ADC DAC

Arvadia Kevin Dharmesh, 170070001
Thorve Rajesh Bhaskar, 170070002
Pawar Atharv Amar, 170070003
Syomantak Chaudhuri, 170070004

May 2, 2019

1 Aim and Overview of the experiment

2 Method

To create the controller circuit that will administrate the working of the ADC and the DAC, we first understood the interface provided by the ADC. The ADC has the following pins (all of which are active low) available for use:

- $\overline{\textbf{CS}}$ **(Chip select)**: all operations are performed only when this pin is low
- $\overline{\textbf{WR}}$ **(Write)**: conversion of analog input to digital value starts when this pin sees a falling edge
- $\overline{\textbf{INTR}}$ **(Interrupt)**: this pin goes low to indicate that the conversion is complete and the digital value is ready to be read
- $\overline{\textbf{INTR}}$ **(Read)**: after applying a falling edge on this pin, the output can be read after some delay
- **Clock**: to supply clock signal to the ADC

The working of the ADC has some additional specifications. When we apply a low input the $\overline{WR}$ pin, we need to to keep the signal low for atleast

100 ns. The $\overline{INTR}$ signal becomes reliable only after 300 ns have passed after the falling edge on $\overline{WR}$ pin. Once we detect an interrupt signal and we apply a low input to the $\overline{RD}$ pin, we need to wait for atleast 135 ns before the converted value becomes available at the ADC's output, at which point we can read the data and again set $\overline{RD}$ to high. Finally, the sampling is to be done once every millisecond.

In all these cases, we need to keep a track of the time required before we can move on to the next step. We know that the frequency of the clock we will be using onboard the Krypton board has a frequency of 50 MHz and a corresponding time period of 20 ns. This suggested that we could use counters to keep a track of the time (in steps of 20 ns).

We chose the following delay values and the corresponding counters for our purposes:

- To ensure sufficient time of application of the write signal, we delayed the signal's removal by 100 ns, equivalent to 5 clock cycles.
- To ensure that the interrupt signal was reliable, we delayed consideration of the signal by 300 ns after applying the write signal, which translates to 15 clock cycles.
- After applying the read input, we decided to wait for 140 ns before reading the converted data, which is the same as 7 clock cycles.
- To measure a time period of 1 ms, we used a counter counting to 50000 clock cycles.

We created a separate counter circuit for each of these count values, which were started by the parent circuit when required, and indicated their timeout to the circuit. Since these actions require one clock cycle each, we reduced the count values which they counted by two, so that the measured time periods were accurate. The 1 ms timer circuit was decreased only by one, since it did not take any input.

The parent circuit was similar to a finite state machine, which used different states for different stages in the process of controlling the ADC, which utilized these timers for timekeeping. The first version simply passed the value read from ADC to the DAC, which automatically generated the expected analog output, and thus created the zeroth order hold version of the input signal, having a hold time of 1 ms.

In the filter version where we output the average of the last 8 readings, we created a separate component that took the digital readings (converted values from ADC), and output the average of the last 8 input values. To accomplish this, the circuit used 8 registers x_0 to x_7, all initialized to zero. The latest input was put into x_7, and each register x_i passed its value to register x_{i-1}, thus acting like a shift register. We also used a register to store the 11 bit sum of all the previous 8 recorded readings (11 bits ensured that the sum never overflows and precision is maintained). Each time a new reading is received, the oldest reading (i.e. x_0) is subtracted from the sum and the latest reading is added. Since the average is simply the sum divided by 8, we output the 8 most significant bits of the sum register as the output of the circuit. This implementation acts as a digital low pass filter.
The low pass filter's DTFT is given by

$$H_d(\omega) = \sum_{n=0}^{8} \frac{1}{8} \cdot e^{-j\omega n}$$

$$H_d(\omega) = \frac{(1 - e^{-8j\omega})}{8(1 - e^{-j\omega})}$$

Therefore, it has zeros at $\omega = k/8$ which corresponds to $\omega = 1000k/8$ in continuous time domain. Therefore, it has zeros at 125Hz, 250Hz, and so on.

