Imperial College
London

# EIE FIRST YEAR PROJECT

Imperial College London

## Table of Contents

## Requirements to Complete This Coursework

- Altera Quartus II (v.12);
- Microsoft Visual Studio 2008 (express);
- Calypto Catapult UV;
- Terasic DE0 board;
- Coursework source files and DE0 documentation.

## Typographic Conventions

ⓘ Important information.

🛠 Work you should deliver and include in your report.

## Suggested Timetable

### End of the Spring Term (27th Mar 2015)

By the end of the Spring Term, you are expected to hand-in an interim report. This should consist of a 3-page write-up of the exercises presented below, followed by 3-pages detailing your reading on Image Processing and your plan for the work next term, *including details of how you propose to split the work between the group members*. **This interim report is not assessed**. You may submit the report earlier than the given deadline if you have a clear idea about your project and you need feedback on your plan.

### End of this project

You will be expected to demonstrate your completed design on a DE0 board and hand-in a final report. The report, the demonstration, and the Catapult C code will form the formal deliverables for this project.

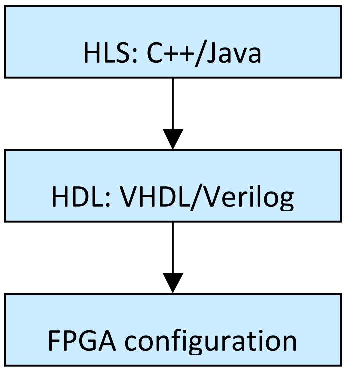
| Dates | Task |
|---|---|
| now – 27 March | Read through this sheet, attend talks and read about Image Processing – particularly convolution masks[1]. |
| 2 March – 27 March | Do the tasks outlined in this lab sheet |
| 2 March – 27 March | Write the interim report |
| 27 April – 18 May (approx) | Implement your plan |

To get you going you will be given one to two lectures by Dr. Bouganis (EEE), providing an outline of the project, a brief introduction to image processing, and an introduction to the programming language and concepts you will be encountering during your project.

[1] Several useful masks can be found at: http://www.sgi.com/software/opengl/advanced97/notes/node152.html

## Introduction

In this coursework you will acquire experience in the design of digital systems using a High-Level Synthesis (HLS) tool to describe your hardware system**.** Often HLS tools are used to create hardware accelerators for digital systems. They take a source, usually C/C++ or Java, and translate it into a Hardware Description Language, such as Verilog or VHDL, allowing for standard tool flow to be used.



In this coursework you are required to design a hardware system that performs real-time image processing to a video stream. The main description of your design will be performed using a HLS tool and the target board is the DE0 from Altera. The twist is that this platform does not contain any processor, but only a Field-Programmable Gate Array (FPGA).

### Image Processing

Image processing has been an active area in Information Systems for a long time[2]. An essential part of image processing is to process an image in order to be able to better extract some information of interest. For example, the detection of edges in an image can be used as a clue for object detection.

One of the major obstacles in the image processing field is the large number of computations that must be performed in order to process a whole image. This makes image processing slow unless performed on a very powerful computer.

### The Field-Programmable Gate Array

We tend to think about the processing power of our computers purely in terms of the clock rate. Such comparisons can only be made if we assume that we can do the same number of computations in each clock period.

[2] M. Sonka, V. Hlavac and R. Boyle, "Image Processing, Analysis and Machine Vision", ITP, 1993.

The Field-Programmable Gate Array (FPGA)[3] is a type of programmable logic device. We can design customised hardware in the FPGA to perform these computations in parallel and thus achieve a very high performance design, even with moderate clock frequencies.

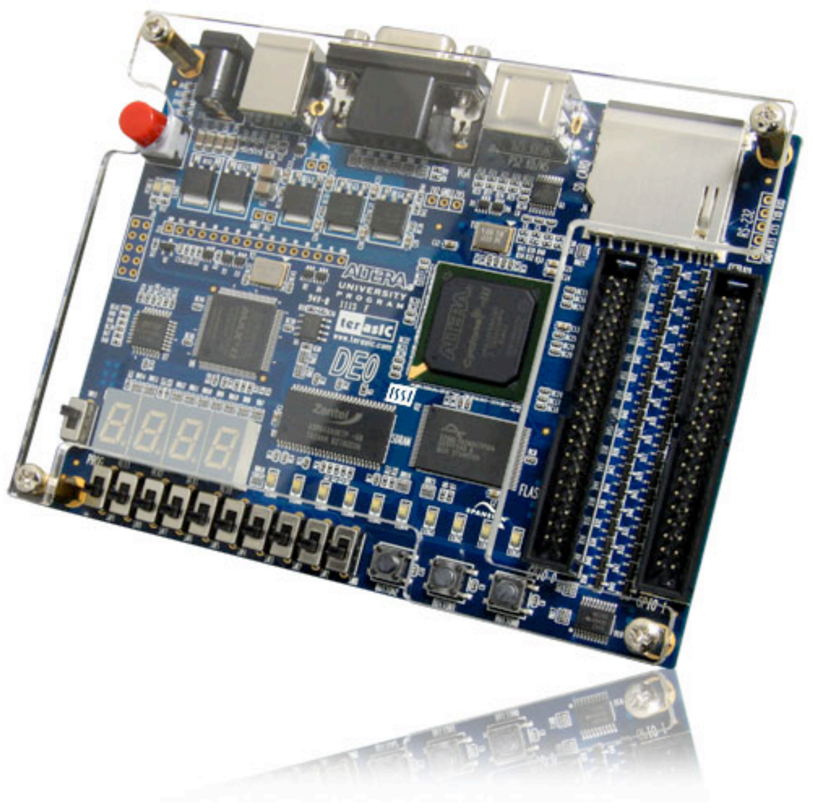
## The DE0 board

The board you will be using for this project is the DE0 board that has an Altera Cyclone III 3C16 FPGA, which contains 15K Logic elements (LEs), 56 M4K memory blocks, and 56 embedded multipliers.

A picture of the DE0 board is shown below. The board contains the FPGA, some Flash RAM, an SDRAM chip, a video digital-to-analogue converter (DAC), a PS/2 mouse/keyboard port and some other connections. Also included is a USB port to allow you to program the FPGA and have access to various components of the board from a PC.

Flash RAM is a type of RAM that does not lose its information when you turn the DE0 power off. We will use this to store any data we want to keep.

A full user-manual of the DE0 board can be downloaded from the project website or from Altera's website. This will be useful if you need to use some of the more advanced features of the board.



[3] http://en.wikipedia.org/wiki/FPGA

## Part I – Introduction to High-Level System Design Project with Catapult C

In the first part of this coursework you will be introduced to Catapult C and Quartus II, and then do a small project to understand their functionality. We will investigate the application of HLS for acceleration of computational intensive mathematical operations.

ⓘ You should make yourself familiar with the Catapult C development environment and high-level synthesis concepts by referring to the provided documentation: "Catapult C Synthesis User's and Reference Manual" – (catapult_useref_uv.pdf) and "High-Level Synthesis Blue Book" (hls_bluebook_uv.pdf) located in folder C:\Program_Files\Calypto Design_Systems\Catapult_Synthesis_2011a.126_Production_Release\Mgc_home\shared\pdfdocs\.

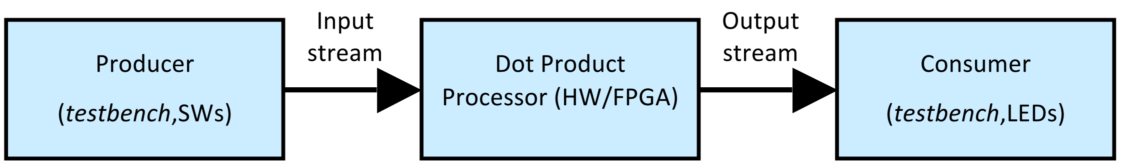
You can request help for the active window by clicking on the help button, on the toolbar.

### Dot product Accelerator

This project will demonstrate how to use Catapult C to design an accelerator to compute the dot product of two 1D vectors, which is defined as follows:

$$a' \cdot b = \sum_{i=0}^{N} a_i b_i$$

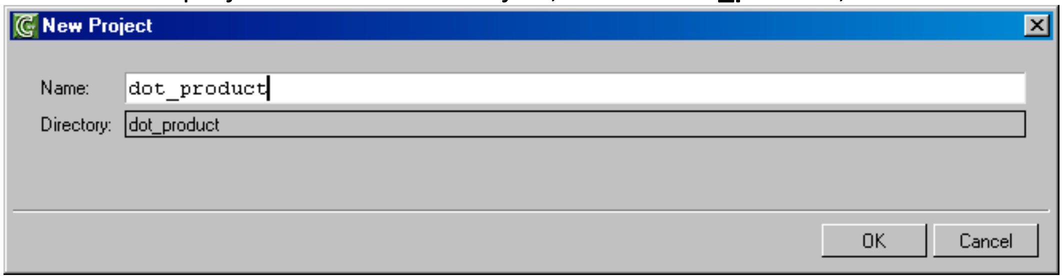
To verify the system's functionality we are going to connect the accelerator to a stimulus block (producer) and read its output (consumer). In the simulation you are going to use a *testbench* and when running on the FPGA board you are going to use peripherals (SWs and LEDs) in order to test its functionality.



