System Level Design of a Multi-channel ADPCM Codec using Single Resource Co-processor

Dinesh Anand Bashkaran, Graduate student,Department of Electrical Engineering ,Rochester Institute of Technology,Rochester,New york
Email: dab8730@g.rit.edu

***Abstract*—Adaptive differential pulse code modulation is a technique for speech compression. The ultimate idea is to reduce the bandwidth of the signal that should be transmitted. This paper talks about how hardware and software is implemented to do this. This is a direct hardware where the description is followed; the hardware is fixed point arithmetic to do the computation for 32 channels Encoder/Decoder. The paper also talks about how software is used to control the hardware. The importance of this application is used in communication industry where the server or lines are used to serve many customers without affecting the quality of the voice signal. The theory behind this technique and the evolution of this technique is also discussed in this paper.**

***Index Terms*—ADPCM, AMBER, A-lAW, U-LAW, SINGLE RESOURCE.**

INTRODUCTION:

The ultimate aim of the project is to reduce the bandwidth of the signal that is being transmitted. This was done in few ways before but then the accuracy and the quality of the signal on the receiver is not appreciable. Little other technique evolved over time but then showed ADPCM which used the difference in the signal and transmits the bits, which consumes less bandwidth but also provides decent quality. The other technique and the evolution of them Is discussed in theory part.

I. THEORY:

The analog signal is converted into digital binary numbers in sampling methodology. This involves sampling, quantizing and binary encoding. When it comes to sampling the signal, Nyquist sampling theorem comes into picture which defines the sampling rate should be least twice the highest frequency contained in the signal. The reason behind is, it enables the signal sampling in appropriate way and also the data generated from this could be easily reframed to get back the analog signal. When analog signal is sampled, it gives series of pulses with varying amplitude between two limits. These signals are quantized, where the sampled analog signals is converted into finite range of discrete values and the vales are encoded in the encoder. There is a set of range, where the encoder generates the encoded words with respect to the codes that matches. There exists some range where the encoder does not change the code, which is referred from quantized error range. The figure 1, shows how the analog signal is covnerted into digital signal using the differenet components explained before. This happens on the end where the analog signal is sampled and such and then transmitted as the digital signal. This basically saves the energy and it also acts like compression technique with less loss in data. However, the Nyquist theorem must be followed to avoid the data loss at the receiver end.

Fig. 1. Analog to digital Encoder

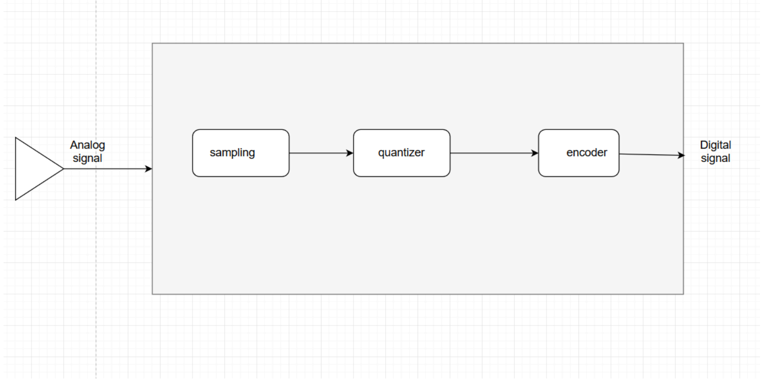


Fig 2, show the image of the decoder that is on the receiver side, where the discrete digital data is converted back into the analog signal. There is no much loss in data and the original analog signal is obtained back without any considerable amount of loss and it provides good results. The bit rate of the pulse coded signals is always the product of the bits per sample times the rate at which sampling is being done. The encoder used decides the bandwidth that is required to transmit the rate. It is generally, the rate at which the encoder could convert the data, decides the throughput and that decides the bandwidth of the output line. However, the discrete digital signals always need more bandwidth than the analog signals.

Fig 3, shows the transmitter and the transmission path and the signal at the receiving end. This is pretty much how it works in real time.

