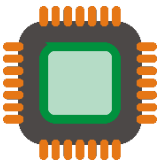
Δευτέρα 06 Απρ. 2019

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ

& ΥΛΙΚΟΥ

«ΗΡΥ 591 ΑΝΑΔΙΑΤΑΣΣΟΜΕΝΑ ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ»



Αναφορά εργασίας εξαμήνου milestone 2

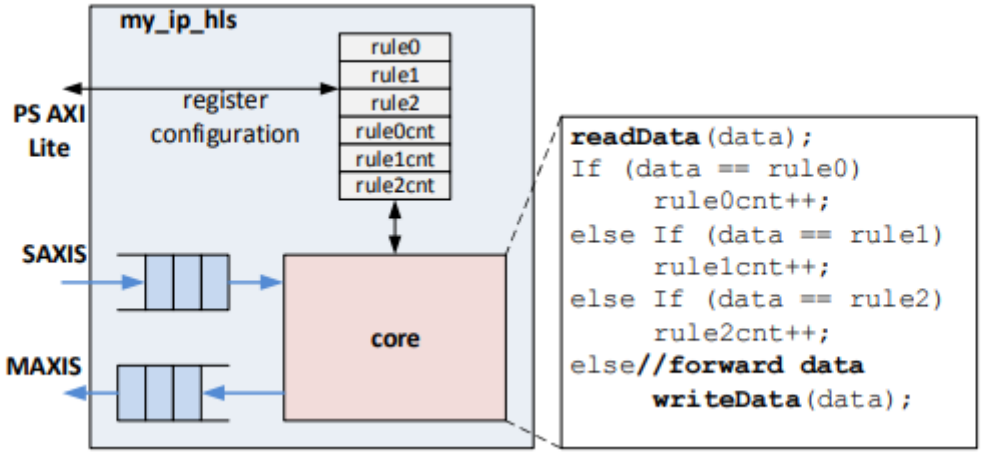
Ομάδα Εργασίας : LAB59140120
Μποκαλίδης Αναστάσιος 2014030069
Χατζηπέτρος Αλέξανδρος 2013030151

Σκοπός εργαστηριακής άσκησης

Σκοπός του milestone 2 είναι η υλοποιήση ενός συστήματος που φιλτράρει μια ροή δεδομένων σε σχέση με προκαθορισμένους κανόνες. Για την υλοποιήση του συστήματος χρησιμοποιήσαμε το Vivado HLS 2017.4 στο οποίο επεκτείναμε τα έτοιμα αρχεία από το δοθέν project. Δηλαδή πραγματώσαμε τις απαραίτητες προσθήκες, ώστε το σύστημα να υποστηρίζει τους καταχωρητές της εκφώνησης καθώς και την επεξεργασία των δεδομένων με βάση τους προκαθορισμένους κανόνες.

Περιγραφή άσκησης

Για την υλοποιήση της εν λόγω εργασίας μας δόθηκε η παρακάτω δομή του συστήματος η οποία αποτελείται από δύο ουρές και ένα core module με σκοπό τον έλεγχο της ροής δεδομένων.



Επιπλέον μας δόθηκε υλοποιήση-κώδικας σε C++ ως reference τον οποίο επεκτείναμε με τις κατάλληλες προσθήκες ώστε να υλοποιηθούν οι λειτουργίες του Milestone 2.

Παρακάτω παρατίθεται η επεξήγηση των module του project:

Rules.cpp

«void rules(uint32 rule0,uint32 &rule0reg,uint32 rule1,uint32 &rule1reg,uint32 rule2,uint32 &rule2reg)»

Στο συγκεκριμένο module δημιουργήσαμε την συνάρτηση **void rule()** στην οποία βάζουμε στους καταχωρήτες rule0reg, rule1reg, rule2reg τις τιμές των κανόνων rule0, rule1, rule2. Οι κανόνες αυτοί ανακτώνται μέσω του PS_AXI_LITE, αυτό επιτυγχάνεται στο my_ip_hls.cpp όπου για κάθε κανόνα δημιουργούμε ένα directive ως interface του PS_AXI_LITE. Συνεπώς όταν ορίζονται τυχαίες τιμές στο my_ip_hls_tb_cpp αυτές προωθούνται και στα υπόλοιπα αρχεία όπου λαμβάνουν χώρα οι κατάλληλοι υπολογισμοί.

