KTH, Dept. Of Electronics
System (ES), School of ICT

Last Revision:
November 8, 2014

Laboration 4

IL2225 Embedded Hardware Design in ASIC and FPGA

Introduction to High Level Synthesis

Name :

Personal Number :

Assistant :

Date :

NOTE: Make it sure that you have downloaded the latest version of lab manual from the course website.

1. Introduction

High level synthesis promises to be one of the solutions to cope with significant increase in the demand for design productivity beyond the start of the art methods. It also offers possibilities to explore the design space in an efficient way by dealing with higher abstraction levels and fast implementation ways to prove the feasibility of algorithms.

In this lab we will try to explore these different possibilities using a HLS tool. We will demonstrate and you will learn about benefits of implementing a design at higher level of abstraction and controlling the generation of RTL using various HLS constraints.

2. G.A.U.T.

G.A.U.T. is a HLS (High Level Synthesis) tool developed at the Universite de Bretagne Sud (UB). Lab-STICC laboratory. G.A.U.T. generates RTL descriptions from a pure bit-accurate algorithmic specification described in C/C++ language. G.A.U.T. fits design flows upstream and targets FPGA and ASICs. The basic principle followed by the tool is shown in the following figure.

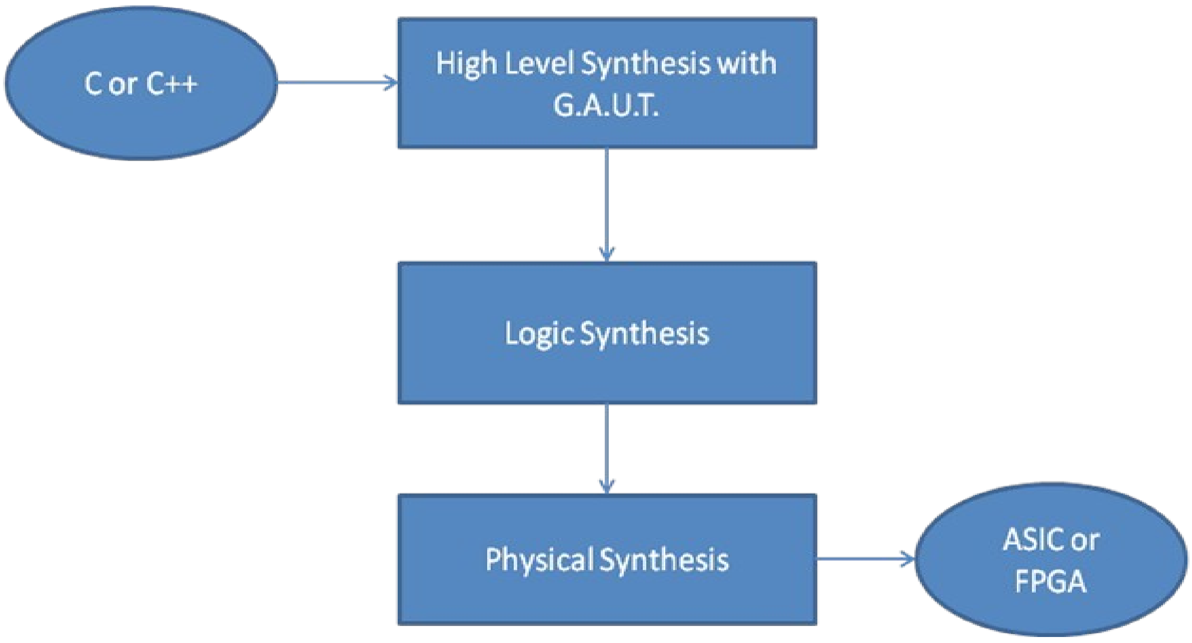


Figure 1 : G.A.U.T. Design Flow

Algorithm description is converted into RTL description during the high level synthesis phase and the RTL description is converted in to the logical description suitable for ASIC/FPGA in the next phase. A physical synthesis placing and routing logical gates on a matrix of sites (ASIC made out of standard cell) or placing and routing CLB on a FPGA (array of CLBs) having a routing topology dynamically reconfigurable.

In this lab we will only be using High Level Synthesis aspect of G.A.U.T. For further details you can consult [1] [2].

a. G.A.U.T. Interface for HLS

G.A.U.T. is software which has inputs and outputs in the shape of files as well as control options for the HLS. These control options are expressed in the command line or by the means of a graphic interface.

The inputs are:

- a file containing the algorithm to be synthesized: this is a .c or .cpp file
- a library of operators characterized for a given technology target: this is a .lib file.

The outputs are :

- An option file containing VHDL RTL code : this is a .vhd file. It has the same name as the .c, but with a .vhd suffix.
- A textual description of the chronogram (timing diagram) of the I/O of the circuit: the .mem file.
- Other files generated to interface G.A.U.T. with other tools for synthesis.

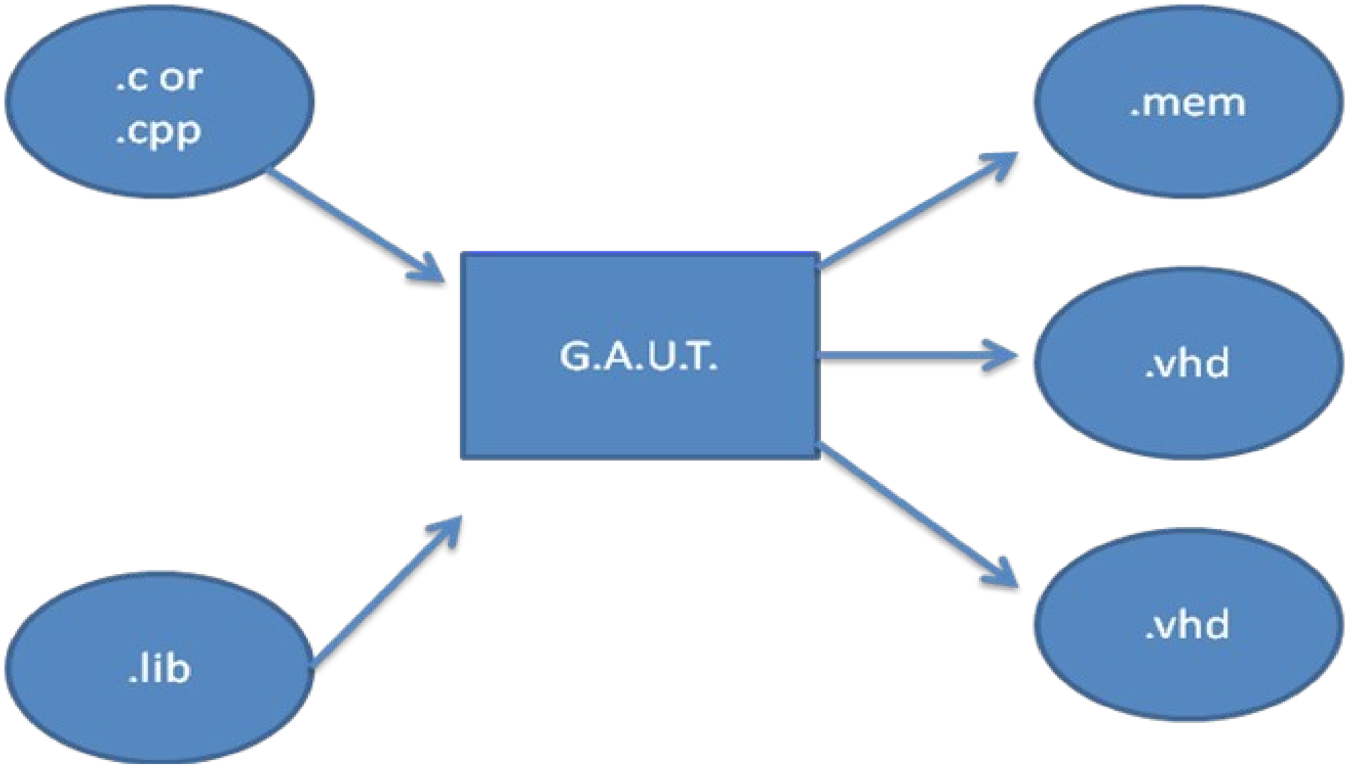


Figure 2 : G.A.U.T. Interface

b. G.A.U.T. HLS Flow

G.A.U.T. takes the functional description of a circuit in form of C code which generates a CDFG. HLS phase takes this CDFG and generates a VHDL RTL after applying scheduling, allocation and binding. Different constraints can be applied on scheduling, allocation and binding phase to explore design space effectively and efficiently. In this lab you will learn to apply these constraints and view the effects of these steps for variations.

