Project Report
On
Implementing Multi-class Classifier on Cortex M3(AT421-MN-80001-r0p0-02rel0) using Design Eval

---

By:

Saswat Padhy (MT2019522)

Under the guidance of:

Prof. Girish S kumar

Introduction:

What is a SOC:

SoC stands for System on Chip. Its highly integrated version of Integrated Circuits. A modern SoC may have billions of transistors integrated into a single device. Alternatively, a SoC is a electronic device which has Hardware and Software as two essential components, working in tandem to realize some function(s).

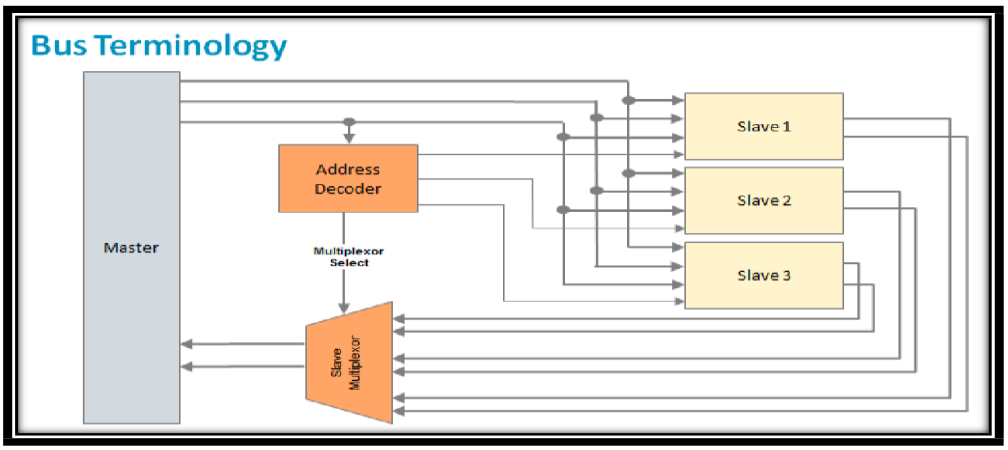
What is inside a SoC: Components of SoC

Almost all the SoCs will have the following elements/components:
Following the list, more about each element of the SoC is explained and its presence justified.

1. I/O pads : These are the pins which lets the SoC communicate with outside world.
2. Power management Unit : Though in some cases PMU can be external to the SoC, it in turn contains Voltage Regulators.
3. Production test logic also called Design for Test Logic (DFT):
4. Trace & Debug Logic:
5. Fuses.
6. Microprocessor(s)
7. Memory(s). Several types of memories. Cache, ROM, Flash, SRAM.
8. Communication Back bone: Also called SoC fabric, System Bus etc.
9. Security logic.
10. Software: This may not be there for all SoCs, and the SoCs without any software in it, will allow the user to put software in its memories post production.
11. Clock source, e.g. crystal oscillator: Though in some cases it is possible to provide a clock to a SoC externally
12. PLLs: Phased Locked Loops.
14. Clock & Reset Logic
15. Power consumption reduction logic. Some times also called low power features.
16. Peripherals: Logic to transfer data off chip, and to receive data from external sources. e.g. USB, I2C, PCI, UART. Ethernet
17. RF Circuit: Not all SoCs have them. But the ones with wireless communication on board must have it.
18. Timers
19. Watchdogs.
20. Interrupt logic.
21. Pin-Muxing logic.
22. Hardware Boot State Machine.
23. Temperature Sensor.
24. AON logic: Always ON Logic.

25. DACs/ADCs
26. DSP processor/DSP logic
27. Ring Oscillators.
28. Random number generator sometimes also called TRNG (True Random Number Generator)
29. Encryption/Decryption logic
30. Power-ON reset generator.
31. Battery Charging Circuit.
32. SoC Control and Status Register Block.
33. SoC ID Register.

Some of the important structures along with the terminologies of an SOC is explained as follows:



A "**master**" (or initiator) refers to the IP component that initiates a read or write data transfer (e.g., processor or DSP).

