

An Object-oriented Micro-kernel Supporting Transparent Distribution of Embedded Control Applications

Stephen H. EDWARDS
Dept. of Computer Science, Virginia Tech
660 McBryde Hall, Mail Stop 0106
Blacksburg, VA 24061, USA
+1 540 231 5723

edwards@cs.vt.edu

ABSTRACT

Traditional approaches to embedded control software design make it difficult to reuse parts of existing designs, build useful component libraries, and transparently map applications across multiple processors when they are available. This paper describes DARK++, an object-oriented micro-kernel that supports dataflow-style control applications. DARK++ is designed specifically to support embedded power electronics control systems. In addition to the natural fit that a dataflow architecture provides for signal flow and control problems, DARK++ provides powerful support for transparent communication among distributed software elements, automatic partitioning of an application across the processing resources available, and self-configuration of control applications in multi-processor environments. This work is an example of computing technology that makes use of basic communications features to better support control applications in a specific application domain—power electronics.

Keywords: Dataflow, power electronics, C++, distributed computing, processor allocation, network protocol

1. INTRODUCTION

Because of limited hardware resources and stringent real-time demands, embedded control software is often designed in a way that is tightly coupled to the problem at hand. This can make it difficult to:

- Reuse parts of an existing software design in a new system,
- Construct or extend designs by composing pre-built components from a software parts library, or
- Enhance performance by mapping an existing application across multiple processors if they are available.

DARK++, the object-oriented Dataflow Architecture Real-time Kernel, is a lightweight real-time operating system that is designed to support embedded control applications that use a dataflow architecture [16]. The benefits of a dataflow architecture for component-based design and extension of control applications have been described elsewhere [11, 12]. This paper focuses on the powerful support this approach provides for transparent communication among distributed software elements, automatic partitioning of an application across the processing resources available, and self-configuration of control applications in multi-processor environments. DARK++ is an example of computing technology that makes use of basic communications features to better support control applications in a specific

application domain—power electronics. While DARK++ was originally designed for modular power electronics systems, the techniques are applicable in many control arenas.

2. BACKGROUND

As computer-based control systems become more complex, so does the control software embedded within them. This problem is even more acute in some control domains, such as power electronics. Current research in power electronics has begun to focus more on distributed control architectures that use a modular set of building blocks to form power stages [3, 17]. This research has led to the Power Electronics Building Block (PEBB) approach, which promotes the use of modular, reusable, interchangeable, standardized physical components.

The PEBB strategy has also led to the design of modular control processors [8] that can supervise such power stage elements over a communications network. Designing general-purpose, reusable control boards that are independent of power stage elements allows this design element to be applied in many different products. Simultaneously, it allows software to be written to a common platform, rather than to a custom, power stage-specific control solution. The presence of a communications network between the controller and the managed devices makes it easy to devise multi-controller systems and design distributed control approaches.

Unfortunately, traditional embedded software design techniques rarely lead to adaptable, flexible control applications that naturally mesh with the modular, distributed, building-block approach being developed for power stage management. How can we best support a similar building-block approach in the control software?

3. DATAFLOW

Dataflow is a style of software architecture [15, 4] in which an application is composed of a collection of independently executing software components that do not directly interact with each other. Instead, software components communicate by sending one-way messages through buffered message queues called data channels. Computation is data-driven: a component waits for data to arrive on its incoming data channel(s), and then produces results that it sends downstream on its outgoing data channels. Dataflow is a natural match for most low-level control tasks because it allows one to directly map signal-flow-based designs into a software structure.

