ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

Προγραμματισμός Ενσωματωμένων Συστημάτων σε Περιβάλλοντα Edge

SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF INFORMATICS AND TELEMATICS

POSTGRADUATE STUDIES

WEB ENGINEERING

Εξαμηνιαία εργασία για το μάθημα

Vivado HLS, Verilog & Arduino

Γεωργάρα Βασιλική - 21107

Τσερπέ Παρασκευή - 21117

Διδάσκοντες:

Sotirios Xydis, Assistant Professor in the area of Digital Design and Reliable Embedded Systems

Dimopoulos Alexandros, Adjunct Professor in the area of Digital Design and Reliable Embedded Systems

ATHENS

MAY 2022

Θέμα 1: Υλοποίηση Convolution accelerator σε FPGA μέσω High Level Synthesis

---

Η συνέλιξη (convolution) είναι μια σημαντική λειτουργία στην επεξεργασία σήματος και εικόνας καθώς και σε εφαρμογές μηχανικής και βαθιάς μηχανικής μάθησης. Η συνέλιξη λειτουργεί σε δύο σήματα (είτε 1D σήματα είτε σε 2D π.χ. εικόνες). Στην περίπτωση της 2D συνέλιξης, μπορείτε να θεωρήσετε το ένα ως την εικόνα "εισόδου" και το άλλο (που ονομάζεται πυρήνας) ως "φίλτρο" στην εικόνα εισόδου. Η λειτουργία της συνέλιξης παράγει μια εικόνα εξόδου, η οποία έχει μετασχηματιστεί σύμφωνα με τη λειτουργία που προδιαγράφεται στον πίνακα του πυρήνα/φίλτρου.
Έστω:
- *$G(c, u, v)$ ο πίνακας του πυρήνα, όπου c: αριθμός καναλιών του πίνακα πυρήνα, u: ύψος του πίνακα πυρήνα G και v: πλάτος του πίνακα πυρήνα G.*
- *D(c, x, y) ο πίνακας της εικόνας εισόδου, όπου c : αριθμός καναλιών του πίνακα εισόδου D, x: ύψος του πίνακα D και y: το πλάτος του πίνακα D.*

Η συνέλιξη μεταξύ του πίνακα της εικόνας εισόδου D(c, x, y) και του πίνακα του πυρήνα $G(c, u, v)$ εκτελείται μέσω της λειτουργίας dot-product:

$$Y_{x,y} = \sum_{c=1}^{C} \sum_{r=1}^{u} \sum_{r=1}^{u} D_{c,x+r} G_{c,r,r}$$

, η οποία μπορεί να περιγραφεί ως εξής: Ο πυρήνας $G(c, u, v)$ εφαρμόζεται πάνω στον πίνακα εισόδου D(c, x, y). Τα στοιχεία στην αντίστοιχη θέση του πίνακα εισόδου D(c, x, y) και του πυρήνα $G(c, u, v)$ πολλαπλασιάζονται και τα γινόμενα τους αθροίζονται ως μία έξοδος (ένα dot-product). Ο πίνακας του πυρήνα ολισθαίνει στον πίνακα εισόδου κατά μία θέση και επαναλαμβάνει τη λειτουργία του dot-product, δηλ. κάθε βήμα υπολογίζει μια θέση του πίνακα εξόδου Y με μέγεθος (c, x - u + 1, y - v + 1). Το παρακάτω σχήμα απεικονίζει ένα βήμα της διαδικασίας συνέλιξης.
Έστω ότι έχετε στη διάθεση σας ένα FPGA τύπου Xilinx Zynq-7000 με τα εξής χαρακτηριστικά {#LUT: 14400, #FF: 28800, #DSP: 66, #BRAM: 50} με speed grade -3. Ζητούμενα:
1) Να υλοποιηθεί μέσω HLS ένας επιταχυντής υλικού για τη λειτουργία της συνέλιξης και να προσομοιωθεί η λειτουργία του σε επίπεδο RTL. Να θεωρήσετε εικόνα εισόδου 128x128 pixel (έστω το κάθε pixel αποτελείται από 8-bit) με 1 κανάλι εισόδου, π.χ greyscale εικόνα.
2) Να προχωρήσετε σε σύνθεση τον επιταχυντή και να καταγράψετε την κατανάλωση των υπολογιστικών πόρων υλικού του FPGA για την περίπτωση που εφαρμόζονται πυρήνες διαφορετικού μεγέθους, 2x2, 4x4 και 8x8. Να θεωρήσετε πυρήνες εξομάλυνσης, για τα στοιχεία των οποίων ισχύει η ακόλουθη σχέση: $G(u, v) = 1 / N$ x M, όπου N και M οι διαστάσεις του εκάστοτε πίνακα.
3) Να εξάγετε και να προτείνετε κατάλληλες "στρατηγικές" χρήσης HLS directives (θεωρήστε κατ' ελάχιστο συνδυασμούς με memory και loop HLS directives), οι οποίες θα παράγουν μια βελτιστοποιημένη υλοποίηση σε ότι αφορά τον χρόνο εκτέλεσης για κάθε έναν από τους επιταχυντές υλικού του ερωτήματος 2, με διαφορετικό μέγεθος πυρήνα. Να εξηγήσετε και αξιολογήσετε τα κέρδη της βελτιστοποίησης.
4) Να επαναλάβετε τη μελέτη του ερωτήματος 2 για δυο διαφορετικές συσκευές FPGA της επιλογής σας, μια με περισσότερα και μια με λιγότερα resources.

Τεχνικη Αναφορα

Περιγραφη

Η συνέλιξη (convolution) είναι μία μαθηματική μέθοδος που χρησιμοποιείται ευρέως στην ανάλυση σημάτων. Σε κάθε pixel της εικόνας εισόδου (input) τοποθετούμε ένα μητρώο συνέλιξης (convolution mask) που ονομάζεται Kernel. Η καινούργια τιμή του pixel στην αντίστοιχη θέση τις εικόνας εξόδου (output) υπολογίζεται από την "μέση βαρύτητα" (weighted average) όλων των γειτονικών pixel. Ενας πίνακας συνέλιξης (kernel) μπορεί να έχει διαστάσεις 1x1, 2x2, 3x3, κ.ο.κ. Οσο μεγαλύτερες διαστάσεις εχει ο πίνακας (kernel) τόσο αυξάνει και ο "βαθμός ελευθερίας" της επεξεργασίας, λόγω του μεγαλύτερου αριθμού γειτονικών pixel που χρησιμοποιούνται. Η βέλτιστη διάσταση ενός

kernel που εξασφαλίζει και αξιοπιστία στη διαδικασία αλλά και ελαχιστοποίηση υπολογιστικού χρόνου είναι 3x3. Κλείνοντας, θα πρέπει να αναφερθεί ότι το μέγεθος του πίνακα εξόδου θα είναι μειωμένο κατά 2 σε κάθε διεύθυνση εξ αιτίας του "φαινομένου ορίου" (edge effect).

Στην εργασία μας επιλέξαμε να εργαστούμε με τη διάσημη φωτογραφία της Lena, η οποία παρουσιάζεται παρακάτω, έχει διαστάσεις 128x128 και είναι greyscale.

Θεωρούμε πυρήνες εξομάλυνσης διαστάσεων 3x3, 5x5 και 7x7, οι οποίοι παρουσιάζονται παρακάτω, τα στοιχεία των οποίων ποικίλουν. Το 3x3 είναι το gaussian blur, το 5x5 είναι οι τιμές του Laplacian of gaussian και το 7x7 είναι το gaussian blur επίσης.

Kernel 3x3:

1	2	1
2	4	2
1	2	1

Kernel 5x5:

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Kernel 7x7:

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Οι παραπάνω πυρήνες εφαρμόστηκαν ξεχωριστά στο πρόγραμμά μας και πάραξαν πίνακες της output εικόνας διαστάσεων 126x126, 124x124 και 122x122.

Η σχέση είναι:
Output Dimensions = Input Dimensions - Kernel Dimensions + 1.

Το source file μας είναι το functions.cpp μαζί με το header file functions.hpp.
Το test bench file μας είναι το convolution.cpp.

Στο test bench file, τα pixel της εικόνας εισόδου δίνονται καρφωτά από ένα python file imageToPixels.py, το οποίο μέσω της βιβλιοθήκης OpenCV μετατρέπει την εικόνα σε pixels.

Αρχικά, δημιουργούμε ένα νέο project στο Vivado HLS 2020.1. Αφού φορτώσουμε τα αρχεία μας, ο τύπος του accelerator που επιλέγουμε, σύμφωνα με την εκφώνηση, είναι ένα FPGA τύπου Xilinx Zynq-7000 με τα εξής χαρακτηριστικά: {#LUT: 14400, #FF: 28800, #DSP: 66, #BRAM: 50} με speed grade -2.

C SIMULATION

Για να διασφαλίσουμε ότι ο C++ κώδικάς μας είναι σωστός και τρέχει χωρίς λάθη, πριν ξεκινήσουμε την ανάλυση, τρέχουμε το C Simulation για κάθε πυρήνα. Το πρόγραμμά μας εκτυπώνει τον output πίνακα.

Για τον πυρήνα 3x3:

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../Downloads/2dconv_2nd/convolution.cpp in debug mode
   Compiling ../../../../Downloads/2dconv_2nd/functions.cpp in debug mode
   Generating csim.exe
THIS IS THE OUTPUT IMAGE
[(2193,2133,2034,1855,1544,1139,811,689,731,816,872,906,928,928,920,929,968,1035,1118,1204,1275,1328,1379,1424,1454,1477,1496,1508,1510,1510,1516,1515,1508,1517,153

Test passed.
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Για τον πυρήνα 5x5:

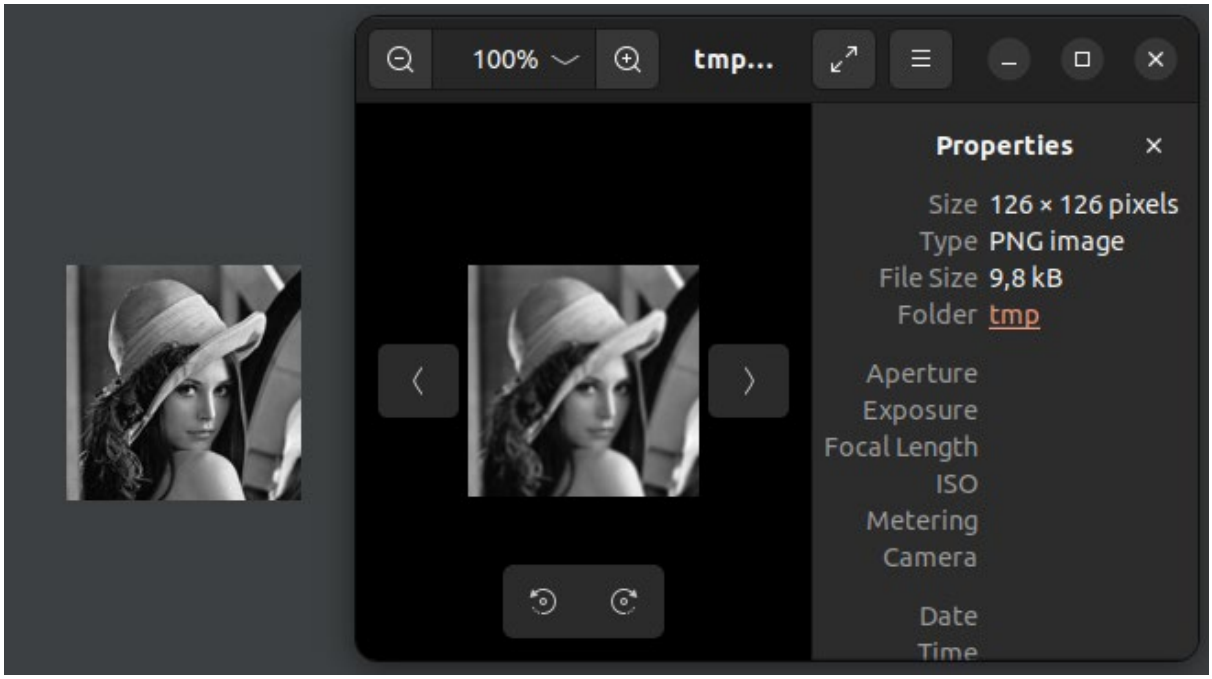
```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../Downloads/2dconv_2nd/convolution.cpp in debug mode
   Compiling ../../../../Downloads/2dconv_2nd/functions.cpp in debug mode
   Generating csim.exe
THIS IS THE OUTPUT IMAGE WITH KERNEL 5x5
[(-21,24,63,55,-65,-115,-80,6,39,14,-5,-21,1,4,0,7,-9,-23,10,22,-3,-17,29,28,0,0,9,8,-28,-2,14,-19,-11,27,0,-15,11,6,9,19,3,-12,11,-2,24,5,-9,4,4,20,23,8,-20,23,-4,

Test passed.
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Για τον πυρήνα 7x7:

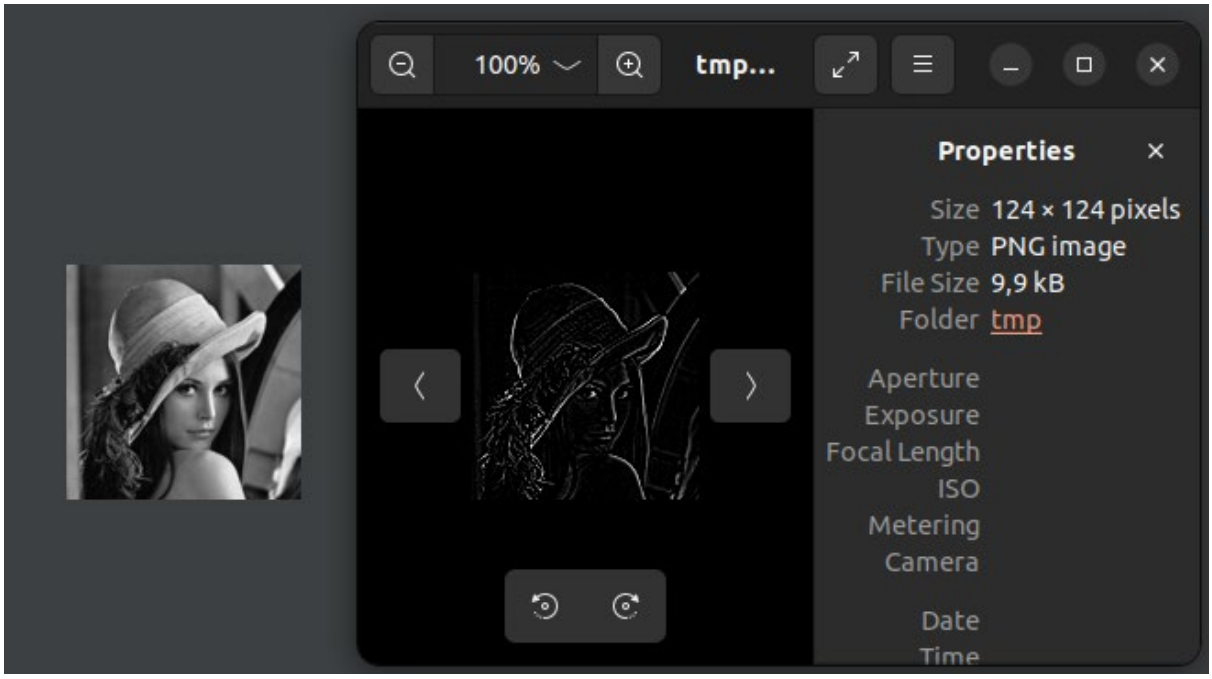
```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../Downloads/2dconv_2nd/convolution.cpp in debug mode
   Compiling ../../../../Downloads/2dconv_2nd/functions.cpp in debug mode
   Generating csim.exe
THIS IS THE OUTPUT IMAGE WITH KERNEL 7x7
[(131201,119877,100462,76865,57315,47971,47933,51817,55450,57423,58131,57933,57991,59199,61945,65954,71174,76627,81260,84582,87204,89739,91746,93472,94575,95193,952

Test passed.
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Για κάθε περίπτωση πυρήνα, για δικό μας οπτικό έλεγχο ότι το πρόγραμμά μας τρέχει λογικά σωστά, ο output πίνακας που εκτυπώνεται, μπαίνει καρφωτά σε ένα άλλο python file pixelsToImage.py, το οποίο μας εμφανίζει στην οθόνη την φιλτραρισμένη εικόνα.

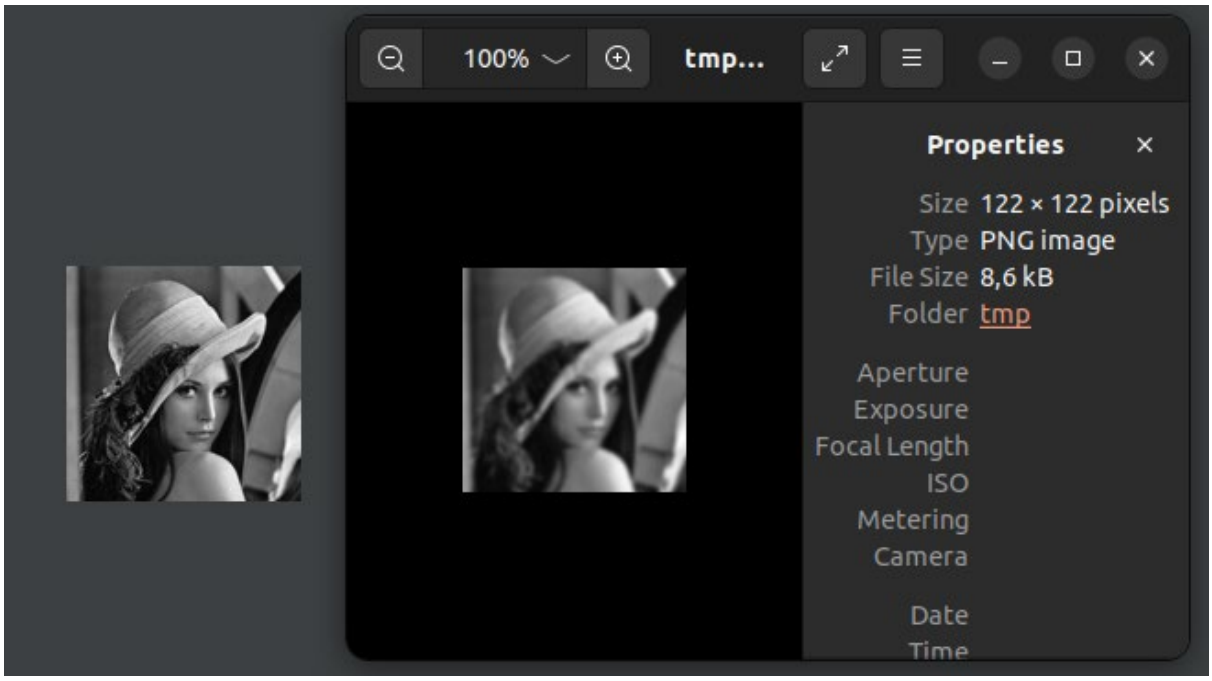
Για τον πυρήνα 3x3:



Για τον πυρήνα 5x5:



Για τον πυρήνα 7x7:



Σε όλες τις περιπτώσεις παρατηρούμε ότι έχει εφαρμοστεί σωστά η συνέλιξη, το πρόγραμμά μας τρέχει ορθά και έχουμε τα αναμενόμενα αποτελέσματα.

C Synthesis

Ύστερα από το επιτυχημένο C Simulation, μπορούμε να προχωρήσουμε και σε C Synthesis, το οποίο θα μετατρέψει τον c++ κώδικα σε Verilog, και θα παραχθεί το hardware της συνάρτησης. Αυτή η διαδικασία μας βγάζει κάποια reports.

Δημιουργήσαμε τρία solutions για κάθε πυρήνα και τα αρχικά αποτελέσματα που πήραμε παρουσιάζονται παρακάτω.

Για τον πυρήνα 3x3:

Synthesis Report for 'convolute'

General Information

Date:	Tue Jun 28 20:49:04 2022
Version:	2020.1 (Build 2897737 on Wed May 27 20:21:37 MDT 2020)
Project:	2d_conv_3rd
Solution:	solution1_kernel3x3_noDirectives
Product family:	zynq
Target device:	xc7z007s-clg225-2

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.050 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
698797	698797	6.988 ms	6.988 ms	698797	698797	none

Detail

- Instance
- Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	139	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	23	-
Register	-	-	230	-	-
Total	0	3	230	162	0
Available	100	66	28800	14400	0
Utilization (%)	0	4	~0	1	0

Όπως βλέπουμε παραπάνω, στο timing report, εμείς κατά την δημιουργία του project ζητήσαμε ρολόι 10ns, αλλά μπόρεσε να το αναπρογραμματίσει στα 7.050ns, το οποίο είναι μια εκτίμηση για να βελτιωθεί ο χρόνος. Το uncertainty είναι η καθυστέρηση του παλμού του ρολογιού με την default τιμή στα 1.25ns.

Στο latency report φαίνονται οι κύκλοι που χρειάστηκαν, οι οποίοι είναι κοντά στους 700.000.

Στο utilization estimates βλέπουμε το hardware που χρειάστηκε.

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	convolute	return value
ap_rst	in	1	ap_ctrl_hs	convolute	return value
ap_start	in	1	ap_ctrl_hs	convolute	return value
ap_done	out	1	ap_ctrl_hs	convolute	return value
ap_idle	out	1	ap_ctrl_hs	convolute	return value
ap_ready	out	1	ap_ctrl_hs	convolute	return value
output_r_address0	out	14	ap_memory	output_r	array
output_r_ce0	out	1	ap_memory	output_r	array
output_r_we0	out	1	ap_memory	output_r	array
output_r_d0	out	32	ap_memory	output_r	array
input_r_address0	out	14	ap_memory	input_r	array
input_r_ce0	out	1	ap_memory	input_r	array
input_r_q0	in	32	ap_memory	input_r	array
kernel_address0	out	4	ap_memory	kernel	array
kernel_ce0	out	1	ap_memory	kernel	array
kernel_q0	in	32	ap_memory	kernel	array

Στην παραπάνω εικόνα βλέπουμε επίσης λεπτομέρειες για το interface. Πληροφορίες όπως αν το σήμα είναι in ή out, πόσα bits χρειάζονται για να υλοποιηθούν τα σήματα, ο τύπος του interface, σε ποιο αντικείμενο αναφέρεται και ο τύπος της C.

Για τους υπόλοιπους πυρήνες σας παρουσιάζουμε τα αποτελέσματα και σχολιάζονται συνολικά. Το interface summary παραλείπεται διότι δεν έχει σημασία.

Για τον πυρήνα 5x5:

Synthesis Report for 'convolute'

General Information

Date:	Tue Jun 28 20:58:33 2022
Version:	2020.1 (Build 2897737 on Wed May 27 20:21:37 MDT 2020)
Project:	2d_conv_3rd
Solution:	solution2_kernel5x5_noDirectives
Product family:	zynq
Target device:	xc7z007s-clg225-2

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.050 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
1722361	1722361	17.224 ms	17.224 ms	1722361	1722361	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	141	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	23	-
Register	-	-	234	-	-
Total	0	3	234	164	0
Available	100	66	28800	14400	0
Utilization (%)	0	4	~0	1	0

Για τον πυρήνα 7x7:

Synthesis Report for 'convolute'

General Information

Date:	Tue Jun 28 21:00:51 2022
Version:	2020.1 (Build 2897737 on Wed May 27 20:21:37 MDT 2020)
Project:	2d_conv_3rd
Solution:	solution3_kernel7x7_noDirectives
Product family:	zynq
Target device:	xc7z007s-clg225-2

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.050 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
3155653	3155653	31.557 ms	31.557 ms	3155653	3155653	none

Detail

- Instance
- Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	3	0	141	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	26	-
Register	-	-	250	-	-
Total	0	3	250	167	0
Available	100	66	28800	14400	0
Utilization (%)	0	4	~0	1	0

Παρατηρούμε, όπως θεωρούμε λογικό, οι κύκλοι των διαφορετικών kernels να αυξάνονται.

Στην παρακάτω εικόνα, βλέπουμε την σύγκριση των τριών solution με τα διαφορετικά kernels.

