EEE404/591 – Real Time DSP – Lab 1
Introduction to CodeWarrior

- ***Hemanth Balaji Dandi***
#1213076538

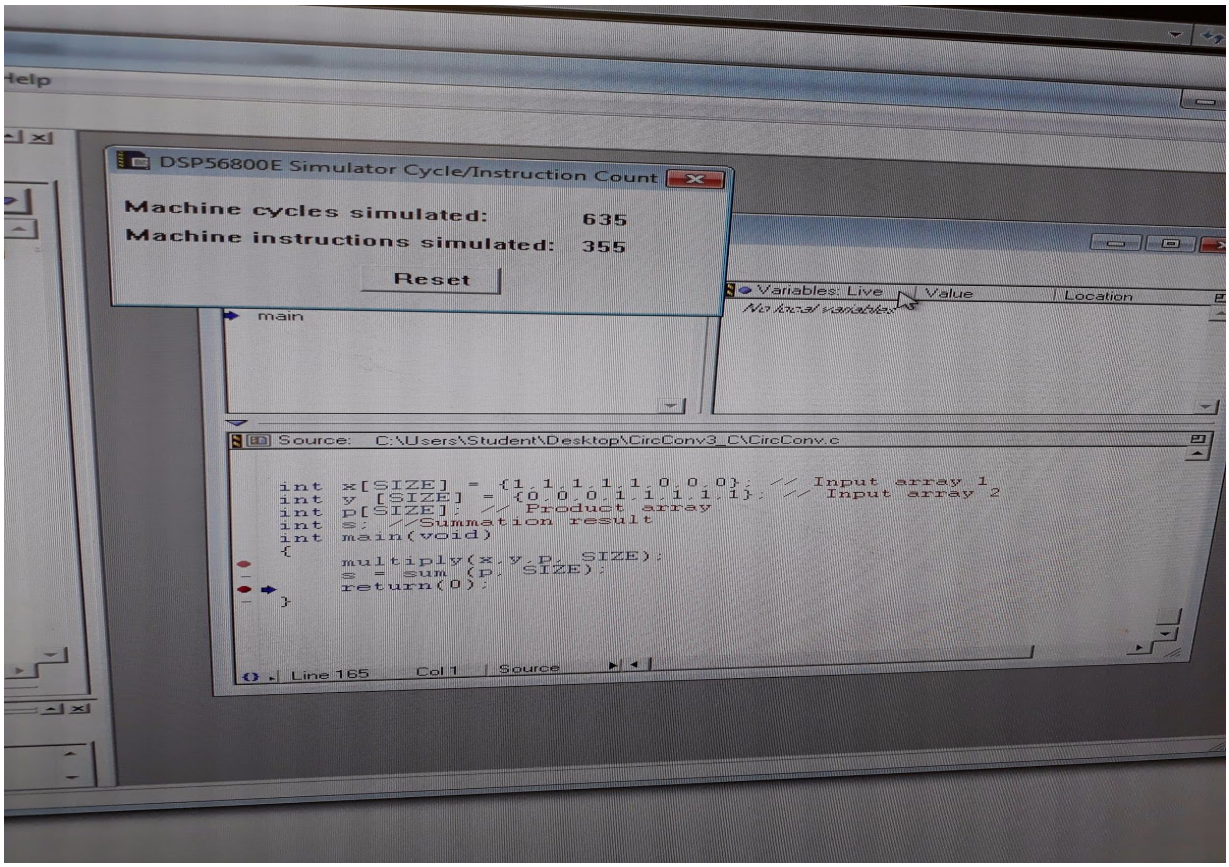
REPORT

Calculating the Machine Cycles & Instruction Counts for MAC Operation

MAC Code: Pure C (Only Main)

```
multiply(x,y,p, SIZE);
s = sum (p, SIZE);
```

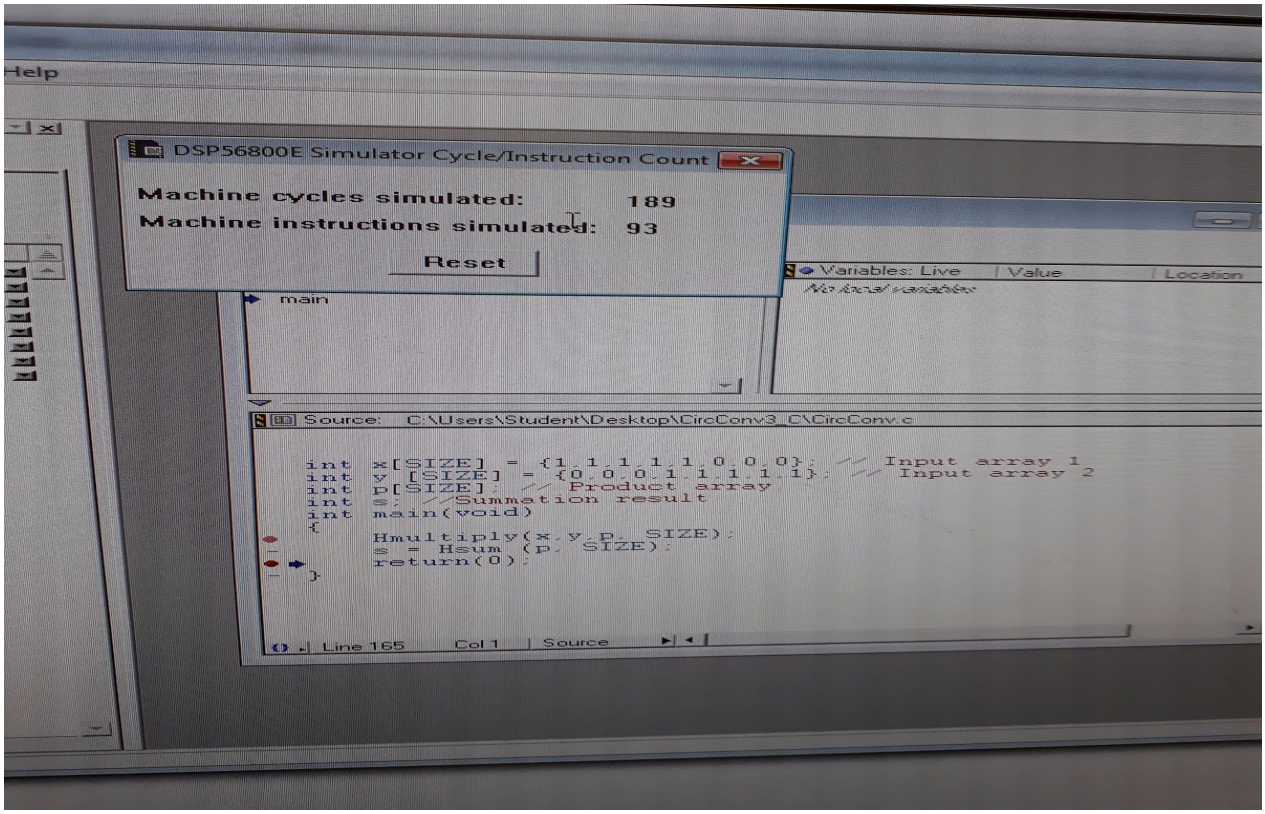
For Pure C:



MAC Code: Hybrid (Only Main)

```
Hmultiply(x,y,p, SIZE);
s = Hsum (p, SIZE);
```

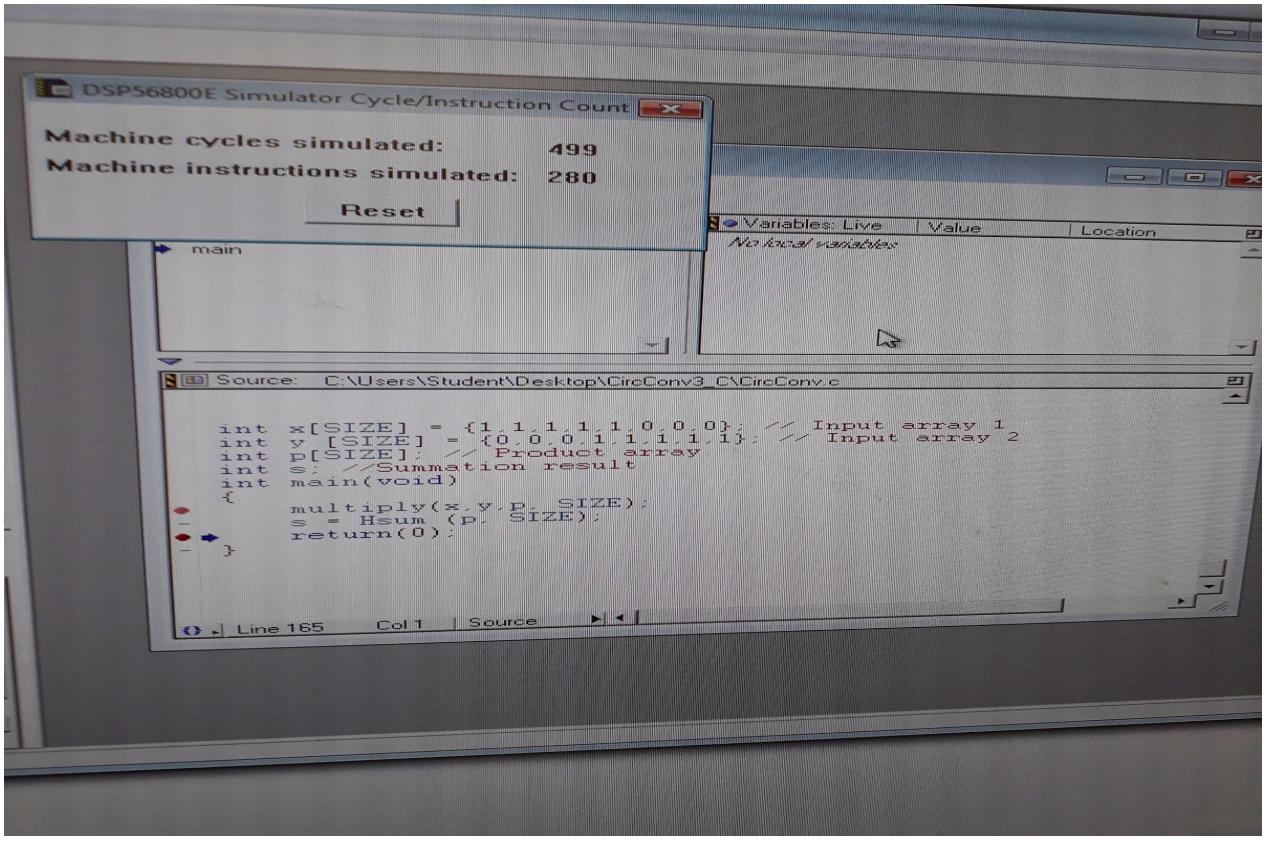
Hybrid : Hmultiply and Hsum



MAC Code: Hybrid (Only Main)

```
multiply(x,y,p, SIZE);
s = Hsum (p, SIZE);
```

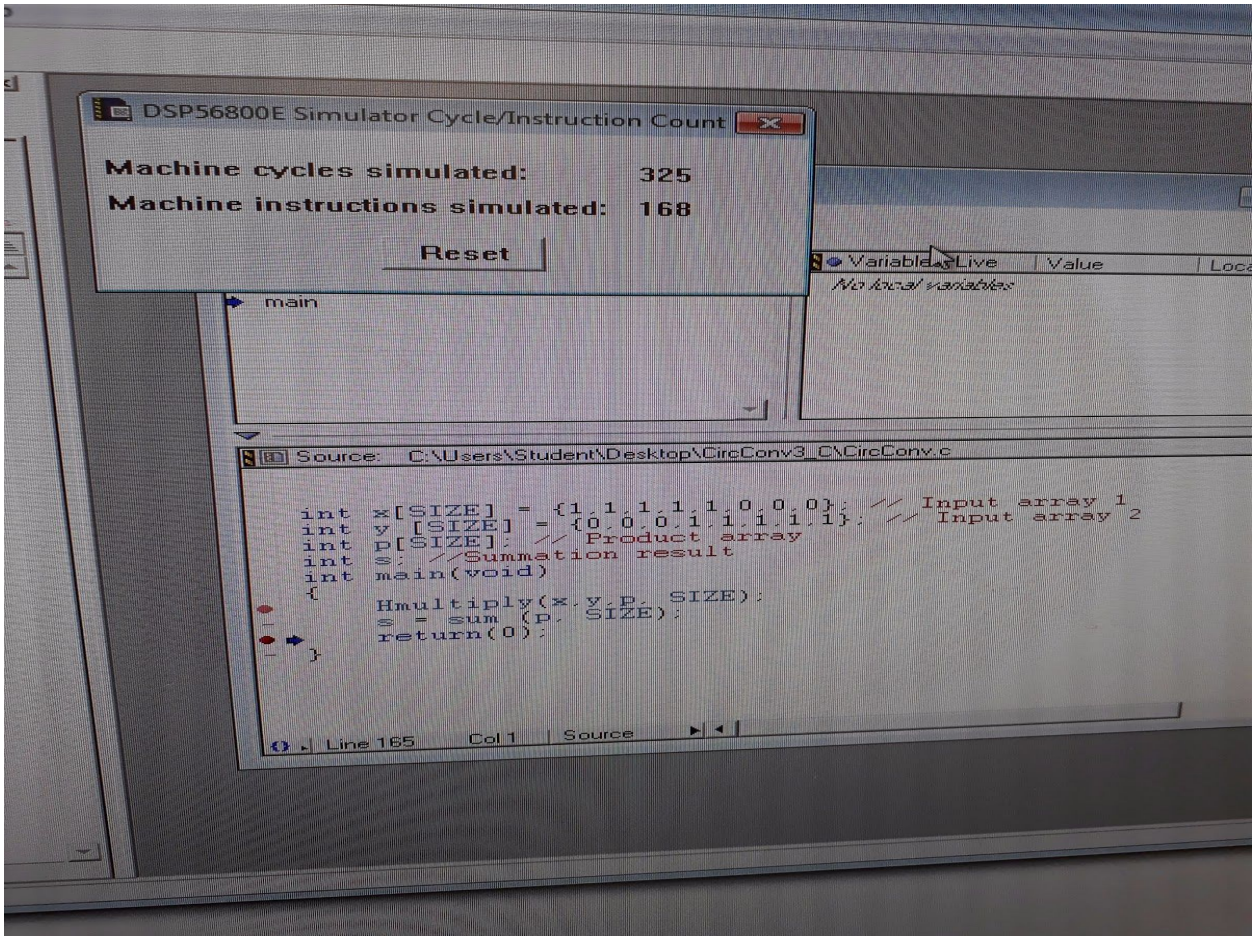
