ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

Εξαμηνιαία Εργασία

Προγραμματισμός Ενσωματωμένων Συστημάτων σε περιβάλλοντα Edge

Αναστασιάδης Παναγιώτης, 22101
Καραβιβέρη Αλεξάνδρα, 22103

2 Ιουλίου 2023

Πίνακας Περιεχομένων

Υλοποίηση Debayering Φίλτρου μέσω High Level Synthesis

Κατασκευή χρήσιμων συναρτήσεων σε Colab Notebook

Πριν την διαδικασία υλοποίησης του ζητούμενου κυκλώματος μέσω HLS, κατασκευάσαμε ένα .ipynb αρχείο στο Colab με βοηθητικά scripts.

Script 1: Εξαγωγή αρχείου .txt με 8-bit Bayer GRBG τιμές από δοσμένη εικόνα

Αρχικά, φτιάξαμε ένα script το οποίο δέχεται μια εικόνα σε μέγεθος 256x256. Στην συνέχεια για το κάθε pixel, μετατρέπουμε τις RGB τιμές του σε binary μορφή και ανάλογα με την θέση του, κρατάμε μόνο μία από αυτές τις τιμές. Τις τιμές αυτές τις αποθηκεύουμε σε ένα txt αρχείο το οποίο θα δοθεί σαν input στον κώδικα του hls. Έχουμε σαν αποτέλεσμα μία εικόνα που έχει περάσει από Bayer GBRG φίλτρο.

Script 2: Υλοποίηση Debayering στο δοσμένο .txt αρχείο και παραγωγή debayering processed εικόνας και αρχείου .txt

Το δεύτερο script διαβάζει το αρχείο που παρήγαγε το προηγούμενο script και κάνει την διαδικασία του Debayering. Για κάθε pixel, μέσω της θέσης που έχει, βρίσκουμε ποια τιμή έχει αποθηκευτεί από τις RGB και στην συνέχεια με τη βοήθεια των γειτονικών του 3x3 pixels παράγει τις τιμές των RGB που του λείπουν. Τέλος, αποθηκεύει την νέα εικόνα που έχει παραχθεί, καθώς και ένα .txt αρχείο με τις παραγόμενες, κατά Debayering RGB, τιμές της εικόνας.

Script 3: Μετασχηματισμός του παραγόμενου από τη debayering διαδικασία αρχείου .txt σε εικόνα.

Το script διαβάζει ένα txt αρχείο με RGB τιμές και παράγει την ανάλογη εικόνα.

Υλοποίηση debayering filter μέσω High Level Synthesis

Ορίζουμε μια συνάρτηση debayering_filter:

- με input έναν 2D πίνακα pixels NxN, όπου κάθε στοιχείο του είναι μια 8-bit (unsigned char) τιμή (Red, Green ή Blue αναλόγως με τη κωδικοποίηση του Bayer για το συγκεκριμένο pixel)
- και output έναν 3D πίνακα output_pixels με διαστάσεις NxNx3, όπου κάθε στοιχείο του απαρτίζει μια τριπλέτα binary τιμών R, G, B που έχουν παραχθεί από την εφαρμογή του debayering φίλτρου.

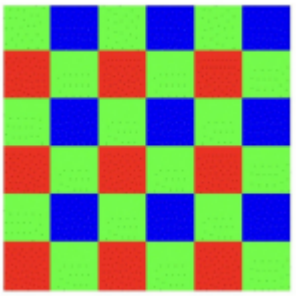
```
#define N 256

void debayering_filter(unsigned char pixels[N][N], unsigned char output_pixels[N][N][3]){
```

Για την εφαρμογή του φίλτρου σε κάθε στοιχείο του πίνακα pixels, διατρέχουμε τον πίνακα στοιχείο προς στοιχείο, ξεκινώντας από το pixel (1,1) και φτάνοντας μέχρι το (N-2, N-2), ενώ αποκλείουμε τα περιμετρικά pixels για τα οποία δεν μπορεί να υπολογιστεί η φόρμουλα του debayering.

```
// we ignore row = 0, column = 0, row = 256, column = 256
// we can't get 3x3 for these rows and columns
for(int row = 1; row < N-1; row++){
    #pragma HLS unroll factor=2
    for(int column = 1; column < N-1; column++){
```

Στο εσωτερικό των for-loops, ορίζουμε το χρώμα του τρέχοντος pixel με βάση το index του στον πίνακα και παράλληλα συμβουλευόμαστε την κατανομή των χρωμάτων κατά το πρότυπο GRBG.

```
#pragma HLS pipeline II=2
// identify which value was stored from {R, G, B}
if((row % 2) == 0 && (column % 2) == 0){
    color = Green;
}
else if((row % 2) == 0 && (column % 2) == 1){
    color = Blue;
}
else if((row % 2) == 1 && (column % 2) == 0){
    color = Red;
}
else if((row % 2) == 1 && (column % 2) == 1){
    color = Green;
}
```



Στη συνέχεια, αποθηκεύουμε σε έναν 3x3 πίνακα το τρέχον pixel και τους γείτονές του.

```
// get 3x3 neighbors, our pixel is the middle one [1][1]
unsigned char matrix[3][3];

matrix[0][0] = pixels[row-1][column-1];
matrix[0][1] = pixels[row-1][column];
matrix[0][2] = pixels[row-1][column+1];

matrix[1][0] = pixels[row][column-1];
matrix[1][1] = pixels[row][column];
matrix[1][2] = pixels[row][column+1];

matrix[2][0] = pixels[row+1][column-1];
matrix[2][1] = pixels[row+1][column];
matrix[2][2] = pixels[row+1][column+1];
```

Με βάση το επιλεγμένο χρώμα για το τρέχον pixel, υπολογίζουμε κατάλληλα τους μέσους όρους, μέσω της φόρμουλας του debayering και αποθηκεύουμε τις παραγόμενες τιμές R,G,B στην αντίστοιχη θέση στον πίνακα output_pixels.

```
// Calculate RGB values based on color
if(color == Red){
// #pragma HLS occurrence cycle=4
    r =  matrix[1][1];
    g = (matrix[0][1] + matrix[1][0] + matrix[1][2] + matrix[2][1]) / 4;
    b = (matrix[0][0] + matrix[0][2] + matrix[2][0] + matrix[2][2]) / 4;
}
else if(color == Green){
// #pragma HLS occurrence cycle=2
    r = (matrix[1][0] + matrix[1][2]) / 2;
    g = matrix[1][1];
    b = (matrix[0][1] + matrix[2][1]) / 2;
}
else if(color == Blue){
// #pragma HLS occurrence cycle=4
    r = (matrix[0][0] + matrix[0][2] + matrix[2][0] + matrix[2][2]) / 4;
    g = (matrix[0][1] + matrix[1][0] + matrix[1][2] + matrix[2][1]) / 4;
    b = matrix[1][1];
}

// Store RGB values in output_pixels
output_pixels[row][column][0] = r;
output_pixels[row][column][1] = g;
output_pixels[row][column][2] = b;
```

Όπως αναφέραμε και παραπάνω, οι τιμές R, G, B των περιμετρικών pixels δεν μπορούν να υπολογιστούν όπως τα εσωτερικά. Η στρατηγική που ακολουθήσαμε για το “γέμισμα” των συγκεκριμένων pixels, είναι να τα “γεμίσουμε” με την τιμή του γειτονικού pixel (π.χ. τα περιμετρικά pixels στα αριστερά θα πάρουν τη τιμή του δεξιού γειτονικού pixel και τα περιμετρικά pixel στα δεξιά με την τιμή του αριστερού γειτονικού pixel).

