View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DSpace at Rice University

Modelling Heterogeneous DSP–FPGA Based System Partitioning with Extensions to the Spinach Simulation Environment

Michael Brogioli
School of Electrical and
Computer Engineering
Rice University
Houston, Texas 77005
Email: brogioli@ece.rice.edu

Dr. Joseph Cavallaro
School of Electrical and
Computer Engineering
Rice University
Houston, Texas 77005
Email: cavallar@ece.rice.edu

***Abstract*— In this paper we present system-on-a-chip extensions to the Spinach simulation environment for rapidly prototyping heterogeneous DSP/FPGA based architectures, specifically in the embedded domain. This infrastructure has been successfully used to model systems varying from multiprocessor gigabit ethernet controllers to Texas Instruments C6x series DSP based systems with tightly coupled FPGA based coprocessors for computational offloading. As an illustrative example of this toolsets functionality, we investigate workload partitioning in heterogeneous DSP/FPGA based embedded environments. Specifically, we focus on computational offloading of matrix multiplication kernels across DSP/FPGA based embedded architectures.**

I. Introduction

In this paper, system-on-a-chip extensions to the Spinach simulation infrastructure for rapidly prototyping heterogeneous and reconfigurable FPGA based architectures for embedded computing are presented. This infrastructure has been successfully used to model systems varying from multiprocessor ethernet controllers to Texas Instruments C6x series DSP based systems with tightly coupled FPGA based coprocessors for computational offloading. As an illustrative example of this toolsets functionality, workload partitioning in heterogeneous DSP-FPGA based embedded environments is investigated. Specifically, matrix multiplication kernels of various sizes are implemented using various system configurations of DSP and integrated FPGA based compute blocks.

There are two main contributions of this paper. The first is the development of DSP and FPGA based reconfigurable computing elements to the Spinach simulation environment. With these extensions, users can model multiprocessor heterogeneous DSP based system topologies, incorporating FPGA based reconfigurable computing elements into their overall system designs. These architectures can now be programmed using Texas Instruments Code Composer Studio in much the same manner as one would develop on the real hardware, running production binaries in a bit true and cycle accurate manner. The second contribution is an exploration of partitioning embedded applications in the presence of a heterogeneous computing environment with DSPS AND reconfigurable FPGA based coprocessing elements. Tradeoffs in performance and system resource utilization are presented as an illustrative example of the toolsets functionality.

II. Background

Simulation has been an established design methodology in architectural exploration for a number of years. By modelling common system components at a high level in software, users can typically prototype system configurations much more rapidly than if this were done at the gate level [2], [3], [4], [8], [9]. Limitations in rapidly prototyping heterogeneous systems typically stem from the fact that the simulated hardware is not well abstracted in software. In modelling heterogeneous ISAs and multiprocessor systems, there is not a one-to-one mapping between system hardware and simulator software modules, nor are there well defined interfaces between the various software modules to accurately model componenets such as data busses, memory arbitration engines, etc. This typically results in an inherent lack of coupling between system timing and data flow, and precludes accurately modelling real time system demands and deadlines.

There are a number of commercially and academically available toolsets for modelling heterogeneous hardware at various levels. Early systems such as SOS from the University of Southern California primarily targeted synthesis of heterogeneous multiprocessor systems based on task level partitioning of a selected input application [6], [10]. Similar to this was the PICO-NPA system from Hewlett Packard, which again was a program-in, chip-out type system [7]. While these tools provide a general solution to the problem at the computational bottleneck level, they fail to provide insight into the overall system component interactions and synergy between application software and application specific hardware at a system wide level.

Commercially available solutions have become readily available in recent years, such as the Seamless system from Mentor Graphics [5]. Seamless provides a hardware/software co-verification environment for multiprocessor heterogeneous

1-4244-0132-1/05/$20.00 ©2005 IEEE

Authorized licensed use limited to: Rice University. Downloaded on June 18, 2009 at 13:22 from IEEE Xplore. Restrictions apply.

environments with custom processing units such as FPGAs. The overall goal is to provide the end user with a toolset of building blocks to prototype a given embedded system. While these tools are useful in the industrial setting, their closed source nature tends to limit the amount of architectural exploration or *under-the-hood* type research that can be performed with them. Modifying processor architecture and bus architectures is precluded, as is the ability for the user to roll their own system components into the overall system topology.