🛠 You are now going to create a Catapult C project to compute the dot product. This will help you to better understand the work-flow from HLS to a system running on a DE0 board.

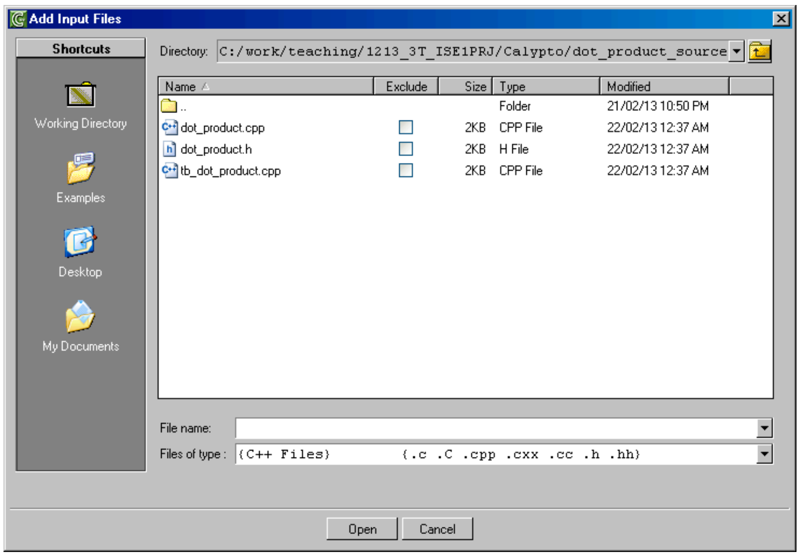
1. Create a folder for your work and download the provided files (i.e. prj1) there (e.g.: H:\lab);
2. Start Catapult using the provided **launch_catapult.bat** file.

3. Create a new project: File→New→Project; name it **dot_product**;



Name: dot_product
Directory: dot_product

4. Set your working directory, by clicking on **set working directory** on the task bar. Specify the path of the folder you created;
5. Click on **Add Input files** and then select the files given inside folder **dot_product**.

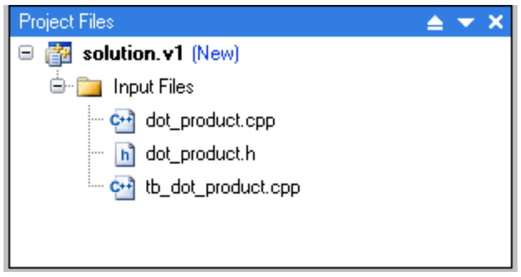


Directory: C:/work/teaching/1213_3T_ISE1PRJ/Calypto/dot_product_source

| Name | Exclude | Size | Type | Modified |
|---|---|---|---|---|
| .. | | | Folder | 21/02/13 10:50 PM |
| dot_product.cpp | ☐ | 2KB | CPP File | 22/02/13 12:37 AM |
| dot_product.h | ☐ | 2KB | H File | 22/02/13 12:37 AM |
| tb_dot_product.cpp | ☐ | 2KB | CPP File | 22/02/13 12:37 AM |

File name:
Files of type : (C++ Files) (.c .C .cpp .cxx .cc .h .hh)

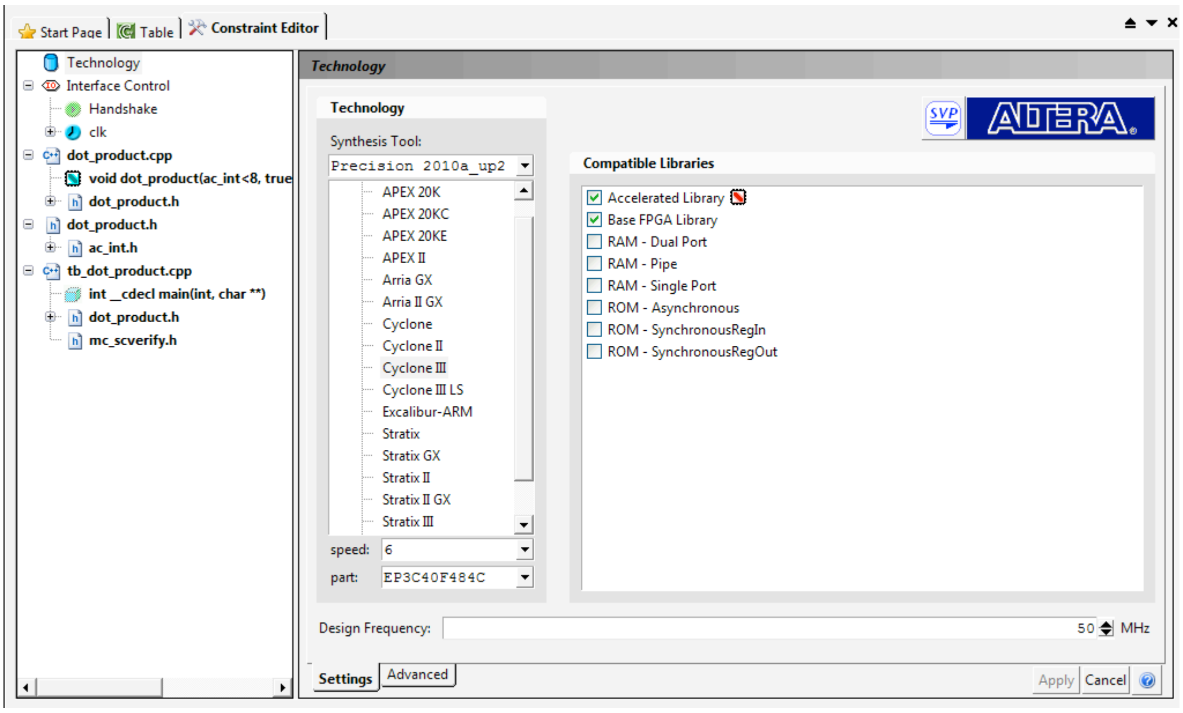
The 3 files should be now included in your project:



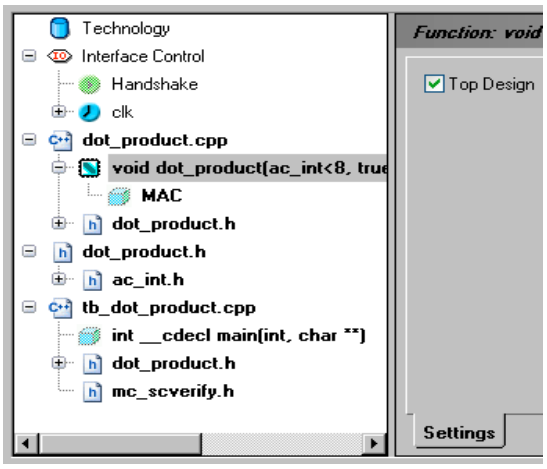
solution.v1 (New)
Input Files
dot_product.cpp
dot_product.h
tb_dot_product.cpp

6. Inspect the added files. Open the dot_product.cpp file. The function calculates the dot product of two vectors of length VECTOR_LEN. The function takes as input two arrays of integers are returns a value (i.e. output). The main body of the function has a for loop that iterates through each element of the two arrays and calculates their dot product.
7. Open the tb_dot_product.cpp file. This is the testbench that we will use to verify the functionality of our system. The file has two parts. The first part (//expected result) the expected result of the provided input is calculated, where the second part (//test design) makes a call to the functions that describes our HW and calculates the result that the HW will return (we are still in simulation).

8. On the Task Bar, under the Synthesis Tasks click on Setup Design. This will open the Constraint Editor on the main window. Click on **Technology** and select Precision 2010a_up2, then click on Altera to expand and select Cyclone III. Two options will show under. Select **speed** 6 and **part** EP3C16F484C. On **Design Frequency** select 50.0 MHz. Finally click on **Apply**. By doing the above steps, you have told to the tools the device that it should target.



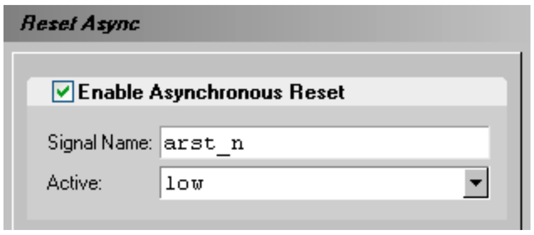
9. On the left side of the **Constraint Editor** window, you now have a list with the all source files you have included with their declarations, functions and its includes. Select your top level function and then tick **Top Design** and click on **Apply**.