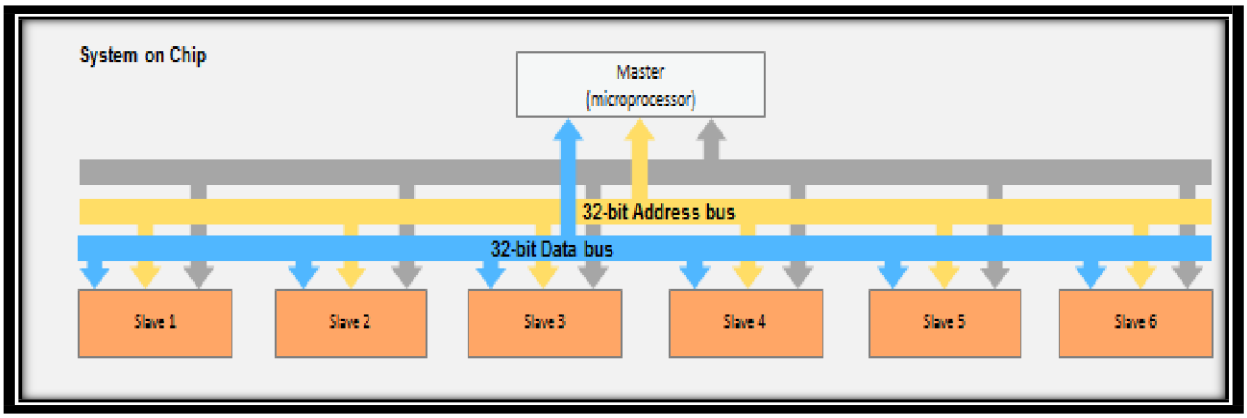
A "**slave**" (or target) refers to the IP component that does not initiate transfers and only responds to incoming transfer requests (e.g., memory block).

A "**decoder**" refers to a logic block that decodes the destination address of a data transfer initiated bya master, and selects the appropriate slave to receive the data.

A **multiplexor** is used to multiplex the read data bus and response signals from the slaves to the master. The decoder provides control for the multiplexor.

A bus typically consists of three types of signal lines:

- The data bus is used to exchange data information.
- The address bus is used to select one of the peripherals (or one register of a peripheral).
- Control signals are used to synchronize and identify transactions, such as ready, write/read, and transfer mode signals.



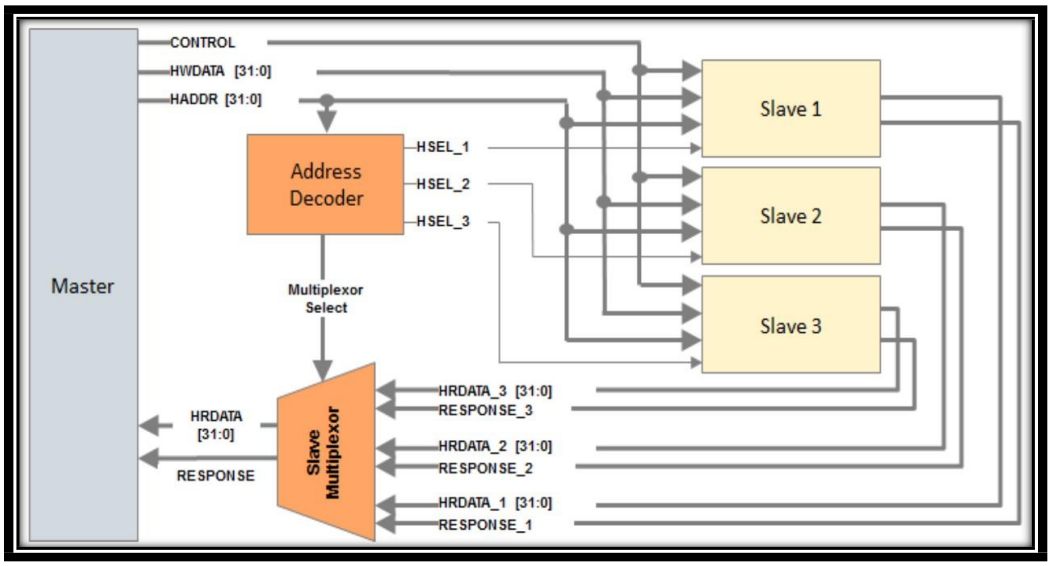
Arm AMBA System Bus

AMBA: Advanced microcontroller bus architecture

- AMBA protocol is an open standard on-chip interconnect specification.
- Provides the interface standard that enables IP reuse
- Facilitates right-first-time development of multiprocessor designs with large numbers of controllers and peripherals
- Widely used in modern portable mobile devices, such as tablets and smartphones

