Technical
University
of Crete

Αναδιατασσόμενα Ψηφιακά Συστήματα

HPY591

Milestone 2

Σπυριδάκης Χρήστος 2014030022
Βίττης Βασίλειος 2015030164

1.Εισαγωγή

Σκοπος του συγκεκριμενου Milestone είναι η επαφή με το εργαλείο Xilinx Vivado HLS, το υπόβαθρο πίσω από την αναδιατασσόμενη λογική σε επίπεδο αφαίρεσης που μας προσφέρει το High Level Synthesis καθώς επίσης και η επαφή με το AXI4 Lite Protocol. Σε πρώτο επίπεδο παρακολουθήσαμε το εισαγωγικό φροντιστήριο που πραγματοποιήθηκε στα πλαίσια του μαθήματος, όπως επίσης και σε δεύτερο επίπεδο είδαμε και μόνοι μας περισσότερα σχετικά με την λειτουργικότητα του HLS. Άρα ήμασταν σε θέση να δημιουργήσουμε το ζητούμενο IP, το οποίο θα είχε την λειτουργικότητα η οποία περιγράφεται στην εκφώνηση.

Σημείωση: Στα σχηματικά και πίνακες τα οποία θα παρουσιαστούν στην συνέχεια της αναφοράς έχει χρησιμοποιηθεί η λογική που υπάρχει στο documentation της HLS - **image 1.1** - σχετικά με το πως αυτή κάνει το synthesize. Με την διαφορά ότι, την ίδια λογική που έχει για τα Top-level function arguments χρησιμοποιήσαμε και εμείς για τα sub-functions τα οποία υπάρχουν. Χωρίς όμως να λάβουμε υπόψιν αυτούσια την χρήση των directives, για να είναι σε ένα προς ένα αναλογία με τον παραδοτέο κώδικα.

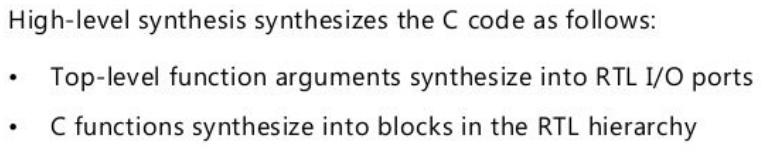


Image 1.1 - HLS synthesize patterns

---

2. Υλοποίηση

Πρώτο βήμα πριν ξεκινήσουμε την υλοποίηση, ήταν τόσο να εισάγουμε στο αναπτυξιακό περιβάλλον το reference code όσο και να το τρέξουμε για να δούμε τα αποτελέσματα του, με γνώμονα να κατανοήσουμε την λειτουργικότητα του κώδικα. Αφού το κάναμε αυτό είδαμε ένα-ένα τα υπάρχοντα modules και παρακάτω καταγράφουμε λίγα πράγματα σχετικά με αυτά, μαζί με τις αλλαγές που χρειάστηκε να κάνουμε.

2.1 my_ip_hls.hpp

Είναι το header αρχείο στο οποίο υπάρχουν όλα τα function *declarations* (prototypes) για όλες τις συναρτήσεις που χρησιμοποιούνται στο IP μαζί με κάποια typedef και struct. Το μόνο που χρειάζεται να αναφερθεί είναι ότι προσαρμόστηκαν τα prototypes των functions σύμφωνα με τις αλλαγές που κάναμε. Δεν δίνεται περαιτέρω επεξήγηση καθώς πέρα από τη

χρήση του **ap_uint<x>** που υπάρχει, και έχει να κάνει με τα σήματα και το πλάτος τους, είναι ένα κλασικό header file.

