CA3 VHDL(System on Chip (SoC))

Amirhossein ilkhani
University of Tehran
Tehran, Iran
hossein.ilkhani1376@gmail.com

System on Chip is an integrated circuit that integrates multiple components including digital, analog, hardware, and software programs in a single chip. The main component in an SoC is a processor for handling different computational tasks within the system.

A. SOC

Our system includes the main processor (PUNEH), FIR, DMA controller, Arbiter, and Programmable interrupt controller. The processor fills the coefficient registers and then set the DMA for reading the related data and putting them in the buffer of the DMA. After this with an in fully interrupt signal processor realizes now the fir is ready to start the calculation with the data we buffered in DMA. When the calculation is finished and the calculated data was written in the buffers, the out full interrupt is set for interrupting PUNEH to save the data in the ram. After finishing this operation out_complete interrupts PUNEH to clear the configuration registers of DMA.

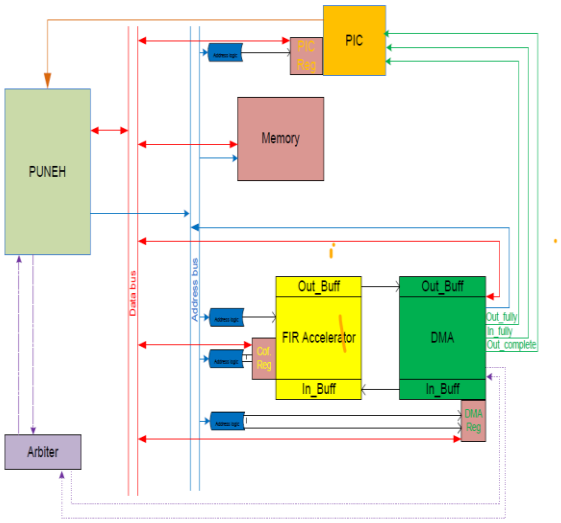
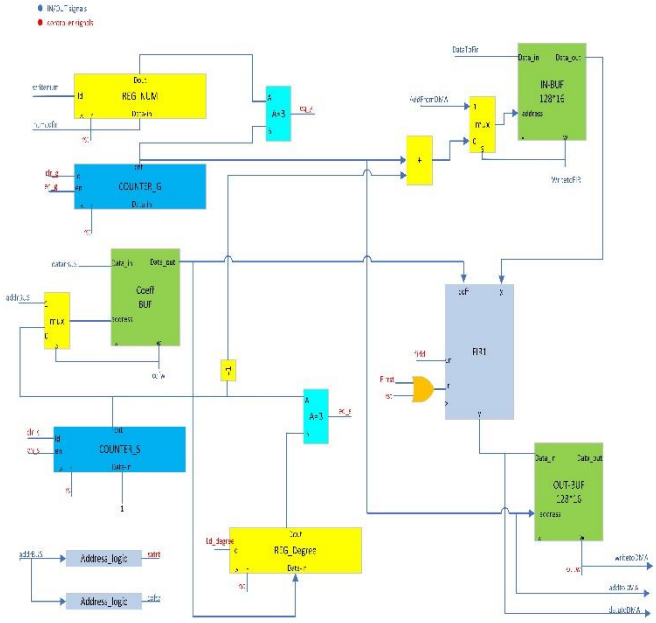


Figure 1

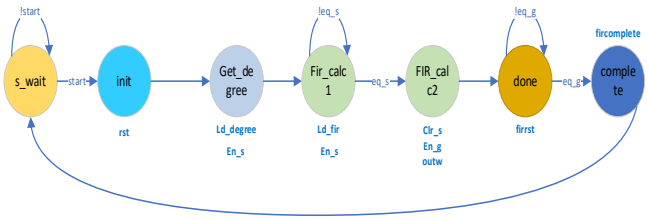
In the next parts, the design of the components will be described and illustrated for realizing their operations.