ⓘ Catapult has a built-in version control mechanism. Every time you change the source files, you will be asked if you want to create a new branch. You have to create a new branch each time, otherwise the environment won't update your work. You can control which version you are using in the solution toolbar. When a new solution is automatically created, you can still change its name for something more meaningful to you, and add a description to it.

## ACCELERATOR INTERFACES

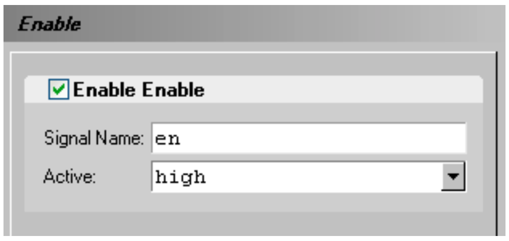
In the **Constraint Editor** dialog you find definitions for the **Interface Control**. For this first example you don't need to specify handshake signals, but you need to disable **synchronous reset** and enable **asynchronous reset** and **enable**. Expand **clk** to see these options. On the correspondent panels, you'll find the name of the signals Catapult will create in your hardware block. Finally click on **Apply**.


*Reset Async*

☑ Enable Asynchronous Reset

Signal Name: arst_n

Active: low


*Enable*

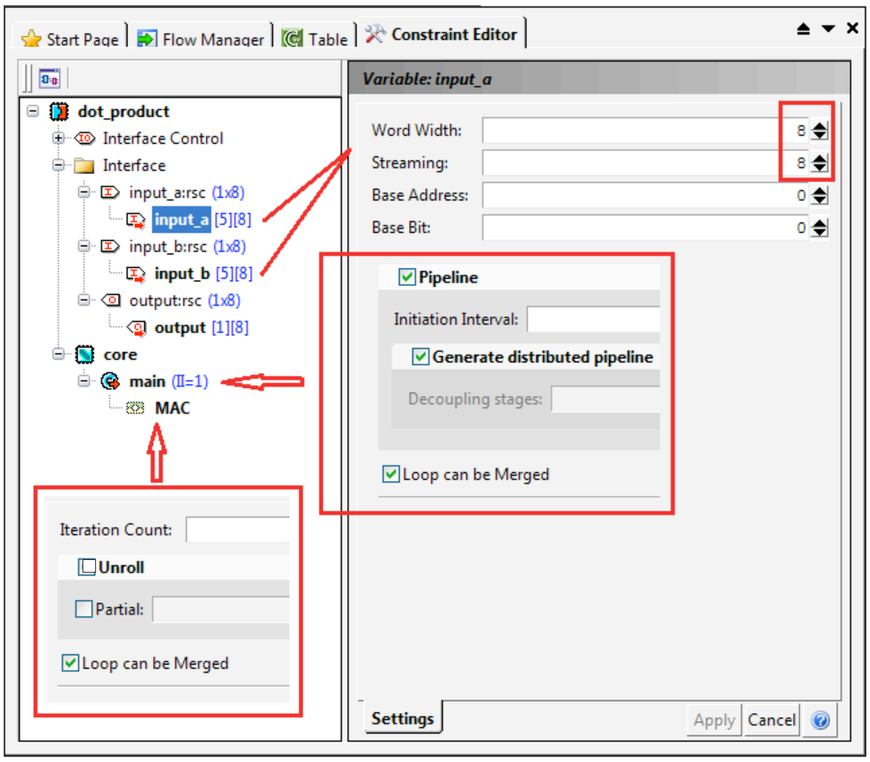
☑ Enable Enable

Signal Name: en

Active: high

Chapter 6 of the user manual provides more details about the interface signals, including handshake signals and their name convention.

Finally, you need to indicate to the tool that the input data will be streamed into your block, rather will be available all at once. Click on the Architecture Constraints (in the Task Bar window) and specify the following:



Variable: input_a

| | |
|---|---|
| Word Width: | 8 |
| Streaming: | 8 |
| Base Address: | 0 |
| Base Bit: | 0 |

☑ Pipeline
Initiation Interval:
☑ Generate distributed pipeline
Decoupling stages:
☑ Loop can be Merged

Iteration Count:
☐ Unroll
☐ Partial:
☑ Loop can be Merged

The above settings, specify that your inputs (a and b) have a streaming interface. This means that the inputs are coming to the HW block one by one. Also, you specified that the main needs to be pipelines with II=1, that is it can process one new input every clock cycle.

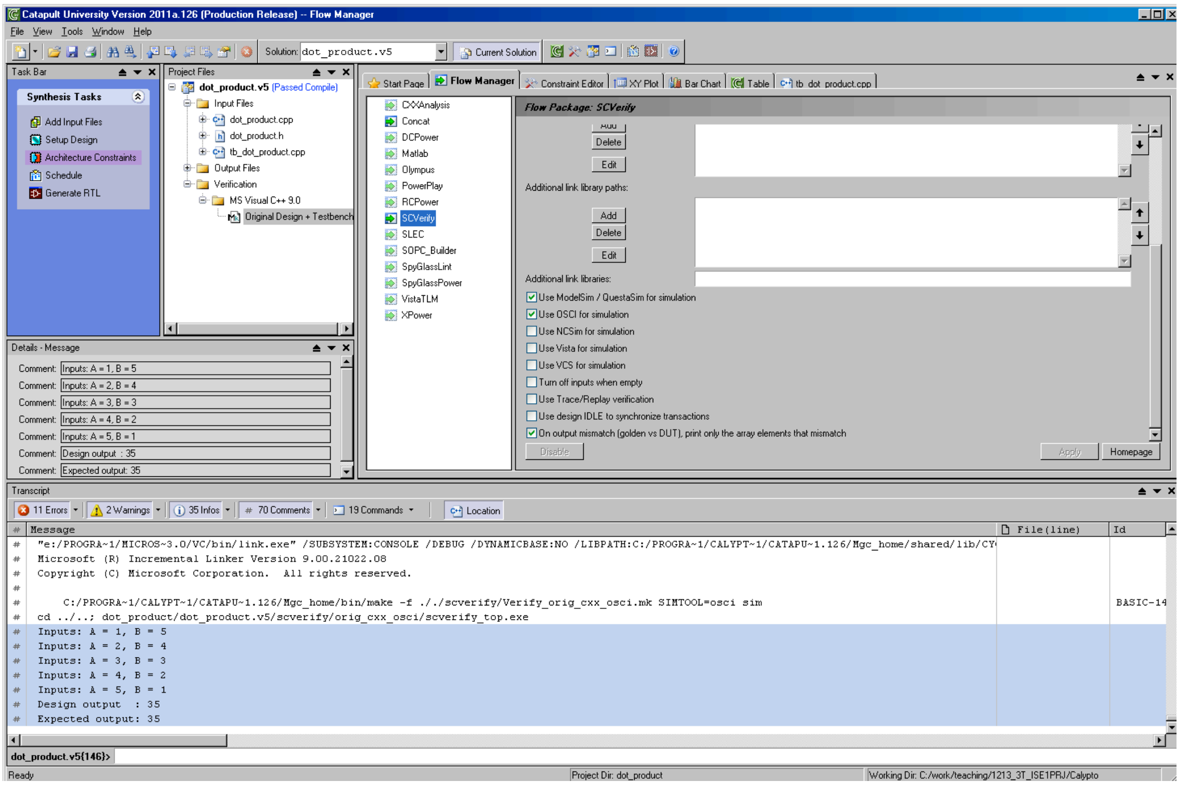
## VERIFY YOUR APPLICATION

*Testbench* is the mechanism available to test your design during development instead of running it in hardware. It consists of a unit that wraps your hardware processor, stimulates its inputs and compares its outputs to a predefined expected result.

Before you'll be able to run a *testbench*, you need to set it up. Click on **Flow Manager** on the main window and select SCVerify. (If you cannot find it, click on View->Other Windows->Flow Manager)

Enable SCVerify with the same options as shown in the figure below. Select the **Architecture Constraints** in the **Task Bar** window.

After this step, you are in position to run the simulation of your design by double-clicking on **Verification→MS Visual C++ 9.0→ Original Design + Testbench**. The output will be displayed on the message window. You should be able to verify that the Design output is the same as the Expected output. Verify that this is the case.



