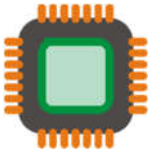
ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ

ΗΡΥ 591 ΑΝΑΔΙΑΤΑΣΣΟΜΕΝΑ

ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2018-19



Εργασία εξαμήνου

Εαρινό εξάμηνο 2018-19, ΕΚΔΟΣΗ 1.2

Δημήτρης Θεοδωρόπουλος

Σύντομη περιγραφή

Στην εργασία αυτή θα έχετε την ευκαιρία να έρθετε σε επαφή με ένα επαναπρογραμματιζόμενο System-on-Chip (reconfigurable SoC - rSoC). Πιο συγκεκριμένα, θα σχεδιάσετε και υλοποιήσετε σε hardware έναν σύστημα, χρησιμοποιώντας (α) τη γλώσσα περιγραφής υλικού VHDL, και (β) γλώσσα υψηλού επιπέδου για την περιγραφή υλικού (High-Level Synthesis HDLs). Όλη τη λογική που θα σχεδιάσετε, θα τη συνδέσετε με έναν ενσωματωμένο επεξεργαστή (embedded processor), και κατόπιν θα δείτε την απόδοση των κυκλωμάτων με προσομοίωση όλου του συστήματος.

Για την εργασία αυτή, θα χρησιμοποιήσετε το περιβάλλον σχεδίασης και υλοποίησης hardware Vivado 2017.4 και Vivado HLS 2017.4 της Xilinx, ενώ η πλατφόρμα υλοποίησης θα είναι ένα zc706 evaluation board. Το τελευταίο περιλαμβάνει ένα Xilinx Zynq7040 SoC που ενσωματώνει έναν dual-core ARM και αναδιατασσόμενη λογική.

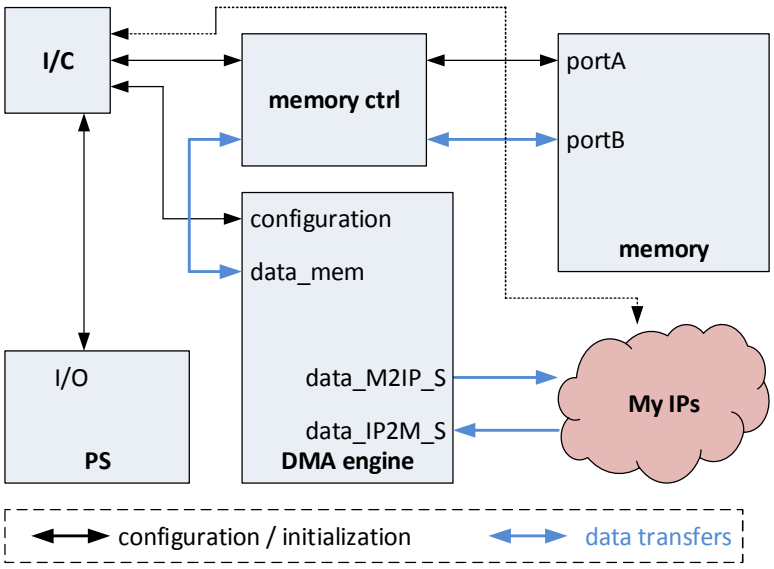
Εισαγωγή

Μία από τις πιο γνωστές προσεγγίσεις κατά τη διάρκεια εκτέλεσης μιας εργασίας σε programmable logic (PL) είναι η χρήση των Direct Memory Access engines (DMAe). Η βασική ιδέα είναι ο επεξεργαστής να προγραμματίσει το DMAe με συγκεκριμένες δομές (descriptors) που θα αναφέρουν (α) πόσα δεδομένα και από διεύθυνση να διαβάσει, ή (β) πόσα δεδομένα να περιμένει και σε ποια διεύθυνση να τα γράψει.

Η Εικόνα 1 δείχνει τη γενική δομή ενός rSoC, που χρησιμοποιεί ένα DMAe για τη μεταφορά δεδομένων από και προς τη μνήμη. Όπως φαίνεται και στο σχήμα, υπάρχουν τα εξής modules:

- Processing System (PS): Ο επεξεργαστής που τρέχει την εφαρμογή και μπορεί να επικοινωνήσει με υλικό στο PL μέσω ενός I/O port.

