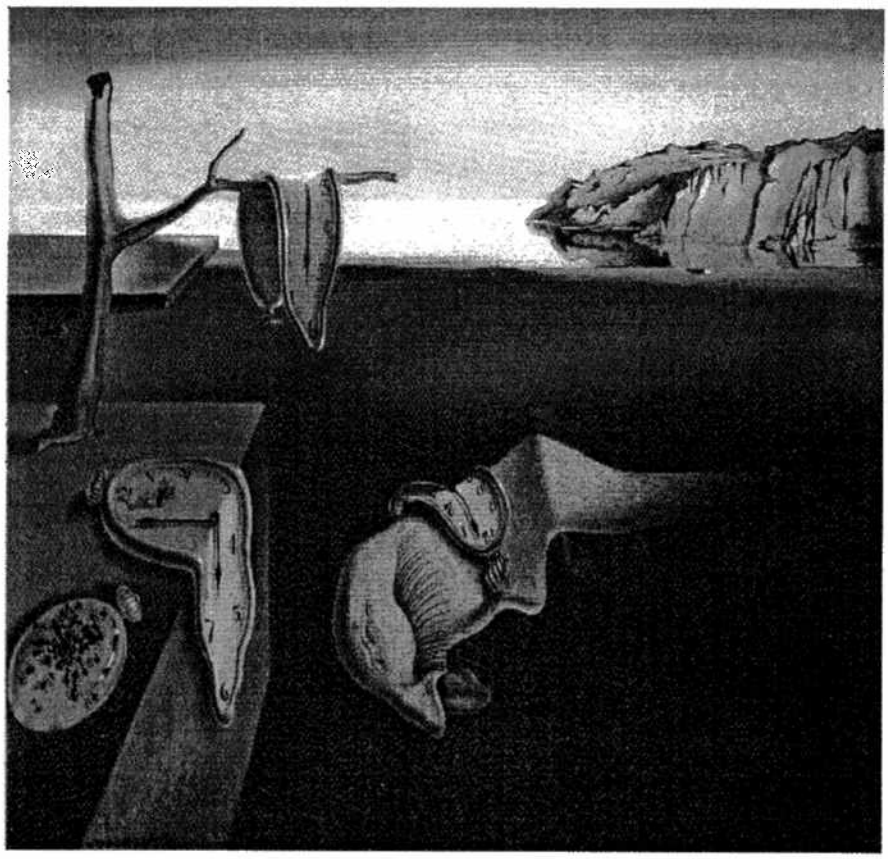
Institut für Automation
Abt. für Automatisierungssysteme

Technische
Universität
Wien

Projektbericht Nr. 183/1-68
October 1996

A Primer to Digital Design with Synopsys and Cadence

Martin Horauer



Salvador Dali, "Die Beständigkeit der Erinnerung"

Notes

Notes

Contents

Notes

Notes

Chapter 1

Synopsys

Here we will give a brief description of the basic programs and shells, that can be used in conjunction with an ASIC development. I will not cover the use of the schematic entry tool `sge`, that is (thats just another prejudice of mine) preferably useful for interconnect management at the top level.

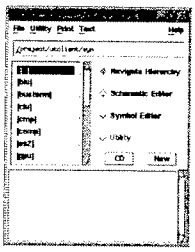


Figure 1.1: SGE fronted after invocation

If you're intending to make use of this tool, see the online documentation for an extensive description.

1.1 Synopsys documentation

The large collection of the Synopsys manuals can be read and searched online via the very powerful inter-leaf viewer. Type **iview** at the Unix command prompt to bring it up onto your screen. Figure 1.2 shows the outline after invocation. Very powerful context search is available via **Search** $\rightarrow$**Collection**. This documentation is an excellent guide to further work, so I'll recommend it for your use not only when you're a novice.
Furthermore there is a collection of books available at our department, that are primarily related to HDL coding [R.94], [P.90], [J.92], [HYE95], [R.93], [D.91], [IEE87], [IEE93], [PT95] and [S.96].

1.2 Analyzing source files

The program **vhdlan** can be used to analyze the VHDL-files for correct syntax.

```
% vhdlan -nc vhdl/test_ent.vhd vhdl/test_beh_arch.vhd
```

Notes

Notes

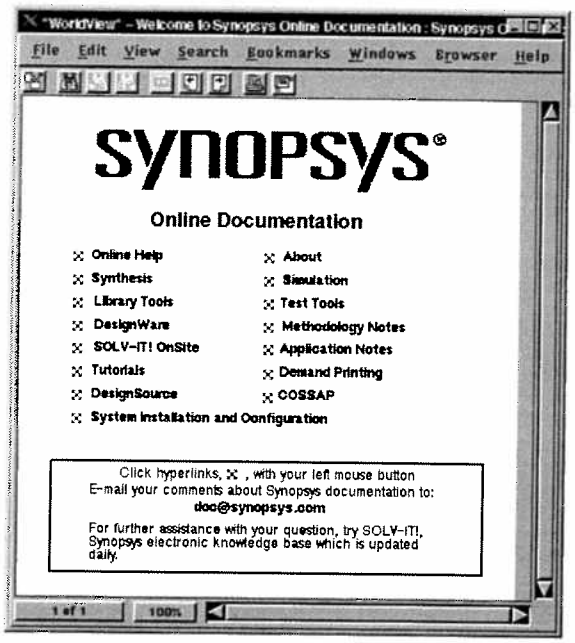
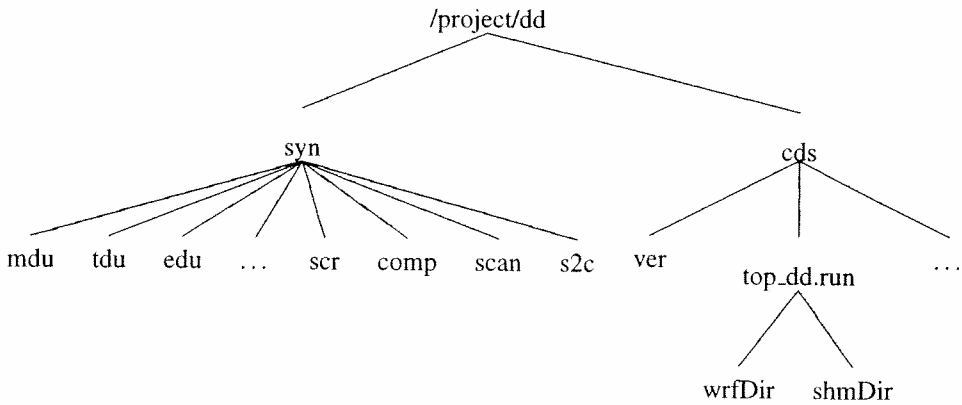


Figure 1.2: Synopsys online documentation

The *-nc* option avoids just an annoying header displayed at every invocation. Beside vhdlan has many other useful options and switches. See the online help *iview* or type **vhdlan --help** to get more information on them. When you're dealing with several source files that are depending on each other you can generate a *Makefile* using the command **simdepends**. Once you've got you're Makefile, you 'll only need to enter **make**, and all depending VHDL-source files, that were modified more recently, are analyzed again. In the case when the analysis of your code succeeds, you can start with your simulation. Otherwise the lines and locations of the error prone code fragments are displayed. You'll have to identify the error an rerun the analysis.

1.3 Directory hierarchy and file-naming policy

To ease file manipulation and navigation through a project, everybody in the design team should follow some general naming and structure conventions. The following directory tree serves as an example how to structure the design.



The *cds* directory holds all Cadence related files, while *syn* all Synopsys related. In the

Notes

Notes

Synopsys tree the subdirectories *mdu*, *tdu* and *exu* hold the VHDL-source files for the three modules of the design. The directory *scr* contains all related script files for synthesis and simulation, under *comp* all synthesized files are stored and *scan* holds all files after scan-path insertion. Finally the contents of directory *s2c* are made up with several files necessary for the design transfer to Cadence (physical place and route).
In the Cadence tree, many subdirectories are instantiated by the tools and therefore you should avoid to manipulate to much in there. Use the tools supplied with Cadence instead.

filename	description
dd_pkg.vhd	project package declaration
dd_pbdy.vhd	project package body
exu_ent.vhd	top-module entity
exu_beh_arch.vhd	top-module architecture
exu_beh_cfg.vhd	top-module configuration
exu_body_ent.vhd	sub-module entity (*body* varies)
exu_body_beh_arch.vhd	sub-module architecture

Table 1.1: Filenaming policy

Table 1.1 summarizes a possible filenaming convention. It is of course a bit tedious to split everything, but benefits of a highly modular design should always be kept in mind. Beside, every file should contain a header, that includes the project name, the name of the designer, the version number, the filename itself, the title and type of the module, the tools it is targeted for, which libraries and packages are used and of course a timestamp. A few of these header lines are of course obvious and could be taken from the file properties itself, but they are nice to have when included also in the header. For examples consult the source-code listings in the further sections.

1.4 Simulation before synthesis

Within Synopsys either the graphical interface **vhdldbx** (see Figure 1.3) or the command line version **vhdlsim** can be used. In both cases you need to specify the name of the configuration you're intending to simulate. Afterwards you'll have to specify the signals you're willing to trace. The later can also be specified in an include file that is applied to the simulator during invocation. You can single-step the code, set/delete breakpoints, evaluate signals and variables, set them to specific values, interrupt the simulation, ... In the waveform window you can add several cursors to measure timings between different signals, zoom in and out, ... Furthermore you can implement file i/o, to ease comparison of your simulation results at different levels of simulation. The later can in addition also be performed via a post-processor program **gpp**. Gpp takes two *wif* files (file-extesion .ow), and compares them at periodic times you specify at the command line. A *wif* file is automatically created when the variable WAVEFORM is set to *wif+waves* within the *.synopsys_vss_setup* configuration file.

The simulator is usually invoked with an include file, which consists of several control sequences for the simulator itself. The following gives an idea of such a file. It traces several signals in different levels of the hierarchy and displays them in the waveform window.

