

Project Report

SYSC 5807Z

Advance Topics in Computer Systems

Winter 2020

Under the guidance of

Prof. Paulo Garcia

Submitted by-

Karanbir singh sandhu

Student id:

101127504

Carleton University

Table of Contents

- Introduction
- Question1: Automatic and manual parallelization
- Question2 :Interface protocols
- Question3 :Pre-defined interfaces
- Question4 :Parameterizing a design
- Question 5 :Verilog Interface
- Question 7 :External memory support
- Question8 :hardware/software interaction
- Project Implementation

Introduction

Field Programmable Gate Arrays, a semiconductor IC which allows us to make changes to its functionality, in simple words a hardware emulator, this can be achieved using RTL languages, Verilog and VHDL. Although nowadays High level languages can also be used to generate the code for RTL level.

Using C/C++ and Vivado HLS allows users to write code with automatic translation to hardware,i.e into register transfer level code i.e Verilog. This allows users to write highly complex algorithm based FPGA's which can be used for various purposes and allows softwarization of hardware components.

By using Vivado HLS with C/C++, we can easily optimize the code for various metrics such as power,throughput and latency.

We have implemented a sobel filter and image segmentation in our project and source code and testbench files are included with this report.

Q.1.) How much automatic parallelization is performed (versus manual parallelization)?

Automatic parallelization : It is the conversion of sequential code in such a manner that it can utilize the multiprocessor for the execution. This is mainly focused on execution of the loops as they utilize the processing power of a machine to the maximum.

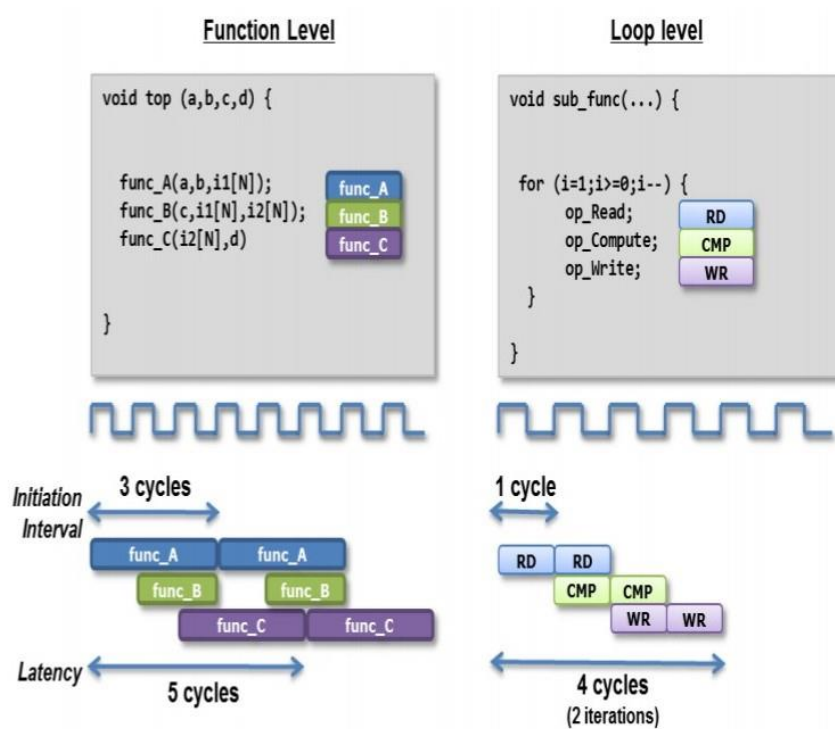
Although by default ,the architecture generated from C/C ++ vivado HLS will be optimized but still be sequential but still by function pipelining and unrolling a high level of parallelization can be achieved

By using C/C++ code in vivado HLS this parallelization is performed by selecting various consecutive tasks and parallelizing those that are independent to one another i.e the data used in the execution of one task is not being used consecutively in the parallel task.

By using

#pragma HLS dataflow

We can trigger the process of pipelining in the Vivado Xilinx and C/C++ code combination. This allows various processes to communicate with each other through channels and parallelization is achieved.