Catapult University Version 2011a.126 (Production Release) -- Flow Manager

Flow Package: SCVerify

Additional link library paths:

Additional link libraries:

- [✓] Use ModelSim / QuestaSim for simulation
- [✓] Use OSCI for simulation
- [ ] Use NCSim for simulation
- [ ] Use Vista for simulation
- [ ] Use VCS for simulation
- [ ] Turn off inputs when empty
- [ ] Use Trace/Replay verification
- [ ] Use design IDLE to synchronize transactions
- [✓] On output mismatch (golden vs DUT), print only the array elements that mismatch

| | |
|---|---|
| Comment: | Inputs: A = 1, B = 5 |
| Comment: | Inputs: A = 2, B = 4 |
| Comment: | Inputs: A = 3, B = 3 |
| Comment: | Inputs: A = 4, B = 2 |
| Comment: | Inputs: A = 5, B = 1 |
| Comment: | Design output : 35 |
| Comment: | Expected output: 35 |

Transcript: 11 Errors, 2 Warnings, 35 Infos, 70 Comments, 19 Commands

```
# Message                                                                                                          File(line)  Id
#  "e:/PROGRA~1/MICROS~3.0/VC/bin/link.exe" /SUBSYSTEM:CONSOLE /DEBUG /DYNAMICBASE:NO /LIBPATH:C:/PROGRA~1/CALYPT~1/CATAPU~1.126/Mgc_home/shared/lib/CY
#  Microsoft (R) Incremental Linker Version 9.00.21022.08
#  Copyright (C) Microsoft Corporation.  All rights reserved.
#
#      C:/PROGRA~1/CALYPT~1/CATAPU~1.126/Mgc_home/bin/make -f ././scverify/Verify_orig_cxx_osci.mk SIMTOOL=osci sim           BASIC-14
#  cd ../..; dot_product/dot_product.v5/scverify/orig_cxx_osci/scverify_top.exe
#  Inputs: A = 1, B = 5
#  Inputs: A = 2, B = 4
#  Inputs: A = 3, B = 3
#  Inputs: A = 4, B = 2
#  Inputs: A = 5, B = 1
#  Design output  : 35
#  Expected output: 35
```

dot_product.v5{146}>

Ready | Project Dir: dot_product | Working Dir: C:/work/teaching/1213_3T_ISE1PRJ/Calypto

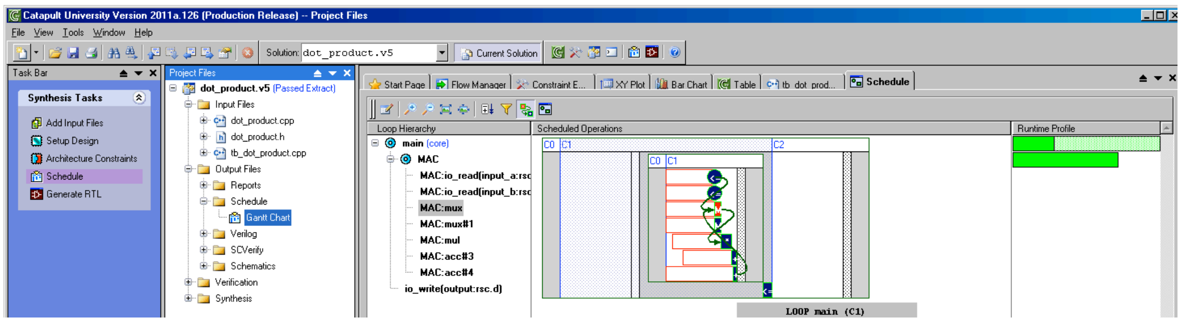
## SCHEDULING

The synthesis tool recognizes patterns in the C++ software and tries to optimize the hardware block according to its functionality.

Now that we verified that the functionality of our block is correct, it is important to understand the relation between the algorithm’s functionality and its implementation constraints (execution time and circuit area).

Click on **Schedule**, on the **Task Bar**. Catapult C will start the scheduling analysis. If the Gantt chart doesn’t show up automatically, double-click on **Gantt Chart** in **Project Files→Output Files→Schedule**.

The Gantt chart shows the execution of your algorithm and its corresponding architectural elements.



Relate the execution steps to your algorithm and verify how many clock cycles it takes to complete each step and the complete execution of your algorithm.

On the **Synthesis Tasks** (or **Task Bar**) panel, click on **Architecture Constraints**. On the **Constraint Editor,** click on **main** and untick Pipeline, then locate the **MAC** unit (multiply-accumulate) and click on it. Tick the **Unroll** box and then click on **Apply**. Check the Project Files window. You will see that different solutions are created automatically by the system.

Now, repeat the scheduling analysis and produce a new Gantt chart. What are the differences between both charts in terms of execution steps, data dependencies and circuit resources? Discuss possible alternative implementations of your algorithm consuming less cycles or resources than the ones generated by Catapult C. What are the trade-offs? Verify that the output is correct.

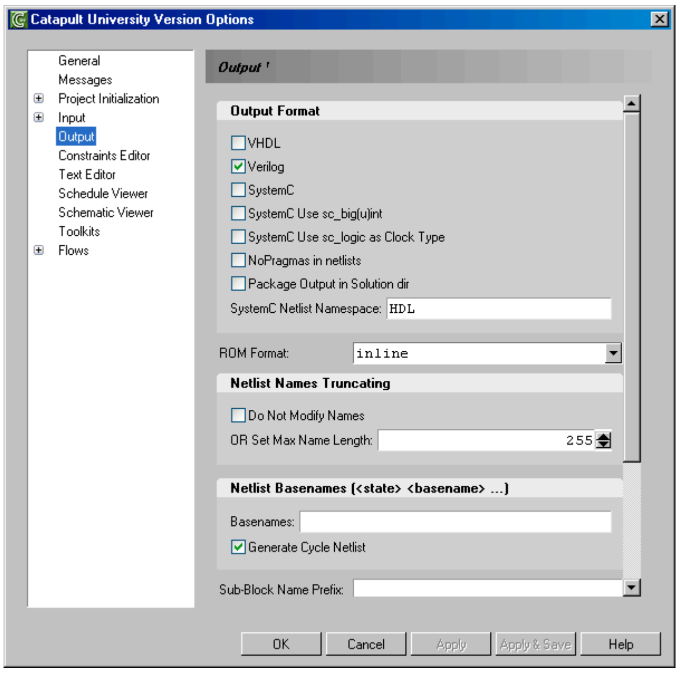
Note: By clicking on the Table tab (Window->Table) you can see a comparison across your explored solutions. Spend some time to understand this.

Continue this handout with the main function being Pipelines and for the MAC unit the **Unroll** option not ticked (first configuration).

## GENERATE HDL FILES

Having verified that your design functions properly and you have found a solution that meets your specifications, you are ready to generate the HDL files to include in your synthesis project in Quartus II.

First you need to instruct Catapult that you want your HDL files in Verilog. To do that click on **Tools→Set Options**, select **Output**, tick **Verilog**, (un-tick VHDL if it is ticked) and then click on OK.



Output Format

- [ ] VHDL
- [✓] Verilog
- [ ] SystemC
- [ ] SystemC Use sc_big(u)int
- [ ] SystemC Use sc_logic as Clock Type
- [ ] NoPragmas in netlists
- [ ] Package Output in Solution dir

SystemC Netlist Namespace: HDL

ROM Format: inline

Netlist Names Truncating

- [ ] Do Not Modify Names

OR Set Max Name Length: 255

Netlist Basenames (<state> <basename> ...)

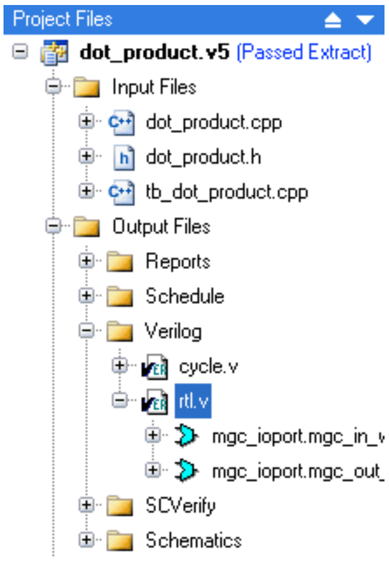
Basenames:

- [✓] Generate Cycle Netlist

Sub-Block Name Prefix:

You can now generate Verilog HDL files to be passed to your project in Quartus II. On the **Task Bar** click on **Generate RTL**.

After that you will see more items on your project tree, similar to the figure below. We are interested in the rtl.v file, and its dependencies (rtl*.v) as you will have to add them to your Quartus II project.



Besides HDL files, Catapult also generates schematic for your hardware block. Expand **Schematics** on your project tree and select RTL. You can double-click on the blocks to open them and examine their details.

If you select Data-path you can see the paths carrying data being processed, which are the data blocks and the control blocks.

When you reach one basic element in the schematic, Catapult will point in the source code where that element is.

🛠 Using the Data-Path diagram, identify the operator(s) and operand(s) of the algorithm in the diagram.

