Implementation of a Digital Down Converter Using Graphics Processing Unit

Xiao Ma, Lixia Deng, Yuping Zhao

Peking University, Beijing 100876, China
andy.maxiao@gmail.com, denglixiapku@gmail.com, yuping.zhao@pku.edu.cn

Abstract: This paper presents a DDC (digital down converter) on NVIDA 580 GTX, which consists of a DDS (direct digital synthesizer), a CIC (cascade integrator comb) decimation filter and a FIR (finite impulse response) filter. The decimating factor of the CIC decimation filter can be arbitrary positive integer and the major concern is concentrated on how to drive it to work well while the decimating factor varies. In our strategy, the problem is ranged into two cases according to whether or not the decimating factor is smaller than 128. Then, we provide suitable method to deal with data in each case. Additionally, we present different ways to construct a root raised cosine filter with 4 times oversampling on GPU (graphics process unit). Through flexible threads assignment and efficient scheduling strategy, the GPU-based DDC is implemented significantly. We evaluate the performance of the designed DDC with respect to its counterpart based on CPU (central processing unit) developed in the C language. Experimental results demonstrate that the DDC shows significant improvements on GPU and achieves a speedup of 387 times.

Keywords: Graphics processing unit; digital down converter; cascade integrator comb

1 Introduction

Nowadays, data explosion and complicated algorithms bring a lot of challenges in digital signal processing. If FPGA (field programmable gate array) and DSP (digital signal processor) are used, concrete programming skills and debugging experience are required. Thus, researchers are looking for a new processor, which is powerful and easily developed. Recently, Graphics processing unit (GPU), a typical kind of multiprocessors becomes widely used for high-speed computing recently. GPU is capable of producing 1 TFLOP (tera floating-point operations per second) and can be developed in the C language easily. Due to its great convenience and computing power, GPU becomes to be used in digital signal processing of wireless communication systems. Although multi-core central processing unit (CPU) is another alternative, it is often not powerful enough in some applications and using CPU will decrease the speed of the other tasks originally assigned to the CPU of the computer. Many studies have been done and prove that some algorithms in communication systems can be greatly improved on GPU compared with CPU and FPGA [1-4].

978-1-4799-0077-0/13/$31.00 ©2013 IEEE

In communication systems, DDC (digital down convertor) is a crucial part and converts a digitized real signal centered at an IF (intermediate frequency) to a baseband complex signal centered at zero frequency. A DDC consists of a DDS (direct digital synthesizer) and a LPF (low pass filter) with a down sampler integrated into it [5]. In addition to down-conversion, DDC decimates input signal to a lower sampling rate and allows follow-on signal processing by lower speed processors. However, the digital LPF contains lots of multipliers with high working frequency and occupies massive resource and high power consumption. Thus, current DDC implementation proposals adopt a CIC (cascaded integrator comb) filter, which is very efficient and economical to deal with decimation and interpolation [6]. Then, a simple FIR (finite impulse response) filter follows the CIC filter, which can work at lower frequency and save resource. In [7], Amit Upadhyay and Yawatkar Shakun Rajan implement a DDC on GPU in GMRT (giant metrewave radio telescope) program and observe that GPU gives about 250 times enhancement in the processing time compared with CPU. However, the GPU-based DDC is designed with a fixed decimating factor and only fits for the GMRT program. To our best knowledge, there is no other research about how to deal with different decimating factors in GPU-based DDC.

In this paper, we implement an efficient GPU-based DDC on 580 GTX with the help of CUDA (compute unified device architecture) SDK (software development kit), which is provided by NVIDIA Corporation to compile the code and to load the compiled code into the GPU [8]. As shown in Fig. 1, Our GPU-based DDC consists of a DDS, a CIC decimation filter and a FIR filter. The DDS generates a complex sinusoid at the intermediate frequency on GPU and performs multiplication of the intermediate frequency with the input signal. The FIR filter is a simple root raised cosine filter with 4 times oversampling and the CIC decimation filter converts the signal's sample rate to 4 times to fit the FIR filter. The decimating factor can be arbitrary positive integer. That is to say, our GPU-based DDC is capable of changing the coefficients and length of the CIC decimation filter automatically without loading new compiled code into GPU when the decimating factor varies. Additionally, we show our strategy about how to make it. Besides, we also present two efficient ways to implement the FIR filter on GPU.