Function and loops with pipelining

We can also unroll loops in a partial manner in subfunctions by using:

```
#pragma HLS pipeline
```

```
#pragma HLS unroll factor=N
```

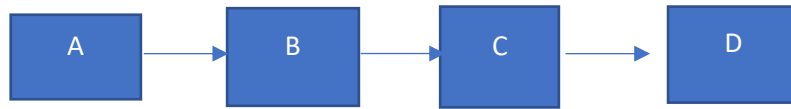
The selection of unroll factor **N** allows scheduling the selected number of accesses **N** each cycle thus allowing task parallelization. Unroll forces parallel execution of instructions in a loop.

Sample code

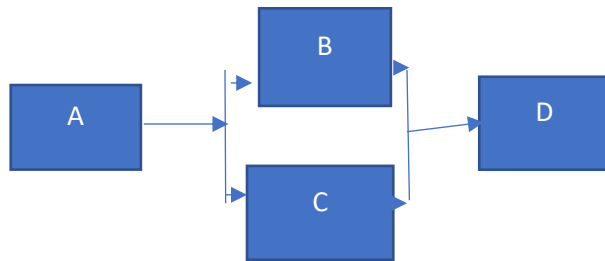
```
void top(int vecIn[10], int vecOut[10])
{
    #pragma HLS DATAFLOW int tmp[10];
    func1(vecIn,tmp);
    func2(tmp,vecOut);
}

void func1(int f1In[10], int f1Out[10])
{
    #pragma HLS PIPELINE
    for(int i=0; i<10;i++) {
        f1Out[i]=f1In[i] *10;
    }
}

Void func2(int f2In[10] , int f2Out[10]) {
    #pragma HLS pipeline
    For(int I =0; I <10; i++) {
        F2Out[i]=f2In[i] +2;
    }
}
```



Sequential execution



Parallel execution with pipelining

Q2. How much support is there to implement interface protocols (e.g., can we create a system that interfaces with the outside world through a protocol that requires an enable signal to be high for one clock cycle, then an address burst followed by aread/write request?)

In a C based design in Vivado Hls we can specify the interface protocols with the use of

#pragma HLS INTERFACE <mode>

The mode can be replaced with the desired interface protocol

We can implement three type of interface protocols

- **Clock and Reset ports** are added to the design by default.

ap_clk and ap_rst

we can also add chip enable port to the design

- **Block-Level Interface Protocols**

ap_start to control the start of processing data

ap_ready indicates when the processor is ready for new input

ap_idle indicates if the hardware is idle

ap_done indicates the completion of an operation

- **Port level Interface Protocols**

ap_none is used for scalar input to a port and specifies no I/O protocol is defined for port.

ap_stable is used for configuration inputs

ap_hs is used for two way handshakes ap_vld and ap_ack for valid and acknowledge handshakes

ap_memory is used for array arguments and includes data ,address,enable and write-enable ports

- **Struct interface**

we can also use structs of C/C++ to manually define interface

eg. typedef struct

```
{ int12 A;
```


int18 B;

int6 C; } ; custom_ports

Various elements of the struct will be treated as separate ports

| Bus Interfaces | | | | | | | | | | | | | | | | |
|----------------|------|--------|----------------|---|----------------|----|---|-------------------|----|---|-------------------|----|---|--------------------|----|---|
| AXI4 | | | Argument | | Variable | | | Pointer Variable | | | Array | | | Reference Variable | | |
| | | | Type | | Pass-by- value | | | Pass-by-reference | | | Pass-by-reference | | | Pass-by-reference | | |
| Stream | Lite | Master | Interface Type | | I | IO | O | I | IO | O | I | IO | O | I | IO | O |
| | | | ap_none | D | | | | D | | | | | | D | | |
| | | | ap_stable | | | | | | | | | | | | | |
| | | | ap_ack | | | | | | | | | | | | | |
| | | | ap_vld | | | | | | | D | | | | | | D |
| | | | ap_ovld | | | | | | D | | | | | | D | |
| | | | ap_hs | | | | | | | | | | | | | |
| | | | ap_memory | | | | | | | | D | D | D | | | |
| | | | ap_fifo | | | | | | | | | | | | | |
| | | | ap_bus | | | | | | | | | | | | | |
| | | | ap_ctrl_none | | | | | | | | | | | | | |
| | | | ap_ctrl_hs | | | | D | | | | | | | | | |
| | | | ap_ctrl_chain | | | | | | | | | | | | | |

Supported Interface

Unsupported Interface

Q3.) How many pre-defined (default) interfaces does it support?