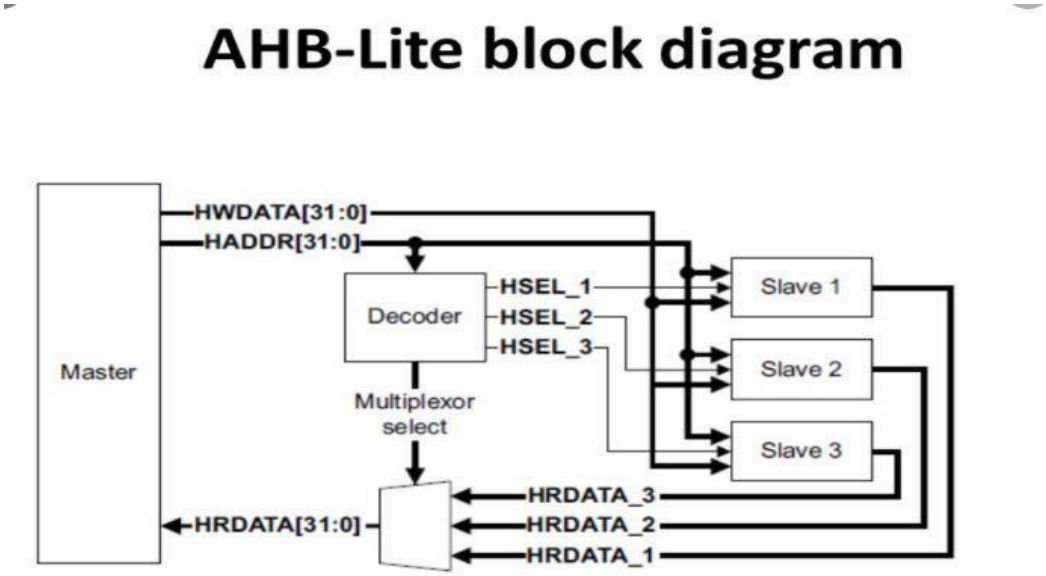
\

Arm AMBA AHB-Lite Protocol

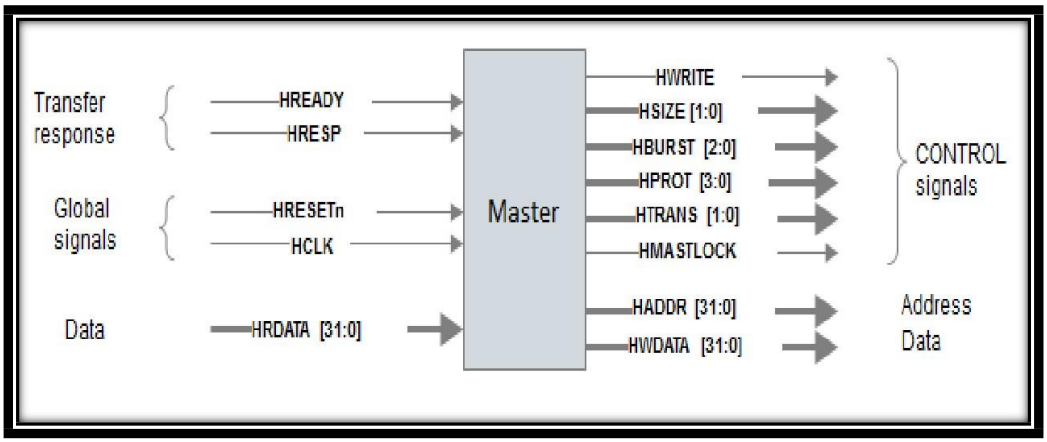


The above shows a single master AHB-Lite system design with one AHB-Lite master and three AHB-Lite slaves. The decoder block is controlled by the master that specifies which slave must be selected.

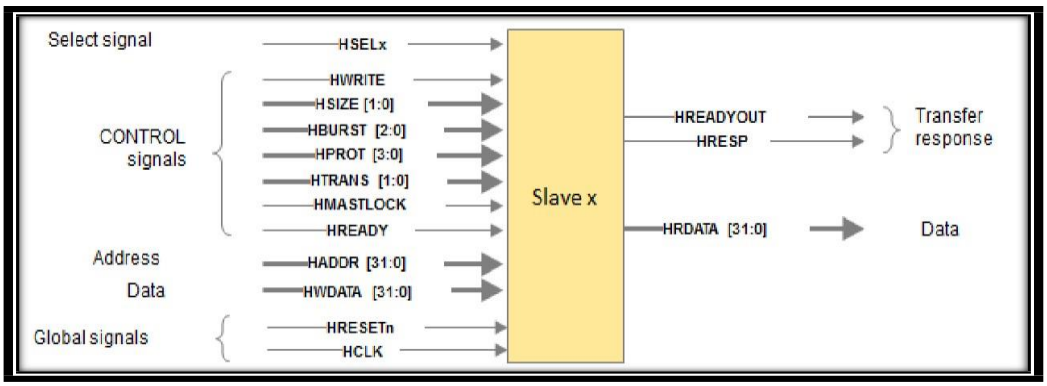
The multiplexor block is controlled by the decoder that chooses which slave output data must be connected to the master.



