

# **A Low-Power Reconfigurable Data-flow Driven DSP System**

**Marlene Wan, Hui Zhang, Martin Benes  
Jan Rabaey**

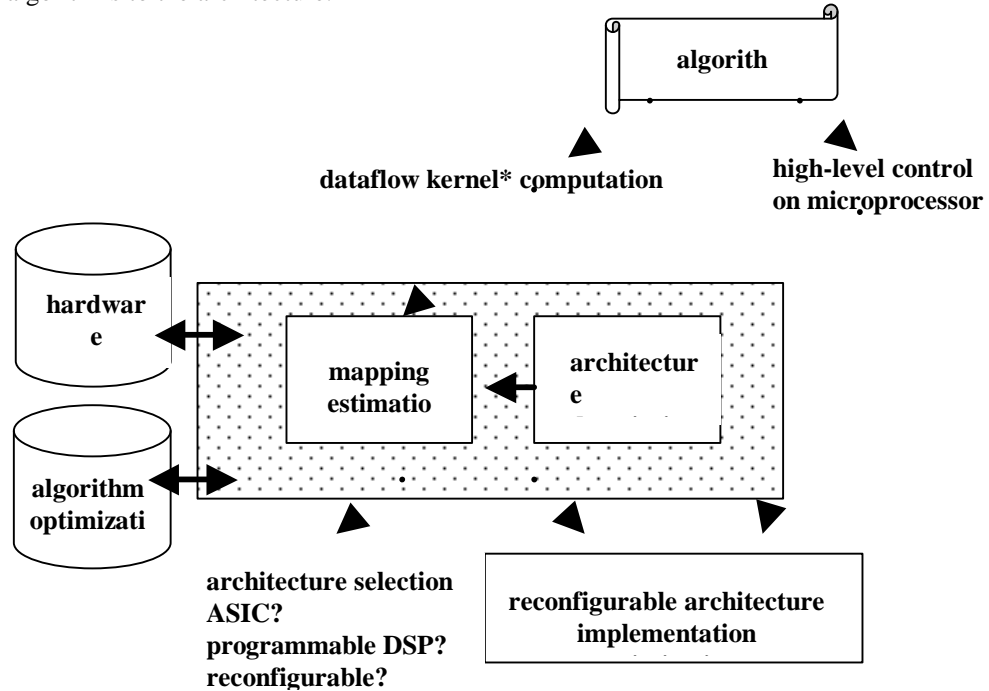
Berkeley Wireless Research Center  
EECS Department, University of California, Berkeley

**ABSTRACT** - Reconfigurable architectures have emerged as a promising implementation platform to provide high-flexibility, high-performance, and low-power solutions for future wireless embedded devices. We discuss in details a reconfigurable data-flow driven architecture, including the computation model, communication mechanism, and implementation. We also describe a set of software tools developed to perform automatic mapping from algorithms to the architecture, as well as to evaluate the resulting performance and energy of the mapping. Finally, we present results on digital signal processing and wireless communication algorithms to show the energy efficiency of the system and the effectiveness of the tools. Our system shows more than one order of magnitude of improvement in terms of energy efficiency when compared to low-power programmable processors.

## **MOTIVATION AND BACKGROUND**

Future wireless multimedia computing devices will be required to adapt their functionality to the changing parameters of the communication link available (e.g., bandwidth, error rates, protocols, etc.) to increase spectral efficiency. Therefore, these devices will have to be flexible enough to accommodate various multimedia services (e.g., different video compression schemes) and communication capabilities (e.g., cellular GSM, PCS and pico-cellular) while meeting the high-performance computational demand. At the same time, low-power consumption will continue to be the predominant design challenge of wireless systems. Reconfigurable architectures have emerged as a promising implementation platform to provide the flexibility, high performance [1] and low power [2] [3] required for future wireless embedded devices. In [4], a heterogeneous reconfigurable architecture template is proposed to meet all these requirements. The reconfigurable architecture template possesses both software programmability and hardware reconfigurability, and consists of a wide range of hardware modules such as embedded processors, arithmetic logic units, embedded memories, address generators and FPGAs. In this paper, we introduce a realization of such an architecture template (in particular, its model of computation and basic processing elements for data-flow computations) and supporting software to assist in direct implementations on such an architecture. The shaded box in Figure 1 shows the scope of this paper: the data-flow driven architecture model is described in detail, then tools to perform mapping and estimation are presented. Finally, the energy efficiency of the proposed realization

is demonstrated by mapping some wireless communication and signal processing algorithms to the architecture.