B. FIR

for this FIR filter, I use a 1 level fir filter consequentially to Degree, IN-BUF, OUT-BUFF, and other parts to reach our goal. the overview of our design and the main signal are illustrated here:



AND the controller for this DATA PATH is shown below:



And the top module of this fir has this ports:

```
ENTITY FIR_TOP IS PORT (
    clk : IN STD_LOGIC;
    r : IN STD_LOGIC;
    databus : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    addbus : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    --to DMA
    DataToFir : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    NUMToFir : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    AddFromDMA : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    writeNUM, writetoFIR : IN STD_LOGIC;
    WriteToDMA : OUT STD_LOGIC;
    DataToDMA : OUT std_logic_vector(15 DOWNTO 0);
    AddToDMA : OUT std_logic_vector(6 DOWNTO 0);
    FirCompelete : OUT STD_LOGIC
);
END ENTITY FIR_TOP;
```

C. PIC

a programmable interrupt controller (PIC) is an integrated circuit that helps a microprocessor (or CPU) handle interrupt requests (IRQ) coming from multiple different sources (like external I/O devices) which may occur simultaneously.[1] It helps prioritize IRQs so that the CPU switches execution to the most appropriate interrupt handler (ISR) after the PIC assesses the IRQ's relative priorities.

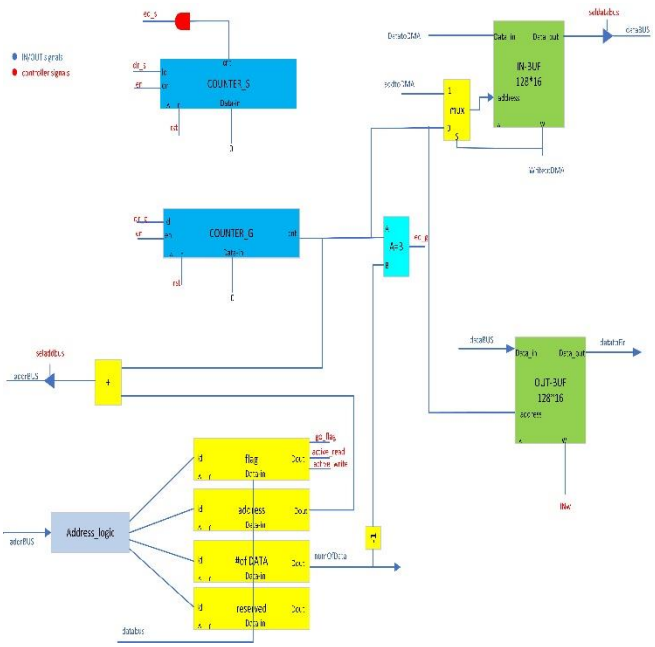
PIC must activate the interrupt of PUNEH and give the isr address of the related interrupt. The codes are shown below:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;

