DIGITAL LAB PROJECT

OV SHASHANK 14D070021 YOGESH MAHAJAN 14D070022

INTRODUCTION

The project is an analog store and play device (Sampling Rate: 1kHz) which has been created using the Altera Max V chip on the Krypton Board while utilising external components, viz. ADC (0804), DAC (0808) and SRAM (HY62256A).

OVERALL DESCRIPTION

The overall system consists of an analog input, 3 digital inputs: reset, capture and display, and one analog output.

1. Analog Input: PIN 5 of the ADC
2. Reset: Switch 8 of the Krypton Board
3. Capture: Switch 2 of the Krypton Board
4. Display: Switch 1 of the Krypton Board
5. Analog Output: PIN 5 of the Op-Amp UA741

Reset: This input is used to reset the system to a default initial state. Any misbehaviour can be eliminated by using this input.

Capture: This input instructs the system to read in the analog data and store it into the SRAM. 13 address bits of the SRAM are used and hence 8192 storage locations are available. And at the fixed sampling rate of 1ms, about 8.192s of data can be stored in a wrap-around fashion.

Display: This input instructs the system to give out the data stored on the SRAM.

The following is the functioning modes of the device

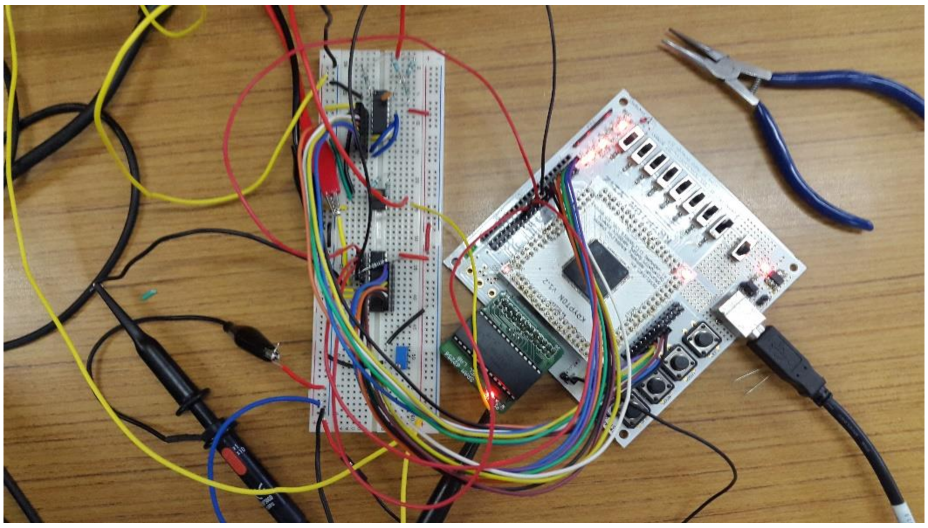
Reset	Capture	Display	Mode
1	X	X	Null (Reset Address = 0)
0	0	0	Null (Do Nothing)
0	1	X	Capture (Read in Data)
0	0	1	Display (Write out Data)

NOTE: As the SRAM does not store data, if the power is removed, and the Display Mode is run without the Capture mode being run for at least 8.192s, the output will be random.

CONNECTIONS

1. The Header 2 is to be used to connect to the SRAM Breakout Board. PIN X is connected to the corresponding PIN X of Header 2
2. Header 1 is to be partly used for the ADC connections.
    a. Odd number pins from 1 to 15 to be used for the ADC output connections with PIN 1 being the LSb
    b. PIN 17 for $\overline{INTR}$
    c. PIN 19 for $\overline{CS}$
    d. PIN 21 for $\overline{RD}$
    e. PIN 23 for $\overline{WR}$
3. Header 0 is to be partly used for the DAC connections. Odd number pins from 1 to 15 to be used for the ADC output connections with PIN 1 being the LSb.

REPRESENTATIONAL IMAGE



SOFTWARE

The software for the MAX V chip has been written in VHDL and compiled and programmed using Quartus Prime Lite 15.1 Edition for Windows and JTAG for Ubuntu.