2.2 my_ip_hls.cpp

Στο συγκεκριμένο module ‘φιλοξενείται’ όλη η λειτουργικότητα του IP. Είναι το αρχείο στο οποίο έπρεπε να δοθεί η περισσότερη προσοχή, καθώς σε αυτό πραγματοποιείται η διασύνδεση με χρήση συγκεκριμένων πρωτοκόλλων με τον ‘’έξω κόσμο’. Για να γνωρίζει το εργαλείο αυτήν την ιδιαιτερότητα των παραμέτρων της **my_ip_hls**, χρειάστηκε να χρησιμοποιήσουμε directives. Αφήσαμε ίδια τα κομμάτια του reference code, που είχαν να κάνουμε με το AXI4 Stream ενώ για το AXI4 Lite χρησιμοποιήσαμε s_axilite INTERFACE και τα βάλαμε όλα στο ίδιο bundle. Παρακάτω παραθέτουμε το κομμάτι κώδικα για το οποίο μόλις αναφερθήκαμε ώστε να γίνει περισσότερο κατανοητό.

my_ip_hls.cpp

```
...
void my_ip_hls( stream<axiWord> &slaveIn,stream<axiWord> &masterOut,
                    uint32 rule0, uint32 rule1, uint32 rule2,
                     uint32 &rule0cnt, uint32 &rule1cnt, uint32 &rule2cnt) {

#pragma HLS DATAFLOW interval=1
#pragma HLS INTERFACE axis register both port=slaveIn
#pragma HLS INTERFACE axis register both port=masterOut
#pragma HLS INTERFACE ap_ctrl_none port=return

// Rules
#pragma HLS INTERFACE s_axilite port=rule0 bundle=psAxiLite
#pragma HLS INTERFACE s_axilite port=rule1 bundle=psAxiLite
#pragma HLS INTERFACE s_axilite port=rule2 bundle=psAxiLite
// Rules' Counters
#pragma HLS INTERFACE s_axilite port=rule0cnt bundle=psAxiLite
#pragma HLS INTERFACE s_axilite port=rule1cnt bundle=psAxiLite
#pragma HLS INTERFACE s_axilite port=rule2cnt bundle=psAxiLite
...
```

Επίσης σε αυτό το αρχείο δημιουργήσαμε τους register στους οποίους θα κρατάμε τα rules, με την χρήση του keyword static.

my_ip_hls.cpp

```
…
static uint32 rule0Reg=0;
static uint32 rule1Reg=0;
static uint32 rule2Reg=0;
...
```

Τέλος απλά καλούμε τις συναρτήσεις που αναλύονται παρακάτω, για να επιτευχθεί η απαραίτητη λειτουργικότητα.

2.3 core.cpp

Είναι ο πυρήνας του IP, σε αυτόν εισέρχονται τα πακέτα του χρειάζεται να φιλτράρουμε, υπάρχουν οι register με τους counters για το κάθε φιλτρο όπως επίσης και από αυτόν εξέρχεται σε περίπτωση που επιτραπεί ένα πακέτο. Ο template κώδικας έχει παραμείνει ίδιος με σημαντική εξαίρεση ότι πλέον γίνεται το φιλτράρισμα ακριβώς με την λογική που υπάρχει στην εκφώνηση του milestone. Τέλος όπως είπαμε εδώ υπάρχουν οι register των counters οι οποίοι όμως είναι δηλωμένοι με ίδια λογική όπως περιγράφηκε παραπάνω, έπειτα αυξάνονται όταν χρειάζεται - και μόνο τότε - ο καθένας και τέλος δίνονται στον έξω κόσμο.