ENTITY PIC IS
    GENERIC(ADDLOGIC :  STD_LOGIC_VECTOR(15 DOWNTO 0):=x"0C05";
            IR1ISR :    STD_LOGIC_VECTOR(15 DOWNTO 0):=x"0900";
            IR2ISR :    STD_LOGIC_VECTOR(15 DOWNTO 0):=x"0910";
            IR3ISR :    STD_LOGIC_VECTOR(15 DOWNTO 0):=x"0920"
            );
    PORT(
    IR1,IR2,IR3 :IN STD_LOGIC;
    ADDRBUS :IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    DATABUS :OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
    INT :OUT STD_LOGIC
    );
END ENTITY;
ARCHITECTURE ARCH OF PIC IS
    TYPE MEMORY IS ARRAY(NATURAL RANGE<>) OF STD_LOGIC_VECTOR(15 DOWNTO 0);
    SIGNAL MEM:MEMORY(2 DOWNTO 0):=(IR3ISR,IR2ISR,IR1ISR);
BEGIN
    INT<='1' WHEN ((IR1 OR IR2 OR IR3)='1') ELSE '0';
    DATABUS<= MEM(0) WHEN (IR1='1' AND ADDRBUS=ADDLOGIC) ELSE MEM(1) WHEN (IR2='1' AND ADDRBUS=ADDLOGIC)
    ELSE MEM(2) WHEN (IR3='1' AND ADDRBUS=ADDLOGIC) ELSE (OTHERS=>'Z');
END ARCHITECTURE;

```

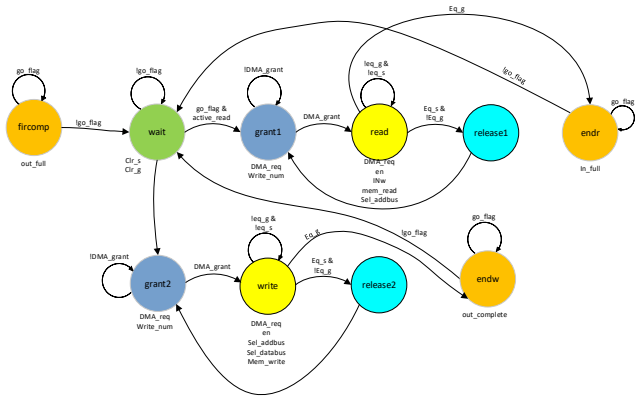
D. DMA

DMA Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer. Our DMA read and writes in burst mode.

The Data path of my DMA is illustrated below:



Also the controller of this is like below:

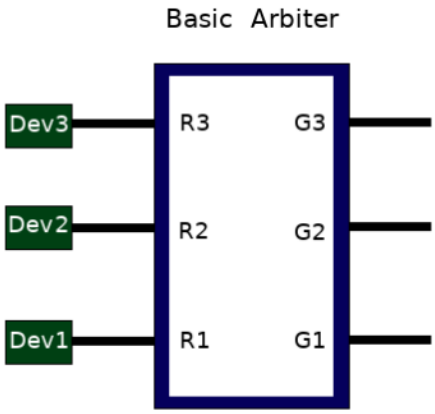


And the top module of this DMA has this ports:

```
ENTITY DMA_TOP IS
    PORT (
        clk ,rst : IN std_logic;
        DMA_grant : IN std_logic;
        writetoDMA : IN std_logic;
        datatoDMA : IN std_logic_vector(15 downto 0);
        addrtoDMA : IN std_logic_vector(6 downto 0);
        firComplete : IN std_logic;
        addrbus : INOUT std_logic_vector(15 downto 0);
        databus : INOUT std_logic_vector(15 downto 0);
        writetoFIR : OUT std_logic;
        DMA_req : OUT std_logic;
        out_full : OUT std_logic;
        in_full : OUT std_logic;
        datatoFIR : OUT std_logic_vector(15 downto 0);
        cnt_g : OUT std_logic_vector(6 DOWNTO 0);
        write_num : OUT std_logic;
        mem_read : OUT std_logic;
        mem_write : OUT std_logic;
        numOfData : OUT std_logic_vector(15 downto 0);
        out_complete : OUT std_logic
        );
END ENTITY DMA_TOP;
```

E.ARBITTER

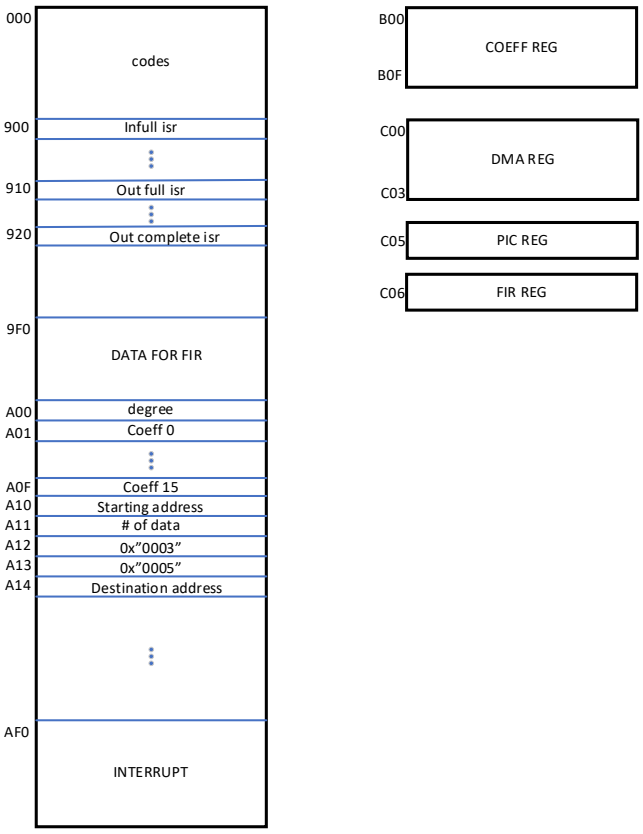
the bus arbiter is a device used in a multi-master bus system to decide which bus master will be allowed to control the bus for each bus cycle.



Requests			Grants		
R1	R2	R3	G1	G2	G3
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0

E.MEMORY MAP ADDRESS

The addresses which I set for testing my work are like this :



The content of the code parts is :