In building an environment for rapidly prototyping heterogeneous embedded architectures, there are a number of features the toolset should support. The library of building blocks should include common elements in the embedded domain. In using these building blocks for system prototyping, multiple clock domains as well as multiple concurrent ISAs should be supported as is commonly the case in today's SOCs. Finally, as a toolset, the user must be able to rapidly prototype a given system topology. Prototyping a system should rapid enough to facilitate an iterative hardware/software design process where synergy exists between input application proposed hardware topology. Input applications should be compiled using production compilers, facilitating a similar experience to programming real hardware, while the entire system remains open source and user extensible amongst various collaborators.

III. Simulation Infrastructure

The Spinach SOC simulation environment is a simulator design tool rather than a simulator for a specific system [1]. Spinach provides the user with a library of composable, user configurable software modules that can be used to build simulators for various embedded architectures. In using Spinach, users create instances of various supplied software modules, and connect them together via well abstracted port mechanisms to simulate a given hardware topology. This provides the simulator designer with an intuitive framework to work within, where there exists a one–to–one mapping between simulator software modules and hardware components in the actual or proposed system. In doing this, system designers deal not with massive amounts of source code, but rather high level module constructs that behave like real hardware components. Furthermore, modelling vastly different system topologies is as simple as reusing existing modules without modifying underlying source code. As an example, going to a multiprocessor DSP based system simply entails creating multiple instances of the DSP module, and connecting the fetch and write-back stages to the appropriate memory hierarchy input ports. While this paper presents extensions to the Spinach simulation environment for modelling DSP and FPGA based computational elements for full system-on-a-chip simulation, further details on the user interfaces and models of computation used within the simulator can be found in [1].

Table I lists the various types of modules included with the SOC extensions to the Spinach simulation environment. Processing modules include bit-true, cycle accurate models of the Texas Instruments C6x series fixed point Digital Signal Processor cores. Specifically the C60x and C62x series, with support for the C64x series pending. Also included are models of the MIPS R4000 embedded microcontrollers. FPGA based compute engines supporting both fine grained and coarse grained functionality. Fine grained reconfigurable FPGA modules have been used to perform compiler controlled custom ISA extensibility at runtime, whereas the coarse grained FPGA models are used to simulate Xilinx style fabrics similar to the Virtex-II and Virtex-IV series devices [11]. FPGA modules include customizable dual ported RAM array sizes, as well as user programmable logic. Timing information for FPGA compute latencies is typically collected externally via real hardware and plugged into simulation timing statistics for overall SOC performance gathering. Communication with FPGA modules within the simulated system is typically done via memory mapped registers in the simulated address space for synchronization and control. Data transfers to and from local FPGA RAM arrays are typically done via burst mode direct memory access transfers or programmed I/O via simulated bus interfaces between modules. FPGA modules are implemented in low level C code, rather than at VHDL or RTL level, which is a limitation of the Liberty Simulation Environment upon which the simulator is built, this permits increased simulation speed for the overall SOC model [1], [8]. We feel this is a fair trade off given the fact that in these types of systems, the overall system component interaction and system performance is of more importance.

TABLE I

Simulator Module Library

Module Type	*Description*
Processing Elements	Texas Instruments C6x series DSPs
	MIPS R4000 Microcontrollers
	Fine Grain FPGA based reconfigurable ALUs
	Coarse grain FPGA based accelerators
Reconfigurable Blocks	Fine Grain FPGA (on-chip for ISA extensions)
	Coarse Grain FPGA (off chip computation offloading)
Memory System	Variable latency SRAM and DRAM modules
	Memory controllers, scratpad SRAMs
	Caches and cache controllers
Interconnect	Memory arbitration units
	Master-slave on-chip crossbar connects, stackable
	Flexible bandwidth bus modules
	Application programmable DMA engines
Compilers/Utilities	Texas Instruments Code Composer Integration
	MIPS GCC Integration
	Runtime profiler, simulation checkpointing

A brief list of major memory system components are listed in Table I, while a full description can be found in Brogioli et. al [1]. Variable latency SRAM and DRAM modules are included, as well as flexible user programmable memory controllers. Instruction and data caches and cache controller modules with flexible storage policies, as well as local fast SRAM based scratchpad memory modules. All memory modules are user configurable and extensible in terms of I/O bandwidth,

Authorized licensed use limited to: Rice University. Downloaded on June 18, 2009 at 13:22 from IEEE Xplore. Restrictions apply.

storage size, clock rate, arbitration and replacement policies. Additionally, all such configurability is user extensible and modifyable as well.

