



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΨΗΦΙΑΚΑ VLSI – 8^ο ΕΞΑΜΗΝΟ

Τζομάκα Αφροδίτη – ΑΜ: 03117107

Bonus Θέμα στο High Level Synthesis

Ζητούμενο 1

1. Προκειμένου το HLS FIR φίλτρο να έχει τον ίδιο αριθμό taps και τα ίδια coefficient με το φίλτρο του ζητούμενου 1 της εργαστηριακής άσκησης 3 θα:

- αλλάξουμε τη συνάρτηση fir(), βγάζοντας την παράμετρο coef_t c[] και βάζοντας πλέον τον πίνακα των coefficient μέσα στην συνάρτηση ως σταθερά, N από 11 σε 8 στο αρχείο fir.h
- φτιάχνουμε στο αρχείο fir_test.c έναν πίνακα signal όπου θα αποθηκεύονται τα (δοθέντα) inputs αντί της μεταβλητής που καθόριζε την τιμή της η ramp_up
- αλλάξουμε το N από 11 σε 8 στο αρχείο fir.h
- αλλάξουμε το out.gold.dat με τα δοθέντα αναμενόμενα outputs.

Οι κώδικες με τις παραπάνω αλλαγές, καθώς και ένα screenshot από την εκτέλεση τους δίνονται παρακάτω:

fir_test.c

```
#include <stdio.h>
#include <math.h>
#include "fir.h"

int main () {
    const int    SAMPLES=17;
    FILE         *fp;
    data_t output;
    data_t signal[SAMPLES] = {40, 248, 245, 124, 204, 36, 107, 234, 202, 245, 0, 0, 0, 0, 0, 0, 0};
    int i, ramp_up;
    ramp_up = 1;

    fp=fopen("out.dat","w");
    for (i=0;i<SAMPLES;i++) {
        // Execute the function with latest input
        fir(&output,signal[i]);
        // Save the results.
        fprintf(fp,"%i %d %d\n",i,signal[i],output);
    }
    fclose(fp);
    printf ("Comparing against output data \n");
    if (system("diff -w out.dat out.gold.dat")) {

        fprintf(stdout, "*****\n");
        fprintf(stdout, "FAIL: Output DOES NOT match the golden output!\n");
        fprintf(stdout, "*****\n");
        return 1;
    } else {
        fprintf(stdout, "*****\n");
        fprintf(stdout, "PASS: The output matches the golden output!\n");
        fprintf(stdout, "*****\n");
        return 0;
    }
}
```

fir.c

```
#include "fir.h"

void fir (data_t *y, data_t x)
{
    coef_t c[N] = {1,2,3,4,5,6,7,8};
    static data_t shift_reg[N];
    acc_t acc;
    data_t data;
    int i;

    acc=0;
    Shift_Accum_Loop: for (i=N-1;i>=0;i--) {
        if (i==0) {
            shift_reg[0]=x;
            data = x;
        } else {
            shift_reg[i]=shift_reg[i-1];
            data = shift_reg[i];
        }
        acc+=data*c[i];
    }
    *y=acc;
}
```

fir.h

```
#ifndef FIR_H_
#define FIR_H_
#define N 8

typedef int coef_t;
typedef int data_t;
typedef int acc_t;

void fir (data_t *y, data_t x);

#endif
```

out.gold.dat

```
0 40 40
1 248 328
2 245 861
3 124 1518
4 204 2379
5 36 3276
6 107 4280
7 234 5518
8 202 6598
9 245 6011
10 0 5203
11 0 5239
12 0 4431
13 0 4931
14 0 4756
15 0 3331
16 0 1960
```

```
Starting C simulation ...
C:/Xilinx/Vivado/2018.2/bin/vivado_hls.bat C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir/solution1/csim.tcl
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2018.2/bin/unwrapped/win64.o/vivado_hls.exe'
INFO: [HLS 200-10] For user 'Afroditi' on host 'desktop-ii5sp0k' (Windows NT_amd64 version 6.2) on Mon Jun 14 11:48:23 +0300 2021
INFO: [HLS 200-10] In directory 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir'
INFO: [HLS 200-10] Opening project 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir'.
INFO: [HLS 200-10] Opening solution 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir/solution1'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Setting target device to 'xc7z020clg400-1'
INFO: [SIM 211-2] ***** CSIM start *****
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
Compiling(apcc) ../.././././fir_test.c in debug mode
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2018.2/bin/unwrapped/win64.o/apcc.exe'
INFO: [HLS 200-10] For user 'Afroditi' on host 'desktop-ii5sp0k' (Windows NT_amd64 version 6.2) on Mon Jun 14 11:48:27 +0300 2021
INFO: [HLS 200-10] In directory 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is apcc_db
INFO: [APCC 202-1] APCC is done.
Compiling(apcc) ../.././././fir.c in debug mode
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2018.2/bin/unwrapped/win64.o/apcc.exe'
INFO: [HLS 200-10] For user 'Afroditi' on host 'desktop-ii5sp0k' (Windows NT_amd64 version 6.2) on Mon Jun 14 11:48:35 +0300 2021
INFO: [HLS 200-10] In directory 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir/solution1/csim/build'
INFO: [APCC 202-3] Tmp directory is apcc_db
INFO: [APCC 202-1] APCC is done.
Generating csim.exe
Comparing against output data
*****
PASS: The output matches the golden output!
*****
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] ***** CSIM finish *****
Finished C simulation.
```

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.510	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
41	41	41	41	none

Detail

Instance

Loop

Loop Name	Latency		Initiation Interval		Trip Count	Pipelined
	min	max	Iteration Latency	achieved target		
- Shift_Accum_Loop	40	40	5	- -	8	no

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	0	81
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	0	-	68	5
Multiplexer	-	-	-	116
Register	-	-	147	-
Total	0	3	215	202
Available	280	220	106400	53200
Utilization (%)	0	1	~0	~0