Various interfaces can be used during design synthesis by using

#pragma HLS interface

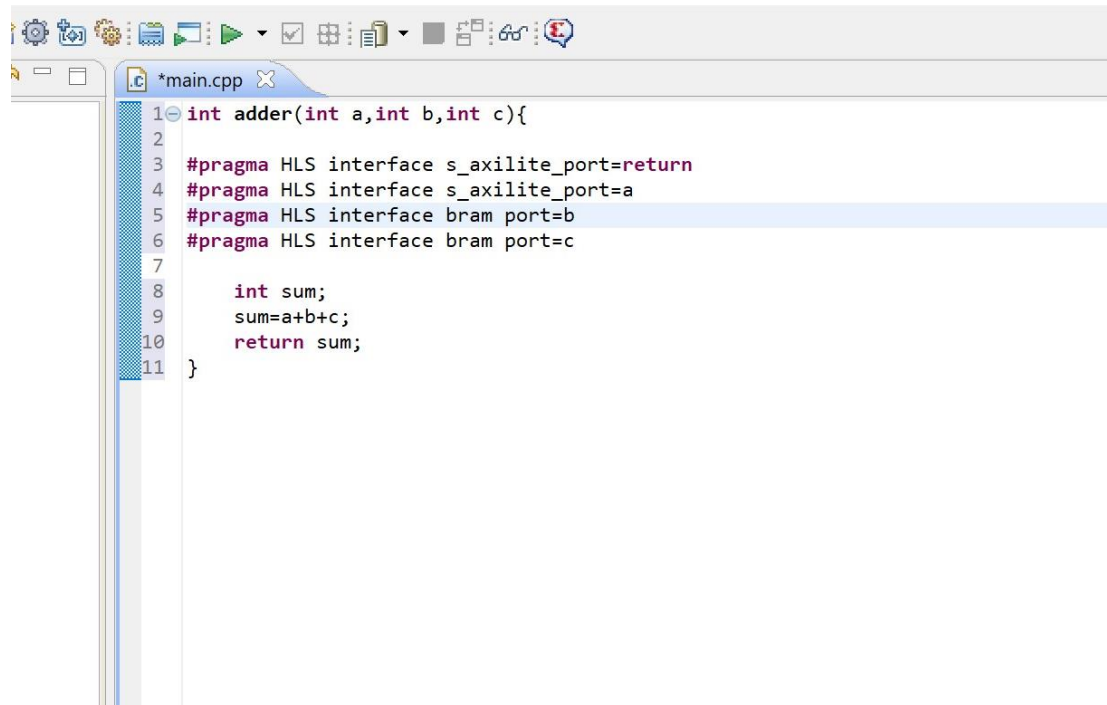
These are :

1.BRAM interface

Fast and reliable

To push data to the memory and reusability of the data

#pragma HLS interface bram port



```
1 int adder(int a,int b,int c){
2
3 #pragma HLS interface s_axilite_port=return
4 #pragma HLS interface s_axilite_port=a
5 #pragma HLS interface bram port=b
6 #pragma HLS interface bram port=c
7
8     int sum;
9     sum=a+b+c;
10    return sum;
11 }
```

2.AXI4 Stream interface-

Used for providing stream of data bursts.

Ideal for image and video processing,convolution filters etc.

#pragma HLS interface axis port

```

1 #include "main.h"
2
3 void thresholdProc(hls::stream<uint_8_channel> &input, hls::stream<int_8_channel> &output, short thresh)
4 {
5     #pragma HLS INTERFACE axis port=input
6     #pragma HLS INTERFACE axis port=output
7     #pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
8     #pragma HLS INTERFACE s_axilite port=thresh bundle=KERNEL_BUS
9
10    // Defining the line buffer and setting the inter dependency to false through pragmas
11    hls::LineBuffer<3,IMG_WIDTH,unsigned char> lineBuff;
12
13
14
15
16
17    int procPix = 0;
18    //delay to fix line-buffer offset
19    int wait = (IMG_WIDTH*(3-1)+3)/2;// 241;
20    int countWait = 0;
21
22
23

