2011 24th Annual Conference on VLSI Design

NoC Based Distributed Partitionable Memory System for a Coarse Grain Reconfigurable Architecture

Muhammad Adeel Tajammul,
Muhammad Ali Shami, Ahmed Hemani
Royal Institute of Technology,Sweden
Email: tajammul,shami,hemani@kth.se

Sridharan Moorthi
NIT,Trichy,India
Email: srimoorthi@nitt.edu

***Abstract*—This paper presents a Network-on-Chip based distributed partitionable memory system for a Dynamic Reconfigurable Resource Array (DRRA). The main purpose of this design is to extend the Register File (RFile) interface with additional data handling capability. The proposed interconnect which enables the interaction between existing partition of computation fabric and the distributed memory system is programmable and partitionable. The system can modify its memory to computation element ratio at runtime. The interconnect can provide multiple interfaces that can support upto 8 GB/s per interface.**

I. Introduction

DRRA is a Coarse Grain Reconfigurable Architecture (CGRA) capable of hosting multiple, complete Radio and Multimedia applications. It has resources for physical layer (PHY layer), Protocol Processing layers (PP layer), application and system control and runtime management. The DRRA fabric for the PHY layer has been implemented in [1] [2] and is shown in Figure 1 along with the proposed memory system. The DRRA PHY layer fabric will be briefly described in section III as the DRRA architecture. A single DRRA cell is composed of a morphable DataPath Units (mDPU), a Register Files (RFile), a sequencer and an interconnect scheme gluing these elements together. At present, the storage of DRRA fabric is restricted to RFiles which are 64 words of 16 bits. This paper presents an on-chip distributed memory architecture for the PHY layer of DRRA with the following salient features:

Distributed: DRRA being a fabric, the computation is distributed across the chip. Multiple threads, algorithms and applications are intended to run in parallel. To achieve true parallelism it is essential to parallelize not only computation but also the interconnect and access to memory. With distributed memory, the proposed design enables multiple private and parallel execution environments (PREX).

Partitioning: The proposed Distributed Memory Architecture (DiMArch) is partitionable. Moreover, the architecture is designed to keep the cost of partitioning and re-partitioning low both in terms of cycles and energy. Partitioning is also a distributed exercise and it happens in parallel which not only speeds up partitioning but also enables runtime re-partitioning.

Streaming: Individual partitions composed of memory banks (mBanks) should be able to act as a unit and stream data to the computational units. These streams have a generic timing model in the form of an initial delay, an intermittent delay between successive read/writes and an end delay. These delay values can be adjusted according to intermediate results to make the streams elastic.

Performance and Energy: The memory architecture should have extremely high bandwidth and very low latency. The distributed nature of memory architecture and the concept of private execution environments enable a short distance between storage and computation, which in turn contributes to low latency. The proposed design deploys wide interconnects to achieve bandwidth upto 8 GB/sec per RFile to mBank interconnect. In addition, the distributed nature of DiMArch and PREX offers effective power management by allowing the unused mBanks to shutdown or put into low power mode.

Scalability: The DiMArch is scalable with the size of memory partitions and clock frequency. The circuit-switched segments of the data Network-on-Chip (dNoC) can be optionally pipelined.

II. Related Work

Reconfigurable architectures of the past decade are investigated for memory organization in [3]. Lambrechts et al. [4] investigates power and performance for multiple forms of interconnect. The design proposed in this paper is very similar to b-neg design in [4]. The programmable pipelined and private partitioning differentiate the proposed design from [4]. Further, the control traffic is routed over bus network as it has lower traffic. Data traffic is routed over the crossbar with programmable pipelined interconnects. Memory systems for MP-SoCs can be either caches or scratch-pad based memory systems. Marescaux [5] provides a case where scratch-pad memory systems behave superior to cache based systems.

Morphosys [6] provides a set of frame buffers for parallel load/store of data between reconfigurable cells. As the size of frame buffer is fixed, the memory element to computational element ratio remains fixed.

