VHDL Simulation과 Synthesis의 기초 실습

- Synopsys Tool을 중심으로 -

인천대학교 공과대학 전자공학과

System ASIC Design Lab.

지도교수 : 조 중 휘

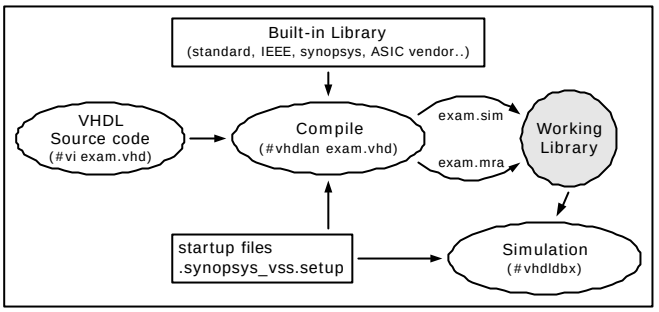
jcho@asiclab.inchon.ac.kr

기초 실습 순서 및 개요

- 기초 실습-I
  - 실습-1 : 1-Bit Full Adder Simulation
    - Synopsys Simulation 환경 설정과 Simulation Tool 이해
  - 실습-2 : 1-Bit Full Adder Synthesis
    - Synopsys Synthesis 환경 설정과 Synthesis Tool 이해
  - 실습-3 : 1-Bit Serial Adder 설계
    - Synopsys Tool에 대한 종합적인 이해
  - 실습-4 : 3-tap FIR Filter 설계
    - 조합논리회로 및 순서논리회로 설계에 대한 이해
  - 실습-5 : Counter와 7-Segment Display 회로에 대한 설계
    - 계층적 설계에 대한 이해
- 기초 실습-II
  - 실습-6 : Traffic Light Controller 설계
    - Digital System에 대한 RTL 회로도 분석 능력
    - VHDL 표현력 검증
    - Synopsys Tool에 대한 이해와 사용
    - Max+PlusII 동작에 대한 이해와 사용
    - FPGA 구현을 통한 종합적인 이해와 응용력 향상
- 질의 및 응답

Simulation을 위한 기본적인 구성요소

- Synopsys VSS Simulator를 위한 기본 구성



- VHDL compile : $ vhdlan {vhdl_source_file_name}
- Built-in Library
  - standard, IEEE, synopsys, ASIC vendor, your own...
- Working Library
  - 설계된 VHDL code를 compile 하여 생성된 데이터(*.sim, *.mra)를 저장하는 공간
- Simulation : $ vhdldbx
- 초기 setup file
  - .synopsys_vss.setup

VSS Simulator를 위한 환경 설정(1)

- .synopsys_vss.setup file의 설정에 대한 기본 모형
  - **Working_Library_Name과 Working_Library_Directory**는 User가 임의로 정할 수 있음
  - **Working_Library_Name 의미** : Compile시 저장되는 장소로써 Symbolic Name
  - **Working_Library_Directory 의미** : Working Library의 Directory Path

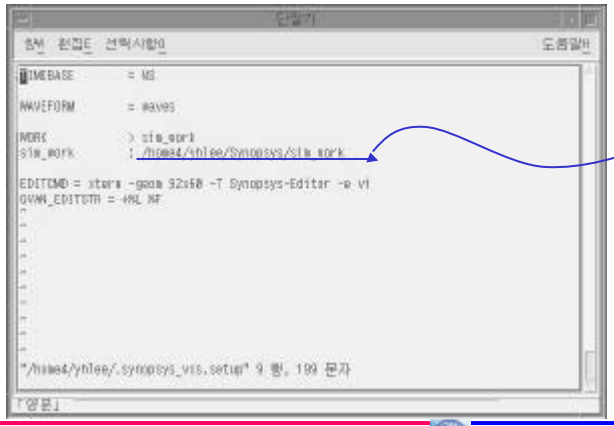
```
TIMEBASE                = NS
WAVEFORM                = waves
WORK                    > Working_Library_Name
Working_Library_Name    : Working_Library_Directory
EDITCMD = xterm -geom 92x60 -T Synopsys-Editor -e vi
```

- 본 실습에서는 다음과 같이 구성한다.
  - 각 User의 Home Directory에 sim_work directory를 Unix 명령어를 사용하여 생성한다.
    $ mkdir ~/sim_work
  - **.synopsys_vss.setup file을 다음과 같이 설정한다.**

```
TIMEBASE                = NS
WAVEFORM                = waves
WORK                    > sim_work
sim_work                : ~/sim_work
EDITCMD = xterm -geom 92x60 -T Synopsys-Editor -e vi
```

VSS Simulator를 위한 환경 설정 (2)

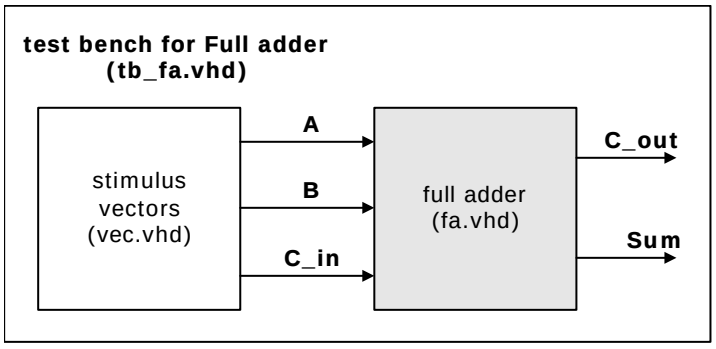
- UNIX 환경에서 사용하는 vi editor를 이용한 ~/.synopsys_vss.setup file 열기 수행
- 필요로 하는 최소한의 정보에 대한 입력
  - Simulation 수행을 위한 Default 시간 단위에 대한 설정 : **TIMEBASE**
  - Simulation 결과에 대한 Default 표현 방식에 대한 정의 : **WAVEFORM**
  - Compile의 과정에서 생성되는 여러 정보들의 기억 공간에 대한 설정 : **WORK**
- 본 설계에서 사용할 예제는 다음과 같다.



자신의 계정에 만든다.

실습-1 : Full Adder를 위한 전체 Block 구조

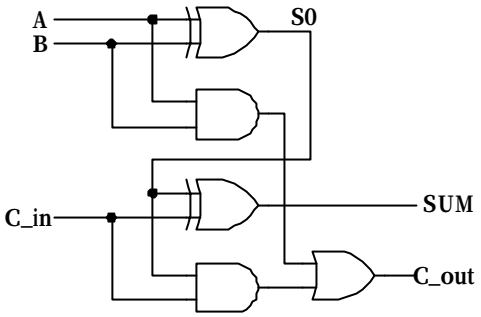
- 설계 사양에 대한 이해와 분석
- 설계 사양에 대한 분석과 설계 대상체에 대한 VHDL 표현
- Simulation을 위한 적절한 Stimulus File 구성과 Test Bench 표현
- VHDL Simulation을 통한 검증



실습-1 : Full Adder에 대한 Data flow 형태의 VHDL 설계

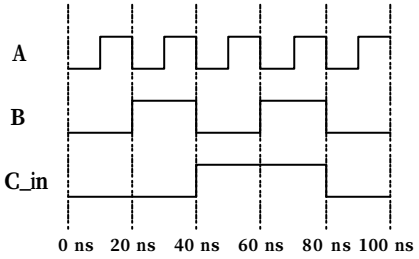
✍ Data Flow 형태의 구현

```
library IEEE;
use IEEE.Std_logic_1164.all;
entity fa is
  port (
        A      : in std_logic;
        B      : in std_logic;
        C_in   : in std_logic;
        C_out: out std_logic;
        Sum   : out std_logic);
end fa;
architecture rtl of fa is
  signal S0 : std_logic;
begin
        S0 <= A xor B;
        Sum <= S0 xor C_in;
        C_out <= (A and B) or (S0 and C_in);
end rtl;
```



실습-1 : Stimulus Vector의 예시와 이에 대한 VHDL 설계

✍ 실습 1을 위해 사용자가 설정한 파형



✍ Stimulus Vector 생성부 설계

```
library IEEE;
use IEEE.Std_logic_1164.all;
entity vec is
    port (A, B, C_in  : out std_logic);
end vec;
--
architecture rtl of vec is
begin
process
begin
        A <= '0'; B <= '0'; C_in <= '0';
        wait for 10 ns;
        A <= '1'; B <= '0'; C_in <= '0';
        wait for 10 ns;
        A <= '0'; B <= '1'; C_in <= '0';
        wait for 10 ns;
        A <= '1'; B <= '1'; C_in <= '0';
        wait for 10 ns;
        A <= '0'; B <= '0'; C_in <= '1';
        wait for 10 ns;
        A <= '1'; B <= '0'; C_in <= '1';
        wait for 10 ns;
        A <= '0'; B <= '1'; C_in <= '1';
        wait for 10 ns;
        A <= '1'; B <= '1'; C_in <= '1';
        wait for 10 ns;
end process;
end rtl;
```

실습-1 : Full Adder를 위한 Test Bench VHDL 설계

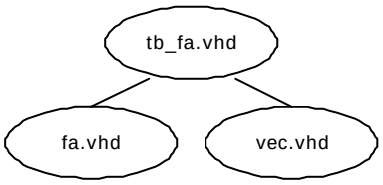
✍ Test Bench를 위한 VHDL 설계

```
library IEEE;use IEEE.std_logic_1164.all;
entity tb_fa is
end tb_fa;
--
architecture rtl of tb_fa is
  component fa
     port (
            A,B,C_in  : in std_logic;
            C_out,Sum: out std_logic);
  end component;
  component vec
     port (
            A,B,C_in  : out std_logic);
  end component;

  -- 내부 Signal Assign
  signal A, B, C_in    : std_logic;
  signal C_out, Sum : std_logic;
  for U1 :  fa use entity work.fa(rtl);
  for U2 :  vec  use entity work.vec (rtl);
begin

  -- U1, U2는 Component Label
  -- 복잡한 계층 구조를 갖는 설계를 검증할 때
  -- 검증하고자 하는 하위 계층을 쉽게 알아볼 수
  -- 있도록 이름을 부여한다.

  U1 : fa
     port map (
            A=> A,B=> B,C_in   => C_in,
            C_out=> C_out,Sum=> Sum);
  U2 : vec
     port map (
            A=> A,B=> B,C_in=> C_in);
end rtl;
```


실습-1 : VHDL Code에 대한 Compile의 수행

✍ Code에 대한 Compile의 수행 순서

✍ Design Tree에서 하위 계층의 VHDL Code부터 Compile을 수행한다.



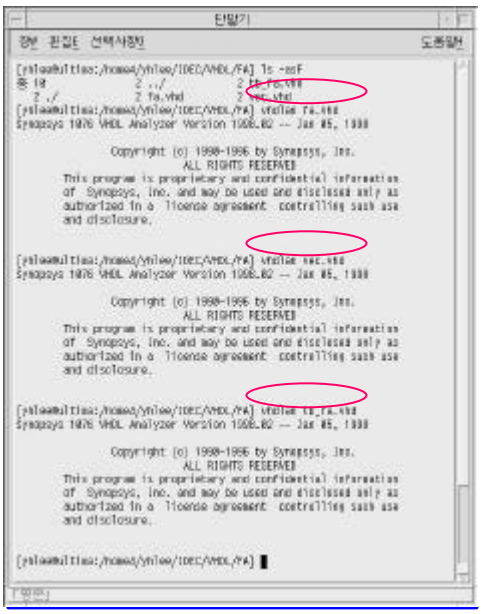
✍ 본 예제에서는 Unix Command에서 다음을 수행한다.

$ **vhdlan fa.vhd**

$ **vhdlan vec.vhd**

$ **vhdlan tb_fa.vhd**

✍ Compile 수행 과정을 표현하는 Window에 대한 예시는 다음과 같다.