System interconnect functionality such as crossbar and arbitration logic is modelled with flexible memory arbitration engines for variable I/O bus bandwidths. Crossbars for SOC component interconnect are included, which are stackable in order to support a master, slave, and slave-grouping structure of interconnect. Flexible bandwidth bus modules are used to connect modules via their port interfaces, and model overall system interconnect. High speed data transfers within the simulated system are performed by the application software programmable DMA enginees. Finally, a user extensible I/O harness is included for simulating traffic coming off of an external medium asynchronously.

All modules are customizable via scripting level parameters by the user. Typical parameters included on modules are: clock rate, bus bandwidth, local data memory size (FPGA RAM arrays), arbitration policys, replacement policies, debugging information, module state logging and checkpointing etc. This allows the system user to ignore the internal details and source code implementation of a given block in the system, and strictly prototype at the script level.

Figure 1 shows how the simulator modules listed in Table I can be used to prototype a given SOC architecture. The topology includes Texas Instruments C6x based DSP processing elements with supporting instruction and data memory busses, and instruction and data memory controllers. Access from 256 bit on chip data busses to external off chip busses must pass through a 256-32 bit bus bridge and through various crossbar and arbitration logic to the appropriate devices. DMA engines and memory mapped control registers are modelled off chip and are accessed via read-write operations from the DSP. Similarly, FPGA engines existing off-chip are modelled explicitly as well, incorporating both local memory mapped registers for control and synchronization, as well as local dual ported RAM arrays for local compute data storage and logic blocks for given functionality. The proposed system topology in Figure 1 was one of many systems modelled in arriving at a heterogeneous DSP/FPGA based SOC architecture for achieving and exceeding 3.5G wireless receiver data rates.

In the next section, we investigate using the toolset for prototyping simple DSP/FPGA based heterogeneous SOCs for matrix multiplication kernels. The intent is not to show the limits of the simulation infrastructure, as much as to show a tractable illustrative example of using the toolset, and the types of statistics and level of modelling detail available to the user.

IV. Case Study: Matrix Multiplication Offloading

As an illustrative example of this toolsets functionality, matrix multiplication kernels are considered. Typically these workloads, specifically dot product computation within matrix multiplication, exhibit larger amounts of instruction and data level parallelism than can be tackled by standard VLIW pipeline widths. The proposed idea is to use coarse grained FPGA based compute engines to exploit large levels of

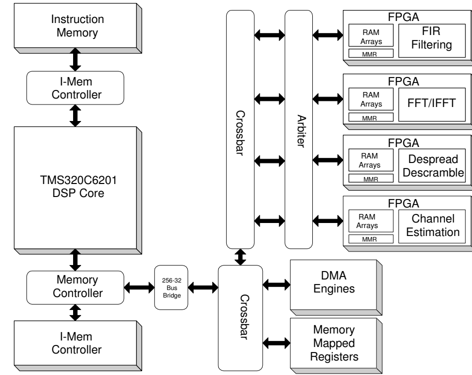


Fig. 1. Simulator Instance

instruction level parallelism, thereby achieving performance enhancements over typical VLIW style architectures. The entire system is modelled, including DSP, FPGA compute engines, DMA engines and bus arbitration and bandwith. Dot product computation in the matrix multiply kernel is offloaded to various FPGA configurations, and synchronization between host DSP and FPGA is the explicit responsibility of the application programmer. The DSP must pole memory mapped registers on the FPGAs and DMA engines to control burst data transfers of input vectors to the FPGA's local RAM arrays for computation. Figure 2 shows the simulated system topology, whereby all data traffic must explicitly occur over either the 256 bit on-chip or 32-bit off chip busses, and be subject to contention with other devices on said shared busses.