Vivado HLS Report Comparison

All Compared Solutions

solution1_kernel3x3_noDirectives: xc7z007s-clg225-2
solution2_kernel5x5_noDirectives: xc7z007sclg225-2
solution3_kernel7x7_noDirectives: xc7z007sclg225-2

Performance Estimates

Timing

Clock		solution1_kernel3x3_noDirectives	solution2_kernel5x5_noDirectives	solution3_kernel7x7_noDirectives
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	7.050 ns	7.050 ns	7.050 ns

Latency

		solution1_kernel3x3_noDirectives	solution2_kernel5x5_noDirectives	solution3_kernel7x7_noDirectives
Latency (cycles)	min	698797	1722361	3155653
	max	698797	1722361	3155653
Latency (absolute)	min	6.988 ms	17.224 ms	31.557 ms
	max	6.988 ms	17.224 ms	31.557 ms
Interval (cycles)	min	698797	1722361	3155653
	max	698797	1722361	3155653

Utilization Estimates

	solution1_kernel3x3_noDirectives	solution2_kernel5x5_noDirectives	solution3_kernel7x7_noDirectives
BRAM_18K	0	0	0
DSP48E	3	3	3
FF	230	234	250
LUT	162	164	167
URAM	0	0	0

Resource Usage Implementation

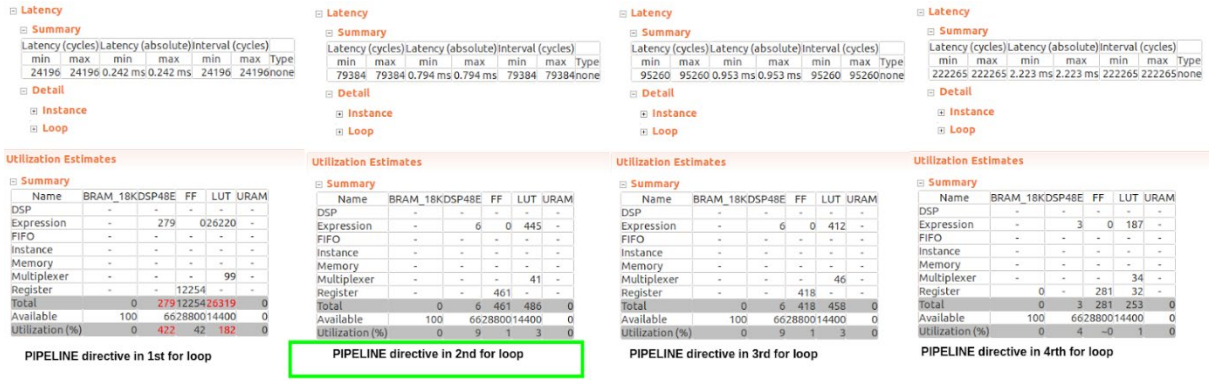
	solution1_kernel3x3_noDirectives	solution2_kernel5x5_noDirectives	solution3_kernel7x7_noDirectives
RTL	verilog	verilog	verilog
SLICE	-	-	-
LUT	-	-	-
FF	-	-	-
DSP	-	-	-
SRL	-	-	-
BRAM	-	-	-

DIRECTIVES

Για την εφαρμογή των στρατηγικών, αποφασίσαμε να ακολουθήσουμε δύο μεθόδους. Η μία μέθοδος θα εφαρμόζει μια στρατηγική τη φορά σε διαφορετικά σημεία του κώδικα και θα παρουσιάζεται το σημείο με τους χαμηλότερους κύκλους και utilization, ενώ η δεύτερη μέθοδος θα εφαρμόζει συνδυαστικά πολλές στρατηγικές και θα παρουσιάζεται πάλι η καλύτερη. Για όλες τις δοκιμές, χρησιμοποιούμε στρατηγικές για loop και memory. Επίσης οι δοκιμές θα παρουσιαστούν για τα kernels 3x3 και 7x7.

Kernel 3x3 - Method 1

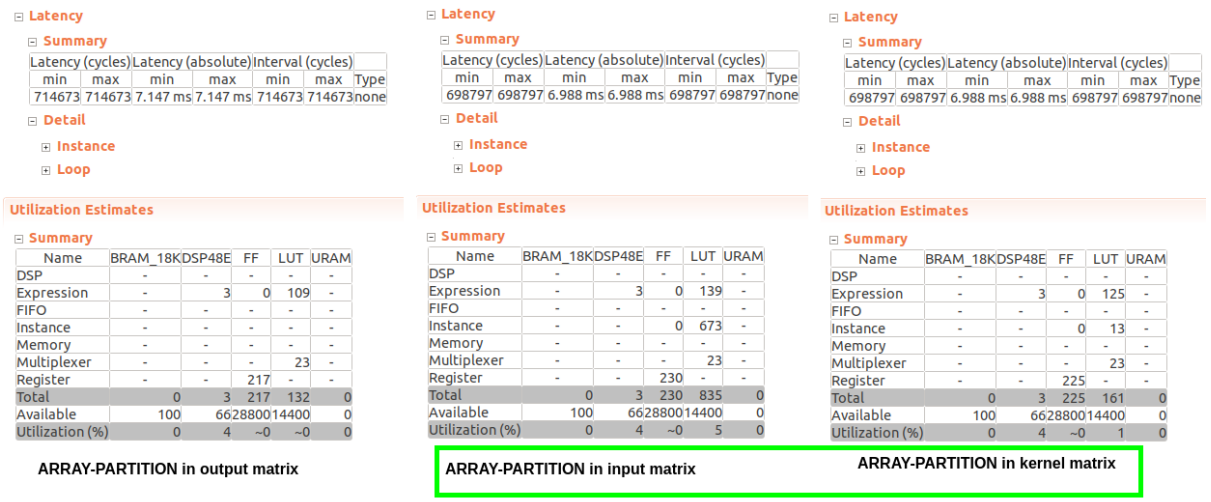
Αρχικά, εφαρμόζουμε την στρατηγική PIPELINE σε κάθε for loop.



Όπως παρατηρούμε από την εικόνα παραπάνω, οι λιγότεροι κύκλοι σε συνδυασμό με το καλύτερο ποσοστό utilization επιτυγχάνονται όταν το pipeline εφαρμοστεί στο δεύτερο for loop. Συμπέρασμα λοιπόν αποτελεί ότι όσο πιο “έξω” σε

επαναλήψεις εφαρμόσουμε το pipeline, τόσο λιγότερους κύκλους έχουμε, αλλά τόσο χειρότερο utilization σε resources. Στην περίπτωση του outer loop μάλιστα τα resources δεν φτάνουν.

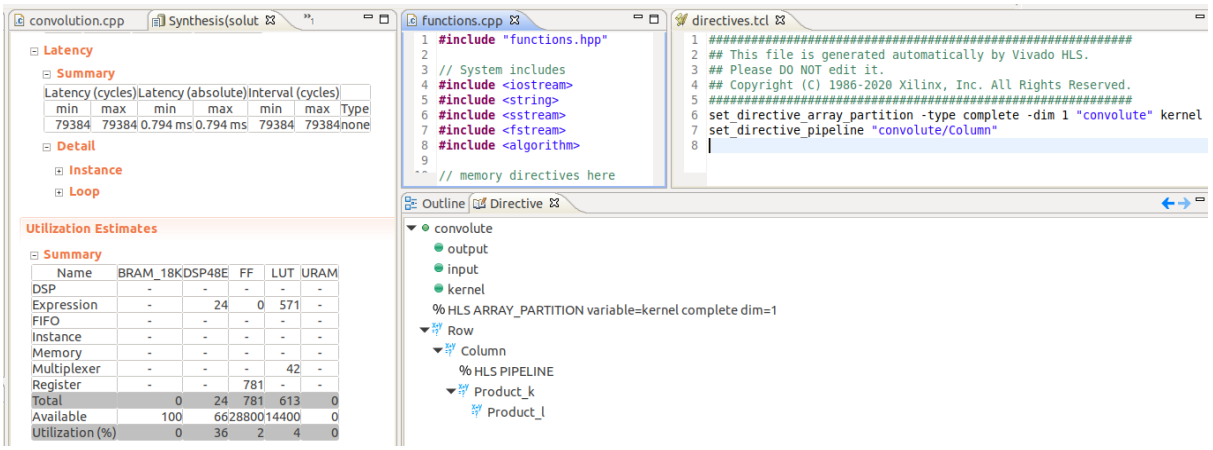
Στη συνέχεια εφαρμόζουμε την στρατηγική ARRAY-PARTITION στην variable output, input και kernel αντίστοιχα.

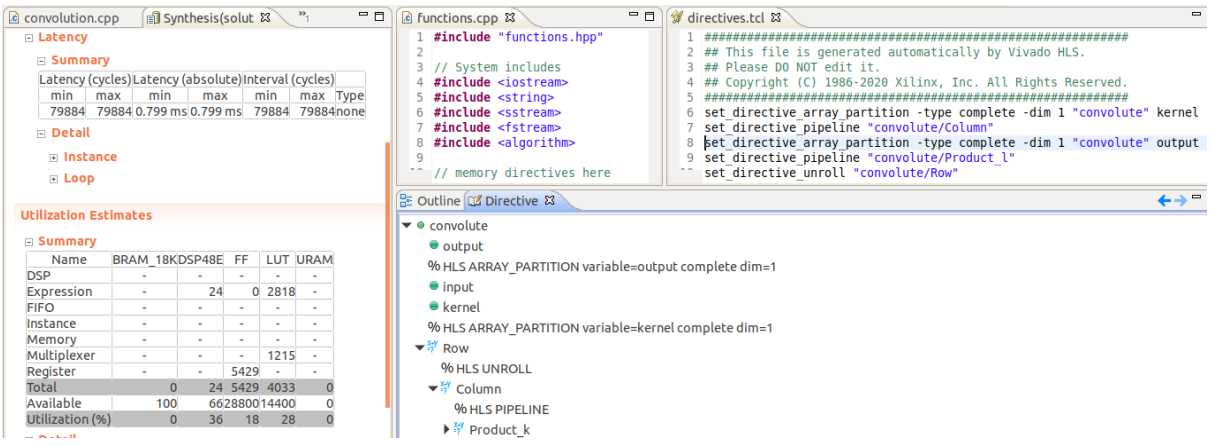


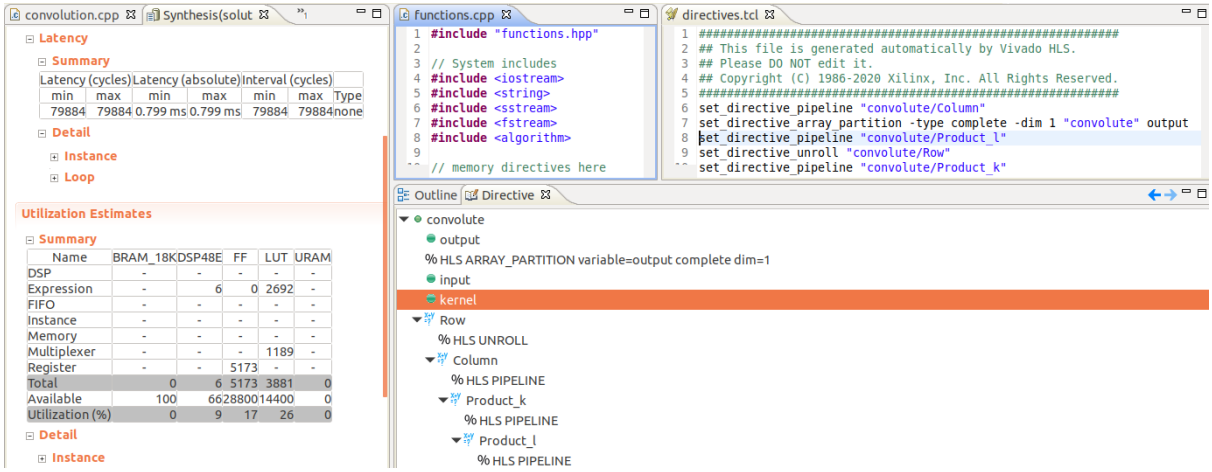
Σύμφωνα με την εικόνα παραπάνω, παρατηρούμε πως τα αποτελέσματα των στρατηγικών στους input και kernel πίνακες δεν διαφέρουν, διότι η πολυπλοκότητά τους στον κώδικα βρίσκεται στον ίδιο σημείο (στην inner-most loop), με καλύτερο latency σε σύγκριση με τον output πίνακα, αλλά παραπάνω resources.