실습-1 : Compile의 수행에 따라 생성되는 File 결과

- Compile을 Error 없이 수행한 경우 ~/sim_work Directory에 생성되는 여러 정보의 예

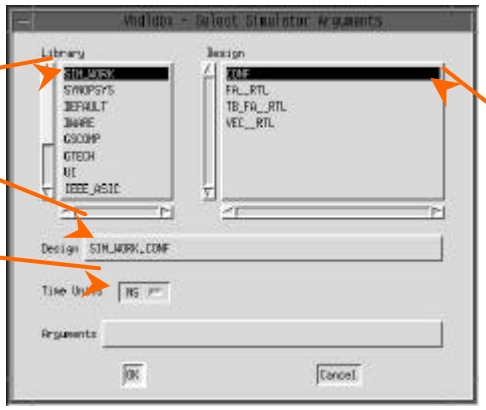
실습-1 : Test Bench Compile 정보에 대한 Loading 수행 시작

- Simulation Engine을 사용하기 위하여 Test Bench VHDL 표현에 대한 Loading을 수행하기 위하여 다음과 같은 명령어를 사용한다.
  - **$ vhdldbx &**를 수행하면 다음과 같은 Window가 나타난다.
- Loading에 따른 Window와 각 블럭에 대한 의미 분석

Working Library 지정

선택된 Design

Simulation 수행에 따른 시간 단위에 대한 설정

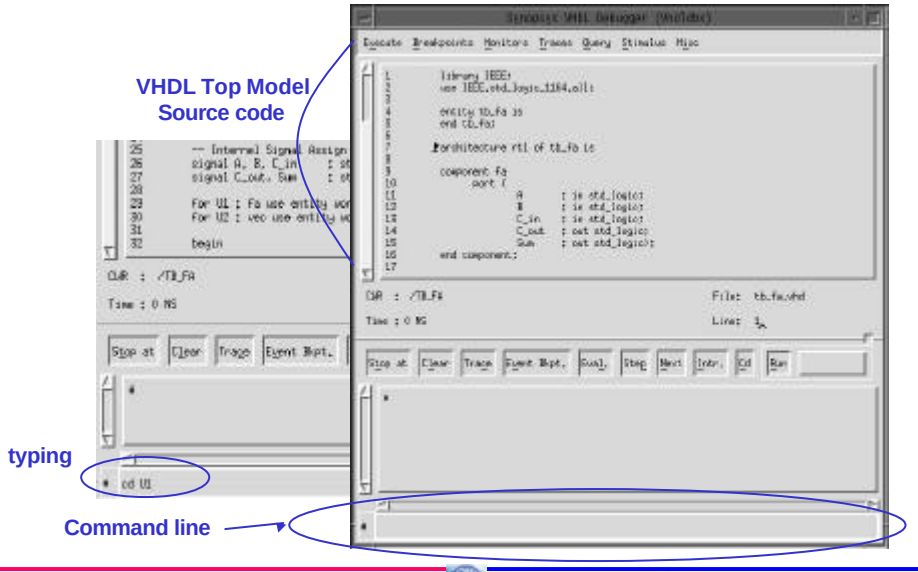


Simulation을 하고자 하는 최상위 Configuration 또는Entity_Architecture_name을 선택 한다.

(여기서는 configuration)

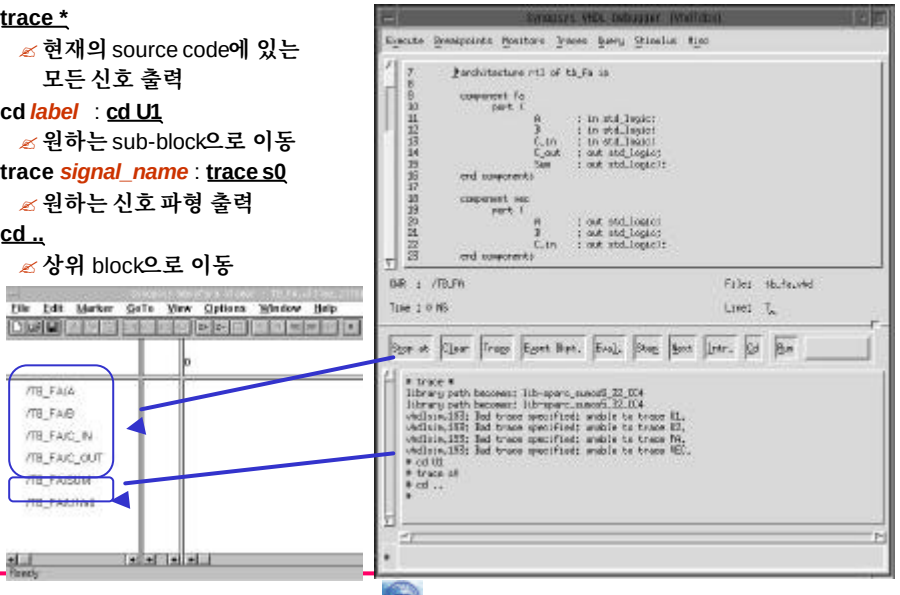
실습-1 : Test Bench Compile 정보에 대한 Loading 수행 결과

- Loading에 따라 생성되는 Window와 각 블럭에 대한 의미 분석



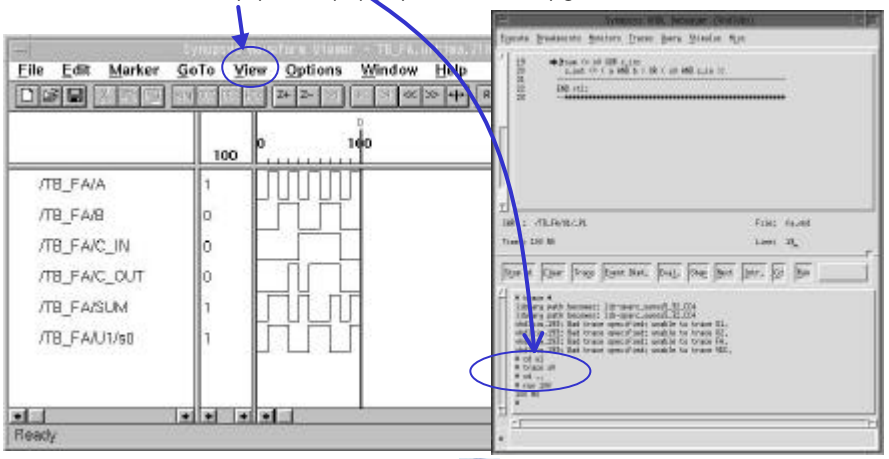
실습-1 : VHDL Debugger의 Command에 대한 정의

- 기본적인 Command 종류와 의미 분석(1)
  - **trace ***
    - 현재의 source code에 있는 모든 신호 출력
  - **cd *label*** : **cd U1**
    - 원하는 sub-block으로 이동
  - **trace *signal_name*** : **trace s0**
    - 원하는 신호 파형 출력
  - **cd ..**
    - 상위 block으로 이동



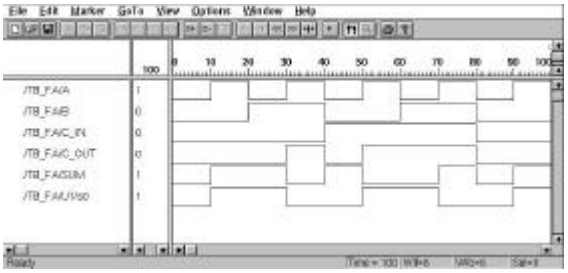
실습-1 : Simulation Engine에 대한 수행

- 기본적인 Command 종류와 의미 분석(2)
  - **run 100**
    - Simulation Engine이 현재 시간부터 100 ns의 Simulation을 수행하도록 하는 명령어
- waveform scale에 대한 조정
  - >tool-bar menu에서 View 에 해당하는 sub-menu 사용

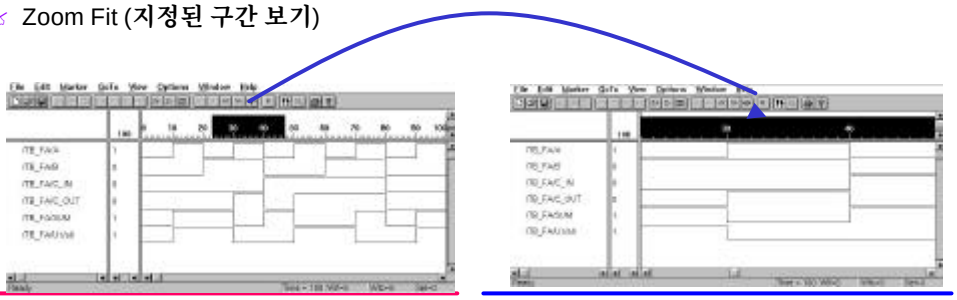


실습-1 : Simulation Waveform에 대한 분석 과정

- Full Fit (전체 보기)

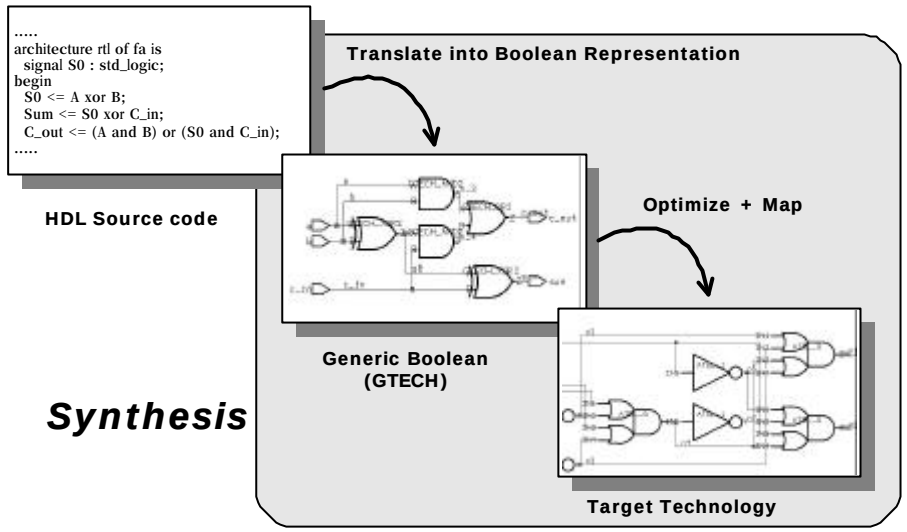


- Zoom In (확대) 또는 Zoom Out (축소)
- Zoom Fit (지정된 구간 보기)



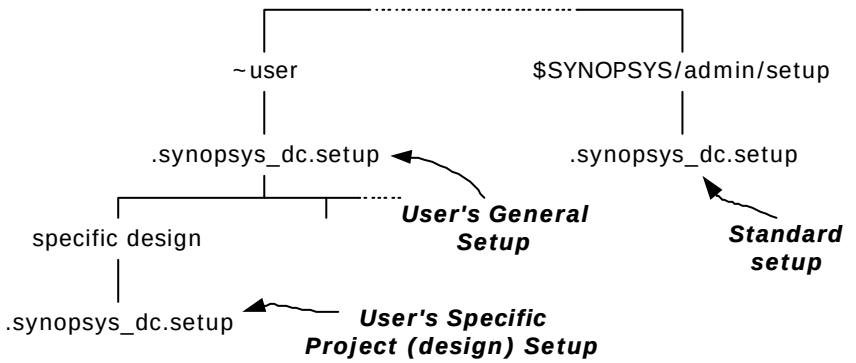
실습-2 : VHDL 표현에 대한 논리 합성의 기본 개념

- 논리 합성(Logic Synthesis)이란?
  - Synthesis = Translation + Optimization + Mapping



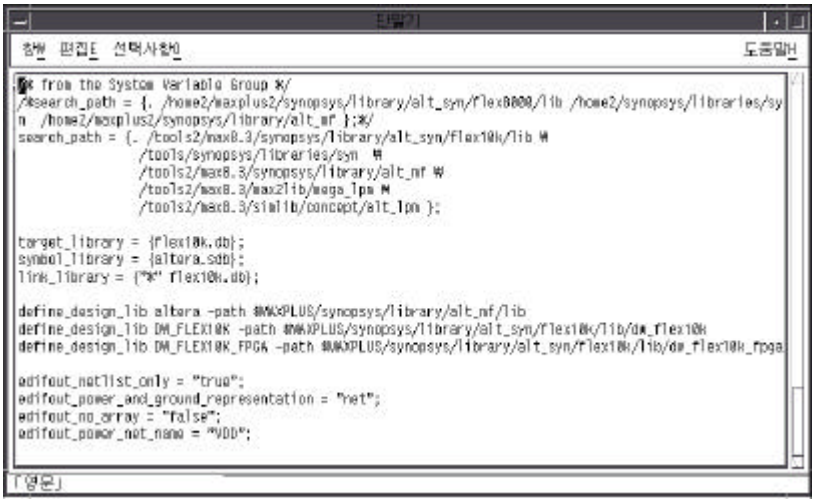
Synopsys Synthesis Tool을 위한 환경 설정 개념

- 환경 설정을 File과 개념 정의
  - Design Compiler의 Setup File : **.synopsys _dc.setup**
  - Setup File의 위치에 따라 미치는 범위가 다르게 된다.
    - Setup File이 없는 경우 : Synopsys Tool이 있는 곳의 Default Setup File을 사용한다.
    - 자신의 계정(account)에 있는 경우 : 자신의 계정에서 작업하는 모든 곳에 사용된다.
    - 특정한 Directory에 있는 경우 : 이 특정한 Directory에 있는 Design Model에 사용된다.