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fir	return value
ap_rst	in	1	ap_ctrl_hs	fir	return value
ap_start	in	1	ap_ctrl_hs	fir	return value
ap_done	out	1	ap_ctrl_hs	fir	return value
ap_idle	out	1	ap_ctrl_hs	fir	return value
ap_ready	out	1	ap_ctrl_hs	fir	return value
y	out	32	ap_vld	y	pointer
y_ap_vld	out	1	ap_vld	y	pointer
x	in	32	ap_none	x	scalar

2. Δεδομένου ότι τα coefficients είναι ακέραιοι αριθμοί από το 1 μέχρι το 8 μπορούμε να τα παραστήσουμε με 4 bits. Επιπλέον έχουμε ότι τα inputs θα είναι αριθμοί των 8 bits άρα οι ενδιάμεσοι πολλαπλασιασμοί θα έχουν μήκος $8 + 4 = 12$ bits. Προκειμένου να έχουμε την ελάχιστη δυνατή ακρίβεια για όλους τους ενδιάμεσους υπολογισμούς θα πρέπει να αναπαραστήσουμε την μεταβλητή acc με 8 διαφορετικές μεταβλητές η κάθε μία με διαφορετικό bit width σύμφωνα με τον κανόνα πως κάθε πρόσθεση στη διαδικασία του accumulation προσθέτει και ένα ακόμα bit ακρίβειας στο αποτέλεσμα. Η ύπαρξη 8 διαφορετικών μεταβλητών οδηγεί και στην αλλαγή του κυρίως κώδικα καθώς πλέον το Sum_Accum_Loop θα πρέπει να ξεδιπλωθεί ώστε να χρησιμοποιηθεί ο κατάλληλος τύπος acc κάθε φορά. Οι κώδικες που υλοποιούν όσα συζητήθηκαν δίνονται παρακάτω:

fir.c

```
#include "fir.h"

void fir (acc_t8 *y, data_t x)
{
    coef_t c[N] = {1,2,3,4,5,6,7,8};
    static data_t shift_reg[N];
    acc_t1 acc1;  acc_t2 acc2;  acc_t3 acc3;  acc_t4 acc4;
    acc_t5 acc5;  acc_t6 acc6;  acc_t7 acc7;  acc_t8 acc8;
    data_t data;
    int i;

    acc1=0;
    shift_reg[7]=shift_reg[6];
    data = shift_reg[7];
    acc1 = data*c[7];

    shift_reg[6]=shift_reg[5];
    data = shift_reg[6];
    acc2 = acc1 + data*c[6];

    shift_reg[5]=shift_reg[4];
    data = shift_reg[5];
    acc3 = acc2 + data*c[5];

    shift_reg[4]=shift_reg[3];
    data = shift_reg[4];
    acc4 = acc3 + data*c[4];

    shift_reg[3]=shift_reg[2];
    data = shift_reg[3];
    acc5 = acc4 + data*c[3];

    shift_reg[2]=shift_reg[1];
    data = shift_reg[2];
    acc6 = acc5 + data*c[2];

    shift_reg[1]=shift_reg[0];
    data = shift_reg[1];
    acc7 = acc6 + data*c[1];

    shift_reg[0]=x;
    data = x;
    acc8 = acc7 + data*c[0];

    *y=acc8;
}
```

fir.h

```
#include <ap_cint.h>          /*Lib for arbitrary precision types*/
#ifndef FIR_H_
#define FIR_H_
#define N      8

typedef uint4   coef_t;
typedef uint8   data_t;
typedef uint12  mult_t;      /*Width of mul operation for 8bit input & 4bit coeff*/
typedef uint12  acc_t1;
typedef uint13  acc_t2;
```

```

typedef uint14 acc_t3;
typedef uint15 acc_t4;
typedef uint16 acc_t5;
typedef uint17 acc_t6;
typedef uint18 acc_t7;
typedef uint19 acc_t8;

void fir (acc_t8 *y, data_t x);

#endif

```

fir_test.c

```

#include <stdio.h>
#include <math.h>
#include "fir.h"

int main () {
    const int SAMPLES=17;
    FILE *fp;
    acc_t8 output;
    data_t signal[SAMPLES] = {40, 248, 245, 124, 204, 36, 107, 234, 202, 245, 0, 0, 0, 0, 0, 0, 0};

    int i, ramp_up;
    ramp_up = 1;

    fp=fopen("out.dat", "w");
    for (i=0; i<SAMPLES; i++) {

        // Execute the function with latest input
        fir(&output, signal[i]);

        // Save the results.
        fprintf(fp, "%i %d %d\n", i, signal[i], output);
    }
    fclose(fp);

    printf ("Comparing against output data \n");
    if (system("diff -w out.dat out.gold.dat")) {

        fprintf(stdout, "*****\n");
        fprintf(stdout, "FAIL: Output DOES NOT match the golden output\n");
        fprintf(stdout, "*****\n");
        return 1;
    } else {
        fprintf(stdout, "*****\n");
        fprintf(stdout, "PASS: The output matches the golden output!\n");
        fprintf(stdout, "*****\n");
        return 0;
    }
}

