An Overseer Control Methodology for Data Adaptable Embedded Systems

Sean Whitsitt
University of Arizona
1230 E. Speedway Blvd
Tucson, Arizona, USA
whitsitt@email.arizona.edu

Jonathan Sprinkle
University of Arizona
1230 E. Speedway Blvd
Tucson, Arizona, USA
sprinkle@ece.arizona.edu

Roman Lysecky
University of Arizona
1230 E. Speedway Blvd
Tucson, Arizona, USA
rlysecky@ece.arizona.edu

ABSTRACT

The performance of software algorithms can be improved by performing those algorithms on specialized embedded hardware. However, complex algorithms that rely on input data at runtime for configuration have a combinatorial explosion of possible configurations, which has historically put hardware acceleration out of reach for applications wishing to serve large configuration spaces. Data adaptable embedded systems overcome this limitation by allowing for hardware reconfiguration during runtime, but the complexity of the specification of these systems is difficult to manage with traditional techniques. In this paper, a modeling approach is discussed in order to concurrently model two aspects of the final system: dependencies between algorithm tasks, and desired hardware configurations for each task. The contribution of the work is the model-based generation of hardware and software tasks, as well as a control scheme customized to each model that oversees the dynamic reconfiguration process.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.2.6 [**Software Engineering**]: Programming Environments—*Integrated environments, Graphical environments*; D.3.2 [**Programming Languages**]: Language Classifications—*Multiparadigm languages*

General Terms

Design; Languages; Performance

1. INTRODUCTION

As the demand for more complex embedded devices grows, a problem arises. As it stands, embedded hardware is limited in its scope by the available hardware real-estate. Data-Adaptable Reconfigurable Embedded Systems (DARES) have been proposed to address this limitation. The purpose of DARESs are to reconfigure hardware at runtime in order to improve the performance of an algorithm while retaining the flexibility and configurability of software [14].

The DARES Modeling Language (DaresML) provides an environment in which a designer can structure an algorithm, and specify implementation parameters [6]. DaresML allows a user to construct a model for each of the tasks and the respective configurations for those tasks involved in his or her solution. Another model can then be created to represent the dataflow interconnections between each of these tasks. From these models all of the necessary code can be generated for the hardware profiles that represent the user's solution.

DaresML requires the designer to already have software and hardware implementations of each task configuration, but the language provides semantic structures to use while writing or porting that software. These semantic structures allow the user to build generic solutions to tasks that can then be interpreted by DaresML into the many different hardware and software configurations for those tasks.

The contribution of this paper is the DARES runtime reconfiguration process, which is partially fixed, and partially derived from the structure and content of the models built in DaresML. This is referred to in this paper as the *Overseer*. This paper extends previous work where the prototype hardware system was demonstrated [13] and the structure of the modeling language was described [6, 14].

2. BACKGROUND

Field Programmable Gate Arrays (FPGAs) are the cornerstone of reconfigurable computing [4], given their ability to perform various tasks in hardware, and to store hardware configurations as files. As such, FPGAs are uniquely suited to algorithm acceleration, since they can perform specific tasks in hardware, and can also include an on-board microprocessor (in hardware) to perform software tasks.

Fig. 1 shows a conceptual example of reconfigurable computing, and is discussed in detail in [13]. In this particular example application there are four tasks to be performed (Tasks A through D). An area off-chip holds several different hardware specific implementations for each of these tasks. Each of these implementations or configurations fits together with a different data profile. In the example shown, the application has received a new data profile and is about to reconfigure the 512x512 version of Task A with the 1024x1024 version of Task A. However, note that the current hardware implementation of Task B matches the new data profile. As such, the microprocessor will begin processing all of the data for Task A, while using the hardware implementation for Task B. As soon as the new Task A is written to the FPGA, it will take over for the microprocessor in processing the data for Task A.