The rest of this paper is organized as follows. In section II, brief overview of NVIDIA GPU and CUDA is provided. Section III presents details of our

GPU-based DDC and our innovation strategy to achieve the goal of a variable decimating factor DDC. In section IV, performances of the designed CIC decimation filter, the FIR filter and the overall DDC system are demonstrated. Finally, Section V concludes this paper.

Figure 1 Structure of the GPU-based DDC

2 Overview of NVIDIA GPU and CUDA

As shown in Fig. 2, NVIDIA GPU contains a number of cores and different kinds of memories. SP (streaming processer) is a basic computing core on GPU and a certain number of SPs compose a SM (streaming multiprocessor) [9]. SM issues one instruction and all the SPs in this SM work together and process different data. There are several types of memories on GPU. The most abundant one with the largest capacity is known as the global memory. The global memory, outside the SMs and with high access latency, can be accessed by all SPs, which is mainly used for data exchange between SMs. Another type of memory, named as shared memory, can be accessed at high speed by SPs in a SM for data exchange inside the SM and cannot be accessed by SPs in other SMs. In addition, there are a certain number of registers on each SM, which are the fastest and most scarce memory on SMs. As the shared memory and registers are limited resources on GPU, whether they are efficiently used has a great impact on the GPU's performance. NVIDIA launches CUDA, which is an extension of the C language, to help developing GPU. The CUDA structure consists of grid, block and thread. The program on a SP is called a thread and all the threads compose a grid. A certain number of threads are organized to run on a single SM simultaneously and these threads compose a block.

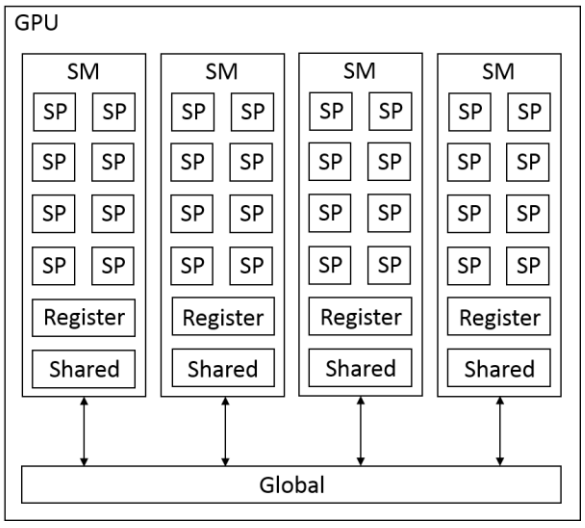

Figure 2 Architecture of Graphics Processing Unit

Generally, the maximum number of threads running on a single SM at the same time is a certain number, according to different kinds of GPU. As for NVIDIA 580 GTX, which owns 16 SMs in total, the maximum number is 1536. Thus, maximum 24,567 threads can run on NVIDIA 580 GTX at the same time and provide huge computing capability. Besides, the number of the blocks will affect the actual number of running threads on a SM. GPU also has restriction on the maximum number of blocks on a single SM, which is 8 for NVIDIA 580 GTX. So, if the number of threads in a block is less than 192, the total actual number will be less than 1536. The actual number of running threads is also restricted by the memory used in a SM. Maximum amount of shared memory per SM for NVIDIA 580 GTX is 48 KB and if each thread uses more than 32B shared memory, the actual number of running threads on a SM must be less than the maximum number for lack of shared memory. This is a simple trade-off. If too many threads are formed into a block and each thread consumes too much memory, then a SM cannot provide enough resources for 8 blocks running on it simultaneously. If a small number of threads are used in a block and the total number of threads on a SM is far less than 1536, then it will result in a large waste of GPU's computing power. Thus, we should take both the algorithms and the resource into account while developing GPU.