```

```

Starting C simulation ...
C:/Xilinx/Vivado/2018.2/bin/vivado_hls.bat C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir/solution2/csim.tcl
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2018.2/bin/unwrapped/win64.o/vivado_hls.exe'
INFO: [HLS 200-10] For user 'Afroditi' on host 'desktop-ii5sp0k' (Windows_NT amd64 version 6.2) on Mon Jun 14 15:23:37 +0300 2021
INFO: [HLS 200-10] In directory 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir'
INFO: [HLS 200-10] Opening project 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir'.
INFO: [HLS 200-10] Opening solution 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir/solution2'.
INFO: [SYN 201-201] Setting up clock 'default' with a period of 10ns.
INFO: [HLS 200-10] Setting target device to 'xc7z020clg400-1'
INFO: [SIM 211-2] ***** CSIM start *****
INFO: [SIM 211-4] CSIM will launch GCC as the compiler.
Compiling(apcc) ../../../../fir_test.c in debug mode
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2018.2/bin/unwrapped/win64.o/apcc.exe'
INFO: [HLS 200-10] For user 'Afroditi' on host 'desktop-ii5sp0k' (Windows_NT amd64 version 6.2) on Mon Jun 14 15:23:42 +0300 2021
INFO: [HLS 200-10] In directory 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir/solution2/csim/build'
INFO: [APCC 202-3] Tmp directory is apcc_db
INFO: [APCC 202-1] APCC is done.
Compiling(apcc) ../../../../fir.c in debug mode
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2018.2/bin/unwrapped/win64.o/apcc.exe'
INFO: [HLS 200-10] For user 'Afroditi' on host 'desktop-ii5sp0k' (Windows_NT amd64 version 6.2) on Mon Jun 14 15:23:51 +0300 2021
INFO: [HLS 200-10] In directory 'C:/Users/Afroditi/Documents/8thSemester/dvlsi/HLS_Bonus/fir/hls1-fir/solution2/csim/build'
INFO: [APCC 202-3] Tmp directory is apcc_db
INFO: [APCC 202-1] APCC is done.
Generating csim.exe
Comparing against output data
*****
PASS: The output matches the golden output!
*****
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] ***** CSIM finish *****
Finished C simulation.

```

Βλέπουμε ότι η υλοποίησή αυτή περνάει επιτυχώς το testing οπότε προχωράμε στη σύνθεση και την σύγκριση των αποτελεσμάτων:

	Χωρίς arbitrary precision types	Με arbitrary precision types
<i>Target Clock(ns)</i>	10.00	10.00
<i>Estimated Clock(ns)</i>	8.51	8.71
<i>Latency(clock cycle)</i>	41	1
<i>BRAM_18K</i>	0	0
<i>DSP48E</i>	3	0
<i>FF</i>	215	72
<i>LUT</i>	202	182

Παρατηρούμε ότι η περίοδος του ρολογιού αυξήθηκε παρόλο που μειώσαμε τον όγκο των δεδομένων ωστόσο το unrolling οδήγησε σε latency ίσο με την μονάδα. Όσον αφορά στο utilization βλέπουμε ότι η χρήση των arbitrary precision types οδήγησε σε μεγάλη μείωση και τα 3 components που χρησιμοποιήθηκαν.

3. Στον πίνακα που ακολουθεί βλέπουμε την σύγκριση του φίλτρου που παρουσιάστηκε στο προηγούμενο ερώτημα με το αντίστοιχο που είχαμε κληθεί να υλοποιήσουμε στο ζητούμενο 1 της εργαστηριακής άσκησης 3.

	FIR Εργ.Ασκ.3	FIR(arbitrary precision types)
<i>Estimated Clock(ns)</i>	6.215	8.71
<i>Latency(clock cycle)</i>	11	1
<i>BRAM_18K</i>	0	0
<i>DSP48E</i>	0	0
<i>FF</i>	0.38	0.06
<i>LUT</i>	0.67	0.34

Παρατηρούμε ότι αν και το προβλεπόμενο ρολόι είναι μικρότερο για το FIR φίλτρο της άσκησης 3 όσον αφορά στο Latency και Utilization εκείνο μένει πίσω σημαντικά.

Ζητούμενο 2

1. Οι κώδικες που υλοποιούν το classification με βάση την δοθείσα εξίσωση για SVM φαίνονται παρακάτω. Πρόκειται για το source code της συνάρτησης που ουσιαστικά εκτελεί τις πράξεις της εξίσωσης (svm.cpp), το testbench για τον έλεγχο του precision (svm_test.cpp) και το αρχείο svm.h όπου ορίζονται οι τύποι των μεταβλητών. Έχει γίνει μελέτη και σύνθεση τόσο για την χρήση floats όσο και για την χρήση arbitrary precision types όπως και στο προηγούμενο ζήτημα ακολουθώντας την σύμβαση του IEEE 754 για την αναπαράσταση των δεκαδικών αριθμών. Επισημαίνεται επίσης ότι τα .csv των support vector και coefficient ενσωματώθηκαν ως πίνακες σε ξεχωριστά αρχεία .h τα οποία γίνονται include στο main πρόγραμμα.

svm.cpp

```
#include <iostream>
#include "svm.h"
#include "supportvectors.h"
#include "coef.h"
#include <cmath>

using namespace std;

void svm (output_t * class_hw,
         input_t x[Dsv])
{
    norm_t norm;
    diff_t dif;
    double sum = 0.0;

    Sum_Outter_Loop:
    for(int i=0; i<Nsv; i++)
    {
        norm = 0.0;

        Sum_Inner_Loop:
        for(int j=0; j<Dsv; j++)
        {
            dif = (x[j] - SupVec[i][j]);
            norm += dif*dif;
        }

        sum += double(Co[i])*exp(double(g*double(norm))) - double(b);
    }

    if (sum > 0) *class_hw = 1;
    else *class_hw = -1;
}
```

svm_test.cpp

```
#include "svm.h"
#include <iostream>
#include <fstream>
#include <string>
#include <cmath>
using namespace std;

int main()
{
    const int SAMPLES = 1000;

    FILE * fin, * annot;

    static input_t x[Dsv];
    output_t * class_hw;

    int gold, counter = 0;

    fin = fopen("testing_set.csv", "r");
    annot = fopen("annotation.csv", "r");

    for(int j=0; j<SAMPLES; j++)
    {
        for(int i=0; i<Dsv; i++)
```

```

    {
        fscanf(fin, "%f", &x[i]);
    }
    fscanf(annot, "%d\n", &gold);
    svm(class_hw, x);

    if (gold != *class_hw) counter++;
}
fclose(fin);
fclose(annot);

cout << "*****" << endl;
cout << "* Precision " << 100.0 - (counter*1.0)/SAMPLES << " %" << endl;
cout << "*****" << endl;
return 0;
}