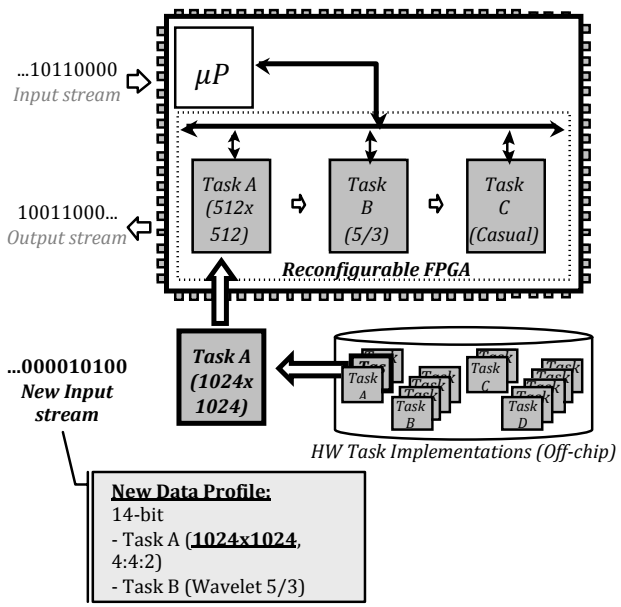

Figure 1: Reconfigurable computing relies on splitting an application into many different tasks. These tasks are then written in hardware and can be loaded onto the chip depending on parameters in a hardware data profile. Image from [13].

It is important to note that the DARES methodology presents a domain-specific approach to design streaming applications. As such, it is particularly well suited to applications that rely on a stream of incoming data stream whose metadata may change. The application shown in Fig. 1 is an adaptation of the streaming JPEG image compression algorithm. The JPEG algorithm provides a simple example on which to test and validate the DARES methodology.

This approach to embedded systems becomes increasingly important as demand for more complex embedded systems increases. Certainly more transistors can be packed onto a chip, but there are physical and economical limitations to that approach. Adding more transistors to solve a problem gets exponentially expensive upon the realization that the number of hardware configurations for an application are exponentially related to the number of parameters for that particular application. The reconfigurable computing approach allows designers to leverage the fact that area on storage devices is much cheaper than on processing devices, as well as the nonfunctional benefit that power consumption in hardware is less than software for many functions [8, 7, 15].

The methodology and tools in this paper provide a domain specific modeling environment in which to design embedded solutions for data streaming applications. Other languages currently support streaming applications for both software (OpenCL, OpenMP, and StreamIt [1]) and hardware (Streams-C [5], ImpulseC [11], ROCCC [12]). However, none of these seek to provide a modeling framework as a solution. As such, the approach in this paper may be readily adapted to provide generated artifacts in the previously mentioned languages.

The model of computation demonstrated in this paper can be best described as a cross between dynamic dataflow and communicating sequential processes. Dynamic dataflow (DDF) is a computation method that is capable of processing data in parallel but not necessarily in order [3, 2]. It takes advantage of the portions of a data stream which are not dependent on one another to do this. However, any dependencies between processing tasks can cause this approach to halt while slower processes catch up to the faster ones. Communicating sequential processes is a formal method for describing how concurrently running components in a system architecture should communicate [9, 10]. This is accomplished by having primitive processes representing the components communicating via events. In such an architecture, an event occurring in one process would have consequences in (thus communication with) other processes. DARES combines these two by using streamed tasks that process information as they receive it (DDF) and that communicate by writing to/reading from FIFO buffers (CSP).

3. DARESML

3.1 Modeling Language Design

DaresML has been described in previous publications [6]. This paper describes new modeling language developments for building models of tasks, configurations for those tasks, and the modeling of data flow in a DARES application, in order to generate the reconfiguration controller.