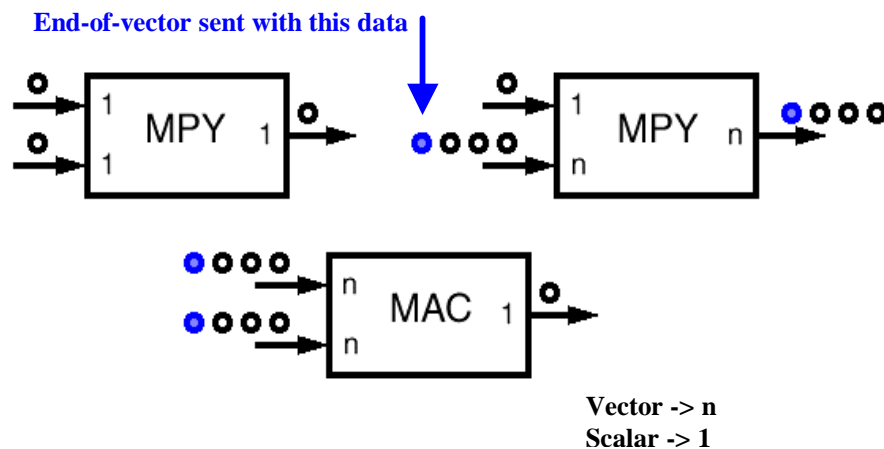
**Figure 1. Reconfigurable Digital Signal Processor Design Flow**

*\*Kernel:computational intensive operations within an algorithm, often correspond to data-flow computations in nested loops*

## **DATA-FLOW DRIVEN ARCHITECTURE DESCRIPTION**

In [4], the proposed architecture consisted of control-flow computation performed on the microprocessor and data-flow computation executed on the heterogeneous satellites. This architecture template fixes the communication scheme between each satellite as well as the interface method between the microprocessor and the satellite. To eliminate the energy overhead of satellite computations, communications between each satellite is data-flow driven and each satellite follows strict execution (i.e. operation starts only when all input data are ready). Reconfigurable interconnection network is used to establish dedicated links between satellites to preserve data correlation in signals, thus reducing energy consumption. In this paper, we will concentrate mainly on the architecture definition and software support of the reconfigurable data-flow driven satellites and satellite communications.

In our realization of the architecture, the data-flow driven satellites are medium to fine-grained according to the definition of [5]. The functionality of the satellites is divided into three categories: source, computation and memory. To support adaptive computations without reconfiguration such as changing the vector length or number of taps for the computation satellites, we have developed a minimum-overhead mechanism for passing data structures (scalar, vector and matrix). Each computation satellite needs to be configured for the data structures it consumes and produces (e.g., vectors to scalar for MAC, shown in Figure 2). The source satellites generate tokens indicating the end of the data structure in parallel with corresponding data.



**Figure 2 Data-flow Driven Operations of the Satellites**

In order to support dedicated links between satellites without reconfiguration overhead and global control, data steering elements are embedded in the reconfigurable network. In general, data steering elements are divided into three categories: static (data goes in a fixed direction in between reconfiguration periods); statically scheduled (data goes in directions instructed by programs configured at reconfiguration times); dynamically determined (data is annotated with the direction). Only the first two are supported by our realization of the architecture template because dynamic data steering imposes too large of an energy overhead for the granularity of the computational satellites.