The details of the modelled system are as follows: DSP cores are 167 MHz Texas Instruments C62x series fixed point processors, 64KB of on-chip instruction and data memory, all accesses must pass through the instruction and memory controllers on-chip via the 256-bit on-chip busses. FPGA based coprocessors are application software controlled explicitly by the programmer via load/store references to memory mapped registers. Data computed on locally by FPGA must be explicitly moved from on-chip instruction memory to local dual ported RAM arrays on FPGA either via programmed I/O by the DSP, or via software controlled burst tranfers from the off-chip DMA engines. All matrix multiply computation is performed at 16-bit fixed point resolution, with all array sizes and offsets known statically at compile time with aggressive loop unrolling performed. In doing this, the optimistic case for DSP performance is modelled due to the nature of the Texas Instruments C62x pipeline functional unit resources. Specifically the C62x pipeline has four adders and two multipliers, each of which are 32-bits wide. When performing 16-bit fixed point computation, however, the ALUs in the DSP pipeline can be used to perform two computations in parallel. This effectively provides eight adders and four multipliers for computation each clock cycle.

Table II shows the parameters used for the various FPGA implementations of dot product simulated. Two implementations of FPGA based dot product computation were modelled:

Authorized licensed use limited to: Rice University. Downloaded on June 18, 2009 at 13:22 from IEEE Xplore. Restrictions apply.

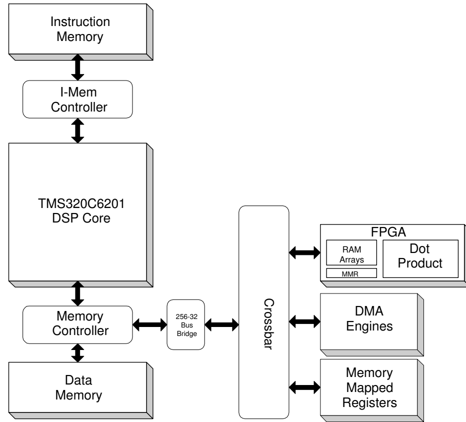


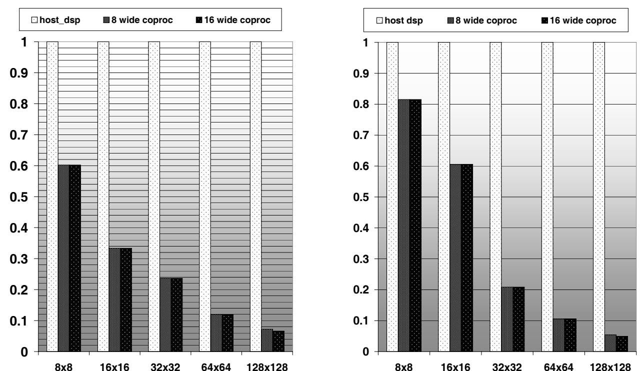
Fig. 2. Matrix Multiplication SOC Diagram

TABLE II
FPGA BASED DOT PRODUCT COMPUTE LATENCY

Dot Product Size	*8 Wide Coproc Latency*	*16 Wide Coproc Latency*
8	2 clock cycles	2 clock cycles
16	3 clock cycles	2 clock cycles
32	6 clock cycles	3 clock cycles
64	11 clock cycles	6 clock cycles
128	21 clock cycles	11 clock cycles

one using an 8-wide parallel multiplier array followed by a 4 level deep adder tree, the other a 16-wide parallel multiplier array followed by a 5 level deep adder tree. The first column in Table II shows the different matrix sizes modelled, specifically 8x8, 16x16, 32x32, 64x64, and 128x128. For each matrix multiplication problem size, the corresponding dot product size is listed in the table showing the computational latency given either 8 or 16 parallel multipliers and corresponding adder trees. It is important to note that these are strictly the number of clock cycles to perform the computation once all data resides locally in FPGA RAM arrays. It is still the application programmers responsibility to initiate the DMA engine data transfers to and from the local RAM arrays on FPGA before computation can begin, as well as manage any synchronization between DSP and FPGA via the memory mapped registers in the address space.