Notes

Notes

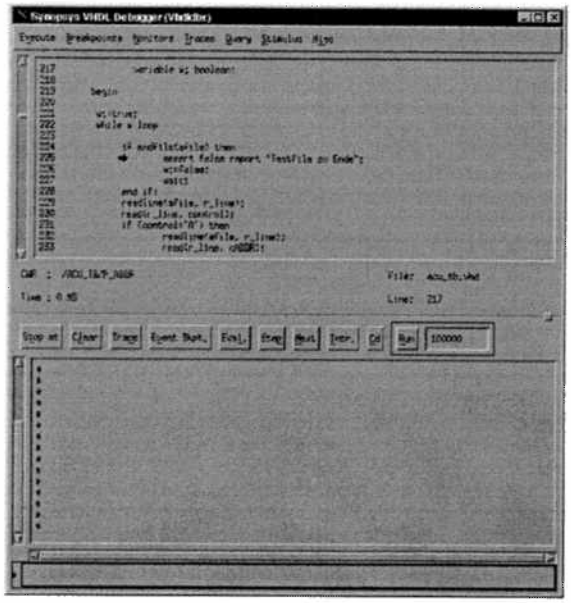


Figure 1.3: The graphical user interface for simulation

```
trace /EXU_TB/TOP/ITCLK
trace /EXU_TB/TOP/ADDR
trace /EXU_TB/TOP/BE
trace /EXU_TB/TOP/RWB
trace /EXU_TB/TOP/CSB
trace /EXU_TB/TOP/RESET
trace /EXU_TB/TOP/UITU/UTCCONF1
trace /EXU_TB/TOP/UITU/UTCSTAT1
trace /EXU_TB/TOP/UITU/UITUBODY/UTCINTSTAT1
trace /EXU_TB/TOP/UITU/UITUBODY/UTCINTEN1
...
```

An analogue file for the post-processor could be for e.g. as the following.

```
load cmp/EXU_TB_LTU_PRE.ow cmp/EXU_TB_LTU_POST.ow
files
timespec t1=40530,40530,repeat 500 times+40000 end
show t1
compare file=1;timespec=t1 all to file=2;timespec=t1 all
...
```

Here two *wif* files, that must be created previously with the simulator, are loaded first. The *files* command displays all files in memory. The *timespec* command declares a periodic point in simulation time, starting at time 40530 and repeating 40000 times every 500 time increments. The resolution for the time increments (ns, ps, ...) is determined via the setup-file *.synopsys_vss_setup*.

1.5 Synthesis

The programs for synthesis are the graphical user interface **design_analyzer** (see Figure 1.4) and the command line shell **dc_shell**. Both have the same functionality, although the shell version is of course more appropriate for most synthesis runs, while the graphical interface can be used primarily for cross-checkings and is easier to use by new users. Both programs have many options and commands built-in. A few of them will be illustrated a little along with some scripts we'll illustrate along with the following sections.

Notes

Notes

To get more information on specific commands just type `help <topic>` at the shell prompt or within the command window.

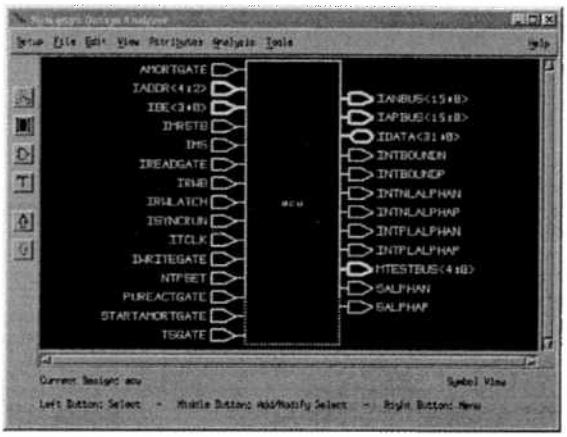


Figure 1.4: The graphical user interface for synthesis

To illustrate the funcionality a bit more in depth, take a look at the following example.

```
PACKAGES={pkg/ExU_pkg.vhd pkg/ExU_pbdy.vhd}
analyze -f vhdl PACKAGES
/* BTU */
BTU_VHDL_FILES={btu/nsumadder_ent.vhd btu/nsumadder_beh_arch.vhd \
btu/btu_body_ent.vhd btu/btu_body_beh_arch.vhd \
btu/btu_mdmux_ent.vhd btu/btu_mdmux_beh_arch.vhd \
btu/btu_mbiu_ent.vhd btu/btu_mbiu_beh_arch.vhd \
btu/btu_ent.vhd btu/btu_beh_arch.vhd}
analyze -f vhdl BTU_VHDL_FILES
elaborate btu
...
current_design{NSUMADDER}
set_max_delay 28 all_outputs()
compile -ungroup_all
...
```

First a dc_shell variable PACKAGES is defined that consists of two files. They are analyzed in the subsequent line. Within this package you could have defined several constants or functions that are used from within several other source files. The subsequent variable holds all VHDL-files of the module in a bootom up fashion. These files are analyzed and elaborated. In the case this operation succeeds, the module is ready to accept several constraints for logic synthesis. Afterwards a submodule NTPADDER, that consits of pure combinational logic is synthesized in front of the rest. Therefore you've to set the current design to this submodule. Then the outputs of this submodule are constrained with a maximum output delay of 28 ns. – Use a realistic value for constraining, otherwise you'll end up with an unnessecarily huge bulk of logic and in addition you'll waste much processing power of your workstation. – The submodule then is synthesiszed and ungrouped, so that you'll end up with a flat logic.

If the module has registered inputs and outputs the following sequence would be an appropriate constraining scenario.

Notes

Notes

```
current_design{btu}
clock_name = ITCLK
create_clock clock_name -period 40 -waveform 0 20
set_dont_touch_network {clock_name}
set_operating_conditions "IND_MAX"
set_wire_load "1407X1407_with_routing" -library ecpd07_ind
set_test_methodology full_scan
set_scan_style multiplexed_flip_flop
set_input_delay 6 find(port,"IANBUS") -clock clock_name
set_input_delay 10 find(port,"ISYNCRUN") -clock clock_name
set_input_delay 12 find(port,"TSGATE") -clock clock_name
set_input_delay 10 find(port,"CSUMSTROBEPURE") -clock clock_name
...
set_output_delay 30 find(port,"ITESTSEL") -clock clock_name
...
...
set_load 0.2 all_outputs()
set_max_fanout 1 all_inputs()
remove_attribute find(port,"ITCLK") max_fanout
set_driving_cell -cell LIBINV -library ecpd07_ind all_inputs()
set_driving_cell -none find(port,"ITCLK")
compile -incremental_map
write -f vhdl -hier -o btu.vhd
write -f db -hier -o btu.db
report_area > btu_area.rep
report_timing > btu_timing.rep
...
```

After the current design is set, a variable *clock_name* is defined with the name of the clock signal. In this case the name of the clock signal is *ITCLK*. Then a clock constraint is instantiated, that defines the clock with a period of 40 ns and the rising edge set to 0 ns and the falling to 20 ns. The next constraint *set_dont_touch_network* prevents the instantiation of buffers within this network. – For e.g. *ES2* uses only one large buffer, strong enough to drive the whole clock tree. Such a buffer is usually instantiated manually after synthesis. – Then according to the technology library you're syntesizing for, you'll have to set *operating conditions* and provide a *wire load model* for area estimation of the interconnections. The next two commands are used if you're intending to insert scan-path logic after synthesis. They set some restrictions for the optimisation of sequential cells to ease the scan-path insertion afterwards. – In the case, you're willing to omit scan insertion, leave these commands aside. – Then set input and output delay constraints in relation to the clock edge to all appropriate signals. – The *set_input_delay* defines the delay of the path to an input. This value is the total time a signal takes to propagate through logic in front of the input port. In contrast the *set_output_delay* value is the delay through the logic hooked to the output port. The common time reference is the rising edge of the clock signal. Include library setup time and instance-specific clock skew in this calculation. – Use the *set_load* command with library dependent typical values to constrain the output ports, and the *set_max_fanout* for the input ports. An inverter is than specified as a *driving cell* for the according inputs. Again these attributes are removed from the clock network, because it should be considered special. Finally compile the module. The switch *incremental_map* takes care of the already precompiled submodule. With the *write* command you are able to save the results in different formats, e.g. vhdl, verilog or the Synopsys internal database format (.db). After synthesis you usually create several reports to verify if the synthesized results meet your desired requirements.

If you're unaware of the meaning or usage of certain commands, you can type **help** <**command**> at the dc_shell prompt or contact the online documentation **iview** from the Unix prompt.

1.6 A design example

To illustrate the design flow and to make it a bit more clear and concise imagine the following example displayed in figure 1.5. This example consists of two submodules, the EXU_BODY, that holds the core functionality of the module, and a submodule called EXU_MDMUX, that is split from the body, due to the fact that it is for e.g. reused within other modules. The block diagram entails the interconnect lines to other modules, and

Notes

Notes

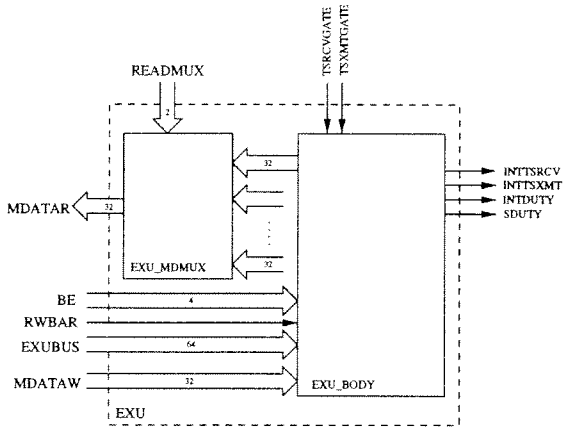


Figure 1.5: A design example: **Ex**ample **U**nit (EXU)

gives a rough impression on the internal structure of the module.

```
-----------------------------------------------------------------------------
--   I C T                              Computer Technology Dept
--                                      University of Technology, Vienna
--                                      All Rights Reserved
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exu_ent.vhd
--   title     :  Example UNIT
--   module    :  entity
--   tools     :  Synopsys
--   ref. lib. :  IEEE
--   ref. pkg. :  1164
--
--   Roadmap   :
--   Date      :  10/96
-----------------------------------------------------------------------------
-- description:
--~~~~~~~~~~~~
-- This file containts the top of the EXU.
--
--           >>>>>   EXU
--                     |
--                 +---+----+
--                 |        |
--            exu_body exu_mdmux
--
--
---------------------------------------------------------------------------

library IEEE;
     use IEEE.std_logic_1164.all;


entity exu is

     port(
  ADDR: in std_logic_vector(1 downto 0);
  RWBAR: in std_logic;
  BE: in std_logic_vector(3 downto 0);
  CLK: in std_logic;
  RESET: in std_logic;
          MDATAR: out std_logic_vector(31 downto 0);
          MDATAW: in std_logic_vector(31 downto 0);
          READMUX: in std_logic_vector( 1 downto 0);
          EXUBUS: in std_logic_vector(63 downto 0);
          TSRCVGATE: in std_logic;
          TSXMTGATE: in std_logic;
          INTTSRCV: out std_logic;
          INTTSXMT: out std_logic;
          INTDUTY: out std_logic;
          SDUTY: out std_logic
         );
end exu;
```

This file which holds the entity ot the EXU top module, is saved in the directory *exu* under the filename *exu_ent.vhd*.

```
-----------------------------------------------------------------------------
--   I C T                              Computer Technology Dept
--                                      University of Technology, Vienna
--                                      All Rights Reserved
```