Currently, the data-flow driven computation is implemented using global asynchronous and local synchronous clocking. A general handshaking scheme has been developed and a library of satellites has been designed using the scheme[6]. Address generators, input ports (with data from microprocessor) and FPGAs can serve as sources and are in charge of generating end of data structure tokens. Implementation issues for low-power reconfigurable interconnection networks are addressed in detail in [7].

## THE SOFTWARE TOOLS

In order to supply fast implementation feedback to the user, we have developed tools to support application specific simulation and direct-mapped synthesis from a high-level language to the satellites. To give effective implementation feedback, the tools utilize energy and performance models of the hardware components described in the previous section. In this section, we first give a short description of the performance models used. We then discuss the simulation and synthesis tool developed for our system.

### Performance Models

Power, delay, and area models have been developed for an extensive library of satellite modules designed at the University of California at Berkeley. Latency and analytical models of the effective switching capacitance ( $C_{eff}$ ) of the modules are derived from circuit level simulation. In this section, we will introduce only models used for energy since the characterization of area and timing is well understood. Since our models are used for high level architecture selection, some degree of inaccuracy is acceptable. Therefore, a white noise signal distribution is assumed instead of statistical signal modeling when obtaining  $C_{eff}$ .

$$Energy_{SAT} = C_{eff}V_{dd} \quad (Eq1)$$

Another important contributor of power consumption in our architecture is the reconfigurable interconnect. Methodology exists [7] to optimize domain specific reconfigurable interconnect architectures such that the energy and performance is close to ASIC implementations. Therefore, ASIC based interconnect power estimation is used for reconfigurable interconnect base cost--the average length (thus switching capacitance,  $C_{ave}$ ) between satellites is predicted based on the area of the modules needed for an application. A preliminary dynamic switching element has also been designed and the power model characterized [6]. The energy for a satellite-to-satellite link is therefore as follows:

$$Energy_{net} = C_{avg}V_{dd} + C_dV_{dd}M \quad (Eq2)$$

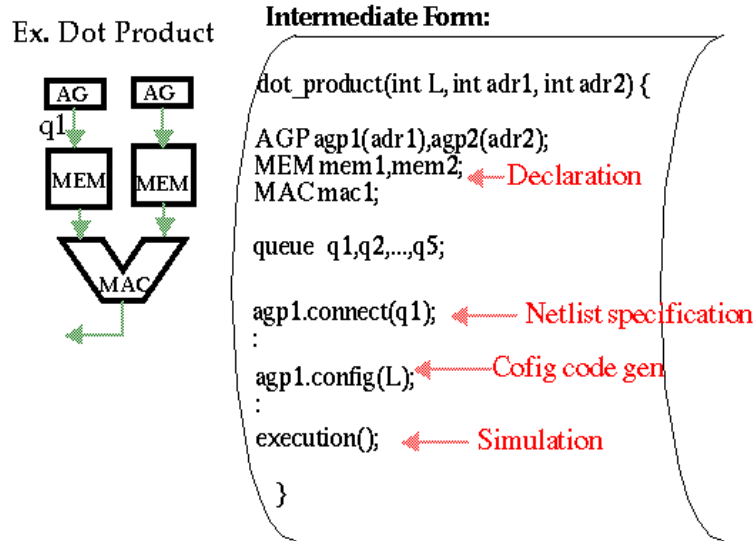
The parameter,  $M$ , specifies the number of dynamic switches required on the particular link which is known at synthesis time.

### Simulation Tool

Based on the realization of the architecture template, a simulation environment is developed to provide an application-specific simulator in a style similar to [8].

Since computation is mapped to clusters of satellites, an object-oriented intermediate form based on the concept of modules (heterogeneous satellites) and queues (links between satellites) is created. A mapped kernel is constructed by

building a netlist using the module and queue library (Figure 3). In order to facilitate verification and performance feedback, wrappers are placed around all modules and queues so modules can be modeled as concurrent processes and queues as synchronized objects. Energy and time stamps are also associated with each module and queue so performance data can be collected. An application specific simulator is automatically instantiated once a netlist is specified.



