Digital Electronics - Laboratory 2
Introduction to VHDL programming

Academic year 2023-2024 / Teaching assistant : Arthur Fyon

Introduction

The goal of this lab is to familiarize you with the software Quartus required to implement your VHDL project on the CPLD.

⚠ Installation (to be done at home before the lab) ⚠

Quartus is a software produced by Intel. To install it, follow the next steps :

1. Create a free account on intel.com.
2. Download the 18.1 Lite version of Quartus, ModelSim Starter and MAX V device support (Windows Linux, under individual files), don't download more recent versions of Quartus as troubles have been reported with the CPLD that you will use.
3. Launch the Quartus setup file and check ModelSim Starter as well as MAX II/V device boxes on the Select Components window at the beginning of the installation.
4. Follow further steps.
5. At the end of the installation, **DO NOT** install the recommended drivers from the installation of Quartus, by unchecking the corresponding box.
6. Before launching Quartus, connect the CPLD to your computer using the USB Blaster.
7. From the device manager ("gestionnaire de peripheriques") of your operating system, right click on Altera USB-Blaster. Then, click on update drivers and select the driver folder available on myUliege (you might need to unzip the archive *drivers.zip*).

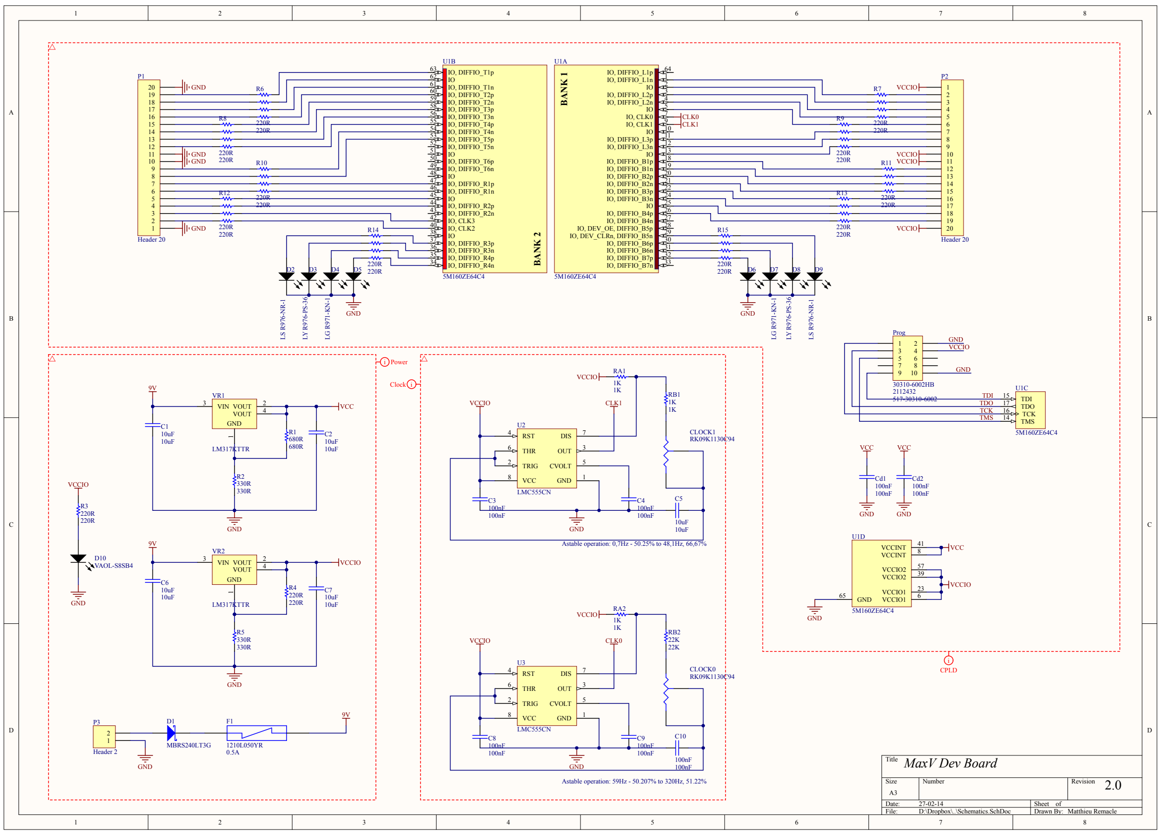
Electronic board

Figure 1 shows the schematic of the electronic board used for the project. As you can see, the CPLD is soldered on the board. Some pins of the CPLD are directly accessible via the pins on the board with a 220 Ω resistor, some are directly connected to LEDs and 2 are connected with the clock signals generated by to 2 555-timers.

1 Development of a simple electrocardiogram

Specification

Let's imagine that an analogue amplification circuit connects to electrodes on a patient, and the output is a voltage between 0 V to 3.3 V. Moreover, the output voltages are such that



MaxV Dev Board
Revision 2.0
BANK 1
BANK 2
5M160ZE64C4
LMC555CN
LM317KTTR
Header 20
Astable operation: 0,7Hz - 50.25% to 48,1Hz, 66,67%
Astable operation: 59Hz - 50.207% to 320Hz, 51.22%

FIGURE 1 – Electronic board

the output can be considered as a 0 most of the time and as a 1 during a heartbeat.

The aim of this example is to design a system that switches on an indicator light when the patient's heart rate exceeds a certain value, set here at 150 bpm (beats per minute), and another light when this rate falls below 40 bpm.

Architecture

One possible solution is to use a fast clock (faster than a heartbeat), and count the number of clock strokes between each beat. The lower the number of strokes, the faster the speed of the beats, and vice versa. So you will need an internal signal to count the number of clock strokes between beats.

The first difficulty is to design the frequency of the clock. Indeed, a high frequency will make it possible to be more precise in the heart rate measurement, but on the other hand, the maximum number of counts is increased, and therefore the number of registers required to store the internal signal will be higher.

A frequency of 60 Hz seems to be a good compromise. Indeed, at 150 bpm, it will take 24 clock strokes, while at 40 bpm, it will take 90 clock strokes. Thus, only 7 bits are required to retain the number of clock strokes.

In addition to the bits used for counting, two other bits should be used to retain the output values of the LEDs between 2 heartbeats.

A possible implementation is an architecture with a single simple process, carrying out the following actions :

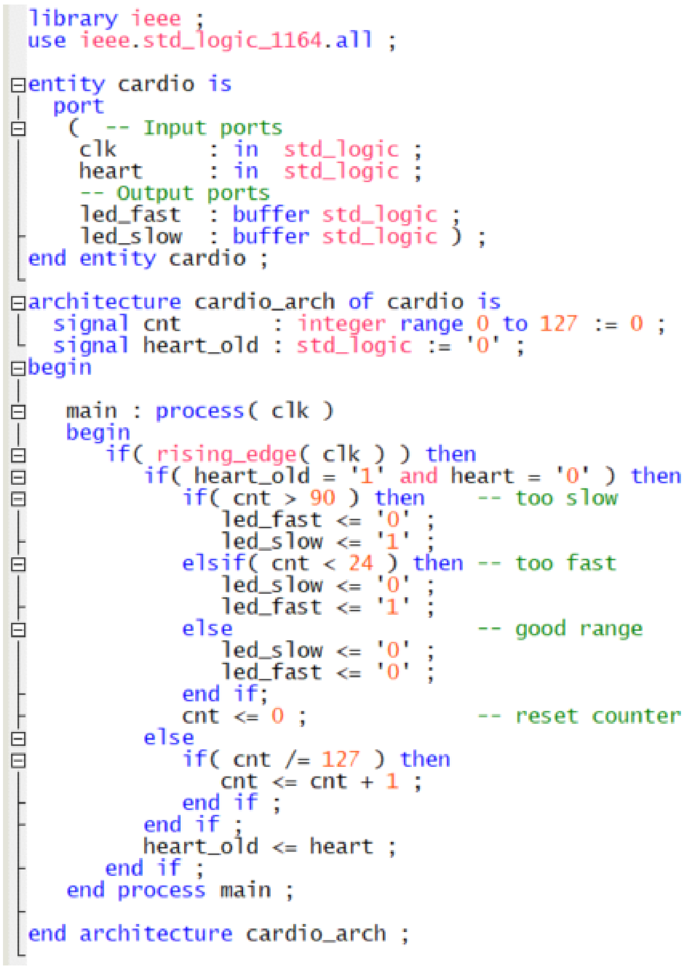
— Count the clock strikes between two heart beat.

— Update the indicator light according the heart frequency during each beat, i.e. when the input is at 1.

On the board with the CPLD, we are going to use a clock signal from a 555 timer to simulate a heartbeat.

Code

Here is an example of a code performing the requested operations.