3 FSM

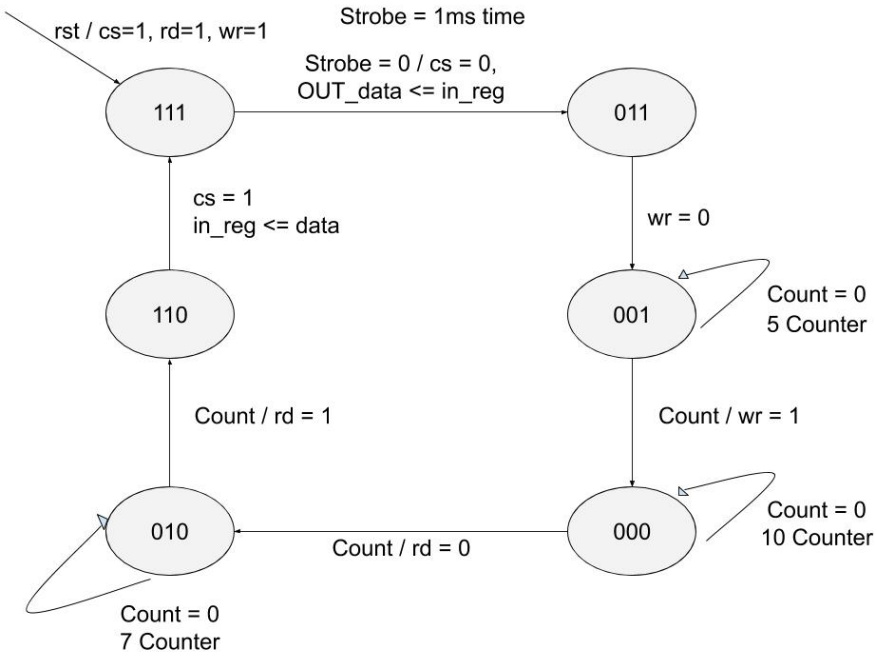

Figure 1: FSM of Part 1

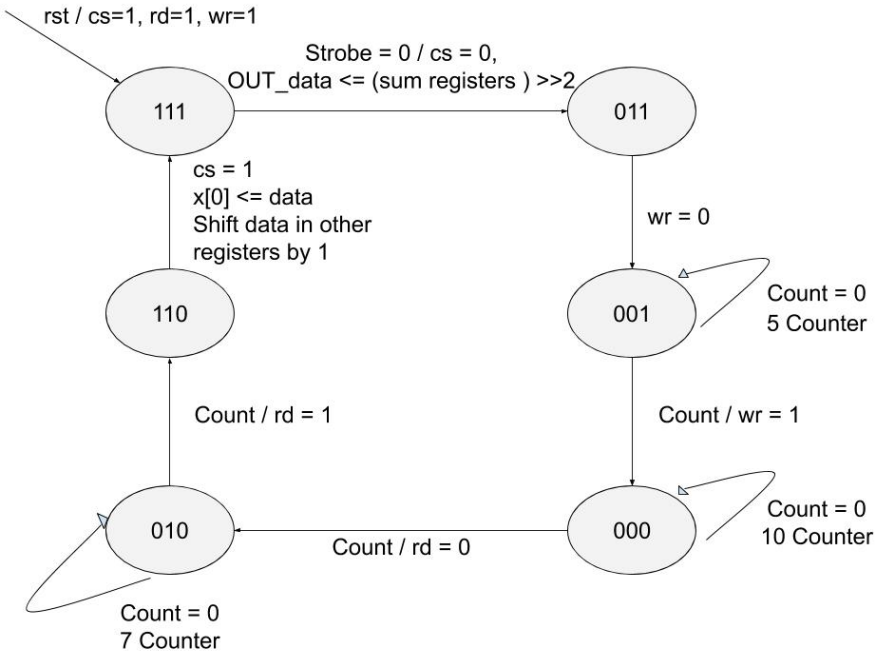

Figure 2: FSM of Part 2

4 Implementation

Our FSM takes reset (on board switch),intl (interrupt of ADC) and data (of ADC) as input. It produces output cs (chip select of ADC), wr (write input of ADC) and rd (read input of ADC). In our code we have used a countdown timer of 50,000 to get a pulse at every 50,000th clock cycle, which corresponds to every 1 ms as clock time period is 20 ns. This timer helps us sample and produce output at the required frequency of 1kHz.

