Individual Project: ADC-DP

Author: Rishi Shukla

Introduction

This individual project presents one of the many components which will be used to optimise the digital signal processing algorithms to process a noisy power signal and derive the frequency by using an integrated set of Application Specific Processors (ASP) in a Heterogeneous Multiprocessor System-on-Chip (HMPSoC).

The ADC-ASP design is the foundation of this HMPSoC, and the following report outlines a brief background of the context, the ADC-ASP's design and features, and instructions on using this ASP.

Background

The frequency of a noisy power signal can be determined through the reference point detection method, which in our case, would be finding two max amplitudes of the wave as a point of reference.[1] The brief method of finding this frequency in the case of our microprocessor is to fetch and sample a power signal via the ADC, average the values using a moving average filter for the reduction of the noise, which is then sent to a correlator for additional processing and which later on is then sent to the peak-to-peak detector that aims to find the maximum values to finally calculate the time and ergo the frequency of the signal.

The Network on Chip (NoC) based on the Time Division Multiple Access - Multistage Interconnect Network (TDMA-MIN) allows data transfer between sources and destination nodes amongst many simultaneous transfers occurring without conflicts. A plus side of using the NoC is that it is fully time-predictable and allows us to process multiple processes simultaneously.

Our primary focus is designing a time-predictable HMPSoC that combines the software and hardware components and develops hardware/software design. We would want to build this as we can perform time-critical executions. Two main mechanisms can allow for this: the RE-CoP processor that was built before that will be integrated alongside the Nios to perform control-driven tasks, and the NoC, which can enable interconnected data transfers without any collisions and provides a predictable latency between any two nodes in the NoC which can help us deliver high performing frequency determination[3].

ADC-ASP Component Features and Objectives

The high-level overview of the ADC-ASP Component can be seen as follows:

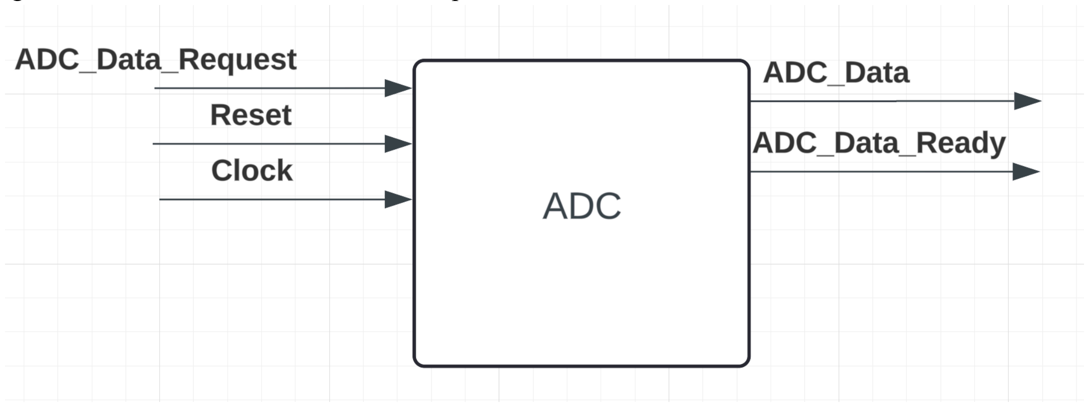


Figure 1: High-Level ADC-ASP Component

Inputs include the clock, reset, and an ADC_data_request, a pulse signal stating to ADC-ASP that new data will be delivered. The latest data generated would then be sent to the output ADC_Data. An ADC_Data_ready flag is enabled as data transmission occurs, letting the following component (the averaging ASP) know the data is prepared for use.

The pulse signal's frequency is dependent on the ADC sampling delay value supplied by the user, and the sampling rate can be derived as:

$$Sampling\ Rate\ = \frac{100MHz}{ADC\ Sampling\ Delay + 1}$$

Equation 1: Sampling Rate Equation.

Since the sampling frequency for the wave is 16KHz, the ADC Sampling delay is calculated as:

$$16KHz\ = \frac{100MHz}{ADC\ Samping\ Delay + 1}$$
$$ADC\ Samping\ Delay\ = 6249$$