Fig. 2 shows the most recent metamodel for DaresML, which is expressed in MetaGME [16]. This metamodel includes updates done for the research in this paper, but the essential elements of previous iterations are also present. There are two paradigms which represent any given DARES project: a *DomainDefinition* and a *DomainSetting*. The *DomainSetting* is where data flow between tasks can be modeled while the *DomainDefinition* is where tasks and their respective configurations are defined. *Tasks* describe unique operations that the resulting embedded system must accomplish. *Configs* are the different methods for achieving those individual operations, through hardware and software. *Parameters* describe how the *Configs* differ from one another. *Ins* and *Outs* describe the inputs and outputs of the *Tasks* while *LConnections* describe how *Ins* and *Outs* atoms are connected. *TINs* and *TOUTs* describe the terminal input and output of a system while *TINConnections* and *TOUTConnections* describe how *TINs* and *TOUTs* are connected to other objects.

The new fields in this metamodel all provide vital information to the interpreter for DaresML so that the artifacts related to the *Overseer* can be generated. The *Includes* field allows a user to specify external include files that may be critical to different task configurations. The *TokenType* and *OtherType* fields provide information to the *Overseer* architecture about the type and therefore size of the data being piped through the system. The *Producer* and *Consumer* fields then allow the user to specify the location for the specialized terminal producer and terminal consumer processes that are meant to contain the new semantic structures for managing the reconfiguration process.

3.2 Instance Model

Fig. 3 shows a simple example that has been used as a demonstration in previous DARES research, for JPEG image compression (which has been extended into the updated DaresML described herein). The process is straightforward and runs a stream of data through four total tasks in succession before outputting a compressed data stream. A benefit of this example is that the task complexity is suitable for discussion in a brief paper.

Fig. 4 demonstrates how parameters are defined in DaresML. At the top level where the task definitions reside, the main parameter list

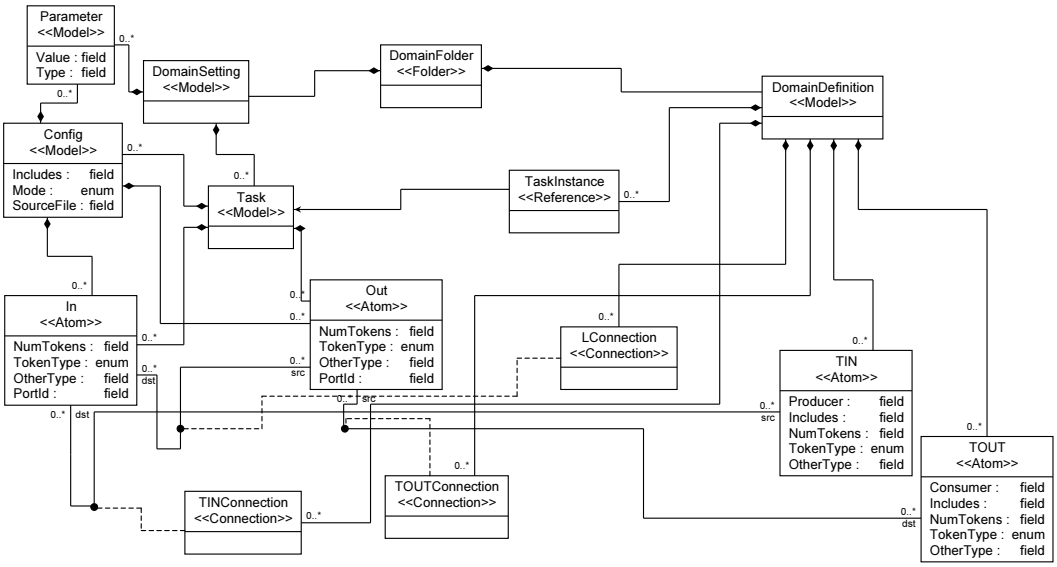

Figure 2: The metamodel for DaresML defines how users can model DARES projects. This model has been updated from [6], though the core components have not changed.

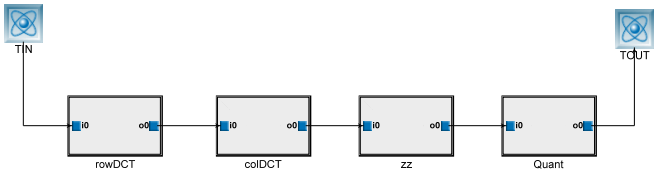

Figure 3: Model of the streaming JPEG example application. It consists of four application tasks plus a terminal producer and terminal consumer.