We tried encode the states by their corresponding output values (the output observed when the given state was reached) with the exception of two states which although has same previous output as an existing state, they were actually different so randomly these exceptions were encoded. When the current state becomes 001 (cs wr rd), we increment the value of a register every clock cycle. When this register holds a value of 5, it means 100ns have passes with wr=0 so we go to next state which is 000 (actually 011 according to nomenclature defined but this is one of the exceptions as 011 is already a

state present). At 000 we wait for 10 clock cycles in a similar manner and then wait for intl to become 0. If so, we move to next state 010. We wait for 7 clock cycles and then move to state 110 (010 according to nomenclature, but its an exception). At state 110, we put the data of ADC into a register. This register transfers its data to output register at the next countdown timer pulse. Therefore, there is a full 1ms (actually 1ms minus the time in ns at start for writing,reading and converting data in ADC) delay in the output. This ofcourse can be eliminated if one directly uses the output of the penultimate register as the output data of the FSM.

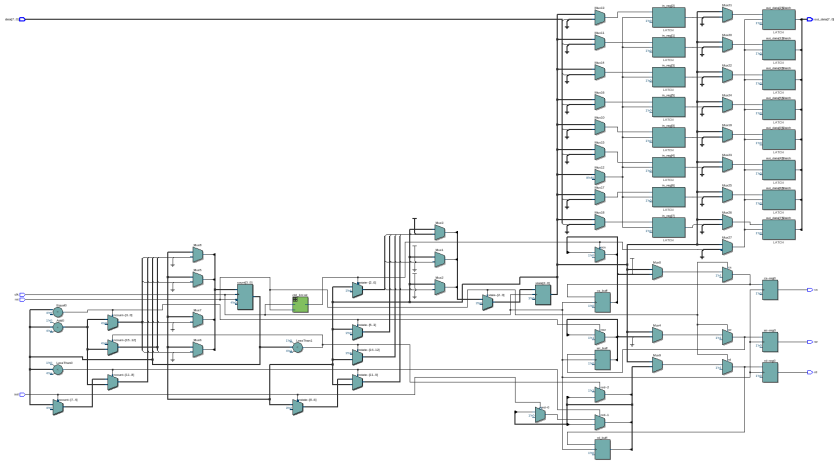

Figure 3: Circuit without filter

For the filter, I used 8 registers (for memory of 8), with each being of 11 bits. Now at state 110, the new data is loaded to a register while the previous data are all shifted by one register (thus, deleting the earliest data). The ADC output is an 8 bit number, but still I used 11 bits to store it because in order to do $\sum_{i=0}^{7} \frac{x[i]}{8}$, I first summed all 8 register values and stored them in another register (this is done in state 111) and this register needs to be 11 bits to avoid overflow. I was unsure whether VHDL allows sum of eight 8 bit vectors to be stored in an 11 bit vector (casting) so I used 11 bit registers to avoid any problem. The output of this register is fed into a 3 bit right shifter, basically a divide by 8 operation. Its output is the required ADC signal and is fed to DAC IC. Since the output is obtained as soon as FSM enters state 111, the 1ms delay is not there anymore.

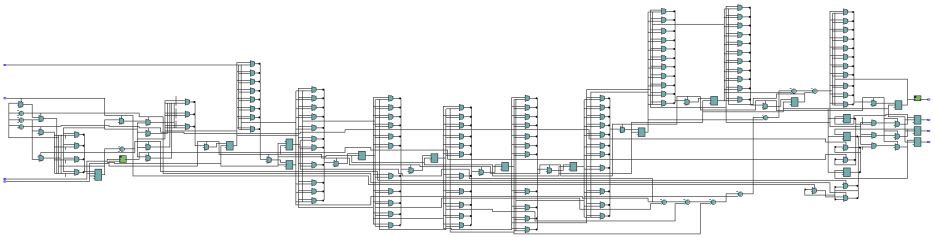

Figure 4: Circuit with filter

5 Observation

The following screenshot is simulation of filter. The register x_i hold the current and past ADC outputs.