COMPONENTS

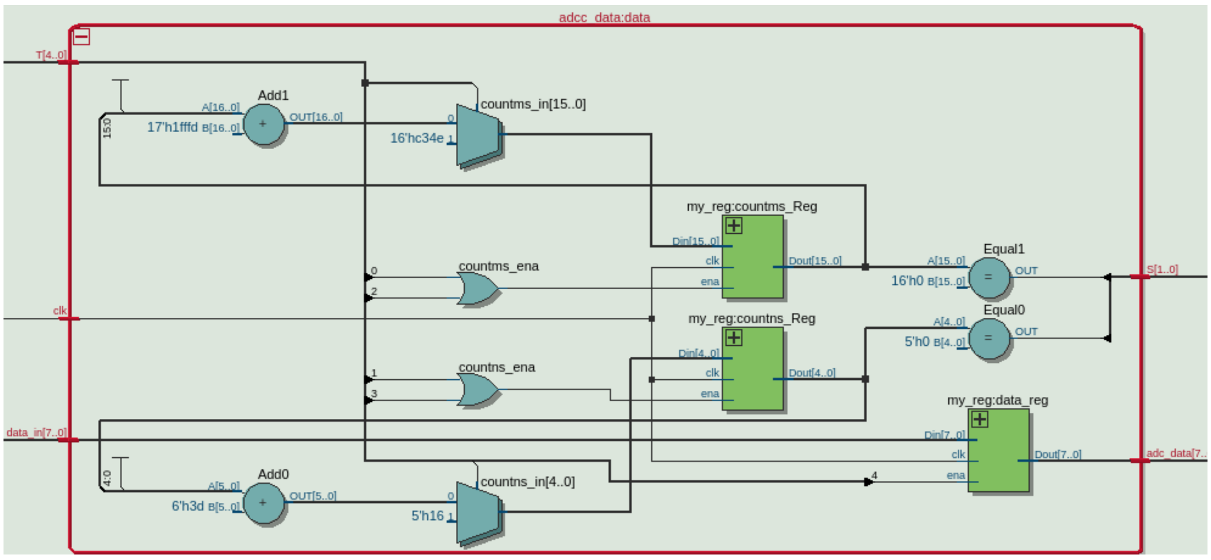
Our system consists of three components which are combined in the toplevel entity named project, which can be found in the file project.vhd (Hyperlinked here).

ADCC

ADCC is the ADC control unit. This interacts with the ADC to give the sampled data at a rate of 1kHz. It abstracts out the hardware interaction for the main control unit. For its proper functioning, it must be provided with a clock of 50 MHz, which is derived from the internal clock of the Krypton Board, because the timing requirements to be met for the ADC have been implemented assuming a clock period of 20ns. The component ports meet the specifications as described in the manual.

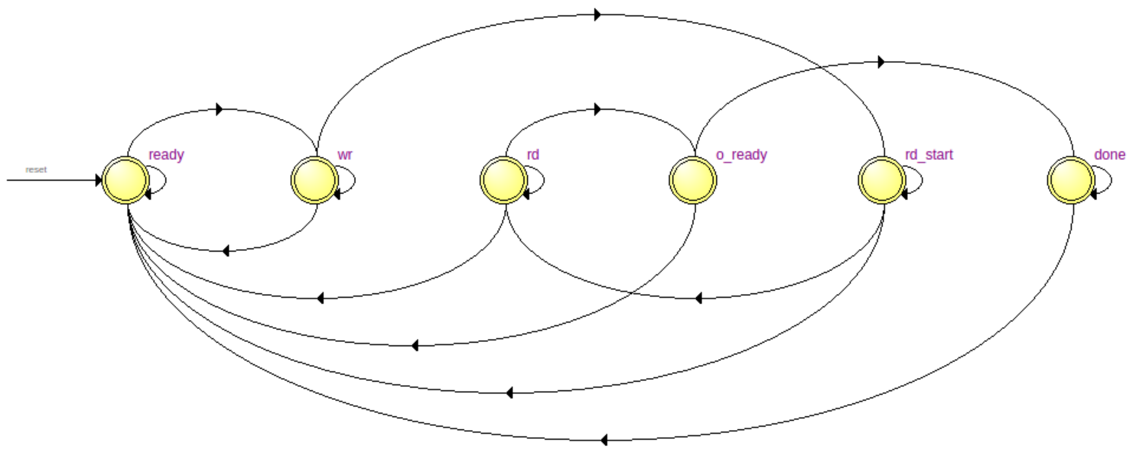
It can found in the file adcc.vhd (Hyperlinked here)