3 Implementation of a GPU-based DDC

As mentioned before, the GPU-based DDC consists of a DDS , a CIC decimation filter and a FIR filter. The DDS simply performs multiplication of the intermediate frequency with the input signal while the other two filters account for most of the DDC's work. Therefore, we will demonstrate how to design the CIC decimation filter and the FIR filter on GPU in the following paragraphs. The DDC follows 4 steps when working on GPU. Firstly, GPU produces the coefficients of the CIC decimation filter according to the decimating factor. Then, the DDS perform multiply operations to shift frequency. Thirdly, the CIC decimation filter decimate the signal. At last, the FIR filter filters the signal and reject the image.

3.1 A GPU-based CIC decimation filter

A CIC decimation filter is an economical alternative to a conventional decimation filter because of its effective structure with no multipliers and few storage elements. Thus, the whole operation can be regarded as a weighted summation of the signal. For detail information about CIC decimation filter, please refer to [10].

As we know, down sampling causes aliasing. Suppose a signal is sampled by f_s and then down sampled by X times. If the spectrum of the signal is not within the $[-f_s/2X, f_s/2X]$ range, aliasing will occur after down sampling. As a result, an anti-aliasing filter is needed before performing the down-sampling operation. Fig. 3 shows the magnitude response of a CIC decimation filter whose decimating factor is 8. After the CIC decimation filter, the spectrum of the useful signal is within the $[-f_s/16, f_s/16]$ range with little aliasing existing. Then a simple low pass FIR filter is sufficiently valid to reject image. In summary, if a signal is down sampled by D

times, the decimating factor of the CIC decimation filter must be D and the spectrum of the output signal is within the $[-f_s/2D, f_s/2D]$ range. Usually, two or three CIC decimation filters are cascaded to provide enough attenuation. The attenuation of a 3-order CIC decimation filter can reach 60dB, which satisfies the need of most communication systems. Therefore, we adopt a 3-order CIC decimation filter in the GPU-based DDC. Besides, the length of a 3-order CIC decimation filter N is related to D, which can be calculated as follows.

$$N = 3D-2 \quad (1)$$

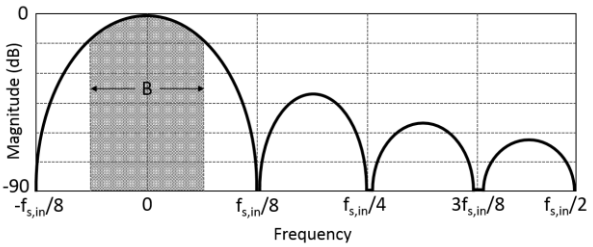

Figure 3 Magnitude response of a D = 8, CIC decimation Filter

In a conventional filter, every single output sample corresponds to an input sample. But due to decimation, every D inputs only leads to one single output in a CIC decimation filter, discarding the other D-1 output samples. Therefore, there is no need to perform summation for every single input sample. As shown in Fig. 4, a summation of N samples is performed for every D input samples and output one sample. The summation is weighted by N coefficients of the CIC decimation filter. Summations are performed at an interval equal to D. For instance, the first summation starts at the first sample and ends at N while the second summation starts at D+1 and ends at N+D. In this way, the summation and the decimation are performed in one function.