- Interconnect (I/C): Διασύνδεση που επιτρέπει την επικοινωνία μεταξύ modules.
- Memory ctlr: Ο memory controller που επιτρέπει την πρόσβαση σε data από το PS αλλά και οποιοδήποτε module στο PL.
- Memory: Η μνήμη που αποθηκεύονται δεδομένα είτε προς επεξεργασία, είτε ως αποτελέσματα κατόπιν επεξεργασίας.
- My IPs: Όλη η λογική που έχουμε βάλει και μπορεί να έχει πρόσβαση στη μνήμη μέσω του DMAe. Πιθανώς η λογική να μην επικοινωνεί μόνο με το DMAe, αλλά να έχει σύνδεση και με το PS, προκειμένου το τελευταίο να μπορεί να περάσει παραμέτρους ή να δει την κατάσταση της λογικής.
- DMA engine (DMAe): Το module που επιτρέπει την πρόσβαση στη μνήμη χωρίς να επηρεάζεται η λειτουργία του PS. Πιο συγκεκριμένα, το DMAe έχει τα εξής σήματα:
    - Configuration: Interface για να μπορεί το PS να προγραμματίσει ή να δει την κατάσταση του DMAe.
    - Data_mem: Memory-mapped interface για να ανταλλάσσει το DMAe δεδομένα με τη μνήμη.
    - Data_M2IP_S: Streaming interface για τη μεταφορά δεδομένων από το DMAe προς τη λογική μας.
    - Data_IP2M_S: Streaming interface για τη μεταφορά δεδομένων από τη λογική μας προς το DMAe.



Εικόνα 1 - Γενική δομή ενός rSoC με DMAe.

Μια τυπική περίπτωση του παραπάνω συστήματος είναι η λογική που έχουμε βάλει να διαβάσει δεδομένα από τη μνήμη, να τα επεξεργαστεί, και στη συνέχεια να γράψει πίσω στη μνήμη τα αποτελέσματα. Η σειρά των βημάτων για τη λειτουργία του παραπάνω συστήματος θα είναι η ακόλουθη:

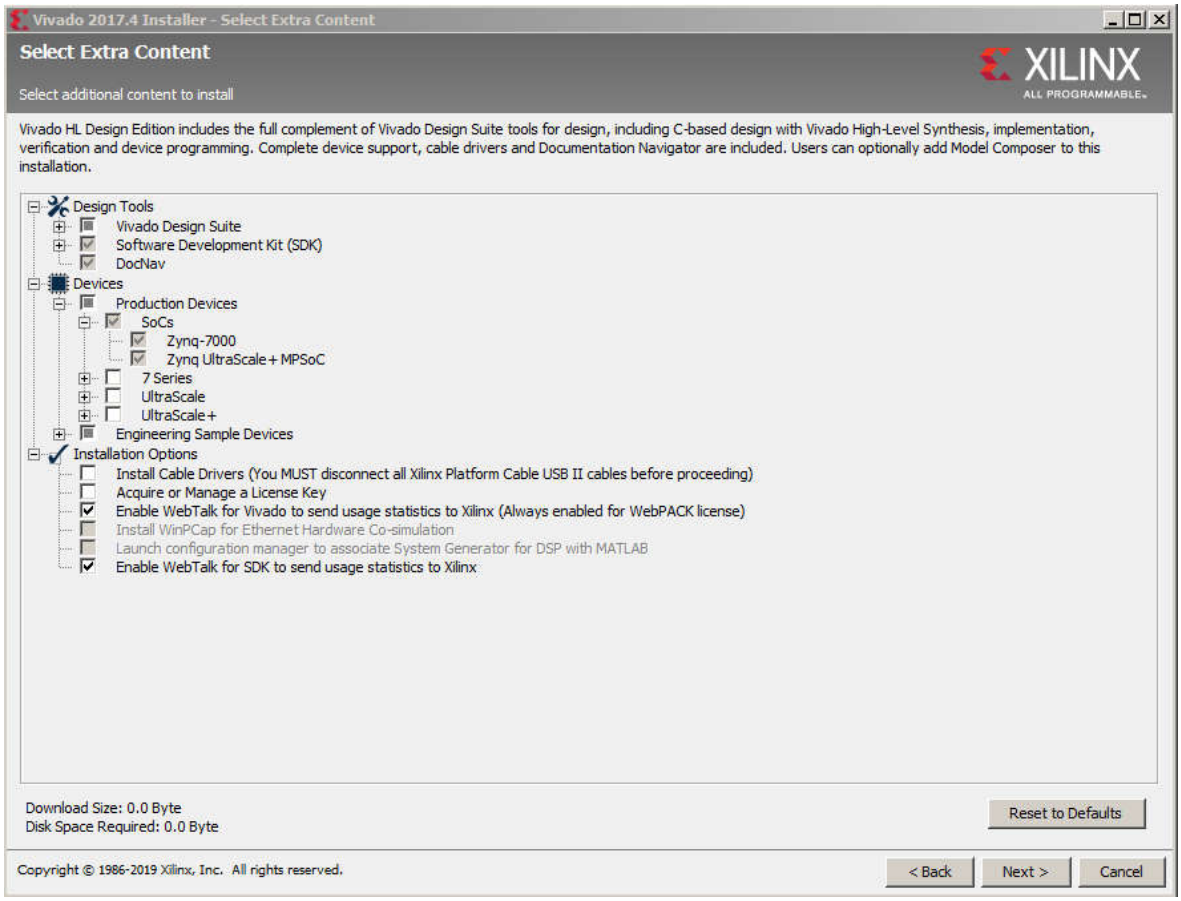
1. Το PS γράφει Χ δεδομένα προς επεξεργασία στη μνήμη με αρχική διεύθυνση Α.
2. Το PS προγραμματίζει τον receiver του DMAe με τη διεύθυνση Β στην οποία θα αποθηκεύσει τα αποτελέσματα, καθώς και τον αναμενόμενο αριθμό τους Υ (DMAe → configuration).
3. Το PS προγραμματίζει τον transmitter του DMAe με τη διεύθυνση Α από την οποία θα διαβάσει τα δεδομένα προς επεξεργασία, καθώς και τον αριθμό τους Χ (DMAe → configuration).
4. Το DMAe ξεκινάει να μεταφέρει όλα τα δεδομένα προς τη λογική μας.