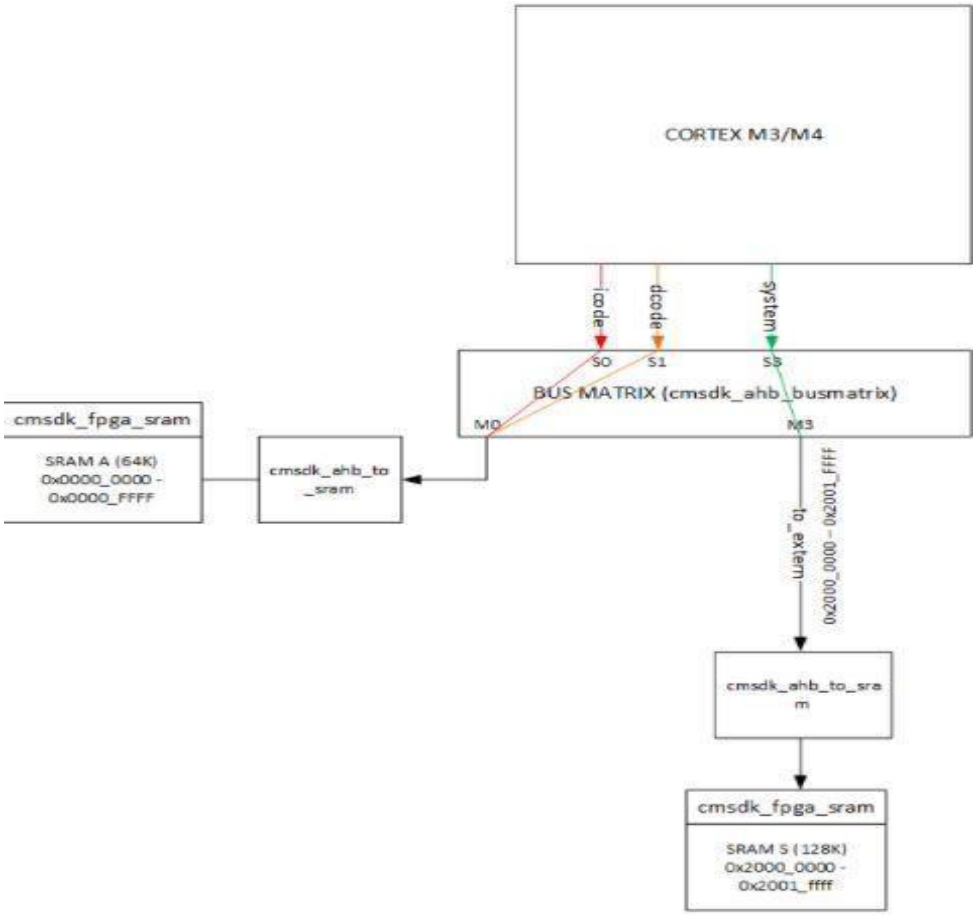
Master bus Interface



Slave Interf



Structure of the SOC being built:



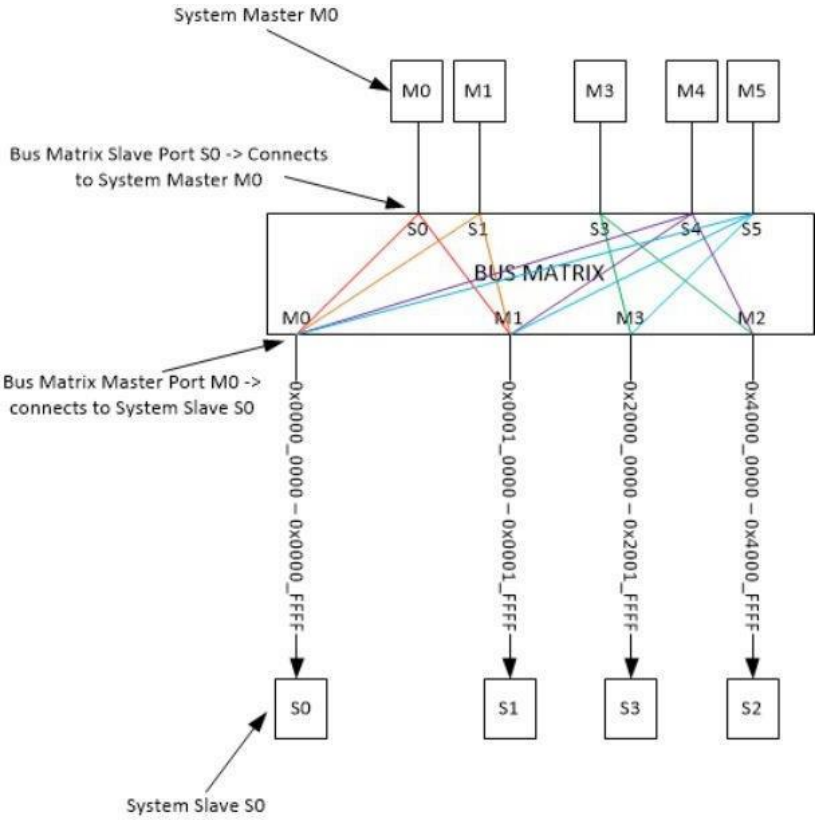
As shown the main processor is ARM Cortex-M3 processor.
The processor is connected to ARM Multilayered Bus Matrix.
The ARM Multilayered Bus Matrix is connected to 2 memories.