So, we have seen how the analog signal is coverted into digital singal but also we noticed that required bandwidth is always higher. We did not compress the data into maximum possibility yet, besides the bandwidth it takes also stays high. So we should probably get to reduce the data being transmitted into further smaller. Talking about this, the delta modulation comes into picture, where the data sent is only the difference between the pulses. For instance, if the signal at time T is higher in amplitude and the signal in time T+1 is lower, then the difference of 1 is being transmitted. Again it depends on the range it varies from higher to lower or lower to higher which changes the sign of the difference. But then this works when the difference is 1 but not when the

Fig. 2. analog to digtal Decoder

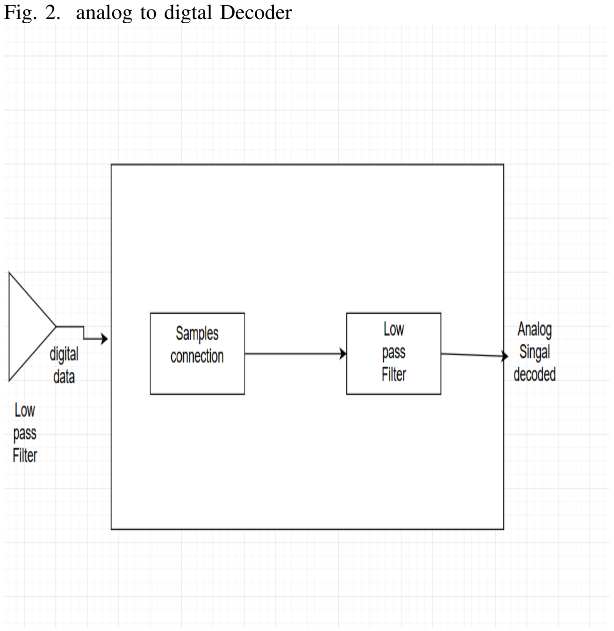
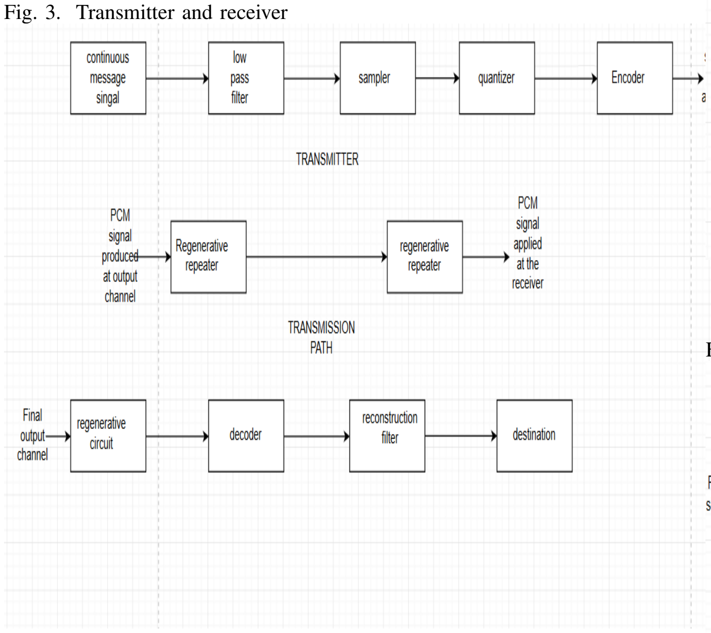


Fig. 3. Transmitter and receiver



difference is more than 1 or when the amplitude varies. Also the noise and distortion could make the signal reliability very less effective. Delta PCM, which uses more than one bit to refer the difference between the signals. Each bit represents the difference in the value between two adjacent signals being sampled. This increase the accuracy at which we could get the original data back. Adaptive delta modulation, the bit that changes in this method is not fixed. The bit varies with respect to the amplitude of the input analog signal in the encoder. This has very high dynamic range due to variable step size. The bandwidth is better utilized as compared to delta modulation. But again we have improved method which is known as differential pulse code modulation, which send the original bits and the changes with respect to the changes in amplitude. The difference being sent is with respect to the sign. This is further optimized into adaptive differential pulse code modulation, the encoded difference is either 2,3,4 or 5 bits range. This is generally a predictive coding scheme that codes the difference from the expected value.This technique takes the advantage of the fact that neighboring signals are similar to each other, this makes it predict the signal. The difference between the prediction and the actual value is only sent. The ADPCM technique produces lower bit rate than PCM.