Οπότε, για τα περιμετρικά pixels που βρίσκονται δεξιά και αριστερά, κάνουμε τις ανάλογες αντιγραφές, έξω από το εσωτερικό for-loop.

```
// Calculate perimeter RGB values for col = 0 and col = N-1
output_pixels[row][0][0] = output_pixels[row][1][0];
output_pixels[row][0][1] = output_pixels[row][1][1];
output_pixels[row][0][2] = output_pixels[row][1][2];

output_pixels[row][N - 1][0] =  output_pixels[row][N - 2][0];
output_pixels[row][N - 1][1] =  output_pixels[row][N - 2][1];
output_pixels[row][N - 1][2] =  output_pixels[row][N - 2][2];
```

Τέλος, μετά το πέρας των επαναλήψεων, η παραγόμενη εικόνα, έχει σχεδόν ολοκληρωθεί. Στο τελευταίο στάδιο, αντιγράφουμε κατάλληλα, μέσω ενός for-loop, τις γραμμές 1 και 254 στις γραμμές 0 και 255 αντιστοίχως για να καλύψουμε και τις οριζόντιες περιμετρικές γραμμές.

```
// Calculate perimeter RGB values for the first and last row
for (int column = 0; column < N; column++) {
    // #pragma HLS pipeline II=1
    // #pragma HLS  unroll factor=2

    output_pixels[0][column][0] = output_pixels[1][column][0];
    output_pixels[0][column][1] = output_pixels[1][column][1];
    output_pixels[0][column][2] = output_pixels[1][column][2];

    output_pixels[N - 1][column][0] = output_pixels[N - 2][column][0];
    output_pixels[N - 1][column][1] = output_pixels[N - 2][column][1];
    output_pixels[N - 1][column][2] = output_pixels[N - 2][column][2];
}
```

Υλοποίηση testbench για την επαλήθευση του κυκλώματος

Για την επαλήθευση του debayering filter φτιάχνουμε ένα testbench με τις εξής λειτουργίες:

- διάβασμα αρχείου image_8bit.txt, το οποίο περιέχει 256 γραμμές με 256 8-bit τιμές R,G ή B (αναλόγως τη θέση του pixel κατα τη bayering κωδικοποίηση) η κάθε μία.
- αποθήκευση των τιμών σε έναν πίνακα pixels 256x256.
- Εκτέλεση του hardware accelerator debayering_filter.
- Αποθήκευση αποτελεσμάτων σε ένα αρχείο output.txt

Σύνθεση χωρίς τεχνικές βελτιστοποίησης

Σε πρώτο στάδιο, πραγματοποιούμε σύνθεση του κυκλώματος χωρίς την χρησιμοποίηση HLS directives.

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	6.940 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
454928	454928	4.549 ms	4.549 ms	454928	454928	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	453390	453390	1785	-	-	254	no
+ Loop 1.1	1778	1778	7	-	-	254	no
- Loop 2	1536	1536	6	-	-	256	no

Όπως παρατηρούμε, το συνολικό latency του κυκλώματος είναι αρκετά μεγάλο (454928 clock cycles), ενώ για να ολοκληρωθεί ένα iteration του εξωτερικού loop χρειάζονται 1785 κύκλους ρολογιού.

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1059	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	384	-
Register	-	-	418	-	-
Total	0	0	418	1443	0
Available	270	240	82000	41000	0
Utilization (%)	0	0	~0	3	0

Το utilization πιάνει το 3% των Lookup Tables.

Σύνθεση με χρήση hls pipeline

Σαν πρώτη βελτιστοποίηση, επιχειρούμε να παραλληλοποιήσουμε τα for-loops του κώδικα, ώστε να μη γίνονται σειριακά και να πετύχουμε επίδοση με λιγότερους κύκλους ρολογιού.

Χρησιμοποιώντας το HLS pipeline με II=1 στο εξωτερικό for-loop (που διατρέχει τα rows), πιάνουμε το 200% του utilization των LUTs, συνεπώς απορρίπτουμε τη συγκεκριμένη βελτιστοποίηση.

```
for(int row = 1; row < N-1; row++){
    #pragma HLS pipeline II=1
```

Total	0	0	13845	82271	0
Available	270	240	82000	41000	0
Utilization (%)	0	0	16	200	0

Χρησιμοποιώντας τη τεχνική του pipeline στο εσωτερικό for-loop (που διατρέχει τα columns), επιτυγχάνουμε initiation interval = 5 (το σύνολο των κύκλων που πρέπει να περάσουν για να δρομολογηθεί το επόμενο iteration) και το συνολικό latency του κυκλώματος μειώνεται στους 326404 κύκλους ρολογιού (μείωση 128524 κύκλοι σε σχέση με πριν).

```
for(int row = 1; row < N-1; row++){
    for(int column = 1; column < N-1; column++){
        #pragma HLS pipeline II=1
```

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
326404	326404	3.264 ms	3.264 ms	326404	326404	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	324866	324866	1279	-	-	254	no
+ Loop 1.1	1271	1271	7	5	1	254	yes
- Loop 2	1536	1536	6	-	-	256	no

Τέλος, το utilization παραμένει στο 3% μετά τη χρήση το pipelining.

Total	0	0	405	1454	0
Available	270	240	82000	41000	0
Utilization (%)	0	0	~0	3	0

Προσθήκη βελτιστοποίησης array partitioning

Η χρήση του pipelining μπορεί να συνδυαστεί με την τεχνική του array partitioning, όπου μπορούμε να διαιρέσουμε τον NxN πίνακα των pixels σε μικρότερα subarrays-blocks για να ενισχύσουμε τη παραλληλοποίηση. Συγκεκριμένα, τα νέα partitions του αρχικού πίνακα μπορούν να συμβάλλουν σε πολλά και παράλληλα accesses στα στοιχεία του πίνακα, το οποίο έχει αποτέλεσμα την παράλληλη εκτέλεση περισσότερων εντολών ανά τα cycles.

Array Block Partitioning

Χρησιμοποιώντας το block array partitioning με factor=3 στο rows dimension, σπάμε τον πίνακα σε partitions 3 σειρών pixels.

```
void debayering_filter(unsigned char pixels[N][N], unsigned char output_pixels[N][N][3]){
#pragma HLS ARRAY_PARTITION variable=pixels block factor=3 dim=1
```

Εκτελώντας τη σύνθεση του κυκλώματος, το αποτέλεσμα δεν επιφέρει καμία βελτιστοποίηση στο συνολικό latency.