The ARM Cortex-M3 processor has following 3 AHB-Lite Interfaces to connect to the system.

1. ICODE Bus
2. DCODE Bus
3. System Bus.

The 'Bus-Matrix' in the above figure is a multi-layered ARM provided AHB Lite Bus Matrix. The '*cmsdk_ahb_busmatrix*' component supports up-to 15 Masters and 15 Slave.

Bus matrix connectivity is shown in the fig. as below:



The Cortex-M3 processor has 3 AHB Lite Interfaces
ICODE -> Connects to AHB Bus Matrix Slave Port S0
DCODE -> Connects to AHB Bus Matrix Slave Port S1
System -> Connects to AHB Bus Matrix Port S3.

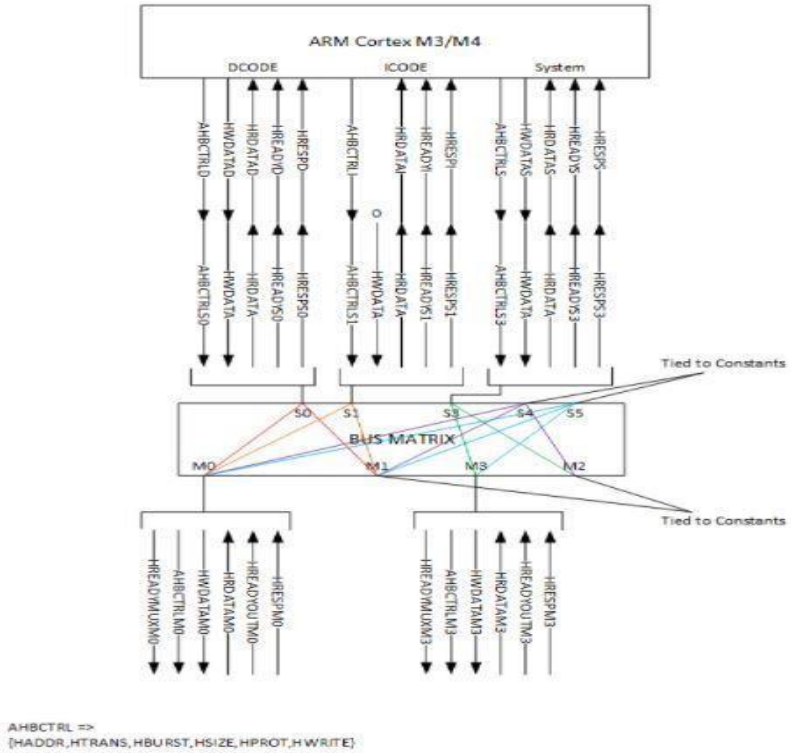


Figure 3.1 ARM Cortex M3 Stage 1 SoC Connections

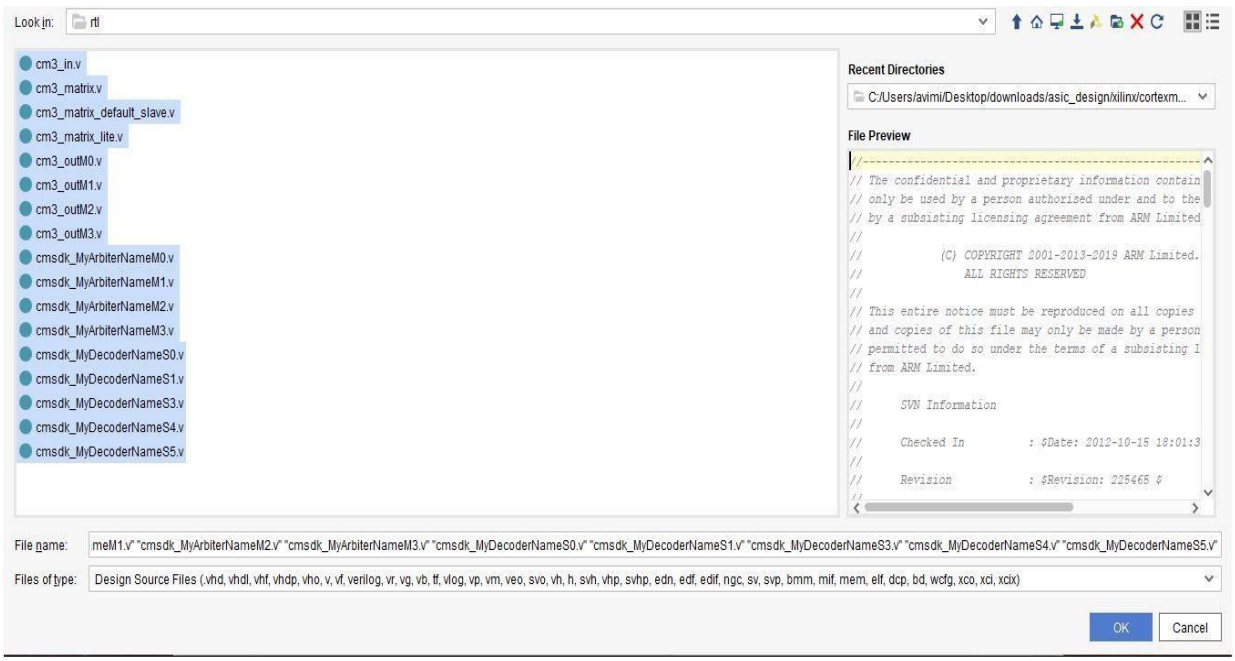
Structure of Cortex M3(AT421-MN-80001-r0p0-02rel0) Integration file

Below is the RTL files overview used in integration [all files are saved in folders having the respective component names]

```
asic_design/xilinx/cortexm3_soc/top/tb/cortexm3_soc_tb.v
asic_design/xilinx/cortexm3_soc/top/rtl/cortexm3_soc.v
asic_design/xilinx/cortexm3_soc/CORTEXM3INTEGRATION/rtl/CORTEXM3INTEGRATIONDS.v
asic_design/xilinx/cortexm3_soc/CORTEXM3INTEGRATION/rtl/cortexm3ds_logic.v
asic_design/xilinx/cortexm3_soc/cmsdk_ahb_to_sram/rtl/cmsdk_ahb_to_sram.v
asic_design/xilinx/cortexm3_soc/cmsdk_fpga_sram/verilog/cmsdk_fpga_sram.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cm3_in.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cm3_matrix.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cm3_matrix_default_slave.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cm3_matrix_lite.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cm3_outM0.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cm3_outM1.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cm3_outM2.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cm3_outM3.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyArbiterNameM0.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyArbiterNameM1.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyArbiterNameM2.