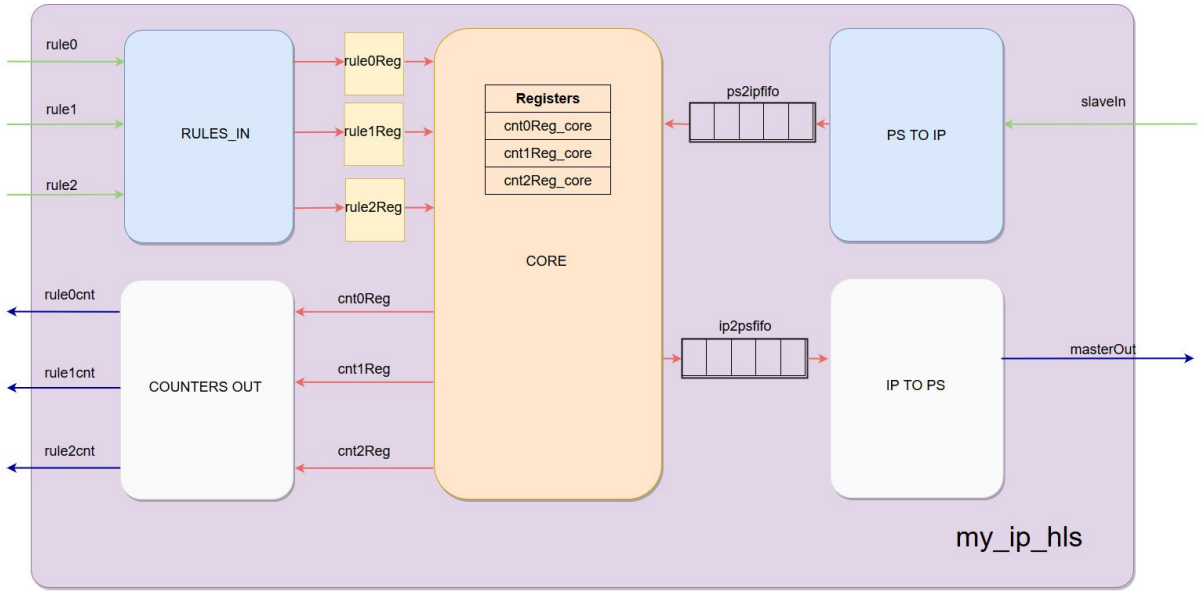
2.4 ps2ip_fifo.cpp και ip2ps_fifo.cpp

Σε αυτά τα δύο modules δεν χρειάστηκε να επέμβουμε καθώς ήταν ήδη λειτουργικά και η δουλειά τους είναι το ένα να διαβάζει από το ένα AXI4 Stream και να βάζει τα δεδομένα σε μία εσωτερική ουρά η οποία θα δινόταν στο core για έλεγχο, ενώ το άλλο ακριβώς το ανάποδο. Παίρνει την ουρά που σε αυτήν έχουν δοθεί δεδομένα από το core και τα διοχετεύει στο δεύτερο AXI4 Stream.

2.5 rules_in.cpp και counters_out.cpp

Αυτά τα modules τα δημιουργήσαμε εμείς προκειμένου να καταφέρουμε να υλοποιήσουμε το project. Ουσιαστικά έχουν παρόμοια χρήση με τα ps2ip_fifo.cpp και ip2ps_fifo.cpp. Όπως λένε και τα ονόματα τους σκοπός του rules_in.cpp είναι να διαβάζει αυτά που έρχονται ως rules από το AXI4 Lite σαν είσοδοι και να τα βάζει σε εσωτερικούς register. Ενώ το counters_out.cpp ακριβώς το ανάποδο, δηλαδή να δίνει ως έξοδο στο AXI4 Lite τις τιμές των register που είναι αποθηκευμένες οι τιμές των counters.

Σύμφωνα με την παραπάνω ανάλυση των επιμέρους modules, προκύπτει το παρακάτω σχηματικό στο οποίο εμφανίζονται οι βασικές συνδέσεις των modules μεταξύ τους.



3. Testing

Προκειμένου να ελέγξουμε την λειτουργικότητα του IP έπρεπε τόσο να δούμε ότι έχει τα επιθυμητά αποτελέσματα σε simulation όσο και ότι το ip μπορεί να γίνει επιτυχώς synthesize.