Figure 3 shows the normalized runtimes and instruction fetch bandwidth utilization for each input matrix multiplication problem size. For each matrix size, we simulate the entire matrix multiplication kernel executing only on the host DSP, as well as DSP using the 8 wide or 16 wide FPGA coprocessor for dot product computation. Looking at Subfigure 3(a), performance improvements greater than 90% can be achieved for the 128x128 matrix problem size. With the smaller 8x8 problem size, modest improvements are gained on the order of 40% improvement, due to data transfers to local RAM arrays on the FPGA must be performed before FPGA based dot product computation can occur. With much smaller vector sizes to DMA back and forth, the offset incurred by the DSP in programming DMA transfers via the off-chip memory mapped registers is significant when compared to overall DMA data transfer time. For the larger vector sizes in the 128x128 case, the penalty of DMA block setup is minimal compared to the total data transfer time. Looking at Subfigure 3(b), similar data trends in the instruction fetch bandwidth can be seen. Here, the number of instruction fetch bundles requested and transferred from on-chip instruction memory to the DSP fetch stage via the on-chip instruction memory controller is shown. When significant computation is offloaded to the FPGA, the amount of instruction fetch decreases significantly, reducing bus contention in the event of shared resources on a single bus. While the program runtime is shorter thereby reducing the number of fetches, the following is important to note: tThe tight polling loop used for synchronization and control within the system via memory mapped registers actually is split across two instruction fetch bundles. This results in fetches occurring during synchronization wait time. This, however, was not apparently until the overall system was simulated. Other solutions thus become apparent: optimize the polling loop to use a single instruction fetch bundle. Furthermore, perhaps interrupt based synchronization is more efficient. Finally, perhaps single bundle sized SRAM based instruction caches can be used to alleviate instruction fetch bus activity during DSP idle time for synchronization loops.



(a) Normalized Program Runtime

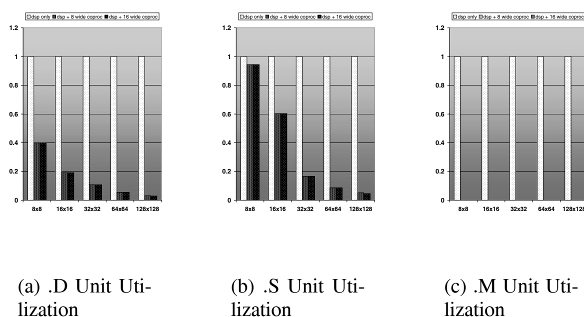
(b) Normalized Instruction Fetch Bundles

Fig. 3. Program Runtimes and Instruction Fetch Bundles

Similar insights into overall system performance can be seen by the fact that both Subfigures in Figure 3 show identical results using the 8-wide and 16-wide FPGA coprocessors until the 128x128 matrix multiplication kernel. That is to say, program runtime and instruction fetch performance is identical regardless of greater amounts of instruction level parallelism residing on the FPGA. This seems counterintuitive upon first glance until we consider the computational latencies of the various FPGA implementations of dot product. The

Authorized licensed use limited to: Rice University. Downloaded on June 18, 2009 at 13:22 from IEEE Xplore. Restrictions apply.

C62x architecture has a lengthy branch delay slot of 5 clock cycles. Looking at the compute latencies in Table II we see that it is not until the 128 element dot product that the difference in computational latency between the two FPGA implementations of dot product exceeds the 5 clock cycle branch delay slot of the host DSP architecture. It is the polling loop for synchronization between DMA transfer and FPGA computation that results in the branch delay slot overshadowing the difference in compute speed between the two FPGA implementations until the 128x128 element matrix. Without simulation at such a level of detail, it is counter intuitive details like this that the application developer and system architect often do not realize until too late in the production process.



(a) .D Unit Utilization (b) .S Unit Utilization (c) .M Unit Utilization

Fig. 4. DSP Functional Unit Utilization

By using an FPGA for the computationally intensive portions of the matrix multiply workload described above, we can see over 90% increase in runtime performance for the larger input problem sizes. These simulations show that despite the non-trivial overhead in configuring DMA based burst data transfers, and synchronization via memory mapped polling registers that whole system performance gains can be achieved. Similarly, we were able to reduce activity on the instruction memory bus by as much as 90% in the larger input problem sizes and were able to show that despite adding more parallelism to the FPGA based coprocessor, memory mapped synchronization methods preclude achieving performance gains due to architectural features of the DSP pipeline.