**Figure 3 An Intermediate Form Specification for a Computation Kernel**

Currently, the intermediate form is implemented in the C++ language and the Solaris thread library [9] (other common thread libraries can be switched in easily). Common satellite processors (such as MAC/multiply processor, ALU processor, memory and address generator, etc.) and data-steering modules have been incorporated in our module library.

### Synthesis Tool

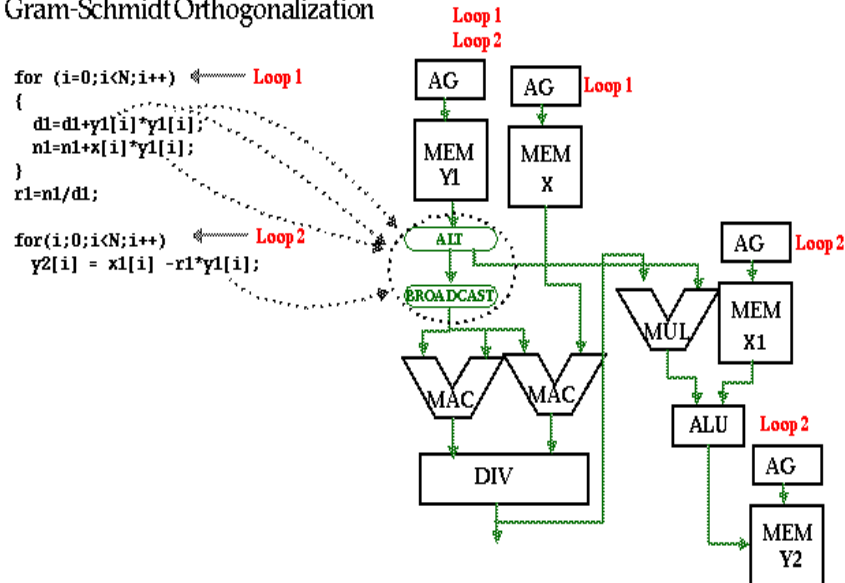
To ease the process of manually mapping algorithms to the architecture, we provide a synthesis tool to translate an algorithm (specified in a subset of C) to the direct-mapped implementation of the architecture. The output is the computation specified in the intermediate form. The kernel performance and energy can then be dynamically collected. In addition, for algorithms with nested loops of constant loop length, energy and performance information is also analyzed statically to avoid the overhead of simulation.

The algorithm is compiled to the Stanford Unified Intermediate Form (SUIF) then converted to hierarchical Control/Data Flow Graph (CDFG [10]) representation. The current conversion from SUIF to CDFG exposes all scalar dependencies and preserves all WAW, RAW, and WAR dependencies in array accesses. The current synthesis tool allocates arrays of the same name to a particular memory and each

operation node in CDFG to a hardware unit (Figure 4 shows an example of a mapped kernel). This assignment of operations gives the *rate-optimal execution* of each computational node in the CDFG graph.

The address generator program for each memory is generated based on the sequence of address expressions and corresponding loop iterations (an end of loop indicates an end of a vector). By merging corresponding fan-outs of each memory data read node and computational node in CDFG, a dedicated data steering element is generated for each output port. As shown in the example in Figure 4, while all other links are static, the output of memory Y1 (*y1* has 4 read nodes in the algorithm) is statically scheduled by a program. Data from memory Y1 has to be broadcast at first to the MAC satellites, but after an end-of-vector (corresponds to Loop1), the direction of the data is changed to the multiplier.

Ex: Gram-Schmidt Orthogonalization



**Figure 4 Direct Mapping from C to Data-flow Driven Implementation**

Static performance estimation for loops with constant loop length is also provided so the overhead of simulation can be avoided. For a hierarchical CDFG, the total energy can be computed by performing a tree search on the graph in  $O(E)$  time:

$$TotalEnergy = IterationNum \times \sum_{\forall comp} Energy_{comp}$$

In the equation, *comp* is either a satellite (Eq1), a link between satellites (Eq2) or another CDFG hierarchy.