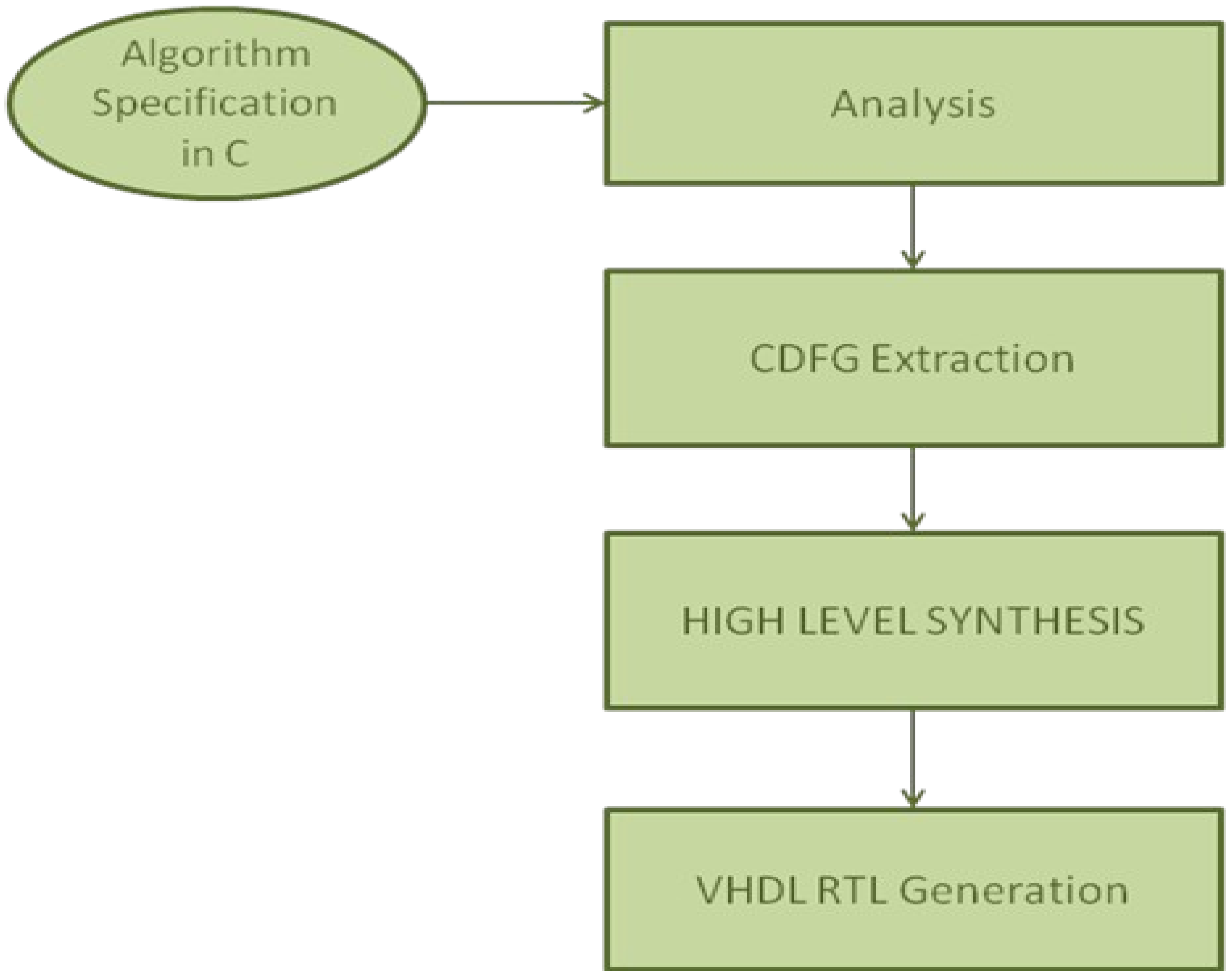


Figure 3 : G.A.U.T. HLS Flow

c. Structure of the synthesized circuit

A circuit synthesized by G.A.U.T. has the following structure:

- A processing unit (PU)
- control signals: clk, reset, enable. When enable = ‘0’, circuit is frozen.
- IO signals connected to external buses with the circuit. “inputs” and “outputs” are the IO as specified in the port of VHDL entity. The external buses do not make parts of the synthesized circuit. **They are there to convey the data between the circuit and external unit storage.** The file .mem describes the scheduling constraints on the data.

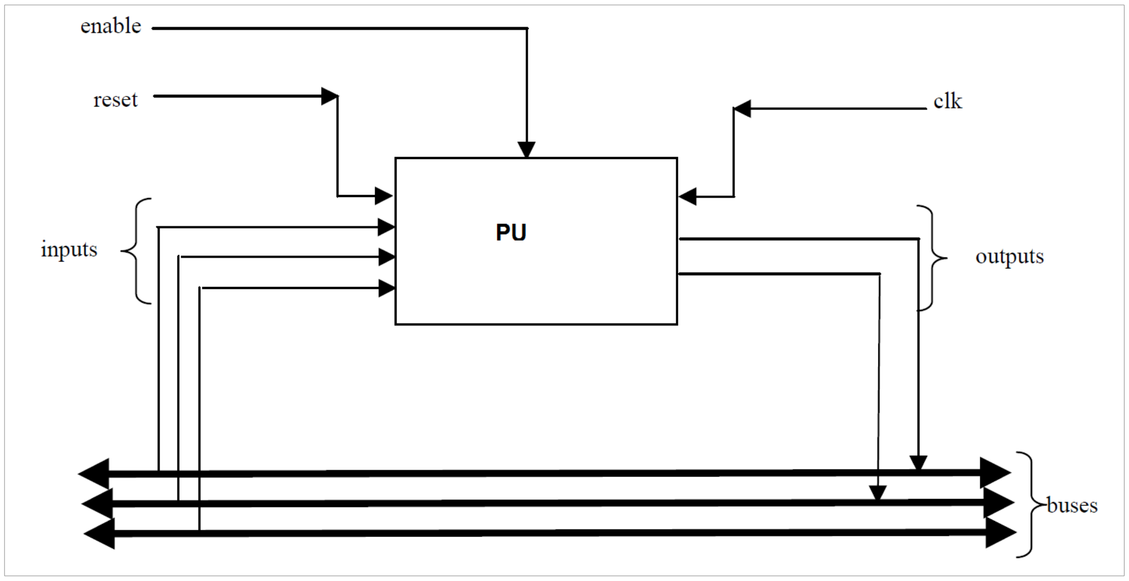


Figure 4: Synthesized Circuit using G.A.U.T.

G.A.U.T. can also synthesize the circuit using the Memory unit. It works on exactly the same principle as synthesizing without memory unit. All aging variables and both static and constant variables are stored in the memory, if circuit is synthesized with the memory unit.

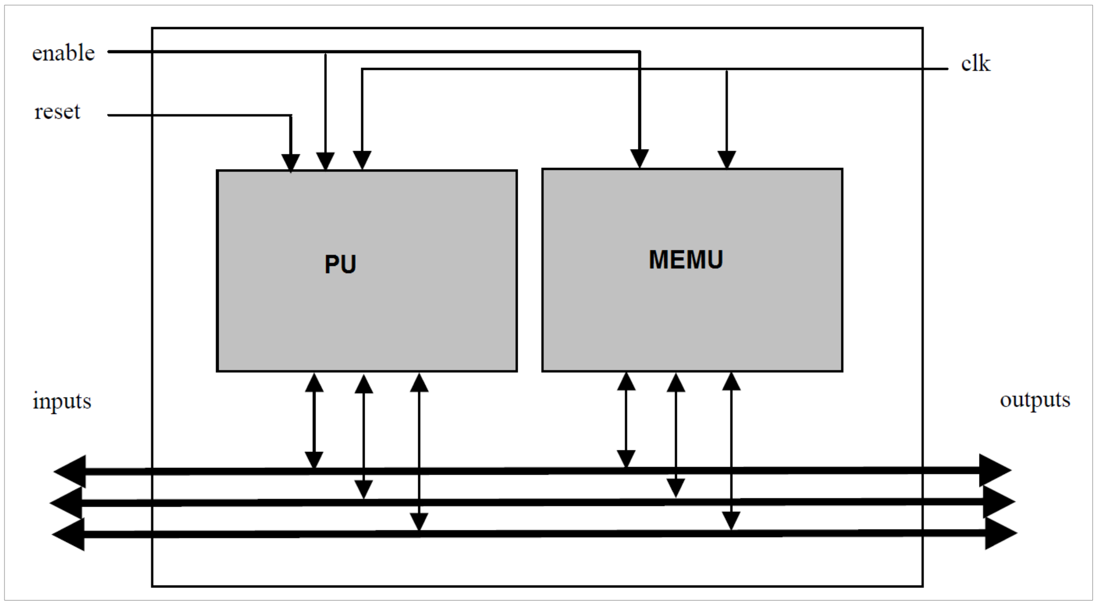


Figure 5: Synthesized Circuit with MEMU using G.A.U.T.

3. Directory Structure

The directory structure shown in Figure 6 is used for this lab course. Assume that you are working in the lab1/ directory, and then this is your current working directory.

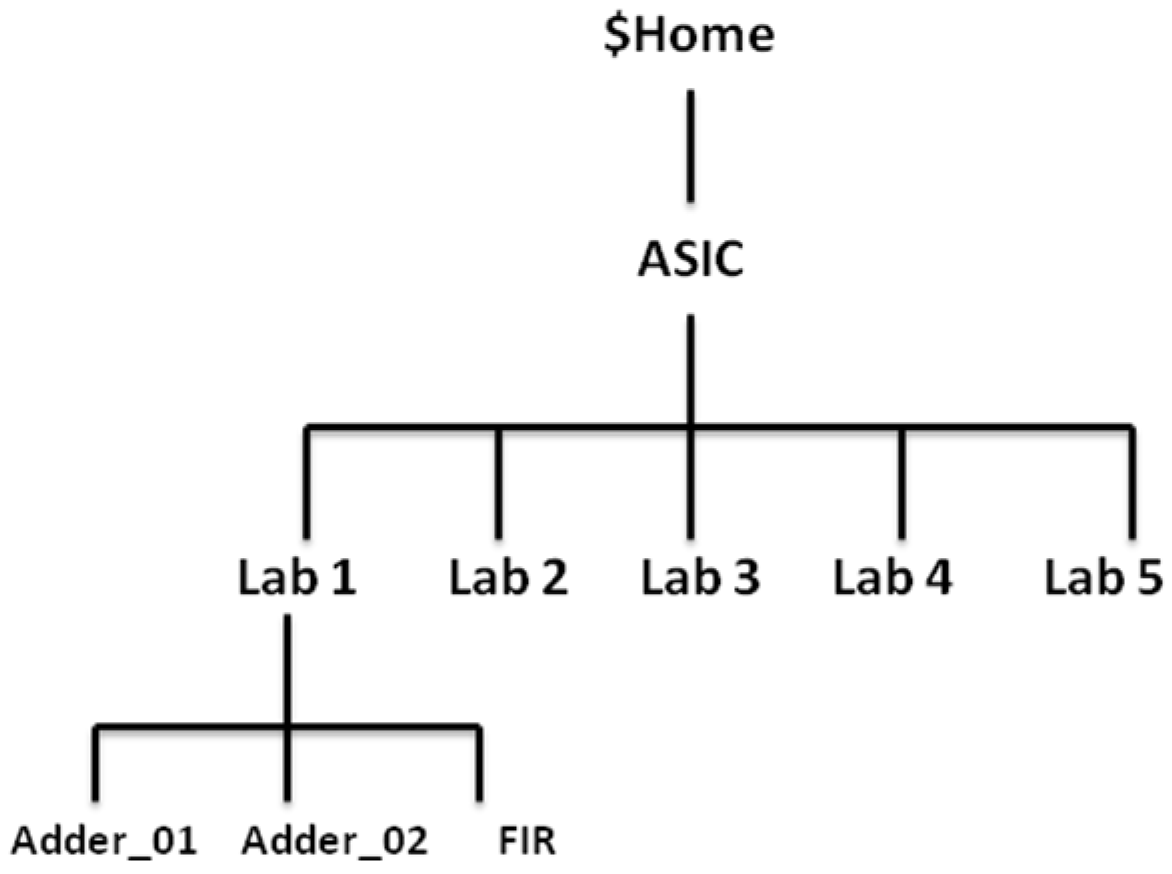


Figure 6 : Directory Structure

4. An Example using G.A.U.T.

Objective:

The primary objective is to give you a quick, hands on tour of the HLS process using G.A.U.T. Upon completion of this exercise you will be able to describe the basic steps involved in the synthesis process and how to explore the design space using various HLS constraints.

Set of Tasks:

- If you are working on a windows machine then simply goto :

  Start -> Programs -> G.A.U.T.

- If you are working on a linux machine then perform the following steps to start the G.A.U.T. software:
  - Open terminal

- type "cd /afs/ict.kth.se/pkg/gaut/2.4.3"
- type ./Gaut &

Figure 7 shows the main interface of the tool with eight colored boxes. Every box has a specific purpose and will be described briefly later.

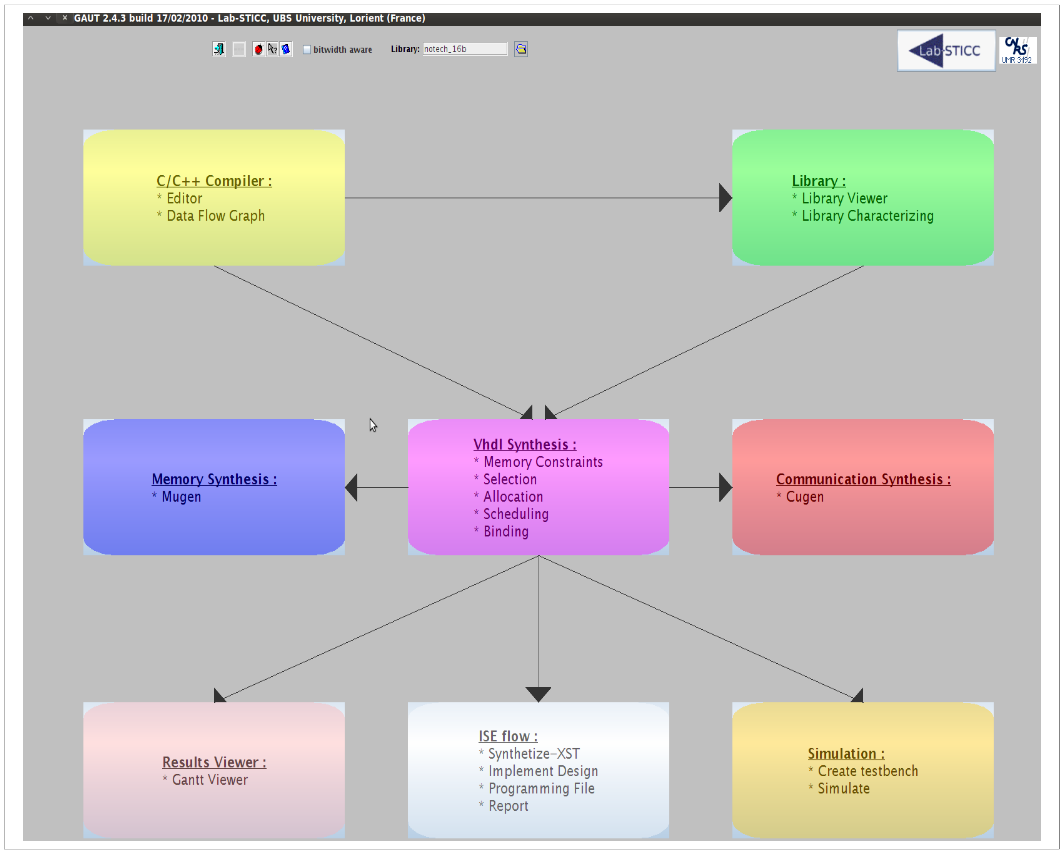
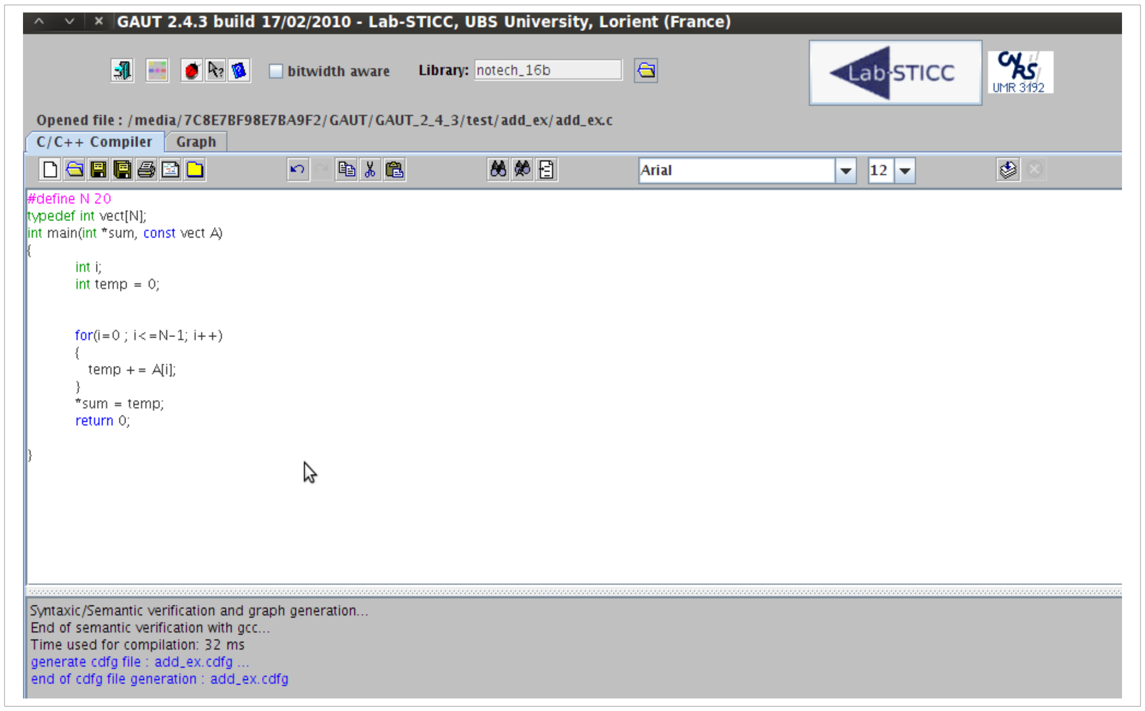


Figure 7 : G.A.U.T. Main Window

a. STEP1 : Compiling the C Code

We will start by compiling a C code using G.A.U.T. This is a simple C code which adds 20 numbers and store the result in the variable "sum".

- **Click on the Yellow box stating "C/C++ Compiler".** (This phase makes it sure that the algorithm specified in C is correct)
- **Load the "adder_1.c" file by clicking on the "open" icon and following the Path :** "$HOME/ASIC/Lab1/adder_1/adder_1.c" where HOME is the path to your home folder.
- **Compile the code by clicking on the "compile button" .**


GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)
bitwidth aware
Library: notech_16b
Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex.c
C/C++ Compiler
Graph
Arial
12

```
#define N 20
typedef int vect[N];
int main(int *sum, const vect A)
{
        int i;
        int temp = 0;


        for(i=0 ; i<=N-1; i++)
        {
          temp += A[i];
        }
        *sum = temp;
        return 0;

}
```

```
Syntaxic/Semantic verification and graph generation...
End of semantic verification with gcc...
Time used for compilation: 32 ms
generate cdfg file : add_ex.cdfg ...
end of cdfg file generation : add_ex.cdfg
```

Figure 8 : C Editor/Compiler Window

- **Click on the graph tab and look at the graph by loading "adder_1.cdfg".**

  This is the cdfg corresponding to the design presented in C code. The cdfg contains 19 additions, data values stored in variables A[0] … A[19] and the variables temp, temp 00001 … temp 00019. These variables came from loop unrolling of the code.

  It should also be noted that the technological target library "notech_16b" is a factitious library. Different sets of technological libraries can be characterized by G.A.U.T. and are beyond the scope of this lab. Interested readers can consult [1] for further details.