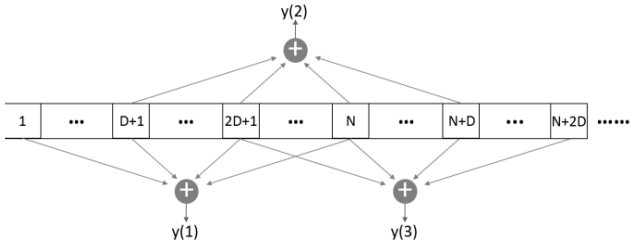

Figure 4 Weighted Summation in the GPU-based CIC Decimation Filter

Here, parallel reduction, a very important class of parallel algorithms, is introduced to solve the problems of summation. Imagine a tree and try to reduce it to the root in parallel. Then, the final sum is output. The very last summation cannot be done in parallel, as parallel threads must not write into one variable. A simple example is illustrated in Fig. 5. Parallel reduction builds a summation tree for a sequence of 8 samples and performs stepwise partial sums. Each step of this partial summation cuts the number of partial sums in half and the final result will be achieved after 3steps.

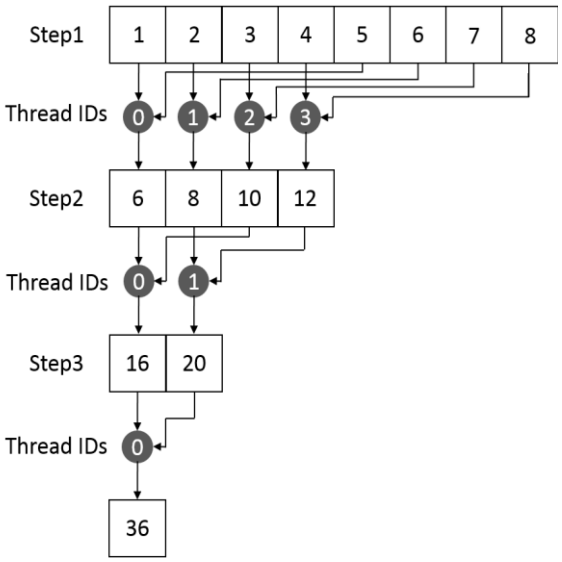

Figure 5 Parallel Reduction on GPU

Maximum number of threads on a single SM is 1536 for NVIDIA 580 GTX. If all the 1536 threads are assigned in a block the parallel reduction can perform a summation of 3072 samples and the largest decimating factor can be 1024 according to equation (1). In this case, all the 1536 threads are formed into one block and there is only one block on a single SM, which takes charge of a summation of 3072 samples. However, if the decimating factor is small, e.g. 8, then N is 22 and only 11 threads are needed to perform the parallel reduction. As mentioned before, only 88 threads of 8 blocks are active on a SM although NVIDIA 580 GTX can support 1536 threads on a SM. An effective way to solve this problem is to perform more reductions in one block when the decimating factor is small. Considering 8 blocks are working on a SM simultaneously, then at most 192 threads are assigned in each block and can complete the summation of 384 samples. Then the situation can be divided into two parts. When D > 128, we need more than 192 threads to perform parallel reduction if each thread sums two samples. In this case, we still use 192 threads but some of the threads work extra times to sum the other samples. For instance, if D = 256, 766 samples are to be summed. Firstly, the total 192 threads sum the first 384 samples and get 192 new samples. Secondly, 190 threads add another 380 samples and get 190 new samples, regardless of the other two original samples. After these partial sums, we get back to the situation that 192 threads sum total 384 samples and original parallel reduction can be performed. When D < 128, L parallel reductions are to be performed in one block. L is related to D according to equation (2). If D equals 8 and 11 threads are needed to perform the parallel reduction, in this case, we can perform 16 parallel reductions and total 176 threads are assigned in a block. This is a huge improvement compared with the case that set only 11 threads in a block. However, when the D is bigger than 64 but smaller than 128, we still perform a single parallel reduction in a block. In the worst case, if D = 65, then only 97 active threads are formed into a block. That means we can take advantage

of at least 50% of total threads, no matter what value D is set.

$$L = Floor((192\times 2+2)/3D) \quad (2)$$

Where $Floor(\cdot)$ rounds the input to the nearest integer less than or equal to it.

