MaRS: A Macro-pipelined Reconfigurable System

Nozar Tabrizi, Nader Bagherzadeh, Amir H. Kamalizad, Haitao Du

Department of Electrical Engineering and Computer Science

University of California, Irvine

Irvine, CA 92697, USA

ABSTRACT

We introduce MaRS, a reconfigurable, parallel computing engine with special emphasis on scalability, lending itself to the computation-/data-intensive multimedia data processing and wireless communication. Global communication between the processing elements (PEs) in MaRS is performed through a 2D-mesh deadlock-free network, avoiding any concerns due to non-scalable bus-based communication. Additionally, we have developed a second layer of inter-PE connection realized by distributed shared register files and conditional operands, to enhance the performance of MaRS for those applications demanding a tightly coupled PE array. We have modeled and verified a major part of MaRS. The promising results of our preliminary analyses show that MaRS can efficiently be tailored to different applications offering flexible data communication, and high performance.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors) - *Interconnection architectures, Multiple-instruction-stream, multiple-data-stream processors (MIMD)*

General Terms

Design

Keywords

2D-mesh network, computer graphics, MIMD, multimedia, reconfigurable architectures, wireless communication.

1. INTRODUCTION

Multimedia processing and wireless communication are gaining a remarkably growing attention in both academia and industry; so that achieving low-power, high-performance, flexible solutions to properly handle complex media and wireless communication tasks in real time and in a cost-effective manner is a challenge. The challenge is significantly highlighted when different requirements of different media in a multimedia environment, with different and also evolving standards are taken into consideration. The existing multimedia processors cover a wide range in terms of both architecture and functionality.

Sun UltralSPARC [19] is a superscalar processor with an issue rate of up to 4, featuring the Visual Instruction Set (VIS) multimedia extension to accelerate data-parallel applications. This extension is a comprehensive SIMD accelerating engine incorporated into a general-purpose processor, where packed multiple data in a register undergo the same operation in parallel, giving rise to "SIMD within a register".

A similar idea was then implemented by other companies such as Intel (MMX and SSE) [11, 18], Motorola (AltiVec) [4] and HP (MAX) [8]. Additionally, several DSPs now also provide SIMD functionality, such as Analog Device's TigerSHARC [5].

DART [2] is a reconfigurable architecture with fixed point arithmetic only. Its current implementation consists of four clusters (macro-pipeline stages in terms of the MaRS terminology), working independently of each other and having access to the same data memory. An external controller has to only allocate the right tasks to the right clusters. Each cluster contains six coarse grain reconfigurable data paths (DPR) and a fine grain FPGA core. The communication between these reconfigurable blocks is performed through a shared memory and also some reconfigurable point-to-point connections (second-level interconnection). A programmable processor is in charge of controlling the whole reconfigurable blocks. Each DPR has four programmable functional units with four local memories. The communication between theses blocks is carried out through some reconfigurable local buses (first-level interconnection).

RAW [17], with over 120 million transistors, is another parallel processor targeting the wire-delay problem. The current implementation of RAW is comprised of an array of 4 by 4 identical programmable tiles interconnected by two 2D-mesh static and two 2D-mesh dynamic networks, which entail 16 input- and 16 output-channels for each tile. One static and two dynamic routers, and an eight-stage single-issue RISC processor with a floating-point arithmetic unit, and a data and an instruction cache build up the backbone of one tile. The size of each tile has been determined such that it takes around one clock period for a signal to travel the longest possible path. This guarantees that no longer wires may be introduced as the number of tiles laid on the silicon is increased. In other words, the clock frequency becomes independent of the number of tiles, facilitating the scalability of RAW.

MaRS is an advanced successor of MorphoSys [13], a reconfigurable SIMD System-on-Chip. MorphoSys was first developed and fabricated in 1999, at UC-Irvine. Several computation-intensive and data-intensive algorithms have successfully been mapped onto MorphoSys [14, 15], and also onto the second version of this processor [1, 3, 6, 7, 10, 12] with some major functional enhancements over the first version. MaRS is targeting some shortcomings of MorphoSys resulting in a scalable and flexible computing engine for wireless communication and multimedia applications.