Kernel 3x3 - Method 2

Σας παρουσιάζουμε παρακάτω συνδυαστικές στρατηγικές που δοκιμάσαμε στον κώδικα.

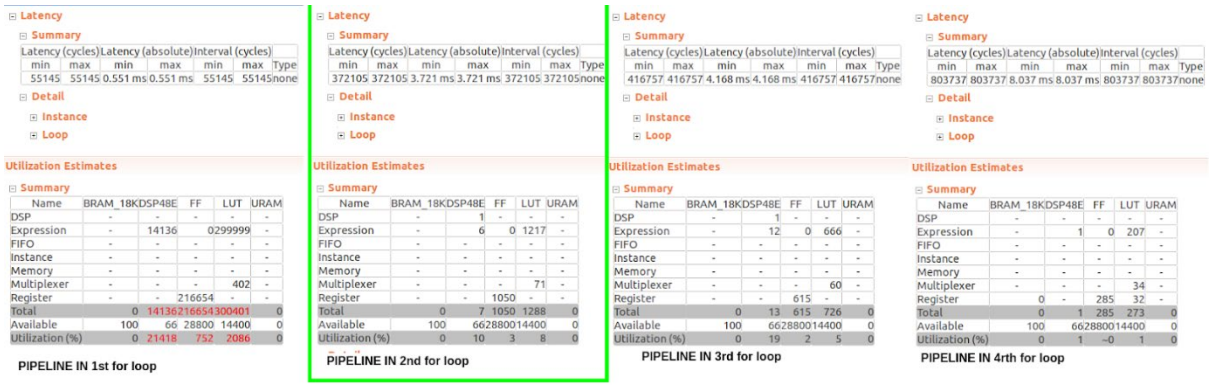






Kernel 7x7 - Method 1

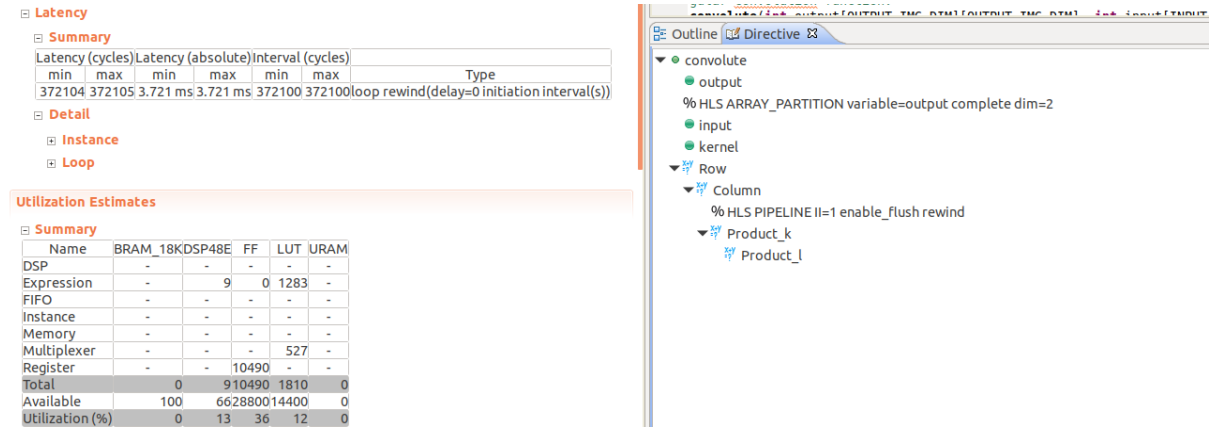
Αρχικά, εφαρμόζουμε την στρατηγική PIPELINE σε κάθε for loop.



Παρατηρούμε ξανά πως δεν μπορούμε να εξοικονομήσουμε resources έχοντας παράλληλα καλό χρόνο, οπότε το καλύτερο αποτέλεσμα φέρνει η PIPELINE στο 2ο εμφωλευμένο for, με τον χρόνο να έχει μειωθεί και οι πόροι να είναι σε ικανοποιητικό επίπεδο.

Kernel 7x7 - Method 2

Συνδυαστικές loop και memory directives με τα αποτελέσματά τους φαίνονται παρακάτω.



Σας παρουσιάζουμε στην παραπάνω εικόνα την εφαρμογή της PIPELINE στην 2η loop και της ARRAY-PARTITION στον output matrix, η οποία δίνει καλό χρόνο και παράληλα καλή εξοικονόμηση πόρων.

ΔΟΚΙΜΕΣ ΣΕ FPGA ΜΕ ΔΙΑΦΟΡΕΤΙΚΑ RESOURCES

Για το τέλος, στα πειράματά μας έπρεπε να συμπεριλάβουμε και δύο επιπλέον FPGA, ένα με λιγότερους πόρους και ένα με περισσότερους πόρους. Το αρχικό μας ήταν εκείνο με ένα Xilinx Zynq-7000 με τα εξής χαρακτηριστικά: {#LUT: 14400, #FF: 28800, #DSP: 66, #BRAM: 50} με speed grade -2.

Το FPGA με τους λιγότερους πόρους που επιλέγουμε είναι το τύπου Xilinx Spartan-7 με τα εξής χαρακτηριστικά: {#LUT: 3750, #FF: 7500, #DSP: 10, #BRAM: 5} με speed grade -1IL.

Το FPGA με τους περισσότερους πόρους που επιλέγουμε είναι το τύπου Xilinx Artix-7 με τα εξής χαρακτηριστικά: {#LUT: 134600, #FF: 269200, #DSP: 740, #BRAM: 365} με speed grade -3.

Για τον kernel 3x3, στην παρακάτω εικόνα συγκρίνουμε το report του C Synthesis για το αρχικό μας FPGA με τα υπόλοιπα δύο. Τα αποτελέσματα φαίνονται κατά σειρά πόρων, από τα λιγότερα προς τα περισσότερα.

Vivado HLS Report Comparison

All Compared Solutions

solution1_kernel3x3_noDirectives_LESS_RESOURCES: xc7s6csga225-1IL
solution1_kernel3x3_noDirectives_INITIAL_RESOURCES: xc7z007s-clg225-2
solution1_kernel3x3_noDirectives_LARGE_RESOURCES: xc7a200tfbg484-3

Performance Estimates

Timing

Clock		solution1_kernel3x3_noDirectives_LESS_RESOURCES	solution1_kernel3x3_noDirectives_INITIAL_RESOURCES	solution1_kernel3x3_noDirectives_LARGE_RESOURCES
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	8.400 ns	7.050 ns	8.539 ns

Latency

		solution1_kernel3x3_noDirectives_LESS_RESOURCES	solution1_kernel3x3_noDirectives_INITIAL_RESOURCES	solution1_kernel3x3_noDirectives_LARGE_RESOURCES
Latency (cycles)	min	698797	698797	555913
	max	698797	698797	555913
Latency (absolute)	min	6.988 ms	6.988 ms	5.559 ms
	max	6.988 ms	6.988 ms	5.559 ms
Interval (cycles)	min	698797	698797	555913
	max	698797	698797	555913

Utilization Estimates

	solution1_kernel3x3_noDirectives_LESS_RESOURCES	solution1_kernel3x3_noDirectives_INITIAL_RESOURCES	solution1_kernel3x3_noDirectives_LARGE_RESOURCES
BRAM_18K	0	0	0
DSP48E	3	3	3
FF	230	230	165
LUT	350	162	347
URAM	0	0	0

Σχετικά με τον χρόνο, παρατηρούμε ότι το αρχικό μας FPGA χρειάζεται λιγότερο χρόνο περιόδου για να προγραμματιστεί, ενώ αυτό με τα περισσότερα resources χρειάζεται περισσότερο κατά 1.5ns και αυτό με τα λιγότερα περισσότερο κατά 1.4ns. Επίσης, οι κύκλοι στο αρχικό και σε αυτό με τα λιγότερα είναι περισσότεροι κατά 100 χιλιάδες περίπου. Τέλος, το FPGA με τα περισσότερα resources χρησιμοποιεί λιγότερα flip flops και τα λιγότερα LUTs τα χρησιμοποιεί το αρχικό FPGA.

Για τον kernel 5x5, στην παρακάτω εικόνα συγκρίνουμε το report του C Synthesis για το αρχικό μας FPGA με τα υπόλοιπα δύο όπως προηγουμένως. Τα αποτελέσματα φαίνονται κατά σειρά πόρων, από τα λιγότερα πρός τα περισσότερα.

Vivado HLS Report Comparison

All Compared Solutions

solution2_kernel5x5_noDirectives_LESS_RESOURCES: xc7s6ftgb196-1IL
solution2_kernel5x5_noDirectives_INITIAL_RESOURCES: xc7z007sclg225-2
solution2_kernel5x5_noDirectives_LARGE_RESOURCES: xc7a200tfbg676-3

Performance Estimates

Timing

Clock		solution2_kernel5x5_noDirectives_LESS_RESOURCES	solution2_kernel5x5_noDirectives_INITIAL_RESOURCES	solution2_kernel5x5_noDirectives_LARGE_RESOURCES
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	8.400 ns	7.050 ns	8.539 ns

Latency

		solution2_kernel5x5_noDirectives_LESS_RESOURCES	solution2_kernel5x5_noDirectives_INITIAL_RESOURCES	solution2_kernel5x5_noDirectives_LARGE_RESOURCES
Latency (cycles)	min	1722361	1722361	1337961
	max	1722361	1722361	1337961
Latency (absolute)	min	17.224 ms	17.224 ms	13.380 ms
	max	17.224 ms	17.224 ms	13.380 ms
Interval (cycles)	min	1722361	1722361	1337961
	max	1722361	1722361	1337961

Utilization Estimates

	solution2_kernel5x5_noDirectives_LESS_RESOURCES	solution2_kernel5x5_noDirectives_INITIAL_RESOURCES	solution2_kernel5x5_noDirectives_LARGE_RESOURCES
BRAM_18K	0	0	0
DSP48E	3	3	3
FF	234	234	169
LUT	356	164	353
URAM	0	0	0

Ισχύουν οι ίδιες παρατηρήσεις.

Για τον kernel 7x7, στην παρακάτω εικόνα συγκρίνουμε το report του C Synthesis για το αρχικό μας FPGA με τα υπόλοιπα δύο όπως προηγουμένως. Τα αποτελέσματα φαίνονται κατά σειρά πόρων, από τα λιγότερα πρός τα περισσότερα.

Vivado HLS Report Comparison

All Compared Solutions

solution3_kernel7x7_noDirectives_LESS_RESOURCES: xc7s6ftgb196-1IL
solution3_kernel7x7_noDirectives_INITIAL_RESOURCES: xc7z007sclg225-2
solution3_kernel7x7_noDirectives_LARGE_RESOURCES: xc7a200tfbg676-3

Performance Estimates

Timing

Clock		solution3_kernel7x7_noDirectives_LESS_RESOURCES	solution3_kernel7x7_noDirectives_INITIAL_RESOURCES	solution3_kernel7x7_noDirectives_LARGE_RESOURCES
ap_clk	Target	10.00 ns	10.00 ns	10.00 ns
	Estimated	8.400 ns	7.050 ns	8.539 ns

Latency

		solution3_kernel7x7_noDirectives_LESS_RESOURCES	solution3_kernel7x7_noDirectives_INITIAL_RESOURCES	solution3_kernel7x7_noDirectives_LARGE_RESOURCES
Latency (cycles)	min	3155653	3155653	2426337
	max	3155653	3155653	2426337
Latency (absolute)	min	31.557 ms	31.557 ms	24.263 ms
	max	31.557 ms	31.557 ms	24.263 ms
Interval (cycles)	min	3155653	3155653	2426337
	max	3155653	3155653	2426337

Utilization Estimates

	solution3_kernel7x7_noDirectives_LESS_RESOURCES	solution3_kernel7x7_noDirectives_INITIAL_RESOURCES	solution3_kernel7x7_noDirectives_LARGE_RESOURCES
BRAM_18K	0	0	0
DSP48E	3	3	3
FF	250	250	185
LUT	363	167	360
URAM	0	0	0

Ισχύουν οι ίδιες παρατηρήσεις.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Μετά από πολλές δοκιμές και συνδυασμούς, αυτό που παρατηρήσαμε είναι πως κάποιες directives παρουσιάζουν περισσότερο impact από κάποιες άλλες. Σίγουρα, ο συνδυασμός κάποιων directives φέρνει καλύτερο latency από άλλους αλλά μπορεί να μη υπάρχει σωστό utilization στα resources. Δηλαδή, είτε θα έχουμε πολύ λίγους κύκλους και θα έχουμε επίδραση στους πόρους, είτε θα έχουμε πολλούς κύκλους και θα στοχεύουμε σε εξοικονόμηση πόρων. Φυσικά στις δοκιμές μας υπήρξαν και τυχαίοι συνδυασμοί, οι οποίοι δεν ήταν αποτελεσματικοί διότι υπήρχε τεράστια αύξηση σε πόρους μέσα σε λίγους κύκλους. Επίσης, στον κώδικά μας, εφαρμόσαμε kernel τύπου int, αλλά προηγουμένως δοκιμάσαμε double, float και char. Διαφορά σε κύκλους είδαμε τη μεγαλύτερη σε τύπου χαρακτήρα, αλλά διατηρήσαμε τύπου ακεραίου για να πιάσουμε τον μέσο όρο. Τέλος, όπως είναι λογικό υπάρχουν χειρότερα και καλύτερα αποτελέσματα σε διαφορετικά FPGA.