Hybrid : multiply and Hsum



MAC Code: Hybrid (Only Main)

```
Hmultiply(x,y,p, SIZE);
s = sum (p, SIZE);
```

Hybrid : Hmultiply and sum

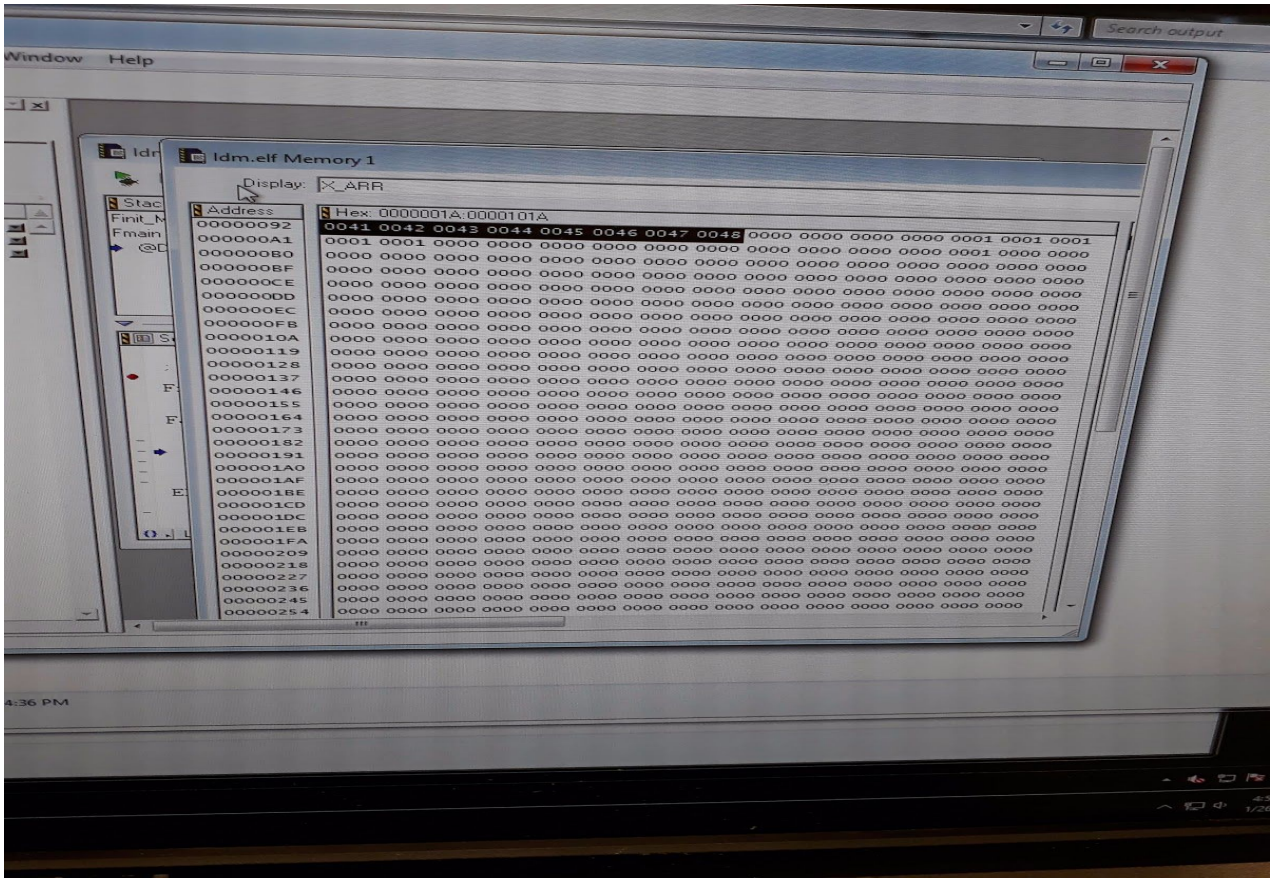


Pure Assembly

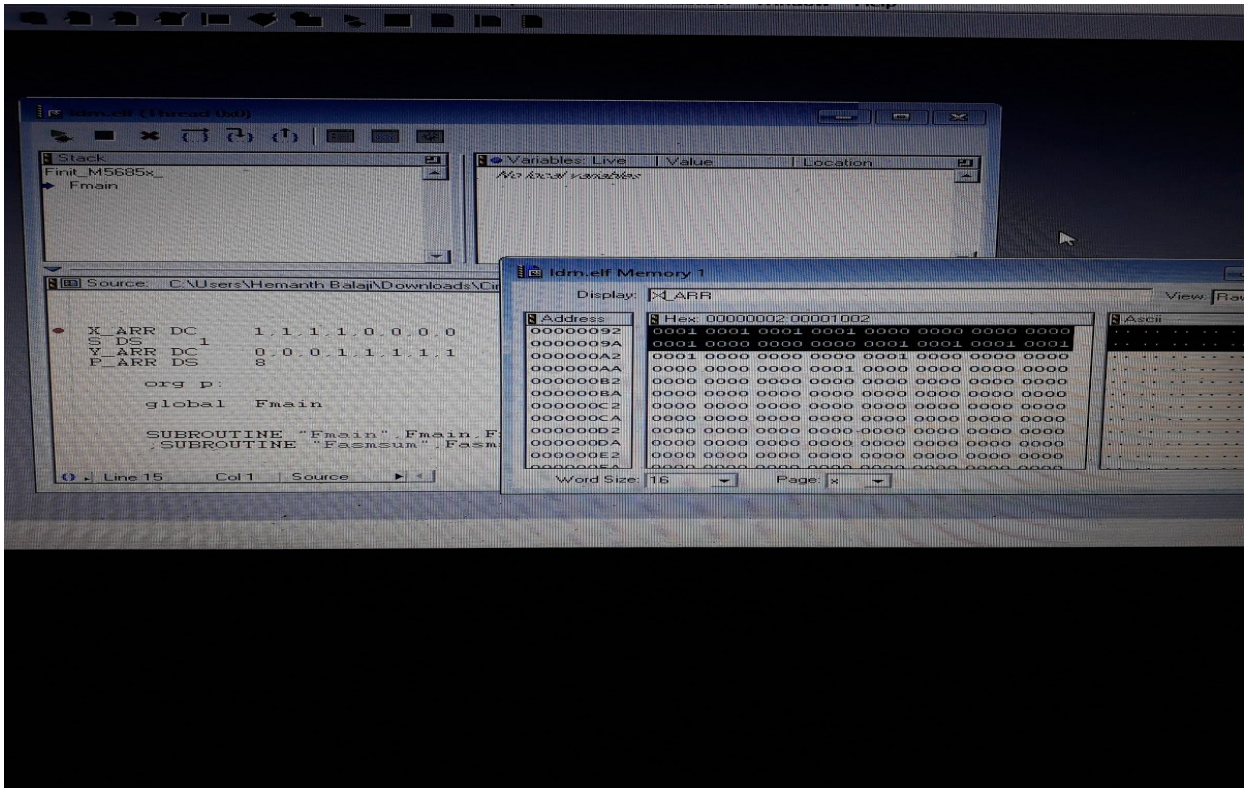
MAC Code:(Main)

```
MOVE.L  #P_ARR, R4