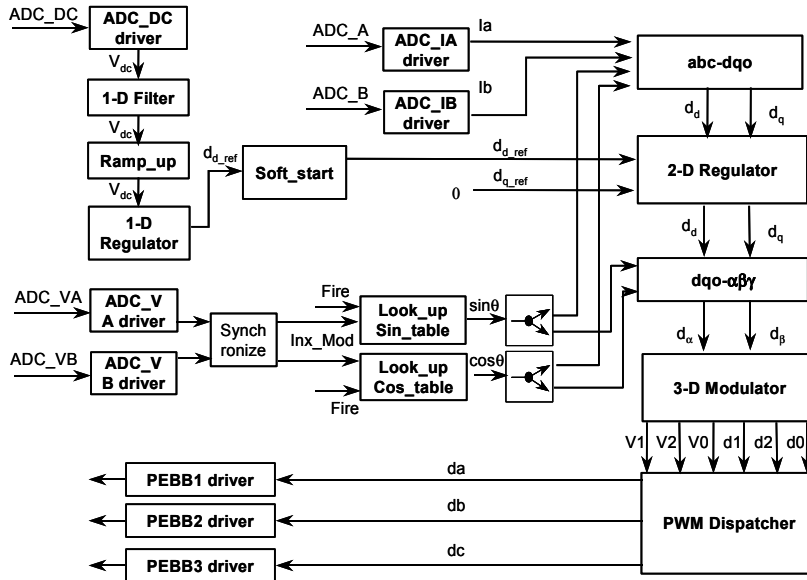


Figure 1: Dataflow-based control design for a closed loop three-phase rectifier.

Dataflow techniques have been used extensively in the control community. Two of the most popular commercial tools using this approach are LabView [10] and SimuLink [1]. SimuLink’s Real-Time Workshop [14] add-on will even allow one to generate compiled code directly from a SimuLink model of a control algorithm. Both dSPACE [7] and ABB’s OPCoDe project build on SimuLink’s powerful environment to provide additional development capabilities in the embedded control domain.

Dataflow designs are easily depicted graphically. Figure 1 shows the dataflow design of the control algorithm for a closed loop three-phase rectifier. Such a dataflow graph (DFG) has vertices representing computation or comparison elements and edges representing the communication channels that interconnect these elements. Each element (vertex) in the graph can fire when there is data present at each of its input ports. Dataflow inherently supports concurrency because data flowing along different parts of the dataflow graph can be carried out in parallel. Dataflow graphs can be cyclic, if desired. In Figure 1, the “control loop” can be thought of as a large cycle. Sensor signals from the power stage flow in at the upper left, are transformed by the control algorithm, and flow back out as commands to the power stage at the lower left. The physical actions of the power stage and the resulting change in sensor values completes the cycle.

In a dataflow graph, a node or actor can have more than one way of being activated or fired; these are described by the input ports to that node that need to have data in them for the node to fire. This means that a node need not always have data in all of its input ports in order to be activated. It can take different actions based on which of its input data channels has/have data in them. In this way, dataflow graphs are different from Petri nets.

A dataflow approach offers significant advantages, including natural support for a parts-based approach to software composition, increased reuse, and greater extensibility. There is also a natural analogy between the logic of dataflow designs and the flow-based metaphors that electrical engineers use regularly. Further, the limited interaction mechanism present in dataflow designs makes it easy to take an application, distribute it among multiple processors, and transparently support the inter-

processor communication without requiring changes to the software design or even to the implementation of the individual software components.

The biggest limitation to dataflow designs is the run-time cost often associated with the approach. If one implements each software part as a separate concurrent process, a significant amount of processor time can be spent switching between processes and managing inter-process communication. If these costs can be overcome, however, the advantages provided by a dataflow approach are compelling.

4. DARK++

We have designed and tested a lightweight real-time micro-kernel to support dataflow-based control applications. The Object-oriented Dataflow Architecture Real-time Kernel (DARK++) [2] is implemented in C++, based on an earlier design in C [16]. The micro-kernel supports all the features needed for embedded dataflow applications, including concurrent execution, priority-driven context switching, data channel communications, and dynamic, data-driven scheduling.

At its heart, DARK++ is a priority-driven micro-kernel for supporting multithreaded applications. It does not use time slicing—instead, the highest-priority thread is always executed. If multiple threads of equal priority are available, the current thread runs to completion and is only switched out once it waits for more data on its input ports. This strategy helps to minimize context switching. In addition, DARK++ can take advantage of dual register sets on hardware such as the Analog Devices ADSP-21160 SHARC. By using one register set for the micro-kernel and one for the application, context switching times can be reduced dramatically.