Ps2ip_fifo.cpp

«void ps2ip_fifo(stream<axiWord> &ps2ip, stream<axiWord> &ps2ipIntFifo)»

Στο module αυτό υλοποιείται η ουρά για τα δεδομένα στην είσοδο,μέσω της συνάρτησης **void ps2ip_fifo()** αποθηκεύουμε σε μια ουρά τη ροή δεδομένων που μας έρχεται μέσω του AXIS_STREAM. Για την αποθήκευση δεδομένων γίνεται έλεγχος αν η ροή δεδομένων δεν είναι άδεια, δηλαδή περιέχει πληροφορίες-δεδομένα, και αν βρίσκεται σε αρχική κατάσταση ο SLAVE, δηλαδή το interface, ώστε να μπορεί να δεχτεί δεδομένα.

Core.cpp

«void core(stream<axiWord> &ps2ipIntFifo, stream<axiWord> &ip2psIntFifo, uint32 rule0reg, uint32 rule1reg,uint32 rule2reg, rulescnts &cnts)»

Είναι το module στο οποίο γίνεται ο έλεγχος των δεδομένων σε σχέση με τους κανόνες που έχουμε ορίσει. Στην συνάρτηση **void core()** όσα δεδομένα έχουν αποθηκευτεί στην παραπάνω ουρά περνάνε μέσα σε αυτή την συνάρτηση μέσω stream και διαδοχικά το ένα μετά το άλλο συγκρίνονται με τους κανόνες που έχουμε θέσει. Στην περίπτωση που κάποια τιμή από τα δεδομένα είναι ίδια με έναν από τους 3 κανόνες τότε αυξάνεται ο μετρητής που αντιστοιχεί σε αυτόν τον κανόνα. Οι μετρητές των κανόνων απλώς κρατάνε τον αριθμό των δεδομένων όπου ήταν ίδια με τον κανόνα αυτό, επιπλέον σε περίπτωση που πετύχουμε κάποιο κανόνα δεν επιτρέπουμε στα δεδομένα να προχωρήσουν σε επόμενες διαδικασίες. Διαφορετικά προχωράνε στην έξοδο της συνάρτησης και κατευθύνονται προς μια νέα ουρά. Τέλος, οι μετρητές των κανόνων δημιουργήθηκαν ως ένα struct από uint32 μεταβλητές και κάθε φορά χρησιμοποιείται κάποιο μέρος του struct.

Ip2ps_fifo.cpp

«void ip2ps_fifo(stream<axiWord> &ip2psIntFifo, stream<axiWord> &ip2ps)»

Το εν λόγω module αποτελεί την ουρά για τα δεδομένα στην έξοδο. Εφόσουν πραγματοποιηθούν οι κατάλληλοι έλεγχοι στην συνάρτηση **void core()** κάποια δεδομένα δεν καταφέρνουν να προχωρήσουν σε επόμενες διαδικασίες ενώ κάποια αντίστοιχα τα καταφέρνουν. Όσα καταφέρουν να περάσουν έρχονται στην συνάρτηση **void ip2ps_fifo()** και μέσω ενός AXI_STREAM εισάγονται σε μια νέα ουρά. Στην συνέχεια απαιτείται έλεγχος για το αν η ουρά είναι έτοιμη να δεχθεί δεδομένα και αν η ροή δεδομένων μας δεν είναι κενή. Κατόπιν τα δεδομένα εισέρχονται στην ουρά πάλι μέσω του AXI_STREAM και μεταφέρονται έξω από το my_ip_hls. Στην προκειμένη περίπτωση το Interface ονομάζεται Master.