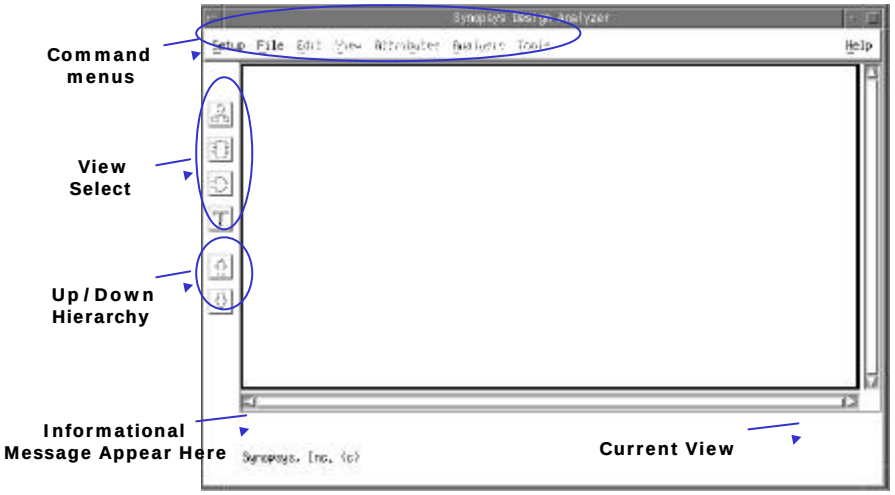
Synopsys Synthesis Tool을 위한 환경 설정의 예제

- 본 실습에서 사용하게 되는 .synopsys_dc.setup File에 대한 예제는 다음과 같다.
  - 가정 : Target Library를 ALTERA FLEX10K로 한 경우이다.



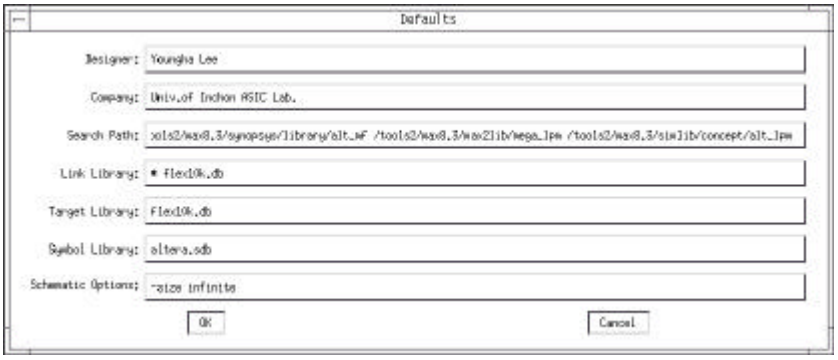
실습-2 : Design Analyzer Window의 설정

- Design Analysis를 위한 Tool의 Loading을 Unix Command로써 다음과 같이 수행한다.
  - $ **design_analyzer &**에 따라 다음과 같은 Window가 나타난다.



실습-2 : Check Design Compiler Setup

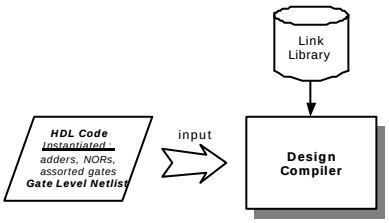
- Design Analysis 의 Command Menu에서 Setup의 Defaults..을 선택.
  - **>Setup > Defaults**로 선택하면 다음과 같은 Window가 Call된다.
  - 다음은 .synopsys_dc.setup File에서 설정된 환경을 Loading 한 경우이다.



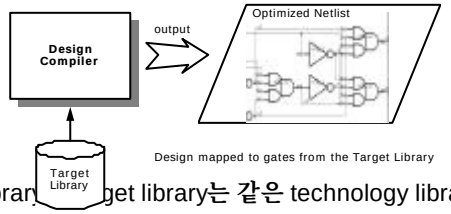
- 만일 설정이 되어 있지 않으면 여기서 다음에 해당하는 사항을 설정하여야 한다.
  - Link Library
  - Target Library
  - Symbol Library

실습-2 : Technology Library

- Link Library
  - Optimization 하기 전에 Design Compiler에게 cells과 wireloads, 동작 조건에 대한 정보를 제공하는 technology library.



- Target Library
  - Synthesis 하는 동안 design compiler가 mapping을 수행하는 technology library.



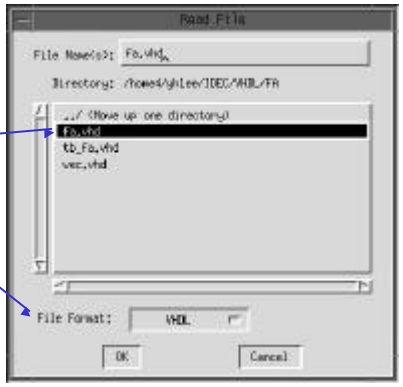
- 일반적으로 link library와 target library는 같은 technology library로 설정한다.

실습-2 : Design Compiler로 File 읽기(1)

- Design Analysis의 Command Menu에서 File의 Read..를 선택한다.
  - **>File > Read**의 선택에 의하여 다음의 Window 가 나타난다.
  - File Name과 File Format을 설정하고, OK를 선택한다.

Reading Design File

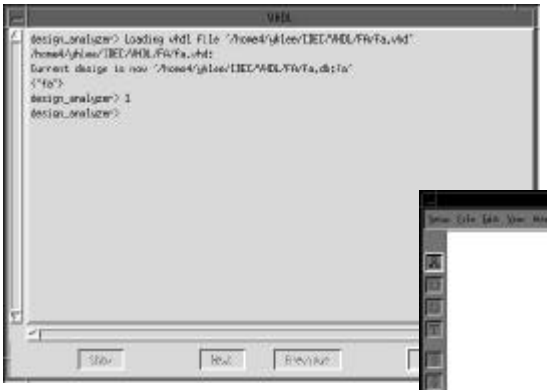
Supports many different formats

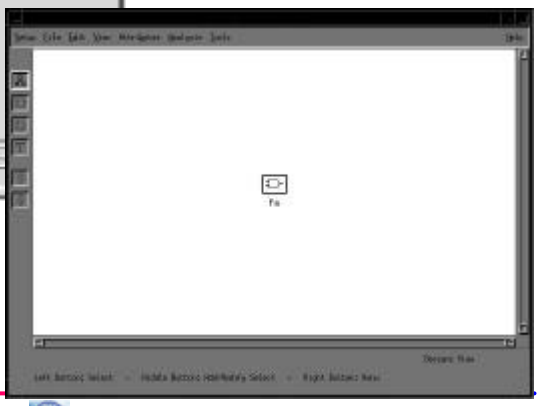


Format	Description
db	Synopsys Internal Format
edif	Electronics Design Interchange format
equation	Synopsys Equation Format
lsi	LSI Logic Corporation Format
mif	Mentor Intermediate Format
pla	Berkeley (Expresso) PLA Format
st	Synopsys State Table Format
tdl	Tegas Design Language Format
verilog	Cadence Design Systems Inc. HDL
vhdl	IEEE Standard VHDL
xnf	Xilinx Netlist Format

실습-2 : Design Compiler로 File 읽기(2)

- File Read를 수행하는 과정에 다음과 같은 Window가 생성된다.

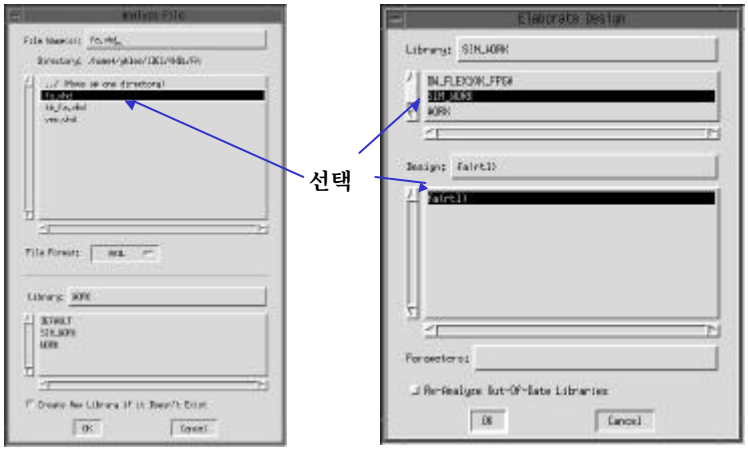




- Error가 없는 경우에 오른쪽과 같은 Window 생성된다.
  - 해당되는 VHDL Code에 따른 Symbol이 나타난다.

실습-2 : HDL Files에 대한 Translate의 수행

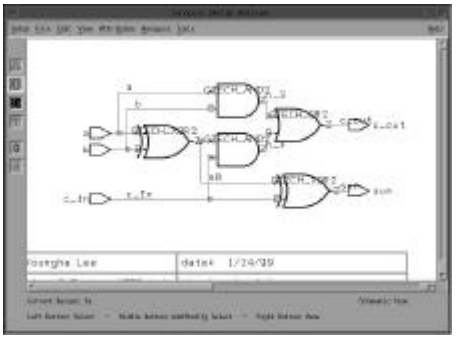
- File Read의 명령어와 동일한 의미로 다음의 명령어를 사용하여 수행할 수 있다.
  - Analyze와 Elaborate 명령
- Design Analysis의 Command Menu에서 다음과 같이 명령어를 선택한다.
  - **>File > Analyze**로 선택한다.
  - **>File > Elaborate**로 선택한다.



실습-2 : HDL Files에 대한 Translate의 수행 결과

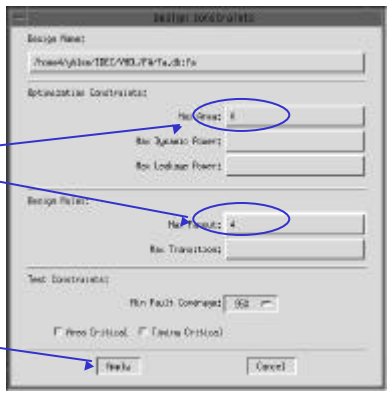
- Translate 수행 결과
  - 생성된 Symbol을 표시하는 Window는 다음과 같다.

- 해당되는 Symbol의 회로도 보기
  - 해당 Symbol을 Double Click 함.
  - Target Library와 설계자에 제시되는 설계 제약 조건을 고려하지 않은 논리 회로도가 생성, 출력됨



실습-2 : Optimization Constraints

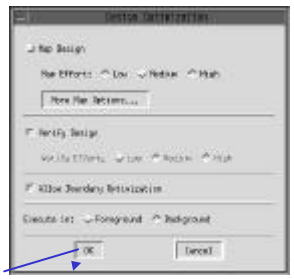
- Optimization을 수행할 Design Block을 선택하고, 다음의 과정을 수행한다.
  - **>Attributes > Optimization Constraints > Design Constraints**에 의하여 선택한다.
- 생성되는 Window는 다음과 같다.



- Optimization Constraints에 대한 설정
  - Max.Area는 최종적으로 줄이고자 하는 Area의 목표 값으로 보편적으로 0으로 설정한다.
  - 단위는 Vendor에 의해 제공되는 Target Technology Library 단위이다.
    - e.g. : 2-input-NAND-gate, inverters, transistors, or square mils, etc.
- Design Rule Constraints 에 대한 설정
  - Max Fanout에 대한 설정
  - Optimization 과정에서 요구되는 Fanout의 최대 개수를 지정한다.
- 설계에 따라 설계자의 다양한 선택이 요청되는 항목이다.

