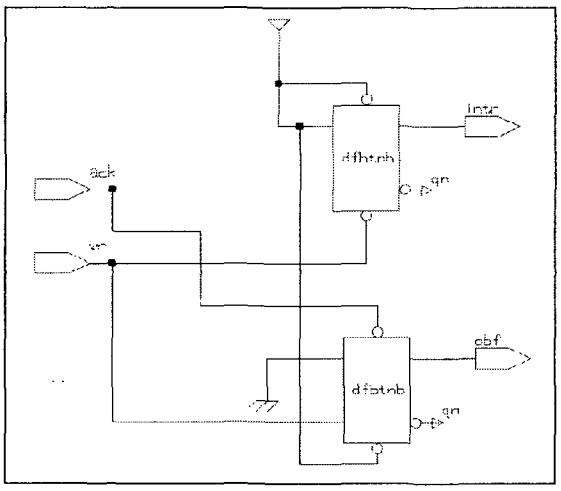
```
library IEEE;
use IEEE.std_logic_1164.all;
entity doubleEdge is
   port ( stb : IN  std_logic;
          rd : IN  std_logic;
          ibf : OUT  std_logic;
          intr : OUT std_logic);
end doubleEdge;
architecture doubleEdge of doubleEdge is
begin
-- #1 flow Start ----
   process (rd, stb)
   begin
      if rd = '0' then
         ibf <= '0';
     elsif stb = '1' and stb'event then
         ibf <= '1';
      end if;
   end process;
-- #1 flow End ----
-- #2 flow Start ----
   process (rd, stb)
   begin
      if stb = '0' then
         intr <= '1';
     elsif (rd = '1' and rd'event) then
         intr <= '0';
      end if;
   end process;
-- #2 flow End ----
end doubleEdge;
```

<그림 12> VHDL 파일의 예
Fig 12. The example of VHDL file

또한 ETRI의 0.8um SOG 디자인 룰을 적용한 로드캡(LODECAP)의 VHDL 합성기에서 <그림 12>처럼 제안된 VHDL 코드를 입력함으로써, <그림 2>와 <그림 3>처럼 제시된 타이밍도에 대하여 <그림 13>과 <그림 14>와 같은 인터페이스 회로도를 합성 할 수 있었다. 또한 로드캡의 파형 편집기를 이용하여 IBF와 INTR에 대하여 <그림 15>와 같은 시뮬레이션 결과를 얻을 수 있었으며, 이는 <그림 2>에서 제시된 타이밍도와 동일함을 입증하였다.[7]



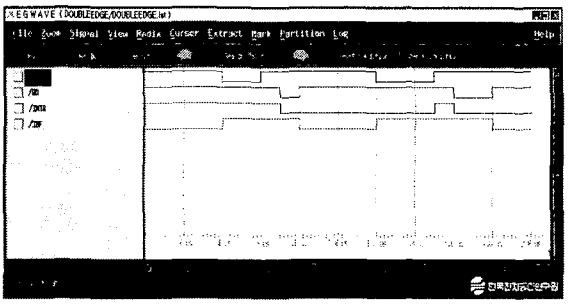
<그림 13> 요구 타이밍의 인터페이스 회로도
Fig 13. Interface circuit for request timing



<그림 14> 응답 타이밍의 인터페이스 회로도
Fig 14. Interface circuit for response timing

한편, <그림 13>과 <그림 14>의 회로도를 살펴보면, 각 입력신호(STB, RD, ack, wr)들이 F/F의 클럭 단자에 접속되고, 구동되므로 주 클럭과 관계없이 비동기로 동작함을 알 수 있으며, 각 출력들은 입력신호(STB, RD, ack, wr)들에 대하여 리얼타임으로 동작함을 알 수 있다.

이와 같은 특징들은 파형변환 알고리즘으로 생성되는 VHDL 코드들에 대하여 IP화 할 수 있는 가능성을 보여준 것이다.



<그림 15> 시뮬레이션 결과
Fig 15. simulation result

5. 결론

본 논문에서는 IP와 IP 간에 인터페이스를 위하여 레벨형 입력과 펄스형 입력이 혼합된 비동기 파형만 제시되었을 때, 이 제시된 파형만으로 인터페이스 회로를 설계할 수 있는 새로운 개념의 "파형변환(Wave2VHDL)" 알고리즘을 제안하였다.