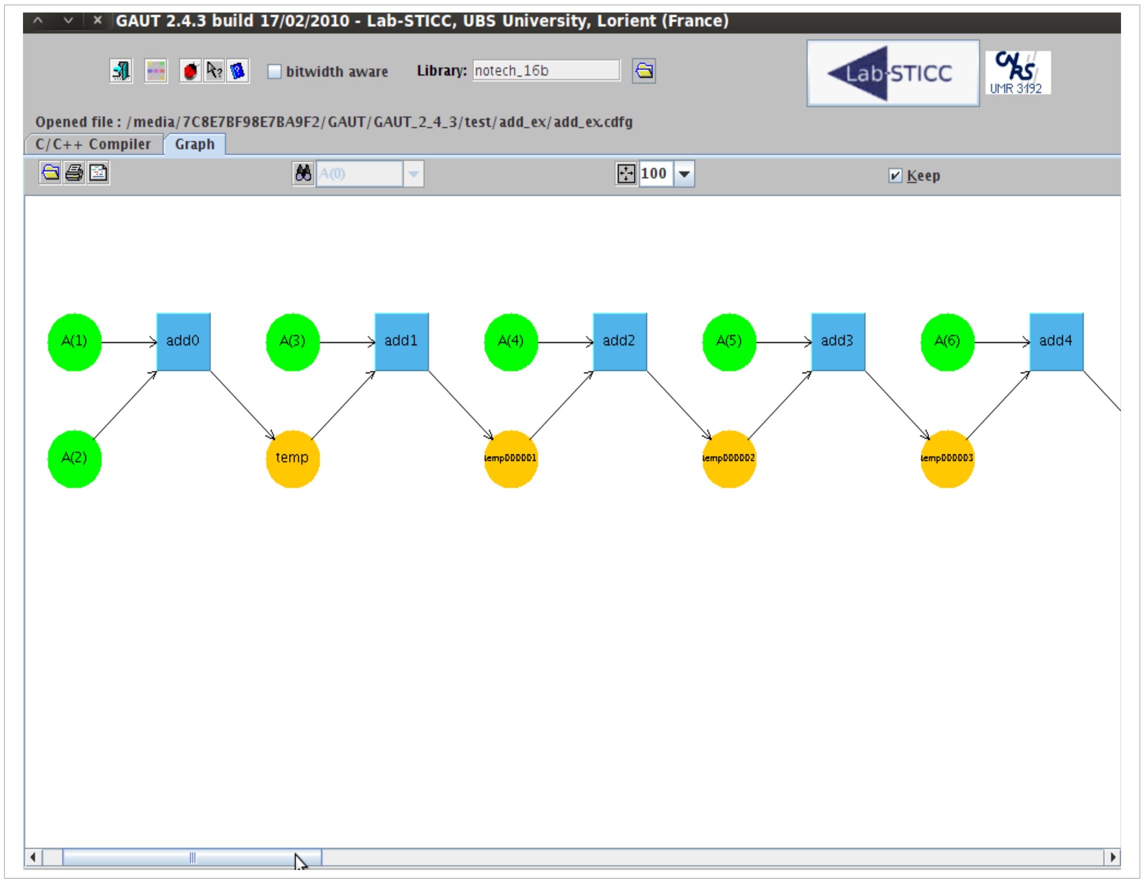


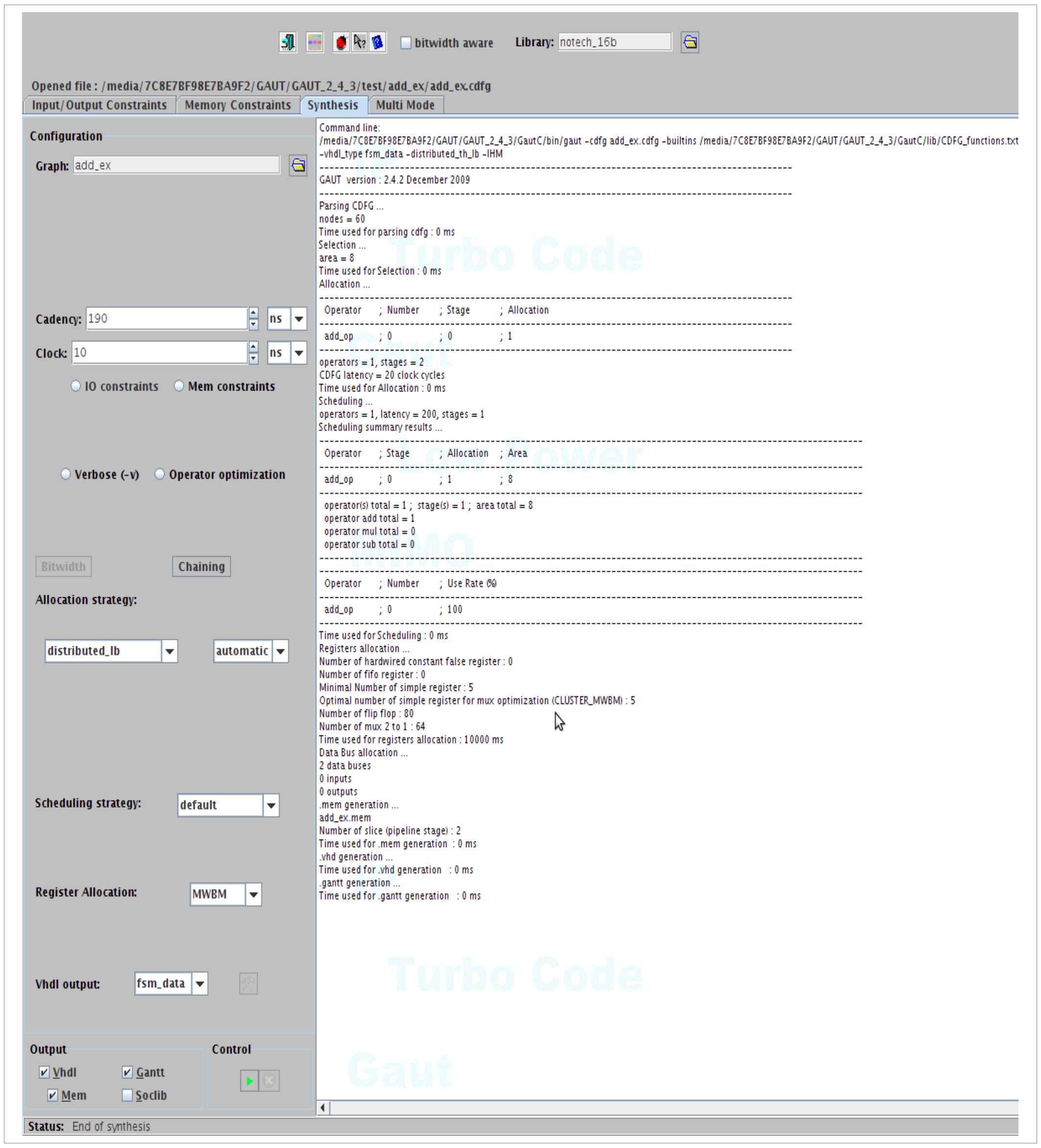
Figure 9 : CDFG of the C code

- **Click on the button and you will come back to the main window as shown in the Figure 7.**

b. STEP 2 : Synthesis procedure

We will apply different constraints on this design for the synthesis purpose. One can use these constraints the change the scheduling, allocation and binding parameters in multiple ways.

- **Click on the Purple box "VHDL synthesis".**



Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex.cdfg

Graph: add_ex

Cadency: 190 ns

Clock: 10 ns

Allocation strategy: distributed_lb automatic

Scheduling strategy: default

Register Allocation: MWBM

Vhdl output: fsm_data

Output: [✓ Vhdl] [✓ Gantt] [✓ Mem] Soclib

Status: End of synthesis

```
Command line:
/media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/GautC/bin/gaut -cdfg add_ex.cdfg -builtins /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/GautC/lib/CDFG_functions.txt
-vhdl_type fsm_data -distributed_th_lb -IHM
---------------------------------------------------------------------------------
GAUT  version : 2.4.2 December 2009
---------------------------------------------------------------------------------
Parsing CDFG ...
nodes = 60
Time used for parsing cdfg : 0 ms
Selection ...
area = 8
Time used for Selection : 0 ms
Allocation ...
---------------------------------------------------------------------------------
 Operator    ; Number     ; Stage       ; Allocation
---------------------------------------------------------------------------------
 add_op      ; 0          ; 0           ; 1
---------------------------------------------------------------------------------
operators = 1, stages = 2
CDFG latency = 20 clock cycles
Time used for Allocation : 0 ms
Scheduling ...
operators = 1, latency = 200, stages = 1
Scheduling summary results ...
-------------------------------------------------------------------------------------------
 Operator    ; Stage      ; Allocation  ; Area
-------------------------------------------------------------------------------------------
 add_op      ; 0          ; 1           ; 8
-------------------------------------------------------------------------------------------
 operator(s) total = 1 ;  stage(s) = 1 ;  area total = 8
 operator add total = 1
 operator mul total = 0
 operator sub total = 0
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
 Operator    ; Number     ; Use Rate (%)
-------------------------------------------------------------------------------------------
 add_op      ; 0          ; 100
-------------------------------------------------------------------------------------------
Time used for Scheduling : 0 ms
Registers allocation ...
Number of hardwired constant false register : 0
Number of fifo register : 0
Minimal Number of simple register : 5
Optimal number of simple register for mux optimization (CLUSTER_MWBM) : 5
Number of flip flop : 80
Number of mux 2 to 1 : 64
Time used for registers allocation : 10000 ms
Data Bus allocation ...
2 data buses
0 inputs
0 outputs
.mem generation ...
add_ex.mem
Number of slice (pipeline stage) : 2
Time used for .mem generation  : 0 ms
.vhd generation ...
Time used for .vhd generation   : 0 ms
.gantt generation ...
Time used for .gantt generation   : 0 ms
```

Figure 10 : Synthesis options

Figure 10 shows the main windows for performing synthesis. As described before, there are various options available. We will discuss few of them for this exercise and rest will be explained in later sections.

Graph : The synthesis part takes the CDFG as the input to start performing the synthesis. This is the same graph which was generated in the previous step.

Cadency : This is the rate of arrival of the sets of data inputs (sampling rate, iteration interval). In other words, this is the throughput of the design. Cadency is one of the primary constraints for synthesizing the design using G.A.U.T.

Clock : You can use this field to specify the desired clock period of the RTL to be generated.

