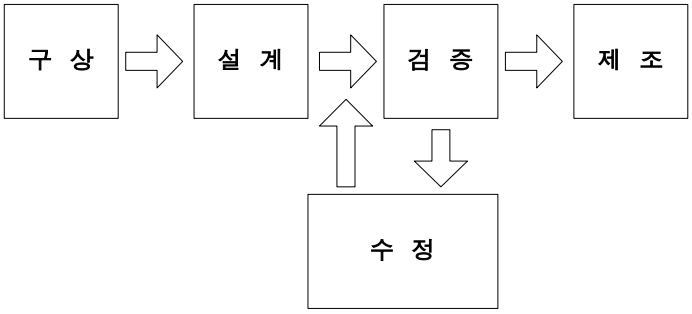
부록 A.
MAX + plus II
기본 사용법

A-1. Design Flow

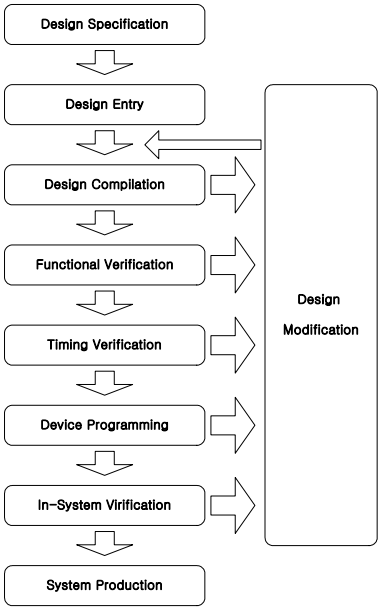
일반적으로 어떤 물건을 만들 때, [그림 A-1]과 같은 과정으로 진행합니다.



[그림 A-1] 제품 생산의 흐름

어떤 물건을 만들 것인지를 생각하고(구상), 그 물건을 만들고 (설계), 그 물건이 의도한 대로 만들어 졌는지 테스트합니다.(검증) 만약에 이상이 발견되면 다시 수정하여 검증하고, 이상이 없다면 제품 출하하는 과정으로 제품을 생산합니다.

하나의 시스템을 설계 하는 것도 마찬가지인데, [그림 A-2]은 MAX + Plus II를 이용해서 시스템을 설계하는 과정입니다.



[그림 A-2] 시스템 설계 시 설계 흐름

먼저 Design Specification에서는 어떤 논리 회로를 설계할 것인가를 구상합니다.
그리고, Design Entry에서 설계하고자 하는 논리 회로를 어떤 방법으로 어떻게 설계할 것인가를 결정하여 논리 회로를 설계합니다.
여기에서 어떤 방법에 해당하는 논리 회로를 설계하는 방법은 여러 가지 있는데, Graphic Design (Schematic Capture에 의한 방법)으로 설계하는 방법, HDL (Hardware Description Language ; 하드웨어 기술 언어)를 사용하여 설계하는 방법, 파형으로 설계하는 방법이 그것입니다.

위에서 설계한 내용에 대해 문법을 검사하여 이상이 있다면 그 이상에 대한 메시지 등을 알려주고, 이상이 없다면 설계한 논리 회로를 하드웨어적으로 만들거나 시뮬레이션 할 수 있는 파일 또는 프로그래밍 할 수 있는 파일 등을 생성하는 부분이 Compilation입니다.

이렇게 문법 오류까지 검증 된 논리 회로는 설계한 의도대로 동작이 잘 되는지에 대한 검증 과정을 합니다. 먼저 기능적으로 문제가 없는지를 살펴보는 Function Verification 과정을 하고, 그 후에 직접 디바이스의 특성을 이용하여 설계한 회로가 사용하고자 하는 디바이스나 클록 주파수에 의해 오동작을 하지 않는지를 살펴보는 Timing Verification 과정을 진행합니다.

위의 Function Verification과 Timing Verification은 PC상에서 소프트웨어로 검증하는 방법이고, 이렇게 검증된 논리 회로가 직접 하드웨어(시스템)에 연결되었을 때 정확하게 동작이 되는지를 검증해야 할 필요가 있습니다. 이 부분이 Device Programming과 In-System Verification입니다. 전자는 말 그대로 디바이스에 설계한 논리 회로를 프로그래밍 한다는 말이고, 후자는 시스템이 꾸며진 상태에서 하드웨어 테스트를 한다는 것입니다.

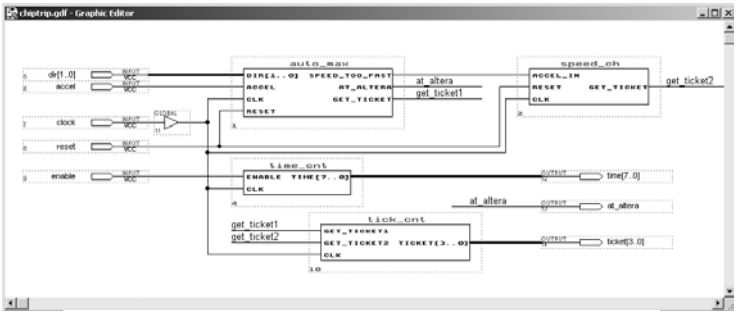
이렇게 모든 검증이 끝난 후에 비로서 제품이 완성되어 제조됩니다.

A-2. Project 선언

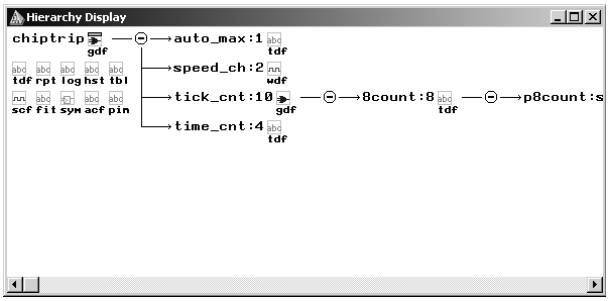
1. 역할

MAX + plus II 프로그램은 모든 설계를 프로젝트 중심으로 관리합니다. 프로젝트에 포함된 설계 파일 및 컴파일한 파일, 환경저장 파일, 시뮬레이션 파일, 프로그래밍 파일이 모두 같은 프로젝트 이름으로 관리가 됩니다.

또, 모든 설계를 프로젝트화 하였기 때문에 전체 프로젝트에 대한 관리도 각각의 프로젝트 별로 관리하게 되어 있습니다. [그림 A-3]은 MAX + plus II에서 기본으로 제공해 주는 예제 중 chiptrip이라는 예제이고, [그림 A-4]는 이 chiptrip 프로젝트에 대한 Hierarchy Display로, chiptrip 프로젝트에 대한 하위 블록들을 계층구조로 보여주는 것입니다. [그림 A-4]와 같이 프로젝트에 포함된 하위 블록 및 각 블록에 포함된 파일들을 확인할 수 있기 때문에 프로젝트를 관리하기 쉽도록 구성되어 있습니다.



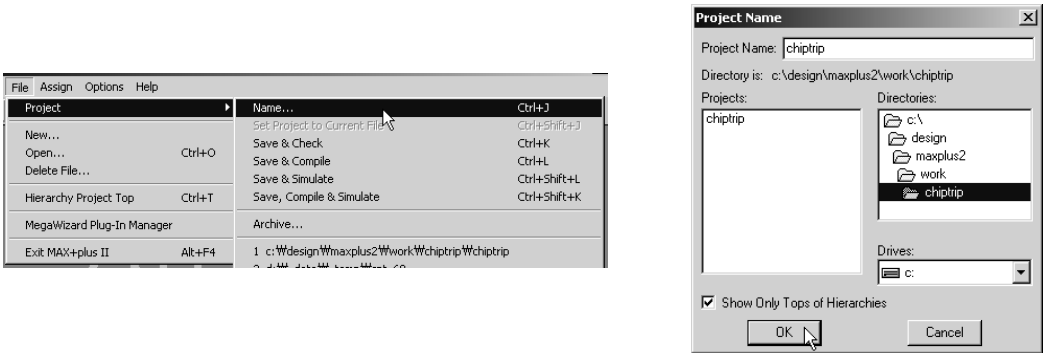
[그림 A-3] chiptrip Project



[그림 A-4] Hierarchy Display

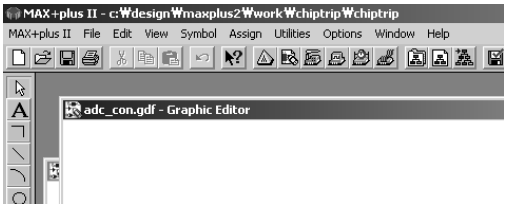
2. Project Name 설정

[그림 A-5]와 같이 MAX + plus II 프로그램의 File -> Project -> Name 메뉴를 선택해서 활성화되는 Project Name 창에서 작업할 새로운 프로젝트에 대한 이름을 설정하거나, 미리 작업된 프로젝트를 선택할 수 있습니다. .



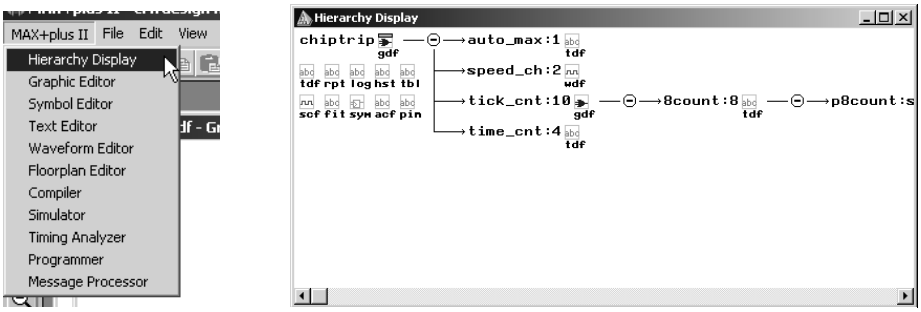
[그림 A-5] 메뉴에서 Project Name 선택과 Project Name 창

이렇게 프로젝트를 선언하면 [그림 A-6]과 같이 MAX + plus II 프로그램의 왼쪽 상단의 타이틀 바에 현재 프로젝트가 선언된 경로명 및 프로젝트의 이름이 표시됩니다.



[그림 A-6] 타이틀 바에 표시된 프로젝트 이름

작업을 진행하는 중에 프로젝트에 포함되어있는 파일들 - 시뮬레이션 파일, 프로그래밍 파일, 하위 블록 및 각 블록에 포함된 파일 - 을 쉽게 확인하고 관리할 수 있는 것이 Hierarchy Display입니다. 이것은 [그림 A-7]과 같이 MAX + plus II 프로그램의 MAX+ plus II -> Hierarchy Display 메뉴를 선택해서 확인할 수 있습니다.
[그림 A-7]의 오른 쪽 그림에서 각 블록별로 가지처럼 나누어져 있는 것을 볼 수 있는데, 각 블록에 포함된 아이콘이 그 프로젝트에 포함된 파일들입니다. 이 아이콘을 마우스의 왼쪽 버튼으로 더블 클릭하여 해당 파일을 불러올 수 있습니다.