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

entity cardio is
   port
   (  -- Input ports
    clk        : in   std_logic ;
    heart      : in   std_logic ;
    -- Output ports
    led_fast   : buffer std_logic ;
    led_slow   : buffer std_logic ) ;
end entity cardio ;

architecture cardio_arch of cardio is
   signal cnt       : integer range 0 to 127 := 0 ;
   signal heart_old : std_logic := '0' ;
begin

   main : process( clk )
   begin
      if( rising_edge( clk ) ) then
         if( heart_old = '1' and heart = '0' ) then
            if( cnt > 90 ) then     -- too slow
               led_fast <= '0' ;
               led_slow <= '1' ;
            elsif( cnt < 24 ) then -- too fast
               led_slow <= '0' ;
               led_fast <= '1' ;
            else                   -- good range
               led_slow <= '0' ;
               led_fast <= '0' ;
            end if;
            cnt <= 0 ;             -- reset counter
         else
            if( cnt /= 127 ) then
               cnt <= cnt + 1 ;
            end if ;
         end if ;
         heart_old <= heart ;
      end if ;
   end process main ;

end architecture cardio_arch ;
```

Figure 2 – Cardio implementation

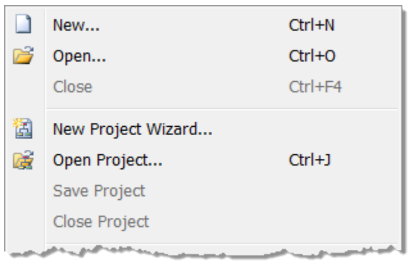
The **heart_old** signal is used to retain the previous value of **heart**. It can therefore detect the end of a heartbeat, i.e. when the **heart** signal changes from 1 (previous rising edge) to 0 (present rising edge). When this condition is detected, the LEDs are lit according to the counter value beat and the counter is then reset.

2 Creation and simulation of a project

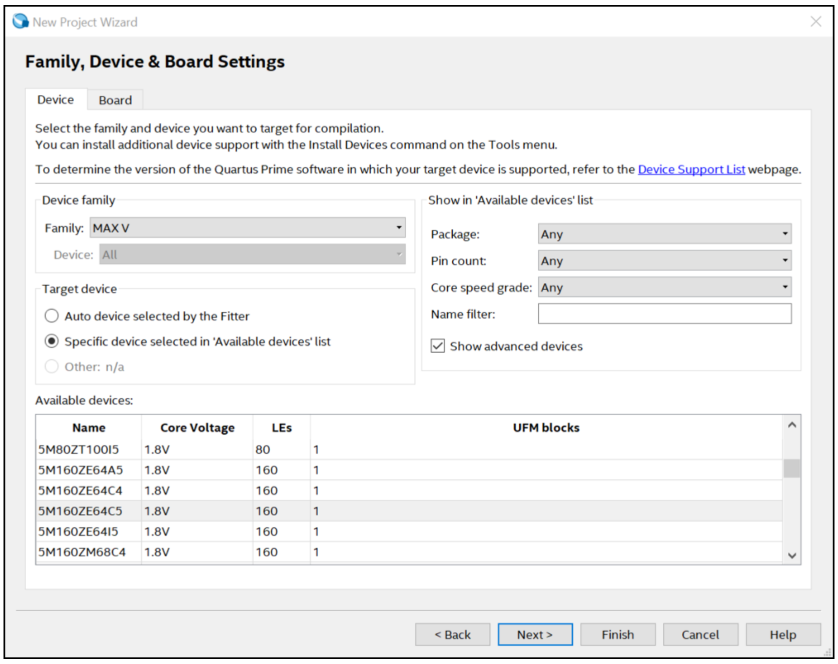
Quartus

This software works with the notion of Projects. A project includes all the source codes, programming files, constraint files, etc., necessary for the programming of a component. You will therefore have a single project for your laboratory. The creation of a project is relatively simple :

1. Click on **File** → **New Project Wizard**.



2. A new window opens. Now, you can configure the characteristics of your project. First, specify the location and the name (put cardio here) of your project. Then, you can skip the step of adding a file (we will create one later). Then, you have to specify which component is going to be used. Select the component reference **5M160ZE64C5**.



New Project Wizard

Family, Device & Board Settings

Device | Board

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the Device Support List webpage.

Device family
Family: MAX V
Device: All

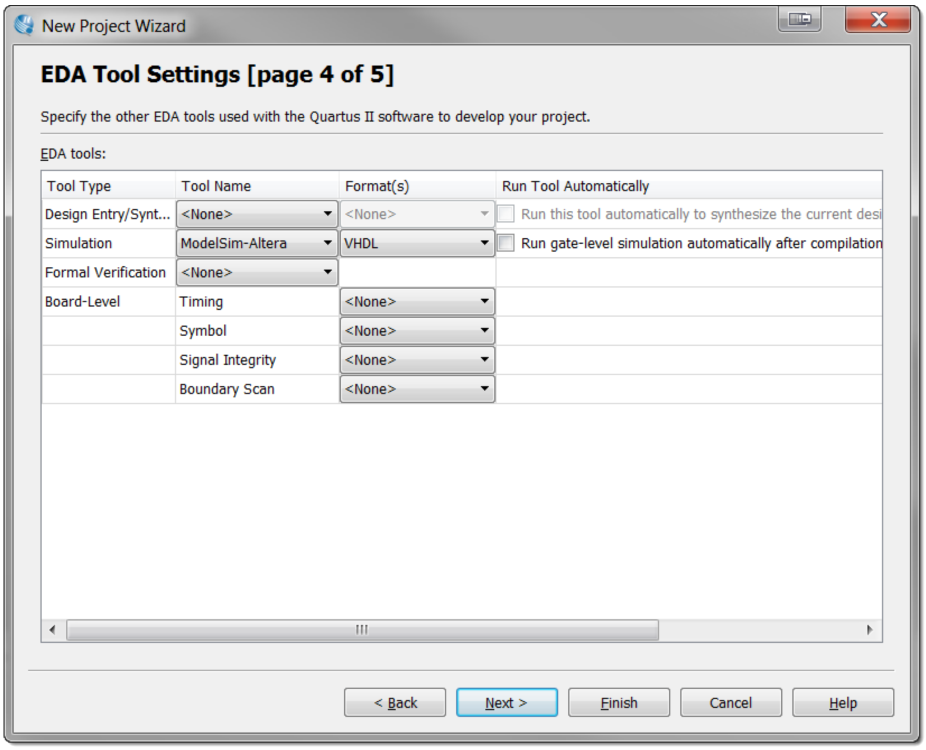
Target device
○ Auto device selected by the Fitter
● Specific device selected in 'Available devices' list
○ Other: n/a

Show in 'Available devices' list
Package: Any
Pin count: Any
Core speed grade: Any
Name filter:
☑ Show advanced devices

Available devices:

Name	Core Voltage	LEs	UFM blocks
5M80ZT100I5	1.8V	80	1
5M160ZE64A5	1.8V	160	1
5M160ZE64C4	1.8V	160	1
5M160ZE64C5	1.8V	160	1
5M160ZE64I5	1.8V	160	1
5M160ZM68C4	1.8V	160	1

3. Finally, in the simulation section, choose **ModelSim Altera** and **VHDL**.



New Project Wizard

EDA Tool Settings [page 4 of 5]

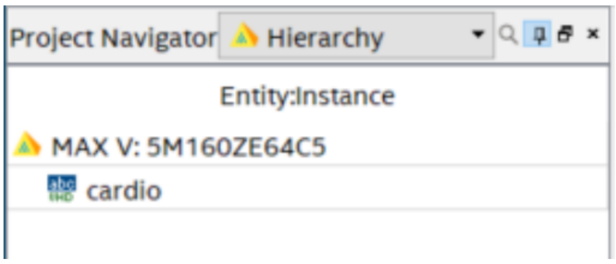
Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synt...	<None>	<None>	Run this tool automatically to synthesize the current desi
Simulation	ModelSim-Altera	VHDL	Run gate-level simulation automatically after compilation
Formal Verification	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

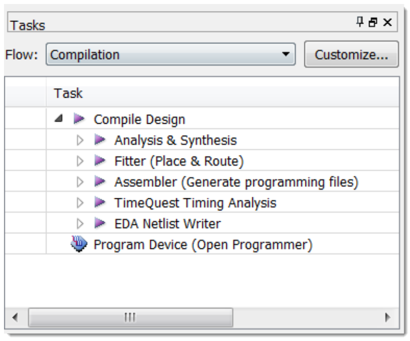
The project is created and you have now access to the main window of Quartus.

Project Navigator : This window provides a summary of the project hierarchy. In this laboratory, the hierarchy will remain simple since we will only have one file. In this window, it is also possible to review all the files of the project.

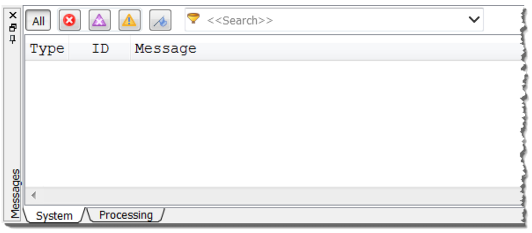


Tasks : This window provides all the information about the tasks to be performed to have an operating project. Each task (for example : the synthesis) can be performed independently by double-clicking on it. If a task requires another task to be done, it will be performed as well.

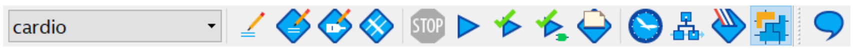
Moreover, it is possible via this window to access the compilation reports, showing what happened during the execution of a task. To do so, simply click on the small arrow to expand the task, and then click on one of the proposed reports.



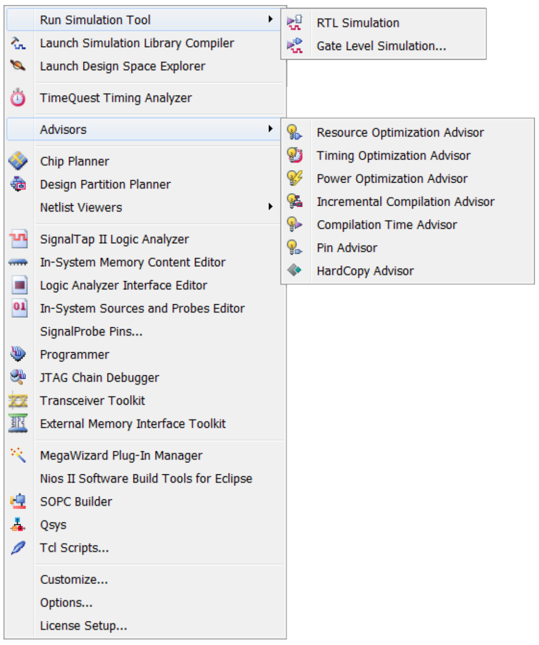
Messages : Generated messages (e.g. compilation errors, etc...) will be displayed here.



Tools : Important tools, such as the launch of a compilation, a timing analysis, or the pin planner can be found in this bar.



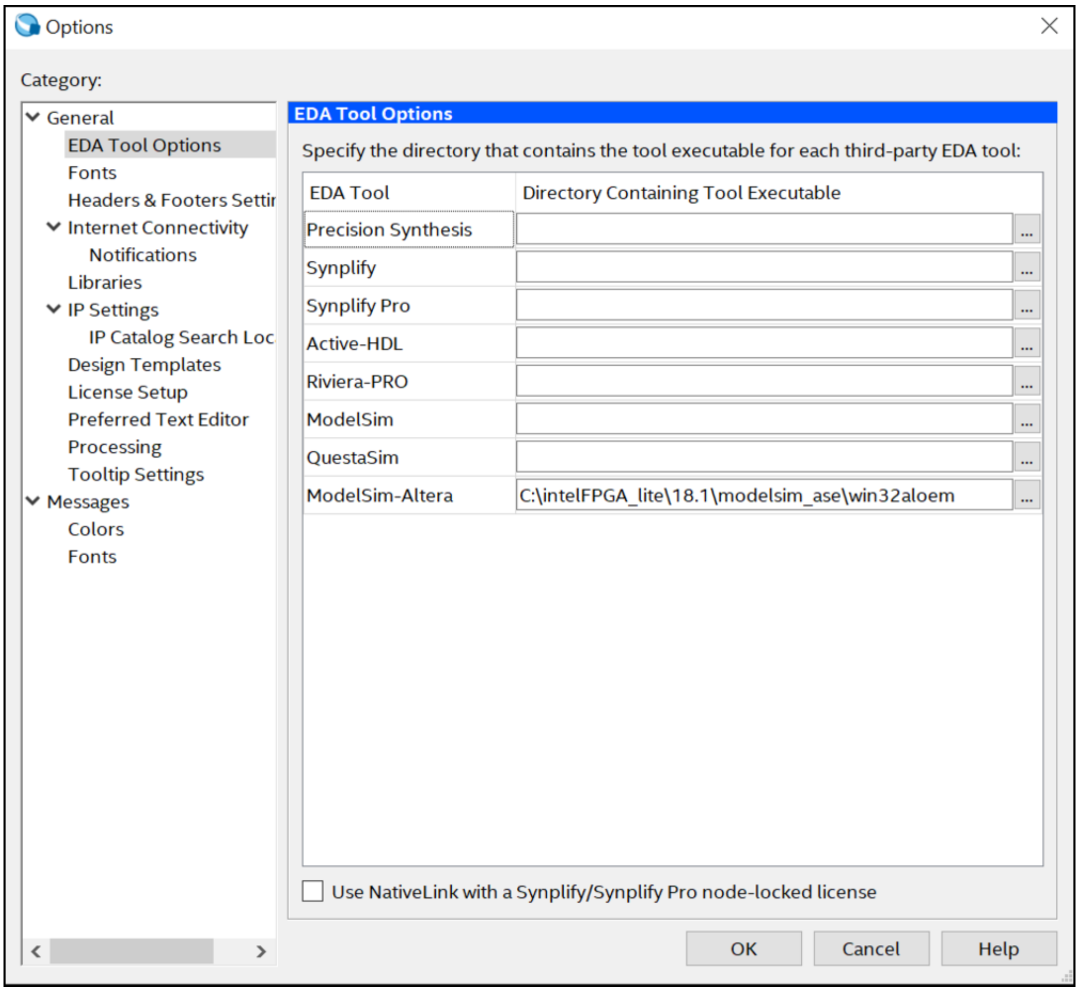
In the top bar, further tools are available under Tools.



Run Simulation Tool
Launch Simulation Library Compiler
Launch Design Space Explorer
TimeQuest Timing Analyzer
Advisors
Chip Planner
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
In-System Sources and Probes Editor
SignalProbe Pins...
Programmer
JTAG Chain Debugger
Transceiver Toolkit
External Memory Interface Toolkit
MegaWizard Plug-In Manager
Nios II Software Build Tools for Eclipse
SOPC Builder
Qsys
Tcl Scripts...
Customize...
Options...
License Setup...
RTL Simulation
Gate Level Simulation...
Resource Optimization Advisor
Timing Optimization Advisor
Power Optimization Advisor
Incremental Compilation Advisor
Compilation Time Advisor
Pin Advisor
HardCopy Advisor

Configuration of ModelSim

To complete the configuration of ModelSim in Quartus, it is necessary to specify the path to win32aloem (in the installation folder of ModelSim, often the same as Quartus) by clicking on Tools → Options → EDA Tool Options.



EDA Tool Options

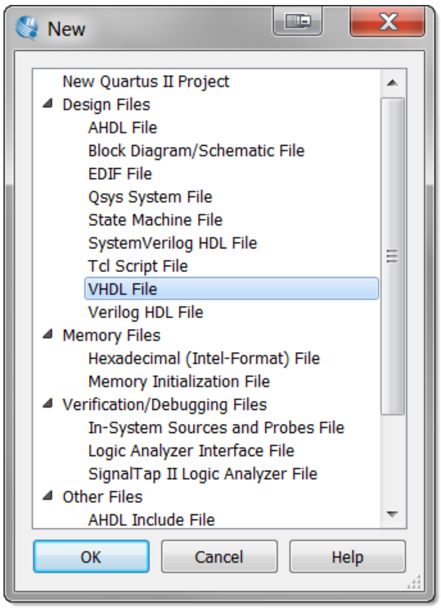
Specify the directory that contains the tool executable for each third-party EDA tool:

EDA Tool	Directory Containing Tool Executable
Precision Synthesis	
Synplify	
Synplify Pro	
Active-HDL	
