Final Report of Internship in Valencia, Spain

Dates of Internship: **3.09.2016 - 3.12.2016.**

Author: **Tsotne Putkaradze**

Contact Person during internship: Taras Iakymchuk, Taras.Yakymchuk@uv.es

Mentor during internship: Alfredo Rosado Munoz, Alfredo.Rosado@uv.es

Receiving University: **University of Valencia**

Sending University: **Tallinn Technical University**

Introduction

During the internship several sub-tasks have been completed. All of the works are briefly mentioned in the documents. latest source codes of the work are on github server: https://github.com/tsotnep/ETSE_GDSP
The Release of the source codes for which this report was written is on the link: https://github.com/tsotnep/ETSE_GDSP/releases/tag/v3

Vhdl interface for the core

Before the start of the internship, matrix multiplier core has been developed. Previously designed core, now has HDL wrapper, that is designed for two purposes:

- to provide AXI4 lite and AXI4 stream interface with the core
- to gain high control of the matrix multiplier IP

The filename is: MMULT_CONTROLLER_2.vhd. _2 is because _1 version only had AXI4 lite interface. The HDL source consists of several processes for axi streaming interfaces, that has been gathered from AXI4 stream templates, meaning they are valid. More important part of the HDL wrapper is FSM. it has following states:

- cntrl_WAIT_FOR_CMD
    - in other words, IDLE state, where FSM is waiting for command
- cntrl_LOAD_G
    - that redirects AXI stream input to G ram
- cntrl_LOAD_P
    - that redirects AXI stream input to P ram
- cntrl_CALCULTE
    - that command sets matrix mulitplier IP in multiplication state, the details of multiplication: P_lower*G or P_upper*TG or etc, can be specified from signal: **cmd_details.** which is "WDATA(DATA_WIDTH + CMD_SIZE * 2 - 1 downto DATA_WIDTH + CMD_SIZE)" And that **WDATA** signal is AXI4 lite write signal. its value is set from C application.
- cntrl_P_to_G
    - sends P lower or P upper data to G ram.
- cntrl_UNLOAD_G
    - sends G ram data directly through AXI streaming interface to DDR3.
- cntrl_RESET_MMULT_IP
    - resets MMULT_ip. all values and datas.

Axi lite interface with the core

During the second month of internship, interfacing module with IP has been developed. it worked in following manner: while data was written into IP, it was simultaneously checked whether that data was for matrix(by checking the first part of written word) and if it was it was saved into array.
And when the first part of the written word identified the command for controlling interface FSM - then that command was executed instead of writing data to array. if that was calculation command, that array was fed to matrix multiplier IP.

P to G command and G ram address generator.

This command has been added that made it possible to send results from P lower or higher to G ram. the order in which the data was written in G ram was specified from G ram address generator. both of these files has been created during third month of internship.

Axi stream interface with the core using DMA engine

During the third month of internship AXI stream interface has been added to the interface IP. Interface IP is able to communicate with DDR3 memory. DMA engine has been used for communicating with memory. processes of streaming interface is taken from xilinx axi stream interface template.

Simulation and comparison scripts

TCL and PYTHON scripts were created that are simulating design and generating output files, and then comparing the results of matrix multiplication in HDL simulator, to the results of matlab matrix multiplication.

HDL simulation script :

```
for {set i $m} {$i < $M + 1} {incr i} {
  set_property generic COLUMN_TOTAL=$i [get_filesets sim_2]
  launch_simulation
  if { [catch { add_condition -name StoppingCondition1
{/sim_TB_MATRIX_MUL_IP_CORE_S_INT_G/when_to_stop_simulation_flag == 1 } {
    puts "Condition when_to_stop_simulation_flag==1 was encountered at [current_time]. Stopping simulation."
```

```
    close_sim \
  } } err] } {puts ""} \
  else { \
    puts "\n\n**********GOOD. STOPPING CONDITION ADDED**********\n\n" \
  }
  run -all
  close_sim
}
exit
```

The script works in following manner, it sets generic parameter of column, from loop. launches simulation, adds condition when to stop the simulation, this condition is "when_to_stop_simulation_flag==1" and that signal is set to "1" when

"UN_LOADING_DONE = '1' and UN_LOADING_DONE'STABLE(1000 ns)" .

Then it exits simulation, increments generic parameter and continues.