[그림 A-7] 메뉴에서 Hierarchy Display 선택 및 창 활성화

A-2. Design Entry

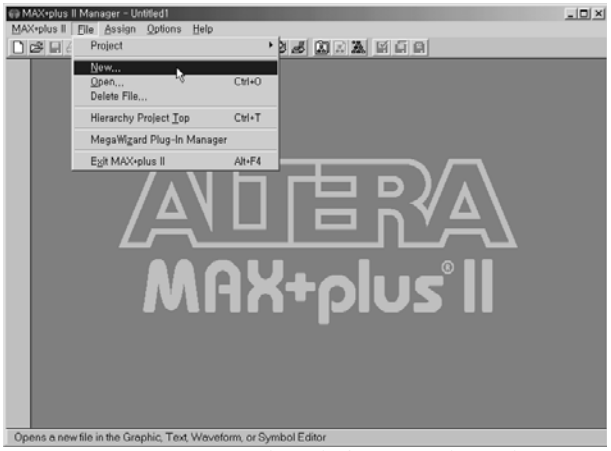
1. Graphic Editor (Schematic Capture에 의한 설계)

(1) 특징

특정 기능을 가진 심볼 라이브러리를 생성하여, 그 심볼 라이브러리의 입 출력 데이터 라인을 선(wire) 등으로 각 심볼 등의 입 출력 데이터 신호를 연결하여 논리 회로를 설계하는 방법입니다.

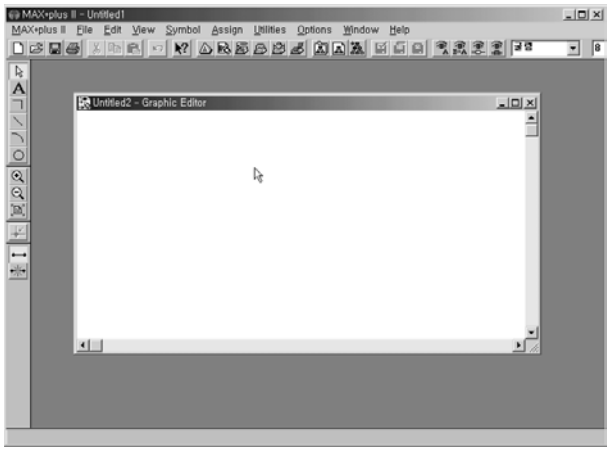
(2) Graphic Editor 선택하기

Graphic Editor를 이용해 하나의 논리 회로를 설계하기 위해서는 Graphic Editor 창을 활성화 시켜야 하는데, [그림 A-8]과 같이 File -> New 메뉴를 선택하여 [그림 A-9]의 New 창을 활성화 한 후 Graphic Editor라는 항목을 선택하여 OK 버튼을 누르면 [그림 A-10]와 같이 Graphic Editor 창이 활성화 됩니다.



[그림 A-8] File메뉴에서 New 항목 선택

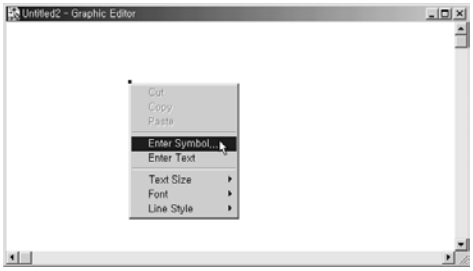
[그림 A-9] Graphic Editor File 선택



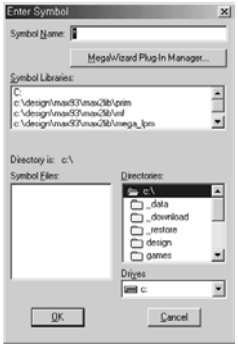
[그림 A-10] Graphic Editor File을 활성화 한 상태

(3) 심볼 라이브러리

MAX + plus II의 Graphic Editor에서는 도면을 마우스의 왼쪽 버튼으로 더블-클릭하는 방법 또는 마우스의 오른쪽 버튼을 눌러 Pop-Up 메뉴의 " Enter Symbol " 항목을 선택하는 방법으로 심볼 라이브러리를 불러 올 수 있습니다.



[그림 A-11] Pop-up 메뉴에서 "Enter Symbol"선택



[그림 A-12] Enter Symbol 창

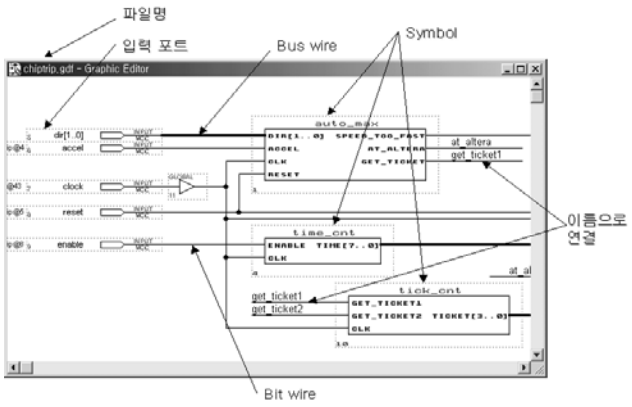
위의 Enter Symbol 창에서 직접 심볼 명을 쓰거나 해당 심볼 라이브러리가 들어 있는 디렉토리에서 찾는 방법으로 Symbol Library를 불러 올 수 있는데, 각 라이브러리 디렉토리에 대해 설명하면 다음과 같습니다.

* PRIM : Primitive를 뜻하는 것으로 가장 기본이 되는 AND 또는 OR 게이트, 입출력 포트 등이 여기에 포함됩니다.
* MF : Macro Function을 뜻하는 것으로 약간의 논리가 포함된 논리 회로 - 예를 들어 4 bit counter, 74 시리즈 등 - 가 포함됩니다.
* MEGA_LPM : 여기에서 LPM이란 Library of Parameterized Module으로 Mega-function이나 high level function module을 기본 설정과 Parameter 값만 변화시켜 줌으로써 쉽게 설계할 수 있도록 해주는 라이브러리입니다. 이처럼 일부 항목을 바꾸어서 범위가 큰 논리 회로를 쉽게 설계할 수 있도록 해줍니다.
* EDIF : Electronic Design Interchange Format을 뜻하는 것으로, 쉽게 논리 회로를 설계하는 다른 설계 소프트웨어(OR-CAD 등)에서도 사용할 수 있는 공통 symbol library라고 생각하면 됩니다.

(4) 각 Symbol의 연결

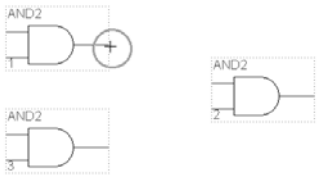
각 Symbol 간의 데이터 연결은 wire 또는 이름으로 연결해 줄 수 있습니다.

[그림 A-13]은 MAX + plus II 에서 제공해주는 기본 예제로 CHIPTRIP.GDF 파일입니다. 각 Symbol 라이브러리 간의 bus 데이터를 연결할 때는 선 굵기가 굵은 bus wire를 사용하고, bit 데이터를 연결할 때는 굵기가 가는 bit wire를 사용합니다.

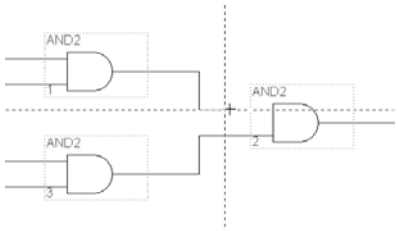


[그림 A-13] chiptrip 예제에서의 연결

Symbol을 연결하는 방법은 [그림 A-14]와 같이 각 Symbol의 데이터 라인이 연결되는 모양을 같고 있는 부분에 마우스를 위치시키면 마우스의 포인터가 십자가 모양으로 바뀝니다. 이 상태에서 마우스의 왼쪽 버튼을 누른 채 드래그 하면 wire가 표시되는 데, [그림 A-15]과 같이 wire를 연결하고자 하는 곳 까지 드래그 하여 연결하면 됩니다.

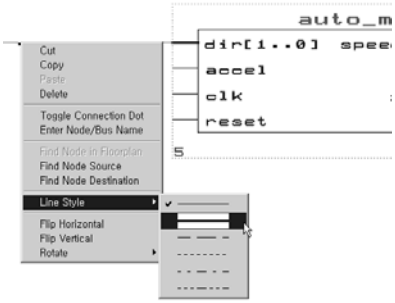


[그림 A-14] 마우스 포인터의 변화



[그림 A-15] wire 연결

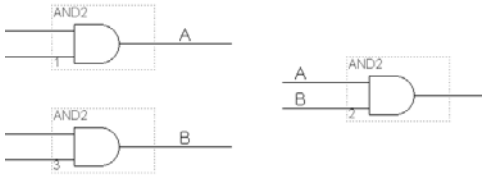
bus wire의 경우에는 위에서처럼 wire을 연결한 후, 마우스 왼쪽 버튼의 더블 클릭으로 그 bus wire 전체를 선택 한 후, 마우스의 오른 쪽 버튼을 눌러 pop-up 메뉴의 line style에서 굵은 선 모양을 선택하면 됩니다.



[그림 A-16] bus wire로 변환

이름으로 연결할 경우에는 연결하고자 하는 Node를 [그림 A-17]와 같이 Symbol에서 wire을 일정 부분 연결한 후 마우스의 왼쪽 버튼을 이용하여 그린 wire를 선택(클릭)하고 Node 명을 적어 주면 됩니다. 버스 데이터의 경우도 bus wire을 그린 후 이름을 버스의 형태로 - 예를 들어, 4bit의 A라는 이름이 있다면 A[3..0] 또는 A[0..3]로.. 이 차이는 MSB(상위 비트)의 번호가 4인지 또는 0인지의 차이이다. - 적어 줍니다.

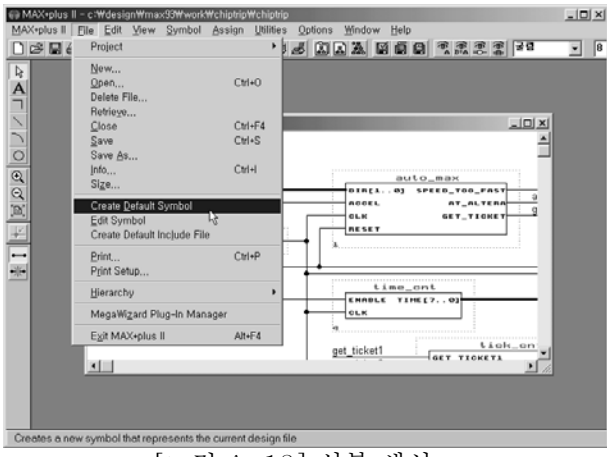
참고, 위에서 A[3..0]의 버스 데이터는 A[3], A[2], A[1], A[0]의 비트 데이터를 묶은 것입니다. 버스 데이터에서 어느 특정 비트 데이터를 연결할 경우에는 A[2]등으로 사용합니다.