DATA PATH



CONTROL PATH

STATE TRANSITION DIAGRAM



STATE FLOW TABLE

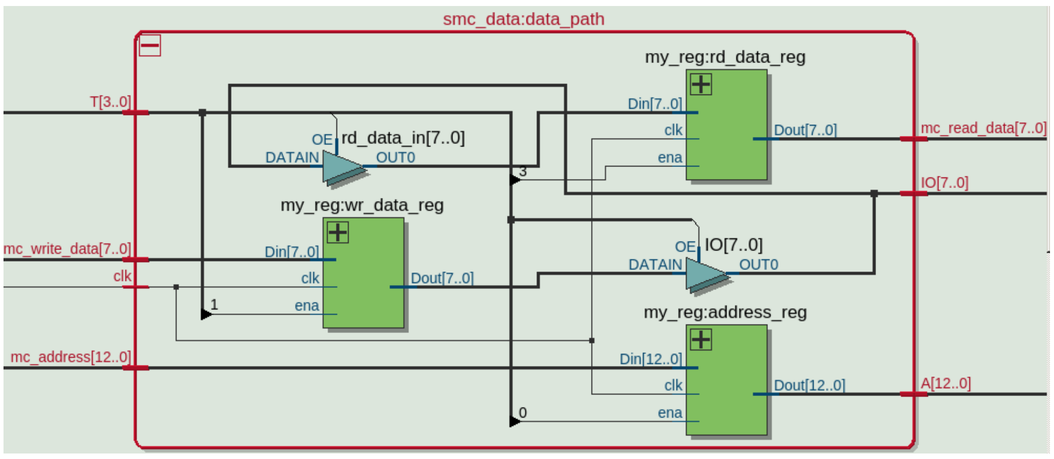
	Source State	Destination State	Condition
1	done	done	(!S[1]).(!reset)
2	done	ready	(!S[1]).(reset) + (S[1])
3	o_ready	done	(!reset)
4	o_ready	ready	(reset)
5	rd	o_ready	(S[0]).(!reset)
6	rd	rd	(!S[0]).(!reset)
7	rd	ready	(reset)
8	rd_start	rd	(!intr_temp).(!reset)
9	rd_start	rd_start	(intr_temp).(!reset)
10	rd_start	ready	(reset)
11	ready	ready	(!adc_run) + (adc_run).(reset)
12	ready	wr	(adc_run).(!reset)
13	wr	rd_start	(S[0]).(!reset)
14	wr	ready	(reset)
15	wr	wr	(!S[0]).(!reset)

SMC

SMC is the SRAM control unit. This interacts with the SRAM to read or write on to it as demanded. It abstracts out the hardware interaction for the main control unit. For its proper functioning it must be provided with a clock of 50 MHz, which is derived from the internal clock of the Krypton Board, because the timing requirements to be met for the SRAM have been implemented assuming a clock period of 20ns. The component ports meet the specifications as described in the manual.

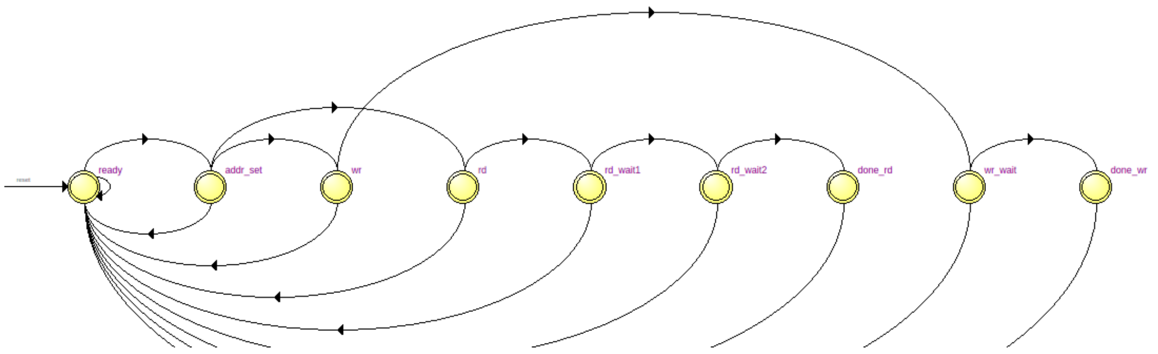
It can found in the file smc.vhd (Hyperlinked here)