5. Καθώς η λογική μας αρχίζει να «παράγει» αποτελέσματα, αυτά επιστρέφουν στο DMAe, το οποίο με τη σειρά του τα προωθεί προς αποθήκευση, ξεκινώντας από τη διεύθυνση B.

Εγκατάσταση εργαλείων

Στα πλαίσια της εργασίας, όπως αναφέρθηκε, θα χρησιμοποιήσετε το Vivado 2017.4 και Vivado HLS 2017.4 της Xilinx. Η πλατφόρμα υλοποίησης θα είναι ένα zc706 evaluation board, που περιλαμβάνει ένα Xilinx Zynq7040 SoC, το οποίο ενσωματώνει έναν dual-core ARM και αναδιατασσόμενη λογική.

Για την εγκατάσταση, θα κατεβάσετε το Vivado 2017.4 HL design edition (full version). Για την εγκατάσταση **χρησιμοποιήστε path που δεν έχει ελληνικούς χαρακτήρες ή κενά, καθώς επίσης να είναι σίγουρο πως θα υπάρχουν write permissions από το OS**. Όταν σας ζητηθούν ποια devices να βάλετε, επιλέξτε τα SoC → Zynq 7000, όπως φαίνεται παρακάτω:



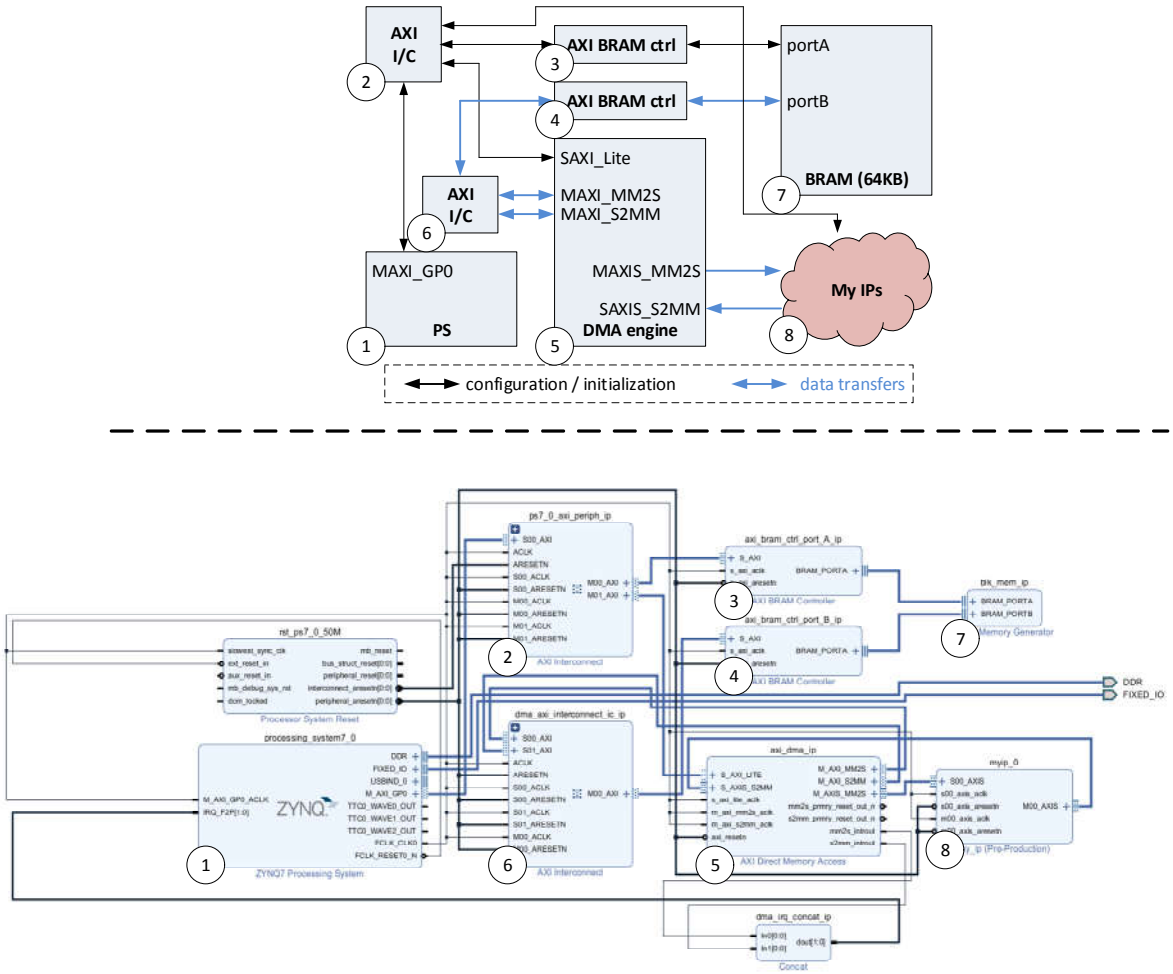
Εικόνα 2 - Επιλογή της Zynq 7000 device κατά τη διάρκεια του installation.