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
326404	326404	3.264 ms	3.264 ms	326404	326404	none

Αυτό πιθανόν συμβαίνει, γιατί εφόσον διατρέχουμε τον πίνακα κατά σειρά και ταυτόχρονα επιδιώκουμε τη παραλληλοποίηση της επανάληψης, υπάρχουν πολλαπλά accesses στο ίδιο block με συνέπεια τον ανταγωνισμό των πόρων που δεν επιφέρει τελικά την προσδοκώμενη βελτιστοποίηση.

Array Cyclic Partitioning

Το παραπάνω πρόβλημα μπορεί να διορθωθεί με το cyclic partitioning. Θέτοντας το συγκεκριμένο directive με dim=0 (το εκτελεί και κατά σειρά και κατά στήλη) και factor=2 (προέκυψε πειραματικά) επιτυγχάνουμε το διαμοιρασμό των στοιχείων κυκλικά, με τη λογική που φαίνεται στον παρακάτω πίνακα (ο κάθε αριθμός αντιπροσωπεύει ένα από τα 4 partitions):

0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

```
void debayering_filter(unsigned char pixels[N][N], unsigned char output_pixels[N][N][3]){

#pragma HLS ARRAY_PARTITION variable=pixels cyclic factor=2 dim=0
```

Ο συγκεκριμένος διαμοιρασμός των στοιχείων φαίνεται πως ενισχύει σημαντικά την παραλληλοποίηση, καθώς κατά το pipelining του εσωτερικού for-loop και με τον τρόπο που διατρέχουμε το κάθε pixel μαζί με τους γείτονες του (για κάθε pixel κοιτάμε τα γειτονικά pixel πάνω, κάτω, δεξιά και αριστερά), είναι εφικτό να πραγματοποιούνται πολλαπλά memory accesses παράλληλα.

Αυτό αποδεικνύεται και από τα νέα αποτελέσματα που αφορούν τα χαρακτηριστικά του κυκλώματος.

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
133364	133364	1.334 ms	1.334 ms	133364	133364	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	131826	131826	519	-	-	254	no
+ Loop 1.1	511	511	6	2	1	254	yes
- Loop 2	1536	1536	6	-	-	256	no

Το συνολικό latency έχει πέσει πλέον στα 133364 cycles, το εξωτερικό loop έχει iteration latency = 519 (προηγουμένως 1279) και τέλος στο εσωτερικό loop πετύχαμε initiation interval = 2 (προηγουμένως 5).

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1317	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	537	-
Register	0	-	541	32	-
Total	0	0	541	1886	0
Available	270	240	82000	41000	0
Utilization (%)	0	0	~0	4	0

Το utilization αυξήθηκε των LUTs αυξήθηκε ελάχιστα και πήγε στο 4%.

Προσθήκη βελτιστοποίησης loop unrolling

Η επόμενη βελτιστοποίηση που επιχειρούμε είναι το loop unrolling στο εξωτερικό loop των rows. Θέτοντας το συγκεκριμένο directive με factor=2 (προέκυψε πειραματικά) διασπάται το εσωτερικό loop σε 2 επιμέρους loops και επιτυγχάνεται περαιτέρω παραλληλοποίηση.

```
// we ignore row = 0, column = 0, row = 256, column = 256
// we can't get 3x3 for these rows and columns
for(int row = 1; row < N-1; row++){
    #pragma HLS unroll factor=2
    for(int column = 1; column < N-1; column++){
        #pragma HLS pipeline II=1
```

Τα νέα αποτελέσματα δείχνουν περαιτέρω μείωση του latency κατά 381 κύκλους.

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
132983	132983	1.330 ms	1.330 ms	132983	132983	none

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	131445	131445	1035	-	-	127	no
+ Loop 1.1	510	510	5	2	1	254	yes
+ Loop 1.2	510	510	5	2	1	254	yes
- Loop 2	1536	1536	6	-	-	256	no

Με τα νέα δεδομένα το utilization αυξήθηκε κατα 2%.

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1966	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	877	-
Register	-	-	822	-	-
Total	0	0	822	2843	0
Available	270	240	82000	41000	0
Utilization (%)	0	0	1	6	0

Αποτυχημένες προσπάθειες βελτιστοποίησης

Αφού πετύχαμε μια ικανοποιητική βελτιστοποίηση του κυκλώματος, επιχειρήσαμε να δοκιμάσουμε επιπλέον directives με μηδενικό αντίκτυπο στην απόδοση. Αυτά ήταν τα εξής:

- pipelining και loop unrolling στο loop που γίνεται η ανάθεση των pixels στις οριζόντιες περιμετρικές σειρές (row = 0, row = 255)

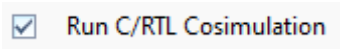
```
// Calculate perimeter RGB values for the first and last row
for (int column = 0; column < N; column++) {
     #pragma HLS pipeline II=1
     #pragma HLS  unroll factor=2

    output_pixels[0][column][0] = output_pixels[1][column][0];
    output_pixels[0][column][1] = output_pixels[1][column][1];
    output_pixels[0][column][2] = output_pixels[1][column][2];

    output_pixels[N - 1][column][0] = output_pixels[N - 2][column][0];
    output_pixels[N - 1][column][1] = output_pixels[N - 2][column][1];
    output_pixels[N - 1][column][2] = output_pixels[N - 2][column][2];
}
```

- hls pragma occurrence στα if-blocks

```
// Calculate RGB values based on color
if(color == Red){
// #pragma HLS occurrence cycle=4
    r =  matrix[1][1];
    g = (matrix[0][1] + matrix[1][0] + matrix[1][2] + matrix[2][1]) / 4;
    b = (matrix[0][0] + matrix[0][2] + matrix[2][0] + matrix[2][2]) / 4;
}
```

Επαλήθευση του κυκλώματος και ανίχνευση μη ομαλής λειτουργίας

Αφού εκτελέσουμε την προσομοίωση του κυκλώματος με την εκτέλεση της παρακάτω επιλογής του Vivado HLS:



Επαληθεύουμε την ορθή λειτουργία του κυκλώματος:

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	132983	132983	132983	NA	NA	NA

Με τη βοήθεια του Colab Notebook, από το οποίο και εξάγαμε το input αρχείο με τις 8-bit τιμές, επαληθεύουμε το παραγόμενο output.txt αρχείο που περιέχει τις rgb τιμές που προέκυψαν από το debayering.

Αφου μετατρέψουμε τις rgb τιμές του αρχείου σε εικόνα, συγκρίνουμε και επαληθεύουμε την αρχική φωτογραφία με αυτή που προέκυψε από το debayering.

