



ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ενσωμάτωση Επεξεργαστή MicroBlaze σε FPGA για Διαχείριση Εφαρμογών

```
puts("Enabling MicroBlaze caches.....");
microblaze_invalidate_icache();
microblaze_enable_icache();

//microblaze_invalidate_dcache();
microblaze_enable_dcache();
puts("Done");

//=====
//      Instantiate Device Drivers
//=====

//Initialize interrupt controller driver
puts("Initializing Interrupt Controller....");
status = XIntc_Initialize(&interrupt_controller, XPAR_INTC_0_DEVICE_ID);
if(status != XST_SUCCESS) {
    puts("Failed");
    return XST_FAILURE;
}
puts("Passed");

//Initialize cfifo driver
puts("Initializing Interrupt Controller....");
status = CFifo_Initialize(&cfifo, XPAR_CFIPO_0_DEVICE_ID);
if(status != XST_SUCCESS) {
    puts("Failed");
    return XST_FAILURE;
}
puts("Passed");

//=====
//      Setup System Interrupts
//=====

puts("Set-up System Interrupts");

//Set options to service all interrupts
puts("Set Interrupt Controller to service all interrupts");
status = XIntc_SetOptions(&interrupt_controller, XIN_SVC_ALL_ISRS_OPTION);
if (status != XST_SUCCESS) {
    puts("Failed");
    return XST_FAILURE;
}
puts("Passed");
```



MicroBlaze

ΠΑΝΑΓΙΩΤΗΣ ΣΚΟΡΔΥΛΑΚΗΣ

ΕΠΙΒΛΕΠΩΝ : ΔΙΟΝΥΣΙΟΣ ΡΕΪΣΗΣ, ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ ΠΑΝΕΠΙΣΤΗΜΙΟΥ
ΑΘΗΝΩΝ

ΑΘΗΝΑ, 2013

Περίομενα

1. Εισαγωγή	1
2 MicroBlaze	2
2.1 Εισαγωγή	2
2.2 Endianness	2
2.3 Instructions.....	3
2.5 Registers.....	3
2.6 Pipeline	4
2.7 Memory Architecture	6
2.8 Virtual-Memory Management.....	6
2.9 Reset, Interrupts and Exceptions	6
2.9.1 Reset	7
2.9.2 Hardware Exceptions.....	7
2.9.4 Interrupt	8
2.10 Interfaces	9
3. AXI Interconnect.....	10
3.1 Εισαγωγή	10
3.2 Use Models	10
3.2.1 Pass Through	11
3.2.2 Conversion Only	11
3.2.3 N-to-1 Interconnect.....	12
3.2.4 1-to-N Interconnect	12
3.2.5 N-to-M Interconnect(Crossbar Mode)	13
3.2.6 M-to-N Interconnect(Shared Access Mode).....	14
3.3 Functionality	15
3.3.1 Μετατροπή Μήκους Αρτηρίας(Width Conversion)	15
3.3.2 Μετατροπή Ρολογιού (Clock Conversion).....	15
3.3.3 Τομή Καταχωρητών Περιφερειακών (Peripheral Register Slices).....	15
3.3.4 Ουρές Datapath (Datapath FIFOs)	15

4. AXI Interrupt Controller.....	16
4.1 Εισαγωγή.....	16
4.2 Interrupt Capture Modes	17
4.3 Interrupt Vector Register	17
4.4 Programmer Registers	17
4.4.1 Interrupt Status Register (ISR).....	17
4.4.2 Interrupt Enable Register (IER).....	17
4.4.3 Interrupt Pending Register (IPR).....	18
4.4.4 Interrupt Acknowledge Register (IAR)	18
4.4.5 Set Interrupt Enables (SIE)	18
4.4.6 Clear Interrupt Enables (CIE).....	18
4.4.7 Interrupt Vector Register (IVR)	18
4.4.8 Master Enable Register (MER)	19
4.5 Interrupt Service Flow	19
5.Παρεχόμενα Περιφεριακά	20
5.1 Εισαγωγή	20
5.2 Bus & Bridges	20
5.3 Clocks, Reset & Interrupt	20
5.4 Communication	21
5.5 DMA & Timer	21
5.6 Debug	22
5.7 GPIO	22
5.8 Memory & Memory Controller	23
6 Custom Peripherals.....	25
6.1 Εισαγωγή.....	25
6.2 AXI IPIF.....	25
6.2.1 AXI IPIF Parameters	26
6.2.2 Πίνακες καθορισμού περιοχών διευθύνσεων	26
6.2.3 C_ARD_ADDR_RANGE_ARRAY	26
6.2.4 C_ARD_NUM_CE_ARRAY	27
6.2.5 C_S_AXI_ADDR_WIDTH & C_S_AXI_DATA_WIDTH.....	27

6.2.6 C_S_AXI_MIN_SIZE	27
2.2.7 C_USE_WSTRB	27
6.2.8 C_DPHASE_TIMEOUT	27
6.3 AXI IPIC	28
6.4 Interrupt Control	28
6.5 Κυκλική Ουρά - Υλοποίηση Custom Περιφερειακού	31
6.5.1 Μνήμη	31
6.5.2 Δείκτες - Pointers	31
6.5.3 User Logic.....	33
6.5.4 Top Module Custom Περιφερειακού	34
6.5.5 Έλεγχος λειτουργίας	37
7. Standalone Software	40
7.1 XParameters.h	40
7.2 C Standard Library	40
7.3 Προγράμματα Οδήγησης Περιφερειακών	40
7.4 Πρώτη Εφαρμογή	41
7.4.1 platform.h/c	42
7.4.2 interrupts.h/c	44
7.4.3 Main.c	44
7.5 Προγράμματα Οδήγησης Custom Περιφερειακών	45
7.5.1 cfifo.h	45
7.5.2 cfifo.c	46
7.5.3 cfifo_sinit.c	47
7.5.4 cfifo_intr.c	47
7.5.5 cfifo_l.h	48
7.6 Δοκιμαστική εφαρμογή προγράμματος οδήγησης ενός custom περιφερειακού ...	49
Παράρτημα A : Κώδικας	52
A.1 Σύστημα.....	52
A.1.1 MHS	52
A.1.2 MSS	58
A.1.3 UCF	60

A.2 Αρχεία Custom Περιφερειακού (VHDL)	61
A.2.1 CFIFO.VHDL	61
A.2.2 USER_LOGIC.VHDL	69
A.2.2 POINTER.VHDL	73
A.2.3 RAM.VHDL	74
A.3 Αρχεία Custom Περιφερειακού (Drivers).....	75
A.3.1 cfifo.h.....	75
A.3.2 cfifo.c	77
A.3.3 cfifo_l.h	79
A.3.4 cfifo_i.h	82
A.3.5 cfifo_g.c	83
A.3.6 cfifo_sinit.c.....	84
A.3.7 cfifo_intr.c	86
A.3.8 cfifo_selftest.c.....	90
Παράρτημα Β : Εργαλεία Ανάπτυξης και Υλοποίησης.....	92
B.1 Integrated Software Environment (ISE)	92
B.2 Embedded Development Kit (EDK)	92
B.3 Software Development Kit (SDK)	93

1. Εισαγωγή

Τα FPGAs είναι ολοκληρωμένα κυκλώματα τα οποία μπορούν να παραμετροποιηθούν ώστε να υλοποιούν ψηφιακά κυκλώματα μετά την παρασκευή τους από τον εκάστοτε σχεδιαστή. Με την εξέλιξη της τεχνολογίας τα ολοκληρωμένα είναι αρκετά πυκνά ώστε να υποστηρίξουν ακόμα και ολόκληρα συστήματα(System On Chip) για πληθώρα εφαρμογών σε διάφορους τομείς όπως των τηλεπικοινωνιών ή του αυτομάτου ελέγχου. Στις περισσότερες των περιπτώσεων σε ένα τέτοιο σύστημα απαιτείται η ύπαρξη ενός επεξεργαστή για τον έλεγχο των λειτουργιών του συστήματος και την ανταπόκριση του σε αλλαγές των εξωτερικών συνθηκών.

Οι πρώτες υλοποιήσεις επεξεργαστών σε FPGA ήταν hard processors, δηλαδή ο επεξεργαστής ήταν υλοποιημένος στο ολοκληρωμένο και ο σχεδιαστής απλά έπρεπε κατά την διάρκεια του σχεδιασμού να ενεργοποιήσει και να συνδέσει το υπόλοιπο σύστημα στον επεξεργαστή ώστε να εκμεταλευνεί τις δυνατότητες του.

Πλέον υπάρχουν αρκετές υλοποιήσεις επεξεργαστών οι οποίοι υλοποιούνται στα LUTs(look up tables), soft processors, του ολοκληρωμένου δύνοντας έτσι το πλεονέκτημα χρήσης όλου του ολοκληρωμένου από συστήματα που δεν χρειάζονται έναν επεξεργαστή αλλά κυρίως δύνοντας την δυνατότητα στο σχεδιαστή που θα χρησιμοποιήσει έναν soft processor επεξεργαστή να παραμετροποιήσει και να συμπεριλάβει μόνο τα στοιχεία του επεξεργαστή τα οποία είναι απαραίτητα.

Στην συνέχεια της διπλωματικής αυτής θα περιγραφεί ο soft processor Microblaze, που προσφέρεται από την Xilinx για υλοποίηση στα ολοκληρωμένα κυκλώματα της. Θα αναφερθούμε στα απαραίτητα περιφερειακά που πρέπει να περιέχει ένα σύστημα με τον συγκεκριμένο επεξεργαστή, καθώς επίσης και το τρόπο με τον οποίο ο σχεδιαστής μπορεί εύκολα να επεκτείνει το σύστημα αυτό υλοποιώντας καινούρια περιφερειακά κατάλληλα ώστε να επικοινωνούν με τον επεξεργαστή.

Τέλος θα γίνει αναφορά στην υλοποίηση εφαρμογών οι οποίες θα εκτελούνται στο επεξεργαστή κάνοντας χρήση των παρεχόμενων βιβλιοθηκών, προγράμματα οδήγησης για τα περιφερειακά του επεξεργαστή αλλά και την δημιουργία προγραμμάτων οδήγησης για περιφερειακά που έχουν υλοποιηθεί από το σχεδιαστή για συγκεκριμένο σκοπό.

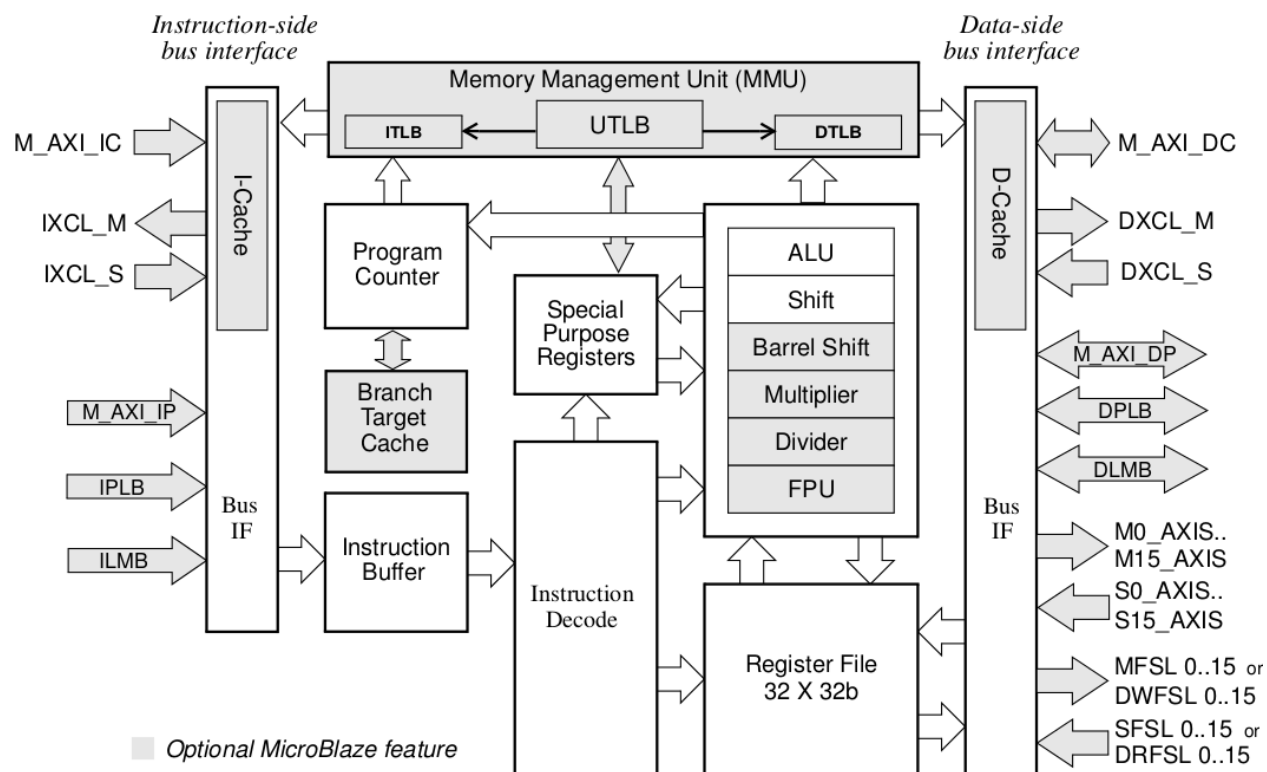
2 MicroBlaze

2.1 Εισαγωγή

Ο Microblaze είναι processor ο οποίος προσφέρεται από την Xilinx για υλοποίηση σε FPGAs. Μερικά από τα βασικά χαρακτηριστικά του είναι :

- Soft Core, υλοποιείται στα luts του fpga
- Παραμετροποιήσιμος σε μεγάλο βαθμό
- 32-bit Architecture
- RISC (Reduced Instruction Set Computer)
- 3 ή 5 στάδια pipeline

Στην εικόνα 2.1 φαίνεται το block διάγραμμα του Microblaze. Τα στοιχεία που είναι σκιαγραφημένα με γκρι είναι προαιρετικά και μπορούν να ενεργοποιηθούν παραμετροποιώντας κατάλληλα τον processor.



Εικόνα 2.1 : Block Διάγραμμα MicroBlaze

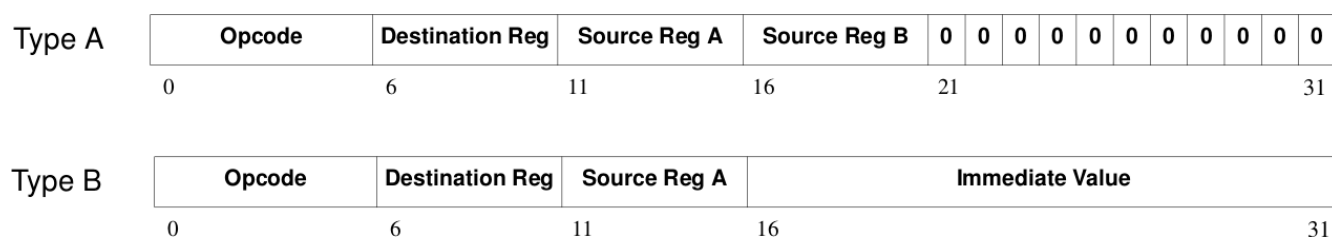
2.2 Endianness

Ο Microblaze υποστηρίζει Big-Endian αλλά και Little-Endian. Ανάλογα με το bus το οποίο θα χρησιμοποιήσουμε για να συνδέσουμε το processor με τα περιφερειακά, θα

πρέπει να χρησιμοποιήσουμε και την κατάλληλη bytes order. Η Xilinx προσφέρει δύο υλοποιήσεις: το PLB, υλοποίηση του CoreConnect της IBM και το AXI, υλοποίηση του AMBA που χρησιμοποιείται στους ARM επεξεργαστές. Εάν επιλέξουμε το PLB, θα πρέπει να χρησιμοποιήσουμε Big-Endianess, ενώ εάν επιλέξουμε το AXI, θα πρέπει να χρησιμοποιήσουμε Little-Endianess. Η επιλογή γίνεται παραμετροποιώντας τον processor και συγκεκριμένα θέτοντας την παράμετρο C_ENDIANNESS.

2.3 Instructions

Οι εντολές του Microblaze είναι όλες 32-bit οι οποίες χωρίζονται σε δύο τύπους, Type A και Type B. Οι type A εντολές δέχονται μέχρι δύο source registers και έναν destination register. Αντίθετα οι εντολές Type B δέχονται έναν source register και έναν 16-bit τελεστή. Το format και των δύο τύπων εντολών φαίνεται στην εικόνα 2.2



Εικόνα 2.2 : Τύποι εντολών

2.5 Registers

Όσο αναφορά τους registers, ο Microblaze έχει 32 registers γενικού σκοπού των 32-bit, καθώς και μέχρι 18 registers συγκεκριμένου σκοπού επίσης των 32-bit, ανάλογα με τις τιμές των παραμέτρων.

Στον παρακάτω πίνακα ακολουθούν οι 18 registers συγκεκριμένου σκοπού με την χρήση τους.

Πίνακας 1 : Καταχωρητές ειδικού σκοπού

Καταχωρητές	Χρήση	Παράμετρος
Program Counter(PC)	Περιέχει την διεύθυνση της επόμενης εντολής που θα εκτελεστεί	-
Machine Status Register(MSR)	Περιέχει flags για έλεγχο της κατάστασης του processor	-
Exception Address Register(EAR)	Περιέχει την διεύθυνση προσπέλασης της εντολής που προκάλεσε το exception	-
Exception Status Register(ESR)	Περιέχει flags σχετικά με exceptions που δημιουργούνται, όπως ο λόγος.	-
Brach Target Register(BTR)	Περιέχει την διεύθυνση	Εάν χρησιμοποιούνται

	διακλάδωσης στις εντολές delay slot που ακολουθούν εντολή διακλάδωσης.	εξαιρέσεις
Floating Point Status Register(FSR)	Περιέχει flags σχετικά με την μονάδα κινητής υποδιαστολής	C_USE_FPU
Exception Data Register(EDR)	Περιέχει τα δεδομένα τα οποία προκάλεσαν εξαίρεση σε συνδέσεις ροών(FSL ή AXIStream)	C_FSL_LINKS, C_FSL_EXCEPTION
Stack Low Register(SLR)	Περιέχει την μικρότερη διεύθυνση της stack για τον εντοπισμό υπερχείλισης	C_USE_STACK_PROTECTION
Stack High Register(SHR)	Περιέχει την μεγαλύτερη διεύθυνση της stack για τον εντοπισμό υποχείλισης	C_USE_STACK_PROTECTION
Process Identifier Register(PID)	Χρησιμοποιείται για να αναγνωρίσει μια διεργασία κατά την λειτουργία της MMU	C_USE_MMU
Zone Protection Register(ZPR)	Χρησιμοποιείται για την παράκαμψη της προστασίας της μνήμης που ορίζεται στις TLB εγγραφές της MMU	C_USE_MMU
Translation Look-Aside Buffer Low Register(TLBLO)	Χρησιμοποιείται για την πρόσβαση σε UTLB εγγραφές της MMU	C_USE_MMU, C_AREA_OPTIMIZED
Translation Look-Aside Buffer High Register(TLBHI)	Χρησιμοποιείται για την πρόσβαση σε UTLB εγγραφές της MMU	C_USE_MMU, C_AREA_OPTIMIZED
Translation Look-Aside Buffer Index Register(TLBX)	Χρησιμοποιείται ως δείκτης στον UTLB όταν χρησιμοποιούνται οι registers TLBLO και TLBHI	C_USE_MMU, C_AREA_OPTIMIZED
Translation Look-Aside Buffer Search Index Register(TLBSX)	Χρησιμοποιείται κατά την αναζήτηση ενός αριθμού εικονικής σελίδας στον UTLB	C_USE_MMU, C_AREA_OPTIMIZED
Processor Version Register(PVR)	Μια σειρά από registers που περιέχουν στοιχεία για τις λειτουργίες που υλοποιεί ο processor	C_PVR

2.6 Pipeline

Οι εντολές του MicroBlaze είναι pipelined. Ο χρόνος εκτέλεσης των εντολών ενός σταδίου του pipeline είναι ένας κύκλος ρολογιού. Έτσι ο συνολικός χρόνος εκτέλεσης των περισσοτέρων εντολών είναι ίσος με τα στάδια του pipeline και κάθε εντολή ολοκληρώνετε σε κάθε ένα κύκλο ρολογιού. Μερικές εντολές χρειάζονται περισσότερους κύκλους ρολογιού για να ολοκληρωθούν στο στάδιο εκτέλεσης, στις συγκεκριμένες εντολές γίνεται καθυστέρηση του pipeline.

Τα στάδια του pipeline μπορούν να παραμετροποιηθούν σε περίπτωση που χρειάζεται να μειώσουμε το κόστος σε area του processor. Έτσι μπορούμε να ορίσουμε ένα pipeline τριών σταδίων ή πέντε σταδίων. Το pipeline τριών σταδίων περιέχει τα στάδια :

- Fetch (IF)
- Decode (OD) και
- Execute (EX)

Στην εικόνα 2.3 φαίνεται η εκτέλεση εντολών με τρία στάδια pipeline, με την δεύτερη εντολή να χρειάζεται τρεις κύκλους στο στάδιο της εκτέλεσης, οπότε η εκτέλεση της τρίτης εντολής καθυστερείται για δύο κύκλους.

	cycle1	cycle2	cycle3	cycle4	cycle5	cycle6	cycle7
instruction 1	Fetch	Decode	Execute				
instruction 2		Fetch	Decode	Execute	Execute	Execute	
instruction 3			Fetch	Decode	Stall	Stall	Execute

Εικόνα 2.3 : Εκτέλεση εντολών με τρία στάδια pipeline

Εάν παραμετροποιήσουμε τον MicroBlaze με πέντε στάδια pipeline τα στάδια του pipeline είναι :

- Fetch (IF)
- Decode (OD)
- Execute (EX)
- Access Memory (MEM) και
- Write Back (WB)

Στην εικόνα 2.4 φαίνεται η εκτέλεση εντολών με πέντε στάδια pipeline, με την δεύτερη εντολή να χρειάζεται τρεις κύκλους στο στάδιο της πρόσβασης στην μνήμη, οπότε το αντίστοιχο στάδιο της τρίτης εντολής καθυστερείται για δύο κύκλους.

	cycle1	cycle2	cycle3	cycle4	cycle5	cycle6	cycle7	cycle8	cycle9
instruction 1	IF	OF	EX	MEM	WB				
instruction 2		IF	OF	EX	MEM	MEM	MEM	WB	
instruction 3			IF	OF	EX	Stall	Stall	MEM	WB

Εικόνα 2.4 : Εκτέλεση εντολών με πέντε στάδια pipeline

Ο καθορισμός των σταδίων του pipeline καθορίζεται από την παράμετρο C_AREA_OPTIMIZED, η τιμή 1 καθορίζει pipeline τριών σταδίων ενώ η τιμή 0 καθορίζει pipeline πέντε σταδίων.

2.7 Memory Architecture

Ο Microblaze έχει σχεδιαστεί με βάση το μοντέλο αρχιτεκτονικής Harvard, δηλαδή χρησιμοποιείται διαφορετικό interface για τις εντολές και διαφορετικό για τα data. Οι περιοχές μνήμης, των δύο interface μπορούν να επικαλύπτονται και έτσι να βρίσκονται στο ίδιο φυσικό μέσο αποθήκευσης.

Ο επεξεργαστής δεν διαχειρίζεται διαφορετικά την πρόσβαση εισόδου/εξόδου από την πρόσβαση στην μνήμη, δηλαδή χρησιμοποιείται memory mapped I/O. Για την πρόσβαση στην μνήμη μπορούμε να έχουμε μέχρι τρία διαφορετικά interfaces :

- Local Memory Bus (LMB)
- Advanced eXtensible Interface (AXI4) ή Processor (PLB)
- Advanced eXtensible Interface (AXI4) ή Xilinx CacheLink (XCL)

2.8 Virtual-Memory Management

Ο MicroBlaze μπορεί να παραμετροποιηθεί έτσι ώστε να υποστηρίζει Virtual Memory. Όταν ενεργοποιηθεί η λειτουργία εσωτερικά του processor προστίθεται μια Memory Management Unit (MMU). Η λειτουργία της MMU είναι να μεταφράζει τις ενεργές διευθύνσεις σε πραγματικές. Με αυτόν το τρόπο ένα σύστημα μπορεί να μεταφέρει δυναμικά τα προγράμματα που είναι φορτωμένα στην μνήμη σε διαφορετική θέση. Επίσης δεν χρειάζεται ο προγραμματιστής να λαμβάνει υπόψιν ποιες φυσικές διευθύνσεις μνήμης αποδίδονται στις υπόλοιπες διεργασίες.

2.9 Reset, Interrupts and Exceptions

Ο Microblaze μπορεί να σχεδιαστεί έτσι ώστε να ανταποκρίνεται σε ορισμένα events. Τα events αυτά είναι reset, interrupts, exceptions και break. Η προτεραιότητα εκτέλεσης των αντίστοιχων callbacks, από την υψηλότερη προς την χαμηλότερη, είναι :

1. Reset
2. Hardware Exception
3. Non-maskable Break
4. Break
5. Interrupt
6. User Vector (Exception)

Για την ανταπόκριση του επεξεργαστή στα events, αυτές οι προτεραιότητες είναι καθορισμένες σε συγκεκριμένες θέσεις μνήμης με σειρά διευθύνσεων(vector) στις οποίες κάνει jump. Οι διευθύνσεις φαίνονται στον παρακάτω πίνακα μαζί με το event, με το οποίο σχετίζονται.

Πίνακας 2 : Jump vectors

Event	Jump Address Vector
Reset	0x00000000 – 0x00000004
User Vector(Exception)	0x00000008 – 0x0000000C
Interrupt	0x00000010 – 0x00000014
Break : Non-maskable hardware	
Break : Hardware	0x00000018 – 0x0000001C
Break : Software	
Hardware Exception	0x00000020 – 0x00000024

2.9.1 Reset

Στην περίπτωση που γίνει reset στον επεξεργαστή, είτε hardware είτε software, τότε αδειάζει το pipeline από τις εντολές τις οποίες εκτελεί την συγκεκριμένη στιγμή και η εκτέλεση μεταφέρεται στο reset vector, στην θέση 0x00000000 από όπου διαβάζεται η επόμενη εντολή που θα εκτελεστεί.

2.9.2 Hardware Exceptions

Ο Microblaze μπορεί να ρυθμιστεί έτσι ώστε να αντιλαμβάνεται και να ενεργεί κατάλληλα όταν συμβαίνουν συγκεκριμένα σφάλματα. Ορισμένα από τα σφάλματα αυτά ενεργοποιούνται όταν ενεργοποιηθούν συγκεκριμένες μονάδες οι οποίες μπορούν να προκαλέσουν τα σφάλματα αυτά.

Αναλυτικότερα τα σφάλματα αυτά είναι :

- Μη έγκυρη εντολή (illegal instruction)
- Σφάλμα εντολής (instruction error)
- Σφάλμα διαύλου (bus error)
- Ανάγνωση μη ευθυγραμμισμένων δεδομένων(unaligned access)
- Σφάλμα διαίρεσης(divide exception), μόνο στην περίπτωση που ο επεξεργαστής έχει ρυθμιστεί να περιέχει διαιρέτη σε hardware.
- Σφάλματα κινητής υποδιαστολής, μόνο στην περίπτωση που στον επεξεργαστή περιλαμβάνεται Floating Point Unit(FPU)
 - underflow
 - overflow
 - float division-by-zero
 - invalid operation
 - denormalized operand error

- Σφάλματα Μονάδας Διαχείρισης Μνήμης(MMU) :
 - Illegal Instruction Exception
 - Data Storage Exception
 - Instruction Storage Exception
 - Data TLB Miss Exception
 - Instruction TLB Miss Exception

Όταν δημιουργηθεί κάποιο σφάλμα η εκτέλεση των εντολών σταματάει και αδειάζει το pipeline του επεξεργαστή, στην συνέχεια η εκτέλεση μεταφέρεται στο hardware exception vector(0x00000020).

Στην περίπτωση που το error προέρχεται από την MMU, εκτός του illegal instruction exception, η εντολή που προκάλεσε το σφάλμα εκτελείται ξανά μετά την εκτέλεση της συνάρτησης που διαχειρίζεται το σφάλμα. Σε όλες τις άλλες περιπτώσεις, η εκτέλεση μεταφέρεται στην επόμενη εντολή.

Όταν δημιουργηθούν παραπάνω από ένα σφάλματα ταυτόχρονα, ο επεξεργαστής τα διαχειρίζεται με την παρακάτω σειρά :

1. Instruction Bus Exception
2. Instruction TLB Miss Exception
3. Instruction Storage Exception
4. Illegal Opcode Exception
5. Privileged Instruction Exception or Stack Protection Violation Exception
6. Data TLB Miss Exception
7. Data Storage Exception
8. Unaligned Exception
9. Data Bus Exception
10. Divide Exception
11. FPU Exception
12. Stream Exception

2.9.4 Interrupt

Ο Microblaze υποστηρίζει μόνο ένα εξωτερικό interrupt. Στην περίπτωση που είναι ενεργοποιημένα τα interrupts, το bit Interrupt Enable(IE) στον Machine Status Register(1) έχει την τιμή 1 και εάν ενεργοποιηθεί η είσοδος του interrupt, τότε η εκτέλεση μεταφέρεται στο interrupt vector στην διεύθυνση(0x00000010). Ο επεξεργαστής μπορεί να ρυθμιστεί να αναγνωρίζει level ή edge interrupt. Χρησιμοποιώντας εξωτερικό περιφερειακό interrupt controller, μπορούν να υποστηριχθούν παραπάνω από ένα interrupts. Ο interrupt controller θα αναλυθεί σε επόμενο κεφάλαιο.

2.10 Interfaces

Ο MicroBlaze μας δίνει την δυνατότητα χρήσης αρκετών διαφορετικών interfaces μέσω του οποίου θα επικοινωνεί με την μνήμη ή κάποιο περιφερειακό. Τα interfaces από τα οποία μπορούμε να επιλέξουμε είναι :

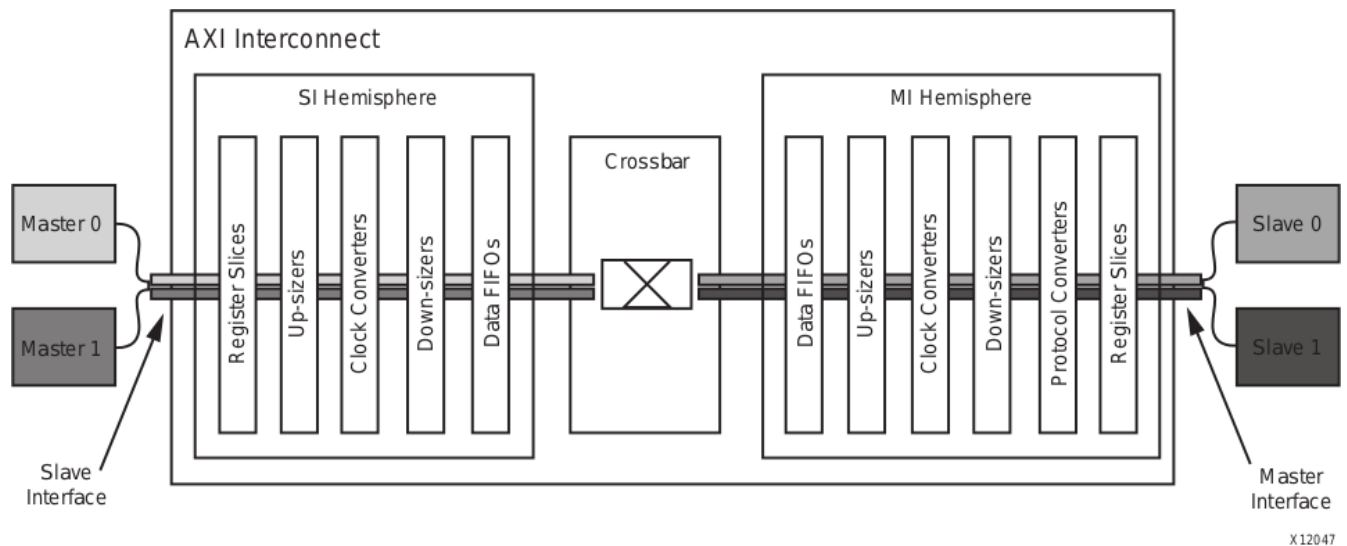
- AXI4 ή AXI4-Lite
- Processor Local Bus(PLB)
- Local Memory Bus(LMB)
- Fast Simplex Link(FSL)
- Xilinx CacheLink(XCL)

Όπως έχει αναφερθεί παραπάνω τα AXI και το PLB interfaces χρησιμοποιούνται για την σύνδεση στα αντίστοιχα interconnects (buses). Το LMB χρησιμοποιείται για την επικοινωνία με BRAM, block μνήμης εσωτερικά του FPGA. Το FSL όπως και το AXIStream χρησιμοποιούνται για point-to-point επικοινωνία με κάποιο περιφερειακό. Τέλος το Xilinx CacheLink κάνει χρήση δύο FSL, προσφέροντας μια γρήγορη λύση για την πρόσβαση σε εξωτερική μνήμη.