DATA PATH

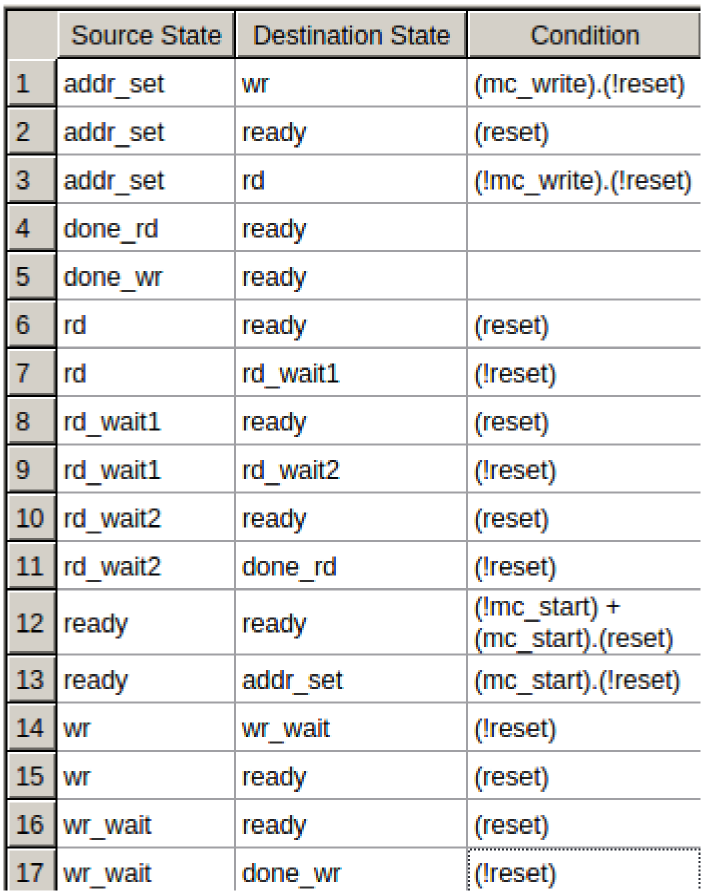


CONTROL PATH

STATE FLOW



STATE TABLE



	Source State	Destination State	Condition
1	addr_set	wr	(mc_write).(!reset)
2	addr_set	ready	(reset)
3	addr_set	rd	(!mc_write).(!reset)
4	done_rd	ready	
5	done_wr	ready	
6	rd	ready	(reset)
7	rd	rd_wait1	(!reset)
8	rd_wait1	ready	(reset)
9	rd_wait1	rd_wait2	(!reset)
10	rd_wait2	ready	(reset)
11	rd_wait2	done_rd	(!reset)
12	ready	ready	(!mc_start) + (mc_start).(reset)
13	ready	addr_set	(mc_start).(!reset)
14	wr	wr_wait	(!reset)
15	wr	ready	(reset)
16	wr_wait	ready	(reset)
17	wr_wait	done_wr	(!reset)

CCU

This is the centre of the system. It takes the inputs from the environment and based upon them, works with the control units to perform the necessary function. The component ports meet the specifications as described in the manual. In the display mode, it reads from the SRAM at a rate of 1kHz.

