TP: Neon Lab

Sonosy Omar, Hegab Habiba & Attia Sarah

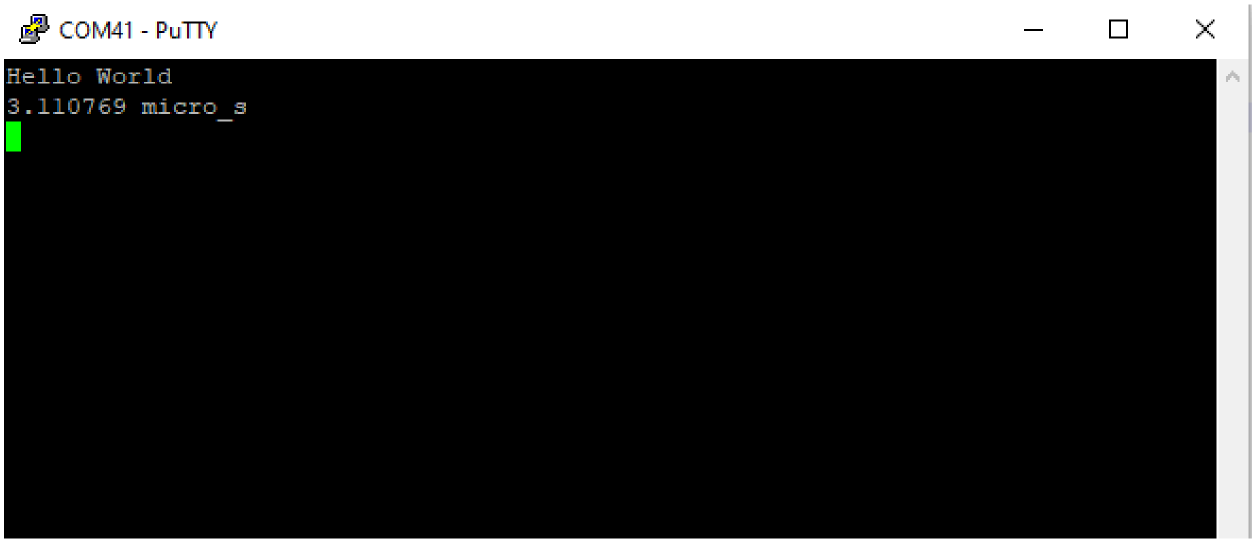
• Introduction:

In this lab we will be working on a Zybo Z7 board in order to learn how to use the SIMD instructions introduced in ARMv7 architecture called NEON.

We will be developing baremetal codes to process some calculations that are performed on multiple data using both C code and NEON Intrinsics, and comparing the speed of the calculations using each of them.

• Ex.1- Standalone Application & Timing:

In this exercise we are learning how to develop, build and run a simple Hello World on the Zybo board and also how to calculate time consumed to execute the code and print it on the console using UART.



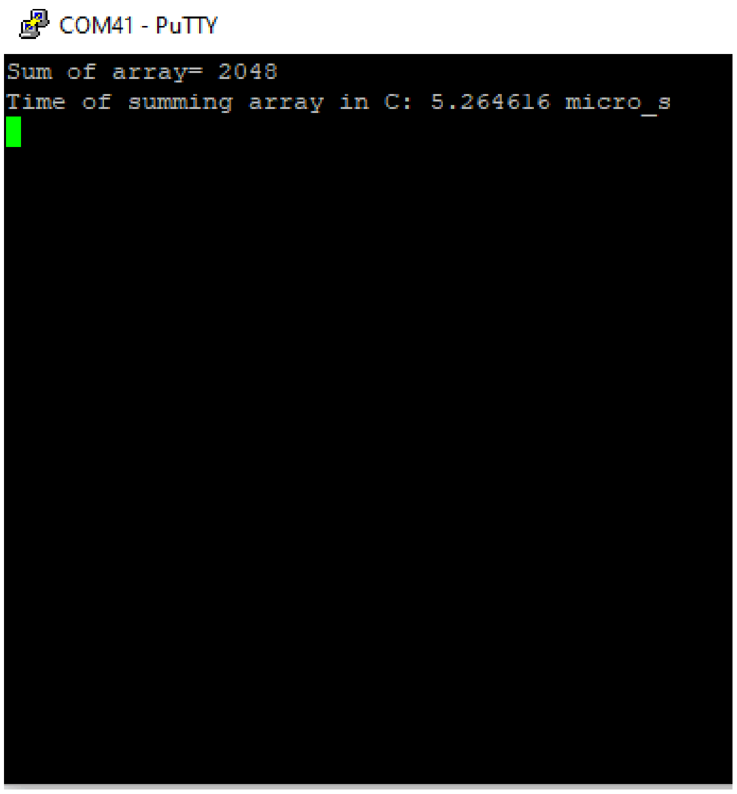
```
Hello World
3.110769 micro_s
```

• <u>Ex.2- Sum of an array:</u>

In this exercise we will be implementing a function that returns a sum of an array of 2048 elements of 16 bits each, firstly we implemented a C function that sum the array using a for loop element by element, find below the code and the result with timing:

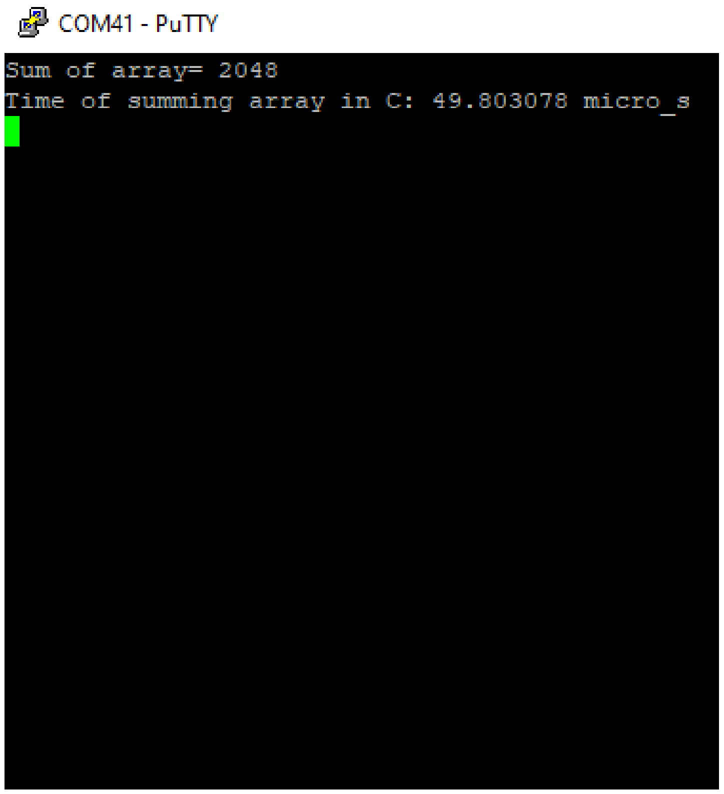
```c
int sum_c(int16_t* arr)
{
    int sum=0;
    for(int i = 0; i < 2048; i++)
    {
        sum+=(int)arr[i];
    }
    return sum;
}
```

With Optimization -O3



```
Sum of array= 2048
Time of summing array in C: 5.264616 micro_s
```

Without Optimization -O3

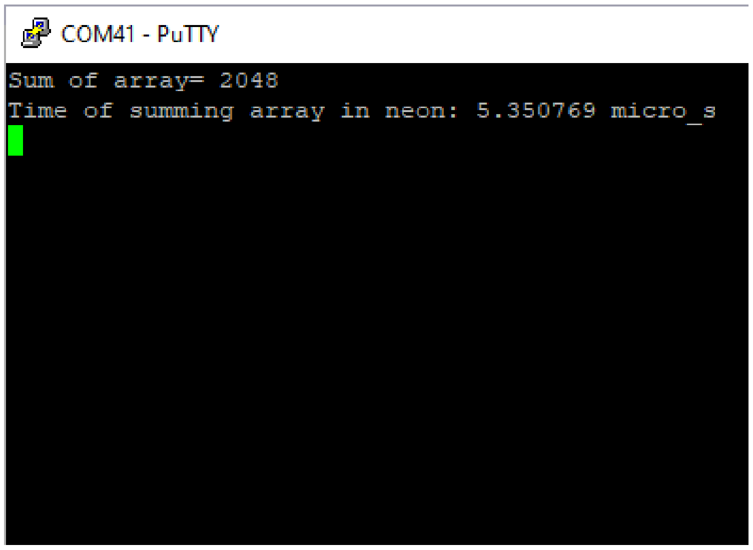


```
Sum of array= 2048
Time of summing array in C: 49.803078 micro_s
```

After that we implemented another function using Neon Intrinsics in order to sum 8 elements at the same step, therefore it should increase the time efficiency, find below the function implemented and and the result with timing:

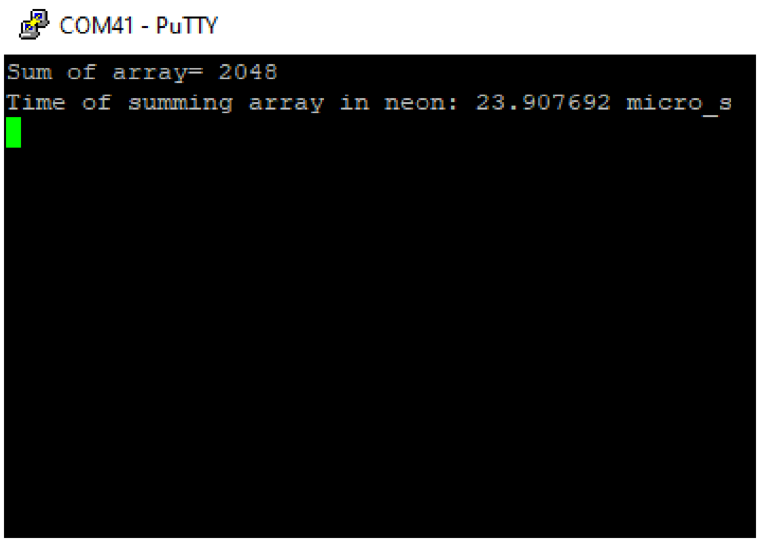
```
int sum_ni(int16_t* arr)
{
    int16x8_t v1,vsum,vtemp;
    vsum= vdupq_n_s16(0);
    for(int i=0; i<2048 ; i=i+8){
         v1=vld1q_s16(&arr[i]);
         vsum = vaddq_s16(vsum, v1);
    }
    int sum=0;
    sum=vgetq_lane_s16(vsum,0);
    sum+=vgetq_lane_s16(vsum,1);
    sum+=vgetq_lane_s16(vsum,2);
    sum+=vgetq_lane_s16(vsum,3);
    sum+=vgetq_lane_s16(vsum,4);
    sum+=vgetq_lane_s16(vsum,5);
    sum+=vgetq_lane_s16(vsum,6);
    sum+=vgetq_lane_s16(vsum,7);
    return sum;
}
```

With Optimization -O3



```
Sum of array= 2048
Time of summing array in neon: 5.350769 micro_s
```

Without Optimization -O3



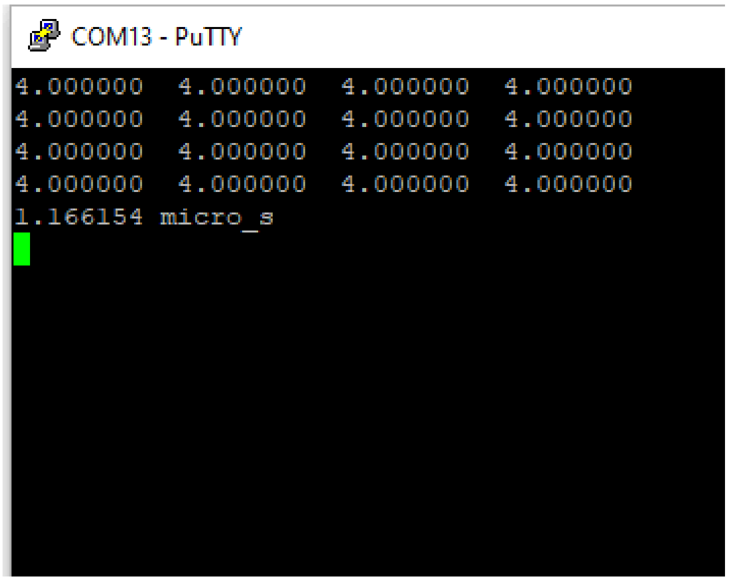
```
Sum of array= 2048
Time of summing array in neon: 23.907692 micro_s
```

• Ex.3- Matrix Multiplication:

In this exercise we want to implement a function that can do a matrix multiplication of two matrices of size 4x4 and type float, firstly we implemented a C function that does the multiplication using three nested for loops in order to calculate the multiplication for each cell, find below the code and the results with timing:

```
void mat_product_c(float A[4][4], float B[4][4], float C[4][4])
{
    //for loop to loop on rows
    for (int i=0; i<4; i++){
        //for loop to loop on columns
        for(int j=0; j<4;j++){
            //initializing each cell in output with zero
            C[i][j]= 0;
            //for loop to accumulate the value of the multiplication for each
cell
            for(int k=0;k<4;k++){
                C[i][j]+=A[i][k]*B[k][j];
            }
        }
    }
}
```