[그림 A-17] 이름으로 연결

(5) 심볼 생성

사용자가 설계하여 만든 논리 회로를 하나의 블록으로 하여 더 큰 프로젝트에 넣을 수 있습니다. 이렇게 하기 위해서는 Graphic Editor 상태에서는 Symbol 형태로 불러오게 되는데, 이 Symbol 형태를 만들기 위해서는 Graphic Editor를 활성화 시킨 상태에서 File -> Create Default Symbol 메뉴를 선택하여 변환시키면 됩니다.



[그림 A-18] 심볼 생성

2. Text Editor (HDL에 의한 설계)

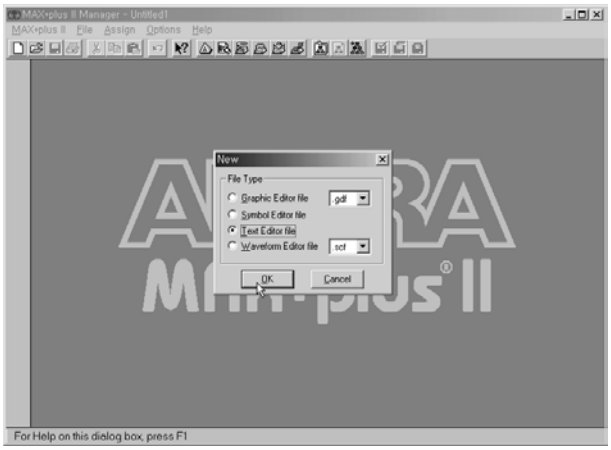
HDL을 이용하여 논리 회로를 설계하는 방법은 각 HDL에 따라 문법을 알아야 하기 때문에, 많은 내용을 설명해야 합니다. 따라서 이 사용자 설명서에서는 가장 기본적으로 Editor 창을 활성화 시키는 방법과 간단한 예제를 보여드리겠습니다. HDL의 문법에 대한 사항은 MAX + plus II에 대한 교재를 참고하시기 바랍니다.

(1) 특징

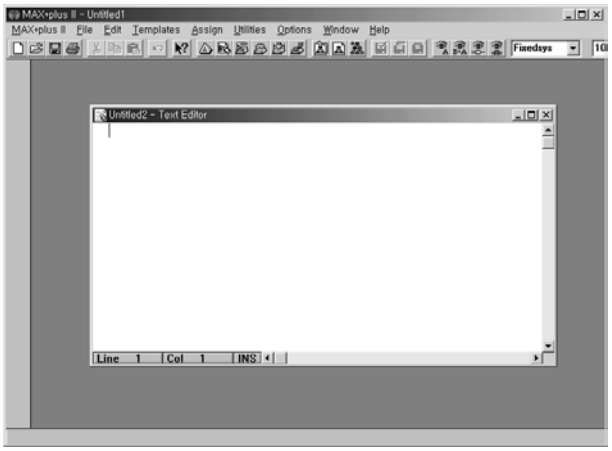
언어를 이용해서 논리 회로를 설계하는 것으로 보통 HDL(Hardware Description Language) 언어를 사용하여 설계합니다. HDL의 종류는 VHDL(Very High-speed integrated circuit HDL), Verilog HDL등이 있는데, MAX + plus II에서는 Text Editor를 이용하여 HDL을 설계합니다.

(2) Text Editor 선택하기

Text Editor를 이용하여 하나의 논리 회로를 설계하기 위해서는 Text Editor 창을 활성화 시켜야 하는데, File -> New 메뉴를 선택한 후 [그림 A-19]과 같은 그림이 활성화 되면 Text Editor file이라는 항목을 선택하여 OK 버튼을 누르면 Text Editor 창이 활성화 됩니다.



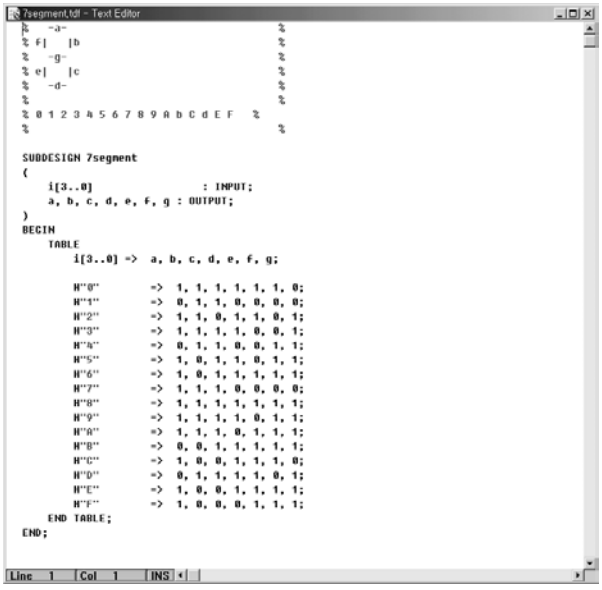
[그림 A-19] New 메뉴에서 Text Editor File 선택



[그림 A-20] Text Editor File을 활성화 한 상태

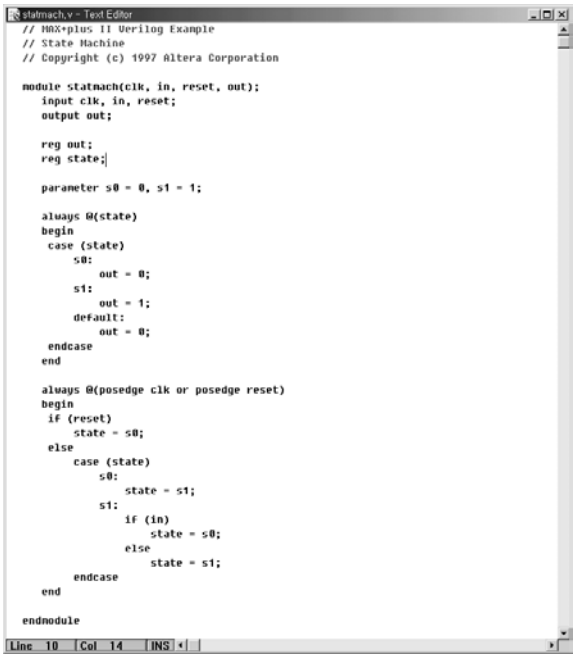
(2) 설계하기

HDL 코드를 사용하여 논리 회로를 설계하기 위해서는 각 HDL (AHDL, VHDL, Verilog HDL등)의 문법등에 대해 알아야 할 것이 많기 때문에 간단히 설계한 예만 보여드리겠습니다. 참고로 각각의 설계파일은 다른 확장자 - VHDL 파일은 .vhd, AHDL 파일은 .tdf, Verilog HDL 파일은 .v - 를 갖는다는 것을 유념해야 합니다.



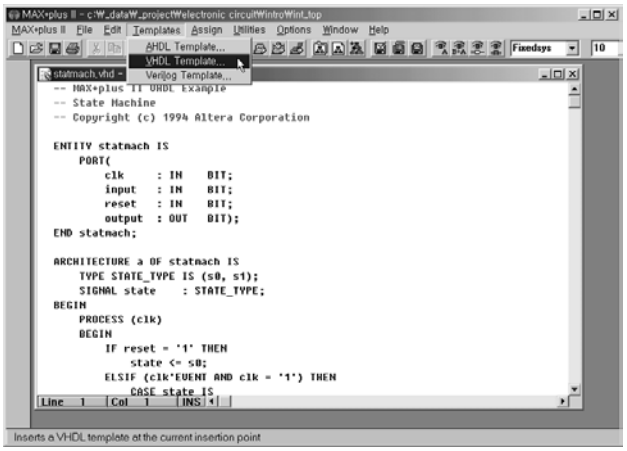
[그림 A-21] AHDL로 설계한 예

[그림 A-22] VHDL로 설계한 예



[그림 A-23] Verilog HDL로 설계한 예

참고. 설계하는 문법은 Text Editor 창을 활성화 시킨 상태에서 [그림 A-24]와 같이 Template 메뉴의 각 언어 - VHDL, Verilog HDL등 - 의 Template 창을 선택하여 설계하는 구문의 문법을 알 수 있습니다.



[그림 A-24] Template 선택

3. Waveform Editor(파형에 의한 설계)

(1) 특징

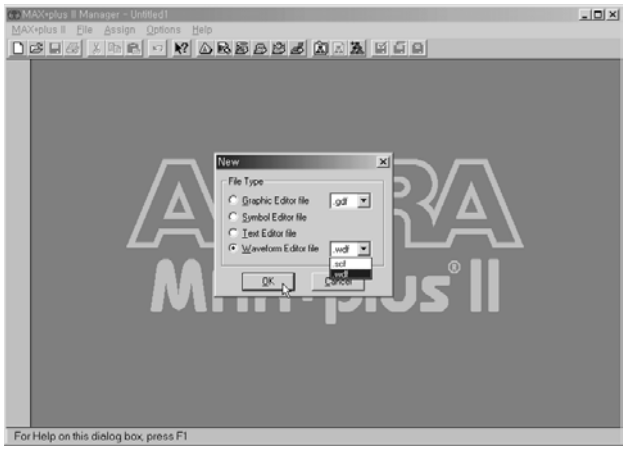
Waveform Editor는 논리 회로를 설계하는 목적과 시뮬레이션 데이터를 만드는 목적으로 사용됩니다.

파형의 상태로 논리 회로를 구성할 때는 입력 조건에 대한 출력 신호 등을 파형의 형태로 설계하여 그것이 하나의 논리 회로로 동작하게 합니다. 장점은 회로를 생각하지 않고, 입력 파형에 대한 출력 파형만 결정하기 때문에 회로 설계가 쉽다는 것이고, 단점은 입력 데이터가 많이 지면 이에 대한 출력 데이터 결과의 경우 수가 많아지기 때문에 복잡한 논리 회로를 설계하는 데에는 어려움이 따른다는 것입니다.

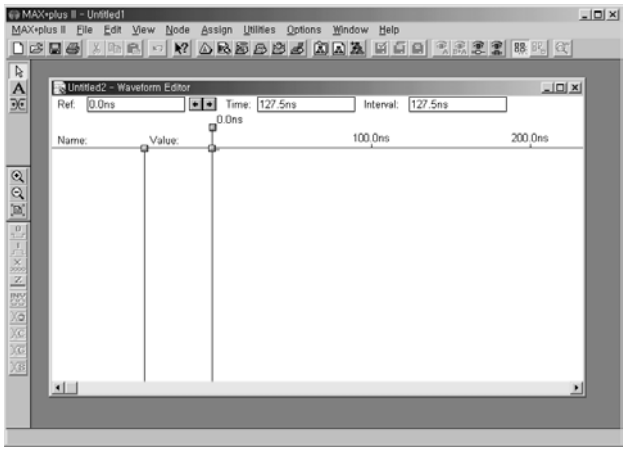
그렇기 때문에 주로 시뮬레이션 파일을 만들 때 사용됩니다. 시뮬레이션 파일에서는 입력 조건만을 설계하여 사용할 수 있기 때문입니다.

