

Behavioural Codes of Clock Integrated Circuit

By

Mohammad Rafi Bin Dan

Dissertation

Submitted in partial fulfilment of
the requirements for the
Bachelor of Electrical and Electronics Engineering (Hons)

June 2007

Universiti Teknologi PETRONAS

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

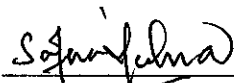
Behavioural Codes of Clock Integrated Circuit

by

Mohammad Rafi Bin Dan

A project dissertation submitted to the
Electrical and Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL AND ELECTRONICS ENGINEERING)

Approved by,



(Ms. Salina Mohmad)

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



(MOHAMMAD RAFI BIN DAN)

ABSTRACT

We are now submerging into a world where big-board-circuits been replaced by integrated circuits. So to goes into this race, I had been given a task to develop a Very-Large-Scale Integrated (VLSI) Circuit that is capable to run and display a real clock on various displays such as computer monitor or any specific display. This built I.C. will receive a clock signal wave from a wide range of frequencies and able to compute exact time up to milliseconds. It shall able not just to display correct local time and date but also can be reset and display time and date for various locality. The first important step is to master and develop the behavioural code of this clock system. There are two choices of codes, which are Verilog and Verilog Hardware Description Language (VHDL). Only then this develop system will be implemented to the Field Programmable Gate Array (FPGA). A FPGA is a semiconductor device containing programmable interconnections. This FPGA should be connected to a computer clock simulator and run the Clock Integrated Circuit built.

ACKNOWLEDGEMENT

The author would like to thank God for revealing some of His knowledge to the author in the course of this project. The successful implementation and completion of this project has been made possible through the help and support of many individuals. First and foremost is the author's supervisor, Ms. Salina Mohmad, for his guidance have enabled the author to understand the subject matter at hand, and overcoming all the obstacles along the way. Thanks also to the Electrical & Electronics Engineering Department lecturers for their teachings, and last but not least, many thanks to the author's course mate for all their input during this project. The author also likes to give his appreciation to his family, for their continuing support for him, and their understanding. They are the main motivator for the author in his studies. Finally the author would like to thank Electrical & Electronics Engineering Department of University Teknologi PETRONAS and all those who have help him in one way or another to make this Final Year Project a success.

TABLE OF CONTENTS

CERTIFICATION OF APPROVAL-----	ii
CERTIFICATION OF ORIGINALITY-----	iii
ABSTRACT-----	iv
ACKNOWLEDGEMENT-----	v
TABLE OF CONTENTS-----	3
Chapter 1 INTRODUCTION-----	5
1.1 Background-----	5
1.2 Problem Statement-----	6
1.3 Objectives and Scope of Duty-----	6
Chapter 2 LITERITURE REVIEW AND THEORY-----	8
Chapter 3 METHODOLOGY/PROJECT WORK-----	11
Chapter 4 RESULTS AND FINDING-----	12
Chapter 5 CONCLUSION-----	20
REFERENCES-----	21
APPENDICES :	
Appendix I : Code for Centisecond's module.	
Appendix II : Code for Second's module.	
Appendix III : Code for Minute's module.	
Appendix IV : Code for Hour's module.	
Appendix V : Code for Day's module.	
Appendix VI : Code for Month's module.	
Appendix VII : Code for Alarm's module.	
Appendix VIII: Code for Stopwatch's module.	
Appendix IX : Code for Time Zone Conversion's module.	

List of figures:

Figure 4.1	: Module blocks for clock counter -----	12
Figure 4.1.1	: Waveform of centisecond's module at the initial value -----	14
Figure 4.2.1	: Waveform of second's module at the initial until clear = '1'-----	14
Figure 4.2.2	: Waveform of second's module at counter reached 60 -----	15
Figure 4.3.1	: Waveform of minute's module at the initial-----	15
Figure 4.3.2	: Waveform of minute's module when counter reached 60-----	16
Figure 4.4.1	: Waveform of hour's module at the initial -----	16
Figure 4.4.2	: Waveform of hour's module when counter reached 24-----	17
Figure 4.5.1	: Waveform of day's module -----	17
Figure 4.6.1	: Waveform of month's module -----	18
Figure 4.7.1	: Waveform of alarm's module-----	18
Figure 4.8.1	: Waveform of stopwatch's module-----	19

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND OF STUDY

An FPGA is a semiconductor device containing programmable logic components and interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders or simple math functions. In most FPGAs, these programmable logic components also include memory elements, which may be simple flip-flops or more complete blocks of memory. A hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected as needed by the system designer. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer so that the FPGA can perform whatever logical function is needed. To define the behaviour of the FPGA, users are provided with a Hardware Description language (HDL) or schematic design. Common HDLs are Verilog and VHDL. Then these behavioural languages would be compiled and simulated using the simulator or compiler such as Xilinx Project Manager, ModelSim, Quartus or many more appropriate simulators/compilers available. Then using an electronic design automation tool, a technology-mapped netlist is generated. This netlist can then be fitted to the actual FPGA architecture using a process called place-and-route. Once this procedure had been performed, they can not be reconfigured to fit the system.

1.2 PROBLEM STATEMENT

Technologies were developed to fulfil human desires. Since inventions started to play a big roll in human life, they are still developing until now and much more useful inventions keep being invented. One of these inventions is clock and it is an extremely useful device in human daily life. There are so many type of clock available in the market with variable sizes and functions. However, what attract costumers more nowadays are the multifunction products and this become the competitive point for all manufacturer worldwide.

1.3 OBJECTIVES & SCOPE OF WORKS

1.3.1 Objectives

Objectives of this project are to develop behavioural codes for a VLS Integrated Circuit that is capable to run and display a real clock on a various type of displays such as seven-segment, computer monitor and many other types of display. This clock will be able to compute exact time up to centisecond and shall display correct time and date for various locality. It also can be reset at any time and has additional function such as stopwatch and alarm. Successfully behavioural codes should create an FPGA implementation and can be further develop as an integrated circuit for a multifunction digital watch that stored data of time and date for various localities.

1.3.2 Scope of Work

Student are given two semesters to work with this project and hope that this time frame will be enough to develop the Hardware Description Languages needed for clock integrated circuit. Below is the scope of work for the project within these two semesters for the project:

First Semester

- a) To do a literature study about the behavioural languages (Verilog and VHDL).
- b) To develop the behavioural code.

Second Semester

- a) To develop full behavioural code of Clock Integrated circuit:
 - i) Basic clock's modules (centisecond, second, minute, hour).
 - ii) Date's module (day, month).
 - iii) Stopwatch's module.
 - iv) Alarm's module
 - v) Time zone's module.
- b) Simulation of the behavioural codes.