Notes

Notes


```
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exu_beh_arch.vhd
--   title     :  Example UNIT
--   module    :  architecture
--   tools     :  Synopsys
--   ref. lib. :  IEEE
--   ref. pkg. :  1164
--
--   Roadmap   :
--   Date      :  10/96
-------------------------------------------------------------------------------
-- description:
---------------
--
--           >>>>>   EXU
--                    |
--               +----+---+
--               |        |
--           exu_body exu_mdmux
--
--
-------------------------------------------------------------------------

architecture BEHAVIORAL of exu is

     signal MDATABR0: std_logic_vector( 31 downto 0);
     signal MDATABR1: std_logic_vector( 31 downto 0);
     signal MDATABR2: std_logic_vector( 31 downto 0);
     signal MDATABR3: std_logic_vector( 31 downto 0);

COMPONENT exu_body
  port(
       EXUBUS: in std_logic_vector(63 downto 0);
       MDATAW: in std_logic_vector(31 downto 0);
       RESET: in std_logic;
       CLK: in std_logic;
       ADDR: in std_logic_vector(1 downto 0);
       BE: in std_logic_vector(3 downto 0);
       RWBAR: in std_logic;
       INTTSRCV: out std_logic;
       TSRCVGATE: in std_logic;
       TSXMTGATE: in std_logic;
       INTTSXMT: out std_logic;
       INTDUTY: out std_logic;
       SDUTY: out std_logic;
       MSRCV : out std_logic_vector(31 downto 0);
       TSRCV : out std_logic_vector(31 downto 0);
       MSXMT : out std_logic_vector(31 downto 0);
       TSXMT : out std_logic_vector(31 downto 0)
      );
end COMPONENT;

COMPONENT exu_mdmux
  PORT(
          MDATABR0: in  std_logic_vector(31 downto 0);
          MDATABR1: in  std_logic_vector(31 downto 0);
          MDATABR2: in  std_logic_vector(31 downto 0);
          MDATABR3: in  std_logic_vector(31 downto 0);
  MDATAR:   out std_logic_vector(31 downto 0);
  READMUX:  in  std_logic_vector( 1 downto 0)
      );
end COMPONENT;

begin

   UEXUMDMUX: exu_mdmux port map(MDATABR0,MDATABR1,MDATABR2,MDATABR3,
                                 MDATAR,READMUX
                                );

   UEXUBODY: exu_body port map(EXUBUS,MDATAW,RESET,
                               CLK,ADDR,BE,RWBAR,INTTSRCV,
                               TSRCVGATE,TSXMTGATE,
                               INTTSXMT,INTDUTY,SDUTY,
                               MDATABR0,MDATABR1,MDATABR2,MDATABR3
                              );
end BEHAVIORAL;
```

The architecture of the EXU top module is saved as *exu/exu_beh_arch.vhd*. Finally the configuration is stored as *exu/exu_beh_cfg.vhd*.

```
-------------------------------------------------------------------------------
--   I C T                          Computer Technology Dept
--                                  University of Technology, Vienna
--                                  All Rights Reserved
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exu_beh_cfg.vhd
--   title     :  Example UNIT
--   module    :  configuration
--   tools     :  Synopsys
--   ref. lib. :  -
--   ref. pkg. :  -
--
--   Roadmap   :
--   Date      :  10/96
-------------------------------------------------------------------------------
-- description:
---------------
-- This file containts the top of the EXU.
```

Notes

Notes

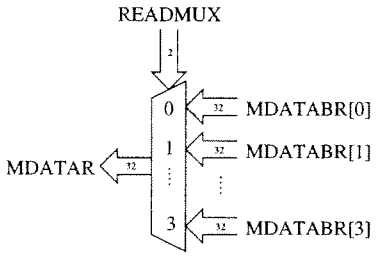


Figure 1.6: Schematic of the submodule EXU_MDMUX

```
--
--          >>>>>   EXU
--                   |
--               +---+----+
--               |        |
--           exu_body   exu_mdmux
--
--------------------------------------------------------------------------

configuration CFG_EXU of exu is
  for BEHAVIORAL
for UEXUMDMUX: exu_mdmux use entity work.exu_mdmux(BEHAVIORAL);
end for;
for UEXUBODY: exu_body use entity work.exu_body(BEHAVIORAL);
end for;
  end for;
end CFG_EXU;
```

Now that the top elements are specified, lets work out the interior of the submodules. Figure 1.6 entails the schematic of the module EXU_MDMUX, which consists only of one large multiplexer, that feeds one of four 32-bit wide buses, depending on the value of the lines READMUX to the output MDATAR. For larger modules a table should summarize the functionality, such as provided with the following submodule, to ease understanding and to write down a functional mapping. The following code saved in *exu/exu_mdmux_ent.vhd* holds the entity.

```
--------------------------------------------------------------------------
--   I C T                              Computer Technology Dept
--                                      University of Technology, Vienna
--                                      All Rights Reserved
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exupkg_mdmux_ent.vhd
--   title     :  Modul Data Bus Multiplexer
--   module    :  entity
--   tools     :  Synopsys
--   ref. lib. :  IEEE
--   ref. pkg. :  1164
--
--   Roadmap   :
--   Date      :  10/96
--------------------------------------------------------------------------


library IEEE;
     use IEEE.std_logic_1164.all;
     USE work.exupkg.all;


entity exu_mdmux is
     port(
          MDATABR0: in  std_logic_vector(31 downto 0);
          MDATABR1: in  std_logic_vector(31 downto 0);
          MDATABR2: in  std_logic_vector(31 downto 0);
          MDATABR3: in  std_logic_vector(31 downto 0);
  MDATAR:   out std_logic_vector(31 downto 0);
  READMUX:  in  std_logic_vector( 1 downto 0)
        );
end exu_mdmux;
```

And in addition *exu/exu_mdmux_beh_arch.vhd* specifies the architecture.

```
--------------------------------------------------------------------------
--   I C T                              Computer Technology Dept
--                                      University of Technology, Vienna
--                                      All Rights Reserved
```

Notes

Notes

```
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exu_mdmux_beh_arch.vhd
--   title     :  Modul Data Bus Multiplexer adopted for EXU
--   module    :  architecture
--   tools     :  Synopsys
--   ref. lib. :  IEEE
--   ref. pkg. :  1164
--
--   Roadmap   :
--   Date      :  10/96
-------------------------------------------------------------------------------


architecture BEHAVIORAL of exu_mdmux is

     begin
-------------------------------------------------------------
-- MDATAMUX
--+++++++++
-- inputs:  MDATABRx, READMUX
-- outputs: MDATAR
-- description: Data Multiplexer
-------------------------------------------------------------

P_READMUX: process(READMUX,MDATABR3,MDATABR2,
                   MDATABR1,MDATABR0)
begin
-------------------------------------------------------
-- TSXMT
-------------------------------------------------------
          if (READMUX = "11") then
   MDATAR <= MDATABR3;
-------------------------------------------------------
-- MSXMT
-------------------------------------------------------
          elsif (READMUX = "10") then
   MDATAR <= MDATABR2;
-------------------------------------------------------
-- TSRCV
-------------------------------------------------------
          elsif (READMUX = "01") then
   MDATAR <= MDATABR1;
-------------------------------------------------------
-- MSRCV
-------------------------------------------------------
          elsif (READMUX = "00") then
   MDATAR <= MDATABR0;
          else
                MDATAR <= ZERO32;
  end if;
end process;

end BEHAVIORAL;
```

The second submodule EXU_BODY consists of an input bus EXUBUS, that provides the EXU with varying data. The registers MSRCV and TSRCV sample the data of this bus on the activation of the gate signal TSRCVGATE, that is exteranlly activated synchronous to the bus. In analogy, the registers MSXMT and TSXMT are sampled by TSXMTGATE. Both signals are additionally fed through this unit (INTTSRCV and INTTSXMT) to a follow up module. All these four 32-bit registers can than be read via a 32-bit access from the address MDATABR[X].

At the bottom, there are two registers DUTYH and DUTYL located, that can be written. The 32-bit DUTYH and the lower 16 bits of DUTYL serve as DUTY input for a comparator module. Bit 17 of register DUTYL enables or disables this comparator, that performs a comparison of 48 bits between EXUBUS[55,8] and the *Duty* value. When the *Exubus* is greater than or equal to *Duty*, a pulse is generated on the output-line INTDUTY, that is active for one clock period, whilst signal SDUTY is active as long as the EXUBUS $\geq$ DUTY.

Element	Width	R/W	ADDR	BE	Reset	description
MSRCV	32	R	0	x	0	Macrostamp receive
TSRCV	32	R	1	x	0	Timestamp receive
MSXMT	32	R	2	x	0	Macrostamp transmit
TSXMT	32	R	3	x	0	Timestamp transmit
DUTYH	32	W	0	3:0	0	DUTYH high
DUTYL	17	W	1	2:0	0	Enable and DUTYL

The following listings belong to the files *exu/exu_body_ent.vhd* and *exu_body_beh_arch.vhd*.

```
-------------------------------------------------------------------------------
--   I C T                          Computer Technology Dept
```

Notes

Notes

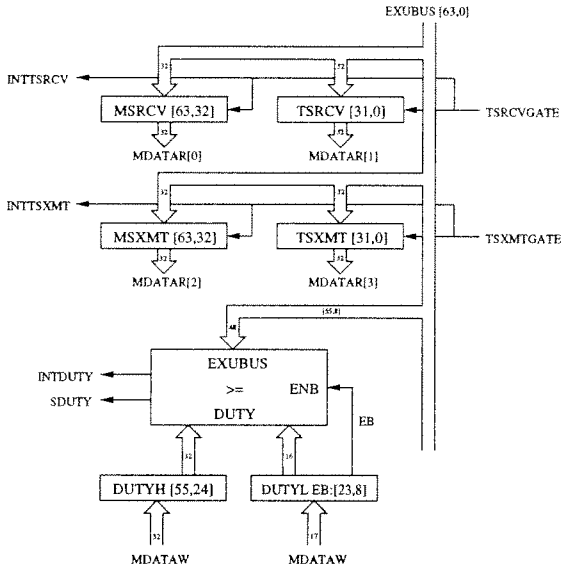


Figure 1.7: Schematic of the submodule EXU_BODY

```
--                                      University of Technology, Vienna
--                                      All Rights Reserved
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exu_body_ent.vhd
--   title     :  Example Unit
--   module    :  entity
--   tools     :  Synopsys
--   ref. lib. :  IEEE
--   ref. pkg. :  1164
--
--   Roadmap   :
--   Date      :  10/96
-------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.">=";

USE work.exupkg.all;


ENTITY exu_body IS
  port(
       EXUBUS: in std_logic_vector(63 downto 0);
       MDATAW: in std_logic_vector(31 downto 0);
       RESET: in std_logic;
       CLK: in std_logic;
       ADDR: in std_logic_vector(1 downto 0);
       BE: in std_logic_vector(3 downto 0);
       RWBAR: in std_logic;
       INTTSRCV: out std_logic;
       TSRCVGATE: in std_logic;
       TSXMTGATE: in std_logic;
       INTTSXMT: out std_logic;
       INTDUTY: out std_logic;
       SDUTY: out std_logic;
       MSRCV : out std_logic_vector(31 downto 0);
       TSRCV : out std_logic_vector(31 downto 0);
       MSXMT : out std_logic_vector(31 downto 0);
       TSXMT : out std_logic_vector(31 downto 0)
      );
END exu_body;


-------------------------------------------------------------------------------
--   I C T                              Computer Technology Dept
--                                      University of Technology, Vienna
--                                      All Rights Reserved
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exu_body_beh_arch.vhd
--   title     :  Example Unit
--   module    :  architecture
--   tools     :  Synopsys
--   ref. lib. :  IEEE
--   ref. pkg. :  1164
--
--   Roadmap   :
--   Date      :  10/96
-------------------------------------------------------------------------------
-- The example unit samples timestamps on external events, as
```