(2) Waveform Editor 선택하기

Waveform Editor를 이용해 논리 회로를 설계하기 위해선 File -> New 메뉴에서 Waveform Editor File을 선택하고, 뒤의 확장자를 시뮬레이션 파일을 만들고자 할 때는 .scf로, 논리 회로를 설계할 때에는 .wdf로 선택하여 OK 버튼을 눌러 Waveform Editor 창을 활성화 시킵니다.



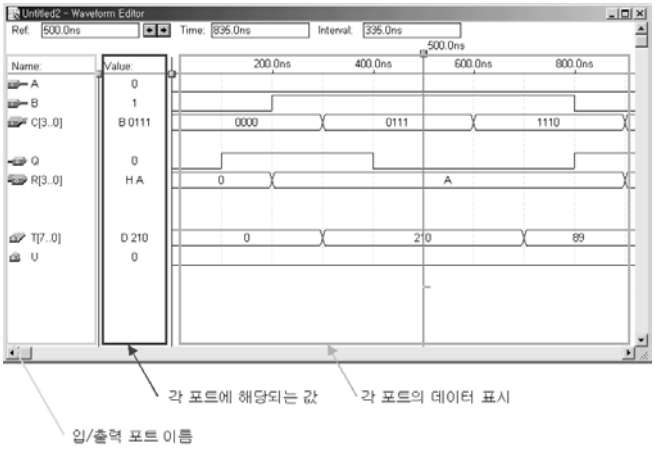
[그림 A-25] Waveform Editor file 선택



[그림 A-26] Waveform Editor File을 활성화한 상태

(3) 구성

아래 [그림 A-27]에서 왼쪽에 선택되어 있는 부분이 입력이나 출력, 변수 등에 해당하는 노드의 이름을 표시하는 부분입니다.  이 입력 노드를 나타내고,  는 출력 노드를 나타냅니다.  는 논리 회로에서 변수형태로 동작하는 노드를 나타내고 이 노드들이 여러 개 모여 있는 형태의 그림  는 각 노드가 버스 형태라는 것을 나타냅니다. Value 부분은 데이터 창에서 포인터라인이 선택된 부분의 값을 보여주는 부분인데, 버스의 경우, B(2진수) H(16진수) D(10진수) O(8진수)의 형태로 나타낼 수 있습니다. 데이터 창 부분은 각 노드들의 데이터를 파형의 형태로 표시하는 부분입니다

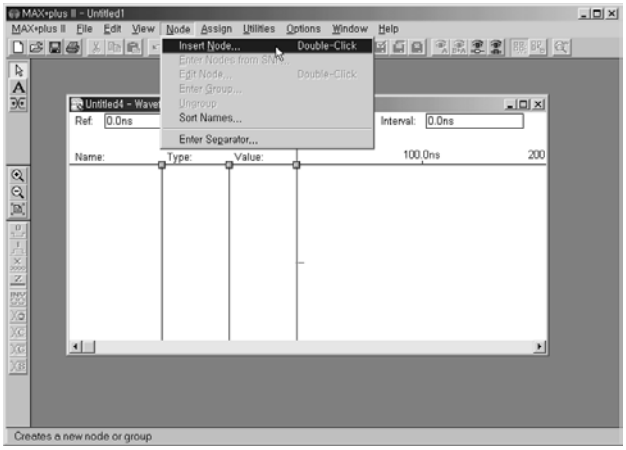


[그림 A-27] Waveform에 대한 설명

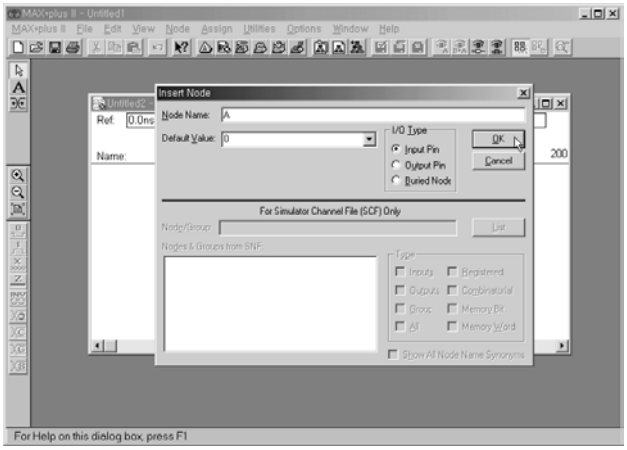
(4) 데이터 수정

① 포트 추가

먼저 [그림 A-28]과 같이 Waveform Editor를 생성 시킨 후 Node -> Insert Node 메뉴를 선택하여 Insert node 창을 활성화 시킵니다.

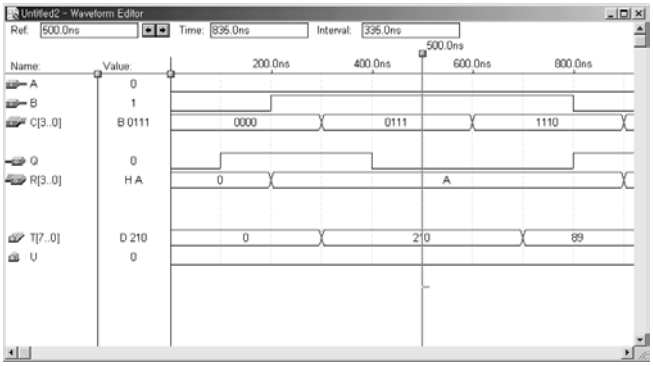


[그림 A-28] insert node



[그림 A-29] Insert node 창

Node Name 항목에 추가하고자 하는 노드의 이름([그림 A-29]에서는 A)을 선택하고 I/O Type에서 해당 노드가 입력 핀인지 출력 핀인지 등을 결정([그림 A-29]에서는 입력 핀으로 설정)하여 OK 버튼을 누르는 것으로 노드를 추가합니다. [그림 A-30]에서 보이듯이 같은 방법으로 다른 노드들도 추가할 수 있습니다.

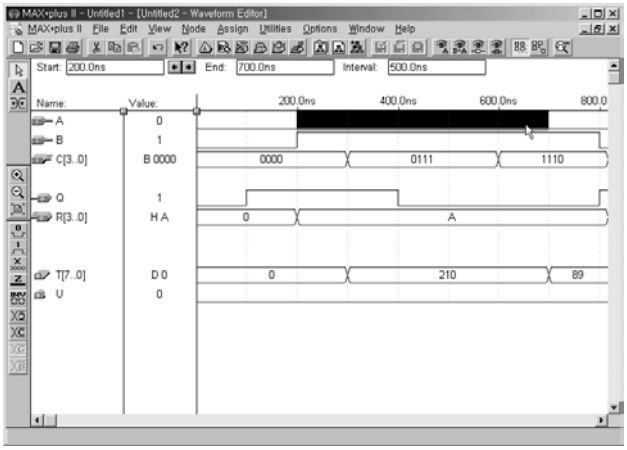


[그림 A-30] 노드 생성

② 데이터 수정

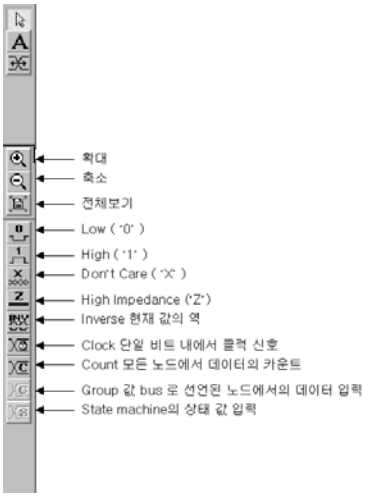
데이터 창의 값을 수정할 때에는 바꾸고자 하는 부분을 마우스의 왼쪽 버튼을 누른 상태에서 드래그하여 선택하고, 프로그램 왼쪽 부분에 있는 아이콘을 눌러 데이터를 수정할 수 있습니다.

다시 설명 드리면 [그림 A-31]과 같이 마우스로 드래그하여 수정하고자 하는 블록을 선택합니다.



[그림 A-31] 수정할 부분 선택

프로그램 왼쪽에 있는 아이콘을 이용하여 수정하려 하는 값을 선택하면 선택한 부분이 그 값으로 바뀝니다.



[그림 A-32] 수정 아이콘

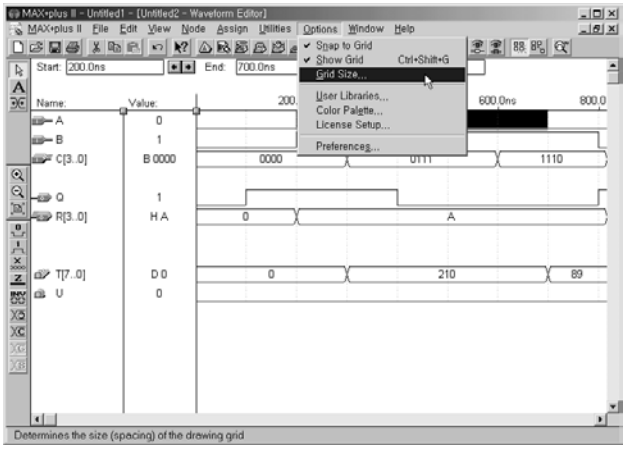
③ 그 외 기능

위에서 설명한 노드 추가 및 데이터의 수정 외에 자주 사용하는 기능에 대해 설명드리겠습니다.

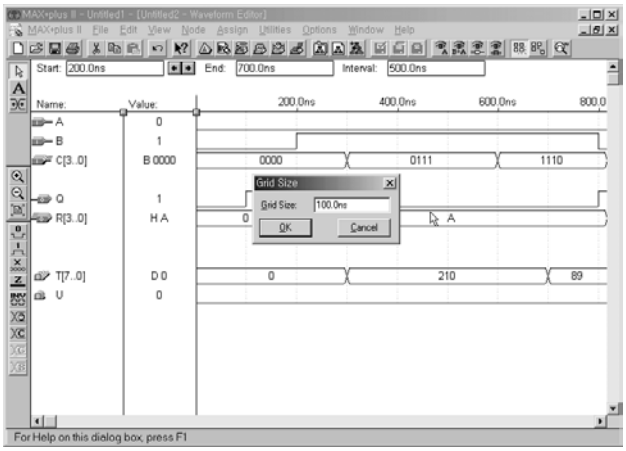
- Grid Size 설정

여기에서 Grid란 데이터가 변하는 최소 단위를 말합니다. 물론 Grid를 무시하고 데이터를 넣을 수 있지만, 이렇게 하였을 경우에는 데이터 출력 결과를 분석하는데 오류가 생길 수 있기 때문에 Grid 단위로 데이터를 수정합니다.