```

3.AXI4 Master interface

Mainly used for burst data access

The bundle option allows us to combine various arguments to a single AXI port

#pragma HLS interface M_AXI

rs\karan\AppData\Roaming\Xilinx\Vivado\practice1)

lp

```

1 void example(hls::stream<data_t>& dout,
2 data_t sample[MAX_SIZE], short sample_input)
3 {
4     #pragma HLS INTERFACE port=return s_axilite bundle=ctrl
5     #pragma HLS INTERFACE port=sig_buf s_axilite bundle=ctrl
6     #pragma HLS INTERFACE port=sig_period s_axilite bundle=ctrl
7     #pragma HLS INTERFACE port=dout axis
8

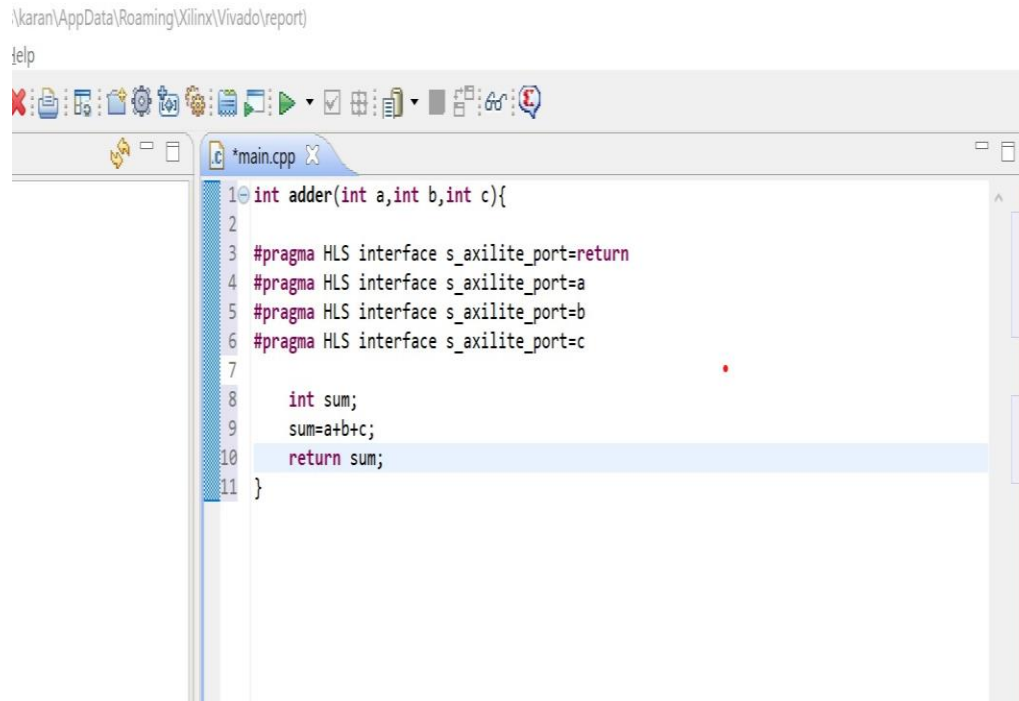
```

4.AXI4 lite slave interface-

Used for IP cores that do not need a stream of data.

To send and receive small size of data

#pragma HLS interface s_axilite port



```
1 int adder(int a,int b,int c){
2
3 #pragma HLS interface s_axilite port=return
4 #pragma HLS interface s_axilite port=a
5 #pragma HLS interface s_axilite port=b
6 #pragma HLS interface s_axilite port=c
7
8     int sum;
9     sum=a+b+c;
10    return sum;
11 }
```

Q4. How parameterizable a design/module is (e.g., can we specify number of bits or type of function as a parameter on sub-module instantiation?)

A design can be parameterized quite comfortably using C/C++ in vivado HLS. We can specify the number of bits by using arbitrary precision data types which allows us to use small bit width variables. This can be achieved by including libraries in our code :

```
#include "ap_cint.h" C support
#include "ap_int.h" C++ support
#include "ap_fixed.h" C++ support
Eg.
```

```
#include "ap_cint.h"
typedef int12 input12t;;
typedef int24 output24t;
dout_t func_sized(input12t x, output24t y)
{
    int z;
    z = (x * y);
    return z;
};
```

In this two 12 bit variables are used as input parameters and a 24 bit output variable is returned. This gives us a multiplier with 24 bit output.