Results Comparison Script

It parses files generated from matlab and hdl simulator. then it subtrackts respective numbers from each other and prints it to folder called: "diff/r". if differences are '0's that is positive, meaning outputs are same. that happens when column size of matrix multiplier is 3,4 and 5. when column size is 6, values already start to differ since DSP size is 18 bits and we get overflow: https://github.com/tsotnep/ETSE_GDSP/blob/master/src/diff-hdlsim-matlab/diff/r/6.txt

Currently results have been uploaded to github for results in range of column size: [3,20].

note that input data to matrices are numbers from 1 to column_size^2. so if we send larget initial data we get overflow earlier.

this link to script is here:
https://github.com/tsotnep/ETSE_GDSP/blob/master/src/diff-hdlsim-matlab/compare_matlab_vhdl_results.py

Baremetal application for interfacing with core

C application has been developed to work with matrix multiplier interface. it works through UART. offers multiple commands to be sent to the matrix multiplier ip. list of allowed commands are following:

:0) Print this list of Commands
:2) load G matrix from memory via stream interface
:3) load P matrix from memory via stream interface
:4) perform calculation of P-lower and G and store in P-s other bank
:41) perform calculation of P-higher and G and store in P-s other bank

:42) perform calculation of P-lower and Gt and store in P-s other bank
:43) perform calculation of P-higher and Gt and store in P-s other bank
:44) perform calculation of Pt-lower and G and store in P-s other bank
:45) perform calculation of Pt-higher and G and store in P-s other bank
:46) perform calculation of Pt-lower and Gt and store in P-s other bank
:47) perform calculation of Pt-higher and Gt and store in P-s other bank
:5) transfer data from P-lower to G
:51) transfer data from P-higher to G
:6) transfer data from G to DDR3 via DMA
:7) print Tx Buffer
:8) print Rx buffer
:11) reset multiplier IP only
:12) reset multiplier IP and it's controller
:13) send multiplier IP's controller in idle state
:21) execute DMA transfer routine 1: identity matrix - 1s on diagonal
:22) execute DMA transfer routine 1s: all values are 1
:23) execute DMA transfer routine 1-9: values from 1 to 9
:24) execute DMA transfer routine 11-33: 11,12,13; 21,22..
:33) perform bundle 1: commands: 24, 3, 24, 2, 4, 51, 6, 8
:34) perform bundle 2: commands: 23, 3, 23, 2, 4, 51, 6, 8
:35) perform bundle 3: commands: 24, 3, 21, 2, 4, 51, 6, 8
:36) perform bundle 4: commands: 23, 3, 22, 2, 4, 41, 5, 6, 8

for example lets take a look on command :36. it does those commands:
23 execute DMA transfer routine 1-9: values from 1 to 9 [1,2,3; 4,5,6; 7,8,9;]
3 load P matrix from memory via stream interface
22 execute DMA transfer routine 1s: all values are 1 [1,1,1; 1,1,1; 1,1,1;]
2 load G matrix from memory via stream interface
4 perform calculation of P-lower and G and store in P-s other bank [store in upper bank]
41 perform calculation of P-higher and G and store in P-s other bank [store in lower bank]
5 transfer data from P-lower to G [from P lower bank to G ram]
6 transfer data from G to DDR3 via DMA [from G ram to DDR3 memory using DMA engine]
8 print Rx buffer [read DDR3 memory addresses, referred as Rx buffer, and print it to uart]

So this gives us the correct values on UART terminal:
18 18 18
45 45 45
72 72 72

Synthesis and Implementation

3x3 core has been implemented on ZYNQ fpga.

	TOTAL LUTs	LOGIC LUTs	LUTRAMs	SRLs	FFs	RAM36	ram18	DSP48 blocks
synthesis	996	987	0	9	1214	0	3	3
implementation	849	840	0	9	1022	0	3	3

reports can be seen by:

```
vivado ETSE_GDSP_DMA_ETHERNET.xpr
open_run synth_1
report_utilization -hierarchical
close_design
#####
open_run impl_1
report_utilization -hierarchical
close_design
#####
```