```

svm.h

```

#ifndef SVM_H_
#define SVM_H_

#include <ap_fixed.h>

#define Nsv 1222
#define Dsv 18

static const float b = 2.8180;
static const float g = -8.0;

//typedef ap_fixed<10,1> input_t;
//typedef char output_t;
//typedef ap_fixed<19,10> coeff_t;
//typedef ap_fixed<10,1> vector_t;
//typedef ap_fixed<16,1> norm_t;
//typedef ap_fixed<11,2> diff_t;

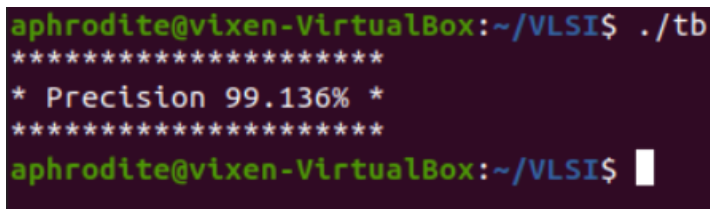
typedef float input_t;
typedef short output_t;
typedef float coeff_t;
typedef float vector_t;
typedef float norm_t;
typedef float diff_t;

void svm (
    output_t * class_hw,
    input_t x[Dsv]
);

#endif

```

Η ακρίβεια των αποτελεσμάτων είναι η εξής:



```

aphrodite@vixen-VirtualBox:~/VLSI$ ./tb
*****
* Precision 99.136% *
*****
aphrodite@vixen-VirtualBox:~/VLSI$

```

Επισημαίνω εδώ το segmentation error που αντιμετώπισα με το simulation στο Vivado και για το οποίο έχω ήδη ενημερώσει την υπεύθυνη του project. Όπως φαίνεται παραπάνω δεν έχει να κάνει με τον κώδικα αυτόν καθ' αυτόν αλλά είτε έχει να κάνει με το stack των windows είτε ίσως με κάποιο bug της version του εργαλείου ή της εγκατάστασης.

Το γεγονός αυτό δεν με άφησε να δω βελτιώσεις στο precision με την χρήση arbitrary precision types ωστόσο δίνονται σχολιασμένοι στο αρχείο svm.h και παρακάτω βλέπουμε και την βελτίωση που αυτοί επιφέρουν στο performance και το utilization:

Χωρίς arbitrary precision types

Με arbitrary precision types

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.232	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
403261	403261	403261	403261	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	146
FIFO	-	-	-	-
Instance	-	45	2889	5644
Memory	68	-	0	0
Multiplexer	-	-	-	394
Register	-	-	628	-
Total	68	45	3517	6184
Available	280	220	106400	53200
Utilization (%)	24	20	3	11

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.232	1.25

Latency (clock cycles)

Summary

Latency		Interval		Type
min	max	min	max	
125867	125867	125867	125867	none

Detail

Instance

Loop

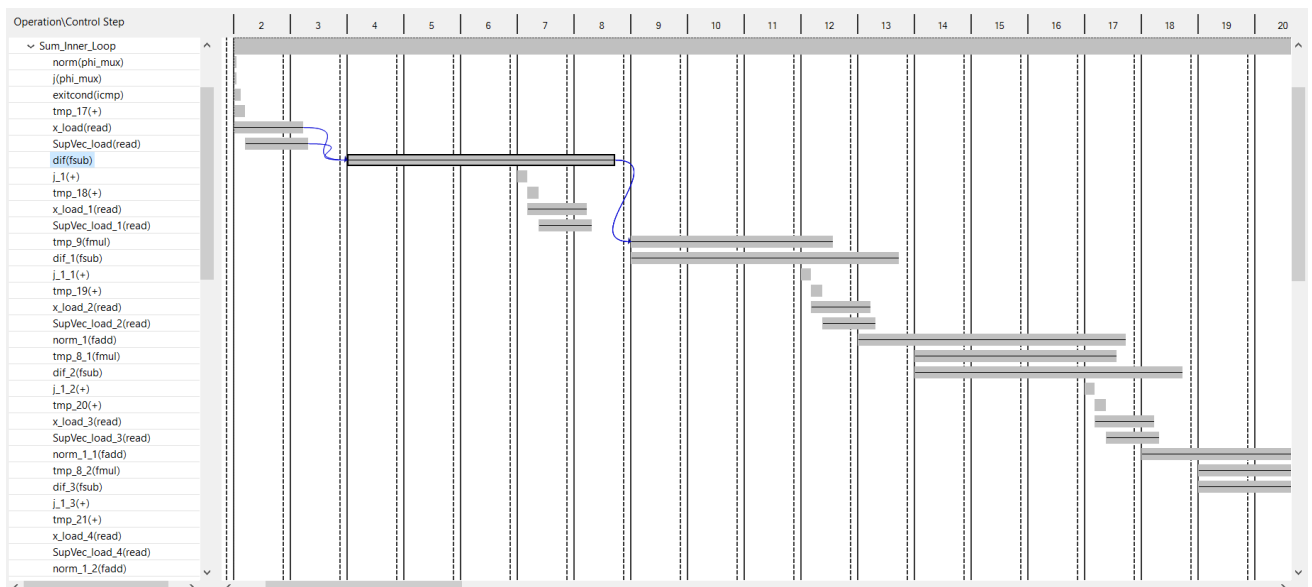
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	345
FIFO	-	-	-	-
Instance	-	40	2853	5440
Memory	23	-	0	0
Multiplexer	-	-	-	340
Register	-	-	670	-
Total	23	41	3523	6125
Available	280	220	106400	53200
Utilization (%)	8	18	3	11

Επειδή όπως αναφέρθηκε και παραπάνω η υλοποίηση με τις `ap_fixed` μεταβλητές δεν μπορεί να ελεγχθεί ως προς την ορθότητα και το precision (η βιβλιοθήκη `ap_fixed` υπάρχει μόνο στο Vivado HLS) θα προχωρήσουμε στην εργασία με την υλοποίηση χωρίς `arbitrary precision types`.