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyArbiterNameM3.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyDecoderNameS0.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyDecoderNameS1.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyDecoderNameS3.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyDecoderNameS4.v
asic_design/xilinx/cortexm3_soc/cm3_matrix/rtl/cmsdk_MyDecoderNameS5.v
```

Steps:

1) RTL files along with the test-bench setup in XILINX VIVADO



2) Multi-classifier assembly code implementation using KEIL:

The multi-classifier code is written in assembly language ,then converted to .hex file using KEIL as shown below:

```
:020000040000FA
:10000000081000020C1000000C9000000CB00000063
:10001000CD00000CF000000D1000000000000000073
:10002000000000000000000000000000D3000000FD
:10003000D5000000000000000D7000000D900000003B
:10004000DB000000DB000000DB000000DB00000044
:10005000DB000000DB000000DB000000DB00000034
:10006000DB000000DB000000DB000000DB00000024
:10007000DB000000DB000000DB000000DB00000014
:10008000DB000000DB000000DB000000DB00000004
:10009000000000000000000000000000000000060
:1000A000000000000000000000000000000000050
:1000B000DB000000000000000DB000000DB000000AF
:1000C0000948047094800047FEE7FEE7FEE7FEE7EC
:1000D000FEE7FEE7FEE7FEE7FEE7FEE704480549 28
:10012000022C01D10A4607E0A4F1010401EB000210
:10013000084611460 22CF7D11A60000004480068F6
:1000E000054A064B704700003D0100009010000071
:1000F000080000200810002008 0C002008 0C002038
:10010000704770477047000004FF000004FF001014A
:100110004FF000534FF00A04012C01D102460BE0CE
:10012000022C01D10A4607E0A4F1010401EB000210
:10013000084611460 22CF7D11A60000004480068F6
:10014000 40F47000024908600248034908607047A3
:1001500088ED00E040787D0100000020 40787D01BE
:1000B000DB000000000000000DB000000DB000000AF
:1000C0000948047094800047FEE7FEE7FEE7FEE7EC
:1000D000FEE7FEE7FEE7FEE7FEE7FEE704480549 28
:1000F000080000200810002008 0C002008 0C002038
:10010000704770477047000004FF000004FF001014A
:04000005000001 09ED
:00000001FF
```

