



Ανάπτυξη Μεθοδολογίας Σχεδιασμού Βέλτιστων Επεξεργαστών Ειδικού Σκοπού

ΠΕΝΕΔ-2003

Τίτλος:	Αποτίμηση της Μεθοδολογίας Σχεδιασμού ΕΕΣ με Προτυποποίηση σε FPGA
Συγγραφείς:	Νικόλαος Καβαβιάς (Ερευνητής Α.Π.Θ.)
Κωδικός:	Π 6.1
Έκδοση:	1.0
Τύπος:	Παραδοτέο
Εμπιστευτικότητα:	Δημόσια
Ημερομηνία:	Νοέμβριος 15, 2007
Πρόγραμμα:	Πρόγραμμα Ενίσχυσης του Ερευνητικού Δυναμικού (ΠΕΝΕΔ 2003)
Λέξεις-Κλειδιά:	επεξεργαστές ειδικού σκοπού, ενσωματωμένα συστήματα, επεξεργασία εικόνας, διάταξη πυλών προγραμματιζόμενου πεδίου, FPGA, εκτίμηση επιδόσεων
Πρόλογος:	Αντικείμενο του παραδοτέου αυτού είναι η εφαρμογή της μεθοδολογίας σχεδιασμού των ΕΕΣ για την περίπτωση του σχεδιασμού ASIP για ομάδα εφαρμογών επεξεργασίας εικόνας. Δίνεται αναλυτική περιγραφή της δομής και τρόπου λειτουργίας πρότυπου ενσωματωμένου συστήματος επεξεργασίας εικόνας ονόματι IPCN (Image Processing Computational Node), και καταγράφονται οι λεπτομέρειες του λογισμικού και του υλικού (ειδικές εντολές) του ASIP. Η ορθή λειτουργία του ενσωματωμένου συστήματος IPCN πιστοποιείται α) με την προσομοίωση αρχιτεκτονικού μοντέλου των επεξεργαστών ByoRISC σε ArchC, β) με την προσομοίωση του RTL VHDL μοντέλου των επεξεργαστών και γ) με την προτυποποίηση του συστήματος σε αναπτυξιακή πλακέτα FPGA.

Ιστορικό

Ημερομηνία	Έκδοση	Σχόλια
Οκτώβριος 21, 2007	0.1	Πρώτη πρόχειρη έκδοση
Νοέμβριος 15, 2007	1.0	Τελική έκδοση

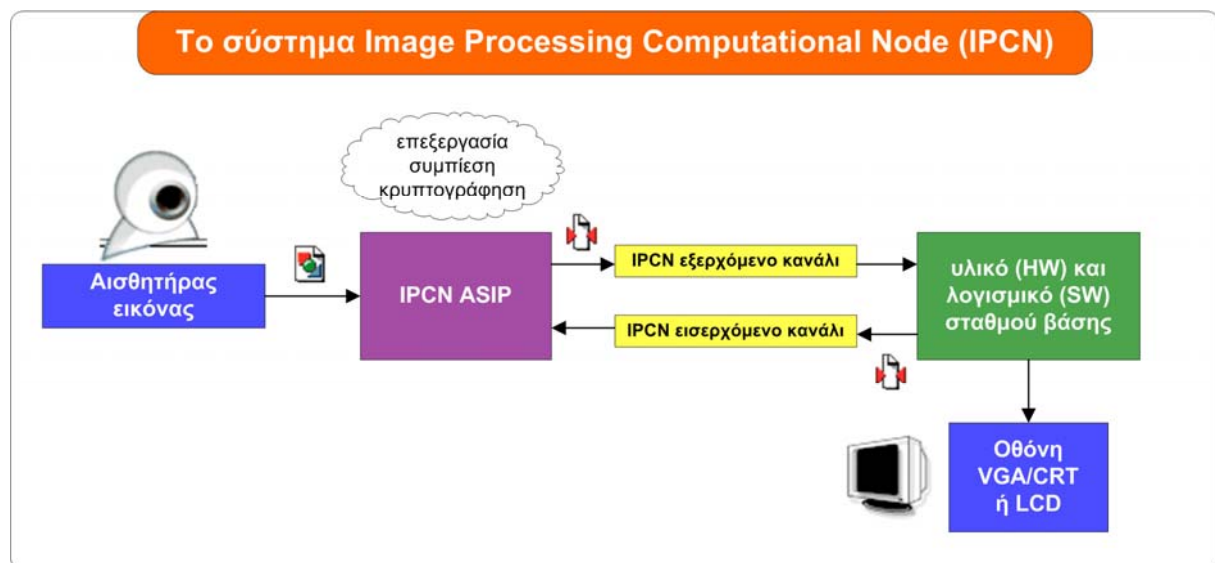
Πίνακας περιεχομένων

Πίνακας περιεχομένων	3
1. ΕΙΣΑΓΩΓΗ: ΤΟ ΕΝΣΩΜΑΤΩΜΕΝΟ ΣΥΣΤΗΜΑ IPCN	4
2. ΑΝΑΛΥΣΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ ΕΦΑΡΜΟΓΩΝ ΤΟΥ IPCN.....	6
2.1. Μίγμα στατικών εντολών (SIMIX) και μίγμα δυναμικών εντολών (DIMIX).....	12
2.2. Ανάλυση τύπων δεδομένων	13
2.3. Εξαγωγή και χαρακτηρισμός των κρίσιμων βασικών μπλοκ των εφαρμογών.....	14
2.4. Καθορισμός του γενικού αρχιτεκτονικού περιγράμματος, γέννηση και επιλογή εντολών	22
3. ΕΚΤΕΛΕΣΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ ΕΦΑΡΜΟΓΩΝ ΣΕ ΠΡΟΣΟΜΟΙΩΤΗ ΕΚΤΙΜΗΣΗΣ ΚΥΚΛΟΥ.....	50
4. Η ΑΝΑΠΤΥΞΙΑΚΗ ΠΛΑΚΕΤΑ FPGA ΓΙΑ ΤΙΣ ΔΟΚΙΜΕΣ ΤΟΥ ΕΝΣΩΜΑΤΩΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ	53
5. ΑΝΑΦΟΡΕΣ.....	55

1. ΕΙΣΑΓΩΓΗ: ΤΟ ΕΝΣΩΜΑΤΩΜΕΝΟ ΣΥΣΤΗΜΑ IPCN

Προκειμένου να επιδειχθεί η χρήση της μεθοδολογίας πάνω σε ρεαλιστικές εφαρμογές, στην ενότητα αυτή αναπτύσσεται και σχεδιάζεται ένας ΕΕΣ με βάση την αρχιτεκτονική ByoRISC ο οποίος στοχεύει εφαρμογές επεξεργασίας εικόνας. Ο ΕΕΣ είναι υπεύθυνος για το υπολογιστικό μέρος των αρμοδιοτήτων ενός υποθετικού ενσωματωμένου συστήματος, το οποίο ονομάζουμε IPCN (Image Processing Computational Node). Το διάγραμμα βαθμίδων ενός συστήματος IPCN απεικονίζεται στο Σχήμα 1.1.

Σύμφωνα με το Σχήμα 1.1, το IPCN αποτελείται από έναν αισθητήρα εικόνας (image sensor) τεχνολογίας CMOS ο οποίος καταγράφει εικόνες από το περιβάλλον σε ανάλυση $M\text{SIZE} \times N\text{SIZE}$ ¹. Το IPCN έχει δυνατότητα για full-duplex ασύρματη επικοινωνία με σταθμό βάσης από τον οποίο μπορεί να λάβει και στον οποίο μπορεί να στείλει δεδομένα εικόνας (σειρές εικονοστοιχείων) σε μορφή πακέτων. Στο σταθμό βάσης γίνεται η απεικόνιση των δεδομένων, τα οποία έχει συλλέξει το IPCN από το περιβάλλον, σε έγχρωμη οθόνη τεχνολογίας VGA/CRT ή LCD. Στην ενότητα αυτή εστιάζουμε στις υπολογιστικές δυνατότητες του IPCN και δεν εξετάζονται διεξοδικά το υποσύστημα του αισθητήρα, το πρωτόκολλο επικοινωνίας με το σταθμό βάσης και τα χαρακτηριστικά του καναλιού επικοινωνίας.



Σχήμα 1.1: Το ενσωματωμένο σύστημα IPCN για επεξεργασία εικόνας.

Σε περιπτώσεις όπου η αποστολή όλων των δεδομένων από το IPCN δεν κρίνεται απαραίτητη κυρίως για λόγους χρόνου μετάδοσης και κατανάλωσης ισχύος (εφόσον η λειτουργία του βασίζεται σε φορητή πηγή ενέργειας), είναι αναγκαίο να δύναται να επεξεργαστεί τα δεδομένα τα οποία έχει συλλέξει ώστε να αποστέλλει μόνο ό,τι απαιτείται στο σταθμό βάσης. Για το λόγο αυτό, αλλά και για να μπορεί να λαμβάνει εντολές αλλά και δεδομένα εικόνας όπως για σύγκριση, είναι αναγκαίο να μπορεί να εξυπηρετήσει ένα σύνολο εφαρμογών, επιτυγχάνοντας επαρκείς επιδόσεις. Επίσης, καθώς η αποστολή/λήψη δεδομένων από το IPCN γίνεται με ασύρματο τρόπο, θα πρέπει να υποστηρίζει κάποια

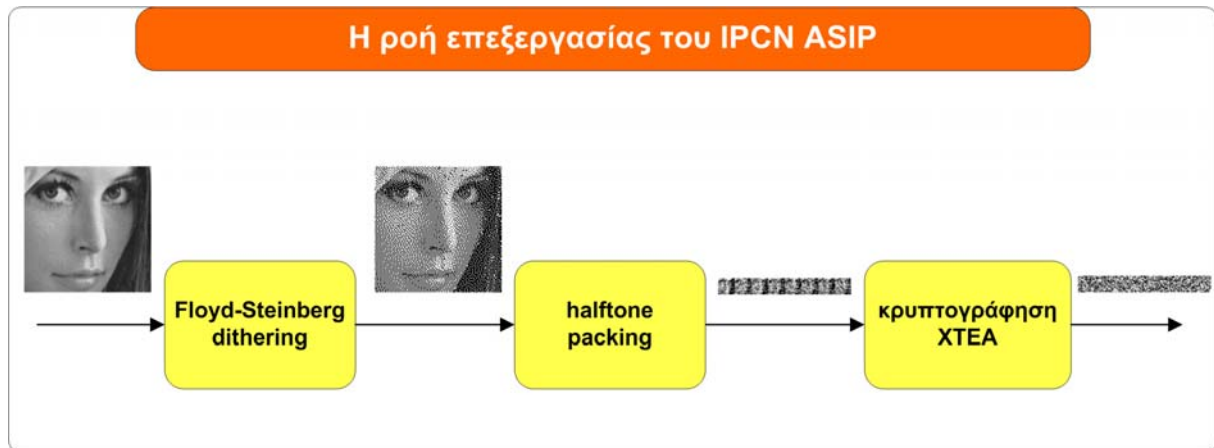
¹ Τυπικές αναλύσεις που θα θεωρήσουμε από εδώ και στο εξής είναι οι: 64×64 , 256×256 , 320×240 , 512×512 και 512×480 .

τεχνική κρυπτογράφησης και ανάκτησης δεδομένων για την ασφάλεια των δεδομένων από ανεπιθύμητους παραλήπτες.

Το ενσωματωμένο λογισμικό του IPCN αποτελείται από ένα πολύ βασικό RTOS, το οποίο χειρίζεται τις χρονοθυρίδες εξυπηρέτησης του λογισμικού εφαρμογών το οποίο αποτελείται από εφαρμογές επεξεργασίας εικόνας, συμπίεσης/ αποσυμπίεσης δεδομένων και κρυπτογράφησης/ αποκρυπτογράφησης δεδομένων. Συγκεκριμένα στο IPCN εκτελούνται οι εφαρμογές:

- **fsdither**: διτονική απόδοση εικόνας κατά Floyd-Steinberg για τη μείωση των επιπέδων του γκρι σε μια greyscale εικόνα από 256 σε 2 [EEMBC]. Ο fsdither πλεονεκτεί σε σχέση με μια απλή εφαρμογή επιβολής κατωφλίου (image thresholding) καθώς εξασφαλίζει την προσέγγιση της αρχής χρωματικής απόδοσης με τεχνική διάχυσης σφαλμάτων (error diffusion).
- **hilcurv**: παραμετρική γεννήτρια καμπυλών Hilbert. Οι καμπύλες Hilbert ανήκουν στην κατηγορία των καμπυλών πλήρωσης χώρου και η χρήση τους που ενδιαφέρει εδώ είναι η παραγωγή εικόνων δοκιμής για τον έλεγχο λειτουργίας της συσκευής IPCN.
- **htpack**: η εφαρμογή htpack (halftone packer) συμπιέζει μη-απωλεστικά μια διτονική εικόνα (οι RGB κωδικοποιήσεις των δύο θεωρούμενων τόνων μπορούν να προγραμματιστούν) κατά 8 φορές. Ο αλγόριθμος htpack στοιβάζει 8 ψηφία ανά byte συμπιεσμένων δεδομένων καθώς οι δύο τόνοι μπορούν να αποδοθούν από ένα μόνο bit. Στην πράξη χρησιμοποιούμε την htpack για τη συμπίεση μιας greyscale εικόνας στην οποία έχει πρώτα εφαρμοστεί η fsdither.
- **htunpack**: η εφαρμογή htunpack (halftone unpacker) αποσυμπιέζει μια διτονική εικόνα η οποία έχει προηγουμένως συμπιεστεί από την htpack. Όπως και στην htpack, οι κωδικοποιήσεις RGB των δύο θεωρούμενων τόνων μπορούν να προγραμματιστούν και ακόμη να είναι και διαφορετικές από αυτές της htpack. Ο htunpack αποσυμπιέζει 8 εικονοστοιχεία ανά byte συμπιεσμένων δεδομένων.
- **xteaenc**: εφαρμογή κρυπτογράφησης κατά τον αλγόριθμο XTEA (eXtended Tiny Encryption Algorithm) [Nee97]. Ο κωδικοποιητής XTEA είναι τύπου block cipher με μήκος κλειδιού 128 bit και μέγεθος μπλοκ 64 bit. Χρησιμοποιεί δομή δικτύου Feistel και ο αριθμός των χρησιμοποιούμενων γύρων είναι ίσος με 32 (64 γύροι Feistel). Ο αλγόριθμος προτάθηκε από τους D. Wheeler και R. Needham για την αντιμετώπιση των αδυναμιών του προγενέστερου TEA. Ο XTEA επιλέχθηκε καθώς α) δεν υπόκειται σε καθεστώς πατέντας (ανήκει στο δημόσιο πεδίο και μπορεί να χρησιμοποιηθεί από τον οποιονδήποτε), β) λόγω του πολύ μικρού μεγέθους του και ιδιαίτερα των περιορισμένων απαιτήσεων του σε RAM και γ) δεν έχει κρυπταναλυθεί πλήρως, άρα προσφέρει κάποια αξιοπιστία στην ασφάλεια δεδομένων. Μέχρι σήμερα έχουν δημοσιευτεί αναφορές επιτυχούς μερικής μόνο (26 από τους 64 γύρους) κρυπτανάλυσης του XTEA [Nee97].
- **xteadec**: εφαρμογή αποκρυπτογράφησης κατά τον αλγόριθμο XTEA. Η λειτουργία του αποκωδικοποιητή είναι σχεδόν συμμετρική με αυτή του κωδικοποιητή.

Μία περίπτωση χρήσης του IPCN σε ρεαλιστικές συνθήκες σκιαγραφείται στο Σχήμα 2.2. Ο αισθητήρας εικόνας καταγράφει μια εικόνα ανάλυσης 128×128 εικονοστοιχείων από το περιβάλλον. Στη συνέχεια, η εικόνα υπόκειται εντός του ΕΕΣ σε επεξεργασία διτονικής απόδοσης, μη-απωλεστική συμπίεση (halftone packing) και κρυπτογράφηση κατά τον αλγόριθμο XTEA. Στη συνέχεια, τα δεδομένα της συμπιεσμένης και κρυπτογραφημένης εικόνας μπορούν να σταλούν στο σταθμό βάσης.



Σχήμα 1.2: Περίπτωση χρήσης του IPCN για επεξεργασία, συμπίεση και κρυπτογράφηση δεδομένων εικόνας.

Το ζητούμενο είναι η ανάδειξη των ειδικών εντολών για το σχεδιασμό των κατάλληλων ΕΛΜ για τον επεξεργαστή ΕΕΣ του IPCN ώστε την επιτάχυνση των εφαρμογών κατά το περισσότερο δυνατό. Τα χαρακτηριστικά και οι περιορισμοί του IPCN δίνονται στον Πίνακα 1.1.

Πίνακας 1.1: Χαρακτηριστικά και περιορισμοί του IPCN.

Παράμετρος	Περιγραφή	Τιμή
IMEMSIZE	Μνήμη προγράμματος	8KB
DMEMSIZE	Μνήμη δεδομένων	8KB-1MB
ASIC_AREA	Συνολική επιφάνεια ολοκληρωμένου για υλοποίηση σε τεχνολογία ASIC	επιπλέον 100,000μ ² του βασικού ByoRISC
FPGA_AREA	Συνολική επιφάνεια ολοκληρωμένου για υλοποίηση σε τεχνολογία FPGA	50% της επιφάνειας του XC3S200

2. ΑΝΑΛΥΣΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ ΕΦΑΡΜΟΓΩΝ ΤΟΥ IPCN

Σύμφωνα με τη μεθοδολογία, το πρώτο βήμα στη διαδικασία ανάπτυξης και σχεδιασμού του ΕΕΣ αποτελεί η ανάλυση των εφαρμογών του ενδιαφέροντος, των οποίων τα γενικά χαρακτηριστικά συνοψίζονται στον Πίνακα 2.1. Ο μεταγλωττιστής που χρησιμοποιήθηκε είναι ο MachSUIF στοχεύοντας την αρχιτεκτονική SUIFvm (IR), ενώ οι εφαρμογές καταγράφηκαν και σε γλώσσα συμβολομεταφραστή (ASM), πραγματοποιώντας χειροκίνητα τα βήματα ενός backend μεταγλωττιστή. Ο θεωρούμενος επεξεργαστής για τις αντίστοιχες μέτρησεις πάνω σε κώδικα συμβολομεταφραστή είναι η βασική αρχιτεκτονική ByoRISC. Για τη μεταγλώττιση υπό τον MachSUIF χρησιμοποιήθηκαν τα περάσματα βελτιστοποίησης: ifconversion, lcase, cfg2ssa, ssa2cfg, reep, reepdeux, cplx_locate, strength_reduct, και έγινε πολλαπλή χρήση του dce.

Πίνακας 2.1: Γενικά χαρακτηριστικά του λογισμικού εφαρμογών του IPCN.