Περιβάλλον εργασίας

Η Εικόνα 3 δείχνει στο πάνω μέρος πώς μπορεί να γίνει η σύνδεση του PS με το DMAe και μια λογική, χρησιμοποιώντας τα AXI4 και AXI4 Stream interfaces. Στο κάτω μέρος της ίδιας εικόνας φαίνεται το block design στο Vivado. Με βάση τους αριθμούς που φαίνονται στο σχήμα είναι:

1. Το PS που τρέχει την εφαρμογή, και θα κάνει configure το DMAe.
2. Το AXI4 interconnect που επιτρέπει την επικοινωνία μεταξύ του PS και της λογικής μας, ενός AXI4 memory controller, και του DMAe.

3. Ο 1ος memory controller που επιτρέπει την προσπέλαση στη μνήμη από το PS.
4. Ο 2ος memory controller που επιτρέπει την προσπέλαση στη μνήμη από το DMAe.
5. Το DMAe που χρησιμοποιείται για τη μεταφορά δεδομένων μεταξύ λογικής ⇔ DMAe.
6. Το AXI4 interconnect που επιτρέπει το DMAe να προσπελάσει τη μνήμη μέσω του memory controller.
7. Η μνήμη που αποθηκεύονται δεδομένα προς επεξεργασία και αποτελέσματα.
8. Η λογική μας.



Εικόνα 3 - Αναπαράσταση του rSoC που υπάρχει με όλα τα interfaces (πάνω) και το αντίστοιχο block diagram στο Vidado).

1ο Milestone

Το project που έχετε κατεβάσει έχει το my_ip_0 module, που στην ουσία είναι ένας wrapper για AXI4 Stream interfaces σε master και slave modes. Σκοπός είναι να αλλάξετε τον κώδικα VHDL που υπάρχει και να φτιάξετε μια FIFO ουρά 64 θέσεων (32 bits μέγεθος η κάθε λέξη), η οποία θα έχει τα παρακάτω σήματα:

Τύπος	Όνομα	Μέγεθος (bits)	Τύπος	Περιγραφή
Slave AXI4 Stream interface	S_AXIS_tvalid	1	Είσοδος	Δηλώνει ότι στο tdata υπάρχουν δεδομένα για αποθήκευση στη FIFO.
	S_AXIS_tdata	32	Είσοδος	Τα δεδομένα προς αποθήκευση στη FIFO.
	S_AXIS_tready	1	Έξοδος	Δηλώνει πως η FIFO είναι έτοιμη να δεχτεί νέα δεδομένα.
Master AXI4 Stream interface	M_AXIS_tvalid	1	Έξοδος	Δηλώνει ότι στο tdata υπάρχουν τα πιο «παλιά» δεδομένα που είχαν αποθηκευτεί στη FIFO.
	M_AXIS_tdata	32	Έξοδος	Τα πιο «παλιά» δεδομένα που είχαν αποθηκευτεί στη FIFO.
	M_AXIS_tready	1	Είσοδος	Δηλώνει πως η το DMAe είναι έτοιμο να δεχτεί νέα δεδομένα.