Imagine [7] provides a three stage memory architecture, where the third stage is an off-chip memory. Beside a Local

1063-9667/11 $26.00 © 2011 IEEE
DOI 10.1109/VLSID.2011.45

Register File (LRF), a scratch pad memory and a single Stream Register File (SRF) is connected as a second stage memory. A single SRF is shared by multiple clusters but the ratio of computational element to the storage resources remains fixed.

SMART CELL [8] describes an innovative reconfigurable architecture with distributed data and instruction memory architecture. These memory units are directly connected to processing elements. Each mBank is 1K and works as a scratch pad memory. Unlike SMARTCELL, the proposed architecture can have array of memories assigned to a single mDPU without any cost on performance of other mDPUs.

MONTIUM Tile processor [9] has a local memory for each tile processor. Each tile processor is then connected to a Network-on-Chip (NoC) which can bring data from an on-chip memory via AHB-bridge interface. The local memory per tile remains fixed in each MONTIUM implementations [9] [10].

The proposed memory system can change its memory to computational resource ratio by changing its partitioning, as discussed in section V.This memory system is integrated with DRRA Architecture [1] [2] which is discussed in section III. Further-more, if the system is running with variable clock, then it can alter its critical path accordingly as discussed in section VI.

This paper is organized as follows: Section I and II present introduction and the related work on memory models of the current CGRAs. Section III presents the computational fabric, the Dynamic Reconfigurable Resource Array (DRRA) architecture. The proposed memory system is explained in section IV. Section V deals with the private execution of multiple processes. The programmable pipelining feature is briefly discussed in section VI. The costs and overheads of the different components of the system are formulated in section VII. Section VIII presents few case studies on the proposed system. Finally, section IX summarizes with concluding remarks.

III. DRRA Architecture

DRRA is a Coarse Grain Reconfigurable Architecture and the proposed memory system is shown in Figure 1. The computational portion [1] [2] is briefly described in this section. DRRA PHY resources are:

1. morphable Data Path Unit (mDPU)
2. Register File (RFile)
3. Micro-coded hierarchical sequencing machine (Sequencer)
4. A Seamless, sliding-window, circuit-switched interconnect fabric.

mDPUs are native 16-bit integer units with four 16-bit inputs corresponding to two complex numbers and two 16-bit output corresponding to one complex number. mDPU provides a) MAC, with internal and external accumulation. This MAC has an adder in front to optionally implement the symmetric FIR MAC, where the coefficients of symmetric samples are added before multiplication by the co-efficient, b) half of radix-2 butterfly for real or imaginary number c) add, subtract trees - 4 input adders/subtracters, two 2-input add/subtract trees, sum-of-difference, difference-of-sum etc. mDPU also has two comparators, one for each output and a counter. The results of comparators, counter and overflow, underflow are logged in a status word read by sequencer. mDPU can do saturation, truncation/rounding, overflow, underflow check. The end result bit-width can be configured to be anything from 8 to 16 bits. RFile - the DRRA Register File is 64 word 16 bit register file with dual read and write ports. RFile has a DSP style AGU(Address Generation Unit) with vectorized, circular buffer and bit reverse addressing that is useful in implementing FFT. Each of these modes can be executed once or in an endless loop. Each mode can have an arbitrary initial delay, a loop delay between each read/write and an end delay before the loop iterates. These delays can be used for synchronization among DSP functions, I/O and DSP rate change functions. Moreover, these delays can be dynamically computed at runtime and is the basis for providing highly sophisticated elastic streaming functionalities. Adjacent RFiles can be daisy chained to implement a shift register for a highly parallel FIR filter.

Sequencer is a micro-coded sequencing machine that controls a single mDPU and a RFile and the switchbox. Sequencer can be daisy chained to allow a single Sequencer to control adjoining Sequencers within the sliding-window reach. This concept is used to implement a hierarchy of controllers, for instance to implement Rx/Tx FSMs of a MODEM or encode/decode FSMs of a CODEC. With elastic streaming capability of RFile together with the proposed memory architecture(described in section IV) the sequencers provide the capability to implement chained elastic streaming functionalities that matches very well the nature of most PHY layers for radio and multi-media applications.