2. Εισάγουμε το Directive του Unroll με factor 9 στο `Sum_Inner_Loop` του κώδικα και ανοίγουμε το analysis tab του Vivado HLS. Η εικόνα για τους χρονισμούς του συγκεκριμένου loop που παίρνουμε από τον scheduler είναι η εξής:



Αρχικά εκείνο που παρατηρούμε είναι τα RAW dependencies μεταξύ των πράξεων της αφαίρεσης, του πολλαπλασιασμού και της πρόσθεσης. Επιπλέον γνωρίζουμε ότι διατίθενται και dual-ported BRAMs και επομένως θα μπορούσαμε να εκμεταλλευτούμε την παραλληλία διαβάζοντας σε κάθε κύκλο 2 στοιχεία από κάθε πίνακα (x[] και SupVec[][]).

Οι παραπάνω αστοχίες στον χρονισμό προκύπτουν διότι το UNROLL directive αντιστοιχεί στο μοτίβο κώδικα:

```
dif0 = (x[j] - SupVec[i][j]);
norm += dif0*dif0;
dif1 = (x[j+1] - SupVec[i][j+1]);
norm += dif1*dif1;
dif2 = (x[j+2] - SupVec[i][j+2]);
norm += dif2*dif2;
...
```

ενώ προκειμένου να επιλύσουμε τα παραπάνω και να επιτύχουμε καλύτερες επιδόσεις χρειαζόμαστε manual unrolling όπως φαίνεται στο παρακάτω ανανεωμένο svm.cpp:

svm.cpp

```
#include <iostream>
#include "svm.h"
#include <cmath>

using namespace std;

void svm(output_t * class_hw,
input_t x[Dsv])
{
    norm_t norm;
    diff_t dif;
    double sum = 0.0;

    Sum_Outer_Loop:
    for(int i=0; i<Nsv; i++)
    {
        norm = 0.0;

        Sum_Inner_Loop:
        for(int j=0; j<Dsv; j+=9)
        {
            diff_t dif0,dif1,dif2,dif3,dif4,dif5,dif6,dif7,dif8;
            norm_t mul0,mul1,mul2,mul3,mul4,mul5,mul6,mul7,mul8;

            dif0 = x[j] - SupVec[i][j];
            dif1 = x[j+1] - SupVec[i][j+1];
            dif2 = x[j+2] - SupVec[i][j+2];
            dif3 = x[j+3] - SupVec[i][j+3];
            dif4 = x[j+4] - SupVec[i][j+4];
            dif5 = x[j+5] - SupVec[i][j+5];
            dif6 = x[j+6] - SupVec[i][j+6];
            dif7 = x[j+7] - SupVec[i][j+7];
            dif8 = x[j+8] - SupVec[i][j+8];

            mul0 = dif0*dif0;
            mul1 = dif1*dif1;
            mul2 = dif2*dif2;
            mul3 = dif3*dif3;
            mul4 = dif4*dif4;
            mul5 = dif5*dif5;
            mul6 = dif6*dif6;
            mul7 = dif7*dif7;
            mul8 = dif8*dif8;

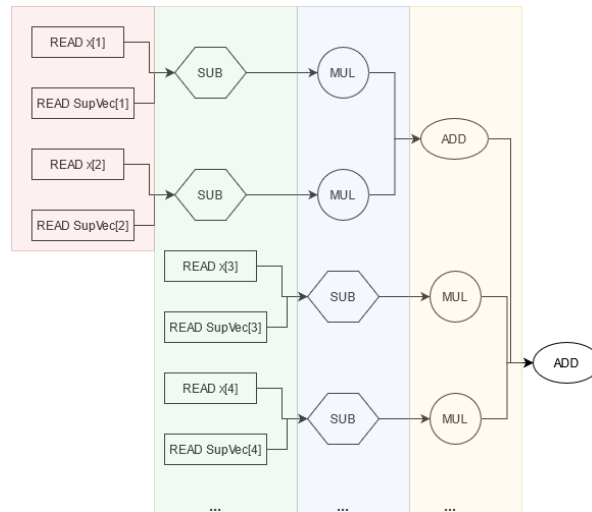
            norm = mul0 + mul1 + mul2 + mul3 + mul4 + mul5 + mul6 + mul7 + mul8;

        }

        sum += double(Co[i])*exp(double(g*double(norm))) - double(b);
    }

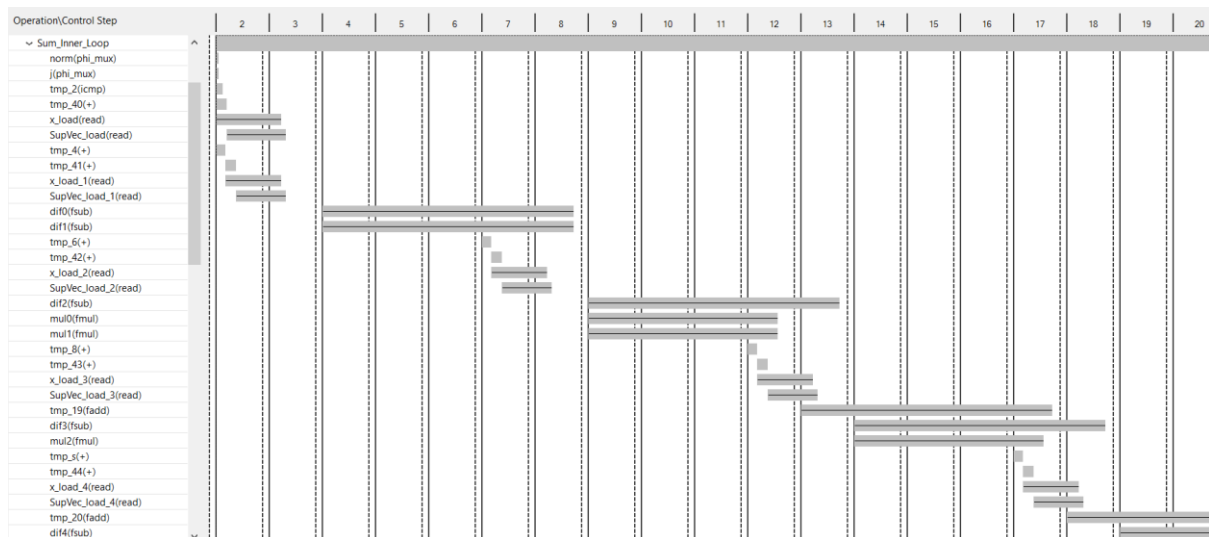
    if (sum > 0) *class_hw = 1;
    else *class_hw = -1;
}
```