Original **After Debayering**

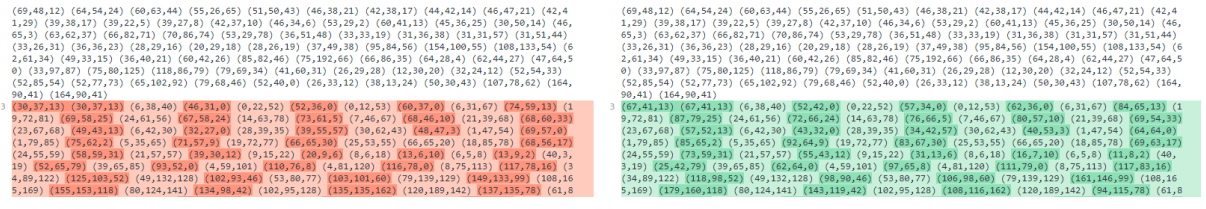
Όπως παρατηρούμε, το φίλτρο φαίνεται να δουλεύει καλά και η επεξεργασμένη εικόνα μοιάζει με την αρχική.

Στο σημείο αυτό, πραγματοποιούμε, για την ίδια φωτογραφία, τη διαδικασία bayering και debayering μέσα από τη python και το Colab Notebook.

Το .txt που παράχθηκε από το script συγκρίνεται με αυτό που παράχθηκε κατά το C\RTL simulation.

Έχοντας ορίσει τη διαδικασία με τέτοιο τρόπο ώστε να παράγει το ίδιο ακριβώς format αρχείου τόσο στο python script όσο και στο testbench του κυκλώματος, χρησιμοποιούμε ένα online δωρεάν εργαλείο **Diffchecker** (https://www.diffchecker.com/text-compare/) για να επαληθεύσουμε την καθολική ομοιότητα των δυο αρχείων.

Συνεπώς και με τη βοήθεια του εργαλείου παρατηρούμε το εξής φαινόμενο:



Στη παραπάνω φωτογραφία απεικονίζεται αριστερά το αρχείο με τα παραγόμενα RGB values του python script και δεξιά το αρχείο που παρήγαγε το κύκλωμα. Το φαινόμενο που παρατηρήθηκε είναι ότι τα στοιχεία που αντιστοιχούσαν στα μπλε pixel του bayering φίλτρου έχουν λανθασμένους μέσους όρους για τις τιμές green και red).

Μετά από ενδελεχές ψάξιμο και εφόσον επαληθεύτηκε η σωστή λειτουργία της προσομοίωσης του κώδικα c διότι παρήγαγε το ίδιο αρχείο με το python script, καταλήξαμε στο γεγονός ότι το πρόβλημα παρατηρείται μόνο στο τελικό synthesized κύκλωμα.

Επίλυση του προβλήματος

Για να πετύχουμε τη σωστή λειτουργία του, δοκιμάσαμε τη προσθαφαίρεση directives, καθώς και την αύξηση της περιόδου του ρολογιού. Αυτό που τελικά πέτυχε ήταν η "χαλάρωση" του initiation interval στο pipelining από 2 κύκλους σε 3. Αυτό είχε ως συνέπεια την επιβάρυνση του τελικού latency.

Τα τελικά αποτελέσματα του κυκλώματος είναι τα παρακάτω:

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.680 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
197499	197499	1.975 ms	1.975 ms	197499	197499	none

Detail

Instance

Loop

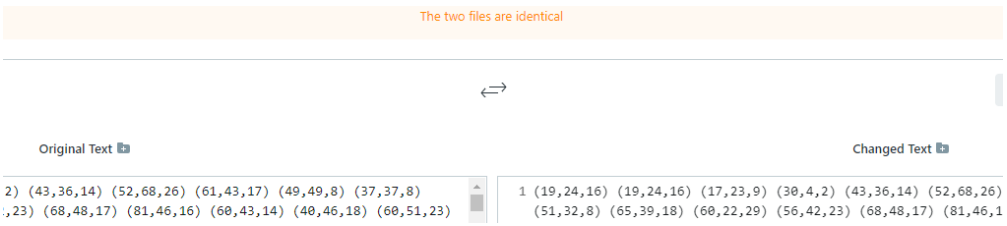
	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	195961	195961	1543	-	-	127	no
+ Loop 1.1	764	764	6	3	3	254	yes
+ Loop 1.2	764	764	6	3	3	254	yes
- Loop 2	1536	1536	6	-	-	256	no

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1966	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	864	-
Register	-	-	856	-	-
Total	0	0	856	2830	0
Available	270	240	82000	41000	0
Utilization (%)	0	0	1	6	0

Η αλλαγή στο initiation interval επιβαρύνει το τελικό latency το οποίο πλέον φτάνει τον αριθμό των 197499 κύκλων (πριν 132983), ενώ το iteration latency για το εξωτερικό loop έφτασε πλέον τους 1543 κύκλους (πριν 1035).

Όπως παρατηρούμε, το νέο κύκλωμα παράγει πλέον το ίδιο αποτέλεσμα με το αντίστοιχο python script (χρήση Diffchecker).



Τελικά συμπεράσματα

Μετά και την επαλήθευση της σωστής λειτουργίας του επιταχυντή, το κύκλωμα έχει πλέον τη τελική του μορφή. Συγκρίνοντας τα αποτελέσματα της απόδοσης του κατά την αρχική μη βελτιστοποιημένη του μορφή, με αυτά της τελικής μορφής και της χρήσης των directives, πετύχαμε τα εξής:

- μείωση **57%** στο συνολικό latency του κυκλώματος
- μείωση **13.5%** στο iteration latency του εξωτερικού for-loop-(iteration over rows)
- αύξηση μόλις **3%** του utilization στα Lookup tables

Υλοποίηση Ενσωματωμένου Συστήματος με χρήση Verilog και Arduino

Το σενάριο είναι ότι, μπορούμε να δεχθούμε έως max_clients. Το total_clients δηλώνει πόσοι πελάτες έχουν έρθει συνολικά να εξυπηρετηθούν πχ μέσα στη μέρα, ανεξαρτήτως αν έχουν εξυπηρετηθεί ή όχι. Το current_client είναι ο πελάτης που εξυπηρετείται εκείνη την στιγμή, και το full δηλώνει ότι έχουμε φτάσει τους max_clients που μπορούμε να εξυπηρετήσουμε.
Όσο το total_clients είναι μικρότερο του max_clients μπορούμε να δεχθούμε πελάτες άρα μπορεί να πατηθεί το κουμπί NEW_CLIENT.
Όταν πατάμε το κουμπί NEW_CLIENT οι πελάτες μπαίνουν σε ένα εικονικό waiting list (δεν έχουμε φτιάξει κάποιο queue) και περιμένουν μέχρι να εξυπηρετηθούν.
Αν το total_clients γίνει ίσο με το max_clients δεν μπορούμε να δεχτούμε νέους πελάτες για να εξυπηρετηθούν, και ανάβουμε το led φωτάκι προς ενημέρωσή τους.
Όσο το current_client είναι μικρότερο του total_clients, μπορούμε να πατήσουμε το κουμπί DONE που σημαίνει ότι έχουμε εξυπηρετήσει τον προηγούμενο πελάτη και έχουμε πάρει καινούργιο πελάτη να εξυπηρετηθεί.
Αν το current_client γίνει ίσο με το total_clients σημαίνει ότι έχουμε εξυπηρετήσει όσους πελάτες περίμεναν στο waiting list και πρέπει να έρθουν καινούργιοι, αν μπορούμε να τους εξυπηρετήσουμε.
Σε οποιαδήποτε φάση, μπορούμε να πατήσουμε το RESET το οποίο θα κάνει initialize τους counters και θα σβήσει το φωτάκι, σαν να είναι η αρχή μιας νέας μέρας, όπου μπορούμε να ξεκινήσουμε την διαδικασία και πάλι από την αρχή.