CHAPTER 2

LITERATURE REVIEW & THEORY

In the beginning, hardware designers were programmers and vice versa. The world of hardware design and software design fragmented into separate camps during the 1950s and 1960s as advancing technology made software programming easier [1]. The industry needs many more programmers than hardware designers and programmers require far less knowledge of the physical machine than hardware designers. Despite this, the role of software designers and hardware designers is essentially the same; solve a problem. Although many hardware designers realized in the 1960s and 1970s that their primary job was to develop an algorithm that solves a problem and translate that algorithm into hardware, some hardware designers lost sight of this essential truth. An early notation for describing digital hardware that provides tremendous clarity in this regard is the Algorithmic State Machine (ASM), which was invented in early 1960s by T.E. Osborne [1]. As the name suggests, the ASM notation emphasizes the algorithmic nature of the machines being designed.

Unfortunately, hardware designers were inundated with the overwhelming technological changes that occurred with semiconductor electronics. Many hardware designers lost track of the advances in design methodology that occurred in software. Around 1980s, as semiconductor technology advanced, it got more and more difficult to design hardware. Designers realized that the ever-increasing power of general-purpose computer could be harnessed to aid them in designing the next generation of chips. The goal of using the current generation of general-purpose computers to help design the next generation of special and general-purpose computers required bringing the worlds of hardware and software back together again.

Out of this union was born the concept of the Hardware Description Language (HDL). Being a computer language, an HDL allows use of many of the timesaving software methodologies that hardware designers had been lacking [1]. But as the computer language, the HDL allows the expression of concepts that previously could only be expressed by manual notations, such as the ASM notation and circuit diagrams.

Common HDLs are VHDL and Verilog. Although there are lot of similarity between those two, but each of them had they own advantages. VHDL stands for the VHSIC Hardware Description Language and VHSIC refers to the Very High-Speed Integrated Circuit program. This VHDL was developed in 1981, initiated by Department of Defense (DoD) of United States aimed to develop a new generation of high-speed integrated circuit [7]. At the early stage, continuous advances in semiconductor technologies increased the complexity of digital system and were found to have a fundamental impact on the economics of the design of military and space electronic systems. Furthermore it became more difficult to share designs of subsystems across contractors. Finally standardized representation of digital system became the prior concern. The contract to develop this language was awarded to a DoD team and this resulting a first version of VHDL, released in 1985. Subsequently this language was transferred to the IEEE for standardization and further development by representatives from industry, government and academe. Subsequently the language was ratified in 1987 and became the IEEE 1076-1987 standard [3].

At the other hand, Verilog was started initially as a proprietary hardware modelling language by Gateway Design Automation Inc. around 1984. It is rumoured that the original language was designed by taking features from the most popular HDL language of the time, called HiLo, as well as from traditional computer languages such as C. At that time, Verilog was not standardized and the language modified itself in almost all the revisions that came out within 1984 to 1990. In 1990, Cadence recognized that if Verilog remained a closed language, the pressures of standardization would eventually cause the industry to shift to VHDL. Consequently, Cadence organized the Open Verilog International (OVI), and in 1991 gave it the

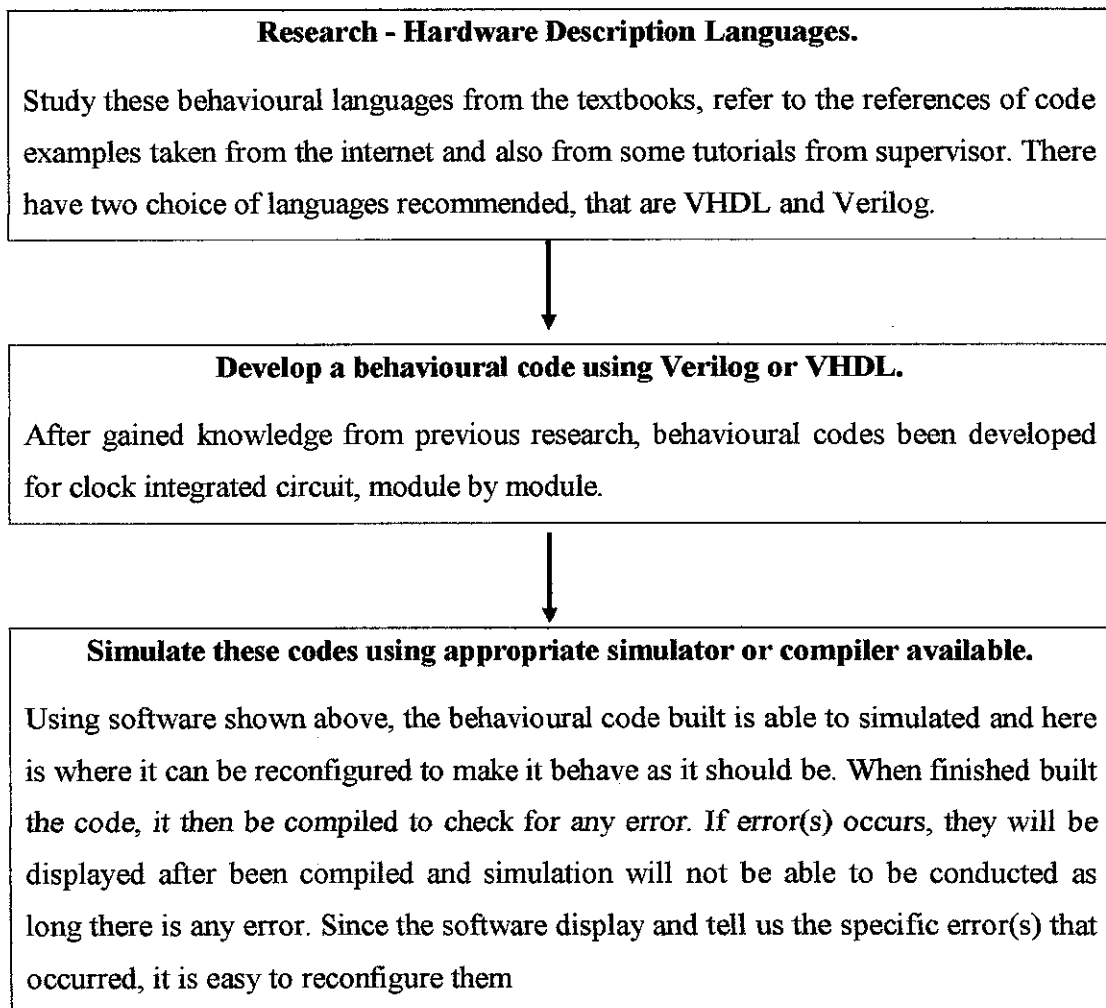
documentation for the Verilog Hardware Description Language. This was the event which "opened" the language [8].