3. AXI Interconnect

3.1 Εισαγωγή

Το AXI interconnect χρησιμοποιείται για την επικοινωνία μεταξύ μιας ή περισσότερων memory mapped master συσκευών, με μια ή περισσότερες memory mapped slave συσκευές. Τα AXI interfaces είναι υλοποίηση της προδιαγραφής AMBA AXI 4 από την ARM.



Εικόνα 3.1 : Block διάγραμμα AXI Interconnect

Το block διάγραμμα του core φαίνεται στην εικόνα 3.1. Το core αποτελείται από τρία μέρη, το slave interface(SI), το master interface(MI) και το crossbar. Τα master devices συνδέονται στο SI, μέσω του οποίου μπορούν να δώσουν εντολές για ανάγνωση και εγγραφή, αντίστοιχα οι slave devices συνδέονται στο MI interface.

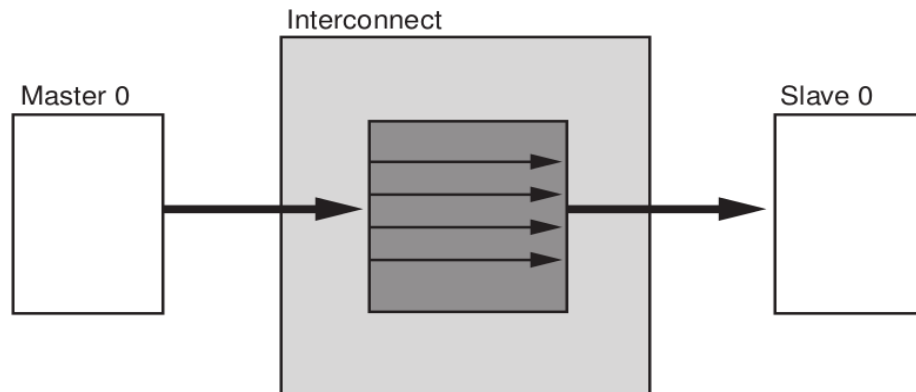
3.2 Use Models

Η υλοποίηση στο FPGA διαφέρει ανάλογα με την χρήση που έχει οριστεί, του AXI Interconnect (δηλαδή τον αριθμό των masters και slaves που είναι συνδεδεμένοι), καθώς και των μετατροπών ή του pipeline . Η υλοποίηση που θα γίνει μπορεί να είναι μια από τις παρακάτω :

- Pass Through
- Conversion Only
- N-to-1 Interconnect
- 1-toN Interconnect
- N-toM Interconnect (Crossbar Mode)
- M-to-N Interconnec (Shared Access Mode)

3.2.1 Pass Through

Στην περίπτωση που είναι συνδεδεμένοι μόνο ένας master και ένας slave και το interconnect δεν κάνει κάποια άλλη λειτουργία μετατροπής ή pipeline, τότε η υλοποίηση μετατρέπεται σε απευθείας σύνδεση των συσκευών, χωρίς να καταναλώνονται πόροι και χωρίς να δημιουργείται κάποια καθυστέρηση.

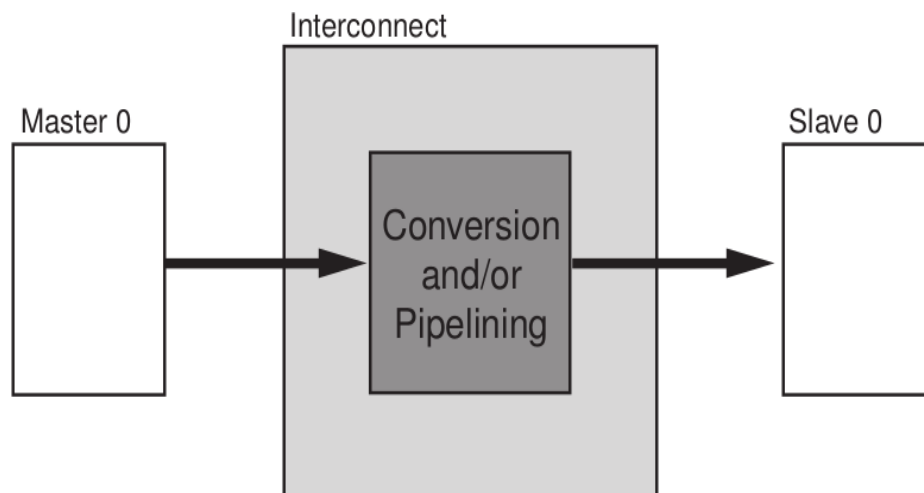


Εικόνα 3.2 : Pass Through υλοποίηση

X12048

3.2.2 Conversion Only

Στην περίπτωση που γίνεται κάποια μετατροπή όπως :



Εικόνα 3.3 : Conversion only υλοποίηση

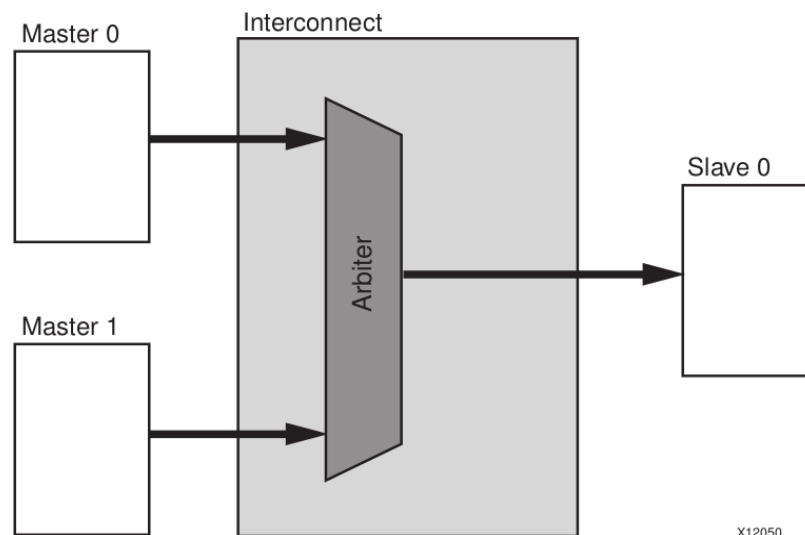
X12049

- Data width conversion
- Clock rate conversion
- AXI4-Lite slave adaption
- Pipelining

η υλοποίηση δεν περιέχει κυκλώματα διαιτησίας (arbitration), αποκωδικοποίησης ή δρομολόγησης.

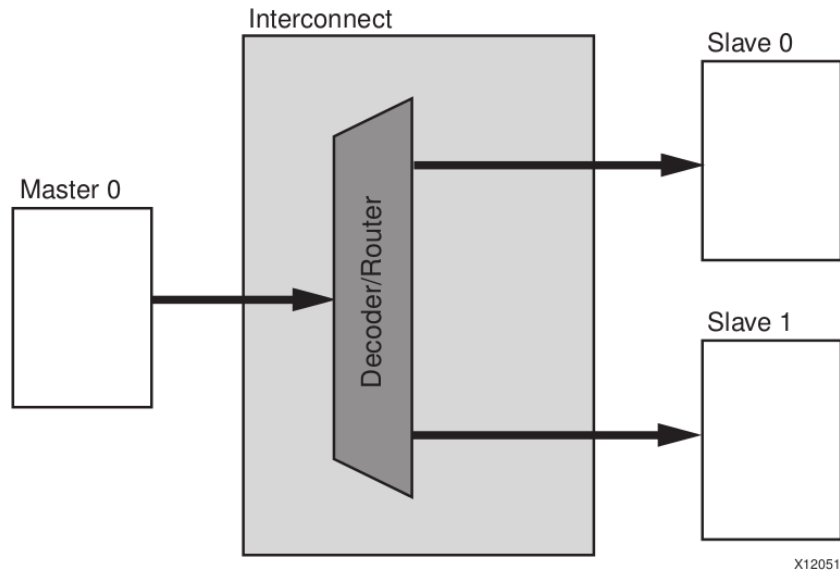
3.2.3 N-to-1 Interconnect

Στην περίπτωση που έχουμε πολλαπλές master συσκευές, αλλά μία μόνο slave συσκευή, η υλοποίηση περιέχει κύκλωμα διαιτησίας (arbiter) για τον καθορισμό της συσκευής που θα αποκτήσει πρόσβαση στη slave συσκευή. Οποιαδήποτε προαιρετική μετατροπή, υποστηρίζεται σε αυτό το mode.



3.2.4 1-to-N Interconnect

Μια περίπτωση που είναι αρκετά συχνή, είναι να έχουμε έναν master με πολλές slave συσκευές. Ένα παράδειγμα που είναι και η περίπτωση μας, είναι να έχουμε έναν processor που προσπελαύνει ένα αριθμό από memory mapped devices. Σε αυτήν την περίπτωση δεν χρειαζόμαστε κυκλώματα διαιτησίας, εφόσον έχουμε έναν master, αλλά κυκλώματα αποκωδικοποίησης και δρομολόγησης για τις διευθύνσεις και τα δεδομένα.

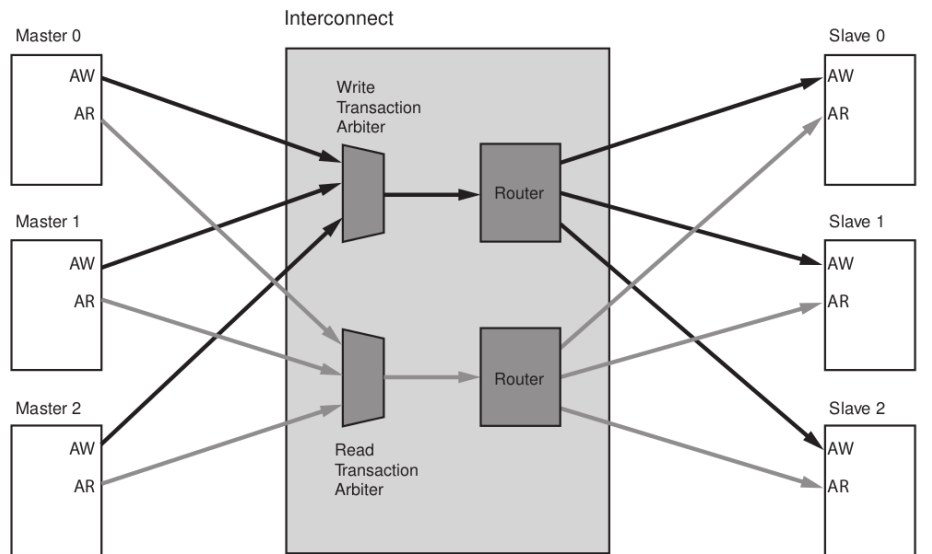


Εικόνα 3.5 : 1-το-N υλοποίηση

X12051

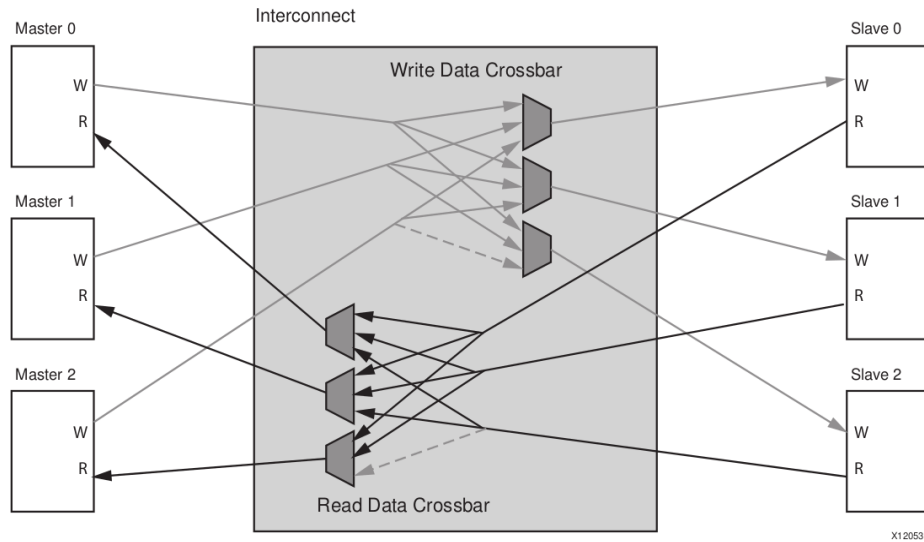
3.2.5 N-to-M Interconnect(Crossbar Mode)

Στην περίπτωση που έχουμε πολλαπλούς masters, slaves, σε crossbar mode, το interconnect έχει ένα διαμοιραζόμενο κύκλωμα διαιτησίας (arbiter) για εγγραφή και ανάγνωση (εικόνα 3.6) καθώς και πολλαπλές διαδρομές για τα data, ανάγνωσης και εγγραφής (εικόνα 3.7)



Εικόνα 3.6 : N-to-M υλοποίηση, διαιτησία εντολών εγγραφής και ανάγνωσης

X12052

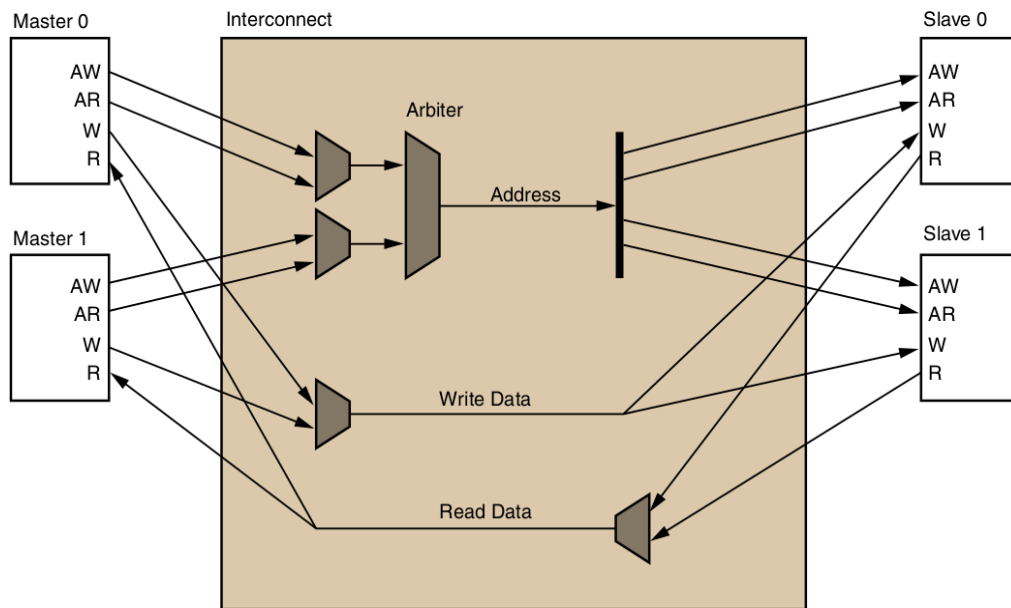


Εικόνα 3.7 : N-to-M υλοποίηση data path

Με αυτόν τον τρόπο δίνεται η δυνατότητα να πραγματοποιούνται ταυτόχρονα transactions από διαφορετικούς masters σε διαφορετικούς slaves.

3.2.6 M-to-N Interconnect(Shared Access Mode)

Η τελευταία περίπτωση είναι η να υλοποιηθεί το interconnect σε shared access mode (εικόνα 3.8). Στην περίπτωση αυτή σε κάθε χρονική στιγμή μπορεί να πραγματοποιείται μόνο ένα transaction. Το πλεονέκτημα σε σχέση με την crossbar mode είναι ότι η υλοποίηση αυτή καταναλώνει λιγότερα resources.



Εικόνα 3.8 : M-to-N υλοποίηση

3.3 Functionality

Το AXI Interconnect core μπορεί να παραμετροποιηθεί ώστε να πραγματοποιεί ορισμένες λειτουργίες μετατροπής κατά την διάρκεια μιας συναλλαγής. Ορισμένες από τις λειτουργίες που θα δούμε παρακάτω είναι :

- Μετατροπή Μήκους Αρτηρίας (Width Conversion)
- Μετατροπή Ρολογιού (Clock Conversion)
- Εισαγωγή τομής καταχωρητών
- Ουρές Datapath (Datapath FIFOs)

3.3.1 Μετατροπή Μήκους Αρτηρίας(Width Conversion)

Το AXI Interconnect έχει εσωτερικά παραμετροποιήσιμο μέγεθος αρτηρίας. Στην περίπτωση που οποιοσδήποτε slave ή master χρησιμοποιεί διαφορετικό μέγεθος αρτηρίας θα πρέπει να γίνει μετατροπή στο μήκος των data, είτε αύξηση είτε μείωση του μήκους.

3.3.2 Μετατροπή Ρολογιού (Clock Conversion)

Μια άλλη μετατροπή που μπορεί να κάνει το axi interconnect είναι η μετατροπή στα ρολόγια των devices (slave & master). Η μετατροπή μπορεί να είναι, είτε μείωση είτε αύξηση κατά ένα λόγο. Και στις δύο περιπτώσεις γίνεται μέσω ακεραίας πράξης, διαίρεση για την μείωση και πολλαπλασιασμό για την αύξηση κατά ένα παράγοντα N.

3.3.3 Τομή Καταχωρητών Περιφερειακών (Peripheral Register Slices)

Προαιρετικά μπορούν να προστεθούν δύο registers σε κάθε SI ή MI υποδοχής, καθώς επίσης και ένας buffer εισόδου-εξόδου αντίστοιχα. Και στις δύο περιπτώσεις ο λόγος είναι η επίτευξη υψηλότερης τιμής ρολογιού, με την εισαγωγή όμως καθυστέρησης ενός κύκλου ρολογιού.

3.3.4 Ουρές Datapath (Datapath FIFOs)

Στις περιπτώσεις που χρειάζεται να γίνει κάποιου είδους μετατροπής, μήκους αρτηρίας ή/και ρυθμού ρολογιού, η ενδιάμεση αποθήκευση (buffering) μπορεί να βελτιώσει την απόδοση του axi interconnect όσο αναφορά την μετάδοση των δεδομένων.

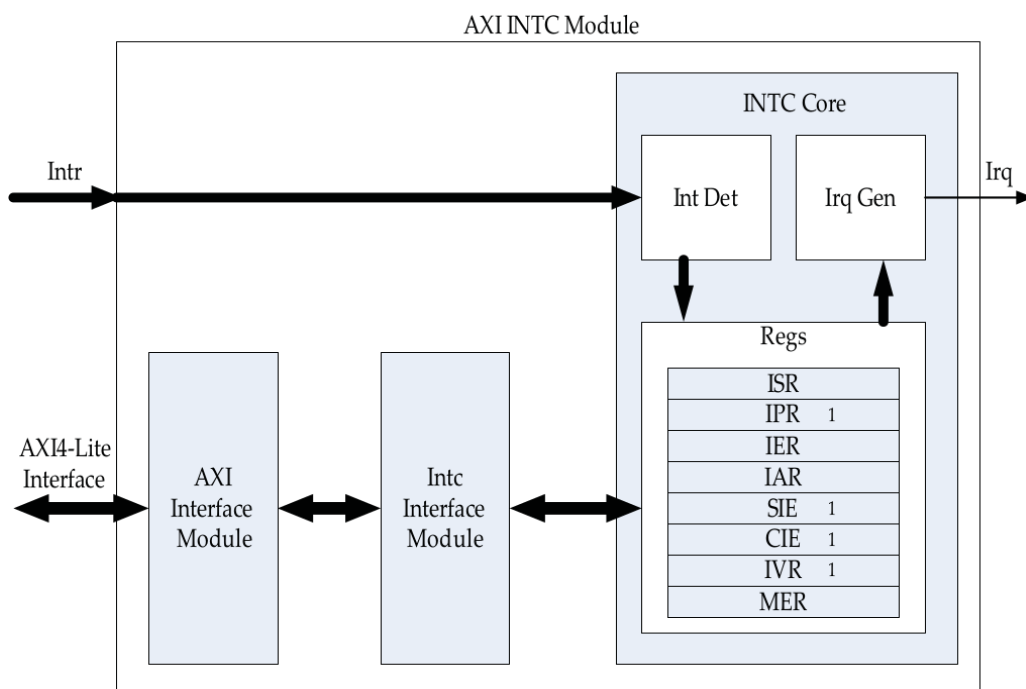
4. AXI Interrupt Controller

4.1 Εισαγωγή

Ο Microblaze έχει μόνο μια είσοδο για interrupt, η υποστήριξη περισσότερων interrupts γίνεται μέσω του interrupt controller. Ο interrupt controller δέχεται πολλαπλά interrupt σήματα από τα περιφερειακά που τα ενώνει σε ένα interrupt σήμα, το οποίο οδηγείται στον επεξεργαστή, επίσης ο interrupt controller μπορεί να παρέχει προαιρετικά, έναν τρόπο προτεραιότητας υπηρετήσης των interrupt signals, τα οποία δέχεται στις εισόδους του.

Οι registers, εσωτερικά του interrupt controller, που χρησιμοποιούνται για την υλοποίηση των interrupt vectors, καθώς επίσης και για την ενεργοποίηση και επιβεβαίωση των interrupts προσπελαύνονται μέσω του axi interface, το οποίο χρησιμοποιείται για την επικοινωνία όλων των περιφερειακών με τον επεξεργαστή.

Ο interrupt controller περιέχει τους κατάλληλους καταχωρητές, οι οποίοι είναι προσβάσιμοι προγραμματιστικά, για την λειτουργία αυτή καθώς και το κατάλληλο interface για την επικοινωνία με τον επεξεργαστή. Στην εικόνα 4.1 φαίνεται το block διάγραμμα του interrupt controller.



1. Registers are OPTIONAL

Εικόνα 4.1 : Block διάγραμμα Interrupt controller

4.2 Interrupt Capture Modes

Ο interrupt controller μπορεί να παραμετροποιηθεί να αναγνωρίζει τα interrupt signals με δύο τρόπους :

- Edge-Sensitive capture mode
- Level-Sensitive capture mode

Στην περίπτωση που η αναγνώριση είναι ακμοπυροδοτούμενη (edge-triggered), μια interrupt συνθήκη αναγνωρίζεται σε μια ενεργή ακμή, ακόμη και όταν δεν υπάρχει κάποια άλλη αναγνωρισμένη interrupt συνθήκη. Η φορά (αύξουσα ή φθίνουσα) της ενεργής ακμής μπορεί να καθοριστεί για κάθε είσοδο ξεχωριστά.

Αντίστοιχα, στην περίπτωση που χρησιμοποιείται το level sensitive capture mode, γίνεται και η αναγνώριση interrupt.

4.3 Interrupt Vector Register

Εάν ο interrupt controller παραμετροποιηθεί ώστε να υλοποιεί τον interrupt vector register, τότε εφαρμόζεται μια ιεραρχία όσο αναφορά την προτεραιότητα των interrupts. Τα interrupts τα οποία προέρχονται από τις μικρότερες εισόδους έχουν μεγαλύτερη προτεραιότητα εξυπηρέτησης.

4.4 Programmer Registers

Ο interrupt controller περιέχει δέκα καταχωρητές, οι οποίοι είναι προσβάσιμοι προγραμματιστικά και χρησιμοποιούνται για την ενεργοποίηση, αναγνώριση και εκκαθάριση των interrupts. Στην συνέχεια ακολουθεί μια αναλυτική περιγραφή των καταχωρητών αυτών και του σκοπού που εξυπηρετούν.

4.4.1 Interrupt Status Register (ISR)

Το περιεχόμενο του συγκεκριμένου καταχωρητή δείχνει εάν υπάρχει κάποιο interrupt, το οποίο πρέπει να εξυπηρετηθεί. Κάθε bit του καταχωρητή αντιστοιχεί σε μία είσοδο του interrupt controller. Η τιμή 1 σε κάποιο από τα bits αυτά, δείχνει την ύπαρξη ενός interrupt στην αντίστοιχη είσοδο. Αντίθετα η τιμή 0 σε κάποιο από τα bits του καταχωρητή, δείχνει την μη ύπαρξη ενός interrupt στην αντίστοιχη είσοδο.

4.4.2 Interrupt Enable Register (IER)

Το κάθε bit του IER χρησιμοποιείται για την ενεργοποίηση ή απενεργοποίηση (masking) του interrupt της αντίστοιχης εισόδου του interrupt controller. Η απενεργοποίηση ενός interrupt (0 στο αντίστοιχο bit) δεν αποτρέπει την αναγνώρισή τους, αλλά μόνο την παραγωγή του IRQ από τον interrupt controller.

4.4.3 Interrupt Pending Register (IPR)

Αντίστοιχα με τον Interrupt Status Register, το κάθε bit του καταχωρητή αντιστοιχείται με μια από τις εισόδους του interrupt controller. Η τιμή του κάθε bit δείχνει την παρουσία ή απουσία (με 1 ή 0 αντίστοιχα) ενός interrupt που είναι ενεργοποιημένο.

Ο IPR χρησιμοποιείται για να μειώσει τις προσβάσεις στον interrupt controller κατά μία, για κάθε interrupt, ενώ η πρόσβαση γίνεται μέσω του bus. Το κάθε bit του καταχωρητή είναι το αποτέλεσμα της πράξης AND των bits των καταχωρητών ISR και IER στις αντίστοιχες θέσεις.

Ο IPR είναι προαιρετικός και αν θα υλοποιηθεί καθορίζεται από την τιμή της παραμέτρου C_HAS_IPR.

4.4.4 Interrupt Acknowledge Register (IAR)

Ο IAR χρησιμοποιείται για την επιβεβαίωση των interrupts. Η επιβεβαίωση ενός interrupt γίνεται γράφοντας την τιμή 1 στο bit που αντιστοιχεί στο εκάστοτε interrupt. Η εγγραφή της τιμής 1 έχει σαν επακόλουθο τον μηδενισμό του αντίστοιχου bit στον καταχωρητή ISR και στον ίδιο τον IAR.

4.4.5 Set Interrupt Enables (SIE)

Ο Set Interrupt Enable χρησιμοποιείται ώστε αν ενεργοποιήσει συγκεκριμένα interrupts με μία μόνο συναλλαγή (transaction), ορίζοντας στο εκάστοτε bit την τιμή 1, έχει σαν αποτέλεσμα να πάρει την ίδια τιμή το αντίστοιχο bit του IER. Αντίθετα αν οριστεί η τιμή 0, τότε η τιμή που έχει το αντίστοιχο bit στον IER παραμένει η ίδια.

Ο SIE είναι προαιρετικός και αν θα υλοποιηθεί καθορίζεται από την τιμή της παραμέτρου C_HAS_SIE.

4.4.6 Clear Interrupt Enables (CIE)

Ο Clear Interrupt Enable έχει την αντίθετη (συμπληρωματική) χρήση από ότι ο Set Interrupt Enable. Γράφοντας την τιμή 1 σε κάποιο bit, έχει σαν αποτέλεσμα την εκχώρηση της τιμής 0 στο αντίστοιχο bit του IER. Αντίθετα γράφοντας την τιμή 0, η τιμή του αντίστοιχου bit παραμένει ίδια.

Ο CIE είναι προαιρετικός και αν θα υλοποιηθεί καθορίζεται από την τιμή της παραμέτρου C_HAS_CIE.

4.4.7 Interrupt Vector Register (IVR)

Ο IVR περιέχει τον αριθμό της εισόδου που είναι ενεργοποιημένη και έχει την μεγαλύτερη προτεραιότητα. Σε περίπτωση που δεν υπάρχει κάποιο ενεργό interrupt, ο

IVR περιέχει την μέγιστη τιμή, δηλαδή όλα τα bits είναι 1. Ο IVR είναι προαιρετικός καταχωρητής και η υλοποίηση του εξαρτάται από την τιμή της παραμέτρου C_HAS_IVR.

4.4.8 Master Enable Register (MER)

Ο τελευταίος καταχωρητής του interrupt controller είναι ο Master Enable Register. Ο συγκεκριμένος καταχωρητής περιέχει μόνο δύο bits, που είναι τα δύο LSB της διεύθυνσης του καταχωρητή.

4.5 Interrupt Service Flow

Όπως αναφέρθηκε ο Microblaze έχει μια μόνο είσοδο για interrupt, καθώς επίσης και μια μόνο συγκεκριμένη διεύθυνση στην οποία μεταφέρεται η εκτέλεση όταν γίνει assert η είσοδος αυτή. Σε περίπτωση που υποστηρίζονται περισσότερα του ενός interrupts μέσω του interrupt controller, ο επεξεργαστής θα πρέπει να αναγνωρίσει ποιο περιφερειακό έχει ενεργοποιήσει το interrupt. Λόγω της ανάγκης αυτής δημιουργείται αρκετή καθυστέρηση στην εξυπηρέτηση ενός interrupt, όπου σε αρκετές περιπτώσεις δεν είναι ανεκτή.

Για τον παραπάνω λόγο, σε νεότερες εκδόσεις του interrupt controller έχει προστεθεί ένα καινούριο mode λειτουργίας, το fast interrupt mode, το οποίο δίνει την δυνατότητα στον επεξεργαστή να κάνει αμέσως jump στην διεύθυνση που βρίσκεται η κατάλληλη service routine για την εξυπηρέτηση του interrupt. Αυτό γίνεται υλοποιώντας περισσότερους καταχωρητές που περιέχουν τις διευθύνσεις των interrupt service routines και παρέχοντας την κατάλληλη διεύθυνση σε ξεχωριστή είσοδό του, παράλληλα με το interrupt σήμα.

5.Παρεχόμενα Περιφερειακά

5.1 Εισαγωγή

Για την υλοποίηση ενός πλήρους συστήματος στο ολοκληρωμένο, η Xilinx παρέχει αρκετά έτοιμα περιφερειακά τα οποία μπορούν να υλοποιηθούν στο FPGA. Τα παρεχόμενα περιφερειακά μπορούν να χωριστούν σε κατηγορίες ανάλογα με την λειτουργικότητα που προσφέρουν. Μερικές από τις κατηγορίες είναι :

- Bus & Bridges
- Clocks, Reset & Interrupts
- Communication
- DMA & Timer
- Debug
- GPIO
- Memory & Memory Controller

5.2 Bus & Bridges

Όπως αναφέρθηκε στην ανάλυση της αρχιτεκτονικής του, ο Microblaze μπορεί να υποστηρίξει ένα πλήθος από αρτηρίες (bus) για την επικοινωνία με τα περιφερειακά. Ένα από τα core που είδαμε και ανήκει στην κατηγορία αυτή, είναι το AXI Interconnect. Ένα άλλο από τα υπόλοιπα cores που υλοποιούν κάποια αρτηρία, το οποίο μπορούμε να το χρησιμοποιήσουμε με τον Microblaze, είναι το Processor Local Bus (PLB), το οποίο είναι υλοποίηση του CoreConnect της IBM. Το PLB ήταν η βασική αρτηρία στα ολοκληρωμένα της Xilinx, αφού χρησιμοποιούταν και από τον επεξεργαστή PowerPC (hard processor) που ήταν υλοποιημένος στο ολοκληρωμένο.

Άλλο ένα ακόμη core της κατηγορίας αυτής, το οποίο χρησιμοποιείται σχεδόν πάντα σε συστήματα με τον επεξεργαστή Microblaze, είναι το Local Memory Bus(LMB). Το core αυτό χρησιμοποιείται για την επικοινωνία του επεξεργαστή με Block Rams (BRAM), οι οποίες είναι υλοποιημένες μέσα στο ολοκληρωμένο.

Εκτός από τα core που υλοποιούν αρτηρίες, υπάρχουν διαθέσιμα και cores, τα οποία υλοποιούν γέφυρες (bridges). Οι γέφυρες χρησιμοποιούνται για διασύνδεση διαφορετικών αρτηριών, έτσι ώστε να είναι δυνατή η χρήση περιφερειακών που είναι σχεδιασμένα ώστε να επικοινωνούν μια μόνο συγκεκριμένη αρτηρία.

5.3 Clocks, Reset & Interrupt

Ένα σύστημα χρειάζεται ένα πλήθος από διαφορετικά ρολόγια για τον ίδιο τον επεξεργαστή, αλλά και τα περιφερειακά που είναι υλοποιημένα μέσα στο ολοκληρωμένο. Για την δημιουργία των κατάλληλων περιόδων που χρειάζονται, χρησιμοποιείται ο clock manager.