Εφαρμογή	Αναφορά	# εντολών (IR)	# εντολών (ASM)	# BB (IR)	# BB (ASM)	Δυναμικές εντολές (ASM)	Κύκλοι εκτέλεσης (IR)*	Κύκλοι εκτέλεσης (ASM)

<i>fsdither</i>	[EEMBC]	80	83	8	12	223503	287116	250248
<i>hilcurv</i>	[War02]	51	56	4	7	667665	164877	725009
<i>htpack</i>	--	87	109	4	3	39818	40976	51421
<i>htunpack</i>	--	96	117	4	3	45410	44562	57013
<i>xteaenc</i>	[Nee97]	132	86	6	10	584217	692760	619033
<i>xteadec</i>	[Nee97]	144	84	6	10	553494	686636	588310

Ο πηγαίος κώδικας για τις 6 εφαρμογές του ενδιαφέροντος δίνεται από τους Κώδικες 2.1.α ως 2.1.στ, με τη σειρά που αναφέρονται οι εφαρμογές στον Πίνακα 2.1, αντίστοιχα. Κοινό χαρακτηριστικό των εφαρμογών είναι ότι αποτελούνται κυρίως από απλά ή διπλά πλήρως φωλιασμένους βρόχους.

```
void fsdither(unsigned int threshold)
{
    int i,j;
    int error;
    int x,q;

    for (i=0; i<NSIZE; i++)
    {
        for (j=0; j<MSIZE; j++)
        {
            source_word[i*MSIZE+j] = source[i*MSIZE+j];
        }
    }

    for (i=0; i<NSIZE; i++)
    {
        for (j=0; j<MSIZE; j++)
        {
            x = source_word[i*MSIZE+j];

            dest[i*MSIZE+j] = ((x > threshold-1) ? 0xff : 0x00);
            q = dest[i*MSIZE+j];

            error = x - q;

            source_word[i*MSIZE+(j+1)] += (error * 7) >> 4;
            source_word[(i+1)*MSIZE+(j-1)] += (error * 3) >> 4;
            source_word[(i+1)*MSIZE+(j)] += (error * 5) >> 4;
            source_word[(i+1)*MSIZE+(j+1)] += (error) >> 4;
        }
    }
}
```

(α) fsdither.

```
for (s = 0; s < N; s++)
{
    state = 0;
    x = 0;
    y = 0;

    for (i = 2*n - 2; i >= 0; i -= 2)
    {
        row = (state << 2) | (s >> i) & 3;
        x = (x << 1) | (0x936C >> row) & 1;
        y = (y << 1) | (0x39C6 >> row) & 1;
        state = (0x3E6B94C1 >> (row << 1)) & 3;
    }

    dest[x*MSIZE+y] = 0xFF;
}
```

(β) hilcurv.



```
void htpack(unsigned int one, unsigned int zero)
{
    unsigned int i;
    unsigned int temp0, temp1, temp2, temp3, temp4, temp5, temp6, temp7;
    unsigned int temp0_or, temp1_or, temp2_or, temp3_or, temp4_or, temp5_or, temp6_or,
        temp7_or;
    unsigned int temp_val;
    unsigned int base_addr;

    for (i=0; i<NSIZE*MSIZE/8; i++)
    {
        base_addr = 8*i;
        temp_val = 0x00;

        temp0 = source[base_addr ];
        temp1 = source[base_addr+1];
        temp2 = source[base_addr+2];
        temp3 = source[base_addr+3];
        temp4 = source[base_addr+4];
        temp5 = source[base_addr+5];
        temp6 = source[base_addr+6];
        temp7 = source[base_addr+7];

        if (temp0 == one)
        {
            temp0_or = (1 << 0);
        }
        else
        {
            temp0_or = (0 << 0);
        }

        if (temp1 == one)
        {
            temp1_or = (1 << 1);
        }
        else
        {
            temp1_or = (0 << 1);
        }

        if (temp2 == one)
        {
            temp2_or = (1 << 2);
        }
        else
        {
            temp2_or = (0 << 2);
        }

        if (temp3 == one)
        {
            temp3_or = (1 << 3);
        }
        else
        {
            temp3_or = (0 << 3);
        }

        if (temp4 == one)
        {
            temp4_or = (1 << 4);
        }
        else
        {
            temp4_or = (0 << 4);
        }

        if (temp5 == one)
        {
            temp5_or = (1 << 5);
        }
        else
        {
```



```
    temp5_or = (0 << 5);
}

if (temp6 == one)
{
    temp6_or = (1 << 6);
}
else
{
    temp6_or = (0 << 6);
}

if (temp7 == one)
{
    temp7_or = (1 << 7);
}
else
{
    temp7_or = (0 << 7);
}

temp_val = temp0_or | temp1_or | temp2_or | temp3_or | temp4_or | temp5_or | temp6_or |
           temp7_or;
dest[i] = temp_val;
}
}
```

(γ) htpack.

```
void htunpack(unsigned int one, unsigned int zero)
{
    unsigned int i;
    unsigned int temp0, temp1, temp2, temp3, temp4, temp5, temp6, temp7;
    unsigned int cond0, cond1, cond2, cond3, cond4, cond5, cond6, cond7;
    unsigned int temps0, temps1, temps2, temps3, temps4, temps5, temps6, temps7;
    unsigned int temp_val;

    for (i=0; i<NSIZE*MSIZE/8; i++)
    {
        temp_val = source[i];

        temp0 = (temp_val & 0x01);
        temp1 = (temp_val & 0x02) >> 1;
        temp2 = (temp_val & 0x04) >> 2;
        temp3 = (temp_val & 0x08) >> 3;
        temp4 = (temp_val & 0x10) >> 4;
        temp5 = (temp_val & 0x20) >> 5;
        temp6 = (temp_val & 0x40) >> 6;
        temp7 = (temp_val & 0x80) >> 7;

        cond0 = temp0 == 1;
        cond1 = temp1 == 1;
        cond2 = temp2 == 1;
        cond3 = temp3 == 1;
        cond4 = temp4 == 1;
        cond5 = temp5 == 1;
        cond6 = temp6 == 1;
        cond7 = temp7 == 1;

        if (cond0)
        {
            temps0 = one;
        }
        else
        {
            temps0 = zero;
        }

        if (cond1)
        {
            temps1 = one;
        }
        else
        {

```



```
    temps1 = zero;
}

if (cond2)
{
    temps2 = one;
}
else
{
    temps2 = zero;
}

if (cond3)
{
    temps3 = one;
}
else
{
    temps3 = zero;
}

if (cond4)
{
    temps4 = one;
}
else
{
    temps4 = zero;
}

if (cond5)
{
    temps5 = one;
}
else
{
    temps5 = zero;
}

if (cond6)
{
    temps6 = one;
}
else
{
    temps6 = zero;
}

if (cond7)
{
    temps7 = one;
}
else
{
    temps7 = zero;
}

dest[8*i ] = temps0;
dest[8*i+1] = temps1;
dest[8*i+2] = temps2;
dest[8*i+3] = temps3;
dest[8*i+4] = temps4;
dest[8*i+5] = temps5;
dest[8*i+6] = temps6;
dest[8*i+7] = temps7;
}
}
```

(δ) htunpack.

```
void encode_xtea(unsigned int v[2], unsigned int result[2])
{
    unsigned int y, z, DELTA;
    unsigned int sum;
```

```
unsigned int i;

y = v[0];
z = v[1];
DELTA = 0x9e3779b9;
sum = 0;

for (i=0; i<N; i++)
{
    y += ((z<<4 ^ z>>5) + z) ^ (sum + keystr[sum & 3]);
    sum += DELTA;
    z += ((y<<4 ^ y>>5) + y) ^ (sum + keystr[(sum>>11) & 3]);
}

result[0] = y;
result[1] = z;
}
```

(ε) xteaenc.

```
void decode_xtea(unsigned int v[2], unsigned int result[2])
{
    unsigned int y, z, DELTA;
    unsigned long sum;
    unsigned int i;

    y = v[0];
    z = v[1];
    DELTA = 0x9e3779b9;
    sum = DELTA * (N);

    for (i=0; i<N; i++)
    {
        z -= ((y<<4 ^ y>>5) + y) ^ (sum + keystr[(sum>>11) & 3]);
        sum -= DELTA;
        y -= ((z<<4 ^ z>>5) + z) ^ (sum + keystr[sum & 3]);
    }

    result[0] = y;
    result[1] = z;
}
```

(στ) xteadec.

Κώδικας 2.1: Πηγαίος κώδικας για το λογισμικό εφαρμογών του IPCN. Ως *source* αναφέρονται δεδομένα τύπου U8, *source_word* δεδομένα εικόνας τύπου U32 ή S32 για τις εικόνες που δίνονται ως είσοδο στην αντίστοιχη εφαρμογή και *dest* δεδομένα τύπου U8 που αντιστοιχούν στην επεξεργασμένη/συμπιεσμένη/κρυπτογραφημένη εικόνα. Οι διαστάσεις των εικόνων δίνονται σε συνάρτηση με τα NSIZE και MSIZE που για την εικόνα δοκιμής είναι ίσα με 64. Ο τύπος δεδομένων (unsigned) long αντιστοιχεί στον MachSUIF τύπο (U|S)32.

Η διαδικασία ανάλυσης των εφαρμογών συμπεριλαμβάνει τις παρακάτω ενέργειες οι οποίες και πραγματοποιούνται πλήρως αυτόματα με τη χρήση του εργαλείου YARDstick:

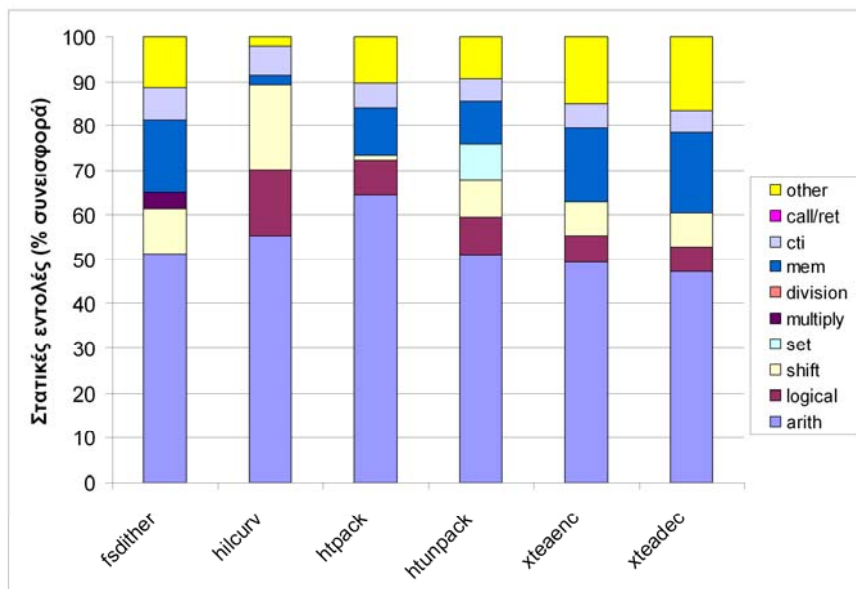
- εξαγωγή στατικού μίγματος εντολών
- εξαγωγή δυναμικού μίγματος εντολών
- εξαγωγή δυναμικών εμφανίσεων τύπων δεδομένων
- ανάδειξη κρίσιμων βασικών μπλοκ (κατάταξη των βασικών μπλοκ κατά φθίνουσα σειρά αριθμού εκτιμώμενων κύκλων)
- εξαγωγή γράφων σε απεικονίσεις VCG και Graphviz (dot) για όλα τα βασικά μπλοκ και γράφους ροής ελέγχου (CFG) των εφαρμογών υπό ανάλυση
- εκτίμηση της μέγιστης και μέσης παραλληλίας βασικών λειτουργιών (εντολές της

SUIFvm IR) για τα κρίσιμα βασικά μπλοκ (maximum ILP, average ILP)

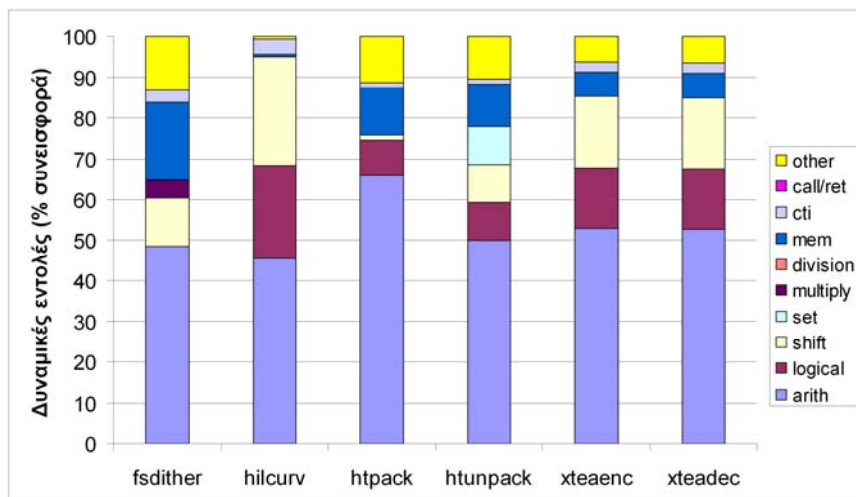
Η διαδικασία (εκτός της εξαγωγής των δεικτών ILP) πραγματοποιείται σε χρόνο ίσο με 61 sec με χρήση προσωπικού υπολογιστή (επεξεργαστής Pentium-4, 2,8 GHz, 368 MB RAM) σε περιβάλλον Cygwin/x86 (έκδοση 1.5.15). Αξίζει να σημειωθεί εδώ ότι οι ίδιες ακριβώς αναλύσεις είναι επιλέξιμες και στην περίπτωση ανάλυσης εφαρμογών που είναι υπό μορφή κώδικα συμβολομεταφραστή.

2.1. Μίγμα στατικών εντολών (SIMIX) και μίγμα δυναμικών εντολών (DIMIX)

Στο Σχήμα 2.1 δίνεται το μίγμα στατικών εντολών για την περίπτωση βασικής βελτιστοποίησης με MachSUIF. Στο Σχήμα 2.2 δίνεται αντίστοιχα το μίγμα δυναμικών εντολών για τον υπολογισμό του οποίου χρησιμοποιήθηκαν οι συχνότητες βασικών μπλοκ για κάθε εφαρμογή.



Σχήμα 2.1: Μίγμα στατικών εντολών για το λογισμικό εφαρμογών του IPCN. Ο συμβολισμός cti (control-transfer instructions) χρησιμοποιείται για το άθροισμα των εντολών διακλάδωσης με (cbr) ή χωρίς (ubr) συνθήκη. Οι λοιπές (other) εντολές συμπεριλαμβάνουν κυρίως λειτουργίες ανάθεσης τύπου (cvt) και φόρτωσης συμβόλου/διεύθυνσης (lda).



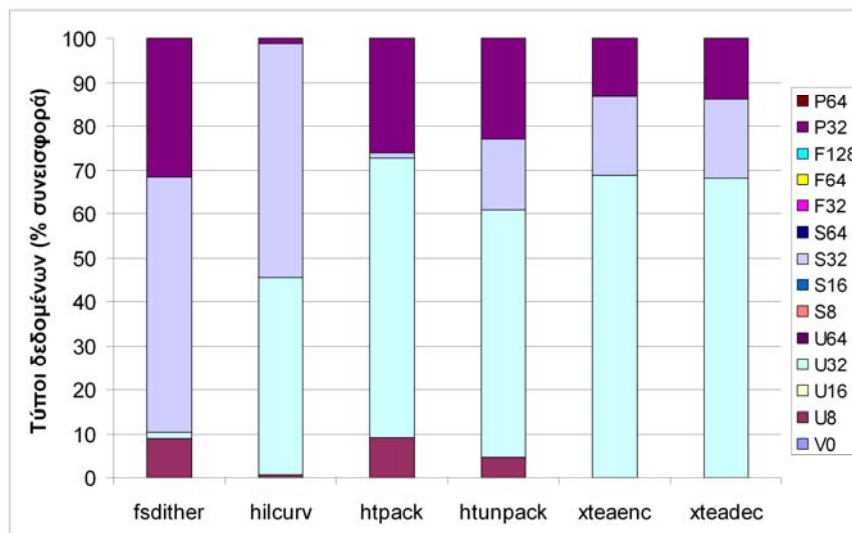
Σχήμα 2.2: Μίγμα δυναμικών εντολών για το λογισμικό εφαρμογών του IPCN.

Συνοψίζοντας, από τα αποτελέσματα του Σχήματος 2.3 προκύπτει ότι στο μίγμα δυναμικών εντολών κυριαρχούν οι αριθμητικές (arith) εντολές, για σχεδόν όλες τις εφαρμογές, πλην της cgc, με κατά μέσο όρο συνεισφορά 52.7%. Κατά δεύτερο λόγο απαντώνται εντολές προσπέλασης στη μνήμη δεδομένων (12.13%), ολίσθησης (8.97%), σύγκρισης (7.15%), και διακλάδωσης (5.83%). Το σημαντικό ποσοστό των «άλλων» (other) εντολών (10.82%) οφείλεται εν μέρει στη συνεισφορά από εντολές ανάθεσης τύπου (CVT). Το ποσοστό των «άλλων» εντολών γενικά περιορίζεται με τη χρήση του περάσματος peepdeux, παρουσιάζεται δε αυξημένο για τις xteaenc και xteadec.

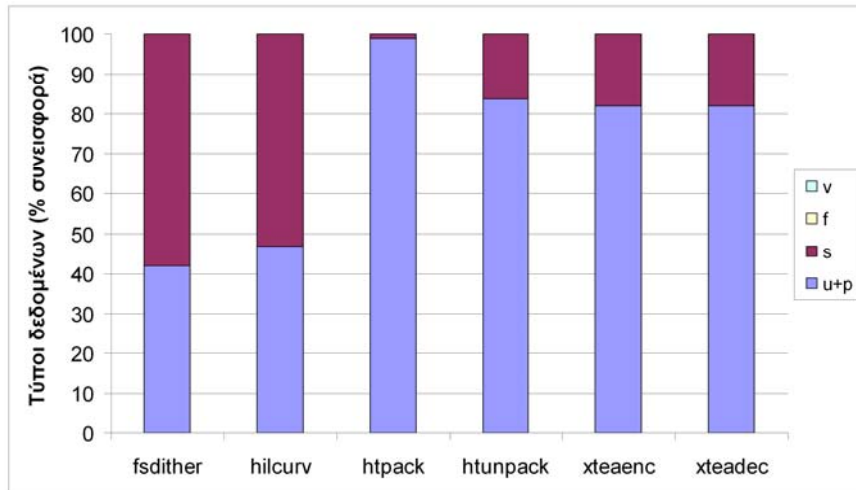
2.2. Ανάλυση τύπων δεδομένων

Αυτό που ενδιαφέρει εδώ είναι να καταγραφεί η αναλογία των δυναμικών εμφανίσεων των διαφόρων τύπων δεδομένων σε όλη τη διάρκεια μιας εφαρμογής. Στο Σχήμα 2.3 δίνονται αποτελέσματα ανάλυσης ΤΔ με τις συνολικές εμφανίσεις τύπων δεδομένων ανά εφαρμογή. Υπολογίζονται με άθροιση των επιμέρους εμφανίσεων ανά συνάρτηση (-dt-dynamic-proc).