이 Grid Size를 변경하기 위해서는 Waveform Editor 창이 활성화 된 상태에서 Option -> Grid size 항목을 선택하여 바꿀 수 있습니다.



[그림 A-33] Grid Size 항목 선택

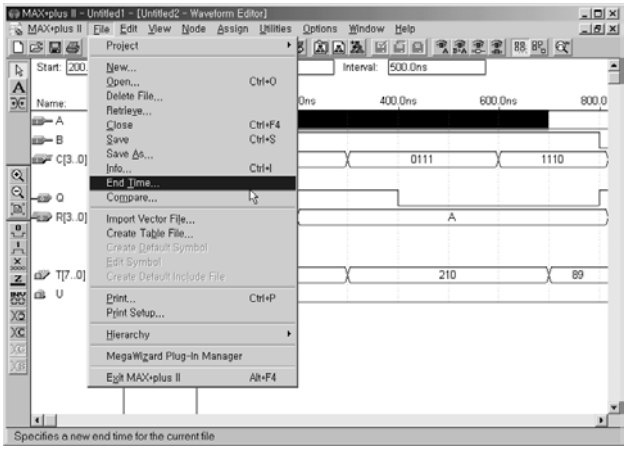


[그림 A-34] Grid Size 설정

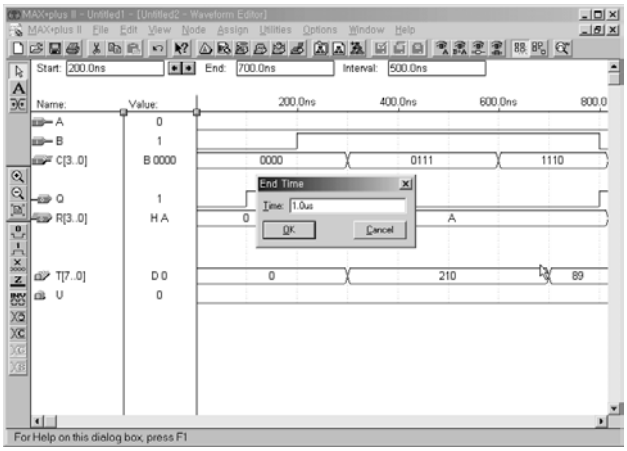
참고로, 초기 값은 100ns로 설정되어 있습니다.

- End Time 설정

시뮬레이션 등을 할 때, 그 시뮬레이션이 끝나는 시간 - 시뮬레이션 하는 길이 - 을 지정해 주는 부분이 End Time입니다. 이것은 Waveform Editor 창을 활성화 한 상태에서 File -> End Time 항목을 선택하여 바꾸어 줄 수 있습니다.



[그림 A-35] End Time 항목 선택



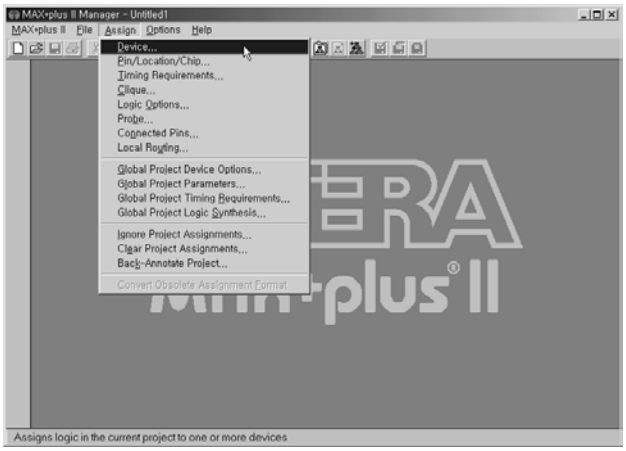
[그림 A-36] End time 설정

A-3. Assign

컴파일에서 설계된 논리 회로를 디바이스에 프로그래밍 할 수 있도록 하드웨어적인 특성을 설정하는데, 이 하드웨어적인 특성인 어느 디바이스를 사용할 것인지, 어떤 핀을 입력 핀으로 사용하는지, 어떤 핀을 출력 핀으로 사용하는지 등에 대한 하드웨어적인 상태를 설정해주는 부분입니다.
일반적으로는 사용하려는 디바이스와 사용하는 핀 번호 등에 대한 설정을 주로 합니다.

1. Device

디바이스는 [그림 A-37]와 같이 Assign -> Device 메뉴를 선택하여 설정합니다.

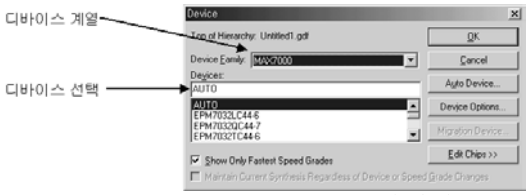


[그림 A-37] Assign -> Device 항목 선택

[그림 A-38]의 디바이스 선택 창에서 선택하려는 디바이스를 선택하여 OK 버튼을 누르면 사용한 설계에 적용하려는 디바이스가 선택합니다.

[그림 A-38]에서 디바이스 계열은 비슷한 특성을 지닌 제품군을 말합니다. 주로 MAX 디바이스 계열과 FLEX 디바이스 계열이 있는데, 두 계열의 차이는 디바이스의 구조가 ROM 방식으로 되어있는지 SRAM 방식으로 되어있는 지에 따릅니다 ROM 방식으로 되어 있다는 것은 디바이스에 프로그래밍 된 논리 회로가 디바이스의 전원 공급을 차단하여도 계속 남아 있어 전원 공급을 다시 해주면 전에 프로그래밍 한 논리 회로의 동작을 한다라고 생각하면 되고, SRAM 방식이라는 것은 전원 공급을 다시 해줄 때 마다 프로그래밍을 새로 해주어야 한다고 생각하면 됩니다.
디바이스를 선택하는 부분은 각 계열이 선택되었을 때마다 그 계열에 포함된 디바이스들이 표시되는데, 위에서 AUTO라고 선택되어 있는 것은 Compile 과정에서

Compiler가 설계한 논리 회로를 포함할 수 있는 디바이스를 자동으로 선택하도록 한다는 것인데, 먼저 논리 회로를 설계하고 그에 따른 디바이스를 선택할 때 유용하게 사용됩니다.



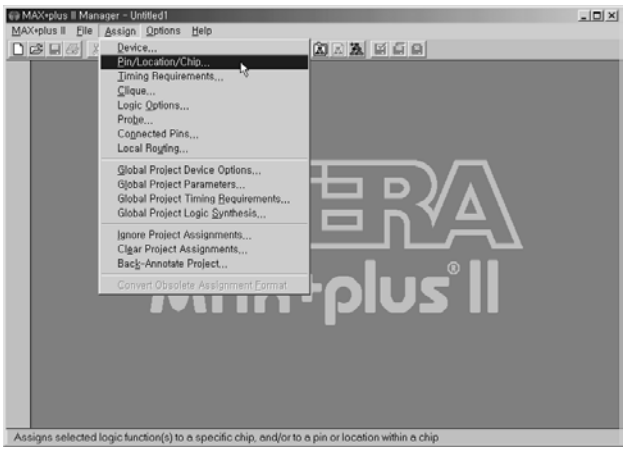
[그림 A-38] Device 창 활성화

참고. Device 창에서 Show Only Fastest Speed Grade 체크 옵션은 선택한 디바이스 계열의 디바이스 중 가장 빠른 속도를 낼 수 있는 디바이스만을 보이는 옵션입니다. 이 부분이 체크 되어 있다면, 체크 된 것을 해지하고 사용하시기 바랍니다.

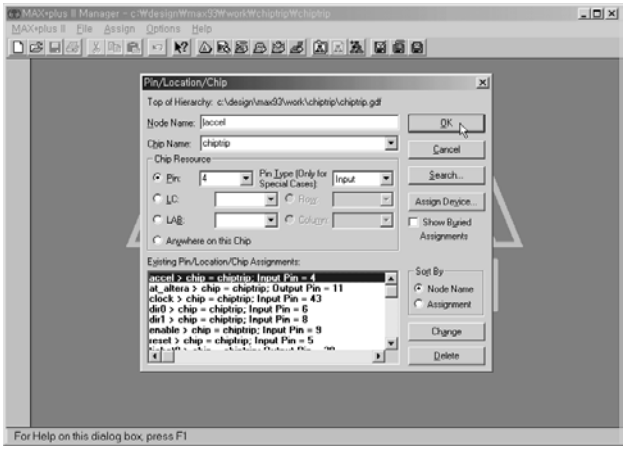
참고. 보드에서 사용하는 디바이스는 FLEX 10K Family의 EPF10K20RC208-4 디바이스입니다.

2. Pin

Pin 번호 설정은 [그림 A-39]와 같이 Assign 메뉴의 Pin/Location/Chip 항목을 선택하여 설정할 수 있습니다.

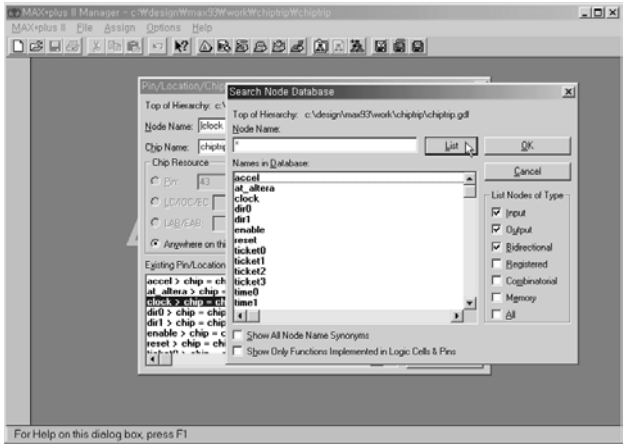


[그림 A-39] Assign 메뉴의 Pin/Location... 항목 선택



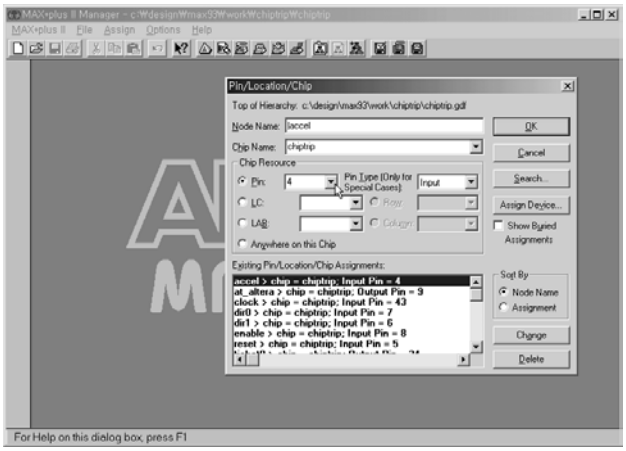
[그림 A-40] Pin/Location/Chip 창 활성화

핀을 설정하려면 [그림 A-40]의 Pin .. 창에서 Search 버튼을 눌러 노드 검색 창을 활성화 시킨 후, 노드 검색 창의 Node Name Edit 창에 "*" 표시를 한 후 List 버튼을 누르면 [그림 A-41]과 같이 설계한 논리 회로에 들어 있는 입/출력 노드의 이름이 전체 검색 되어 List 가 출력됩니다. 이 때 Node Name에 다른 식으로 설정 - 예를 들면 a* (a로 시작하는 모든 이름) 으로 하였을 때 - 하면 그 식에 맞는 노드 이름을 갖는 노드들의 list가 출력됩니다. 노드 검색 창 오른 쪽의 List Nodes of Type 부분에서 Node의 형을 선택하여 검색할 수도 있습니다.