(with default values) can be constructed. These top level parameters define what parameters are available to configure the hardware profiles of the system. At the lower level inside of the configurations of a particular task is where the parameter list for that task can be constructed. Any of the parameters defined at this lower level will define the necessary parameter requirements for that task. For instance, in the example shown in Fig. 4, the hardware implementation of the *colDCT* task requires that the *BlockSize* parameter be set to *64* (the value attribute shown in the figure) in order for it to work properly. As such, any time the *BlockSize* parameter is set to *64*, the system will use the hardware version of the *colDCT* task, whereas when the parameter is not *64* it will use the software version.

Note that these parameters also allow the modeling language to generate multiple configurations from the same user generated source file (see Section 5.3).

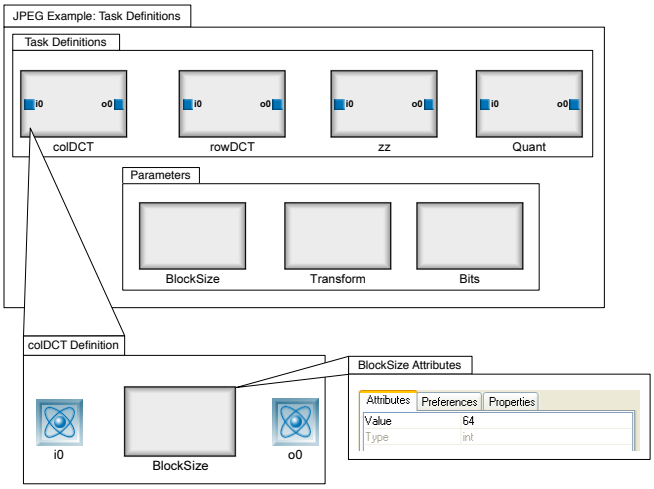

Figure 4: The JPEG example has three main parameters: *BlockSize*, *Transform*, and *Bits*. Those parameters and their values define which hardware configuration will be used for any given data stream. The inside of a configuration of the *colDCT* task is also shown in the figure. This shows that the *BlockSize* parameter is the only parameter that defines that task. It also shows that the *BlockSize* parameter has attributes for its value and that value's type.

4. OVERSEER METHODOLOGY

It is important for the reader to note that the Overseer Methodology described in this paper is not a single artifact (nor even a collection of many artifacts). Instead this methodology describes the process and semantics by which runtime FPGA reconfiguration is achieved.

The semantics necessary to the Overseer Methodology are two fold. First, DaresML makes no assumptions about the data stream for

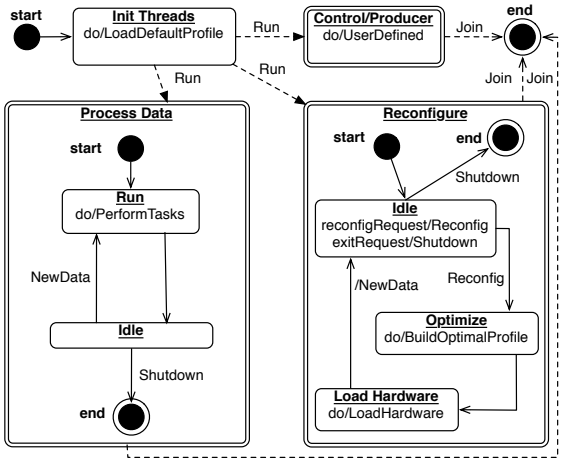

Figure 5: The state-chart diagram of the Overseer Methodology. This figure shows that the basic concept is to have a user defined control thread, a generated reconfiguration thread, and the task threads (shown here as one 'process' thread for simplicity). These three threads are spawned by an initialization process then the application completes when all three threads complete.