Riviera-PRO	
ModelSim	
QuestaSim	
ModelSim-Altera	C:\intelFPGA_lite\18.1\modelsim_ase\win32aloem

☐ Use NativeLink with a Synplify/Synplify Pro node-locked license

VHDL File

Now, let's implement our electrocardiogram. Click on File → New and in the window that appears, select VHDL file.


New
New Quartus II Project
Design Files
AHDL File
Block Diagram/Schematic File
EDIF File
Qsys System File
State Machine File
SystemVerilog HDL File
Tcl Script File
VHDL File
Verilog HDL File
Memory Files
Hexadecimal (Intel-Format) File
Memory Initialization File
Verification/Debugging Files
In-System Sources and Probes File
Logic Analyzer Interface File
SignalTap II Logic Analyzer File
Other Files
AHDL Include File
OK
Cancel
Help

An empty file opens in the workspace, all you have to do is to copy the example code displayed in the Figure 2 (in *cardio.txt* available on myUliege). Once it's done, you can check that what you have written is correct with the button . If the compilation is successful, a message will be displayed in the message window and a (empty) compilation report in the workspace.

As your file is syntactically correct, it is now time to launch the analysis and the simulation steps.

ModelSim

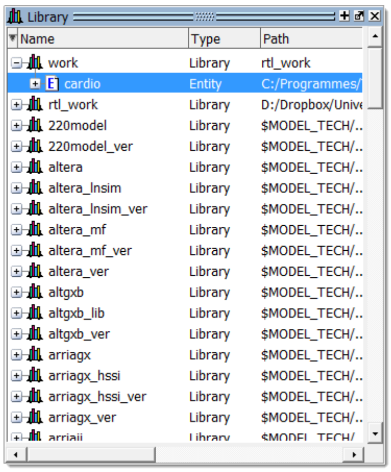
In order to check the system implementation, it can be simulated. Click on Tools → Run Simulation Tool → RTL Simulation, which will launch the ModelSim program with all the parameters already loaded. As for Quartus, the workspace is divided into different areas. The most important are the following :

Libraries : All necessary libraries are listed in this window. In particular, your program can be found in **work**. Double-click on the name of your program to start the simulation.

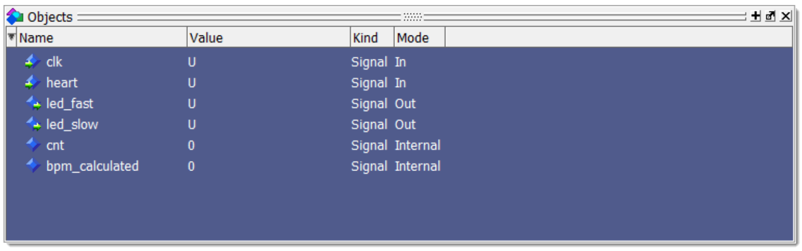
Objects : All available signals will appear in this window. You can drag and drop them into the Wave window with the mouse to display their evolution during the simulation.

Wave : All requested simulation results will be accessible in this window. Here, the **clk** object has been moved (If the window does not appear, do View → Wave).

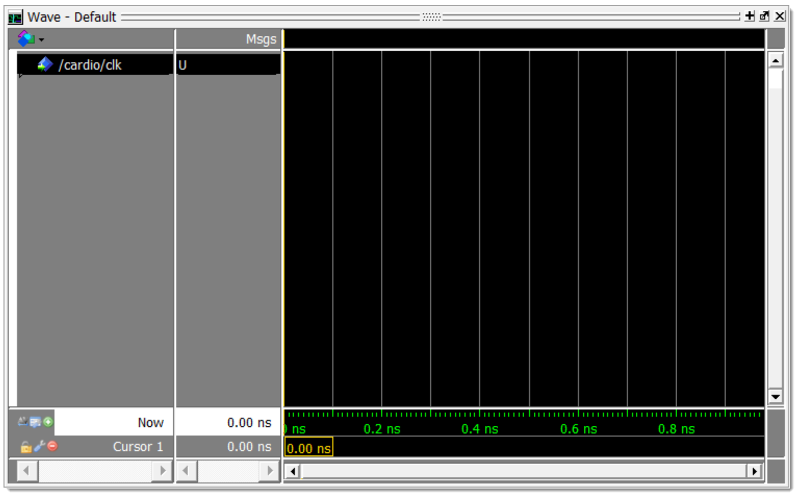
Console : It is possible to control the software using a console, i.e. with commands in the console. I do not recommend this method.


Library

Name	Type	Path
work	Library	rtl_work
cardio	Entity	C:/Programmes/
rtl_work	Library	D:/Dropbox/Unive
220model	Library	$MODEL_TECH/..
220model_ver	Library	$MODEL_TECH/..
altera	Library	$MODEL_TECH/..
altera_lnsim	Library	$MODEL_TECH/..
altera_lnsim_ver	Library	$MODEL_TECH/..
altera_mf	Library	$MODEL_TECH/..
altera_mf_ver	Library	$MODEL_TECH/..
altera_ver	Library	$MODEL_TECH/..
altgxb	Library	$MODEL_TECH/..
altgxb_lib	Library	$MODEL_TECH/..
altgxb_ver	Library	$MODEL_TECH/..
arriagx	Library	$MODEL_TECH/..
arriagx_hssi	Library	$MODEL_TECH/..
arriagx_hssi_ver	Library	$MODEL_TECH/..
arriagx_ver	Library	$MODEL_TECH/..
arriaii	Library	$MODEL_TECH/


Objects

Name	Value	Kind	Mode
clk	U	Signal	In
heart	U	Signal	In
led_fast	U	Signal	Out
led_slow	U	Signal	Out
cnt	0	Signal	Internal
bpm_calculated	0	Signal	Internal



When you double-clicked on your program in the Libraries window, you started a simulation. In fact, your simulation was initialized but nothing happened yet. Indeed, you have to first give the software stimuli, i.e. determine the system inputs. With ModelSim, there are 2 main ways of doing it.

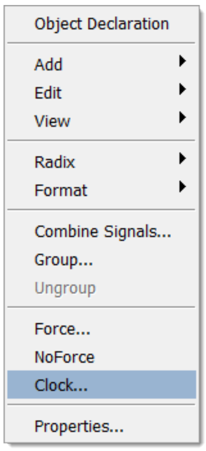
— Launch the simulation without any information and specify the stimuli directly in the simulation.

— Start the simulation with a test bench file, which contains all the stimuli specifications.

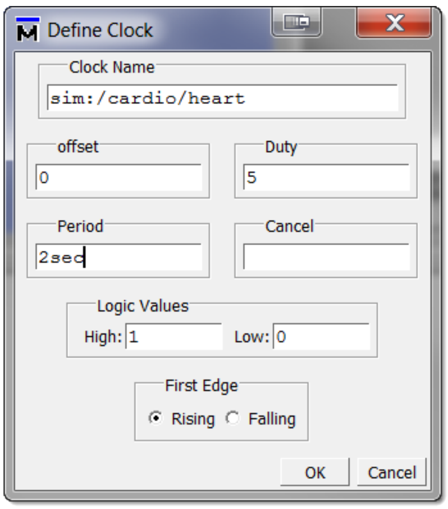
1st option

Start by first moving all the objects in the Wave window, which will allow you to view the signals. Note that inputs, outputs, but also internal signals are available, which is very useful for debugging your code.

Then, right click on the **clk** item. Then, in the menu that appears, choose Clock.



In the Period field, enter "16ms". Repeat the operation with the heart signal, but this time with a period of "2sec", and a duty cycle of 5, which will simulate a heartbeat.


Define Clock

Clock Name: sim:/cardio/heart

offset: 0

Duty: 5

Period: 2sec

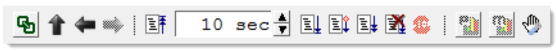
Cancel:

Logic Values — High: 1 Low: 0

First Edge — [✓ Rising] Falling

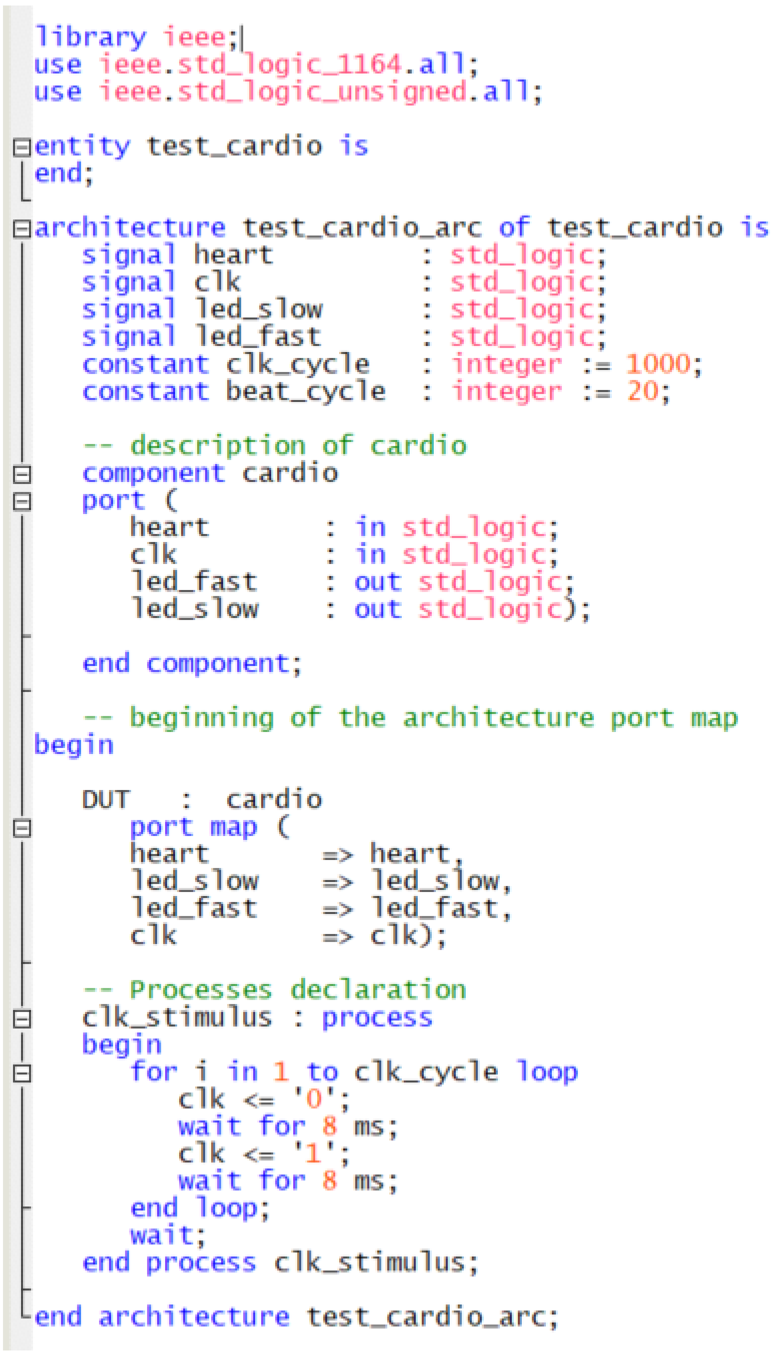
OK Cancel

Finally, run the simulation for 10 seconds. In order to so so, write 10 sec in the appropriate field and click on the button .



2nd option : Test bench

Now, we will see the second possibility of simulation : the use of a test bench file. It's a VHDL code that includes the entity that is being tested, and runs different test processes. The unit under test is called Unit Under Test (UUT), or Device Under Test (DUT). The code test is shown in Figure 3.



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

entity test_cardio is
end;

architecture test_cardio_arc of test_cardio is
    signal heart            : std_logic;
    signal clk              : std_logic;
    signal led_slow         : std_logic;
    signal led_fast         : std_logic;
    constant clk_cycle      : integer := 1000;
    constant beat_cycle     : integer := 20;

    -- description of cardio
    component cardio
    port (
        heart       : in std_logic;
        clk         : in std_logic;
        led_fast    : out std_logic;
        led_slow    : out std_logic);

    end component;

    -- beginning of the architecture port map
begin

    DUT   :  cardio
        port map (
        heart       => heart,
        led_slow    => led_slow,
        led_fast    => led_fast,
        clk         => clk);

    -- Processes declaration
    clk_stimulus : process
    begin
        for i in 1 to clk_cycle loop
            clk <= '0';
            wait for 8 ms;
            clk <= '1';
            wait for 8 ms;
        end loop;
        wait;
    end process clk_stimulus;

end architecture test_cardio_arc;
```