Fig. 4. ADPCM ENCODER

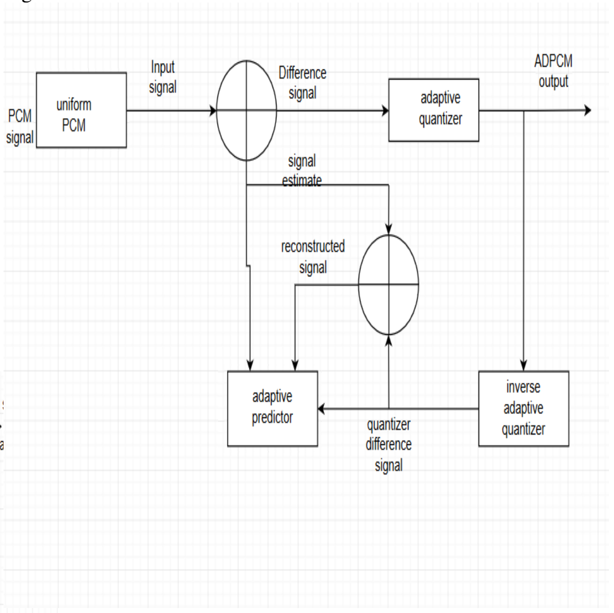
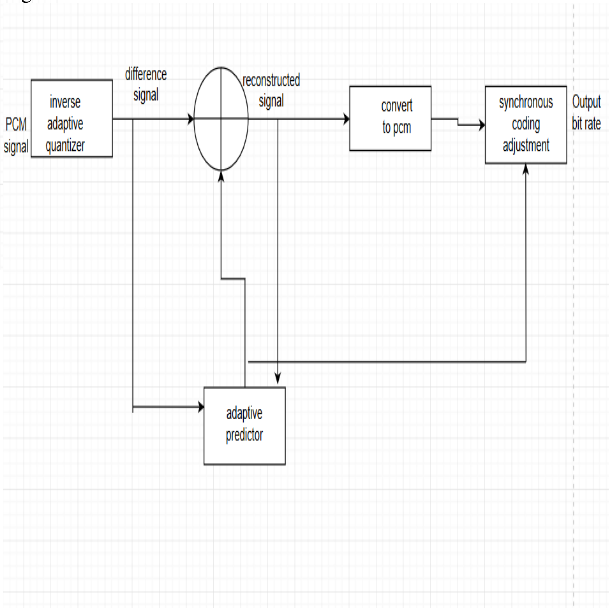


Fig. 5. ADPCM DECODER



II. Test Vector generation:

As discussed earlier, the hardware is made of several individual components. Out of them, co-processor is the one that acts as ALU or the computational unit for the majority of process. The amber could do the computation too but to keep it fast and simple, we show up with co-processor. So basically, we design a hardware that does the computation function as well other hardware units that works along with this unit. It is know that any computational unit is tested with input vectors to test it functionality. So talking about the gate level, the set of inputs are changed and the output response is noticed at every instance to check the functionality of the gate. The number of input vectors fed depends on the test coverage. However, 99% is generally preferred and usually random combination of test vectors are selected to avoid the repetition of same response at the output end. Other point to notice is that, the testing done here is to check test the functionality of the hardware synthesized and not the verification that is done to check the physical design of the hardware. Verification usually comes after testing and hardware being manufactured. Verification using involves stuck at faults. But as of now, only testing is being performed. So, in order to have a set of test vectors that test the functionality of the co-processor, they are being generated first from C model. Co-processor contains several small modules and FMULT-ACCUM together APRSC. The generated test vectors are fed into all low level modules and the FMULT-ACCUM. The modules perform its desired function on these testing variables and provides the output. The output is being verified for every input vector being fed into the module. All the low level modules and APRSC is checked functionally by verifying the passage of all test vectors.