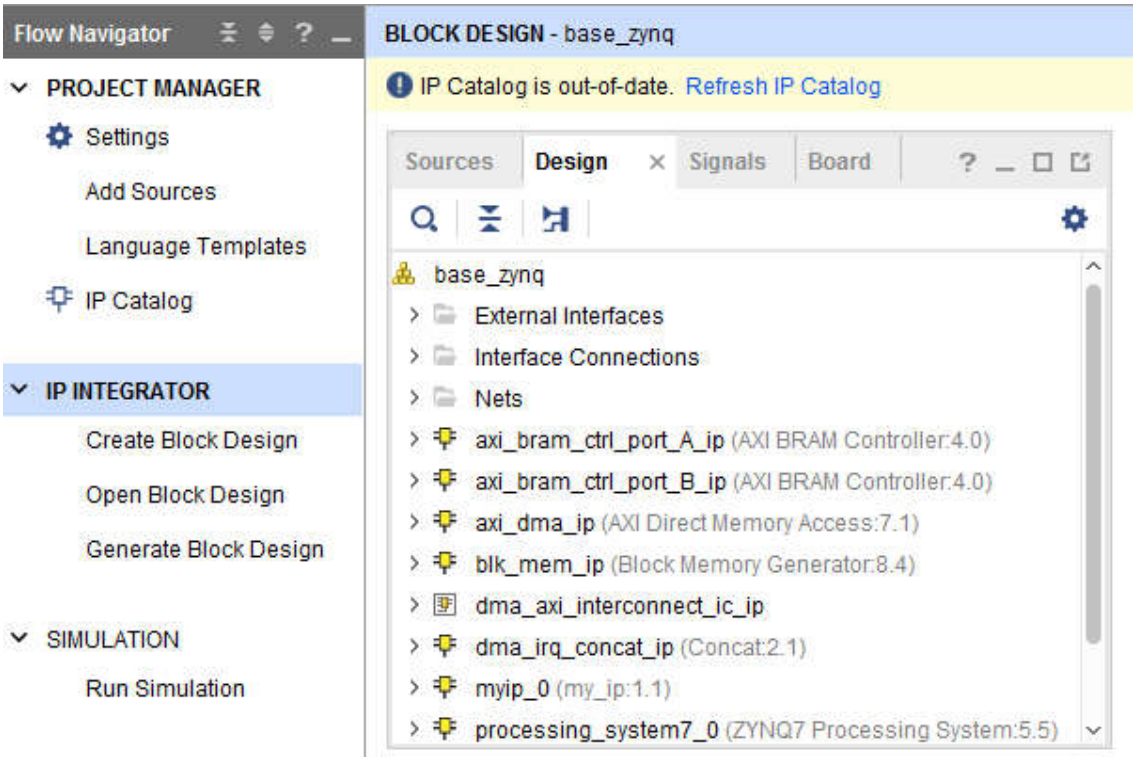
- Γράψιμο στη FIFO:
    - Το γράψιμο στη FIFO θα γίνεται από το slave interface, κάθε φορά που έχουμε ένα valid AXI4 stream transaction, δηλαδή:
      If (S_AXIS_tvalid==1 and S_AXIS_tready==1) then
      FIFO[i] <= S_AXIS _tdata; , όπου i η 1η κενή θέση στη FIFO.
- Διάβασμα από τη FIFO:
    - Το διάβασμα της πιο «παλιάς» λέξης που έχει γραφτεί στη FIFO, θα γίνεται κάθε φορά που το DMAe είναι έτοιμο να διαβάσει μια valid λέξη, δηλαδή:
      If (M_AXIS_tvalid==1 and M_AXIS_tready==1) then
      M_AXIS _tdata<=FIFO[0];

Επεξεργασία της λογικής

Για να επεξεργαστείτε τη λογική, κάνετε τα εξής βήματα:

1. Ανοίγετε το project με το 2017.4
2. Flow navigator → IP catalog → Στα δεξιά που ανοίγει το IP catalog παράθυρο → User repository → AXI peripheral → my ip → δεξί κλικ → Edit in IP packager. **Σημείωση:** Αν δεν υπάρχει το IP, σημαίνει πως δεν επιλέξατε το SoC → Zynq 7000 devices στο installation. Για να το προσθέσετε θα κάνετε μέσα από το project του Vivado: Help -> Add design tools or devices. Βάλτε user name / password, και στην επόμενη οθόνη επιλέξτε Upgrade installation to Vivado HL Design Edition. Μετά επιλέξτε Devices -> Production Devices -> SoCs -> Zynq 7000.
3. Θα ανοίξει ένα καινούριο instance του Vivado, όπου εκεί θα κάνετε τις αλλαγές του VHDL κώδικα.

4. Στο νέο project που ανοίγει → Sources → Design Sources → myip_v1_1.vhd κάνετε κλικ για να ανοίξει ο κώδικας του top level module. Κάνοντας expand το myip_v1_1.vhd θα δείτε και τα άλλα δυο components που υλοποιούν τα AXI Stream master και slave interfaces.
5. Όταν κάνετε αλλαγές στον κώδικα, κάνετε μετά save. Κατόπιν επιλέγετε Project Manager → Package IP. Εμφανίζεται μια νέα καρτέλα στα δεξιά "Package IP – myip" μαζί με Packaging steps.
6. Στο identification αλλάξτε το version, ώστε να είστε σίγουροι πως μετά στο simulation χρησιμοποιείτε το updated IP.
7. Στο Review and Package step πατήστε Re-Package IP. Στο αρχικό project που είναι για όλο το σύστημα, θα εμφανιστεί μήνυμα IP catalog is out-of-date:



8. Πατήστε Refresh IP Catalog και μετά στο κάτω μέρος → Upgrade selected. Στο παράθυρο "Generate Output Products" πατήστε Skip.
9. Αν έχετε ήδη ανοιχτό το simulation, θα πρέπει να το ξεκινήσετε από την αρχή.