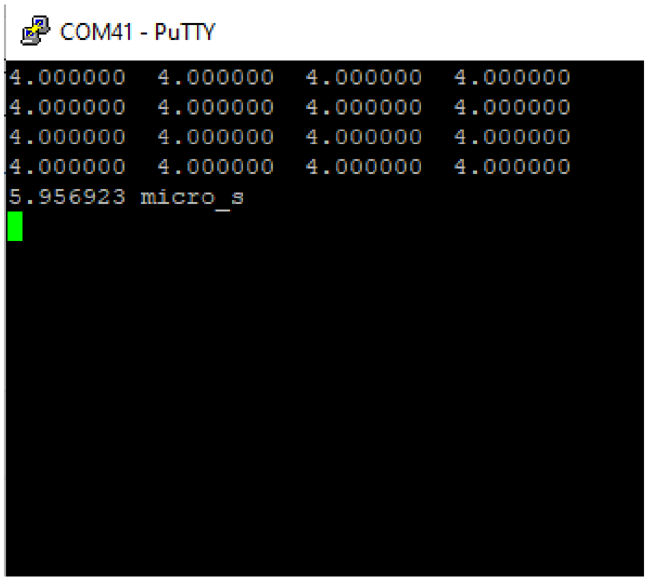
With Optimization -O3



COM13 - PuTTY

```
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
1.166154 micro_s
```

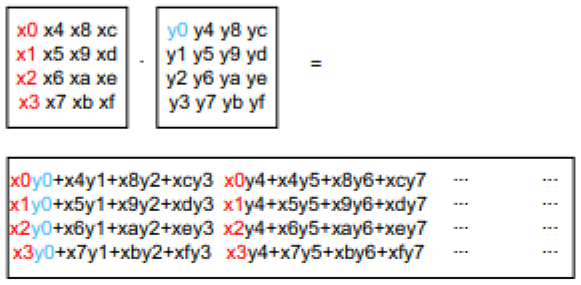
Without Optimization -O3



COM41 - PuTTY

```
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
5.956923 micro_s
```

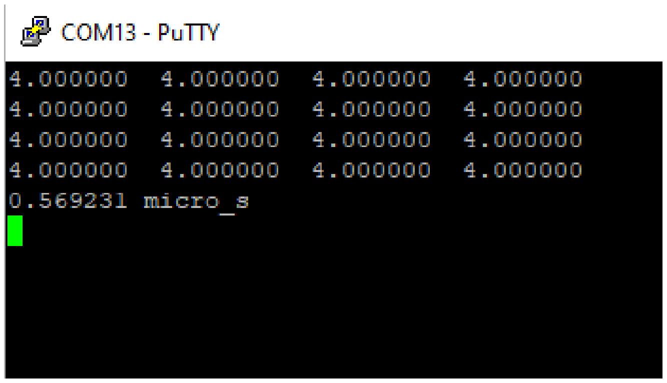
After we implemented another function that optimizes the operations using Neon Intrinsics, and the matrix multiplication algorithm presented in the following figure, please find the implementation and the result with timing below:



x0 x4 x8 xc
x1 x5 x9 xd
x2 x6 xa xe
x3 x7 xb xf
y0 y4 y8 yc
y1 y5 y9 yd
y2 y6 ya ye
y3 y7 yb yf
x0y0+x4y1+x8y2+xcy3 x0y4+x4y5+x8y6+xcy7
x1y0+x5y1+x9y2+xdy3 x1y4+x5y5+x9y6+xdy7
x2y0+x6y1+xay2+xey3 x2y4+x6y5+xay6+xey7
x3y0+x7y1+xby2+xfy3 x3y4+x7y5+xby6+xfy7