III. System Overview:

The figure 6 shows the top level view of the encoder and the decoder. The encoder takes the PCM data of 8 bits, the config modules decides the rate and the law. The amber and co-processor performs the computation and the ADPCM is signal is given out as the output. The decoder takes the ADPCM signal the sofware controls the amber to use the same hardware but perform PCM output. This is the high level view, where the operation of the entire block is simply explained. Further down, every single block is explained in detail.

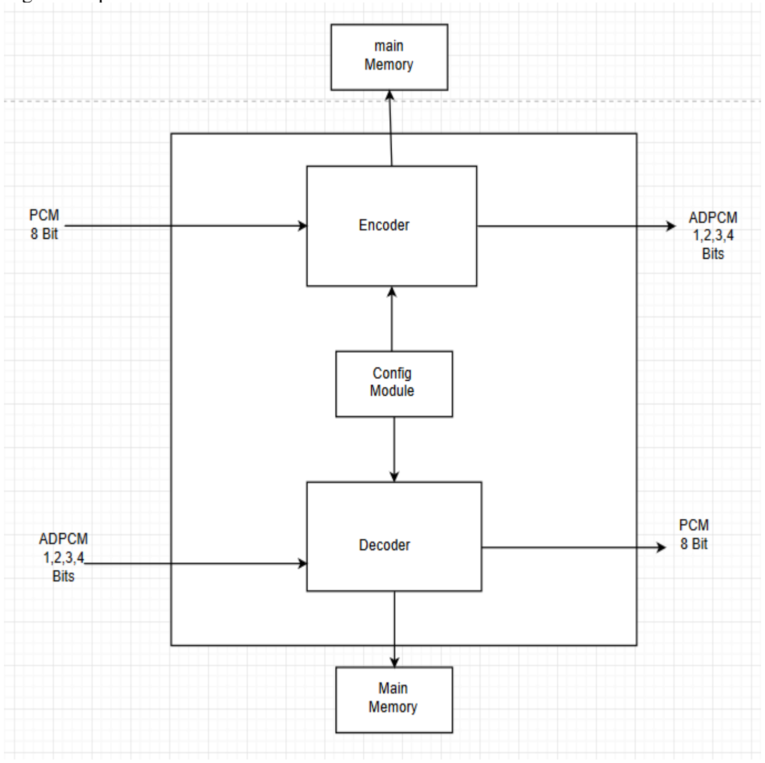
A. Hardware Flow :

The entire architecture consists of different components that do its own functionality. The Top level view of the desired hardware is shown below in the Figure 10. The hardware components in the encoder and the decoder are pretty much same expect the software that decides what instruction to perform on the hardware. So talking about the components, in the order of data flow TDMI comes as the first unit that takes the PCM or ADPCM as input.

B. TDMI:

Basically irrespective of the type of input, the input fed is always parallel data. The input is of the size 8 bits. The parallel input data should be converted into serial data and store it in the memory for computation process. TDMI is the module that takes in the input parallel data and converts it into

Fig. 6. Top level view of encoder and decoder



serial data. TDMI works on input channel clock, input frame sync and wishbone clock. Input channel clock is the clock with which TDMI converts the data. Basically, every input channel clock could possibly have 1 bit data which has to be converted. Input frame sync is the synchronize signal that tells the TDMI when to start the conversion function. This Frame sync signals, helps in keeping the data synchronized with other modules and the primary output unit. Wishbone clock, Is used by TDMI to access wishbone and store the data in the main memory. The functionality of TDMI is basically, taking the input, converting into serial data and storing it in memory. Wishbone is the interconnect to connect different modules and it is controlled using arbiter which acts as multiplexer giving access to the units which requests. We will talk about it in detail in further down the paper.