Conventional semiconductor technology will someday reach its limit (based on the minimum size of transistor and the speed of light). Technologies based on recombine DNA, photonics, quantum mechanics, superconductivity and nanomechanics are all contenders to be the computer technology of the twenty-first century [1]. The point is that it does not matter; technology changes every day, but the concepts endure.

CHAPTER 3

METHODOLOGY/PROJECT WORK

There are certain steps to be followed while managing this project and they are as shown below:



CHAPTER 4

RESULTS AND FINDINGS

Base on literature review and some tutorials with supervisors throughout these two semesters, this project had come out with module blocks for Centisecond's, Second's, Minute's, Hour's, Day's, Alarm's, Stopwatch's and last, the Time Conversion's module. These blocks are interconnected within each other and will affect each other changes in counter. These module blocks are as shown below:

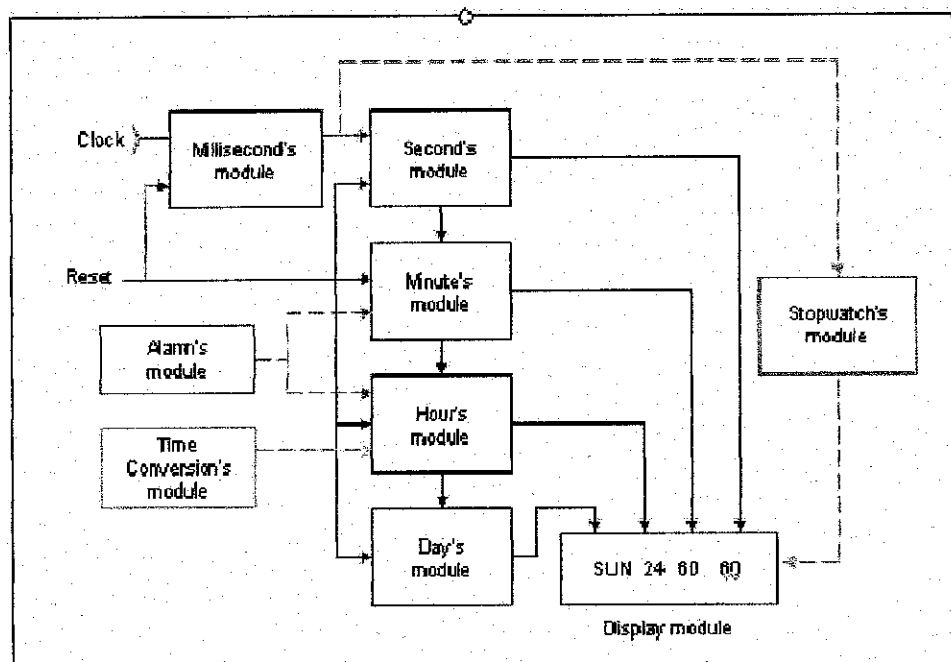


Figure 4.1: Modules' blocks for clock counter.

On top of everything, the counter is initiated by clock signal. The signal must have a time, $t = 1$ centisecond for each cycle, so from the t given, we will get the value for the frequency, f by using equation (1), (2) and (3)

$$f = 1/t \quad \text{--- (1)}$$

$$f = 1/1 \times 10^{-2} \quad \text{--- (2)}$$

$$f = 100 \text{ Hz} \quad \text{--- (3)}$$

Every time the clock signal trigger from 0 to 1, the Centisecond's module will count for 1 until it reaches 100. Once it reaches 100, it will reset to 0 again and gives the Second's module a count '1'. This situation keep repeating until Second's module reaches 60 count and will reset to 0 again and gives Minute's module a count '1'. Same happen to Minute's module, it will reset when it count until 60 and give Hour's module a count '1'. This module will reset once it count to 24 and trigger an input for Day's module. There are seven outputs for Day's module and they are Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday. They will change and loop to Monday again at every input triggered. All of these modules' outputs, except for Centisecond's module will be display at Display's module.

However, this centisecond will be displayed when the stopwatch's module is activated. While this module is actives, the counter will count and display only from centisecond up until hour. Day will not be included because it is inadequate for a stopwatch to count until day. Differ from centisecond's module, alarm's module usually displayed only when to set the timing for the alarm to be activated. When hour and minute for the alarm has been set, basic clock's module will be activated back. When hour and minute in this module reach the same values set in the alarm's module, alarm will be triggered ON.

Time Zone Conversion's module is a module where local time can be converted to a different time zone. It will add or subtract some amount of hour from the previous value refer to what time zone is applied. This module is directly linked with the basic clock's module where it can convert value of hour directly from the hour's module.

From these module blocks and its behaviour created, codes are constructed. Please refer to the appendices section for these codes. Below, are the results from those behavioural codes:

i. Centisecond's module:

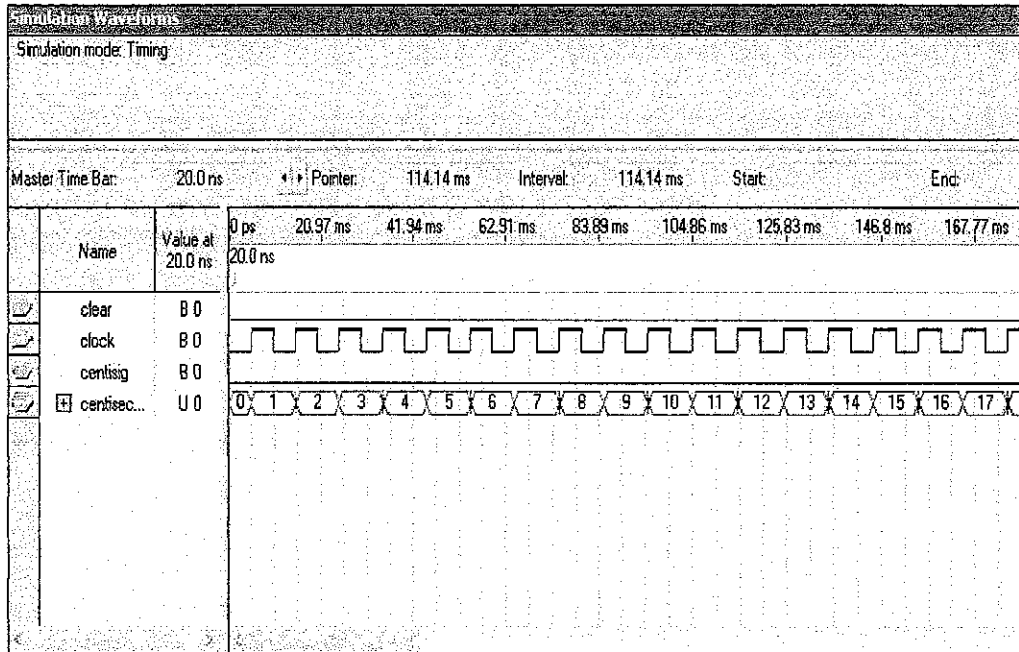


Figure 4.1.1: Waveform of centisecond's module at the initial value.