Θέμα 2: Υλοποίηση Ενσωματωμένου Συστήματος με χρήση Verilog και Arduino

Ένα (υποτυπώδες) σύστημα συναγερμού βασίζεται σε ένα Ενσωματωμένο Σύστημα (ΕΣ) και χρησιμοποιείται για την προστασία ενός μικρού χώρου. Το ΕΣ έχει τέσσερις εισόδους (Motion1, Motion2, Reed και Code) και δύο εξόδους (Active, Alarm).

- *Οι είσοδοι Motion1 και Motion2 είναι συνδεδεμένες σε δύο αισθητήρες κίνησης εντός του χώρου. Όταν ανιχνευτεί κίνηση ενεργοποιούνται, δηλαδή δίνουν έξοδο 1.*
- *Η είσοδος Reed είναι συνδεδεμένη σε αισθητήρα τύπου Reed Switch στην πόρτα εισόδου. Όταν ανιχνευτεί άνοιγμα της πόρτας ενεργοποιείται, δηλαδή δίνει έξοδο 1.*
- *Η είσοδος Code είναι 5 bit και οδηγείται από ένα υποτυπώδες πληκτρολόγιο με 32 πλήκτρα, αριθμημένα από το 0 μέχρι το 30 και 1 πλήκτρο Arm. Ανάλογα με το τί πλήκτρο πατάει ο χρήστης, ο αντίστοιχος δυαδικός αριθμός εγγράφεται στα 5 bit του σήματος Code ή 31 στην περίπτωση του Arm.*
- *Η έξοδος Active δείχνει πως ο συναγερμός έχει ενεργοποιηθεί*
- *Η έξοδος Alarm είναι συνδεδεμένη σε μια σειρήνα και ενεργοποιείται όταν ανιχνευτεί κίνηση ή άνοιγμα της πόρτας, ενώ ο συναγερμός είναι ενεργός*

Κάθε φορά που ο χρήστης πατάει το πλήκτρο Arm, ο συναγερμός οπλίζει και ενεργοποιεί την έξοδο Active. Ο ενεργός συναγερμός μπορεί να αφοπλιστεί μόνο με το πλήκτρο με τον αριθμό 4. Όσο είναι οπλισμένος ο συναγερμός, αν ανιχνευτεί κίνηση ή άνοιγμα της πόρτας, τότε ενεργοποιείται η σειρήνα (σήμα Alarm) μέχρι ο χρήστης να τον αφοπλίσει μέσω του πλήκτρου 4. Σε κάθε άλλη περίπτωση, η σειρήνα δεν ενεργοποιείται και η έξοδος Active είναι ανενεργή

A. Υλοποιήστε και επαληθεύστε ένα module σε Verilog το οποίο να υλοποιεί τις ανωτέρω προδιαγραφές

Στο παρόν θέμα καλούμαστε να υλοποιήσουμε ένα σύστημα συναγερμού σε Verilog. Η εργασία μας αναπτύχθηκε και δοκιμάστηκε στο περιβάλλον του edaplayground, την οποία μπορείτε και να δείτε να εκτελείται σε αυτόν τον σύνδεσμο. Παρακάτω θα πραγματοποιηθεί εξήγηση των παραδοχών που πήραμε, του κώδικα καθώς και τα παραδείγματα εκτέλεσης. Ακολουθούν επίσης screenshots.

ΠΑΡΑΔΟΧΕΣ

Στο πρόγραμμά μας συμπεριλαμβάνουμε το σενάριο όπου ο συναγερμός ενεργοποιείται με το πάτημα του πλήκτρου 31. Αν ο συναγερμός είναι ενεργός και ανιχνεύσει κίνηση, είτε από τους σένσορες ανίχνευσης κίνησης στον χώρο, είτε από τον διακόπτη των παραθύρων, τότε θα ηχήσει η σειρήνα. Η σειρήνα δεν θα πάψει να ηχεί, παρά μόνο αν ο χρήστης πατήσει το πλήκτρο 4. Επίσης, η σειρήνα δεν θα ανάψει αν ο συναγερμός είναι ανενεργός και πατιούνται πλήκτρα διαφορετικά του 31. Επομένως, ο γενικός κανόνας είναι πως η κατάσταση του συναγερμού δεν θα αλλάξει αν πληκτρολογεί ο χρήστης πλήκτρα διαφορετικά του 31 ή του 4. Τέλος, αν πατηθεί πλήκτρο εκτός του εύρους [0,31] τότε εμφανίζεται στην κονσόλα μήνυμα «INVALID passcode» και η κατάσταση του συναγερμού δεν αλλάζει.

ΚΩΔΙΚΑΣ VERILOG

Το πρόγραμμά μας αποτελείται από δύο modules τα οποία υλοποιούν ξεχωριστές λειτουργίες και από το stimulus module.

Module Passcodes_Converter

```
module Passcodes_Converter(
    input  [31:0] in, //inputs
    output reg [4:0] passcode //5-bits
    );

    always @(in)
    begin
        $display("Entered passcode: %0d", in);
        case (in)
            0:  passcode = 5'b00000;
            1:  passcode = 5'b00001;
            2:  passcode = 5'b00010;
            3:  passcode = 5'b00011;
            4:  passcode = 5'b00100;
            5:  passcode = 5'b00101;
            6:  passcode = 5'b00110;
            7:  passcode = 5'b00111;
            8:  passcode = 5'b01000;
            9:  passcode = 5'b01001;
            10: passcode = 5'b01010;
            11: passcode = 5'b01011;
            12: passcode = 5'b01100;
            13: passcode = 5'b01101;
            14: passcode = 5'b01110;
            15: passcode = 5'b01111;
            16: passcode = 5'b10000;
            17: passcode = 5'b10001;
            18: passcode = 5'b10010;
            19: passcode = 5'b10011;
            20: passcode = 5'b10100;
            21: passcode = 5'b10101;
            22: passcode = 5'b10110;
            23: passcode = 5'b10111;
            24: passcode = 5'b11000;
            25: passcode = 5'b11001;
            26: passcode = 5'b11010;
            27: passcode = 5'b11011;
            28: passcode = 5'b11100;
            29: passcode = 5'b11101;
            30: passcode = 5'b11110;
            31: passcode = 5'b11111;
            default: $display("INVALID passcode");
        endcase
      //$display("passcode= %b", passcode);
    end

endmodule
```

Το module *Passcodes_Converter* μετατρέπει το δεκαδικό input του χρήστη σε δυαδικό των 5 bit. Το επιτρεπτό εύρος είναι [0,31] και οτιδήποτε διαφορετικό από αυτό εμφανίζει το μήνυμα invalid passcode.

Γραμμές 1 – 4 :
Τα ορίσματα του module είναι:

Input [31:0] in: Το input από το πληκτρολόγιο του χρήστη με εύρος από το 0 ως το 31. Είναι wire μεταβλητή διότι είναι input.

Output reg [4:0] passcode: Το output είναι τύπου register μεταβλητή γιατί θέλουμε να βάζουμε εμείς τιμές και βγάζει στην έξοδο τα 5 bit.

Γραμμές 6 – 44:
Το always block εκτελείται σε loop με event expression την μεταβλητή in. Μέσα στο always εφαρμόζεται ένα case για τα διαφορετικά inputs και μετατρέπεται το δεκαδικό input σε δυαδικό 5 bit. Το αποτέλεσμα αποθηκεύεται στην μεταβλητή passcode η οποία είναι το output. Για λόγους debugging στην γραμμή 43 είναι σχολιασμένο το display που εκτυπώνει στην κονσόλα τον κωδικό των 5 bit.

Module Home_Alarm_System

```
1   module Home_Alarm_System(
2       input wire motion1,
3       input wire motion2,
4       input wire reed,
5       input wire [4:0] code,
6     output reg alarm,
7     output reg active
8       );
9
10    always @(posedge motion1 or posedge motion2 or posedge reed or code
11  or active or alarm)
12      begin
13        // arm system with code 31
14        if(code == 5'b11111) begin
15          active = 1'b1; //active = 1
16          alarm = (motion1 | motion2 | reed) & active;
17        end
18        // disarm system with code 4
19        else if(code == 5'b00100) begin
20          active = 1'b0; // active = 0
21          alarm = 1'b0; //alarm = 0
22        end
23        else begin
24          alarm = (motion1 | motion2 | reed) & active;
25        end
26
27        if(alarm == 1'b1) begin
28          //$display("=======>> !!! Siren !!! <<=======");
29        end
30      end
31
32  endmodule
```

Το module *Home_Alarm_System* υλοποιεί όλη τη λογική του συστήματος.

Γραμμές 1 – 8:

Τα ορίσματα του module είναι:

input wire motion1: Η μεταβλητή τύπου wire διότι είναι input η οποία υποδηλώνει με 1 αν ανιχνευτεί κίνηση στον χώρο και με 0 αν δεν ανιχνευτεί.

input wire motion2: Όμοια με το motion1, για την ανίχνευση κίνησης με τα bit 1 και 0.

input wire reed: Όμοια με τα motion1 και motion2, για την ανίχνευση κίνησης των παραθύρων. Με τιμές 1 και 0.

input wire [4:0] code: Μεταβλητή 4-ων bit που περιέχει την μετατροπή του πληκτρολογίου σε δυαδικών των 5 bit.

output reg alarm: Μεταβλητή τύπου output γιατί θέλουμε να είναι έξοδος και reg γιατί θέλουμε εμείς να βάζουμε τιμές. Αλλάζει η τιμή του alarm σε 0 ή 1 ανάλογα αν ανιχνεύτηκε κίνηση ενώ ο συναγερμός είναι ενεργός.

output reg active: Μεταβλητή τύπου output γιατί θέλουμε να είναι έξοδος και reg γιατί θέλουμε εμείς να βάζουμε τιμές. Όμοια με το alarm, παίρνει την τιμή 1 ή 0, ανάλογα τον κωδικό που θα δοθεί.

Γραμμές 10 – 11:

Σε αυτό το σημείο, το always block θα εκτελείται συνέχεια, κάθε φορά που αλλάζουν τιμές τα motion1, motion2, reed, code, active ή alarm. Πιο συγκεκριμένα, εκτελείται σε κάθε θετικό μέτωπο του παλμού του motion1, motion2, reed (δηλαδή κάθε φορά που από 0 γίνεται 1).

Γραμμές 14 – 25:

Η λειτουργία του if είναι η εξής:

Αν δοθεί ο κωδικός 31 ή αλλιώς 11111, τότε ο συναγερμός θα ενεργοποιηθεί και η τιμή του active θα γίνει 1, και το alarm θα πάρει την τιμή του αποτελέσματος or(motion1,motion2,reed) and active.

Αν δοθεί ο κωδικός 4 ή αλλιώς 00100, τότε θεωρούμε ότι αφοπλίζεται ο συναγερμός και κλείνει η σειρήνα. Δηλαδή οι τιμές του active και alarm γίνονται 0.

Σε οποιονδήποτε άλλον κωδικό, απλά θα εφαρμοστεί η έκφραση για το alarm, με συνέπεια να μην αλλάξει η κατάσταση του συναγερμού.

Γραμμές 27 – 29:

Αυτό το σημείο του κώδικα υπάρχει για λόγους debugging και είναι σχολιασμένο για την αποφυγή πολλών περιττών μηνυμάτων στην κονσόλα. Στην ουσία, αν η μεταβλητή alarm πάρε την τιμή 1, εκτυπώνεται στην κονσόλα το "=======>> !!! Siren !!! <<=======" για να διακρίνουμε εύκολα ότι «χτυπάει» ο συναγερμός.

Module Top_Module – Stimulus Module

```
1   module Top_Module();
2       reg Motion1, Motion2, Reed;
3       integer keyboard;
4       output wire [4:0] Code;
5       output wire Alarm;
6       output wire Active;
7
8       //instantiating the two modules
9       Passcodes_Converter P1(keyboard,Code);
10      Home_Alarm_System S1(Motion1, Motion2, Reed, Code, Alarm, Active);
```

Στο stimulus module υπάρχουν τα εξής ορίσματα:

Γραμμές 2 – 6:

reg Motion1, Motion2, Reed: Οι μεταβλητές ανίχνευσης κίνησης του ενός bit.
integer keyboard: Μεταβλητή τύπου integer για την είσοδο από το πληκτρολόγιο του χρήστη.
output wire [4:0] Code: Μεταβλητή που δέχεται το 5-μπιτο κωδικό.
output wire Alarm: Μεταβλητή για το αν ηχεί η σειρήνα του συναγερμού ή όχι με τιμές 0 ή 1.
output wire Active: Μεταβλητή για το αν είναι ενεργοποιημένος ο συναγερμός με τιμές 0 ή 1.

Γραμμές 8 – 10:

Το κάλεσμα των modules που αναφέραμε προηγουμένως, με τις εισόδους και εξόδους που περιεγράφηκαν.

```
11  initial
12      begin
13          $dumpfile("test.vcd");
14          $dumpvars(0);
5           $monitor("\t\ttime=%0tns\tMotion1=%b   Motion2=%b   Reed=%b   C
16  ode=%2d   Active=%b   Alarm=%b", $time, Motion1, Motion2, Reed,
17  keyboard, Active, Alarm);
18      end
```

Γραμμές 11 – 18:

Το πρώτο από τα δύο initial blocks, στο οποίο γίνεται τα dumpfile, dumpvars, monitor. Με τα dumpfile και dumpvars μας επιτρέπεται να γράψουμε αυτόματα όσες πληροφορίες χρειάζονται για να δημιουργηθεί το διάγραμμα με το 0 και 1 που παρουσιάζουμε παρακάτω. Το dumpvars δέχεται την τιμή 0 ώστε να παρακολουθούμε όλες τις μεταβλητές με τις αλλαγές τους.
Η monitor μας επιτρέπει να εκτυπώνουμε στην κονσόλα ένα μήνυμα κάθε φορά που πραγματοποιείται μια αλλαγή στις μεταβλητές που της δίνονται. Μπορεί να υπάρξει μόνο μια monitor στο πρόγραμμα.

```
19  initial
20      begin
21          keyboard = 31;
22          {Motion1, Motion2, Reed} = 3'b000;
23          #5 {Motion1, Motion2, Reed} = 3'b001;
24          #5 {Motion1, Motion2, Reed} = 3'b010;
25          #5 {Motion1, Motion2, Reed} = 3'b011;
26          #5 {Motion1, Motion2, Reed} = 3'b100;
27          #5 {Motion1, Motion2, Reed} = 3'b101;
28          #5 {Motion1, Motion2, Reed} = 3'b110;
29          #5 {Motion1, Motion2, Reed} = 3'b111;
```

Γραμμές 19 – 29:

Το δεύτερο και τελευταίο initial block, το οποίο εκτελείται σε παραλληλία με το πρώτο, στο οποίο υλοποιούνται διαφορετικά σενάρια προσομοίωσης του συναγερμού.
Το πρώτο σενάριο προσπαθούμε να ενεργοποιήσουμε τον συναγερμό και να δούμε τη συμπεριφπρά των μεταβλητών για όλες τις δυνατές περιπτώσεις ανίχνευσης κίνησης.

```
30          #5 keyboard = 4;
31          {Motion1, Motion2, Reed} = 3'b000;
32          #5 {Motion1, Motion2, Reed} = 3'b001;
33          #5 {Motion1, Motion2, Reed} = 3'b010;
34          #5 {Motion1, Motion2, Reed} = 3'b011;
35          #5 {Motion1, Motion2, Reed} = 3'b100;
36          #5 {Motion1, Motion2, Reed} = 3'b101;
37          #5 {Motion1, Motion2, Reed} = 3'b110;
38          #5 {Motion1, Motion2, Reed} = 3'b111;
```

Γραμμές 30 – 38:
Δεύτερο σενάριο υλοποίησης, κατά το οποίο δίνουμε τον κωδικό απενεργοποίησης 4 και παρατηρούμε την συμπεριφορά των μεταβλητών για όλα τα σενάρια ανίχνευσης κίνησης.

```
39          #5 keyboard = 19;
40          {Motion1, Motion2, Reed} = 3'b000;
41          #5 {Motion1, Motion2, Reed} = 3'b001;
42          #5 {Motion1, Motion2, Reed} = 3'b010;
43          #5 {Motion1, Motion2, Reed} = 3'b011;
44          #5 {Motion1, Motion2, Reed} = 3'b100;
45          #5 {Motion1, Motion2, Reed} = 3'b101;
46          #5 {Motion1, Motion2, Reed} = 3'b110;
47          #5 {Motion1, Motion2, Reed} = 3'b111;
```

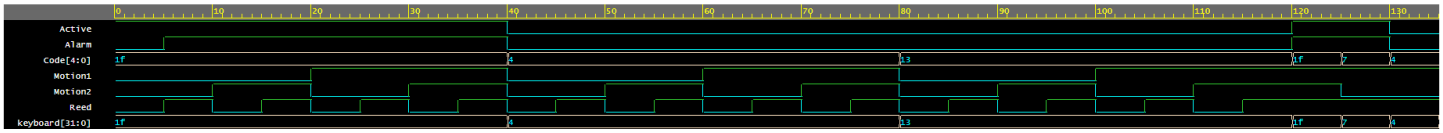
Γραμμές 39 – 47:
Τρίτο σενάριο υλοποίησης, κατά το οποίο δίνουμε έναν μεν valid κωδικό αλλά χωρίς ρόλο και όλα τα σενάρια ανίχνευσης κίνησης ώστε να επαληθεύσουμε ότι η κατάσταση του συναγερμού δεν αλλάζει.

```
48          #5 keyboard = 31;
49          {Motion1, Motion2, Reed} = 3'b111;
50
51          #5 keyboard = 7;
52          {Motion1, Motion2, Reed} = 3'b101;
53
54          #5 keyboard = 4;
55          {Motion1, Motion2, Reed} = 3'b101;
56
57          #5 keyboard = 44;
58      end
59
60  endmodule
```

Γραμμές 48 – 60:
Τέταρτο, πέμπτο, έκτο και έβδομο σενάρια υλοποίησης, κατά τα οποία μπορούμε να δούμε την συμπεριφορά του συναγερμού αν ενεργοποιηθεί και ανιχνεύσει κίνηση, μετά αν δοθεί άσχετος αριθμός, αν απενεργοποιηθεί και αν δοθεί invalid κωδικός.

SCREENSHOTS

Το διάγραμμα από το EPWave μετά από την εκτέλεση της υλοποίησής μας είναι το παρακάτω:



Το output της κονσόλας είναι το παρακάτω:

Log | Share

```
[2022-06-01 14:24:20 EDT] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile test.vcd opened for output.
Entered passcode: 31
                time=0ns        Motion1=0   Motion2=0   Reed=0   Code=31   Active=1   Alarm=0
                time=5ns        Motion1=0   Motion2=0   Reed=1   Code=31   Active=1   Alarm=1
                time=10ns       Motion1=0   Motion2=1   Reed=0   Code=31   Active=1   Alarm=1
                time=15ns       Motion1=0   Motion2=1   Reed=1   Code=31   Active=1   Alarm=1
                time=20ns       Motion1=1   Motion2=0   Reed=0   Code=31   Active=1   Alarm=1
                time=25ns       Motion1=1   Motion2=0   Reed=1   Code=31   Active=1   Alarm=1
                time=30ns       Motion1=1   Motion2=1   Reed=0   Code=31   Active=1   Alarm=1
                time=35ns       Motion1=1   Motion2=1   Reed=1   Code=31   Active=1   Alarm=1
Entered passcode: 4
                time=40ns       Motion1=0   Motion2=0   Reed=0   Code= 4   Active=0   Alarm=0
                time=45ns       Motion1=0   Motion2=0   Reed=1   Code= 4   Active=0   Alarm=0
                time=50ns       Motion1=0   Motion2=1   Reed=0   Code= 4   Active=0   Alarm=0
                time=55ns       Motion1=0   Motion2=1   Reed=1   Code= 4   Active=0   Alarm=0
                time=60ns       Motion1=1   Motion2=0   Reed=0   Code= 4   Active=0   Alarm=0
                time=65ns       Motion1=1   Motion2=0   Reed=1   Code= 4   Active=0   Alarm=0
                time=70ns       Motion1=1   Motion2=1   Reed=0   Code= 4   Active=0   Alarm=0
                time=75ns       Motion1=1   Motion2=1   Reed=1   Code= 4   Active=0   Alarm=0
Entered passcode: 19
                time=80ns       Motion1=0   Motion2=0   Reed=0   Code=19   Active=0   Alarm=0
                time=85ns       Motion1=0   Motion2=0   Reed=1   Code=19   Active=0   Alarm=0
                time=90ns       Motion1=0   Motion2=1   Reed=0   Code=19   Active=0   Alarm=0
                time=95ns       Motion1=0   Motion2=1   Reed=1   Code=19   Active=0   Alarm=0
                time=100ns      Motion1=1   Motion2=0   Reed=0   Code=19   Active=0   Alarm=0
                time=105ns      Motion1=1   Motion2=0   Reed=1   Code=19   Active=0   Alarm=0
                time=110ns      Motion1=1   Motion2=1   Reed=0   Code=19   Active=0   Alarm=0
                time=115ns      Motion1=1   Motion2=1   Reed=1   Code=19   Active=0   Alarm=0
Entered passcode: 31
                time=120ns      Motion1=1   Motion2=1   Reed=1   Code=31   Active=1   Alarm=1
Entered passcode: 7
                time=125ns      Motion1=1   Motion2=0   Reed=1   Code= 7   Active=1   Alarm=1
Entered passcode: 4
                time=130ns      Motion1=1   Motion2=0   Reed=1   Code= 4   Active=0   Alarm=0
Entered passcode: 44
INVALID passcode
                time=135ns      Motion1=1   Motion2=0   Reed=1   Code=44   Active=0   Alarm=0
Finding VCD file...
./test.vcd
[2022-06-01 14:24:20 EDT] Opening EPWave...
Done
```

Στο output επαληθεύουμε πως η κατάσταση του συναγερμού δεν αλλάζει αν δοθεί λάθος κωδικός. Η σειρήνα θα ηχεί για όσο ο χρήστης δεν δίνει τον αριθμό 4 της απενεργοποίησης.

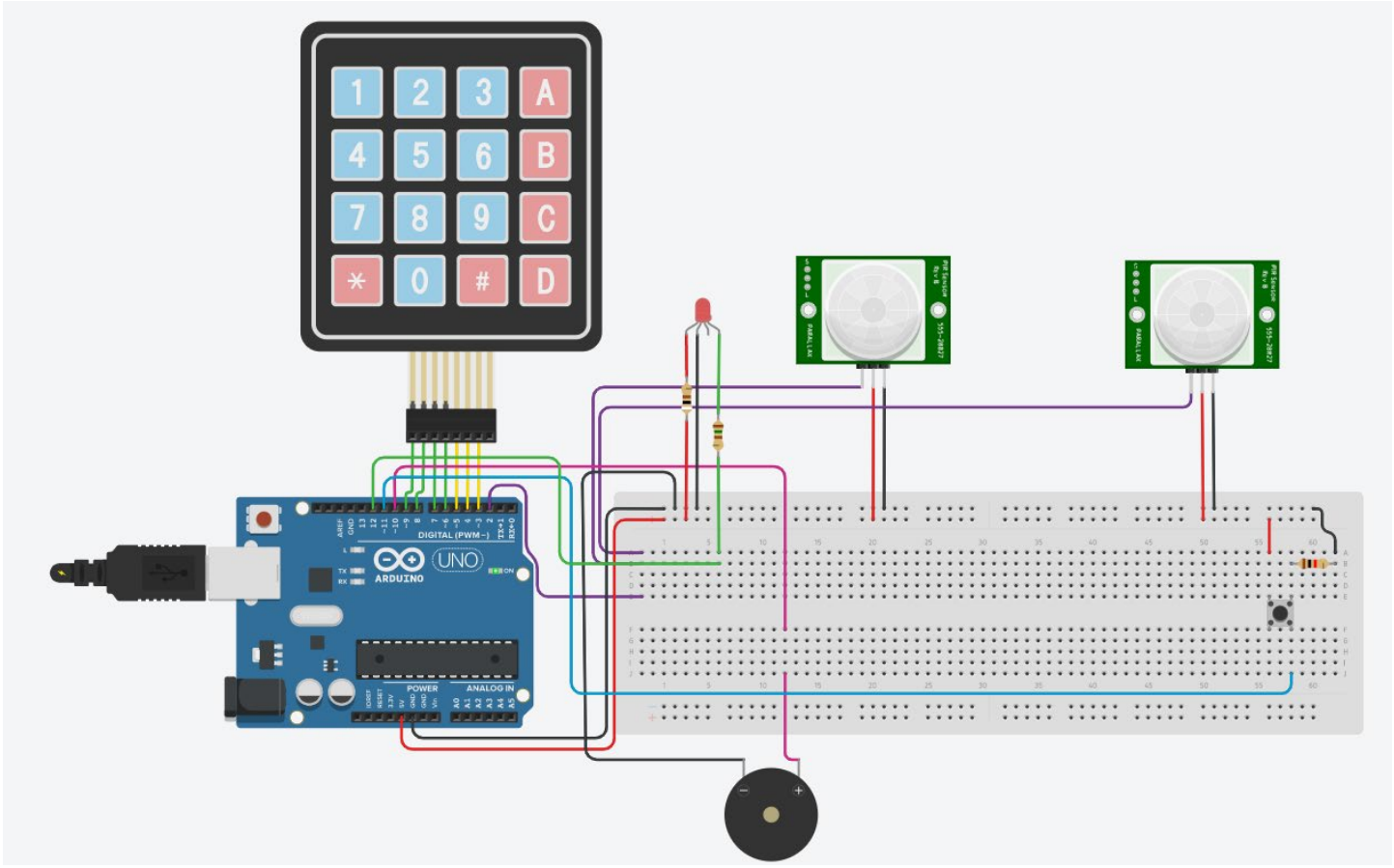
Β. Υλοποιήστε σύστημα σε Arduino το οποίο να υλοποιεί τις ανωτέρω προδιαγραφές.

Η υλοποίηση του προαναφερθέντος συστήματος σε Arduino υλοποιήθηκε στην πλατφόρμα tinkercad.

ΠΑΡΑΔΟΧΕΣ

Στην παρούσα υλοποίηση της εργασίας, θεωρούμε δύο PIR Sensors ως τα motion1 και motion2 και ένα push button ως το reed. Αυτό σημαίνει ότι αν ανιχνευτεί κίνηση στο εύρος των PIR Sensors ή αν πατηθεί το push button, τότε θα ακουστεί ήχος από ένα buzzer module (piezo). Υποθέτουμε πως ο συναγερμός θεωρείται απενεργοποιημένος και αυτό θα φανεί με ένα LED RGB το οποίο θα είναι κόκκινο όσο ο συναγερμός είναι απενεργοποιημένος και πράσινο όταν ενεργοποιηθεί. Επίσης, χρησιμοποιούμε το keypad 4x4, το οποίο παρόλο που περιέχει κλειδιά διαφορετικά των ζητούμενων, αυτά δεν ενεργοποιούνται, ούτε λαμβάνονται υπόψιν. Οι εκτυπώσεις που θα γράφονται στην κονσόλα είναι για λόγους debugging.

ΣΥΝΔΕΣΜΟΛΟΓΙΑ



Στο screenshot που σας παρουσιάζουμε παραπάνω, έχουμε τα διαφορετικά components. Αρχικά, χρησιμοποιούμε το Arduino, την υπολογιστική μονάδα που τα κάνουμε όλα, το breadboard στο οποίο συνδέονται τα καλώδια, τους ανιχνευτές κίνησης, το pushbutton που θεωρούμε ως reed, δηλαδή όταν πατηθεί το pushbutton έχουμε κίνηση στο παράθυρο, το ηχείο (piezo) που κάνει θόρυβο, το πληκτρολόγιο, τις αντιστάσεις και το LED που μας υποδεικνύει αν λειτουργεί ο συναγερμός ή όχι. Το Arduino έχει υποδοχές, ή αλλιώς pins, τα οποία συνδέονται με τα καλώδια.

Στην κάτω σειρά των υποδοχών, συνδέουμε το ρεύμα των 5V με το breadboard και αυτόματα αυτή η σειρά αποκτά ρεύμα με 5V. Ομοίως, συνδέουμε το GND του Arduino με την πάνω γραμμή του breadboard, όπου και δημιουργείται η γείωση σε αυτή την γραμμή.

Οπότε, για τους PIR Sensors, που έχουν τρία pins, συνδέουμε αυτό που αντιστοιχεί στο Ground με τη γραμμή του GRD, αυτό που αντιστοιχεί στο Power με τη γραμμή των 5V. Δεν συνδέουμε το Power pin των PIR Sensor κατευθείαν με το Arduino γιατί μας ενδιαφέρει να είναι συνέχεια σε λειτουργία. Θα μπορούσαμε να τους είχαμε συνδέσει στο Arduino για εξοικονόμηση ρεύματος, αλλά θεωρήσαμε ως παραδοχή να είναι συνέχεια σε

λειτουργία. Το 3ο pin των PIR Sensors είναι το Signal, το οποίο αν ανιχνεύσει κίνηση θα δώσει ρεύμα, αλλιώς δεν δίνει ρεύμα.

Επιπλέον, επειδή μας ενδιαφέρει έστω ένα σένσορας να ανιχνεύσει κίνηση, τα Signal pins συνδέονται με την 1η γραμμή του breadboard, οπότε είτε αν δώσει το 1ο είτε το 2ο pin ρεύμα, τότε ένα καλώδιο που υπάρχει στην ίδια στήλη (γραμμή E) θα πάρει ρεύμα κι θα το πάει στο Arduino στο pin 2.

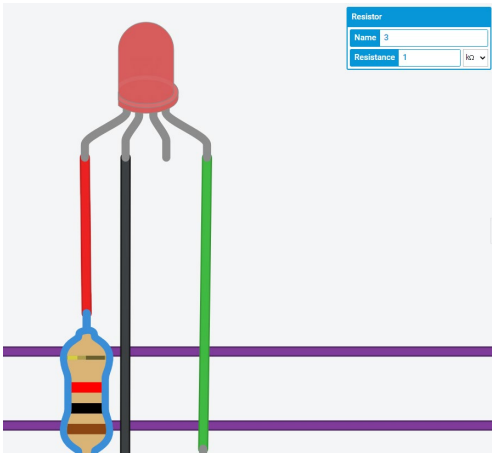
Αντίστοιχα, για το component του keypad, έχει 4 pins για τις γραμμές και 4 pins για τις στήλες. Καθώς δεν υπάρχει άλλο keypad που να εξυπηρετεί τον σκοπό της εργασίας μας, εργαστήκαμε με το συγκεκριμένο 4x4, με τη λειτουργία του να ενεργοποιείται μόνο στα αριθμητικά πλήκτρα οπότε και η4η στήλη δεν συνδέεται καν με το Arduino καθώς περιέχει σύμβολα. Συνδέουμε τα rows με τα pins του Arduino 9,8,7,6 και τα columns με τα pins -5, 4 και 3.

Το επόμενο component είναι το push button, του οποίου η αρχιτεκτονική είναι ακριβώς ίδια. Δηλαδή, του δίνω ρεύμα από τη μία και δημιουργώ γείωση από την άλλη. Όταν το πατήσω θα δώσει ρεύμα στο γαλανό καλώδιο, το οποίο μπαίνει στο pin 11.

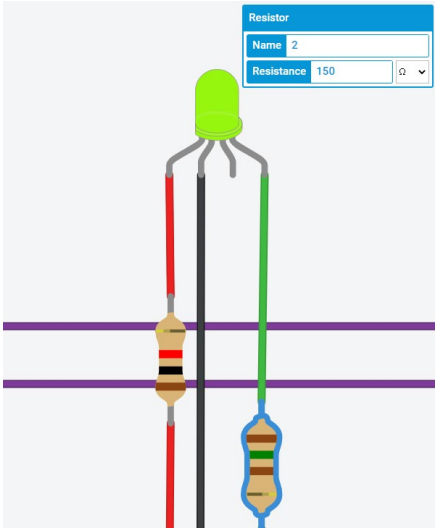
Στη συνέχεια, για το Piezo, συνδέουμε το negative pin με τη γείωση και το positive pin με το pin 10 του Arduino.

Τέλος, έχουμε ένα LED RGB για την ένδειξη αν ο συναγερμός μας είναι ενεργοποιημένος ή όχι. Συνδέουμε το red pin με το ρεύμα, το cathode pin με τη γείωση καθώς και το green pin με το pin 12 του Arduino, το οποίο δίνει ρεύμα αν λειτουργεί ο συναγερμός.

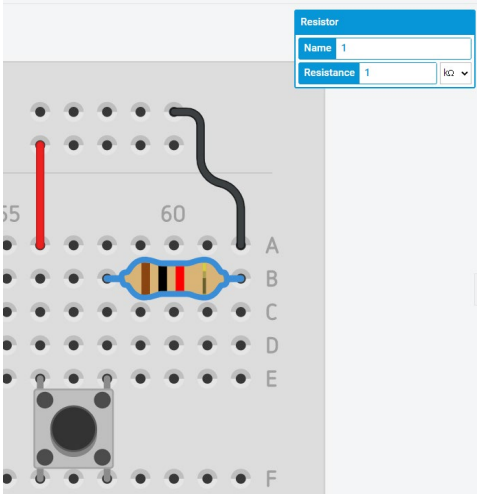
Με την έναρξη του simulation, ο συναγερμός θεωρείται απενεργοποιημένος, οπότε το λαμπάκι ανάβει κόκκινο χρώμα. Αυτό επιτυγχάνεται με έναν αντιστάτη στο red pin του led με την default τιμή στο 1kΩ.



Αντίστοιχα, αν πατηθεί ο σωστός κωδικός, το 31, τότε το λαμπάκι θα ανάψει πράσινο, και αυτό το επιτύχαμε με έναν αντιστάτη στο green pin αντίστασης 150Ω.



Ένας ακόμα αντιστάτης χρησιμοποιήθηκε για το push button με την default τιμή του 1kΩ.



ΚΩΔΙΚΑΣ

Σας παρουσιάζουμε τον κώδικα καθώς και την εξήγησή του.

```
1   #include <Keypad.h>
2
3   const int ROW_NUM = 4; //four rows
4   const int COLUMN_NUM = 4; //four columns
5
6   char keys[ROW_NUM][COLUMN_NUM] = {
7     {'1','2','3', 'A'},
8     {'4','5','6', 'B'},
9     {'7','8','9', 'C'},
10    {'*','0','#', 'D'}
11  };
12
13  byte pin_rows[ROW_NUM] = {9, 8, 7, 6}; //connect to the row pinouts of
14  the keypad
15  byte pin_column[COLUMN_NUM] = {5, 4, 3}; //connect to the column
16  pinouts of the keypad
17
18  Keypad keypad = Keypad( makeKeymap(keys), pin_rows, pin_column,
19  ROW_NUM, COLUMN_NUM );
```

Γραμμή 1:
Η δήλωση της βιβλιοθήκης keypad, η οποία είναι απαραίτητη για matrix style keypads με το Arduino.

Γραμμές 3 – 4:
Δήλωση σταθερών μεταβλητών για τις γραμμές και στήλες του πληκτρολογίου, το οποίο είναι ένα έτοιμο 4x4.