C. TDMO:

TDMO is the module through which the primary output of the block is provided. This works on output channel clock where the data is passed for every cycle, the output frame sync decides what time the process should be started and then the wishbone clock to acquire data from the main memory. The working is again simple; it just converts the serial data into parallel form and gives it as primary output.

D. AMBER:

Amber 25 is used here, it is a five stage pipelined ARM RISC processor core. It is open source code which makes it flexible. The five stage pipeline includes instruction fetch, instruction decode, execution, memory and write back. The

Fig. 7. TDMI

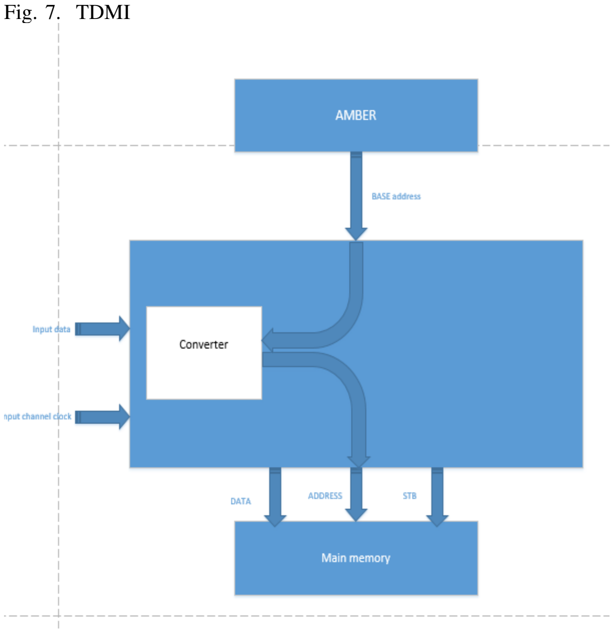
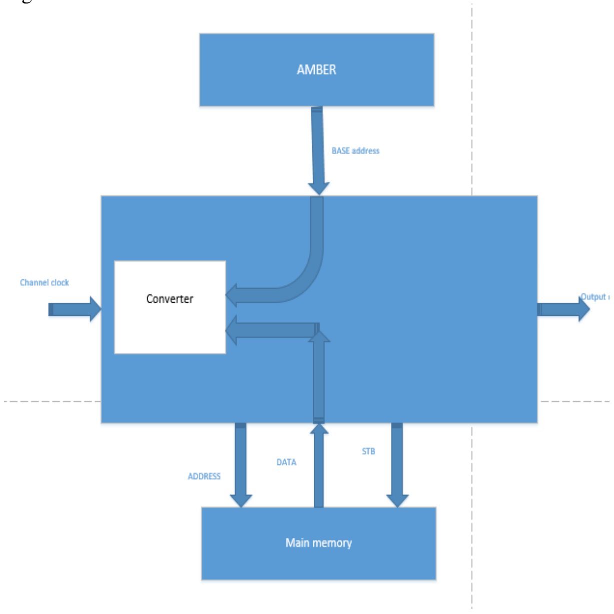