any given application, so the user needs the ability to determine when the FPGA will be reconfigured. Second, the user needs to be able to link the different hardware configurations for tasks to the data in a particular stream. The first concept is handled by several semantic commands given to the user, while the second is handled by parameters and parameter profiles. Fig. 5 shows the state flow of the Overseer methodology. The state-chart shows three threads spawned by an initialization process from a main thread. The *Process Data* thread is a simplification of the multi-threaded task architecture that would process the data for a given application. The *Reconfigure* thread processes reconfiguration requests made by the user. It does this by selecting an optimal hardware profile based on the current profile parameters. Then loading that profile is loaded to the FPGA before the thread returns to an idle state.

Fig. 6 shows an example of how a user might define the *Control/Producer* thread. Aside from loading data, DaresML provides the semantics that the user can employ to define the states shown in the chart. The *Initial Reconfigure* state shown in the figure is accomplished by the same semantic structures as the *Update Hardware* state. *Initial Reconfigure* simply updates the hardware with the default configuration. Also, the *Update Hardware* state first performs some necessary actions to ensure that the system begins processing data using software tasks if the necessary hardware configurations for those tasks are not currently in hardware. This is accomplished by the *ShiftToSoftware* method. The *Reconfig* method refers to Fig. 5. When run from the *Control/Producer* thread, *Reconfig* informs the *Reconfigure* thread that it should start reconfiguring the system again. If for some reason the *Reconfigure* thread is still processing a previous reconfiguration request, it will perform the new reconfiguration once it reaches it's *Idle* state. DaresML does provide the user with a *WaitForReconfigure* command that would prevent *Control/Producer* thread from continuing until the *Reconfigure* thread reaches it's idle state, but semantic structure is unused in this simple example.

4.1 Parameters

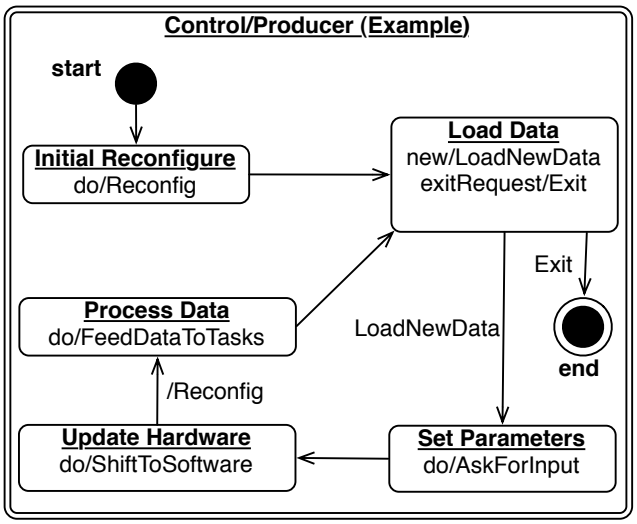

Figure 6: This is an example of how a user defined control thread might act. DaresML provides the user with the semantic structures to set parameters, update the hardware, and the initial reconfiguration. The user is responsible for loading the data stream. Note that the *Reconfig* method refers to Fig. 5.

Parameters are a new feature of DaresML. Parameters are specified in the application model at both the DomainSetting level and the Configuration level (Fig. 2). All parameters specified in a model, regardless of where they are specified, must be specified with a value. The value given to a parameter in the model is the default value for that parameter. Parameters specified in the DomainSetting are global parameters. These are the parameters that will define the overall hardware profile for any input data stream. Parameters specified in a configuration of a given task are then the necessary parameters for that configuration to be valid.

For example, in Fig. 1 the data profile shows two parameters, one that stores a Block Size, and one that stores a type of Wavelet transform. These are parameters specified at the DomainSetting level. Also, in Fig. 1 there are two configurations shown for Task A. The current configuration of Task A has a parameter for a Block Size of 512, while the configuration that is about to replace it has a parameter for a Block Size of 1024. In both of these configurations this parameter would be specified at the configuration level.