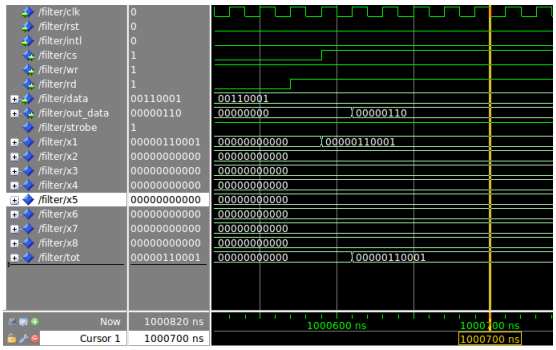

Figure 5: Very little delay

Frequency (Hz)	Output (V)
5	1.14
10	1.1
20	1.06
40	0.96
50	0.88
60	0.768
70	0.688
75	0.608
80	0.568
85	0.504
90	0.452
95	0.392
100	0.336

Table 1: Frequency Response

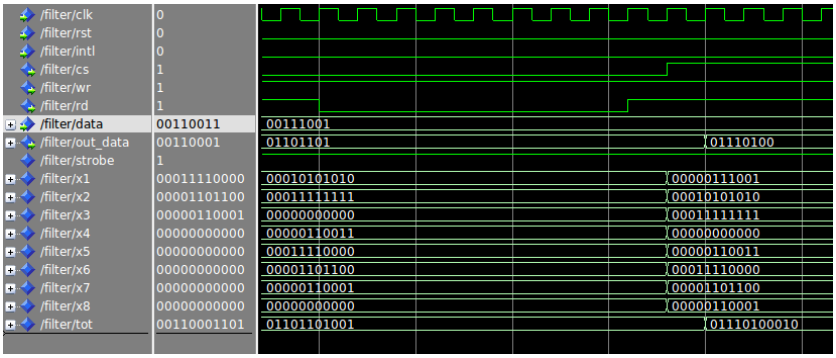

Figure 6: The registers of the circuit showing past values

We tried characterising the filter. Using the values obtained in table 1 with input peak-peak voltage of 1V, we can plot the frequency response. We observed that for frequencies above 100 Hz, there was too much noise/distortion to take reliable readings.