파형변환 알고리즘이란 규칙에 따라서 입력 파형의 성분을 펄스와 레벨로 구분한 후 펄스에 해당하는 구간 시작 지점부터 파형의 동작 순서에 따라서 개별신호들을 흐름도로 바꾸고, 흐름도의 단계별로 VHDL 코드를 대입시킨 후 VHDL 소스를 완성함으로써 비동기식 인터페이스 회로를 생성시킬 수 있는 새로운 전자회로 설계에 관한 알고리즘이다.

본 논문에서는 제안된 파형변환 알고리즘으로부터 완성한 VHDL 소스코드를 로드캠의 VHDL 합성기(VHDL Synthesis)에 적용하여 IP 인터페이스를 위한 비동기식 전자회로가 생성됨을 확인하였고, 로드캠의 파형 편집기(Waveform Editor)에 적용한 시뮬레이션 결과와 제시된 타이밍도가 일치함을 확인함으로써 알고리즘의 유효성을 입증하였으며, 2종의 또 다른 CAD 툴(Tool)에서도 동일한 결과가 발생됨을 확인하였다.

본 논문에서 제안한 파형변환 알고리즘은 비교적 간단하면서 전자회로 구현이 매우 용이하므로

시스템 설계자들이 쉽게 숙지할 수 있으며, 결과물이 VHDL 코드로 생성되어 주 클럭을 생략할 수 있기 때문에 칩 구현시 디자인 룰이 바뀌어도 무난하게 설계할 수 있을 것으로 기대되므로 IP화하기에도 적합하다고 판단된다.

앞으로의 연구 과제는 본 파형변환 알고리즘을 이용하여 제시된 타이밍도의 파형들만 입력하는 것으로도 VHDL 코드가 생성될 수 있도록 프로그램화하는 방향으로 이에 대한 연구가 진행되어야 하겠다.

참고 문헌

[1] J.V. Woods, P. Day, S.B. Furber, J.D. Garside, N.C. Paver and S.Temple, "AMULET1: An Asynchronous ARM Processor," IEEE Trans. on Computer, Vol.46, No.4, pp.385-398, April 1997.

[2] 정성태, 캐리 선택과 캐리 우회 방식에 의거한 비동기 가산기의 CMOS 회로 설계 한국정보처리학회 제5권 제11호 P29802988, 1998년 11월

[3] Pic 16/17 *Micro controller data book*, Microchip Technology Inc, 1995

[4] C.L. Seitz, *System timing in Introduction to VLSI Systems*, Addison-Wesley, 1980.

[5] A. Bellaouar and M.I. Elmasry, "Low-power Digital VLSI Design." Kluwer Academic Publisher, 1995.

[6] Cheon-Hee, Yi, and Ki-Jong, Ha, "Development of Statistical Analizing Tool and System of Automatic Magnetizer", Journal of Korea Information Processing Society, Vol. 3, No. 4, P1014-1025, July 1996.

[7] Y. H. Bae, etc. "VHDL based ASIC design CAD system suppotting manual & automatic design," World Korean Scientist Workshop, July 1996.

[8] J. F. Wakerly, *Digital Design Principles and Practices*, Prentice-Hall, Englewood Cliffs, N.J., P265-295, 1990.

[9] Gaetano Borriello, and Randy H. Katz, "Synthesis and Optimization of Interface Transducer Logic," Proc. ICCAD '87, pp.274-277, (Event Graph)

[10] 이천희 외 1명, "인터럽트 발생기를 사용한 접속 비트 전환식 양방향 접속장치의 설계," 대한전자공학회 36권 C편 7호, P17-26, 1999년 7월

저자소개

우경환

1977년 청주대학교 졸업
1986년 숭실대학교 대학원 전자계산과 졸업(공학석사)
1998년 청주대학교 대학원 전자공학과 박사과정수료
1990년 ~현재 우송공업대학 부교수
관심분야 : VLSI & CAD, ASIC, 실시간처리

이천희

1971년 한양대학교 전자공학과 졸업(공학사)
1975년 성균관대학교 대학원 전자자료처리과 졸업(석사)
1981년 한양대학교 대학원 전자공학과 졸업(공학석사)
1987년 성균관대학교 대학원 전자공학과 졸업(공학박사)
1971년 2월 ~ 1972년 8월 한국마벨(전자업체)
1972년 9월 ~ 1977년 수송전기 공업고등학교 교사
1977년 3월 ~ 1979년 2월 동양공업전문대학 전자과 전임강사
1979년 3월 ~ 현재 청주대학교 전자공학과 교수
1983년 8월 ~1985년 8월 미국 산호세 캘리포니아 주립대학교 전자계산과 객원교수
관심분야 : VLSI & CAD, ASIC, System Design