Το θεωρητικά αναμενόμενο αποτέλεσμα εδώ θα ήταν να έχουμε 4 reads (2 για κάθε πίνακα) ανά κύκλο το οποίο οδηγεί σε μία εκτέλεση όπως περιγράφεται από το παρακάτω σχηματικό:



όπου με διαφορετικό χρώμα σημειώνονται οι διαφορετικοί κύκλοι.

Ωστόσο βλέπουμε ότι ο χρονισμός του εργαλείου δεν συμφωνεί και μετά τα 4 πρώτα ταυτόχρονα reads εκτελεί τα υπόλοιπα όπως και πριν. Επίσης φαίνεται να μην εκμεταλλεύεται καθόλου την έλλειψη dependencies από τον τρόπο που έχει γραφεί ο κώδικας

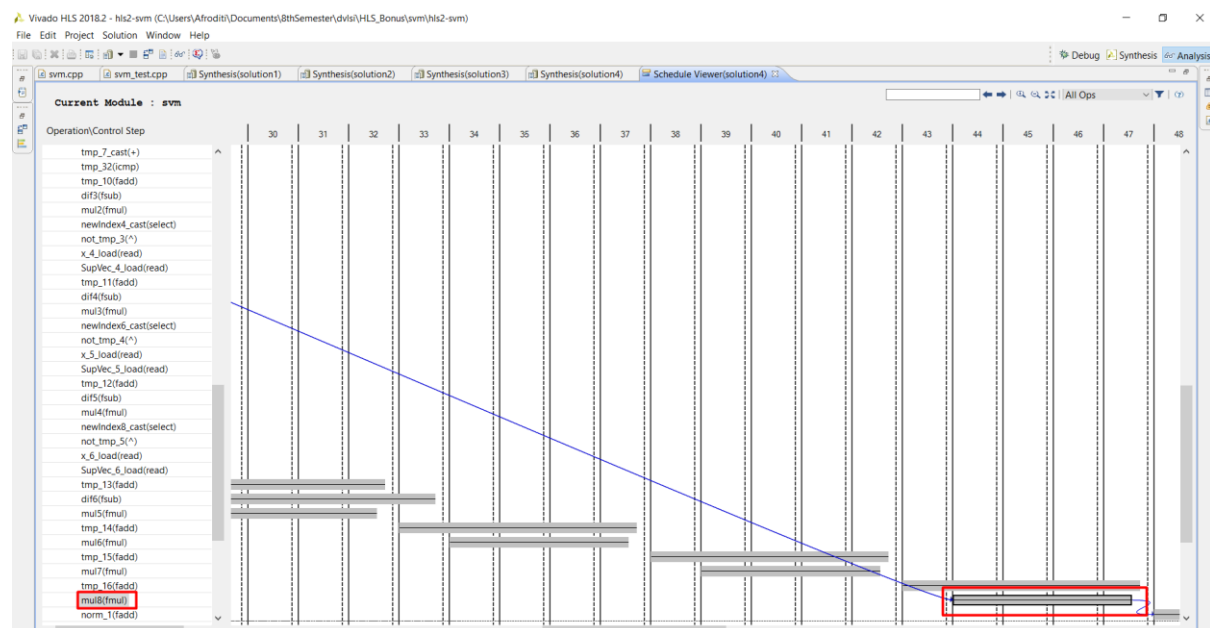
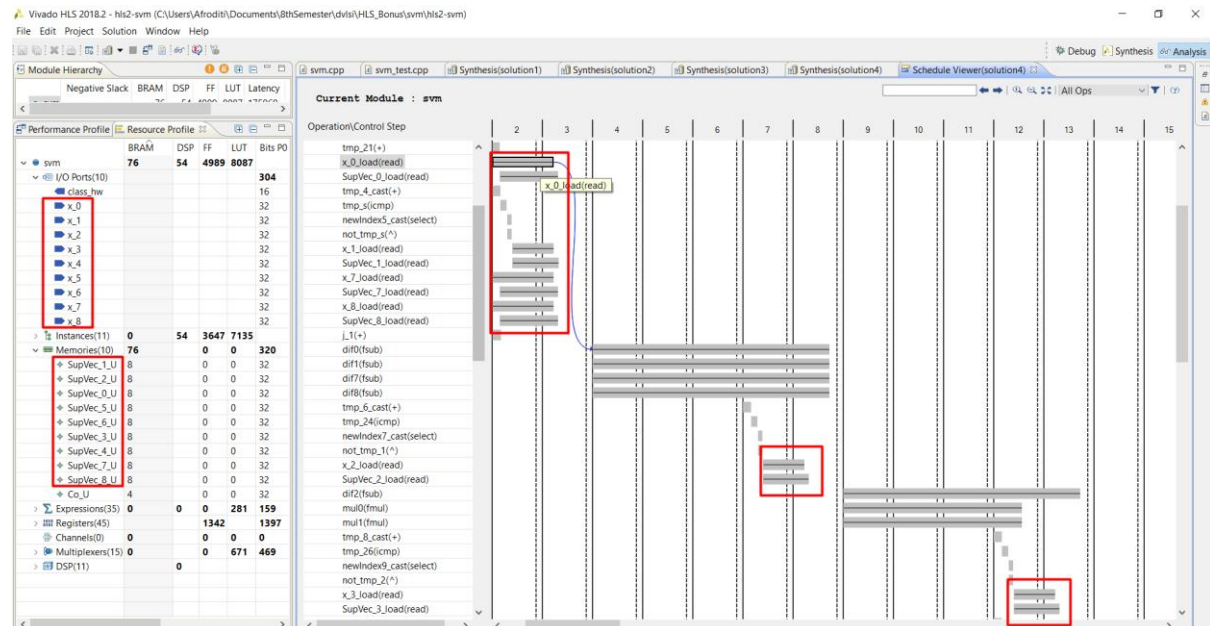