ii. Second's module:

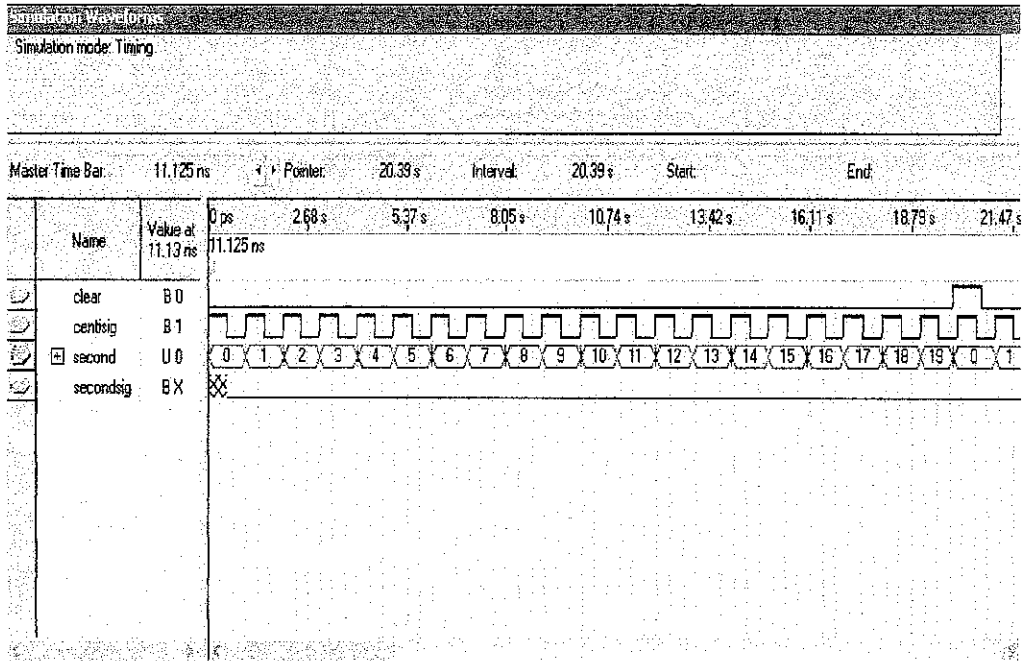


Figure 4.2.1: Waveform of second's module at the initial value until clear = '1'.

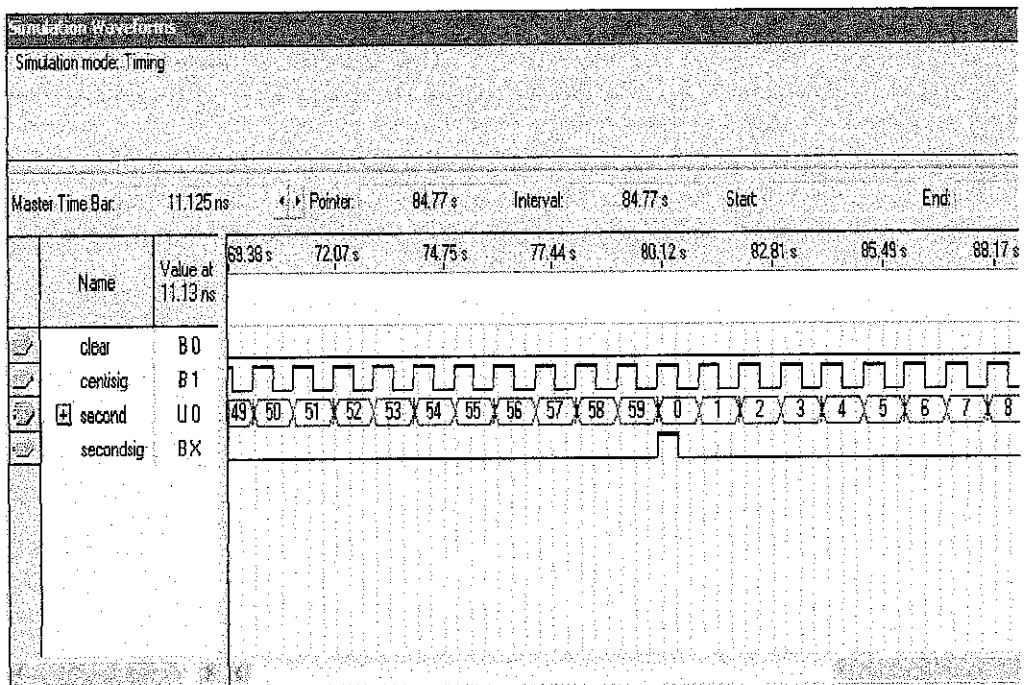


Figure 4.2.2: Waveform of second's module at counter reached 60.

iii. Minute's module

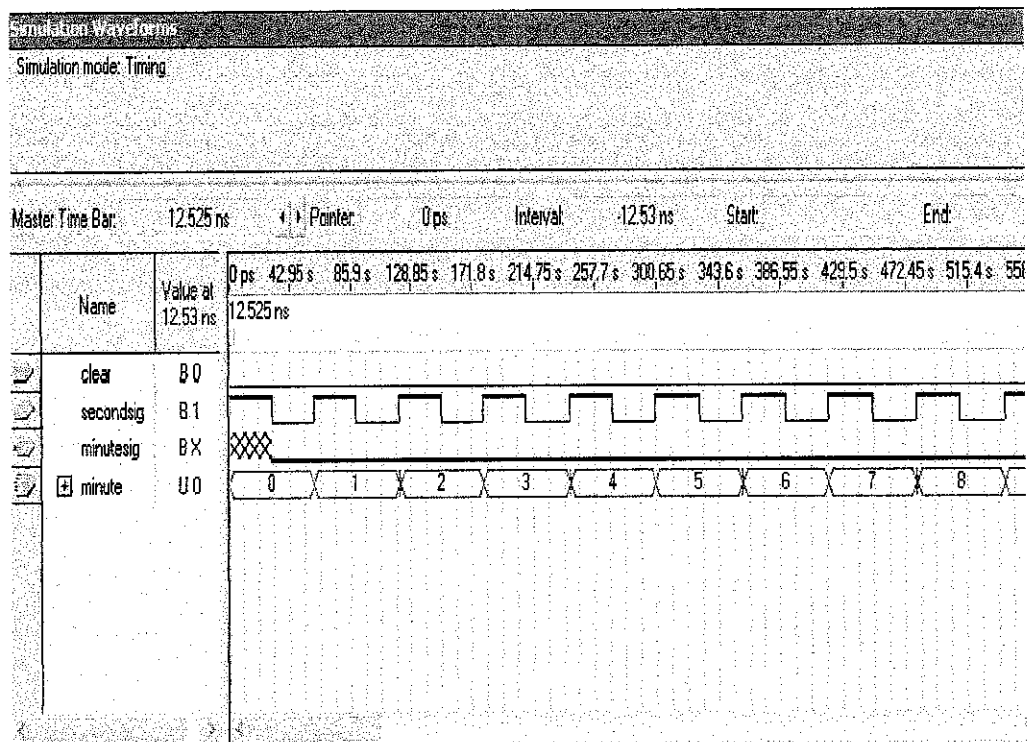


Figure 4.3.1: Waveform of minute's module at the initial value.