The seamless sliding-window circuit-switched interconnect fabric connects the fabric of mDPUs, RFiles and sequencers organized in two rows as shown in Figure 1. In principle, multiple such rows can be organized but the present experiments are based on outputs of mDPUs and RFiles are carried by horizontal buses, 3 columns on each side. A circuit-switched interconnect, at the intersection of horizontal and vertical buses, controlled by sequencer, selects the programmed outputs from the horizontal buses and loads them onto the vertical buses that feeds the inputs of mDPUs and RFiles as shown in Figure 1. Each column can reach 3 columns to the right and 3 columns to the left. Thus including the middle column, 7 columns constitute the window that seamlessly slides across each such 7 column window. With this scheme, each mDPU and RFile input can receive data from the output of 14 mDPUs and 14 RFiles, including itself. The fabric can be as large as the die allows; several thousand DRRA cells can be accommodated in a 45 nm 300 mm^2 die. Figure 1 shows only a fragment for clarity.

IV. Distributed Memory Architecture

The proposed Distributed Memory Architecture (DiMArch) for DRRA extends [12] which is composed of (a) a set of distributed memory banks (mBanks), (b) a circuit-switched data Network-on-Chip (dNoC) (that transports data between

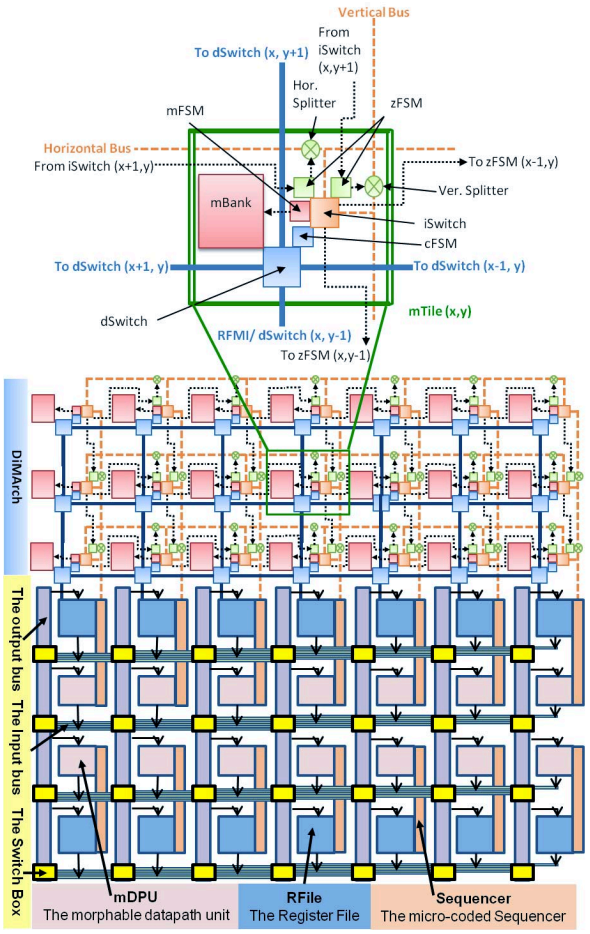

Fig. 1. DRRA Architecture

mBanks and RFiles (DRRA Register Files)), (c) a packet-switched instruction Network-on-Chip (iNoC), a NoC and bus hybrid used to create partitions, program mBanks to stream data and transport instructions from sequencer to the instruction Switch (iSwitch).