Fig. 8. TDMO

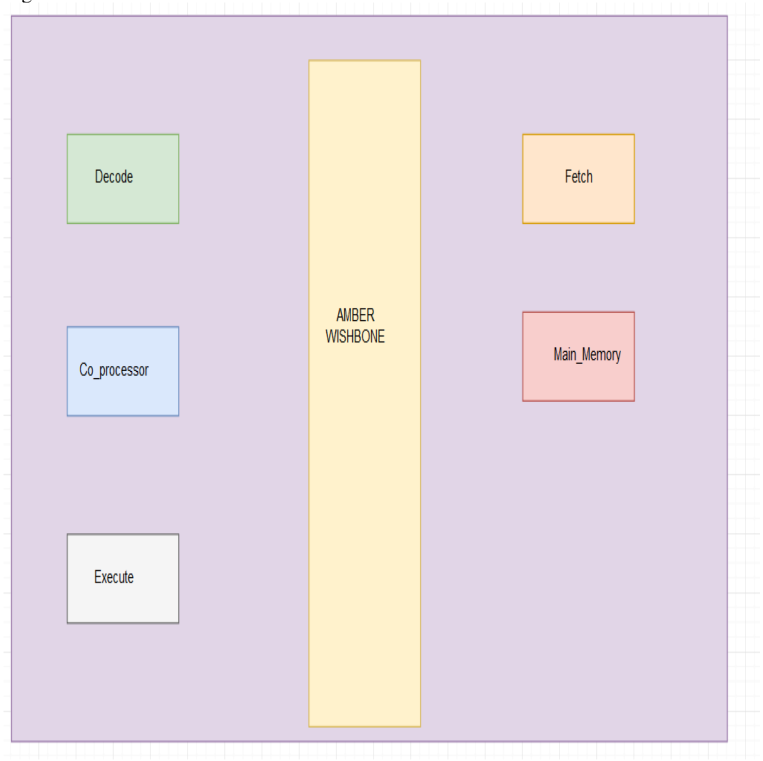


amber is good enough to run the PCM or ADPCM to convert them accordingly but then amount of time taken is extremely high and it gets complex. The amber was coded in Verilog and made to run on FPGA. Due to this low throughput, we need a other hardware that does the computation process.

E. Arbiter:

The Arbiter, is the unit that controls the interconnect buses between the various modules. All the modules are connected through wishbone bus and these buses are controlled using arbiter. This is basically a multiplexer that selects what line should be connected. But again, the arbiter is coded in Verilog and synthesized to make it into hardware implemented. Master is the unit that initiates the communication and the slave

Fig. 9. AMBER low level



is the unit that responds to this. The protocol for arbiter is simple. Master gives stobe signal to the wishbone, the arbiter finds what slave the master is trying to communicate from the address of the slave. The arbiter then gives this stobe to the slave. The slave responds on this stobe by giving back acknowledgment back to the wishbone. The slave gives acknowledgement only when the slave is free or not communication with other module already. This acknowledges sent to the wishbone; will be sent back to the master. Once the stobe and acknowledge goes high, the two modules are connected and the data lines of these modules are connected after which the data could be read or written. This is basically the protocol for hand-shake technique.

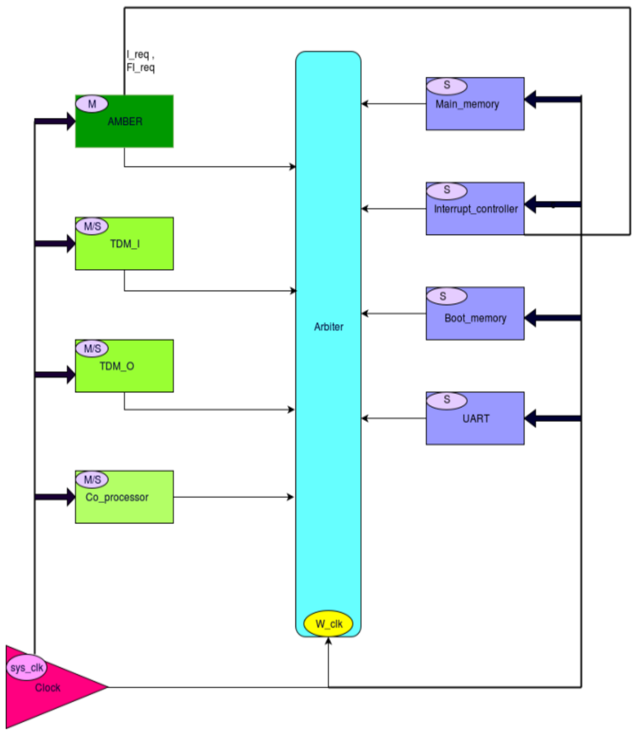
F. Boot memory:

Boot Memory, holds the instructions for the amber. The boot memory was designed for FPGA. The amber holds its own instruction set to being the communication with the boot-memory.