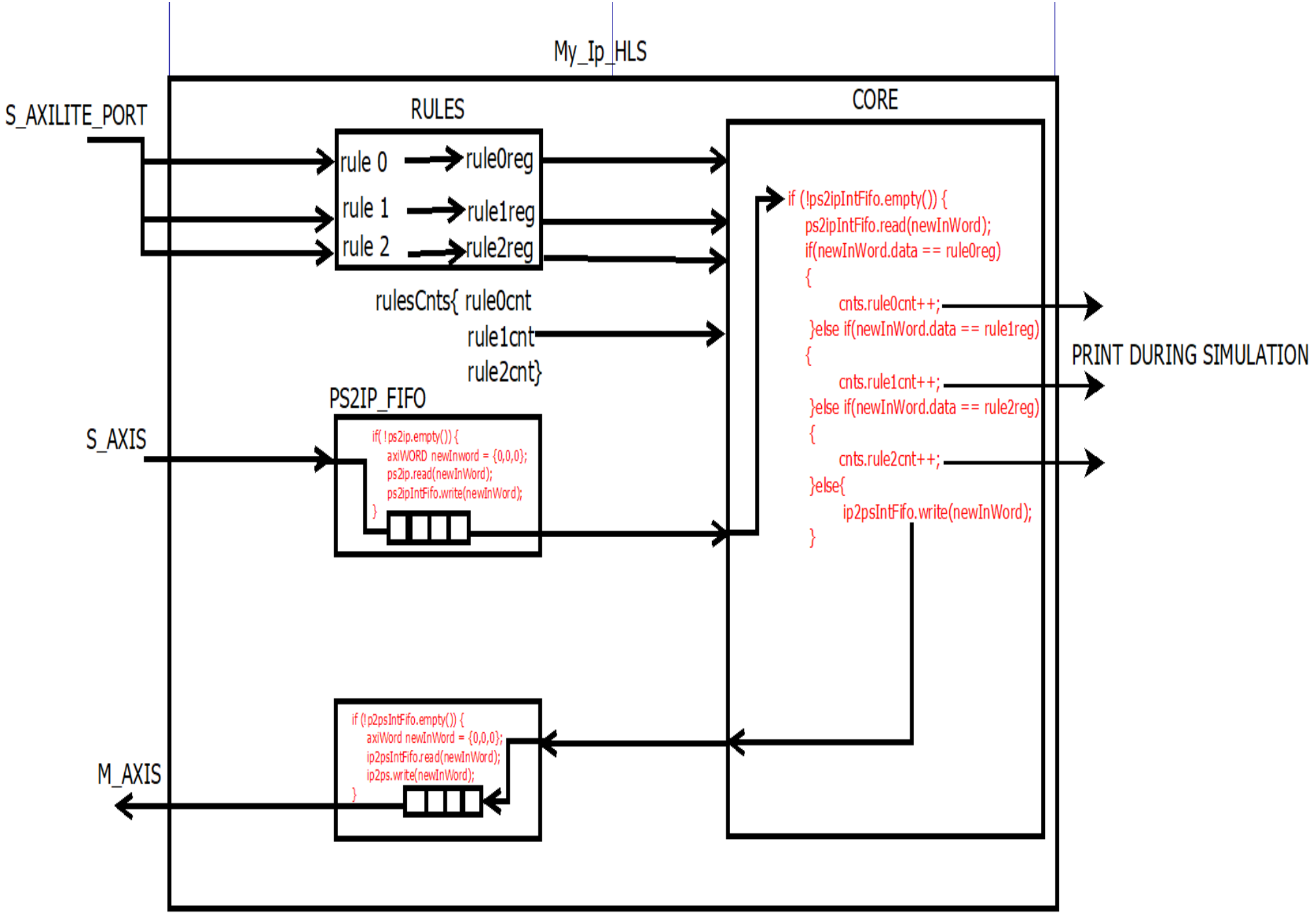
Εν κατακλείδι, μέσω αυτών των διαδικασιών καταφέρνουμε να φιλτράρουμε μια ροή δεδομένων, μπορούμε δηλαδή να ελέγξουμε ποιές πληροφορίες μας είναι χρήσιμες αλλά και να ελέγξουμε τυχόν ύπαρξη κακόβουλων δεδομένων που προέρχονται από τον εξωτερικό κόσμο στο συνολικό σύστημά μας.

Simulation

Για την προσομοίωση του συστήματος στο testbench αρχείο που μας δόθηκε ορίσαμε τους 3 κανόνες ως 10,30,50. Τα δεδομένα που κινούνται μέσω stream παράγονται με μια for loop . Παράγονται 3 φορές ανά δεκάδες οι τιμές από 0 έως 90 , δηλαδή 10,20,30,40,50,60,70,80,90. Στα αποτελέσματα κάθε φορά που κάποιο δεδομένο πετυχαίνει κανόνα εμφανίζεται ως "no data available". Στο τέλος εκτυπώνεται και ο αριθμός των counter που έχουμε για να μετράνε το πόσες φορές πετύχαμε τους κανόνες.

Σχηματική αναπαράσταση modules, Simulation, Synthesis

Παρακάτω παρατίθεται η σχηματική αναπαράσταση των module κατόπιν παρέμβασής μας καθώς και οι μεταξύ τους:


My_Ip_HLS
S_AXILITE_PORT
RULES
rule 0
rule 1
rule 2
rule0reg
rule1reg
rule2reg
rulesCnts{ rule0cnt
rule1cnt
rule2cnt}
CORE
if (!ps2ipIntFifo.empty()) {
ps2ipIntFifo.read(newInWord);
if(newInWord.data == rule0reg)
{
cnts.rule0cnt++;
}else if(newInWord.data == rule1reg)
{
cnts.rule1cnt++;
}else if(newInWord.data == rule2reg)
{
cnts.rule2cnt++;
}else{
ip2psIntFifo.write(newInWord);
}
PRINT DURING SIMULATION
PS2IP_FIFO
S_AXIS
if(!ps2ip.empty()) {
axiWORD newInword = {0,0,0};
ps2ip.read(newInWord);
ps2ipIntFifo.write(newInWord);
}
if (!p2psIntFifo.empty()) {
axiWord newInWord = {0,0,0};
ip2psIntFifo.read(newInWord);
ip2ps.write(newInWord);
}
M_AXIS