A. Memory Banks (mBank)

As stated earlier, the DRRA memory architecture is realized as distributed memory banks that are SRAM macros, typically 2 to 4 KB, a design time decision that is strongly influenced by the size of the DRRA cell (see Figure 1), as the goal is to align mBanks with the columns of the DRRA fabric. mBanks are controlled by mFSMs - state machines that also acts as interface between mBanks and the data Switches (dSwitch). mFSMs act as programmable address generation unit with a general timing model. They implement single read/write, vectorized read/writes with programmable address offset, circular buffer and bit reversed addressing. Besides the flexibility of typical DSP like address generation unit capability, mFSMs also provide a general purpose timing model using three delays, an initial delay before a loop, an intermittent delay before every read or write within a loop and an end delay at the end of the loop before repeating the next iterations. These delays are used to synchronize the memory to register file streams with the computation. Individual delays can be changed depending on the intermediate results of the computation to make the streaming behavior elastic. mFSMs are programmed via iNoC with special instructions.

B. Data Network-on-Chip (dNoC)

dNoC is a half-duplex circuit-switched mesh Network-on-Chip. The streaming nature of applications, the inherent QoS guarantees and improved latency compared to packet-switched network were the motivations for using circuit-switched network. A memory partition is defined as a set of contiguous memory banks. A computation partition is defined as a set of contiguous set of RFiles, mDPUs and its associated sequencers and switchbox (see Figure 1). A memory partition together with a computation partition is called private execution environment (PREX). The interface between memory and computational partition is as wide as the number of RFiles involved; the width here implies the number of dNoC connections, each dNoC being 256 bit wide, which is a design time decision can be changed as it is a GENERIC VHDL parameter in a template. Since the data traffic at each RFile/dNoC (RFMI) interface can only be read or written, half-duplex interconnects are proposed. dNoC is realized as a mesh network of dSwitches. As shown in Figure 2, each dSwitch is made up of five dSwitch cells (dCell) serving the N, E, W, S and the mBank directions. Each dCell has four inputs coming from the other four directions; one of these four inputs is multiplexed out in the output mode; in the input mode, data from the associated direction enters the dCell. The bi-directional I/O is optionally buffered to cope with long wires and provide flexibility to implement the planned Dynamic Voltage Frequency Scaling.

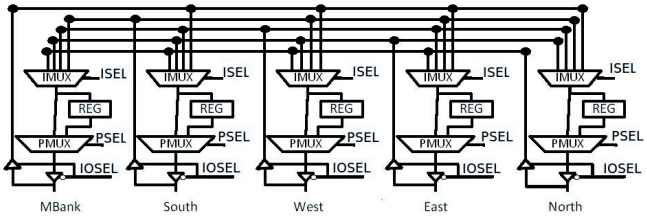

Fig. 2. dSwitch

cFSMs control the temporal behavior of dSwitch. They are essential to make multiple mBanks behave as a contiguous memory. Figure 3 shows an example of a memory partition made up of three mBanks A, B and C that bring data to a single Register File (RFile) for processing and also take the data back to the mBanks once processed. cFSMs associated with each dSwitch are programmed to time multiplex the path to and from register file in a co-ordinated way so that it appears as if RFile is reading from/writing to one large contiguous memory. Compiler ensures that the computation is synchronized with the behavior of cFSMs controlling the memory transactions. This works fine for the targeted signal processing application

with deterministic cyclo-stationary behavior. The ability to partially reprogram these streams, allows these streams to be elastic as well. The DRRA sequencers, the sequencers (see Figure 1) have the hooks to chain these elastic streams but the present DiMArch does not support chained elastic streams. The architecture can deal with the degenerate case of non-deterministic random individual memory transactions as well like a normal processor; this case will obviously not benefit from the efficiency of autonomic (elastic) streaming capability of cFSMs in DiMArch. cFSMs are programmed by special instructions via iNoC.