- **Verilog**

```
module ClientService(

  input wire btnNew,
  input wire btnDone,
  input wire btnReset,

  output reg [7:0] current_client = 0,
  output reg [7:0] total_clients = 0,

  output reg full = 1'b0
);

  parameter max_clients = 10;
```

Τα wires αναπαριστούν τα κουμπιά του συστήματος (NEW_CLIENT, DONE, RESET).
Οι 8-bit registers αναπαριστούν τους counters (current_client , total_clients).
Ο 1-bit register full χρησιμοποιείται σαν bool μεταβλητή όπου το 1'b0 δηλώνει το false και το 1'b1 δηλώνει το true.
Το parameter χρησιμοποιείται σαν const και δηλώνει τον μέγιστο αριθμό των πελατών που μπορούμε να εξυπηρετήσουμε.

```
module ClientService_tb;

  reg btnNew, btnDone, btnReset;
  wire [7:0] current_client, total_clients;
  wire full;

  // Instantiate the module under test
  ClientService dut (
    .btnNew(btnNew),
    .btnDone(btnDone),
    .btnReset(btnReset),
    .current_client(current_client),
    .total_clients(total_clients),
    .full(full)
  );

  // Clock generation
  reg clk;
  always #5 clk = ~clk;

  // Stimulus generation
  initial begin
    clk = 0;
    btnNew = 0;
    btnDone = 0;
    btnReset = 0;
```

Στο testbench, δηλώνουμε τους registers και τα wires που χρειαζόμαστε για τα buttons και τους counters.
Δηλώνουμε έναν register που θα αναπαριστά το clock signal και δηλώνουμε ότι το state του clock θα πρέπει να αλλάζει κάθε 5 sec και να συγχρονίζονται τα processes.

Όταν πατάμε το btnNew, το total_clients αυξάνεται μόνο αν δεν έχουμε ξεπεράσει τους max_clients. Αν έχουμε φτάσει τον μέγιστο αριθμό πελατών που μπορούμε να εξυπηρετήσουμε βγαίνει ενημερωτικό μήνυμα και ανάβει το φωτάκι.
Πριν από την εκτύπωση των μηνυμάτων που αναγράφουν τις τιμές των current_client και total_clients βάζουμε καθυστέρηση ώστε να προλάβουν να ενημερωθούν οι τιμές.

```
always @(posedge btnNew) begin

  $display("NEW_CLIENT button pressed..");

  if (total_clients < max_clients) begin
    total_clients <= total_clients + 1;
  end
  else begin
    $display("Reached maximum amount of clients, come back tomorrow!");
  end

  if (total_clients == max_clients  && full == 1'b0) begin
    full <= 1'b1;
    $display("-----------------------------------");
    $display("The light is turned on!");
    $display("Reached maximum amount of clients, come back tomorrow!");
    $display("-----------------------------------");
  end

  #1;
  $display("Current client: %d", current_client);
  $display("Total clients: %d", total_clients);
  $display("-----------------------------------");

end
```

```
// Press the "New" button 2 times
repeat (2) begin
  #10 btnNew = 1;
  #10 btnNew = 0;
end
```

```
NEW_CLIENT button pressed..
Current client:   0
Total clients:   1
-----------------------------------
NEW_CLIENT button pressed..
Current client:   0
Total clients:   2
```

```
-----------------------------------
NEW_CLIENT button pressed..
Current client:   7
Total clients:  10
-----------------------------------
NEW_CLIENT button pressed..
Reached maximum amount of clients, come back tomorrow!

The light is turned on!
Reached maximum amount of clients, come back tomorrow!
-----------------------------------
```

Παραπάνω απεικονίζεται το σενάριο όπου έχουμε φτάσει τους max_clients, άρα δεν μπορούμε να εξυπηρετήσουμε άλλους πελάτες, έχουμε εξυπηρετήσει όλους τους clients που περιμένουν και συνεχίζουμε να πατάμε το done. Σε αυτή την περίπτωση βγαίνουν μηνύματα στον χρήστη για να ενημερωθεί για την κατάσταση και δεν ενημερώνεται κάποιος counter.

Όταν πατάμε το btnDone, το current_client αυξάνεται μόνο αν δεν έχουμε ξεπεράσει τους total_clients. Αν έχουμε φτάσει τους total_clients σημαίνει ότι δεν περιμένουν άλλοι πελάτες να εξυπηρετηθούν και βγαίνει αντίστοιχο μήνυμα ενημέρωσης.
Πριν από την εκτύπωση των μηνυμάτων που αναγράφουν τις τιμές των current_client και total_clients βάζουμε καθυστέρηση ώστε να προλάβουν να ενημερωθούν οι τιμές.

```
always @(posedge btnDone) begin

  $display("DONE button pressed..");

  if (current_client + 1 > total_clients) begin
    $display("No client waiting in line!");
  end
  else begin
    current_client <= current_client + 1;
  end

  #1;
  $display("Current client: %d", current_client);
  $display("Total clients: %d", total_clients);
  $display("-----------------------------------");

end
```

```
// Press the "Done" button 3 times
repeat (3) begin
  #10 btnDone = 1;
  #10 btnDone = 0;
end
```

```
-----------------------------------
DONE button pressed..
Current client:   1
Total clients:   2
-----------------------------------
DONE button pressed..
Current client:   2
Total clients:   2
-----------------------------------
DONE button pressed..
No client waiting in line!
Current client:   2
Total clients:   2
-----------------------------------
```

```
-----------------------------------
DONE button pressed..
Current client:   9
Total clients:  10
-----------------------------------
DONE button pressed..
Current client:  10
Total clients:  10
-----------------------------------
DONE button pressed..
No client waiting in line!
Current client:  10
Total clients:  10
-----------------------------------
```

Παρακάτω φαίνεται η υλοποίηση και το αποτέλεσμα του να πατήσει κάποιος το btnReset. Με το Reset μηδενίζονται οι counters και σβήνει το λαμπάκι, στην περίπτωση που έχει ανοίξει. Και εδώ ενημερώνουμε τον χρήστη με τις τιμές των current_client και total_clients.