Ο clock manager δέχεται στην είσοδο ένα σήμα ρολογιού, το σήμα αυτό χρησιμοποιείται για την λειτουργία του ολοκληρωμένου, όπου κάνοντας τις κατάλληλες διαιρέσεις συχνότητας, παράγει μια σειρά από εξόδους, ως σήματα ρολογιών, με διαφορετικές περιόδους. Χρησιμοποιεί μια σειρά από primitives (αρχαίτυπα) στοιχεία, υλοποιημένα (hard core) στο ολοκληρωμένο, παραμετροποιώντας τα κατάλληλα, ώστε να πετύχει όσο τον δυνατόν πιο κοντινές τιμές στην επιθυμητή περίοδο, αλλά παράλληλα παράγοντας σήματα με όσο το δυνατόν λιγότερο θόρυβο.

Ένα από τα primitives που χρησιμοποιεί ο clock generator είναι το Mixed Mode Clock Module core, το οποίο μπορούμε να το χρησιμοποιήσουμε στο σύστημα στην περίπτωση που θέλουμε να παράγουμε επιπλέον τιμές ρολογιών. Το MMCM core δέχεται επίσης ένα σήμα ρολογιού με βάση το οποίο μπορεί να δημιουργήσει μια σειρά από εξόδους με διαφορετικές τιμές περιόδου. Το MMCM μπορεί να συνδεθεί σειριακά με το clock manager core, αν και στις περισσότερες φορές προτιμάται παράλληλη συνδεσμολογία για να μην δημιουργείται θόρυβος στα σήματα ρολογιού που παράγει.

Στην κατηγορία αυτή επίσης μπορούμε να βρούμε cores τα οποία προσφέρουν λειτουργικότητα που έχει να κάνει με την διαχείριση των interrupts, όπως ο AXI Interrupt Controller.

5.4 Communication

Σε περίπτωση που χρειάζεται να έχουμε κάποιου είδους επικοινωνία με το σύστημα, θα χρειαστούμε cores τα οποία να υλοποιούν κάποιο πρωτόκολλο επικοινωνίας. Το πιο συνηθισμένο πρωτόκολλο επικοινωνίας και το πιο απλό είναι το RS232. Για την επικοινωνία με RS232 χρησιμοποιείται το core Uartlite. Σε περίπτωση που είναι υλοποιημένο στο ολοκληρωμένο και συνδεδεμένο στο σύστημα, μπορεί να επιλεγεί ως η κύρια είσοδος/έξοδος (stdio) του συστήματος.

Ένας άλλος τρόπος επικοινωνίας είναι μέσω ethernet. Για την επικοινωνία μέσω ethernet μπορούμε να χρησιμοποιήσουμε το core axi ethernet lite. Το core αυτό μπορεί να χρησιμοποιηθεί για επικοινωνία 10/100 Mbps. Για αυτήν την επικοινωνία φυσικά το σύστημα θα πρέπει να είναι ρυθμισμένο έτσι ώστε να τρέχει κάποιο service, όπως telnet ή/και ssh, οπότε να μπορεί κάποιος να συνδεθεί μετά την εκκίνηση του.

5.5 DMA & Timer

Το core AXI CDMA προσφέρει Direct Memory Access λειτουργικότητα, μεταξύ δύο memory mapped διευθύνσεων, μιας πηγής και ενός προορισμού, σε ένα ενσωματωμένο σύστημα. Το core έχει δύο interfaces, AXI4 και AXI4-Lite για την σύνδεση στα αντίστοιχα interconnect. Μέσω του AXI4 interconnect πραγματοποιείται η σύνδεση με την μνήμη όπου αυτή συνδέεται, ενώ μέσω του AXI4-Lite επιτυγχάνεται η επικοινωνία με τον επεξεργαστή και τα περιφερειακά.

Προαιρετικά μπορεί να ενεργοποιηθεί η λειτουργία Scatter-Gather, όπου μέσω αυτής μπορούν να προγραμματιστούν και να πραγματοποιηθούν αυτόματα DMA μεταφορές. Ο προγραμματισμός γίνεται στο από πρόγραμμα που εκτελείται στο επίπεδο χρήστη (userland) και πραγματοποιείται χρησιμοποιώντας μια λίστα από προκαθορισμένες εντολές.

Για τα περιφερειακά που χρησιμοποιούν το AXI Stream interface επικοινωνίας, παρέχεται το core AXI DMA Engine με αντίστοιχες δυνατότητες.

Στην κατηγορία αυτή μπορούμε επίσης να βρούμε cores που υλοποιούν timers και που μπορούν να χρησιμοποιηθούν από το σύστημα μας. Έχουμε την δυνατότητα να χρησιμοποιήσουμε δύο cores στο σύστημα μας: το axi timebase watchdog και το axi timer.

Το core axi timebase watchdog παρέχει έναν 32-bit up timer που μπορεί να χρησιμοποιηθεί ως γενικού σκοπού ή ως watchdog timer.

Αντίθετα το core axi timer μπορεί να παραμετροποιηθεί ώστε να περιέχει έναν ή δύο timers, των 32 ή 64 bit ο καθένας, με την δυνατότητα της επιλογής του τρόπου μέτρησης, up ή down.

5.6 Debug

Κατά την διαδικασία της υλοποίησης ενός συστήματος, στις περισσότερες φορές, υπάρχει το στάδιο της αποσφαλμάτωσης. Για την εύκολη/γρήγορη αποσφαλμάτωση των προγραμμάτων που τρέχουν στο σύστημα, καθώς επίσης και περιφερειακών που έχουμε δημιουργήσει οι ίδιοι, μπορεί να χρησιμοποιηθεί το Microblaze Debug Module(MDM). Το MDM είναι ένα core που συνδέεται με τον επεξεργαστή και μέσω του οποίου μπορούμε να ελέγξουμε την κατάσταση στην οποία βρίσκεται μια ορισμένη στιγμή, διαβάζοντας τις τιμές στους καταχωρητές και την μνήμη. Με το ίδιο core επίσης μπορούμε να δοκιμάσουμε ορισμένα test cases καθώς μας δίνει την δυνατότητα να γράψουμε στους καταχωρητές και την μνήμη. Η ανάγνωση και η εγγραφή της μνήμης μας δίνει επίσης την δυνατότητα να ελέγξουμε την λειτουργικότητα ενός περιφερειακού στην φάση της υλοποίησης, καθώς όπως έχουμε αναφέρει όλα τα περιφερειακά είναι memory mapped οπότε δεν χρειάζεται κάποια περαιτέρω ενέργεια για την ανάγνωση ή την εγγραφή σε αυτά.

5.7 GPIO

Ένα σύστημα θα πρέπει να συνδέεται με εξωτερικές συσκευές. Αυτό στις περισσότερες φορές επιτυγχάνεται χρησιμοποιώντας συγκεκριμένου τύπου core ως controller για την εξωτερική συσκευή, όπως το AXI ethernet lite ή κάποιος memory controller. Σε απλές περιπτώσεις, όπου θέλουμε να ελέγξουμε απλά τις τιμές σε ορισμένα pins του ολοκληρωμένου, μπορεί να χρησιμοποιηθεί το core AXI GPIO (General Purpose I/O).

Το core επικοινωνεί μέσω του interface AXI4-lite, και παρέχει ένα ή δύο κανάλια για επικοινωνία, ενώ μπορεί να παραμετροποιηθεί το μήκος των data από 1 έως 32-bit.

5.8 Memory & Memory Controller

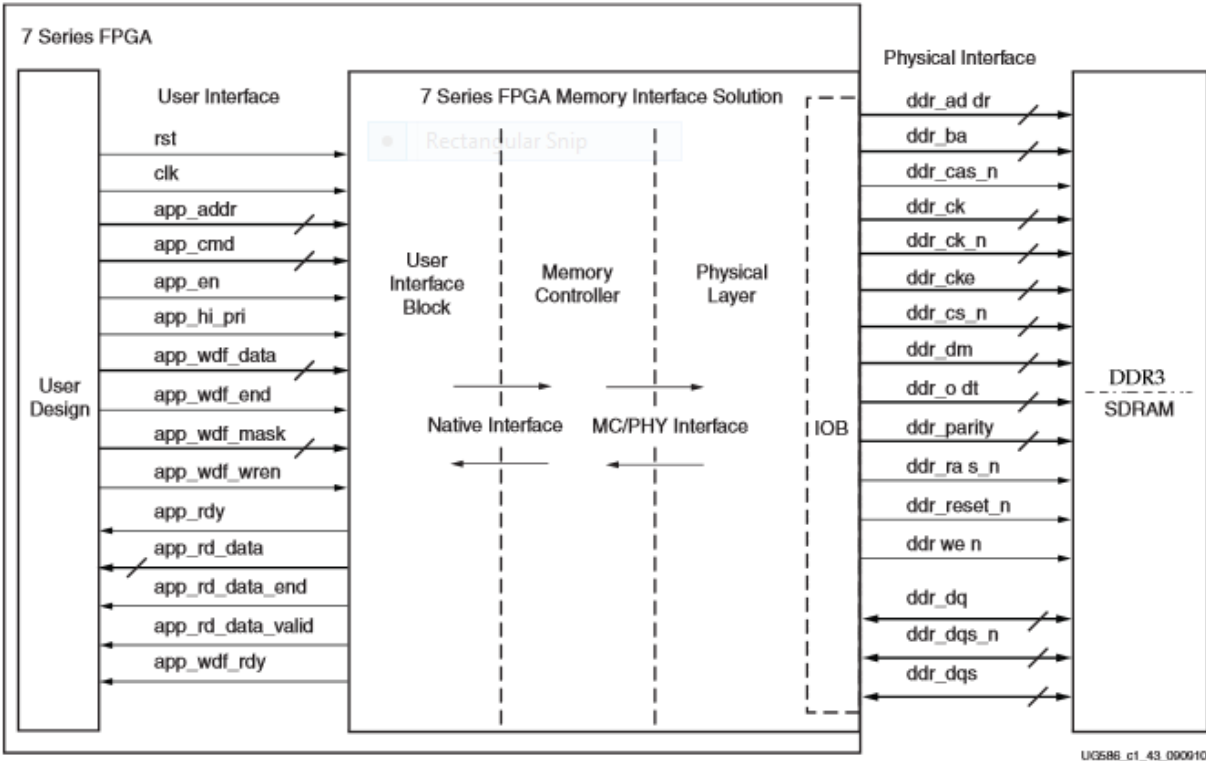
Στην τελευταία κατηγορία μπορούμε να βρούμε cores που έχουν να κάνουν με την μνήμη. Τέτοια cores είναι η υλοποίηση μνήμης εσωτερικά του FPGA(Block RAM), καθώς και controllers για την επικοινωνία με αυτή. Επίσης υπάρχουν controllers για την επικοινωνία με εξωτερική μνήμη.

Εάν πρέπει να χρησιμοποιήσουμε μνήμη εσωτερικά του ολοκληρωμένου, για παράδειγμα cache μνήμη, θα χρησιμοποιήσουμε το core bram το οποίο δεσμεύει block ram primitives για το σύστημα μας. Εκτός από την περίπτωση της cache μπορούμε να χρησιμοποιήσουμε επιπλέον μνήμη εσωτερικά του ολοκληρωμένου, η οποία θα γίνει memory mapped στο εύρος της μνήμης που μπορεί να προσπελάσει ο επεξεργαστής, ώστε να πετύχουμε καλύτερους χρόνους απόκρισης.

Η επικοινωνία με την bram μπορεί αν γίνει, είτε με το core LMB BRAM Controller, είτε με το core AXI BRAM Controller. Όπως έχουμε αναφέρει σε προηγούμενη ενότητα, το LMB (Local Memory Bus) είναι ιδικά σχεδιασμένο για την γρήγορη πρόσβαση σε block rams primitives. Το AXI BRAM Controller core υποστηρίζει σύνδεση με το AXI4 ή το AXI4-lite interconnect. Επίσης μπορεί να συνδεθεί απευθείας με κάποιο master device.

Για την επικοινωνία με εξωτερική μνήμη ram, θα πρέπει να χρησιμοποιήσουμε ειδικό core που να υλοποιεί τον controller της μνήμης. Το core που θα πρέπει να χρησιμοποιήσουμε εξαρτάται από την οικογένεια του ολοκληρωμένου με το οποίο θα υλοποιήσουμε το σύστημά μας.

Τα δύο interface του controller προς το υλοποιημένο κύκλωμα του ολοκληρωμένου, αλλά και προς την μνήμη φαίνονται στην εικόνα 5.1



Εικόνα 5.1 : Block διάγραμμα memory controller

Ο controller μπορεί να παραμετροποιηθεί σε μεγάλο βαθμό, ώστε να μπορεί να υποστηρίξει ένα μεγάλο εύρος από διαφορετικά memory modules, αλλά να δώσει και μια ελευθερία στο σχεδιαστή. Στις περιπτώσεις όπως αυτή που εξετάζουμε, στην οποία χρησιμοποιείται το AXI Bus, ο memory controller υλοποιείται έτσι ώστε να συμπεριλαμβάνει το AXI interface.

Μια από τις ευκολίες που παρέχει το core, είναι η επιλογή των memory banks του module που θα ελέγχει. Με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε παραπάνω του ενός controller, οι οποίοι θα ελέγχουν διαφορετικές banks της μνήμης και θα μπορούν να προσπελαθούν από διαφορετικά στοιχεία ενός συστήματος. Φυσικά υπάρχουν ορισμένοι περιορισμοί, όπως ο αριθμός των controllers που μπορούμε να υλοποιήσουμε, οι περιοχές μνήμης που μπορεί να δει ο κάθε controller αλλά και το interface με το οποίο θα επικοινωνούν. Συγκεκριμένα όλοι οι controllers θα πρέπει να επικοινωνούν με το ίδιο interface, για παράδειγμα το AXI4.

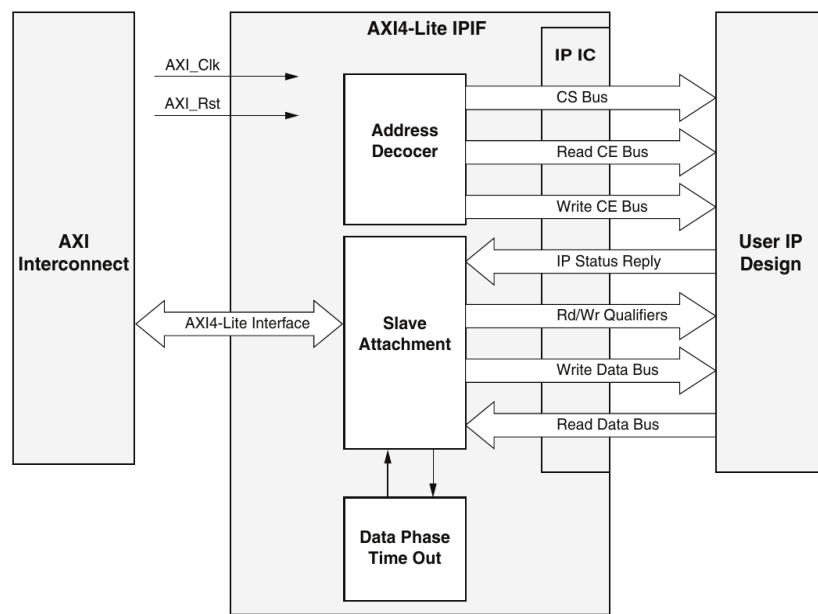
6 Custom Peripherals

6.1 Εισαγωγή

Η χρήση του FPGA ως πλατφόρμα στην οποία τρέχουν ο επεξεργαστής και τα περιφερειακά, μας δίνει την δυνατότητα να υλοποιήσουμε τα δικά μας περιφερειακά και να τα προσαρτήσουμε στον επεξεργαστή. Η διαδικασία, τουλάχιστον όσο αφορά το hardware, είναι αφού δημιουργήσουμε το περιφερειακό μας, να το προσαρμόσουμε κατάλληλα ώστε να μπορεί να επικοινωνεί μέσω του bus το οποίο χρησιμοποιούμε.

6.2 AXI IPIF

Για να είναι πιο εύκολη η διαδικασία ενσωμάτωσης ενός custom περιφερειακού, η Xilinx, έχει καθορίσει το AXI IPIF core. Η λειτουργία του core είναι να μετατρέπει το AXI interface σε ένα πιο απλό interface, το οποίο να μπορούμε να το χρησιμοποιήσουμε εύκολα ως custom περιφερειακό. Το core χειρίζεται όλα τα σήματα του AXI interface και αρκετές λειτουργίες όπως ο χρονισμός. Παραμετροποιώντας κατάλληλα το core μπορούμε να προσθέσουμε στο περιφερειακό ένα μεγάλο αριθμό από registers, οι οποίοι είναι προσβάσιμοι μέσω του bus από διαφορετικές διευθύνσεις(memory mapped). Επίσης το core αναλαμβάνει και το χειρισμό των interrupts που μπορεί να δημιουργήσει το περιφερειακό μας. Στην εικόνα 6.1 φαίνεται το block διάγραμμα του axi ipif core.



DS765_01

Εικόνα 6.1 : Block διάγραμμα AXI IPIF core

6.2.1 AXI IPIF Parameters

Το AXI IPIF core μπορεί να παραμετροποιηθεί ώστε να χρησιμοποιούνται μόνο οι πόροι που είναι απαραίτητοι για την υποστήριξη του περιφερειακού. Οι παράμετροι που μπορούμε να ορίσουμε παραθέτονται στον παρακάτω πίνακα

Πίνακας 3 : Παράμετροι AXI IPIF core

Όνομα Παραμέτρου	Περιγραφή
C_FAMILY	Οικογένεια FPGA chip στην οποία θα υλοποιηθεί το σύστημα
C_USE_WSTRB	Χρήση του Write Strobe
C_DPHASE_TIMEOUT	Data phase time out
C_S_AXI_ADDR_WIDTH	Μήκος της AXI αρτηρίας διευθύνσεων
C_S_AXI_DATA_WIDTH	Μήκος της AXI αρτηρίας δεδομένων
C_S_AXI_MIN_SIZE	Ελάχιστος αριθμός διευθύνσεων για το περιφερειακό
C_ARD_ADDR_RANGE_ARRAY	Πίνακας με ζεύγη Base Address/High Address για κάθε περιοχή διευθύνσεων
C_ARD_NUM_CE_ARRAY	Πίνακας με τον αριθμό των chip enable σημάτων για κάθε περιοχή διευθύνσεων

Στην συνέχεια ακολουθεί μια λεπτομερέστερη ανάλυση των παραμέτρων, με τις τιμές που μπορούν να λάβουν καθώς και την λειτουργία που επηρεάζουν.

6.2.2 Πίνακες καθορισμού περιοχών διευθύνσεων

Μια από τις βασικές λειτουργίες του axi ipif core είναι να αποκωδικοποιεί την διεύθυνση και να παράγει τα κατάλληλα σήματα chip enable(CE)/chip select(CS) για το custom περιφερειακό. Για τον καθορισμό της αποκωδικοποίησης που θα γίνει στα πεδία διευθύνσεων, χρησιμοποιούνται δύο πίνακες :

- C_ARD_ADDR_RANGE_ARRAY και
- C_ARD_NUM_CE_ARRAY

Οι πίνακες δεν έχουν προκαθορισμένο μέγεθος και μπορούν να έχουν όσες εγγραφές χρειάζεται το περιφερειακό. Το ότι οι πίνακες δεν έχουν προκαθορισμένο μέγεθος, δίνει το πλεονέκτημα ότι υλοποιούνται ακριβώς τα κυκλώματα που χρειάζονται, χωρίς να γίνεται σπατάλη των πόρων του ολοκληρωμένου. Τα κυκλώματα που θα προκύψουν είναι βελτιστοποιημένα με τέτοιο τρόπο ώστε να ανταποκρίνονται μόνο στις συγκεκριμένες περιοχές διευθύνσεις που καθορίζονται.

6.2.3 C_ARD_ADDR_RANGE_ARRAY

Οι διευθύνσεις μιας περιοχής διευθύνσεων, καθορίζονται στον πίνακα αυτό ως ζεύγη base address/high address. Από την πλευρά του επεξεργαστή, οι περιοχές διεύθυνσης εμφανίζονται ως συνεχόμενες διευθύνσεις μνήμης της μνήμης που μπορεί να

προσπελάσει. Ένα παράδειγμα καθορισμού δύο περιοχών διευθύνσεων μέσω αυτού του πίνακα, φαίνεται στο παρακάτω τμήμα κώδικα.

6.2.4 C_ARD_NUM_CE_ARRAY

Σε κάθε περιοχή διευθύνσεων που καθορίζονται στο πίνακα C_ARD_ADDR_RANGE_ARRAY, μπορούμε αντιστοιχίσουμε οποιονδήποτε αριθμό από registers που θέλουμε. Για τον λόγο αυτό θα πρέπει να οριστεί και ο αριθμός των CE σημάτων, τα οποία θα παράγονται για κάθε register. Ο αριθμός των CE σημάτων που θα παράγονται καθορίζεται από τον πίνακα C_ARD_NUM_CE_ARRAY. Ο αριθμός των CE σημάτων, που ορίζονται σε κάθε περιοχή μνήμης πρέπει να είναι θετικός σε δυνάμεις του 2, δηλαδή 2, 4, 8, 16 κ.λ.π, για τον λόγο αυτό εάν επιθυμούμε να έχουμε αριθμό που δεν είναι δύναμη του 2, για παράδειγμα το 6, θα πρέπει να επιλέξουμε τον αμέσως μεγαλύτερο αριθμό από registers, δηλαδή 8. Εδώ πρέπει να προσέξουμε ότι μια περιοχή διευθύνσεων θα πρέπει να είναι ορισμένη έτσι ώστε να μπορεί υποστηρίξει τον αριθμό των registers. Ένας καταχωρητής, 32-bit, χρειάζεται 4 διευθύνσεις μνήμης(8-bit).

Ένα παράδειγμα καθορισμού του αριθμού των CE σημάτων για περιοχές διευθύνσεων φαίνεται στο παρακάτω τμήμα κώδικα.

6.2.5 C_S_AXI_ADDR_WIDTH & C_S_AXI_DATA_WIDTH

Οι παράμετροι C_S_AXI_ADDR_WIDTH και C_S_AXI_DATA_WIDTH καθορίζουν το μήκος της αρτηρίας διευθύνσεων και δεδομένων αντίστοιχα. Η μοναδική τιμή που μπορούν να πάρουν είναι το 32. Ο λόγος που καθορίζονται ως παράμετροι είναι για την υποστήριξη 64-bit σε μελλοντικές εκδόσεις.

6.2.6 C_S_AXI_MIN_SIZE

Η τιμή της παραμέτρου καθορίζει την ελάχιστη συνολική περιοχή διευθύνσεων του περιφερειακού. Αυτή πρέπει να είναι δύναμη του 2 και μεγαλύτερη από την high address της μέγιστης περιοχής διευθύνσεων που καθορίζεται στον πίνακα C_ARD_ADDR_RANGE_ARRAY.

2.2.7 C_USE_WSTRB

Σε περίπτωση που χρησιμοποιείται το Write Strobe τότε το σήμα byte enable από το AXI Interface θα μεταφερθεί στο interface του custom περιφερειακού.

6.2.8 C_DPHASE_TIMEOUT

Καθορίζει έναν αριθμό από κύκλους ρολογιού, στον οποίο το περιφερειακό θα πρέπει να απαντήσει στις περιπτώσεις ανάγνωσης και εγγραφής.

6.3 AXI IPIC

Το interface από την πλευρά του περιφερειακού (IPIC) περιέχει μια σειρά από σήματα, τα οποία γίνονται enabled όταν υπάρχει κάποιο αίτημα συναλλαγής, ανάγνωσης ή εγγραφής, και κάποια σήματα τα οποία μπορεί να γράψει το περιφερειακό ως απάντηση στο αίτημα. Στον παρακάτω πίνακα συνοψίζονται τα σήματα αυτά με την σημασία τους.

Πίνακας 4 : Σήματα IP διεπαφής

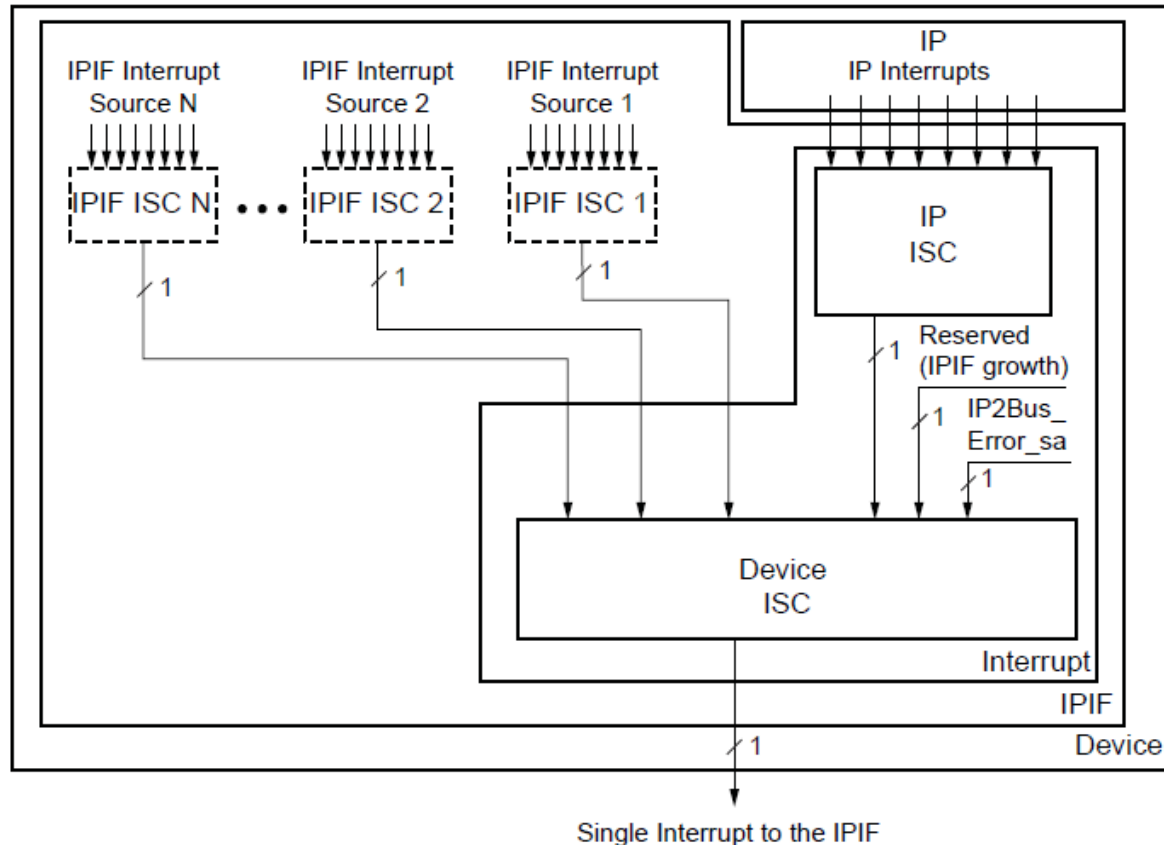
Όνομα Σήματος	Περιγραφή
Bus2IP_Addr	
Bus2IP_Data	
Bus2IP_RNW	
Bus2IP_BE	
Bus2IP_CS	
Bus2IP_RdCE	
Bus2IP_WrCE	
IP2Bus_Data	
IP2Bus_RdAck	
IP2Bus_WrAck	
IP2Bus_Error	
Interrupt	

Τα σήματα του IPIC interface καθορίζονται όμοια για τους διαφορετικούς τύπους αρτηρίας, που μπορούν να χρησιμοποιηθούν με τον επεξεργαστή, όπως το PLB. Παρέχοντας το ίδιο interface στα περιφερειακά δεν χρειάζεται να γίνει κάποια επιπλέον προσαρμογή, όσο αναφορά το περιφερειακό σε περίπτωση χρήσης διαφορετικής αρτηρίας.

6.4 Interrupt Control

Για την υποστήριξη interrupts που μπορεί να παραγάγει το περιφερειακό, θα πρέπει να χρησιμοποιήσουμε το interrupt control component, το οποίο παρέχει ως βιβλιοθήκη η Xilinx. Το component υλοποιείται στο επίπεδο του axi lite ipif, το οποίο χρησιμοποιείται ως attachment στο τμήμα που επικοινωνεί με το bus. Αυτό το component μπορεί να υποστηρίξει πολλαπλά interrupts του περιφερειακού, ενώ παράγει μια έξοδο η οποία θα πρέπει να συνδεθεί στον interrupt controller, που διαχειρίζεται όλα τα interrupts του συστήματος, όπως περιγράφηκε σε προηγούμενη ενότητα.

Στην εικόνα φαίνεται το block διάγραμμα του interrupt control.



DS516_01_022107

Εικόνα 6.2 : Block διάγραμμα interrupt control

Το component υποστηρίζει μια σειρά από μεθόδους που μπορούν να χρησιμοποιηθούν για την καταγραφή των interrupts. Η επιλογή εξαρτάται από την λειτουργικότητα που έχει υλοποιηθεί στο περιφερειακό το οποίο παράγει τα interrupts. Πιο συγκεκριμένα οι μέθοδοι καταγραφής των interrupts είναι :

- Pass Through(inverting ή non-inverting)
- Registered Level (inverting ή non-inverting)
- Positive ή Negative Edge Detect

Στην πρώτη περίπτωση το interrupt σήμα απλά προωθείτε από το interrupt controll και η διαχείριση του πρέπει να γίνει στο επίπεδο του περιφερειακού. Στην δεύτερη περίπτωση το interrupt διαχειρίζεται από το interrupt control αλλά πρέπει να παραμείνει ενεργό το αντίστοιχο σήμα για δύο κύκλους του ρολογιού, ώστε να μπορέσει να το αναγνωρίσει. Τέλος στην τρίτη περίπτωση η αναγνώριση γίνεται στις παρυφές του σήματος και έτσι το περιφερειακό δεν χρειάζεται να κάνει κάποια επιπλέον ενέργεια.

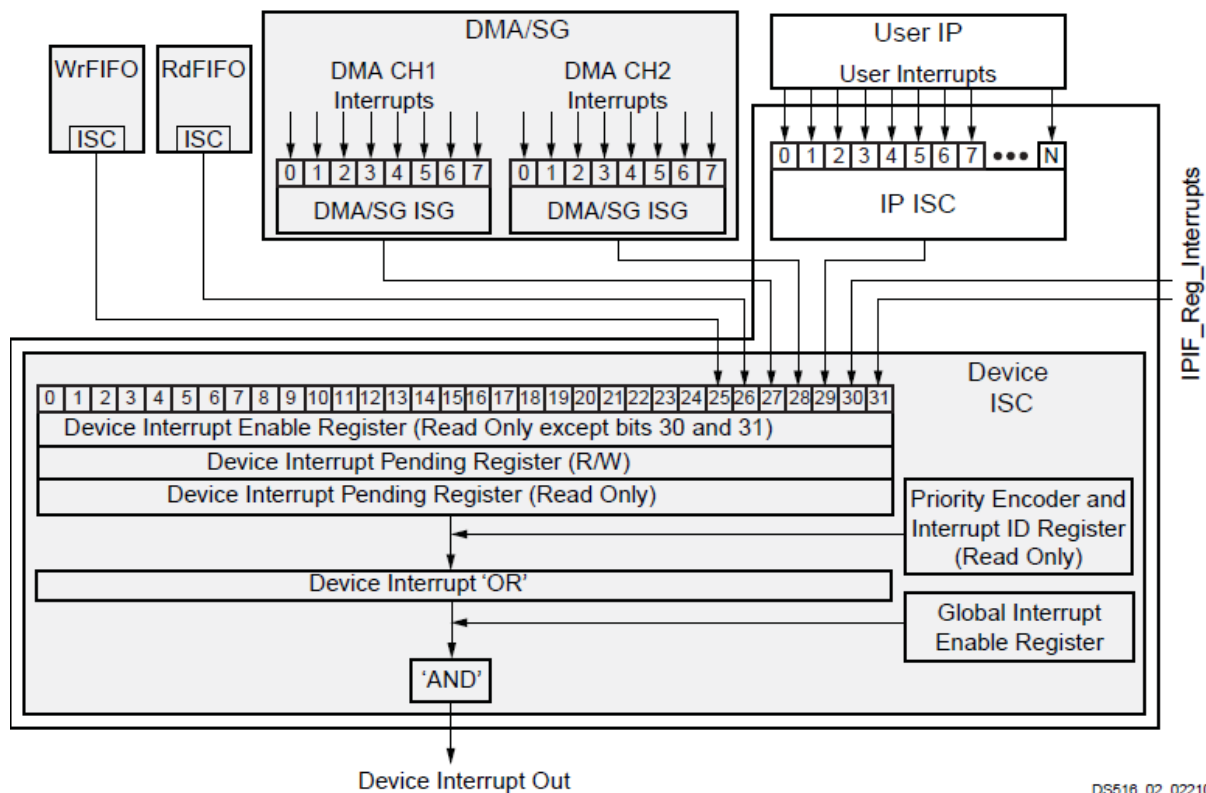
Το interrupt control παρέχει μια σειρά από registers, προσβάσιμους από τον επεξεργαστή και άρα από λογισμικό (drivers), οι οποίοι μπορούν να χρησιμοποιηθούν

για την διαχείριση των interrupts της συγκεκριμένης συσκευής στο σύστημα μας. Για την προσπέλαση των registers του component θα πρέπει να προσθέσουμε ένα επιπλέον εύρος διευθύνσεων, το οποίο θα το αναθέσουμε σε αυτό. Οι τελικές διευθύνσεις προσπέλασης των components καθορίζονται από την πρώτη διεύθυνση του εύρους και ένα προκαθορισμένο offset.