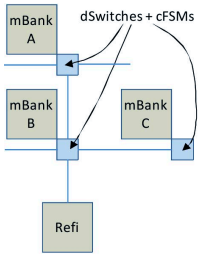

Fig. 3. Single Memory Partition

C. Instruction Network-on-Chip (iNoC)

iNoC is a packet-switched network used in DiMArch to program the cFSMs and mFSMs. The decision to use packet-switched network was taken because it is primarily used for short programming messages and life of a certain path is very short. Additionally, the generality of packetized network to reach any node of the DiMArch from any sequencer. Even if iNoC is used to transport short programming messages, agility of programming or reconfiguring DiMArchs partitions and behaviors is a key goal of the DRRA architecture to make it dynamically reconfigurable. To achieve this agility, while retaining the generality of packet-switched network, two architectural measures have been taken. The first is that the horizontal and vertical segments of the iNoC are a hybrid of bus and NoC behaviors. Any message asserted on an iSwitch is broadcast along its entire length of vertical segment, behaving like a bus as the broadcast happens in a single cycle. Every iSwitch on the vertical segment analyzes the message in parallel to check if the message address is on its associated horizontal segment and if it is, a second broadcast happens on the horizontal segment. Again, every iSwitch on the horizontal segment listens to the broadcast and analyzes if the message is addressed to it and if it is, it forwards it to zFSM that analyzes it and appropriately acts on it. By having a bus like behavior, the message is broadcast in a single cycle, i.e., each iSwitch can be reached in two cycles. The second measure to improve the agility of programming and re-programming is that these bus segments can be split to enable parallel programming of the split segments. Each horizontal and vertical segment can be potentially split with a simple message that toggles the split status; initially all segments are split. So, the first action by the sequencers is to combine the multiple segments horizontally or vertically by closing the splitters at appropriate places. This is explained with a simple example in section V. DRRA targets hosting multiple applications(MODEMs and CODECs) simultaneously. So at first level, the iNoC split segments are combined to enable each application to program in parallel and allocate appropriate memory space. At second level, each such application is very likely going to be implemented as a pipelined datapath of functions like FFTs, FIRs, Viterbis etc. Each such function will have its own thread and set of sequencers and they can program their memory partitions according to their need in parallel.

V. Private Partitioning of Multiple Processes

The concept of private partition is illustrated in Figure 4. Consider three Sequencers which need access to multiple memory banks (mBanks). mBanks in the first row have dedicated access from Sequencer in the same column. All splitters are open at the start. Sequencers 1 and 2 issue an instruction for respective instruction Switch (iSwitch) to close vertical splitter for top-to-down access. Horizontal splitters are set to remain open. Instruction Switch takes one cycle to process this instruction. After a wait of one cycle, Sequencers 1 and 2 can now access iSwitches in second row (row 1). This one cycle wait can be used to configure mFSM or cFSM for required traffic patterns. Sequencer 1 issues another instruction for iSwitch (1, 1) to get access to iSwitch (1, 2). Sequencer 1 issues instruction to close the vertical splitter top-down. Then Sequencer 1 instructs iSwitch (1,2) to close horizontal splitter right-left between iSwitch (0,2) and iSwitch (1,2). At this point all sequencers have access to their desired private partitions.

At run-time Sequencer 2 can gain access to iSwitch (2,2) to allocate more memory for additional memory requirements. However, since the proposed architecture does not provide memory locks, all access conflicts are resolved at compile time.

When two PREX needs a shared memory space, then a shared iSwitch is specified. e.g. Sequencer 0 and Sequencer 1 can specify iSwitch (0,1) as a shared space. In that case, iSwitch(0,1) will receive instructions from both PREX. Furthermore, the traffic for both cases should be deterministic and conflicts are resolved at compile time.

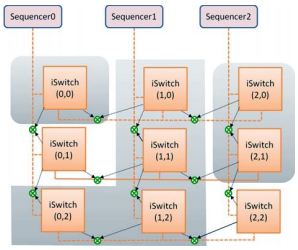

Fig. 4. Private Partitioning

VI. Programmable Pipelining

Consider an example where three RFiles connected to nine MTiles in 3x3 configuration. For a single block transfer from mBank to RFile, mBank data is first sent to dSwitch. dSwitch works either in pipelined or unpipelined mode. The pipelined mode is used when PMUX is programmed to use the register

in its path (see Figure 2). The pipelined path is omitted for Single Cycle Multi-Hop Transfer (SCMHT) mode. The concerned dSwitch routes the data to neighboring dSwitch. At destination, data is directly loaded into RFile from neighboring dSwitch. The number of cycles of each transfer is not always equal to the number of hops. The cycles may be reduced if any dSwitch is in SCMHT mode.