my_ip_hls_tb.cpp | my_ip_hls_csim.log

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../my_ip_hls_tb.cpp in debug mode
   Generating csim.exe
0: read data: 0
1: no data available!
2: read data: 20
3: no data available!
4: read data: 40
5: no data available!
6: read data: 60
7: read data: 70
8: read data: 80
9: read data: 90
0: read data: 0
1: no data available!
2: read data: 20
3: no data available!
4: read data: 40
5: no data available!
6: read data: 60
7: read data: 70
8: read data: 80
9: read data: 90
0: read data: 0
1: no data available!
2: read data: 20
3: no data available!
4: read data: 40
5: no data available!
6: read data: 60
7: read data: 70
8: read data: 80
9: read data: 90

rule0cnt : 3 rule1cnt : 3 rule2cnt : 3


INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_rulesconf_AWVALID	in	1	s_axi	rulesconf	scalar
s_axi_rulesconf_AWREADY	out	1	s_axi	rulesconf	scalar
s_axi_rulesconf_AWADDR	in	6	s_axi	rulesconf	scalar
s_axi_rulesconf_WVALID	in	1	s_axi	rulesconf	scalar
s_axi_rulesconf_WREADY	out	1	s_axi	rulesconf	scalar
s_axi_rulesconf_WDATA	in	32	s_axi	rulesconf	scalar
s_axi_rulesconf_WSTRB	in	4	s_axi	rulesconf	scalar
s_axi_rulesconf_ARVALID	in	1	s_axi	rulesconf	scalar
s_axi_rulesconf_ARREADY	out	1	s_axi	rulesconf	scalar
s_axi_rulesconf_ARADDR	in	6	s_axi	rulesconf	scalar
s_axi_rulesconf_RVALID	out	1	s_axi	rulesconf	scalar
s_axi_rulesconf_RREADY	in	1	s_axi	rulesconf	scalar
s_axi_rulesconf_RDATA	out	32	s_axi	rulesconf	scalar
s_axi_rulesconf_RRESP	out	2	s_axi	rulesconf	scalar
s_axi_rulesconf_BVALID	out	1	s_axi	rulesconf	scalar
s_axi_rulesconf_BREADY	in	1	s_axi	rulesconf	scalar
s_axi_rulesconf_BRESP	out	2	s_axi	rulesconf	scalar
ap_clk	in	1	ap_ctrl_none	my_ip_hls	return value
ap_rst_n	in	1	ap_ctrl_none	my_ip_hls	return value
slaveIn_TDATA	in	32	axis	slaveIn_V_data_V	pointer
slaveIn_TSTRB	in	4	axis	slaveIn_V_strb_V	pointer
slaveIn_TLAST	in	1	axis	slaveIn_V_last_V	pointer
slaveIn_TVALID	in	1	axis	slaveIn_V_last_V	pointer
slaveIn_TREADY	out	1	axis	slaveIn_V_last_V	pointer
masterOut_TDATA	out	32	axis	masterOut_V_data_V	pointer
masterOut_TSTRB	out	4	axis	masterOut_V_strb_V	pointer
masterOut_TLAST	out	1	axis	masterOut_V_last_V	pointer
masterOut_TVALID	out	1	axis	masterOut_V_last_V	pointer
masterOut_TREADY	in	1	axis	masterOut_V_last_V	pointer
cnts_rule0cnt_V_i	in	32	ap_ovld	cnts_rule0cnt_V	pointer
cnts_rule0cnt_V_o	out	32	ap_ovld	cnts_rule0cnt_V	pointer
cnts_rule0cnt_V_o_ap_vld	out	1	ap_ovld	cnts_rule0cnt_V	pointer
cnts_rule1cnt_V_i	in	32	ap_ovld	cnts_rule1cnt_V	pointer
cnts_rule1cnt_V_o	out	32	ap_ovld	cnts_rule1cnt_V	pointer
cnts_rule1cnt_V_o_ap_vld	out	1	ap_ovld	cnts_rule1cnt_V	pointer
cnts_rule2cnt_V_i	in	32	ap_ovld	cnts_rule2cnt_V	pointer
cnts_rule2cnt_V_o	out	32	ap_ovld	cnts_rule2cnt_V	pointer
cnts_rule2cnt_V_o_ap_vld	out	1	ap_ovld	cnts_rule2cnt_V	pointer