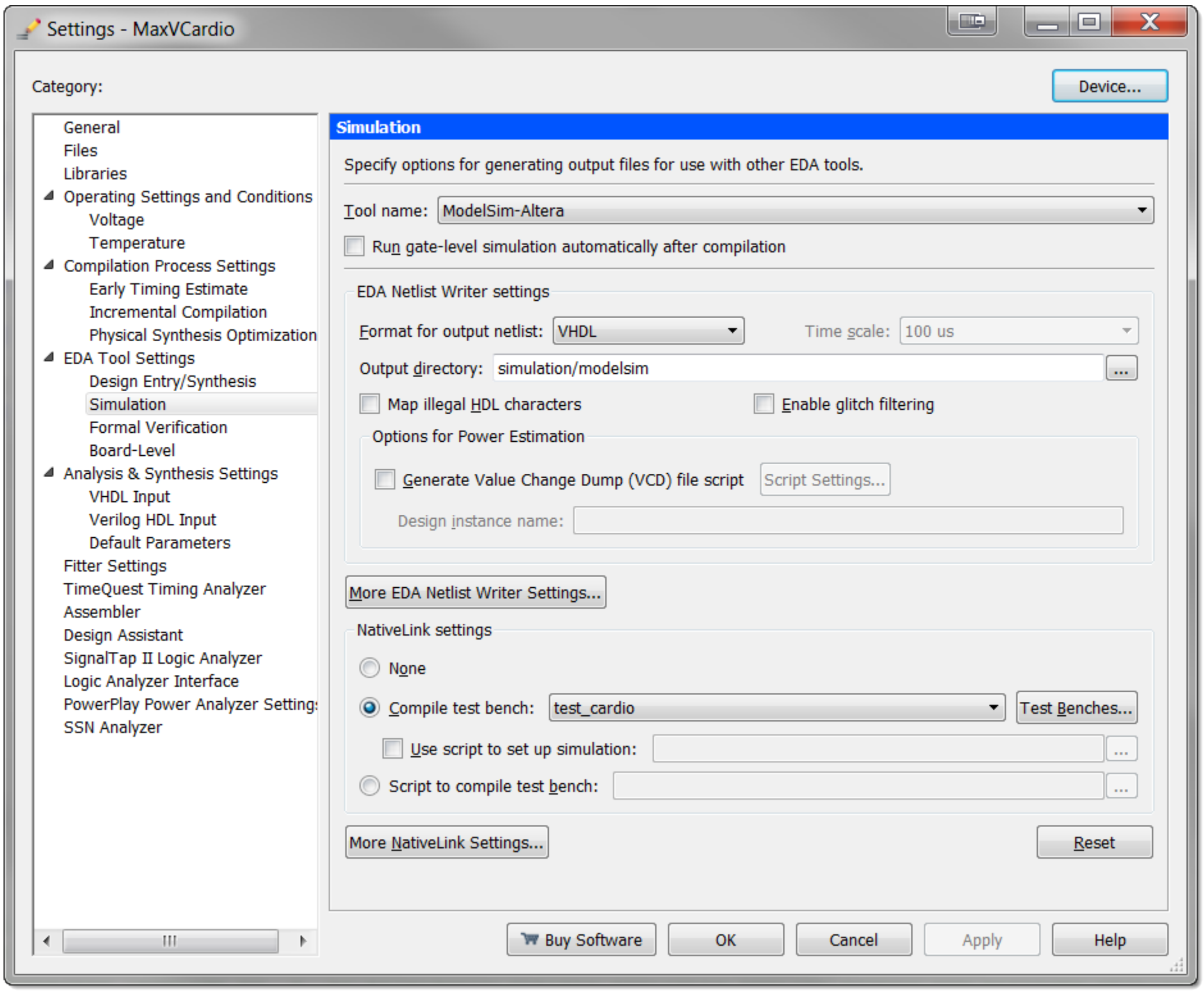
FIGURE 3 – Test bench implementation

The code structure is as follows :