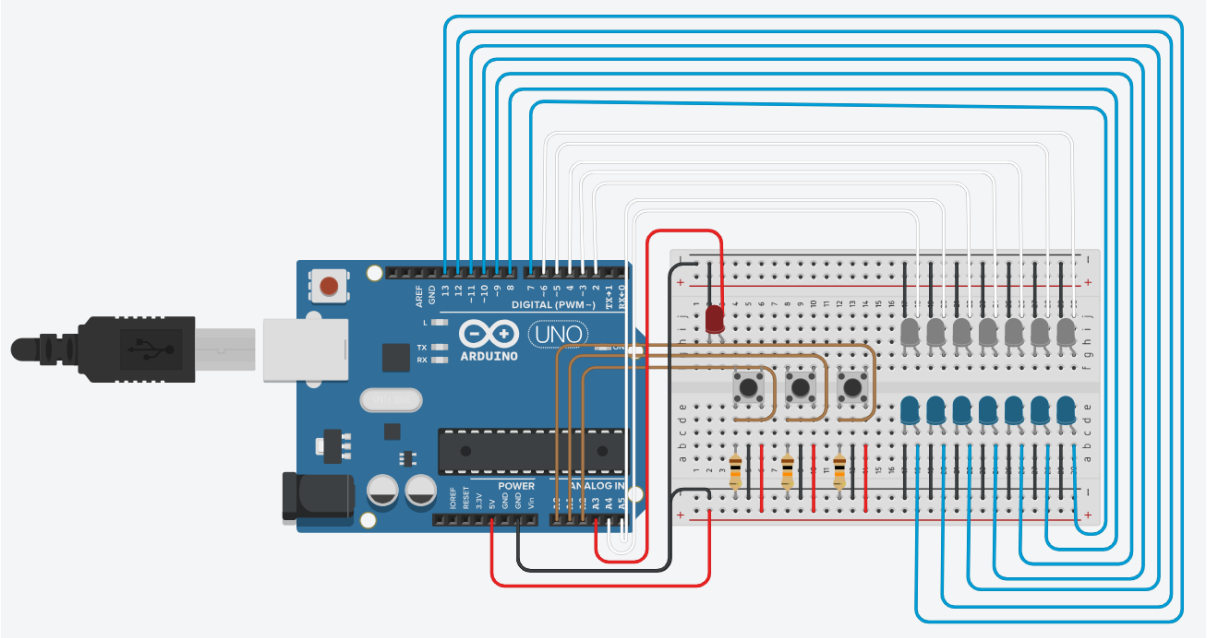
```
always @(posedge btnReset) begin

  $display("RESET button pressed..");

  current_client = 0;
  total_clients = 0;
  full = 1'b0;

  #1;
  $display("The light is turned off!");
  $display("Current client: %d", current_client);
  $display("Total clients: %d", total_clients);
  $display("-----------------------------------");

end
```

```
// Press the "Reset" button
#10 btnReset = 1;
#10 btnReset = 0;
```

```
-----------------------------------
RESET button pressed..
The light is turned off!
Current client:   0
Total clients:   0
-----------------------------------
```

- **Arduino**

Χρησιμοποιούμε το Arduino Uno R3 για να ενώσουμε και να ελέγχουμε τις υπόλοιπες συσκευές που θα χρησιμοποιήσουμε στο project. Επιπλέον, χρησιμοποιούμε ένα Breadboard ώστε να συνδέσουμε τα κουμπιά, τα καλώδια και τα led φωτάκια.
Στο Breadboard τοποθετούμε με κατάλληλο τρόπο τα buttons, τα led φωτάκια, τα καλώδια και τα resistors ώστε να επικοινωνούν σωστά με το πρόγραμμά μας.



Αρχικά, ορίζουμε τους αριθμούς των pins για τα κουμπιά και τα led φωτάκια. Τα buttonPinNew, buttonPinDone και buttonPinReset είναι αναλογικά pins που στην περίπτωσή μας τα χρησιμοποιούμε ως ψηφιακά, διότι δεν μας φτάνουν τα pins για τις ανάγκες του σεναρίου. Για τα led φωτάκια που αναπαριστούν τον αριθμό των total_clients και current_client χρησιμοποιούμε τα ψηφιακά pins, με εξαίρεση τα pins TX και RX τα οποία είναι καλό να μην χρησιμοποιούμε για διαφορετικό λόγο από σειριακή επικοινωνία. Έτσι, αποφασίσαμε να χρησιμοποιήσουμε δύο αναλογικά pins στη θέση τους. Τέλος, για το max_clients που αναπαριστά τον μέγιστο αριθμό πελατών που μπορούν να εξυπηρετηθούν πχ σε μια ημέρα.
Στα πλαίσια του demo, θέσαμε το max_clients με 10 αντί για 100 για πιο γρήγορα αποτελέσματα.

```
const int buttonPinNew = A2;
const int buttonPinDone = A1;
const int buttonPinReset = A0;

const int ledPin = A3;

const int totalClientsPins[] = {6, 5, 4, 3, 2, A5, A4};
const int currentClientPins[] = {7, 8, 9, 10, 11, 12, 13};

int buttonStateNew = 0;
int buttonStateDone = 0;
int buttonStateReset = 0;

const int max_clients = 10;

unsigned char current_client = 0;
unsigned char total_clients = 0;
bool full = false;
```

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);

  // Initialize LED pins as outputs
  for (int i = 0; i < 8; i++) {
    pinMode(totalClientsPins[i], OUTPUT);
  }
  for (int i = 0; i < 8; i++) {
    pinMode(currentClientPins[i], OUTPUT);
  }

  // initialize the pushbutton pins as an input:
  pinMode(buttonPinNew, INPUT);
  pinMode(buttonPinDone, INPUT);
  pinMode(buttonPinReset, INPUT);

  Serial.begin(9600);
}
```

Στο setup ορίζουμε τα inputs και τα outputs, και αρχικοποιούμε την σειριακή επικοινωνία.

```
void loop() {
  // read the state of the pushbutton value:
  buttonStateNew = digitalRead(buttonPinNew);
  buttonStateDone = digitalRead(buttonPinDone);
  buttonStateReset = digitalRead(buttonPinReset);