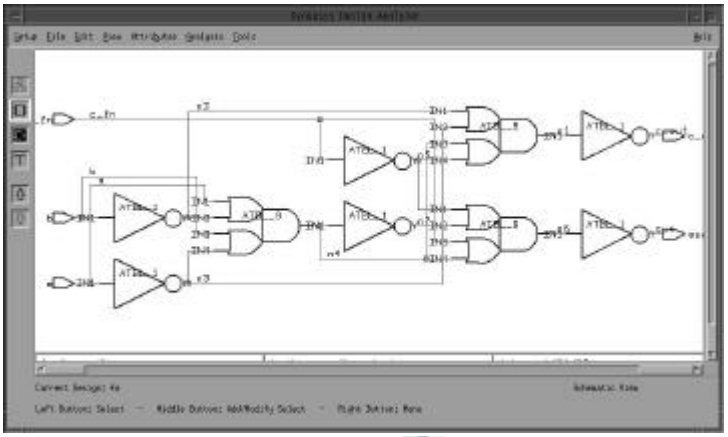
실습-2 : Optimization의 수행

- Optimization을 수행할 Design Block을 선택하고, 다음의 과정을 수행한다.
  - **>Tools > Design Optimization**에 의하여 선택한다.
- Design Optimization Window에서 "OK"를 선택한다.
  - Optimization의 과정을 표시하는 Window 가 나타나면서 수행된다.



실습-2 : Optimization의 수행에 따른 회로도 출력

- 각 Symbol에 대하여 Target Library를 사용한 설계 도면의 생성
  - 각 Symbol을 Double Click 하여 선택하면 다음과 같은 논리 회로도 도면이 생성된다.
  - VHDL Modeling 형태에 따라 다르다.
    - Behavioral Modeling : 논리 회로도 출력
    - Structural Modeling : 해당 Component로 연결되는 Symbol 구성도 출력



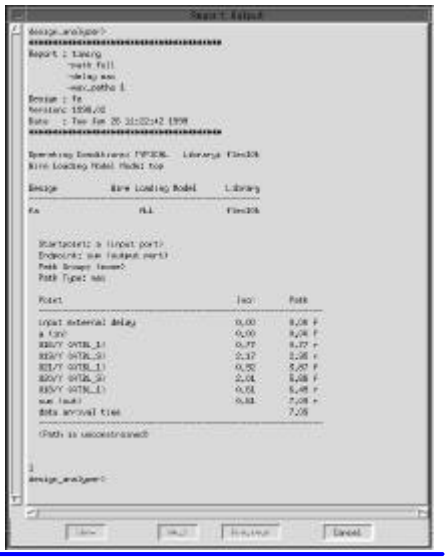
실습-2 : Optimization의 수행에 따른 Report 분석(1)

- 사용된 Cell에 따른 면적 정보, Delay 정보 및 Critical Path 정보 등을 분석한다.
  - Command Menu에서 **>Analysis>Report**를 선택하면 다음과 같은 Window 가 설정된다.

실습-2 : Optimization의 수행에 따른 Report 분석(2)

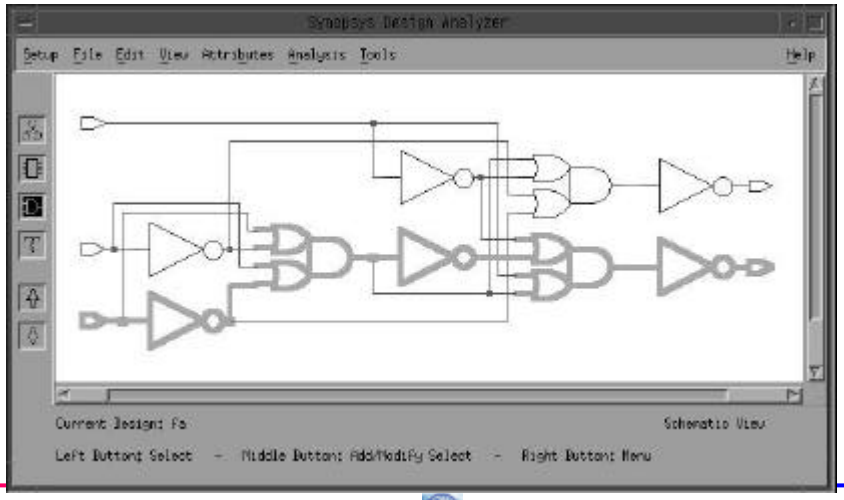
- 사용된 Area에 대한 Report File의 출력
  - 앞의 Window에서 **Area**를 선택한다.

- Delay 정보에 대한 Report File의 출력
  - 앞의 Window에서 **Timing**을 선택한다.



실습-2 : Optimization의 수행에 따른 Critical Path에 대한 분석

- 회로도의 Path 중에서 Timing Delay가 가장 크게 분석되는 Path의 설정을 한다.
  - Command Menu에서 **>Analysis>Highlight>Critical Path**를 선택하면 다음 Window가 나타난다.
  - 도면에서 굵은 선으로 표시된 Path가 Critical Path이다.

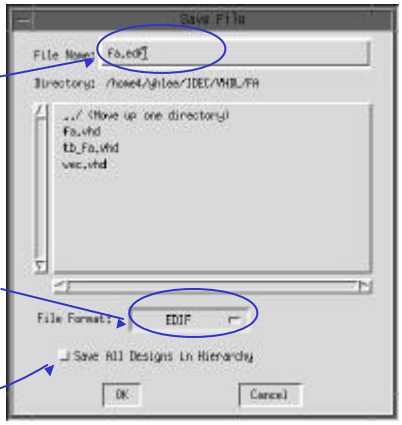


실습-2 : 설계 결과 저장

- Synthesis 된 결과를 저장
  - ALTERA Library를 사용하는 경우 **edif** 형식으로 저장

저장할 file 이름

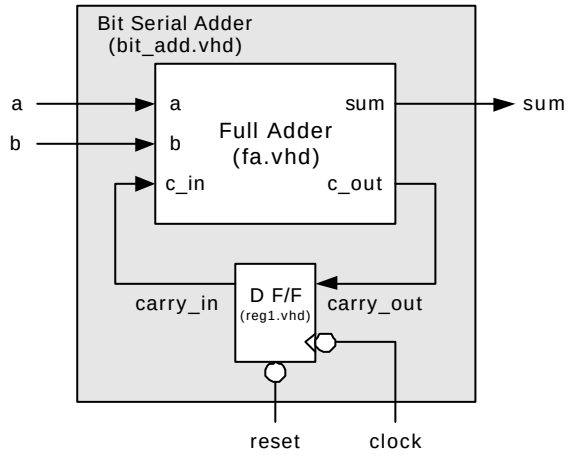
저장할 file 형식 설정



- Recommendation : Always use the save ALL Designs in Hierarchy option.

실습-3 : 직렬 가산기의 핵심 블럭에 대한 설계

- 실습1에서 설계한 Full Adder를 이용하여 N-Bit 직렬 가산기에 요구되는 핵심 블럭에 대한 VHDL 표현을 행한다.
- 직렬로 입력되는 두 개의 값을 더하여 직렬로 출력하는 구조이다.
  - carry를 저장하기 위하여 1-Bit의 Flip-Flop이 요구된다.



실습-3 : 직렬 가산기 블럭에 대한 VHDL 설계

- 1-Bit Flip-Flop에 대한 VHDL 표현

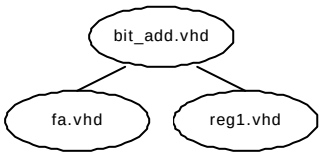
```
library IEEE;use IEEE.std_logic_1164.all;
entity reg1 is
 port ( reset, clock: in std_logic;
        d_in          : in std_logic;
        d_out         : out std_logic);
end reg1;
architecture rtl of reg1 is
begin
 process ( reset, clock, d_in )
 begin
  if reset = '0' then
   d_out <= '0';
  elsif clock = '0' and clock'event then
   d_out <= d_in;
  end if;
 end process;
end rtl;
```

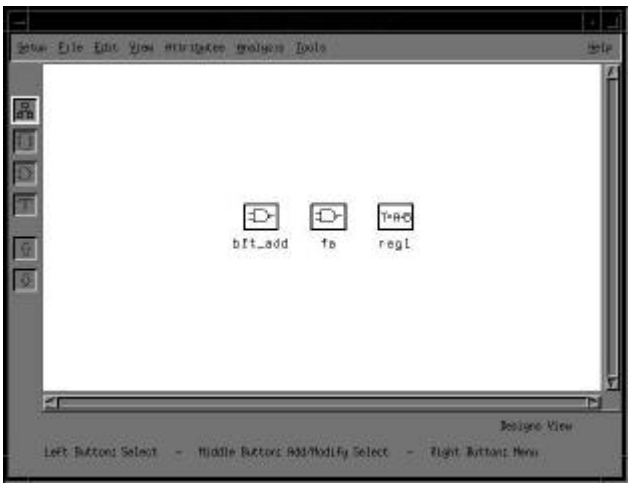
- 직렬 가산기에 대한 VHDL 표현

```
library IEEE; use IEEE.std_logic_1164.all;
entity bit_add is
 port ( a, b, reset, clock : in std_logic;
        sum : out std_logic);
end bit_add;
architecture rtl of bit_add is
 component fa
 port ( a, b, c_in : in std_logic;
        sum, c_out : out std_logic);
 end component;
 component reg1
 port ( reset, clock, d_in : in std_logic;
         d_out : out std_logic);
 end component;
 for u0 : fa use entity work.fa(rtl);
 for u1 : reg1 use entity work.reg1(rtl);
 signal carry_in, carry_out : std_logic;
begin
 u0 :fa port map (a, b, carry_in, sum, carry_out);
 u1 : reg1 port map (reset, clock, carry_out, carry_in);
end rtl;
```


실습-3 : 직렬가산기 블럭에 대한 File Read

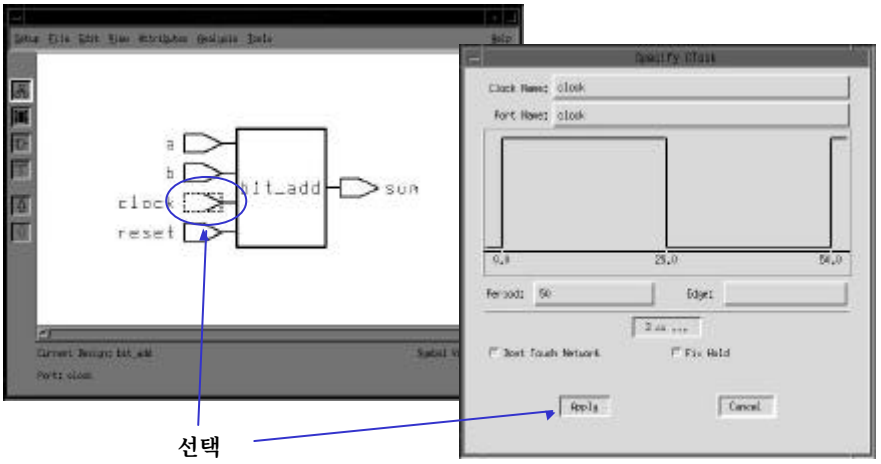
- 실습1과 같이 합성 하고자 하는 3가지 VHDL Code에 대하여 File Read를 수행한다.
  - 계층적 설계의 경우 하위 블럭에 대응하는 VHDL Code부터 File Read를 수행한다.
  - **>File>Read>fa.vhd**
  - **>File>Read>reg1.vhd**
  - **>File>Read>bit_add.vhd**





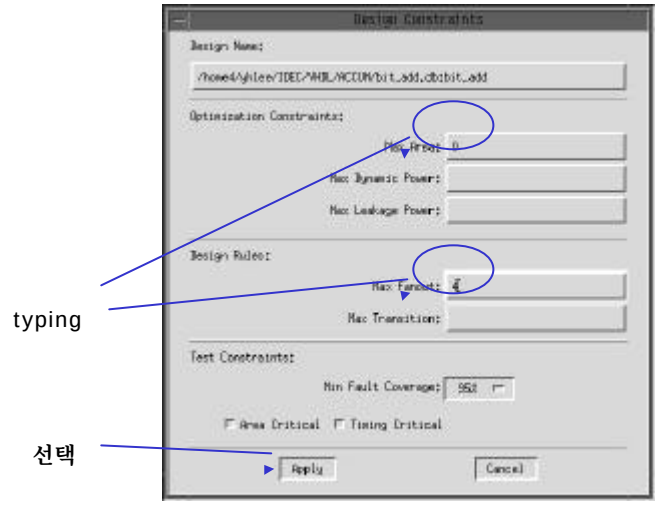
실습-3 : Clock 주기에 대한 설정

- Clock에 attributes 를 주는 경우
  - Design Analyzer에서 clock에 대응하는 신호를 선택하여 원하는 동작 속도를 지정한다.
  - Design Analysis의 Command Menu에 서 Attributes의 Clock, Specify를 선택한다.
    - **>Attributes > Clock > Specify**에 의하여 지정한다.