Two other important facts. First, parameters that are defined at the configuration level, but not at the DomainSetting level act as constants. Second, parameters specified in a software configuration are not considered necessary for that configuration since software configurations are the catchall if there is no hardware configuration available that meets the necessary parameter requirements. Instead software configurations simply access the current parameter profile for parameter values.

4.2 Semantics

DaresML provides the user with a total of four semantic commands that he or she can issue at any point in any task. It is expected that these commands will be issued in a terminal producer which manages the raw input data stream to the application, but that is not a strict limitation placed on the user by DaresML. All DARES appli-

cations are created with two parameter profiles for these commands to operate on: a current profile which stores the configuration that the application tasks access, and a new profile which stores the configuration for the next data stream.

4.2.1 Update

The update command begins the reconfiguration process. First, an atomic reconfiguration occurs that reconfigures the FPGA using the new profile, based on available hardware tasks. Second, as a part of the atomic reconfiguration the new profile is copied into the current profile. Third, a reconfiguration task is primed and eventually executed to move new hardware tasks onto the FPGA.

4.2.2 Reset

The reset command resets the new parameter profile to the default parameter values that the user can specify in the application model.

4.2.3 Wait

The wait command causes a task to block until a reconfiguration has occurred. This is useful (but not necessary) to prevent a new data stream from being operated on by tasks with the incorrect parameter configuration.

4.2.4 Set Parameter

The set parameter command allows the user to set the value of parameters in the new parameter profile.

4.3 Code Generation

There are two sections of code that are always generated as a part of the Overseer Methodology.

4.3.1 Atomic Reconfiguration

The atomic reconfiguration process that occurs first in the update semantic command is necessary to ensure that the new profile is copied into the current profile before any new data is processed. Also, this function helps prevent new data streams from being processed by hardware tasks with inappropriate parameter configurations.

4.3.2 FPGA Reconfiguration Task

The FPGA reconfiguration task is a software task that is present in any DARES project. This task handles optimizing the FPGA hardware profile for the application and actually managing moving hardware tasks to and from the FPGA during runtime.

5. USER GENERATED SEMANTICS

As mentioned previously, DaresML provides the user with certain semantic structures that can be used to control the reconfiguration process. These semantic structures are used in the development of the code for application tasks and for the terminal producer and terminal consumer tasks.

5.1 Terminal Producer

The terminal producer is the main entry point for an application. This is also where a user would define the *Control/Producer* thread shown in Fig. 5 and Fig. 6. It is here that a user would collect data and stream it into the application tasks that he or she has modeled with DaresML. This particular task provides a clear demarcation between processing application data and controlling the reconfiguration process as per the Overseer Methodology. However, as DaresML makes no assumptions about data streams the user is not limited to controlling the reconfiguration process from this task. It is only highly recommended.

5.2 Terminal Consumer

The terminal consumer is the main exit point for an application. It is here that a user should collate the data stream into whatever form the user requires. Under the Overseer Methodology there are no necessary procedures that the user must follow when finalizing data output, but this is a good location for a user to indicate when a data stream has finished processing. In this way the user can prevent a reconfiguration from occurring before a given data stream has completed processing. As with the terminal producer, DaresML does not force a terminal consumer on the user. DaresML only encourages the user to make use of this clear demarcation between data control and data processing.

5.3 Parameters

DaresML provides a simple way for users to access parameters in the current parameter profile. A user can simply use the name of the parameter as if it were a variable in his or her code. The modeling language takes over from there and handles accessing that parameter correctly.

Parameters allow DaresML to generate multiple hardware/software configurations from a single user generated source file. For example, the difference in C++ code for an implementation with a *BlockSize* of *64* versus *32* may just be the number of times a loop iterates. However, in a hardware implementation this difference may be drastic, but the C++ code that represents that can just be a parameter. DaresML then interprets that parameter differently for each task configuration, depending on the specifics that the user has specified in the model. This provides a simplification for the user where only one source file need be created for each task. Though, the user does have the option of using different source files for the configurations of a task in case there are differences that cannot be encapsulated in a parameter.