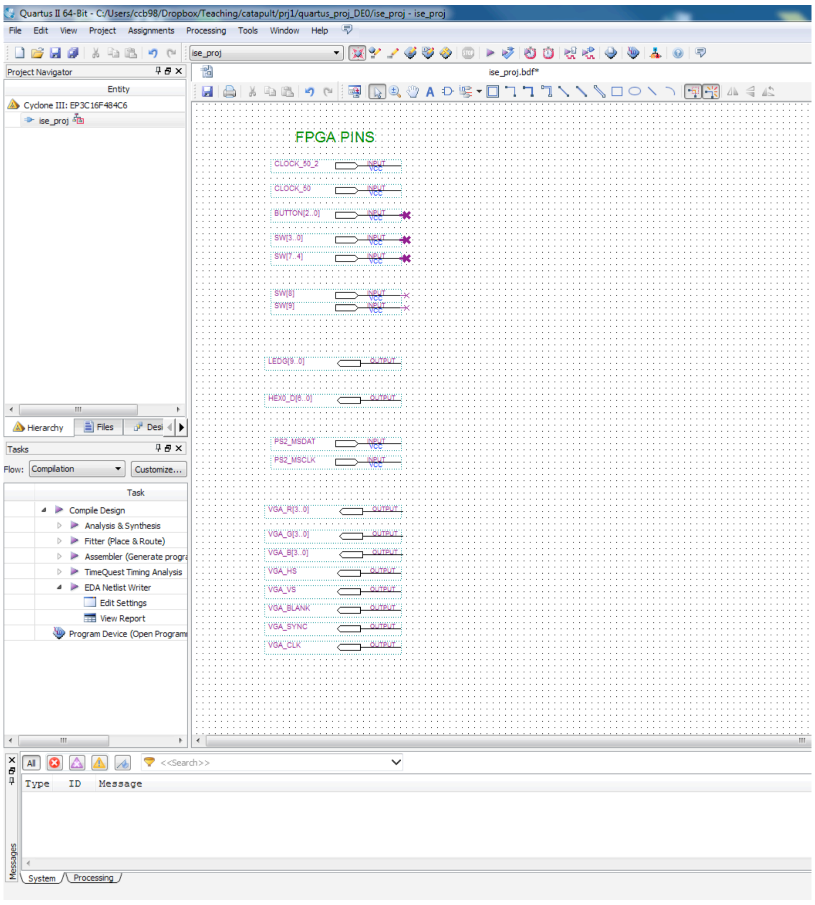
## Hardware Synthesis with Quartus II

After you have successfully completed the generation of the HDL files for your hardware block in Catapult C, you need to attach it to your project in the synthesis tool, Quartus II.

In the current example, the control and data signals will be connected to the board. The user will provide the input data, where the output will be presented on the LEDs of the board.

Open the Quartus II project given to you. It has the top entity (**ise_proj**) and the interface pins for switches, LEDs, 7-segment displays, LCD, PS2, VGA and video input on the DE0 board. It also has the correct FPGA selected.

Double-click on the top entity name (**ise_proj**) on the panel on the left to open it. It should look like this:

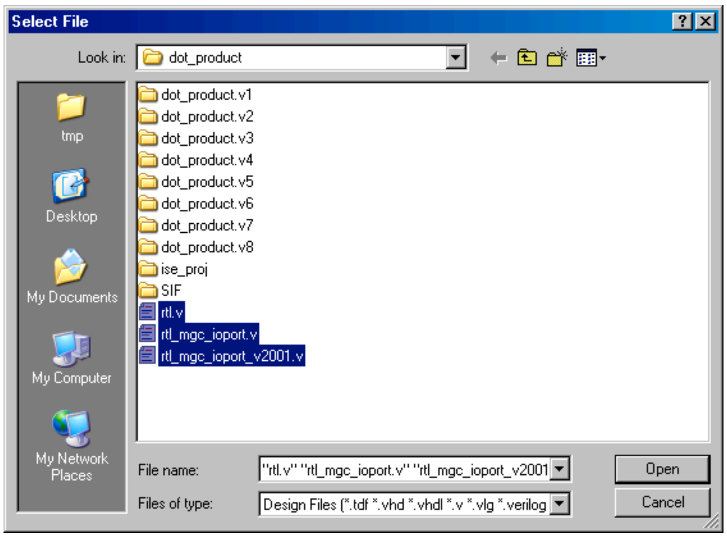


Since the processing element has just been generated by Catapult, you need to add the source files to the existing project (**Project→Add/Remove Files in Project**). Later you will connect the new units to the pins on the FPGA.

ⓘ Before you add the generated files to your Quartus II project, make a copy of them into the project's main folder, and use these files for inclusion in your Quartus design. The reason for doing this is to be able to update your design with new versions of your accelerator design without having to remove and add files in your Quartus II project.

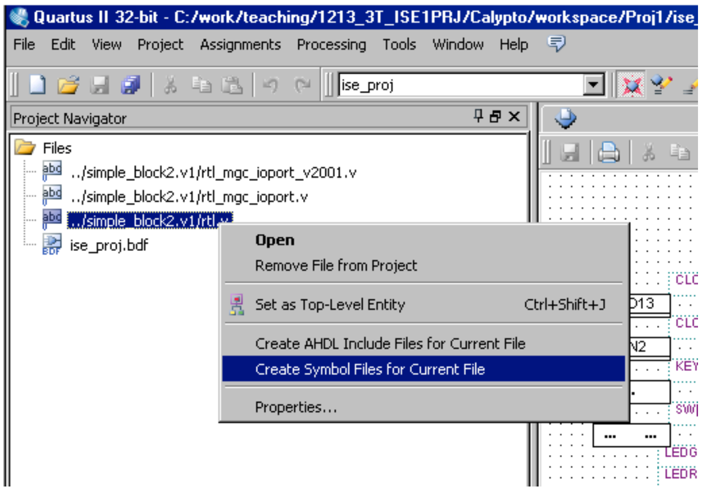
More information about the Quartus II development environment can be found here:

ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Tutorials/Schematic/Quartus_II_Introduction.pdf



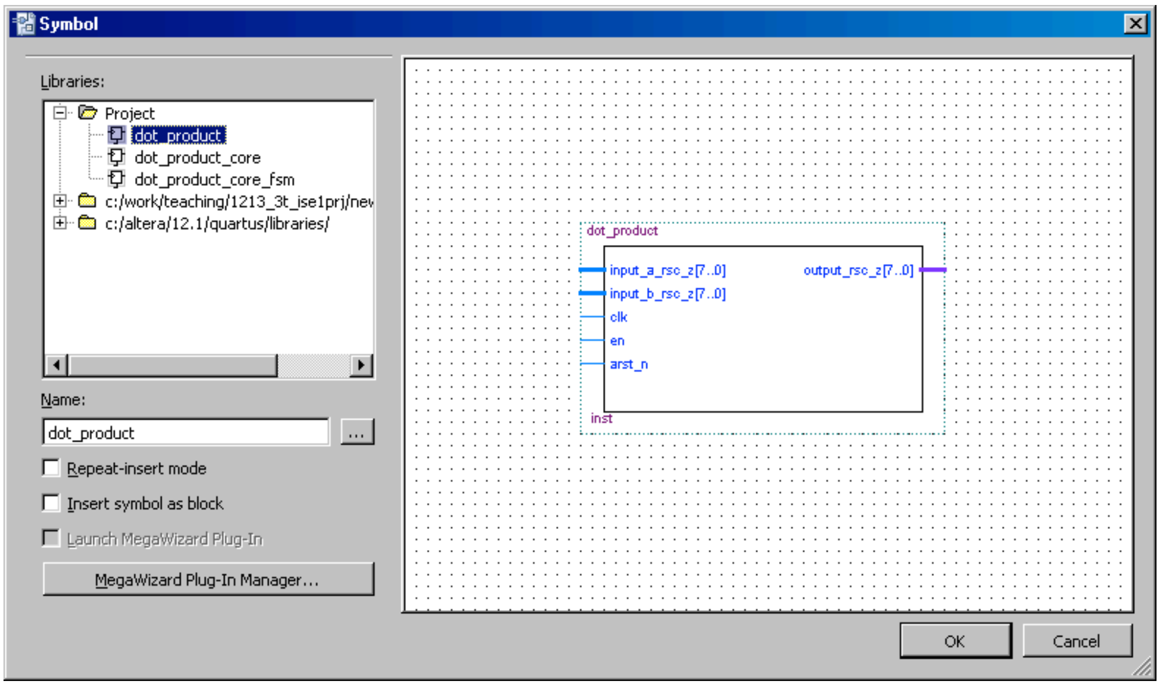
Locate the generated files in your dot_product project folder (**rtl.v** and ***mgc_*.v**) and add the files to your project (these should be under your latest solution).