```
00000 1A00
00001 3B00
00002 1A01
00003 3B01
00004 1A02
00005 3B02
00006 1A03
00007 3B03
00008 1A04
00009 3B04
0000a 1A05
0000b 3B05
0000c 1A06
0000d 3B06
0000e 1A07
0000f 3B07
00010 1A08
00011 3B08
00012 1A09
00013 3B09
00014 1A0A
00015 3B0A
00016 1A0B
00017 3B0B
00018 1A0C
00019 3B0C
0001a 1A0D
0001b 3B0D
0001c 1A0E
0001d 3B0E
0001e 1A0F
0001f 3B0F
00020 1A10
00021 3C01
00022 1A11
00023 3C02
00024 1A12
00025 3C00
00026 0000
00027 0000
00028 0000
00029 0000
```

Until 1f the coefficients registers will be filled and from 20 to 25 the DMA registers will be set for reading the data and filling them in the fir input buffer.

--And the contents of interrupt is :

AF0: PC
AF1: SIC(FC)
AF2: JMN(C05)
AF3: RIC
AF4: EIN
AF5: JMN(AF0)
.
.
.
AFC: AC
AFD: OF
AFE: IN
AFF: SR

--And the contents of infull isr is :

900: STA(C06)
901: ACZ
902: STA(C00)
903: JMA(AF3)

--And the contents of out full isr is :

910: LDA(A14)
911: STA(C01)
912: LDA(A13)
913: STA(C00)
914: JMA(AF3)

--And the contents of out complete isr is :

920: ACZ
921: STA(C00)
922: JMA(AF3)

--The destination address is A15

```
0a14 |0A15
```

--# of data is 8

```
)0a11 |0008
```

--starting address is 9F0

```
)0a10 |09F0
```

--the degree is 3 and coefficients are :