Equation 2: Calculation of ADC Sampling Delay

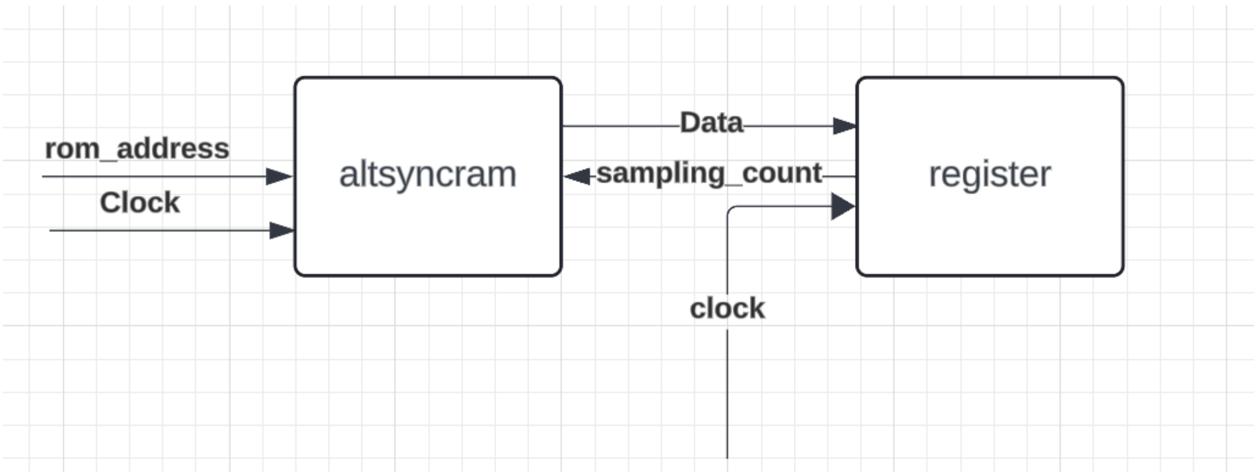
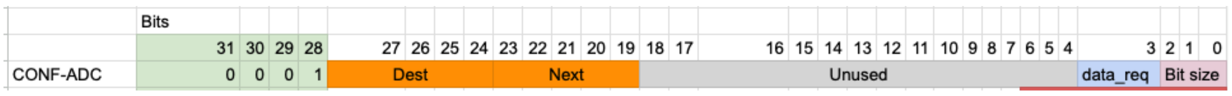


Figure 2: ADC-ASP component overview

Inside the ADC-ASP component(as shown in Figure 2), we have a generic component *altsyncram,* which acts as our read-only memory, with an input of the .mif file, which emulates a quantised signal which would either be of 8, 10, or 12 bits. This memory is accessed every clock cycle and only outputted when the ADC data is requested.

The following component would also interface with the NoC, TDMA-MIN, which would tell the ASPs the configurations necessary from the user. In the case of the ADC-ASP, the TDMA-MIN can send configuration messages that change the output of the ADC component, namely configuring the number of bits utilised by the ASP.



	Bits								
	31	30	29	28	27 26 25 24	23 22 21 20 19	18 17 16 15 14 13 12 11 10 9 8 7 6 5 4	3	2 1 0
CONF-ADC	0	0	0	1	Dest	Next	Unused	data_req	Bit size

Figure 3: TDMA-MIN receive data configure message

As shown in Figure 3, the NoC would let the system know that the ADC-ASP is being used with the bits highlighted 31 down to 28, with the bits following mapping out the destination and possible address to which the results could be forwarded. Bits 23 to 19 are significant as they would map out to the next component and give the data (which, in our case, would be to the AVG-ASP to average the results). Finally, the final four bits would be the configuration bits indicating the data size of the ASP, whether 8, 10, or 12.

This means that the following component would follow the following datapath:

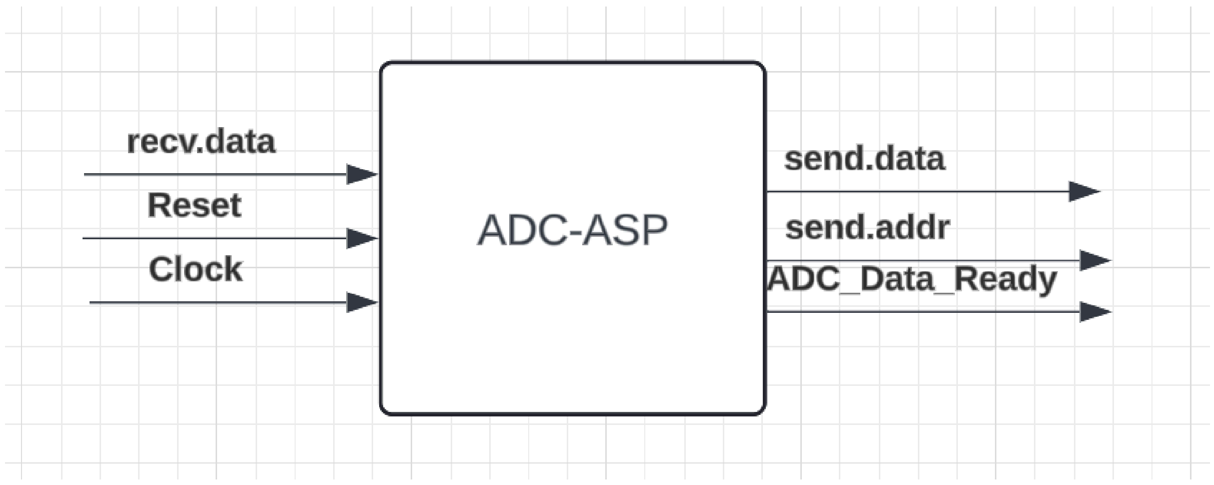


Figure 4: Revised datapath

The inside of the ASP would work just the same. However, it is notable that the data_request is now in the recv.data packet from TDMA-MIN, where a request bit would be enabled by the packet. As for the output of the ADC-ASP, only the bottom 16 bits will be accessed by the AVG-ASP, Ergo, future work is to have a viable code to let the next ASP know how to retrieve the data from the ADC-ASP.

Usage Instructions

The following project consists of four main VHDL files that are noteworthy:

1. Aspadc.vhd - The main vhdl file without integration of the TDMA-MIN
2. Test_adc.vhd - The testbench for the above code
3. Aspadc_tdma.vhd - The main vhdl file with integration of the TDMA-MIN
4. Test_adc_tdma.vhd - The testbench for the above code

To run the simulations and observe the waveform of an 8-bit quantized signal, follow these steps, ensure all VHDL files are compiled alongside other dependencies such as the TDMA-MIN. Run the test_adc file for 50 ms to generate and observe the waveform of the 8-bit quantised signal.

The input .mif file used in the aspadc or aspadc_tdma components can be changed by updating the init_file parameter inside the altsyncram component within the respective VHDL files.

The `test_adc_tdma` and `aspadc_tdma` files are integrated with the NoC. To configure ports for the TDMA, you can use the recv_port(0).data variable. Assign the last three digits to one of the following values to configure the data width:

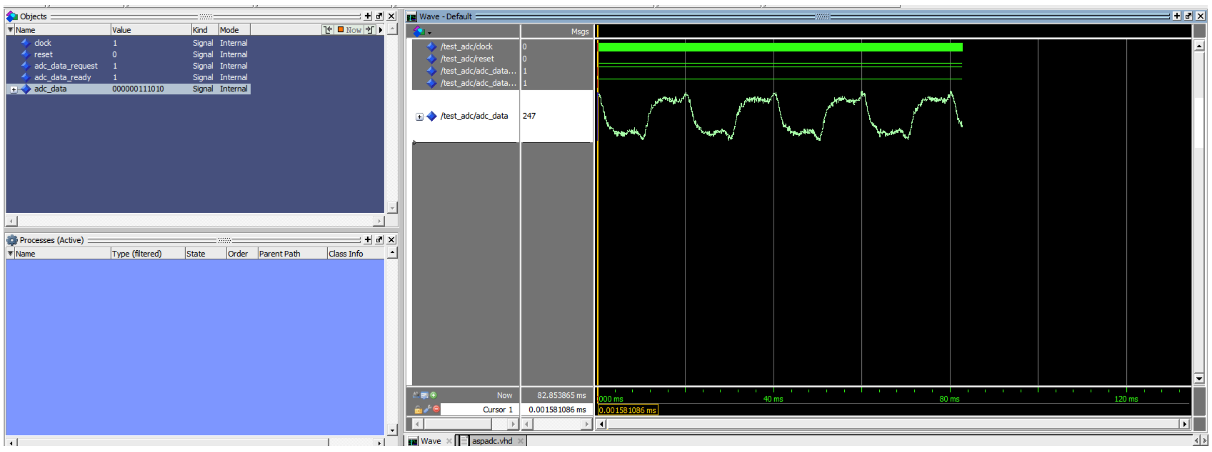
- ‘001’ for 8-bit configuration
- ‘010’ for 10-bit configuration
- ‘011’ for 12-bit configuration

To enable the data transfer, set recv.data(3) to ‘1’, and to disable transfer set to ‘0’.

For the quartus compilation, go to Lab2-Reference -> cs701 and load the project in Quartus using the .qpf file.

The Python code for generating the `.mif` files is in the `input-generator` folder. This code generates the samples of the modeled power signal, which can then be used as input for the VHDL simulations.

Results and Discussion



Name	Value	Kind	Mode
clock	1	Signal	Internal
reset	0	Signal	Internal
adc_data_request	1	Signal	Internal
adc_data_ready	1	Signal	Internal
adc_data	000000111010	Signal	Internal

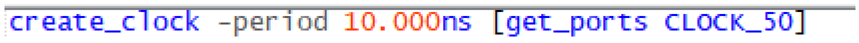
```
/test_adc/clock         0
/test_adc/reset         0
/test_adc/adc_data...   1
/test_adc/adc_data...   1
/test_adc/adc_data      247

Now        82.853865 ms
Cursor 1   0.001581086 ms
```

Figure 5: ADC-ASP output in ModelSim

As shown in the figure above, the ADC-ASP takes the input and successfully showcases the data-samples stored in the ROM, with easy configurability to showcase the 8,10 and 12-bit samples.