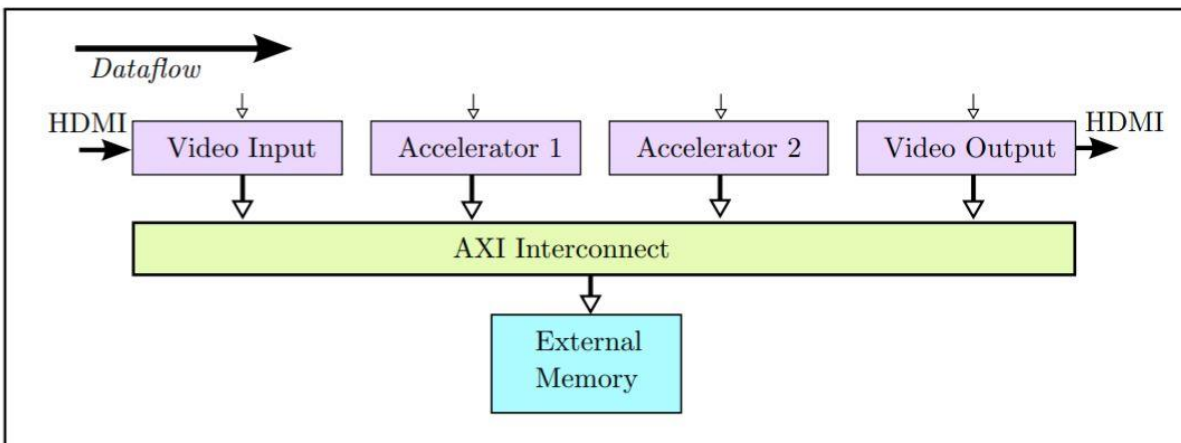
Q5. How well can it interface with Verilog (e.g., can we instantiate sub-modules designed in Verilog?)

Using C/C++ and Vivado HLS is not able to instantiate sub modules designed in Verilog.

It is only able to instantiate sub modules that are designed using C/C++ High level code based sub modules and modules. Although Vivado HLS allows IP Integration using the IP Integrator.

Q7. How much native support is there for interfacing with off-chip systems (e.g., external DRAM)?

While using C/C++ and Vivado HLS ,it is quite easy to access the external memory through AXI4 slave interface that interfaces with and external memory controller connected to external memory.



```
void video filter(pixel t pixel in[MAX HEIGHT][MAX WIDTH], pixel t pixel  
out[MAX HEIGHT][MAX WIDTH]) {
```

```
#pragma HLS interface m_axi port = pixel out
```

```
#pragma HLS interface m_axi port = pixel in
```

This is required usually during video processing applications for large bursts of data that are fed from the external memory.

Q8. How much support is there for hardware/software interactions? None; oneway (software calling hardware accelerators): two-way (hardware calling software-functions as well)?

Even though there is no support for hardware/Software interactions as of now but oneway interaction that is software calling hardware functions can be achieved with IP integration solutions: Altera SOPC and Vivado IP Integrator are two such tools that allow RTL IP Integration by properly interfacing the various components using C/C++.

Project Implementation

Version1

Sobel filter :

In this implementation we have applied a sobel filter on a stream of pixels from our input image and passing them through a linebuffer followed by a iteration on all the pixels and the sobel filter has been applied using a 3*3window and 3*3 sobel kernel 2dconvolution on the window that transverses all the pixels of the image.For every loop HLS pipeline pragma has been used for parallelization to minimize the latency

For the testbench the image has first been converted to a grayscale image and our sobel filter function was applied on our input data.

Libraries used :

hls_video

hls_opencv

Sobel kernel used=

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Input Image:320 *240



Output Image :



Version 2:

Image segmentation:

Image segmentation can be done many ways Out of them most common are K-means algorithm and threshold based algorithm. In the version 2 implementation,

I tried to use the K-means algorithm but due to hardware based limitations and complexity of the algorithm,I opted for a threshold based algorithm .

The truncated -threshold based algorithm for image segmentation has been used in this implementation.

For the testbench the image has first been converted to a grayscale image and our threshold based segmentation function has been applied on our input data.

Threshold value used:100

Input Image:320*240



Output Image:



References:

1. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug902-vivado-high-level-synthesis.pdf
2. https://www.theregister.co.uk/2019/10/01/xilinx_vitis_fpga_launch/
3. <http://kastner.ucsd.edu/wpcontent/uploads/2018/03/admin/pp4fpgas11.12.2018.pdf>
4. <https://forums.xilinx.com/>
5. https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf
6. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug871-vivado-high-level-synthesis-tutorial.pdf