```
0a00 |0003
0a01 |0001
0a02 |0001
0a03 |0002
0a04 |0003
0a05 |0004
0a06 |0005
0a07 |0006
0a08 |0007
0a09 |0008
0a0a |0009
0a0b |000A
0a0c |000B
0a0d |000C
0a0e |000D
0a0f |000E
```

--the input data for calculation is :

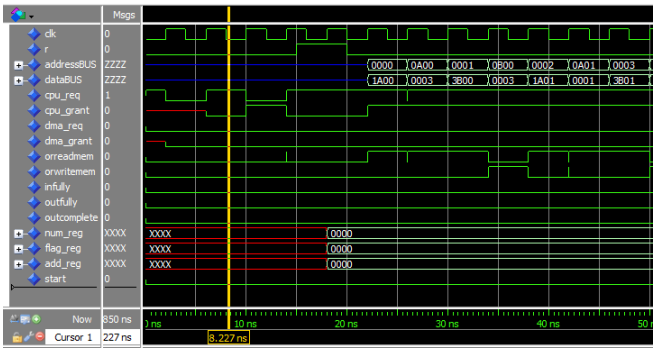
```
)09f0 |0000
)09f1 |0001
)09f2 |0002
)09f3 |0003
)09f4 |0004
)09f5 |0005
)09f6 |0006
)09f7 |0007
)09f8 |0008
)09f9 |0009
)09fa |000A
)09fb |000B
)09fc |000C
)09fd |000D
)09fe |000E
)09ff |000F
```

Because the number of date is 8 we just use 0,1,2,3,4,5,6,7.

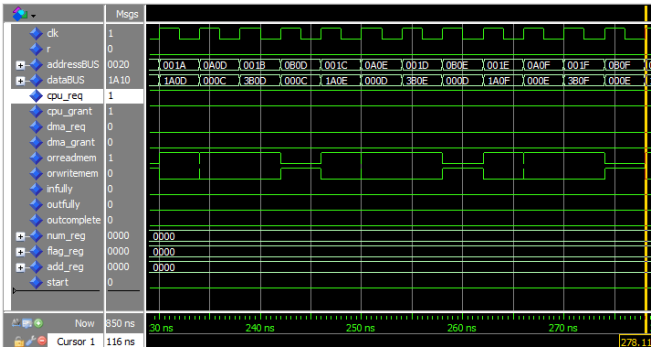
F.TESTING

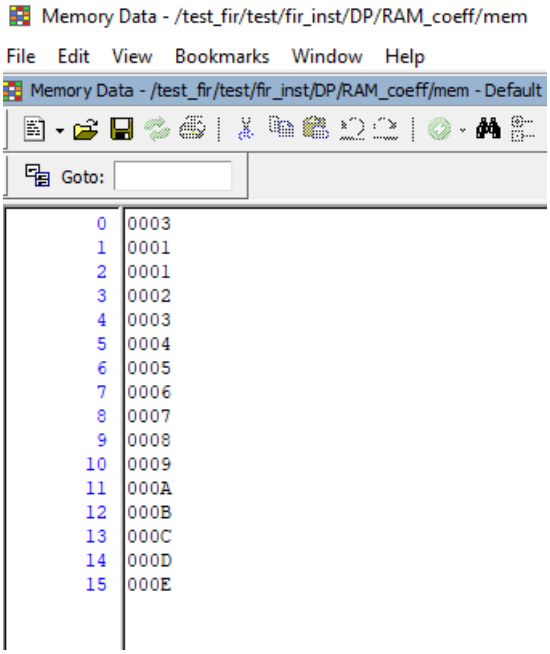
For the verification of this code, according to the memory contents of the previous part, a test bench was made(TB) which proves the authenticity of our design.