Instance	Module	TOTAL LUTs	LOGIC LUTs	LUTRAMs	SRLs	FFs	RAM36	ram18	DSP48
ETSE_GDSP	top	994	985	0	9	1214	0	3	3
ETSE_GDSP	top	0	0	0	0	0	0	0	0
-ETSE_GDSP_AXI_inst	ETSE_GDSP_AXI	994	985	0	9	1214	0	3	3
-ETSE_GDSP_AXI_inst	ETSE_GDSP_AXI	47	47	0	0	105	0	0	0
--MMULT_CONTROLLER_2_inst	MMULT_CONTROLLER_2	947	938	0	9	1109	0	3	3
--MMULT_CONTROLLER_2_inst	MMULT_CONTROLLER_2	337	336	0	1	255	0	0	0
---MATRIX_MUL_IP_CORE_S_INT_G_inst	MATRIX_MUL_IP_CORE_S_INT_G	610	602	0	8	854	0	3	3
---MATRIX_MUL_IP_CORE_S_INT_G_inst	MATRIX_MUL_IP_CORE_S_INT_G	11	3	0	8	7	0	0	0
----BLOCK_A_DSP_GEN_1_DSP	DSP_INPUT_C	54	54	0	0	169	0	0	1
----BLOCK_A_DSP_GEN_2_DSP	DSP_INPUT_C_HD9	54	54	0	0	169	0	0	1
----BLOCK_A_MEM_GEN_0_MEMA	BRAM_WRAPPER_V2	1	1	0	0	24	0	1	0
----BLOCK_A_MEM_GEN_1_MEMA	BRAM_WRAPPER_V2__parameterized0	1	1	0	0	24	0	1	0
----BLOCK_A_MEM_GEN_2_MEMA	BRAM_WRAPPER_V2__parameterized1	2	2	0	0	26	0	1	0
----FIRST_DSP	DSP_INPUT_C_HD23	54	54	0	0	169	0	0	1
----FSM_UNIT	CONTROL_UNIT_S_INT_G	266	266	0	0	46	0	0	0
----GRAM	STANDARD_RAM	72	72	0	0	216	0	0	0
----GRAM_ADDRESS_GENERATOR_inst	GRAM_ADDRESS_GENERATOR	95	95	0	0	4	0	0	0

reports can be seen by:

```
vivado ETSE_GDSP_DMA.xpr
launch_runs synth_1
open_run synth_1
report_utilization -hierarchical
close_design
```

TCL scripts

Following TCL scripts have been developed during internship:

build_ip_ETSE_GDSP.tcl

This script is doing following,

- it gets HDL source files, of matrix multiplier
- gets HDL file for interfacing matrix multiplier and which has AXI4 streaming interface modules inside.
- gets HDL files of AXI4 lite that was AXI4 Stream interfaces wires inside.
- add files for simulation, in case we want to simulate the design later.
- packages IP and stores it in directory : `${origin}/tmp/ip_repo/ETSE_GDSP`

build_project_ETSE_GDSP_DMA_ETHERNET.tcl

This script is doing following:

- gets the IP from `${origin}/tmp/ip_repo/ETSE_GDSP`
- adds it to block design
- adds DMA engine, ZYNQ PS and necessary AXI4 modules in the block design.
- runs synthesis and implementation.
- exports design to SDK, in case user wants to use GUI interface, otherwise, rest of the tasks and all of the tasks before can be done without any GUI.

build_app_ETSE_GDSP_DMA.tcl

after exporting the design, this app should be run to gather C sources and build the ELF file that will be run on ARM core.

-

build_app_ETSE_GDSP_DMA_ETHERNET.tcl

this is similar to build_app_ETSE_GDSP_DMA.tcl but it should use ethernet for interfacing instead of UART connection. currently it is not completed due to limited dates of internship, the status is development is following, it simply builds built in C software that is actually an echo server, so that, after programming it will setup and echo server, and it will print the IP of the zedboard on UART terminal like that:

*Board IP: **193.40.246.250***
Netmask : 255.255.255.0
Gateway : 193.40.246.254
TCP echo server started @ port 7

Then it can be tested like this:

echo 1 2 3 4 5 6 7 8 9 | netcat 193.40.246.250 7

and it will print the data in the same terminal.

build_app_FSBL.tcl

This script can be used to create FSBL. in case you want to run the app from SD card.

program_fpga_BITSTREAM_APP.tcl

This script :

- programs fpga with bitstream
- chooses one of the elf files generated before:
    - FSBL
    - ETSE_GDSP_DMA
    - ETSE_GDSP_DMA_ETHERNET
- and programs fpga with that ELF file
- runs the application.

simulate_ip_ETSE_GDSP.tcl

This script was also tcl script mentioned in subchapter: HDL simulation script. It is for simulating the design. It requires two additional variables compared to other scripts, these are: **m** and **M** that are the ranges in which column size of matrix will wary and in which ranges the design will be simulated.