Οι πιο σημαντικοί registers και αυτοί που θα χρησιμοποιήσουμε στους drivers παρουσιάζονται στον πίνακα 5, όπου ακολουθεί, μαζί με το offset από την πρώτη διεύθυνση του εύρους.

Πίνακας 5 : Καταχωρητές interrupt controll

Καταχωρητής	Offset	Περιγραφή
Global Interrupt Enable	0x1C	Ενεργοποιεί τα interrupts της συσκευής, MSB = '1'
IP Interrupt Status Register	0x20	Καταγράφει τα interrupts που παράγονται από την συσκευή
IP Interrupt Enable Register	0x28	Ενεργοποιεί συγκεκριμένα interrupts της συσκευής



Εικόνα 6.3

6.5 Κυκλική Ουρά - Υλοποίηση Custom Περιφερειακού

Ως παράδειγμα θα υλοποιηθεί μια κυκλική ουρά (circular queue), η οποία θα χρησιμοποιηθεί ως περιφερειακό και θα επικοινωνεί με τον επεξεργαστή μέσω της αρτηρίας axi4, η οποία έχει επιλεγεί ως αρτηρία στο σύστημα που έχει υλοποιηθεί για τις δοκιμές.

6.5.1 Μνήμη

Το περιφερειακό αποτελείται από τα αρχεία ram.vhdl, όπου υλοποιείται μια dual port ram και χρησιμοποιείται ως το μέσο αποθήκευσης των δεδομένων. Η μνήμη έχει δύο εισόδους διευθύνσεων, την διεύθυνση εγγραφής και την διεύθυνση ανάγνωσης. Η χρήση dual port ram μας δίνει την δυνατότητα να μην χρησιμοποιήσουμε mux για την επιλογή της κατάλληλης διεύθυνσης, ανάλογα με την ενέργεια που θέλουμε να εκτελέσουμε, δηλαδή εγγραφή ή ανάγνωση. Επίσης η dual port ram δίνει την δυνατότητα για ταυτόχρονη εγγραφή και ανάγνωση, αρκεί οι διευθύνσεις να μην περιέχουν την ίδια τιμή, κάτι που μπορεί να χρησιμοποιηθεί σε υλοποίηση ουράς, αν και δεν γίνεται στο παράδειγμα μας για λόγους απλότητας.

Τα σήματα εισόδου εξόδου με την σημασία τους, περιγράφονται στον πίνακα 6 που ακολουθεί.

Πίνακας 6 : Σήματα μνήμης

Σήμα	Κατεύθυνση	Περιγραφή
Clk	Είσοδος	Ρολόι
Wa(Write Address)	Είσοδος	Διεύθυνση Εγγραφής
We(Write Enable)	Είσοδος	Ενεργοποίηση Εγγραφής
Ra(Read Address)	Είσοδος	Διεύθυνση Ανάγνωσης
Datain	Είσοδος	Δεδομένα Εισόδου
Dataout	Έξοδος	Δεδομένα Εξόδου

Όταν το σήμα We ενεργοποιηθεί (πάρει την τιμή 1), τα δεδομένα που βρίσκονται στην είσοδο datain, εγγράφονται στην διεύθυνση που υποδεικνύεται από το σήμα Wa. Το σήμα dataout σε όλες τις χρονικές στιγμές περιέχει τα δεδομένα που βρίσκονται στην μνήμη στην διεύθυνση που υποδεικνύεται από το σήμα Ra. Στο σχεδιάγραμμα της εικόνας 6.2 φαίνεται η λειτουργία της ram όπως θα χρησιμοποιηθεί ως component στο περιφερειακό. Ο κώδικας για την μνήμη υπάρχει στο παράρτημα Α.

6.5.2 Δείκτες - Pointers

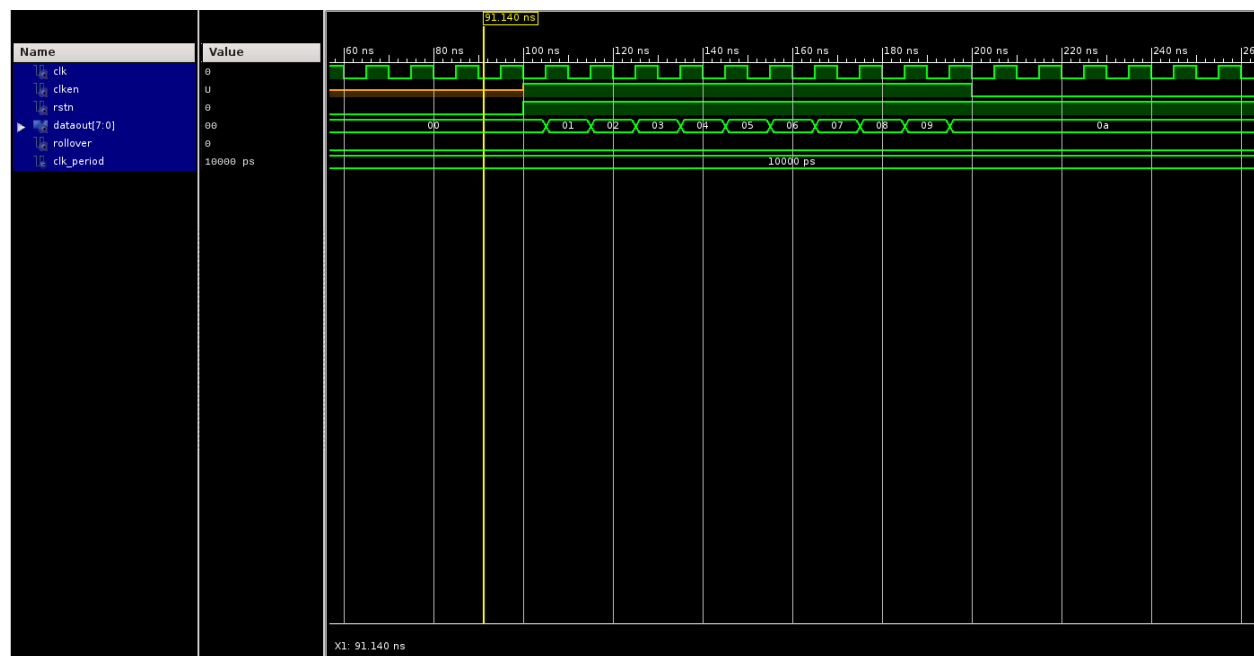
Το δεύτερο στοιχείο που χρειάζεται για την υλοποίηση της κυκλικής ουράς είναι δύο δείκτες: ένας που δείχνει την αρχή της ουράς όπου εγγράφονται τα δεδομένα που εισάγονται στην ουρά και ένας όπου θα δείχνει την διεύθυνση από όπου θα γίνεται ανάγνωση των δεδομένων.

Τα σήματα εισόδου-εξόδου του στοιχείου περιγράφονται στον πίνακα 7 που ακολουθεί.

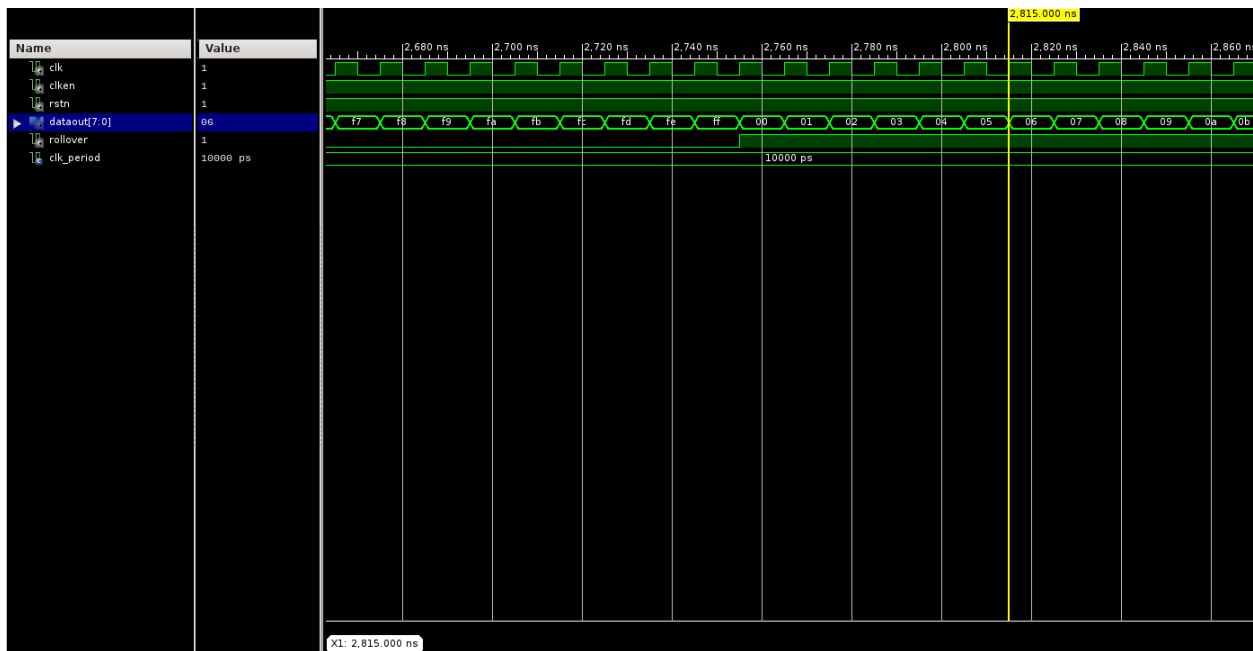
Πίνακας 7 : Σήματα pointer

Σήμα	Κατεύθυνση	Περιγραφή
Clk	Είσοδος	Ρολόι
Clken	Είσοδος	Ενεργοποίηση ρολογιού
Rstn	Είσοδος	Επαναφορά
Dataout	Έξοδος	Διεύθυνση
Rollover	Έξοδος	Αναδίπλωση

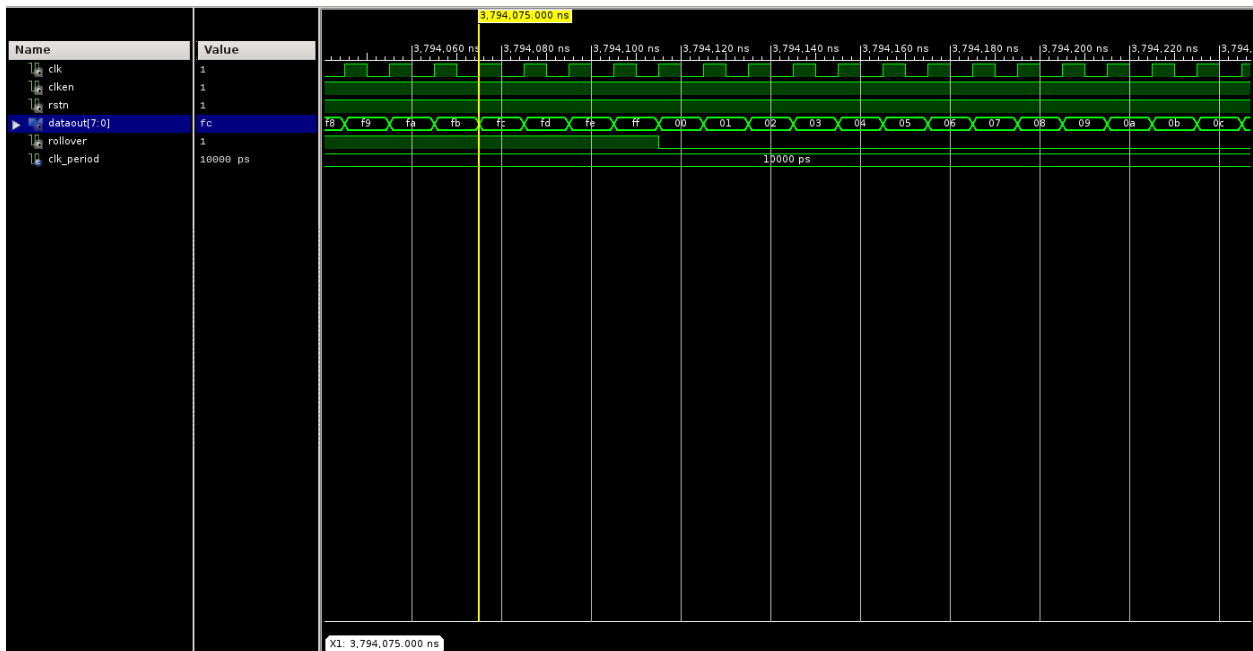
Ο δείκτης περιέχει έναν μετρητή, ο οποίο αυξάνεται σε ανερχόμενη παρυφή του ρολογιού όταν το σήμα clken είναι ενεργοποιημένο (clken = '1'). Το σήμα rstn, αρνητικής λογικής, επαναφέρει το στοιχείο στην αρχική κατάσταση. Το σήμα dataout περιέχει το περιεχόμενο του counter σε κάθε χρονική στιγμή. Τέλος το σήμα rollover υποδεικνύει εάν έχει γίνει αναδίπλωση στην τιμή που περιέχει ο counter. Το σήμα rollover θα το χρησιμοποιήσουμε ώστε να μπορούμε να αναγνωρίσουμε εάν η ουρά είναι γεμάτη ή άδεια όπως θα περιγράψουμε στην επόμενη υποενότητα. Στις εικόνες 6.4, 6.5 και 6.6 φαίνεται η λειτουργικότητα του στοιχείου. Ο κώδικας του στοιχείου παρατίθεται στο παράρτημα Α.



Εικόνα 6.4 : Ενεργοποίηση pointer για 10 κύκλους ρολογιού



Εικόνα 6.5 : Rollover pointer(1)



Εικόνα 6.6 : Rollover pointer(2)

6.5.3 User Logic

Το στοιχείο user logic χρησιμοποιεί τα δύο στοιχεία που περιγράφηκαν προηγουμένως, μνήμη και pointers, κατάλληλα συνδεδεμένα προσθέτοντας την κατάλληλη λογική, ώστε να μπορεί να επικοινωνήσει με τον επεξεργαστή. Το στοιχείο χρησιμοποιεί το interface IPIC για να επικοινωνήσει με το slave attachment στοιχείο, το οποίο υλοποιεί το interface axi4 lite. Από το IPIC interface δεν χρησιμοποιούνται όλα τα σήματα, αλλά

μόνο αυτά που είναι απαραίτητα για την υλοποίηση της απαραίτητης λειτουργίας. Πιο συγκεκριμένα τα σήματα που χρησιμοποιούνται είναι :

- Bus2IP_Data
- Bus2IP_RdCe
- Bus2IP_WrCe
- IP2Bus_Data
- IP2Bus_RdAck
- IP2Bus_WrAck
- IP2Bus_Error
- IP2Bus_Interrupt

Έχοντας παραμετροποιήσει κατάλληλα το slave attachment, το οποίο θα περιγράψουμε στην επόμενη υποενότητα, ξέρουμε ότι όταν υπάρχει αίτημα για εγγραφή στην ουρά, θα ενεργοποιηθεί το MSB του σήματος Bus2IP_WrCe, καθώς επίσης και το σήμα Bus2IP_Data που περιέχει τα δεδομένα τα οποία πρέπει να εισαχθούν στην ουρά, ενώ εάν υπάρχει αίτημα για ανάγνωση από την ουρά, θα ενεργοποιηθεί το MSB του σήματος Bus2IP_RdCe. Στην περίπτωση της ανάγνωσης, το component θα πρέπει να επιστρέψει στον επεξεργαστή τα δεδομένα που αναγνώστηκαν από την ουρά χρησιμοποιώντας το σήμα IP2Bus_Data.

Σε κάθε περίπτωση μετά την εκτέλεση της εκάστοτε ενέργειας, θα πρέπει να ενεργοποιηθεί το αντίστοιχο σήμα acknowledge IP2Bus_WrAck και IP2Bus_RdAck.

Στις περιπτώσεις που δεν μπορεί να πραγματοποιηθεί για κάποιο λόγο η ζητούμενη ενέργεια (στην περίπτωση μας εάν η ουρά είναι γεμάτη και ζητηθεί εγγραφή ή είναι άδεια και ζητηθεί ανάγνωση), θα πρέπει να ενεργοποιηθεί το σήμα IP2Bus_Error ώστε να ακυρωθεί η ενέργεια.

Τέλος στις περιπτώσεις που χρειάζεται να πραγματοποιηθεί κάποια ενέργεια από τον επεξεργαστή, ενεργοποιείται το σήμα IP2Bus_Interrupt. Οι περιπτώσεις που διεγείρουμε κάποιο interrupt είναι όταν αλλάξει η κατάσταση σε γεμάτη ή άδεια. Τα interrupts θα μπορούσαν να χρησιμοποιηθούν στην περίπτωση που ένα πρόγραμμα διαχειρίζεται πάνω από μια ουρές και αναθέτει διαφορετικές προτεραιότητες εγγραφής και ανάγνωσης ανάλογα με την κατάσταση της εκάστοτε ουράς.

6.5.4 Top Module Custom Περιφερειακού

Το top module είναι το τελικό αρχείο που θα δημιουργήσουμε, το αρχείο αυτό περιέχει το user logic component, καθώς και τα κατάλληλα component για την διαχείριση των σημάτων του AXI4 interface, επίσης περιέχει και την υποστήριξη των interrupts. Ο wizard του εργαλείου που προσφέρει η Xilinx παράγει ένα αρκετά ολοκληρωμένο αρχείο

που απαιτεί μόνο μια μικρή παραμετροποίηση για να προσαρμοστεί στις ανάγκες του εκάστοτε περιφερειακού.

Πιο αναλυτικά το top module που παράγει ο wizard περιέχει το axi lite ipif component το οποίο διαχειρίζεται όλα τα σήματα του axi interface και είναι υπεύθυνο ώστε να παράγει τα κατάλληλα σήματα του ipic interface με το οποίο μιλάει το περιφερειακό και το οποίο αναλύσαμε προηγουμένως.

Το axi lite ipif επίσης περιέχει τον κατάλληλα παραμετροποιημένο decoder, ο οποίος παράγει τα σήματα Bus2IP_RdCe, Bus2IP_WrCe και Bus2IP_Cs, τα οποία κάνει assert, όταν υπάρχει κάποιο αίτημα ανάγνωσης ή εγγραφής σε μια διεύθυνση μνήμης, μέσα στο εύρος τιμών του περιφερειακού. Η αντιστοιχία των διευθύνσεων μνήμης και των κατάλληλων chip enable και chip select σημάτων που θα γίνουν assert, γίνεται με βάση τις τιμές που θα ορίσουμε στους πίνακες C_ARD_ADDR_RANGE_ARRAY και C_ARD_NUM_CE_ARRAY, όπως αναφέρθηκε και προηγουμένως. Για την υλοποίηση της κυκλικής ουράς καθορίζουμε δύο εύρη πεδίων μνήμης, το πρώτο εύρος θα χρησιμοποιηθεί για την επικοινωνία με το περιφερειακό και το δεύτερο εύρος για τον χειρισμό του interrupt control.

Ο κώδικας για τον καθορισμό των διευθύνσεων μνήμης, δηλαδή ο καθορισμός των δύο πινάκων φαίνεται στο παρακάτω τμήμα του κώδικα.

```
constant ZERO_ADDR_PAD : std_logic_vector(0 to 31) := (others => '0');
constant USER_SLV_BASEADDR : std_logic_vector := C_BASEADDR or X"00000000";
constant USER_SLV_HIGHADDR : std_logic_vector := C_BASEADDR or X"0000000F";
constant INTR_BASEADDR : std_logic_vector := C_BASEADDR or X"00000100";
constant INTR_HIGHADDR : std_logic_vector := C_BASEADDR or X"0000013F";
constant IPIF_ARD_ADDR_RANGE_ARRAY : SLV64_ARRAY_TYPE :=
(
    ZERO_ADDR_PAD & USER_SLV_BASEADDR, -- user logic slave space base address
    ZERO_ADDR_PAD & USER_SLV_HIGHADDR, -- user logic slave space high address
    ZERO_ADDR_PAD & INTR_BASEADDR, -- interrupt slave space base address
    ZERO_ADDR_PAD & INTR_HIGHADDR -- interrupt slave space high address
);
constant USER_SLV_NUM_REG : integer := 4;
constant USER_NUM_REG : integer := USER_SLV_NUM_REG;
constant INTR_NUM_CE : integer := 16;
constant TOTAL_IPIF_CE : integer := USER_NUM_REG + INTR_NUM_CE;
constant IPIF_ARD_NUM_CE_ARRAY : INTEGER_ARRAY_TYPE :=
(
    0 => USER_SLV_NUM_REG, -- number of ce for user logic slave space
    1 => INTR_NUM_CE -- number of ce for interrupt control space
);
```

Για το πρώτο εύρος διευθύνσεων ορίζουμε την παραγωγή τεσσάρων CE σημάτων ενώ για το δεύτερο δεκαέξι. Όσο αναφορά το περιφερειακό, αν και θα χρησιμοποιήσουμε μόνο ένα σήμα CE, για εγγραφή και ανάγνωση, ορίζουμε την παραγωγή τεσσάρων CE

σημάτων, καθώς ο μεγαλύτερος αριθμός σημάτων οδηγεί σε αποκωδικοποιητή που απαιτεί λιγότερους πόρους του ολοκληρωμένου.

Το δεύτερο στοιχείο που περιέχει το top module είναι το user logic το οποίο αντιστοιχεί στο custom περιφερειακό και το οποίο αναλύσαμε στην προηγούμενη υποενότητα.

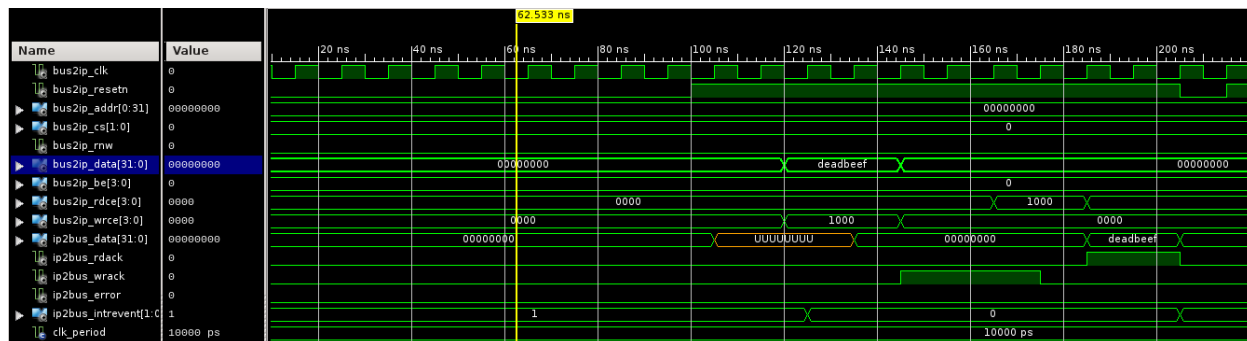
Τέλος θα πρέπει να προσθέσουμε και να συνδέσουμε κατάλληλα το component interrupt control που προσφέρεται από την Xilinx, το οποίο δεν προστίθεται αυτόματα από τον Wizard. Η προσθήκη του interrupt control φαίνεται στο παρακάτω τμήμα του κώδικα.

```
INTERRUPT_CONTROL_1 : entity interrupt_control_v2_01_a.interrupt_control
generic map
(
    C_NUM_CE => INTR_NUM_CE,
    C_NUM_IPIF_IRPT_SRC => INTR_NUM_IPIF_IRPT_SRC,
    C_IP_INTR_MODE_ARRAY => INTR_IP_INTR_MODE_ARRAY,
    C_INCLUDE_DEV_PENCODER => INTR_INCLUDE_DEV_PENCODER,
    C_INCLUDE_DEV_ISC => INTR_INCLUDE_DEV_ISC,
    C_IPIF_DWIDTH => IPIF_SLV_DWIDTH
)
port map
(
    -- Inputs From the IPIF Bus
    Bus2IP_Clk => ipif_Bus2IP_Clk,
    Bus2IP_Reset => ipif_Bus2IP_Reset,
    Bus2IP_Data => ipif_Bus2IP_Data,
    Bus2IP_BE => ipif_Bus2IP_BE,
    Interrupt_RdCE => ipif_Bus2IP_RdCE(INTR_CE_INDEX to INTR_CE_INDEX+INTR_NUM_CE-1),
    Interrupt_WrCE => ipif_Bus2IP_WrCE(INTR_CE_INDEX to INTR_CE_INDEX+INTR_NUM_CE-1),
    -- Interrupt inputs from the IPIF sources that will
    -- get registered in this design
    IPIF_Reg_Interrupts => intr_IPIF_Reg_Interrupts,
    -- Level Interrupt inputs from the IPIF sources
    IPIF_Lvl_Interrupts => intr_IPIF_Lvl_Interrupts,
    -- Inputs from the IP Interface
    IP2Bus_IntrEvent => user_IP2Bus_IntrEvent,
    -- Final Device Interrupt Output
    Intr2Bus_DevIntr => IP2INTC_Irpt,
    -- Status Reply Outputs to the Bus
    Intr2Bus_DBus => intr_IP2Bus_Data,
    Intr2Bus_WrAck => intr_IP2Bus_WrAck,
    Intr2Bus_RdAck => intr_IP2Bus_RdAck,|
    Intr2Bus_Error => intr_IP2Bus_Error,
    Intr2Bus_Retry => open,
    Intr2Bus_ToutSup => open
);
```

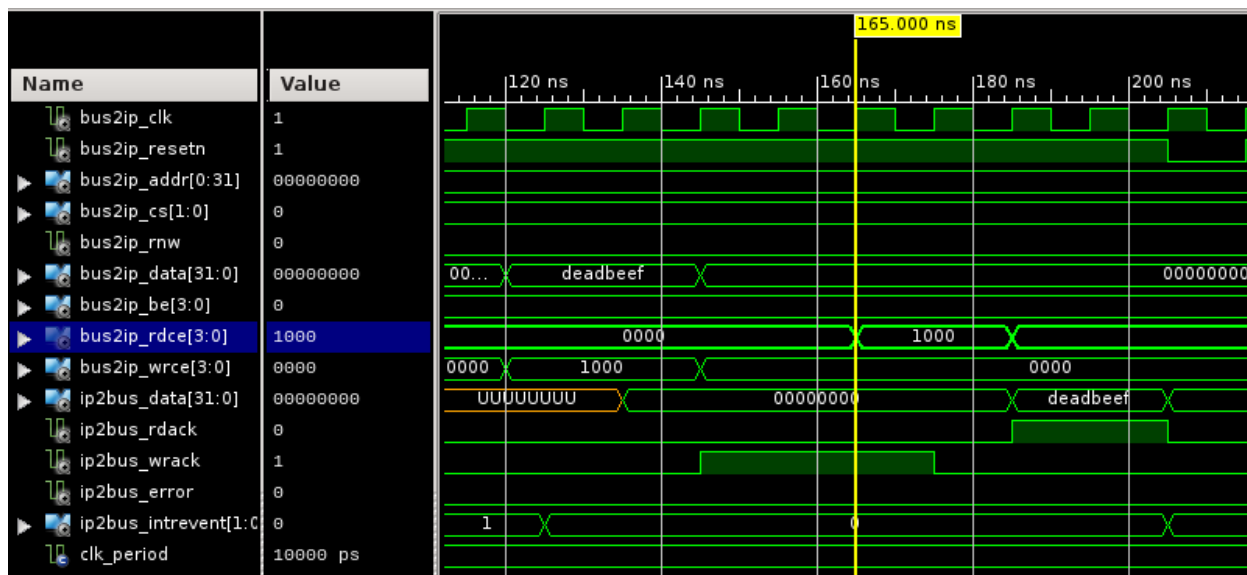
Ο κώδικας του αρχείου ολοκληρωμένος υπάρχει στο παράρτημα Α.

6.5.5 Έλεγχος λειτουργίας

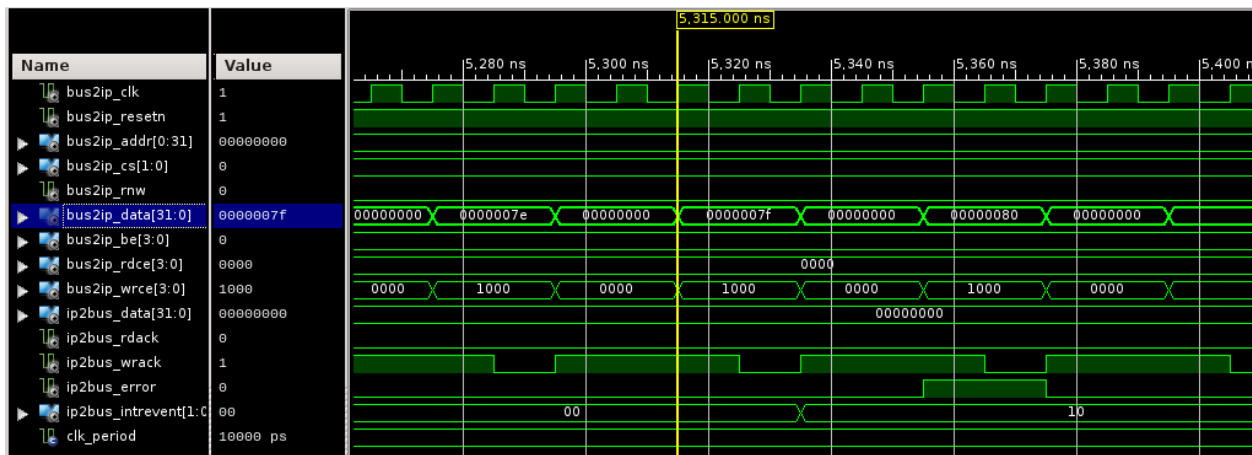
Ο έλεγχος του περιφερειακού γίνεται σε δύο επίπεδα, το πρώτο είναι κατά την διάρκεια της ανάπτυξης με χρήση testbench. Η λειτουργικότητα που ελέγχεται είναι η εγγραφή και η ανάγνωση, καθώς και οι περιπτώσεις σφάλματος και interrupt. Οι εικόνες που ακολουθούν είναι screenshots από τις εξομοιώσεις που εκτελέστηκαν και παρουσιάζουν την αναμενόμενη συμπεριφορά του component με βάση τον σχεδιασμό που έγινε.



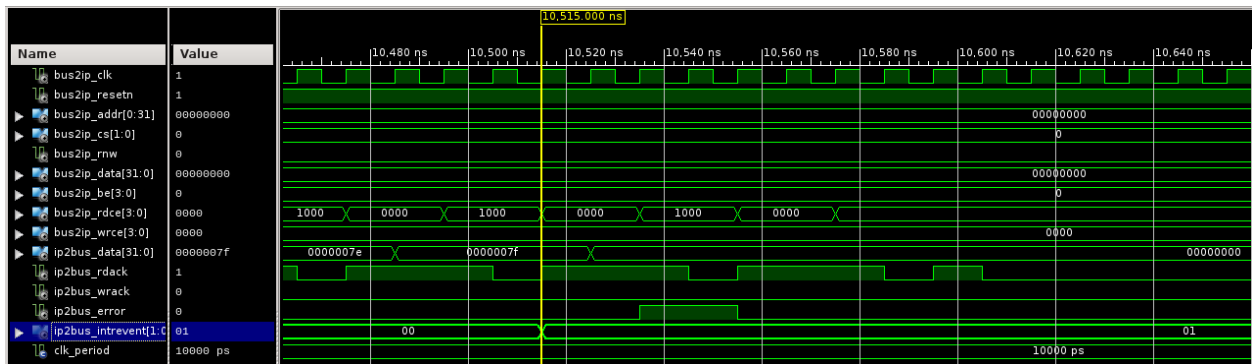
Εικόνα 6.7 : Διαδικασία εγγραφής στο custom περιφερειακό



Εικόνα 6.8 : Διαδικασία ανάγνωσης από το custom περιφερειακό



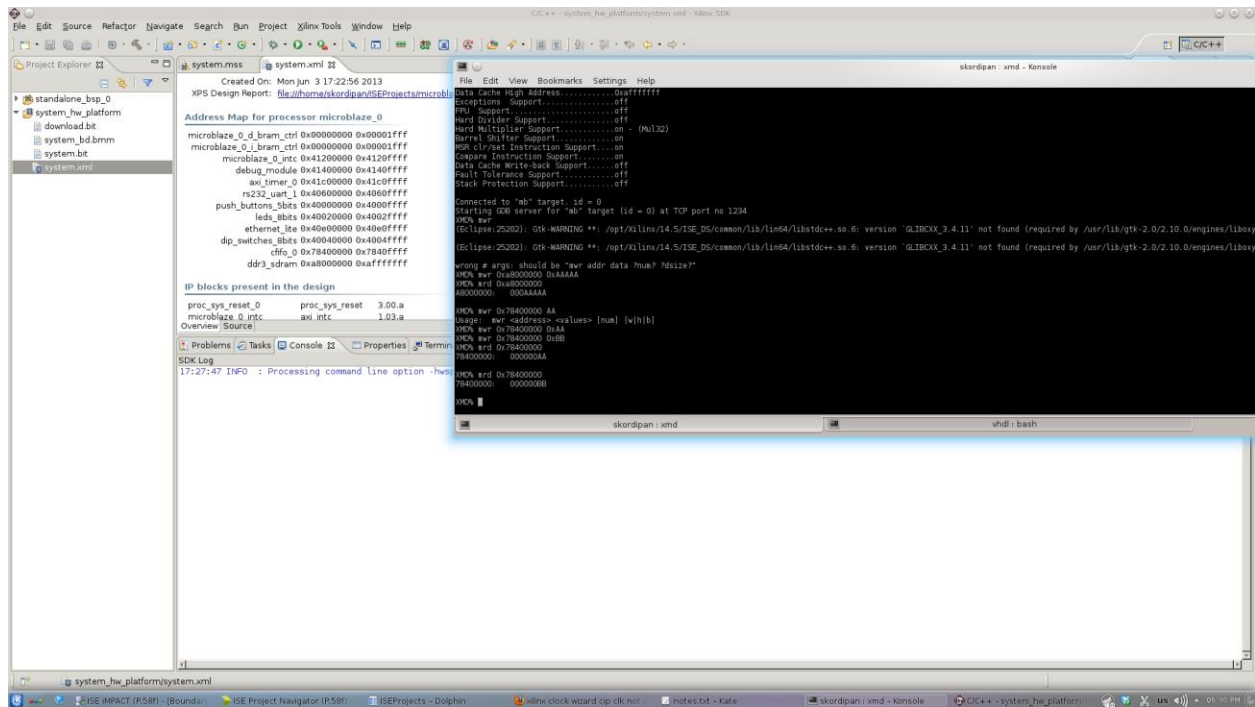
Εικόνα 6.9 : Σφάλμα εγγραφής στο custom περιφερειακό



Εικόνα 6.10 : Σφάλμα ανάγνωσης από το custom περιφερειακό

Σε δεύτερο στάδιο υλοποιούμε πρώτα το τελικό σύστημα, αφού πρώτα έχουμε προσθέσει σε αυτό το custom περιφερειακό και του έχουμε αναθέσει ένα εύρος διευθύνσεων μνήμης (memory mapped). Για να δοκιμάσουμε την λειτουργία χρησιμοποιούμε το εργαλείο XMD, το οποίο συνδέεται μέσω του MDM στον επεξεργαστή και μέσω του οποίου μπορούμε να εκτελέσουμε εντολές όπως εγγραφή και ανάγνωση από τους καταχωρητές, την μνήμη κ.λ.π.

Για να ελέγξουμε την σωστή λειτουργία του περιφερειακού θα εγγράψουμε στην πρώτη θέση μνήμης από το εύρος που του έχουμε αναθέσει μια λέξη των 32-bit, όσο είναι και το εύρος την ουράς και στην συνέχεια θα διαβάσουμε πάλι από την ίδια θέση μνήμης. Εφόσον όλα έχουν υλοποιηθεί σωστά, η λέξη που διαβάζουμε πρέπει να είναι ίδια με αυτήν που γράψαμε. Η διαδικασία με τα αποτελέσματα της εμφανίζεται στην εικόνα 6.8.



Εικόνα 6.11 : Επιβεβαίωση λειτουργικότητας στην αναπτυξιακή κάρτα.

7. Standalone Software

Εφόσον έχουμε υλοποιήσει το σύστημα, μπορούμε να το χρησιμοποιήσουμε με bare metal software ή χρησιμοποιώντας κάποιου είδους λειτουργικό στο οποίο μπορούμε στην συνέχεια να εκτελέσουμε τα προγράμματά μας. Ανάλογα με τις επιλογές μας μπορούμε να ρυθμίσουμε τον επεξεργαστή κατάλληλα, ώστε αν πετύχουμε καλύτερη επίδοση, είτε σε ταχύτητα είτε σε area.

Σε αυτήν την ενότητα θα αναφερθούμε σε standalone software, ενώ στην επόμενη θα αναφερθούμε στην χρήση του Linux ως λειτουργικού συστήματος.

Μια standalone εφαρμογή χωρίζεται σε δύο τμήματα. Το πρώτο τμήμα περιέχει τις απαραίτητες βιβλιοθήκες για την διαχείριση του επεξεργαστή, την πρόσβαση στα περιφερειακά του (drivers) και την υλοποίηση της standard βιβλιοθήκης της C/C++. Το δεύτερο τμήμα περιέχει την εφαρμογή την οποία εκτελεί ο επεξεργαστής.

7.1 XParameters.h

Ένα από τα πιο βασικά αρχεία είναι το xparameters.h το οποίο παράγεται από το libgen. Το συγκεκριμένο αρχείο περιέχει definitions που περιγράφουν πλήρως το υλοποιημένο σύστημα, από τις δυνατότητες του επεξεργαστή μέχρι τις δυνατότητες του εκάστοτε περιφερειακού καθώς και τις διευθύνσεις από τις οποίες μπορούν να προσπελαστούν. Για παράδειγμα στο παρακάτω τμήμα κώδικα από το αρχείο που έχει παραχθεί για το σύστημα που έχουμε υλοποιήσει, φαίνονται τα definitions για το GPIO core το οποίο συνδέεται στα δοκιμαστικά LED's της κάρτας και θα χρησιμοποιηθεί για το πρώτο πρόγραμμα που θα υλοποιηθεί.

7.2 C Standard Library

Η υλοποίησης της standard βιβλιοθήκης της C που παρέχεται είναι η newlib, η οποία είναι κατάλληλη για χρήση σε ενσωματωμένα συστήματα. Παρέχει τις υλοποιήσεις για τις πιο κοινές συναρτήσεις, όπως αυτές που παρέχονται στα stdio, stdlib και string. Επίσης παρέχεται μια εξελιγμένη μαθηματική βιβλιοθήκη που βασίζεται στην libm της newlib και παρέχει τις πιο κοινές μαθηματικές συναρτήσεις.

Σε αυτό το σημείο θα πρέπει να επισημανθεί ότι σε περίπτωση που η εφαρμογή θα πρέπει να κάνει πράξεις με δεκαδικούς, θα πρέπει να έχει ενσωματωθεί και ρυθμιστεί σωστά η FPU(floating point unit). Σε διαφορετική περίπτωση θα έχουμε χαμηλή απόδοση λόγω της εξομίωσης σε λογισμικό των πράξεων κινητής υποδιαστολής.

7.3 Προγράμματα Οδήγησης Περιφερειακών

Για τα cores που παρέχονται και μπορούν να χρησιμοποιηθούν ως περιφερειακά στο σύστημα μας, παρέχονται επίσης και βιβλιοθήκες που χρησιμοποιούνται ως drivers για τις συσκευές και μπορούν να χρησιμοποιηθούν για να προσπελαστούν μέσω software.

Οι drivers έχουν μια κοινή δομή ώστε να υπάρχει μια ομοιομορφία στα προγράμματα, η ίδια δομή έχει χρησιμοποιηθεί και για την ανάπτυξη του αντίστοιχου driver για το custom περιφερειακό που αναπτύχθηκε και παρουσιάστηκε στην προηγούμενη ενότητα.

Οι drivers παρέχουν συνήθως δύο data structures, ένα που περιέχει τα κατάλληλα fields για τον χειρισμό της συσκευής, όπως η base address του περιφερειακού, ενώ το δεύτερο data structure περιέχει τις ρυθμίσεις που έχουν γίνει σε αυτό. Εκτός από τα structures οι drivers περιέχουν και τις κατάλληλες συναρτήσεις για τον σωστό χειρισμό τους, όπως ανάγνωση, εγγραφή αλλά και ενεργοποίηση απενεργοποίηση των interrupts όπου αυτά υποστηρίζονται.

7.4 Πρώτη Εφαρμογή

Η πρώτη εφαρμογή που έχει υλοποιηθεί είναι αρκετά απλή, αλλά μέσω αυτής θα παρουσιαστούν δύο βασικά τμήματα ενός προγράμματος, η εγγραφή και ανάγνωση δεδομένων από περιφερειακά και η χρήση interrupts. Η εφαρμογή χρησιμοποιεί δύο από τα GPIO cores τα οποία έχουν υλοποιηθεί στο σύστημα, το πρώτο συνδέεται στα push buttons που έχει η αναπτυξιακή κάρτα και το δεύτερο στα LED's. Επίσης χρησιμοποιείται ο interrupt controller για την αναγνώριση και εξυπηρέτηση των interrupts και ο timer που έχει ενσωματωθεί στο σύστημα για την παραγωγή των interrupts σε συγκεκριμένα χρονικά διαστήματα.

Τα push buttons χρησιμοποιούνται ως το μέσο με το οποίο ο χρήστης εισάγει εντολές και τις οποίες το πρόγραμμα πρέπει να εκτελέσει. Η κάρτα έχει πέντε push buttons τα οποία τα αντιστοιχούμε στις παρακάτω εντολές ενέργειας.

Πίνακας 8 : Λειτουργίες Push Buttons

Κουμπί	Εντολή/Ενέργεια
N	Ενεργοποίηση των LED
S	Απενεργοποίηση των LED
W	Αναβοσβήνει τα LED
E	Σταματάει να αναβοσβήνει τα LED
C	Έξοδος

Για να μην δεσμευθεί ο επεξεργαστής στον έλεγχο εισόδου από τον χρήστη, τα push buttons είναι ρυθμισμένα έτσι ώστε να παράγουν interrupt όταν ο χρήστης πατήσει κάποιο από αυτά. Ελέγχοντας την τιμή που διαβάζουμε από το GPIO μπορούμε να διακρίνουμε το κουμπί το οποίο πατήθηκε. Το άναμμα ή το σβήσιμο των LED's γίνεται εύκολα εγγράφοντας την τιμή 0xFF ή 0x00 αντίστοιχα στο GPIO που είναι συνδεδεμένο με τα LED's της κάρτας. Με την ίδια λογική μπορούμε να πετύχουμε και τις άλλες δύο λειτουργίες, χρησιμοποιώντας όμως τον timer έτσι ώστε να παράγει ανά καθορισμένο

χρονικό διάστημα ένα interrupt, το οποίο θα ανάβει και θα σβήνει εναλλάξ τα LED's. Η ενεργοποίηση και απενεργοποίηση επιτυγχάνεται εκκινώντας και σταματώντας τον timer.

Το πρόγραμμα αποτελείται από πέντε στο σύνολο συναρτήσεις, μια για την αρχικοποίηση του συστήματος και εκκαθάρισή του, δύο συναρτήσεις για την εξυπηρέτηση των interrupts και την main η οποία εκτελείται με την εκκίνηση του συστήματος.

Το πρόγραμμα χωρίζεται σε πέντε διαφορετικά αρχεία, τα interrupts.h/c που περιέχουν τις δύο συναρτήσεις που χειρίζονται τα interrupts, τα platform.h/c και το main.c που περιέχει την main συνάρτηση. Στο main.c έχουν δηλωθεί επίσης ως global οι μεταβλητές που αντιστοιχούν στους drivers του συστήματος.

7.4.1 platform.h/c

Τα αρχεία platform.h και platform.c περιέχουν τα πρωτότυπα και τις υλοποιήσεις, αντίστοιχα, των συναρτήσεων platform_init και platform_cleanup. Η συνάρτηση platform_init είναι υπεύθυνη για την σωστή αρχικοποίηση του συστήματος, η οποία περιλαμβάνει την αρχικοποίηση των drivers των συσκευών που θα χρησιμοποιηθούν από το πρόγραμμα καθώς και την αρχικοποίηση του συστήματος των interrupts.

Η αρχικοποίηση των drivers γίνεται μέσω των συναρτήσεων XY_Initialize, όπου Y αντικαθίστατε από το προσδιοριστικό του εκάστοτε περιφερειακού, για παράδειγμα η αρχικοποίηση του driver για τον interrupt controller, γίνεται καλώντας την συνάρτηση XIntc_Initialize όπως φαίνεται στο παρακάτω τμήμα κώδικα.

```
//Initialize interrupt controller driver
puts("Initializing Interrupt Controller....");
status = XIntc_Initialize(&interrupt_controller, XPAR_INTC_0_DEVICE_ID);
if(status != XST_SUCCESS) {
    puts("Failed");
    return XST_FAILURE;
}
puts("Passed");
```

Για κάθε συσκευή ακόμα και εάν είναι του ίδιου τύπου, θα πρέπει να αρχικοποιήσουμε ένα διαφορετικό στιγμιότυπο του driver. Πιο συγκεκριμένα στο σύστημα μας έχουμε τρεις συσκευές GPIO από τις οποίες έχουν χρησιμοποιηθεί οι δύο, για τον έλεγχο των push buttons και των LED's. Στο πρόγραμμα έχουν δημιουργηθεί δύο στιγμιότυπα του driver, ένα για κάθε συσκευή.

Για την αρχικοποίηση του συστήματος των interrupts, θα πρέπει να γίνει η ενεργοποίηση αυτών στον επεξεργαστή, καθώς και στα περιφερειακά τα οποία θα τα παράγουν, καθώς και ο καθορισμός των συναρτήσεων που θα εξυπηρετούν τα εκάστοτε interrupt. Στο πρόγραμμα έχουν υλοποιηθεί δύο τέτοιες συναρτήσεις, μια για

την εξυπηρέτηση των interrupts που θα παράγουν τα push button όταν ο χρήστης πιέσει ένα από αυτά, και μια για την εξυπηρέτηση του interrupt που θα παράγει ο timer μετά από ένα καθορισμένο, από το πρόγραμμα, χρονικό διάστημα.

Η ενεργοποίηση των interrupts σε κάθε συσκευή γίνεται με την κλήση των κατάλληλων συναρτήσεων. Πιο συγκεκριμένα για την GPIO συσκευή, θα πρέπει να καλέσουμε την συνάρτηση για την ενεργοποίηση των interrupts στο κανάλι, μια GPIO συσκευή προσφέρει δύο κανάλια, όπου συνδεδεμένα τα push buttons και στην συνέχεια πρέπει να ενεργοποιήσουμε γενικά την παραγωγή των interrupt από την συσκευή. Η ενεργοποίηση γίνεται με την κλήση των συναρτήσεων που φαίνονται στο παρακάτω τμήμα κώδικα

```
//enable interrupts in push buttons
XGpio_InterruptEnable(&push_buttons, XGPIO_IR_CH1_MASK);
XGpio_InterruptGlobalEnable(&push_buttons);
```

Αντίστοιχα θα πρέπει να ενεργοποιηθούν και τα interrupts στον timer που όμως γίνεται με την συνάρτηση που φαίνεται παρακάτω

```
//enable interrupts in timer
puts("Enabler interrupts in timer");
XTmrCtr_SetOptions(&timer, 0x0, XTC_INT_MODE_OPTION);
```

Η σύνδεση της συνάρτησης που θα εξυπηρετήσει το interrupt γίνεται με την συνάρτηση XIntc_Connect όπως φαίνεται στην συνέχεια

```
status = XIntc_Connect(&interrupt_controller,
XPAR_INTC_0_TMRCTR_0_VEC_ID,(XInterruptHandler)timer_handler, &timer);
    if (status != XST_SUCCESS) {
        puts("Failed");
        return XST_FAILURE;
    }
```

Αφού καθοριστούν οι συναρτήσεις εξυπηρέτησης των interrupts, θα πρέπει να γίνει εκκίνηση του interrupt controller σε real mode ώστε να αναγνωρίζει τα interrupt που παράγονται από τις συσκευές.

```
//Start interrupt controller in real mode
status = XIntc_Start(&interrupt_controller, XIN_REAL_MODE);
    if (status != XST_SUCCESS) {
        return XST_FAILURE;
    }
```

Τέλος θα πρέπει να καθοριστεί η συνάρτηση, παρέχεται έτοιμη, η οποία εξυπηρετεί το interrupt που παράγεται από τον interrupt controller και να ενεργοποιηθούν τα interrupts στον επεξεργαστή.

```
microblaze_register_handler( (XInterruptHandler) XIntc_DeviceInterruptHandler,
XPAR_MICROBLAZE_0_INTC_DEVICE_ID);
microblaze_enable_interrupts();
```

7.4.2 interrupts.h/c

Για την εξυπηρέτηση των interrupts, που έχουμε ενεργοποιήσει για τις ανάγκες του προγράμματος καθορίζονται δύο συναρτήσεις. Η πρώτη η οποία διαχειρίζεται τα interrupts που παράγει η GPIO συσκευή, η οποία είναι συνδεδεμένη με τα push buttons, όταν ο χρήστης πιέσει ένα από αυτά, πρώτα διαβάζει την τιμή που έχει η συσκευή, ώστε να αναγνωρίσει πιο κουμπί πατήθηκε και στη συνέχεια αλλάζει την τιμή μιας μεταβλητής, ώστε να αλλάξει την κατάσταση του προγράμματος κατάλληλα. Με αυτόν τον τρόπο λειτουργεί σαν interface και μέσω εισόδου εντολών από τον χρήστη.

Η δεύτερη συνάρτηση η οποία εξυπηρετεί τα interrupts του timer απλά αλλάζει την τιμή που έχει ο GPIO που ελέγχει τα LED's της κάρτας. Έτσι τα LED's αναβοσβήνουν ανά ένα συγκεκριμένο χρονικό διάστημα.

Και στις δύο περιπτώσεις αφού εξυπηρετηθεί το interrupt, θα πρέπει να γίνει clear το interrupt ώστε να μην αναγνωριστεί ξανά από τον interrupt controller. Η ενέργεια αυτή γίνεται με κατάλληλη συνάρτηση η οποία παρέχεται από τους drivers της εκάστοτε συσκευής, για παράδειγμα στην περίπτωση της GPIO, θα πρέπει να γίνει κλήση στην συνάρτηση XGpio_InterruptClear.

```
XGpio_InterruptClear(GpioPtr, XGPIO_IR_CH1_MASK);
```

7.4.3 Main.c

Στην κυρίως συνάρτηση πρώτα καλείται η συνάρτηση για την αρχικοποίηση του συστήματος και στην συνέχεια γίνεται ο καθορισμός του χρονικού διαστήματος, σε κύκλους ρολογιού, ανά του οποίου ο timer θα παράγει το interrupt. Τέλος χρησιμοποιείται ένας βρόγχος χωρίς σώμα, ώστε ο επεξεργαστής να παραμένει ενεργός και να μπορεί να εξυπηρετήσει τα interrupts τα οποία θα παραχθούν

```
/*
 * Switch LEDs on & off
 * based on user input
 * (user buttons)
 */
int main(void) {

    int status;
    //Initialize system
    puts("Initializing System....");
    status = platform_init();
    if(status != XST_SUCCESS) {
        puts("Initialization Failed");
        return XST_FAILURE;
    }
}
```

```

puts("Initialization Success");

//set timer value to 0
XTmrCtr_SetResetValue(&timer, 0x0, 0x00FFFFFF);

//set timer down count
XTmrCtr_SetOptions(&timer, 0x0, XTC_DOWN_COUNT_OPTION | TmrCtr_GetOptions(&timer, 0x0));

puts("Menu :\nN : turn leds on\nS : turn leds off\nW : start blinking\nE : stop linking\nC : exit\n");

//wait for exit
while(!exit_flag);

//Cleanup system
puts("Cleaning up System....");
platform_cleanup();
puts("Done");

return XST_SUCCESS;
}

```

7.5 Προγράμματα Οδήγησης Custom Περιφερειακών

Στο πρόγραμμα που περιγράφηκε παραπάνω είδαμε ένα παράδειγμα πως μπορούμε να χρησιμοποιήσουμε τους drivers που μας παρέχονται για τον χειρισμό των αντίστοιχων περιφερειακών. Στην περίπτωση όμως που υπάρξει η ανάγκη για υλοποίηση ενός custom περιφερειακού, θα πρέπει να μπορεί να είναι και διαχειρίσιμο μέσω του λογισμικού, το οποίο εκτελείται στον επεξεργαστή. Για να γίνει αυτό δυνατό θα πρέπει να υλοποιηθεί και ο αντίστοιχος driver. Στην συνέχεια θα παρουσιαστεί ο driver που υλοποιήθηκε για το custom περιφερειακό που αναλύσαμε στην προηγούμενη ενότητα.

Για την υλοποίηση των συναρτήσεων χρησιμοποιήθηκε αντίστοιχη δομή με αυτές του GPIO driver, ώστε να υπάρχει μια ομοιομορφία στον κώδικα του προγράμματος που θα τους χρησιμοποιήσει. Ο driver της συσκευής παρέχει στον χρήστη τις βασικές συναρτήσεις για εγγραφή, ανάγνωση και διαχείριση των interrupts που παράγει, καθώς επίσης τα structures και τις συναρτήσεις αρχικοποίησης για τον ίδιο τον driver.

7.5.1 cfifo.h

Όλα τα πρωτότυπα των συναρτήσεων καθώς και τα structures καθορίζονται στο αρχείο cfifo.h. Το structure το οποίο αντιπροσωπεύει την συσκευή ορίζεται ως

```

typedef struct {
    u32 BaseAddress;          /* Device base address */
    u32 IsReady;              /* Device is initialized and ready */
    int InterruptPresent;     /* Are interrupts supported in h/w */
} CFifo;

```

Το structure περιέχει τρία πεδία, το BaseAddress, όπου μετά την αρχικοποίηση περιέχει την πρώτη διεύθυνση μνήμης με το εύρος που έχει ανατεθεί στην συσκευή, το IsReady, όπου μετά την αρχικοποίηση περιέχει την τιμή XIL_COMPONENT_IS_READY και το πεδίο InterruptPresent, το οποίο δείχνει εάν η συσκευή είναι συνδεδεμένη με τον interrupt controller και υποστηρίζονται τα interrupts, ενώ η τιμή που καταχωρείται παράγεται από το εργαλείο δημιουργίας του λογισμικού (libgen).

Για την υποστήριξη πολλαπλών συσκευών του ίδιου τύπου σε ένα σύστημα, όλες οι συναρτήσεις δέχονται ως παράμετρο μια μεταβλητή τύπου CFifo από την οποία εξάγεται η base address του εκάστοτε περιφερειακού και η ενέργεια πραγματοποιείται μόνο στην συγκεκριμένη συσκευή.

7.5.2 cfifo.c

Η υλοποίηση των βασικών συναρτήσεων, όπως εγγραφή, ανάγνωση και αρχικοποίηση του driver βρίσκονται στο αρχείο cfifo.c. Η συνάρτηση εγγραφής δέχεται μόνο ένα όρισμα που είναι η διεύθυνση μνήμης του structure, και το οποίο αντιπροσωπεύει μια συσκευή στο σύστημα μας. Επειδή έχουμε ορίσει ότι στην πρώτη διεύθυνση μνήμης θα γίνεται η ανάγνωση και η εγγραφή, η συνάρτηση απλά διαβάζει την base address και επιστρέφει την τιμή.

```
u32 CFifo_Read(CFifo *InstancePtr)
{
    Xil_AssertNonvoid(InstancePtr != NULL);
    Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    return CFifo_ReadReg(InstancePtr->BaseAddress,
                        0x0);
}
```

Η συνάρτηση εγγραφής είναι παρόμοια με την διαφορά ότι δέχεται μια επιπλέον παράμετρο, τα δεδομένα που θα γράψει, όμως δεν επιστρέφει κάποια τιμή.

```
void CFifo_Write(CFifo *InstancePtr, u32 Mask)
{
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    CFifo_WriteReg(InstancePtr->BaseAddress,
                    0x0,
                    Mask);
}
```

Η ανάγνωση όπως και η εγγραφή γίνεται με την χρήση μιας macro συνάρτησης, η οποία δέχεται μια διεύθυνση μνήμης και ένα offset, ενώ επιστρέφει την τιμή που βρίσκεται στην διεύθυνση μνήμης όπου σχηματίζεται από τον συνδυασμό τους. Με την χρήση της macro συνάρτησης ο driver γίνεται κατά το μεγαλύτερο μέρος του portable.

7.5.3 cfifo_sinit.c

Η συνάρτηση αρχικοποίησης διαβάζει και αναθέτει τις κατάλληλες τιμές στο structure του οποίου την διεύθυνση δέχεται ως παράμετρο. Οι τιμές, όπως αναφέραμε και προηγουμένως, παράγονται από το εργαλείο libgen, μαζί με ένα DeviceID το οποίο χρησιμοποιείται ώστε να καθοριστεί η εκάστοτε συσκευή.

```
int CFifo_Initialize(CFifo * InstancePtr, u16 DeviceId)
{
    CFifo_Config *ConfigPtr;
    /*
     * Assert arguments
     */
    Xil_AssertNonvoid(InstancePtr != NULL);

    /*
     * Lookup configuration data in the device configuration table.
     * Use this configuration info down below when initializing this
     * driver.
     */
    ConfigPtr = CFifo_LookupConfig(DeviceId);
    if (ConfigPtr == (CFifo_Config *) NULL) {
        InstancePtr->IsReady = 0;
        return (XST_DEVICE_NOT_FOUND);
    }

    return CFifo_CfgInitialize(InstancePtr, ConfigPtr,
                               ConfigPtr->BaseAddress);
}
```

7.5.4 cfifo_intr.c

Οι υλοποιήσεις των συναρτήσεων διαχείρισης των interrupts της συσκευής περιέχονται στο αρχείο cfifo_intr.c. Οι συναρτήσεις χρησιμοποιούνται για την ενεργοποίηση/απενεργοποίηση γενικά ή συγκεκριμένων interrupts που μπορούν να παραχθούν από την συσκευή.

Η ενεργοποίηση και απενεργοποίηση γίνεται εγγράφοντας τις κατάλληλες τιμές στους καταχωρητές που περιέχονται στον interrupt control του περιφερειακού, όπως περιγράφηκε στην ενότητα 6.

Για να ενεργοποιηθούν γενικά τα interrupts της συσκευής θα πρέπει να γίνει set το MSB του Device Global Interrupt Enable Register.

```
void CFifo_InterruptGlobalEnable(CFifo * InstancePtr)
{
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertVoid(InstancePtr->InterruptPresent == TRUE);

    CFifo_WriteReg(InstancePtr->BaseAddress, CFIFO_GIE_OFFSET,
```

```

        CFIFO_GIE_GINTR_ENABLE_MASK);
}

```

Αντίστοιχα για την ενεργοποίηση του εκάστοτε interrupt θα πρέπει να εγγραφεί η κατάλληλη τιμή στον IP Interrupt Enable Register. Για να γίνει αλλαγή μόνο σε συγκεκριμένο bit του καταχωρητή και έτσι να μην αλλάξει η κατάσταση στα υπόλοιπα interrupts, πρώτα διαβάζονται τα περιεχόμενα του καταχωρητή, οπότε η τιμή που εγγράφεται είναι το αποτέλεσμα της λογικής πράξης, της τιμής που περιείχε ο καταχωρητής και αυτής που δέχεται ως όρισμα η συνάρτηση.

```

void CFifo_InterruptEnable(CFifo * InstancePtr, u32 Mask)
{
    u32 Register;

    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertVoid(InstancePtr->InterruptPresent == TRUE);

    /*
     * Read the interrupt enable register and only enable the specified
     * interrupts without disabling or enabling any others.
     */

    Register = CFifo_ReadReg(InstancePtr->BaseAddress, CFIFO_IER_OFFSET);
    CFifo_WriteReg(InstancePtr->BaseAddress, CFIFO_IER_OFFSET,
        Register | Mask);
}

```

Για λόγους μεταφερισιμότητας του κώδικα και στην περίπτωση αυτή οι τιμές δεν είναι hardcoded, αλλά καθορίζονται σε άλλο αρχείο ως defines.

7.5.5 cfifo_l.h

Το τελευταίο αρχείο που θα αναλυθεί είναι το cfifo_l.h. Το αρχείο αυτό περιέχει defines και macro συναρτήσεις, για την εγγραφή σε συγκεκριμένους καταχωρητές του περιφερειακού. Τα defines που περιέχει είναι διευθύνσεις μνήμης για τους καταχωρητές, όπως αυτοί της συσκευής και του interrupt controller. Οι διευθύνσεις αυτές εξαρτώνται από την σχεδίαση που έχει γίνει στο hardware του περιφερειακού. Για παράδειγμα το εύρος μνήμης το οποίο έχει ανατεθεί στον interrupt controller, καθορίζεται από τα offset 0x100 έως 0x13F της base address του περιφερειακού. Ο Device Global Interrupt Enable Register έχει offset 0x1C ως προς την base address του interrupt controller, συνεπώς η διεύθυνση μνήμης του register από την πλευρά του επεξεργαστή, είναι το offset 0x11C της base address του περιφερειακού. Αντίστοιχα καθορίζονται και οι τιμές για τους υπόλοιπους registers.

```

#define CFIFO_GIE_OFFSET    0x11C /**< Global interrupt enable register */
#define CFIFO_ISR_OFFSET    0x120 /**< Interrupt status register */
#define CFIFO_IER_OFFSET    0x128 /**< Interrupt enable register */

```

Στο αρχείο καθορίζονται επίσης defines με τις κατάλληλες τιμές που πρέπει να εγγραφούν για την ενεργοποίηση των interrupts στους καταχωρητές.

```
#define CFIFO_IR_EMPTY_MASK 0x1 /** Mask empty interrupt bit */
#define CFIFO_IR_FULL_MASK 0x2 /** Mask full interrupt bit */
#define CFIFO_IR_MASK 0x3 /** Mask of all bits */
#define CFIFO_GIE_GINTR_ENABLE_MASK 0x80000000
```

Τέλος στο αρχείο καθορίζονται ως macros οι low level συναρτήσεις που θα πρέπει να κληθούν για την εγγραφή και ανάγνωση των διευθύνσεων μνήμης που αντιστοιχούν στους καταχωρητές.

```
#define CFifo_WriteReg(BaseAddress, RegOffset, Data) \
\
    CFifo_Out32((BaseAddress) + (RegOffset), (u32)(Data))

#define CFifo_ReadReg(BaseAddress, RegOffset) \
\
    CFifo_In32((BaseAddress) + (RegOffset))
```

7.6 Δοκιμαστική εφαρμογή προγράμματος οδήγησης ενός custom περιφερειακού

Για την επιβεβαίωση της σωστής λειτουργίας του προγράμματος οδήγησης, υλοποιήθηκε μια εφαρμογή η οποία πραγματοποιεί εγγραφή και ανάγνωση στο περιφερειακό καθώς και υλοποιήσεις συναρτήσεων για την εξυπηρέτηση των interrupts που μπορούν να παραχθούν.

Η δομή των αρχείων είναι ίδια με αυτή της πρώτης εφαρμογής που αναλύσαμε στην ενότητα 7.4. Η εκτέλεση του προγράμματος ξεκινάει με την αρχικοποίηση του συστήματος ενώ στην συνέχεια καλείται η συνάρτηση selftest για το custom περιφερειακό. Εφόσον ο έλεγχος είναι επιτυχής γίνεται στη συνέχεια εγγραφή στην ουρά έως ότου γεμίσει, από όπου και θα παραχθεί το αντίστοιχο interrupt. Στην συνέχεια πραγματοποιούνται αναγνώσεις έως ότου αδειάσει η ουρά, οπότε παράγεται και αυτό το κατάλληλο interrupt.

```
int main(void) {
    int status;

    //Initialize system
    puts("Initializing System....");
    status = platform_init();
    if(status != XST_SUCCESS) {
        puts("Initialization Failed");
        return XST_FAILURE;
    }
    puts("Initialization Success");

    puts("Self test cfifo...");
    if(CFifo_SelfTest(&cfifo) != XST_SUCCESS) {
```

```

        puts("Failed");
        return XST_FAILURE;
    }
    puts("Passed");

    write_flag = 1;
    read_flag = 1;
    u32 data = 0;

    for(;;) {
        while(write_flag == 1) {
            CFifo_Write(&cfifo, data);
            data++;
            printf("%lu queued\n", data);
        }
        while(read_flag == 1) {
            data = CFifo_Read(&cfifo);
            printf("%lu dequeued\n", data);
        }
        break;
    }

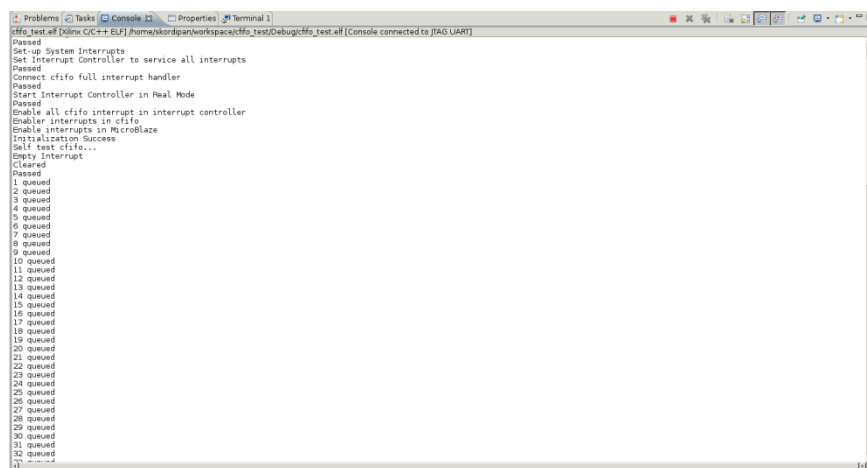
    //Cleanup system
    puts("Cleaning up System....");
    platform_cleanup();
    puts("Done");

    for(;;);

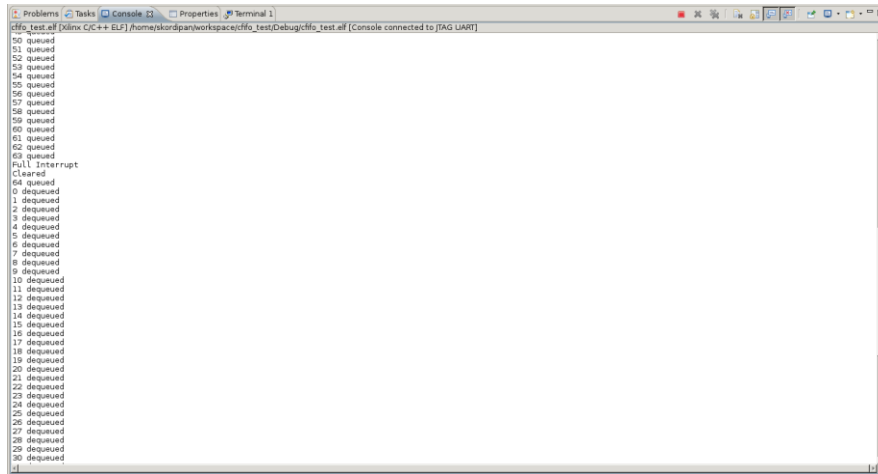
    return XST_SUCCESS;
}

```

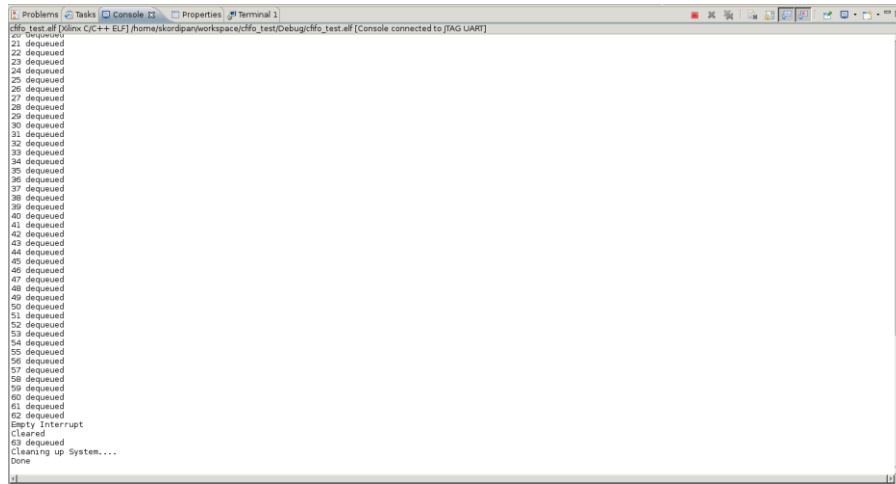
Το αποτέλεσμα της εκτέλεσης της εφαρμογής, φαίνεται στις εικόνες 7.1 - 7.2, διακρίνοντας διαφορετικά στάδια της εφαρμογής.



Εικόνα 7.1 : Εκκίνηση εφαρμογής & Cfifo Selftest



Εικόνα 7.2 : Full Interrupt



Εικόνα 7.3 : Empty Interrupt & Cleanup

Παράρτημα Α : Κώδικας

A.1 Σύστημα

A.1.1 MHS

```
# #####  
# Created by Base System Builder Wizard for Xilinx EDK 14.5 Build EDK_P.58f  
# Fri May 31 17:16:01 2013  
# Target Board: xilinx.com kc705 Rev C  
# Family: kintex7  
# Device: xc7k325t  
# Package: ffg900  
# Speed Grade: -2  
# #####  
PARAMETER VERSION = 2.1.0  
  
PORT sm_fan_pwm_net_vcc = net_vcc, DIR = O  
PORT ddr_memory_we_n = ddr_memory_we_n, DIR = O  
PORT ddr_memory_ras_n = ddr_memory_ras_n, DIR = O  
PORT ddr_memory_odt = ddr_memory_odt, DIR = O  
PORT ddr_memory_dqs_n = ddr_memory_dqs_n, DIR = IO, VEC = [0:0]  
PORT ddr_memory_dqs = ddr_memory_dqs, DIR = IO, VEC = [0:0]  
PORT ddr_memory_dq = ddr_memory_dq, DIR = IO, VEC = [7:0]  
PORT ddr_memory_dm = ddr_memory_dm, DIR = O, VEC = [0:0]  
PORT ddr_memory_ddr3_rst = ddr_memory_ddr3_rst, DIR = O  
PORT ddr_memory_cs_n = ddr_memory_cs_n, DIR = O  
PORT ddr_memory_clk_n = ddr_memory_clk_n, DIR = O, SIGIS = CLK  
PORT ddr_memory_clk = ddr_memory_clk, DIR = O, SIGIS = CLK  
PORT ddr_memory_cke = ddr_memory_cke, DIR = O  
PORT ddr_memory_cas_n = ddr_memory_cas_n, DIR = O  
PORT ddr_memory_ba = ddr_memory_ba, DIR = O, VEC = [2:0]  
PORT ddr_memory_addr = ddr_memory_addr, DIR = O, VEC = [13:0]  
PORT RS232_Uart_1_sout = RS232_Uart_1_sout, DIR = O  
PORT RS232_Uart_1_sin = RS232_Uart_1_sin, DIR = I  
PORT RESET = RESET, DIR = I, SIGIS = RST, RST_POLARITY = 1  
PORT Push_Buttons_5Bits_TRI_I = Push_Buttons_5Bits_TRI_I, DIR = I, VEC = [4:0]  
PORT LEDs_8Bits_TRI_O = LEDs_8Bits_TRI_O, DIR = O, VEC = [7:0]  
PORT Ethernet_Lite_TX_EN = Ethernet_Lite_TX_EN, DIR = O  
PORT Ethernet_Lite_TX_CLK = Ethernet_Lite_TX_CLK, DIR = I  
PORT Ethernet_Lite_TXD = Ethernet_Lite_TXD, DIR = O, VEC = [3:0]  
PORT Ethernet_Lite_RX_ER = Ethernet_Lite_RX_ER, DIR = I  
PORT Ethernet_Lite_RX_DV = Ethernet_Lite_RX_DV, DIR = I  
PORT Ethernet_Lite_RX_CLK = Ethernet_Lite_RX_CLK, DIR = I  
PORT Ethernet_Lite_RXD = Ethernet_Lite_RXD, DIR = I, VEC = [3:0]  
PORT Ethernet_Lite_PHY_RST_N = Ethernet_Lite_PHY_RST_N, DIR = O  
PORT Ethernet_Lite_MDIO = Ethernet_Lite_MDIO, DIR = IO  
PORT Ethernet_Lite_MDC = Ethernet_Lite_MDC, DIR = O  
PORT Ethernet_Lite_CRS = Ethernet_Lite_CRS, DIR = I  
PORT Ethernet_Lite_COL = Ethernet_Lite_COL, DIR = I  
PORT DIP_Switches_TRI_I = DIP_Switches_TRI_I, DIR = I, VEC = [3:0]  
PORT CLK_P = CLK, DIR = I, DIFFERENTIAL_POLARITY = P, SIGIS = CLK, CLK_FREQ = 200000000  
PORT CLK_N = CLK, DIR = I, DIFFERENTIAL_POLARITY = N, SIGIS = CLK, CLK_FREQ = 200000000
```

```

BEGIN proc_sys_reset
  PARAMETER INSTANCE = proc_sys_reset_0
  PARAMETER HW_VER = 3.00.a
  PARAMETER C_EXT_RESET_HIGH = 1
  PORT MB_Debug_Sys_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
  PORT Dcm_locked = proc_sys_reset_0_Dcm_locked
  PORT MB_Reset = proc_sys_reset_0_MB_Reset
  PORT Slowest_sync_clk = clk_100_0000MHzPLLE0
  PORT Interconnect_aresetn = proc_sys_reset_0_Interconnect_aresetn
  PORT Ext_Reset_In = RESET
  PORT BUS_STRUCT_RESET = proc_sys_reset_0_BUS_STRUCT_RESET
END

```

```

BEGIN axi_intc
  PARAMETER INSTANCE = microblaze_0_intc
  PARAMETER HW_VER = 1.03.a
  PARAMETER C_BASEADDR = 0x41200000
  PARAMETER C_HIGHADDR = 0x4120ffff
  BUS_INTERFACE S_AXI = axi4lite_0
  BUS_INTERFACE INTERRUPT = microblaze_0_interrupt
  PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
  PORT INTR = RS232_Uart_1_Interrupt & DIP_Switches_8Bits_IP2INTC_Irpt & Push_Buttons_5Bits_IP2INTC_Irpt & Ethernet_Lite_IP2INTC_Irpt & axi_timer_0_Interrupt & Cfifo_IP2INTC_Irpt
END

```

```

BEGIN lmb_v10
  PARAMETER INSTANCE = microblaze_0_ilmb
  PARAMETER HW_VER = 2.00.b
  PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
  PORT LMB_CLK = clk_100_0000MHzPLLE0
END

```

```

BEGIN lmb_bram_if_cntlr
  PARAMETER INSTANCE = microblaze_0_i_bram_ctrl
  PARAMETER HW_VER = 3.10.c
  PARAMETER C_BASEADDR = 0x00000000
  PARAMETER C_HIGHADDR = 0x00001fff
  BUS_INTERFACE SLMB = microblaze_0_ilmb
  BUS_INTERFACE BRAM_PORT = microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
END

```

```

BEGIN lmb_v10
  PARAMETER INSTANCE = microblaze_0_dlmb
  PARAMETER HW_VER = 2.00.b
  PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
  PORT LMB_CLK = clk_100_0000MHzPLLE0
END

```

```

BEGIN lmb_bram_if_cntlr
  PARAMETER INSTANCE = microblaze_0_d_bram_ctrl
  PARAMETER HW_VER = 3.10.c
  PARAMETER C_BASEADDR = 0x00000000
  PARAMETER C_HIGHADDR = 0x00001fff

```

```
BUS_INTERFACE SLMB = microblaze_0_dldb
BUS_INTERFACE BRAM_PORT = microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END
```

```
BEGIN bram_block
PARAMETER INSTANCE = microblaze_0_bram_block
PARAMETER HW_VER = 1.00.a
BUS_INTERFACE PORTA = microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
BUS_INTERFACE PORTB = microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END
```

```
BEGIN microblaze
PARAMETER INSTANCE = microblaze_0
PARAMETER HW_VER = 8.50.a
PARAMETER C_INTERCONNECT = 2
PARAMETER C_USE_BARREL = 1
PARAMETER C_USE_FPU = 0
PARAMETER C_DEBUG_ENABLED = 1
PARAMETER C_ICACHE_BASEADDR = 0xa8000000
PARAMETER C_ICACHE_HIGHADDR = 0xffffffff
PARAMETER C_USE_ICACHE = 1
PARAMETER C_CACHE_BYTE_SIZE = 8192
PARAMETER C_ICACHE_ALWAYS_USED = 1
PARAMETER C_DCACHE_BASEADDR = 0xa8000000
PARAMETER C_DCACHE_HIGHADDR = 0xffffffff
PARAMETER C_USE_DCACHE = 1
PARAMETER C_DCACHE_BYTE_SIZE = 8192
PARAMETER C_DCACHE_ALWAYS_USED = 1
BUS_INTERFACE ILMB = microblaze_0_ilmb
BUS_INTERFACE DLMB = microblaze_0_dldb
BUS_INTERFACE M_AXI_DP = axi4lite_0
BUS_INTERFACE M_AXI_DC = axi4_0
BUS_INTERFACE M_AXI_IC = axi4_0
BUS_INTERFACE DEBUG = microblaze_0_debug
BUS_INTERFACE INTERRUPT = microblaze_0_interrupt
PORT MB_RESET = proc_sys_reset_0_MB_Reset
PORT CLK = clk_100_0000MHzPLLE0
END
```

```
BEGIN mdm
PARAMETER INSTANCE = debug_module
PARAMETER HW_VER = 2.10.a
PARAMETER C_INTERCONNECT = 2
PARAMETER C_USE_UART = 1
PARAMETER C_BASEADDR = 0x41400000
PARAMETER C_HIGHADDR = 0x4140ffff
BUS_INTERFACE S_AXI = axi4lite_0
BUS_INTERFACE MBDEBUG_0 = microblaze_0_debug
PORT Debug_SYS_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
END
```

```
BEGIN clock_generator
PARAMETER INSTANCE = clock_generator_0
PARAMETER HW_VER = 4.03.a
```

```

PARAMETER C_CLKIN_FREQ = 200000000
PARAMETER C_CLKOUT0_FREQ = 400000000
PARAMETER C_CLKOUT0_PHASE = 337.5
PARAMETER C_CLKOUT0_GROUP = PLLE0
PARAMETER C_CLKOUT0_BUF = FALSE
PARAMETER C_CLKOUT1_FREQ = 400000000
PARAMETER C_CLKOUT1_GROUP = PLLE0
PARAMETER C_CLKOUT1_BUF = FALSE
PARAMETER C_CLKOUT2_FREQ = 250000000
PARAMETER C_CLKOUT2_PHASE = 10
PARAMETER C_CLKOUT2_DUTY_CYCLE = 0.0625
PARAMETER C_CLKOUT2_GROUP = PLLE0
PARAMETER C_CLKOUT2_BUF = FALSE
PARAMETER C_CLKOUT3_FREQ = 100000000
PARAMETER C_CLKOUT3_GROUP = PLLE0
PARAMETER C_CLKOUT4_FREQ = 200000000
PARAMETER C_CLKOUT4_GROUP = PLLE0
PARAMETER C_CLKOUT0_DUTY_CYCLE = 0.500000
PARAMETER C_CLKOUT0_VARIABLE_PHASE = FALSE
PARAMETER C_CLKOUT1_DUTY_CYCLE = 0.500000
PARAMETER C_CLKOUT1_PHASE = 0
PARAMETER C_CLKOUT1_VARIABLE_PHASE = FALSE
PARAMETER C_CLKOUT2_VARIABLE_PHASE = FALSE
PARAMETER C_CLKOUT3_BUF = TRUE
PARAMETER C_CLKOUT3_DUTY_CYCLE = 0.500000
PARAMETER C_CLKOUT3_PHASE = 0
PARAMETER C_CLKOUT3_VARIABLE_PHASE = FALSE
PARAMETER C_CLKOUT4_BUF = TRUE
PARAMETER C_CLKOUT4_DUTY_CYCLE = 0.500000
PARAMETER C_CLKOUT4_PHASE = 0
PARAMETER C_CLKOUT4_VARIABLE_PHASE = FALSE
PORT LOCKED = proc_sys_reset_0_Dcm_locked
PORT CLKOUT3 = clk_100_0000MHzPLLE0
PORT RST = RESET
PORT CLKOUT2 = clk_25_0000MHz10PLLE0_nobuf
PORT CLKOUT1 = clk_400_0000MHzPLLE0_nobuf
PORT CLKOUT0 = clk_400_0000MHz1PLLE0_nobuf
PORT CLKOUT4 = clk_200_0000MHzPLLE0
PORT CLKIN = CLK
END

```

```

BEGIN axi_timer
PARAMETER INSTANCE = axi_timer_0
PARAMETER HW_VER = 1.03.a
PARAMETER C_COUNT_WIDTH = 32
PARAMETER C_ONE_TIMER_ONLY = 0
PARAMETER C_BASEADDR = 0x41c00000
PARAMETER C_HIGHADDR = 0x41c0ffff
BUS_INTERFACE S_AXI = axi4lite_0
PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
PORT Interrupt = axi_timer_0_Interrupt
END

```

```

BEGIN axi_interconnect
  PARAMETER INSTANCE = axi4lite_0
  PARAMETER HW_VER = 1.06.a
  PARAMETER C_INTERCONNECT_CONNECTIVITY_MODE = 0
  PORT INTERCONNECT_ARESETN = proc_sys_reset_0_Interconnect_aresetn
  PORT INTERCONNECT_ACLK = clk_100_0000MHzPLLE0
END

```

```

BEGIN axi_interconnect
  PARAMETER INSTANCE = axi4_0
  PARAMETER HW_VER = 1.06.a
  PORT interconnect_aclk = clk_100_0000MHzPLLE0
  PORT INTERCONNECT_ARESETN = proc_sys_reset_0_Interconnect_aresetn
END

```

```

BEGIN axi_uartlite
  PARAMETER INSTANCE = RS232_Uart_1
  PARAMETER HW_VER = 1.02.a
  PARAMETER C_BAUDRATE = 9600
  PARAMETER C_DATA_BITS = 8
  PARAMETER C_USE_PARITY = 0
  PARAMETER C_ODD_PARITY = 1
  PARAMETER C_BASEADDR = 0x40600000
  PARAMETER C_HIGHADDR = 0x4060ffff
  BUS_INTERFACE S_AXI = axi4lite_0
  PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
  PORT TX = RS232_Uart_1_sout
  PORT RX = RS232_Uart_1_sin
  PORT Interrupt = RS232_Uart_1_Interrupt
END

```

```

BEGIN axi_gpio
  PARAMETER INSTANCE = Push_Buttons_5Bits
  PARAMETER HW_VER = 1.01.b
  PARAMETER C_GPIO_WIDTH = 5
  PARAMETER C_ALL_INPUTS = 1
  PARAMETER C_INTERRUPT_PRESENT = 1
  PARAMETER C_IS_DUAL = 0
  PARAMETER C_BASEADDR = 0x40000000
  PARAMETER C_HIGHADDR = 0x4000ffff
  BUS_INTERFACE S_AXI = axi4lite_0
  PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
  PORT GPIO_IO_I = Push_Buttons_5Bits_TRI_I
  PORT IP2INTC_Irpt = Push_Buttons_5Bits_IP2INTC_Irpt
END

```

```

BEGIN axi_gpio
  PARAMETER INSTANCE = LEDs_8Bits
  PARAMETER HW_VER = 1.01.b
  PARAMETER C_GPIO_WIDTH = 8
  PARAMETER C_ALL_INPUTS = 0
  PARAMETER C_INTERRUPT_PRESENT = 0
  PARAMETER C_IS_DUAL = 0
  PARAMETER C_BASEADDR = 0x40020000
  PARAMETER C_HIGHADDR = 0x4002ffff

```

```

BUS_INTERFACE S_AXI = axi4lite_0
PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
PORT GPIO_IO_O = LEDs_8Bits_TRI_O
END

```

```

BEGIN axi_ethernetlite
PARAMETER INSTANCE = Ethernet_Lite
PARAMETER HW_VER = 1.01.b
PARAMETER C_BASEADDR = 0x40e00000
PARAMETER C_HIGHADDR = 0x40e0fff
BUS_INTERFACE S_AXI = axi4lite_0
PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
PORT PHY_tx_en = Ethernet_Lite_TX_EN
PORT PHY_tx_clk = Ethernet_Lite_TX_CLK
PORT PHY_tx_data = Ethernet_Lite_TXD
PORT PHY_rx_er = Ethernet_Lite_RX_ER
PORT PHY_dv = Ethernet_Lite_RX_DV
PORT PHY_rx_clk = Ethernet_Lite_RX_CLK
PORT PHY_rx_data = Ethernet_Lite_RXD
PORT PHY_rst_n = Ethernet_Lite_PHY_RST_N
PORT PHY_MDIO = Ethernet_Lite_MDIO
PORT PHY_MDC = Ethernet_Lite_MDC
PORT PHY_crs = Ethernet_Lite_CRS
PORT PHY_col = Ethernet_Lite_COL
PORT IP2INTC_Irpt = Ethernet_Lite_IP2INTC_Irpt
END

```

```

BEGIN axi_gpio
PARAMETER INSTANCE = DIP_Switches_8Bits
PARAMETER HW_VER = 1.01.b
PARAMETER C_GPIO_WIDTH = 4
PARAMETER C_ALL_INPUTS = 1
PARAMETER C_INTERRUPT_PRESENT = 1
PARAMETER C_IS_DUAL = 0
PARAMETER C_BASEADDR = 0x40040000
PARAMETER C_HIGHADDR = 0x4004fff
BUS_INTERFACE S_AXI = axi4lite_0
PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
PORT GPIO_IO_I = DIP_Switches_TRI_I
PORT IP2INTC_Irpt = DIP_Switches_8Bits_IP2INTC_Irpt
END

```

```

BEGIN axi_7series_ddrx
PARAMETER INSTANCE = DDR3_SDRAM
PARAMETER HW_VER = 1.08.a
PARAMETER C_MEM_PARTNO = CUSTOM
PARAMETER C_INTERCONNECT_S_AXI_AR_REGISTER = 8
PARAMETER C_INTERCONNECT_S_AXI_AW_REGISTER = 8
PARAMETER C_INTERCONNECT_S_AXI_R_REGISTER = 8
PARAMETER C_INTERCONNECT_S_AXI_W_REGISTER = 8
PARAMETER C_INTERCONNECT_S_AXI_B_REGISTER = 8
PARAMETER C_DM_WIDTH = 1
PARAMETER C_DQS_WIDTH = 1
PARAMETER C_DQ_WIDTH = 8
PARAMETER C_ROW_WIDTH = 14

```

```

PARAMETER C_INTERCONNECT_S_AXI_MASTERS = microblaze_0.M_AXI_DC & microblaze_0.M_AXI_IC
PARAMETER C_MEM_BASEPARTNO = MT8JTF12864HZ-1G6
PARAMETER C_S_AXI_BASEADDR = 0xa8000000
PARAMETER C_S_AXI_HIGHADDR = 0xfffffff
BUS_INTERFACE S_AXI = axi4_0
PORT clk = clk_100_0000MHzPLLE0
PORT ddr_we_n = ddr_memory_we_n
PORT ddr_ras_n = ddr_memory_ras_n
PORT ddr_odt = ddr_memory_odt
PORT ddr_dqs_n = ddr_memory_dqs_n
PORT ddr_dqs_p = ddr_memory_dqs
PORT ddr_dq = ddr_memory_dq
PORT ddr_dm = ddr_memory_dm
PORT ddr_reset_n = ddr_memory_ddr3_rst
PORT ddr_cs_n = ddr_memory_cs_n
PORT ddr_ck_n = ddr_memory_clk_n
PORT ddr_ck_p = ddr_memory_clk
PORT ddr_cke = ddr_memory_cke
PORT ddr_cas_n = ddr_memory_cas_n
PORT ddr_ba = ddr_memory_ba
PORT ddr_addr = ddr_memory_addr
PORT sync_pulse = clk_25_0000MHz10PLLE0_nobuf
PORT mem_refclk = clk_400_0000MHzPLLE0_nobuf
PORT freq_refclk = clk_400_0000MHz1PLLE0_nobuf
PORT clk_ref = clk_200_0000MHzPLLE0
PORT pll_lock = proc_sys_reset_0_Dcm_locked
END

```

```

BEGIN cfifo
PARAMETER INSTANCE = cfifo_0
PARAMETER HW_VER = 1.00.a
PARAMETER C_INTERRUPT_PRESENT = 1
PARAMETER C_BASEADDR = 0x78400000
PARAMETER C_HIGHADDR = 0x7840ffff
BUS_INTERFACE S_AXI = axi4lite_0
PORT S_AXI_ACLK = clk_100_0000MHzPLLE0
PORT IP2INTC_Irpt = Cfifo_IP2INTC_Irpt
END

```

A.1.2 MSS

```
PARAMETER VERSION = 2.2.0
```

```

BEGIN OS
PARAMETER OS_NAME = standalone
PARAMETER OS_VER = 3.09.a
PARAMETER PROC_INSTANCE = microblaze_0
PARAMETER STDIN = debug_module
PARAMETER STDOUT = debug_module
END

```

```

BEGIN PROCESSOR
PARAMETER DRIVER_NAME = cpu
PARAMETER DRIVER_VER = 1.14.a

```

```
PARAMETER HW_INSTANCE = microblaze_0
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = tmrctr
PARAMETER DRIVER_VER = 2.04.a
PARAMETER HW_INSTANCE = axi_timer_0
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = axi_7series_ddrx
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = ddr3_sdram
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = uartlite
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = debug_module
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = gpio
PARAMETER DRIVER_VER = 3.00.a
PARAMETER HW_INSTANCE = dip_switches_8bits
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = emaclite
PARAMETER DRIVER_VER = 3.03.a
PARAMETER HW_INSTANCE = ethernet_lite
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = gpio
PARAMETER DRIVER_VER = 3.00.a
PARAMETER HW_INSTANCE = leds_8bits
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = bram
PARAMETER DRIVER_VER = 3.01.a
PARAMETER HW_INSTANCE = microblaze_0_d_bram_ctrl
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = bram
PARAMETER DRIVER_VER = 3.01.a
PARAMETER HW_INSTANCE = microblaze_0_i_bram_ctrl
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = intc
PARAMETER DRIVER_VER = 2.06.a
PARAMETER HW_INSTANCE = microblaze_0_intc
END
```

```
BEGIN DRIVER
  PARAMETER DRIVER_NAME = gpio
  PARAMETER DRIVER_VER = 3.00.a
  PARAMETER HW_INSTANCE = push_buttons_5bits
END
```

```
BEGIN DRIVER
  PARAMETER DRIVER_NAME = uartlite
  PARAMETER DRIVER_VER = 2.00.a
  PARAMETER HW_INSTANCE = rs232_uart_1
END
```

```
BEGIN DRIVER
  PARAMETER DRIVER_NAME = cfifo
  PARAMETER DRIVER_VER = 1.00.a
  PARAMETER HW_INSTANCE = cfifo_0
END
```

```
BEGIN LIBRARY
  PARAMETER LIBRARY_NAME = lwip140
  PARAMETER LIBRARY_VER = 1.04.a
  PARAMETER PROC_INSTANCE = microblaze_0
END
```

A.1.3 UCF

```
#
```

```
# pin constraints
```

```
#
```

```
NET CLK_N LOC = "AD11" | IOSTANDARD = "DIFF_SSTL15";
NET CLK_P LOC = "AD12" | IOSTANDARD = "DIFF_SSTL15";
NET DIP_Switches_TRI_I[0] LOC = "Y29" | IOSTANDARD = "LVCMOS25";
NET DIP_Switches_TRI_I[1] LOC = "W29" | IOSTANDARD = "LVCMOS25";
NET DIP_Switches_TRI_I[2] LOC = "AA28" | IOSTANDARD = "LVCMOS25";
NET DIP_Switches_TRI_I[3] LOC = "Y28" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_COL LOC = "W19" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_CRS LOC = "R30" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_MDC LOC = "R23" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_MDIO LOC = "J21" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_PHY_RST_N LOC = "L20" | IOSTANDARD = "LVCMOS25" | TIG;
NET Ethernet_Lite_RXD[0] LOC = "U30" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_RXD[1] LOC = "U25" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_RXD[2] LOC = "T25" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_RXD[3] LOC = "U28" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_RX_CLK LOC = "U27" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_RX_DV LOC = "R28" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_RX_ER LOC = "V26" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_TXD[0] LOC = "N27" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_TXD[1] LOC = "N25" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_TXD[2] LOC = "M29" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_TXD[3] LOC = "L28" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_TX_CLK LOC = "M28" | IOSTANDARD = "LVCMOS25";
NET Ethernet_Lite_TX_EN LOC = "M27" | IOSTANDARD = "LVCMOS25";
```

```

NET LEDs_8Bits_TRI_O[0] LOC = "AB8" | IOSTANDARD = "LVCMOS15";
NET LEDs_8Bits_TRI_O[1] LOC = "AA8" | IOSTANDARD = "LVCMOS15";
NET LEDs_8Bits_TRI_O[2] LOC = "AC9" | IOSTANDARD = "LVCMOS15";
NET LEDs_8Bits_TRI_O[3] LOC = "AB9" | IOSTANDARD = "LVCMOS15";
NET LEDs_8Bits_TRI_O[4] LOC = "AE26" | IOSTANDARD = "LVCMOS25";
NET LEDs_8Bits_TRI_O[5] LOC = "G19" | IOSTANDARD = "LVCMOS25";
NET LEDs_8Bits_TRI_O[6] LOC = "E18" | IOSTANDARD = "LVCMOS25";
NET LEDs_8Bits_TRI_O[7] LOC = "F16" | IOSTANDARD = "LVCMOS25";
NET Push_Buttons_5Bits_TRI_I[0] LOC = "G12" | IOSTANDARD = "LVCMOS25";
NET Push_Buttons_5Bits_TRI_I[1] LOC = "AC6" | IOSTANDARD = "LVCMOS15";
NET Push_Buttons_5Bits_TRI_I[2] LOC = "AB12" | IOSTANDARD = "LVCMOS15";
NET Push_Buttons_5Bits_TRI_I[3] LOC = "AG5" | IOSTANDARD = "LVCMOS15";
NET Push_Buttons_5Bits_TRI_I[4] LOC = "AA12" | IOSTANDARD = "LVCMOS15";
NET RESET LOC = "AB7" | IOSTANDARD = "LVCMOS15";
NET RS232_Uart_1_sin LOC = "M19" | IOSTANDARD = "LVCMOS25";
NET RS232_Uart_1_sout LOC = "K24" | IOSTANDARD = "LVCMOS25";
NET sm_fan_pwm_net_vcc LOC = "L26" | IOSTANDARD = "LVCMOS25";
# Added for RevC board
CONFIG DCI_CASCADE = "33 32 34";

```

A.2 Αρχεία Custom Περιφερειακού (VHDL)

A.2.1 CFIFO.VHDL

```

-----
-- cfifo.vhd - entity/architecture pair
-----
--
-----
-- Filename:      cfifo.vhd
-- Version:       1.00.a
-- Description:    Top level design, instantiates library components and user logic.
-- Date:          Wed May 22 18:17:20 2013
-- VHDL Standard: VHDL'93
-----
-- Naming Conventions:
-- active low signals:      "*_n"

-- clock signals:          "clk", "clk_div#", "clk_#x"
-- reset signals:          "rst", "rst_n"
-- generics:               "C_*"
-- user defined types:     "*_TYPE"
-- state machine next state: "*_ns"
-- state machine current state: "*_cs"
-- combinatorial signals:  "*_com"
-- pipelined or register delay signals: "*_d#"
-- counter signals:        "*cnt*"
-- clock enable signals:   "*_ce"
-- internal version of output port:   "*_j"
-- device pins:           "*_pin"
-- ports:                  "- Names begin with Uppercase"
-- processes:              "*_PROCESS"
-- component instantiations: "<ENTITY>_I_<#>|FUNC>"
-----

```

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

```

```

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;
use proc_common_v3_00_a.ipif_pkg.all;

```