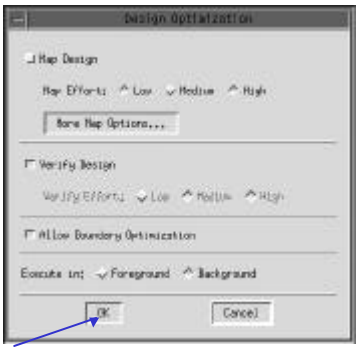
실습-3 : Design Constraint에 대한 설정

- Optimization을 수행 하기 전에 Optimization 조건과 Design Rule에 대하여 설정한다.
  - 계층적 설계인 경우 최상위 symbol을 선택한 다음 수행한다.
  - **>Attributes > Optimization Constraints > Design Constraints** 에 의하여 선택된다.



실습-3 : Optimization에 대한 조건 설정과 수행

- 먼저 Optimization을 수행할 Design Block을 선택하고, 다음 과정을 수행한다.
  - **>Tools > Design Optimization**에 의하여 선택한다.

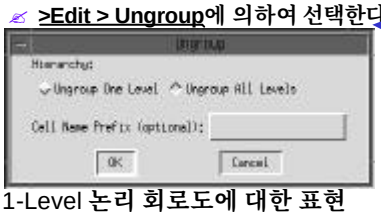


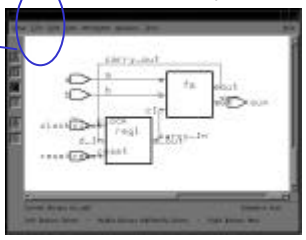
선택

- Design Optimization Window에서 "OK"를 선택한다.
  - Optimization의 과정을 표시하는 Window 가 나타나면서 수행된다.
- 사용된 Cell 정보 및 Delay 정보 등의 출력을 분석하여 보자.
  - Report File의 생성과 분석

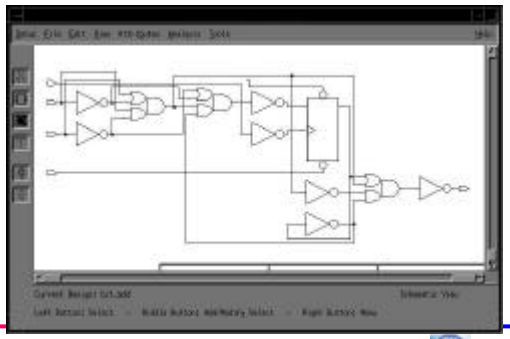
실습-3 : 계층적 설계 도면의 1-Level로의 변환 과정 (선택과정)

- 계층적으로 합성된 경우 1-Level로 변환하려면, 변환하려는 부분을 왼쪽 마우스로 영역을 선택한 후 다음과 같은 일을 수행한다.
  - **>Edit > Ungroup**에 의하여 선택한다.



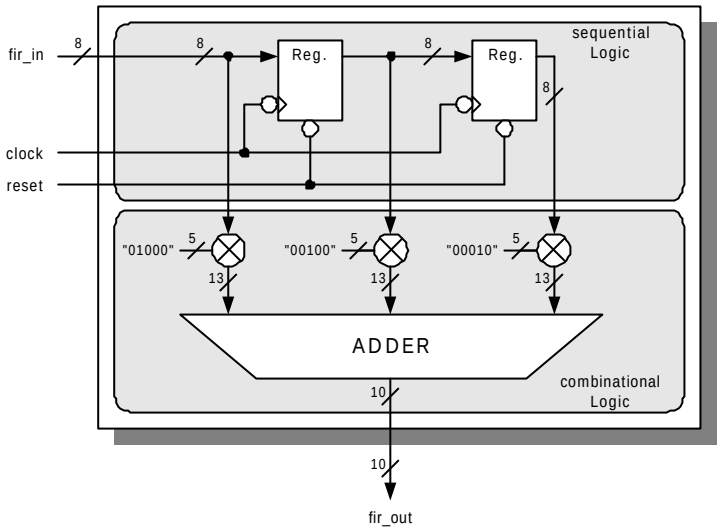


- 1-Level 논리 회로도에 대한 표현



실습-4 : 3-Tap FIR Filter에 대한 RTL 회로

- 3-Tap FIR Filter에 대한 사양 분석에 따라 다음과 같은 RTL 회로를 생성한다.
- VHDL 표현을 행하고, Simulation과 Synthesis 를 수행한다.



실습-4 : 3-Tap FIR Filter에 대한 VHDL 표현

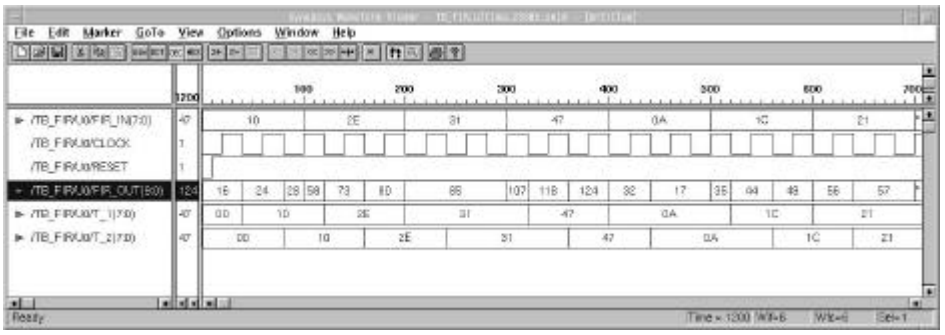
```
library IEEE; use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
entity fir is
  port ( fir_in : in std_logic_vector(7 downto 0);
         clock : in std_logic;
         reset : in std_logic;
         fir_out : out std_logic_vector(9 downto 0));
end fir;
--
architecture rtl of fir is
  signal t_1, t_2  : std_logic_vector(7 downto 0);
-- filter 계수에 대한 선언
  constant a1 : std_logic_vector(4 downto 0) :=
                                          "01000";
  constant a2 : std_logic_vector(4 downto 0) :=
                                          "00100";
  constant a3 : std_logic_vector(4 downto 0) :=
                                          "00010";
begin
--filter register에 대한 표현
   process(clock,reset) begin
     if reset = '0' then
        t_1 <= (others => '0');
        t_2 <= (others => '0');
     elsif clock'event and clock = '0' then
        t_1 <= fir_in;
        t_2 <= t_1;
     end if;
   end process;
--filter 값 계산을 위한 산술 연산 블럭의 표현
   process(fir_in, t_1, t_2)
     variable b_0, b_1, b_2, tmp :
           std_logic_vector(12 downto 0);
   begin
      b_0 := a1 * fir_in;    b_1 := a2 * t_1;
      b_2 := a3 * t_2;    tmp := b_0 + b_1 + b_2;
      fir_out <= tmp(12 downto 3);
   end process;
end rtl;
```

실습-4 : FIR Filter의 검증을 위한 Test Bench VHDL 표현

```
library IEEE; use IEEE.std_logic_1164.all;
entity tb_fir is end tb_fir;
--
architecture rtl of tb_fir is
--
  component fir
   port ( fir_in   : in std_logic_vector(7 downto 0);
          clock    : in std_logic;
          reset    : in std_logic;
        fir_out : out std_logic_vector(9 downto 0));
  end component;
--
  for u0: fir use entity work.fir(rtl);
--
  signal fir_in   : std_logic_vector(7 downto 0);
  signal clock, reset  : std_logic := '0';
  signal fir_out  : std_logic_vector(9 downto 0);
--
begin
  u0: fir port map ( fir_in, clock, reset, fir_out);
  clock <= not clock after 20 ns;
--
  reset <= '1' after 10 ns;
--
  process
  begin
   fir_in <= "00010000", "00101110" after 100 ns,
             "00110001" after 200 ns,
             "01000111" after 300 ns,
             "00001010" after 400 ns,
             "00011100" after 500 ns,
             "00100001" after 600 ns,
             "00000111" after 700 ns;
    wait for 800 ns;
  end process;
end rtl;
--
configuration conf of  tb_fir is
  for rtl  end for;
end conf;
```

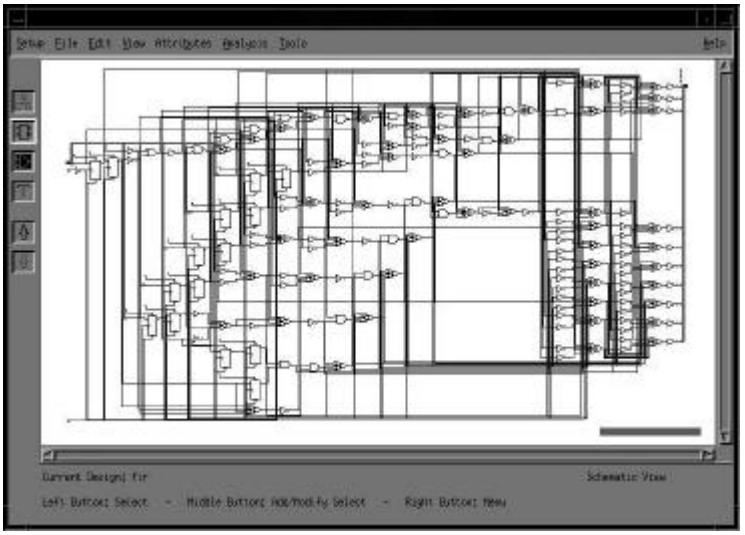

실습-4 : Synopsys Simulator를 이용한 VHDL 검증

- Waveform의 생성에 따른 설계 사양과의 기능 일치성 분석



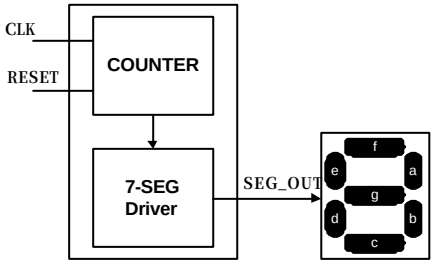
실습-4 : Synopsys Synthesis Tool을 이용한 회로도 합성

- 논리 회로도 합성과 설계 조건에 따른 만족성 분석
  - 설계도면 분석, 각종 Report File에 대한 분석



실습-5 : Synthesis Tool 실습을 위한 예제의 동작 사양

- 목적
  - Simulation Tool에 대한 요약과 회로 합성에 대한 단계별 설명과 실습
- 설계 사양
  - 10 진 Counter의 설계
  - 10 진 Counter의 출력 값 표시를 위한 Seven-Segment 드라이버 설계



- 10진 Counter

RESET	CLK	CNT_OUT(n)	CNT_OUT(n+1)
1	X	X	“0000”
0	?	“1001”	“0000”
0	?	“1001” 이외 값	CNT_OUT(n) + ‘1’

- Segment

CNT_OUT	SEG_OUT
“0000”	"1111110"
“0001”	"1100000"
“0010”	"1011011"
“0011”	"1110011"
“0100”	"1100101"
“0101”	"0110111"
“0110”	"0111111"
“0111”	"1100110"
“1000”	"1111111"
“1001”	"1110111"

실습-5 : 예제의 Behavioral Level VHDL 설계

Counter의 VHDL 설계

```
library IEEE;  use IEEE.Std_logic_1164.all;
use IEEE.Std_logic_unsigned."+";
entity Counter is
      port (   RESET : in std_logic;
      Clk  : in std_logic;
      CNT_out : out std_logic_vector(3 downto 0));
end Counter;
architecture Counter_Arch of Counter is
  signal CNT : std_logic_vector(3 downto 0);
begin
      process(RESET, CLK)
      begin
         if RESET = '1' then
           -- same expression of CNT <= "0000";
           CNT <= (others => '0');
         elsif Clk'event and Clk = '1' then
           if CNT = "1001" then
            CNT <= (others => '0');
           else
            CNT <= CNT + '1';
           end if;
         end if;
      end process;
      CNT_out <= CNT;
end Counter_Arch;
```