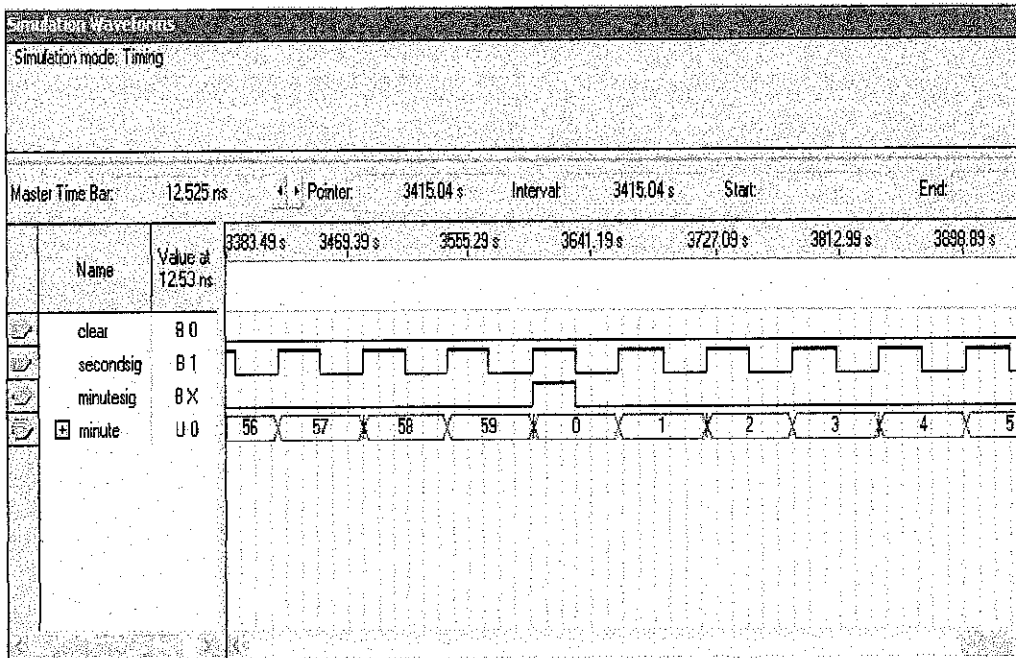


Figure 4.3.2: Waveform of minute's module when counter reached 60.

iv. Hour's module

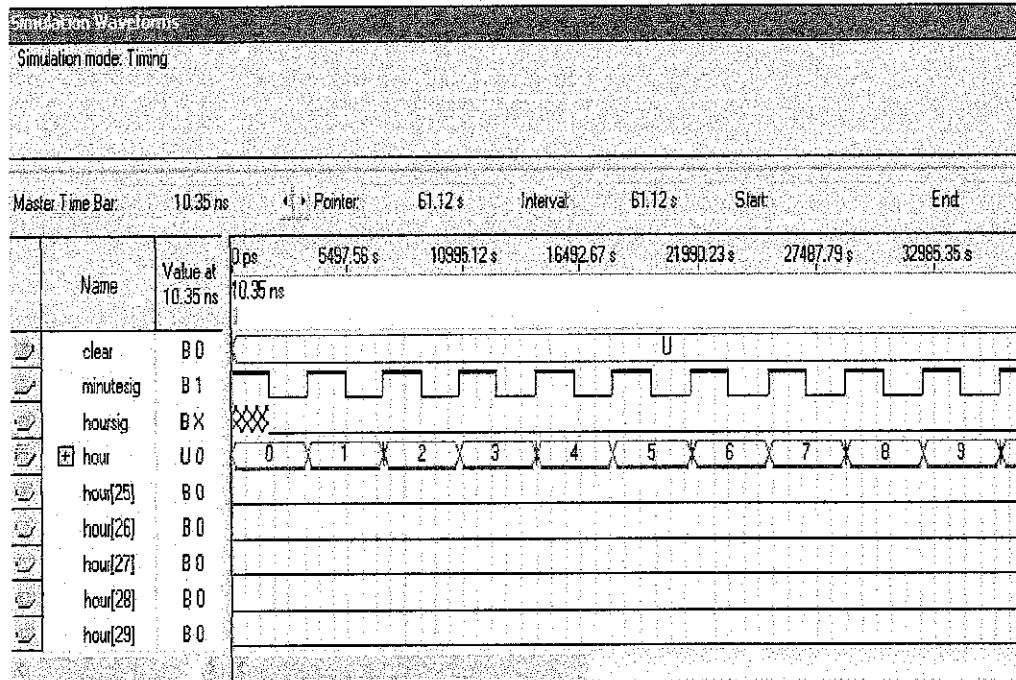


Figure 4.4.1: Waveform of hour's module at the initial value.