Notes

Notes

```
-- can be transmit or receive pacckets. Receive packets trigger via TSRCV a time-
-- stamp, which is sampled into the MSRCV and TSRCV registers. Transmit packets
-- trigger TSXMT wich samples a timestamp into MSXMT and TSXMT.
-- The duty-timer DUTY is responsible to periodically trigger S/R/D/A-duties.
-- The duty-registers are loaded via two 32-bit wide registers and are enabled via the
-- 17th bit of the lower register. When the EXUBUS>=DUTY an INTDUTY + SDUTY are
-- driven active. EXUBUS<DUTY restores SDUTY w. the consecutive rising edge of CLK.
-- Otherwise SDUTY is restored one clock-period after the falling edge of the enable
-- of the current timer. The INTDUTY is just one period active.
-------------------------------------------------------------------------------

ARCHITECTURE BEHAVIORAL OF exu_body IS


SIGNAL DUTYH : std_logic_vector(31 downto 0);
SIGNAL DUTYL : std_logic_vector(15 downto 0);
SIGNAL EB : std_logic;
SIGNAL LSDUTY : std_logic;
SIGNAL TMPINTDUTY,TMPSDUTY : std_logic;
SIGNAL QBARSDUTY : std_logic;



BEGIN  --  BEHAVIORAL

-------------------------------------------------------------------------------
--  instantiation
-------------------------------------------------------------------------------

    INTTSRCV <= TSRCVGATE;
    INTTSXMT <= TSXMTGATE;
    SDUTY <= LSDUTY;


-------------------------------------------------------------------------------
--  Process:   P_MSRCV
--  ================
--  Purpose:
--  Inputs:  EXUBUS,RESET,MTCLK,TSRCVGATE
--  Outputs: MSRCV
-------------------------------------------------------------------------------

P_MSCRV : PROCESS (EXUBUS,TSRCVGATE,CLK,RESET)
  BEGIN
   IF (RESET='0') THEN
     MSRCV  <= ZERO32;
     ELSE
       IF (CLK='1' AND CLK'event) THEN
         IF (TSRCVGATE='1') THEN
           MSRCV <= EXUBUS(63 downto 32);
         END IF;
       END IF;
   END IF;
END PROCESS P_MSCRV;

-------------------------------------------------------------------------------
--  Process:   P_TSRCV
--  ================
--  Purpose:
--  Inputs:  EXUBUS,RESET,CLK,TSRCVGATE
--  Outputs: TSRCV
-------------------------------------------------------------------------------

P_TSRCV : PROCESS (RESET,CLK,TSRCVGATE,EXUBUS)
  BEGIN
   IF (RESET='0') THEN
     TSRCV <= ZERO32;
   ELSE
     IF (CLK='1' and CLK'event) THEN
       IF (TSRCVGATE='1') THEN
         TSRCV <= EXUBUS(31 downto 0);
       END IF;
     END IF;
   END IF;
END PROCESS P_TSRCV;


-------------------------------------------------------------------------------
--  Process:   P_MSXMT
--  ================
--  Purpose:
--  Inputs:   CLK,RESET,EXUBUS,TSXMTGATE
--  Outputs:  MSXMT
-------------------------------------------------------------------------------
P_MSXMT : PROCESS (CLK,RESET,EXUBUS,TSXMTGATE)
  BEGIN
   IF (RESET='0') THEN
     MSXMT <= ZERO32;
   ELSE
     IF (CLK='1' and CLK'event) THEN
       IF (TSXMTGATE='1') THEN
         MSXMT <= EXUBUS(63 downto 32);
       END IF;
     END IF;
   END IF;
END PROCESS P_MSXMT;


-------------------------------------------------------------------------------
--  Process:   P_TSXMT
--  ================
--  Purpose:
--  Inputs:    RESET,CLK,EXUBUS,TSXMTGATE
--  Outputs:   TSXMT
-------------------------------------------------------------------------------
P_TSXMT : PROCESS (RESET,CLK,EXUBUS,TSXMTGATE)
  BEGIN
   IF (RESET='0') THEN
     TSXMT <= ZERO32;
   ELSE
     IF (CLK='1' and CLK'event) THEN
```

Notes

Notes

```
        IF (TSXMTGATE='1') THEN
          TSXMT <= EXUBUS(31 downto 0);
        END IF;
      END IF;
    END IF;
END PROCESS P_TSXMT;


-----------------------------------------------------------------------------
--  Process:    P_DUTYH
--  ================
--  Purpose:
--  Inputs:     RWBAR,BE,RESET,CLK,MDATAW
--  Outputs:    DUTYH
-----------------------------------------------------------------------------
P_DUTYH : PROCESS (RWBAR,BE,RESET,CLK,MDATAW)
  BEGIN
    IF (RESET='0') THEN
      DUTYH <= ZERO32;
    ELSE
      IF (CLK='1' and CLK'event) THEN
        IF (ADDR="10" and RWBAR='0' and BE="0001") THEN
          DUTYH(7 downto 0) <= MDATAW(7 downto 0);
        ELSIF (ADDR="10" and RWBAR='0' and BE="0010") THEN
          DUTYH(15 downto 8) <= MDATAW(15 downto 8);
        ELSIF (ADDR="10" and RWBAR='0' and BE="0100") THEN
          DUTYH(23 downto 16) <= MDATAW(23 downto 16);
        ELSIF (ADDR="10" and RWBAR='0' and BE="1000") THEN
          DUTYH(31 downto 24) <= MDATAW(31 downto 24);
        ELSIF (ADDR="10" and RWBAR='0' and BE="0011") THEN
          DUTYH(15 downto 0) <= MDATAW(15 downto 0);
        ELSIF (ADDR="10" and RWBAR='0' and BE="1100") THEN
          DUTYH(31 downto 16) <= MDATAW(31 downto 16);
        ELSIF (ADDR="10" and RWBAR='0' and BE="1111") THEN
          DUTYH <= MDATAW;
        END IF;
      END IF;
    END IF;
END PROCESS P_DUTYH;


-----------------------------------------------------------------------------
--  Process:    P_DUTYL
--  ================
--  Purpose:
--  Inputs:  ADDR,RWBAR,BE,RESET,CLK,MDATAW
--  Outputs: DUTYL,EB
-----------------------------------------------------------------------------
P_DUTYL : PROCESS (RWBAR,BE,RESET,CLK,MDATAW)
  BEGIN
    IF (RESET='0') THEN
      DUTYL <= ZERO16;
      EB <= '0';
    ELSE
      IF (CLK='1' and CLK'event) THEN
        IF (ADDR="01" and RWBAR='0' and BE="0001") THEN
          DUTYL(7 downto 0) <= MDATAW(7 downto 0);
        ELSIF (ADDR="01" and RWBAR='0' and BE="0010") THEN
          DUTYL(15 downto 8) <= MDATAW(15 downto 8);
        ELSIF (ADDR="01" and RWBAR='0' and BE="0100") THEN
          EB <= MDATAW(16);
        ELSIF (ADDR="01" and RWBAR='0' and BE="0011") THEN
          DUTYL(15 downto 0) <= MDATAW(15 downto 0);
        ELSIF (ADDR="01" and RWBAR='0' and BE="1100") THEN
          EB <= MDATAW(16);
        ELSIF (ADDR="01" and RWBAR='0' and BE="1111") THEN
          DUTYL <= MDATAW(15 downto 0);
          EB <= MDATAW(16);
        END IF;
      END IF;
    END IF;
END PROCESS P_DUTYL;


-----------------------------------------------------------------------------
--  Process:    P_DUTY
--  ===============
--  Purpose:
--  Inputs:     DUTYH,DUTYL,EB,EXUBUS,CLK,RESET
--  Outputs:    INTDUTY,SDUTY
-----------------------------------------------------------------------------

INTDUTY <= (TMPSDUTY AND QBARSDUTY);
LSDUTY <= TMPSDUTY;


P_DUTY : PROCESS (DUTYH,DUTYL,EB,EXUBUS,CLK,RESET)
  BEGIN
    IF (RESET='0') THEN
      TMPSDUTY <= '0';
    ELSE
    IF (CLK='1' and CLK'event) THEN
      IF ((EXUBUS(55 downto 8) >= (DUTYH & DUTYL)) and EB='1') THEN
        TMPSDUTY <= '1';
      ELSIF ((EXUBUS(55 downto 8) >= (DUTYH & DUTYL)) and EB='0') THEN
        TMPSDUTY <= '0';
      ELSE
        TMPSDUTY <= '0';
      END IF;
    END IF;
  END IF;
END PROCESS P_DUTY;


-----------------------------------------------------------------------------
--  Process:    P_DUTYGATE
--  ==================
--  Purpose:
--  Inputs:     CLK,TMPSDUTY,RESET
--  Outputs:    QBARSDUTY
```

Notes

Notes

```
-----------------------------------------------------------------------------

P_DUTYGATE : PROCESS (TMPSDUTY,CLK,RESET)
  BEGIN
    IF (RESET='0') THEN
      QBARSDUTY <= '0';
    ELSE
    IF (CLK='1' and CLK'event) THEN
      QBARSDUTY <= not(TMPSDUTY);
    END IF;
  END IF;
END PROCESS P_DUTYGATE;


END BEHAVIORAL;
```

The Figure 1.5, the previously description and the Table ?? are a fundamental description that should exist in any case before one starts coding. It eases VHDL-coding drastically, clarifies the the dataflow and gives always a brief overview of the module.