MOVE.L  #S, R5
MOVE.W  #LENGTH,Y0
MOVE.L  #Y_ARR, R2
MOVE.L  #X_ARR, R3
JSR  Fasmmultiply
MOVE.W #LENGTH,Y0
MOVE.L #P_ARR,R2
JSR  Fasmsum
MOVE.W Y0,X:(R5)
```

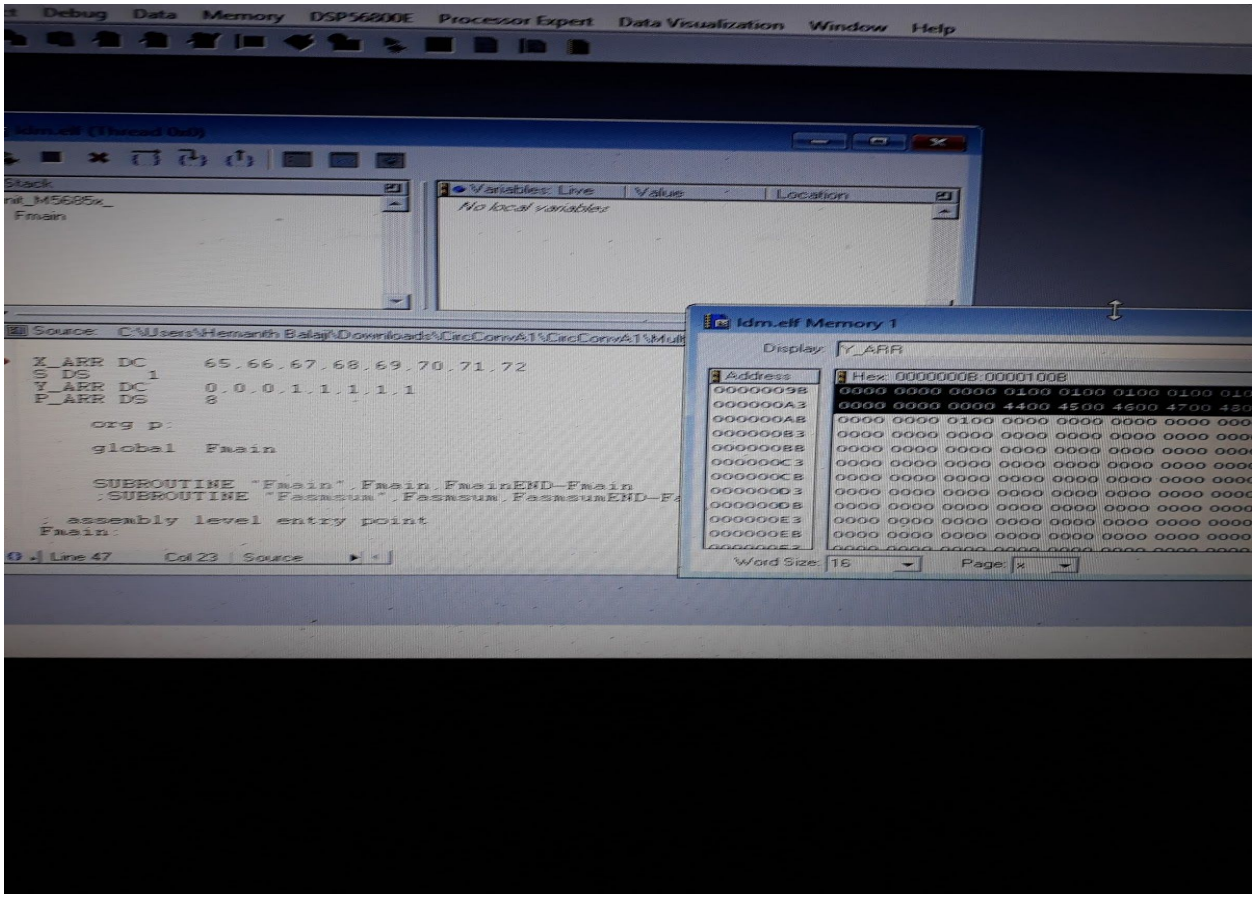
Data Memory: X_ARR= [65,66,67,68,69,70,71,72] (Below values are in ASCII)