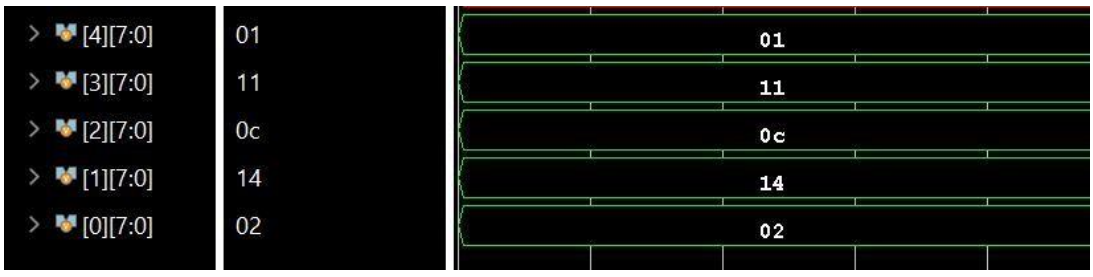
Now this .hex file is placed inside the .sim directory generated by running the behavioral simulation in VIVADO.

Then the behavioral simulation s is run giving appropriate time scales using the TCL console in VIVADO. [command used is run 10000 =>10000 ps]

Simulation Waveforms:
[Note: All the values in waveform are in hex]

Here first the contents of the sensor file is imported from the Thingspeak platform; then is converted into binary and then stored in nets named BRAM3 as shown below.

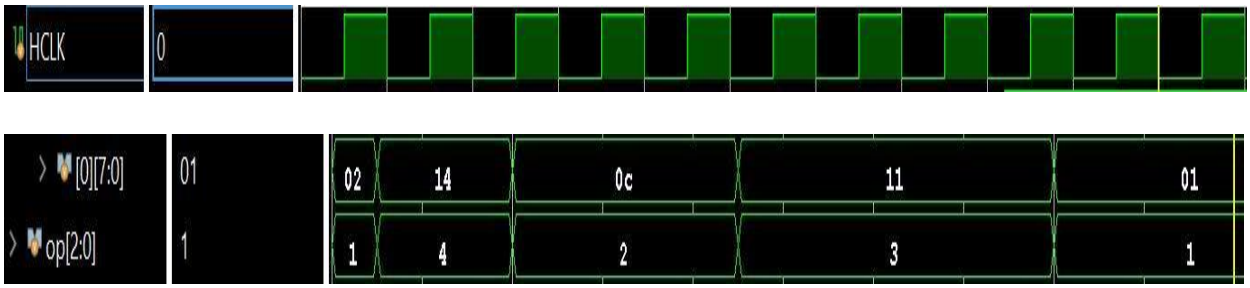
5 values corresponding to different materials are stored in the BRAM3.



Now these values are read into the vis_ro_0 register w.r.t clock and then these values are classified into 4 different types of materials (whether it is paper, plastic, steel or rubber)

The o/p reg shows 4 different labels for 4 different materials as op reg should show as;

1: steel, 2: paper, 3: plastic ,4: rubber corresponding to the readings in vis_ro_0 reg.



Hall effect values <= 1 should be shown as steel; so op reg should show 1

Similarly Hall effect values <= 14 should be shown as paper; so op reg should show 2

Similarly Hall effect values <= 18 should be shown as plastic; so op reg should show 3

Similarly Hall effect values > 18 should be shown as rubber; so op reg should show 4.

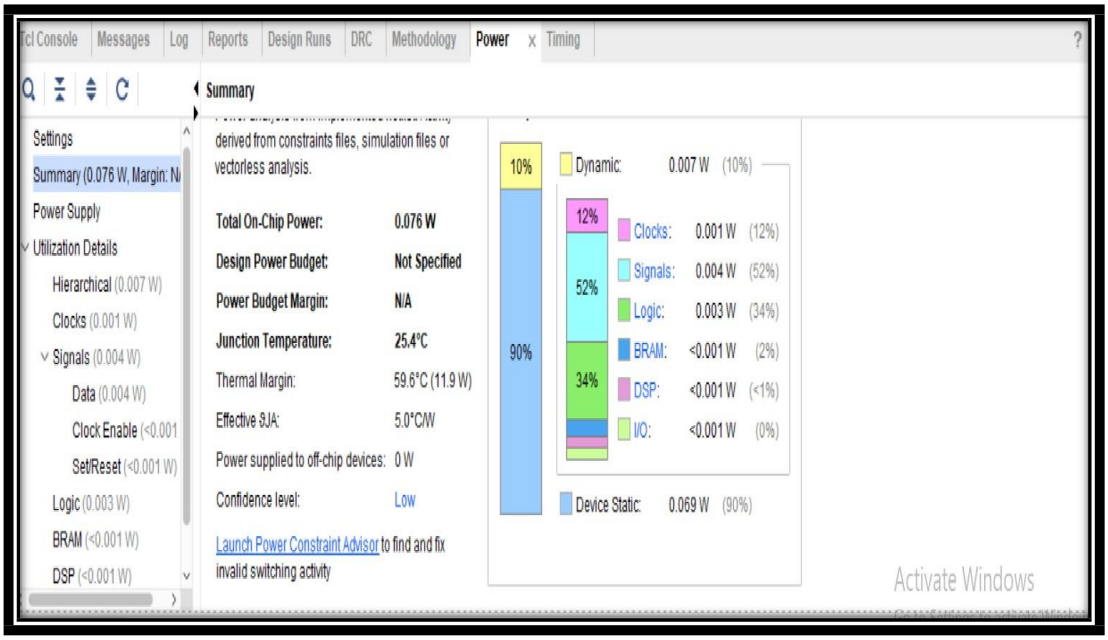
Note: All these hall value classes were obtained by using permeability as the classifying parameter.