Διαπιστώνουμε ότι οι εφαρμογές χρησιμοποιούν μόνο ακέραιους τύπους δεδομένων ενώ οι συχνότερες εμφανίσεις τύπων δεδομένων κυριαρχούνται από τους τύπους U32, S32, P32 και U8. Ιδιαίτερα, οι χρήσεις απρόσημης αριθμητικής φαίνεται ότι είναι σημαντικά περισσότερες (δυναμικές εμφανίσεις των αντίστοιχων ΤΔ: 72.5%) από τις χρήσεις προσημασμένης αριθμητικής.



(α)



(β)

Σχήμα 2.3: % εμφανίσεις τύπων δεδομένων σε δυναμικές εντολές ανά εφαρμογή για το λογισμικό εφαρμογών του IPCN. (α) Ανά τύπο δεδομένων. (β) Με ομαδοποίηση σε απρόσημους (u+p) και προσημασμένους ακέραιους (s), κινητής υποδιαστολής (f) και άλλο (v).

2.3. Εξαγωγή και χαρακτηρισμός των κρίσιμων βασικών μπλοκ των εφαρμογών

Για τον προσδιορισμό των κρίσιμων βασικών μπλοκ της εφαρμογής, έγινε κατάταξή τους, σύμφωνα με τον αριθμό δυναμικών κύκλων (αριθμός στατικών κύκλων BB × συχνότητα εκτέλεσης BB). Στον Πίνακα 2.2 δίνονται:

- το όνομα της εφαρμογής, της υπορουτίνας και ο αύξων αριθμός του κρίσιμου BB
- ο αριθμός στατικών εντολών του
- ο αριθμός εκτιμώμενων δυναμικών κύκλων
- το ποσοστό που καταλαμβάνουν οι κύκλοι εκτέλεσης επί των κύκλων της συνολικής εφαρμογής.

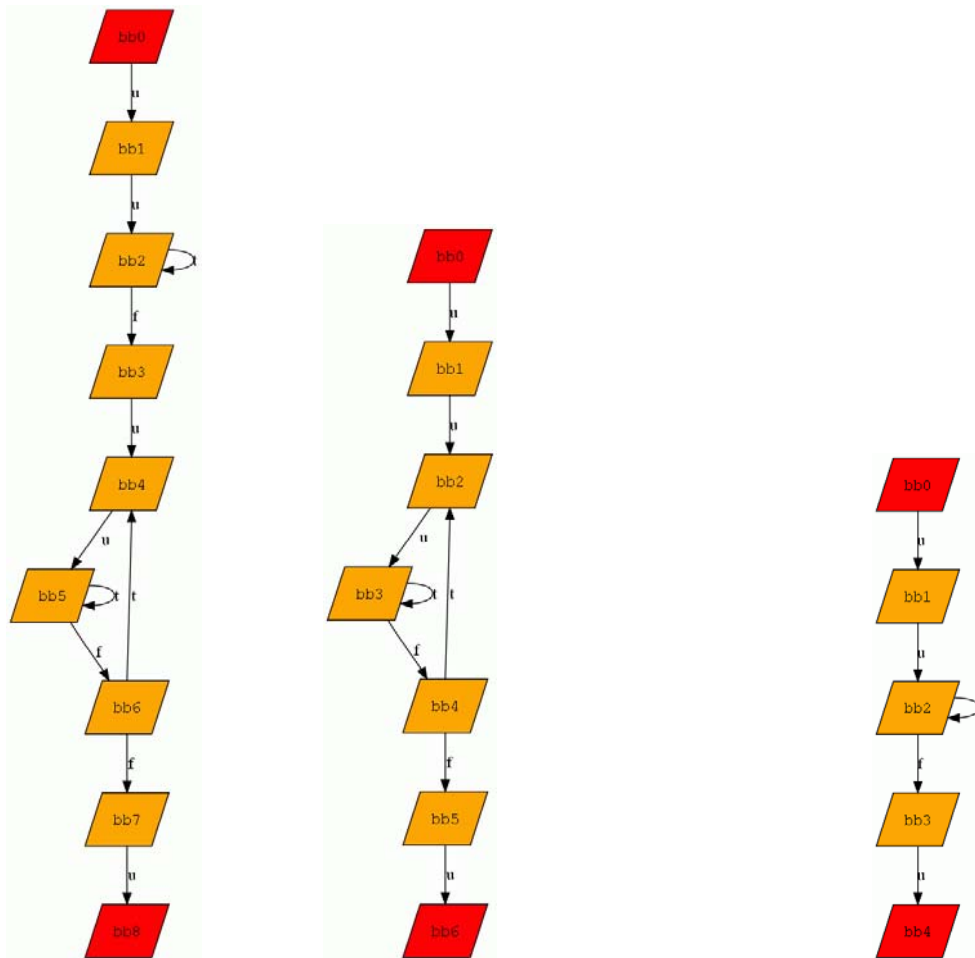
Παρατηρώντας την τελευταία στήλη του Πίνακα 2.2 (% κύκλων BB επί της συνολικής εφαρμογής) διαπιστώνουμε ότι κατά μέσο όρο, το κρίσιμότερο βασικό μπλοκ καταλαμβάνει το 95%. Αυτό το ποσοστό είναι ιδιαίτερα υψηλό και οφείλεται στα χαρακτηριστικά των εφαρμογών, και πιο συγκεκριμένα στο ότι υλοποιούν επεξεργασία πάνω σε πολυδιάστατες σειρές δεδομένων (εικόνες). Επίσης το μέγεθος των βασικών μπλοκ είναι αρκετά μεγάλο (49, το οποίο είναι πολύ πάνω από τη μέση τιμή του ~5 που έχει παρατηρηθεί εμπειρικά για τις εφαρμογές γενικού σκοπού) αλλά αυτό είναι κάτι που καθορίζεται εν μέρει επίσης από τη χρησιμοποιούμενη IR. Η χρήση μετασχηματισμού if-conversion συνεισφέρει στη συνένωση βασικών μπλοκ με λειτουργίες προδικασμού (predication) όπως στη περίπτωση αυτή είναι οι εντολές select. Για τον ίδιο λόγο χρησιμοποιούνται και τεχνικές ξετυλίγματος βρόχων.

Πίνακας 2.2: Γενικά χαρακτηριστικά των «κρίσιμων» (hot) βασικών μπλοκ του λογισμικού εφαρμογών του IPCN. Απεικονίζονται τα σημαντικότερα BB για κάθε εφαρμογή (λογίζονται ως αυτά που συνεισφέρουν άνω του 5% των συνολικών κύκλων εκτέλεσης).

Εφαρμογή	Όνομα υπορουτίνας	Αύξων αριθμός BB	Αριθμός εντολών	Αριθμός εκτιμώμενων δυναμικών κύκλων	% συνολικής εφαρμογής
fsdither	fsdither	5	59	237568	85.19

fsdither	fsdither	2	10	40960	14.69
hilcurv	main	3	30	712704	94.56
hilcurv	main	4	10	40960	5.43
htpack	htpack	2	80	40448	99.98
htunpack	htunpack	2	88	44032	99.98
xteaenc	encode_xtea	2	44	638976	95.63
xteaenc	main	2	59	29184	4.37
xteadec	decode_xtea	2	44	638976	94.76
xteadec	main	2	71	35328	5.24

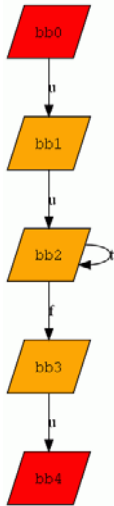
Στο σημείο αυτό, και για την καλύτερη εποπτεία των εφαρμογών του IPCN δίνονται πρώτα στο Σχήμα 2.4 οι CFG για τις συναρτήσεις των εφαρμογών που περιλαμβάνουν τα κρίσιμα βασικά μπλοκ, και έπειτα στο Σχήμα 2.5 οι DDG των 10 κρίσιμων βασικών μπλοκ.



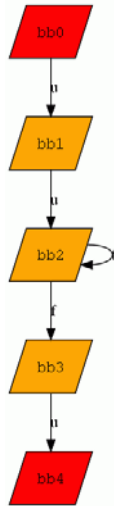
(α) fsdither:fsdither.

(β) hilcurv:main.

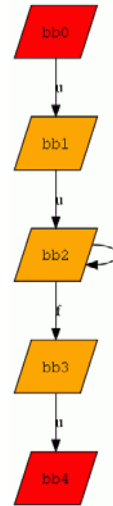
(γ) htpack:htpack.



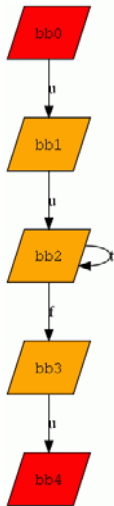
(δ) htunpack:htunpack.



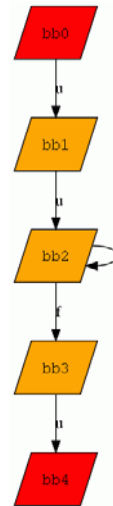
(ε) xteaenc:encode_xtea.



(στ) xteaenc:main.



(ζ) xteadec:decode_xtea.

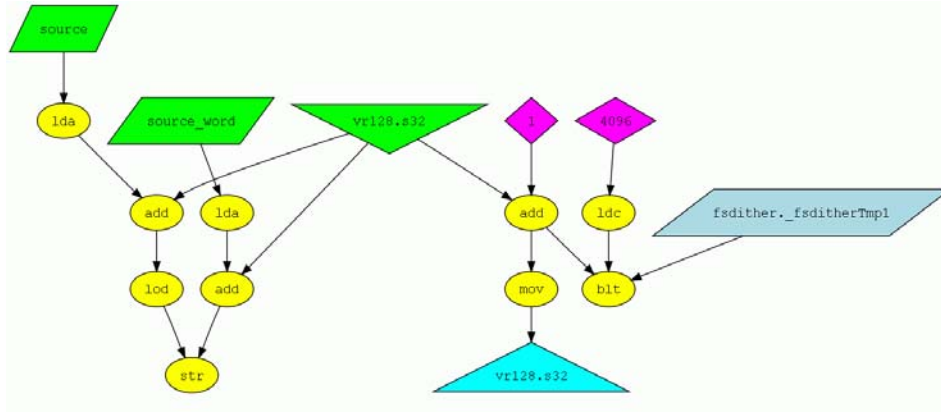


(η) xteadec:main.

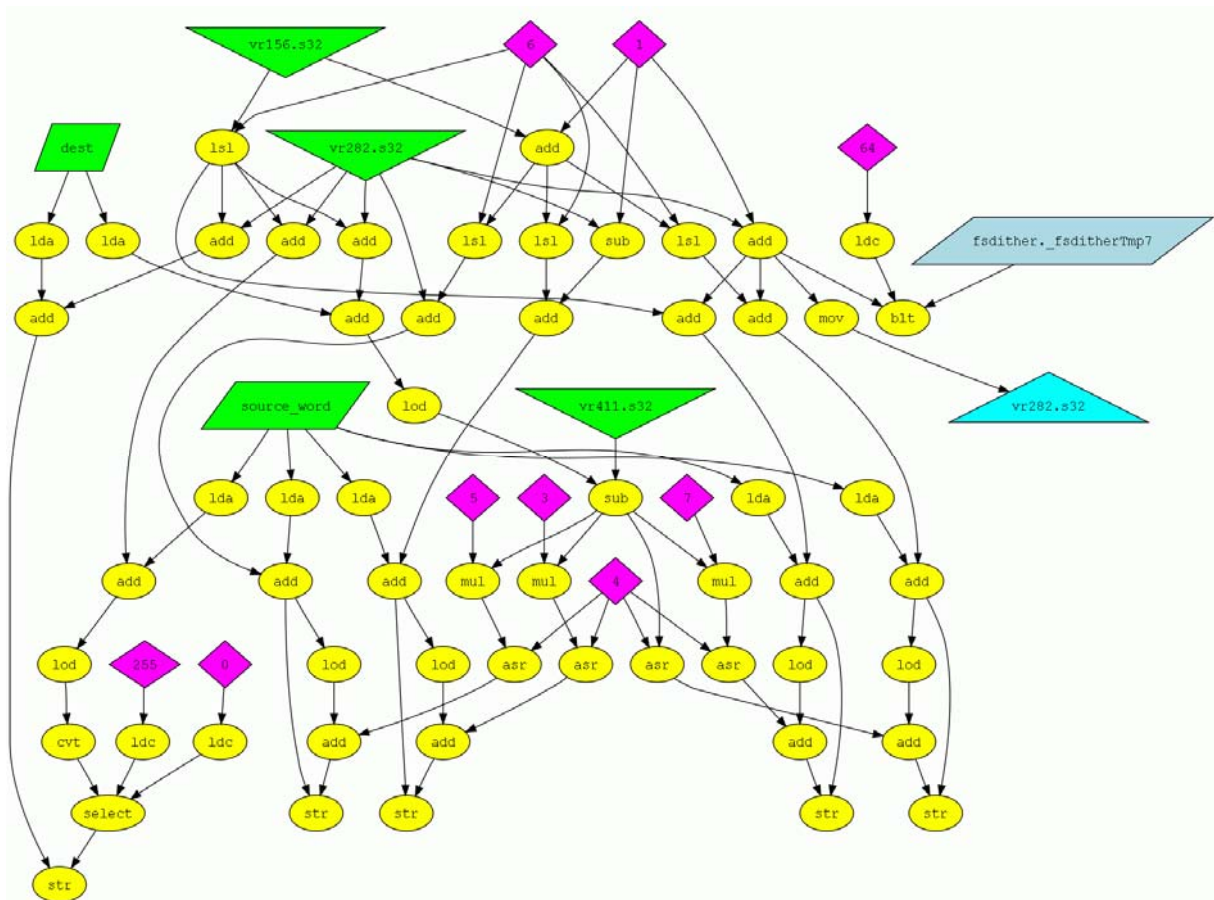
Σχήμα 2.4: Οπτικοποιημένοι γράφοι ροής ελέγχου (CFGs) για συναρτήσεις των εφαρμογών του IPCN οι οποίες περιλαμβάνουν τα κρίσιμα βασικά μπλοκ. Τα ονόματα των συναρτήσεων κωδικοποιούνται ως εξής: translation-unit:procedure. Μία μονάδα μετάφρασης (translation unit) αντιστοιχεί σε ένα αρχείο πηγαίου κώδικα C και περιλαμβάνει (στην περίπτωση μας) μια ολόκληρη εφαρμογή. Για τη χρωματική απόδοση χρησιμοποιούνται οι εξής επιλογές: ψευδοκόμβος πηγής/προορισμού (κόκκινο/διαμάντι), βασικό μπλοκ (πορτοκαλί/διαμάντι). Οι ακμές ελέγχου φέρουν μία από τις ετικέτες “u” (χωρίς-συνθήκη), “t” (αληθές) και “f” (ψευδές).

Παρατηρώντας τα CFG του Σχήματος 2.4 μπορούμε άμεσα να διακρίνουμε τα CFG με διπλά (περιπτώσεις α,β) και απλά φωλιασμένους βρόχους (γ-η). Σε κάθε περίπτωση σημειώνονται και οι συμβολικοί κόμβοι πηγής (source) και καταβόθρας (sink) που χρησιμοποιούνται κατά σύμβαση από αλγορίθμους ανάλυσης ροής ελέγχου.

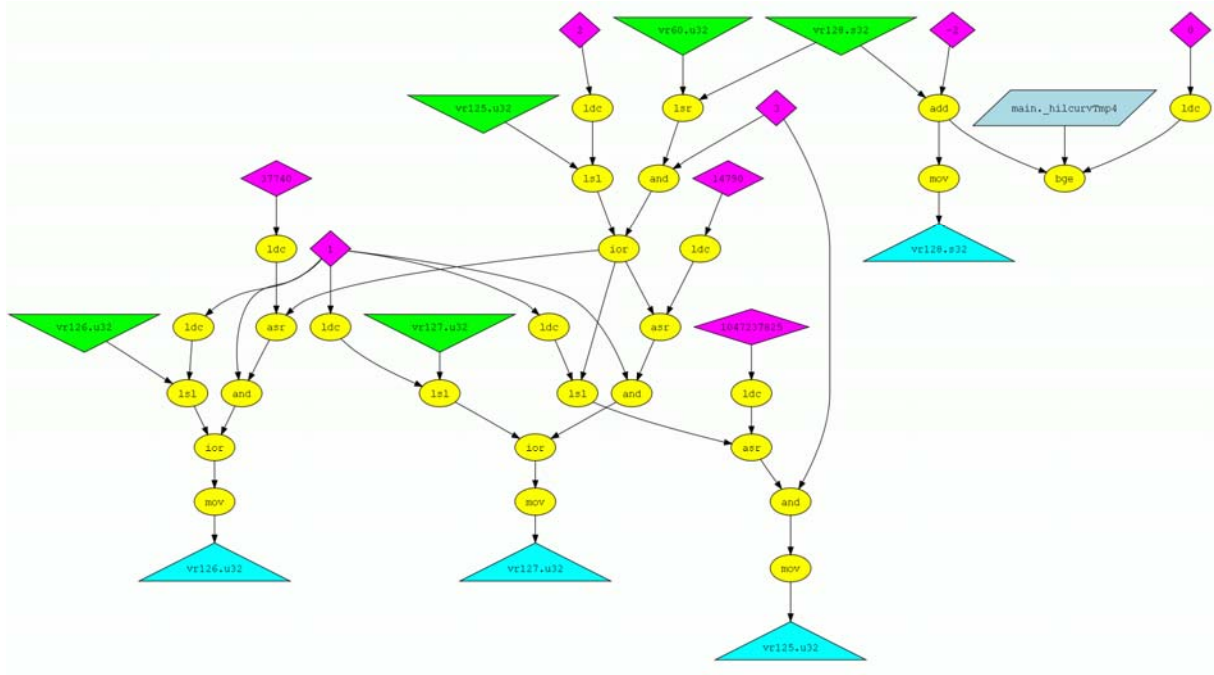
Στο Σχήμα 2.4 απεικονίζονται τα DDG για επιλεγμένα βασικά μπλοκ ορισμένων εφαρμογών. Για τη διάταξη και οπτικοποίηση χρησιμοποιείται το λογισμικό VCG. Για την απεικόνιση χρησιμοποιούνται διαφορετικοί χρωματισμοί και σχήματα για εντολές, ορίσματα εισόδου/εξόδου και σταθερές.



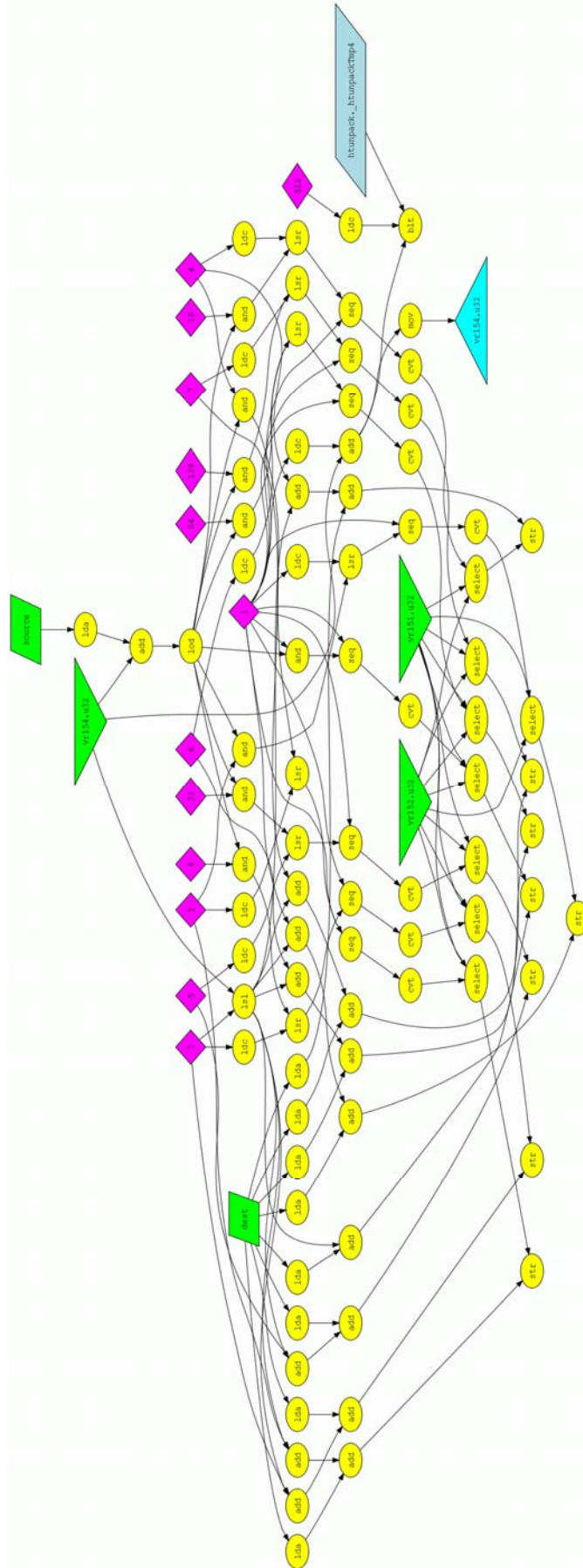
(α) fsdither:fsdither:2.



(β) fsdither:fsdither:5.



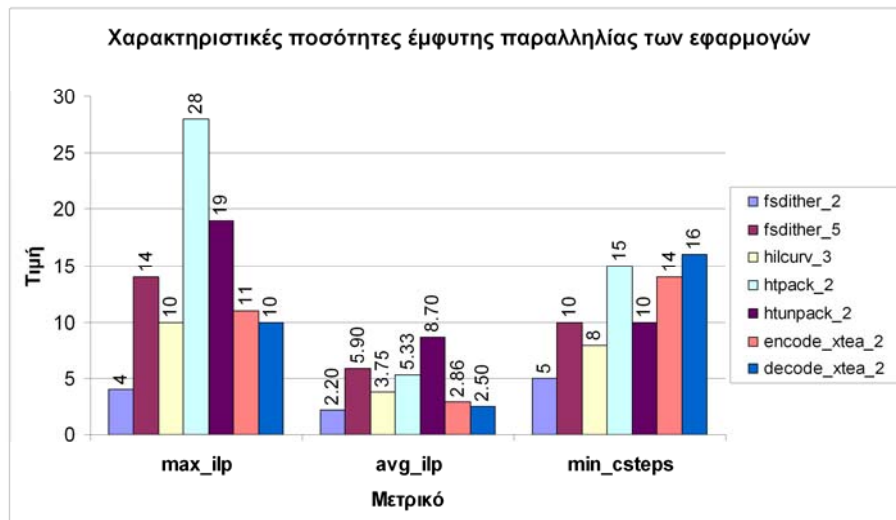
(γ) hilcurv:main:3.



(ε) htunpack:htunpack:2.

Σχήμα 2.5: Οπτικοποιημένα DDG για επιλεγμένα κρίσιμα βασικά μπλοκ από το λογισμικό εφαρμογών του IPCN. Η διάταξη των γράφων έγινε με το πρόγραμμα VCG. Η χρωματική/σχηματική αποτύπωση γίνεται με τις εξής επιλογές: εντολές (κίτρινο/έλλειψη), ορίσματα εισόδου (πράσινο/κατωφέρές τρίγωνο), σταθερές (πορφυρό/ρόμβος), ορίσματα εξόδου (κυανό/ανωφέρές τρίγωνο), ετικέτες (γκρι/παράλληλόγραμμο), συμβολικές διευθύνσεις (πράσινο/παράλληλόγραμμο).

Στη συνέχεια εξήχθησαν αρχεία `iseqinfo` για κάθε βασικό μπλοκ ξεχωριστά και έγινε αυτόματη μετάφραση σε αναπαράσταση CDFG. Τα CDFG αυτά χρονοδρομολογήθηκαν κατά ASAP με εργαλείο `asapalar`. Για κάθε BB μπορούν να υπολογιστούν τα μετρικά: `max_ilp`, `min_csteps`, και `avg_ilp`. Στο Σχήμα 2.6 δίνονται αναλυτικά οι τρεις ποσότητες όπως αυτές υπολογίζονται για τα κρίσιμα βασικά μπλοκ των εφαρμογών που εκτελούνται στον ΕΕΣ του IPCN.



Σχήμα 2.6: Χαρακτηριστικά χρονοδρομολόγησης των «κρίσιμων» (hot) βασικών μπλοκ του λογισμικού εφαρμογών του IPCN. Απεικονίζεται η μέγιστη παραλληλία (`max_ilp`), μέση παραλληλία (`avg_ilp`), και ελάχιστος αριθμός βημάτων ελέγχου (`min_csteps`) για κάθε BB.

Μια παρατήρηση είναι ότι με εξαίρεση την εφαρμογή `fsdither`, η μέγιστη χρήσιμη παραλληλία στο υλικό είναι πάνω από 10 το οποίο δείχνει το βαθμό ανεξαρτησίας μεταξύ των αντίστοιχων λειτουργιών που μπορούν να δρομολογηθούν στο ίδιο βήμα ελέγχου.

2.4. Καθορισμός του γενικού αρχιτεκτονικού περιγράμματος, γέννηση και επιλογή εντολών

Στο επόμενο βήμα της μεθοδολογίας γίνεται καθορισμός² του αρχιτεκτονικού περιγράμματος του επεξεργαστή. Το αρχιτεκτονικό περίγραμμα βασίζεται στην αρχιτεκτονική ByoRISC με τις εξής επισημάνσεις:

- δεν χρησιμοποιούνται τοπικοί συνεπεξεργαστές (`HAVE_COP=0`)
- δεν χρησιμοποιείται η πρόωρη σηματοδότηση διακλάδωσης (`BRANCH_EARLY=0`)
- ισχύουν οι εξής τιμές παραμέτρων: `IMEMSIZE=8KB`, `DMEMSIZE=8KB` (στον

² Για τον οποίο θα μπορούσε να διεξαχθεί και διερεύνηση του πολυδιάστατου αρχιτεκτονικού πεδίου λύσεων με αποτίμηση καμπυλών Pareto.

προσομοιωτή εκτίμησης κύκλων μπορούν να ρυθμιστούν μέχρι τα αρκετά MB), OW=8, RAW=8

- Οι παράμετροι NRP, NWP θα καθοριστούν από τα αποτελέσματα του επόμενου σταδίου της μεθοδολογίας (γένεση ειδικών εντολών).
- OPTIONAL_LS=1 καθώς χρησιμοποιούνται και άλλοι τύποι δεδομένων πλην των [U|S|P]32.
- OPTIONAL_DIV=0.

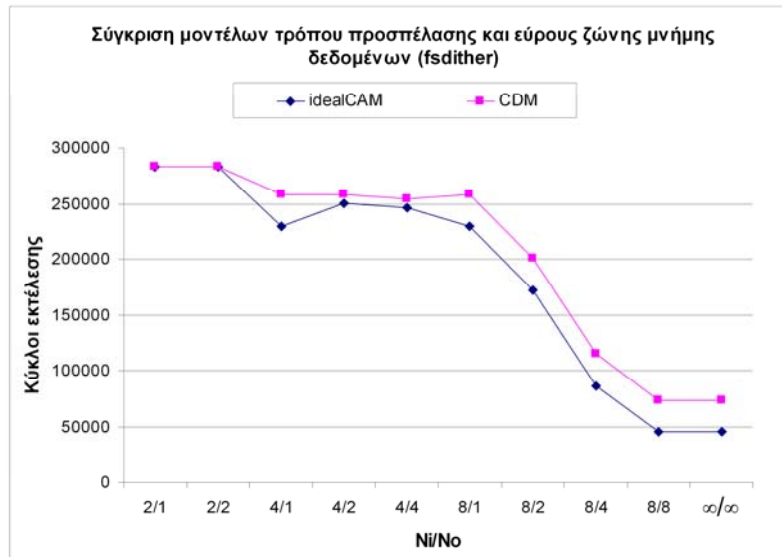
Για τη διαδικασία της γέννησης και επιλογής ειδικών εντολών, όλες οι CTI εντολές (`-forbid beqz bnez j jr jal`) έχουν αποκλειστεί από τη συμπερίληψή τους σε ειδικές εντολές για τον BgoRISC. Στη συνέχεια και κάτω από τις παραπάνω προϋποθέσεις, χρησιμοποιείται το εργαλείο YARDstick [Kav07] για τη γέννηση και επιλογή MIMO εντολών. Ο αντίστοιχος αλγόριθμος χρησιμοποιεί αναδρομική αναζήτηση πάνω σε δυαδικό δένδρο αποφάσεων, για το αν πρέπει να συμπεριληφθεί ή όχι ο τρέχων κόμβος ενός βασικού μπλοκ σε ειδική εντολή [Poz06], κάνοντας όμως χρήση μιας γρήγορης ευρεστικής τεχνικής, παρόμοιας με αυτή της εργασίας [Pot07]. Για την ενεργοποίηση του αλγορίθμου χρησιμοποιούνται οι επιλογές (`-mimo -fast`). Εντός των βασικών μπλοκ, οι κόμβοι δρομολογούνται με τοπολογική ταξινόμηση (`-topsort`). Για τη γέννηση ειδικών εντολών τύπου MIMO μπορεί να χρησιμοποιηθεί και ο αλγόριθμος γραμμικής πολυπλοκότητας του παραδοτέου Π4.1.

Τα εξαγόμενα MIMO μοτίβα φιλτράρονται μέσω ελέγχων ισομορφισμών γράφου (`-isomorph-vf2`) για την απομάκρυνση των πλεοναζόντων μοτίβων για τα οποία έχουν ήδη αναγνωριστεί ισομορφικά τους. Για την επιλογή ειδικών εντολών χρησιμοποιείται ένας άπληστος επιλογέας (`-iselect-cycgain-greedy-fastexit`) για τη συνάρτηση προτεραιότητας κέρδους κύκλων.

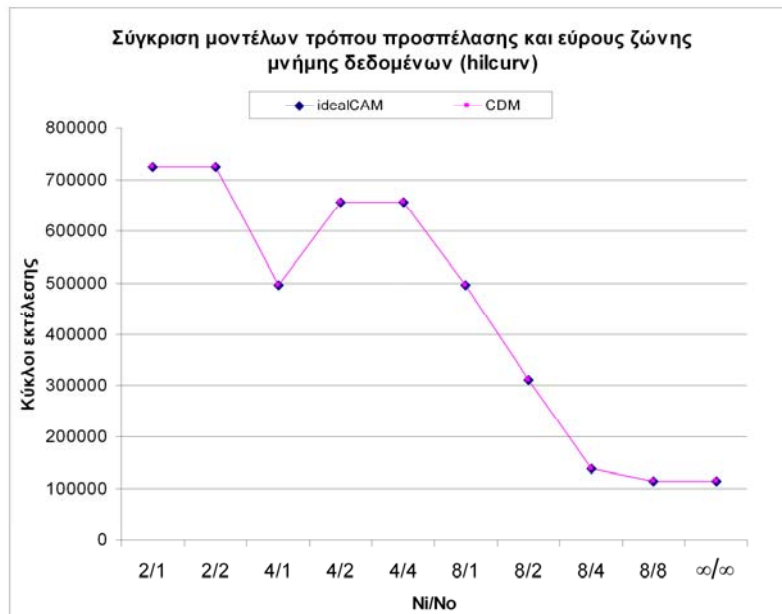
Η διερεύνηση διεξάγεται για δύο διαφορετικά μοντέλα προσπέλασης στη μνήμη δεδομένων:

- Το μοντέλο συνεπούς μνήμης δεδομένων, όπου η ΕΛΜ προσπελαίνει άμεσα τη μνήμη δεδομένων για το οποίο κάνουμε τη συντηρητική υπόθεση ότι οι εντολές φόρτωσης και αποθήκευσης πρέπει πάντοτε να είναι σε σειρά (`-memmodel 0`). Η εκδοχή αυτή σημειώνεται με "CDM".
- Το μοντέλο ιδανικής συνεπούς μνήμης ΕΛΜ, όπου κάθε φόρτωση/αποθήκευση προς/από τη μνήμη δεδομένων μετασχηματίζεται σε διαφανή προσπέλαση στην τοπική μνήμη ΕΛΜ (`-memmodel 1`). Η εκδοχή αυτή σημειώνεται με "idealCAM".

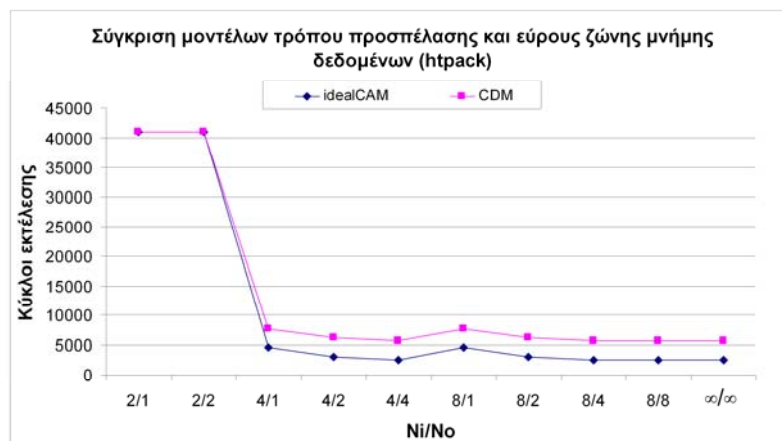
Τα αποτελέσματα της διερεύνησης των διαφορετικών μοντέλων προσπέλασης μνήμης δεδομένων για ενδεικτικούς συνδυασμούς αριθμού έντελων εισόδου/εξόδου (N_i/N_o) δίνονται στο Σχήμα 2.7 για τις 6 εφαρμογές του IPCN. Στον Πίνακα 2.3 δίνονται αναλυτικά οι μετρήσεις των σχετικών επιταχύνσεων για τις ίδιες εφαρμογές και στον Πίνακα 2.4 δίνεται η αυξητική επιτάχυνση για τις ειδικές εντολές με τη σημαντικότερη συνεισφορά, όπως αυτές προκύπτουν από την επιλογή ειδικών εντολών.



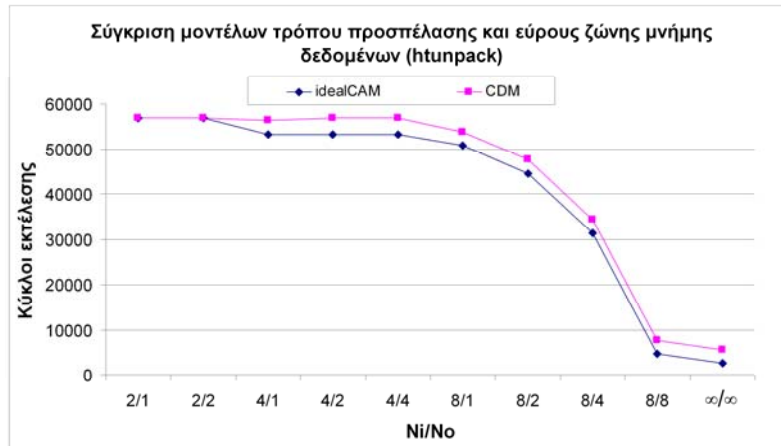
(α) fsdither.



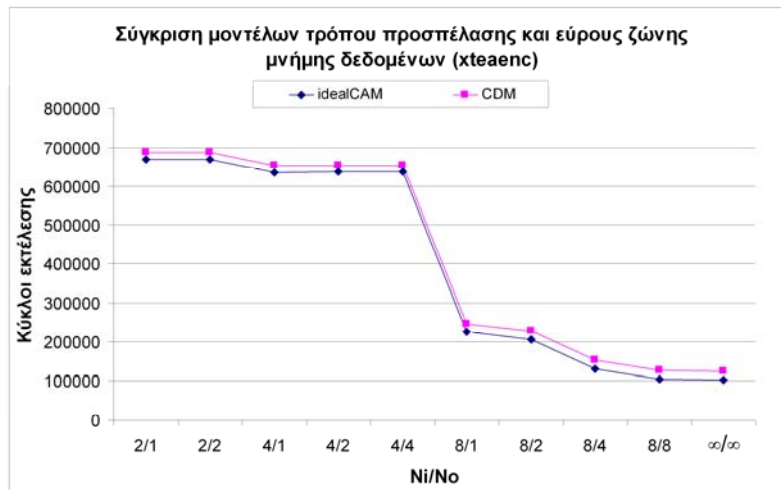
(β) hilcurv.



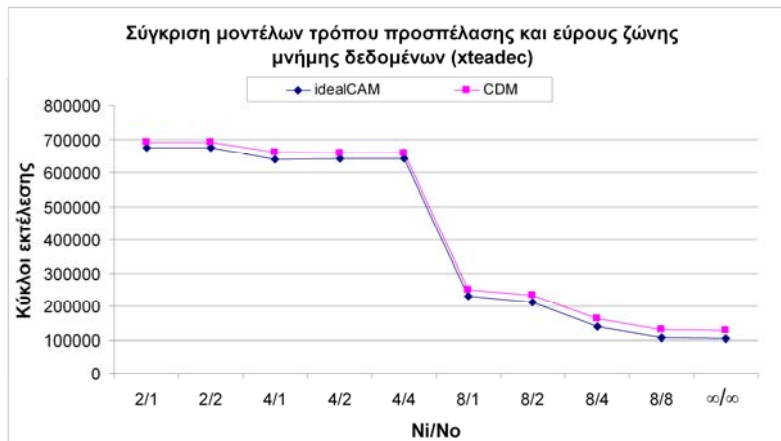
(γ) htpack.



(δ) htunpack.



(ε) xteaenc.



(στ) xteadec.

Σχήμα 2.7: Επίδραση των προσπελάσεων στη μνήμη δεδομένων πάνω στην επιτάχυνση εφαρμογής που οφείλεται στις ειδικές εντολές. Οι προσπελάσεις στη μνήμη δεδομένων απαιτούν επιβάρυνση ενός κύκλου μηχανής.

Πίνακας 2.3: Οι εκτιμήσεις που ελήφθησαν με το YARDstick για την επιτάχυνση εφαρμογών λόγω της χρήσης των εξαγόμενων ειδικών εντολών για το λογισμικό εφαρμογών του IPCN.

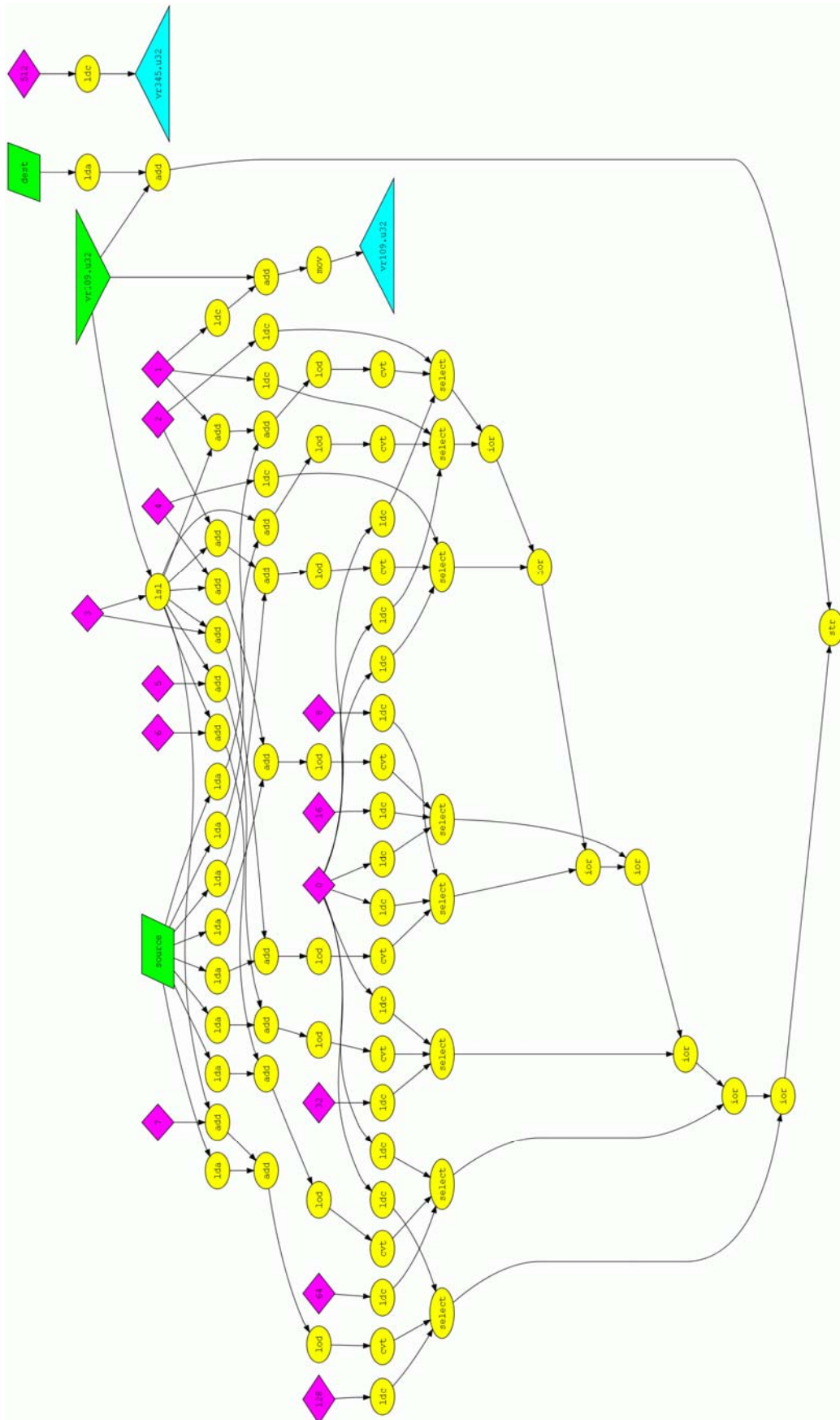
Παράμετρος			Εφαρμογή					
N_i	N_o	MemModel	fsdither	hilcurv	htpack	htunpack	xteaenc	xteadec
2	1	idealCAM	1.01	1.08	1.00	1.00	1.03	1.02
2	1	CDM	1.01	1.08	1.00	1.00	1.01	0.99
2	2	idealCAM	1.01	1.08	1.00	1.00	1.03	1.02
2	2	CDM	1.01	1.08	1.00	1.00	1.01	0.99
4	1	idealCAM	1.25	1.58	8.86	1.07	1.09	1.07
4	1	CDM	1.11	1.58	5.32	1.01	1.06	1.04
4	2	idealCAM	1.15	1.19	13.27	1.07	1.09	1.07
4	2	CDM	1.11	1.19	6.65	1.00	1.06	1.04
4	4	idealCAM	1.17	1.19	15.92	1.07	1.09	1.07
4	4	CDM	1.13	1.19	7.26	1.00	1.06	1.04
8	1	idealCAM	1.25	1.58	8.86	1.12	3.08	2.97
8	1	CDM	1.11	1.58	5.32	1.06	2.84	2.74
8	2	idealCAM	1.67	2.51	13.27	1.28	3.37	3.25
8	2	CDM	1.43	2.51	6.65	1.19	3.05	2.93
8	4	idealCAM	3.33	5.62	15.92	1.82	5.31	4.98
8	4	CDM	2.50	5.62	7.26	1.66	4.54	4.24
8	8	idealCAM	6.33	6.82	15.92	12.30	6.63	6.51
8	8	CDM	3.88	6.82	7.26	7.39	5.48	5.30
∞	∞	idealCAM	6.33	6.82	15.92	22.09	6.73	6.67
∞	∞	CDM	3.88	6.82	7.26	10.07	5.54	5.41

Πίνακας 2.4: Αυξητική επιτάχυνση εφαρμογών λόγω των ειδικών εντολών για την περίπτωση $N_i/N_o = 8/8$ και MemModel=CDM.

Κατάταξη ειδικής εντολής (κατά φθίνουσα σειρά σημαντικότητας)	Εφαρμογή					
	fsdither	hilcurv	htpack	htunpack	xteaenc	xteadec
1η ειδική εντολή	2.69	5.46	7.25	7.39	4.70	5.34
2η ειδική εντολή	3.87	6.82	NA	NA	5.43	5.37

Πίνακας 2.5: Χαρακτηριστικά των κυριότερων ειδικών εντολών όπως αυτές προέκυψαν για το λογισμικό εφαρμογών του IPCN.

Εφαρμογή	Κωδικοποίηση ειδικής εντολής	Χαρακτηριστικά DDG ειδικής εντολής						Κέρδος κύκλων	Αυξητική επιτάχυνση εφαρμογής	Κύκλοι SW	Κύκλοι HW	Εκτίμηση χρόνου HW	Επιφάνεια υλικού (MAU)
		N _n	N _e	N _i	N _o	N _s	N _c						
fsdither	fsdither_cfg0_bb5_ise8	57	64	5	2	2	9	180224	2.69	57	13	12.38	4.56
fsdither	fsdither_cfg0_bb2_ise2	9	6	3	2	2	2	32768	3.87	9	1	0.40	0.18
hilcurv	main_cfg0_bb3_ise2	28	25	5	5	0	8	638976	5.46	28	2	1.25	0.35
htpack	htpack_cfg0_bb2_ise2	78	82	3	2	2	14	35328	7.25	78	9	8.55	1.10
htunpack	htunpack_cfg0_bb2_ise1	81	87	5	0	2	12	36864	5.79	81	9	8.24	1.33
xteaenc	encode_xtea_cfg0_bb2_ise1	38	38	6	5	1	7	540672	4.55	38	5	4.08	0.83
xteadec	decode_xtea_cfg0_bb2_ise1	38	38	6	5	1	7	540672	4.70	38	5	4.52	0.78

(δ) `htpack_cfg0_bb2_ise2`

Σχήμα 2.8: Οπτικοποιημένα DDG των επιλεγμένων ειδικών εντολών για την επιτάχυνση του λογισμικού εφαρμογών του IPCN. Η διάταξη των γράφων έγινε με το πρόγραμμα Graphviz. Η χρωματική/σχηματική αποτύπωση γίνεται με τις επιλογές του Σχήματος 2.5.

Ο Πίνακας 2.6 συγκρίνει τις εκτιμήσεις υψηλού επιπέδου όπως προκύπτουν από την ανάλυση των εφαρμογών και την γέννηση ειδικών εντολών από το επίπεδο της SUIFvm IR με τις μετρήσεις αριθμού κύκλων εκτέλεσης στο προσομοιωτή ArchC για τις εφαρμογές του IPCN. Παρατηρούμε ότι λόγω της προσθήκης των επιλεγμένων ειδικών εντολών έχουμε μια εκτιμώμενη επιτάχυνση ίση με περίπου 6 φορές (6.02) ενώ η πραγματική αποτιμάται σε λίγο μεγαλύτερη από 5 (5.09). Εξαιτίας της έλλειψης επαναστοχεύσιμου μεταγλωττιστή (η προσθήκη υποστήριξης ενός νέου επεξεργαστή συνήθως απαιτεί πολλούς ανθρωπομήνες ενασχόλησης όσον αφορά υποδομές μεταγλωττιστή όπως οι GCC, LLVM και SUIF/MachSUIF), οι εφαρμογές του IPCN μεταγλωττίστηκαν σε κώδικα assembly του ByoRISC χειρωνακτικά με βάση τον κώδικα στην C και την εμπειρία του προγραμματιστή. Παρά την ύπαρξη αυτής της πηγής ανομοιογένειας και το γεγονός ότι μία IR μεταγλωττιστή (οι θεμελιώδεις λειτουργίες της) δεν έχει απαραίτητα ένα-προς-ένα ταύτιση με εντολές επιπέδου συμβολομεταφραστή, κατά μέσο όρο η εκτίμηση της επιτάχυνσης εφαρμογής αποκλίνει κατά 9.1% σε σχέση με τις τιμές που προκύπτουν από την προσομοίωση του επεξεργαστή ByoRISC. Είναι σημαντικό το γεγονός ότι η απόκλιση αυτή είναι μονότονη, δηλαδή σε όλες τις περιπτώσεις η εκτίμηση επιτάχυνσης εφαρμογής είναι μεγαλύτερη (κατά 1.3% ως 16.8%) από την επιτάχυνση που δίνει ο προσομοιωτής. Μάλιστα για ορισμένες εφαρμογές, ο μετασχηματισμός της ενδιάμεσης αναπαράστασης από μορφή CFG σε CFG/SSA και πάλι σε CFG (για την απομάκρυνση εγχύσεων/ πληρώσεων προσωρινών μεταβλητών προς/από τη μνήμη δεδομένων και τη μετάθεσή τους σε προσωρινούς καταχωρητές) συνεισφέρει σε εντολές μετακίνησης (move). Αυτές οι εντολές μετακίνησης (σε ορισμένες εφαρμογές το φαινόμενο είναι εντονότερο από ότι σε άλλες) όταν συμπεριλαμβάνονται σε ειδικές εντολές, συντελούν σε μεγαλύτερες επιταχύνσεις λόγω της (σχεδόν) μηδενικής τους συνεισφοράς στο κρίσιμο μονοπάτι καθυστέρησης του υλικού που υλοποιεί την ειδική εντολή. Αντίθετα αυτή η επίδραση αυτή δεν εμφανίζεται απαραίτητα όταν ο προγραμματιστής συντάσσει τον κώδικα συμβολομεταφραστή χειρωνακτικά και χωρίς να πραγματοποιεί την μετατροπή σε μορφή CFG/SSA.

Πίνακας 2.6: Σύγκριση εκτιμώμενων-πραγματικών κύκλων εκτέλεσης και επιταχύνσεων εφαρμογών για το λογισμικό εφαρμογών του IPCN.

	Benchmark						Αποτίμηση
	fsdither	hilcurv	htpack	htunpack	xteaenc	xteadec	
Αρχικοί κύκλοι εκτέλεσης (ArchC)	250248	725009	51421	57013	619033	588310	
Κύκλοι εκτέλεσης με ειδικές εντολές (ArchC-CI)	74133	143377	7183	8207	158230	144406	
Επιτάχυνση εφαρμογής (ArchC-CI)	3.38	5.06	7.16	6.95	3.91	4.07	Μέση τιμή: 5.09
Επιτάχυνση εφαρμογής (YARDstick-CI)	3.87	5.46	7.25	7.39	4.70	4.55	Μέση τιμή: 6.02
Κύκλοι εκτέλεσης με ειδικές εντολές και ZOLC (ArchC-CI-ZOLC)	37269	114705	6671	6671	141334	110614	
Επιτάχυνση εφαρμογής (ArchC-CI-ZOLC)	6.71	6.32	7.71	8.55	4.38	5.32	Μέση τιμή: 6.50
%σφάλμα μεταξύ ArchC-CI και YARDstick-CI	12.77	7.39	1.26	6.00	16.76	10.46	Μέσο σφάλμα: 9.1

Προκειμένου τη διευκόλυνση στην ανάπτυξη προσομοιωτών (αλλά και άλλων εργαλείων ακόμη και μεταγλωττιστών), ώστε να ενημερώνονται με τις περιγραφές των ειδικών εντολών, μπορεί να παραχθεί αυτόματα, από κάθε ειδική εντολή, πρωτότυπος κώδικας

προσομοίωσης με εκτίμηση αριθμού κύκλων (υπό μορφή ντιρεκτίβας #pragma). Στον Κώδικα 2.2 δίνονται οι μακροεντολές για όλους τους επιτρεπόμενους τρόπους φόρτωσης και αποθήκευσης τιμών από και προς τη μνήμη δεδομένων (DM) του ArchC μοντέλου του επεξεργαστή ByoRISC. Στον Κώδικα 2.3 δίνονται οι ANSI C κώδικες για την προσομοίωση των ειδικών εντολών του Πίνακα 2.5 και Σχήματος 2.8.

```
#define LODB(address)      (DM.read_byte(address))
#define LODBU(address)    ((unsigned char)DM.read_byte(address))
#define LODH(address)    (DM.read_half(address))
#define LODHU(address)   ((unsigned short)DM.read_half(address))
#define LODW(address)    (DM.read(address))
#define STRB(address,data) (DM.write_byte(address,data))
#define STRH(address,data) (DM.write_half(address,data))
#define STRW(address,data) (DM.write(address,data))
```

Κώδικας 2.2: Μακροαντικαταστάσεις για την προσομοίωση λειτουργιών φόρτωσης και αποθήκευσης με το ArchC μοντέλο του επεξεργαστή ByoRISC.

```
void fsdither_5(
    int *d0
    ,int      *d1
    ,int      s0
    ,int      s1
    ,int      s2
    ,unsigned int * s3
    ,unsigned int * s4
)
{
    int vr403_s32;
    int vr404_s32;
    unsigned int * vr405_p32;
    unsigned int * vr408_p32;
    unsigned char vr409_u8;
    unsigned int  vr412_u32;
    int vr515_s32;
    int vr516_s32;
    int vr416_s32;
    int vr421_s32;
    unsigned int * vr422_p32;
    unsigned int * vr425_p32;
    int vr428_s32;
    unsigned int * vr429_p32;
    unsigned int * vr432_p32;
    unsigned char vr433_u8;
    int vr436_s32;
    int vr440_s32;
    int vr441_s32;
    unsigned int * vr442_p32;
    unsigned int * vr445_p32;
    unsigned char vr446_u8;
    int vr449_s32;
    int vr451_s32;
    int vr452_s32;
    int vr455_s32;
    int vr457_s32;
    int vr459_s32;
    int vr460_s32;
    unsigned int * vr461_p32;
    unsigned int * vr464_p32;
    unsigned char vr465_u8;
    int vr468_s32;
    int vr470_s32;
    int vr471_s32;
    int vr476_s32;
    int vr477_s32;
    unsigned int * vr478_p32;
    unsigned int * vr481_p32;
    unsigned char vr482_u8;
    int vr485_s32;
    int vr487_s32;
```



```
int vr488_s32;
int vr493_s32;
int vr496_s32;
unsigned int *   vr497_p32;
unsigned int *   vr500_p32;
unsigned char    vr501_u8;
int vr504_s32;
int vr505_s32;
*d1 = 64;
vr497_p32 = (void *)&s3;
vr478_p32 = (void *)&s3;
vr461_p32 = (void *)&s3;
vr459_s32 = s0-1;
vr455_s32 = s2+1;
vr493_s32 = vr455_s32<<6;
vr476_s32 = vr455_s32<<6;
vr477_s32 = vr476_s32+s0;
vr481_p32 = vr478_p32+vr477_s32;
vr482_u8 = LODBU(vr481_p32);
vr457_s32 = vr455_s32<<6;
vr460_s32 = vr457_s32+vr459_s32;
vr464_p32 = vr461_p32+vr460_s32;
vr465_u8 = LODBU(vr464_p32);
vr442_p32 = (void *)&s3;
vr440_s32 = s0+1;
*d0 = vr440_s32;
vr496_s32 = vr493_s32+vr440_s32;
vr500_p32 = vr497_p32+vr496_s32;
vr501_u8 = LODBU(vr500_p32);
vr429_p32 = (void *)&s4;
vr422_p32 = (void *)&s4;
vr516_s32 = 0;
vr515_s32 = 255;
vr405_p32 = (void *)&s3;
vr403_s32 = s2<<6;
vr441_s32 = vr403_s32+vr440_s32;
vr445_p32 = vr442_p32+vr441_s32;
vr446_u8 = LODBU(vr445_p32);
vr428_s32 = vr403_s32+s0;
vr432_p32 = vr429_p32+vr428_s32;
vr433_u8 = LODBU(vr432_p32);
vr436_s32 = s1-vr433_u8;
vr504_s32 = vr436_s32>>4;
vr505_s32 = vr501_u8+vr504_s32;
STR(vr500_p32, vr505_s32);
vr485_s32 = vr436_s32*5;
vr487_s32 = vr485_s32>>4;
vr488_s32 = vr482_u8+vr487_s32;
STR(vr481_p32, vr488_s32);
vr468_s32 = vr436_s32*3;
vr470_s32 = vr468_s32>>4;
vr471_s32 = vr465_u8+vr470_s32;
STR(vr464_p32, vr471_s32);
vr449_s32 = vr436_s32*7;
vr451_s32 = vr449_s32>>4;
vr452_s32 = vr446_u8+vr451_s32;
STR(vr445_p32, vr452_s32);
vr421_s32 = vr403_s32+s0;
vr425_p32 = vr422_p32+vr421_s32;
vr404_s32 = vr403_s32+s0;
vr408_p32 = vr405_p32+vr404_s32;
vr409_u8 = LODBU(vr408_p32);
vr412_u32 = (unsigned char)vr409_u8;
if (vr412_u32) {
    vr416_s32 = vr515_s32;
} else {
    vr416_s32 = vr516_s32;
}
STR(vr425_p32, vr416_s32);
#pragma cycles_est_total = 13
}
```

(α) fsdither_cfg0_bb5_ise8

```
void fsdither_2(
```



```
int *d0
,int      *d1
,int      s0
,unsigned int * s1
,unsigned int * s2
)
{
unsigned int * vr142_p32;
unsigned int * vr145_p32;
unsigned char vr146_u8;
unsigned int * vr148_p32;
unsigned int * vr151_p32;
int vr153_s32;
*d1 = 4096;
vr153_s32 = s0+1;
*d0 = vr153_s32;
vr148_p32 = (void *)&s2;
vr151_p32 = vr148_p32+s0;
vr142_p32 = (void *)&s1;
vr145_p32 = vr142_p32+s0;
vr146_u8 = LOD(vr145_p32);
STRB(vr151_p32,vr146_u8);
#pragma cycles_est_total = 1
}
```

(β) fsdither_cfg0_bb2_ise2

```
void main_3(
unsigned int      *d0
,unsigned int      *d1
,unsigned int      *d2
,int              *d3
,int              *d4
,unsigned int      s0
,unsigned int      s1
,int              s2
,unsigned int      s3
,unsigned int      s4
)
{
int vr170_s32;
unsigned int      vr172_u32;
unsigned int      vr174_u32;
unsigned int      vr176_u32;
unsigned int      vr177_u32;
int vr178_s32;
unsigned int      vr180_u32;
int vr181_s32;
int vr183_s32;
int vr185_s32;
unsigned int      vr187_u32;
int vr188_s32;
unsigned int      vr190_u32;
int vr191_s32;
int vr193_s32;
int vr195_s32;
unsigned int      vr197_u32;
int vr198_s32;
int vr199_s32;
unsigned int      vr201_u32;
int vr203_s32;
int vr205_s32;
int vr208_s32;
*d4 = 0;
vr208_s32 = s2+(-2);
*d3 = vr208_s32;
vr199_s32 = 1;
vr198_s32 = 1047237825;
vr191_s32 = 14790;
vr188_s32 = 1;
vr190_u32 = s4<<vr188_s32;
vr181_s32 = 37740;
vr178_s32 = 1;
vr180_u32 = s3<<vr178_s32;
```



```
vr174_u32 = ((unsigned long)s1) >>s2;
vr176_u32 = vr174_u32&3;
vr170_s32 = 2;
vr172_u32 = s0<<vr170_s32;
vr177_u32 = vr172_u32|vr176_u32;
vr201_u32 = vr177_u32<<vr199_s32;
vr203_s32 = vr198_s32>>vr201_u32;
vr205_s32 = vr203_s32&3;
*d2 = vr205_s32;
vr193_s32 = vr191_s32>>vr177_u32;
vr195_s32 = vr193_s32&1;
vr197_u32 = vr190_u32|vr195_s32;
*d1 = vr197_u32;
vr183_s32 = vr181_s32>>vr177_u32;
vr185_s32 = vr183_s32&1;
vr187_u32 = vr180_u32|vr185_s32;
*d0 = vr187_u32;
#pragma cycles_est_total = 2
}
```

(γ) hilcurv_cfg0_bb3_ise2

```
void htpack_2(
    unsigned int      *d0
    ,unsigned int      *d1
    ,unsigned int      s0
    ,unsigned int *    s1
    ,unsigned int *    s2
)
{
    unsigned int      vr234_u32;
    unsigned int *    vr236_p32;
    unsigned int *    vr239_p32;
    unsigned char     vr240_u8;
    unsigned int      vr242_u32;
    unsigned int      vr244_u32;
    unsigned int *    vr245_p32;
    unsigned int *    vr248_p32;
    unsigned char     vr249_u8;
    unsigned int      vr251_u32;
    unsigned int      vr253_u32;
    unsigned int *    vr254_p32;
    unsigned int *    vr257_p32;
    unsigned char     vr258_u8;
    unsigned int      vr260_u32;
    unsigned int      vr262_u32;
    unsigned int *    vr263_p32;
    unsigned int *    vr266_p32;
    unsigned char     vr267_u8;
    unsigned int      vr269_u32;
    unsigned int      vr271_u32;
    unsigned int *    vr272_p32;
    unsigned int *    vr275_p32;
    unsigned char     vr276_u8;
    unsigned int      vr278_u32;
    unsigned int      vr280_u32;
    unsigned int *    vr281_p32;
    unsigned int *    vr284_p32;
    unsigned char     vr285_u8;
    unsigned int      vr287_u32;
    unsigned int      vr289_u32;
    unsigned int *    vr290_p32;
    unsigned int *    vr293_p32;
    unsigned char     vr294_u8;
    unsigned int      vr296_u32;
    unsigned int      vr298_u32;
    unsigned int *    vr299_p32;
    unsigned int *    vr302_p32;
    unsigned char     vr303_u8;
    unsigned int      vr305_u32;
    unsigned int      vr362_u32;
    unsigned int      vr363_u32;
    unsigned int      vr307_u32;
    unsigned int      vr364_u32;
```

```
unsigned int      vr365_u32;
unsigned int      vr310_u32;
unsigned int      vr366_u32;
unsigned int      vr367_u32;
unsigned int      vr313_u32;
unsigned int      vr368_u32;
unsigned int      vr369_u32;
unsigned int      vr316_u32;
unsigned int      vr370_u32;
unsigned int      vr371_u32;
unsigned int      vr319_u32;
unsigned int      vr372_u32;
unsigned int      vr373_u32;
unsigned int      vr322_u32;
unsigned int      vr374_u32;
unsigned int      vr375_u32;
unsigned int      vr325_u32;
unsigned int      vr376_u32;
unsigned int      vr377_u32;
unsigned int      vr328_u32;
unsigned int      vr330_u32;
unsigned int      vr331_u32;
unsigned int      vr332_u32;
unsigned int      vr333_u32;
unsigned int      vr334_u32;
unsigned int      vr335_u32;
unsigned int      vr336_u32;
unsigned int *    vr338_p32;
unsigned int *    vr341_p32;
int vr342_s32;
unsigned int      vr344_u32;
*d1 = 512;
vr342_s32 = 1;
vr344_u32 = s0+vr342_s32;
*d0 = vr344_u32;
vr338_p32 = (void *)&s2;
vr341_p32 = vr338_p32+s0;
vr377_u32 = 0;
vr376_u32 = 128;
vr375_u32 = 0;
vr374_u32 = 64;
vr373_u32 = 0;
vr372_u32 = 32;
vr371_u32 = 0;
vr370_u32 = 16;
vr369_u32 = 0;
vr368_u32 = 8;
vr367_u32 = 0;
vr366_u32 = 4;
vr365_u32 = 0;
vr364_u32 = 2;
vr363_u32 = 0;
vr362_u32 = 1;
vr299_p32 = (void *)&s1;
vr290_p32 = (void *)&s1;
vr281_p32 = (void *)&s1;
vr272_p32 = (void *)&s1;
vr263_p32 = (void *)&s1;
vr254_p32 = (void *)&s1;
vr245_p32 = (void *)&s1;
vr236_p32 = (void *)&s1;
vr234_u32 = s0<<3;
vr298_u32 = vr234_u32+7;
vr302_p32 = vr299_p32+vr298_u32;
vr303_u8 = LODBU(vr302_p32);
vr305_u32 = (unsigned char)vr303_u8;
if (vr305_u32) {
    vr328_u32 = vr376_u32;
} else {
    vr328_u32 = vr377_u32;
}
vr289_u32 = vr234_u32+6;
vr293_p32 = vr290_p32+vr289_u32;
vr294_u8 = LODBU(vr293_p32);
vr296_u32 = (unsigned char)vr294_u8;
if (vr296_u32) {
    vr325_u32 = vr374_u32;
```



```
} else {
    vr325_u32 = vr375_u32;
}
vr280_u32 = vr234_u32+5;
vr284_p32 = vr281_p32+vr280_u32;
vr285_u8 = LODBU(vr284_p32);
vr287_u32 = (unsigned char)vr285_u8;
if (vr287_u32) {
    vr322_u32 = vr372_u32;
} else {
    vr322_u32 = vr373_u32;
}
vr271_u32 = vr234_u32+4;
vr275_p32 = vr272_p32+vr271_u32;
vr276_u8 = LODBU(vr275_p32);
vr278_u32 = (unsigned char)vr276_u8;
if (vr278_u32) {
    vr319_u32 = vr370_u32;
} else {
    vr319_u32 = vr371_u32;
}
vr262_u32 = vr234_u32+3;
vr266_p32 = vr263_p32+vr262_u32;
vr267_u8 = LODBU(vr266_p32);
vr269_u32 = (unsigned char)vr267_u8;
if (vr269_u32) {
    vr316_u32 = vr368_u32;
} else {
    vr316_u32 = vr369_u32;
}
vr253_u32 = vr234_u32+2;
vr257_p32 = vr254_p32+vr253_u32;
vr258_u8 = LODBU(vr257_p32);
vr260_u32 = (unsigned char)vr258_u8;
if (vr260_u32) {
    vr313_u32 = vr366_u32;
} else {
    vr313_u32 = vr367_u32;
}
vr244_u32 = vr234_u32+1;
vr248_p32 = vr245_p32+vr244_u32;
vr249_u8 = LODBU(vr248_p32);
vr251_u32 = (unsigned char)vr249_u8;
if (vr251_u32) {
    vr310_u32 = vr364_u32;
} else {
    vr310_u32 = vr365_u32;
}
vr239_p32 = vr236_p32+vr234_u32;
vr240_u8 = LODBU(vr239_p32);
vr242_u32 = (unsigned char)vr240_u8;
if (vr242_u32) {
    vr307_u32 = vr362_u32;
} else {
    vr307_u32 = vr363_u32;
}
vr330_u32 = vr307_u32|vr310_u32;
vr331_u32 = vr330_u32|vr313_u32;
vr332_u32 = vr331_u32|vr316_u32;
vr333_u32 = vr332_u32|vr319_u32;
vr334_u32 = vr333_u32|vr322_u32;
vr335_u32 = vr334_u32|vr325_u32;
vr336_u32 = vr335_u32|vr328_u32;
STR(vr341_p32,vr336_u32);
#pragma cycles_est_total = 9
}
```

(δ) htpack_cfg0_bb2_ise2

```
void htunpack_2(
    unsigned int    s0
    ,unsigned int    s1
    ,unsigned int    s2
    ,unsigned int *  s3
```



```
,unsigned int *   s4
)
{
unsigned int *    vr330_p32;
unsigned int *    vr333_p32;
unsigned char     vr334_u8;
unsigned int      vr338_u32;
unsigned int      vr340_u32;
int vr341_s32;
unsigned int      vr343_u32;
unsigned int      vr345_u32;
int vr346_s32;
unsigned int      vr348_u32;
unsigned int      vr350_u32;
int vr351_s32;
unsigned int      vr353_u32;
unsigned int      vr355_u32;
int vr356_s32;
unsigned int      vr358_u32;
unsigned int      vr360_u32;
int vr361_s32;
unsigned int      vr363_u32;
unsigned int      vr365_u32;
int vr366_s32;
unsigned int      vr368_u32;
unsigned int      vr370_u32;
int vr371_s32;
unsigned int      vr373_u32;
int vr375_s32;
unsigned int      vr376_u32;
int vr378_s32;
unsigned int      vr379_u32;
int vr381_s32;
unsigned int      vr382_u32;
int vr384_s32;
unsigned int      vr385_u32;
int vr387_s32;
unsigned int      vr388_u32;
int vr390_s32;
unsigned int      vr391_u32;
int vr393_s32;
unsigned int      vr394_u32;
int vr396_s32;
unsigned int      vr397_u32;
unsigned int      vr399_u32;
unsigned int      vr401_u32;
unsigned int      vr403_u32;
unsigned int      vr405_u32;
unsigned int      vr407_u32;
unsigned int      vr409_u32;
unsigned int      vr411_u32;
unsigned int      vr413_u32;
unsigned int      vr416_u32;
unsigned int *    vr417_p32;
unsigned int *    vr420_p32;
unsigned int      vr425_u32;
unsigned int *    vr426_p32;
unsigned int *    vr429_p32;
unsigned int      vr434_u32;
unsigned int *    vr435_p32;
unsigned int *    vr438_p32;
unsigned int      vr443_u32;
unsigned int *    vr444_p32;
unsigned int *    vr447_p32;
unsigned int      vr452_u32;
unsigned int *    vr453_p32;
unsigned int *    vr456_p32;
unsigned int      vr461_u32;
unsigned int *    vr462_p32;
unsigned int *    vr465_p32;
unsigned int      vr470_u32;
unsigned int *    vr471_p32;
unsigned int *    vr474_p32;
unsigned int      vr479_u32;
unsigned int *    vr480_p32;
unsigned int *    vr483_p32;
vr480_p32 = (void *)&s4;
```



```
vr471_p32 = (void *)&s4;
vr462_p32 = (void *)&s4;
vr453_p32 = (void *)&s4;
vr444_p32 = (void *)&s4;
vr435_p32 = (void *)&s4;
vr426_p32 = (void *)&s4;
vr417_p32 = (void *)&s4;
vr416_u32 = s2<<3;
vr479_u32 = vr416_u32+7;
vr483_p32 = vr480_p32+vr479_u32;
vr470_u32 = vr416_u32+6;
vr474_p32 = vr471_p32+vr470_u32;
vr461_u32 = vr416_u32+5;
vr465_p32 = vr462_p32+vr461_u32;
vr452_u32 = vr416_u32+4;
vr456_p32 = vr453_p32+vr452_u32;
vr443_u32 = vr416_u32+3;
vr447_p32 = vr444_p32+vr443_u32;
vr434_u32 = vr416_u32+2;
vr438_p32 = vr435_p32+vr434_u32;
vr425_u32 = vr416_u32+1;
vr429_p32 = vr426_p32+vr425_u32;
vr420_p32 = vr417_p32+vr416_u32;
vr371_s32 = 7;
vr366_s32 = 6;
vr361_s32 = 5;
vr356_s32 = 4;
vr351_s32 = 3;
vr346_s32 = 2;
vr341_s32 = 1;
vr330_p32 = (void *)&s3;
vr333_p32 = vr330_p32+s2;
vr334_u8 = LODBU(vr333_p32);
vr370_u32 = vr334_u8&128;
vr373_u32 = ((unsigned long)vr370_u32) >>vr371_s32;
vr396_s32 = vr373_u32==1;
vr397_u32 = (int)vr396_s32;
if (vr397_u32) {
    vr413_u32 = s0;
} else {
    vr413_u32 = s1;
}
STR(vr483_p32, vr413_u32);
vr365_u32 = vr334_u8&64;
vr368_u32 = ((unsigned long)vr365_u32) >>vr366_s32;
vr393_s32 = vr368_u32==1;
vr394_u32 = (int)vr393_s32;
if (vr394_u32) {
    vr411_u32 = s0;
} else {
    vr411_u32 = s1;
}
STR(vr474_p32, vr411_u32);
vr360_u32 = vr334_u8&32;
vr363_u32 = ((unsigned long)vr360_u32) >>vr361_s32;
vr390_s32 = vr363_u32==1;
vr391_u32 = (int)vr390_s32;
if (vr391_u32) {
    vr409_u32 = s0;
} else {
    vr409_u32 = s1;
}
STR(vr465_p32, vr409_u32);
vr355_u32 = vr334_u8&16;
vr358_u32 = ((unsigned long)vr355_u32) >>vr356_s32;
vr387_s32 = vr358_u32==1;
vr388_u32 = (int)vr387_s32;
if (vr388_u32) {
    vr407_u32 = s0;
} else {
    vr407_u32 = s1;
}
STR(vr456_p32, vr407_u32);
vr350_u32 = vr334_u8&8;
vr353_u32 = ((unsigned long)vr350_u32) >>vr351_s32;
vr384_s32 = vr353_u32==1;
vr385_u32 = (int)vr384_s32;
```



```
if (vr385_u32) {
    vr405_u32 = s0;
} else {
    vr405_u32 = s1;
}
STR(vr447_p32,vr405_u32);
vr345_u32 = vr334_u8&4;
vr348_u32 = ((unsigned long)vr345_u32) >>vr346_s32;
vr381_s32 = vr348_u32==1;
vr382_u32 = (int)vr381_s32;
if (vr382_u32) {
    vr403_u32 = s0;
} else {
    vr403_u32 = s1;
}
STR(vr438_p32,vr403_u32);
vr340_u32 = vr334_u8&2;
vr343_u32 = ((unsigned long)vr340_u32) >>vr341_s32;
vr378_s32 = vr343_u32==1;
vr379_u32 = (int)vr378_s32;
if (vr379_u32) {
    vr401_u32 = s0;
} else {
    vr401_u32 = s1;
}
STR(vr429_p32,vr401_u32);
vr338_u32 = vr334_u8&1;
vr375_s32 = vr338_u32==1;
vr376_u32 = (int)vr375_s32;
if (vr376_u32) {
    vr399_u32 = s0;
} else {
    vr399_u32 = s1;
}
STR(vr420_p32,vr399_u32);
#pragma cycles_est_total = 9
}
```

(ε) htunpack_cfg0_bb2_ise1

```
void encode_xtea_2(
    unsigned int    *d0
    ,unsigned int    *d1
    ,unsigned int    *d2
    ,unsigned int    *d3
    ,unsigned int    *d4
    ,unsigned int    s0
    ,unsigned int    s1
    ,unsigned int    s2
    ,unsigned int *  s3
    ,unsigned int    s4
    ,unsigned int    s5
)
{
    int vr149_s32;
    unsigned int    vr151_u32;
    int vr152_s32;
    unsigned int    vr154_u32;
    unsigned int    vr155_u32;
    unsigned int    vr156_u32;
    unsigned int    vr158_u32;
    int vr159_s32;
    unsigned int *  vr160_p32;
    unsigned int *  vr163_p32;
    unsigned int    vr164_u32;
    unsigned int    vr166_u32;
    unsigned int    vr167_u32;
    unsigned int    vr168_u32;
    unsigned int    vr169_u32;
    int vr170_s32;
    unsigned int    vr172_u32;
    int vr173_s32;
    unsigned int    vr175_u32;
    unsigned int    vr176_u32;
}
```



```
unsigned int      vr177_u32;
int vr178_s32;
unsigned int      vr180_u32;
unsigned int      vr182_u32;
int vr183_s32;
unsigned int *    vr184_p32;
unsigned int *    vr187_p32;
unsigned int      vr188_u32;
unsigned int      vr190_u32;
unsigned int      vr191_u32;
unsigned int      vr192_u32;
int vr193_s32;
unsigned int      vr195_u32;
*d4 = 32;
vr193_s32 = 1;
vr195_u32 = s5+vr193_s32;
*d3 = vr195_u32;
vr184_p32 = (void *)&s3;
vr178_s32 = 11;
vr173_s32 = 5;
vr170_s32 = 4;
vr169_u32 = s2+s4;
*d2 = vr169_u32;
vr180_u32 = ((unsigned long)vr169_u32) >>vr178_s32;
vr182_u32 = vr180_u32&3;
vr183_s32 = vr182_u32<<2;
vr187_p32 = vr184_p32+vr183_s32;
vr188_u32 = LOD(vr187_p32);
vr190_u32 = vr169_u32+vr188_u32;
vr160_p32 = (void *)&s3;
vr158_u32 = s2&3;
vr159_s32 = vr158_u32<<2;
vr163_p32 = vr160_p32+vr159_s32;
vr164_u32 = LOD(vr163_p32);
vr166_u32 = s2+vr164_u32;
vr152_s32 = 5;
vr154_u32 = ((unsigned long)s0) >>vr152_s32;
vr149_s32 = 4;
vr151_u32 = s0<<vr149_s32;
vr155_u32 = vr151_u32^vr154_u32;
vr156_u32 = vr155_u32+s0;
vr167_u32 = vr156_u32^vr166_u32;
vr168_u32 = s1+vr167_u32;
*d1 = vr168_u32;
vr175_u32 = ((unsigned long)vr168_u32) >>vr173_s32;
vr172_u32 = vr168_u32<<vr170_s32;
vr176_u32 = vr172_u32^vr175_u32;
vr177_u32 = vr176_u32+vr168_u32;
vr191_u32 = vr177_u32^vr190_u32;
vr192_u32 = s0+vr191_u32;
*d0 = vr192_u32;
#pragma cycles_est_total = 5
}
```

(στ) encode_xtea_cfg0_bb2_ise1

```
void decode_xtea_2(
    unsigned int *d0
    ,unsigned int *d1
    ,unsigned int *d2
    ,unsigned int *d3
    ,unsigned int *d4
    ,unsigned int s0
    ,unsigned int s1
    ,unsigned int s2
    ,unsigned int *s3
    ,unsigned int s4
    ,unsigned int s5
)
{
    int vr163_s32;
    unsigned int vr165_u32;
    int vr166_s32;
    unsigned int vr168_u32;
```

```
unsigned int      vr169_u32;
unsigned int      vr170_u32;
int vr172_s32;
unsigned int      vr174_u32;
unsigned int      vr176_u32;
int vr177_s32;
unsigned int *    vr178_p32;
unsigned int *    vr181_p32;
unsigned int      vr182_u32;
unsigned int      vr185_u32;
unsigned int      vr186_u32;
unsigned int      vr187_u32;
unsigned int      vr189_u32;
int vr190_s32;
unsigned int      vr192_u32;
int vr193_s32;
unsigned int      vr195_u32;
unsigned int      vr196_u32;
unsigned int      vr197_u32;
unsigned int      vr200_u32;
int vr201_s32;
unsigned int *    vr202_p32;
unsigned int *    vr205_p32;
unsigned int      vr206_u32;
unsigned int      vr209_u32;
unsigned int      vr210_u32;
unsigned int      vr211_u32;
int vr212_s32;
unsigned int      vr214_u32;
*d4 = 32;
vr212_s32 = 1;
vr214_u32 = s5+vr212_s32;
*d3 = vr214_u32;
vr202_p32 = (void *)&s3;
vr193_s32 = 5;
vr190_s32 = 4;
vr189_u32 = s2-s4;
*d2 = vr189_u32;
vr200_u32 = vr189_u32&3;
vr201_s32 = vr200_u32<<2;
vr205_p32 = vr202_p32+vr201_s32;
vr206_u32 = LOD(vr205_p32);
vr209_u32 = vr189_u32+vr206_u32;
vr178_p32 = (void *)&s3;
vr172_s32 = 11;
vr174_u32 = ((unsigned long)s2) >>vr172_s32;
vr176_u32 = vr174_u32&3;
vr177_s32 = vr176_u32<<2;
vr181_p32 = vr178_p32+vr177_s32;
vr182_u32 = LOD(vr181_p32);
vr185_u32 = s2+vr182_u32;
vr166_s32 = 5;
vr168_u32 = ((unsigned long)s0) >>vr166_s32;
vr163_s32 = 4;
vr165_u32 = s0<<vr163_s32;
vr169_u32 = vr165_u32^vr168_u32;
vr170_u32 = vr169_u32+s0;
vr186_u32 = vr170_u32^vr185_u32;
vr187_u32 = s1-vr186_u32;
*d1 = vr187_u32;
vr195_u32 = ((unsigned long)vr187_u32) >>vr193_s32;
vr192_u32 = vr187_u32<<vr190_s32;
vr196_u32 = vr192_u32^vr195_u32;
vr197_u32 = vr196_u32+vr187_u32;
vr210_u32 = vr197_u32^vr209_u32;
vr211_u32 = s0-vr210_u32;
*d0 = vr211_u32;
#pragma cycles_est_total = 5
}
```

(ζ) decode_xtea_cfg0_bb2_ise1

Κώδικας 2.3: Κώδικες προσομοίωσης σε ANSI C για τις επιλεγμένες ειδικές εντολές του λογισμικού εφαρμογών του IPCN.

Προκειμένου τη σύνθεση των ΕΛΜ σε VHDL, το YARDstick προσφέρει τη δυνατότητα μετατροπής τους από τη διαμόρφωση ISeq στην cdfg του εργαλείου CDFGtool [CDFGtool] ώστε έπειτα να είναι εφικτή η μετατροπή τους σε VHDL με το εργαλείο cdfg2vhdl. Το εργαλείο αυτό υποστηρίζει τις ειδικές εντολές που εξάγονται από το YARDstick με κάποιους περιορισμούς (εντολές select της IR, πόροι αποθήκευσης, εκτέλεση σε πολλαπλούς κύκλους). Επίσης υπάρχει η δυνατότητα σύνθεσης του κώδικα VHDL των ΕΛΜ απευθείας από τον κώδικα προσομοίωσης σε C, χρησιμοποιώντας τα δωρεάν εργαλεία σύνθεσης υψηλού επιπέδου SPARK [SPARK] και GAUT [GAUT], εντός των περιορισμών που αυτά επιβάλλουν.

Στον Κώδικα 2.4 δίνονται δύο ενδεικτικές περιπτώσεις κώδικα VHDL για δύο αρκετά σύνθετες ΕΛΜ που αντιστοιχούν στις (β) και (γ) στα Σχήματα 2.8 και Κώδικα 2.3. Πρόκειται για την 2η ειδική εντολή της εφαρμογής fsdither και την ειδική εντολή που εξάγεται για τη γεννήτρια καμπύλων Hilbert (hilcurv). Οι υλοποιήσεις του Κώδικα 2.4 είναι τύπου FSM (Finite-State Machine with Datapath) [Bai01]. Στην περίπτωση της ειδικής εντολής fsdither_cfg0_bb2_ise2, δίνεται και ο κώδικας της διεπαφής της ειδικής εντολής με το υποσύστημα φόρτωσης/αποθήκευσης μνήμης δεδομένων (το οποίο βρίσκεται στο στάδιο διοχέτευσης MEM του επεξεργαστή). Στη συγκεκριμένη περίπτωση, η εντολή fsdither_cfg0_bb2_ise2 έχει υλοποιηθεί με 3 καταστάσεις καθώς οι λειτουργίες φόρτωσης (LOAD_0) και αποθήκευσης (STORE_0) που απαιτούνται έχουν μετατεθεί σε ξεχωριστές καταστάσεις διάρκειας ενός κύκλου μηχανής.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity ci_fsdither0 is
  generic (
    OW      : integer := 8;
    WIDTH   : integer := 32
  );
  port (
    clk      : in  std_logic;
    reset    : in  std_logic;
    ci_en    : in  std_logic;
    opcode   : in  std_logic_vector(OW-1 downto 0);
    opd_src0 : in  std_logic_vector(WIDTH-1 downto 0);
    opd_src1 : in  std_logic_vector(WIDTH-1 downto 0);
    opd_src2 : in  std_logic_vector(WIDTH-1 downto 0);
    opd_dst0 : out std_logic_vector(WIDTH-1 downto 0);
    opd_dst1 : out std_logic_vector(WIDTH-1 downto 0);
    opd_lod  : in  std_logic_vector(WIDTH-1 downto 0);
    opd_str  : out std_logic_vector(WIDTH-1 downto 0);
    addr_ls  : out std_logic_vector(WIDTH-1 downto 0);
    mode_ls  : out std_logic_vector(3 downto 0);
    done     : out std_logic
  );
end ci_fsdither0;

architecture synth_mcycle of ci_fsdither0 is
  --
  -- Type declarations
  type STATE_TYPE is (COMPUTATION_0, LOAD_0, STORE_0);
  --
  -- Signal declarations
  signal current, following : STATE_TYPE;
  signal start              : std_logic;
  signal ci_cycle           : std_logic_vector(7 downto 0);
  signal vr142_p32         : std_logic_vector(31 downto 0);
  signal vr145_p32         : std_logic_vector(31 downto 0);
  signal vr146_u8          : std_logic_vector(7 downto 0);
  signal vr148_p32         : std_logic_vector(31 downto 0);
  signal vr151_p32         : std_logic_vector(31 downto 0);
end synth_mcycle;
```



```
signal vr153_s32      : std_logic_vector(31 downto 0);
--
begin

-- Formation of multi-cycle controller (FSM) start signal
start <= '1' when (opcode = X"81" and ci_en = '1' and ci_cycle =
    conv_std_logic_vector(0,8)) else '0';

-- Next state logic
process (current, start)
begin
    case current is
        when COMPUTATION_0 =>
            if (start = '1') then
                following <= LOAD_0;
            else
                following <= COMPUTATION_0;
            end if;
        when LOAD_0 =>
            following <= STORE_0;
        when STORE_0 =>
            following <= COMPUTATION_0;
    end case;
end process;

-- Current state logic
process (clk, reset)
begin
    if (reset = '1') then
        current <= COMPUTATION_0;
    elsif (clk'event and clk='1') then
        current <= following;
    end if;
end process;

-- Miscellaneous synchronous logic
process (clk, reset)
begin
    if (reset = '1') then
        ci_cycle <= (others => '0');
    elsif (clk'event and clk='1') then
        ci_cycle <= ci_cycle + 1;
    end if;
end process;

-- Output logic
process (current)
begin
    case current is
        when COMPUTATION_0 =>
            --
            opd_dst1 <= conv_std_logic_vector(4096,WIDTH);
            vr153_s32 <= opd_src0 + 1;
            opd_dst0 <= vr153_s32;
            vr148_p32 <= opd_src2;
            vr151_p32 <= vr148_p32 + opd_src0;
            vr142_p32 <= opd_src1;
            vr145_p32 <= vr142_p32 + opd_src0;
            ci_cycle <= ci_cycle + 1;
            done <= '0';
        when LOAD_0 =>
            --
            -- vr146_u8 = LOD(vr145_p32);
            addr_ls <= vr145_p32;
            vr146_u8 <= opd_lod(7 downto 0);
            mode_ls <= "1000";
            ci_cycle <= ci_cycle + 1;
            done <= '0';
        when STORE_0 =>
            --
            -- DM.write_byte(vr151.p32,vr146.u8);
            opd_str <= (WIDTH-1 downto 8 => '0') & vr146_u8;
            addr_ls <= vr151_p32;
            mode_ls <= "1000";
            ci_cycle <= (others => '0');
            done <= '1';
    end case;
end process;
```



```
end process;  
  
end synth_mcycle;
```

(α) fsdither_cfg0_bb2_ise2

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.std_logic_unsigned.all;  
use IEEE.numeric_std.all;  
use WORK.useful_functions_pkg.all;  
  
entity ci_hilcurv0 is  
  generic (  
    OW      : integer := 8;  
    WIDTH   : integer := 32  
  );  
  port (  
    clk      : in  std_logic;  
    reset    : in  std_logic;  
    ci_en    : in  std_logic;  
    opcode   : in  std_logic_vector(OW-1 downto 0);  
    opd_src0 : in  std_logic_vector(WIDTH-1 downto 0);  
    opd_src1 : in  std_logic_vector(WIDTH-1 downto 0);  
    opd_src2 : in  std_logic_vector(WIDTH-1 downto 0);  
    opd_src3 : in  std_logic_vector(WIDTH-1 downto 0);  
    opd_src4 : in  std_logic_vector(WIDTH-1 downto 0);  
    opd_dst0 : out std_logic_vector(WIDTH-1 downto 0);  
    opd_dst1 : out std_logic_vector(WIDTH-1 downto 0);  
    opd_dst2 : out std_logic_vector(WIDTH-1 downto 0);  
    opd_dst3 : out std_logic_vector(WIDTH-1 downto 0);  
    opd_dst4 : out std_logic_vector(WIDTH-1 downto 0);  
    done     : out std_logic  
  );  
end ci_hilcurv0;  
  
architecture synth_mcycle of ci_hilcurv0 is  
  --  
  -- Type declarations  
  type STATE_TYPE is (COMPUTATION_0, COMPUTATION_1);  
  --  
  -- Signal declarations  
  signal current, following : STATE_TYPE;  
  signal start              : std_logic;  
  signal ci_cycle           : std_logic_vector(7 downto 0);  
  signal vr170_s32         : std_logic_vector(31 downto 0);  
  signal vr172_u32         : std_logic_vector(31 downto 0);  
  signal vr174_u32         : std_logic_vector(31 downto 0);  
  signal vr176_u32         : std_logic_vector(31 downto 0);  
  signal vr177_u32         : std_logic_vector(31 downto 0);  
  signal vr178_s32         : std_logic_vector(31 downto 0);  
  signal vr180_u32         : std_logic_vector(31 downto 0);  
  signal vr181_s32         : std_logic_vector(31 downto 0);  
  signal vr183_s32         : std_logic_vector(31 downto 0);  
  signal vr185_s32         : std_logic_vector(31 downto 0);  
  signal vr187_u32         : std_logic_vector(31 downto 0);  
  signal vr188_s32         : std_logic_vector(31 downto 0);  
  signal vr190_u32         : std_logic_vector(31 downto 0);  
  signal vr191_s32         : std_logic_vector(31 downto 0);  
  signal vr193_s32         : std_logic_vector(31 downto 0);  
  signal vr195_s32         : std_logic_vector(31 downto 0);  
  signal vr197_u32         : std_logic_vector(31 downto 0);  
  signal vr198_s32         : std_logic_vector(31 downto 0);  
  signal vr199_s32         : std_logic_vector(31 downto 0);  
  signal vr201_u32         : std_logic_vector(31 downto 0);  
  signal vr203_s32         : std_logic_vector(31 downto 0);  
  signal vr205_s32         : std_logic_vector(31 downto 0);  
  signal vr208_s32         : std_logic_vector(31 downto 0);  
  --  
  signal vr177_u32_r       : std_logic_vector(31 downto 0);  
  signal vr178_s32_r       : std_logic_vector(31 downto 0);  
  signal vr183_s32_r       : std_logic_vector(31 downto 0);
```



```
signal vr188_s32_r      : std_logic_vector(31 downto 0);
signal vr193_s32_r      : std_logic_vector(31 downto 0);
signal vr198_s32_r      : std_logic_vector(31 downto 0);
signal vr199_s32_r      : std_logic_vector(31 downto 0);
signal opd_src3_r       : std_logic_vector(31 downto 0);
signal opd_src4_r       : std_logic_vector(31 downto 0);
--
begin
-- Formation of multi-cycle controller (FSM) start signal
start <= '1' when (opcode = X"82" and ci_en = '1' and ci_cycle = X"00") else '0';

-- Current state logic
process (clk, reset)
begin
  if (reset = '1') then
    current <= COMPUTATION_0;
  elsif (clk'event and clk='1') then
    current <= following;
  end if;
end process;

-- Miscellaneous synchronous logic
process (clk, reset)
begin
  if (reset = '1') then
    ci_cycle <= (others => '0');
    vr177_u32_r <= (others => '0');
    vr178_s32_r <= (others => '0');
    vr183_s32_r <= (others => '0');
    vr188_s32_r <= (others => '0');
    vr193_s32_r <= (others => '0');
    vr198_s32_r <= (others => '0');
    vr199_s32_r <= (others => '0');
    opd_src3_r <= (others => '0');
    opd_src4_r <= (others => '0');
  elsif (clk'event and clk='1') then
    ci_cycle <= ci_cycle + 1;
    vr177_u32_r <= vr177_u32;
    vr178_s32_r <= vr178_s32;
    vr183_s32_r <= vr183_s32;
    vr188_s32_r <= vr188_s32;
    vr193_s32_r <= vr193_s32;
    vr198_s32_r <= vr198_s32;
    vr199_s32_r <= vr199_s32;
    opd_src3_r <= opd_src3;
    opd_src4_r <= opd_src4;
  end if;
end process;

-- Next state logic
process (current, start)
begin
  case current is
    when COMPUTATION_0 =>
      if (start = '1') then
        following <= COMPUTATION_1;
      else
        following <= COMPUTATION_0;
      end if;
    when COMPUTATION_1 =>
      following <= COMPUTATION_0;
  end case;
end process;

-- Output logic
process (current)
-- Variable declarations
variable shamt1, shamt2 : integer range 0 to WIDTH-1;
variable shamt3, shamt4 : integer range 0 to WIDTH-1;
variable shamt5, shamt6 : integer range 0 to WIDTH-1;
variable shamt7, shamt8 : integer range 0 to WIDTH-1;
--
begin
  case current is
    when COMPUTATION_0 =>
      --

```

```
    opd_dst4 <= X"00000000";
    vr208_s32 <= opd_src2 - X"00000010";
    opd_dst3 <= vr208_s32;
    vr199_s32 <= X"00000001";
    vr198_s32 <= X"3E6B94C1";
    vr191_s32 <= X"000039C6";
    vr188_s32 <= X"00000001";
    vr181_s32 <= X"000093C6";
    vr178_s32 <= X"00000001";
--    vr174_u32 <= ((unsigned long)s1) >>s2;
    shamt1 := to_integer(unsigned(opd_src2(log2c(WIDTH-1) downto 0)));
    vr174_u32 <= std_logic_vector(shift_right(unsigned(opd_src1), shamt1));
    vr176_u32 <= vr174_u32 and X"00000011";
    vr170_s32 <= X"00000010";
--    vr172_u32 = s0<<vr170_s32;
    shamt2 := to_integer(unsigned(vr170_s32(log2c(WIDTH-1) downto 0)));
    vr172_u32 <= std_logic_vector(shift_left(unsigned(opd_src0), shamt2));
    vr177_u32 <= vr172_u32 or vr176_u32;
--    vr193_s32 <= vr191_s32>>vr177_u32;
    shamt3 := to_integer(unsigned(vr177_u32(log2c(WIDTH-1) downto 0)));
    if (vr191_s32(WIDTH-1) = '1') then
        vr193_s32 <= not(std_logic_vector(shift_right(unsigned(not vr191_s32), shamt3)));
    else
        vr193_s32 <= std_logic_vector(shift_right(unsigned(vr191_s32), shamt3));
    end if;
--    vr183_s32 <= vr181_s32>>vr177_u32;
    shamt4 := to_integer(unsigned(vr177_u32(log2c(WIDTH-1) downto 0)));
    if (vr181_s32(WIDTH-1) = '1') then
        vr183_s32 <= not(std_logic_vector(shift_right(unsigned(not vr181_s32), shamt4)));
    else
        vr183_s32 <= std_logic_vector(shift_right(unsigned(vr181_s32), shamt4));
    end if;
--
--
done    <= '0';
when COMPUTATION_1 =>
--
--    vr190_u32 <= s4<<vr188_s32;
    shamt5 := to_integer(unsigned(vr188_s32_r(log2c(WIDTH-1) downto 0)));
    vr190_u32 <= std_logic_vector(shift_left(unsigned(opd_src4_r), shamt5));
--    vr180_u32 <= s3<<vr178_s32;
    shamt6 := to_integer(unsigned(vr178_s32_r(log2c(WIDTH-1) downto 0)));
    vr180_u32 <= std_logic_vector(shift_left(unsigned(opd_src3_r), shamt6));
--    vr201_u32 = vr177_u32<<vr199_s32;
    shamt7 := to_integer(unsigned(vr199_s32_r(log2c(WIDTH-1) downto 0)));
    vr201_u32 <= std_logic_vector(shift_left(unsigned(vr177_u32), shamt7));
--    vr203_s32 = vr198_s32>>vr201_u32;
    shamt8 := to_integer(unsigned(vr201_u32(log2c(WIDTH-1) downto 0)));
    if (vr198_s32_r(WIDTH-1) = '1') then
        vr203_s32 <= not(std_logic_vector(shift_right(unsigned(not vr198_s32_r), shamt8)));
    else
        vr203_s32 <= std_logic_vector(shift_right(unsigned(vr198_s32_r), shamt8));
    end if;
    vr205_s32 <= vr203_s32 and X"00000011";
    opd_dst2 <= vr205_s32;
    vr195_s32 <= vr193_s32 and X"00000001";
    vr197_u32 <= vr190_u32 or vr195_s32;
    opd_dst1 <= vr197_u32;
    vr185_s32 <= vr183_s32_r and X"00000001";
    vr187_u32 <= vr180_u32 or vr185_s32;
    opd_dst0 <= vr187_u32;
--
done    <= '1';
end case;
end process;

end synth_mcycle;
```

(β) hilcurv_cfg0_bb3_ise2

Κώδικας 2.4: Κώδικες VHDL επιπέδου RTL για επιλεγμένες ειδικές εντολές που προέκυψαν από εφαρμογή της μεθοδολογίας στο λογισμικό εφαρμογών του IPCN.

3. ΕΚΤΕΛΕΣΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ ΕΦΑΡΜΟΓΩΝ ΣΕ ΠΡΟΣΟΜΙΩΤΗ ΕΚΤΙΜΗΣΗΣ ΚΥΚΛΟΥ

Το λογισμικό εφαρμογών του IPCN δοκιμάστηκε στον προσομοιωτή εκτίμησης κύκλου για τον επεξεργαστή ByoRISC. Το μοντέλο προσομοίωσης του ByoRISC διαθέτει αρκετή μνήμη (4MB) ώστε η επεξεργασία εικόνων των μεγαλύτερων διαστάσεων για τις δοκιμές (512×512) να μην αποτελεί πρόβλημα. Για κάθε ειδική εντολή, το μοντέλο προσομοίωσης χρειάζεται να ενημερωθεί με την προσθήκη της καταχώρησης ονόματος εντολής, αποκωδικοποίησης, διαμόρφωσης και συμπεριφοράς για την αντίστοιχη ΕΛΜ. Στον Κώδικα 3.1 δίνονται τα αντίστοιχα τμήματα κώδικα, τα οποία εισήχθησαν στην ArchC περιγραφή του προσομοιωτή, για τη γνωστοποίηση σε αυτόν, της ειδικής εντολής "hilcurv0" (πλήρες όνομα για το YARDstick: hilcurv_cfg0_bb3_ise2).

```
ac_instr<Type_B> hilcurv0;
```

(α) Καταχώρηση ονόματος εντολής

```
hilcurv0.set_decoder(op=0x52);
```

(β) Καταχώρηση αποκωδικοποίησης εντολής

```
hilcurv0.set_asm("hilcurv0 %exp", occ);
```

(γ) Καταχώρηση διαμόρφωσης εντολής

```
//!Instruction hilcurv0 behavior method.
void ac_behavior( hilcurv0 )
{
    my_ac_cycle += 2;
    dbg_printf("hilcurv0 %d\n", occ);

    // Read operands from SID storage
    decode_sid_src(ci_src, ci_src_addr, occ);

    dbg_printf("Source operand addresses = %#x, %#x, %#x, %#x, %#x\n",
        ci_src_addr[0], ci_src_addr[1], ci_src_addr[2], ci_src_addr[3], ci_src_addr[4]);

    dbg_printf("Source operands = %#x, %#x, %#x, %#x, %#x\n",
        ci_src[0], ci_src[1], ci_src[2], ci_src[3], ci_src[4]);

    //
    // CI: hilcurv0
    //
    // row = (state << 2) | (s >> i) & 3;
    // x = (x << 1) | (0x936C >> row) & 1;
    // y = (y << 1) | (0x39C6 >> row) & 1;
    // state = (0x3E6B94C1 >> (row << 1)) & 3;
    //
    // v8 = row,
    // v6 = x, v7 = y, v5 = state, v4 = i, v3 = s
    //
    // READ : v6, v7, v5, v3, v4
    // WRITE: v6, v7, v5
    // hilcurv0 $6, $7, $5, $6, $7, $5, $3, $4
    //
    int x, y, row, state, s, i;

    x      = ci_src[0];
    y      = ci_src[1];
    state  = ci_src[2];
    s      = ci_src[3];
    i      = ci_src[4];
}
```

```
row = (state << 2) | (s >> i) & 3;
x = (x << 1) | (0x936C >> row) & 1;
y = (y << 1) | (0x39C6 >> row) & 1;
state = (0x3E6B94C1 >> (row << 1)) & 3;

ci_dst[0] = x;
ci_dst[1] = y;
ci_dst[2] = state;

decode_sid_dst(ci_dst, ci_dst_addr, occ);

dbg_printf("Dest operand addresses = %#x, %#x, %#x\n",
ci_dst_addr[0], ci_dst_addr[1], ci_dst_addr[2]);

dbg_printf("Result = %#x, %#x, %#x\n", RB[ci_dst_addr[0]], RB[ci_dst_addr[1]],
RB[ci_dst_addr[2]]);
};
```

(δ) Καταχώρηση συμπεριφοράς εντολής

Κώδικας 3.1: Ενημέρωση της περιγραφής του επεξεργαστή σε ArchC με την περιγραφή της ειδικής εντολής "hilcurv0".

Με το πέρας της προσομοίωσης, αλλά και σε οποιοδήποτε άλλο σημείο αυτό ζητηθεί, μπορούμε να οπτικοποιήσουμε τα περιεχόμενα μέρους (από δοσμένη αρχική μέχρι τελική διεύθυνση) της μνήμης δεδομένων. Στο Σχήμα 3.1 δίνονται οι αρχικές εικόνες δοκιμής, οι οποίες είναι greyscale 256 επιπέδων: fruits (διαστάσεων 512×480), lenna (512×512), και monarch (256×256). Για τις εφαρμογές htpack, htunpack και xtheadec, χρησιμοποιούνται ως εικόνες εισόδου, οι προκύπτουσες εικόνες από την εφαρμογή των fsdither, htpack και xteaenc, αντίστοιχα. Οι εικόνες αυτές διακρίνονται από τις καταλήξεις fsd, htp και cbc, αντίστοιχα. Οι εφαρμογές htunpack και xtheadec είναι συμπληρωματικές των htpack και xteaenc, αντίστοιχα, με αποτέλεσμα οι προκύπτουσες εικόνες να είναι ακριβώς ίδιες με αυτές που χρησιμοποιήθηκαν ως είσοδοι στις δεύτερες. Στα Σχήματα 3.2-3.5 δίνονται απεικονίσεις των αποτελεσμάτων από την επεξεργασία που επιτελείται από κάθε εφαρμογή στις διαφορετικές εικόνες δοκιμής. Σε κάθε περίπτωση, η ορθότητα των περιεχομένων της μνήμης δεδομένων επιβεβαιώθηκε με δύο διαφορετικές συγκρίσεις: α) ως προς τα αποτελέσματα της εκτέλεσης του αντικείμενου κώδικα που παράγεται από τον πηγαίο κώδικα αναφοράς της εφαρμογής, και β) ως προς τα αποτελέσματα προσομοίωσης του VHDL κώδικα του επεξεργαστή στα περιβάλλοντα Modelsim [Modelsim] και GHDL-0.26 [GHDL]. Όλες οι μετρήσεις ελήφθησαν σε περιβάλλον Windows-XP/x86.

Να σημειωθεί ότι στις εφαρμογές εκτός της hilcurv, το άνω μέρος (άνω σελίδα μνήμης) της εικόνας απεικονίζει τα αρχικά περιεχόμενα της μνήμης δεδομένων και το κάτω μέρος (κάτω σελίδα μνήμης) τα αποτελέσματα λόγω της εφαρμογής του αλγορίθμου από την επεξεργασία της αρχικής εικόνας. Η εφαρμογή hilcurv αποτελεί εξαίρεση καθώς δεν επεξεργάζεται την αρχική εικόνα. Στην περίπτωση αυτή, τα αποτελέσματα της hilcurv, γράφονται στην άνω σελίδα μνήμης. Για την hilcurv λαμβάνεται στιγμιότυπο της έπειτα από 985 επαναλήψεις του εξώτερου βρόχου της, ώστε να είναι ευδιάκριτη η λειτουργία της ως καμπύλη πλήρωσης χώρου.



(α) fruits.



(β) lenna.



(γ) monarch.

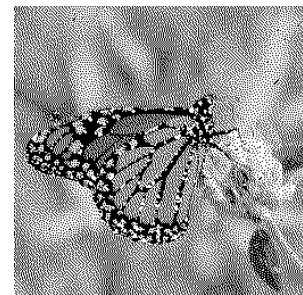
Σχήμα 3.1: Αρχικές εικόνες για τις εφαρμογές fsdither και xteaenc. (α) fruits (512×480), (β) lenna (512×512) και (γ) monarch (256×256).



(α) fruits_fsd.



(β) lenna_fsd.

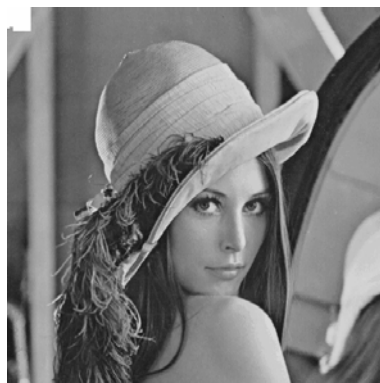


(γ) monarch_fsd.

Σχήμα 3.2: Αποτελέσματα της fsdither για τις εικόνες: (α) fruits, (β) lenna και (γ) monarch.



(α) fruits_hil.



(β) lenna_hil.

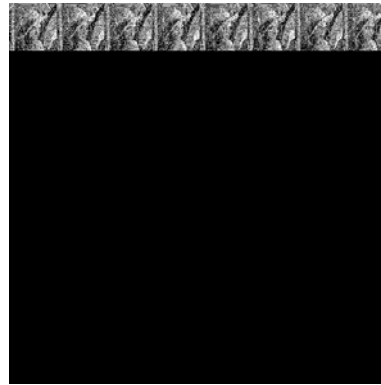


(γ) monarch_hil.

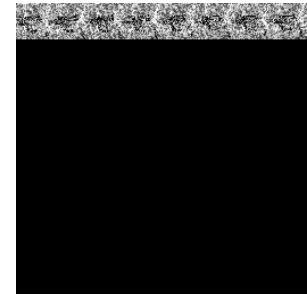
Σχήμα 3.3: Στιγμιότυπο της εκτέλεσης της εφαρμογής hilcurv για τις εικόνες: (α) fruits, (β) lenna και (γ) monarch.



(α) fruits_htp.

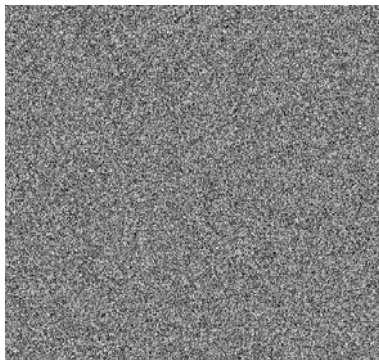


(β) lenna_htp.

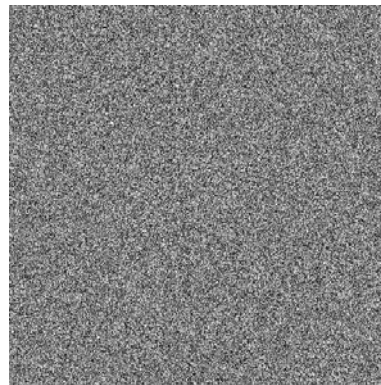


(γ) monarch_htp.

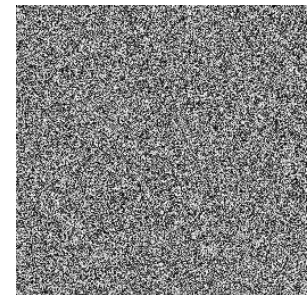
Σχήμα 3.4: Αποτελέσματα της htpack για τις εικόνες: (α) fruits, (β) lenna και (γ) monarch, δεχόμενη ως εισόδους τις αντίστοιχες εικόνες του Σχήματος 3.2.



(α) fruits_cbc.



(β) lenna_cbc.



(γ) monarch_cbc.

Σχήμα 3.5: Αποτελέσματα της xteaenc για τις εικόνες: (α) fruits, (β) lenna και (γ) monarch.

Τα αποτελέσματα αυτά επιβεβαιώνονται και από την προσομοίωση του VHDL μοντέλου του επεξεργαστή.

4. Η ΑΝΑΠΤΥΞΙΑΚΗ ΠΛΑΚΕΤΑ FPGA ΓΙΑ ΤΙΣ ΔΟΚΙΜΕΣ ΤΟΥ ΕΝΣΩΜΑΤΩΜΕΝΟΥ ΣΥΣΤΗΜΑΤΟΣ

Στο Σχήμα 4.1 φαίνεται η αναπτυξιακή πλακέτα Spartan-3 Starter Kit στο εργαστηριακό περιβάλλον χρήσης της. Η διασύνδεση VGA υποστηρίζει 8 μόνο χρώματα (1-bit χρωματικής πληροφορίας ανά R, G, B) αλλά η χρήση του κυκλώματος σκάλας το οποίο φαίνεται υλοποιημένο στο ράστερ στο αριστερά μέρος της εικόνας επιτρέπει την απεικόνιση greyscale εικόνων με 64 επίπεδα (καλή ποιότητα) ή 256/512 επίπεδα του γκρι (μέτρια ποιότητα απεικόνισης).



Σχήμα 4.1: Εργαστηριακή διάταξη για την χρήση του επεξεργαστή ByoRISC προκειμένου την επεξεργασία εικόνας.

Με χρήση της εν λόγω πειραματική διάταξης πιστοποιήθηκε η σωστή λειτουργία του επεξεργαστή, καθώς τα λαμβανόμενα αποτελέσματα (περιεχόμενα της μνήμης δεδομένων για εικόνες διαστάσεων 64x64 εικονοστοιχείων) είναι σε απόλυτη συμφωνία με τις προσομοιώσεις σε επίπεδο ArchC και VHDL.

5. ΑΝΑΦΟΡΕΣ

[Bai01] B. Bailey and D. Gajski, "RTL Semantics and Methodology," in Proceedings of the 2001 International Symposium on System Synthesis, pp. 69-74, Montreal, Quebec, Canada, Oct. 1-3, 2001.

[CDFGtool] CDFG toolset. <http://poppy.snu.ac.kr/CDFG/cdfg.html>

[EEMBC] EEMBC – The Embedded Microprocessor Benchmark Consortium. <http://www.eembc.com>

[GAUT] GAUT High-Level Synthesis tool. <http://lester.univ-ubs.fr:8080/tools/gaut/tool.htm>

[GHDL] GHDL. <http://ghdl.free.fr>

[Kav07] N. Kavvadias and S. Nikolaidis, "YARDstick: Automation for custom processor development," presented at the University Booth of the Design, Automation and Test in Europe Conference (DATE'07), Nice, France, April 16-20, 2007.

[Modelsim] Mentor Modelsim. <http://www.mentor.com>

[Nee97] Roger M. Needham and David J. Wheeler, "Tea extensions," Technical report, Computer Laboratory, University of Cambridge, October 1997.

[Pot07] N. Pothineni, A. Kumar, and K. Paul, "Application specific datapath extension with distributed I/O functional units," in Proceedings of the 20th International Conference on VLSI Design (VLSI Design 2007), 6th International Conference on Embedded Systems (ICES 2007), pp. 551-558, Bangalore, India, Jan. 2007.

[Poz06] L. Pozzi, K. Atasu, and P. lenne, "Exact and Approximate Algorithms for the Extension of Embedded Processor Instruction Sets," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 25, No. 7, pp. 1209-1229, July 2006.

[War02] H.S. Warren, Hacker's Delight, Addison-Wesley Publishing, July 2002.