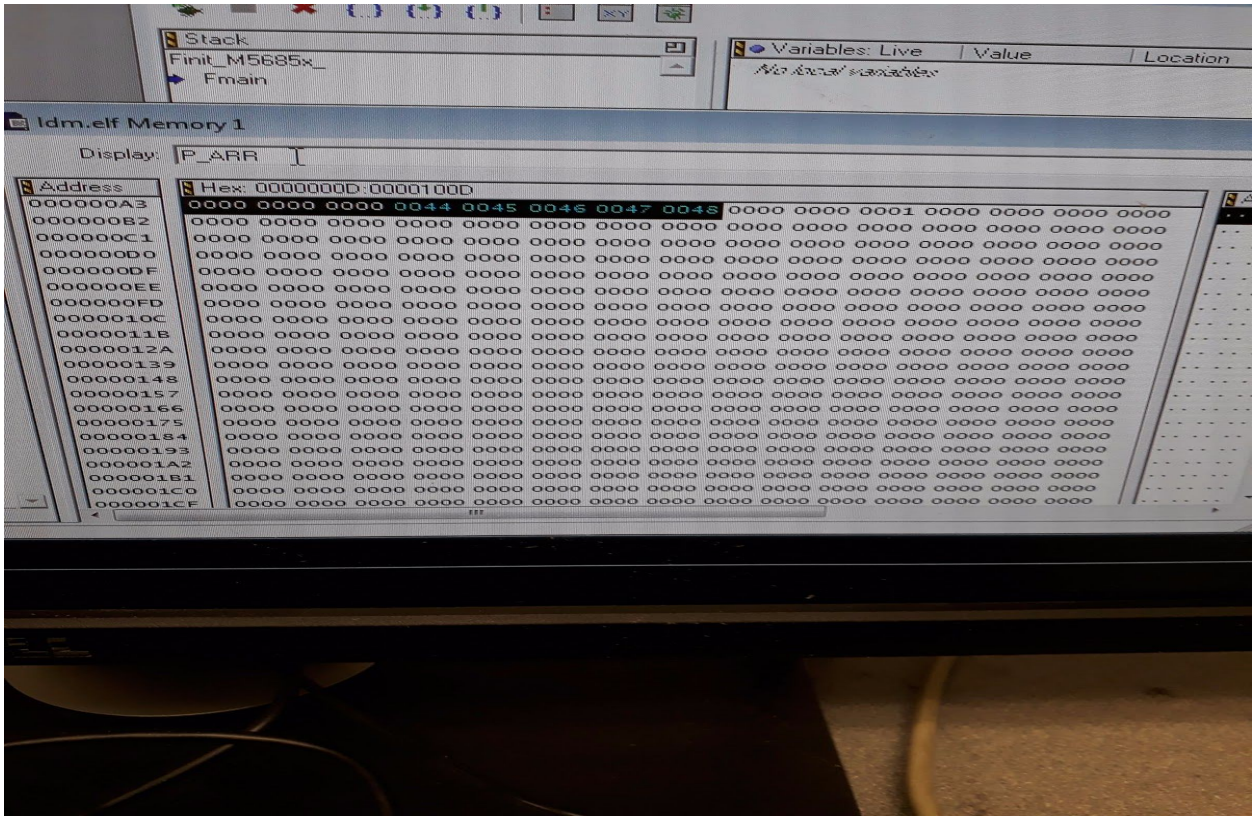
Data Memory: X_ARR=[1,1,1,1,0,0,0,0]



Data Memory: Y_ARR=[0,0,0,0,1,1,1,1]



Data Memory: P_ARR



Data Memory: S- (015E)