3.2 A GPU-based FIR filter

FIR is implemented by computing convolution between window function and input signal in time domain. The function is defined as follows:

$$y(n) = x(n) \otimes h(n) = \sum_{m=0}^{M-1} h(m) \times x(n-m) \quad (3)$$

Where, $x(n)$ is a length-N sequence of numbers considered to be the input signal, $h(n)$ is a length-M sequence of numbers considered to be the filter coefficients and $y(n)$ is the filter output.

In this paper, we evaluate two ways to construct a root raised cosine filter with the roll-off factor being 0.5 and the length being 49 on GPU. Firstly, we calculate equation (3) on CPU directly. We write a simple loop with a single accumulator variable to construct the sum. Then we perform equation (3) on NVIDIA 580 GTX. Like CIC decimation filter, parallel reduction is also used in the FIR filter. As the length of the FIR filter is fixed, we can figure a good solution easily. Here, we assign 245 threads in each block and total 6 blocks can be set in a SM at the same time. Thus 95.7% of the threads are used in a SM. 245 threads are divided into 5 parts, each part take charge of one output. Another way to achieve convolution is using the overlap-discard method [11]. The strategy is to divide the signal into segments and performs FFT (fast fourier transform) operation in each segment. It is noteworthy that CUDA API (application programming interface) provides the library of FFT algorithm. According to NVIDIA's research, best performance can be achieved when GPU performs length-4096 FFT. Thus, we evaluate the efficiency of the convolution using length-4096 FFT.

4 Experimental results

In this section, we present the performance of the DDC. The GPU-based one is developed to run on NVIDIA GeForce 580 GTX, and is compared with its CPU-based counterpart which is developed in C++ to run under an Intel(R) Core(TM) i7 3.07GHz. To be specific, experiments are performed on the CIC decimation filter and the FIR filter.

Firstly, we evaluate the performance of the proposed CIC filter. As the GPU-based CIC filter has different functions while the decimating factor varies, we select two typical values and perform test. Table I is a comparison of the execution time when decimating factor is 8 and 256 in the GPU-based CIC decimation filter. Apparently, the execution time is larger when the decimating factor is 256, especially when there is more signal to be processed. As mentioned before, parallel reductions are performed in each block and half of the threads will be idle after each step. According to equation (2), there are 16 parallel reductions in each block when the decimating factor is 8, and at least 16 threads are working at the last step. However, there is only one parallel reduction in each block when the decimating factor is 256 and only one thread is working at the last step. That means the utilization of threads is different when the decimating factor varies, which leads to different performances. Besides, when dealing with a small amount of data, the difference of throughput is only 3 times However, when data size increases, consuming time for D = 256 is 17.5 times more than that of D = 8.

Table I Comparison of the GPU-based CIC decimation filter

Data size (KB)	Time (ms)	
	D = 8	D = 256
51200	0.02303	0.40430
25600	0.01254	0.22147
12800	0.00674	0.10353
6400	0.00386	0.05596
3200	0.00255	0.03037
1600	0.00191	0.01698
800	0.00141	0.00930
400	0.00120	0.00630
200	0.00120	0.00520
100	0.00119	0.00390
50	0.00119	0.00350

Fig. 6 and Fig. 7 demonstrate the throughput of the CPU-based and the GPU-based CIC decimation filter when the decimating factor is 8 and 256. The experimental results show that, with the increasing amount of signal, CPU's execution time increases linearly. However, GPU's execution time increases much more slowly when processing a small amount of signal and increases linearly when there is a large amount of signal. The reason is that a smaller number of blocks, especially smaller than the number of SMs, will lead to a portion of SM remaining idle. As a result, the program cannot fully exploit the parallelism of GPU when there are only a small number of signal to be processed. As for CPU, there is little difference for the DDC to deal with a certain amount of data when the decimating factor varies. The results also reveal that, there is a large difference in execution time between the CPU-based CIC decimation filter and the GPU-based one. When dealing with large amount of signal, CPU needs more processing time than GPU and sometimes the time can be 300 times longer. That means GPU can improve the performance greatly.