	Pragma	Manual Unroll
<i>Estimated Clock(ns)</i>	9.379	8.317
<i>Latency(clock cycle)</i>	188189	175969
<i>BRAM_18K</i>	24	24
<i>DSP48E</i>	20	22
<i>FF</i>	3	3
<i>LUT</i>	12	14

3. Αρχικά, με την παραδοχή πως γίνονται 2 reads σε κάθε πίνακα ανά κύκλο και πως χρειαζόμαστε διαδοχικά στοιχεία ένα κυκλικό partition με factor 5 θα ήταν αρκετό για να διαβάσουμε τα 9 στοιχεία που χρειαζόμαστε λόγω του unroll που κάναμε παραπάνω. Ωστόσο βλέπουμε ότι κάτι τέτοιο δίνει σημαντικά αυξημένο Latency σε σχέση με τις προηγούμενες υλοποιήσεις (320165). Έτσι προτιμάμε κυκλικό partition με factor 9 ώστε να μπορούν θεωρητικά να διαβαστούν και τα 9 στοιχεία του κάθε πίνακα ταυτόχρονα, μέσω των

```
#pragma HLS array_partition variable=x cyclic factor=9
#pragma HLS array_partition variable=SupVec cyclic factor=9 dim=2
```

. Ωστόσο ξανά στο analysis perspective του εργαλείου βλέπουμε μία περίεργη συμπεριφορά :



Πιο συγκεκριμένα :

1. Αρχικά διαβάζονται τα 2 πρώτα και 2 τελευταία στοιχεία των πινάκων (indexes 0,1,7,8), στην συνέχεια γίνονται κάποια subs/diffs και μετά διαβάζεται 1 στοιχείο σε

- κάθε "κύκλο" (θεωρητικά θα έπρεπε να διαβαστούν όλα ταυτόχρονα εξ αρχής εφόσον υπάρχει η δυνατότητα και όπως δείχνουν τα resources έχει γίνει το partition),
2. Στο δεύτερο screenshot, σημειώνεται ο πολλαπλασιασμός mul8 που ουσιαστικά χρησιμοποιεί το diff8. Βλέπουμε πως είναι σαν να το περιμένει ενώ το diff8 είναι από τα πρώτα diffs που έχουν υπολογιστεί (όπως φαίνεται και από το μπλε βελάκι που δείχνει την εξάρτηση και πάει προς τα πάνω, προς την αρχή δηλαδή του loop).
 3. Επίσης φαίνεται οι τελευταίοι πολλαπλασιασμοί να γίνονται όλοι σειριακά, σαν να έχει φτιαχτεί μόνο ένας πολλαπλασιαστής και να χρησιμοποιείται αυτός.

Τέλος και η απόδοση φαίνεται να μένει σταθερή ενώ θα περιμέναμε να δούμε το optimization που επιφέρει το partition:

	Without Partition	With Partition
<i>Estimated Clock(ns)</i>	8.317	8.437
<i>Latency(clock cycle)</i>	175969	175969
<i>BRAM_18K</i>	24	27
<i>DSP48E</i>	22	24
<i>FF</i>	3	4
<i>LUT</i>	14	15

4. Προκειμένου να καταφέρουμε να ενσωματώσουμε στο FPGA $n(>1)$ παράλληλα instances του SVM ουσιαστικά θα προσθέσουμε μια ακόμη συνάρτηση στα sources και θα ορίσουμε πλέον αυτήν ως top function. Ονομάζουμε την συνάρτηση αυτή svm_top.cpp.