Now that the generated HDL files are included in the project, before you include them in the diagram, you need to create the corresponding symbol, so that the tool knows how to represent it in the diagram. Select **Files** panel on the **Project Navigator**. Right-click on **rtl.v** and select **Create Symbol Files for Current File**.


Quartus II 32-bit - C:/work/teaching/1213_3T_ISE1PRJ/Calypto/workspace/Proj1/ise_
File Edit View Project Assignments Processing Tools Window Help
ise_proj
Project Navigator
Files
../simple_block2.v1/rtl_mgc_ioport_v2001.v
../simple_block2.v1/rtl_mgc_ioport.v
../simple_block2.v1/rtl.v
ise_proj.bdf
Open
Remove File from Project
Set as Top-Level Entity Ctrl+Shift+J
Create AHDL Include Files for Current File
Create Symbol Files for Current File
Properties...

You have to repeat this step whenever you change your accelerator (Catapult C design), otherwise its changes won't be seen in the diagram. If the unit is already in your block diagram, after generating the symbol, right-click on it and then select update symbol.

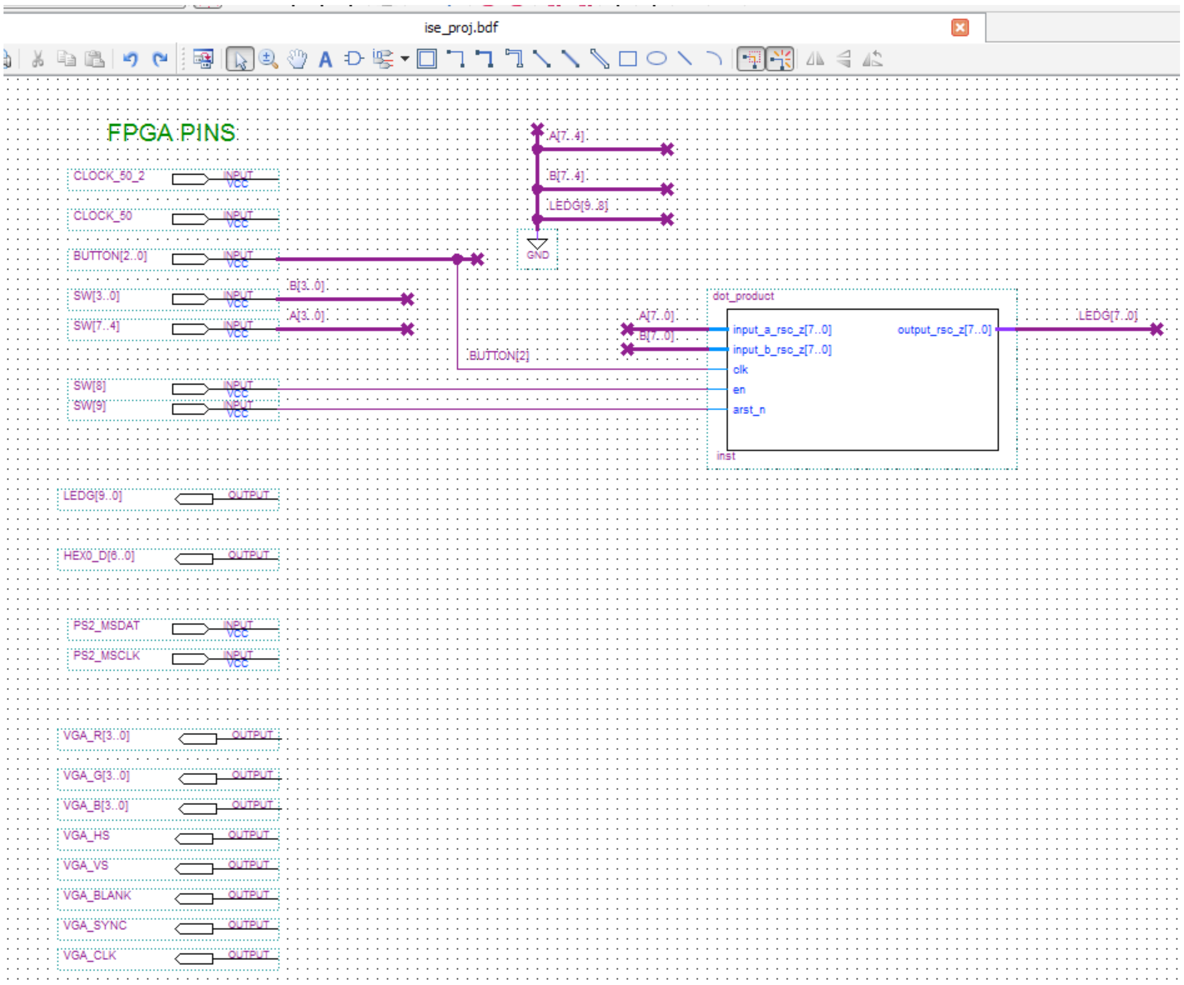
On the block diagram, double-click on an empty area and the dialog window to insert symbols will show up:



Select **dot_product** and click on OK to place in the schematic. Now you will create the connections between the processing unit and the FPGA. The connections will define how the user will operate with the unit. Connect all the inputs to switches (SW) and push-buttons (KEY), and the outputs to LEDs. Use to table below to guide you through the process.

| Hardware Block | FPGA pin | Description |
|---|---|---|
| clk | BUTTON[2] | Clock signal |
| en | SW[8] | Enable unit |
| arst_n | SW[9] | Asynchronous reset (active low) |
| input_a_...[7..0] | SW[3..0] | Input vector A (Check diagram) |
| input_b_...[7..0] | SW[7..4] | Input vector B (Check diagram) |
| output_...[7..0] | LEDG[7..0] | Dot product result |

You should have a schematic similar to this:



With the hardware block connected to the peripherals on the board, you can compile the design **Processing → Start Compilation (CTRL+L)** and verify that the synthesis process completes without error. The output of this compilation is a circuit than can be downloaded to the FPGA on the DE0 board (Ignore any timing violations, as the clock is simulated by the user).

Start the programmer to run your circuit on the DE0 board **Tools→Programmer**.

To test your design on the board, use switches to encode your input stream, and the LEDs to verify that the output produced by the processor generates the correct output.

🛠 Previously in this handout you saw that you can implement different scheduling for the operations in your algorithm (rolled *vs* unrolled). Repeat the compilation of the design, now using the other scheduling. What is the impact, in Quartus II, of unrolling your design? Compare the results in terms of area reported by Quartus II against the area reported by Catapult-C. Discuss results. You can read the results in Quartus II in the **Compilation Report** panel→**Analysis & Synthesis**→**Resource Utilization by Entity**

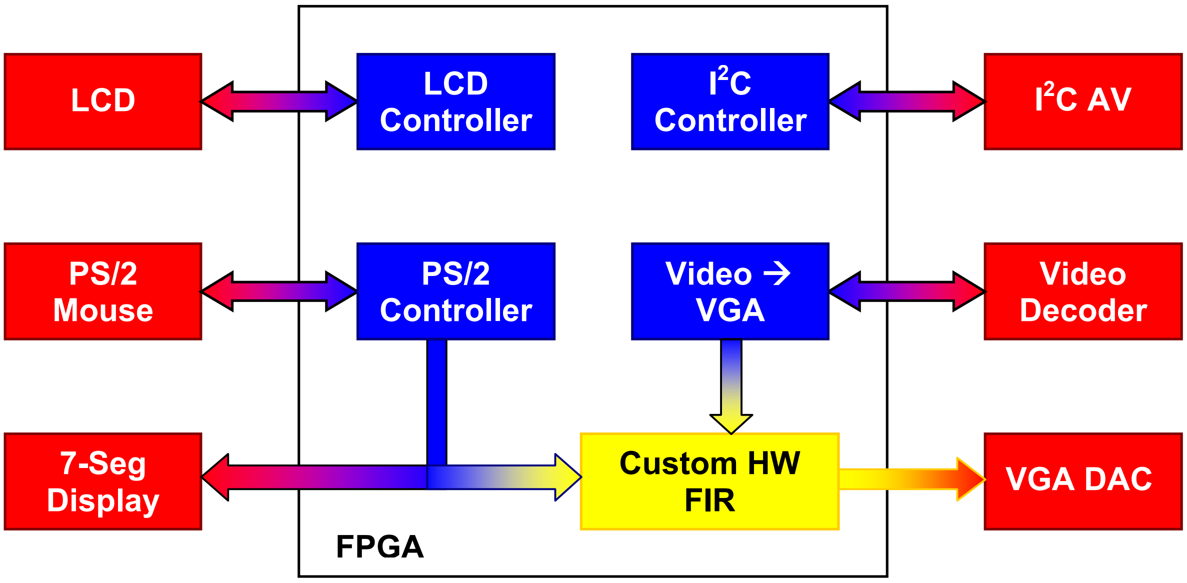
🛠Try to implement other functionalities and test your designs. (i.e. Addition, multiplication in a non streaming way), in order to familiarise yourselves with the tool.