Προσομοίωση του συστήματος

Για να κάνετε προσομοίωση του συστήματος, ακολουθήστε τα παρακάτω βήματα:

1. Ανοίγετε το project με το 2017.4
2. IP INTEGRATOR → Open Block Design
3. SIMULATION → Run Simulation → Run Behavioral Simulation
4. Όταν πλέον ανοίξει το simulation, κλείστε την Untitled κυματομορφή (αν υπάρχει) κα ανοίξτε την «tb_behav.wcfg» ως εξής: File → Open Waveform Configuration → tb_behav.wcfg
5. Αριστερά της κυματομορφής υπάρχουν τα Obejcts και δίπλα σε αυτά 2 tabs, Scope και Resources. Επιλέξτε τα Resources → Simulation Sources → sim_1 → zynq_tb.v για να ανοίξετε το testbench.
6. Πατήστε το "Run all" κουμπί για να τρέξετε την προσομοίωση και μόλις τελειώσει μπορείτε να συνεχίσετε την προσομοίωση πατώντας το "Run for 1000 ns" κουμπί, ώστε να συνεχίζετε την προσομοίωση για 1000 nsec κάθε φορά. Μπορείτε να δείτε τα σήματα στην κυματομορφή. Η προσομοίωση τελειώνει λίγο πριν τα 8000 nsec.

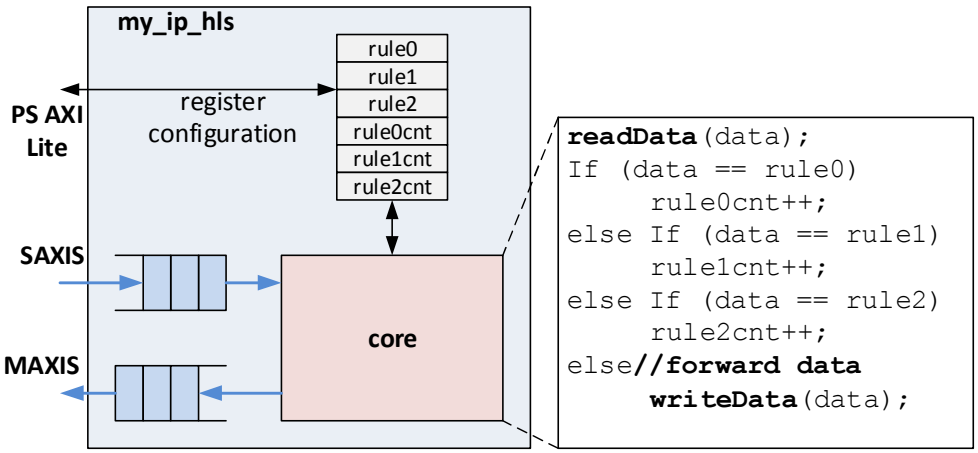
Η παρούσα λογική αυτό που κάνει είναι αρχικά να στέλνει κάποιες λέξεις στο DMAe μέσω του slave AXI4 Stream interface. Μόλις το DMAe προγραμματιστεί για να δεχθεί τις λέξεις αυτές (από το benchmark), τις γράφει στην BRAM. Από την άλλη, μόλις το DMAe προγραμματιστεί (από το benchmark) για να διαβάσει κάποιες λέξεις από τη BRAM, θα τις δείτε να εισέρχονται στη λογική από το master AXI4 Stream interface.

Παραδοτέα 1ου milestone

1. Αναφορά που περιλαμβάνει σχηματική αναπαράσταση των παράλληλων modules που έχετε χρησιμοποιήσει στον κώδικά σας. Θα πρέπει να φαίνεται με βέλη ο τρόπος αλληλεπίδρασης μεταξύ των κομματιών του κώδικα που έχετε γράψει, σε επίπεδο process.
2. Κυματομορφή που δείχνει τη σωστή λειτουργία της ουράς. Για την προσομοίωση, χρησιμοποιήστε το behavioral simulation που είναι ήδη έτοιμο στο project που έχετε κατεβάσει.
3. Πηγαίος κώδικας (όχι όλο το project).