6. CONCLUSIONS AND FUTURE WORK

This paper demonstrates that the Overseer Methodology for the control of FPGA reconfiguration provides a user with the necessary tools to effectively model a whole reconfigurable computing system. DaresML provides modeling and semantic tools for generating task architecture and configurations, data flow models, and reconfiguration control.

So far the scope of this research has been rather limited in its focus. The next step will be to tackle a more complicated example to further verify the efficacy of the Overseer Methodology. In order to keep DARES within the trend of image compression streaming applications the next research task will be to implement JPEG2000. This will allow for the refinement of DaresML.

Furthermore, a JPEG2000 implementation will allow for the testing of other optimization techniques. For instance, the configuration of data in the input stream may effect performance beyond simple estimates for task latency. A future project may involve testing optimization techniques that actively test new strategies for optimization by processing old streams of data with new hardware profiles during system idle time.

7. REFERENCES

[1] S. Amarasinghe, M. I. Gordon, M. Karczmarek, J. Lin, D. Maze, R. M. Rabbah, and W. Thies. Language and compiler design for streaming applications. *Int. J. Parallel Program.*, 33(2):261–278, June 2005.

[2] Arvind and R. Nikhil. Executing a program on the mit tagged-token dataflow architecture. *Computers, IEEE Transactions on*, 39(3):300 –318, mar 1990.

[3] J. L. e Silva and E. Marques. Executing algorithms for dynamic dataflow reconfigurable hardware -the operators protocol. In *Reconfigurable Computing and FPGA's, 2006. ReConFig 2006. IEEE International Conference on*, pages 1 –7, sept. 2006.

[4] P. Garcia, K. Compton, M. J. Schulte, E. R. Blem, and W. Fu. An overview of reconfigurable hardware in embedded systems. *EURASIP J. Emb. Sys.*, 2006, 2006.

[5] M. Gokhale, J. Stone, J. Arnold, and M. Kalinowski. Stream-oriented fpga computing in the streams-c high level language. In *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, pages 49 –56, 2000.

[6] V. S. Gopinath, J. Sprinkle, and R. Lysecky. Modeling of data adaptable reconfigurable embedded systems. In *Proceedings of the 8th IEEE Workshop on Model-Based Development for Computer-Based Systems*, pages 276–285, April 2011.

[7] J. Henkel. A low power hardware/software partitioning approach for core-based embedded systems. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 122–127, New York, NY, USA, 1999. ACM.

[8] J. Henkel and Y. Li. Energy-conscious hw/sw-partitioning of embedded systems: a case study on an mpeg-2 encoder. In *Proceedings of the 6th international workshop on Hardware/software codesign*, CODES/CASHE '98, pages 23–27, Washington, DC, USA, 1998. IEEE Computer Society.

[9] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[10] C. A. R. Hoare. Process algebra: A unifying approach. In *25 Years Communicating Sequential Processes*, pages 36–60, 2004.

[11] Impulse Accelerated Technologies. Impulse codeveloper, 2012.

[12] Jacqaurd Computing. Roccc 2.0, 2001.

[13] S. Mahadevan, V. Gopinath, R. Lysecky, J. Sprinkle, J. Rozenblit, and M. Marcellin. Hardware/software communication middleware for data adaptable embedded systems. In *Proceedings of the 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pages 34–43. IEEE Computer Society Press, 2011.

[14] A. Milakovich, V. S. Gopinath, R. Lysecky, and J. Sprinkle. Automated software generation and hardware coprocessor synthesis for data-adaptable reconfigurable systems. In *Proceedings of the 19th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pages 15–23, 2012.

[15] J. Mu and R. Lysecky. Autonomous hardware/software partitioning and voltage/frequency scaling for low-power embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 15(1):2:1–2:20, Dec. 2009.

[16] Ákos Lédeczi, Árpad Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing domain-specific design environments. *IEEE Computer*, 34(11):44–51, November 2001.