## Part II – Real-Time Image Processing Accelerator

Now that you are familiar with the flow of the system design tools, you are going to use them to generate an accelerator for an image processing application.

Besides computationally intensive applications, digital systems are often used in applications that require real-time processing. Such applications demand a high throughput that general purpose CPUs are unable to meet.

In order to help you to develop your system, a design template is given to you. The template provides support for interfacing with the various components of the board (i.e. mouse, VGA output, etc), so you can focus on the main block of your system that performs the intended operations. As an example, a system provided that blurs the input stream in real-time. The main processing is performed in the “Custom HW FIR” block that has been designed through Catapult C.

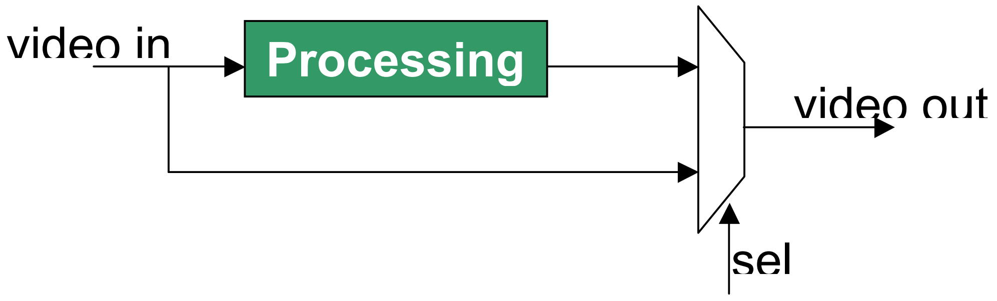


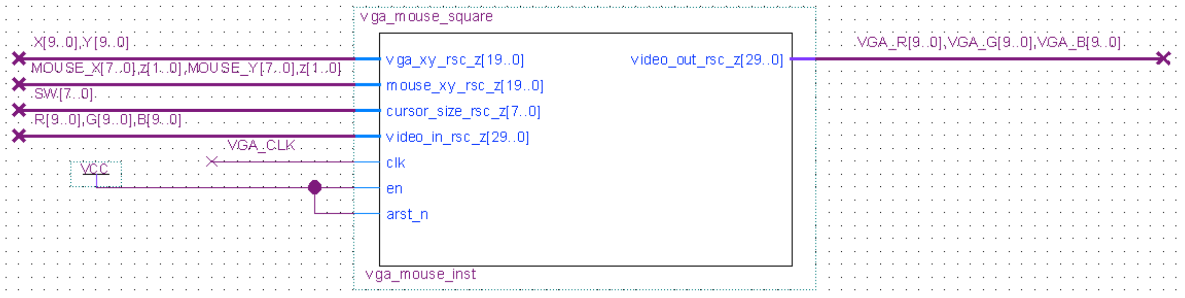
In the files made available to you (**prj2.zip – DE0_CAMERA_MOUSE**), you will find Catapult C source files for the example for a blur filter (vga_blur), and a Quartus II project with the controllers for all the peripherals above. The demo presents two design examples (i.e. VGA Mouse and VGA Blur) that have been designed through Catapult-C and put together in common design for demonstration reasons. The corresponding Catapult-C files can be found in the prj2->catapult_proj directory. The provided demo utilises the files generated by the Catapult-C tool flow through the catapult_ip directory. You are advised to copy any new designs (*.v files) generated by Catapult-C in that folder in order to incorporate them in the existing demo.

## VGA Mouse

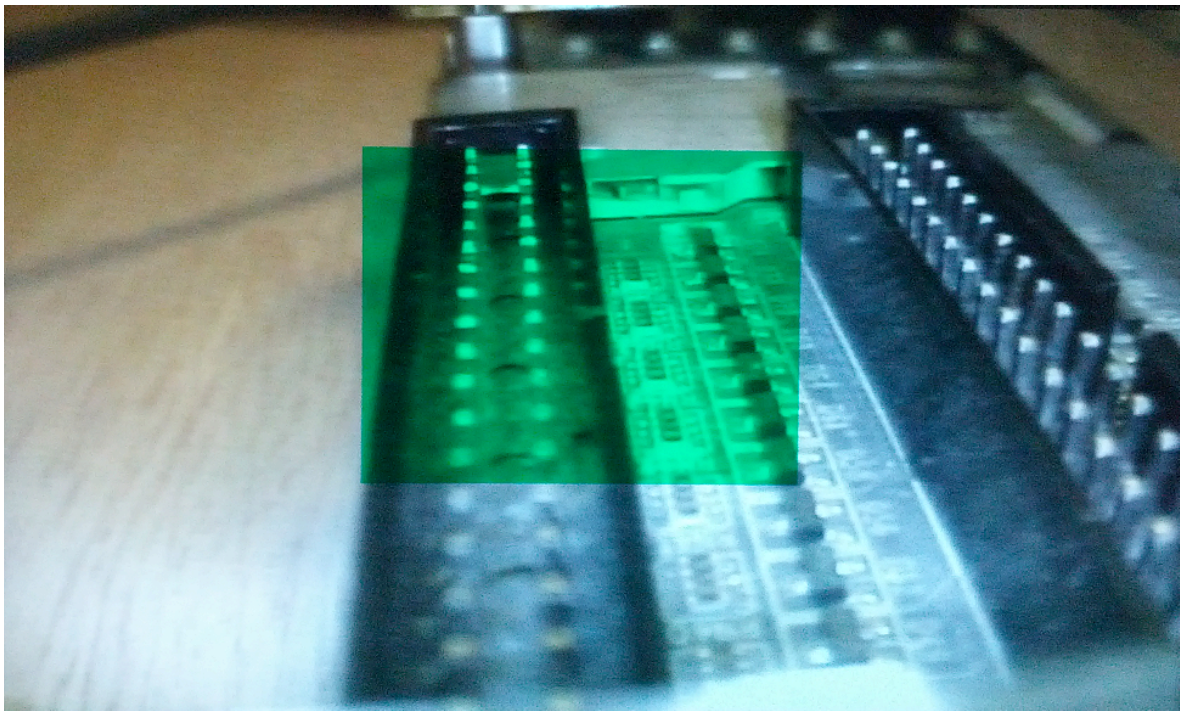
In this example, the mouse controls the position of a window on the input frame and some basic processing is performed within this window. In this example, the purpose of the system is to change the colour of the pixels depending on whether it is inside the mouse cursor area or not. The figures below illustrate the design specified in Catapult C and the block diagram for the instantiation of the generated block. Please read the "instructions .doc" file for more details on the control of the demo.

The system takes as inputs the video stream, the SWs, the mouse coordinates and the coordinates of the pixels that are rendered by the VGA output. The output of the system is the pixels that correspond to coordinates of the VGA output. The design creates two streams internally. The first stream is the one that performs the colour conversion, where the second one just passes the input stream unaltered. A multiplexer selects which one of the two streams should be passed to the output of the system. The multiplexer is controlled by a block that takes as input the pixel coordinates and the mouse position and checks whether the output pixel is within the mouse region.





Demonstration of the design running.



## VGA Blur

In this example, the aim is to apply a 5x5 blurring filter to the input video stream. The custom hardware block receives the lines buffered and produces the average of all pixels.

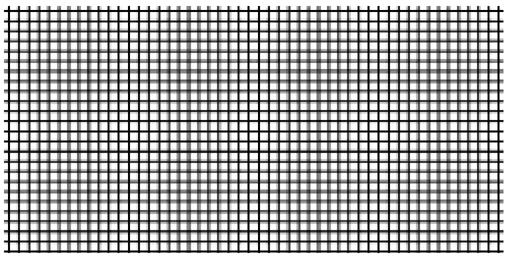
### MATLAB IMPLEMENTATION

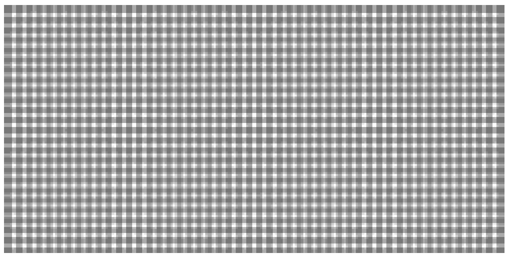
Before focusing on the hardware implementation you should verify how a filter works using a faster implementation and visualization method. In this case you are going to use Matlab.

Example of Matlab source code for a blur filter:

```
k = [1 1 0 1 1; 1 1 0 1 1; 0 0 0 0 0; 1 1 0 1 1; 1 1 0 1 1 ];
l = repmat(k, 50,50);
fm = ones(3)/9; Y = filter2(fm, l);
imshow(l), figure, imshow(Y)
```

The code above creates a matrix with black (0) and white (1) mesh and then applies the filter to that matrix  (image). The filter kernel is a 3x3 matrix with constant values. Below there are the original (left) and result (right) pictures.