- Declaration of an empty entity for the simulation.
- Declaration of the internal signals that correspond to the inputs/outputs of the DUT.
- Connection of the DUT within our architecture.
- Specification of DUT inputs.

The test bench can be seen as a block box with no inputs/outputs in which the DUT is tested.

Let's add the test bench file to our project. Create a new VHDL file, copy the code (in *test_ cardio.txt* available on myUliege) and save the file. Then, in Quartus, click on Assignments → Settings. In the window that opens, choose the Simulation part under EDA Tools Settings.



Specify options for generating output files for use with other EDA tools.

Tool name: ModelSim-Altera

☐ Run gate-level simulation automatically after compilation

EDA Netlist Writer settings

Format for output netlist: VHDL   Time scale: 100 us

Output directory: simulation/modelsim

☐ Map illegal HDL characters   ☐ Enable glitch filtering

Options for Power Estimation

☐ Generate Value Change Dump (VCD) file script

Design instance name:

NativeLink settings

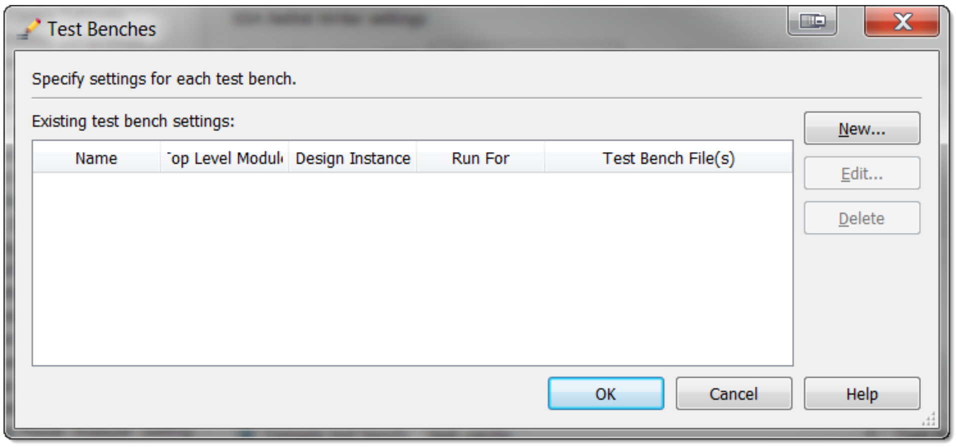
○ None

● Compile test bench: test_cardio

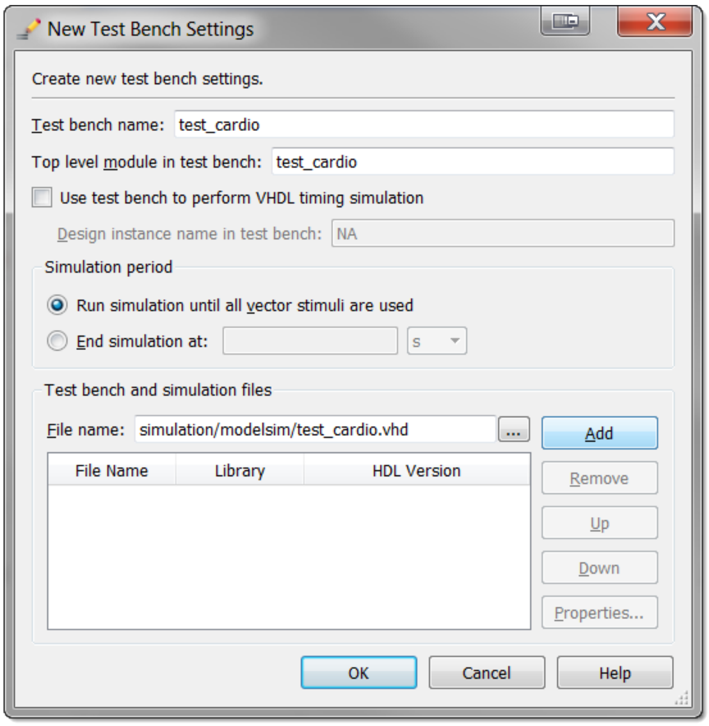
☐ Use script to set up simulation:

○ Script to compile test bench:

Then, in the NativeLink settings section, check Compile test bench, and click on the Test Benches button.



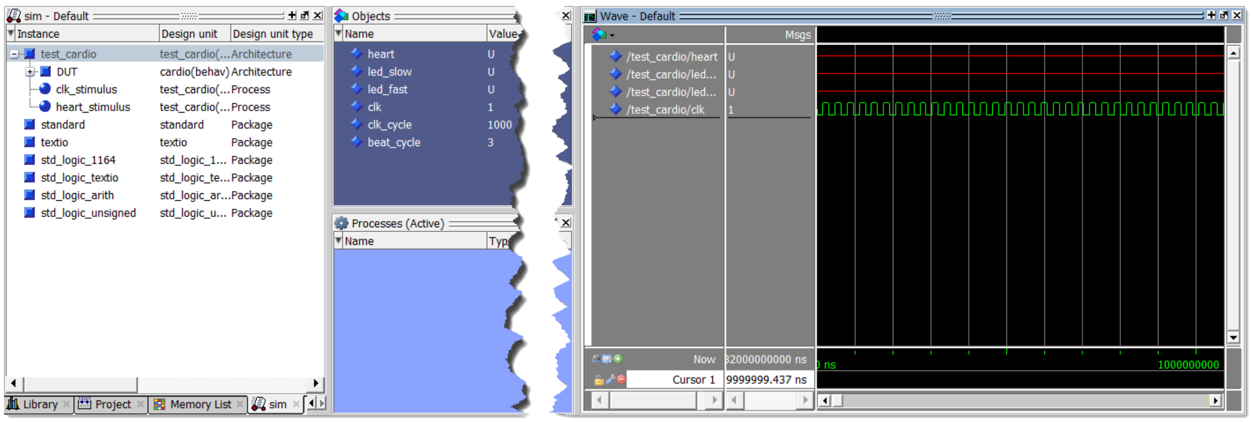
In the window that opens, press New.



Test bench name: test_cardio
Top level module in test bench: test_cardio
Design instance name in test bench: NA
Run simulation until all vector stimuli are used
File name: simulation/modelsim/test_cardio.vhd

Indicate a name for the test bench, following the example in the screenshot. choose the duration of the simulation, and finally, using the button ..., select your test file. Don't forget to click on Add.

Then return to the main Quartus window. Then click on Tools → Run Simulation Tool → RTL Simulation. The simulation starts. After adjusting the zoom in the Wave window with the O and I keys of your keyboard, you should get the window represented in the figure below.

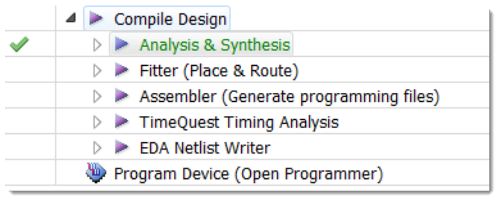


As shown in the screenshot, your simulation starts automatically, with the information contained in the Test bench file. However, the test bench does not contain any information about the heart signal. The heart signal appears in red.

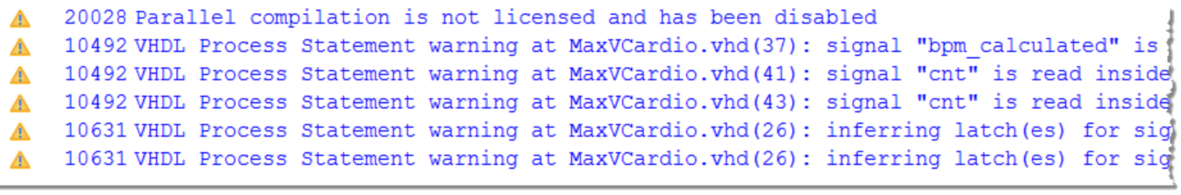
Your task is to modify the Test bench file to implement your test scenario. Check that the system behaves as desired.

Analysis & Synthesis

During this step, the software will analyse your VHDL file and translate it into logical elements. To start the analysis, simply click on Analysis & Synthesis in the task window. If this operation is successful, you should see a small ✓, like this :


Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programming files)
TimeQuest Timing Analysis
EDA Netlist Writer
Program Device (Open Programmer)

In addition, a compilation report window has opened in your workspace. If this is not the case, check your VHDL file again.


```
20028 Parallel compilation is not licensed and has been disabled
10492 VHDL Process Statement warning at MaxVCardio.vhd(37): signal "bpm_calculated" is
10492 VHDL Process Statement warning at MaxVCardio.vhd(41): signal "cnt" is read inside
10492 VHDL Process Statement warning at MaxVCardio.vhd(43): signal "cnt" is read inside
10631 VHDL Process Statement warning at MaxVCardio.vhd(26): inferring latch(es) for sig
10631 VHDL Process Statement warning at MaxVCardio.vhd(26): inferring latch(es) for sig
```

Once the file has been analysed, Some information on what has been achieved is now available. You can access the compilation report, if it is not already open, with a right click on Analysis & Synthesis then View Report.

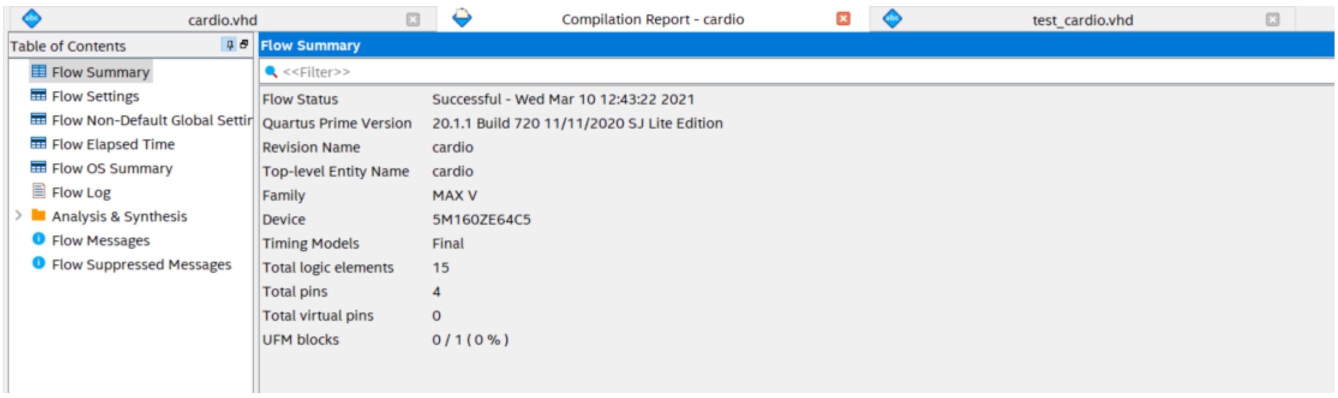

cardio.vhd | Compilation Report - cardio | test_cardio.vhd

Table of Contents
- Flow Summary
- Flow Settings
- Flow Non-Default Global Settir
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Flow Messages
- Flow Suppressed Messages

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Mar 10 12:43:22 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	cardio
Top-level Entity Name	cardio
Family	MAX V
Device	5M160ZE64C5
Timing Models	Final
Total logic elements	15
Total pins	4
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

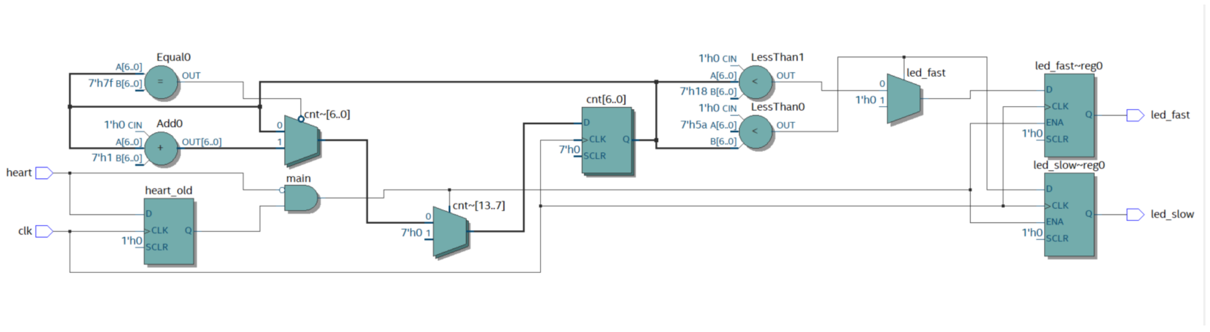
This window contains two important pieces of information :

— The number of logic elements used by your program.

— The number of I/O required.

These two pieces of information have to be compatible with the hardware you have ! Otherwise you will have to modify your code to use less resources. You can also, out of curiosity, go and see the other parts of the report, such as the Resource Usage Summary.

Furthermore, it is possible to see the physical implementation of the system. Click on Tools → Netlist Viewers → RTL Viewer.



Here you can see the translation of your code into a logic circuit with registers, multiplexers, etc... For example, the **heart_old** signal led to the addition of a register.

The last step now consists of assigning the signals of our system to inputs/outputs of the component. Only certain I/O of the CPLD are accessible on the development board. It will also be necessary to specify to the software what to do with the unused I/O and finally to indicate which voltage to use.

In this project, there are two types of signals : classical signals and clock signals. A signal is automatically considered as a clock signal by Quartus when you use the **rising_edge** function or other specific functions. You must be able to determine from the code you have written which signal is a clock signal but you can get confirmation in Quartus by going to the Fitter report and then navigating to Resource Section, Control Signal.

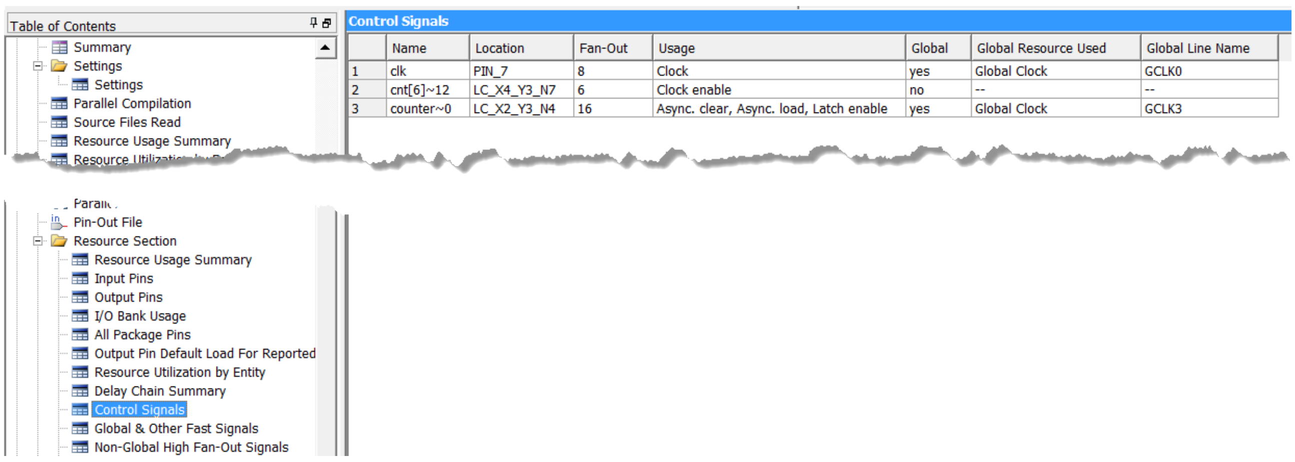


Table of Contents

- Summary
- Settings
  - Settings
- Parallel Compilation
- Source Files Read
- Resource Usage Summary
- Resource Utilizatio
- Paral
- Pin-Out File
- Resource Section
  - Resource Usage Summary
  - Input Pins
  - Output Pins
  - I/O Bank Usage
  - All Package Pins
  - Output Pin Default Load For Reported
  - Resource Utilization by Entity
  - Delay Chain Summary
  - Control Signals
  - Global & Other Fast Signals
  - Non-Global High Fan-Out Signals

Control Signals

	Name	Location	Fan-Out	Usage	Global	Global Resource Used	Global Line Name
1	clk	PIN_7	8	Clock	yes	Global Clock	GCLK0
2	cnt[6]~12	LC_X4_Y3_N7	6	Clock enable	no	--	--
3	counter~0	LC_X2_Y3_N4	16	Async. clear, Async. load, Latch enable	yes	Global Clock	GCLK3

The signals are described in the Usage field.

Now you must tell the software where each of your signals are located on the CPLD. To do this, click on the button (Pin Planner), which opens the window shown in Figure 4.

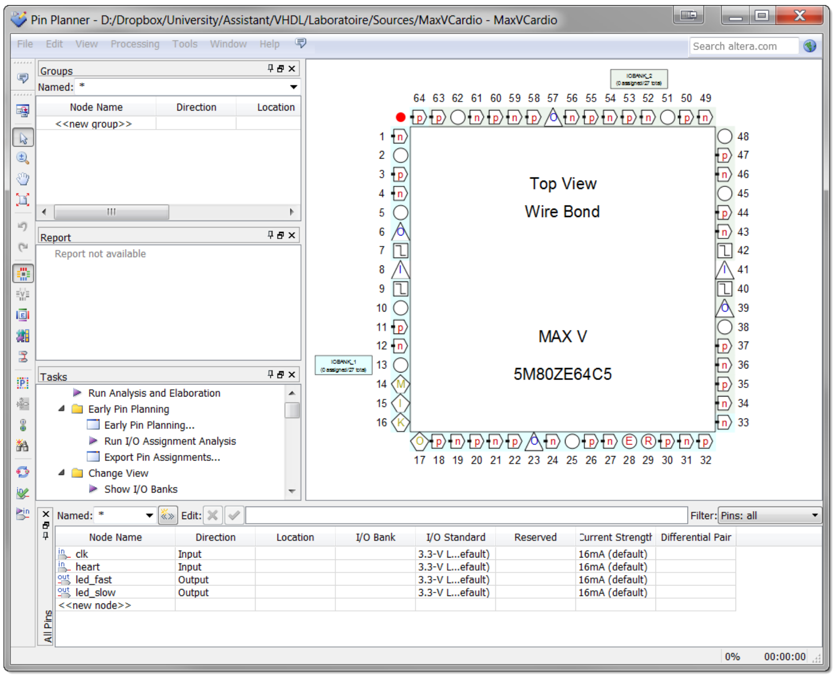
You can move a signal (e.g. clk) to an input on the CPLD. However, please note the following :

— The clocks in your system must be on CPLD clock inputs, symbolised by .

— Some I/O are reserved (such as programming I/O's, represented by) or power supply I/O's (represented by).

Don't forget to refer to the schematic of the electronic board represented in Figure 1 to place your inputs and outputs in the right pins. The two 555 timers of the board are connected directly to the CLK0 and CLK1 inputs of the CPLD. Finally, you can also specify the output voltage of the pin, using the Standard I/O column. Check that everything is in 3.3-V LVTTL (default).

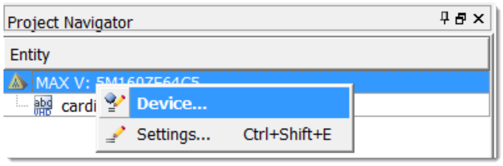
I recommend you to assign **clk** to the fastest clock of the board and **heart** to the slowest. Moreover, assign **led_slow** and **led_fast** to LEDs integrated on the board.


Pin Planner - D:/Dropbox/University/Assistant/VHDL/Laboratoire/Sources/MaxVCardio - MaxVCardio
Groups
Report
Report not available
Tasks
Run Analysis and Elaboration
Early Pin Planning
Early Pin Planning...
Run I/O Assignment Analysis
Export Pin Assignments...
Change View
Show I/O Banks
Top View
Wire Bond
MAX V
5M80ZE64C5
Named: *
Filter: Pins: all

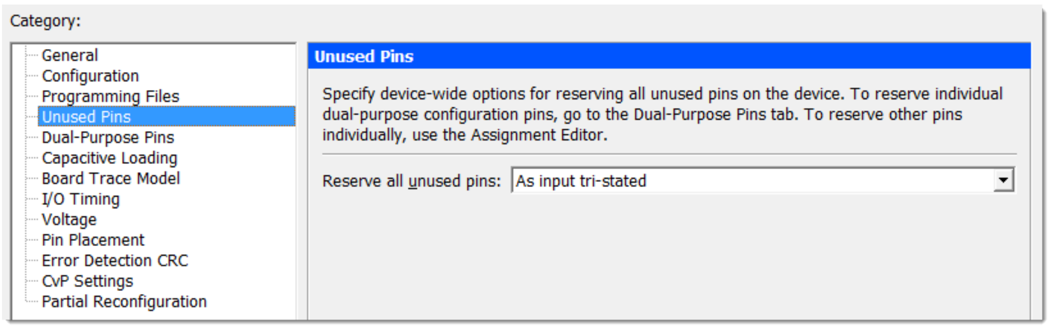
Node Name	Direction	Location	I/O Bank	I/O Standard	Reserved	Current Strength	Differential Pair
clk	Input			3.3-V L...efault)		16mA (default)	
heart	Input			3.3-V L...efault)		16mA (default)	
led_fast	Output			3.3-V L...efault)		16mA (default)	
led_slow	Output			3.3-V L...efault)		16mA (default)	
<<new node>>							

FIGURE 4 – Pin planner

The last thing to do is to tell the software what to do with the unused I/O. By default, the CPLD will impose a voltage of 0 V. However, this may cause problem as there is a risk of conflict with a voltage imposed by a external device (as the 555 timers). Therefore, it's better to change this default behaviour and set all unused pins to Input mode. To do so, right-click on the name of the CPLD in the Project Navigator, and click Device.



In the window that appears, click Device and Pin Options. A new window appears : go to the Unused Pin section. Finally, in the section Reserve all unused pins, choose As input tri-stated.

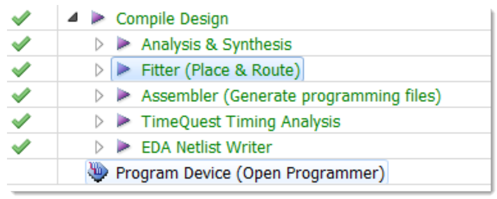


Specify device-wide options for reserving all unused pins on the device. To reserve individual dual-purpose configuration pins, go to the Dual-Purpose Pins tab. To reserve other pins individually, use the Assignment Editor.

Reserve all unused pins: As input tri-stated

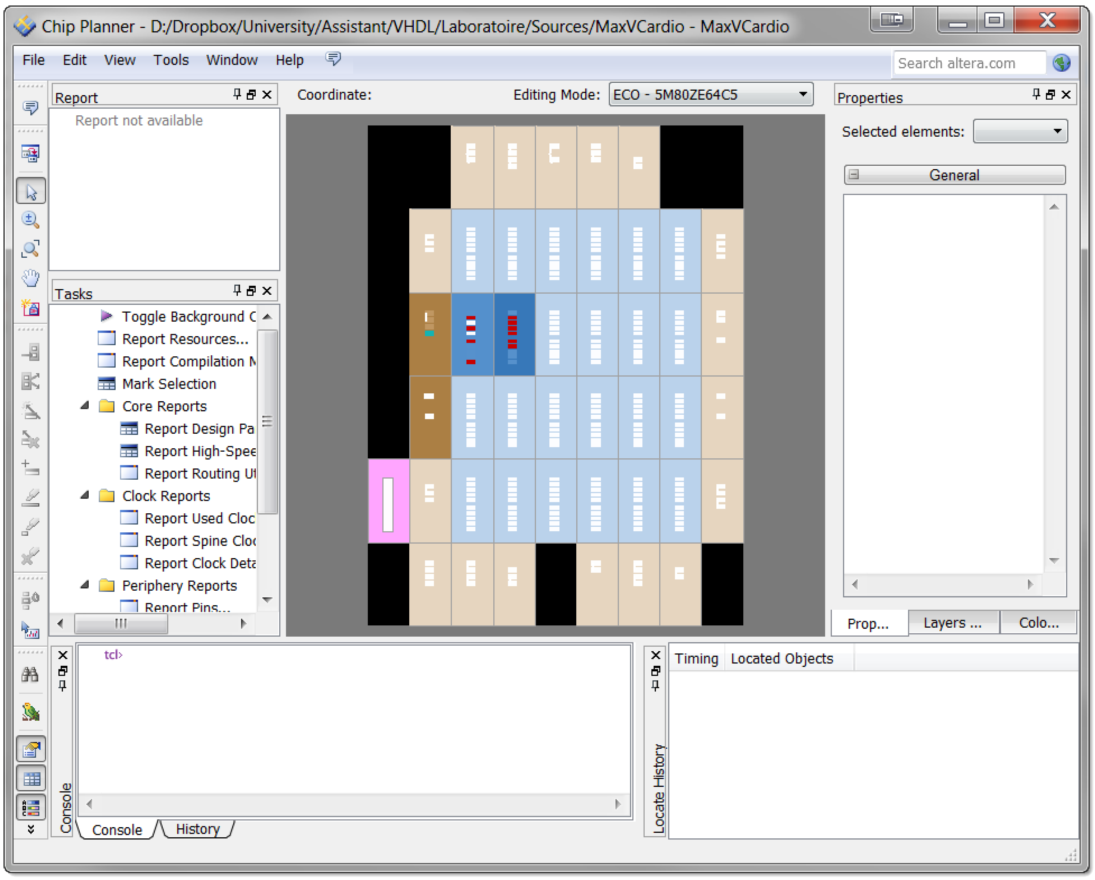
Place & Route

At this stage, your project has been coded, simulated and analysed. All that remains to do is to synthesise it, i.e. to build the logic circuit that can be implemented on the CPLD. Click on the button to launch the complete compilation.

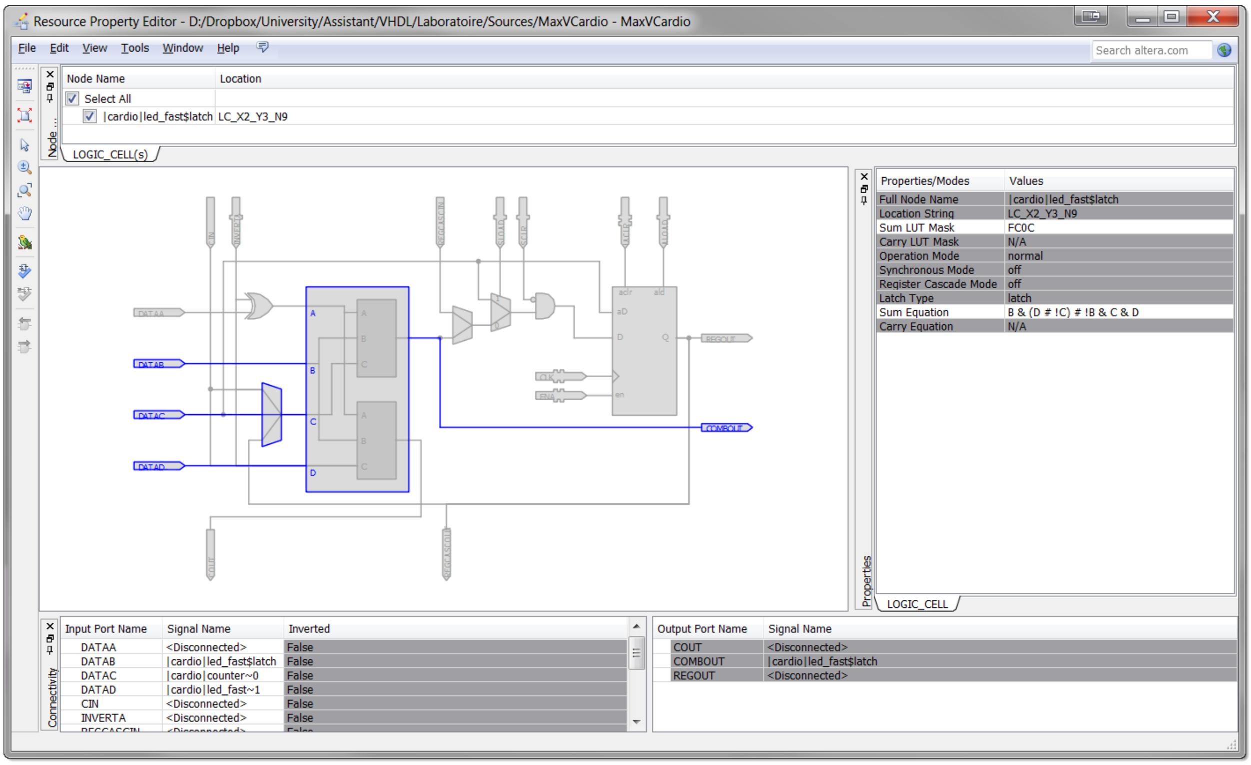


✓ Compile Design
✓ Analysis & Synthesis
✓ Fitter (Place & Route)
✓ Assembler (Generate programming files)
✓ TimeQuest Timing Analysis
✓ EDA Netlist Writer
Program Device (Open Programmer)

At the end of this stage, a number of reports are available. For example, it is possible to see how the code has been implemented on the component. To do so, click on (Chip Planer), which opens the following window :



The darker blue a block is, the more it is used. Thus, in the program created here, only 2 logical units are used. You can go through the program to see exactly how your system has been implemented. you can see the logical equations, registers, etc. with a double click on a block to see details of the implementation.