Sometimes in the previous files we've included our own package. This could look like the following one. In the package we declare constants, functions and procedures that are frequently used, whilst their code remains in the package_body section. In our example there are only constants, which can be used either for HDL-coding or for simulation. Note that the type *time* must be hidden from the synthesis tools, therefore use the Synopsys synthesis off and on switches (they are special comments treated seperately) as illustrated.

```
-----------------------------------------------------------------------------
--   I C T                              Computer Technology Dept
--                                      University of Technology, Vienna
--                                      All Rights Reserved
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exupkg_pkg.vhd
--   title     :  -
--   module    :  -
--   tools     :  Synopsys
--   ref. lib. :  IEEE
--   ref. pkg. :  1164
--
--   Roadmap   :
--   Date      :  10/96
-----------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

package exupkg is

   constant ZERO16 :std_logic_vector:= "0000000000000000";
   constant ZERO32 :std_logic_vector:= "00000000000000000000000000000000";

        --synopsys synthesis_off

        constant t_CLK_period : time := 40 ns;
        constant t_half_CLK_period : time := 20 ns;

        constant t_WAITCYCLE: time := 0 ns;            --  x Waitestates (x*CLK)
        constant t_waitcycle_pulse_width: time := 0 ns;

        constant t_ADDR_setup : time :=  5 ns;
        constant t_ADDR_hold  : time :=  5 ns;
        constant t_ADDR_valid : time := (t_ADDR_setup+t_CLK_period+t_ADDR_hold);
        constant t_ADDR_finish: time := (t_CLK_period-t_ADDR_hold);

        constant t_rwb_setup : time := 5 ns;
        constant t_rwb_hold  : time := 5 ns;
        constant t_rwb_valid : time := (t_ADDR_valid+t_rwb_setup+t_rwb_hold);
        constant t_rwb_finish: time := (t_ADDR_finish-t_rwb_hold);

        constant t_data_write_setup : time := 10 ns ;
        constant t_data_write_hold  : time :=  5 ns ;
        constant t_data_write_finish: time := (t_CLK_period-t_data_write_hold);

        constant t_be_setup : time :=  5 ns;
        constant t_be_hold  : time :=  5 ns;
        constant t_be_valid : time :=  (t_ADDR_valid+t_be_setup+t_be_hold);
        constant t_be_finish: time :=  (t_ADDR_finish-t_be_hold);

        constant t_gate_hold  : time :=  3 ns;
        constant t_gate_setup : time :=  t_half_CLK_period;
        constant t_gate_finish: time :=  (t_CLK_period-t_gate_hold);
        --synopsys synthesis_on


end exupkg;

package body exupkg is
end exupkg;
```

Notes

Notes

Now analyze the files with the command:
%vhdlan -nc -spc exu_pkg.vhd exu_mdmux_ent.vhd exu_mdmux_beh_arch.vhd exu_body_ent.vhd exu_body_beh_arch.vhd exu_body_beh_arch.vhd exu_ent.vhd exu_beh_arch.vhd exu_beh_cfg.vhd

The first switch avoids printing a header to the standard output, and the second (*-spc*) forces the analyzer to check the code for fragments that wouldn't be syntesizable. When all errors are diminished, proceed with simulation.


```
-----------------------------------------------------------------------------
--   I C T                              Computer Technology Dept
--                                      University of Technology, Vienna
--                                      All Rights Reserved
--
--   project   :  UTCSU
--   designer  :  Martin HORAUER
--   version   :  1.0
--   file name :  exu_tb.vhd
--   title     :  Example Unit - Testbench
--   module    :  IEEE
--   tools     :  Synopsys
--   ref. lib. :  IEEE
--   ref. pkg. :  1164,arith
--
--   Roadmap   :
--   Date      :  10/96
-----------------------------------------------------------------------------


library IEEE;
     use IEEE.std_logic_1164.all;
     use IEEE.std_logic_arith.all;
---- synopsys translate_off
     use IEEE.std_logic_textio.all;
     use std.textio.all;
---- synopsys translate_on
     USE work.exupkg.all;

entity exu_tb is
end;

architecture tb of exu_tb is

FILE afile: TEXT is in "exu_control.dat";
FILE rfile: TEXT is in "exu_control.dat";
FILE dfile: TEXT is in "exu_control.dat";
FILE bfile: TEXT is in "exu_control.dat";
FILE hfile: TEXT is in "exu_control.dat";
FILE yfile: TEXT is in "exu_control.dat";
FILE ffile: TEXT is in "exu_control.dat";
FILE ifile: TEXT is in "exu_control.dat";
FILE jfile: TEXT is in "exu_control.dat";

        signal ADDR: std_logic_vector(1 downto 0);
        signal RWBAR: std_logic;
        signal BE: std_logic_vector(3 downto 0);
        signal CLK: std_logic;
        signal RESET: std_logic;
        signal MDATAW: std_logic_vector(31 downto 0);
        signal MDATAR: std_logic_vector(31 downto 0);
        signal INTPBUS: std_logic_vector(63 downto 0);
        signal TSRCVGATE: std_logic;
        signal TSXMTGATE: std_logic;
        signal INTTSRCV:  std_logic;
        signal INTTSXMT: std_logic;
        signal INTDUTY: std_logic;
        signal SDUTY: std_logic;
        signal READMUX: std_logic_vector(1 downto 0);


  component exu
     port(
  ADDR: in std_logic_vector(1 downto 0);
  RWBAR: in std_logic;
  BE: in std_logic_vector(3 downto 0);
  CLK: in std_logic;
  RESET: in std_logic;
          MDATAR: out std_logic_vector(31 downto 0);
          MDATAW: in std_logic_vector(31 downto 0);
          READMUX: in std_logic_vector(1 downto 0);
          INTPBUS: in std_logic_vector(63 downto 0);
          TSRCVGATE: in std_logic;
          TSXMTGATE: in std_logic;
          INTTSRCV: out std_logic;
          INTTSXMT: out std_logic;
          INTDUTY: out std_logic;
          SDUTY: out std_logic
          );
   end component;


begin


 example_unit: exu
       port map(ADDR,RWBAR,
                BE,CLK,RESET,MDATAR,MDATAW, READMUX,
                INTPBUS,TSRCVGATE,TSXMTGATE,INTTSRCV,
                INTTSXMT,INTDUTY,SDUTY
               );

P_CLK: process
```

Notes

Notes

```
    begin
         CLK <= '1';
         wait for t_half_CLK_period;
         CLK <= '0';
         wait for t_half_CLK_period;
end process P_CLK;


P_ADDR:PROCESS
variable r_line: line;
variable control: character;
variable cADDR: std_logic_vector(1 downto 0);
         variable w: boolean;

begin
 w:=true;
 while w loop

if endfile(afile) then
assert false report "Testfile zu Ende";
w:=false;
wait;
end if;
readline(afile, r_line);
read(r_line, control);
if (control='A') then
readline(afile, r_line);
read(r_line, cADDR);
ADDR <= "00";
                wait for (t_CLK_period-t_ADDR_setup);
ADDR <= cADDR;
                wait for t_ADDR_valid;
                wait for t_WAITCYCLE;
ADDR <= "00";
                wait for t_ADDR_finish;
else
readline(afile, r_line);
end if;
 end loop;
 wait;
end process P_ADDR;


P_RWBAR:PROCESS
variable r_line: line;
variable control: character;
variable crwb: std_logic;
         variable w: boolean;

begin
 w:=true;
 while w loop
if endfile(rfile) then
assert false report "Testfile zu Ende";
w:=false;
wait;
end if;
readline(rfile, r_line);
read(r_line, control);
if (control='R') then
readline(rfile, r_line);
read(r_line, crwb);
RWBAR <= '1';
                wait for (t_CLK_period-t_rwb_setup);
RWBAR <= crwb;
                wait for t_rwb_valid;
                wait for t_WAITCYCLE;
RWBAR <= '1';
                wait for t_rwb_finish;
else
readline(rfile, r_line);
end if;
 end loop;
 wait;
end process P_RWBAR;


P_DATA:PROCESS

variable r_line: line;
variable control: character;
variable cdata: std_logic_vector(31 downto 0);
         variable w: boolean;

begin

 w:=true;
 while w loop

if endfile(dfile) then
assert false report "Testfile zu Ende";
w:=false;
wait;
end if;
readline(dfile, r_line);
read(r_line, control);
if (control='D') then
readline(dfile, r_line);
hread(r_line, cdata);
                MDATAW <= "00000000000000000000000000000000";
                wait for (2*t_CLK_period+t_WAITCYCLE-t_data_write_setup);
MDATAW <= cdata;
                wait for (t_data_write_setup+t_data_write_hold);
                MDATAW <= "00000000000000000000000000000000";
                wait for t_data_write_finish;
else
readline(dfile, r_line);
end if;
 end loop;
 wait;
end process P_DATA;
```

Notes

Notes

```
P_BE:PROCESS

variable r_line: line;
variable control: character;
variable cbe: std_logic_vector(3 downto 0);
        variable w: boolean;

begin

 w:=true;
 while w loop

if endfile(bfile) then
assert false report "Testfile zu Ende";
w:=false;
wait;
end if;
readline(bfile, r_line);
read(r_line, control);
if (control='B') then
readline(bfile, r_line);
hread(r_line, cbe);
                BE <= "0000";
                wait for (t_CLK_period-t_ADDR_setup);
BE <= cbe;
                wait for t_be_valid;
                wait for t_WAITCYCLE;
                BE <= "0000";
                wait for t_be_finish;
else
readline(bfile, r_line);
end if;
 end loop;
 wait;
end process P_BE;


P_RESET:PROCESS

variable r_line: line;
variable control: character;
variable creset: std_logic;
        variable w: boolean;

begin
RESET <= '1';
 w:=true;
 while w loop

if endfile(hfile) then
assert false report "Testfile zu Ende";
w:=false;
wait;
end if;
readline(hfile, r_line);
read(r_line, control);
if (control='H') then
readline(hfile, r_line);
read(r_line, creset);
RESET <= creset;
                wait for 3*t_CLK_period;
                RESET <= '1';
                wait for t_WAITCYCLE;
else
readline(hfile, r_line);
end if;
 end loop;
 wait;
end process P_RESET;

P_READMUX:PROCESS
variable r_line: line;
variable control: character;
variable creadmux: std_logic_vector(1 downto 0);
        variable w: boolean;

begin
READMUX <= "00";
 w:=true;
 while w loop

if endfile(jfile) then
assert false report "Testfile zu Ende";
w:=false;
wait;
end if;
readline(hfile, r_line);
read(r_line, control);
if (control='J') then
readline(hfile, r_line);
read(r_line, creadmux);
READMUX <= creadmux;
                wait for 3*t_CLK_period;
                READMUX <= "00";
                wait for t_WAITCYCLE;
else
readline(jfile, r_line);
end if;
 end loop;
 wait;
end process P_READMUX;


P_INTPBUS:PROCESS

variable r_line: line;
variable control: character;
variable cINTPBUS: std_logic_vector(63 downto 0);
```

Notes

Notes

```
        variable w: boolean;

 begin


  w:=true;
  while w loop

 if endfile(yfile) then
 assert false report "Testfile zu Ende";
 w:=false;
 wait;
 end if;
 readline(yfile, r_line);
 read(r_line, control);
 if (control='Y') then
 readline(yfile, r_line);
 hread(r_line, cINTPBUS);
                    WAIT for 3 ns;
 INTPBUS <= cINTPBUS;
                    wait for 3*t_CLK_period - 3 ns;
                    wait for t_WAITCYCLE;
 else
 readline(yfile, r_line);
 end if;
  end loop;
  wait;
 end process P_INTPBUS;


 P_TSRCVGATE:PROCESS

 variable r_line: line;
 variable control: character;
 variable ctsrcv: std_logic;
        variable w: boolean;

 begin

  w:=true;
  while w loop

 if endfile(ffile) then
 assert false report "Testfile zu Ende";
 w:=false;
 wait;
 end if;
 readline(ffile, r_line);
 read(r_line, control);
 if (control='F') then
 readline(ffile, r_line);
 read(r_line, ctsrcv);
                TSRCVGATE <= ctsrcv;
                wait for 3*t_CLK_period;
                wait for t_WAITCYCLE;
 else
 readline(ffile, r_line);
 end if;
  end loop;
  wait;
 end process P_TSRCVGATE;


 P_TSXMTGATE:PROCESS

 variable r_line: line;
 variable control: character;
 variable ctsxmt: std_logic;
        variable w: boolean;

 begin

  w:=true;
  while w loop

 if endfile(ifile) then
 assert false report "Testfile zu Ende";
 w:=false;
 wait;
 end if;
 readline(ifile, r_line);
 read(r_line, control);
 if (control='I') then
 readline(ifile, r_line);
 read(r_line, ctsxmt);
 TSXMTGATE <= ctsxmt;
                wait for 3*t_CLK_period;
                wait for t_WAITCYCLE;
 else
 readline(ifile, r_line);
 end if;
  end loop;
  wait;
 end process P_TSXMTGATE;

 end tb;

 configuration cfg_exu_tb of exu_tb is
    for tb
      for example_unit: exu use entity work.EXU(BEHAVIORAL);
      end for;
    end for;
 end cfg_exu_tb;
```