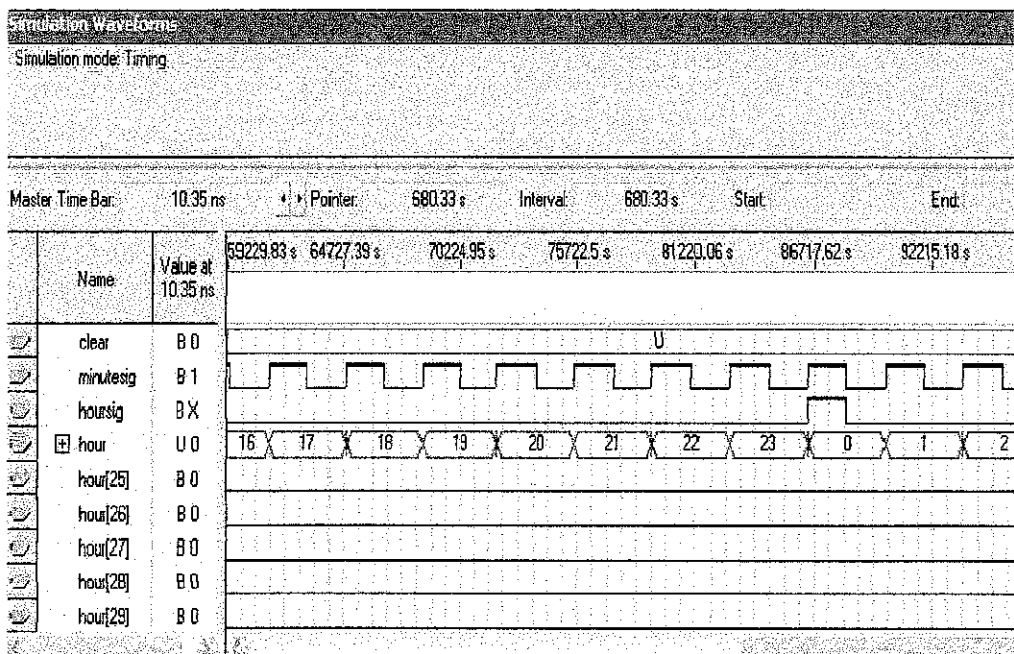


Figure 4.4.2: Waveform of hour's module when counter reached 24.

v. Day's module

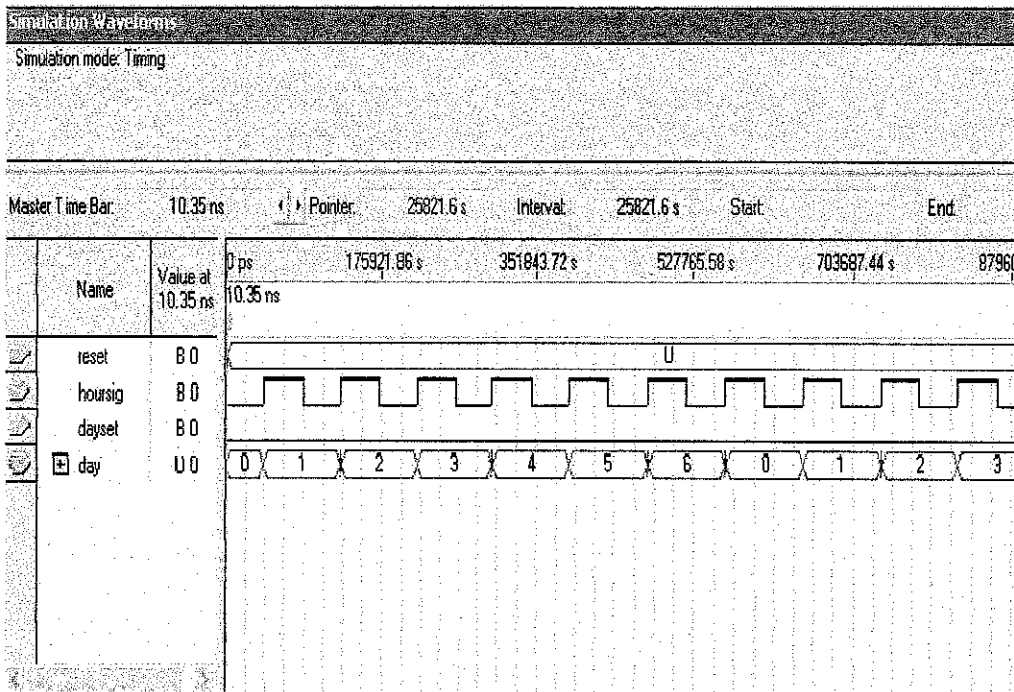


Figure 4.5.1: Waveform of day's module.

vi. Month's module

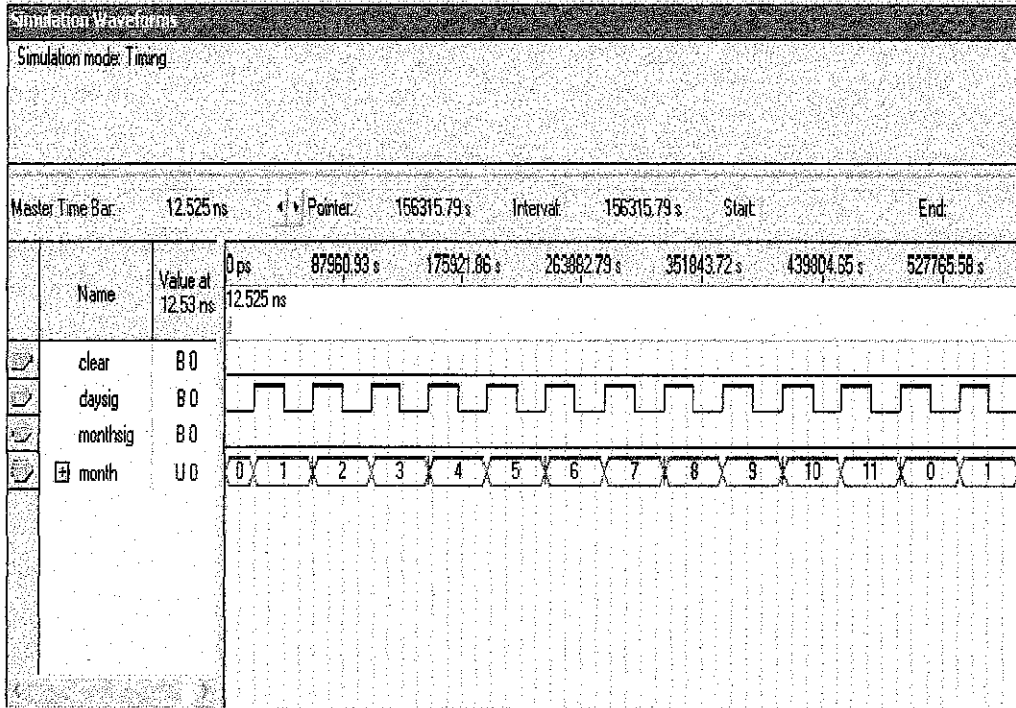


Figure 4.6.1: Waveform of month's module.