3.1 Simulation

Προκειμένου να ελέγξουμε την λειτουργία του κώδικα ακολουθήσαμε το παράδειγμα της εκφώνησης και βάλαμε στους κανόνες rule1, rule2 και rule3 τις τιμές 10, 30 και 50 αντίστοιχα.
Έτσι, όπως είναι προφανές με τις εισόδους 10,20,30,40,50,60,70 και 50, η πρώτη είσοδος θα αντιστοιχηθεί με τον πρώτο κανόνα μια φορά ενώ η τρίτη είσοδος με τον δεύτερο κανόνα επίσης μια φορά και η είσοδος με τον αριθμό 50 θα αντιστοιχηθεί με τον τρίτο κανόνα δυο φορές. Άρα περιμένουμε οι αντίστοιχοι μετρητές να ειναι για τον πρώτο κανόνα ένα, για τον δεύτερο ένα και για τον τρίτο κανόνα δυο. Ενώ οι υπόλοιπες είσοδοι θα περάσουν τον έλεγχο και θα γραφτούν κανονικά στην FIFO μας.
Παρακάτω βλέπουμε τα αποτελέσματα από το simulation που επιβεβαιώνουν τον παραπάνω συλλογισμό μας.

```
Rule0: 10       | Rule1: 30     | Rule2: 50
Stream Data:
 10, 20, 30, 40, 50, 60, 70, 50
---------------------------------
Begin Simulation...
0: no data available!
1: read data: 20
2: no data available!
3: read data: 40
4: no data available!
5: read data: 60
6: read data: 70
7: no data available!


Results:
  Counter 0 : 1
  Counter 1 : 1
  Counter 2 : 2
---------------------------------
```

Image 3.1.1 - Reference simulation

3.2 Synthesize

Παρακάτω θα παρουσιάσουμε κάποια ενδιαφέρον σημεία από το Synthesis Report. Προκειμένου να το κάνουμε αυτό, παραθέτουμε τις παρακάτω εικόνες στις οποίες μπορούμε να δούμε αρχικά την ύπαρξη των έξι registers οι οποίοι χρησιμοποιήθηκαν, τρεις για τα rules και τρεις τα counters. Ενώ στην συνέχεια μπορούμε να δούμε το τελικό Summary στο οποίο εμφανίζεται το Interface του IP module το οποίο δημιουργήθηκε.

Register

Name	FF	LUT	Bits	Const Bits
ap_sync_reg_channel_write_cnt0Reg_V	1	0	1	0
ap_sync_reg_channel_write_cnt1Reg_V	1	0	1	0
ap_sync_reg_channel_write_cnt2Reg_V	1	0	1	0
ap_sync_reg_channel_write_rule0Reg_V_channel	1	0	1	0
ap_sync_reg_channel_write_rule1Reg_V_channel	1	0	1	0
ap_sync_reg_channel_write_rule2Reg_V_channel	1	0	1	0
Total	6	0	6	0

Image 3.2.1 - Register

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_psAxiLite_AWVALID	in	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_AWREADY	out	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_AWADDR	in	6	s_axi	psAxiLite	pointer
s_axi_psAxiLite_WVALID	in	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_WREADY	out	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_WDATA	in	32	s_axi	psAxiLite	pointer
s_axi_psAxiLite_WSTRB	in	4	s_axi	psAxiLite	pointer
s_axi_psAxiLite_ARVALID	in	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_ARREADY	out	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_ARADDR	in	6	s_axi	psAxiLite	pointer
s_axi_psAxiLite_RVALID	out	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_RREADY	in	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_RDATA	out	32	s_axi	psAxiLite	pointer
s_axi_psAxiLite_RRESP	out	2	s_axi	psAxiLite	pointer
s_axi_psAxiLite_BVALID	out	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_BREADY	in	1	s_axi	psAxiLite	pointer
s_axi_psAxiLite_BRESP	out	2	s_axi	psAxiLite	pointer
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

Image 3.2.2 - Interface

---

4. Μελλοντική δουλειά

Αν ένα από τα ζητούμενα του συγκεκριμένου Milestone ήταν πώς θα μπορούσε να βελτιωθεί/διαφοροποιηθεί σε θέμα λειτουργικότητας ο παραδοτέος κώδικας, η απάντηση μας θα ήταν ότι θα προσπαθούσαμε να μηδενιζουμε τους counters όταν βλέπουμε ότι εισάγεται ένα διαφορετικό rule σε μία θέση που την ακριβώς προηγούμενη στιγμή υπάρχει κάποιο άλλο. Μεν δεν ήταν απαραίτητο σε αυτήν την φάση του project σύμφωνα με τις προδιαγραφές, αλλά θα μας έδινε την δυνατότητα να αλλάξουμε κατά την λειτουργία τα ‘φιλτρα’ τα οποία θέλουμε να ελέγξουμε. Έτσι θα ήμασταν κοντά σε πιο ρεαλιστικά πλάνα λειτουργίας.