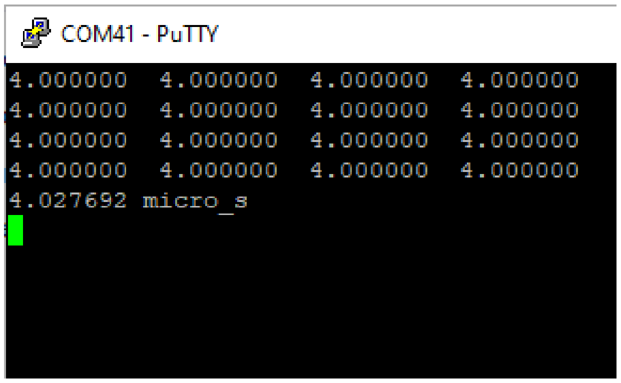
```
void mat_product_ni(float A[4][4], float B[4][4], float C[4][4])
{
    //Declaring the neon registers which will hold the matrices
    float32x4_t V_A[4],V_B[4],result_temp[6];
    float32x4x4_t V_B_4,V_A_4,result;
    //loading the matrices into 4 different registers, each one holding a row
    V_A_4=vld4q_f32(&A[0][0]);
    V_A[0]=V_A_4.val[0];
    V_A[1]=V_A_4.val[1];
    V_A[2]=V_A_4.val[2];
    V_A[3]=V_A_4.val[3];
    V_B_4=vld4q_f32(&B[0][0]);
    V_B[0]=V_B_4.val[0];
    V_B[1]=V_B_4.val[1];
    V_B[2]=V_B_4.val[2];
    V_B[3]=V_B_4.val[3];
    //one for loop to implement the algorithm of matrix multiplication
    for(int i=0;i<4;i++){
        result_temp[0]=vmulq_n_f32(V_A[0],vgetq_lane_f32(V_B[i],0));
        result_temp[1]=vmulq_n_f32(V_A[1],vgetq_lane_f32(V_B[i],1));
        result_temp[2]=vmulq_n_f32(V_A[2],vgetq_lane_f32(V_B[i],2));
        result_temp[3]=vmulq_n_f32(V_A[3],vgetq_lane_f32(V_B[i],3));
        result_temp[4]=vaddq_f32(result_temp[0],result_temp[1]);
        result_temp[5]=vaddq_f32(result_temp[2],result_temp[3]);
        result.val[i]=vaddq_f32(result_temp[4],result_temp[5]);
    }
    //storing the result in the output matrix
    vst4q_f32(&C[0][0],result);
}
```

With Optimization -O3



```
COM13 - PuTTY
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
0.569231 micro_s
```

Without Optimization -O3



```
COM41 - PuTTY
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.000000  4.000000  4.000000  4.000000
4.027692 micro_s
```

- ## <u>Ex.4- Edge Detection:</u>

In this exercise we would like to implement a function that can detect the edge in a 2D array of size 10x10 of 8-bits elements with an edge introduced inside, in a way that the upper half is filled with zeros and the lower half filled with 100, the edge will be detected by calculating this formula for each cell:

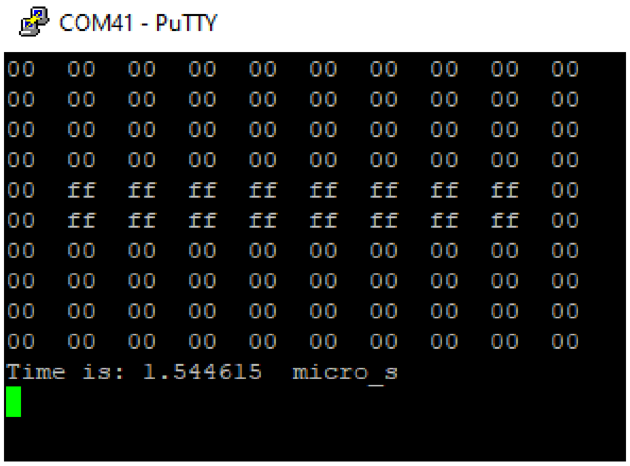
$|G| = |E - W| + |N - S|$ and an edge will be detected if G>threshold.

Firstly we implemented a C function to check for edge for each cell by looping on each cell:

```c
void edge(unsigned char arr[10][10], unsigned char arr2[10][10]){

    for (int i=1; i<9; i++){
        for (int j=1; j<9; j++){
            int G= abs(arr[i][j+1] - arr[i][j-1])+abs(arr[i+1][j]-arr[i-1][j]);
            if (G >= 100){
                arr2[i][j]= 255;
            }
            else{
                arr2[i][j]= 0;
            }
        }
    }
}
```