Since the synthesis tool performs a direct mapping of CDFP to hardware, the latency of the implementation is characterized by the longest path and iteration period bound of the CDFG graph. The longest path is calculated by performing a topological sort of the CDFG graph ( $O(E)$ ) and the iteration period bound is calculated in  $O(VE \log E)$  [12][13].

$$\text{Latency} = \text{LongestPath} + (\text{IterationNumber} - 1) \times \text{IterationPeriodBound}$$

The output of the static performance analysis also includes energy and performance macro-models [14] with *IterationNum*'s as parameters. The users can then evaluate the performance of the algorithm with different loop lengths (for example, different filter orders in adaptive filters).

## CASE STUDIES

We show the low-energy feature of the system and effective performance feedback of the tools by using the performance information in several architecture selection processes. All energy and performance models of all satellite modules and interconnects are based on physical implementations designed in 0.25  $\mu\text{m}$  technology. Detailed reconfigurable interconnects are characterized in [7] also. The preliminary overhead of steering elements is included as well.

### Multuser Detection Channel Estimator

The first example is the mapping of an adaptive LMS filter for a multuser detection (MUD) channel estimator in direct sequence code division multiple access systems. The specification, documented in [15], provides a symbol rate of 1.67MHz and a spreading factor of 15.

The algorithm, specified in C, is synthesized to the architecture. Performance and energy data are determined statically and verified dynamically using the simulator. Table 1 shows the results of implementing the algorithm on the data-flow driven reconfigurable architecture along with two other implementations. The analysis [15] of the algorithm implemented on TMS320C54x is based on the same C algorithm compiled down to a low power version of the processor [16]. Based on the comparison offered by Table 1, users can select a specific architecture style based on the power/area/flexibility requirement.

Architecture	Power (mW)	Area (mm <sup>2</sup> )
TMS320C54x	460	1089
Data-flow Driven Satellites	<b>18.04</b>	<b>5.07</b>
ASIC	3 [15]	1.5

**Table 1 Different Architecture Implementation of LMS for MUD**

## VSELP Speech Coding

For more complex algorithms with both control-flow and data-flow computations, the data-flow computations often become the performance and energy bottleneck. Therefore, the data-flow driven satellites serve as a good accelerator to a microprocessor in the implementations of such systems. In this case study, we mapped all kernels in the VSELP [17] speech-coding algorithm to the architecture as an accelerator to a low-power embedded microprocessor (ARM8). The following is a list of kernels in the algorithm, discovered by the system compilation tool described in [18]:

*Dot\_product, FIR, IIR, VectorSumScalarMul, Compute\_Code.*

All of the kernels were synthesized and simulated, and performance information was gathered. Table 2 shows the energy utilized by each kernel to process 50 frames of voice data for the two different implementations. Given the same performance constraint, the data-flow driven architecture is able to run at much a lower voltage. Therefore, the data-flow driven architecture offers orders of magnitude of energy improvement for the kernels. The performance and energy information can then be passed up to a system level evaluation tool [18] to evaluate the effect of the architecture selection. In addition, the netlist generated by the synthesis step is passed down to the implementation step for further optimizations [7]. The final implementation of the VSELP algorithm (including ARM8 and satellites) is 1.11 mW (dropped from 37.29 mW, which is the power consumption of the ARM8 implementation).

	Energy on ARM8 (2.5V)	Energy on Data-Driven Reconfigurable Architecture (1V)
Dot_product	11550 $\mu$ J	153.7 $\mu$ J
FIR	5690 $\mu$ J	96.10 $\mu$ J
VectorSumScalarMul	4800 $\mu$ J	23.95 $\mu$ J
Compute_Code	1550 $\mu$ J	2.195 $\mu$ J
IIR	390 $\mu$ J	1.200 $\mu$ J

**Table 2 Comparisons of Two Architectures for VSELP Kernels**

## CONCLUSION

We have presented a low-power reconfigurable data-flow driven digital signal processing system, described architecture concept in detail, and shown the energy efficiency of the architecture in the case studies. The examples in the case study also

illustrate how the tools introduced in this paper allow rapid architecture selection and serve as the basis of future optimizations. Utilizing the ideas introduced in this paper, future work will include algorithm level transformations (loop transformation and parallelism), implementation optimizations as well as more application mappings in adaptive filtering for the wireless communication domain.

