**Journal**

**Exercise 3 HW/SW Co-design**

Date: **03/11-10**

Author: **Anders Hvidgaard Poder**

Stud.nr.: 19951439

# Introduction

In this journal I will answer the different questions posted in the text as well as my experiences with implementing an Architectural model in SystemC. However, before any of this can be done it is important to understand the design of the LAVMU model.

# LAVMU model breakdown (reverse engineering)

The first thing that springs to mind when looking at this design is that the IIRFilter is not in the data-path. Looking further into the code you notice that the Algo class implements the IIR Filter. Then what is the purpose of the IIRFilter class? We dive a little deeper into the PView class which uses the IIRFilter class to find out. It turns out that even though the IIRFilter class has the ability to filter samples, it is only used to generate the IIRFilter coefficients. Therefore the name IIRFilter is quite poorly chosen, and the name IIRFilterFactory would have been better, but as we only use the coefficients and not the filter an even better name would be IIRFilterCoefficentsFactory. Also the IIRFilter is purely composite under PView, so the class diagram could beneficially be updated to reflect this

The Top class creates the binding between the Algo and the PView, and also between the Algo and the Source and Sink Codec. This allows the data sent by the SourceCodec to be received by Algo and the data sent by Algo to be received by the SingCodec. It also allows the PView to transfer its coefficients to the Algo class. All this communication is done using abstract busses.

The Algo class accepts values from the SourceCodec and filter them according to the IIR filter implemented in the class with the coefficients received from the PView (who got them from the IIRFilter). The result of the filtering is then sent to the SinkCodec. The filter implementation in the Algo class does not include timing. This means that a filtration of a sample takes 0 time and can be done in one clock. Naturally it would be possible to determine the required computation clock cycles for the filter and thereby make the filter up to CCAM accurate, but this has not been done and the Algo implementation (model) is therefore untimed TLM. The Algo class has two clocks, the AudioClock, which is use by the SourceCodec and SinkCodec (12MHz) and a main clock (50MHz), which it shares with the PView.

Communication with the Source- and SinkCodec is done via channels. Here it may be seen that the SourceCodec uses an internal AudioClock of 12MHz, which it divides down by 125 to reach the 96kHz sampling frequency. Its main loop is executed with audio clock frequency (12MHz) and for every 125 loops a sample is read from the file and written to the Algo class channel. Furthermore synchronization signals are set to indicate to the Algo class that a value is now ready. In a real life example there is naturally no file and the sample is taken from an A/D Converter. As no extra wait is executed it is simulated that the sampling of the A/D and the writing to the Algo filter takes 0 time units, but the sampling is Cycle accurate, as it is done on a divided clock, just like the actual implementation would be. As the Algo filter has no computation delay it would also make very little sense to include the delay, as it is negligible compared to the filter computation. However if correct waits were added to Algo to make it computation cycle accurate, then it would be important to include the sampling and communication delay to see the total Source to Sink delay.

# Assignment 3.12.1

TLM is a system level model and is in SystemC a matter of augmenting the MoC (in SystemC) with timing constraints. The TLM may be made timed. This may be done by augmenting the code with wait statements for the different blocks. At the lowest level of implementation it should be for every Basic Block, but it may be done by simply estimating how much an entire transaction will take and then place the wait after that. It is possible to extend the TLM with more detail, like “implementing” the communication busses in SystemC and then augmenting this implementation with timing.

CAM is a model that is accurate down to the individual logic level changes, and is therefore accurate to the lowest timing level. A CAM can be simulated and be fully accurate compared to the actual execution an FPGA or ASIC.

In the SystemC class diagram the IIR algorithm is implemented in the Algo class and that no consideration is given to execution time of the algorithm, making it an untimed TLM. The interface to the IIR algorithm is implemented in the Codec Source and Codec Sink. Here it may be seen that the sampling of the input signal, and the reception of output from the Algo class is clock accurate, meaning that it is done on the AudioClock, sampled down to 96kHz, making it BCAM accurate on the A/D converter sampling. The CodecSink awaits data from the

CodecSource and CodecSink is AudioClock accurate, meaning that the timing in the communication is accurate down to the clock level, i.e. BCAM – the difference between CAM and BCAM (Bus CAM) is that BCAM only include fully accuracy on the communication level. The computation part is modelled in CCAM (Computation CAM), and CAM is the combination of BCAM and CCAM. The IIR algorithm is modelled in the Algo class, but with no timing what so ever. This makes it TLM (not timed TLM), as it only model the functionality not the timing. Later in the design process this implementation might be augmented with timing, first perhaps just from estimation or form testing the algorithm on the actual HW and then adding this number, and even later a CCAM version may be created making the model fully accurate (but also a lot slower).

The PView module is an abstraction of the HW/SW boundary. It is used to communicate from the SW to the HW (setting the coefficients and gain) and form the HW to the SW (getting the peak value).

# Assignment 3.12.2

After reading the assignment I decided that since it called for a replacement of the IIR filter the easiest approach was to simply duplicate the LAVMU\_Model project and then work from that.

However first we need to complete the design. The design is created by using the design from Assignment 3.12.1 and modifying it accordingly. Firstly we replace the IIR filter with the LMS filter and adjust the interfaces to supply the configuration required by the LMS filter (convergence factor) and not the coefficients required by the IIR filter

The LAVMU\_Model already reads it input from two files, representing the left and right channels of the stereo signal respectively. The LMS filter takes a noise and noise+signal input, but as there is no requirement that the filtered signal should be a stereo signal, we can simply assume two mono microphones. Alternatively we would need two CodecSources and two LMS filters and then a mixer, but that is not what we are going to do ☺

This gives us the design below:

Naturally to do this design we should draw a lot more diagrams. A Block diagram and a few inner block diagrams would be an absolute minimum to show ownership, dependencies and communication channels. Furthermore a sequence/activity diagram or FSM to show the sampling and finally naturally a deployment diagram to show the HW/SW layout. All of these things will however be left as an exercise for the reader ☺

as described we get the following possible SystemC class design:

# Conclusion