G. Co-Processor:

Adaptive predictor and reconstructed signal calculator is the co-processor unit that works along with amber to do the computation process of the input data. The major part of the computation is done in APRSC because the hardware is designed to do this in a faster way to give high throughput and have less latency. The architecture remains the same for encoder and decoder, but the instruction from amber varies accordingly. This is basically a hardware controlled by the software. The co-processor is fixed point hardware with G.726

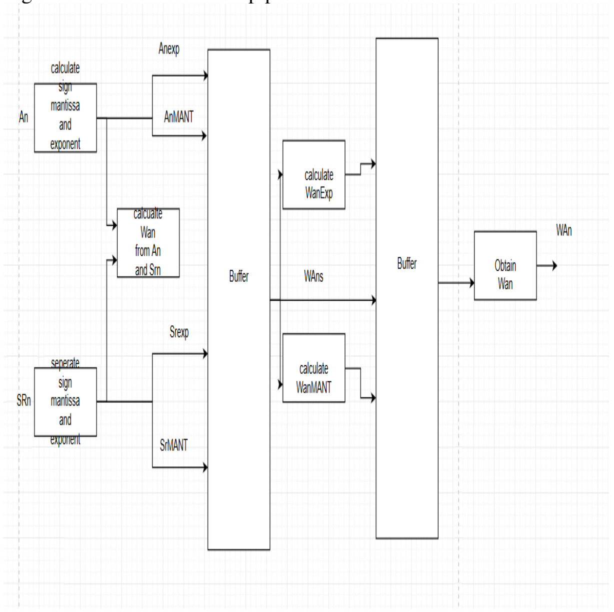
Fig. 10. Wishbone Arbiter



standard. It has 6 all zero filters and 2 all pole filters. The co-efficient of all pole filters is A1 and A2, co-efficient of all zero filters is B1 to B6. These signals are multiplied with the DQ signal and SR signal to generate signal estimate (SE) and partial signal estimate (SEZ). The computation done by FMULT is floating point which is the part that takes more clock cycles. This is the worst case delay in this FMULT unit. So, now it's very obvious why software takes long time to compute floating point operation. The variables are then obtained in fixed point variables. The hardware constrain is single FMULT-ACCUM, that is the multiplier and adder is physically present by number of one but the operation it should perform for each channel is 8 times. So the variables are fed into it like a cycle one after the other.

1) FMULT-ACCUM:: Like discussed before, the APRSC consists of 8 floating point multiplications. The operation performed by FMULT should be parallel to avoid the time consumption and keep the timing budget in range. To perform the multiplication for one set of variables it takes 3 clock cycles. Again the variables should be fetched from the main memory and fed to the FMULT unit. To make the operation faster, the FMULT was divided into 3 stages, which is shown in the figure 11. Basically, 16 bit signed bit signed variables Bn and An are converted into floating point signed bit which contains 4 mantissa and 6 exponents. SRn and DQn needs to be broken down as they are represented in floating point. The floating point operation is performed to obtain WAn and Wbn and adding exponents to WAnEXP/WBnEXP and then multiplying the mantissa to obtain WAnMANT/WBnMANT. The worst case delay is considered from mantissa. The ACCUM is basically a adder that performs A=A+B. It adds FUMULT output to the pervious output. Overall the FMULT-ACCUM takes 14 clock cycles to the operation when variables are ready, but then considering the wishbone operation where the data should be fetched from the main-memory it increases the number of clock cycles the entire FMULT-ACCUM takes. Generally, one wishbone read/write takes over 3 clock cycles. To have 8 variables being written and read and then the operation, it ends up in roughly around 50-60 clock cycles.

Fig. 11. FMULT ACCUM pipelined


An
calculate sign mantissa and exponent
Anexp
AnMANT
calcualte Wan from An and Srn
SRn
seperate sign mantissa and exponent
Srexp
SrMANT
Buffer
calculate WanExp
WAns
calculate WanMANT
Buffer
Obtain Wan
WAn

H. Main Memory:

The main memory is separate hardware for encoder and decoder. The size of the memory is of 128 Mb. The memory is portioned for individual modules and then used accordingly. This is the block where the data is stored and read back. The units that access main memory are TDMI, TDMO, AMBER, CO-PROCESSOR.