Since all inter-thread communication in DARK++ must occur through data channels, inter-thread synchronization concerns are minimal. Each data channel has one reader and one writer. By ensuring that read and write operations happen atomically from the application’s perspective, no other synchronization support is needed.

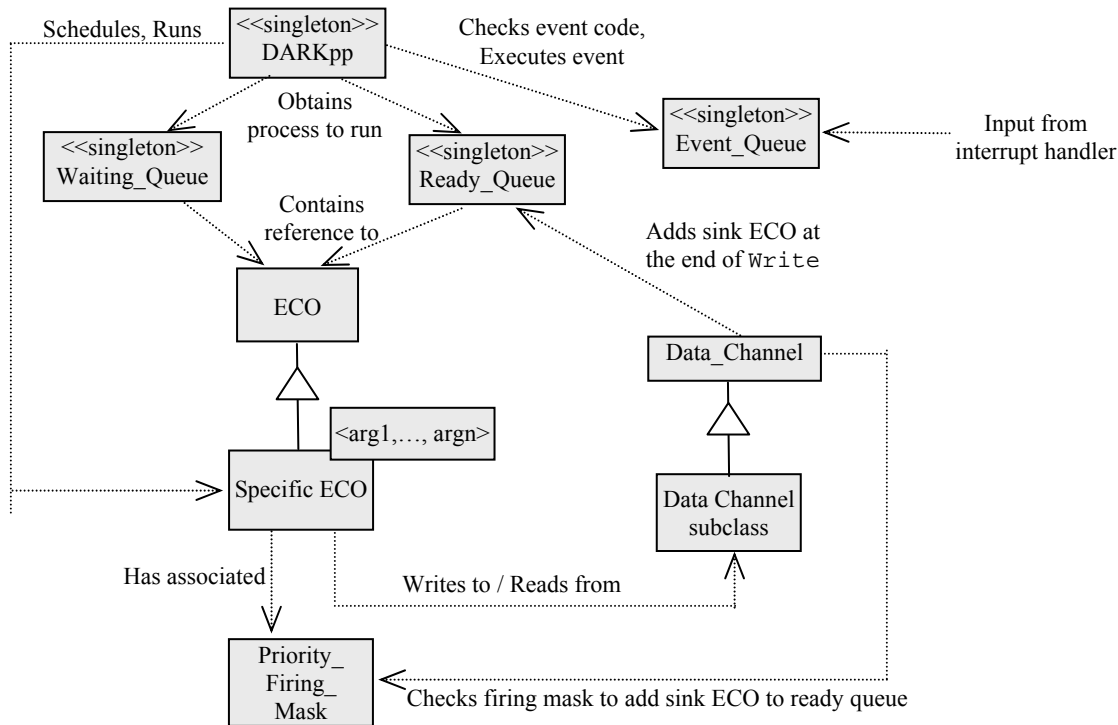


Figure 2: DARK++ class diagram.

Data channels also play a large role in thread management. Each thread has an associated firing rule that determines when it is ready to wake up and process incoming data. A firing rule is a priority-ordered list of input conditions, each of which is expressed as a simple bit mask indicating which input ports must contain data. Each bit mask in the firing rule is also associated with a new priority, which the thread will take on if awakened by the corresponding mask. DARK++ automatically maintains information about the empty/non-empty status of each data channel for use in testing a thread’s firing rule. After each write operation, the receiving thread’s firing rule is tested. If any mask in the rule triggers, the receiver becomes ready for execution with the associated priority, preempting the current thread if necessary.

Figure 2 depicts the main classes in DARK++. The micro-kernel is represented as a singleton object of the class `DARKpp`. Individual threads are also objects, all of which descend from a common `ECO` base class. These threads, or “elementary control objects,” can be instantiated from a class library of existing dataflow elements. Alternatively, new custom ECOs can easily be created from scratch, or as extensions to existing ECO classes. The `ECO` base class defines the API functions that threads use to communicate with the micro-kernel.