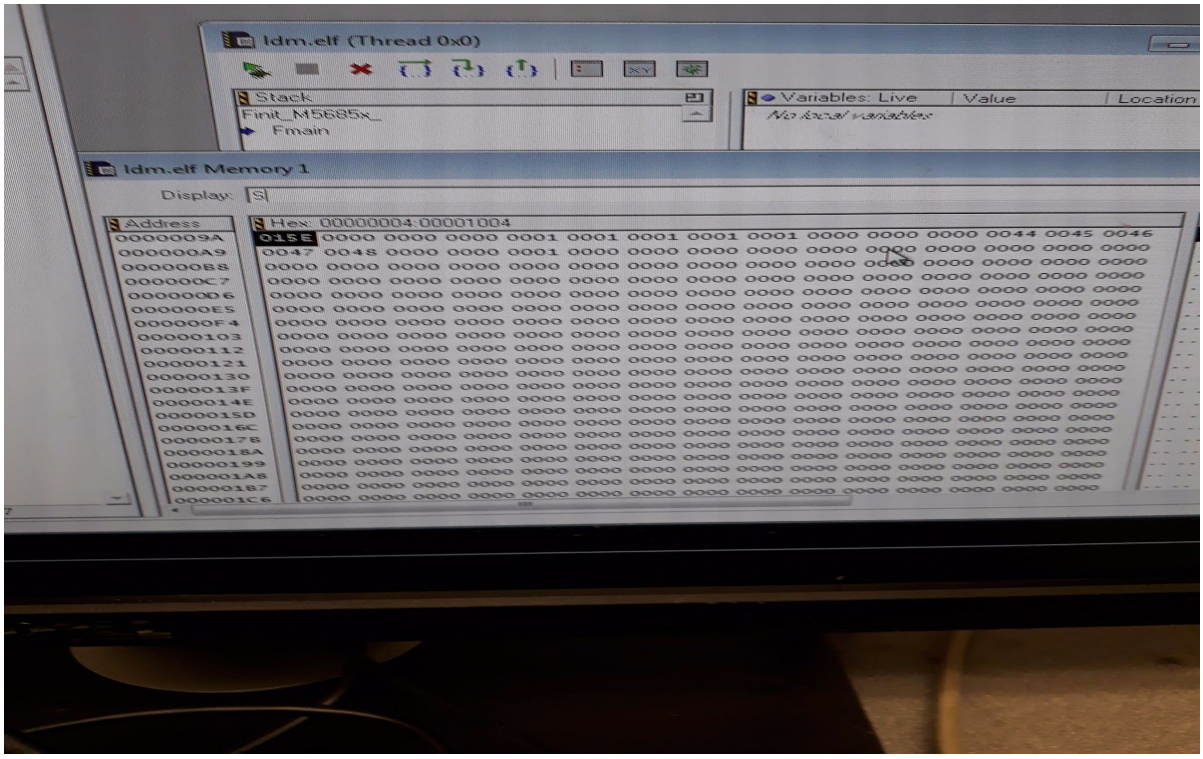
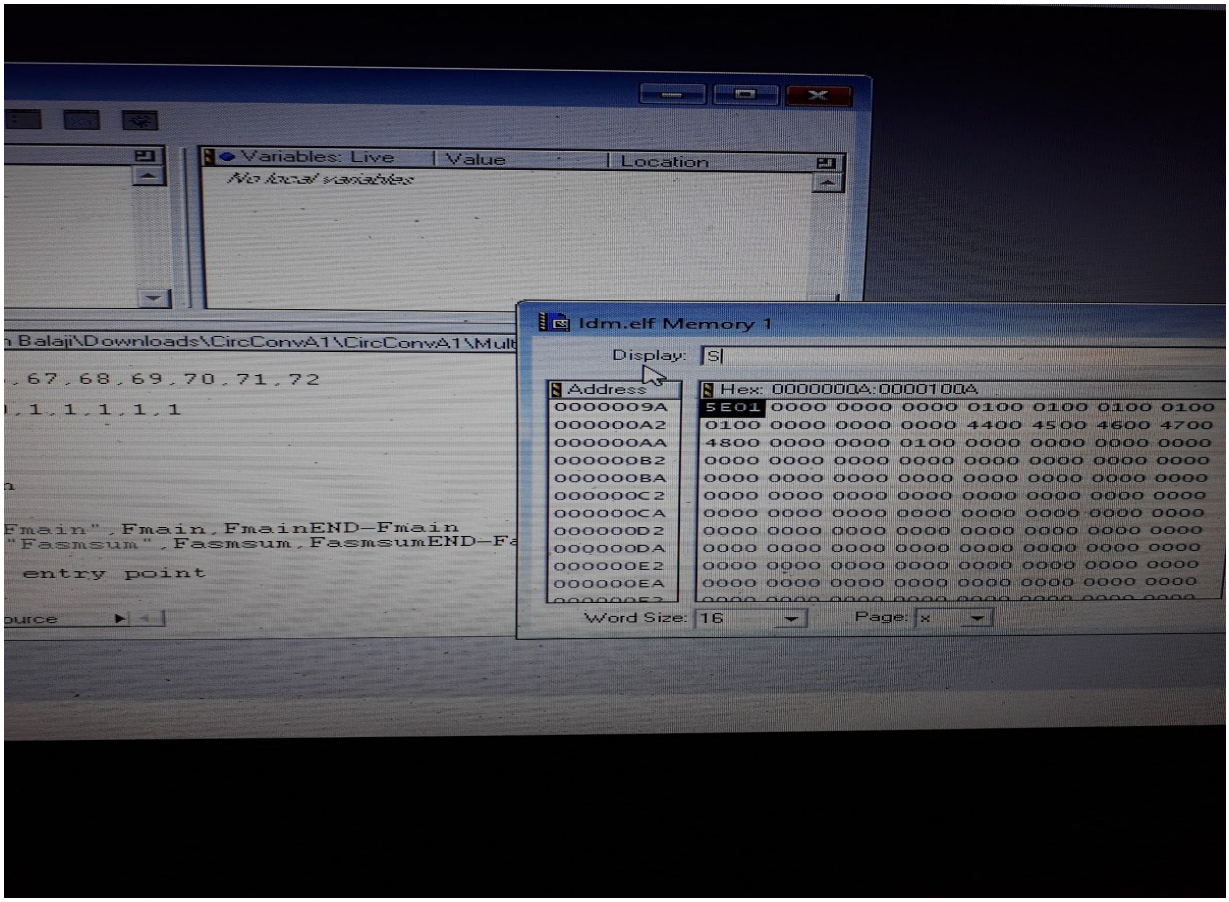


Table 1 : MAC Operation

	Pure C	Hybrid			Pure Assembly
		Hmultiply & Hsum	Multiply & Hsum	Hmultiply & sum	
Machine Cycles Simulated	635	189	499	325	197
Machine Instructions simulated	335	93	280	168	101
Size of ldm.elf Files (in bytes)	6169 bytes	6169 bytes			5927 bytes

Data Memory S: (After Memory>Swap Endian)- (5E01)