VHDL output **:** The tool can generate different styles of VHDL code. It should be noted that each style of VHDL performs the same type of functionality.

For this exercise set the following fields :

- **Set the cadency value as 190 ns.**
- **Set the clock period as 10 ns.**
- **Set the VHDL output type as "fsm_regs". Keep the rest of the values as default**
- **Synthesize the design by pressing the "control" button.**

Some of the information presented in the report is as follows:

The CDFG parsing step

Parsing CDFG . . . nodes = 60

The Allocation step

Allocation … Operators = 1, stages = 2

CDFG Latency = 20 clock cycles

The scheduling step

Scheduling … Operators = 1, Latency = 200 , stages = 1

Register allocation

Bus Allocation . . . 2 data buses

Question 01 : Can you view the synthesis report generated by G.A.U.T.? Is this is a serial or parallel solution?

____

____

Question 02 : What is the latency of this design? How may clock steps were taken by the tool to complete the job?

____

____

Question 03 : How many adders are used by the tool for generating this solution? What is the minimum delay of a single adder? (You may need to check the Library Viewer)

____

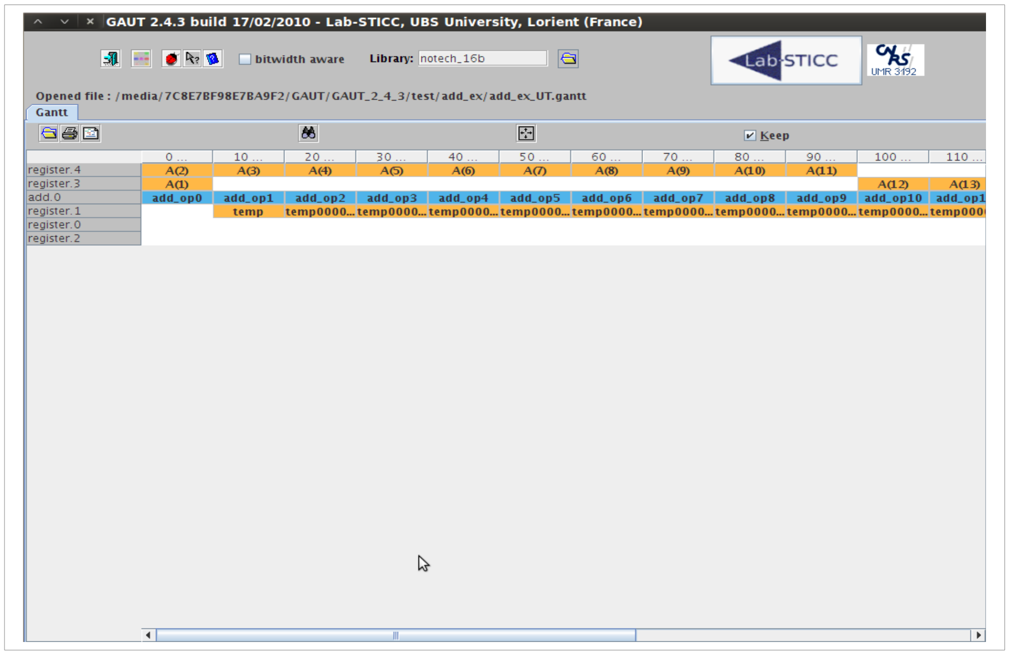
____

- **Click on the button and you will come back to the main window as shown in the Figure 8.**

c. STEP 3 : Viewing the Results

Pink box "Results Viewer", is use to generate Gantt charts of the scheduled operations. This chart explains the results of the scheduling steps. It also contains information about the contents of the circuit in terms of operators and registers.

- **Click on the "Results Viewer" and open the file "adder_1_UT*.gantt".** Horizontally the clue color defines the execution of the operations and the orange color defines the variables and registers in which they are stored. Vertically the names of the operators and registers are defined. (*UT has the same as PU)



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

bitwidth aware Library: notech_16b

Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex_UT.gantt

Gantt

✓ Keep

	0 ...	10 ...	20 ...	30 ...	40 ...	50 ...	60 ...	70 ...	80 ...	90 ...	100 ...	110 ...
register.4	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)	A(10)	A(11)		
register.3	A(1)										A(12)	A(13)
add.0	add_op0	add_op1	add_op2	add_op3	add_op4	add_op5	add_op6	add_op7	add_op8	add_op9	add_op10	add_op1
register.1		temp	temp0000...	temp0000...	temp0000...	temp0000...	temp0000...	temp0000...	temp0000...	temp0000...	temp0000...	temp000
register.0												
register.2												

Figure 11 : GANTT Chat of the synthesized design

Question 04 : How can you co relate your synthesized design with this gantt chart ?

____

- **(OPTIONAL) Open the following file (using a text editor such as gedit or notepad) :**

  "$Home/ASIC/Lab1/adder_1/adder_1.mem".

  This file shows the I/O chronogram and the temporal access conflicts. The contents of this file tells us :

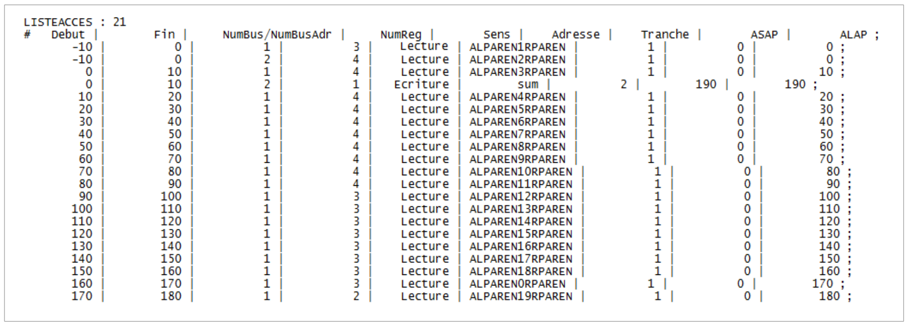
-10ns to 0ns	:	A1 is presented to data bus 1
-10ns to 0ns	:	A2 is presented to data bus 2
0ns to 10ns	:	A3 is presented on data bus 1

  .

  .

  .

  "lecture" means "Read" and "Ecriture" means "Write". This set of information can be used resolving temporal conflicts when synthesizing using memory.



```
LISTEACCES : 21
#   Debut |        Fin |     NumBus/NumBusAdr |     NumReg |       Sens |     Adresse |     Tranche |        ASAP |       ALAP ;
      -10 |          0 |            1 |            3 |    Lecture | ALPAREN1RPAREN |         1 |          0 |          0 ;
      -10 |          0 |            2 |            4 |    Lecture | ALPAREN2RPAREN |         1 |          0 |          0 ;
        0 |         10 |            1 |            4 |    Lecture | ALPAREN3RPAREN |         1 |          0 |         10 ;
        0 |         10 |            2 |            1 |   Ecriture |        sum |         2 |        190 |        190 ;
       10 |         20 |            1 |            4 |    Lecture | ALPAREN4RPAREN |         1 |          0 |         20 ;
       20 |         30 |            1 |            4 |    Lecture | ALPAREN5RPAREN |         1 |          0 |         30 ;
       30 |         40 |            1 |            4 |    Lecture | ALPAREN6RPAREN |         1 |          0 |         40 ;
       40 |         50 |            1 |            4 |    Lecture | ALPAREN7RPAREN |         1 |          0 |         50 ;
       50 |         60 |            1 |            4 |    Lecture | ALPAREN8RPAREN |         1 |          0 |         60 ;
       60 |         70 |            1 |            4 |    Lecture | ALPAREN9RPAREN |         1 |          0 |         70 ;
       70 |         80 |            1 |            4 |    Lecture | ALPAREN10RPAREN |         1 |          0 |         80 ;
       80 |         90 |            1 |            4 |    Lecture | ALPAREN11RPAREN |         1 |          0 |         90 ;
       90 |        100 |            1 |            3 |    Lecture | ALPAREN12RPAREN |         1 |          0 |        100 ;
      100 |        110 |            1 |            3 |    Lecture | ALPAREN13RPAREN |         1 |          0 |        110 ;
      110 |        120 |            1 |            3 |    Lecture | ALPAREN14RPAREN |         1 |          0 |        120 ;
      120 |        130 |            1 |            3 |    Lecture | ALPAREN15RPAREN |         1 |          0 |        130 ;
      130 |        140 |            1 |            3 |    Lecture | ALPAREN16RPAREN |         1 |          0 |        140 ;
      140 |        150 |            1 |            3 |    Lecture | ALPAREN17RPAREN |         1 |          0 |        150 ;
      150 |        160 |            1 |            3 |    Lecture | ALPAREN18RPAREN |         1 |          0 |        160 ;
      160 |        170 |            1 |            3 |    Lecture | ALPAREN0RPAREN |         1 |          0 |        170 ;
      170 |        180 |            1 |            2 |    Lecture | ALPAREN19RPAREN |         1 |          0 |        180 ;
```