The critical path for single cycle transfer for any given wireload model is variable; increase in number of hops will increase the critical path exponentially. So, increasing the maximum number of hops in SCMHT mode will reduce the clock speed of the system by the same rate. Hence, the number of hops should only be increased for such cases when the gain of SCMHT mode is more than degradation due to lower clock frequency.

VII. System Costs and Overheads

The system has three types of costs and overheads interms of cycles: 1. Computational Cost ($C_{Computation}$), the time spent (cycles) in processing data by the DPU. 2. Reconfiguration Overhead, the time which is required to reconfigure the interconnect partitioning. This time is directly proportional to the number of splitters to be programmed. Programming a single splitter takes three cycles (Instruction identification, Decoding and Partition set/reset). 3. Interconnect Overhead, deals with the amount of time (cycles) it takes for the data to move between mBank and RFile.

All the costs and overheads are dependent on the method of its use. The computational cost is dependent on the mode of mDPU and data sample width since the mDPU is pipelined. For example, cost of computation of an FFT butterfly using two mDPUs is given by equation (1).

$$C_{Computation} = (N._{Sample}/2) + O_{Computation} \quad (1)$$

$$where \begin{cases} N_{Sample} = Number\ of\ Samples \\ O_{Computation} = Overhead\ of\ Computation \end{cases}$$

Overhead of computation is directly dependent on the mode of mDPU. The cost of pipeline ($C_{Pipeline}$) changes with each mDPU mode as number of stages of pipeline change.

$$O_{Computation} = C_{Pipeline} + C_{Load\ Store} \quad (2)$$

$where\ C_{Load\ Store} = RFile\ load\ store\ Cost$ Computation Interconnect Reconfiguration cost ($C._{CIR}$) is the cost of reconfiguration for the computational fabric which directly depend on the number of interconnects to reconfigure and the cost of reconfiguration. This cost can have the maximum value of six cycles.

$$C_{CIR} = N_{Interconnect} * C_{Comp.\ Reconf} \quad (3)$$

mBank Interconnect Reconfiguration cost (C_{MIR}) directly proportional to the number of instructions used to reconfigure the memory partitions (C_{PR}).

$$C_{MIR} = N_{Instruction} * C_{PR} \quad (4)$$

Reconfiguration is performed when 1. new mBank is to be allocated, 2. the instruction partitioning interconnects are reconfigured to change direction of instruction flow (or) 3. computational fabric interconnects are reconfigured. The interconnect cost ($C_{Interconnect}$) depends on data transfers and is represented as:

$$C_{Interconnect} = N_{Transfers} * C_{Num.ofhops} \quad (5)$$

$$where \begin{cases} N_{Transfers} = Number\ of\ Transfers \\ C_{Num.ofhops} = Cost\ in\ cycles/data\ transfer \end{cases}$$

VIII. Case Studies

A. Mapping one dimensional point FFT

An implementation of radix-2 FFT butterfly was carried out in mDPU [1]. Four mDPUs are used to perform two butterfly operations (one real and one imaginary). The operations are pipelined and are performed in six cycles. Depending on data access patterns, the butterfly can be reused to implement various stages of FFT. Hence, an algorithm can be defined for FFT mapping by defining the traffic pattern between RFile, mBank and mDPU. In this case, FFT traffics are manually mapped. A maximum of sixteen butterflies are used in parallel. A single butterfly is fed with data samples and twiddle factors from RFiles that can perform 32 operations in 40 pipelined cycles. Between each FFT stage, reordering and reconfiguration is performed. Reordering using the data Network is performed by common three stage data transfer. If the number of butterfly operations per stage are more than the number of available butterflies, then additional butterflies are processed serially. If re-ordering is required between neighbouring RFiles, then interconnect reconfiguration is performed instead [2].During reordering the mBank truly behaves as a scratch-pad memory, where intermediate data is stored.