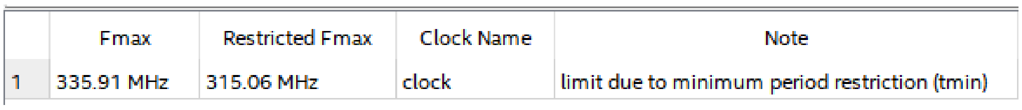
Timing Analysis

A timing analysis was performed where the clock was converted to a 100MHz clock, and a complete compilation was run to give the following results:



```
create_clock -period 10.000ns [get_ports CLOCK_50]
```

Figure 6: Clock changed to 100MHz



	Fmax	Restricted Fmax	Clock Name	Note
1	335.91 MHz	315.06 MHz	clock	limit due to minimum period restriction (tmin)

Figure 7: Operating Frequency of the ASP.

The component, as shown, has a max frequency of 335.91 MHz, which means that it's operating at a period of:

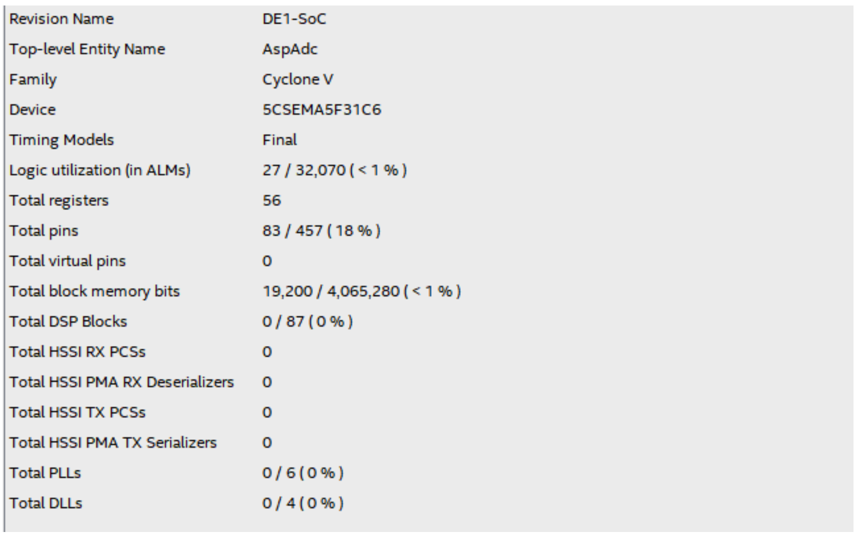
$$T = \frac{1}{F}$$

$$T = \frac{1}{335.91 MHz}$$

$$T = 2.985\ ns$$

Equation 3: Calculation of period

Resource Usage



Revision Name	DE1-SoC
Top-level Entity Name	AspAdc
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	27 / 32,070 (< 1 %)
Total registers	56
Total pins	83 / 457 (18 %)
Total virtual pins	0
Total block memory bits	19,200 / 4,065,280 (< 1 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 8: Hardware Resource Usage

Upon compilation, we can see that this component is highly resourceful and isn't a strain on the hardware portion, with only 27 Logic Elements being used. The total memory blocks could also be reduced if the altsyncram components' data width is configurable with the number of bits the signal is configured to; however, this is considered future work/implementation.

Conclusion

In conclusion, the above is a current implementation of the ADC-ASP, which interfaces with the NoC TDMA-MIN to receive and enable easy configurability of the data bits needing to be used and, later on, send data to other components in the ASP. The component uses minimal resources, is high-performing as an ASP, and can be easily integrated into the ASP.

Future work would be configuring the altsyncram with the data bits the received command gives and including a reset functionality. However, it is hopeful that a correct result will be obtained that provides a foundation upon which to build and later on integrate with the other components to aid the greater context of a high-performance frequency relay system.

References

1. Z Salcic, R Mikhael, 2000. A new method for instantaneous power system frequency measurement using reference points detection, Electric Power Systems Research, Volume 55, Issue 2, 1 August 2000, Pages 97-102, https://doi.org/10.1016/S0378-7796(99)00102-9
2. Salcic, Z., Nadeem, M. and Striebing, B, 2016, A Time Predictable Heterogeneous Multicore Processor for Hard Real-time GALS Programs. ARCS 2016
3. Salcic, Z., Park, H., Biglari-Abhari, M., and Teich, J., 2019, SystemGALS – A language for the design of GALS software systems, Embedded Systems Research Group, University of Auckland, Internal document, Embedded Systems Research Group, available to C701 class