2ο Milestone

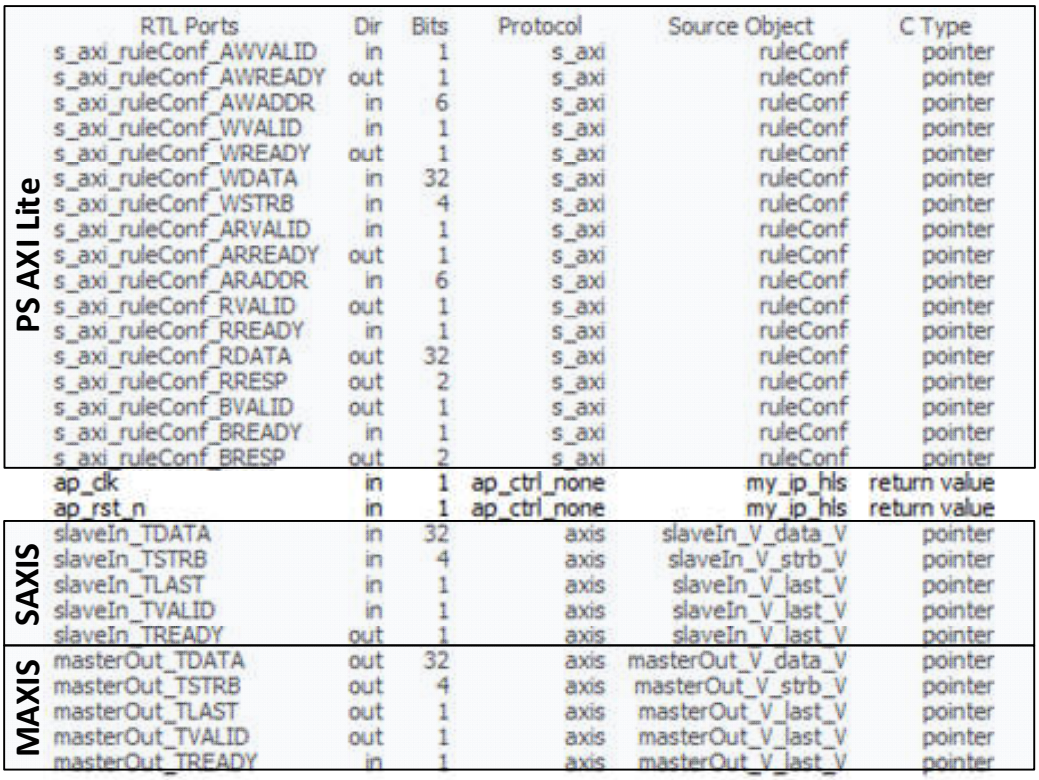
Στο 2ο milestone θα ασχοληθούμε με την υλοποίηση ενός συστήματος που «φιλτράρει» μια ροή δεδομένων σε σχέση με προκαθορισμένους κανόνες.



Εικόνα 4 – Το σύστημα που θα φιλτράρει μια ροή δεδομένων σε σχέση με προκαθορισμένους κανόνες.

Η Εικόνα 4 δείχνει τη δομή του συστήματος αυτού, το οποίο αποτελείται από ένα σύνολο από registers, δύο ουρές και το core module που ελέγχει τη ροή δεδομένων. Πιο συγκεκριμένα:

- Οι registers είναι 6 και μπορούν να γραφτούν/διαβαστούν από την επεξεργαστή μέσω ενός AXI Lite interface. Οι 3 πρώτοι έχουν την τιμή του κάθε κανόνα, και οι υπόλοιποι χρησιμοποιούνται ως μετρητές που αναφέρουν πόσες φορές δεδομένα απορρίφθηκαν, επειδή ταυτίζονταν με κάποιο κανόνα. Για παράδειγμα, αν rule0=10, rule1=30, rule2=50, και η ροή δεδομένων είναι 10, 20, 30, 40, 50, 60, 70, 50, τότε στο τέλος θα πρέπει οι μετρητές να είναι rule0cnt=1, rule1cnt=1, rule2cnt=2.
- Οι δύο ουρές χρησιμοποιούνται για προσωρινή αποθήκευση των ροών δεδομένων στην είσοδο και την έξοδο, ακολουθώντας το πρωτόκολλο AXI Stream.
- Το core module διαβάζει δεδομένα από την ουρά εισόδου, ελέγχει αν κάποιο από αυτά ταυτίζεται με οποιοδήποτε από τους υπάρχοντες κανόνες, και είτε ανανεώνει τον αντίστοιχο μετρητή, είτε προωθεί τα δεδομένα στην έξοδο. Για το παραπάνω παράδειγμα όπου η είσοδος ήταν 10, 20, 30, 40, 50, 60, 70, 50, η έξοδος θα είναι 20, 40, 60, 70.



	RTL Ports	Dir	Bits	Protocol	Source Object	C Type
PS AXI Lite	s_axi_ruleConf_AWVALID	in	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_AWREADY	out	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_AWADDR	in	6	s_axi	ruleConf	pointer
	s_axi_ruleConf_WVALID	in	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_WREADY	out	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_WDATA	in	32	s_axi	ruleConf	pointer
	s_axi_ruleConf_WSTRB	in	4	s_axi	ruleConf	pointer
	s_axi_ruleConf_ARVALID	in	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_ARREADY	out	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_ARADDR	in	6	s_axi	ruleConf	pointer
	s_axi_ruleConf_RVALID	out	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_RREADY	in	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_RDATA	out	32	s_axi	ruleConf	pointer
	s_axi_ruleConf_RRESP	out	2	s_axi	ruleConf	pointer
	s_axi_ruleConf_BVALID	out	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_BREADY	in	1	s_axi	ruleConf	pointer
	s_axi_ruleConf_BRESP	out	2	s_axi	ruleConf	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
SAXIS	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
MAXIS	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

Εικόνα 5 – Είσοδοι και έξοδοι του συστήματος.