  if (buttonStateNew == HIGH)
  {
    Serial.println("NEW_CLIENT button pressed..");
    NewClientState();
  }
  else if (buttonStateDone == HIGH)
  {
    Serial.println("DONE button pressed..");
    ClientIsDoneState();
  }
  else if (buttonStateReset == HIGH)
  {
    Serial.println("RESET button pressed..");
    ResetState();
  }
}
```

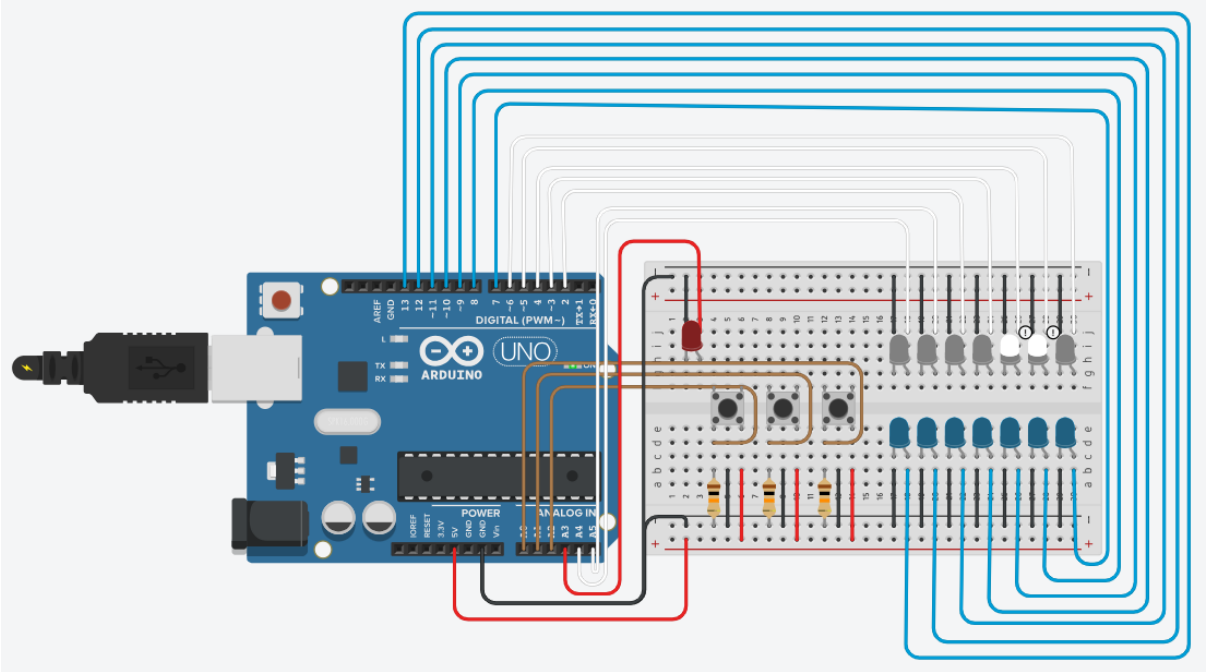
Στο loop διαβάζεται συνέχεια το state των buttons. Αν η τιμή είναι LOW το κουμπί δεν πατιέται εκείνη τη στιγμή, αν η τιμή είναι HIGH σημαίνει ότι μόλις πατήσαμε το κουμπί.

Όποτε πατιέται ένα από τα κουμπιά, ανεξαρτήτως αν οι μετρητές total_clients και current_client θα αυξηθούν ή όχι, εκτυπώνονται οι τιμές τους για να γνωρίζουμε τις τιμές τους πάντα. Σε αυτό το πλαίσιο, μετατρέπουμε τις τιμές τους σε binary μορφή και ελέγχουμε αν συμφωνεί το κάθε bit με το αντίστοιχό του pin. Αν όχι, ανάβουμε και σβήνουμε τα φωτάκια όπως πρέπει για να είναι σωστή η αναπαράσταση του αριθμού, διαβάζοντας από δεξιά προς τα αριστερά. Τα άσπρα led αναπαριστούν τους total_clients και τα μπλε τον current_client.

```
void PrintClients(){

  SetTotalClientsLEDs();
  SetCurrentClientLEDs();

  Serial.println("-----------------------------------");
  Serial.print("Current client: ");
  Serial.println(current_client);

  Serial.print("Total clients: ");
  Serial.println(total_clients);
  Serial.println("-----------------------------------");
}

void SetTotalClientsLEDs(){

  for (int i = 0; i < 8; i++) {
    // Turn on or off the LED based on the bit value
    bool bit = (total_clients >> i) & 1;
    if (bit == 1 && totalClientsPins[i] != HIGH) {
        digitalWrite(totalClientsPins[i], HIGH);  // Turn on the LED
    }
    else if (bit == 0 && totalClientsPins[i] != LOW) {
        digitalWrite(totalClientsPins[i], LOW);   // Turn off the LED
    }
  }

}

void SetCurrentClientLEDs(){

  for (int i = 0; i < 8; i++) {
    // Turn on or off the LED based on the bit value
    bool bit = (current_client >> i) & 1;
    if (bit == 1 && currentClientPins[i] != HIGH) {
        digitalWrite(currentClientPins[i], HIGH);  // Turn on the LED
    }
    else if (bit == 0 && currentClientPins[i] != LOW) {
        digitalWrite(currentClientPins[i], LOW);   // Turn off the LED
    }
  }