```

library axi_lite_ipif_v1_01_a;
use axi_lite_ipif_v1_01_a.axi_lite_ipif;

```

```

library interrupt_control_v2_01_a;
use interrupt_control_v2_01_a.interrupt_control;

```

```

library cfifo_v1_00_a;
use cfifo_v1_00_a.user_logic;

```

```

-----
-- Entity section
-----

```

```

-- Definition of Generics:

```

```

-- C_S_AXI_DATA_WIDTH      -- AXI4LITE slave: Data width
-- C_S_AXI_ADDR_WIDTH      -- AXI4LITE slave: Address Width
-- C_S_AXI_MIN_SIZE        -- AXI4LITE slave: Min Size
-- C_USE_WSTRB              -- AXI4LITE slave: Write Strobe
-- C_DPHASE_TIMEOUT        -- AXI4LITE slave: Data Phase Timeout
-- C_BASEADDR              -- AXI4LITE slave: base address
-- C_HIGHADDR              -- AXI4LITE slave: high address
-- C_FAMILY                -- FPGA Family
-- C_NUM_REG                -- Number of software accessible registers
-- C_NUM_MEM                -- Number of address-ranges
-- C_SLV_AWIDTH            -- Slave interface address bus width
-- C_SLV_DWIDTH            -- Slave interface data bus width
--

```

```

-- Definition of Ports:

```

```

-- S_AXI_ACLK              -- AXI4LITE slave: Clock
-- S_AXI_ARESETN           -- AXI4LITE slave: Reset
-- S_AXI_AWADDR            -- AXI4LITE slave: Write address
-- S_AXI_AWVALID           -- AXI4LITE slave: Write address valid
-- S_AXI_WDATA             -- AXI4LITE slave: Write data
-- S_AXI_WSTRB             -- AXI4LITE slave: Write strobe
-- S_AXI_WVALID           -- AXI4LITE slave: Write data valid
-- S_AXI_BREADY            -- AXI4LITE slave: Response ready
-- S_AXI_ARADDR           -- AXI4LITE slave: Read address
-- S_AXI_ARVALID          -- AXI4LITE slave: Read address valid
-- S_AXI_RREADY           -- AXI4LITE slave: Read data ready
-- S_AXI_ARREADY          -- AXI4LITE slave: read address ready
-- S_AXI_RDATA            -- AXI4LITE slave: Read data
-- S_AXI_RRESP            -- AXI4LITE slave: Read data response

```

```

-- S_AXI_RVALID      -- AXI4LITE slave: Read data valid
-- S_AXI_WREADY      -- AXI4LITE slave: Write data ready
-- S_AXI_BRESP       -- AXI4LITE slave: Response
-- S_AXI_BVALID      -- AXI4LITE slave: Resonse valid
-- S_AXI_AWREADY     -- AXI4LITE slave: Wrt address ready
-- IP2INTC_Irpt      -- Interrupt output to processor
-----

```

entity cfifo is

generic

```

(
  C_FIFO_AWIDTH      : integer      = 8;
  C_FIFO_DWIDTH      : integer      := 32;
  -- Bus protocol parameters
  C_S_AXI_DATA_WIDTH  : integer      := 32;
  C_S_AXI_ADDR_WIDTH  : integer      := 32;
  C_S_AXI_MIN_SIZE    : std_logic_vector := X"000001FF";
  C_USE_WSTRB         : integer      := 0;
  C_DPHASE_TIMEOUT    : integer      := 8;
  C_BASEADDR          : std_logic_vector := X"FFFFFFFF";
  C_HIGHADDR          : std_logic_vector := X"00000000";
  C_FAMILY            : string        := "virtex6";
  C_NUM_REG           : integer      := 1;
  C_NUM_MEM           : integer      := 1;
  C_SLV_AWIDTH        : integer      := 32;
  C_SLV_DWIDTH        : integer      := 32;
  C_INTERRUPT_PRESENT : integer      := 1
);

```

port

```

(
  -- Bus protocol ports, do not add to or delete
  S_AXI_ACLK          : in std_logic;
  S_AXI_ARESETN       : in std_logic;
  S_AXI_AWADDR        : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  S_AXI_AWVALID       : in std_logic;
  S_AXI_WDATA         : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  S_AXI_WSTRB         : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
  S_AXI_WVALID       : in std_logic;
  S_AXI_BREADY        : in std_logic;
  S_AXI_ARADDR        : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  S_AXI_ARVALID       : in std_logic;
  S_AXI_RREADY        : in std_logic;
  S_AXI_ARREADY       : out std_logic;
  S_AXI_RDATA         : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  S_AXI_RRESP         : out std_logic_vector(1 downto 0);
  S_AXI_RVALID       : out std_logic;
  S_AXI_WREADY        : out std_logic;
  S_AXI_BRESP         : out std_logic_vector(1 downto 0);
  S_AXI_BVALID       : out std_logic;
  S_AXI_AWREADY       : out std_logic;
  IP2INTC_Irpt        : out std_logic
  -- DO NOT EDIT ABOVE THIS LINE -----
);

```

```

attribute MAX_FANOUT : string;
attribute SIGIS : string;
attribute MAX_FANOUT of S_AXI_ACLK : signal is "10000";
attribute MAX_FANOUT of S_AXI_ARESETN : signal is "10000";
attribute SIGIS of S_AXI_ACLK : signal is "Clk";
attribute SIGIS of S_AXI_ARESETN : signal is "Rst";
attribute SIGIS of IP2INTC_Irpt : signal is "INTR_LEVEL_HIGH";
end entity cfifo;

-----
-- Architecture section
-----

architecture IMP of cfifo is
    constant USER_SLV_DWIDTH : integer := C_S_AXI_DATA_WIDTH;
    constant IPIF_SLV_DWIDTH : integer := C_S_AXI_DATA_WIDTH;

    constant ZERO_ADDR_PAD : std_logic_vector(0 to 31) := (others => '0');
    constant USER_SLV_BASEADDR : std_logic_vector := C_BASEADDR or X"00000000";
    constant USER_SLV_HIGHADDR : std_logic_vector := C_BASEADDR or X"0000000F";
    constant INTR_BASEADDR : std_logic_vector := C_BASEADDR or X"00000100";
    constant INTR_HIGHADDR : std_logic_vector := C_BASEADDR or X"0000013F";
    constant IPIF_ARD_ADDR_RANGE_ARRAY : SLV64_ARRAY_TYPE :=
    (
        ZERO_ADDR_PAD & USER_SLV_BASEADDR, -- user logic slave space base address
        ZERO_ADDR_PAD & USER_SLV_HIGHADDR, -- user logic slave space high address
        ZERO_ADDR_PAD & INTR_BASEADDR, -- interrupt slave space base address
        ZERO_ADDR_PAD & INTR_HIGHADDR -- interrupt slave space high address
    );

    constant USER_SLV_NUM_REG : integer := 4;
    constant USER_NUM_REG : integer := USER_SLV_NUM_REG;
    constant INTR_NUM_CE : integer := 16;
    constant TOTAL_IPIF_CE : integer := USER_NUM_REG + INTR_NUM_CE;
    constant IPIF_ARD_NUM_CE_ARRAY : INTEGER_ARRAY_TYPE :=
    (
        0 => USER_SLV_NUM_REG, -- number of ce for user logic slave space
        1 => INTR_NUM_CE -- number of ce for interrupt control space
    );

    -----
    -- Number of device level interrupts
    -----
    constant INTR_NUM_IPIF_IRPT_SRC : integer := 4;

    -----
    -- Capture mode for each IP interrupt (generated by user logic)
    -- 1 = pass through (non-inverting)
    -- 2 = pass through (inverting)
    -- 3 = registered level (non-inverting)
    -- 4 = registered level (inverting)
    -- 5 = positive edge detect
    -- 6 = negative edge detect
    -----
    constant USER_NUM_INTR : integer := 2;

```

```

constant USER_INTR_CAPTURE_MODE      : integer      := 5;
constant INTR_IP_INTR_MODE_ARRAY      : INTEGER_ARRAY_TYPE :=
(
    0 => USER_INTR_CAPTURE_MODE,
    1 => USER_INTR_CAPTURE_MODE
);

-----
-- Device priority encoder feature inclusion/omission
-- true  = include priority encoder
-- false = omit priority encoder
-----

constant INTR_INCLUDE_DEV_PENCODER    : boolean      := false;

-----
-- Device ISC feature inclusion/omission
-- true  = include device ISC
-- false = omit device ISC
-----

constant INTR_INCLUDE_DEV_ISC         : boolean      := false;

-----
-- Index for CS/CE
-----

constant USER_SLV_CS_INDEX            : integer      := 0;
constant USER_SLV_CE_INDEX            : integer      :=
alc_start_ce_index(IPIF_ARD_NUM_CE_ARRAY, USER_SLV_CS_INDEX);
constant INTR_CS_INDEX                 : integer      := 1;
constant INTR_CE_INDEX                 : integer      :=
calc_start_ce_index(IPIF_ARD_NUM_CE_ARRAY, INTR_CS_INDEX);
constant USER_CE_INDEX                 : integer      := USER_SLV_CE_INDEX;

-----
-- IP Interconnect (IPIC) signal declarations
-----

signal ipif_Bus2IP_Clk                  : std_logic;
signal ipif_Bus2IP_Resetn               : std_logic;
signal ipif_Bus2IP_Addr                 : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
signal ipif_Bus2IP_RNW                  : std_logic;
signal ipif_Bus2IP_BE                   : std_logic_vector(IPIF_SLV_DWIDTH/8-1 downto 0);
signal ipif_Bus2IP_CS                   : std_logic_vector((IPIF_ARD_ADDR_RANGE_ARRAY'LENGTH)/2-1
downto 0);
signal ipif_Bus2IP_RdCE                  : std_logic_vector(0 to calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)-1);
signal ipif_Bus2IP_WrCE                  : std_logic_vector(0 to calc_num_ce(IPIF_ARD_NUM_CE_ARRAY)-1);
signal ipif_Bus2IP_Data                  : std_logic_vector(IPIF_SLV_DWIDTH-1 downto 0);
signal ipif_IP2Bus_WrAck                 : std_logic;
signal ipif_IP2Bus_RdAck                 : std_logic;
signal ipif_IP2Bus_Error                 : std_logic;
signal ipif_IP2Bus_Data                  : std_logic_vector(IPIF_SLV_DWIDTH-1 downto 0);
signal ipif_Bus2IP_Reset                 : std_logic;
signal intr_IPIF_Reg_Interrupts          : std_logic_vector(0 to 1);
signal intr_IPIF_Lvl_Interrupts          : std_logic_vector(0 to INTR_NUM_IPIF_IRPT_SRC-1);
signal intr_IP2Bus_Data                  : std_logic_vector(0 to IPIF_SLV_DWIDTH-1);
signal intr_IP2Bus_WrAck                 : std_logic;
signal intr_IP2Bus_RdAck                 : std_logic;
signal intr_IP2Bus_Error                 : std_logic;

```

```

signal user_Bus2IP_RdCE      : std_logic_vector(USER_NUM_REG-1 downto 0);
signal user_Bus2IP_WrCE      : std_logic_vector(USER_NUM_REG-1 downto 0);
signal user_IP2Bus_Data      : std_logic_vector(USER_SLV_DWIDTH-1 downto 0);
signal user_IP2Bus_RdAck     : std_logic;
signal user_IP2Bus_WrAck     : std_logic;
signal user_IP2Bus_Error     : std_logic;
signal user_IP2Bus_IntrEvent : std_logic_vector(0 to USER_NUM_INTR-1);

```

```
begin
```

```
-----
-- instantiate axi_lite_ipif
-----
```

```
AXI_LITE_IPIF_I : entity axi_lite_ipif_v1_01_a.axi_lite_ipif
```

```
generic map
```

```
(
    C_S_AXI_DATA_WIDTH      => IPIF_SLV_DWIDTH,
    C_S_AXI_ADDR_WIDTH      => C_S_AXI_ADDR_WIDTH,
    C_S_AXI_MIN_SIZE        => C_S_AXI_MIN_SIZE,
    C_USE_WSTRB              => C_USE_WSTRB,
    C_DPHASE_TIMEOUT        => C_DPHASE_TIMEOUT,
    C_AR_ADDR_RANGE_ARRAY   => IPIF_AR_ADDR_RANGE_ARRAY,
    C_AR_NUM_CE_ARRAY       => IPIF_AR_NUM_CE_ARRAY,
    C_FAMILY                 => C_FAMILY
)
```

```
port map
```

```
(
    S_AXI_ACLK              => S_AXI_ACLK,
    S_AXI_ARESETN           => S_AXI_ARESETN,
    S_AXI_AWADDR            => S_AXI_AWADDR,
    S_AXI_AWVALID           => S_AXI_AWVALID,
    S_AXI_WDATA             => S_AXI_WDATA,
    S_AXI_WSTRB             => S_AXI_WSTRB,
    S_AXI_WVALID            => S_AXI_WVALID,
    S_AXI_BREADY            => S_AXI_BREADY,
    S_AXI_ARADDR            => S_AXI_ARADDR,
    S_AXI_ARVALID           => S_AXI_ARVALID,
    S_AXI_RREADY            => S_AXI_RREADY,
    S_AXI_ARREADY           => S_AXI_ARREADY,
    S_AXI_RDATA             => S_AXI_RDATA,
    S_AXI_RRESP             => S_AXI_RRESP,
    S_AXI_RVALID            => S_AXI_RVALID,
    S_AXI_WREADY            => S_AXI_WREADY,
    S_AXI_BRESP             => S_AXI_BRESP,
    S_AXI_BVALID            => S_AXI_BVALID,
    S_AXI_AWREADY           => S_AXI_AWREADY,
    Bus2IP_Clk              => ipif_Bus2IP_Clk,
    Bus2IP_Resetn           => ipif_Bus2IP_Resetn,
    Bus2IP_Addr             => ipif_Bus2IP_Addr,
    Bus2IP_RNW              => ipif_Bus2IP_RNW,
    Bus2IP_BE               => ipif_Bus2IP_BE,
    Bus2IP_CS               => ipif_Bus2IP_CS,
    Bus2IP_RdCE             => ipif_Bus2IP_RdCE,
    Bus2IP_WrCE             => ipif_Bus2IP_WrCE,
    Bus2IP_Data             => ipif_Bus2IP_Data,

```

```

        IP2Bus_WrAck          => ipif_IP2Bus_WrAck,
        IP2Bus_RdAck          => ipif_IP2Bus_RdAck,
        IP2Bus_Error          => ipif_IP2Bus_Error,
        IP2Bus_Data           => ipif_IP2Bus_Data
    );

-----
-- instantiate Interrupt Controller
-----

    intr_IPIF_Reg_Interrupts(0) <= '0';
    intr_IPIF_Reg_Interrupts(1) <= '0';

    intr_IPIF_Lvl_Interrupts(0) <= '0';
    intr_IPIF_Lvl_Interrupts(1) <= '0';
    intr_IPIF_Lvl_Interrupts(2) <= '0';
    intr_IPIF_Lvl_Interrupts(3) <= '0';

    INTERRUPT_CONTROL_I : entity interrupt_control_v2_01_a.interrupt_control
    generic map
    (
        C_NUM_CE              => INTR_NUM_CE,
        C_NUM_IPIF_IRPT_SRC   => INTR_NUM_IPIF_IRPT_SRC,
        C_IP_INTR_MODE_ARRAY  => INTR_IP_INTR_MODE_ARRAY,
        C_INCLUDE_DEV_PENCODER => INTR_INCLUDE_DEV_PENCODER,
        C_INCLUDE_DEV_ISC     => INTR_INCLUDE_DEV_ISC,
        C_IPIF_DWIDTH         => IPIF_SLV_DWIDTH
    )
    port map
    (
        -- Inputs From the IPIF Bus
        Bus2IP_Clk             => ipif_Bus2IP_Clk,
        Bus2IP_Reset           => ipif_Bus2IP_Reset,
        Bus2IP_Data            => ipif_Bus2IP_Data,
        Bus2IP_BE              => ipif_Bus2IP_BE,
        Interrupt_RdCE         => ipif_Bus2IP_RdCE(INTR_CE_INDEX to
    INTR_CE_INDEX+INTR_NUM_CE-1),
        Interrupt_WrCE         => ipif_Bus2IP_WrCE(INTR_CE_INDEX to
    INTR_CE_INDEX+INTR_NUM_CE-1),

        -- Interrupt inputs from the IPIF sources that will
        -- get registered in this design
        IPIF_Reg_Interrupts    => intr_IPIF_Reg_Interrupts,

        -- Level Interrupt inputs from the IPIF sources
        IPIF_Lvl_Interrupts    => intr_IPIF_Lvl_Interrupts,

        -- Inputs from the IP Interface
        IP2Bus_IntrEvent        => user_IP2Bus_IntrEvent,

        -- Final Device Interrupt Output
        Intr2Bus_DevIntr        => IP2INTC_Irpt,

        -- Status Reply Outputs to the Bus
        Intr2Bus_DBus           => intr_IP2Bus_Data,
        Intr2Bus_WrAck          => intr_IP2Bus_WrAck,

```

```

    Intr2Bus_RdAck    => intr_IP2Bus_RdAck,
    Intr2Bus_Error    => intr_IP2Bus_Error,
    Intr2Bus_Retry     => open,
    Intr2Bus_ToutSup   => open
);

-----
-- instantiate User Logic
-----

USER_LOGIC_I : entity cfifo_v1_00_a.user_logic
generic map
(
    -- MAP USER GENERICS BELOW THIS LINE -----
    awidth => C_FIFO_AWIDTH,
    dwidth => C_FIFO_DWIDTH,
    -- MAP USER GENERICS ABOVE THIS LINE -----
    C_NUM_REG => USER_NUM_REG,
    C_SLV_DWIDTH => USER_SLV_DWIDTH,
    C_NUM_INTR => USER_NUM_INTR
)
port map
(
    Bus2IP_Clk => ipif_Bus2IP_Clk,
    Bus2IP_Resetn => ipif_Bus2IP_Resetn,
    Bus2IP_Addr => ipif_Bus2IP_Addr,
    Bus2IP_CS => ipif_Bus2IP_CS,
    Bus2IP_RNW => ipif_Bus2IP_RNW,
    Bus2IP_Data => ipif_Bus2IP_Data,
    Bus2IP_BE => ipif_Bus2IP_BE,
    Bus2IP_RdCE => user_Bus2IP_RdCE,
    Bus2IP_WrCE => user_Bus2IP_WrCE,
    IP2Bus_Data => user_IP2Bus_Data,
    IP2Bus_RdAck => user_IP2Bus_RdAck,
    IP2Bus_WrAck => user_IP2Bus_WrAck,
    IP2Bus_Error => user_IP2Bus_Error,
    IP2Bus_IntrEvent => user_IP2Bus_IntrEvent
);

-----
-- connect internal signals
-----

IP2BUS_DATA_MUX_PROC : process( ipif_Bus2IP_CS, user_IP2Bus_Data, intr_IP2Bus_Data ) is
begin
    case ipif_Bus2IP_CS is
        when "10" => ipif_IP2Bus_Data <= user_IP2Bus_Data;
        when "01" => ipif_IP2Bus_Data <= intr_IP2Bus_Data;
        when others => ipif_IP2Bus_Data <= (others => '0');
    end case;
end process IP2BUS_DATA_MUX_PROC;

ipif_IP2Bus_WrAck <= user_IP2Bus_WrAck or intr_IP2Bus_WrAck;
ipif_IP2Bus_RdAck <= user_IP2Bus_RdAck or intr_IP2Bus_RdAck;
ipif_IP2Bus_Error <= user_IP2Bus_Error or intr_IP2Bus_Error;
user_Bus2IP_RdCE <= ipif_Bus2IP_RdCE(USER_CE_INDEX to USER_CE_INDEX+USER_NUM_REG-1);
user_Bus2IP_WrCE <= ipif_Bus2IP_WrCE(USER_CE_INDEX to USER_CE_INDEX+USER_NUM_REG-1);

```

```

        ipif_Bus2IP_Reset <= not ipif_Bus2IP_Resetn;
end IMP;

```

A.2.2 USER_LOGIC.VHDL

```

-----
-- user_logic.vhd - entity/architecture pair
-----
--
-----
-- Filename:      user_logic.vhd
-- Version:       1.00.a
-- Description:    User logic.
-- Date:          Wed May 22 18:17:20 2013
-- VHDL Standard: VHDL'93
-----
-- Naming Conventions:
-- active low signals:      "*_n"
-- clock signals:           "clk", "clk_div#", "clk_#x"
-- reset signals:           "rst", "rst_n"
-- generics:                "C_*"
-- user defined types:      "*_TYPE"
-- state machine next state: "*_ns"
-- state machine current state: "*_cs"
-- combinatorial signals:   "*_com"
-- pipelined or register delay signals: "*_d#"
-- counter signals:         "**cnt*"
-- clock enable signals:    "*_ce"
-- internal version of output port:   "*_j"
-- device pins:            "*_pin"
-- ports:                  "- Names begin with Uppercase"
-- processes:              "*_PROCESS"
-- component instantiations: "<ENTITY>_I_<#>|FUNC>"
-----

```

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_misc.all;

```

```

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;

```

```

library cfifo_v1_00_a;
use cfifo_v1_00_a.ram;
use cfifo_v1_00_a.pointer;

```

```

-- Entity section
-----

```

```

-- Definition of Generics:
-- C_NUM_REG          -- Number of software accessible registers
-- C_SLV_DWIDTH       -- Slave interface data bus width
--
-- Definition of Ports:

```

```

-- Bus2IP_Clk          -- Bus to IP clock
-- Bus2IP_Resetn       -- Bus to IP reset
-- Bus2IP_Addr         -- Bus to IP address bus
-- Bus2IP_CS           -- Bus to IP chip select
-- Bus2IP_RNW          -- Bus to IP read/not write
-- Bus2IP_Data         -- Bus to IP data bus
-- Bus2IP_BE           -- Bus to IP byte enables
-- Bus2IP_RdCE         -- Bus to IP read chip enable
-- Bus2IP_WrCE         -- Bus to IP write chip enable
-- IP2Bus_Data         -- IP to Bus data bus
-- IP2Bus_RdAck        -- IP to Bus read transfer acknowledgement
-- IP2Bus_WrAck        -- IP to Bus write transfer acknowledgement
-- IP2Bus_Error        -- IP to Bus error response
-- IP2Bus_IntrEvent    -- IP to Bus interrupt events
-----

```

entity user_logic is

generic

(

```

    dwidth : integer := 8;
    awidth : integer := 8;
    -- Bus protocol parameters
    C_NUM_REG : integer := 4;
    C_SLV_DWIDTH : integer := 32;
    C_NUM_INTR : integer := 2

```

);

port

(

```

    -- Bus protocol ports
    Bus2IP_Clk : in std_logic;
    Bus2IP_Resetn : in std_logic;
    Bus2IP_Addr : in std_logic_vector(0 to 31);
    Bus2IP_CS : in std_logic_vector(0 to 1);
    Bus2IP_RNW : in std_logic;
    Bus2IP_Data : in std_logic_vector(C_SLV_DWIDTH-1 downto 0);
    Bus2IP_BE : in std_logic_vector(C_SLV_DWIDTH/8-1 downto 0);
    Bus2IP_RdCE : in std_logic_vector(C_NUM_REG-1 downto 0);
    Bus2IP_WrCE : in std_logic_vector(C_NUM_REG-1 downto 0);
    IP2Bus_Data : out std_logic_vector(C_SLV_DWIDTH-1 downto 0);
    IP2Bus_RdAck : out std_logic;
    IP2Bus_WrAck : out std_logic;
    IP2Bus_Error : out std_logic;
    IP2Bus_IntrEvent : out std_logic_vector(C_NUM_INTR-1 downto 0)

```

);

```

    attribute MAX_FANOUT : string;
    attribute SIGIS : string;
    attribute SIGIS of Bus2IP_Clk : signal is "CLK";
    attribute SIGIS of Bus2IP_Resetn : signal is "RST";

```

end entity user_logic;

-- Architecture section

architecture IMP of user_logic is

component pointer is

```
generic (width : integer := 8);
port(    clk : in std_logic;
        clken : in std_logic;
        rstn : in std_logic;
        dataout : out std_logic_vector(width-1 downto 0);
        rollover : out std_logic
    );
```

end component;

component ram is

```
generic (dwidth : integer := 8;
        awidth : integer := 8
    );
port (    clk : in std_logic;--clock
        we : in std_logic; --write enable
        wa : in std_logic_vector(awidth-1 downto 0);--write address
        ra : in std_logic_vector(awidth-1 downto 0);--read address
        datain : in std_logic_vector (dwidth-1 downto 0);--input data
        dataout : out std_logic_vector(dwidth-1 downto 0)--output data
    );
```

end component;

--inputs

signal datain_r : std_logic_vector(dwidth-1 downto 0) := (others => '0');

--outputs

signal dataout_r : std_logic_vector(dwidth-1 downto 0) := (others => '0');

--internal signals & registers

```
signal p_read_en : std_logic := '0';
signal p_readaddr : std_logic_vector(awidth-1 downto 0) := (others => '0');
signal p_readroll : std_logic := '0';
signal p_write_en : std_logic := '0';
signal p_writeaddr : std_logic_vector(awidth-1 downto 0) := (others => '0');
signal p_writeroll : std_logic := '0';
signal memwe : std_logic := '0'; -- memory write enable
signal memaddr : std_logic_vector(awidth-1 downto 0) := (others => '0');
signal memout : std_logic_vector(dwidth-1 downto 0) := (others => '0');
```

--high address to check for full

```
signal rollover : std_logic := '0';
signal fe_check : std_logic;
```

-- full/empty check

```
signal full_s : std_logic;
signal empty_s : std_logic;
signal error_s : std_logic;
```

--delays

```
type rdce_delay is array(0 to 2) of std_logic;
type wrce_delay is array(0 to 2) of std_logic;
signal rdce_delay_signal : rdce_delay;
signal wrce_delay_signal : wrce_delay;
```

```

begin
process(Bus2IP_Clk)
begin
    if rising_edge(Bus2IP_Clk) then
        if (Bus2IP_Resetn = '0') then
            for I in 0 to 2 loop
                rdce_delay_signal(i) <= '0';
                wrce_delay_signal(i) <= '0';
            end loop;
            datain_r <= (others => '0');
            dataout_r <= (others => '0');
        else
            rdce_delay_signal(0) <= Bus2IP_RdCE(C_NUM_REG-1);
            wrce_delay_signal(0) <= Bus2IP_WrCE(C_NUM_REG-1);

            for I in 1 to 2 loop
                rdce_delay_signal(i) <= rdce_delay_signal(i-1);
                wrce_delay_signal(i) <= wrce_delay_signal(i-1);
            end loop;

            datain_r <= Bus2IP_Data(dwidth-1 downto 0);
            dataout_r <= memout;
        end if;
    end if;
end process;

--read pointer
p_read_en <= Bus2IP_RdCE(C_NUM_REG-1) and not empty_s;
p_read : pointer
generic map
( width => awidth)
port map
( clk => Bus2IP_Clk,
  clken => p_read_en,
  rstn => Bus2IP_Resetn,
  dataout => p_readaddr,
  rollover => p_readroll);

--write pointer
p_write_en <= Bus2IP_WrCE(C_NUM_REG-1) and not full_s;
p_write : pointer
generic map
( width => awidth )
port map
( clk => Bus2IP_Clk,
  clken => p_write_en,
  rstn => Bus2IP_Resetn,
  dataout => p_writeaddr,
  rollover => p_writeroll );

rollover <= p_readroll xor p_writeroll;

```

```

-- memory
memwe <= Bus2IP_WrCE(C_NUM_REG-1) and not full_s;

mem : ram
    generic map
        ( awidth => awidth,
          dwidth => dwidth )
    port map
        ( clk => Bus2IP_Clk,
          we => memwe,
          wa => p_writeaddr,
          ra => p_readaddr,
          datain => datain_r,
          dataout => memout);

--full/empty check
fe_check <= and_reduce(p_writeaddr xnor p_readaddr); -- 1 same, 0 dif

--empty wp=rp and not rollover
empty_s <= fe_check and not rollover;

--full wp=rp and rollover
full_s <= fe_check and rollover;

--error
error_s <= (Bus2IP_RdCE(C_NUM_REG-1) and empty_s) or (Bus2IP_WrCE(C_NUM_REG-1) and full_s);

IP2Bus_Data(dwidth-1 downto 0) <= dataout_r;
IP2Bus_WrAck <= wrce_delay_signal(2);
IP2Bus_RdAck <= rdce_delay_signal(2);
IP2Bus_Error <= error_s;
IP2Bus_IntrEvent <= empty_s & full_s;
end IMP;

```

A.2.2 POINTER.VHDL

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_misc.all;

entity pointer is
    generic (width : integer := 8);
    port(
        clk : in std_logic;
        clken : in std_logic;
        rstn : in std_logic;
        dataout : out std_logic_vector(width-1 downto 0);
        rollover : out std_logic
    );
end pointer;

architecture behavioral of pointer is
    signal cnt : std_logic_vector(width-1 downto 0) := (others => '0');
    signal roll : std_logic;
    signal roll_reg : std_logic;
begin

```

```

process(clk)
begin
    if rising_edge(clk) then
        if (rstn = '0') then
            cnt <= (others => '0');
        elsif (clken = '1') then
            cnt <= cnt + 1;
        end if;
    end if;
end process;

roll <= and_reduce(cnt);

process(clk, rstn)
begin
    if (rstn = '0') then
        roll_reg <= '0';
    elsif rising_edge(clk) then
        if (clken = '1') then
            roll_reg <= (not roll and roll_reg) or (roll and not roll_reg);
        end if;
    end if;
end process;

rollover <= roll_reg;
dataout <= cnt;
end architecture;

```

A.2.3 RAM.VHDL

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ram is
    generic (awidth : integer := 8;
            dwidth : integer := 8
            );

    port (
        clk : in std_logic;--clock
        we : in std_logic;--write enable
        wa : in std_logic_vector(awidth-1 downto 0);--write address
        ra : in std_logic_vector(awidth-1 downto 0);--read address
        datain : in std_logic_vector(dwidth-1 downto 0);--input data
        dataout : out std_logic_vector(dwidth-1 downto 0);--output data
    );

end ram;

architecture behavioral of ram is
    type ram_type is array ((2**awidth-1) downto 0) of std_logic_vector(dwidth-1 downto 0);
    signal RAM : ram_type;

begin
    process (clk)
    begin

```

```

        if rising_edge(clk) then
            if (we = '1') then
                RAM(conv_integer(wa)) <= datain;
            end if;
        end if;
    end process;

    dataout <= RAM(conv_integer(ra));
end architecture;

```

A.3 Αρχεία Custom Περιφερειακού (Drivers)

A.3.1 cfifo.h

```

/*****
* Filename:      /home/skordipan/ISEProjects/microblaze/system/drivers/cfifo_v1_00_a/src/cfifo.h
* Version:      1.00.a
* Description:   cfifo Driver Header File
* Date:        Tue Jun  4 00:09:03 2013
*****/
/*****/
/**
 * @file cfifo.h
 *
 * This file contains the software API definition of the Cfifo Custom Peripheral
 *
 * <b>Initialization & Configuration</b>
 *
 * The CFifo_Config structure is used by the driver to configure itself. This
 * configuration structure is typically created by the tool-chain based on HW
 * build properties.
 *
 * To support multiple runtime loading and initialization strategies employed
 * by various operating systems, the driver instance can be initialized in one
 * of the following ways:
 *
 * - CFifo_Initialize(InstancePtr, DeviceId) - The driver looks up its own
 * configuration structure created by the tool-chain based on an ID provided
 * by the tool-chain.
 *
 * - CFifo_CfgInitialize(InstancePtr, CfgPtr, EffectiveAddr) - Uses a
 * configuration structure provided by the caller. If running in a system
 * with address translation, the provided virtual memory base address
 * replaces the physical address present in the configuration structure.
 *
 *
 * @note
 *
 * This API utilizes 32 bit I/O to the CFIFO registers. With less than 32 bits,
 * the unused bits from registers are read as zero and written as don't cares.
 *
 * <pre>
 * MODIFICATION HISTORY:
 * </pre>