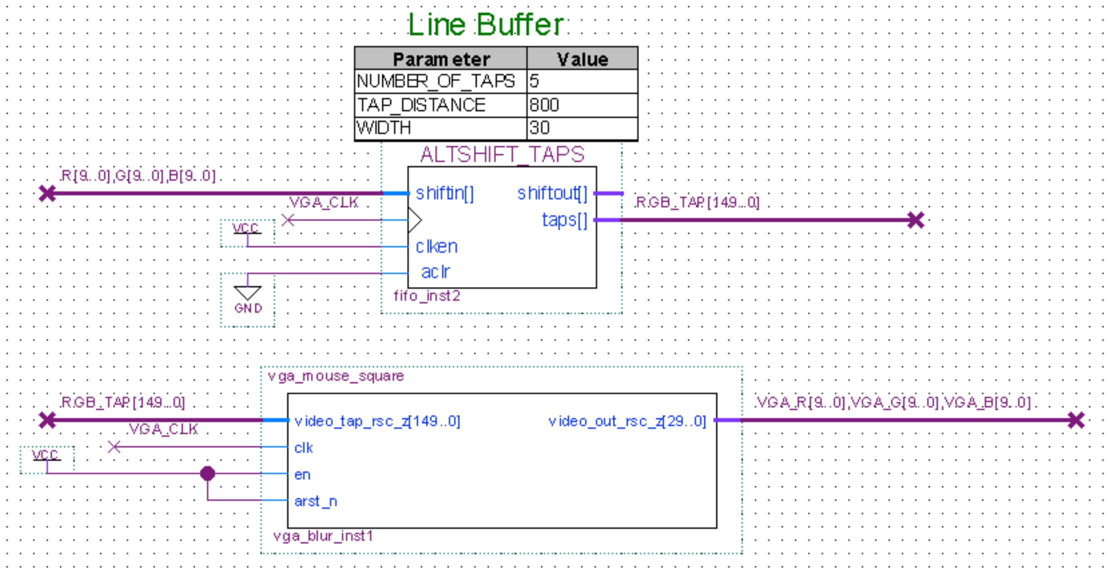


What image does a 5x5 blur filter produce?

## HARDWARE IMPLEMENTATION

Below there's a screenshot of the top level block diagram of this design. It consists of 3 blocks:

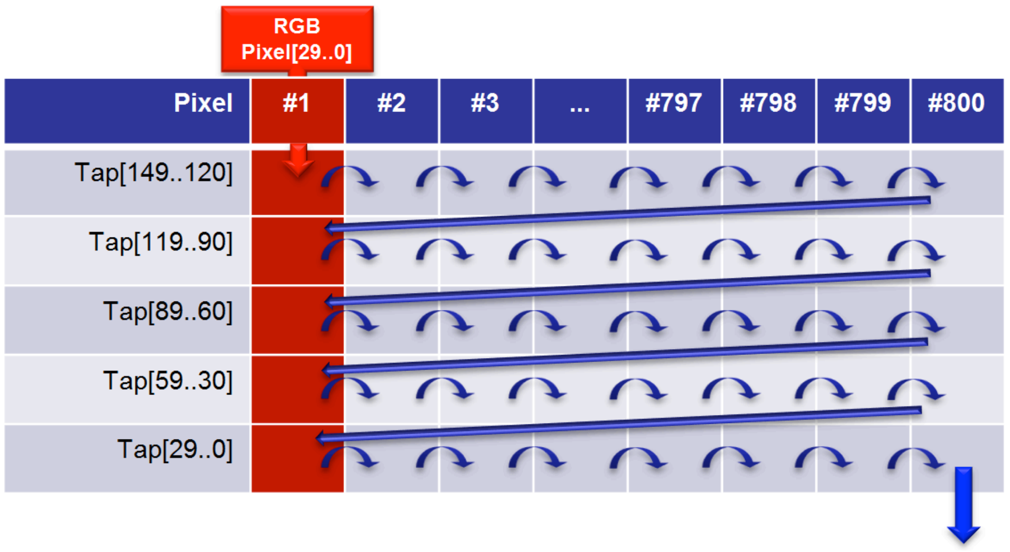
- Line Buffer: is a shift register that stores N lines of pixels according to the window size of the filter. Delivers the pixel in the current VGA scanning coordinates and the N-1 pixels in that column (N-1 previous lines). This unit could have been included in the filter (VGA_Blur), but for simplicity has been left outside.
- VGA_Blur: hardware low-pass filter. Receives the N pixels from the Line Buffer unit.



Line Buffer

| Parameter | Value |
|---|---|
| NUMBER_OF_TAPS | 5 |
| TAP_DISTANCE | 800 |
| WIDTH | 30 |

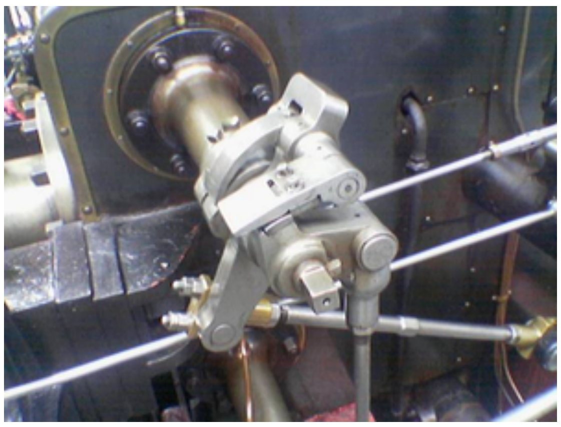
The figure below illustrates the internal operation of the line buffer unit. The matrix represents the pixel storage. The red arrow represents the data from the video decoder unit and the blue arrows represent the "movement" of the data inside.

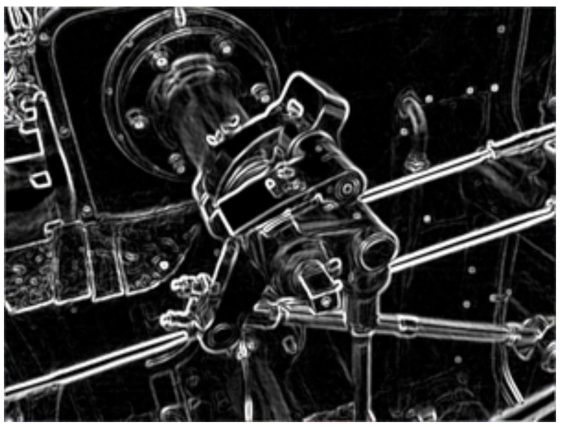


The Line Buffer stores the contents for 5 lines, and each line has 800 pixels. In every clock cycle a new pixel is stored (shifted-in) and another discarded (shifted-out). The signal RGB_TAP[149..0] signal holds the values for the 5 taps. The outputs of the tap signal are updated every clock cycle with the new pixel and the pixels in the same column but from the previous lines. Each tap has 3 colour components R (most significant), G, B (least significant). Each colour component has 10 bits to encode their intensity.

In the designs provided you will find signals have been grouped, e.g. **mouse_xy** corresponds to **mouse_x** in the most significant bits and **mouse_y** in the least significant bits. The word-length of the new signal is the sum of the word-lengths of the original signals. In the Catapult C source files the input signals are "sliced" to recover the individual signals.

Make the necessary changes to the systems on **prj2.zip** to perform an edge-detection filter.





Example of edge detection applied to a colour image. This example can be found here: https://en.wikipedia.org/wiki/Sobel_filter

If you want to increase the size of the window on the filter, you will need to store more lines of your video frame. This can be done in the Line Buffer block.

While developing your block, whenever you update your C source files and want to update the system running on the board, you have to repeat the following steps:

1. Generate RTL (Catapult C);
2. Copy the new generated files to the project location (**rtl*.v**);
3. Open the new **rtl.v** file in your Quartus II project;
4. Create symbol files for current file;
5. Update the symbol in your block diagram;
6. Compile your design;
7. Program Device.

This sequence can take some time to execute. You're advised to rely on simulation to verify your design before running it on the board.

## Part III – Your project

You are expected to implement a system demonstrating image processing techniques and take advantage of parallelism offered by the FPGA. You are encouraged to use the peripherals existent on the board: VGA, PS2 mouse, LEDs, SWs, KEYs, and reuse the controllers given so you don't spend time developing and debugging them. Inside Quartus II there's an application that allows you to generate customized IP blocks and add them to your project. It's called MegaWizard and for each block there's documentation associated with it, so you know how to interface them.

Your project should make use of the video stream as input, and your application should make use of image processing techniques.