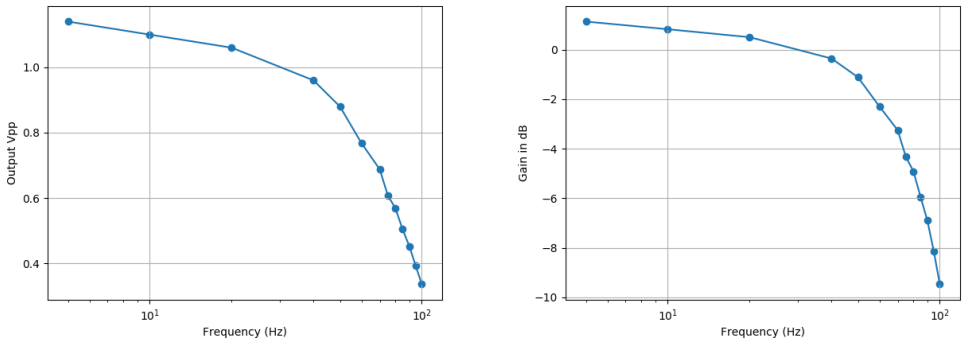

Figure 7: Frequency response of the filter

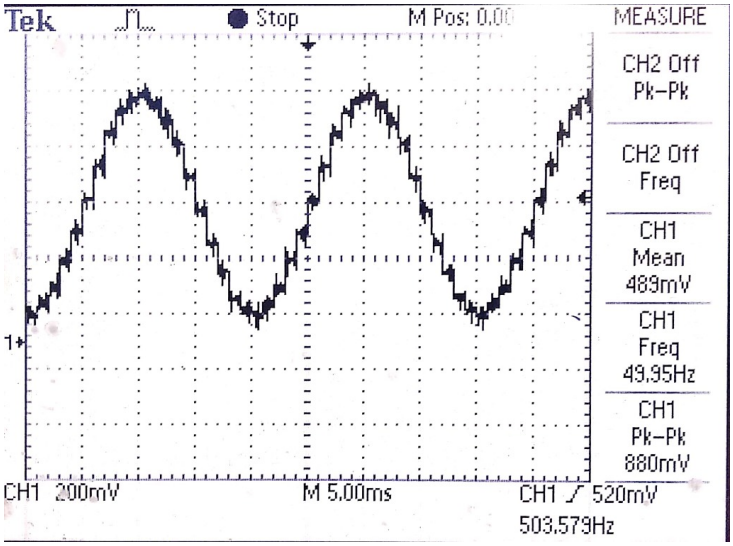

Figure 8: DAC Output distorted at 500Hz

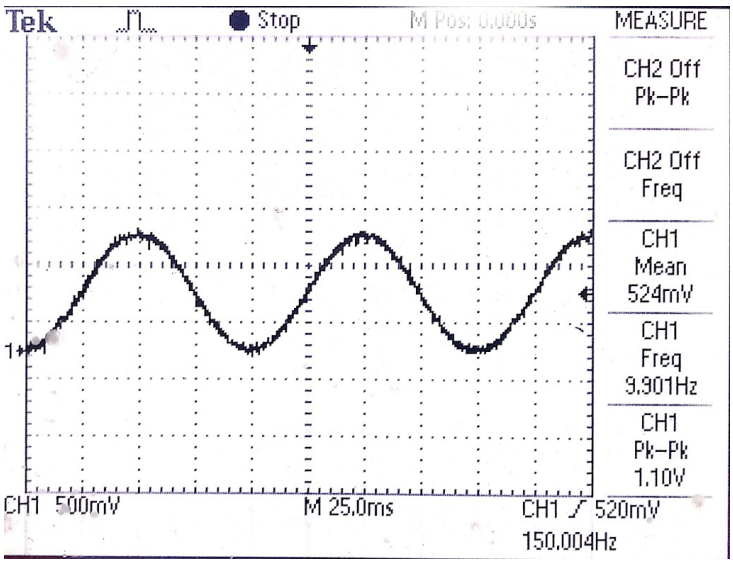

Figure 9: DAC Output at 150Hz

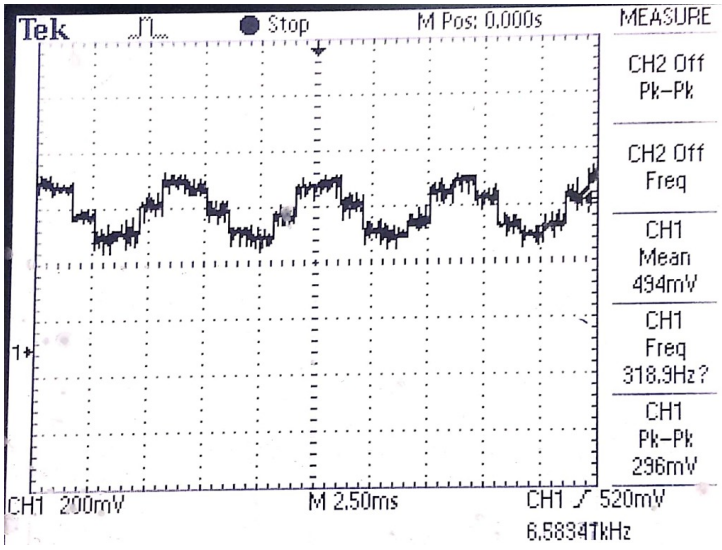

Figure 10: DAC Output distorted at 6.5kHz