Inference

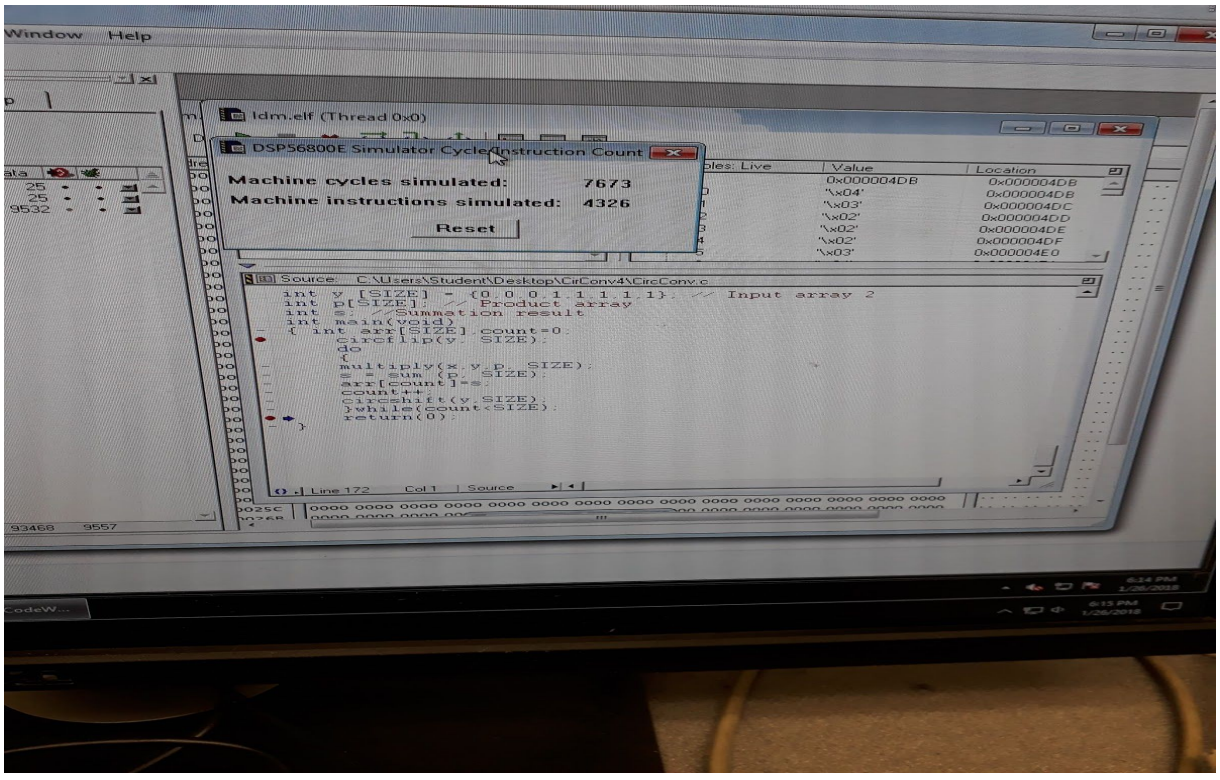
We see that the Hybrid- Hmultiply & Hsum code is more efficient as fewer machine cycles and instructions are executed comparatively than the others.
However, the Pure Assembly code consumes less memory, making it efficient is a way as well, thus there is a sort of trade-off between Machine Instructions & Cycles and Memory size with regards to efficiency.
Nevertheless, the Pure C code was simpler to code mainly because the language was known beforehand by us unlike Assembly, though I am suspecting that once we get a grip on the latter, it will turn out to be much simpler to code than C.

Machine Cycles & Instruction Count for Pure C, Hybrid & Pure Assembly Circular convolution on DSP 568500E

Code: (Pure C)-Main Function

```
int count=0; // Initializing to zero. This is used as the index for the final array, arr[SIZE}
circflip(y, SIZE); // Circular flip the y array and store it in the same array
do                  // Start a do-while loop
{
multiply(x,y,p, SIZE); // Dot product of each value of the x array and circular flipped  array y, and store in p array
 s = sum (p, SIZE); // Summation of all the values in p and store in a variable 's'
arr[count]=s; // Store the 's' variable value in an array, arr, corresponding to how many times 'arr' is being invoked-taken care by the 'count' variable.
count++; // Increment count, so that next time, 's' will be stored in that corresponding incremented count index in arr.
 circshift(y,SIZE); // The circular flipped array,y, is now circular shifted through mod operation
}while(count<SIZE); // Condition for the do-while loop- Breaks if count becomes greater than SIZE
```