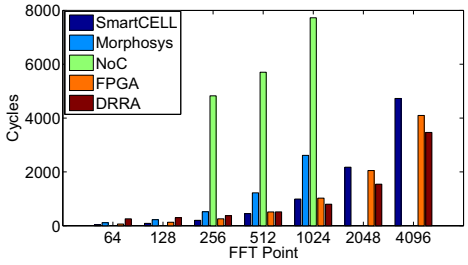

Fig. 5. FFT throughput

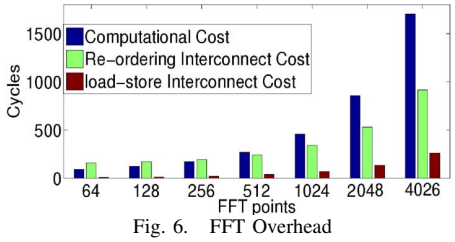

Fig. 6. FFT Overhead

First step in this mapping is loading the correct data in the correct RFile. Data is loaded from the memory to all the RFiles. By keeping a correct order at instruction level, data is picked up by the correct RFile. Twiddle factors are also loaded to RFile which act as a Look Up Table (LUT). Figure 5 extends the results of [11] for DRRA architecture. To keep the comparison fair, equal number of computational elements

are used. The last two results for SMARTCell and FPGA are interpolated based on [11]. The proposed system outperforms others by an order of magnitude more than the expected error in interpolation. For FFT larger than 512, the SmartCell is 1.24 to 1.36 times slower than DRRA.

The implementations of FFT smaller than 512 on such parallel system do not exploit locality. Figure 6 illustrates that the data for small-sized FFTs (less than 512) spends more time in motion rather than in computation. The memory-data interconnects overhead takes more or comparable times than the time spent in computation. This can be further elaborated by the fact that it takes marginally less cycles to compute the 64 point FFT using a single butterfly (real and imaginary) set. Such case uses 16 times less resources compared to the case presented (16 butterflies).

B. 2D Mapping vs McNoC [13]

The two dimensional FFT is performed in two steps. First, row-wise FFTs are calculated. This step is followed by column-wise FFT calculation. Hence two dimensional FFTs are broken down into multiple one-dimensional FFTs. It is mapped using the same principles as in section VIII-A. In this experiment, the size of FFT remains constant and number of resources are increased. Further more, an extra step is added where horizontal to vertical translation is performed. The results for such mapping are given in Figure 7.

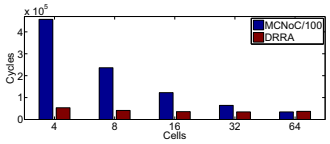

Fig. 7. Cycle count for 2D-FFT

C. Mapping Matrix Multiplication

Consider a matrix multiplication of two matrix [64, 1]with [1, 64] which results in [64, 64] matrix. Only such computation require initial load and final store operations. All the multiplications are unique and can be performed completely parallel. The performance comparison between mapping on proposed design and McNoC [13] is shown in Figure 8. The variation in cost can be better understood by looking at the overhead breakdown of such mapping given in Figure 9. When the decrease in computation cost is less than the increase in interconnect cost, then it is feasible to parallelize such system.

