ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστημάτων

4η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΑΝΑΦΟΡΑ

ΟΜΑΔΑ: 16
ΙΩΑΝΝΗΣ ΔΑΝΙΑΣ, Α.Μ: 03120831
ΣΤΑΥΡΟΣ ΜΗΤΡΟΠΟΥΛΟΣ, Α.Μ.: 03120847

Άσκηση 1:

Α) Κάνουμε build το project με επιλεγμένο το estimate performance, για την forward_propagation σαν HW function.

Details

Performance estimates for 'forward_propagation in main.cp ...

HW accelerated (Estimated cycles)	683780

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	3	80	3.75
BRAM	16	60	26.67
LUT	1760	17600	10
FF	892	35200	2.53

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
104625	104625	104626	104626	none

Detail

Instance

Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- read_input	1568	1568	4	-	-	392	no
- layer_1	36456	36456	93	-	-	392	no
+ layer_1.1	90	90	3	-	-	30	no
- layer_1_act	60	60	2	-	-	30	no
- layer_2	4600	4600	92	-	-	50	no
+ layer_2.1	90	90	3	-	-	30	no
- layer_3	61936	61936	158	-	-	392	no
+ layer_3.1	150	150	3	-	-	50	no

Β)

Estimation και actual speedup έχουν διαφορά μόλις 1000 κύκλους, το οποίο λόγω τάξης μεγέθους δεν είναι σημαντικό.

Έχουμε speed up 2.16

```
sh-4.3# ./GAN_accelerator.elf
Starting dataset parsing...
Parsing finished...
Starting hardware calculations...
Hardware calculations finished.
Starting software calculations...
Software calculations finished.
Hardware cycles : 682934
Software cycles : 1475575
Speed-Up        : 2.16064
Saving results to output.txt...
sh-4.3#
```

Γ)

Δοκιμάσαμε διάφορα HLS directives (dataflow, array partitioning, DSP resources allocation directives etc) και καταλήξαμε σε pipeline design με memory partitioning των πινάκων. Έχοντας το δεδομένο από πριν ότι estimation cycles και hardware cycles είναι πολύ κοντά, οι 12038 κύκλοι είναι ικανοποιητικοί καθώς προσεγγίζουν το 120 speedup. Βλέπουμε ότι στο hardware πιάνουμε 12329 κύκλους και speedup 119.59. Είναι fully pipelined design με II=1, το οποίο δίνει μεγάλη επίδοση καθώς αποφεύγει πολλά stalls. Έχουμε συνεπώς πολύ μεγάλη κλιμάκωση σε σχέση με την unoptimized έκδοση.

Details

Performance estimates for 'forward_propagation in main.cp ...

HW accelerated (Estimated cycles)	12038

Resource utilization estimates for HW functions

Resource	Used	Total	% Utilization
DSP	80	80	100
BRAM	40	60	66.67
LUT	7335	17600	41.68
FF	7584	35200	21.55

```
sh-4.3# ./GAN_accelerator.elf
Starting dataset parsing...
Parsing finished...
Starting hardware calculations...
Hardware calculations finished.
Starting software calculations...
Software calculations finished.
Hardware cycles : 12261
Software cycles : 1475714
Speed-Up        : 120.358
Saving results to output.txt...
```

Δ)

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- read_input	395	395	5	1	1	392	yes
- layer_1	393	393	3	1	1	392	yes
- layer_1_act	30	30	1	1	1	30	yes
- layer_2	52	52	4	1	1	50	yes
- layer_3	400	400	10	1	1	392	yes

Παρατηρούμε ότι το τελευταίο loop έχει το μεγαλύτερο latency και ακολουθεί το read_input. Αυτό συμβαίνει διότι αυτά επικοινωνούν με την DRAM η οποία δεν είναι πάνω στο FPGA και άρα είναι αρκετά πιο αργή η πρόσβαση σε αυτή και δεν υπάρχει ταυτοχρονισμός στην πρόσβαση της.
Το design είναι fully pipelined, το οποίο δίνει κλιμακώνει το πρόγραμμα καθώς έχουμε άμεση επαναχρησιμοποίηση των MODULES που δημιουργεί το FPGA.

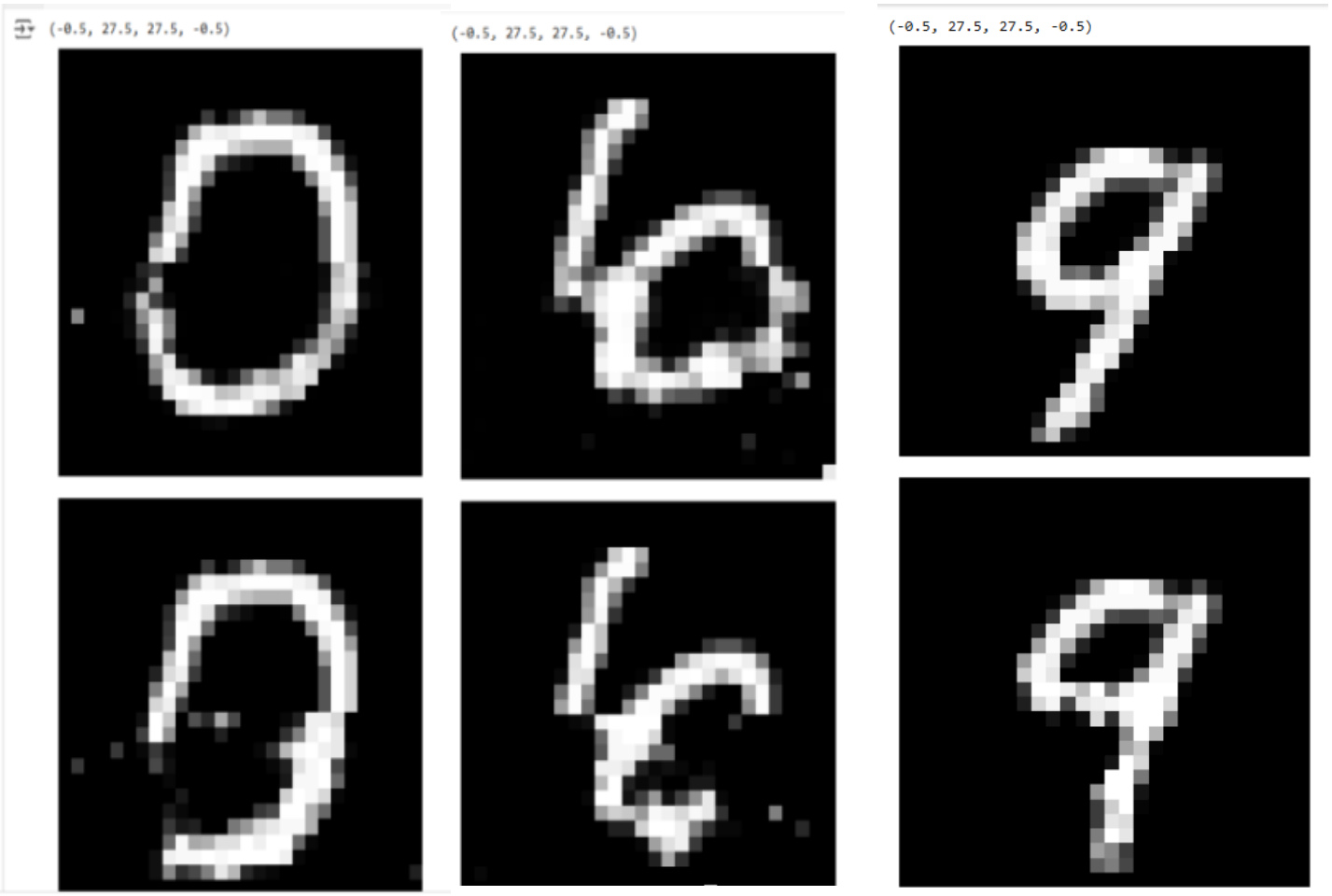
	BRAM	DSP	FF	LUT	Bits P0	Bits P1	Bits P2	B
forward_propagation	**81**	**80**	**7584**	**7256**				
I/O Ports(2)					**64**			
Instances(3)	**0**	**0**	**268**	**830**				
Memories(139)	**81**		**833**	**316**	**1286**			**1**
Expressions(474)	**0**	**80**	**0**	**4497**	**4640**	**4996**	**1544**	
-	**0**	**0**	**0**	**78**	**16**	**78**	**0**	
*	**0**	**80**	**0**	**0**	**1233**	**764**	**0**	
+	**0**	**0**	**0**	**2483**	**2962**	**2930**	**0**	
and	**0**	**0**	**0**	**5**	**5**	**5**	**0**	
ashr	**0**	**0**	**0**	**161**	**54**	**54**	**0**	
icmp	**0**	**0**	**0**	**83**	**223**	**79**	**0**	
or	**0**	**0**	**0**	**7**	**7**	**7**	**0**	
select	**0**	**0**	**0**	**1571**	**95**	**1030**	**1544**	
shl	**0**	**0**	**0**	**88**	**32**	**32**	**0**	
xor	**0**	**0**	**0**	**21**	**13**	**17**	**0**	

Σχετικά με τα expressions, βλέπουμε πως το DSP κάνει μόνο πολλαπλασιασμούς.

Άσκηση 2:

Α) Τρέξαμε για τον optimized κώδικα του προιγούμενου ερωτήματος και μετά την εισαγωγή του output.txt έχουμε από την εκτέλεση στο notebook τα παρακάτω δεδομένα για idx: 10, 11 και 12 με την σειρά

B)
BITS = 4



BITS = 10

Παρατηρούμε ότι για την μικρή ακρίβεια τον 4 bit έχουμε εικόνες από το hardware με πολύ μεγάλη διαφορά από ότι αυτές που υλοποιούνται στο software ενώ σε αντίθεση για τα 10 bit ακρίβεια έχουμε πολύ καλύτερη ευκρίνεια και σχεδόν πανομοιότυπα με αυτά του software. Γενικά μπορούμε να πούμε με βεβαιότητα ότι όσα περισσότερα bit έχουμε για τους υπολογισμούς τόσο καλύτερα και ακριβέστερα τα αποτελέσματα θα είναι, το trade off όμως είναι οι πόροι που χρησιμοποιούνται καθώς και μετά από κάποιο όριο η ταχύτητα.

Γ)
4 BITS:

```
Max pixel error:  255
Peak Signal-to-Noise Ratio:  13.525831164368576
```

5 BITS:

```
Max pixel error:  229
Peak Signal-to-Noise Ratio:  19.407263480245117
```

6 BITS:

```
Max pixel error:  93
Peak Signal-to-Noise Ratio:  29.23557136050292
```

8 BITS:

```
Max pixel error:  13
Peak Signal-to-Noise Ratio:  47.065287020211215
```

10 BITS:

```
Max pixel error:  4
Peak Signal-to-Noise Ratio:  53.76982650203158
```

Όπως είναι αναμενόμενο όσα περισσότερα bits έχουμε τόσο μικρότερο είναι το σφάλμα που έχουμε, αφού έχουμε περισσότερα bits να αποτυπώσουμε την εικόνα. Από την άλλη παρατηρούμε πως όσα περισσότερα είναι τα bits χάνουμε σε Performance. Από τα 8 στα 10 bits έχουμε ένα σκαλοπάτι και στο speedup πέφτοντας από το φράγμα του 120 στο 85 speedup. Λαμβάνοντας αυτό υπόψη, ανάλογα τις απαιτήσεις της εφαρμογής, επιλέγουμε το 8bit για συνδυασμό επίδοσης-ακρίβειας και το 10bit για την καλύτερη δυνατή ακρίβεια.