Other statistics gathered during simulation can be found in Figure 4 where the utilization of host DSP ALUs is shown. We can see that the .D unit, used for load-store operations, has significantly reduced utilization. Similarly the .S unit used for integer computation and control register file accesses is reduced dramatically as well. Finally, the .M unit used for multiplication drops to zero, as would be expected when all dot product computation is offloaded to the FPGA. While these results are as to be expected, it is interesting to note that in the presence of FPGA compute elements, it may be beneficial to utilize function unit gating in certain host DSP architectures, or to perhaps overlap computation amongst both the FPGA and DSP. Other statistics gathered and reported at simulation time, but not visualized in this paper for sake of brevity, are: bus utilization, arbitration policy and contention statistics, DMA engine utilization and activity, FPGA activity as well as compute latencies and dynamic data transfer latencies, instruction and data level parallelism tradeoffs in the input appliction, host processor dynamic ISA utilization and dynamic runtime profiling of the input application. All of these statistics are modelled at the granularity of clock cycles and reported to the user.

V. CONCLUSIONS

This paper presents extensions to the Spinach simulation environment for modelling heterogeneous DSP/FPGA based reconfigurable embedded architectures. Specifically, support for bit true, cycle accurate DSP cores, fine grained and coarse grained FPGA based compute elements is presented. As an illustrative case study, the toolset is used to investigate the benefits of computational offloading various matrix multiplication kernels in heterogeneous DSP/FPGA based system on a chip architectures. In doing this, performance gains of approximately 90% are shown to be achieved using both coarse grained FPGA compute engines in tandem with modern DSP cores. Additionally, insights into total system on a chip performance are presented that are only apparent when modelling no only computational elements in the system, but overall chip interconnect and memory system.

ACKNOWLEDGMENT

This work was supported in part by Nokia Inc., Texas Instruments, Inc., and NSF under grants EIA-0224458 and EIA-0321266.

REFERENCES

[1] M. Brogioli, P. Willmann, and V. Pai. Spinach: A Liberty-Based Simulator for Programmable Network Interface Architectures. In *Languages Compilers and Tools for Embedded Systems 2004*, pages 100–110, 2004.

[2] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A Framework for Simulating and Prototyping Heterogenous Systems. *Int. Journal in Computer Simulation*, 4(2), 1994.

[3] D. Burger and T. M. Austin. The Simplescalar Tool Set, Version 2.0. *SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[4] C. J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve. RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors. *IEEE Computer*, 35(2):40–49, February 2002.

[5] Mentor Graphics. Seamless Co–Verification, Texas Instruments Processor Support Packages.

[6] S. Prakash and A. Parker. SOS: Synthesis of Application-Specific Heteroeneous Multiprocessor Systems. In *Journal on Parallel and Distributed Computing, Volume 1*, pages 338–351, 1992.

[7] R. Schreiber, S. Aditya, B. Rau, V. Kathail, S. Mahlke, S. Abraham, and G. Snider. PICO-NPA: High-Level Synthesis of Nonprogrammable Hardware Accelerators, 2000.

[8] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August. Microarchitectural Exploration with Liberty. In *Proceedings of the 35th Annual International Symposium on Microarchitecture*, pages 271–282, November 2002.

[9] H. Wang, X. Zhu, L. Peh, and S. Malik. Orion: A Power-Performance Simulator for Interconnection Networks. In *Proceedings of the 35th Annual International Symposium on Microarchitecture*, pages 294–305, November 2002.

[10] W. Wolf. A Decade of Hardware/Software Codesign. In *IEEE Computer, Volume 36, No. 4*, pages 38–43, 2003.

[11] Xilinx Corporation Incorporated. Virtex–II Pro Platform FPGA Users Guide.

Authorized licensed use limited to: Rice University. Downloaded on June 18, 2009 at 13:22 from IEEE Xplore. Restrictions apply.