```

```

*****/

#ifndef CFIFO_H
#define CFIFO_H

#ifdef __cplusplus
extern "C" {
#endif
/***** Include Files *****/

#include <xil_types.h>
#include <xil_assert.h>
#include <xstatus.h>
#include <cfifo_l.h>

/***** Constant Definitions *****/

/***** Type Definitions *****/

/**
 * This typedef contains configuration information for the device.
 */
typedef struct {
    u16 DeviceId; /* Unique ID of device */
    u32 BaseAddress; /* Device base address */
    int InterruptPresent; /* Are interrupts supported in h/w */
} CFifo_Config;

/**
 * The CFifo driver instance data. The user is required to allocate a
 * variable of this type for every CFIFO device in the system. A pointer
 * to a variable of this type is then passed to the driver API functions.
 */
typedef struct {
    u32 BaseAddress; /* Device base address */
    u32 IsReady; /* Device is initialized and ready */
    int InterruptPresent; /* Are interrupts supported in h/w */
} CFifo;

/***** Macros (Inline Functions) Definitions *****/
/***** Function Prototypes *****/
/*
 * Initialization functions in cfifo_sinit.c
 */
int CFifo_Initialize(CFifo *InstancePtr, u16 DeviceId);
CFifo_Config *CFifo_LookupConfig(u16 DeviceId);

/*
 * API Basic functions implemented in cfifo.c
 */
int CFifo_CfgInitialize(CFifo *InstancePtr, CFifo_Config * Config,
    u32 EffectiveAddr);
u32 CFifo_Read(CFifo *InstancePtr);

```

```

void CFifo_Write(CFifo *InstancePtr, u32 Mask);

/*
 * API Functions implemented in cfifo_selftest.c
 */
XStatus CFifo_SelfTest(CFifo *InstancePtr);

/*
 * API Functions implemented in cfifo_intr.c
 */
void CFifo_InterruptGlobalEnable(CFifo *InstancePtr);
void CFifo_InterruptGlobalDisable(CFifo *InstancePtr);
void CFifo_InterruptEnable(CFifo *InstancePtr, u32 Mask);
void CFifo_InterruptDisable(CFifo *InstancePtr, u32 Mask);
void CFifo_InterruptClear(CFifo *InstancePtr, u32 Mask);
u32 CFifo_InterruptGetEnabled(CFifo *InstancePtr);
u32 CFifo_InterruptGetStatus(CFifo *InstancePtr);

#ifdef __cplusplus
}
#endif
#endif /** CFIFO_H */

```

A.3.2 cfifo.c

```

/*****
 * Filename:      /home/skordipan/ISEProjects/microblaze/system/drivers/cfifo_v1_00_a/src/cfifo.c
 * Version:      1.00.a
 * Description:   cfifo Driver Source File
 * Date:         Tue Jun  4 00:09:03 2013 (by Create and Import Peripheral Wizard)
 *****/
/**
 * @file cfifo.c
 *
 * The implementation of the CFifo driver's basic functionality. See cfifo.h
 * for more information about the driver.
 *
 * @note
 *
 * None
 *
 * <pre>
 * MODIFICATION HISTORY:
 *
 * </pre>
 */
*****/

/***** Include Files *****/
#include "cfifo.h"
#include <xstatus.h>

/***** Constant Definitions *****/
/***** Type Definitions *****/
/***** Macros (Inline Functions) Definitions *****/

```

```

/***** Variable Definitions *****/
/***** Function Prototypes *****/

/***** Function Definitions *****/
/**
 * Initialize the CFifo instance provided by the caller based on the
 * given configuration data.
 *
 * Nothing is done except to initialize the InstancePtr.
 *
 * @param InstancePtr is a pointer to an CFifo instance. The memory the
 * pointer references must be pre-allocated by the caller. Further
 * calls to manipulate the driver through the CFifo API must be
 * made with this pointer.
 * @param Config is a reference to a structure containing information
 * about a specific CFifo device. This function initializes an
 * InstancePtr object for a specific device specified by the
 * contents of Config. This function can initialize multiple
 * instance objects with the use of multiple calls giving different
 * Config information on each call.
 * @param EffectiveAddr is the device base address in the virtual memory
 * address space. The caller is responsible for keeping the address
 * mapping from EffectiveAddr to the device physical base address
 * unchanged once this function is invoked. Unexpected errors may
 * occur if the address mapping changes after this function is
 * called. If address translation is not used, use
 * Config->BaseAddress for this parameters, passing the physical
 * address instead.*
 *
 * @return
 * - XST_SUCCESSInitialization was successfull.
 *
 * @note None.
 */
*****/
int CFifo_CfgInitialize(CFifo *InstancePtr, CFifo_Config * Config,
                       u32 EffectiveAddr)
{
    /**
     * Assert arguments
     */
    Xil_AssertNonvoid(InstancePtr != NULL);

    /**
     * Set some default values.
     */
    #if (XPAR_CFIFO_USE_DCR_BRIDGE != 0)
        InstancePtr->BaseAddress = ((EffectiveAddr >> 2)) & 0xFFF;
    #else
        InstancePtr->BaseAddress = EffectiveAddr;
    #endif

    InstancePtr->InterruptPresent = Config->InterruptPresent;

```

```

    /*
     * Indicate the instance is now ready to use, initialized without error
     */
    InstancePtr->IsReady = XIL_COMPONENT_IS_READY;
    return (XST_SUCCESS);
}

/*****

/**
 * Read from CFifo.
 *
 * @param      InstancePtr is a pointer to an CFifo instance to be worked on.
 *
 * @return      Current copy of the discretes register.
 *
 * @note
 *
 *****/
u32 CFifo_Read(CFifo *InstancePtr)
{
    Xil_AssertNonvoid(InstancePtr != NULL);
    Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);

    return CFifo_ReadReg(InstancePtr->BaseAddress,
                        0x0);
}

/*****

/**
 * Write to CFifo
 *
 * @param      InstancePtr is a pointer to an CFifo instance to be worked on.
 * @param      Data is the value to be written to the discretes register.
 *
 * @return      None.
 *
 * @note
 *
 *****/
void CFifo_Write(CFifo *InstancePtr, u32 Mask)
{
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    CFifo_WriteReg(InstancePtr->BaseAddress,
                    0x0,
                    Mask);
}

```

A.3.3 cfifo_l.h

```

/*****

/**
 * @file cfifo_l.h
 *

```

```

* <pre>
* MODIFICATION HISTORY:
* </pre>
*
*****/

#ifndef CFIFO_L_H                /* prevent circular inclusions */
#define CFIFO_L_H                /* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif
/***** Include Files *****/
#include <xil_types.h>
#include <xil_assert.h>
#include <xil_io.h>

/*
 * XPAR_CFIFO_USE_DCR_BRIDGE has to be set to 1 if the CFIFO device is
 * accessed through a DCR bus connected to a bridge
 */
#define XPAR_CFIFO_USE_DCR_BRIDGE 0

#if (XPAR_CFIFO_USE_DCR_BRIDGE != 0)
#include "xio_dcr.h"
#endif

/***** Constant Definitions *****/

/** @name Registers
 *
 * Register offsets for this device.
 * @{
 */
#if (XPAR_CFIFO_USE_DCR_BRIDGE != 0)

#define CFIFO_DATA_OFFSET    0x0  /**< Data register/channel for CFIFO */

#define CFIFO_GIE_OFFSET    0x47  /**< Global interrupt enable register */
#define CFIFO_ISR_OFFSET    0x48  /**< Interrupt status register */
#define CFIFO_IER_OFFSET    0x4A  /**< Interrupt enable register */

#else

#define CFIFO_DATA_OFFSET    0x0  /**< Data register/channel for CFIFO */

#define CFIFO_GIE_OFFSET    0x11C  /**< Global interrupt enable register */
#define CFIFO_ISR_OFFSET    0x120  /**< Interrupt status register */
#define CFIFO_IER_OFFSET    0x128  /**< Interrupt enable register */

#endif

/** @} */

```

```

/** @name Interrupt Status and Enable Register bitmaps and masks
 *
 * Bit definitions for the interrupt status register and interrupt enable
 * registers.
 * @{
 */
#define CFIFO_IR_EMPTY_MASK 0x1 /** Mask empty interrupt bit */
#define CFIFO_IR_FULL_MASK 0x2 /** Mask full interrupt bit */

#define CFIFO_IR_MASK 0x3 /** Mask of all bits */

/* @} */

/** @name Global Interrupt Enable Register bitmaps and masks
 *
 * Bit definitions for the Global Interrupt Enable register
 * @{
 */

#define CFIFO_GIE_GINTR_ENABLE_MASK 0x80000000

/* @} */

/***** Type Definitions *****/

/***** Macros (Inline Functions) Definitions *****/

/*
 * Define the appropriate I/O access method to memory mapped I/O or DCR.
 */

#if (XPAR_CFIFO_USE_DCR_BRIDGE != 0)

#define CFifo_In32 XIo_DcrIn
#define CFifo_Out32 XIo_DcrOut

#else

#define CFifo_In32 Xil_In32
#define CFifo_Out32 Xil_Out32

#endif

/*****
 **
 *
 * Write a value to a CFIFO register. A 32 bit write is performed. If the
 * CFIFO core is implemented in a smaller width, only the least significant data
 * is written.
 *
 * @param BaseAddress is the base address of the CFIFO device.
 * @param RegOffset is the register offset from the base to write to.
 * @param Data is the data written to the register.
 */

```

```

* @return      None.
*
* @note        C-style signature:
*               void CFifo_WriteReg(u32 BaseAddress, u32 RegOffset, u32 Data)
*
*****/

#define CFifo_WriteReg(BaseAddress, RegOffset, Data) \
    CFifo_Out32((BaseAddress) + (RegOffset), (u32)(Data))

/*****
**
* Read a value from a CFIFO register. A 32 bit read is performed. If the
* CFIFO core is implemented in a smaller width, only the least
* significant data is read from the register. The most significant data
* will be read as 0.
*
* @param        BaseAddress is the base address of the CFIFO device.
* @param        RegOffset is the register offset from the base to read from.
*
* @return        Data read from the register.
*
* @note        C-style signature:
*               u32 CFifo_ReadReg(u32 BaseAddress, u32 RegOffset)
*
*****/

#define CFifo_ReadReg(BaseAddress, RegOffset) \
    CFifo_In32((BaseAddress) + (RegOffset))

/***** Function Prototypes *****/
/***** Variable Definitions *****/
#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */

```

A.3.4 cfifo_i.h

```

/*****
**
* @file cfifo_i.h
*
* This header file contains internal identifiers, which are those shared
* between the files of the driver. It is intended for internal use only.
*
* NOTES:
*
* None.
*
* <pre>
* MODIFICATION HISTORY:

```

```

* </pre>
*****/

#ifndef CFIFO_I_H          /* prevent circular inclusions */
#define CFIFO_I_H          /* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif

/***** Include Files *****/

#include "cfifo.h"

/***** Constant Definitions *****/

/***** Type Definitions *****/

/***** Macros (Inline Functions) Definitions *****/

/***** Function Prototypes *****/

/***** Variable Definitions *****/

extern CFifo_Config CFifo_ConfigTable[];

#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */

```

A.3.5 cfifo_g.c

```

/*****/
/**
 *
 * @file cfifo_g.c
 *
 * This file contains a configuration table that specifies the configuration
 * of CFIFO devices in the system.
 *
 * <pre>
 * MODIFICATION HISTORY:
 * </pre>
 *
 *****/

/***** Include Files *****/

#include "cfifo.h"

#include <xparameters.h>

/***** Constant Definitions *****/

/***** Type Definitions *****/

```

```
/****** Macros (Inline Functions) Definitions *****/
```

```
/****** Function Prototypes *****/
```

```
/****** Variable Prototypes *****/
```

```
/**
```

```
 * This table contains configuration information for each CFIFO device  
 * in the system.
```

```
*/
```

```
CFifo_Config CFifo_ConfigTable[] = {  
    {  
        XPAR_CFIFO_0_DEVICE_ID,  
        XPAR_CFIFO_0_BASEADDR,  
        XPAR_CFIFO_0_INTERRUPT_PRESENT}  
};
```

A.3.6 cfifo_sinit.c

```
/****** *****/
```

```
/**
```

```
 * @file cfifo_sinit.c
```

```
 *
```

```
 * The implementation of the CFifo driver's static initialization  
 * functionality.
```

```
 *
```

```
 * @note
```

```
 *
```

```
 * None
```

```
 *
```

```
 * <pre>
```

```
 * MODIFICATION HISTORY:
```

```
 * </pre>
```

```
 *
```

```
*****
```

```
/****** Include Files *****/
```

```
#include <xstatus.h>
```

```
#include <xparameters.h>
```

```
#include "cfifo_i.h"
```

```
/****** Constant Definitions *****/
```

```
/****** Type Definitions *****/
```

```
/****** Macros (Inline Functions) Definitions *****/
```

```
/****** Variable Definitions *****/
```

```
/****** Function Prototypes *****/
```

```
*****
```

```
/**
```

```
 * Lookup the device configuration based on the unique device ID. The table
```

```

* ConfigTable contains the configuration info for each device in the system.
*
* @param      DeviceId is the device identifier to lookup.
*
* @return
*
*             - A pointer of data type CFifo_Config which points to the
*             device configuration if DeviceID is found.
*             - NULL if DeviceID is not found.
*
* @note      None.
*
*****/

CFifo_Config *CFifo_LookupConfig(u16 DeviceId)
{
    CFifo_Config *CfgPtr = NULL;
    int Index;
    for (Index = 0; Index < XPAR_CFIFO_NUM_INSTANCES; Index++) {
        if (CFifo_ConfigTable[Index].DeviceId == DeviceId) {
            CfgPtr = &CFifo_ConfigTable[Index];
            break;
        }
    }
    return CfgPtr;
}

/*****
* Initialize the CFifo instance provided by the caller based on the
* given DeviceID.
*
* Nothing is done except to initialize the InstancePtr.
*
* @param      InstancePtr is a pointer to an CFifo instance. The memory the
*             pointer references must be pre-allocated by the caller. Further
*             calls to manipulate the instance/driver through the CFifo API
*             must be made with this pointer.
* @param      DeviceId is the unique id of the device controlled by this CFifo
*             instance. Passing in a device id associates the generic CFifo
*             instance to a specific device, as chosen by the caller or
*             application developer.
*
* @return
*
*             - XST_SUCCESS if the initialization was successfull.
*             - XST_DEVICE_NOT_FOUND if the device configuration data was not
*             found for a device with the supplied device ID.
*
* @note      None.
*
*****/

int CFifo_Initialize(CFifo * InstancePtr, u16 DeviceId)
{
    CFifo_Config *ConfigPtr;

```

```

/*
 * Assert arguments
 */

Xil_AssertNonvoid(InstancePtr != NULL);

/*
 * Lookup configuration data in the device configuration table.
 * Use this configuration info down below when initializing this
 * driver.
 */

ConfigPtr = CFifo_LookupConfig(DeviceId);
if (ConfigPtr == (CFifo_Config *) NULL) {
    InstancePtr->IsReady = 0;
    return (XST_DEVICE_NOT_FOUND);
}

return CFifo_CfgInitialize(InstancePtr, ConfigPtr,
                          ConfigPtr->BaseAddress);
}

```

A.3.7 cfifo_intr.c

```

/*****
**
* @file cfifo_intr.c
*
* Implements CFIFO interrupt processing functions for the CFIFO driver.
* See cfifo.h for more information about the driver.
*
* The functions in this file require the hardware device to be built with
* interrupt capabilities. The functions will assert if called using hardware
* that does not have interrupt capabilities.
*
* <pre>
* MODIFICATION HISTORY:
* </pre>
*
*****/

/***** Include Files *****/

#include "cfifo.h"

/***** Constant Definitions *****/

/***** Type Definitions *****/

/***** Macros (Inline Functions) Definitions *****/

/***** Variable Definitions *****/

/***** Function Prototypes *****/

*****/

```

```

/**
 * Enable the interrupt output signal. Interrupts enabled through
 * CFifo_InterruptEnable() will not be passed through until the global enable
 * bit is set by this function. This function is designed to allow all
 * interrupts to be enabled easily for exiting a critical
 * section. This function will assert if the hardware device has not been
 * built with interrupt capabilities.
 *
 * @param      InstancePtr is the CFIFO instance to operate on.
 *
 * @return      None.
 *
 * @note      None.
 */
*****/

void CFifo_InterruptGlobalEnable(CFifo * InstancePtr)
{
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertVoid(InstancePtr->InterruptPresent == TRUE);

    CFifo_WriteReg(InstancePtr->BaseAddress, CFIFO_GIE_OFFSET,
                   CFIFO_GIE_GINTR_ENABLE_MASK);
}

/*****/
/**
 * Disable the interrupt output signal. Interrupts enabled through
 * CFifo_InterruptEnable() will no longer be passed through until the global
 * enable bit is set by CFifo_InterruptGlobalEnable(). This function is
 * designed to allow all interrupts to be disabled easily for
 * entering a critical section. This function will assert if the hardware
 * device has not been built with interrupt capabilities.
 *
 * @param      InstancePtr is the CFIFO instance to operate on.
 *
 * @return      None.
 *
 * @note      None.
 */
*****/

void CFifo_InterruptGlobalDisable(CFifo * InstancePtr)
{
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertVoid(InstancePtr->InterruptPresent == TRUE);

    CFifo_WriteReg(InstancePtr->BaseAddress, CFIFO_GIE_OFFSET, 0x0);
}

/*****/
/**
 * Enable interrupts. The global interrupt must also be enabled by calling
 * CFifo_InterruptGlobalEnable() for interrupts to occur. This function will

```

```

* assert if the hardware device has not been built with interrupt capabilities.
*
* @param InstancePtr is the CFIFO instance to operate on.
* @param Mask is the mask to enable. Bit positions of 1 are enabled.
* This mask is formed by OR'ing bits from CFIFO_IR* bits which
* are contained in cfifo_l.h.
*
* @return None.
*
* @note None.
*
*****/

void CFifo_InterruptEnable(CFifo * InstancePtr, u32 Mask)
{
    u32 Register;
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertVoid(InstancePtr->InterruptPresent == TRUE);

    /*
     * Read the interrupt enable register and only enable the specified
     * interrupts without disabling or enabling any others.
     */

    Register = CFifo_ReadReg(InstancePtr->BaseAddress, CFIFO_IER_OFFSET);
    CFifo_WriteReg(InstancePtr->BaseAddress, CFIFO_IER_OFFSET,
        Register | Mask);
}

/*****
**
* Disable interrupts. This function allows specific interrupts
* to be disabled. This function will assert if the hardware device
* has not been built with interrupt capabilities.
*
* @param InstancePtr is the CFIFO instance to operate on.
* @param Mask is the mask to disable. Bits set to 1 are disabled. This
* mask is formed by OR'ing bits from CFIFO_IR* bits which are
* contained in cfifo_l.h.
*
* @return None.
*
* @note None.
*
*****/

void CFifo_InterruptDisable(CFifo * InstancePtr, u32 Mask)
{
    u32 Register;
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertVoid(InstancePtr->InterruptPresent == TRUE);

```

```

    /*
    * Read the interrupt enable register and only disable the specified
    * interrupts without enabling or disabling any others.
    */

    Register = CFifo_ReadReg(InstancePtr->BaseAddress, CFIFO_IER_OFFSET);
    CFifo_WriteReg(InstancePtr->BaseAddress, CFIFO_IER_OFFSET,
        Register & (~Mask));

}

/*****
**
* Clear pending interrupts with the provided mask. This function should be
* called after the software has serviced the interrupts that are pending.
* This function will assert if the hardware device has not been built with
* interrupt capabilities.
*
* @param InstancePtr is the CFIFO instance to operate on.
* @param Mask is the mask to clear pending interrupts for. Bit positions
* of 1 are cleared. This mask is formed by OR'ing bits from
* CFIFO_IR* bits which are contained in cfifo_l.h.
*
* @return None.
*
* @note None.
*
*****/
void CFifo_InterruptClear(CFifo * InstancePtr, u32 Mask)
{
    u32 Register;
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertVoid(InstancePtr->InterruptPresent == TRUE);

    /*
    * Read the interrupt status register and only clear the interrupts
    * that are specified without affecting any others. Since the register
    * is a toggle on write, make sure any bits to be written are already
    * set.
    */

    Register = CFifo_ReadReg(InstancePtr->BaseAddress, CFIFO_ISR_OFFSET);
    CFifo_WriteReg(InstancePtr->BaseAddress, CFIFO_ISR_OFFSET,
        Register & Mask);
}

/*****
**
* Returns the interrupt enable mask. This function will assert if the
* hardware device has not been built with interrupt capabilities.
*
* @param InstancePtr is the CFIFO instance to operate on.
*
* @return A mask of bits made from CFIFO_IR* bits which are contained in

```

```

*          cfifo_l.h.
*
* @return    None.
*
* @note      None.
*
*****/
u32 CFifo_InterruptGetEnabled(CFifo * InstancePtr)
{
    Xil_AssertNonvoid(InstancePtr != NULL);
    Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertNonvoid(InstancePtr->InterruptPresent == TRUE);

    return CFifo_ReadReg(InstancePtr->BaseAddress, CFIFO_IER_OFFSET);
}

/*****/
/**
 * Returns the status of interrupt signals. Any bit in the mask set to 1
 * indicates that the channel associated with the bit has asserted an interrupt
 * condition. This function will assert if the hardware device has not been
 * built with interrupt capabilities.
 *
 * @param      InstancePtr is the CFIFO instance to operate on.
 *
 * @return      A pointer to a mask of bits made from CFIFO_IR* bits which are
 *              contained in cfifo_l.h.
 *
 * @note
 *
 * The interrupt status indicates the status of the device irregardless if
 * the interrupts from the devices have been enabled or not through
 * CFifo_InterruptEnable().
 *
*****/
u32 CFifo_InterruptGetStatus(CFifo * InstancePtr)
{
    Xil_AssertNonvoid(InstancePtr != NULL);
    Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Xil_AssertNonvoid(InstancePtr->InterruptPresent == TRUE);

    return CFifo_ReadReg(InstancePtr->BaseAddress, CFIFO_ISR_OFFSET);
}

```

A.3.8 cfifo_selftest.c

```

/*****/
* Filename:      /home/skordipan/ISEProjects/microblaze/system/drivers/cfifo_v1_00_a/src/cfifo_selftest.c
* Version:      1.00.a
* Description:   Contains a diagnostic self-test function for the cfifo driver
* Date:         Tue Jun 4 00:09:03 2013
*****/

/***** Include Files *****/

```

```

#include "cfifo.h"
#include <stdio.h>
#include <xio.h>
#include <xparameters.h>

/***** Constant Definitions *****/

/***** Variable Definitions *****/

/***** Function Definitions *****/

/**
 *
 * Run a self-test on the driver/device. Note this may be a destructive test if
 * resets of the device are performed.
 *
 * If the hardware system is not built correctly, this function may never
 * return to the caller.
 *
 * @param InstancePtr is a pointer to an CFifo instance. The memory the
 * pointer references must be pre-allocated by the caller. Further
 * calls to manipulate the driver through the CFifo API must be
 * made with this pointer.
 *
 * @return
 *
 * - XST_SUCCESS if all self-test code passed
 * - XST_FAILURE if any self-test code failed
 *
 * @note Caching must be turned off for this function to work.
 * @note Self test may fail if data memory and device are not on the same bus.
 */

XStatus CFifo_SelfTest(CFifo *InstancePtr)
{
    CFifo_Write(InstancePtr, 0xDEADBEEF);
    if(0xDEADBEEF == CFifo_Read(InstancePtr)) {
        return XST_SUCCESS;
    }

    return XST_FAILURE;
}

```

Παράρτημα Β : Εργαλεία Ανάπτυξης και Υλοποίησης

Επειδή τα εργαλεία της Xilinx αλλάζουν αρκετά από έκδοση σε έκδοση, και είναι πολύ πιθανό να έχουν αλλάξει σε μεγάλο βαθμό σε επόμενες εκδόσεις, η όλη διαδικασία υλοποίησης του συστήματος περιγράφεται στο παράρτημα Β που ακολουθεί.

Τα εργαλεία που χρησιμοποιούνται είναι το ISE, το EDK και το SDK τα οποία υλοποιούν διαφορετικές λειτουργίες το καθένα. Στην πράξη και τα τρία εργαλεία είναι ολοκληρωμένα IDEs τα οποία καλούν άλλα κατάλληλα εργαλεία, παρέχοντας την σωστή είσοδο με βάση το design που έχουμε δημιουργήσει και τα options τα οποία έχουμε επιλέξει. Η κλήση των επιμέρους εργαλείων γίνεται με την κατάλληλη σειρά, ώστε να επιτευχθεί η σωστή υλοποίηση του συστήματος.

B.1 Integrated Software Environment (ISE)

Το ISE είναι το βασικό εργαλείο σχεδίασης και υλοποίησης συστημάτων στην πλατφόρμα της Xilinx. Στην περίπτωση ενός ενσωματωμένου συστήματος δεν αποτελεί εξαίρεση, και η όλη διαδικασία ξεκινάει με την δημιουργία ενός καινούργιου Project. Κατά την δημιουργία του project επιλέγουμε το ολοκληρωμένο για το οποίο θα υλοποιηθεί το σύστημα. Μπορούμε να επιλέξουμε επίσης ένα evaluation board και το tool θα επιλέξει τις κατάλληλες τιμές στα υπόλοιπα πεδία. Στην περίπτωση μας επιλέγουμε το evaluation board KC705, το οποίο περιέχει το ολοκληρωμένο XC7K325T-2FFG900C.

Αφού ολοκληρωθεί η διαδικασία δημιουργίας του καινούργιου project, αυτό το project είναι άδειο. Επιλέγουμε την δημιουργία νέου αρχείου (New Source), από την λίστα επιλέγουμε το embedded processor και συμπληρώνουμε ένα όνομα, π.χ. system. Αφού ολοκληρωθεί η διαδικασία θα εκκινήσει αυτόματα το EDK, το οποίο θα περιγράψουμε στην συνέχεια.

Αν και το EDK μας δίνει την δυνατότητα να δημιουργήσουμε και να διαχειριστούμε έναν σύστημα χωρίς την βοήθεια του ISE, η χρήση του μας δίνει την δυνατότητα να προσθέσουμε components στο design, τα οποία θα λειτουργούν παράλληλα με τον επεξεργαστή αλλά δεν χρειάζεται να έχουν άμεση επικοινωνία με αυτόν.

B.2 Embedded Development Kit (EDK)

Με την δημιουργία του embedded processor source στο ISE, θα εκκινήσει αυτόματα το EDK, μέσω του οποίου μπορούμε να δημιουργήσουμε ένα αρχικό σύστημα και στην συνέχεια να κάνουμε τις απαραίτητες παραμετροποιήσεις ή/και προσθέσεις, ώστε να έχουμε ένα πλήρες σύστημα, το οποίο θα είναι ικανό στην συνέχεια να προσφέρει την λειτουργικότητα που επιθυμούμε.

Στην περίπτωση που έχουμε επιλέξει κάποιο evaluation board, αρκετά από τα απαραίτητα αρχεία παράγονται αυτόματα. Επίσης ο wizard δημιουργίας νέου συστήματος έχει προεπιλεγμένες αρκετές επιλογές.

Μέσω του wizard μπορούμε να επιλέξουμε το κυρίως bus, σε αυτό που θα χρησιμοποιηθεί, οι δυνατές τιμές είναι δύο. Το OPB και το AXI έχουν περιγραφεί. Το OPB υπάρχει κυρίως για θέματα συμβατότητας, αφού είναι αυτό που χρησιμοποιούσαν παλαιότερα ολοκληρωμένα της Xilinx, τα οποία περιείχαν το hard core του επεξεργαστή PowerPC 4xx. Ο MicroBlaze μπορεί να χρησιμοποιηθεί και με τις δύο αρτηρίες, αλλά στις νεότερες εκδόσεις προτιμάται το AXI4.

Αφού ολοκληρωθεί η διαδικασία δημιουργίας του συστήματος, το EDK μας δίνει την δυνατότητα να παραμετροποιήσουμε τα επιμέρους στοιχεία, δηλαδή του ιδίου του επεξεργαστή, αλλά και των περιφερειακών που είναι συνδεδεμένα με αυτόν. Επίσης μέσω του EDK μπορούμε να κάνουμε τις κατάλληλες διασυνδέσεις μεταξύ των περιφερειακών και να αναθέσουμε τις διευθύνσεις μνήμης που θέλουμε σε αυτά. Η ανάθεση των ευρών των διευθύνσεων μνήμης σε κάθε περιφερειακό, παράγεται αυτόματα από το EDK, μπορεί όμως να αλλαχθεί από τον σχεδιαστή αν αυτό κρίνεται απαραίτητο. Τέλος το EDK μας δίνει την δυνατότητα να δημιουργήσουμε εύκολα ένα custom περιφερειακό, παράγοντας templates για τα αρχεία custom_component.vhdl και user_logic.vhdl τα οποία μπορούν να παραμετροποιηθούν κατάλληλα, όπως περιγράφεται στην ενότητα 6.

Όλες οι επιλογές που κάνουμε στο EDK αποθηκεύονται σε μια σειρά από αρχεία, τα οποία χρησιμοποιούνται από τα επιμέρους εργαλεία για την υλοποίηση του συστήματος. Τα πιο βασικά από αυτά τα αρχεία είναι το MHS και MSS, τα οποία περιγράφουν το hardware και software σχεδιασμό του συστήματος αντίστοιχα. Τα δύο αρχεία για το σύστημα που υλοποιήθηκε παρέχονται στο παράρτημα Α.

B.3 Software Development Kit (SDK)

Τέλος το SDK παρέχει τα κατάλληλα εργαλεία για την δημιουργία εφαρμογών που θα εκτελούνται στο σύστημα που έχει υλοποιηθεί. Το SDK είναι βασισμένο στο IDE eclipse, παραμετροποιημένο κατάλληλα για την εύκολη υλοποίηση των εφαρμογών.

Για την δημιουργία ενός software project, πρώτα πρέπει να δημιουργήσουμε ένα hardware specification platform project, που μπορεί να παραχθεί αυτόματα από το ISE εάν επιλέξουμε να κάνουμε export to design στο SDK. Το project αυτό περιέχει τα κατάλληλα αρχεία τα οποία περιγράφουν το hardware του συστήματος και χρησιμοποιούνται για την δημιουργία των απαραίτητων βιβλιοθηκών.

Το δεύτερο project που θα πρέπει να δημιουργήσουμε είναι το board support package, το οποίο περιέχει όλες τις βιβλιοθήκες που μπορούν να χρησιμοποιηθούν για την

ανάπτυξη μιας εφαρμογής, όπως οι drivers για τα περιφερειακά του συστήματος αλλά και η newlib. Δεν υπάρχει περιορισμός στον αριθμό των συγκεκριμένων projects που μπορούμε να δημιουργήσουμε, αλλά πρέπει όλα να αναφέρονται σε ένα hardware specification platform project

Αφού έχουν δημιουργηθεί τα δύο παραπάνω projects, μπορεί να δημιουργηθεί ακόμη ένας απεριόριστος αριθμός από εφαρμογές, οι οποίες θα αναφέρονται στο ίδιο ή σε διαφορετικά board support packages.

Τέλος ένα ακόμα εργαλείο το οποίο υπάγεται στο SDK είναι το XMD (Xilinx MicroBlaze Debugger), το οποίο συνδέεται στο περιφερειακό MDM, εάν αυτό υπάρχει στο σύστημα. Μέσω του XMD μπορούμε να ελέγξουμε την κατάσταση του επεξεργαστή, εκκινώντας ή σταματώντας την εκτέλεσή του, αλλά και να αλλάξουμε την ροή εκτέλεσης διαβάζοντας και γράφοντας στους καταχωρητές του, καθώς επίσης και σε όποια άλλη διεύθυνση μνήμης θέλουμε. Το XMD προσθέτει λειτουργίες στον προγραμματισμό του ολοκληρωμένου, όπως την μεταφορά μεταγλωττισμένων προγραμμάτων για τον επεξεργαστή και την εκτέλεσή τους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Xilinx Inc, MicroBlaze Processor Reference Guide Embedded Development Kit EDK 14.5 User Guide, http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/mb_ref_guide.pdf, March 20, 2013
2. Xilinx Inc, LogiCORE IP AXI Interconnect(v1.04) Product Manual Specification, http://www.xilinx.com/support/documentation/ip_documentation/ds747_axi_intc.pdf, October 19, 2011
3. Xilinx Inc, LogiCORE IP AXI Interconnect (v1.04.a) Product Manual Specification, http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v1_04_a/ds768_axi_interconnect.pdf, October 19, 2011
4. Xilinx Inc, LogiCORE IP AXI4-Lite IPIF (v1.01.a) Product Manual Specification, http://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif_ds765.pdf, January 18, 2012
5. Xilinx Inc, LogiCORE IP Interrupt Control (v2.01a) Product Manual Specification, http://www.xilinx.com/support/documentation/ip_documentation/interrupt_control.pdf, July 25, 2012
6. Xilinx Inc, OS and Libraries Document Collection User Guide, March 20, 2013
7. Xilinx Inc, Device Driver Programmer Guide December 14, 2007
8. Xilinx Inc, EDK Concepts Tools and Techniques User Guide, http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/edk_ctt.pdf, March 20, 2013