vii. Alarm's module

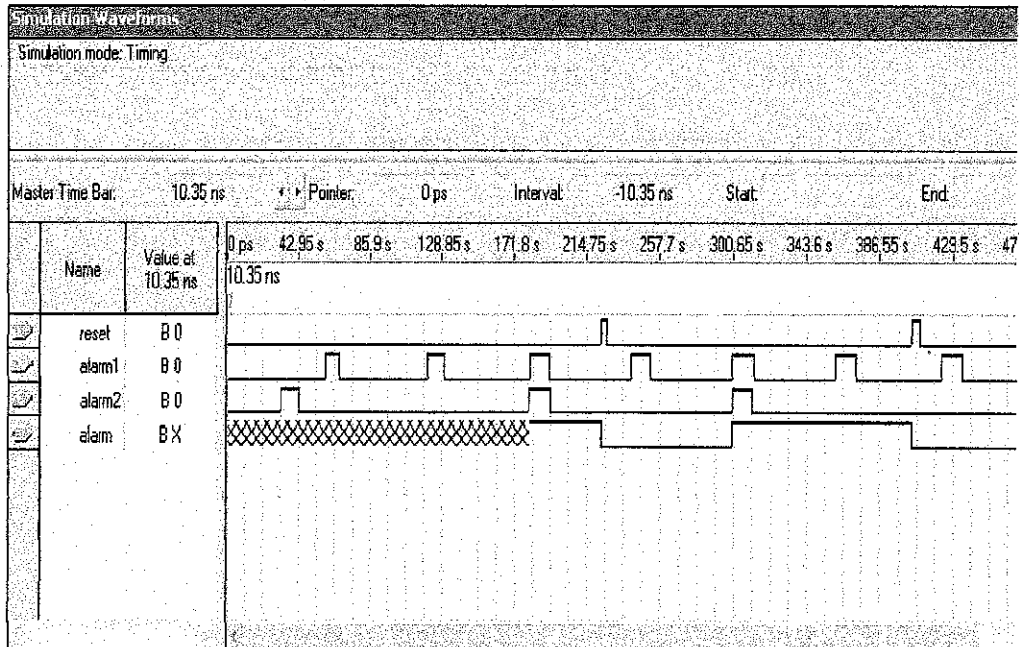


Figure 4.7.1: Waveform of alarm's module.

viii. Stopwatch's module

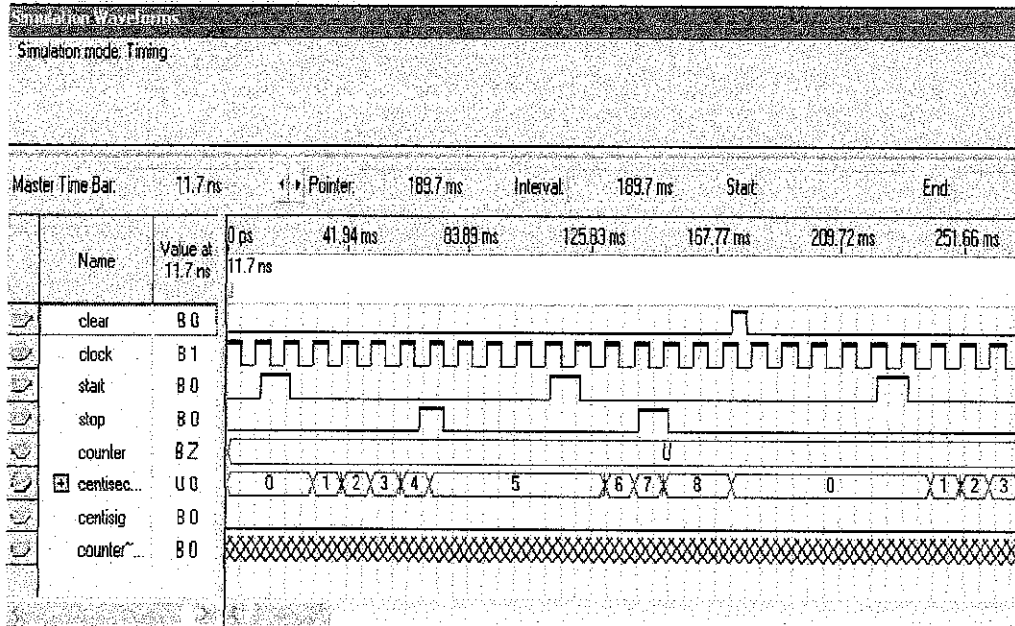


Figure 4.8.1: Waveform of stopwatch's module.

CHAPTER 5

CONCLUSION

The study of these computer languages is the critical part in this project, where it is the key to develop a system that can perform behaviours of a clock as per project's requirement. Then, a familiarization to the simulator or compiler is needed in order to simulate this behaviour codes. After all the modules are ready and simulated, the appropriate waveforms been produced. It can be concluded that these behavioural codes are ready to be implemented into an FPGA to produce an integrated circuit for a multifunctional digital clock.

REFERENCES

- [1] Verilog Digital Computer Design, Algorithms to Hardware, Mark Gordon Arnold, Prentice Hall.
- [2] VHDL Programming by Example, Douglas L. Perry, Mc Graw Hill.
- [3] Introductory VHDL from Simulation to Synthesis, Sudhakar Yalamanchili, Prentice Hall.
- [4] <http://en.wikipedia.org/wiki/FPGA>
- [5] www.webopedia.com/TERM/F/FPGA
- [6] <http://esd.cs.ucr.edu/labs/tutorial>
- [7] http://www.doulos.com/knowhow/vhdl_designers_guide/a_brief_history_of_vhdl/
- [8] <http://www.asic-world.com/verilog/history.html>

APPENDICES

APPENDIX I

Behavioral code for centisecond's module

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity centisecond is

generic (width      : integer :=100);
port  (clock      : in std_logic;
       clear      : in std_logic;
       centisecond : out std_logic_vector(width downto 0);
       centisig   : out std_logic);

end centisecond;

-----
architecture behv of centisecond is

    signal pre_centisecond: std_logic_vector(width downto 0);

begin

    -- behavior describe the centisecond

process(clock, clear)
begin
    if clear = '1' then
        pre_centisecond <= pre_centisecond - pre_centisecond;
    elsif (clock = '1' and clock'event) then
        pre_centisecond <= pre_centisecond + 1;
    end if;
    if pre_centisecond > 99 then
        pre_centisecond <= pre_centisecond - pre_centisecond;
    end if;
    if pre_centisecond > 99 then
        centisig <= '1';
    elsif clock = '0' then
        centisig <= '0';
    end if;
end process;

    -- concurrent assignment statement
centisecond <= pre_centisecond;

end behv;
```

APPENDIX II

Behavioral code for second's module

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity second is

generic (width      :integer :=60);
port  (centisig     :in std_logic;
       clear        :in std_logic;
       second       :out std_logic_vector(width downto 0);
       secondsig    :out std_logic);

end second;

-----

architecture behv of second is

    signal Pre_second: std_logic_vector(width downto 0);

begin

-- behavior describe the second

process(centisig, clear)
begin
    if clear = '1' then
        Pre_second <= Pre_second - Pre_second;
    elsif (centisig = '1' and centisig'event) then
        Pre_second <= Pre_second + 1;
    end if;
    if Pre_second > 59 then
        Pre_second <= Pre_second - Pre_second;
    end if;
    if Pre_second > 59 then
        secondsig <= '1';
    elsif centisig = '0' then
        secondsig <= '0';
    end if;
end process;

-- concurrent assignment statement
second <= Pre_second;

end behv;
```