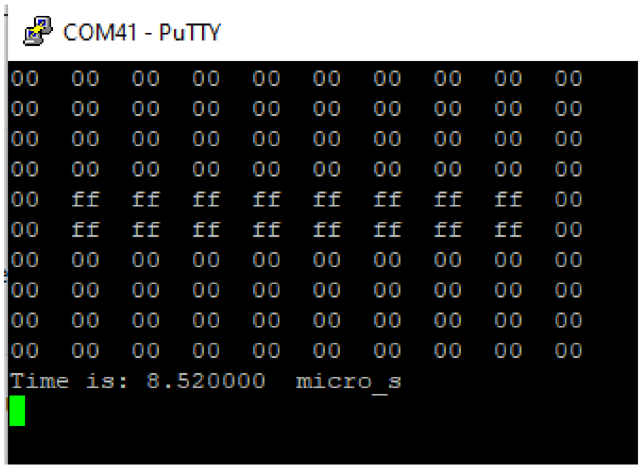
With Optimization -O3



COM41 - PuTTY

```
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  ff  ff  ff  ff  ff  ff  ff  ff  00
00  ff  ff  ff  ff  ff  ff  ff  ff  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
Time is: 1.544615  micro_s
```

Without Optimization -O3



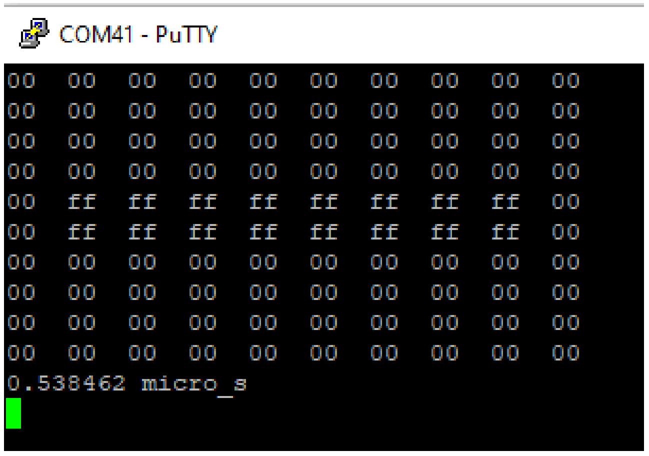
COM41 - PuTTY

```
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  ff  ff  ff  ff  ff  ff  ff  ff  00
00  ff  ff  ff  ff  ff  ff  ff  ff  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
Time is: 8.520000  micro_s
```

After that we implemented another function to detect edges using Neon Intrinsics that can detect the edge for full row at one time, please find the code implemented and the results with timing below:

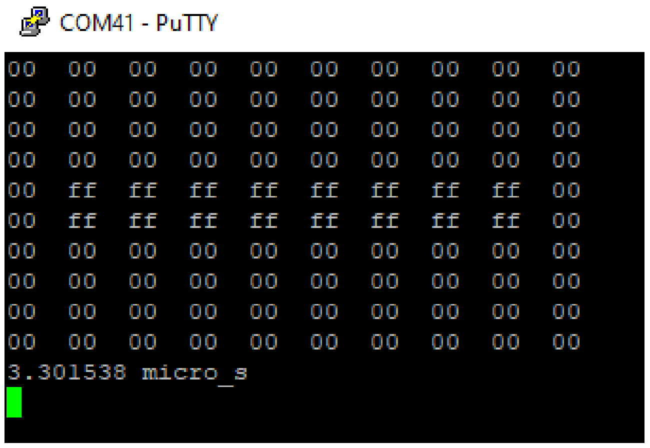
```c
void edge_neon(unsigned char arr[10][10], unsigned char arr2[10][10]){
    uint8x8_t N,S,W,E;
    uint8x8_t reg_out;
    uint8x8_t sub1,sub2;
    uint8x8_t sum;
    uint8x8_t threshold;
    threshold=vdup_n_u8(100);
    for(int i=1;i<9;i++){
        N= vld1_u8(&arr[i-1][1]);
        S= vld1_u8(&arr[i+1][1]);
        W= vld1_u8(&arr[i][0]);
        E= vld1_u8(&arr[i][2]);
        sub1=vabd_u8(N,S);
        sub2=vabd_u8(W,E);
        sum=vadd_u8(sub1,sub2);
        sum= vcge_u8(sum,threshold);
        vst1_u8(&arr2[i][1],sum);
    }
}
```

With Optimization -O3


COM41 - PuTTY

```
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  ff  ff  ff  ff  ff  ff  ff  ff  00
00  ff  ff  ff  ff  ff  ff  ff  ff  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
0.538462 micro_s
```

Without Optimization -O3


COM41 - PuTTY

```
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  ff  ff  ff  ff  ff  ff  ff  ff  00
00  ff  ff  ff  ff  ff  ff  ff  ff  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00
3.301538 micro_s
```