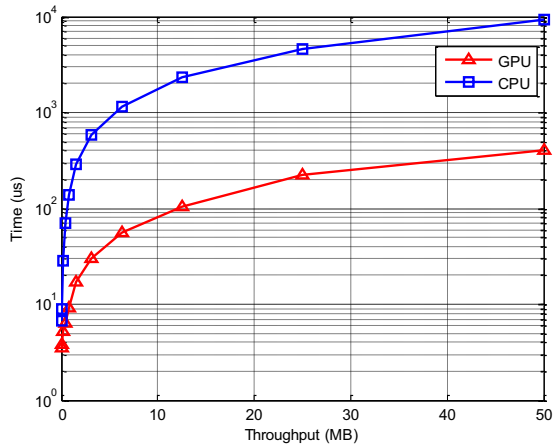

Figure 6 The execution time comparison in a D=256 CIC filter

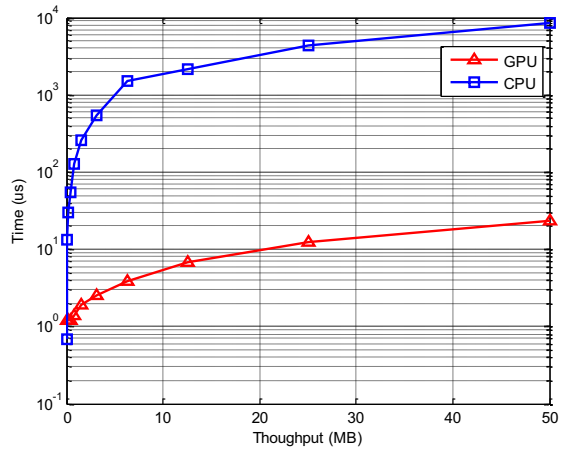

Figure 7 The execution time comparison in a D=8 CIC filter

As observed from Fig. 8, the performances of FIR filter are demonstrated by measuring the amount of data processed in one second. The left column and the middle bar show the performances of CPU-based and GPU-based implementation using equation (3). The right one shows the performance of using overlap-discard method on GPU. Firstly, The results are 5.89 MB/s, 515.84 MB/s and 2.23 GB/s separately. The results firstly show that the CPU-based FIR filter performs worse than the GPU-based one if using the same algorithm according to equation (3). The speedup reaches nearly up to 87 times on GPU. The CPU-based FIR filter does not exploit parallelism and uses simple loop to do summation. Secondly, the GPU-based FIR filter achieves the best performance using overlap-discard method. As we know, using overlap-discard method to achieve fast convolution is more efficient than common convolution on CPU. The results demonstrate that this rule works for GPU as well. It can reach a speedup of 387 times compared with the common convolution CPU.

As a whole, the GPU-based DDC also shows better performance than the CPU-based one. Due to the simple structure and algorithm, the throughput of the CIC decimation filter is much larger than that of FIR filter on GPU. Thus, the throughput of the whole DDC is restricted by the FIR filter. Obviously, it's very useful to optimize algorithm of the FIR filter on GPU. According to the experimental result, the throughput of the GPU-based DDC reaches 2.23 GB/s while the CPU-based one could only reach 5.89 MB/s and the speedup of the whole DDC is 387.