Segment의 VHDL 설계

```
library IEEE;  use IEEE.Std_logic_1164.all;
entity Segment is
      port ( Seg_IN : in std_logic_vector(3 downto 0);
         Seg_OUT : out std_logic_vector(6 downto 0));
end Segment;
architecture Segment_Arch of Segment is
begin
  process(Seg_IN)
  begin
      case Seg_IN is
               when "0000" => Seg_OUT <= "1111110";
               when "0001" => Seg_OUT <= "1100000";
               when "0010" => Seg_OUT <= "1011011";
               when "0011" => Seg_OUT <= "1110011";
               when "0100" => Seg_OUT <= "1100101";
               when "0101" => Seg_OUT <= "0110111";
               when "0110" => Seg_OUT <= "0111111";
               when "0111" => Seg_OUT <= "1100110";
               when "1000" => Seg_OUT <= "1111111";
               when "1001" => Seg_OUT <= "1110111";
               when others => Seg_OUT <= "0000000";
               end case;
      end process;
end Segment_Arch;
```


실습-5 : 예제의 Structural Level 및 Test Bench VHDL 설계

CNT_SEG의 VHDL 설계

```
library ieee; use ieee.std_logic_1164.all;
entity CNT_SEG is
port ( RESET : in std_logic;
        Clk : in std_logic;
        Seg_out : out std_logic_vector(6 downto 0));
end CNT_SEG;
architecture CNT_SEG_Arch of CNT_SEG is
  component Counter
  port ( RESET : in std_logic;
          Clk : in std_logic;
          CNT_out : out std_logic_vector(3 downto 0));
  end component;
  component Segment
  port ( Seg_IN : in std_logic_vector(3 downto 0);
          Seg_OUT : out std_logic_vector(6 downto 0));
  end component;
  signal CNT_out : std_logic_vector(3 downto 0);
  for CNT : Counter
       use entity work.Counter(Counter_Arch);
  for Seg : Segment
       use entity work.Segment(Segment_Arch);
begin
  CNT : Counter port map
        (RESET=> RESET,Clk=> Clk,CNT_out=> CNT_out);
  Seg : Segment port map
        (Seg_IN=> CNT_out,Seg_OUT=> Seg_OUT);
end CNT_SEG_Arch;
configuration CNT_SEG_C of CNT_SEG is
  for CNT_SEG_Arch
  end for;
end CNT_SEG_C;
```

Test-Bench를 위한 VHDL 설계

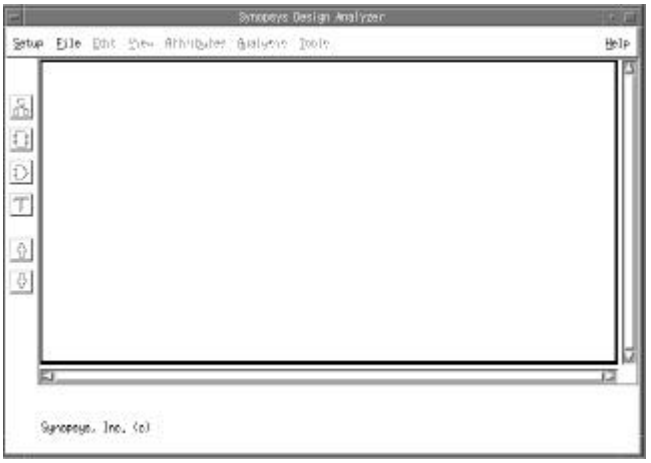
```
library IEEE; use IEEE.Std_logic_1164.all;
entity TB_CNT_SEG is
end TB_CNT_SEG;
--
architecture TB_CNT_SEG_Arch of TB_CNT_SEG is
component CNT_SEG
  port ( RESET : in std_logic;
          Clk : in std_logic;
          Seg_out : out std_logic_vector(6 downto 0));
end component;
  signal RESET          : std_logic := '1' ;
  signal Clk: std_logic := '1' ;
  signal Seg_out        : std_logic_vector(6 downto 0);
begin
  RESET <= '0' after 5 ns;
  Clk <= not Clk after 10 ns;
  Counter : CNT_SEG port map (
               RESET     => RESET,
               Clk       => Clk,
               Seg_out   => Seg_out);
end TB_CNT_SEG_Arch;
--
configuration TB_CNT_SEG_C of TB_CNT_SEG is
 for TB_CNT_SEG_Arch
  end for;
end TB_CNT_SEG_C;
```

실습-5 : VSS Simulator를 사용한 기능 검증

- 각 VHDL Source Code에 대하여 Analysis 를 수행한다.
  $ **vhdlan Counter.vhd**
  $ vhdlan Segment.vhd
  $ vhdlan CNT_SEG.vhd
  $ vhdlan TB_CNT_SEG.vhd
  - File_Name.scr에 위의 내용을 저장하고, Script File을 수행한다. (예제 : Comp.scr)
- Debugging Tool의 Loading과 Simulation Library 및 Design Model을 선택한다.
  $ **vhdldbx &**
  - 예제에서의 Design Model은 **SIM_WORK.TB_CNT_SEG_C**이다.
- Waveform 생성단계
  - 다음 명령어를 하나씩 수행하면 Waveform과 값이 생성된다.
    trace *
    trace /TB_CNT_SEG/Counter/CNT_out
    run 300
  - Script File을 이용하는 경우
    - File_Name.scr에 위의 내용을 저장한다.
    - Command 명령어에서 **include Comp.scr**을 실행한다.
    - Waveform과 값이 생성된다.

실습-5 : Design Analyzer Window의 설정

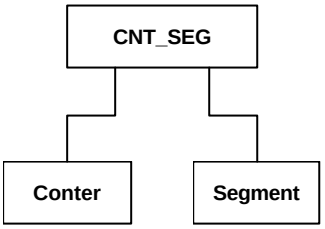
- Design Analysis 를 위한 Tool의 Loading을 수행한다.
  $ design_analyzer &
  - 다음과 같은 Window가 Call된다.

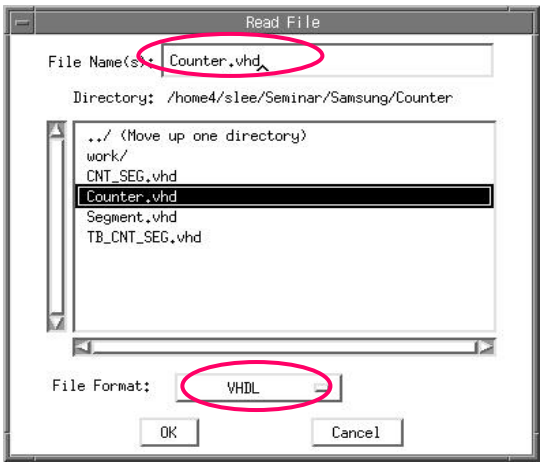


실습-5 : VHDL File의 Read 수행

- VHDL Source File을 Read 하여 내부 데이터 형태로 저장한다.
  - Window 상에서 **File -> Read**를 선택하면 다음과 같은 Window가 Call된다.
  - Window상에서 Bottom-Up 즉, 하위부터 VHDL Code를 Read 한다.
    - Counter.vhd
    - Segment.vhd
    - CNT_SEG.vhd

계층구조

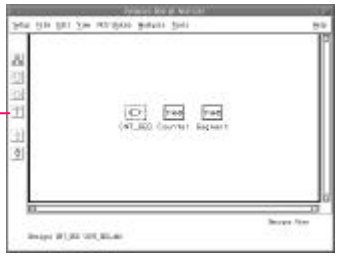




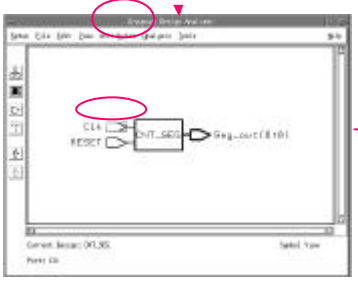
실습-5 : Clock 신호에 대한 지정

- Clock 신호와 주기를 결정한다.

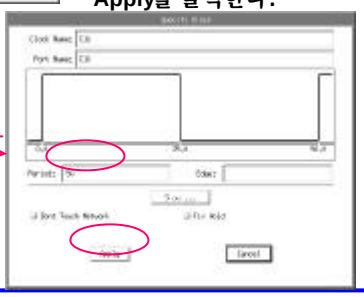
1. CNT_SEG를 Double클릭한다.



2. Clk핀을 클릭한다.

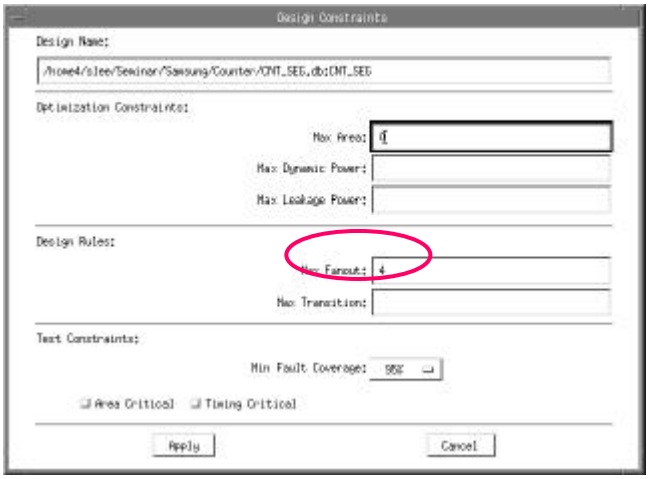


3. Attribute -> clock -> Specify를 클릭하고 다음 그림과 같이 설정하고 Apply를 클릭한다.



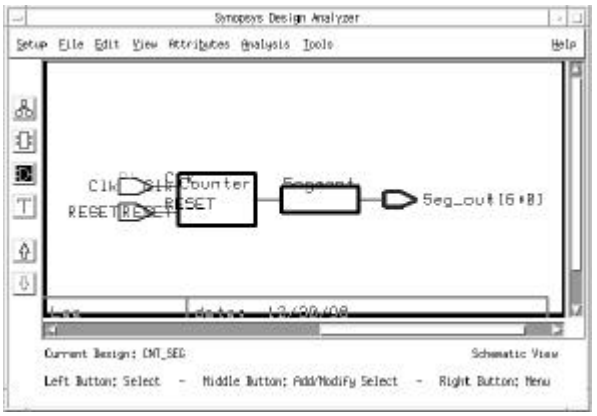
실습-5 : Design Constraint의 설정

- Optimization 조건과 Design Rule을 설정한다.
  - 계층적 설계인 경우 최상위 symbol을 선택한 다음 수행한다.
  - Attributes -> Optimization Constraints ->Design Constraints 에 의하여 선택된다.



실습-5 : 논리 회로도의 생성

- 논리 회로를 생성하며, 관찰한다.
  - 다음은 설계 예제의 최상위 Block인 **CNT_SEG**의 회로도 이다.
  - 하위 Level의 논리 회로를 보기 위해서는 회로 상에서 보고자 하는 Component Unit을 Double Click한다.



실습-5 : Physical Design을 위한 EDIF File 생성

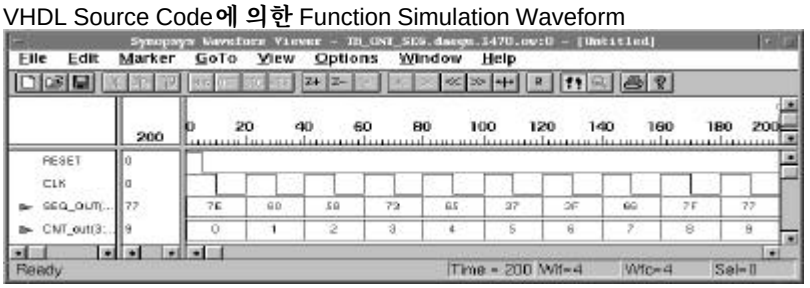
- EDIF File 생성
  - File -> Save as에 의하여 File Format을 **EDIF**로 선택
  - File Name을 본 예제의 경우 **CNT_SEG.edf** 로 설정하여 다음과 같이 입력한다.