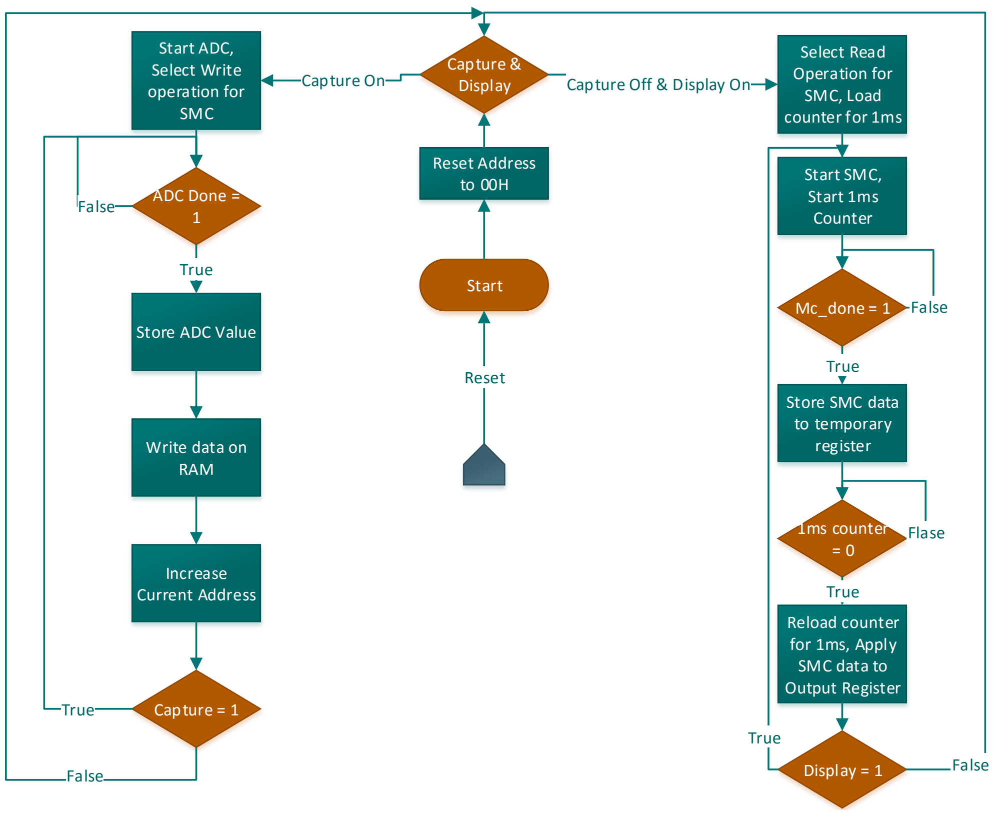
It can be found in the file ccu.vhd (Hyperlinked here)

Its working algorithm is as follows:

ALGORITHM

```
--CCU Control Path

rest : all signals to '0' except curr_address
      if(capture = '1') then
         next state = write
      else if ((capture = '0') and (display = '1')) then
         next state = read
      else
         next state = rest
---------------------------------------------------------------
**write (wrap around write fashion)-->
      curr_address = address of current selection so for reading operatio
this will be start point

wr : adc_start <= '1' so ADCC will provide sample after every 1ms
        mc_write <= '1'  write signal will always be high, so SMC is
controlled by mc_start
         next state = getADC

getADC : if(adc_output_reday = '1') then
            store adc_data in reg
            next state = wrOnRAM

wrOnRAM : mc_start <= '1'
          if (mc_done = '1')then
            next state = updateWrite

updateWr : curr_address ++
               if(capture = '0') then
                  next state = rest
               else
               next state = getGCD
---------------------------------------------------------------
**read (wrap around read fashion) --> start address is curr_address

rd : mc_write <= '0';
         next state = readRAM
         load 1ms counter

readRAM : mc_start <= '1'
          start 1ms counter  --update Tvar in every state
         next state = wait1

wait1 : if(mc_done = '1') then
         store output in RAMoutReg
         next state = wait2

wait2 : if(1ms counter = "0") then
         stop 1ms counter
         next state = updateRead

updateRd : curr_address ++
             reload 1ms counter
             apply RAMoutReg to DACrag
            if(display  = '0') then
               next state = rest
           else
               next state = readRAM
```

ALGORITHM (FLOW CHART)



TESTING

Test benches have been created and tested using GHDL and GTKwave for each component. They can be found in the Testbench Folder. Several signals have been tested on to the system and the system has provided the expected outputs every time.

PROBLEMS FACED

- The first problem we faced was when the ADCC was tested. It would stop working in a while. After debugging and suggestions from professors, we realised that we had not synchronised the inputs from the asynchronous ADC. Adding a flip-flop rectified the issue.
- The second problem was that the SMC test would succeed even in the absence of the SRAM. This fault was detected by chance because of the simple test we were performing (not changing the SRAM address) due to lack of necessary hardware. The issue was tracked to the read tristates which had always been enabled and hence would read in the data even while writing to the I/O ports. It was rectified by adding the conditional enable.
- The last problem was a plain communication gap which led to different assumptions. This was quickly identified by observing various signals on the LEDs of the Krypton Board.

FUTURE SCOPE

The project, though very basic, opens up a wide area of research and development. The sampled analog data memory allows any amount of processing to be done on it and the ability to display facilitates a large number of operations to be performed with the analog signal, like rectification, filtering, etc. very easily on a digital platform. This system can act as a processing unit for analog systems by taking input and providing analog signals.