As you can see that the op reg is producing the outputs as expected thus successfully implementing a multi-class classifier [4 class classifier]

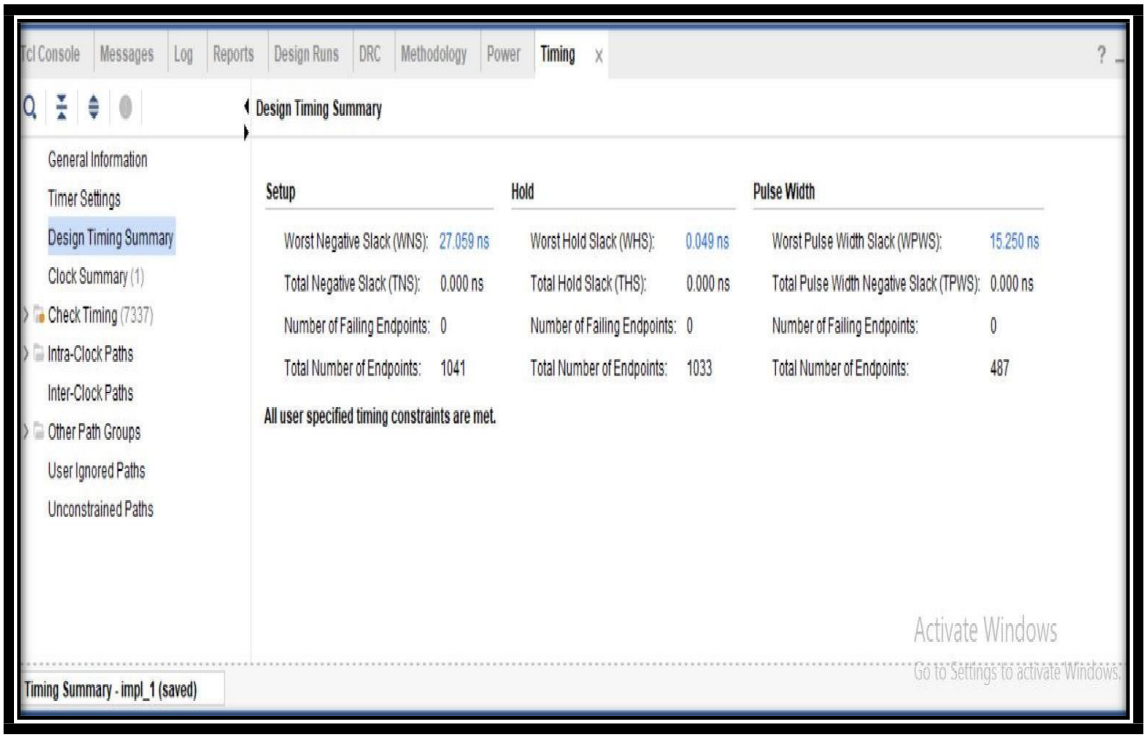
Results:

1) Power and Timing Reports post Synthesis:

Power consumed during code synthesis



Timing summary



Conclusion:

As defined earlier an SOC achieves an intended function by using both hardware and software. The data intensive functions are generally implemented on hardware part so as to speed up the process and it also gives dynamicity to the function. Implementing a function on software although does not give advantages in speed but it consumes less bandwidth and memory as it is static in nature.

In this project a SOC has been developed in which the AMBA bus protocol AHB-Lite is implemented on hardware and the intended application is using it as a multi-class classifier is implemented on software assembly language and they are co-simulated using XILINX VIVADO.