svm.cpp

```
#include <iostream>
#include "svm.h"
#include <cmath>

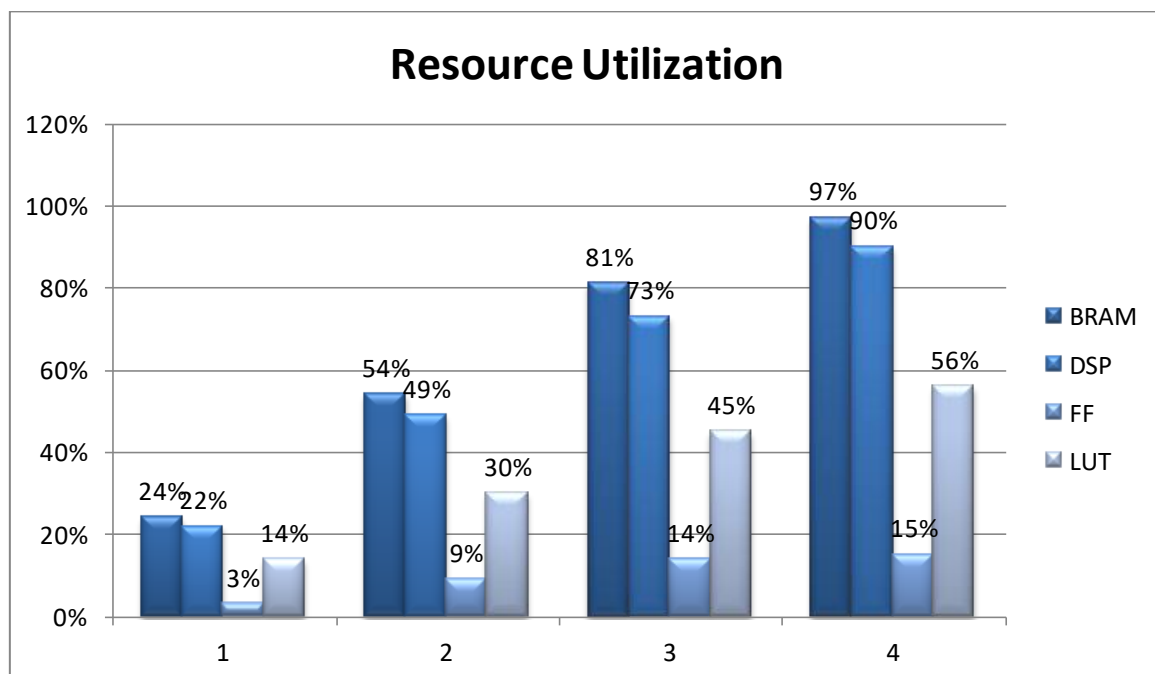
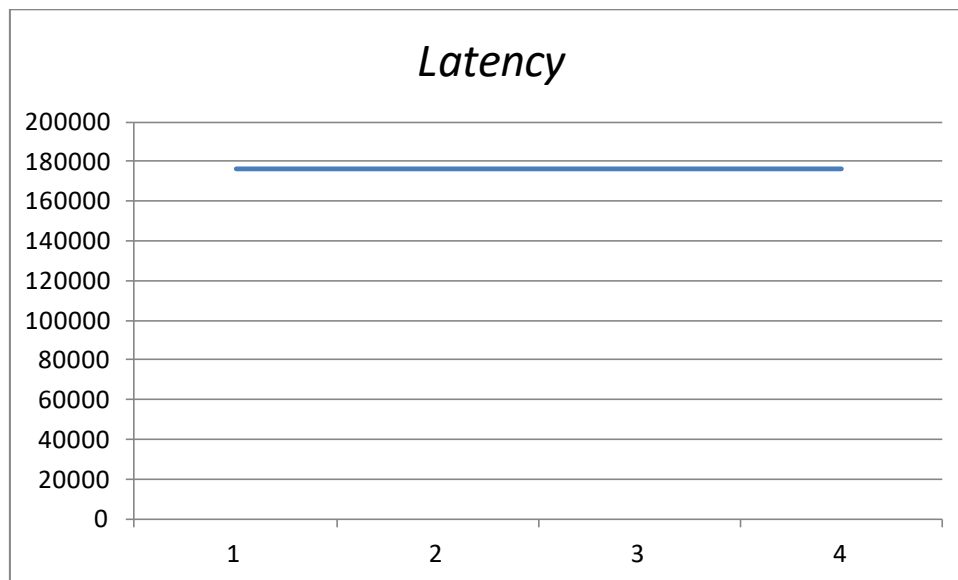
using namespace std;

void svm_top(
    output_t * class_hw0,
    output_t * class_hw1,
    output_t * class_hw2,
    output_t * class_hw3,
    input_t x0[Dsv],
    input_t x1[Dsv],
    input_t x2[Dsv],
    input_t x3[Dsv])
{
    #pragma HLS DATAFLOW
    svm(class_hw0, x0);
    svm(class_hw1, x1);
    svm(class_hw2, x2);
    svm(class_hw3, x3);
}
```

Όπως φαίνεται αυτό που κάνει η συνάρτηση αυτή είναι να καλεί πολλές φορές την svm() που ορίσαμε παραπάνω έτσι ώστε χρησιμοποιώντας το pragma του DATAFLOW να μπορέσουν τα n αυτά instances να εκτελούνται παράλληλα. Τα synthesis reports δίνουν τα ακόλουθα δεδομένα για τις υλοποιήσεις (ως optimized SVM kernel χρησιμοποιείται εκείνος χωρίς το array partition σύμφωνα με τα προηγούμενα αποτελέσματα του εργαλείου):

	1 SVM Kernel	2 SVM Kernels
<i>Estimated Clock(ns)</i>	8.317	8.427
<i>Latency(clock cycle)</i>	175969	175969
<i>BRAM_18K</i>	24	54
<i>DSP48E</i>	22	49
<i>FF</i>	3	9
<i>LUT</i>	14	30

Αυξάνοντας κατά 1 τα instances βλέπουμε ότι μπορούμε να «χωρέσουμε» έως 4 πάνω στο FPGA καθώς στο 5^ο κάνει exceed τα διαθέσιμα resources (BRAMs και DSPs). Ακολουθούν συγκριτικά διαγράμματα για τις υλοποιήσεις με 1, 2, 3 και 4 πυρήνες:



Τέλος, επισημαίνεται πως τόσο η version του Vivado όσο και η εγκατάσταση του σε Windows δημιουργούσε προβλήματα στην λειτουργία του simulation όταν εισάγονταν η βιβλιοθήκη `ap_fixed` για `arbitrary precision types`. Σε περίπτωση που χρησιμοποιούνταν αυτά ίσως να μπορούσαν ένα «χωρέσουν» και παραπάνω από 4 instances με την προσθήκη του `pragma RESOURCE` για τον καθορισμό συγκεκριμένου αριθμού DSPs προκειμένου να μην υπερβούμε το 100% στο utilization και να χρησιμοποιηθούν στη θέση τους LUTs.