[그림 A-41] 노드 검색 창 활성화

이 노드 리스트 중 한 항목을 선택하여 OK 버튼을 누르면 [그림 A-41]과 같이 노드가 선택된 Pin... 창이 활성화 되는데, Pin 부분에서 해당 핀 번호를 적고 Add 버튼을 눌러 해당 핀의 설정을 합니다. List의 나머지 부분도 위와 동일한 방법으로 하여 설계한 논리 회로의 핀에 대한 설정을 합니다.



[그림 A-41] Pin 번호 설정

참고. 각 Device마다 I/O로 사용할 수 있는 핀 번호가 결정되어 있고, HBE-DTK-INTRO에서는 사용하는 응용 모듈에 대한 핀 번호가 결정되어 있기 때문에 이에 해당하는 번호로 사용하여야 합니다.

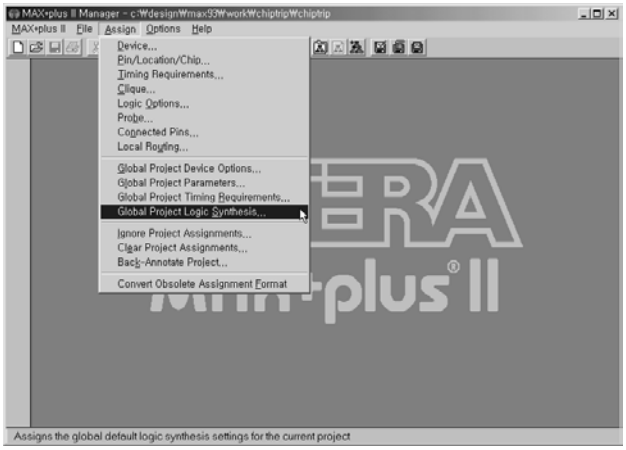
3. 그 외

Assign 메뉴에 해당하는 기능은 많이 있지만 그 중 많이 사용되는 기능에 대해 설명드리겠습니다..

(1) Global Project Logic Synthesis

Logic Synthesis라는 것은 설계한 논리 회로를 하드웨어적으로 어떤 특성을 지니도록 합성하는 것입니다. 그 중 Global Project Logic Synthesis는 Project에 포함된 모든 설계에 대하여 어떤 특성으로 논리 회로를 합성 할 것인지를 설정하는 부분입니다..

아래 [그림 A-42], [그림 A-43]과 같이 Assign 메뉴의 Global Project Logic Synthesis 항목을 선택하여 창을 활성화 합니다.



[그림 A-42] Global Project Logic Synthesis.. 항목 선택

[그림 A-43] Global Project Logic Synthesis 창 활성화

Global Project Synthesis Style

논리 회로의 합성 방식을 Normal, Fast, WYSIWYG 이렇게 세 가지로 나누어 지원합니다.

Normal : 논리 회로는 PLD에 Logic Cell이라는 형태로 프로그램 되는데, 이 때 Logic Cell간의 지연 시간을 최소로 하는 합성 방법입니다.

Fast : 어떤 클럭에 동기되는 논리 회로일 때, 그 동기되는 클럭이 최대의 빠르기를 줄 수 있도록 하는 합성 방법입니다.

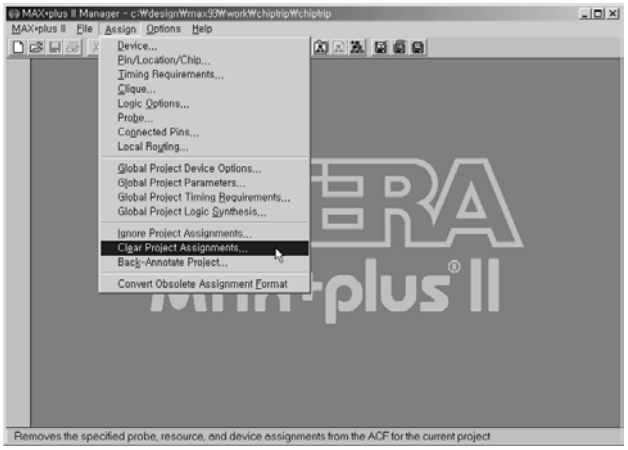
WYSIWYG (What You See Is What You Get) : MAX + plus II 프로그램의 OEM 툴에서 논리 회로에 대한 컴파일 및 합성을 하였을 때, EDIF 형식으로 MAX + plus II에 불러올 수 있는데, OEM 툴에서 사용한 합성 옵션 그대로 MAX + plus II에서 사용하기 위한 방법입니다.

Optimize

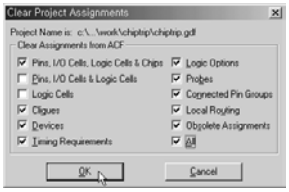
논리 회로를 합성할 때, 바(bar)가 Area 쪽에 가까울 수록 디바이스에서 차지하는 공간이 작아지고, 바(bar)가 Speed 쪽에 가까울 수록 동작 스피드가 빨라집니다. 공간과 스피드는 서로 상대적입니다.

(2) Clear Project Assignment

지금까지 설명한 Pin과 디바이스, 합성 옵션 등을 설정한 것을, 초기화(Pin 등의 설정을 지우는 것) 시키고자 할 때 사용합니다. [그림 A-44]가 Clear Project Assignment 창인데, 이곳에서 초기화할 사항을 check box에 클릭하여 선택한 후 OK 버튼을 누르면 설정된 부분이 초기화 됩니다.



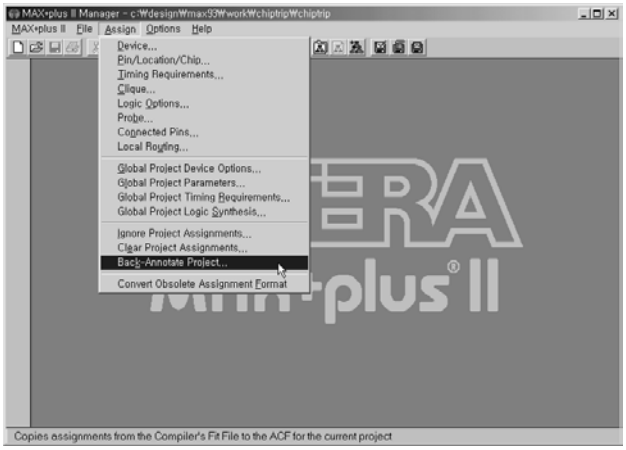
[그림 A-44] Clear Project Assignment...



[그림 A-45] 창 활성화

(3) Back-Annotate Project

디바이스나 핀 등에 대한 하드웨어적인 설정 없이 compile하면 compiler가 자동으로 핀 등에 대한 배치를 해 주는데, 이 배치를 사용자가 변경할 수 있도록 Compiler가 설정한 핀 번호로 고정하는 것입니다. 이 작업 후 (프로젝트 명).acf 파일을 이용하여 하드웨어의 설정을 바꿀 수 있습니다.



[그림 A-46] Back Annotate Project...

[그림 A-47] Back.... 활성화 창

A-4. Compile

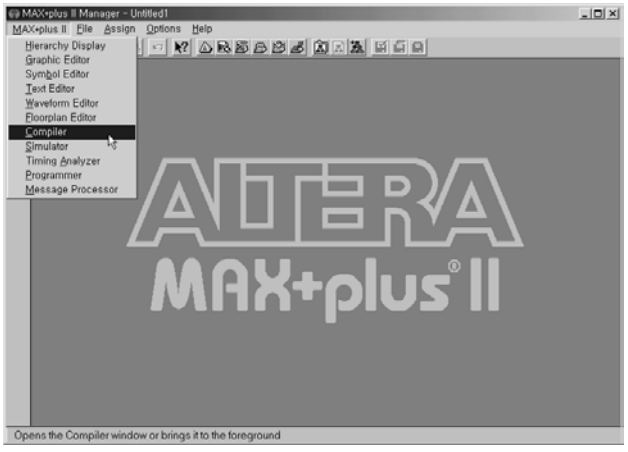
1. 역할

컴파일러는 프로젝트에 포함된 모든 디자인 파일을 통합하는 과정과 문법 오류와 일반적인 설계의 오류를 찾는 과정, 논리 회로 합성 및 설계한 논리 회로가 하드웨어에서 이루어지는 가장 기본 단위인 로직 셀로 나누고, 각 셀의 위치를 결정하는 과정, 시뮬레이션과 시간 측정을 할 수 있도록 하는 파일을 생성하고, 타겟 디바이스에 프로그래밍 할 수 있는 파일을 만드는 과정을 진행합니다.

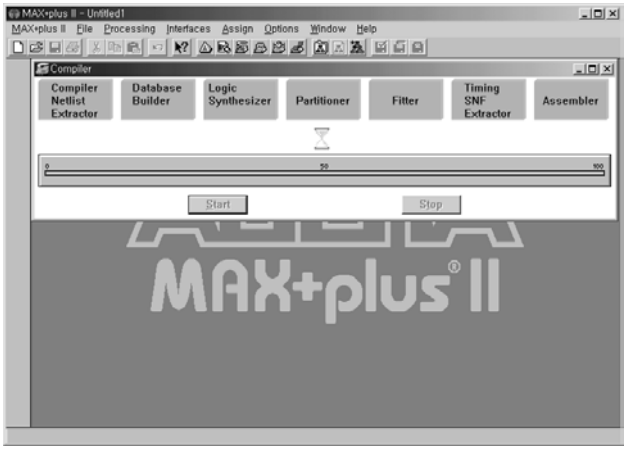
2. Compiler

컴파일러는 두 가지 방법이 있는데, 기능적인 컴파일러(Functional Compiler)와 시간 특성을 지닌 컴파일러(Timing Compiler)입니다. 간단하게 말하자면 기능적인 컴파일러는 프로젝트에 포함되어 있는 설계에 대한 기능적인 검증, 즉 문법적으로 옳은지에 대한 여부를 체크하고, 기능적으로 설계한 프로젝트의 특성만 파악할 수 있는 시뮬레이션이 가능토록 해 줍니다. 이에 비해 타이밍 컴파일러(시간 특성을 지닌 컴파일러 : Timing Compiler)는 사용할 디바이스가 선택되었을 때, 프로젝트에 포함되어 있는 설계한 부분을 그 디바이스의 특성 등에 맞추어 그 특성에 따른 시뮬레이션과 지연시간 등의 측정할 수 있는 파일을 생성하며, 또, 그 디바이스에 프로그램할 수 있는 프로그래밍 파일을 만드는 과정을 합니다.

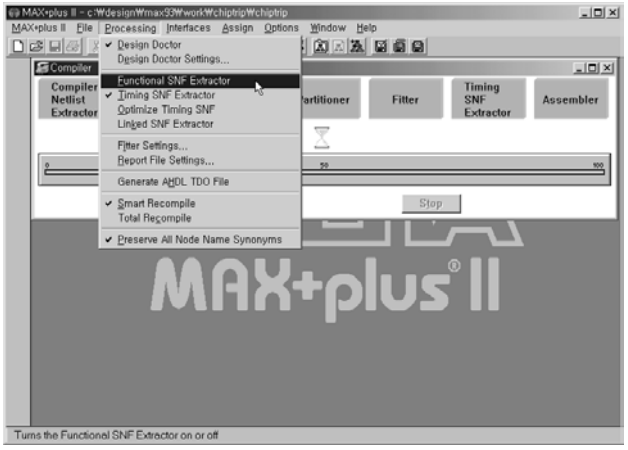
[그림 A-48]과 같이 MAX + plus II 메뉴에서 Compiler 항목을 선택하면 [그림 A-49]와 같이 Timing Compiler가 활성화 되는데, 이 상태에서 [그림 A-50]와 같이 Processing 메뉴의 Functional SNF Extractor 항목을 선택하면 [그림 A-51]과 같이 Compiler가 Functional Compiler로 바뀝니다. 다시 Timing Compiler의 상태로 돌리기 위해서는 Functional SNF Extractor 항목을 다시 선택하면 됩니다.



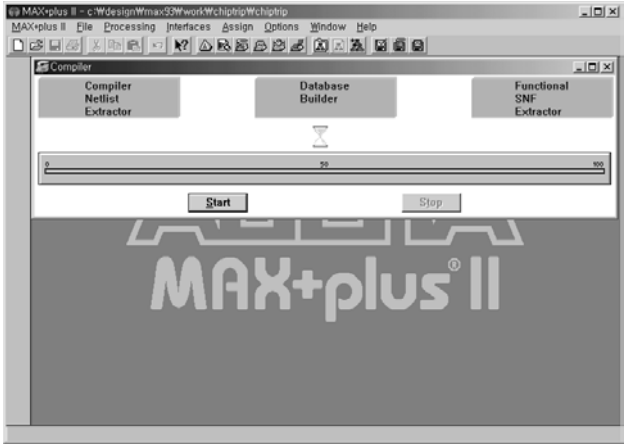
[그림 A-48] MAX + plus II -> Compiler 항목 선택



[그림 A-49] Compiler 창 활성화



[그림 A-50] Processing -> Functional SNF Extractor 항목 선택



[그림 A-51] Functional Compiler 활성화

3. 각 Compiler의 수행 과정

(1) Functional Compiler

[그림 A-51]을 통해 알 수 있듯이 Functional Compiler에서는 세 가지 과정을 수행하는데, Compiler Netlist Extractor에서 문법의 오류를 체크하고 Netlist 파일을 생성합니다. 다음의 Database Builder에서는 Node 이름에 따른 데이터 베이스를 구축하고, 마지막으로 Functional SNF Extractor에서 기능적으로 검증할 수 있도록 시뮬레이션 할 수 있도록 하는 파일을 만듭니다.

(2) Timing Compiler

[그림 A-49]을 통해 알 수 있듯이 Functional Compiler의 과정보다는 상당히 많은 과정이 있는데, Compiler Netlist Extractor와 Database Builder 과정에서는 Functional Compiler와 마찬가지로 문법의 오류를 찾고 Netlist 데이터 베이스를 구축합니다. Logic Synthesizer에서 설계한 회로의 합성과 논리 회로를 최소화시키는 과정을 진행합니다. Partitioner와 Fitter에서는 컴파일 할 때의 주변 옵션등에 대한 사항 및 디바이스가 허용하는 한계치에 비해 어느 정도 사용했는가를 나타내는 사용 등을 나타내는 report 파일과 설계된 파일을 디바이스에 구현하기 위한 가장 작은 단위인 Logic Cell로 나누고, 나누어진 Logic Cell을 디바이스의 어느 위치에 놓을 것인지에 대한 연산을 하는 과정을 진행합니다. Timing SNF Extractor에서는 디바이스의 시간 특성을 고려한 시뮬레이션과 시간 특성에 대한 측정 등으로 활용할 수 있는 .snf 파일을 만듭니다. 마지막으로 Assembler에서 설계한 논리 회로를 디바이스에 프로그래밍 하기 위한 프로그래밍 파일을 만드는 과정을 수행합니다.

A-5. Simulation

1. 역할

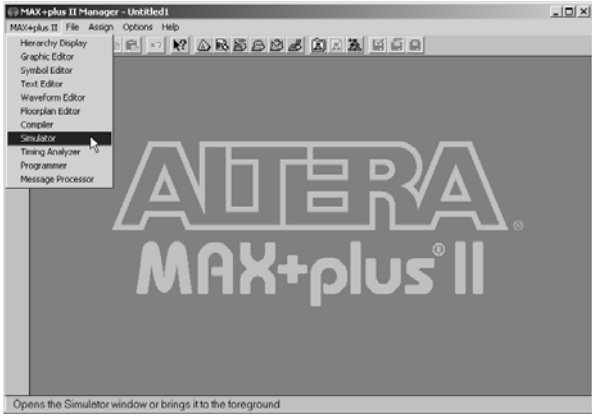
설계한 논리 회로를 하드웨어를 이용해서 검증하는 경우에, 처리하는 데이터량이 많거나 속도가 빠를 경우에는 검증하는데 시간이 많이 걸리고 그 원인을 파악하지 못하는 경우가 있기 때문에 소프트웨어 적으로 먼저 검증하는 것이 필요합니다. MAX + plus II에서는 이를 위해 파형으로 검증하는 시뮬레이션을 제공하는데, 이 시뮬레이션에서 동작하는데 필요한 입력 파형의 조건을 설정해 주면 그에 따른 출력 결과의 파형이 출력되어 쉽게 확인해 볼 수 있습니다.

2. Simulator

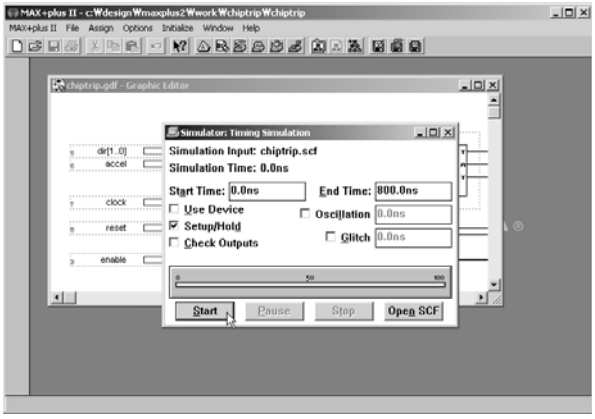
컴파일러와 마찬가지로 시뮬레이터를 이용한 시뮬레이션에도 두 가지 방법이 있습니다. 기능적인 시뮬레이션(Functional Simulation)과 시간 특성을 지닌 시뮬레이션(Timing Simulation)입니다. 기능적인 시뮬레이션은 논리적으로 설계한 논리 회로가 설계한 의도대로 동작하는지를 파형으로 검증하는 부분입니다. 기능적인 시뮬레이션을 하기 위해서는 먼저 기능적인 컴파일을 해야 합니다. 위의 기능적인 시뮬레이션이 단지 설계한 회로를 논리적으로만 검증 한다면, 타이밍 시뮬레이션(시간 특성을 지닌 시뮬레이션:Timing Simulation)은 설계자가 결정한 하드웨어의 상태 즉, 사용하려고 하는 디바이스와 핀 번호 등을 고려하여 파형으로 검증하는 것입니다. 타이밍 시뮬레이션을 하기 위해서는 먼저 타이밍 컴파일을 해야 합니다.

먼저 기능적 시뮬레이션을 통해서 논리적으로 동작이 되는지를 검증한 다음, 타이밍 시뮬레이션으로 목적으로 하는 디바이스의 시간적 특성을 적용한 검증을 하는 것이 정확하게 검증하는 방법입니다. 앞에서도 설명하였듯이, 기능적인 시뮬레이션만으로 검증하였을 때는 사용하는 디바이스에 따라서 동작이 다르게 될 수 있고, 타이밍 시뮬레이션만으로 검증하였을 때는 논리 회로를 조금 변경하거나 다른 디바이스로 목적 디바이스를 바꾸었을 때는 동작이 안될 수 있기 때문입니다.

[그림 A-52]과 같이 MAX + plus II 메뉴에서 Simulator 항목을 선택하면 [그림 A-53]과 같이 Simulator이 활성화 되는데, 왼쪽 아래 부분의 Start 버튼을 누르면 시뮬레이션을 하게 됩니다. 이 시뮬레이션을 하기 위해서는 먼저 시뮬레이션 파일을 만들어야 합니다.

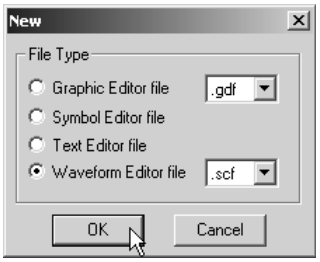


[그림 A-52] Simulator 메뉴 선택



[그림 A-53] Simulator 창 활성화

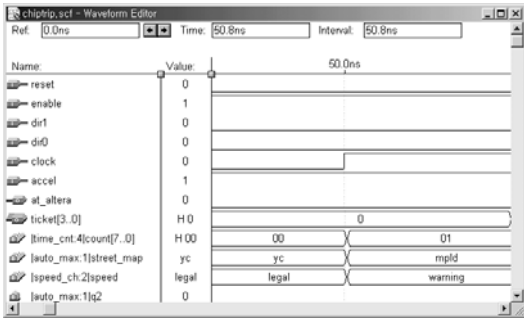
시뮬레이션 파일은 앞의 Waveform Editor에서 파형에 의한 논리 회로를 설계하는 방법과 거의 비슷합니다. 다만, 처음에 파일을 생성할 때 [그림 A-54]와 같이 확장자를 .scf로 설정해야 하고, 입력에 대한 조건 만을 설정한다는 부분에서 차이가 있습니다. 물론 저장할 때에도 .scf 파일로 저장합니다.



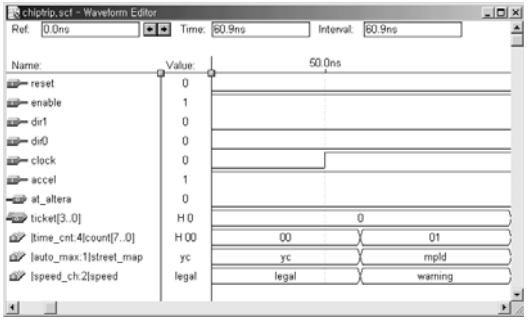
[그림 A-54] New 창에서 시뮬레이션 파일 선택

Function Simulation과 Timing Simulation하였을 때의 출력되는 결과의 차이에 대해 [그림 A-55]와 [그림 A-56]을 이용하여 확인해 보겠습니다. [그림 A-55]와 [그림 A-56]은 같은 논리 회로를 각각 Function Compile과 Timing Compile하여 시뮬레이션 한 결과입니다.

Functional Simulation한 결과인 [그림 A-55]에서는 clock의 입력이 있을 때 바로 데이터가 전달되는데 비해, Timing Simulation한 결과인 [그림 A-56]에서는 일정 시간이 지연된 후에 데이터가 전달되는 것을 볼 수 있습니다.



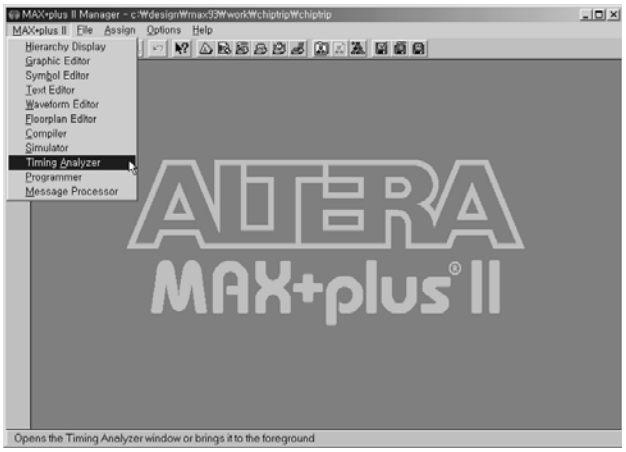
[그림 A-55] Functional Simulation 예



[그림 A-56] Timing Simulation 예

A-5. Timing Analyzer

시뮬레이션에서 파형에 의해 설계된 논리 회로를 검증할 수 있지만, 지연 시간 등을 수치적으로 정확하게 보여주는 부분이 Timing Analyzer입니다. Timing Analyzer는 MAX + plus II 메뉴의 Timing Analyzer 항목을 선택하여 활성화 시킵니다.



[그림 A-57] Timing Analyzer 항목 선택

[그림 A-58] Timing Analyzer 창 활성화

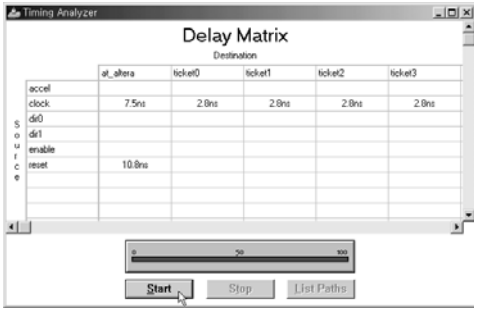
1. Delay Matrix

입력노드의 신호에 대한 출력 노드의 지연 시간을 측정해 주는 부분입니다.

이 항목은 Timing Analyzer 창을 활성화 한 상태에서 Analysis 메뉴의 Delay Matrix 항목을 선택하여 활성화 시키며, Start 버튼을 눌러 지연시간을 측정합니다.

[그림 A-60]에서 왼쪽의 세로줄이 입력 노드이며, 위쪽의 노드들이 출력 노드를 나타냅니다.

[그림 A-59] Delay Matrix 선택

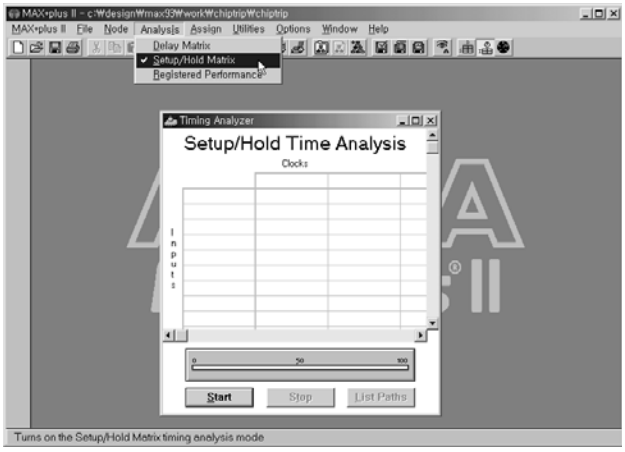


[그림 A-60] Delay Matrix 실행

2. Setup/Hold time Matrix

설계한 논리 회로에 플립-플롭이 사용되었을 때 이 플립 플롭의 Setup time과 Hold Time을 측정해 주는 부분입니다.

이 항목은 Timing Analyzer 창을 활성화 한 상태에서 Analysis 메뉴의 Setup/Hold Matrix 항목을 선택하여 활성화 시키며, Start 버튼을 눌러 Setup / Hold Time을 측정합니다.



[그림 A-61] Setup/ Hold Matrix 선택

[그림 A-62] Setup / Hold Time Analysis 실행

참고, Setup Time과 Hold Time이란?

플립 플롭은 어떤 신호를 클럭에 동기 시켜 내보내 주는 레지스터의 역할을 합니다. 이 때, 플립플롭이 값을 읽어 들이기 위해 필요한 최소의 시간이 Setup Time이고, 플립 플롭에서 값이 출력되는데 걸리는 시간이 Hold Time입니다.

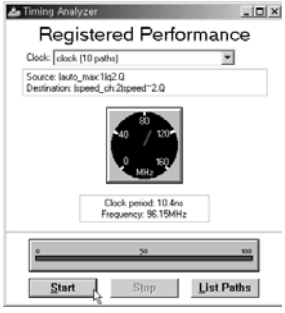
3. Registered Performance

설계한 논리 회로가 어떤 클럭 등에 의해 동기되어 움직일 경우, 그 동기되는 클럭의 최대 사용 주파수를 측정해 주는 부분입니다.

이 항목은 Timing Analyzer 창을 활성화 한 상태에서 Analysis 메뉴의 Registered Performance 항목을 선택하여 활성화 시키며, Start 버튼을 눌러 Performance를 측정합니다.

[그림 A-64]에서 clock에 대한 최대 동작 주파수가 96.15MHz인 것을 확인 할 수 있습니다.

[그림 A-63] Registered Performance 항목 선택



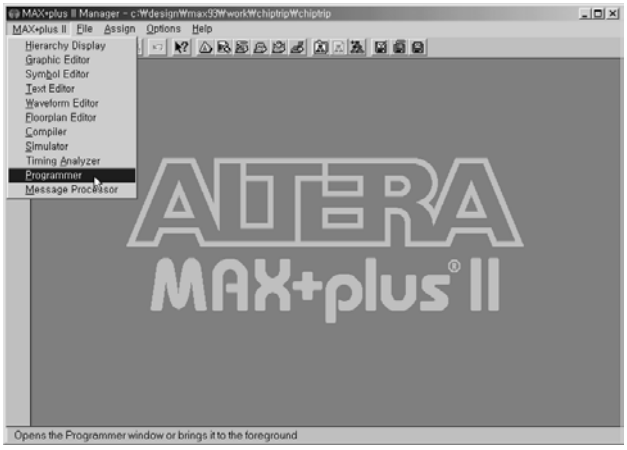
[그림 A-64] Registered Performance 실행

A-6. Programming

위에서 설계 및 검증이 끝난 논리 회로를 디바이스에 프로그래밍 부분입니다.

1. Programmer 활성화

[그림 A-65]과 같이 MAX + plus II 메뉴에서 Programmer 항목을 선택하여 Programmer 창을 활성화 시켜 사용합니다.

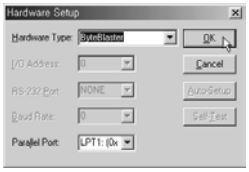


[그림 A-65] Max + plus II 메뉴에서 Programmer 항목 선택

[그림 A-66] Programmer 창 활성화

2. 하드웨어 설치

처음에 Programmer 창을 활성화 하면 Hardware Setup 창이 나온다. ALTERA 디바이스를 프로그램하는 장치를 선택하는 부분인데, 이 장비에서는 ByteBlaster라는 장치를 이용하기 때문에 [그림 A-67]와 같이 ByteBlaster(또는 ByteBlaster MV)를 선택하고 OK 버튼을 누릅니다.



[그림 A-67] Hardware Setup

참고. ALTERA 사에서 지원하는 프로그래밍 장치 목록

PLP6 + MPU + Adapter

PLP6는 PC의 ISA 슬롯에 꽂는 카드입니다. 그리고, MPU는 PLD에 프로그래밍 하기 위한 장치입니다. Adapter는 프로그램 하고자 하는 PLD의 종류 등에 따라 모델이 다른데, MPU에 이 Adapter를 연결하고, PLP6와 MPU를 연결하여 사용합니다.

BitBlaster

BitBlaster는 Serial Port를 이용한 프로그래밍 장치입니다.

ByteBlasterMV

3.3V와 5V를 지원하는 Parallel Port를 이용한 프로그래밍 장치입니다.

MasterBlaster

Serial/USB Port를 이용한 프로그래밍 장치입니다.

3. 프로그래밍

하나의 프로젝트가 선언되고 컴파일을 통해 프로그래밍 파일이 만들어 졌을 때, Programmer 창을 활성화 하면 Configure 버튼이 활성화 됩니다. 보드의 전원을 키고, 다운로드 케이블을 PC의 Parallel Port와 보드의 다운로드 port에 연결한 후 Configure 버튼을 누르면 프로그래밍이 됩니다.

만약에 활성화 되지 않았다면, 위의 하드웨어 설치 부분을 다시 한번 확인 바랍니다. 사용하는 컴퓨터의 OS가 Windows NT 또는 Windows 2000일 때에는 ByteBlaster를 드라이버를 통해 설정해야 합니다. 설정하는 방법은 저희 회사 홈페이지 (www.hanback.co.kr)의 자료실을 참고하시기 바랍니다.

참고.

Programmer에서 Program과 Configure 버튼이 활성화 되는 것은 어느 디바이스를 목적으로 하고 컴파일 했는가에 따라서 달라집니다. MAX 계열의 디바이스를 목적으로 하는 경우에는 Program 버튼이, FLEX 계열의 디바이스를 목적으로 하는 경우에는 Configure 버튼이 활성화 되고, 프로그래밍 파일로는 각각 확장자가 .pof와 .sof인 파일이 선택됩니다.