Η Εικόνα 5 παρέχει το σύνολο των εισόδων και εξόδων του συστήματος. Το PS AXI Lite interface χρησιμοποιείται για διάβασμα και το γράψιμο των registers, ενώ τα SAXIS / MAXIS interfaces παρέχουν την είσοδο και την έξοδο των ροών δεδομένων αντίστοιχα, με βάση το πρωτόκολλο AXI Stream.

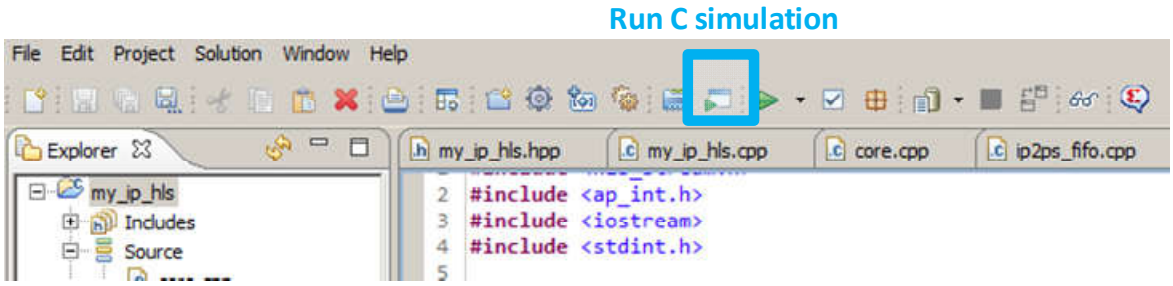
Επεξεργασία της λογικής

Για την υλοποίηση του συστήματος, θα χρησιμοποιήσουμε το Vivado HLS 2017.4. Στο courses.ece.tuc.gr θα βρείτε ένα project που έχει ήδη έτοιμα ορισμένα αρχεία για να ξεκινήσετε άμεσα την υλοποίηση:

- my_ip_hls.hpp: είναι το header file του project, στο οποίο κάνουμε include βιβλιοθήκες, καθώς επίσης δηλώνουμε και τις συναρτήσεις που έχουμε.
- my_ip_hls.cpp: είναι το top-level αρχείο του project, στο οποίο δηλώνουμε τις εισόδους / εξόδους, τα interfaces που θέλουμε (πχ AXI Lite, AXI Stream), καθώς και καλούμε τις επιμέρους συναρτήσεις που χρειάζονται για την επεξεργασία των δεδομένων.
- core.cpp: είναι το module στο οποίο θα γίνεται ο έλεγχος των δεδομένων σε σχέση με τους κανόνες που έχουμε ορίσει.
- ps2ip_fifo.cpp: είναι η ουρά για τα δεδομένα στην είσοδο.
- ip2ps _fifo.cpp: είναι η ουρά για τα δεδομένα στην έξοδο.
- my_ip_hls_tb.cpp: είναι το αρχείο που περιγράφει το testbench.

Σκοπός είναι να γίνουν οι απαραίτητες προσθήκες, ώστε το σύστημα να υποστηρίζει τους καταχωρητές που φαίνονται στην Εικόνα 4, καθώς και την επεξεργασία των δεδομένων με βάση τους κανόνες (λεπτομέρειες θα δοθούν και στο tutorial).

Προσομοίωση του συστήματος



Εικόνα 6 - Προσομοίωση συστήματος με C.

Για την προσομοίωση του συστήματος, ανοίγετε το my_ip_hls_tb.cpp, και όπως φαίνεται στην Εικόνα 6, επιλέγετε Run C simulation. Η κονσόλα θα δείξει το αποτέλεσμα, όπου στην προκειμένη περίπτωση όλα τα δεδομένα εισόδου, προωθούνται απευθείας στην έξοδο.

Παραδοτέα 2ου milestone

1. Αναφορά που περιλαμβάνει σχηματική αναπαράσταση των modules που έχετε χρησιμοποιήσει στον κώδικά σας. Θα πρέπει να φαίνεται με βέλη ο τρόπος αλληλεπίδρασης μεταξύ των κομματιών του κώδικα που έχετε γράψει.
2. Πηγαίος κώδικας (όχι όλο το project).

3ο Milestone

Αναμένεται σύντομα η περιγραφή.

Χρήσιμες αναφορές / links

Στο courses.ece.tuc.gr μπορείτε να βρείτε χρήσιμα manuals / datasheets σχετικά με την εργασία.