2010 12th International Conference on Computer Modelling and Simulation

Performance Prediction of Parallel Computation of Streaming Applications on FPGA Platform

Radha Guha
CSE Dept.
Amrita Univ., Coimbatore, India 641105
radhaguha@yahoo.com

David Al-Dabass
School of Computing and Informatics
Nottingham Trent Univ., NG11 8NS
David.al-dabass@ntu.ac.uk

Abstract— **This paper analyzes possible performance improvement of streaming applications by the parallel computation platform of FPGAs. Software developers still are not familiar with the hardware implementation details of applications and will benefit from this analysis. First the available logic and memory resources of modern FPGAs like Xilinx's Virtex-5 and Virtex-6 devices are explored to determine how many parallel processors can be configured with the available logic resources and how many parallel processors' speed can be sustained by simultaneous data access from the on-chip memory. A portion of the on-chip memory is first set aside for pre-fetching of reconfiguration bits for dynamic reconfiguration of FPGA. The rest of the on-chip memory is used for data memory requirements of the concurrent processors. Based on input data rate and required output throughput of data, a number of on-chip cache memory locations and their sizes are determined. Finally a quantitative analysis of possible performance gain of streaming applications by FPGA implementation is compared to that of sequential implementations by a 2.5 GHz dual-core Intel microprocessor.**

Keywords-Streaming Applications; FPGA Platform; Performance Prediction

I. INTRODUCTION

There are many data intensive streaming applications which require compute intensive algorithms like finite impulse response (FIR) filter, discrete cosine transform (DCT), fast Fourier transform (FFT) and data encryption standard (DES) etc. Digital filters are widely used in digital signal processing (DSP). DCT algorithm is one of the kernels of video compression standards like H.264 and MPEG-4. DES is one of the encryption standards recommended by Federal Information Processing Standards (FIPS) of the National Institute of Standards and Technology (NIST) for transmitting and storing of all sensitive data. The characteristics of the streaming applications are high-performance computations and simulations requiring low latency, high throughput and high I/O bandwidth. These applications can be heavily pipelined or highly parallelized to take advantage of the abundant logic and memory resources of FPGA for performance improvement. FPGA platform is also ideal for prototyping of applications as it can be dynamically reconfigured for different modules to execute on it at different times.

To increase the concurrent processing, FPGA device platform is evolving with more logic and memory resources from one generation to the next. This has been possible because of process technology's continual shrinking trend following Moore's Law which is enabling FPGA to pack billions of smaller and faster transistors on the same chip. The logic resources can be configured for parallel processing elements (PEs) with flexible data path width of 1, 2, 4, 8, 16 bits etc. in contrast to the 32 or 64 bits fixed data path width of a general purpose microprocessor. The on-chip distributed RAM and less distributed block RAM of FPGA can be used as registers, FIFO memory, random access memory, lookup tables (LUTs), or high performance state machines. This on-chip data cache helps sustaining the parallel processing speed of the concurrent processors on FPGA. FPGA also provides interface for larger external main memory access. But memory access from external main memory is slower than that of on-chip memory. Bringing data from external memory to the hardware of FPGA through bandwidth limited bus limits the acceleration power of FPGA itself. Because of this processor-memory speed gap, on-chip cache memory provisions should be sufficient depending on input data rate and required output throughput of applications.

This paper analyzes such on-chip memory requirements for maximizing performance gain of streaming applications by the FPGA platform. The software application developers, who usually lack the expertise to explore the HW resources of the FPGA platform, will benefit from this analysis.

In Section II, Virtex-5 and Virtex-6 device platforms are explored (Table 1) [1], [2] for available logic and memory resources. How much logic and memory resources are required for the arithmetic functions is measured and their operating speed is verified using Xilinx's ModelSim in Table 2. Existing processor-memory speed gap for limited external memory interface capability of FPGA is also evaluated in this section. In this section, related research work to overcome the processor-memory speed gap problem and performance optimization of FPGA platform is also surveyed. In Section III the required on-chip memory is estimated which depends on input/output data rate of applications. In this section, first an example of a FIR filter

978-0-7695-4016-0/10 $26.00 © 2010 IEEE
DOI 10.1109/UKSIM.2010.111

is given which can be heavily pipelined. Second an example of a DCT kernel is given which can be highly parallelized on 8*8 data block. With an efficient logic and memory architecture design which reduces processor-memory speed gap, the amount of logic and memory resources of modern FPGAs determine maximum performance gain of FPGA platform. In Section IV, software simulation is used to estimate data transfer rate from external memory to on-chip memory. In Section V, performance gain of FPGA platform is compared with that of a sequential general purpose microprocessor. Section VI concludes the paper.

II. LITERATURE SURVEY

A. FPGA Overview, Logic and Memory Resources

Field Programmable Gate Array (FPGA) is a semi-conducting device with programmable configurable logic blocks (CLBs), input output blocks (IOBs), block random access memory (BRAM) as embedded memory blocks (EMBs), interconnects, clock resources and configuration circuitry (Figure 1). CLBs are lookup table (LUT) based where a k-input LUT can store any logic function of k-binary variables. The inter connections between the CLBs are a two dimensional network of connectors and routers called connection network. There is also provision for temporary on-chip data storage, as all computations require data. An array of CLBs on FPGA is surrounded by a number of IOBs for data transfer to and from external main memory. LUT-based FPGA stores logic functions as well as the configuration bits that drive the switching matrix of the FPGA.

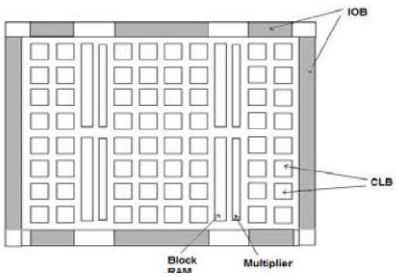

Figure 1: FPGA Architecture

Table 1: Virtex-5 & Virtex-6 Resources

Virtex-5	XC5VSX95T	XC5VLX330T	Virtex-6	XC6VSX475T	XC6VLX760
Speed (MHz)	550	550	Speed (MHz)	600	600
I/O Banks	19	27	I/O Banks	21	30
Logic Slices	14,720	51,840	Logic Slices	74,400	118,560
DSP48 SLICES	640	192	DSP48 SLICES	2016	864
BRAM (18Kbits)	488	648	BRAM (18Kbits)	2128	1440

As data intensive streaming applications with high performance and high I/O bandwidth requirements are considered in this paper, FPGAs which have high memory to logic ratio are considered here. In Table 1 the available logic and memory resources of some of the Virtex-5 and Virtex-6 devices [1], [2] suitable for streaming applications are listed. Table 2 lists basic functional units like adder and multiplier and corresponding logic resource usage, critical path delay and frequency of operation when synthesized on FPGA.

Table 2: Resource Usage and Critical Path

Arith. Fn.	Data Width	# Used Input LUTs	# Used Logic Slices	# Used DSP 48 Slices	# Used 4 Input IOBs	Maxm. Delay ns	Freq. Of Ope MHZ
Adder	8	8	4	0	25	6	170
	16	16	8	0	49	7	162
	32	32	16	0	97	7	149
	64	64	32	0	193	8	128
Multipl	8	0	0	1	32	9	120
	16	0	0	1	64	9	120
	32	0	0	4	128	14	72
	64	158	79	16	256	19	54
	8	64	32	0	32	20	50
	16	256	128	0	64	21	45
	32	1024	512	0	128	23	40
	64	4096	2048	0	256	25	35

Modern FPGAs also provide fast and dedicated ASIC (application specific integrated circuit) quality arithmetic operation units such as multiply and accumulate (MAC) units as DSP blocks for computation intensive algorithms. When the number of DSP slices is more, number of logic slices is less. As the functional units in DSP slices are faster they are used first before configuring additional functional units from logic slices. MAC units in DSP slices are called adder-chain based MAC units which have dedicated routing. When the DSP slices are used up the logic resources can be configured for distributed arithmetic MACs (DAMACs) by LUT based distributed arithmetic approach.

The CLBs, IOBs, DSPs and EMBs are all run-time reprogrammable or reconfigurable in the order of milliseconds to create an efficient interface between the logic data paths and the on-chip memory. Modern FPGAs like Virtex-II Pro, Virtex-4, Virtex-5 and Virtex-6 are partially reconfigurable at run-time [3], [4]. A portion of the FPGA can be reconfigured when execution continues on other parts of the FPGA. Partial reconfiguration reduces configuration bits memory size and configuration time overhead. By pre-fetching and caching the configuration bits in on-chip BRAM, configuration overhead can be hidden at the cost of area. Reconfiguration memory, reconfiguration time and power consumption overhead of FPGA can also be minimized by efficient configuration bit-stream compression. During configuration these compressed

bit stream can be decompressed by a decompressor circuit and used for configuring a data path. It is a design decision to determine how much percentage of BRAM can be set aside for configuration bits depending on frequency of reconfiguration requirements and data memory requirements.

In all modern FPGAs like Actel MX and DX, Altera FLEX10K, FLEX10KE, APEX20K, Xilinx's Virtex-5 and Virtex-6 device family, there are on-chip distributed small memories and less distributed larger block memories. In a fine grained data parallel programming model the distributed small memories can be used as registers, register files and LUT or level 1 (L1) cache. Whereas in coarse grained data parallel programming model larger block memories called embedded memory blocks (EMBs) can be used as level 2 (L2) cache. One EMB may be of a few CLBs in height and width. For optimal use of the chip area the EMBs must be fully utilized by the applications. Data can be brought from off-chip SDRAM to on-chip L2 cache and then distributed to L1 cache. Any data path that is configured on FPGA logic fabric is a number of basic functional units like adder, multiplier etc. connected in a particular fashion. Data is fed to the data path from the memory banks and after processing is stored back to the memory banks [5] for further use by another data path or for transport to off-chip main memory.

In Virtex-5 XC5VLX330T device the block memory size is 11.6 Mb whereas in Virtex-6 XC6VSX475T total block memory (BRAM) size is improved to 38.3 Mb (Table 1). Each block can store up to 36 Kbits of data which can be configured as 2 independent 18 Kbits block RAMs or one 36 Kbits block RAM. BRAM can be configured as dual-ported RAM or FIFO memory. Port width can be variable. In Virtex-5 the BRAM transfers data at 550 MHz and in Virtex-6 the BRAM transfers data at 600 MHz. The BRAMs are cascadable to configure larger memory block with deeper and wider memory interfaces. As for example 16 BRAMs can be cascaded to create 576 Kb RAM. The maximum number of parallel PEs that can be sustained will depend on number of simultaneous read access from BRAMs. As for example Virtex-5, XC5VLX330T has total 324 dual ported BRAMs each of 36 Kb in size. So that chip can support maximum 324*2 = 648 read operations. If each PE has 4 registers in its data path which need data in the same clock cycle, then there can be maximum 648/4 = 162 PEs working in parallel. If each register is 32 bits wide then memory bandwidth requirement is 162*4*32 = 20736 bits.

B. External Memory Interface Capability of FPGA

External memory interface capability of FPGA is improving slowly [6]. In 1990 data transfer rate was at single data rate (SDR) from synchronous dynamic random access memory (SDRAM). Now in 2009 it has evolved to double data rate (DDR) SDRAM or DDR2 SDRAM operating at 667 Mb/s or higher per pin. The data rate is expected to double every four years reaching 1.2 Gb/s by 2010 with DDR3 SDRAM. When the data rate per pin is fixed by the technology limitation, the data bandwidth can be variable by configuring narrow bus or wide bus.

FPGA can be configured as low-performance low-cost device or high-performance high-cost device. In a high performance system a faster external memory interface is provided. In low performance system the memory interface is designed to support 133 Mb/s data transfer per pin from DDR SDRAMs whereas in high performance design the data transfer rate can be 667 Mb/s per pin from DDR2 SDRAMs [6]. Spartan-3, Virtex-4, Virtex-5 and Virtex-6 FPGAs offer low cost data bus width of 72 bits and high cost data bus width of up to 576 bits. Thus in a high performance high cost system 667*576 = 384 Gb/s data transfer is possible. A high performance and high cost design can even provide data transfer through multiple interfaces from multiple off-chip SDRAM. FPGA memory interface with external SDRAM memory is controlled by a software (SW) processor. A SW processor can initiate direct memory access (DMA) transfer to the on-chip L2 cache. Inside the FPGA there can be another memory controller between the L2 cache and the data path of the functional unit.

C. Processor Memory Speed Gap

The peak clock frequency of Virtex -5 device is 550 MHz and Virtex-6 device is 600 MHz. As FPGA is very generic, complex functions designed from LUT-based logic fabric can approximately reach processing speed of 100 to 300 MHz (Table 2). As multiple processing elements (PEs) can be configured to run parallely on FPGA their cumulative data consumption rate is very high. Suppose we can configure 100 PEs on FPGA and each PE can perform one addition operation (OP) in one cycle then the cumulative throughput of the system will be 100 OPS*300 MHz = 30000 MOPS (mega operations) = 30 GOPS (giga operations). If every operation involves two 32 bits operands then total 30*2*32 =1920 Gb/s data supply is required. Thus the required data supply rate is much higher than even what the high performance high cost external memory interface can provide as analyzed in Section II.B above.

Memory latency from SDRAM main memory to FPGA can be reduced by pre- fetching and caching onto on-chip cache memory. The main memory is off -chip SDRAM memory and large blocks of shared memory or L2 cache is regularly distributed throughout the chip. Smaller and dedicated L1 cache is near each processing element. If more concurrent data paths are configured more number of L2 caches are better design choice instead of a single centralized L2 cache. For video processing applications large amount of array data is transferred into the chip. This

kind of data access pattern is regular and periodic. Thus memory controller can be designed to transfer data in pipeline mode from on-chip SRAM or page mode from off-chip SDRAM memories.

D. Performance Optimization

Performance of an application improves by parallel processing on FPGA chip but the area and the power cost of the chip also increases significantly. One consideration against abundance of on-chip memory provisions is its associated cost. For deeper memory the cost of memory address generation circuitry is significant and memory is more expensive than complex logic data paths. For data intensive streaming application such as video processing involving multidimensional array data, on-chip memory area requirement is high. A considerable percent of the power consumption of the chip is due to data storage, data transfer and cache controllers. Many memory optimization techniques can be applied to reduce memory requirement of streaming applications. Cache memory size is designed to be much smaller than external SDRAM main memory size and only the data that will be needed immediately by the processors are pre-fetched to cache memory. Correct estimation of cache memory size, memory bandwidth and memory locations is the key to maximizing performance gain of streaming applications and power, area savings of the chip.

Research is going on for reduced and efficient usage of memory. In reference [7] they have surveyed the state-of-the-art techniques used in performing data and memory related optimizations for embedded systems. Memory optimization can be achieved by architecture independent source level code transformation to reduce storage and memory bandwidth. Code transformation is also required to exploit the spatial and temporal concurrency of an application for parallel processing which in turn results in changing the memory access pattern in the original code. As architecture dependent memory optimization techniques they have surveyed graph coloring, clique partitioning and in-place mapping etc. for different levels of granularity from register to register file to on-chip cache memory.

In reference [8] they have estimated background memory area requirement for video and image processing applications based on data-flow analysis. In reference [9], system level memory optimization and global reorganization of the sub-modules or tasks are done first to reduce area and power cost of data storage and transfer before partitioning the tasks between HW and SW for performance improvement. In reference [10], they have described memory interface design both for on-chip SRAM and off-chip SDRAM memory. Their proposed design has low area and fairly good timing with pipeline and page-mode data transfer operations.

In reference [11], they have proposed performance improvement of streaming applications on FPGA platform by optimizing HW synthesis methodology by hierarchical compilation strategy.

The contribution in this paper is different from the research work for memory optimization and performance optimization as mentioned above. The main purpose of this paper is to familiarize the software developers with the HW implementation details of the FPGA platform. The paper analyzes possible performance improvement of streaming applications on FPGA platform depending on its available logic and memory resources, its interface capability with external memory and input and output data rate of applications. For that purpose what portions of the available on-chip memory is to be kept for reconfiguration bits for dynamic reconfiguration of the chip and for data for processors are estimated first. L1 and L2 level data cache memory size and locations depending on input data rate and output throughput of data are analyzed in Table 3 and Table 5. Finally performance gain of the FPGA platform in comparison to a 2.5 GHz dual-core Intel microprocessor is reported in Table 8.

III. On-Chip Memory Requirement Analysis

A. Data Input/Output Rate Specific Memory Requirement

In a low performance design fewer resources are dedicated for completing an application. More sequential operations are performed than parallel operations to reduce area and power cost. In a high performance design more resources are dedicated for more parallel operations than sequential operations thus incurring more area and power cost. Data input/output rate specific memory architecture for N-tap FIR filter is designed and analyzed here.

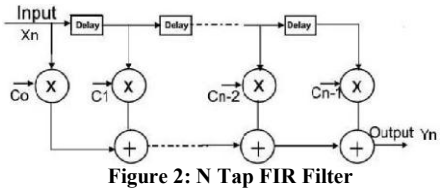

Figure 2: N Tap FIR Filter

N-tap FIR filter can be mathematically represented as $Y[n]=\sum C[i]X[\ n-i]$, where $C[i]$ is the filter coefficient where i varies from 0 to n-1, X is the input data stream, $Y[n]$ is the current output signal and n is the number of taps. N-tap FIR filter computes N multiply and (N-1) add operations as shown in the block diagram of FIR filter in Figure 2. Each output signal is the convolution sum of one current and N-1 previous input data multiplied by N coefficient values.

In a sequential implementation by a microprocessor, N delayed samples are multiplied by the coefficients one by one, by one multiplier and added together by one accumulator unit. In the parallel implementation on FPGA, all N multiplications and additions can be performed in parallel by N multiplier and (N-1) adder units.

Input data rate is often slower than the processing speed of the processors. Thus if the input data rate cannot be matched with the cumulative processing power of *N* processors, having multiple processors will not achieve the expected performance gain. Configuration of different logic and memory architecture for 400 taps FIR filter for 1MSPS (million samples per seconds), 2MSPS and 400 MSPS data rate are analyzed here.

For a large application like 400 taps filter after using the DSP blocks the logic slices can be configured for parallel multiplier and adder units. Number of bits required to represent the input data and the constant coefficients determine the data path width and memory size. Depending on the processing speed in the logic slice and the input data rate the logic and memory architecture design will vary. For a 400 taps FIR filter for calculating one output data, 400 multiplications and 400 additions are required. Suppose input data and coefficient data are both 16 bits wide and in each cycle one multiplication and one addition can take place. For a data rate of 1 MSPS from off-chip memory, 400 taps filter needs 400M multiplications and 400M additions per second. If the processing speed is 400 MHz then a pair of multiplier and adder can perform all 400 M multiplication and addition operations in one second, one by one. For one processing element (PE) there is a need for 400 word deep input data and 400 word deep coefficient data in a dedicated, level one (L1) on-chip cache memory as shown in Figure 3.

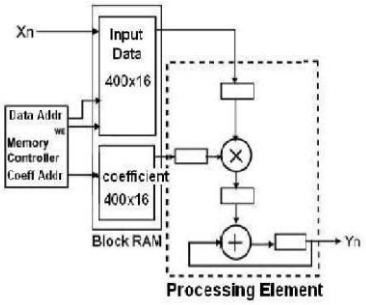

Figure 3: 1 MSPS, 400 Taps FIR Filter

If the data rate is 2 MSPS then we can have two PEs each calculating 200 taps. So there is a need for two 200 word deep coefficient memory and 200 word deep input data memory near each PE. If the data rate is 400 MSPS then we can have a pair of multiplier and adder for every tap of the filter to keep up with the data rate. In this case there is no need for dedicated on-chip coefficient memory and on -chip input data memory. Instead distributed registers near each PE are sufficient. Thus using more real estate for 400 parallel PEs, the design can keep up with the input data rate.

From the above analysis it can be concluded that higher processing power and lower input data rate require less logic resources but more memory resources and vice versa. If the data rate is slower than the processing power then multiple data channel processing such as red, green and blue channel for color images and videos can be exploited effectively by the same processing element instead of having separate processing element for each channel. If the data rate is higher than the processing speed then more concurrent processing elements can be configured to process data faster.

Huge concurrent resources of FPGA can be exploited in another way. Irrespective of input data rate, output data rate can be increased by configuring more parallel PEs. To sustain output throughput, data can be pre-fetched and cached in larger L2 cache made of cascading block RAMs. Obviously the transfer time of the data from external SDRAM to on-chip BRAM will affect the performance gain by the FPGA. In 400 taps FIR filter design all the 400 PEs are pipelined or chained with one another and there is one entry point of input data which traverse through 400 delay registers. For this design one central L2 cache is sufficient.

Table 3 shows a list of logic and memory configurations depending on input data rate and required output throughput of 400 taps FIR filter.

Table 3: I/O Rate vs. Logic and Memory Usage

400 Taps FIR Filter, 16 bits, 10 MSPS input data, 16 bits coefficient, 400 MHz Processor							
Output data rate (MSPS)	#HW PEs	#4 input LUTs	#Logic slices	#L1 bits	Size	#L2	Size (Kb)
10	10	272X10 = 2,720	2720	10	1280	1	10
40	40	272X40 = 10,880	5440	40	320	1	100
100	100	272X100= 27,200	13600	100	128	1	500
200	200	272X200= 54,400	27200	200	64	1	1000
400	400	272X400= 108,800	40800	300	32	1	2000

For many other applications like DCT, FFT, DES more concurrent data paths can be configured and multiple number of L2 cache distributed regularly on the chip is a better design choice. DCT algorithm (Table 4) is block based and work on 8*8 data blocks. Two multipliers and one adder can be employed in each data path of a PE which iteratively computes the DCT algorithm for a data block.

Table 4: 2D DCT Kernel

```
for ( i = 0 ; i < N ; i++ )
{ for ( j = 0 ; j < N ; j++ )
{DCTvalues[i][j]= 0;
for ( k = 0 ; k < N ; k++ )
{DCTvalues[i][j]+= inputData[i][k]*encode_cos_table[k][j];}
DCTvalues[i][j] *= c_table[i][j];}}
```

Table 5: 2-D DCT Logic and Memory Usage

Image Size	#HW PEs	#4 input LUTs	#Logic Slices	#L1 Cache	L1 Cache Size	#L2 Cache	L2 Cache Size
256x 256	16	2000x16 = 32000	8000	16	4096	4	1000
512x 512	32	2000x32 = 64000	16000	32	4096	8	1000
1024x 1024	64	2000x64 = 128000	32000	64	8192	8	1000

Table 5 shows how the concurrent resources of FPGA can be exploited for 2 -D DCT algorithm for various image sizes. For an image size of 1024*1024, 64 PEs are constructed each having two multipliers and one adder. As all PEs can operate concurrently and have no data dependency between them multiple numbers of BRAMs can be equally distributed for concurrent data access by the PEs. Instead of constructing 64 BRAMs, 8 BRAMs are constructed each supplying data for a group of 8 PEs. As each PE has two multipliers and one adder the processing speed of this data path is conservatively estimated as 50 MHz. 1 Mb of data storage in each BRAM will sustain 1000000/(32*2) = 15,625 read cycles for double ported 32 bit data transfer. This amounts to at least 52 ms of uninterrupted operation at a memory speed of 300 MHz.

IV. METHODOLOGY

A. SW Simulation

To verify the preliminary memory analysis regarding data production and data consumption rate by the on- chip BRAM and the data paths, a design consisting of one on-chip L2 cache supplying data to 4 data-paths through 4 FIFO channel interfaces are simulated using SystemC programming language. Xilinx's CoreGenerator EDA tool, VHDL and SystemC like SW/HW modeling language to verify design at a higher level of abstraction are used. VHDL and SystemC support all the constructs of hardware design like events timing and events synchronization to verify an actual system. Creating block RAM with Xilinx's CoreGenerator tool is easy as just by specifying memory size and port width the corresponding VHDL code is generated automatically which determines the data address width (Table 6). Table 7 shows the code snippet of the simulation model of a design with one on-chip L2 cache supplying data to 4 data paths through 4 FIFO channel.

Table 6: VHDL Code Generated by CoreGenerator

```
port (addra: IN std_logic_VECTOR(10 downto 0);
addrb: IN std_logic_VECTOR(8 downto 0);
clka: IN std_logic;
clkb: IN std_logic;
dina: IN std_logic_VECTOR(63 downto 0);
dinb: IN std_logic_VECTOR(255 downto 0);
douta: OUT std_logic_VECTOR(63 downto 0);
doutb: OUT std_logic_VECTOR(255 downto 0);
wea: IN std_logic;
web: IN std_logic);
```

Table 7: Producer and Consumer Model

```
#include <systemc.h>
//Write interface
class write_if : virtual public sc_interface
{public: virtual void write(int) = 0;
virtual void reset() = 0;};
//Read interface
class read_if : virtual public sc_interface
{public: virtual void read(int &) = 0;
virtual int num_available() = 0; };
//FIFO Channel, implements the read and write interface
class fifo : public sc_channel, public write_if, public read_if
{ public: fifo(sc_module_name name) : sc_channel(name),
num_elements(0),first(0) {}
void write(int c) {
if (num_elements == max)
{ wait(read_event); }
data[(first + num_elements) % max] = c;
++ num_elements;
write_event.notify(); }
void read(int &c){
if (num_elements == 0)
{ wait(write_event); }
c = data[first];
-- num_elements;
first = (first + 1) % max;
read_event.notify();}};
SC_MODULE(top)
{ fifo *FIFO1,*FIFO2,*FIFO3,*FIFO4; //4 FIFO ch
bram1 *BRAM; //one producer
datapath *DP1,*DP2,*DP3,*DP4; //4 consumer
SC_CTOR(top)
{ //Instantiate 1 producer, 4 consumer and 4 FIFO channels and
portmap them properly
FIFO1 = new fifo("Fifo1");
FIFO2 = new fifo("Fifo2");
FIFO3 = new fifo("Fifo3");
FIFO4 = new fifo("Fifo4");
PE1 = new pe1("Producer1");
BRAM->out1(*FIFO1);
BRAM ->out2(*FIFO2);
BRAM ->in1(*FIFO3);
BRAM ->in2(*FIFO4);
```

V. RESULTS

A. Performance Gain

Finally in Table 8, performance improvement of two streaming algorithms FIR filter and DCT algorithm by FPGA implementation are compared with software implementation by a sequential micro -processor. A micro-processor's peak performance is the number of 64 bits floating point operation it can perform in one second. If the micro-processor can perform one multiply and one add

operation per cycle then the 2.5 GHz dual-core Intel processor will perform (2.5*2*2) = 10 GFLOPS/seconds. In FPGA data bus width can be customized to any number of bits as required. 64 bits floating point operation consumes too much resources, so in FPGA all operations are either integer or fixed point operations on customized data path width and FPGA performance is measured in terms of GOPS. By configuring multiple processors on FPGA and providing enough on-chip data memory the output data throughput can be increased as desired only limited by the available hardware resources.

The available logic resources in Virtex-5 and Virtex-6 devices can allow up to 40 times performance improvement over sequential microprocessor.

Table 8: Performance Gain

Algorithm	Virtex-5 Output Datarate(GOPs)	Dual core Intel Microprocessor	Performance Boost by FPGA platform
FIR	10	10 GFLOPS	1x
	100		10x
	400		40x
DCT	10		1x
	100		10x
	400		40x

VI. CONCLUSION

In this paper possible performance improvement of streaming applications by parallel computation on FPGA platform is analyzed and compared with sequential execution of application by a general purpose microprocessor. Required memory architecture for supporting desired performance boost of streaming application is designed, analyzed and simulated. The memory architecture design is dependent not only on the logic to memory ratio of the chip but also on considerations like reconfiguration memory requirements, desired power and area savings of the chip and input/output data rate. In future we will implement our complete design on FPGA to verify the results with the software simulation.

REFERENCES

[1] Xilinx.Virtex-6 Family Overview. DS250 (v2.1) Nov. 2009.

[2] Xilinx. Virtex-5 Family Overview. DS100 (v5) Feb. 2009.

[3] R. Hymel, A. George and H. Lam. Evaluating Partial Reconfiguration for Embedded FPGA Applications. NSF Center for High-Performance Reconfigurable Computing (CHREC),University of Florida.

[4] P. Butel, G. Habay, A. Rachet. Managing Partial Dynamic

[5] M. Gokhales and P.Graham. Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays (Springer, 2005).

[6] Adrian Cosoroaba. Memory Interfaces Made Easy with Xilinx FPGAs and the Memory Interface Generator. Xilinx WP260 (V1.0) February 16, 2007.

[7] P. Panda, F. Catthoor, N. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle and P. Kjeldsberg. Data and Memory Optimization Techniques for Embedded Systems. ACM Transactions on Design Automation of Electronic Systems, 2001.

[8] F. Balasa, F. Catthoor and H. Man. Background Memory Area Estimation for Multidimensional Signal Processing Systems. IEEE Trans. on VLSI Systems, Vol.3, No.2, pages 157-172, June 1995.

[9] K. Danckaert, K. Masselos, F. Catthoor, H. Man and C. Goutis. Strategy for Power-Efficient Design of Parallel Systems. IEEE Transaction on VLSI Systems, Vol 7, June 1995.

[10] J. Park and P. Diniz. Synthesis and Estimation of Memory Interfaces for FPGA-based Reconfigurable Computing Engines. In Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'03), 2003.

[11] A. H. Hormati1, M. Kudlur1, D. Bacon2, S. Mahlke1, and R. Rabbah2. Optimus: Efficient Realization of Streaming Applications on FPGAs. CASES'08, October 19–24, 2008.