Γραμμές 6 – 11:
Η ανάθεση των τιμών των κλειδιών του πληκτρολογίου που φαίνονται και οπτικά.

Γραμμές 13 – 19:
Πού έχουμε συνδέσει το κάθε pin του πληκτρολογίου με τα pin του Arduino.

Ο παραπάνω κώδικας αναφέρεται επίσης σε αυτήν την ιστοσελίδα.

```
20  int motionPin = 2;
21  int soundPin = 10;
22  int indicatorPin = 12;
23  int reedPin = 11;
24
25  bool arm = 0;
26  char previous = '0';
```

Γραμμές 20 – 26:
Ο ορισμός των pins. Στο 2 βάζουμε την κίνηση των sensors, στο 10 βάζουμε τον ήχο, στο 12 βάζουμε το λαμπάκι και στο 11 το push button (reed). Επίσης θέτουμε μια μεταβλητή Boolean που είναι 0 ή 1, αν ο συναγερμός είναι απενεργοποιημένος ή ενεργός αντίστοιχα, και μια μεταβλητή previous, η οποία κρατάει το κουμπί που πατήθηκε. Εμάς μας ενδιαφέρει μόνο αν πατήθηκε το 3 στην περίπτωση του previous, οπότε κρατάμε μόνο το 3. Σε οποιαδήποτε άλλη περίπτωση, το previous είναι παίρνει την τιμή 0.

```
27  void setup(){
28    pinMode(motionPin,INPUT);
29    pinMode(reedPin,INPUT);
30
31    pinMode(soundPin,OUTPUT);
32    pinMode(indicatorPin,OUTPUT);
33
34    Serial.begin(9600);
35  }
```

Γραμμές 27 – 35:
Σε αυτό το κομμάτι κώδικα καλούμε την setup function. Η setup καλείται όταν ξεκινάει το sketch. Χρησιμοποιείται για να αρχικοποιήσει μεταβλητές και pin modes. Η setup θα τρέξει μόνο μία φορά, μετά από κάθε powerup ή για να γίνει reset. Εμείς ρυθμίζουμε τα motionPin και reedPin ως input, δηλαδή δέχονται ρεύμα μόνο, ενώ το soundPin και indicatorPin θα δώσουν ρεύμα, δεν θα δεχτούν οπότε θα είναι output.
Το Serial.begin(9600) περνάει την τιμή 9600 στο speed parameter, δηλαδή, «λέει» στο Arduino να ετοιμαστεί να ανταλλάξει μηνύματα με το Serial Monitor σε ένα data rate των 9600 bits per second. Στην παρούσα εργασία αυτός ο αριθμός δεν έχει σημασία.

```
36  void loop(){
37    char key = keypad.getKey();
38    Serial.println(String("Key: ") + key);
39    Serial.println(String("Previous: ") + previous);
40    Serial.println(String("Arm: ") + arm);
41    Serial.println("======================");
42
43    if (key == '1' && previous == '3'){
44      arm = 1;
45      previous = '0';
46    } else if (key == '3'){
47      previous = '3';
48    } else if (keypad.getState() == PRESSED){
49      previous = '0';
50    }
51
52    if (arm == 1 && (digitalRead(motionPin) == HIGH || digi-
53  talRead(reedPin) == HIGH)){
54      digitalWrite(soundPin,HIGH);
55      Serial.println("======ALARM======");
56    }
57    if (key == '4'){
58      digitalWrite(soundPin,LOW);
59      Serial.println("===DEACTIVATED===");
60      arm = 0;
61    }
62    if (arm == 1) {
63      digitalWrite(indicatorPin,HIGH);
64    } else {
65      digitalWrite(indicatorPin,LOW);
66    }
67  }
```

Γραμμή 36:
Η συνάρτηση loop εκτελείται συνεχόμενα επιτρέποντας στο πρόγραμμα μας να αλλάζει και να απαντάει.

Γραμμή 37:
Στην μεταβλητή key αποθηκεύουμε το κλειδί που πατιέται. Αν δεν έχουμε πατήσει κάποιο κουμπί τότε θα είναι κενό.

Γραμμές 38 – 41:
Σε αυτές τις γραμμές εμφανίζουμε στην κονσόλα κάποιες πληροφορίες για λόγους debugging. Συγκεκριμένα, εμφανίζουμε το κλειδί που πατήθηκε, το προηγούμενο κουμπί που πατήθηκε, αν αυτό είναι το 3 και αν ο συναγερμός είναι ενεργοποιημένος.

Γραμμές 43 – 50:
Σε αυτό το σημείο χωρίζουμε τις περιπτώσεις για το previous. Αν έχει πατηθεί προηγουμένως το 3 και μετά πατηθεί το 1, τότε κάνε το arm 1 γιατί δόθηκε ο σωστός κωδικός και μηδένισε ξανά το previous. Αν πατηθεί το 3, τότε αποθήκευσε το στο previous. Οποιοδήποτε άλλο κλειδί αν πατηθεί τότε κάνε το previous 0. Με αυτό το κομμάτι κώδικα βλέπουμε αν έχει δοθεί ο σωστός κωδικός ή όχι.

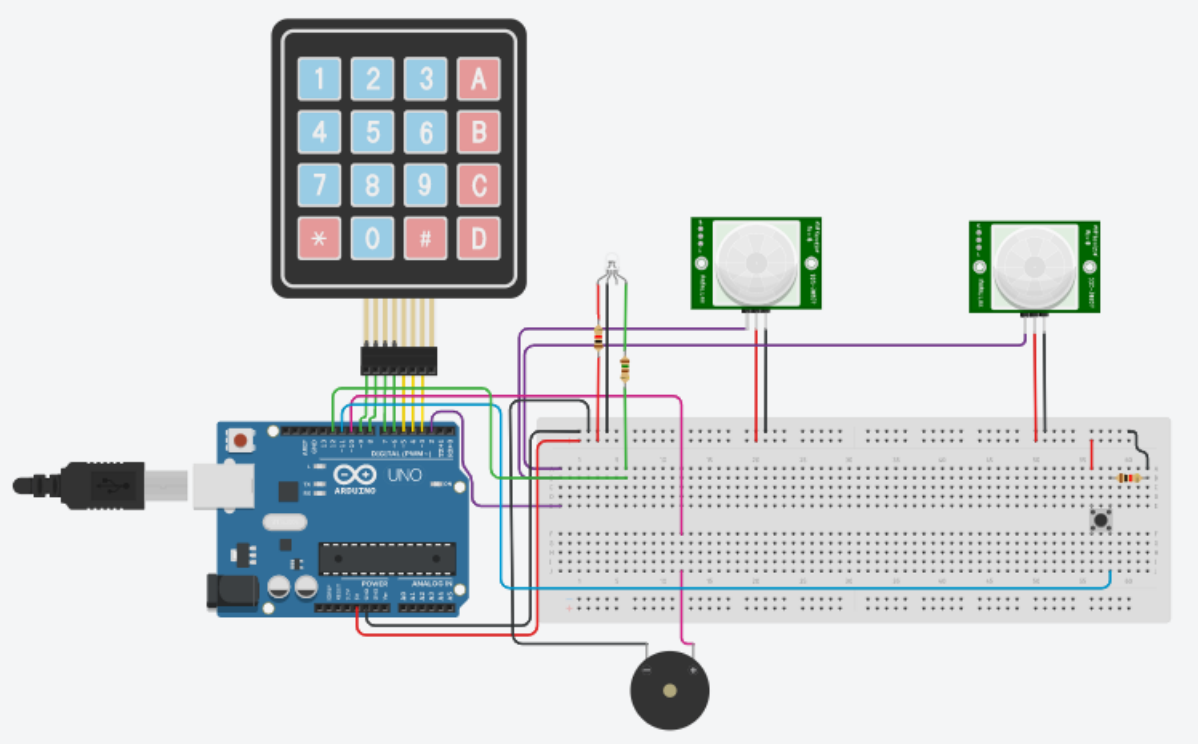
Γραμμές 52 – 56:
Στην περίπτωση που είναι ενεργός ο συναγερμός και την ίδια στιγμή εντοπισθεί κίνηση είτε στο motionPin είτε στο reedPin, τότε ενεργοποίησε τον συναγερμό. Διαβάζουμε το motionPin αν είναι HIGH (δηλαδή το pin 2) και διαβάζουμε το reedPin αν είναι HIGH (δηλαδή το pin 11). Στη συνέχεια, γράφουμε στο soundPin το HIGH (δίνουμε ρεύμα) και εκτυπώνουμε στην κονσόλα το ALARM.

Γραμμές 57 – 61:
Αν πατηθεί ο κωδικός 4, τότε δεν δίνουμε ρεύμα πια στο soundPin (pin 10) περνώντας του την τιμή LOW , εκτυπώνουμε στην κονσόλα το μήνυμα της απενεργοποίησης και αναθέτουμε στην μεταβλητή arm την τιμή 0.

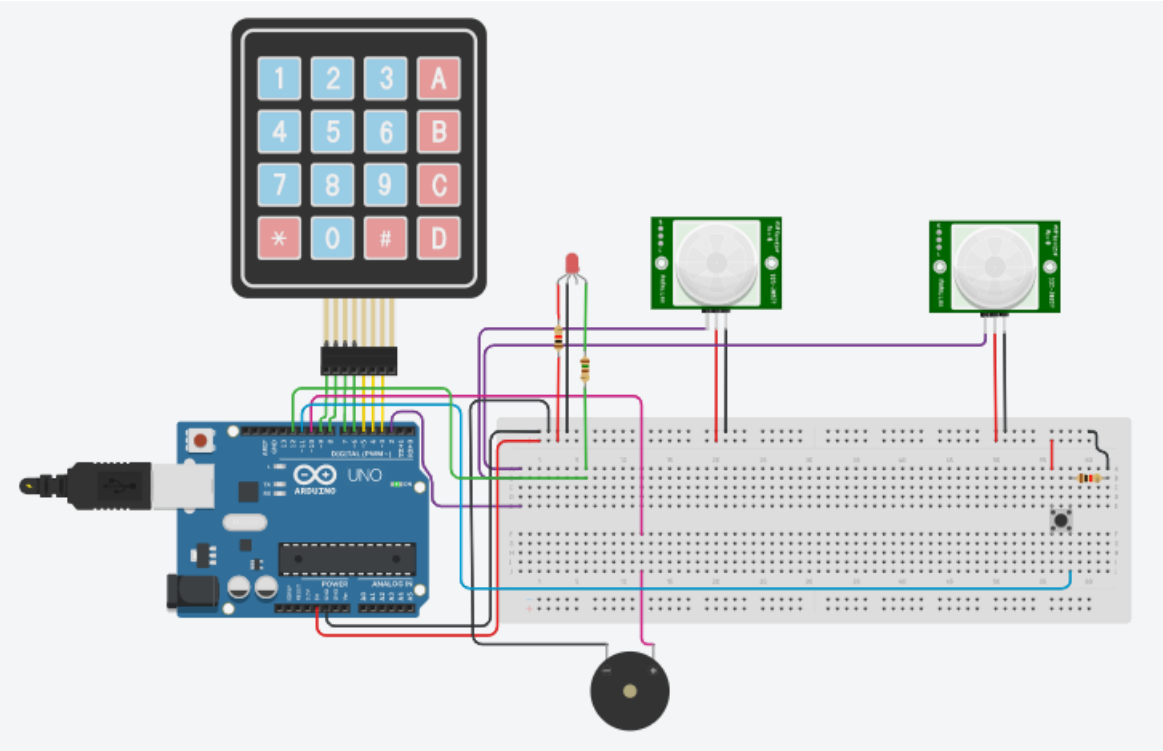
Γραμμές 62 – 66:
Τέλος, έχουμε υλοποιήσει τον κώδικα για το indicatorPin, δηλαδή το λαμπάκι, όπου δίνουμε ρεύμα για να γίνει πράσινο όταν είναι ενεργός ο συναγερμός, ενώ δεν δίνουμε ρεύμα όταν είναι απενεργοποιημένος και φαίνεται το κόκκινο χρώμα.

SCREENSHOTS

Συναγερμός εκτός ρεύματος - λευκό λαμπάκι



Απενεργοποιημένος συναγερμός – κόκκινο λαμπάκι



Ενεργοποιημένος συναγερμός – πράσινο λαμπάκι