Now we can see After resetting the system the arbiter gives permission the CPU to get the bus and run the codes:

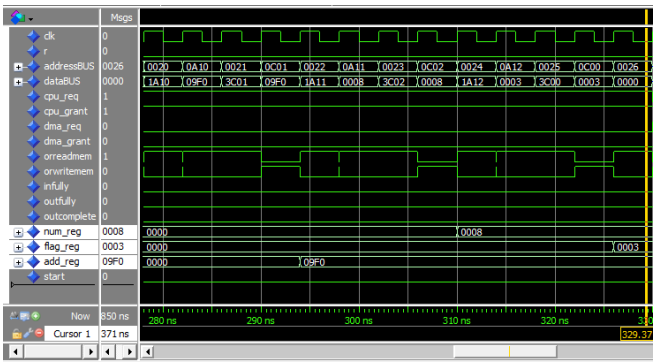


After running the codes until the address 1F, we fill the FIR coefficient ram with the data in memory from A00 to A0F:

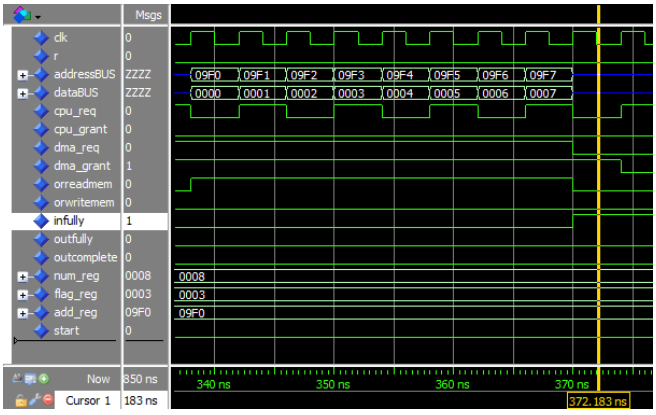


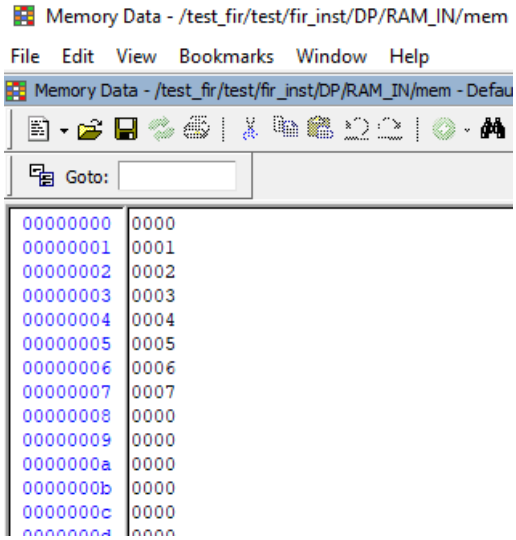


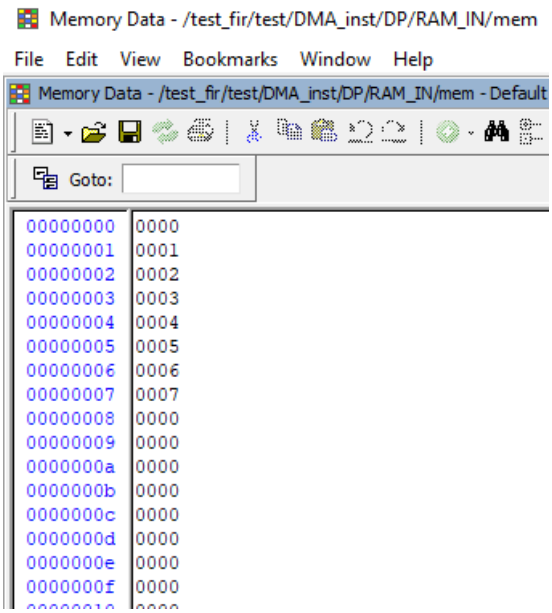
with running the codes from 20 to 25 the registers of DMA will be set :



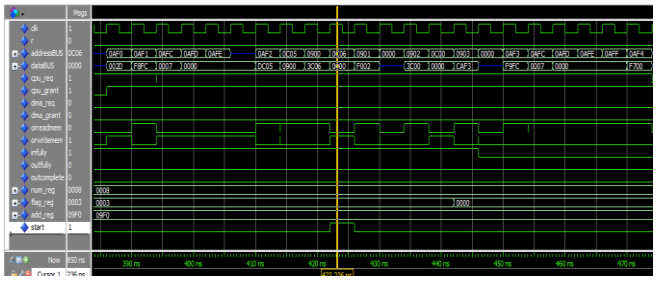
Now the DMA wants to start reading from memory so get the bus and write the data in the input buffer of DMA and FIR, after this the DMA set the in full interrupt :



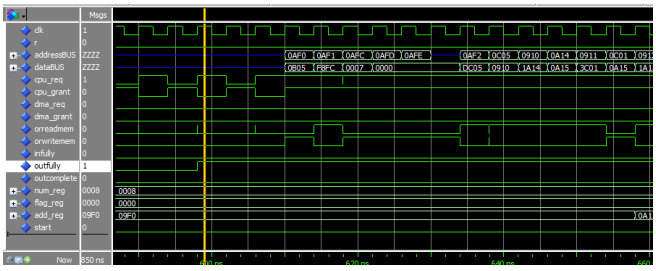




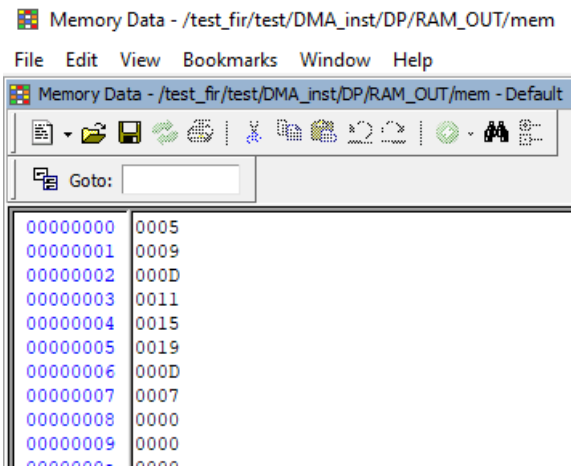