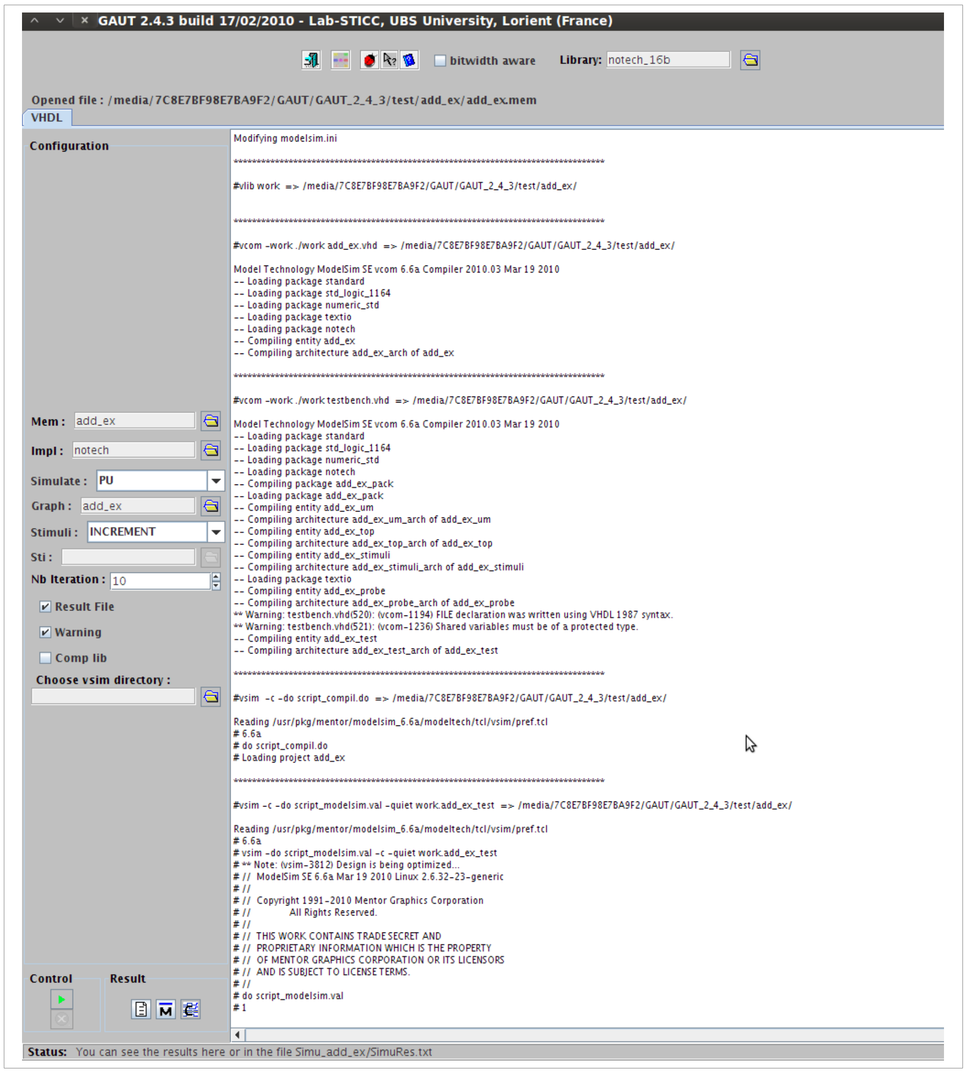
d. STEP 4: Simulating the Design

The tool also generates a VHDL RTL of the synthesized design in the working folder.

- **Open the VHDL file which is located at "ASIC\Lab1\adder_1\adder_1.vhd".**

Question 05a : Can you correlate your VHDL with the synthesis report and gantt charts? Are the number of states are matching the number of clock steps ?

- **Click the orange button to simulate the design using Modelsim.**
- **Keep the default values and simulate the design by clicking on the "control” button.**


GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)
bitwidth aware
VHDL
Configuration
Control
Result

Library: notech_16b

Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex.mem

Mem : add_ex
Impl : notech
Simulate : PU
Graph : add_ex
Stimuli : INCREMENT
Sti :
Nb Iteration : 10
[✓] Result File
[✓] Warning
[ ] Comp lib
Choose vsim directory :

```
Modifying modelsim.ini

********************************************************************************

#vlib work  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/


********************************************************************************

#vcom -work ./work add_ex.vhd  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/

Model Technology ModelSim SE vcom 6.6a Compiler 2010.03 Mar 19 2010
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package numeric_std
-- Loading package textio
-- Loading package notech
-- Compiling entity add_ex
-- Compiling architecture add_ex_arch of add_ex

********************************************************************************

#vcom -work ./work testbench.vhd  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/

Model Technology ModelSim SE vcom 6.6a Compiler 2010.03 Mar 19 2010
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package numeric_std
-- Loading package notech
-- Compiling package add_ex_pack
-- Loading package add_ex_pack
-- Compiling entity add_ex_um
-- Compiling architecture add_ex_um_arch of add_ex_um
-- Compiling entity add_ex_top
-- Compiling architecture add_ex_top_arch of add_ex_top
-- Compiling entity add_ex_stimuli
-- Compiling architecture add_ex_stimuli_arch of add_ex_stimuli
-- Loading package textio
-- Compiling entity add_ex_probe
-- Compiling architecture add_ex_probe_arch of add_ex_probe
** Warning: testbench.vhd(520): (vcom-1194) FILE declaration was written using VHDL 1987 syntax.
** Warning: testbench.vhd(521): (vcom-1236) Shared variables must be of a protected type.
-- Compiling entity add_ex_test
-- Compiling architecture add_ex_test_arch of add_ex_test

********************************************************************************

#vsim  -c -do script_compil.do  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/

Reading /usr/pkg/mentor/modelsim_6.6a/modeltech/tcl/vsim/pref.tcl
# 6.6a
# do script_compil.do
# Loading project add_ex

********************************************************************************

#vsim -c -do script_modelsim.val -quiet work.add_ex_test  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/

Reading /usr/pkg/mentor/modelsim_6.6a/modeltech/tcl/vsim/pref.tcl
# 6.6a
# vsim -do script_modelsim.val -c -quiet work.add_ex_test
# ** Note: (vsim-3812) Design is being optimized...
# //  ModelSim SE 6.6a Mar 19 2010 Linux 2.6.32-23-generic
# //
# //  Copyright 1991-2010 Mentor Graphics Corporation
# //            All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPERTY
# //  OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
# //  AND IS SUBJECT TO LICENSE TERMS.
# //
# do script_modelsim.val
# 1
```

Status: You can see the results here or in the file Simu_add_ex/SimuRes.txt

Figure 12 : GANTT Chart of the synthesized design

- **To view the simulation, click on the "Modelsim" button available at the bottom of the tool. This part will automatically generate the testbench for verifying the design.**
- **Verify your simulations using the generated gantt chart and correlate them.**

NOTE: if the modelsim simulator does not start, you have to specify the path of the vsim exacutable, which is /afs/it.kth.se/pkg/mentor/modelsim/6.5/modeltech/bin/vsim

Question 05b : Explain to the teacher how do you correlate waveform and gantt chart ?

____

____

TASKS FOR THE LABS

TASK 1

Objective

In the previous section you learnt about the basic usage of G.A.U.T. and synthesized an example C code. In this section you'll be asked to perform synthesis on different designs while varying various constraints.

Set of Tasks

- **Re-synthesize this design by setting the cadency constraint to 140 ns and clock period to 10 ns.**

Question 06 : How many adders are generated this time?

____

____

Question 07 : Read the synthesis output of the tool and report the number of clock steps ? Is this information correct ? If not then how many clock steps should have taken to the tool to implement this design by using the number of adders reported in the previous question?

____

____

- **View the gantt charts "adder_1_UT" and "adder_1_UT_PIPE" generated by the tool.**

Question 08 : What is the difference between the two charts

____

____

- **Simulate your design and verify the results. This design would produce output after every N cycles.**

TASK 2

Objective

In these set of tasks you will view the effects of varying the cadency constraint on the total gate count of circuit.

Set of Tasks

- **Re-synthesize the same design by setting the cadency constraints to 190 ns, 140 ns, 90 ns, 50 ns and 30 ns.**
- **Fill Table 1 accordingly.**

Apply your previous knowledge gained in Digital Designs to calculate the gate count of the design. You might have to view the generated VHDL and the library viewer for reading the technology file.

Cadency (ns)	**Gate Count**
190	
140	
90	
50	
30	

Table 01 : Gate Count of the Design

Cadency value inferior to the latency value generates pipeline architectures.

TASK 3

Objective

In these set of tasks we will try to elaborate the relationship between latency and area of the design.

Set of Tasks

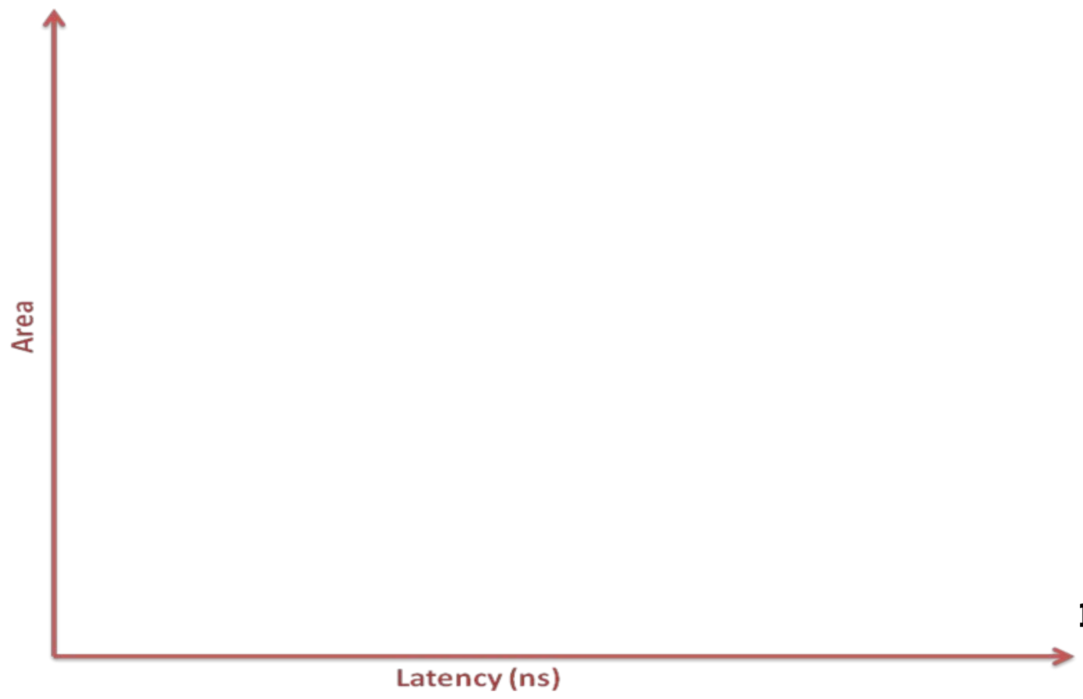
- **From G.A.U.T. main window select "C/C++ Compiler".**
- **Click on the "Open" button and open "adder_2.c" by following the path :** $Home/ASIC/Lab1/adder_2/adder_2.c".
- **"Compile" the code and view the CDFG using the "Graph Tab".**

Question 09 : Is this is a serial or parallel solution? How many adders are used by default?