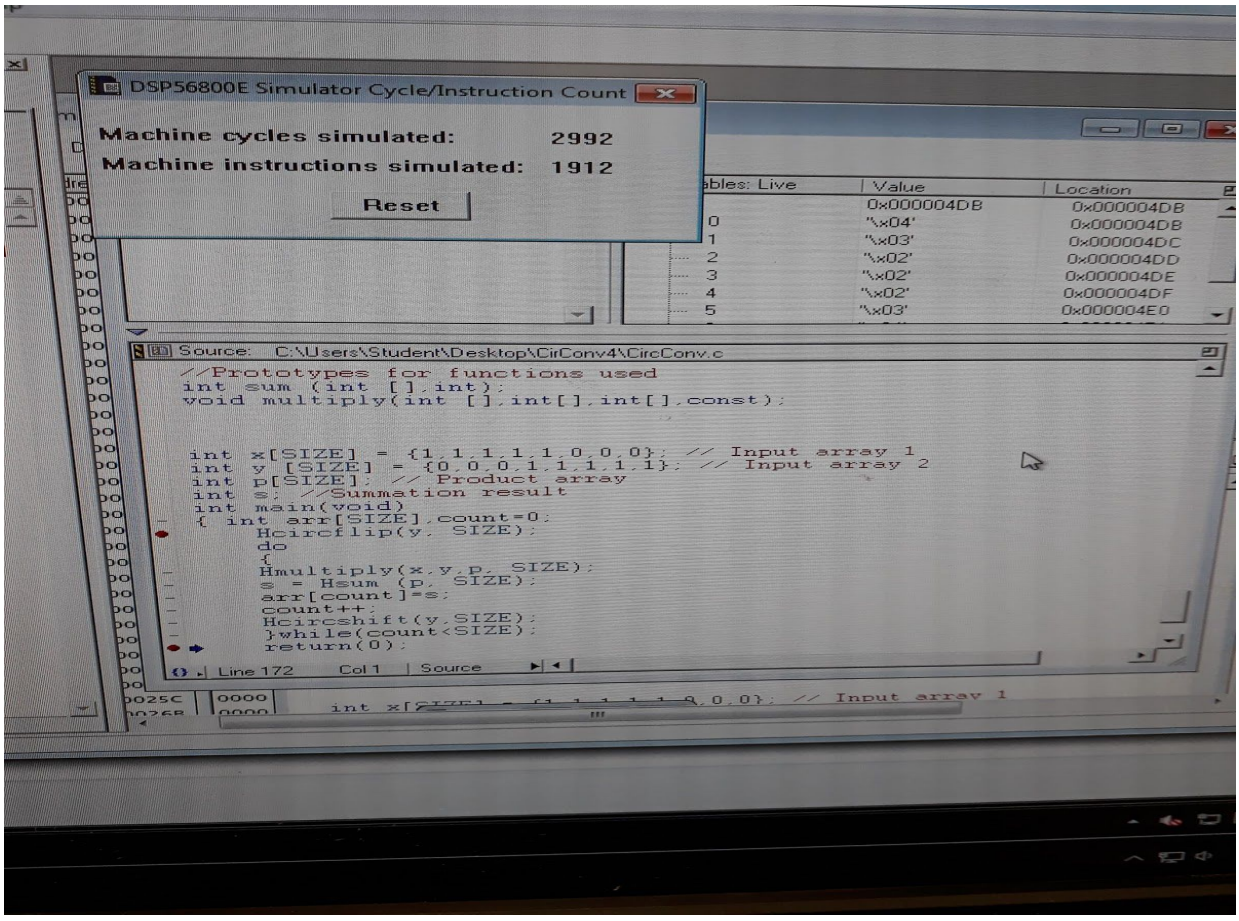
Pure C



Code: (Hybrid)-Main Function

int count=0; // Initializing to zero. This is used as the index for the final array, arr[SIZE}
Hcircflip(y, SIZE); // Circular flip the y array and store it in the same array
do // Start a do-while loop
{
Hmultiply(x,y,p, SIZE); // Dot product of each value of the x array and circular flipped array y, and store in p array
s = Hsum (p, SIZE); // Summation of all the values in p and store in a variable 's'
arr[count]=s; // Store the 's' variable value in an array, arr, corresponding to how many times 'arr' is being invoked-taken care by the 'count' variable.
count++; // Increment count, so that next time, 's' will be stored in that corresponding incremented count index in arr.
Hcircshift(y,SIZE); // The circular flipped array,y, is now circular shifted through mod operation
}while(count<SIZE); // Condition for the do-while loop- Breaks if count becomes greater than SIZE

Hybrid



Code: (PureAssembly)-Fmain Function

```
        MOVEU.W  #HX_BUF, R4 ; Move array pointer of X_BUF to Register 4
        MOVEU.W  #Y_BUF, R5  ; Move  array pointer of Y_BUF to Register 2
        MOVE.W #LENGTH,Y0   ; Move the Size (Length=8) to Y0
        MOVEU.W  #ARR_BUF, R2 ; Move  array pointer of ARR_BUF to Register 2
        JSR Fasmcircflip          ; Perform Circular shift on ARR_BUF by Jumping to Fasmcircshift routine
        DO #LENGTH,OUTER_LOOP  ; Perform a Loop for all the below functions for 'Length' times (8)
        MOVEU.W  #X_BUF, R3  ; Move  array pointer of X_BUF to Register 3
        MOVEU.W #ARR_BUF,R2
        MOVE.W #LENGTH,Y0
        MOVEU.W #HX_BUF, R4
        JSR Fasmmultiply          ; Perform array dot product multiplication of every value by Jumping to
Fasmmultiply routine
        MOVE.W #LENGTH,Y0
        MOVEU.W #ARR_BUF,R2
        JSR Fasmcircshift         ; Perform Circular shift on ARR_BUF by Jumping to Fasmcircshift routine
        MOVEU.W  #HX_BUF, R2
        MOVE.W #LENGTH,Y0
        JSR Fasmsum  ; Perform Circular shift by Jumping to Fasmsum routine
        MOVE.W Y0,X:(R5)+ ; Moving the final circular convolved result from the Y0 registerto internal
register R5, and increment R5 each time to hold each of the values
```

Pure Assembly

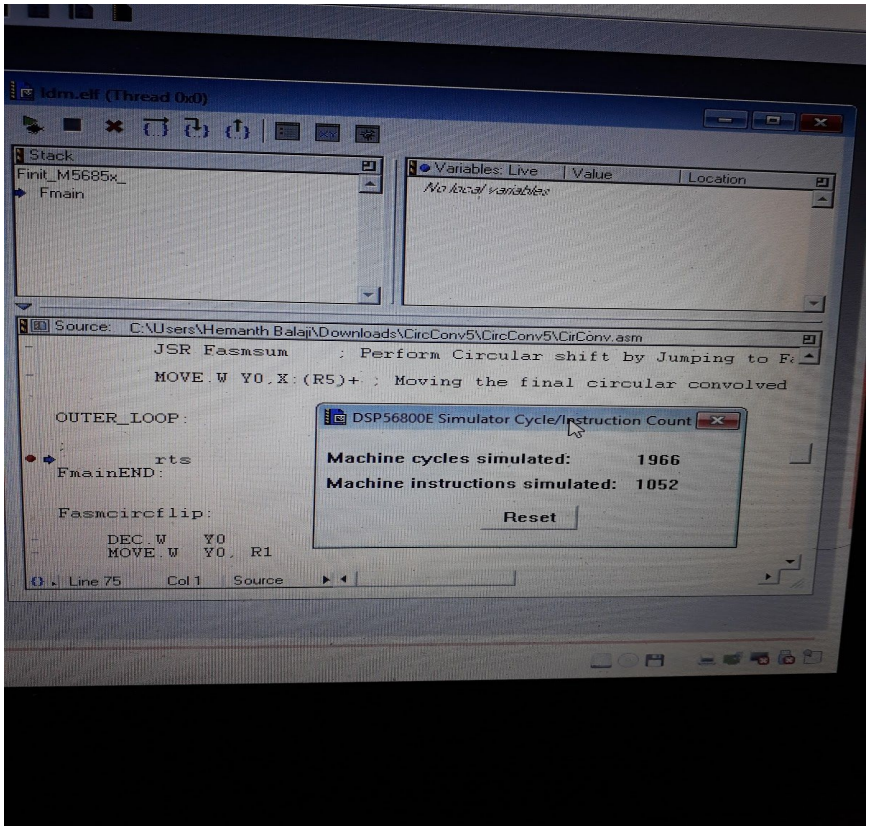


Table 2: Circular Convolution with 3 methods

	Pure C	**Hybrid**	**Pure Assembly**
Machine Cycles simulated	**7673**	**2992**	**1966**
Machine Instructions simulated	**4326**	**1912**	**1052**

Inference:

Therefore we see that Pure Assembly is the most efficient of the lot.