The testbench that is illustrated now should explain a way of coding to make testing a bit more modular and easy to use. It is based on some preassumptions, that when they

Notes

Notes

are met by the design, make it easy to use the testbench with little modifications on all different kinds of modules. The assumption is that you have a periodic bus-cycle and that all signals that don't apply to such a category can be modeled in the same periodic fashion. What follows is a testbench file that reads stimuli data from an ASCII data file. According to the periodic signals, the stimuli are taken from this file only, when they should be valid. In the gaps between, when the values are of no interest for the bus-cycle, they are set to a default value, that is usually related to the idle time. The testbench is structured into several processes, where every process reads simultaneously to all others the stimuli data. The data file consists of a comment line followed by the data. The relevant data for a process checks, if the first letter in a comment line corresponds to the letter that is applied to it. If the comparison is true the following data in the next line is applied to the described models. Therefore to keep all processes simultaneous, within every bus-cycle must exist a data value for every process.


```
A - Address **** RESET **************
00
R - RWB
0
D - MDATAW
22226789
B - Byte Enable
F
H - RESET
0
J - REAMUX
00
Y - INTPBUS
0123ACDE5432F012
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** DUTYH **************
10
R - RWB
0
D - MDATAW
FFFFFFFF
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFFFD00
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** DUTYL **************
01
R - RWB
0
D - MDATAW  ------- EB=1
0A37FFFE
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFFFD00
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** nothing **************
00
R - RWB
0
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFFFE00
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** nothing **************
00
R - RWB
0
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFFFF00
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** DUTYH **************
10
R - RWB
0
D - MDATAW
FFF0FFFF
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFF0000
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** DUTYL **************
01
R - RWB
0
D - MDATAW  ------- EB=1
0A37FFFE
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFFFD00
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** nothing **************
00
R - RWB
0
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFFFE00
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** nothing **************
00
R - RWB
0
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFFFF00
F - TSRCVGATE
0
I - TSXMTGATE
0
```

Notes

Notes

```
A - Address **** DUTYL **************
01
R - RWB
0
D - MDATAW  ------- EB=0
0A30FFFE
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
00FFFFFFFFFFFF00
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** nothing **************
00
R - RWB
0
D - MDATAW
03C10E23
B - Byte Enable
F
L - Big/Little Endian
0
H - RESET
1
J - REAMUX
00
Y - INTPBUS
0000111100001100
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** TSRCV **************
01
R - RWB
1
D - MDATAW
22226789
B - Byte Enable
F
H - RESET
1
J - REAMUX
01
Y - INTPBUS
000000FFFFFFFE00
F - TSRCVGATE
1
I - TSXMTGATE
0
A - Address **** nothing **************
00
R - RWB
0
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
000000AAAAAAAA00
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** TSXMT **************
11
R - RWB
1
D - MDATAW
22226789
B - Byte Enable
F
H - RESET
1
J - REAMUX
11
Y - INTPBUS
0123ACDE5432F012
F - TSRCVGATE
0
I - TSXMTGATE
1
A - Address **** MSRCV **************
00
R - RWB
1
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
00
Y - INTPBUS
0123ACDE5432F012
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** TSRCV **************
01
R - RWB
1
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
01
Y - INTPBUS
0123ACDE5432F012
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** MSXMT **************
10
R - RWB
1
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
10
Y - INTPBUS
0123ACDE5432F012
F - TSRCVGATE
0
I - TSXMTGATE
0
A - Address **** TSXMT **************
11
R - RWB
1
D - MDATAW
03C10E23
B - Byte Enable
F
H - RESET
1
J - REAMUX
11
Y - INTPBUS
0123ACDE5432F012
F - TSRCVGATE
0
I - TSXMTGATE
0
```

Due to the fact that simulation is often a very interactive process, create an include file that holds the relevant commands for the simulator (e.g. which signals to trace). The following could be an example for the previously described EXU.

```
cd /EXU_TB
trace CLK
trace RESET
trace ADDR
trace BE
trace RWBAR
trace MDATAW
trace MDATAR
trace EXUBUS

cd /EXU_TB/EXAMPLE_UNIT/UEXUBODY
trace DUTYH
trace DUTYL
trace EB
trace INTDUTY
```

Notes

Notes

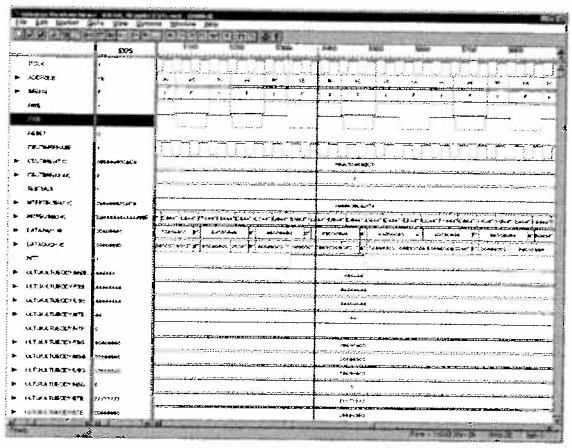


Figure 1.8: The waveform window

```
trace SDUTY

trace TSRCVGATE
trace TSXMTGATE
trace MSRCV
trace TSRCV
```

Now analyze the testbench with the command

%vhdlan -nc exu_tb.vhd

and start the simulator either via

%vhdldbx -i exu_tb.inc CFG_EXU_TB
%vhdlsim -i exu_tb.inc CFG_EXU_TB.

In addition to the commands specified in the include file, you can enter commands at the command prompt (e.g. *run 3000*). The waveform window (see Figure 1.8) is rather intuitive.

When the simulation runs perform as intended, you can continue with synthesis. The major functionalities for synthesis were already described in the prior sections. Therfore the commands of the following script should already be at hand.

Notes

Notes

```
/* targeted for SynopsysV3.4a */
/*****************************/
PACKAGES={exu_pkg.vhd}
analyze -f vhdl PACKAGES

/* EXU */
/* *** */
EXU_VHDL_FILES={exu_body_ent.vhd exu_body_beh_arch.vhd exu_mdmux_ent.vhd
exu_mdmux_beh_arch.vhd exu_ent.vhd exu_beh_arch.vhd}
analyze -f vhdl EXU_VHDL_FILES
elaborate exu
current_design{exu}
clock_name = CLK
create_clock clock_name -period 40 -waveform {0 20}
set_dont_touch_network {clock_name}
set_operating_conditions "IND_MAX"
set_test_methodology full_scan
set_scan_style multiplexed_flip_flop
set_input_delay 6 find(port,"EXUBUS") -clock clock_name
set_input_delay 8 find(port,"TSRCVGATE") -clock clock_name
set_input_delay 8 find(port,"TSXMTGATE") -clock clock_name
set_output_delay 20 find(port,"INTTSRCV") -clock clock_name
set_output_delay 20 find(port,"INTTSXMT") -clock clock_name
set_load 0.2 all_outputs()
set_max_fanout 1 all_inputs()
remove_attribute find(port,"CLK") max_fanout
set_driving_cell -cell LIBINV -library ecpd07_ind all_inputs()
set_driving_cell -none find(port,"CLK")
compile
write -f vhdl -hier -o comp/exu.vhd
write -f db -hier -o comp/exu.db
report_area > comp/exu_area.rep
report_timing > comp/exu_timing.rep

/* SCANpath insertion */
/**********************/
/* In the case you have a bidirectional databus
   create a new level of hierarchy, where only
   the bidirectional function of the data bus is
   described. On top of this file insert scan
   path logic. After scan-insertion you'll
   have to replace the file with the bidir
   information with a file where the pads are
   instantiated. */
set_test_methodology full_scan
set_scan_style multiplexed_flip_flop
test_default_period = 1000.0
test_default_delay = 50.0
test_default_bidir_delay = 550.0
test_default_strobe = 950.0
create_test_clock CLK -waveform {450.0 550.0}
check_test
insert_test -no_disable -max_scan_chain_length 500
check_test > scan/check_test_after_testinsertion.rep
/* generate buffer tree for test_scan_enable signal */
set_max_fanout 1 find(port,"test_se")
compile -only_design_rule
write -f db -hier -o scan/exu_scaned.db
write -f vhdl -hier -o scan/exu_scaned.vhd
check_test
create_test_patterns > scan/patterns.log
/* generate reports on the final design */
/****************************************/
check_design > scan/check_exu.rep
report_area > scan/exu_area.rep
report_cell > scan/exu_cell.rep
report_net > scan/exu_net.rep
report_reference > scan/exu_reference.rep
report_hierarchy > scan/exu_hierarchy.rep
report_timing > scan/exu_timing.rep
report_transitive_fanout -clock_tree  > scan/exu_fanout.rep
report_test -port > scan/exu_ports.rep
report_test -scan_path > scan/exu_scan_path.rep
report_test -coverage > scan/exu_coverage.rep
report_test -faults > scan/exu_faults.rep
report_test -dont_fault > scan/exu_dont_fault.rep
report_test -mask_fault > scan/exu_mask_fault.rep
report_test -assertions > scan/exu_assertions.rep
report_test -atpg_conflicts > scan/exu_atpg_conflicts.rep

/* instantiate the pads */
/************************/
read -f vhdl top_exu.vhd

/* now prepare for CADENCE */
/***************************/
current_design top_exu
verilogout_single_bit = true
symbol_library = {StdLib_cdk.sdb, OscLib_cdk.sdb, PadLib_cdk.sdb,
basic_cdk.sdb, sheets_cdk.sdb, ripper_cdk.sdb }
synthetic_library = standard.sldb
verilogout_no_tri=true
define_name_rules cadence_verilog -allowed "A-Z0-9_()[]";
change_names -rules cadence_verilog -hierarchy
current_design exu_mdmux
ungroup -all
write -f verilog -o s2c/exu_mdmux.v
current_design exu_body_test_1
ungroup -all
write -f verilog -o s2c/exu_body_test_1.v
current_design exu
write -f verilog -o s2c/exu.v
current_design top_exu
write -f verilog -o s2c/top_exu.v
```

Notes

Notes

The first part of this script analyzes, constrains and syntesizes the example unit (EXU). Refer to previous sections for an explanation. Afterwards scan-path-insertion is performed. First the scan methodology is set to *full-scan* and the scan-style is set to *multiplexed flip flop*. The next few lines define the timing related test attributes given by the ES2 documents. The *check_test* command is useful to identify non-scanable sequential cells. Then with the command *insert_test* scan-cells are inserted and the scan-path is instantiated. Afterwards the test select input (*test_se*) is constrained and buffers are instantiated to fix design-rule violations. The *create_test_patterns* instruction calculates test patterns for this path. Next the reports give an overview of the derived results.