Running TCL scripts

running scripts are easy.

All scripts are tested on **Linux Mint 18** with : **XILINX Vivado Design Suite 2016.1** they should work on Windows. (have not tested on Windows, but should be very much similar, since that TCL script is built in VIVADO and is not specific to operating system).

all TCL scripts are constructed in following manner:
line1: setting variable **origin** that is pointing to the directory of ETSE_GDSP folder.
line2: [not crucial, commented] command to source xilinx tools [all of those commands are same for all scripts]
line3: [not crucial, commented] command to run this specific script. [you can also modify and use this ready command]

For example:

```
set origin /home/tsotne/ownCloud/git/ETSE_GDSP
# source /cad/x_16/Vivado/2016.1/settings64.sh
# vivado -mode tcl -source /home/tsotne/ownCloud/git/ETSE_GDSP/src/tcl/build_ip_ETSE_GDSP.tcl
```

on **Windows** it might require to write first line in following manner:

```
set origin c://user//tsotne//Desktop//ETSE_GDSP
```

in other words, double slashes, worth trying.

Logs of real interference with the IP from UART.

```
            picocom v1.7
port is        : /dev/ttyACM0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
escape is      : C-x
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv
imap is        :
omap is        :
emap is        : crcrlf,delbs,

Terminal ready
```

```
>>>>>>>>>> ENTERING MAIN <<<<<<<<<<
DMA MM2S and S2MM reset bits set
DMA Normal mode is enabled
DMA interrupts are disabled


Please Enter New Command Number (enter '0' for info, '36' for simple test
 that should print: 18,18,18, 45,45,45, 72,72,72) :
Command Received : 0


Please Enter New Command Number :
:0) Print this list of Commands
:2) load G matrix from memory via stream interface
:3) load P matrix from memory via stream interface

:4)  perform calculation of P-lower  and G and store in P-s other bank
:41) perform calculation of P-higher and G and store in P-s other bank
:42) perform calculation of P-lower  and Gt and store in P-s other bank
:43) perform calculation of P-higher and Gt and store in P-s other bank
:44) perform calculation of Pt-lower  and G and store in P-s other bank
:45) perform calculation of Pt-higher and G and store in P-s other bank
:46) perform calculation of Pt-lower  and Gt and store in P-s other bank
:47) perform calculation of Pt-higher and Gt and store in P-s other bank

:5)  transfer data from P-lower to G
:51) transfer data from P-higher to G
:6)  transfer data from G to DDR3 via DMA

:7) print Tx Buffer
:8) print Rx buffer
:11) reset multiplier IP only
:12) reset multiplier IP and it's controller
:13) send multiplier IP's controller in idle state

:21) execute DMA transfer routine 1: identity matrix - 1s on diagonal
:22) execute DMA transfer routine 1s: all values are 1
:23) execute DMA transfer routine 1-9: values from 1 to 9
:24) execute DMA transfer routine 11-33: 11,12,13; 21,22..

:33) perform bundle 1: commands: 24, 3, 24, 2, 4, 51,  6, 8
```

:34) perform bundle 2: commands: 23, 3, 23, 2, 4, 51, 6, 8
:35) perform bundle 3: commands: 24, 3, 21, 2, 4, 51, 6, 8
:36) perform bundle 4: commands: 23, 3, 22, 2, 4, 41, 5, 6, 8
Command Completed Successfully
Command Received : 36
This example, loads P and G matrixes, then executes calculation of P-lower * G,
 and stores it in P-higher, then executes calculation of P-higher to G,
 and stores it in P-lower, and then loads that into G ram and unloads it into DDR3 and prints on UART
DMA is busy
DMA transfer scheduled...
1 2 3
4 5 6
7 8 9
DMA transfer scheduled...
1 1 1
1 1 1
1 1 1
Data written from DMA to DDR3:
18 18 18
45 45 45
72 72 72
Command Completed Successfully

Summary

During internship following skills have been acquired:

- DMA engine and DDR3 from both, C and HDL perspective
- axi streaming interface
- moderate level of ethernet IP in baremetal applications
- designing HDL interfaces for AXI4 lite and streaming interfaces.
- simulating AXI4 lite and streaming interfaces
- writing and organizing TCL scripts for
    - multiple simulation
    - synthesis and
    - baremetal application generation