Resource Property Editor - D:/Dropbox/University/Assistant/VHDL/Laboratoire/Sources/MaxVCardio - MaxVCardio
File Edit View Tools Window Help
Search altera.com
LOGIC_CELL(s)
DATAA
DATAB
DATAC
DATAD
REGOUT
COMBOUT
LOGIC_CELL

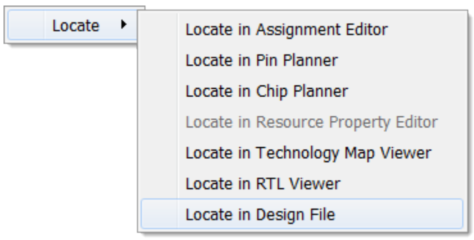
Node Name	Location
Select All	
|cardio|led_fast$latch	LC_X2_Y3_N9

Properties/Modes	Values
Full Node Name	|cardio|led_fast$latch
Location String	LC_X2_Y3_N9
Sum LUT Mask	FC0C
Carry LUT Mask	N/A
Operation Mode	normal
Synchronous Mode	off
Register Cascade Mode	off
Latch Type	latch
Sum Equation	B & (D # !C) # !B & C & D
Carry Equation	N/A

Input Port Name	Signal Name	Inverted
DATAA	<Disconnected>	False
DATAB	|cardio|led_fast$latch	False
DATAC	|cardio|counter~0	False
DATAD	|cardio|led_fast~1	False
CIN	<Disconnected>	False
INVERTA	<Disconnected>	False
REGCASCIN	<Disconnected>	False

Output Port Name	Signal Name
COUT	<Disconnected>
COMBOUT	|cardio|led_fast$latch
REGOUT	<Disconnected>

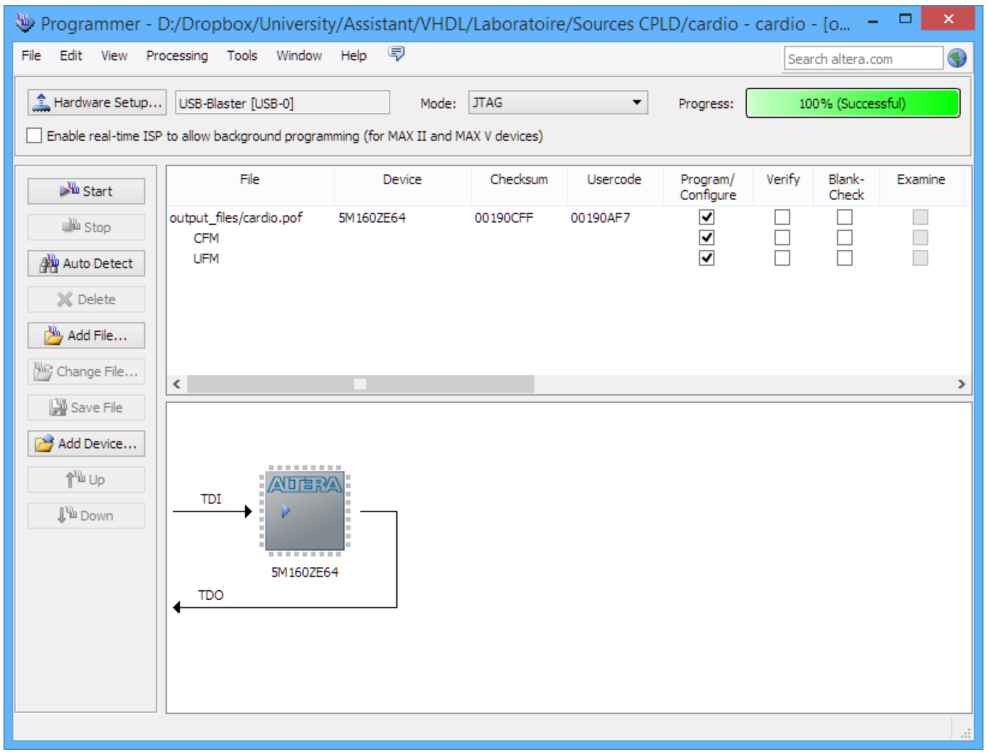
You can see where the item has been declared in the code by right-clicking and then Locate → Locate in Design File.


Locate
Locate in Assignment Editor
Locate in Pin Planner
Locate in Chip Planner
Locate in Resource Property Editor
Locate in Technology Map Viewer
Locate in RTL Viewer
Locate in Design File

In future more complex projects, you will most likely have to do timing analysis, i.e. check the time it takes for the signal to go from the input to the output of the CPLD. Here, there is no need to do this analysis since the system is used at a very slow speed.

Programming of the CPLD

The last step is to program the component. Connect the programmer to your computer. Open the programmer by clicking on the button (Programmer) in the Tools part of Quartus. A new window opens.



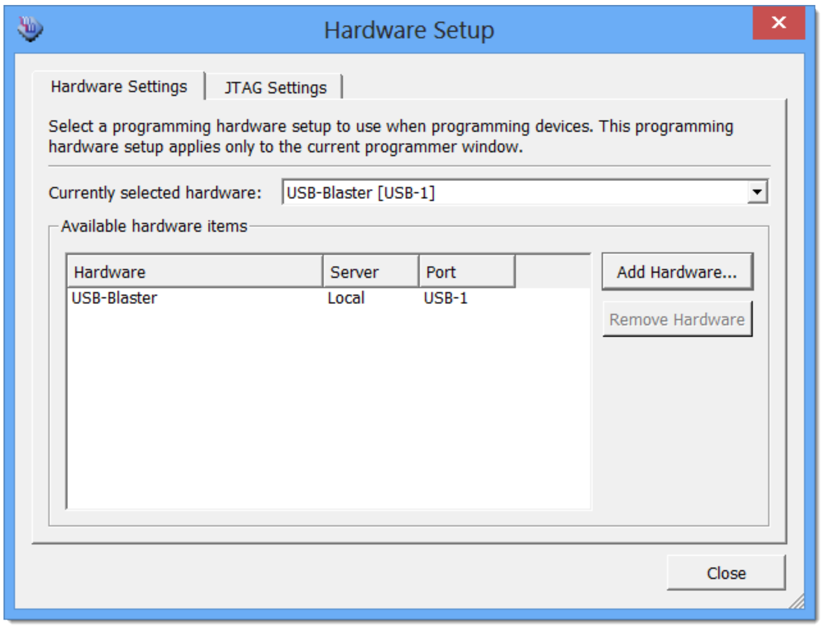
Programmer - D:/Dropbox/University/Assistant/VHDL/Laboratoire/Sources CPLD/cardio - cardio - [o...

USB-Blaster [USB-0] Mode: JTAG Progress: 100% (Successful)

☐ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine
output_files/cardio.pof	5M160ZE64	00190CFF	00190AF7	☑	☐	☐	☐
CFM				☑	☐	☐	☐
UFM				☑	☐	☐	☐

Click the button Hardware Setup... to select and configure your programmer. A new window will open, choose USB-Blaster in Currently Selected Hardware.



Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: USB-Blaster [USB-1]

Available hardware items

Hardware	Server	Port
USB-Blaster	Local	USB-1

Press Close to return to the programming window. Check the Program/Configure boxes. Finally, click on start. If everything went well, your component is programmed (don't forget to supply your board with the battery before clicking on start) !

Set the CLK0 potentiometer fully to the left (when you are facing the card, the notch should be directed to the left), and move the cursor of the other potentiometer to change the heart beat. Observe the changes.

In the laboratory, you would be able to measure the frequency range of the clocks thanks to an oscilloscope. The frequency of the **clk0** signal located in the pin 7 of the CPLD is between 50 Hz and 266 Hz. The frequency of the **clk1** signal located in the pin 9 of the CPLD is between 1.24 Hz and 90 Hz.

3 Exercise

For this exercise, I invite you to perform an improved version of the cardiogram. Indeed, instead of one LED indicating a too low heart rate and one LED indicating a too high heart rate, you will realize a system with 8 LEDs. The number of LEDs lit must be proportional to the heart rate. You can choose the intermediate heart rates, but the evolution of the LEDs lighting up must be proportional to the heart rate. In addition, the two danger thresholds (approx. 45bpm and 200bpm) should be indicated with red LEDs as shown in the Figure 5.

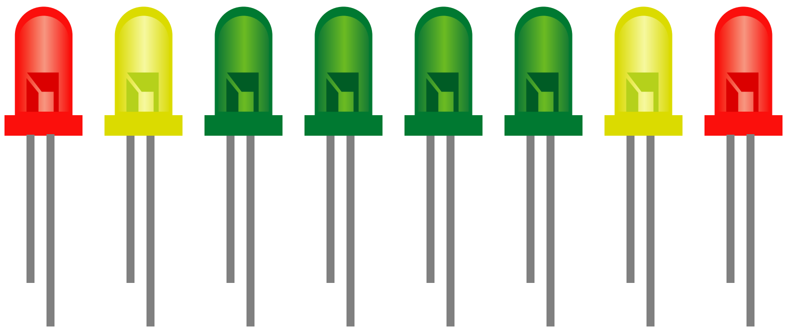


FIGURE 5 – LED configuration