____

____

- **Synthesize the design by setting the cadency value as 190ns, 140ns, 110ns, 70ns, 30ns, 10ns. Keep rest of settings as default and draw the graph represented below.**



TASK 4

- **Set the cadency value as 110 ns, 50 ns and synthesize the design by using the default settings.**
- **Save the gantt charts by clicking at**
- **Use the following naming conventions :**

  “adder_110ns.html” and “adder_50ns.html”.

TASK 5

Objective

In this phase we will change the scheduling constraints on the same design and observe their effects on the synthesis process.

Set of Tasks

- **Synthesize the design by setting the cadency value as 50 ns and scheduling strategy as "no_pipeline".**
- **View the synthesis report and respective gantt chart.**

Question 10 : How many resources are used and what is the latency of the design? Explain the effects of this scheduling step as compared to the TASK 4?

____

____

____

____

- **Save the gantt chart as "adder_50ns_nopipe.html".**
- **Synthesize the design by setting the cadency value as 50 ns and scheduling strategy as "no_pipeline". Also select "operator optimization".**
- **View the synthesis report and also save the gantt chart in "adder_50ns_nopipe_opt.html".**

Question 11 : Can you explain the effects of operator optimization? What happened as compared to previous design?

____

____

Question 12 : Compare the RTLs generated in Task 5. How they are different from each other

TASK 6

Objective

In this phase we will change allocation constraint on the same design and observe their effects on the synthesis process.

Set of Tasks

- **Set the cadency value as 50 ns and scheduling strategy as "no_pipeline". Also select "operator optimization". Select Allocation strategy as manual.**
- **After clicking the "control" button, set 4 as number of adders.**
- **View the synthesis report and also save the gantt chart in "adder_50ns_manual.html".**

Question 13 : What happened? How results differ from the previous step ?

____

____

____

____

- **Change register allocation as "none".**

Question 14 : Compare your gantt chart with the previous one and explain what happened by using this allocation strategy.

____

____

____

____

TASK 7

- **Complete the Table 2 based on task 3 to task 6. Use the information available in the gantt charts saved in these steps.**

Cadency Value	**Operator Optimization**	**Scheduling Strategy**	**Allocation Strategy**	**Resources**		**Latency**
				Registers	**Adders**	
110 ns	NO	Default	Automatic			
50 ns	NO	Default	Automatic			
50 ns	NO	No_pipeline	Automatic			
50 ns	YES	No_pipeline	Automatic			
50 ns	YES	No_pipeline	Manual		5	
50 ns	YES	No_pipeline	Manual		6	

TASK 8

Objective

We will apply different constraints on a FIR Filter which is a more complex design as compared to two input adders. We will apply different set of constraints and explore the design space in terms of serial/parallel tradeoffs.

Set of Tasks

- **From G.A.U.T. main window select "C/C++ Compiler".**
- **Click on the "Open" button and open "fir.c" by following the path :** $Home/ASIC/Lab1/FIR/fir.c".

Question 15 : What type of filter is this ?

____

____

- **Shows the impacts of latency on the area using the methods explained in previous tasks and complete the following table.**

Cadency	**Allocation Strategy**	**Area**	**Multipliers**	**Adders**	**Buses**	**Latency**
250	global_ub					
180	global_ub					
100	global_ub					
60	global_ub					

TASK 9

Objective

We will give a brief introduction on a commercial tool based on the high level synthesis concepts introduced in the previous tasks. The tool is called **Cadence C-to-Silicon**. Starting from a SystemC implementation of a FIR filter, we will show how SystemC code can be transformed to hardware, and which are the benefits.

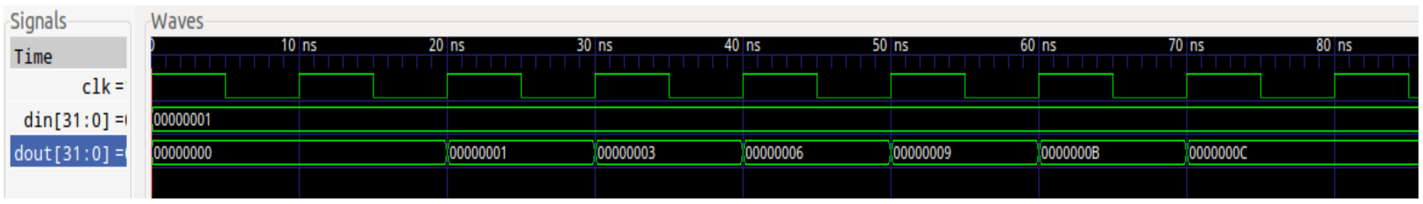
Introduction

….

Tasks

Download the fir_systemc.zip file from the course webpage. Unzip the file and look at the files contained. For the students who took the IL2452 System Design Language course, you should be familiar with this code.

If you want to check how to simulate such code, install the SystemC libraries on your PC. More instructions can be found at http://www.ict.kth.se/courses/IL2452/, under the Laboratory 1 instructions. Running such C code can generate the following output signals:



Question 16 : Which kind of filter implementation does the SystemC code model (pipelined, fully parallel,...)?
How could you modify it to get a different implementation?

____

____

Open a Linux shell and browse to the folder where you have your SystemC files and open the C-to-Silicon tool using the following commands (after being logged on the afs):

```
export ctos=/afs/ict.kth.se/pkg/cadence/ctos102/10.20.100/tools.lnx86/ctos/bin/64bit
export PATH=$PATH:$ctos
export LM_LICENSE_FILE=@lic1.ict.kth.se
export MGLS_LICENSE_FILE=@lic1.ict.kth.se
ctosgui
```

Select the New Design icon from the Tool Bar, or select

File -> New Design from the Menu Bar.

This will bring up the Create New Design Wizard to step you through setting up a new design.

For the first page, since this is a SystemC Design, just click Next.

For the next page, specify the SystemC source and the top module name in the SystemC Source area. In the Source Files field, type in main.c, or use the Add button to browse to this file. In the Top Module field, use the ... button to get a list of modules in the design, and select the only available. Click Next. For the next page, specify the name of clock is 'clk'. Click the Add Clock button, which adds an entry with a default Period of 20000, Rise of 0, and Fall of 10000. Change the name of the clock to clk by typing over clock_0, change Period to 10000 (and Fall should automatically change to 5000), and click Next.

For the next page, specify the Implementation-level details. In the Technology Library field, type tutorial.lbr tutorial.lib. Check Maximize Clock Gating and leave all the other defaults and click Next. For the next page, specify FIR as the Name and Save Directory.

Leave the other defaults and click Next. In the final page, leave Build Now selected, and click Finish.

Question 17 : What does the Build command do?

____

____

After the build has completed, the Input Source viewer, showing the SystemC source, is displayed, as is the Command Output window (which you can minimize by clicking the icon immediately to the left of the command prompt).In the Task Window (checkmark icon), Specify Micro-architecture pane, something is displayed in red: which indicates that something must be resolved before you can schedule.

What is happening? Look at the *for_ln40* and *for_ln35* loops in the Control Data Flow Graph by finding it in the Hierarchy Window, right-clicking, and selecting Show in CDFG, and try to give an explanation.

After that, double-click on Combinational Loops to display the Specify Micro-architecture dialog.

Set action for Click for_ln*35* by double-clicking in the Action column, and

select unroll.

Do the same for the loop for_ln40.

Question 18 : Which does the unroll command do?
How does the GDFG looks after unrolling the ONLY loop *for_ln35*?
How does the GDFG looks after unrolling the both loop *for_ln35* and *for_ln40*?

____

____

Now we can analyze and implement the design:

Examine the critical path by selecting Report -> Timing -> Cycle Analysis from the Menu Bar.

Examine the area by selecting Report -> Area from the Menu Bar.

Sometimes can happen that these reports are not generated. However, when this happens, it is possible to use manual commands in the terminal of the CtoS tool.

Select File -> Generate -> RTL from the Menu Bar. In the Generate RTL dialog, and click OK.

Check the generated *DUT_rtl.v* file in the *model* folder. Now we are ready to simulate the synthesized project using a new simulation tool: ncsim.

Include the provided testbench (testbench.v) and modify if not used the right name

.

Double check that the model folder contains at least testbench.v and DUT_rtl.v

.

Run the following command from a linux terminal:

```
nclaunch
```

If the tool does not start check that you have exported the path:

```
export PATH=$PATH:/afs/ict.kth.se/pkg/cadence/incisiv141/14.10.004/tools/bin
```

Select multi-step.

For each .v file from left column, select it (left click) then

Tools->verilog compiler->ok

check worklib folder on the right column is generated

.

Select DUT_RTL_tb from the right column in worklib folder. Then click on:

Tools->elaborator-> ok

Now we are ready to start the actual simulation:

Tools->simulator->snapshot select worklib.DUT_RTL_tb -> ok

Select DUT_RTL_tb on the left, you will see 4 signals on the right column

Select all signals on the right and right click and send to waveform window.

Press F2 to run the simulation

Are the results similar to the picture in page 26? Show them to the assistent.

If you reached this point the mandatory laboratory tasks are concluded! The next sessions are used to improve/modify the Labs for the future.

Hope you enjoyed it!

TASK 9 - OLD

Objective

We will show how high level synthesis can improve the speed of the design flow, and how it can be integrated in commercial tools targeting FPGAs. We will create an embedded system composed by a processor and some pheripherals, and we will show how C code can be transformed to hardware, and which are the benefits of it.