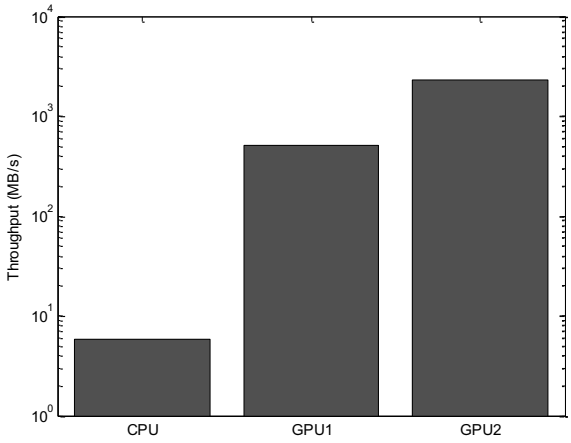

Figure 8 Performances comparison in FIR module

5 Conclusions

DDC is an important part of a communication system, especially in many military applications. To our best knowledge, there is no research about how to deal with different decimating factors in GPU-based DDC. In this paper, a DDC, including a DDS, a CIC decimation filter and a FIR filter are implemented efficiently with CUDA on NVIDIA 580 GTX. Benefiting from the serious computational power of GPU, the GPU-based DDC gains a huge performance improvement with data processing 2.23 GB/s. Besides, the execution time is larger for the CIC decimation filter to process a certain amount of data on GPU when the decimating factor is bigger than 128. What's more, we present two efficient ways to construct a root raised cosine filter with 4 times oversampling on GPU. It turns out that using overlap-discard method to construct a FIR filter is more efficient on GPU than common convolution both on GPU and CPU.

There are many advantages of GPU, such as energy efficiency, cheapness, powerful computing ability and so on. However, there are also some disadvantages of GPU. Firstly, not all algorithms can have theoretical speedup. GPU is not optimized for processing data, which is not large enough or cannot be blocked. Besides, GPU is harder to program than CPU. When solving a problem, it needs to be really parallelizable. Then we must micromanage ram and think about how to access ram in the right way to speed it up due to caching and lots of very low-level stuff. Sometimes, the speed can be increased by 8-10 times just by optimizing memory accesses. Regardless of these disadvantages, GPU still shows great prospects in general computing. Many new GPUs with more powerful computing ability and lower cost will be delivered in the future and provide more solutions for common problems.

Acknowledgements

This work was supported by the Hi-Tech Research and Development Program of China (2011AA01A106).

References

[1] Guohui Wang, Michael Wu, Yang Sun, Joseph R. Cavallaro. GPU accelerated scalable parallel decoding of LDPC codes. 2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers, 2011: 2053 – 2057.

[2] Gao Xiaopeng, Dong Deping. 2011 International Conference on System Science, Engineering Design and Manufacturing Informatization, 2011:293– 296.

[3] Jeff Chase, Brent Nelson, John Bodily, Zhaoyi Wei, Dah-Jye Lee. Real-time optical flow calculations on fpga and gpu architectures: A comparison study. 16th International Symposium on Field-Programmable Custom Computing Machines, 2008: 173-182.

[4] Michael Wu, Yang Sun, Joseph R. Cavallaro. Reconfigurable real-time MIMO detector on GPU. 2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers, 2009: 690-694

[5] T. Hollis, R. Weir. The Theory of Digital Down Conversion. Hunt Engineering, 2003.

[6] Eugene B. Hogenauer. An economic class of digital filtere for decimation and interpolation. IEEE Transaction on Acoustrics, Speech and Signal Processing, 1981, 29(2): 155-162.

[7] Amit Upadhyay, Yawatkar Shakun Rajan. Implementation of digital down converter in GPU. Department of Avionics, Indian Institute of Space Science and Technology, 2012.

[8] NVDIA Corporation. CUDA C Programming Guide. 2008.

[9] David Luebke. CUDA: Scalable parallel programming for high-performance scientific computing. 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2008:836-838.

[10] Ronald E. Crochiere, Lawrence R. Rabiner. Further considerations in the design of decimators and interpolators. IEEE Transaction on Acoustrics, Speech and Signal Processing, 1976, 24(4):296-311.

[11] Rabiner, R. Lawrence, B. Gold. Theory and Application of Digital Signal Processing. Englewood Cliffs, NJ, Prentice-Hall, Inc., 1975.