Before the modules are made ready for transfer to Cadence, pads should be instantiated. One way would be to write a structural VHDL-source file, that instantiates several VERILOG-files after this stage, that can be read into CAdence via the Verilog-In procedure. –Refer to the next section how to continue then with the layout.– Now before you proceed with Cadence, you should perform a post-synthesis simulation. – Take the last saved design unit (in our case *scan/exu_scaned.db*) and load it into dc_shell:

```
>read -f db scan/exu_scaned.db
>current_design exu_body_test_1
>ungroup -all
>current_design exu
>ungroup -all
>write -f vhdl -o scan/exu_synthesized.vhd
```

Than edit this VHDL-file, to check that the lines for the library inclusion are correct (e.g. *use work.ECPD07.all;*), and the name of the instantiated architecture (e.g. *SYN_BEHAVIORAL*). In general both items are handled via the setup-files and should be already instantiated as intended.

In addition you have to edit your testbench file now to add a process that drives the test-select and test-input lines, the component to match the entity with the newly added test-ports, the port-mapping to the component and finally the configuration at the end of the testbench to point to the correct architecture.

Analyze the VHDL-netlist *exu_synthesized.vhd* and the modified testbench and invoke the simulator the same way you did for simulation at the logic level.

Notes

Notes

Chapter 2

Back-End design with CADENCE

Before you start with the back-end design of your project using CADENCE, make sure you got an own directory from within your project directory available (e.g. /project/< *projectname* >/cds). You can make a soft-link from within your home-directory to ease traversing through the system. Furthermore you need to have two files (or two links) in this directory called *.cdsinit* and *.simrc*. All this should already have been done by your project head, if not contact him.
Before invoking Cadence you must allow the program to display the windows on your terminal. Therefore type `% xhost +`[*terminalname*] within an xterm window. Change to the working directory (e.g. /project/< *projectname* >/cds) and enter `% icfb &`. The command interface window (CIW) is displayed as illustrated in figure 2.1.

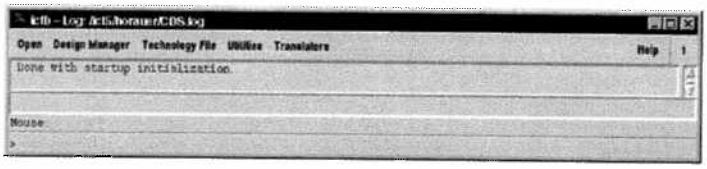


Figure 2.1: The CIW interface

First create a new library that corresponds to the name of your project (e.g. exu) with **Open** ↣**Library** ↩. In the displayed form fill in: Library Name [exu] ↩. You are now prompted to create a new library - select ↩. Then in the next form specify a working and/or user group and apply the according permissions, leave all other items to their default. Open the library browser **Design Manager** ↣**Library Browser** you should see your newly created library.

At the CIW command line prompt enter `load "/project/proto/VerilogIn.il"`. The form for importing Verilog files, with some defaults preset, is displayed in table 2.1. From several items check and adjust the following settings:

The import session can take several minutes up to several hours - stay patient. – The CIW reports many Warnings and Errors that should be checked carefully. – After the import process has finished you should have at least one schematic view in your design library, which you should open (e.g. via the Library Browser). Within the schematic view of your top-level design perform a **Check and Save** in every level of the hierarchy in a

Notes

Notes

Target Library Name	exu
Reference Libraries	StdLib PadLib basic
Verilog Design Files	/project/exu/syn/s2c
Verilog Options	/users/ict2/staff/es2lib/ecpd07/utl/verilog/dllib_ind −y /project/dd/exu/syn/s2c +libext+.v
Verilog Cell Modules	♦Import
View	♦Schematic

Table 2.1: The VerilogIn form

bottom up manner. Before doing so at the top instance, perform the last final changes as instantiating and connecting an oscillator cell, a power-on reset and all the other elements that make up the final design. Furthermore you require for e.g. a cell called LIBTOPNETS, that adds information for the supply nets, and some pads for both supply of core and peripheral. How many and what kind of pads have to be used therefore is described in [ES294] on page 3-3 ff.

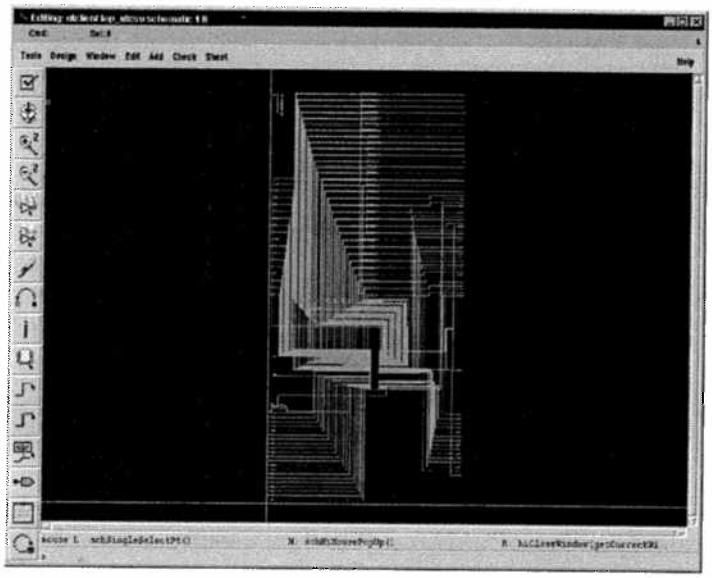


Figure 2.2: The Schematic View

When you have fixed everything and done the final 'check and save' operation, bring up the hierarchy browser via **Tools** ↣**Floorplan/Schematics** and **Floorplan** ↣**Hierarchy Browser**. For a flat design, click on the top instance and perform **Hierarchy** ↣**Generate Physical Hierarchy**. The views *autoLayout* and *autoAbstract* are created.

For a hierarchical design, the above action should be repeated for every level of hierarchy - therefore expand the top element **by instance**. Alternatively you can also open the flat *autoLayout* and use the browser to cross select the instances of each hierarchy and perform a **Create** ↣**Softblock**. Anyway don't *regroup* the hierarchy due to the fact that CADENCE would rename several nets. Then you would no longer be able to run an LVS check later on.

The previously generated *autoLayout* can now be opened into Cell Ensemble by selecting **Tools** ↣**Floorplan/P&R** ↣**Cell Ensemble**. As a first step load the net information for routing (track widths and separation of distinct nets).

Notes

Notes

Route ⟶Modify Net ⟶Net Properties File
♦/users/ict2/staff/es2lib/ecpd07/utl/startup/netData
♦read

This file contains information for the nets *vdd!* and *gnd!*. Now add glue cells such as LIBCORNER with **Place ⟶Glue Cell ⟶Add**. – To move the cells out of the placement region perform **Floorplan ⟶Reinitialize**. Then make *Region* selectable in the Object Selection Window (OSW) and click on the default region, that was created after reinitialisation. Perform **Floorplan ⟶Analyze Floorplan Objects** and enter the desired number of rows (use an even number preferably) and a percentage for the desired row utilization. Both values require some experience and must be derived in an interactive manner. (e.g. 90% row utilization) – Keep in mind that the router expands primarily up and a bit on both sides but not down! – Drag the placement region in a way that the routing channels between the standard cell rows become a bit smaller in height than the standard cells. Therefore considering these aspects, the width of the placement region should be greater than the height in case you have a square core area in mind. Now adjust the design outline to the default region and perform a second reinitialisation with only the *instance status* left on.

Load the floorplan file **Place ⟶I/O commands ⟶Read Initial File** to place the pads around the placement region. When the file is correct all but the LIBCORNER pads should be moved to their desired places. In some cases the placer swaps a few pads, therefore control the placement carefully. Than move [m] and rotate [F3] the corner cells to their locations and use the command **Edit ⟶Align Cells** to position them and change the property [q] from *unplaced* to *placed*.

```
|VDDPY1            left     1
|PIN_TEST_SE       left     2
|PIN_TEST_SI1      left     3
|..
|GNDPY1            bottom   1
|PIN_DATAI_0       bottom   2
|PIN_DATAI_1       bottom   3
...
|PIN_CSUTIME13     right   23
|VDDPY5            right   24
|GNDPY2            top      1
|PIN_CSUTIME14     top      2
...
```

The example on the left shows how such a placement file should be constructed. The first column lists the pin names, the second the position and the third the order they are positioned. A counter clockwise placement is performed.

Start the placement of the standard cells with **Place ⟶Automatic**

♦Insert Feedthru
Feedthru Library Name: **StdLib**
Feedthru Master Name: **LIBFEED**
Placement Snap Grid: **0.1** (Ecpd07)

Sometimes after placement of the standard cells you have to fix up the position of the corner cells again. Now bring up the menu for the power cells **Place ⟶Power Cell ⟶Add Manual**. Depending on the size of the placement region you add either only cap cells on each end of a row, or when the length of a row exceeds some values, influenced by the operating frequency (see [ES294] on page 3-6), you'll have to add power bars. For the later take care that the cells have all the same x-positions after placement,

Notes

Notes

otherwise the routing becomes worse. In the **Define Power Cell** menu select LIBLCAP and LIBRCAP for the cap cells. For powerbar grid routing add also LIBPGB40 and LIBPGFILL and disable the distribute feedthru option. Usually the later two should be omitted in most cases ↩. Now draw two rectangles over the left and right side of the standard cell rows and select ↩. Power cells should be inserted, if not correct the placement and repeat the above step. Figure 2.3 gives a glimpse of an exemplary

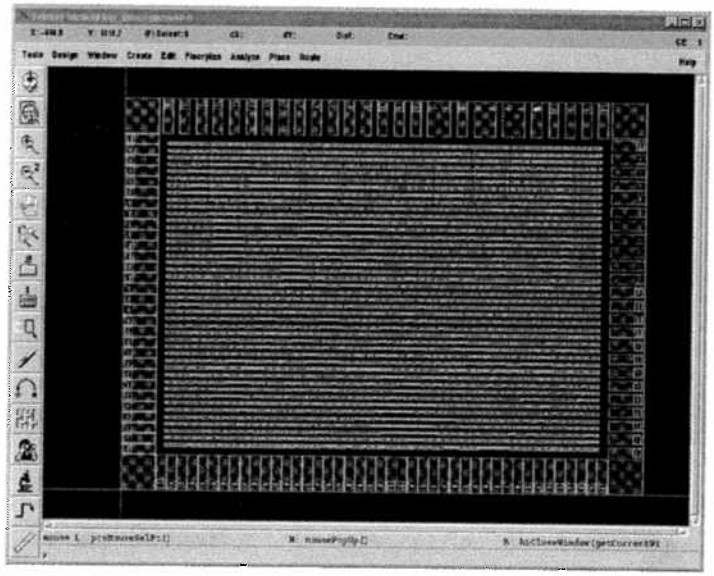


Figure 2.3: A placed design

placement.