## ACKNOWLEDGEMENTS

The authors would like to acknowledge DARPA's support of the Pleiades project (DABT-63-96-C-0026) and all the Pleiades members for their input on the research topic discussed in this article.

## REFERENCES

- [1] G. R. Goslin, "A Guide to Using Field Programmable Gate Arrays for Application Specific Digital Signal Processing Performance", *Proceedings of SPIE*, vol. 2914, p321-331.
- [2] Abnous et al, "Evaluation of a Low-Power Reconfigurable DSP Architecture", *Proceedings of the Reconfigurable Architecture Workshop*, Orlando, Florida, USA, March 1998.
- [3] M. Goel and N. R. Shanbhag, "Low-Power Reconfigurable Signal Processing via Dynamic Algorithm Transformations (DAT)", *Proceedings of Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, November, 1998.
- [4] Arthur Abnous and Jan Rabaey, "Ultra-Low-Power Domain-Specific Multimedia Processors", *Proceedings of the IEEE VLSI Signal Processing Workshop*, San Francisco, California, USA, October 1996.
- [5] P. Lieverse, E.F. Deprettere, A.C.J. Kienhuis and E.A. de Kock, "A Clustering Approach to Explore Grain-sizes in the Definition of Weakly Programmable Processing Elements", In *1997 IEEE Workshop on Signal Processing Systems: Design and Implementation*, pp. 107-120, De Montfort University, Leicester, UK, November 3-5 1997.
- [6] Martin Benes, Master Thesis, University of California at Berkeley, 1999
- [7] H. Zhang, M. Wan, V. George, J. Rabaey, "Interconnect Architecture Exploration for Low Energy Reconfigurable Single-Chip DSPs", *Proceedings of the WVLSI*, Orlando, FL, USA, April 1999.
- [8] B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf, "An Approach for Quantitative Analysis of Application Specific Dataflow Architectures", In

*Proc. 11th Int. Conf. on Application-specific Systems, Architectures and Processors*, Zurich, Switzerland, July 14-16 1997.

- [9] SunSoft Press, "Solaris Multithreaded Programming Guide".
- [10] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures". *IEEE Design & Test of Computers*, vol.8, (no.2), June 1991. p.40-51.
- [11] D. Messerschmitt, "Breaking The Recursive Bottleneck", in *Performance Limits in Communication Theory and Practice*, Kluwer Academic Publishers, 1988.
- [12] Shan-Hsi Huang and Rabaey, J.M. "An Integrated framework for optimizing transformations", *Proceedings of VLSI Signal Processing IX*, p. 263-72.
- [13] C. Leiserson and F. Rose, "Optimizing Synchronous Circuitry by Retiming", *Third Caltech Conf. On VLSI*, March 1983.
- [14] D. Lidsky and J. Rabaey, "Early Power Exploration – a World Wide Web Application", *Proceedings of Design Automation Conference*, Las Vegas, NV, June 1996.
- [14] N. Zhang, "Implementation Issues in a Wideband Receiver Using Multiuser Detection", Master's Thesis, University of California at Berkeley, 1998.
- [16] W. Lee et. al "A 1-V Programmable DSP for Wireless Communications", *IEEE Journal of Solid-State Circuits*, Nov. 1997, vol.32, (no.11):1766-76.
- [17] Gerson and M. Jasiuk, "Vector Sum Excited Linear Prediction (VSELP) Speech Coding at 8Kbps", *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pp. 461-464, April 1990.
- [18] M. Wan, Yuji Ichikawa, David Lidsky, Jan Rabaey, "An Energy Conscious Methodology for Early Design Exploration of Heterogeneous DSPs" *Proceedings of the Custom Intergrated Circuit Conference*, Santa Clara, CA, USA, May 1998