실습-5 : FPGA Compiler에 의한 Physical Design

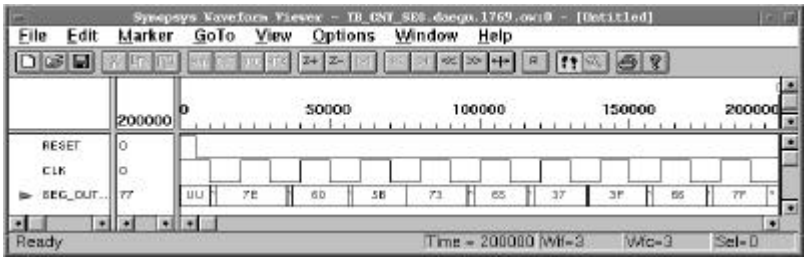
- 본 예제에서는 MAX+PLUSII를 사용함.
- $ maxplus2&를 실행한다
- File -> Project -> Name 에 서 **CNT_SEG.edf** 를 선택한다.
- MAX+PLUSII -> Compile를 클릭한다.
- Interface -> Edif Netlist Reader Settings에서 Synopsys 를 선택한다.
- Interface -> VHDL Netlist Writer 를 클릭한다. 출력 VHDL File은 **Project Name.vho**로 출력되며 Timing 정보를 이용한 검증을 위해 사용된다.
- Assign -> Device 를 선택한다.
  - Family는 FLEX8000을 선택한다.
  - Devices 는 예 제 에 서 는 EPF8282LC84-3을 선택한다.
- Compiler Window에서 Start를 클릭한다.

실습-5 : 합성된 회로에 대한 Timing Simulation

- VHDL Source Code에 의한 Function Simulation Waveform

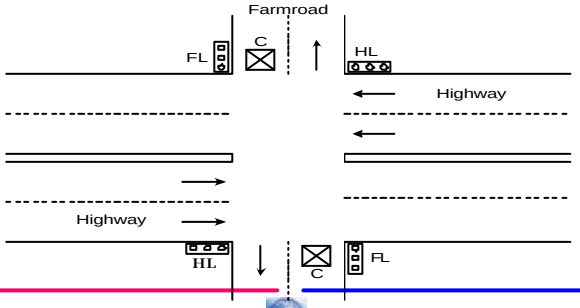


- FPGA Compiler에 의하여 생성된 Timing 정보를 갖는 VHDL File에 대한 Timing Simulation Waveform



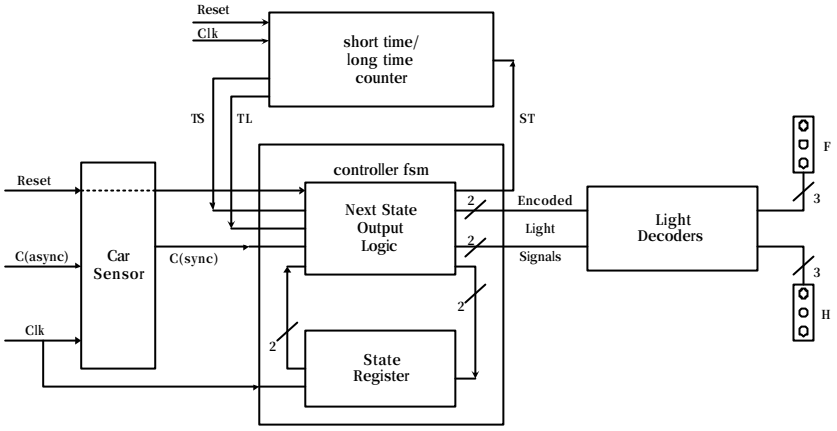
실습-6 : Traffic Light Controller의 설계 사양 및 분석

- Word Description 형태의 설계 사양에 대한 분석
  - Highway와 Farm road가 교차하는 지점에 대한 신호등의 제어기 설계
  - Farm Road에 차가 대기하고 있음은 Car 검출기에 의하여 검출됨
  - 초기적으로 Highway 방향으로 Green 신호를 출력하는데 TL 시간이 경과하면 Car 검출 신호를 조사하여 검출되면 Highway Yellow Mode로 전환하나, 검출되지 않으면 유지한다.
  - Highway Yellow Mode 전환 후에는 TS 시간이 경과하면, Highway Red Mode로 전환한다.
  - Farm Road Green Mode에서는 TL 시간이 경과하면 Car 검출 여부에 관계없이 Farm Road Yellow Mode 전환하며, Car 가 검출되지 않으면 TL 시간 도달 전일지라도 Farm Road Yellow Mode로 전환한다.
  - Farm Road Yellow Mode에서는 TS 시간이 경과하면 Highway Green Mode로 전환한다.
  - TL 시간은 (16개 CLK)*(CLK 주기 30초)로 하며, TS 시간은 (1개 CLK)*(CLK 주기 30초)로 한다.



실습-6 : Traffic Light Controller의 사양 분석과 Block 설계

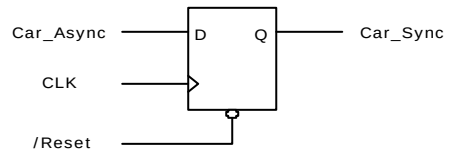
- 설계 사양 분석으로 Block Diagram 의 추출
  - 여러 개의 Module로 개념에 따라 분할 : Modular Design
  - I/O Pin에 대한 추출과 정의



실습-6 : Traffic Light Controller의 기능 및 구조 설계

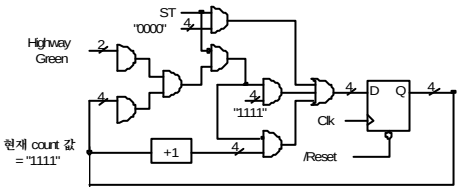
- Car Sensor 기능 및 구조 설계
  - 외부 입력 신호인 Car-Async 신호를 CLK에 동기 시켜 내부에서 사용하기 위한 회로

/Reset	Car_Async	CLK	Car_Sync
0	x	x	0
1	0	↑	0
1	1	↑	1
위 와다른경 우			변화 없 음



- Short Time 및 Long Time 신 호 의 생성을 위한 4-비트 Counter 기능 및 구조 설계

/Reset	ST	현재상태	현재 count 값	다음 count 값
0	x	x	x	"0000"
1	1	x	x	"0000"
1	0	Highway Green	"1111"	"1111"
위와 다른경우				현재 count 값+'1'

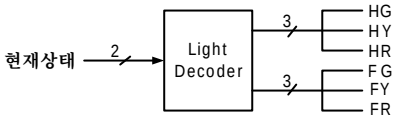


현재 count 값	TL	TS
"1111"	1	0
"0000"	0	1
위와 다른경우	0	0

실습-6 : Traffic Light Controller 기능 및 구조 설계

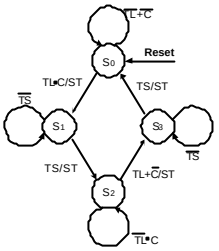
- Light Decoders 기능 및 구조 설계
  - 4가지 상태에 따른 분류
  - High-Way 방향과 Farm-Road 방향에 대한 신호등의 색깔을 4가지 상태에 따라 결정하는 Decoder 회로

H0(F0)	H1(F1)	HG(FG)	HY(FY)	HR(FR)	Light
0	0	1	0	0	Green
0	1	0	1	0	Yellow
1	0	0	0	1	Red
1	1	Not Allowed			



- Traffic Light State Controller 기능 설계
  - 4가지 상태의 설정과 이들 사이의 천이를 나타내는 State Table과 State Diagram

Reset	TS	TL	C	Current State	Next State	ST (Next state)
0	x	x	x	S0	S0	0
1	x	0	x	S0	S0	0
1	x	x	0	S0	S0	0
1	x	1	1	S0	S1	1
1	0	x	x	S1	S1	0
1	1	x	x	S1	S2	1
1	x	0	1	S2	S2	0
1	x	1	x	S2	S3	1
1	x	x	0	S2	S3	1
1	0	x	x	S3	S3	0
1	1	x	x	S3	S0	1



실습-6 : Traffic Light Controller의 VHDL 설계

- VHDL을 이용한 표현

```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned."+";
entity TRAFFIC is
port (RESET, CLK, C_ASYNC : in  std_logic;
      HG, HY, HR, FG, FY, FR : out std_logic);
end TRAFFIC;
architecture RTL of TRAFFIC is
   type STATE is (S0, S1, S2, S3);
   type LIGHT is (GREEN, YELLOW, RED);
   signal C_STATE, N_STATE : STATE;
   signal H, F : LIGHT;
   signal C_SYNC, ST, TS, TL : std_logic;
   signal COUNT:std_logic_vector(3 downto 0);
begin
  A: block begin
   process (RESET, CLK) begin
      if RESET = '0' then
        C <= '0';
      elsif CLK = '1' and CLK'event then
        C <= C_ASYNC;
     end if;
   end process;
```

```
--Timer 회로를 counter를 이용하여 표현
-- HG에서의 count 값 변화에 유의할 것
  process (RESET, CLK) begin
    if RESET='0' then
       COUNT <= "0000";
    elsif CLK = '1' and CLK'event  then
      if ST = '1' then
        COUNT <= "0000";
      elsif C_STATE=S0 and COUNT="1111" then
        COUNT <= COUNT; --NULL 구문과 동일
      else
        COUNT <= COUNT + '1';
      end if;
    end if;
  end process;
--Long time, short time 신호에 대한 생성
  TS<='1' when COUNT="0000" else '0';
  TL<='1' when COUNT="1111" else '0';
--High-way 신호등 ON/OFF 신호 생성
  HG<='1' when H=GREEN  else '0';
  HY<='1' when H=YELLOW else '0';
  HR<='1' when H=RED else '0';
```

실습-6 : Traffic Light Controller의 VHDL 설계

```
  --Farm road의 신호등 ON/OFF 신호 생성
  FG<='1' when F=GREEN else '0';
  FY<='1' when F=YELLOW else '0';
  FR<='1' when F=RED else '0';
end block;
--상태 기억 변수의 설정에 대한 표현
B: block begin
  process (RESET, CLK) begin
    if RESET = '0' then
      C_STATE <= S0;
    elsif CLK = '1' and CLK'event then
      C_STATE <= N_STATE;
    end if;
  end process;
  --다음 상태 결정과 timer 제어 신호 결정
  process (C_STATE, TS, TL, C_SYNC) begin
    case C_STATE is
      when S0 => H <= GREEN; F <= RED;
        if (TL = '1') and (C_SYNC = '1') then
          N_STATE <= S1; ST <= '1';
        else
          N_STATE <= S0; ST <= '0';
        end if;
      when S1 => H <= YELLOW; F <= RED;
        if TS = '1' then
          N_STATE <= S2; ST <= '1';
        else
          N_STATE <= S1; ST <= '0';
        end if;
      when S2 => H <= RED; F <= GREEN;
        if (TL = '1') or (C_SYNC = '0') then
          N_STATE <= S3; ST <= '1';
        else
          N_STATE <= S2; ST <= '0';
        end if;
      when S3 => H <= RED; F <= YELLOW;
        if TS = '1' then
          N_STATE <= S0; ST <= '1';
        else
          N_STATE <= S3; ST <= '0';
        end if;
    end case;
  end process;
end block;
end RTL;
```


실습-6 : Traffic Light Controller Test Bench 설계

- Simulation을 위한 Test Bench 표현
  - Stimulus Vector 생성부
  - Traffic Light Controller 본체부

```
library IEEE; use IEEE.std_logic_1164.ALL;
--Empty entity에 대한 표현
entity TB_TRAFFIC is end TB_TRAFFIC;
architecture RTL of TB_TRAFFIC is
  --Test 할 component에 대한 선언
  component TRAFFIC
    port (RESET,CLK,C_AYNC : in std_logic;
          HG,HY,HR,FG,FY,FR : out std_logic);
  end component;
  --연결함에 요구되는 신호에 대한 선언
  signal RESET : std_logic;
  signal CLK : std_logic := '0';
  signal C_ASYNC : std_logic;
  signal HG,HY,HR,FG,FY,FR : std_logic;
  --component에 대한 configuration 선언
  for U0 : TRAFFIC use entity work.TRAFFIC(RTL);
begin
  --주기적 신호에 대한 vector 생성 구문
  CLK <= not CLK after 15 sec;
  --초기화 신호에 대한 vector 생성 구문
  RESET <= '0', '1' after 1 sec;
  --Random 신호에 대한 vector 생성 구문
  process begin
      C_ASYNC <= '0';
      wait for 10 min;
      C_ASYNC <= '1';
      wait for 5 min;
      C_ASYNC <= '0';
      wait for 5 min;
      C_ASYNC <= '1';
      wait for 5 min;
  end process;

  --동작 회로의 Pin 정보와 Wire 정보의 연결
  U0 : TRAFFIC
    port map (RESET, CLK, C_ASYNC, HG, HY,
              HR, FG, FY, FR);
end RTL;
```