Now select *all* instances and do a **Place** ↦**Snap to Grid** with a value of 0.1.

Placement is now finished and can be saved as for e.g. *placed.*

The next step is **Route** ↦**Channels** ↦**Create** and to modify the net properties of dedicated nets, such as for e.g. the clock net, via **Route** ↦**Modify Net** ↦**Modify Net Properties**. – To determine the width and separation of the clock net see [ES294] on page C-6. – Set the preferred layer to CME2, the signal type to *clock* and the criticality to 105.
Then invoke interactive global route on the clock and supply nets. **Route** ↦**Global Route** ↦**Interactive Global Route**, select **Initialize Net** and enter a net name for global routing (e.g. |Clock, vdd!, gnd!). With **Settings** you can enlarge the *snap points* from 1 to say 19. You should recognize all available snap points for the selected net. You can select those you want to connect either by clicking on them or via the **Select Scan Chain** menu. – Take care of the order you select points, because it can influence the desired routing order. – When you've all desired nets selected, **Connect Set** will tie them together. Via **Modify Set** you can apply a routing width and a preferred layer for the selected segments. Therefore if you want different widths of segments (e.g. the main clock trunk is in general far thicker than the clock net within a channel), you've to modify those separate parts individually. Before you push the **Exit Net** button, the modified snap points should be selected, otherwise you can get malicious routing results. On exiting the net, select the **Fix Global Route** option and to remove all the snap points. If you're prompted for an edge connected attribute, you've obviously

Notes

Notes

forgotten to connect some instances.
After all special nets are global routed, you can exit the **Interactive Global Route** menu by hitting the **Done** button. Now perform global routing **Route** ⟶**Global Route** ⟶**Automatic** ↩. This takes a while - take a look at the CIW. Then continue with **Route** ⟶**Detail Route** ⟶**Automatic**. Choose the **Compact** button to set the routing grid to 0.1 microns and start the detail router. If errors occur they are prompted in the CIW whilst routing. Then save the design under a meaningful name (e.g. layout), as shown in figure 2.4.

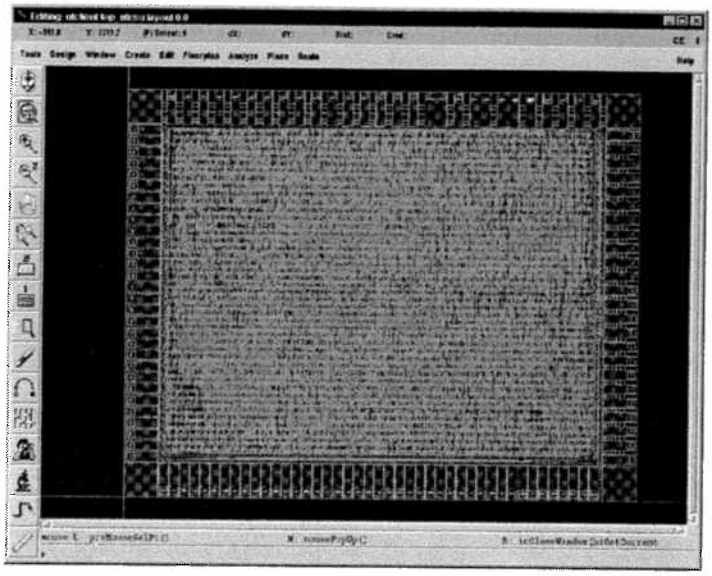


Figure 2.4: The final layout

Now that placement and routing is finished, finalize the chip by performing a DRC and LVS run followed by a final sign-off simulation.
Open the previously saved view and change to the layout interface **Tools** ⟶**Layout**. In the CIW window select **Technology File** ⟶**Compile Technology** and specify

/users/ict2/staff/es2lib/ecpd07/utl/startup/diva.PR♦load

In the layout view choose **Verify** ⟶**DRC** in full and flat mode, with the switches *grid* and *correct* set. In the case that errors are reported, you can view them via **Verify** ⟶**Markers** ⟶**Find**.

TIP: In the layout view you can modify segments easily by changing the values of the geometries you get from the properties window. Modification takes place on a solely segmented basis, therefore if you move a segment it is disconnected from the rest. In contrast the cell ensemble view remains the segments tied to each other when modified.

If you're not able to select a wire, you must first descend into the desired module/channel and perform the check there once more to locate the same errors again. If you want to circumfere the required steps of traversing through the hierarchy, you can explode the channels from within the cell-ensemble view by **Route** ⟶**Detail Route** ⟶**Explode Channels**, which will promote the channels to the upper hierarchy level.

Notes

Notes

In the case when the DRC checker reports zero faults proceed with **Verify** ⟶**Extract** and select ♦**Macro Cell** as extract method. When succeeding a view called extracted is created, that is used for the LVS run.
Prior to the LVS run compile the technology file *diva.lvs*, that resides under the same directory as the one before for the DRC run. Then start **Verify** ⟶**LVS** to compare the schematic versus the extracted view. In the case the run succeeds check the log file if the two views match; – if not search the directories LVS/schematic and LVS/layout for files with a .out extension (e.g. mergenet.out). These files contain information about the nets that caused the mismatch. (e.g. A merged net in the schematic is mostly caused by a short in the layout.) Open the layout in cell ensemble and highlight the conflicting nets with **Edit** ⟶**Search**. Zoom into the regions where these nets cross and search for violations. Correct them and run the DRC, EXTRACT and LVS again. Hopefully you've corrected everything without introducing new errors, and the netlists now match. – Save the design!

How to perform a post layout simulation? Within the cell ensemble view select all global nets with |.* and perform an **Analyze** ⟶**Parasitics** ⟶**Extract** on them. Than make sure that you have enough swap space available, before succeeding with the next step **Analyze** ⟶**Parasitics** ⟶**Write reduced SPF** with the option *physical name mapping* ♦. The name of the SPF file should be < *top_cellname* > *.spf*. At the CIW command prompt enter > **ES2generateSDF**. This brings up a form, where you have to enter the library name, the name of the top_cell and which kind of SDF (min, typ or max) to generate. When you select all three, you'll end up with six files (three sdf files and three reports with an extension .out). Check the reports carefully whether you're violating fanout and fan-in rules or not. If you got violations cross check them with the library data book.

NOTE: If you connect two clock buffers in parallel and the load is greater than the one a buffer can drive, you'll see a warning. Check if the given load is less than the added load of the buffers.

Now lets start with simulation. Open from within the schematic via **Tools** ⟶**Simulation** the **Verilog-XL** environment. Change **Setup** ⟶**Record Signals** from *Top Level I/O* to *All Signals*. Enter at the CIW prompt > **ES2simTemplate** and leave all options to their default values. Change to the simulation run-directory and edit the generated file *ES2testfixture.v*. At the bottom of the file you'll see a line for inclusion of an SDF file (top.sdf) and an inclusion of another verilog file (default_stim.v) that supplies the user provided stimuli. An example of such a stimuli file is provided in appendix **??**. Change the name to point to your file. What is still missing, is a file top.sdf in the simulation directory that has valid back-annotation information. Therefore enter at the CIW > **ES2sdfTranslation**. This copies the appropriate SDF file into the directory, generates a report that should be checked, and produces a soft-link called top.sdf that points to the real SDF file.

TIP: If you have connected two clock buffers in parallel, check that the back-annotated SDF values for the buffer delays don't differ too much, otherwise your simulation run won't be successful. Make the two delays for CKBUF1 and CKBUF2 equal.

Now press the **Start Interactive** button and simulation will start unless you haven't introduced any error in a previous stage. When simulation ends bring up the waveform viewer via **Debug** ⟶**Utilities** ⟶**View Waveform**. The simulation results that are stored below *shmDir/shm.db* in the run-directory is loaded automatically. You can now add signals of different levels to the waveform window by **Edit** ⟶**Browser/Display Tool**. When all desired signals are on screen save the setup for other/later sessions. After

Notes

Notes

you've checked the simulation (see figure 1.8) rename the directory *shm.db* and edit the previously saved setup-file to point there. This will avoid that the simulation run can be overwritten by follow-up runs.

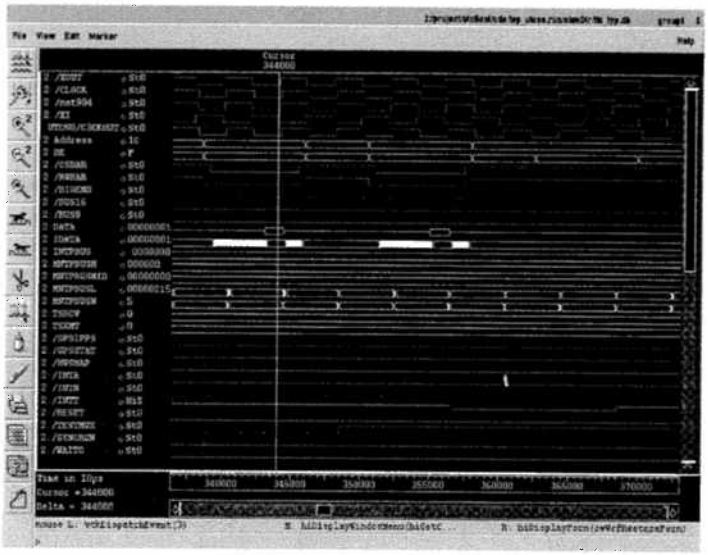


Figure 2.5: The Waveform window

When everything 's fine you can create a *gds2* file out of the final layout view via **Translators →Stream Out**from the CIW. Fill in the working directory, choose Stream DB, Library Version 5.0, microns, append a layer map table (see appendix **??** for details) and activate in the options form the items report bad polygons and precision report. In the end you can load the script `>load "/project/proto/retr.il"`. Then evaluate these script via `>eval (retr())` to get some statistics and reports of the final layout.

Notes

Notes

Bibliography

[D.91] Perry D. *VHDL.* McGraw Hill, 1991.

[ES294] *ES2 ECPD07 Library Handbook, ES2 Process-Independent Library Handbook, ES2 Asic Design Handbook*, 1994.

[HYE95] Liu J. Hsu Y., Tsai K. and Lin E. *VHDL Modelling for Digital Design Synthesis.* Kluwer Academic Publishers, 1995.

[IEE87] *IEEE Standard VHDL Language Reference Manual - Std 1076-1987*, 1987.

[IEE93] *IEEE Standard VHDL Language Reference Manual - Std 1076-1993*, 1993.

[J.92] Basker J. *A VHDL Primer.* Prentice Hall, 1992.

[P.90] Ashden P. *The VHDL Cookbook.* available in postscript on the Internet, South Australia, 1990.

[PT95] Kurup P. and Abbasi T. *Logic Synthesis Using Synopsys.* Kluwer Academic Publishers, 1995.

[R.93] Lipsett R. *VHDL: Hardware Description and Design.* Kluwer Academic Publishers, 1993.

[R.94] Ariau R. *Circuit Synthesis with VHDL.* Kluwer Academic Publishers, 1994.

[S.96] Palnitkar S. *Verilog HDL, A Guide to Digital Design and Synthesis.* Prentice Hall, 1996.