APPENDIX III

Behavioral code for minute's module

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity minute is

generic (width      : integer :=60);
port  (secondsig    : in std_logic;
       clear        : in std_logic;
       minute       : out std_logic_vector(width downto 0);
       minutesig    : out std_logic);

end minute;

-----

architecture behv of minute is

    signal Pre_minute: std_logic_vector(width downto 0);

begin

-- behavior describe the minute

process(secondsig, clear)
begin
    if clear = '1' then
        Pre_minute <= Pre_minute - Pre_minute;
    elsif (secondsig = '1' and secondsig'event) then
        Pre_minute <= Pre_minute + 1;
    end if;
    if Pre_minute > 59 then
        Pre_minute <= Pre_minute - Pre_minute;
    end if;
    if Pre_minute > 59 then
        minutesig <= '1';
    elsif secondsig = '0' then
        minutesig <= '0';
    end if;
end process;

-- concurrent assignment statement
minute <= Pre_minute;

end behv;
```

APPENDIX IV

Behavioral code for hour's module

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity hour is

generic (width      : integer :=24);
port  (minutesig    : in std_logic;
       clear        : in std_logic;
       hour         : out std_logic_vector(width downto 0);
       hoursig      : out std_logic);
```

```
end hour;
```

```
architecture behv of hour is
```

```
    signal Pre_hour: std_logic_vector(width downto 0);
```

```
begin
```

```
-- behavior describe the hour
```

```
process(minutesig, clear)
```

```
begin
```

```
    if clear = '1' then
```

```
        Pre_hour <= Pre_hour - Pre_hour;
```

```
    elsif (minutesig = '1' and minutesig'event) then
```

```
        Pre_hour <= Pre_hour + 1;
```

```
    end if;
```

```
    if Pre_hour > 23 then
```

```
        Pre_hour <= Pre_hour - Pre_hour;
```

```
    end if;
```

```
    if Pre_hour > 23 then
```

```
        hoursig <= '1';
```

```
    elsif minutesig = '0' then
```

```
        hoursig <= '0';
```

```
    end if;
```

```
end process;
```

```
-- concurrent assignment statement
```

```
hour <= Pre_hour;
```

```
end behv;
```

APPENDIX V

Behavioral code for day's module

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity day is

generic (width           :integer :=40);
port  (hoursig           :in std_logic;
       clear             :in std_logic;
       day               :out std_logic_vector(width downto 0);
       -- daysig         :out std_logic);

end day;

-----

architecture behv of day is

    signal Pre_day: std_logic_vector(width downto 0);

begin

-- behavior describe the day

process(hoursig, clear)
begin
    if clear = '1' then
        Pre_day <= Pre_day - Pre_day;
    elsif (hoursig = '1' and hoursig'event) then
        Pre_day <= Pre_day + 1;
    end if;
    if Pre_day > 29 then
        Pre_day <= Pre_day - Pre_day;
    end if;
    if Pre_day > 29 then
        daysig <= '1';
    elsif hoursig = '0' then
        daysig <= '0';
    end if;
end process;

-- concurrent assignment statement
day <= Pre_day;

end behv;
```

APPENDIX VI

Behavioral code for month's module

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity month is

generic (width      :integer :=12);
port  (daysig      :in std_logic;
       clear        :in std_logic;
       month        :out std_logic_vector(width downto 0);
       monthsig     :out std_logic);

end month;

-----

architecture behv of month is

    signal Pre_month: std_logic_vector(width downto 0);

begin

-- behavior describe the month

process(daysig, clear)
begin
    if clear = '1' then
        Pre_month <= Pre_month - Pre_month;
    elsif (daysig = '1' and daysig'event) then
        Pre_month <= Pre_month + 1;
    end if;
    if Pre_month > 11 then
        Pre_month <= Pre_month - Pre_month;
    end if;
end process;

-- concurrent assignment statement
month <= Pre_month;

end behv;
```

APPENDIX VII

Behavioral code for alarm's module

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity alarm is
generic ( width : integer := 60 );
port   ( reset : in std_logic;
        minute : in std_logic_vector(width downto 0);
        hour   : in std_logic_vector(width downto 0);
        set_min : in std_logic;
        set_hour : in std_logic;
        minutealarm : inout std_logic_vector(width downto 0);
        houralarm : inout std_logic_vector(width downto 0);
        alarm1 : inout std_logic;
        alarm2 : inout std_logic;
        alarm : out std_logic);
end alarm;
```

architecture behv of alarm is

signal pre_alarm: std_logic;

begin

--behavior describe the counter

process (reset, set_min, set_hour)

begin

if reset = '1' then

 pre_alarm <= '0';

elsif alarm1 = '1' and alarm2 = '1' then

 pre_alarm <= '1';

end if;

if set_min = '1' then

 minutealarm <= minutealarm + 1;

end if;

if minutealarm > 59 then

 minutealarm <= minutealarm - minutealarm;

end if;

if set_hour = '1' then

 houralarm <= houralarm + 1;

end if;


```
if houralarm > 23 then
    houralarm <= houralarm - houralarm;
end if;
if minute = minutealarm then
    alarm1 <= '1';
    alarm1 <= '0' after 5ns;
end if;

if hour = houralarm then
    alarm2 <= '1';
    alarm2 <= '0' after 5ns;
end if;

end process;

--concurrent assignment statement
alarm <= pre_alarm;

end behv;
```

APPENDIX VIII

Behavioral code for stopwatch's module

```
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity stopwatch is
```

```
generic (width      : integer :=100);
port  (clock      : in std_logic;
       clear      : in std_logic;
       start      : in std_logic;
       stop       : in std_logic;
       counter    : inout std_logic;
       centisecond : out std_logic_vector(width downto 0);
       centisig   : out std_logic);
```

```
end stopwatch;
```

```
-----
architecture behv of stopwatch is
```

```
    signal pre_centisecond: std_logic_vector(width downto 0);
```

```
begin
```

```
    -- behavior describe the centisecond
```

```
proc1: process(clock, clear)
begin
    if clear = '1' then
        pre_centisecond <= pre_centisecond - pre_centisecond;
    elsif (clock = '1' and clock'event) then
        if counter = '1' then
            pre_centisecond <= pre_centisecond + 1;
        end if;
        if pre_centisecond > 99 then
            pre_centisecond <= pre_centisecond - pre_centisecond;
        end if;
        if pre_centisecond > 99 then
            centisig <= '1';
        elsif clock = '0' then
            centisig <= '0';
        end if;

        if start = '1' then
```

```
        counter <= '1';
    elsif stop = '1' then
        counter <= '0';
    end if;

    end if;
end process;
```

```
-- concurrent assignment statement
centisecond <= pre_centisecond;
```

```
end behv;
```