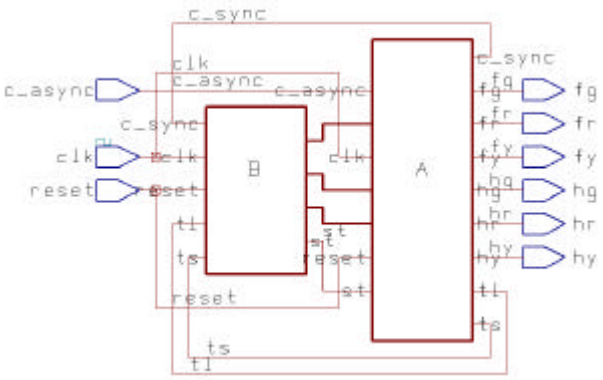
실습-6 : Traffic Light Controller VHDL 표현에 대한 검증

- Test Bench에 대한 Synopsys Simulation에 따른 파형

실습-6 : Traffic Light Controller의 계층적 논리 합성

- Script File에 대한 예

```
read -f vhdl traffic.vhd
  group -hdl_block A
  group -hdl_block B
create_clock clk
max_delay 0 -to all_outputs()
compile -map_effort high
create_schematic
write -f db

write -f edif -hierarchy -o traffic.edf
```

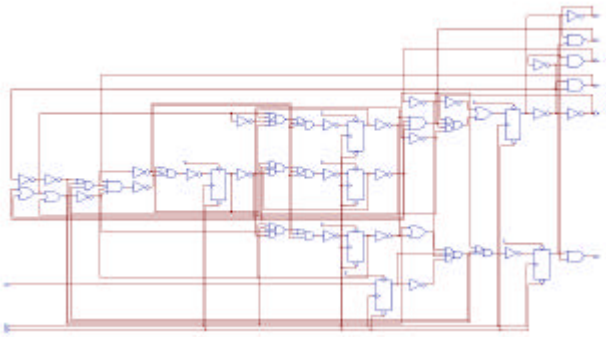


- 2개의 Sub-Block으로 구성됨을 확인
  - 회로도의 Block에 대한 Label
  - VHDL Block 구문의 Label 일치

실습-6 : Traffic Light Controller의 1-Level 논리 합성

- 1-Level 논리 회로 구성을 위한 Script File과 논리 합성 결과

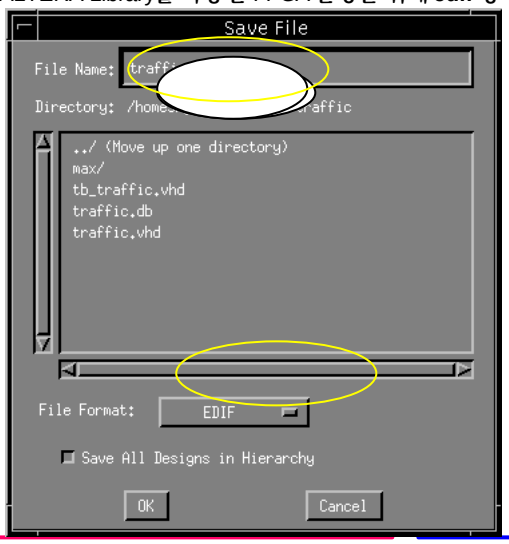
```
read -f vhdl traffic.vhd
create_clock clk
max_delay 0 -to all_outputs()
compile -map_effort high
create_schematic
write -f db
write -f edif -hierarchy -o traffic.edf
```



- 7개의 Flip-Flop과 Random Logic으로 구성됨을 확인

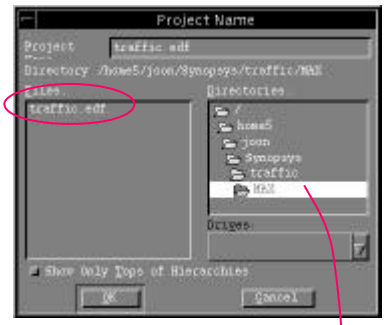
실습-6 : Traffic Light Controller VHDL의 FPGA 구현

- Synthesis 된 결과를 저장 (실습 NO-33과 동일)
  - ALTERA Library를 사용한 FPGA 환경을 위해 **edif** 형식으로 저장



실습-6 : Traffic Light Controller VHDL의 FPGA 구현

- 실습자 개인 directory에 MAX 라는 Sub-directory를 만든다
- 생성된 traffic.edf 화일을 MAX directory로 옮긴다
- MAX directory에서 Max+plus를 실행한다.
  - 실행명령어 : max2win&
- Project의 설정
  - File -> Project -> Name 에서 traffic.edf를 선택
- Device 설정
  - 설계 사양에 알맞은 Device를 선택한다.
    - 이번 설계실습에서는 FLEX 8000 시리즈 중 EPF8282LC84-3을 사용하기로 한다
  - Assign -> Device 에서 EPF8282LC84-3 선택

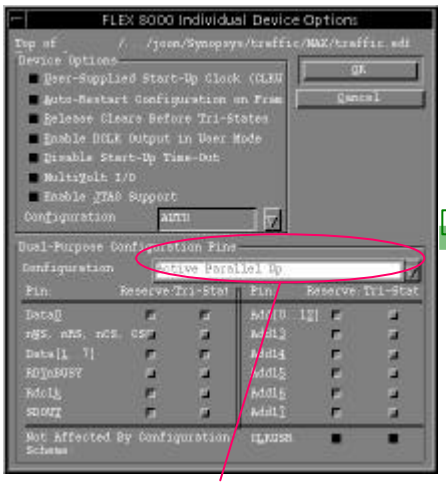


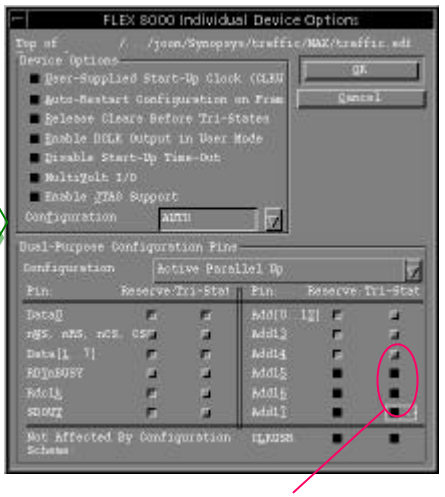
실습자의 MAX directory로 이동

선택되어 있으면 버튼을 해제한다

실습-6 : Traffic Light Controller VHDL의 FPGA 구현

- 오른쪽 버튼 중 Device Options를 선택





- Configuration 설정은 Active Parallel Up 선택
- Add15,16,17을 두 번씩 클릭하여 선택을 해제

실습-6 : Traffic Light Controller VHDL의 FPGA 구현

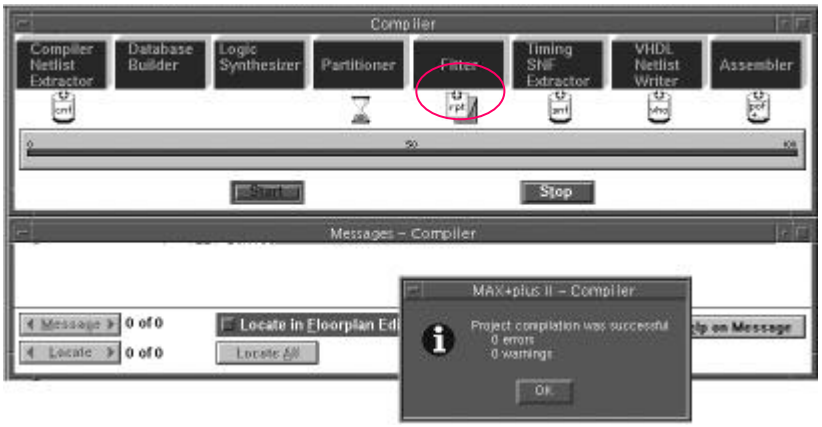
- Compiler 실행
  - Max+plus II -> Compiler 선택 후
    - Processing -> Preserve All Node Name Synonyms 선택
      (1 bit 이상의 다수의 Node를 같은 Name으로 지정하게 한다)
    - Interfaces -> Edif Netlist Reader Settings 선택

Vendor를 Synopsys 선택

- Interfaces -> VHDL Netlist Writer Settings 선택
  선택되어 있는지 확인
- Interfaces -> VHDL Netlist Writer 선택
  (실제 Gate Delay를 고려한 Timing Simulation을 위한 .vho 파일 생성)

실습-6 : Traffic Light Controller VHDL의 FPGA 구현

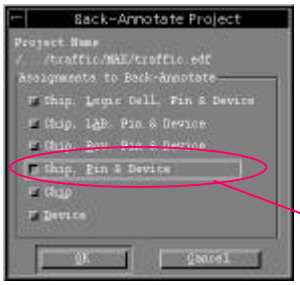
- Start 버튼을 눌러 Compiler를 실행한다

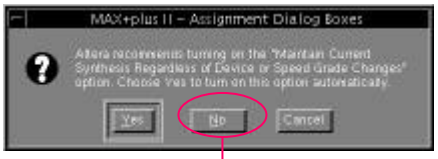


- error가 없음을 확인하고 빠져 나온다
  - 만일 error 가 체크되면 해당 내용을 살펴본 후 수정하도록 한다
- rpt 파일을 두 번 클릭하여 compile 결과와 그 외 여러 가지 내용을 확인해 보도록 한다

실습-6 : Traffic Light Controller VHDL의 FPGA 구현

- Pin Re-Fitting
  - rpt 파일 내용을 통해, compile 후 핀 배치가 이루어짐을 알 수 있다
  - 이번 설계에서는 이미 구성된 FPGA 보드를 통해 구현하여야 하므로, 핀 배치를 재구성할 필요가 있다
  - Max+plus II -> Compiler 선택 후, Assign -> Back-Annotate Project 를 선택





check후 OK를 누른다 NO 를 선택(주의할것)

  - Hierarchy Display를 선택 -> acf 파일을 두 번 클릭하여 활성화 시킨다
    - FPGA Chip 내 의 여러 가지 핀 정보들이 나와 있음을 알 수 있다
    - 자세히 확인하여, acf 파일 내에서 In, Out port의 핀에 관한 정의 부분을 찾는다

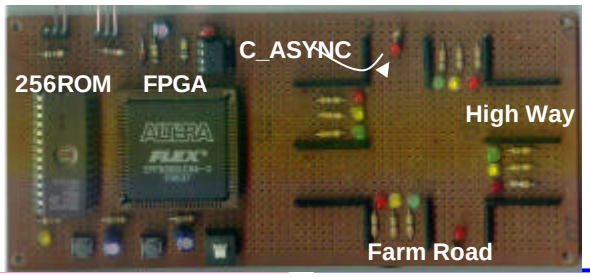
실습-6 : Traffic Light Controller VHDL의 FPGA 구현

- 해당 부분을 아래와 같이 수정한 후, 저장한다

```
| C_ASYNC :    INPUT_PIN = 73
| CLK :        INPUT_PIN = 12;
| RESET :      INPUT_PIN = 31;
| FG :         OUTPUT_PIN = 30;
| FR :         OUTPUT_PIN = 56;
| FY :         OUTPUT_PIN = 25;
| HG :         OUTPUT_PIN = 24;
| HR :         OUTPUT_PIN = 28;
| HY :         OUTPUT_PIN = 16;
```

- 다시 한번 compiler 를 실행시킨 후, Max+plus II를 종료한다
- MAX directory 내 의 traffic.hex 파일을 27C256 ROM에 Download하여 구성된 FPGA

  테스트 보드를 통하여 동작여부를 확인한다.



실습-6 : FPGA를 이용한 Prototyping과 검증

Traffic Light Controller에 대하여 FPGA를 이용한 검증 Board의 예