Introduction

In the previous tasks we have analyzed HLS techniques used in the GAUT tool. Similar techniques are used in some commercial tools to synthesize C functions to pure hardware. Altera commercialize a set of tools for system design and synthesis to FPGAs. In the following tasks we will:

- create an embedded system composed by a processor and some peripherals running on an FPGA;
- run the FIR filter function used in task 8 on the processor, implemented as pure software function;
- synthesize to hardware the FIR function
- run the FIR filter function synthesized as hardware accelerator
- compare the performances between the 2 design approaches

Note: depending on the Altera design suite version you are using, the steps to follow can be slightly different. Those Lab notes have been tested using the 10.0 suite.

Note2: we can provide a virtual machine where all the Altera tools are set up. You can install them on your computer if you want, but we cannot ensure fully support.
ftp://ftp.altera.com/outgoing/release/

Tasks

In this section, you are going to use the Nios II C2H Compiler to accelerate the software implementation of your FIR filter. Given a function name, the C2H compiler tries to generate an efficient, functionally-equivalent hardware in form
of an accelerator on the bus. Integrating the generated accelerator and invoking it in place of the original function is fully automated and does not require direct user interference.
Make sure you have studied the C2H compiler user guide [3], especially the first three chapters, and you have gone through the tutorial in chapter 2 of it.

Answer the following questions:

Question 16 : Describe what is the equivalent hardware generated by C2H for the following software constructs:
- function arguments and global variables,
- simple and nested loops,
- sub-function calls (functions called inside an accelerated function).

____

____

Step 1: create Quartus project, import files and create the hw for FPGA

Open the Quartus tool (on the provided VM, open the nios shell and write *quartus&*).
<u>If you want to use the files as provided, name the project "final_lab", otherwise change the final_lab expression in the final_lab.vhd file.</u>
Create a new project and select the correct family device: Cyclone IV, EP4CE115F29C7(N).
Leave the default values for all other fields and press Finish.

This process will create a folder containing a .qpf file (quartus project file).
In the same folder, copy the provided files: alt_pll.vhd, (final_lab).vhd, softcore.sopc.

Open the sopc builder tool from quartus (tools->sopc builder). This will open automatically the softcore.sopc file, containing a description of the system. Press generate and close sopc builder when finished.

Question 17 : What is the configuration of the system?

____

____

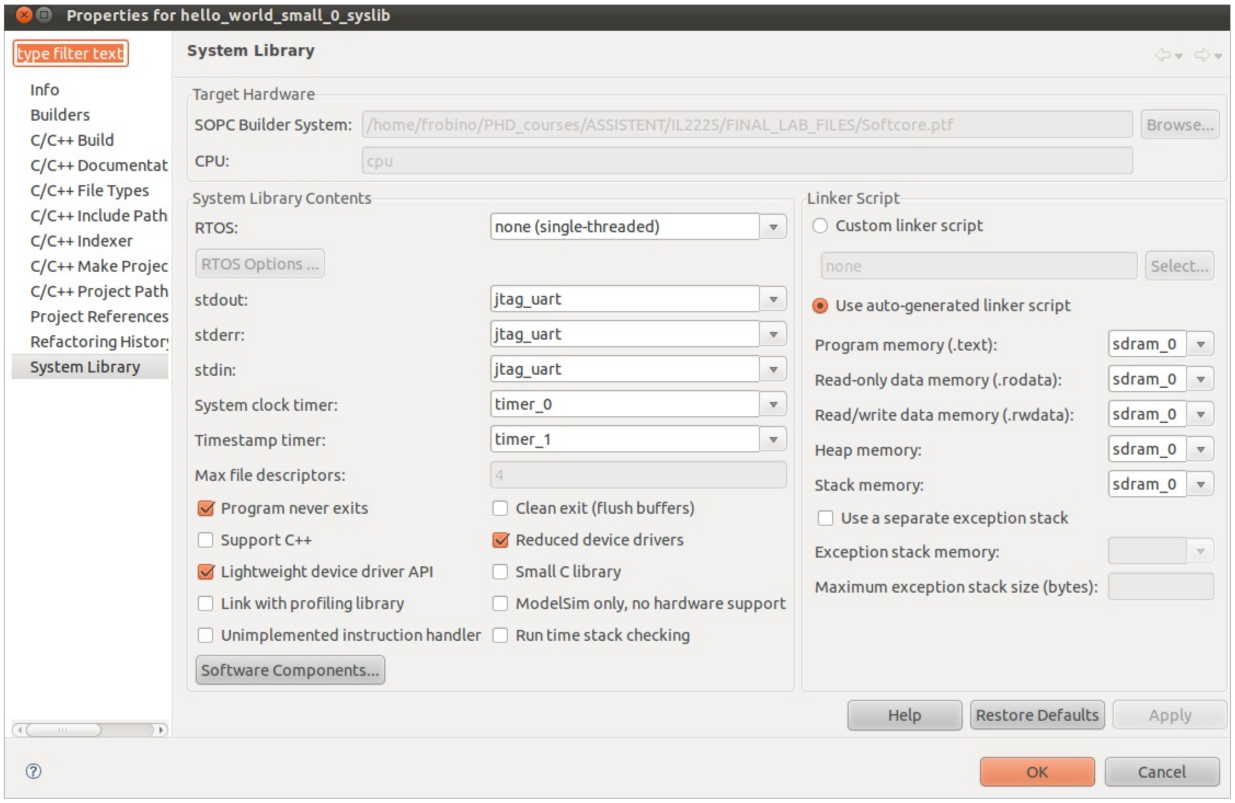
The described system has been generated. Now it must be synthesized to the target FPGA.
Copy the content of the pin_ass file inside the (final_lab).qsf file (open it with a text editor).
Run the synthesis (double click on the compile button on the left) and wait for a while.
When it's done connect the FPGA to your PC (USB cable in the BLASTER port of the board) and from the nios shell run the command *nios2-configure-sof <path_to_sof_file>*

Step 2: create and run a software implementation of the filter for the Nios core

Open the nios ide tool (from nios shell: *nios2-ide &*) and create a new project, using the

hello_world_small template.
Leave the defaults and click next until the project does appear in the left sided bar.

Right click on the project name on the left sided bar and select System Library Properties. Fix it so that if fits the following picture.


Properties for hello_world_small_0_syslib

type filter text

- Info
- Builders
- C/C++ Build
- C/C++ Documentat
- C/C++ File Types
- C/C++ Include Path
- C/C++ Indexer
- C/C++ Make Projec
- C/C++ Project Path
- Project References
- Refactoring Histor
- System Library

System Library

Target Hardware

SOPC Builder System: /home/frobino/PHD_courses/ASSISTENT/IL2225/FINAL_LAB_FILES/Softcore.ptf [Browse...]

CPU: cpu

System Library Contents

RTOS:	none (single-threaded)
stdout:	jtag_uart
stderr:	jtag_uart
stdin:	jtag_uart
System clock timer:	timer_0
Timestamp timer:	timer_1
Max file descriptors:	4

- [✓] Program never exits
- [ ] Clean exit (flush buffers)
- [ ] Support C++
- [✓] Reduced device drivers
- [✓] Lightweight device driver API
- [ ] Small C library
- [ ] Link with profiling library
- [ ] ModelSim only, no hardware support
- [ ] Unimplemented instruction handler
- [ ] Run time stack checking

Linker Script

○ Custom linker script: none
● Use auto-generated linker script

Program memory (.text):	sdram_0
Read-only data memory (.rodata):	sdram_0
Read/write data memory (.rwdata):	sdram_0
Heap memory:	sdram_0
Stack memory:	sdram_0

- [ ] Use a separate exception stack

Exception stack memory:

Maximum exception stack size (bytes):

Remove the hello_world_small.c file from the directory and paste in the provided .c file. The implemented functions use timers to find out the execution time of the FIR function.

The provided .c files implement the adder_1 studied in Tasks 1 and 2. The code has been tested and you can use it as a starting point to test your environment. When you have understood the provided code, you must modify it and adapt it to implement a FIR function.
Implement your own FIR function, using as a template the function provided in Task 8.
IMPORTANT: implement the function in a self standing function definition, such as

```
void aac(int *sum, const vect A)
```

Question 18 : How should the C code be changed to run on the Nios core, compared to the one processed by the GAUT HLS tool?

____

____

Build your software project in the nios ide so that it does not contain errors (right click on the project folder->build all).
If no errors, right click again and Run as Hardware (the FPGA must be connected and the configured with nios2-configure-sof).

Question 19 : What do you see in the terminal? Do you expect this result? How long does the function take if implemented as pure software?

Regarding Question 19, if you try to run the provided .c files implementing adder_1, the result should be BE (hex), and the time as pure SW: 1E ticks. In the answer you have to refer to your own FIR implementation.

Step 3: HLS using C2H and hardware implementation of the filter

This is the step where the High Level Synthesis tool is used. We will use the C2H tool, embedded in the Altera suite. We will indicate to the tool that we want to synthesize the C fir function directly to pure hardware.
To do so, in the Nios IDE, right click on the name of the function representing the fir and click *Accellerate with C2H*.

Re-build the project in the nios ide.
Re-compile the quartus project and run again the nios2-configure-sof command, to configure the FPGA with the new harware.
Re-run as hardware the nios ide project.

Question 20 : What do you see in the terminal? Do you expect this result? How long does the function take if implemented as pure hardware?

Question 21 : Open the SOPC builder tool in quartus. Is something changed from the Step 1? If yes, what?

Question 22 : Search for "product" in the RTL viewer of Quartus. How is the FIR synthesized? Is it optimized for speed, area or...?

____

____

NOTES:

If you find difficulties in using C2H, refer to the Nios II C2H compiler user guide, there you can find a very good step by step tutorial.

5. References

1. G.A.U.T. user Manual
2. How to write a C Code in G.A.U.T.
3. Nios II C2H compiler user guide