}
```

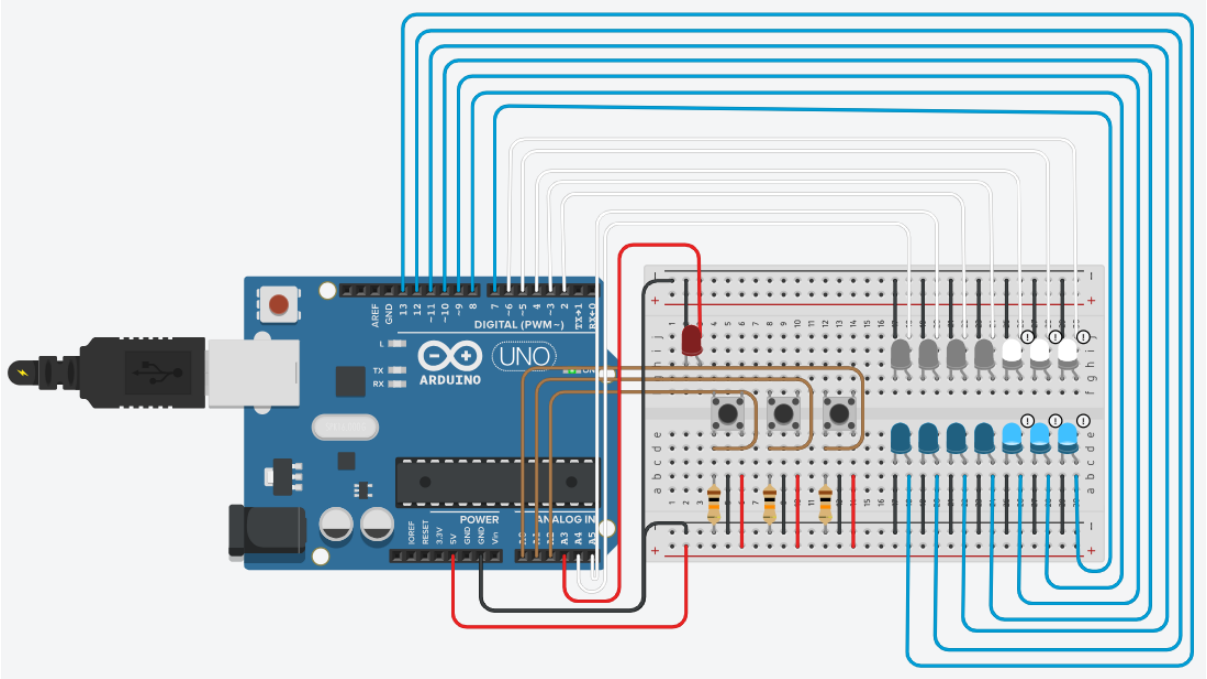
```
NEW_CLIENT button pressed..
New client is waiting in line..
-----------------------------------
Current client: 0
Total clients: 4
-----------------------------------
NEW_CLIENT button pressed..
New client is waiting in line..
-----------------------------------
Current client: 0
Total clients: 5
-----------------------------------
NEW_CLIENT button pressed..
New client is waiting in line..
-----------------------------------
Current client: 0
Total clients: 6
```

Όταν πατάμε το κουμπί NEW_CLIENT, ενημερώνουμε ότι περιμένει νέος πελάτης στη σειρά και στην συνέχεια εκτυπώνουμε ποιος πελάτης έχει εξυπηρετηθεί και πόσοι πελάτες έχουν έρθει γενικότερα για να εξυπηρετηθούν.
Αριστερά βλέπουμε ότι έχει πατηθεί το κουμπί 6 φορές και δεν έχει εξυπηρετηθεί κανείς, άρα υπάρχουν στην αναμονή 6 άτομα.



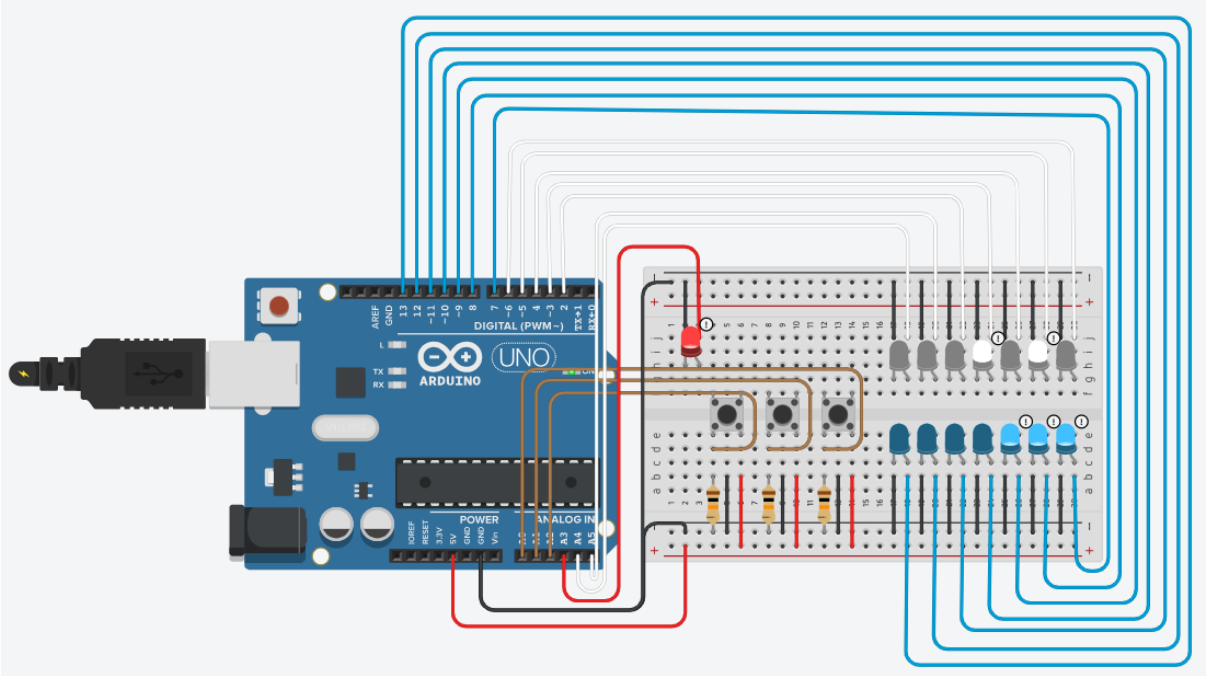
```
------------------------------------
Current client: 4
Total clients: 7
------------------------------------
DONE button pressed..
Client is done, getting next one..
------------------------------------
Current client: 5
Total clients: 7
------------------------------------
DONE button pressed..
Client is done, getting next one..
------------------------------------
Current client: 6
Total clients: 7
------------------------------------
DONE button pressed..
Client is done, getting next one..
------------------------------------
Current client: 7
Total clients: 7
------------------------------------
DONE button pressed..
Client is done, getting next one..
No client waiting in line!
------------------------------------
Current client: 7
Total clients: 7
------------------------------------
```

Πατάμε το κουμπί DONE, για να δείξουμε ποιος πελάτης μόλις εξυπηρετηθηκε. Στην συνέχεια, εκτυπώνουμε ποιος πελάτης έχει εξυπηρετηθεί και πόσοι πελάτες έχουν έρθει γενικότερα για να εξυπηρετηθούν.
Στην περίπτωση που πατήσουμε DONE και δεν περιμένει κάποιος άλλος πελάτης βγαίνει ενημερωτικό μήνυμα. Αυτό συμβαίνει όταν ισχύει το current_client+1 > total_clients.



```
-----------------------------------
Current client: 7
Total clients: 9
-----------------------------------
NEW_CLIENT button pressed..
New client is waiting in line..
-----------------------------------
Current client: 7
Total clients: 10
-----------------------------------
NEW_CLIENT button pressed..
New client is waiting in line..
Reached maximum amount of clients, come back tomorrow!
-----------------------------------
Current client: 7
Total clients: 10
-----------------------------------
NEW_CLIENT button pressed..
New client is waiting in line..
Reached maximum amount of clients, come back tomorrow!
-----------------------------------
Current client: 7
Total clients: 10
-----------------------------------
```

Αν πατήσουμε το κουμπί NEW_CLIENT περισσότερες από 100 φορές (10 για το demo) βγαίνει ενημερωτικό μήνυμα ότι δεν μπορούμε να δεχτούμε περισσότερους πελάτες.
Επιπλέον, όταν φτάνουμε το max των πελατών που μπορούμε να εξυπηρετήσουμε ανάβει και ένα led φωτάκι, προς ενημέρωσή μας.



```
-----------------------------------
Current client: 9
Total clients: 10
-----------------------------------
DONE button pressed..
Client is done, getting next one..
-----------------------------------
Current client: 10
Total clients: 10
-----------------------------------
DONE button pressed..
Client is done, getting next one..
No client waiting in line!
-----------------------------------
Current client: 10
Total clients: 10
-----------------------------------
```

Το κουμπί DONE μπορούμε να το πατάμε μέχρι να φτάσουμε τους total_clients, δεν εξαρτάται από το max_clients οπως το NEW_CLIENT.

```
Current client: 10
Total clients: 10
-----------------------------------
DONE button pressed..
Client is done, getting next one..
No client waiting in line!
-----------------------------------
Current client: 10
Total clients: 10
-----------------------------------
RESET button pressed..
Resetting..
Reset is complete..
-----------------------------------
Current client: 0
Total clients: 0
-----------------------------------
```

Όταν πατάμε το RESET, ανεξαρτήτως αν έχει φτάσει το max_clients ή όχι, μηδενίζει τα current_client και total_clients.
Επίσης, σβήνει το led φωτάκι.