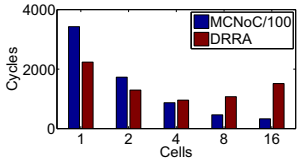

Fig. 8. MM Cycles

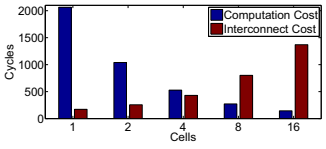

Fig. 9. MM Overhead

IX. Conclusion

This paper proposes a programmable interconnect interface as a method of communication between mBanks and RFiles. The programmable interconnect supports pipelined or bufferless modes. The instruction interconnect also has private partitioning capability where multiple sequencers can communicate with different mBanks using different segments of the network. The controllers within the system help in providing patterns of data which can be routed based on the instructions. The FFT and 2D experiments show the overhead of the interconnect, compared to the computational cost. The results show that for the given programmable interconnect the best throughput is obtained when reasonable computational resources are utilized with good locality to the data. Moreover, gate-level synthesis results show that the system can run up-to 400 MHz on 90nm technology.

As a part of future work, another sequencer will be added which will act as a dedicated main controller for memory. The instruction memory topology will be investigated for Von Neumann or Harvard architecture. More elaborate memory mapping for video application can be explored. A compiler to automate the process of mapping is also under development.

Acknowledgment

The authors would like to thank the Higher Education Commission of Pakistan for funding this research.

References

[1] M. A. Shami and A. Hemani, "Morphable dpu: Smart and efficient data path for signal processing applications," vol. SiPS, 2009, pp. 167–172.

[2] ——, "Partially reconfigurable interconnection network for dynamically reprogrammable resource array," in *IEEE 8th International Conference on ASIC (ASICON'09)*, 2009, pp. 122–125.

[3] M. Herz, R. Hartenstein, M. Miranda, and E. Brockmeyer, "Memory addressing organization for streaming-based reconfigurable computing,"*Conference on Electronics Circuits and Systems,* vol. 2, 2002,pp. 813–817.

[4] A. Lambrechts, P. Raghavan, M. Jayapala, B. Mei, F. Catthoor, and D. Verkest, "Interconnect exploration for energy versus performance tradeoffs for coarse grained reconfigurable architectures,"*IEEE TRAN. VLSI Systems,* vol. 17, no. 1, JANUARY 2009, pp. 151–155.

[5] T. Marescaus, E. Brockmeyer, and H. Corporaal, "The impact of higher communication layers on noc supported mp-socs," in *Proceeding of the First International Symposium on Network-on-Chip (NOCS'07)*, 2007, pp. 107–116.

[6] H. Singh, L. Ming-Hau, L. Guangming, F. Kurdahi, N. Bagherzadeh, and E. C. Filho, "Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications,"*IEEE TRAN. Computers,* vol. 49, 2000, pp. 465–481.

[7] S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U. J. Kapasi, and J. D. Owens, "Register organization for media processing," in *Sixth International Symposium on High-Performance Computer Architecture*, Jan. 2000, pp. 375–386.

[8] C. Liang and X. Huang, "Smartcell: A power efficient reconfigurable architecture for data streaming application," in *IEEE Workshop on Signal Processing Systems (SiPS'08)*, 2008, pp. 257–262.

[9] G. Rauwerda, P. Heysters, and G. Smit, "Towards software defined radios using coarse-grained reconfigurable hardware,",*IEEE TRAN. on VLSI*, vol. 16, no. 1, 2008, pp. 3–13.

[10] L. Smit, A. Molclerink, P. Wolkotte, and G. Smit, "Implementation of 2-d 8x8 idct on reconfigurable montium core," in *International Conference on Field Programmable Logic and Applications*, 2007, pp. 562–566.

[11] C. Liang and X. Huang, "Mapping parallel fft algorithm onto smart-cell coarse-grained reconfigurable architecture," in *Proc. IEEE Design Automation Conference, 2001* , 2001, pp. 231–234.

[12] M.A. Tajammul, el. Al. " A NoC Based Distributed Memory Architecture with programmable and partitionable Capabilities,"*NORCHIP 2010*,Tampere Finland.

[13] X. Chen, Z. Lu, A. Jantsch, and S. Chen, "Supporting distributed shared memory on multi-core network-on-chip using a dual microcoded controller,"*DATE* 2010, pp. 39–44.