MorphoSys is an SIMD processor with a matrix of 8 by 8 PEs, functioning as tightly coupled peripherals to a central general-purpose processor. Every instruction to be executed by each PE is directly determined by this controlling processor. Each instruction not only specifies the type of operation, but also determines the operand sources. An operand source may be an internal register, another PE, or a global data memory. This PEs-shared memory is also under direct control of the central processor. These different aspects of dependence on the central controlling processor may easily result in poor PE utilization, and eventually performance degradation.

MorphoSys is not scalable. Data and instruction broadcast to the PE array, and also some inter-PE communication in this processor are performed over global buses. However, this type of signal path cannot easily be scaled as our hypothetical MorphoSys utilizes larger PE arrays. Even in the current versions long buses have to be considered thoroughly, and are usually a major source of design backtracking. As a matter of fact, wire delay is becoming a major constraint in the implementation of large processors, so that while wires used to interconnect logic gates in the past, in today's technology the situation is being reversed: wires are said to be interconnected by logic gates. Therefore, generous use of wires is no longer consistent with modern, high performance massively parallel processors.

Memory hierarchy in MorphoSys has to be improved for the growing PE array, as a global data memory and a global instruction memory cannot efficiently exploit the possible spatial and temporal localities. Furthermore, the current off-chip memory bandwidth is a rigid bottleneck for non-streaming data intensive applications, such as the binary spatial partitioning- (BSP-) based ray tracing.

A major by-product of the above shortcomings is that concurrent kernels are not supported by MorphoSys, resulting in either performance degradation or much-complicated interfacing between several single-kernel engines.

MaRS is an attempt to relax the above concerns, hence to provide a breakthrough computing engine for efficient mapping of highly parallel data- and/or computation-intensive wireless communication and multimedia applications.

There will eventually be hundreds of PEs in MaRS loosely coupled to some group controllers, as illustrated in Figure 1, resulting in a much higher performance for the intended applications than what is normally achieved through traditional processors, and also domain specific processors such as MorphoSys. The group controllers bind together a number of PEs as a macro-pipeline stage. Then several macro-pipelines may operate at the same time executing different kernels concurrently, and hence tailoring the system to the intended application. Notice that macro-pipeline stages are reconfigurable in terms of shape and the number of participating PEs. We have successfully modeled and verified the whole PE array. We have also synthesized most of the major blocks such as the ALU and MAC unit, in the 0.13μm TSMC technology, followed by some successful post-synthesis simulations.

The backbone of MaRS is a 2D mesh of 32-bit PEs. Each PE is comprised of a *router* and an *execution unit* (EU). Each EU in the current implementation features guarded instructions, 8kB of two-port data RAM, 1kB of instruction RAM, a 16-word register file, an ALU, and a multiply-accumulate (MAC) unit supporting signed/unsigned 16-bit real/8-bit complex multiplications.

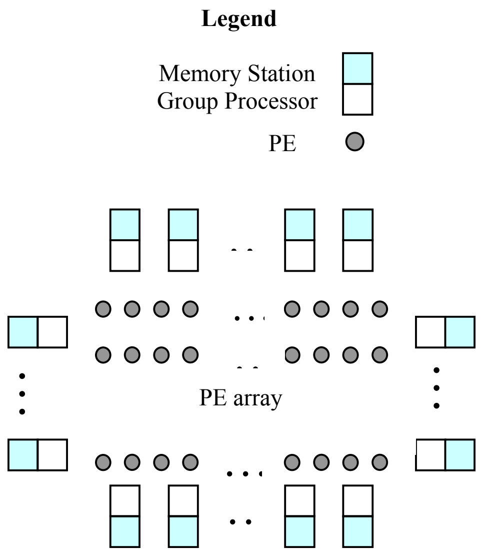


Figure 1- The top level MaRS

We have developed and implemented an adaptive, minimal routing, circuit-switched communication network for MaRS based on a deadlock-free protocol. This network supports point-to-point single transaction, and also point-to-point/multicast block transfer, initiated by any of the PEs, or the surrounding memory stations.

The exact number of PEs in the network is implementation dependent. Each PE is interconnected to its four neighboring PEs through 12 channels as discussed shortly. Any of the participating PEs may be replaced with a floating-point unit (FPU). Each added FPU is able to provide any PE in the network with the requested floating-point operation, using the same network protocol.

In order to further enhance the performance of MaRS for some specific applications such as the Viterbi decoder requiring a tightly coupled array of local PEs, and in addition to the above global communication network, we have also developed a second layer of inter-PE connections realized by distributed shared register files and conditional operands. In this layer each PE has direct but limited access to the register file of some remote PEs. Conditional operands, introduced in the second communication layer, provide an efficient handshaking and synchronization scheme for the producer/consumer sides of every commutation carried out in this layer.

Only the inter-PE-communication-based features of the MaRS architecture are elaborated in the rest of this paper, which is organized as follows. In Sections 2 and 3 the PE and the FPU, respectively, are discussed. In Section 4 the second layer of inter-PE connection is addressed. Section 5 shows some preliminary analysis for three prospective applications, namely the BSP-based ray tracing, a 64-point FFT, and the Viterbi decoder. Section 6 is the conclusion and the ongoing work.

2. PE

The PE is the major component of the network. The participating PEs are interconnected through channels. Each PE is comprised of an execution unit (EU) and a router. All data processing tasks are performed in the EU. The router is in charge of directing the ongoing traffic toward the corresponding destination PEs. The incoming data/instructions are also absorbed by the router once they reach the destination. The router also lets the locally generated blocks enter and then ripple through the network to reach the destination.

2.1 EU

In addition to the traditional instruction set, each EU supports different types of communication through three network-specific instructions, namely *GET*, *PUT* and *PUT BLK*.

The instruction pair PUT (on the transmitter side) and GET (on the receiver side) realizes the single transaction point-to-point communication, and is normally used for process synchronization between the PEs. PUT dispatches one word of data to the specified PE in the network, while GET receives one word of data from the corresponding source PE. More specifically, the instruction "*PUT R1, R2;*" injects the content of register R1 to the network to eventually reach the PE pointed by register R2. The complementary instruction "*GET R1, R2;*" on the other hand, receives a word from the PE pointed by register R2. R1 is the destination register. Notice that due to possible network congestion and also unknown instant of instruction fetch, both PUT and GET have a nondeterministic period of execution cycle.

The GET instruction is bound to the corresponding PUT instruction; that is, the former has to wait until the required data arrives. In case of an early arrival, the data is temporarily saved in a content addressable memory (CAM), and then upon the execution of the GET instruction the right data is located and fetched from the CAM. This mechanism also supports multiple early arrivals.

The "*PUT BLK R1;*" instruction initiates the transfer of up to 1k-byte blocks, leaving the local RAM of the source PE, or a memory station, and heading to the local RAM of the destination PE, or a memory station in point-to-point (one-to-one) block transfers, or the local RAMs of a group of PEs in multicast (one-to-many) mode. Register R1 points to the beginning of the block in the source RAM. Instruction blocks, of course, are not allowed to leave instruction RAMs. They normally flow from the memory stations towards the local instruction RAMs in different PEs. Multicast mode results in a significant saving in power dissipation, comparing to the equivalent multiple point-to-point block transfers. In this mode the destination PEs (a macro-block) may be specified and arranged in an arbitrary shape. According to our current implementation, a macro-block may be comprised of a stack of up to four 8- by 8-PE (or smaller) rectangles, with an indentation of up to 7 PEs for each rectangle. Figure 2 shows a multicast to a 4-rectangle macro-block, initiated from a memory station. The values in parentheses show the corresponding vertex coordinates to be specified in the header. For each rectangle two vertices located on the left-to-right diagonal have to be specified.

Upon the power-on-reset, each EU is forced to execute a single-instruction wait loop until an instruction block reaches the instruction RAM. Then the execution path will be redirected towards the newly downloaded piece of code. Considering that instruction pumping into the network during an instruction-block transfer by a memory station may not be interrupted once the block header has reached the destination, the PE does not have to wait for the end of instruction-block transfer to begin execution. In order to leave the PE in a waiting situation when the execution of a piece of code is over, MaRS features a software reset instruction, *HALT*, to be used at the logical end of programs. The HALT instruction forces the EU to enter the same single-instruction wait loop again.

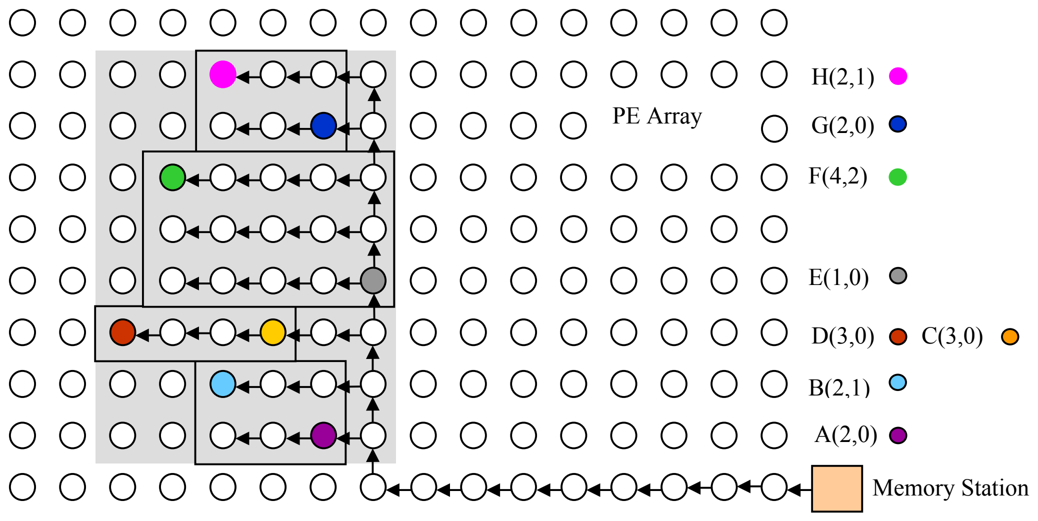


Figure 2- A multicast to a 4-rectangle macro-block

2.2 Routers and Channels

We have developed and implemented a 64-bit (double word) adaptive, circuit-switched, 2D-mesh communication network for MaRS featuring a deadlock-free protocol. Figure 3*a* illustrates a single router with 6 input- and 6 output-channel ports, employed in every node of the network. The 12 surrounding channels are also shown in this figure.

There are two north and two south channel pairs reaching each router, providing two disjoint sub-networks for the west-to-east and east-to-west traffics using the channel sets {W-in, N1, E-out, S1} and {E-in, N2, W-out, S2}, respectively. This allows the network avoid cycles in its channel-dependency-graph, resulting in a deadlock-free operation [9]. Each channel is comprised of a 4-double-word FIFO, and the corresponding sets of physical wires, as shown in Figure 3*b*.

As soon as an outgoing channel is allocated to a double-word header, the channel will remain dedicated to the corresponding block until the tail end of the block passes through the channel. This guarantees that an instruction block leaving a memory station will not be interrupted once the header has reached the destination. However, that is not true for data-block transfers initiated by a PE in our implementation, as an incoming data block heading to the same node does stop the outgoing data transfer already in progress.

Notice that in single transactions no block body follows the header. Now in fact a 32-bit header (short header) is appended to the 32-bit data. The resulting double-word data/header then ripples through the network exactly in the same way that a 64-bit header does in a point-to-point block transfer.

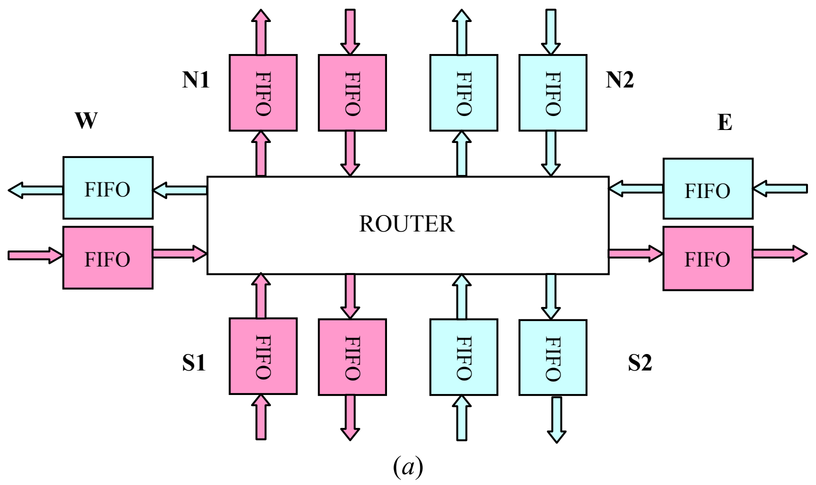
The route traversed by a block header is nondeterministic, as each header adapts its direction to the current situation while stepping from one node to a neighboring node. For outgoing channel allocation the router applies a fixed priority scheme to the incoming headers reached the corresponding node simultaneously: for each outgoing channel, the possible incoming channels have a descending order of priority in a clock-wise direction. For example, for the outgoing channel W, the channels N2, E and S2 are the three possible incoming channels in descending order of priority.

In addition to the above four incoming channels, there are two more sources requesting an outgoing channel in each node, namely the local RAM when a PUT BLK instruction is executed, and the execution unit when a PUT instruction is executed. We have allocated the lowest and the second lowest priority, respectively, to these two sources in our current implementation.

Notice that the route traversed during a block transfer is strictly *monotonic*; in other words for each incoming header there are at most two logically possible outgoing channels, always resulting in a *minimal* route. If the first-choice outgoing channel cannot be allocated to a header, the second choice (if any) will be granted if it has not already been dedicated, and there is no priority violation either.

3. FPU

Notice that there is no floating-point arithmetic unit in the PE. However FPUs may replace as many PEs in the network as required. The idea of distributed FPUs utilized in MaRS provides sufficient support for multimedia processing, while silicon area overhead due to floating-point-capable PEs is still avoided. Each added FPU is able to provide any PE in the network with the requested floating-point service using the same network protocol, while the FPU remains transparent to the ongoing traffic in the network. Each FPU is also comprised of a *router* (FP-router) and an *execution unit* (FPEU), as articulated in the following subsections.



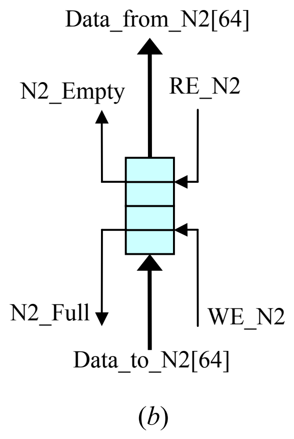


Figure 3- (*a*) A 12-channel router, and (*b*) a single outgoing channel

3.1 FPEU

The FPEU supports the IEEE 754-based single precision floating-point addition, subtraction, and multiplication. In the current implementation FPEU is a multicycle unit. Its pipelined version will be utilized in the upcoming implementations of MaRS. Notice that all supported floating-point operations need 32-bit operands, and therefore one double-word block transfer by the requesting PE suffices to provide the FPEU with both of the operands. The operation type and the source/destination addresses are transmitted in the block header. As soon as the computation is carried out, a single transaction is initiated by the FPEU's controller to dispatch the 32-bit result to the requesting PE. The matching GET instruction on the PE side will receive the operation result. Notice that there is no local instruction or data RAM in the FPEU.

3.2 FP-router

The FP-router is still in charge of routing the ongoing traffic, which reaches the corresponding FPU. Furthermore, all floating-point requests and the computation results are directed by this router. Notice that the incoming blocks to a FPU are handled differently. These requests enter a floating-point FIFO (instead of the local RAM) in the FPU, and then are served on a first-in first-served basis. Due to the non-pipelined and multicycle architecture of the FPEU the floating-point FIFO is likely to become full under heavy load conditions. There are two more major changes to the FP-router: 1- there is no outgoing block transfer from this router, 2- since the FPU cannot be a destination for a multicast, the FP-router simply ignores all such requests.

4. THE SECOND LAYER OF INTER-PE CONNECTIONS

Some applications, such as the Viterbi decoder, demand a closer interconnection between the participating PEs to achieve the required performance. In order to realize this layer of inter-PE communication we utilize distributed shared register files; so that in the current implementation of MaRS half of the register file (16 remote registers located in four remote blocks) of each PE (the root PE) is distributed to four remote PEs, namely PE-N2, PE-N4, PE-E2, and PE-E4, as shown in Figure 4.

To support distributed shared register files we introduce 33-bit conditional operands consisting of the normal 32-bit data field of a register concatenated with one valid bit. Write-into-register and read-from-register operations have now two different modes: unconditional (normal) and conditional. A conditional write waits for the destination register to become invalid (if it is not already invalid) before the data is written into that register, and then the destination is marked as valid, indicating a valid operand in the 32-bit data field of the register. A conditional read, on the other hand, will read the 32-bit data-field of the source register if only it is marked as valid. The source register is then flagged as invalid when the read operation is carried out.

For example the instruction *ADD R27c, R17, R29c* with a conditional read from register R29 (signified by the suffix "c"), and a conditional write into register R27 (signified again by the suffix "c"), waits until registers R29 and R27 become valid and invalid respectively, and then saves the result of *R17 + R29* into R27, while R29 and R27 are flagged as invalid, and valid respectively.

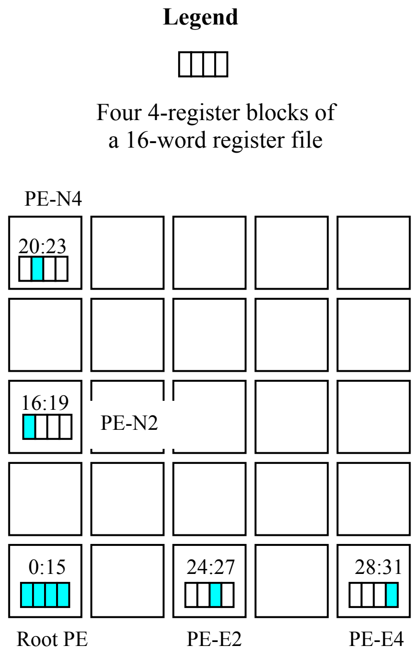


Figure 4- Distributed shared register file

Notice that in addition to a tighter interconnection, the conditional read/write operations provide the participating PEs with a fast and straightforward handshaking and synchronization mechanism as well.

The valid bit is totally ignored in normal-mode read operations; that is, the read operation is performed unconditionally, and the corresponding valid bit remains unchanged after such a read. Write operation, on the other hand, is still subject to an invalid destination. However, the destination remains invalid after such a write.

In the current implementation of MaRS each block of remote registers (located in one remote PE) allows only one access at a time, however, four remote registers in four different remote blocks may be written simultaneously by the root PE.

5. SOME PRELIMINARY STUDIES ON APPLICATION MAPPING

MaRS is targeted at large concurrent kernels. As the very first steps of its performance evaluation we have carried out some preliminary studies on three applications, as presented in the following three subsections.

5.1 BSP-based Ray Tracing

Ray tracing is a technique to render highly realistic images of 3D scenes. It works by simulating how photons travel recursively in the real world. In the BSP algorithm [16] the universal cell, representing the environment, is recursively partitioned into two sub-cells up to a specific depth, and the resulting sub-cells are described by a binary tree called BSP tree. A ray first traverses the tree until a terminal cell (leaf) is reached. Then the objects inside the cell are tested to see whether or not the ray hits any of them. A hit implies that an object has successfully been detected at some specific point in the space; otherwise the ray continues to traverse the tree. Therefore, there are two different processes in ray tracing: *ray traversing* (along BSP tree) and *ray-object intersecting*. Considering MaRS as the target platform, these processes are two kernels running in parallel on two macro-pipeline stages of this processor.

Notice that the traversing procedure is not a data-streaming task, and has little spatial and temporal locality, demanding a high memory bandwidth. Furthermore, a large BSP tree prevents it from being an on-chip resident, and this gives rise to further memory bandwidth requirement. Therefore, while the corresponding high computation demand can easily be satisfied by the on-chip computing resources, the high memory bandwidth requirement has to thoroughly be studied to avoid serious performance degradation. Our approach to overcome this problem, hence efficiently utilize MaRS for ray tracing, is to sufficiently shorten the BSP tree to reach an acceptable off-chip memory bandwidth, at the cost of some computation overhead, which can be tolerated by the sufficient on-chip computing resources in MaRS.

5.2 FFT

FFT is widely used in multimedia applications, and therefore the corresponding algorithms are very decisive in evaluating the MaRS performance. As the first attempt for this evaluation we have developed a code and mapped it on a 4 by 4 array of PEs in MaRS to perform a 64 point-complex FFT, including the presorting stage. Table 1 shows the instruction statistics of this parallel code. The total number of cycles is 70 including 8 cycles to perform presorting. Considering 1104 clock cycles required to do the same job on a TMS320C62x, our achievement on MaRS is promising.

Table 1: Instruction statistics for FFT

Instruction	Instants	Instruction	Instants
PUT	176	Complex Add	192
GET	176	Complex Sub	192
Complex Mul	129	Load Imm	53
Move	16	NOP	160

5.3 Viterbi Decoder

Convolutional encoding is a choice of considerable interest in high-speed reliable digital communication. In a binary convolutional code (CC) specified as (n,k,m), k and n are the number of input and output data bits, respectively, and 'm' is the memory length of the encoder.

The Viterbi decoder is a well-known maximum likelihood technique for decoding convolutionally encoded bits. It is a communication intensive algorithm, which can easily be implemented as an ASIC. This, however, is not a flexible solution; hence we seek to design a programmable engine lending itself to different and also evolving standards.

The IEEE 802.11a standard supports bit-rates up to 54 Mbps for high speed WLAN transceivers. This standard uses CC(2,1,7), consisting of 64 states, and 2 branches entering each state. Considering the high bit rate of this application, only fully node-parallel Viterbi architecture may be utilized to achieve the required performance.

We use 16-entry look-up tables to generate soft-decision bit metrics. Our preliminary mappings on the current implementation of MaRS show that the Viterbi algorithm may be performed in 6 cycles. We expect to reduce this amount to 4, on the next version of MaRS. This improvement will be achieved by increasing the network communication frequency by a factor of two.

6. CONCLUSION AND ONGOING WORK

In this paper we have introduced MaRS, a macro-pipelined reconfigurable system with the backbone of a 2D-mesh circuit-switched network, targeting wireless communication and multimedia applications. The first implementation of the network, PE, and FPU has successfully been verified. The proposed communication architecture in MaRS is scalable, and this is the main achievement when MaRS is compared to MorphoSys, the MaRS' predecessor. Considering the promising results we have achieved in our first implementation, and the preliminary application mappings on MaRS, the ongoing work is being carried out in the following directions:

Memory architecture: The memory stations discussed above are part of the whole memory hierarchy to be designed and implemented. We need more research to identify data access patterns in different applications and more specifically temporal and spatial locality of references to tailor MaRS's memory hierarchy to the wireless communication and multimedia processing needs. The off-chip memory bandwidth requirement will also be investigated to achieve an optimum power-performance trade-off.

Group processors: These processors are in charge of setting up the intended macro pipelines and initializing the participating PEs by transferring proper instruction and data blocks from the off-chip RAM to the on-chip memory stations, and controlling the memory stations as well. Some more work is required to extend scalability to the group processors.

Synchronous clock zones: Clock skew has always been a major concern in the complexity management of large designs such as MaRS. In two-phase logic clock skew (Δt) can totally be hidden at the cost of Δt loss (dead time) out of the whole timing budget (one clock period). Therefore, for a fixed clock skew, the higher the clock frequency, the higher overhead due to the clock skew. To overcome this problem in the single-phase MaRS, we introduce *synchronous clock zones* in which clock skew is guaranteed to remain within the safe range. Then signals passing over the border of adjacent clock zones have to be synchronized with the destination clock. Considering our implementation and also the available clock-tree generators, more studies are required to approach a realistic and safe estimate for the area of a synchronous clock zone.

Software support for MaRS: For a sophisticated system such as MaRS, proper software tools are inevitable. Considering our potential resources we are planning to develop an assembler and a cycle-accurate simulator for MaRS.

7. ACKNOWLEDGMENTS

This work was sponsored by State of California CoRe funded research in cooperation with Broadcom Corporation.

8. REFERENCES

[1] Anido M. L., Tabrizi N., Du H., Sanchez-Elez M., and Bagherzadeh N., "Interactive Ray Tracing Using a SIMD

Reconfigurable Architecture," in *Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing, SBAC-PAD*, Vitoria/ES – Brazil, October 28-30, 2002.

[2] David R., et al, "A Dynamically Reconfigurable Architecture Dealing with Future Mobile Telecommunications Constraints," in *proceedings of Parallel and Distributed Processing Symposium, IPDPS,* pp. 156 -163, 15-19 April 2002.

[3] Du H., Sanchez-Elez M., Tabrizi N., Bagherzadeh N., Anido M. L., and Fernandez M., "Interactive Ray Tracing on Reconfigurable SIMD MorphoSys," in *Proceedings of Design, Automation and Test in Europe (DATE03)*, pp. 144-149, Munich, Germany, March 3-7, 2003.

[4] http://www.motorola.com/SPS/PowerPC/AtiVec/facts/.html

[5] http://www.analog.com/processors/processors/tigerSHARC/whitePapers/newArch.html

[6] Kamalizad A. H., Pan C., and Bagherzadeh N., "A Very Fast 8192-Point Complex FFT Implementation Using MorphoSys Reconfigurable DSP," in *Proceedings of 15th Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2003*, pp. 254-258, São Paulo, SP – Brazil, November 10-12, 2003.

[7] Koohi A., Bagherzadeh N., and Pan C., "A Fast Parallel Reed-Solomon Decoder on a Reconfigurable Architecture," *CODES+ISSS*, pp. 59-64, Newport Beach, California, October 1-3, 2003.

[8] Lee R. B., "Subword parallelism with MAX-2," *IEEE Micro*, vol. 16, pp. 51–59, Aug. 1996.

[9] Ni L., and McKinley P., "A Survey of Wormhole Routing Techniques in Direct Networks," *COMPUTER*, vol. 26, pp. 62-76, 1993.

[10] Pan C., Bagherzadeh N., Kamalizad A. H., and Koohi A., "Design and analysis of a programmable single-chip architecture for DVB-T base-band receiver," in *Proceedings of Design, Automation and Test in Europe (DATE03)*, pp. 468-473, Munich, Germany, March 3-7, 2003.

[11] Raman S. K., Pentkovski V., and Keshava J., "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE Micro*, vol. 20, pp. 47–57, July/Aug. 2000.

[12] Sanchez-Eleza M., Du H., Tabrizi N., Long Y., Bagherzadeh N., and Fernandez M., "Algorithm optimizations and mapping scheme for interactive ray tracing on a reconfigurable architecture," *Journal of Computers & Graphics,* Vol. 27, No. 5, 2003.

[13] Singh H., Lee M., Lu G., Kurdahi F. J, Bagherzadeh N., and Filho E. M. C., "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation –Intensive Applications," *IEEE Trans. Computers*, Vol. 49, No. 5, pp. 465-481, May 2000.

[14] Singh H., Lee M., Lu G., Kurdahi F. J., Bagherzadeh N., Filho E. M. C. and Maestre R., "MorphoSys: Case Study of A Reconfigurable System for Multimedia Applications," in *Proceedings of Design Automation Conference (DAC 2000)*, Los Angeles, June 2000.

[15] Singh H., Lee M., Lu G, Kurdahi F. J., Bagherzadeh N., and Filho E. M. C., "MorphoSys: A Reconfigurable Processor Targeted for High-Performance Image Applications," in *Proceedings of the Reconfigurable Architectures Workshop*, Puerto Rico, April 1999.

[16] Sung K., and Shirley P., "Ray Tracing with the BSP-Tree," *Graphics Gem III*, 271-274, Academic Press 1992.

[17] Taylor M. B., et al, "THE RAW MICROPROCESSOR: A COMPUTATIONAL FABRIC FOR SOFTWARE CIRCUITS AND GENERAL-PURPOSE PROGRAMS," *IEEE Micro* Volume: 22 Issue: 2, pp. 25 -35, March-April 2002.

[18] Thakkur S., and Huff T., "Internet streaming SIMD extensions," *Computer*, vol. 32, no. 12, pp. 26–34, Dec. 1999.

[19] Tremblay M., and O'Connor J. M., "UltraSparc I: A four-issue processor supporting multimedia," *IEEE Micro*, vol. 16, pp. 42–50, April 1996.