I. Configuration Module:

This is the module that holds the information about the rate and the law. The architecture of this module consists of 32 bit registers, mux and mailbox registers. These are separate for encoder and decoder. Mailbox register is used to synchronize data from test bench and internal modules. rgy.

J. Regression System :

The regression system Is used to store the result of the individual modules in the SQL database. The Perl scripting is used to store the generated results into the SQL database. Information like area, slack, test coverage, power consumption is noted and compared to check the performance.

K. Timing Requirement:

One of the constrain of this project, is to process the channel within the time scale of 3.9 Ms. This means, the point from which the data is taken from the main memory and then the processing of those data along with the data fed back into the memory should happen in 3.9 Ms. To meet the timing requirements, the Encoder should be tested to produce the data first and check the current scale and vary the frequency of the module accordinly to bring the scale down to 3.9 Ms.

IV. Methodology:

A. Design:

This is a complex hardware project, though the hardware is being controlled using software. The design is top-down approach, which means the top level has to be understood and then start working from the bottom level to reach the top. To do this, the lower level modules of the co-processor is designed and verified with test bench. Then these modules along with APRSC are made to get a mid-level view. There are certain other mid-level modules that are independent of co-processor like TDMI, TDMO and wishbone which has its own functionality. These units come into mid blocks. Once, all the mid-level blocks are ready they could make up to high level view of coder by just connection them through wires.

B. Testing:

Testing is already discussed, the low level modules and APRSC inside co-processor is tested using test vectors generated from the c-model. The other mid-level blocks like TDMI, TDMO, Wishbone, Configuration module is tested using behavioral test bench where the input is fed and the output is checked. Synthesis: All the modules are made in 65nm library. The low level modules inside co-processor were synthesized at 200 Mhz, the FMULT was synthesized at 500 Mhz. TDMI, TDMO, works on their input channel clock frequency and synthesized at 2.048 Mhz.

V. Results:

The encoder software was simulated for 32 channels. The project was not successfully completed to meet the requirements. The project was not too complex to understand, but it required some set of skills which consumed time like scripting, understanding the architecture, pipeline technique, verification methods. In major part of the project, testing was the issue. For modules that does nottest vectors, the vectors are fed but for other modules, the setup of test bench and verifying the test bench consumed time because it is critical to have appropriate test bench to check the RTL. As of now, the mid-level blocks are synthesized, low level blocks are synthesized. They are simulated to perform their function. The APRSC is tested without wishbone but the code is written for wishbone interface and now the APRSC is working with wishbone interface and is able to throw data into memory and read from it.

VI. Conclusions & Discussion:

The project involved top down methodology for designing hardware and a software model. The modules designed were tested in official ITU. The Project involves various modules communication with each other through wishbone for every channel and keep the track of time and data in synchronized way is the biggest challenge which we didn't get chance to even face it. But I believe, this helps in understanding the working of modules when it comes to function of block. This also helps in understand how to operate the modules at different frequency and understand the behavior of modules through wave forms. With respect to the area, the FMULT consumes major area and clock generator that generates high frequency clock consumes more power.

Acknowlegment

I would like to thank Mark. A Indovina who helped me and my teammate in getting fair amount of work done. I would also like to thank my teammates Vedant, Manikandan, Tejas, Himaja, chankiya, Rakesh who helped me in understanding the project and getting the project closer to reach the goal.

References

[1] ITU-REC-G.726-199012-Spec-1990-E1951
[2] ITU-REC-G.711-198811-E7107
[3] Wishbone B4 Specifications
[4] Google- random sources for verilog

About the author

Dinesh Anand Bshkaran is a graduate student currently pursuing Masters in Electrical Engineering in Rochester Institute of Technology. His Fields of Interest are Design of Digital Systems, Embedded Systems, Power Electronics and his hobbies are Gym, Martial Arts.