Now the CPU goes to the interrupt part and run the in full isr codes. Start of fir will set here:



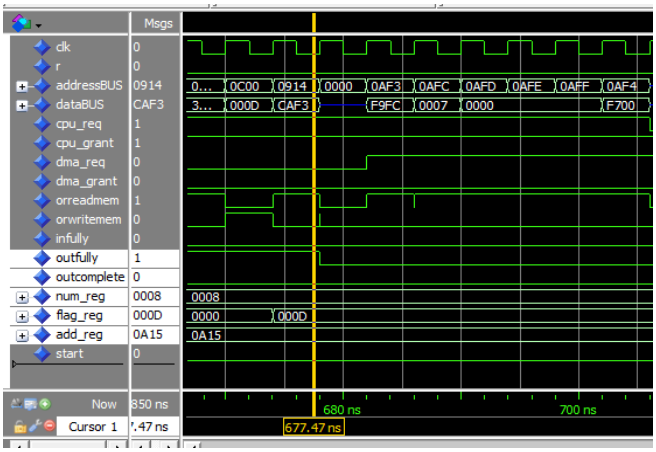
When the fir completes its job and puts the calculated data in out buffer of DMA, the DMA set the out full interrupt and out full isr will be run:



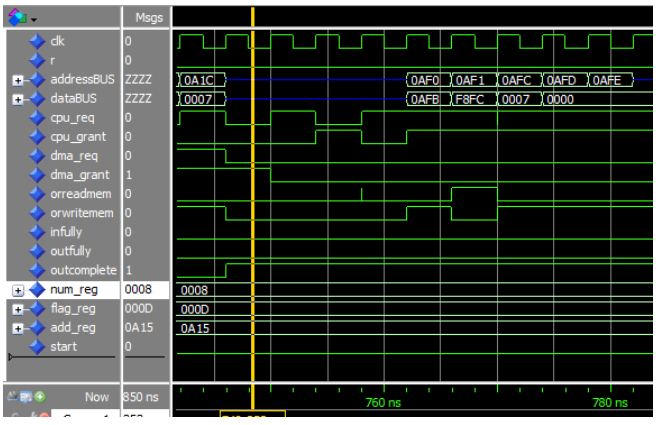
You can see the calculated data of fir in out buffer of DMA. Remind that the number if our data is 8 and the degree is 3 so the data from 0 to 5 are valid and others are invalid:



The ISR of out full set the DMA to write the data in the destination address of the memory:



After completing the act of writing calculated data in memory, the out complete interrupt will be set and the ISR clear the flag register of DMA:



We can see the calculated data were written in the destination address(from A15) of memory: