Unifying manycore and FPGA processing with the RUSH architecture

Brandon Beresini
Maxentric Technologies, LLC
beresini@maxentric.com

Scott Ricketts
Maxentric Technologies, LLC
sricketts@maxentric.com

Michael Bedford Taylor
Angstromush, LLC
michael.b.taylor@angstromush.com

Abstract

Because of the constraints of space computing, the set of available processing technologies is limited. Conventionally, designers have had to choose from programmable rad-hard processors and fixed ASIC solutions. FPGAs provide significantly better power-performance efficiency than general purpose processors, but are more costly to program and are less flexible. For terrestrial applications, manycore processors have been adopted for a class of applications where both performance and flexible programmability are important metrics. Maestro, the first rad-hard manycore processor, has the potential to enable new capabilities for space computation. However, for many applications, certain timing-critical tasks still require the performance efficiency of an FPGA co-processor. Moreover, integrating such heterogeneous systems is challenging because the individual processing substrates have differing internal programming models. As a result, data sharing and dynamic workload scheduling across heterogeneous architectures are often suboptimal and hindered by poor scalability. The Rad-hard Unified Scalable Heterogeneous (RUSH) architecture is a heterogeneous processing platform with both a manycore chip and an FPGA. RUSH provides a unified programming model across both chips to allow for rapid development of scalable and efficient implementations. This paper overviews RUSH's technical approach and presents an example application: a WiMAX radio transceiver.

1. Introduction

Space presents a challenging environment for computing. Table 1 shows a categorized spectrum of the current landscape in radiation hardened processing substrates. On one end are general purpose processors (GPP) like the RAD750, which offer excellent serial performance and low effort for mapping an application to the processor. On the other end, ASICs, such as those implemented in a rad-hard-by-design (RHBD) flow, offer best-of-class performance and energy efficiency on parallel codes. But they have high application mapping effort, are fixed function, and have high NRE manufacturing costs. Between, there are two classes. FPGAs, such as the Virtex-5QV, have similar mapping costs to ASICs but allow late-binding changes and offer performance and energy-efficiency on parallel codes superior to GPP's but less than ASICs. Manycore processors, which have recently gained momentum in the terrestrial domain, are the class of multicore processors that are scalable to 10s or 100s of processors. Manycores inherit many of the benefits of GPPs: low mapping costs, excellent serial performance and low NREs. At the same time, they offer superior parallel performance to GPPs, which depending on the application, can be lower or higher than an FPGA. Terrestrial examples include Tilera's TilePro64 and Tile-Gx series [17] and the massively parallel architectures driven by graphics computing in Graphics Processing Units (GPUs) from companies like Nvidia [12] and AMD [3].

Class	**GPP**	**Manycore**	**FPGA**	**ASIC**
e.g.	RAD750	Maestro	Virtex-5QV	RHBD
More parallel performance and energy efficiency $\longrightarrow$				
$\longleftarrow$ Lower mapping effort, lower NRE cost				

Table 1. Four Classes of Processing Substrates for Space. Generally, there is a tension between power-performance efficiency and programmability.

Maestro, a 49-core rad-hard processor from the OPERA program [18], is the first manycore processor to target space applications. Figure 1 overviews the Maestro architecture.

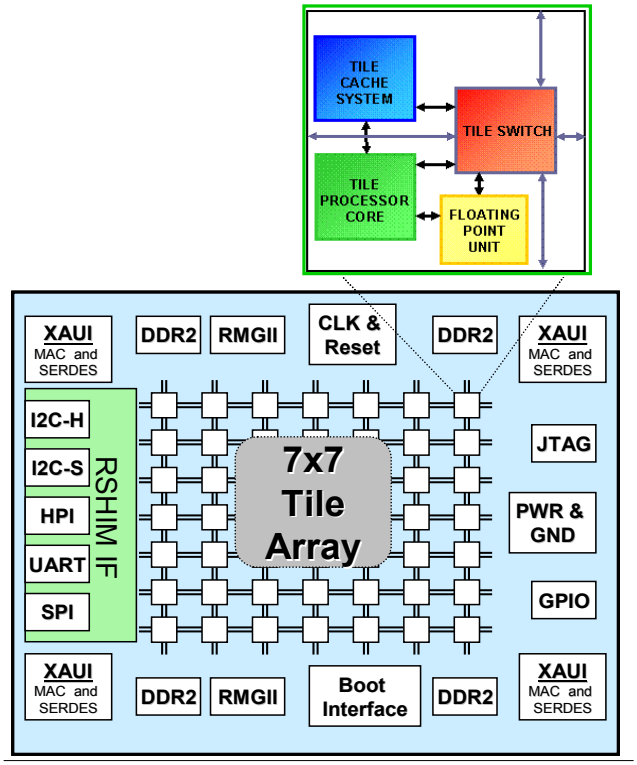

Figure 1. The Maestro Rad-hard Manycore Processor. Maestro has 49 tiles as pictured. Tiles are connected with dynamic and static on-chip interconnects. Image source: [13].

Maestro arranges an array of general purpose *tiles*, each of which includes a processing core based on the Tilera TILE64 [4], a floating point unit, a cache system, and a switch that interfaces with the on-chip interconnect. The on-chip interconnect implements low latency communication between tiles, allowing for message passing for data flow and synchronization. Additionally, the chip provides a cache-coherent shared memory system.

Maestro is the result of a significant body of research and development across government, academic, and private institutions. The DARPA PCA program led to the development of the 16-tile Raw processor at MIT [16], which would later be commercialized by Tilera as the 64-tile TILE64. Meanwhile, the DARPA/DTRA RHBD program, in partnership with Boeing SSED, drove the development of a 90 nm RHBD process that would later be used to harden the TILE64 tile in 2008. This finally led to the fabrication of the first release of 49-tile Maestro chips in 2011. Maestro uses IBM's 9SF 90 nm CMOS process and targets a total ionizing dose (TID) of at least 500 krad (Si). As a general purpose manycore processor, Maestro runs Linux and supports C/C++ programmability through a familiar gcc- and eclipse-based toolchain.

The Maestro on-chip interconnect is called the *iMesh*, consisting of one static network and four dynamic networks. The static network is for data flow that can be determined at compile time: routers are programmed along with the cores to move data throughout the chip. The dynamic networks are packet-based, encoding routing information in headers for dynamic routing and latency insensitive data flow. The four dynamic networks are the user dynamic network (UDN), tile dynamic network (TDN), memory dynamic network (MDN), and IO dynamic network (IDN). As their names indicate, the UDN is for user-level message passing, the TDN is for tile-to-tile messaging, the MDN is for memory-subsystem traffic, and the IDN is for communication with I/O devices. This design thus provides a uniform, network-based message passing interface to all of the fundamental architectural resources of the chip: the tiles, the L1 and L2 caches, the off-chip memory, and I/O devices. That is to say that each of these architectural resources is effectively a client on the interconnect and can communicate with any other client using the appropriate messaging protocol and the uniform packet-based iMesh mechanism.

Figure 2(a) shows an example manycore application, namely a WiMAX [1] software-defined transceiver targeting Maestro. This software-defined radio (SDR) physical layer workload is typically implemented on FPGAs, but our benchmark results indicate that it is indeed feasible to support this workload with Maestro. That said, for higher bit rates or other more demanding performance requirements, it might make sense to offload the demanding computational blocks to an FPGA. For example, as shown in Figure 2(b), we might consider tasking an FPGA with the FFT and Viterbi decoder blocks from the WiMAX receive physical layer. We estimate that this would save roughly half of the Maestro chip for other tasks or for low power sleep mode.

Thus, a heterogeneous processing platform built from a manycore chip and an FPGA could provide a solution that has excellent power-performance efficiency while keeping development costs down by keeping most of the logic on the highly flexible manycore chip and offloading just the computational bottlenecks to the FPGA. However, integration of heterogeneous processing platforms, such as the Maestro+FPGA architecture as described, is a challenging effort:

1. **Hardware integration** requires that access be provided across chip boundaries to architectural resources such as processors, memory systems, and IP cores. Moreover, physical layer communication protocols between the chips must be reconciled.

2. **Software integration** requires that software must use the correct semantics and interfaces for cross-chip communication – e.g. message formats, synchronization, data streaming models, etc.

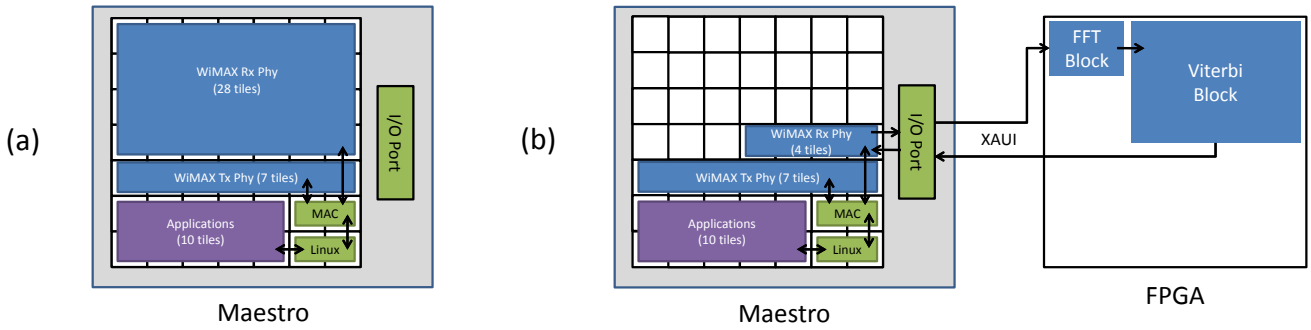

Figure 2. Offloading SDR bottlenecks to an FPGA. Pictured are two approaches to a software-defined WiMAX transceiver. In (a), all functionality is implemented on Maestro. In (b), the FFT and Viterbi decoding blocks are computed on an FPGA co-processor. Such computationally demanding signal processing blocks can be more efficiently run on an FPGA, and thus the system pictured in (b) has the potential for better overall efficiency.

To alleviate this integration effort, we have designed the Rad-hard Unified Scalable Heterogeneous (RUSH) architecture. RUSH is based on the insight that manycore architectures themselves actually solve a similar problem – the manycore architecture unifies a heterogeneous community of resources including tiles, caches, DRAMs, and I/Os. The mechanism that Maestro uses to integrate these clients is the iMesh interconnect. RUSH extends this iMesh model across the chip boundary and onto the FPGA to allow for resources on both Maestro and the FPGA to be treated as clients on a single virtual interconnect.

This paper proceeds as follows. In Section 2 we describe the details of the RUSH architecture and the technical approach. In Section 3 we describe the anticipated challenges of implementing RUSH. In Section 4 we present preliminary results in the context of the aforementioned SDR application. In Section 5 we discuss the body of previous research related to this work. Finally, in Section 6, we conclude.

2. Architecture

RUSH was designed for space applications and thus targets two rad-hard processing elements: Maestro, a 49-core rad-hard by design (RHBD) processor [13] and Xilinx's Virtex-5QV [21], a radiation hardened Virtex-5. Maestro and the FPGA will communicate over the XAUIs on each end. The current generation Maestro part has four half-duplex XAUIs that can be configured as two full-duplex interfaces. On the FPGA side, Xilinx provides a XAUI core for its FPGAs [22].

2.1. Extending the interconnect model to the FPGA

In order to unify the Maestro and FPGA chips, RUSH extends the manycore-style interconnect onto the FPGA as shown in Figure 3. IP blocks on the FPGA are treated as virtual tiles that can send and receive dynamic messages just as a physical tile would on the Maestro side. Since cache misses and other factors can cause timing to vary on

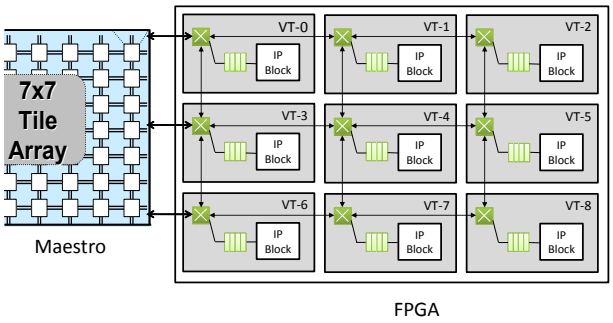

Figure 3. Extension of the Manycore Topology. RUSH extends the manycore grid geometry onto the FPGA, where IP blocks are arranged as virtual tiles.

Maestro, the system needs to be designed to be tolerant to variations in exact message arrival times. The FPGA logic that interfaces to the network must be coded to be flow-controlled and latency insensitive so as to decouple the exact timing of the two chips. To support this functionality, RUSH provides blocks for lightweight virtual tile implementation, including a router and FIFO.

The virtual tile is configurable to support three communication idioms based on three data constructs. We define a *sample* as a unit of data that can be accessed atomically off of the network without special synchronization. A *frame* is a set of samples. *Random access* is an arbitrary access pattern to an array of data. From these constructs, we can then list the three communication idioms:

1. Sample-driven – computation is synchronized to the arrival of a new sample (e.g. Viterbi decoder)

2. Frame-driven – computation is synchronized to the arrival of a new frame to ensure that all of the necessary samples have arrived before the circuit logic attempts to process the frame (e.g. FFT)

3. Random Access – computation requires generalized access to data (e.g. graph-based algorithms)

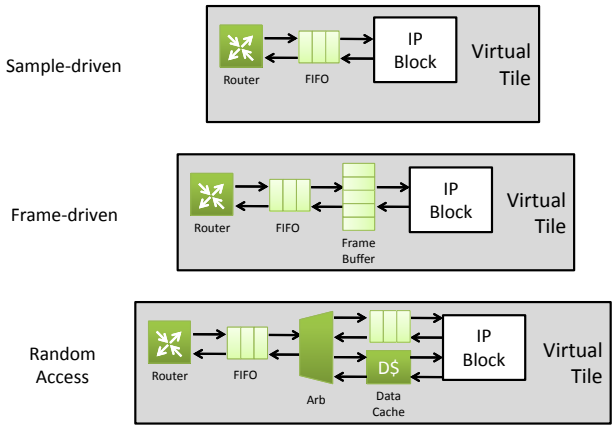

Figure 4. The RUSH Virtual Tile Communication Idioms. Virtual tiles can be configured to support the idiom of communication that drives the signal processing block. Sample-driven blocks attach directly to the network router, which may have some internal buffering to reduce congestion. Frame-driven blocks need a frame buffer to accumulate a full frame before beginning computation. Random access blocks need a data cache controller with access to the network router for memory accesses as well as a general purpose interface to the router for all other network messages.

The first two idioms (sample- and frame-driven) are likely to be most common for streaming-style signal processing workloads. The Random Access idiom then provides a more generalized idiom for algorithms that do not fit the sample- or frame-driven models. For sample-driven algorithms, the virtual tile needs minimal buffering and a network router. For frame-driven algorithms, the virtual tile needs at least a frame of buffering and a network router. For random access algorithms, the virtual tile needs an interface to the memory system such as a data cache controller. These configurations are shown in Figure 4.

Using these mechanisms, a virtual tile then has access to resources on the Maestro chip. To send data to a physical Maestro tile, the virtual tile can place the appropriate header on the network and begin sending payload words. The header and payload will be sent over XAUI to the Maestro chip, where the message will be routed to the appropriate destination using one of the dynamic networks. This allows the virtual tile to communicate with a physical tile using a range of well known protocols such as message passing, DMA, or shared memory. Since Maestro provides cache-coherent shared memory, RUSH can add a virtual tile to the memory system configuration in the Maestro Translation Lookaside Buffers (TLBs) to extend the cache coherence to the FPGA. This would allow a random access style algorithm to keep a local data cache that is coherent with the shared memory system, effectively unifying the main memory across both chips.

2.2. The RUSH software layer

Our vision for the RUSH development experience is grounded in familiar existing frameworks. Maestro-side development should continue to use the gcc- and eclipse-based toolchain while FPGA designs will continue to be developed in vendor supplied toolchains. However, there are three categories of software modules that will be required for rapid adoption of the RUSH approach.

Maestro-side libraries. On the manycore side, there are several modules that will be needed to allow the user to access virtual tiles on the FPGA side. An interface module will manage all cross-chip communication to ensure that I/O packets are routed to the correct physical network. Also, to support cross-chip cache-coherent shared memory, virtual tiles on the FPGA side will need to be recognized by the Maestro-side cache coherence system. This accounting is recorded in the TLBs, and thus a RUSH module will need to be added to the hypervisor to help manage cache coherence.

FPGA-side libraries. A virtual tile on the FPGA is built from an IP block that defines the processing kernel and a lightweight interface to the physical network. This interface at a minimum is a network router, but might also include buffering for frame-driven computation and a cache controller for random access computation. All of these modules should be provided as Verilog or VHDL designs as part of the RUSH libraries on the FPGA side.

Integration tools. Keeping track of design details across the two chips is challenging. Maestro source code is compiled by gcc-based tools while FPGA bit streams are generated with vendor-supplied CAD tools. Source files within the differing flows must access common specifications, such as interfaces and virtual tile addresses. However, this can be managed largely by automatic tools. For example, an interface generator can automatically generate C header files and Verilog module interface definitions based on a list of the names of existing processing blocks in the system. This decouples physical placement of blocks from source code addressing of those blocks, making code easier to read and reducing code duplication. Moreover, co-simulation of the two systems will be required, and the necessary mechanisms for integrating disparate simulation environments should be provided.

3. Challenges

As we move forward with the implementation of the RUSH system, there are a number of challenges that will need to be addressed. Here we categorize and discuss these as engineering research problems.

3.1 Power efficiency

RUSH adds clients on both the Maestro and FPGA sides for connecting the chips. Moreover, on the FPGA side, virtual tiles are implemented by adding a router and buffering to the IP block. These are the main contributors to RUSH overhead. Of course, the promise of RUSH is that its architecture will improve overall energy efficiency. An open research question is how to reduce RUSH overhead and thus maximize efficiency gains.

3.2 Test and verification

During prototyping, RUSH hardware may not yet be available. In this case, the developer will need co-simulation between the Maestro and FPGA processing for test and verification. This will require a software testbed to connect the XAUIs in an efficient way that is transparent to the programmer.

At the same time, RUSH provides a useful mechanism for debugging systems before all components are fully developed. Consider a system with a subset of signal processing blocks targeting Maestro and another subset targeting the FPGA. If the FPGA blocks are not fully verified, the developer could instead run a C model of those blocks on Maestro tiles. Since the RUSH communication model is uniform across both Maestro and the FPGA, we can then quickly test the full system with the C models instead of the FPGA blocks. C models are typically cheaper to develop than FPGA blocks. Moreover, the C models could be designed to be mock ups without any real processing in order to test the data flow of the application. This model provides a great deal of flexibility in the debugging process.

3.3 Communication bandwidth

An open engineering question is the supportable bandwidth over the XAUIs between the chips. The RUSH clients on either end will need to be light weight to provide maximal streaming efficiency. Moreover, future generations of the chips will likely improve the XAUI capabilities – or provide new interfaces that support higher rates. In either case, the RUSH clients should be scalable.

3.4 Extending the memory system to the FPGA

An interesting RUSH capability is the extension of the shared memory system to the FPGA. A data cache on a virtual tile on the FPGA should be configurable to be cache-coherent with the caches on the Maestro side. Currently, cache coherence on Maestro is managed in the TLB entries. Thus, any revision of this design will likely require modifications to the hypervisor code running on Maestro so that the memory system recognizes cache clients on virtual tiles on the FPGA.

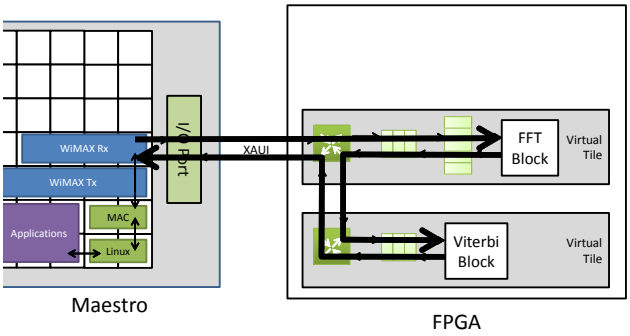

Figure 5. Software Defined Radio on RUSH. A software-defined radio transceiver might use RUSH to offload key kernels of the receiver, such as FFT and Viterbi decoding, to an FPGA, as shown here.

4. Results

In this section we present preliminary results. To date we have focused on an SDR application: a software-defined WiMAX physical layer implementation partitioned across Maestro and an FPGA, shown in Figure 5. On the receive side, the Viterbi decoder and FFT are the computational hotspots. Thus, we might want to offload these blocks to the FPGA. Assume that the signal generation begins on Maestro and is streamed to the FPGA. If we stream to the virtual tile containing the FFT block, the virtual tile can use its buffering to accumulate a frame of data, compute, and then send a packet with the frame payload to the Viterbi decoder virtual tile. This tile can read one sample at a time using its network router. It can then send output onto the network back to the Maestro chip. On the Maestro side, this physical communication might use the UDN, but this detail can potentially be abstracted from the IP block designer using the RUSH blocks.

First, we look at the tile allocations on Maestro based on our own benchmarking. Then, we estimate resource consumption of FFT and the Viterbi decoder on the rad-hard Virtex-5QV part based on vendor-supplied data IP block data sheets.

4.1. Software Defined Radio on Maestro

We set our study in the context of a WiMAX [1] physical layer prototype. WiMAX is a long range wireless standard for fixed and mobile devices with a large set of applications.

In our study, we looked at a frame size of 2048, which is the largest and most computationally intensive for WiMAX – providing a data rate of about 40 Mbit/s for QPSK modulation at 11k frames/s. We use a rate 1/2, K=7 convolutional encoder and corresponding Viterbi decoder to comply with the WiMAX standard. We determined through algorithm complexity analysis that the most computationally demanding signal processing kernels are the FFT and Viterbi blocks. The FFT and Viterbi blocks are used in the

WiMAX receive chain. We used the performance results measured for FFT and Viterbi, and, given our estimated performance after serial optimization, estimated the number of tiles needed to meet the 11k frames/sec requirement based on near-linear speedup. This estimation is shown as part of the floorplan in Figure 2(a). We allocated 22 tiles for the Viterbi decoder and 4 tiles for the 2048-pt FFT.

Note that 10 tiles remain independent of the WiMAX radio and are used for application layer processing. This is an added benefit of a Maestro-based platform. Typically, applications are run on a host CPU that interfaces with the radio. Maestro provides an opportunity to bring the applications onto the radio, reducing size, weight, and power (SWaP) and the latency between the application layer and the lower layers.

Similarly, the MAC layer is co-located with the physical layer, allowing for extremely low latency between these layers. Inter-tile communication latency can be on the order of nanoseconds using the fast on-chip networks. In traditional SDRs, this MAC-PHY latency can be a problem unless the developer has access to the NIC hardware, which often uses proprietary drivers, a fixed ASIC, or both [14].

Moreover, Maestro can switch between protocols quite rapidly by jumping in the binary or context switching to a new binary using the linux operating system. On an FPGA, reconfiguration is slow. On Maestro, the SDR control would send out a message on the on-chip network to switch to a new point in the code (either as a function call or context switch). This could be quite fast – in the optimal case in the 10s or 100s of nanoseconds depending on the method and number of differing blocks between protocols.

Finally, the applications tiles could also be allocated to fault tolerance objectives. While the Maestro chip is rad-hard by design, we can also employ software-based fault tolerance for applications that require an added layer of reliability, availability, and serviceability (RAS) guarantees.

4.2. Viterbi Decoding and FFT on FPGAs

The Virtex-5QV has 20,480 slices [21]. We use slices as a metric for relative resource consumption. From the data sheets for the Xilinx-supplied IP blocks for a 2048-pt FFT [19] and a K=7, 1/2 rate Viterbi decoder [20], we benchmark these blocks at 322 and 726 Virtex-5 slices, respectively. These numbers are then used as approximations for Virtex-5QV slice counts.

4.3. Analysis

Table 2 summarizes our estimates from the previous subsections. Notice that the FFT and Viterbi blocks consume a significantly smaller percentage of the FPGA than they would Maestro. We estimate that the FFT consumes just 1.2% of the Virtex-5QV vs. 8.2% of the Maestro tiles. Similarly, we estimate that the Viterbi decoder would consume 3.5% of the Virtex-5QV vs. 44.9% of the Maestro tiles.

	Maestro		**Virtex-5QV**	
	Tiles	**% of Total**	**Slices**	**% of Total**
FFT	4	8.2%	322	1.2%
Viterbi	22	44.9%	726	3.5%

Table 2. Estimated Resource Consumption of Key WiMAX Blocks. The table compares resource consumption of key WiMAX kernels on Maestro and the rad-hard Virtex-5QV FPGA. The Maestro tile numbers come from our benchmarking results as part of prototyping a Maestro-based WiMAX software-defined transceiver. The Virtex-5QV numbers are based on the numbers of LUTs and flip flops reported in Xilinx's data sheets for the FFT IP block [19] and Viterbi decoder IP block [20]. These numbers are based on targeting the Virtex-5 (the terrestrial part) and are used here as an approximation. The table also shows the chip resource consumption as a percentage of the 49 tiles on Maestro and the 20,480 slices available on the Virtex-5QV.

Since power is correlated with the percentage of the chip that is active, we can expect that the power consumption of these blocks on the FPGA would be much less than on Maestro. Although data exists for these blocks on non rad-hard parts, the power estimates for the rad-hard parts are forthcoming.

5. Related Work

The introduction of Maestro as the first manycore processor for space has led to excitement in the space processing community about newly enabled opportunities. Recent publications have examined feature tracking [7], rover autonomy [5], and FFT and CAF [15]. Moreover, the fault tolerance challenges on multicore have been examined [18, 9]. Our work looks at adding an FPGA co-processor to Maestro within a unified architecture, creating an efficient heterogeneous system.

Such heterogeneous computing platforms are well studied. For example, Gillan et. al. looked at an atomic physics computation on two hybrid systems using an FPGA and a GPU as a co-processor, respectively [8]. Heterogeneous systems on a chip with FPGAs are commercially available [2] and more future looking designs have been proposed [6].

Heterogeneous computing tools are also a well populated area of research. CIGAR [11] is a tool and platform for using data-parallel co-processors alongside CPUs. OpenCL [10] is a standard for parallel programming of heterogeneous systems, particularly CPU-GPU systems.

6. Conclusion

We have presented a heterogeneous computing architecture that provides a manycore substrate and FPGA substrate under a unified programming model. By providing multiple substrates that each target a subset of the anticipated workload, the RUSH heterogeneous architecture can offer a diverse processing toolset to the application developer. Moreover, RUSH provides a unified programming model across chips. This ability to use the "right tool for the job" has the potential to improve power-performance efficiency for demanding space computing applications, which often require supporting a diverse set of algorithms and requirements. A future RUSH platform will target these applications while addressing the challenges discussed in this paper.

References

[1] "IEEE standard for local and metropolitan area networks part 16: Air interface for broadband wireless access systems." *IEEE Std 802.16-2009 (Revision of IEEE Std 802.16-2004)*, may. 2009.

[2] Actel. "Smartfusion intelligent mixed signal FPGA.", 2011. http://www.actel.com/products/SmartFusion/default.aspx.

[3] AMD. "Global provider of innovative graphics, processors and media solutions.", 2011. http://www.amd.com.

[4] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect." In *IEEE Solid-State Circuits Conference*, Feb. 2008.

[5] B. Borenstein, T. Estlin, B. Clement, and P. Springer. "Using a multicore processor for rover autonomous science." In *Aerospace Conference, 2011 IEEE*, 2011.

[6] E. Chung, P. Milder, J. Hoe, and K. Mai. "Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?" In *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, 2010.

[7] T. Gallgher, S. Weiss, and J. Hahn. "Natural feature tracking on the opera maestro platform." In *Aerospace Conference, 2011 IEEE*, 2011.

[8] C. Gillan, T. Steinke, J. Bock, S. Borchert, I. Spence, and N. Scott. "Programming challenges for the implementation of numerical quadrature in atomic physics on FPGA and GPU accelerators." In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, May 2010.

[9] K. Gostelow. "The design of a fault-tolerant, real-time, multi-core computer system." In *Aerospace Conference, 2011 IEEE*, 2011.

[10] K. O. W. Group. "The OpenCL Specification Version 1.0.", 2009. http://www.khronos.org/opencl.

[11] J. Kelm, I. Geladot, M. Murphy, N. Navarro, S. Lumetta, and W. mei Hwu. "CIGAR: Application partitioning for a cpu/coprocessor architecture." In *Parallel Architecture and Compilation Techniques, 2007. PACT 2007. 16th International Conference on*, 2007.

[12] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. "NVIDIA tesla: A unified graphics and computing architecture." *Micro, IEEE*, march-april 2008.

[13] M. Malone. "OPERA RHBD multi-core.", 2009. Presentation at Military and Aerospace Programmable Logic Devices (MAPLD).

[14] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste. "Enabling MAC protocol implementations on software-defined radios." In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, 2009.

[15] K. Singh, J. Walters, J. Hestness, J. Suh, C. Rogers, and S. Crago. "FFTW and complex ambiguity function performance on the maestro processor." In *Aerospace Conference, 2011 IEEE*, 2011.

[16] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. "The raw microprocessor: A computational fabric for software circuits and general-purpose programs." *IEEE Micro*, 2002.

[17] Tilera. "Tilera has solved the multi-processor scalability problem.", 2011. http://www.tilera.com.

[18] C. Vallalpando, D. Rennels, R. Some, and M. Cabanas-Holmen. "Reliable Multicore Processors for NASA Space Missions." In *Aerospace Conference, 2011 IEEE*, 2011.

[19] Xilinx. "LogiCORE IP Fast Fourier Transform v7.1.", 2011. http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf.

[20] Xilinx. "LogiCORE IP Viterbi Decoder v7.0.", 2011. http://www.xilinx.com/support/documentation/ip_documentation/viterbi_ds247.pdf.

[21] Xilinx. "Space-grade Virtex-5QV FPGAs.", 2011. http://www.xilinx.com/products/virtex5qv/index.htm.

[22] Xilinx. "Xilinx : XAUI.", 2011. http://www.xilinx.com/products/design_resources/conn_central/protocols/xaui.htm.