Alongside the `ECO` base class, the `Data_Channel` base class defines the features needed to interconnect threads. The `Data_Channel` base class defines the common features possessed by all data channel objects. It has two main subclasses that are both C++ templates: one for “mailbox” data channels and one for “queued” data channels. A mailbox channel can hold only one data item at a time, and is highly optimized internally to act much like a single “shared variable” with built-in, streamlined synchronization. A queued channel is a FIFO buffer with a fixed capacity. Either class template can be instantiated with any type to create a strongly-typed data connection between two ECO threads. All type-checking is performed

at compile time, and any mistyped connection errors in the dataflow graph are detected statically by the compiler.

Figure 2 also shows the three main internal structures used by the micro-kernel: the ready queue, the waiting queue, and the event queue. The ready queue is a priority queue of threads that are ready to execute—only threads that have fired are held in the queue. When the current thread completes its work and waits for new input values to arrive, DARK++ switches to the next ready thread. The API also allows a thread to sleep for a specific time period, either with or without its firing rule enabled. A thread invoking a clock-based wait is placed on the waiting queue, arranged in sorted order according to absolute wakeup time. The event queue is used for interrupt management. The external event ISR in DARK++ simply logs incoming event interrupts into an event queue, a shared data structure used to communicate with the micro-kernel. After the ISR completes, the micro-kernel takes control, switching to its own register set if needed. Each incoming event is translated into a corresponding message on a specific “interrupt-driven” data channel. For example, incoming sensor values or network messages easily can be mapped onto such interrupt-driven data channels. This allows external events to be seamlessly merged into the dataflow model used by the application.

In addition, DARK++ allows key features to be tuned at compile time without affecting application source code. DARK++ allows the application designer to choose whether or not to include active preemption, whether to use a fixed, cyclic execution order or a fully dynamic schedule, whether to use data-driven process wakeup rules, and whether to use queued data channels or simple mailboxes. The choice between queued channels or mailboxes can be made on a per-channel basis simply by changing the declaration of the corresponding data channel object in the DFG.

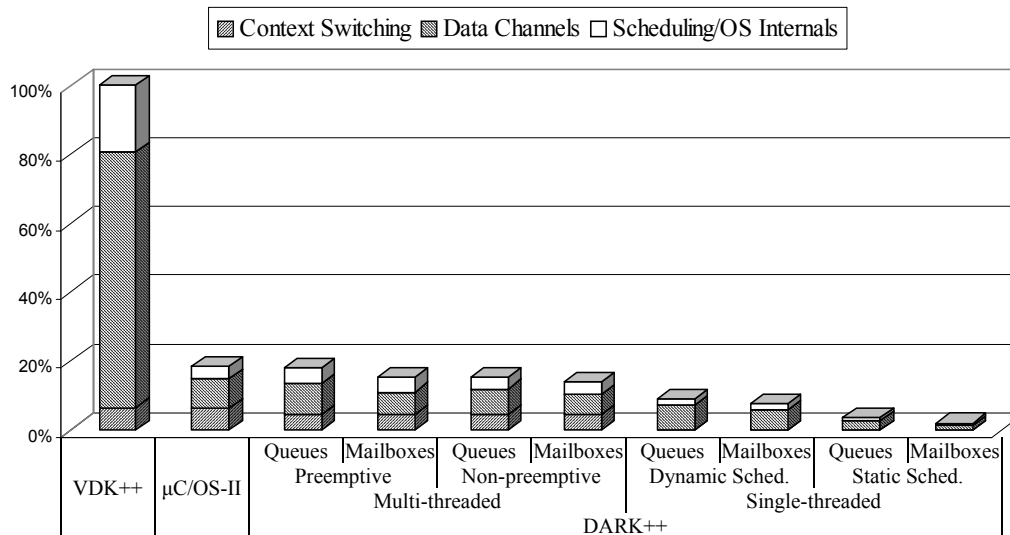


Figure 3: Relative overhead of DARK++ with different feature sets enabled, compared to two commercial RTOSes.

Figure 3 summarizes the overhead associated with dataflow processes in DARK++, compared to two commercial real-time operating systems: VDK++, which is marketed by Analog Devices specifically for the DSP used in the experiment, and μ C/OS-II. The overhead is subdivided into three categories: the time spent context switching, either between application threads or between the application and the micro-kernel, the time spent in data channel calls, including the cost of checking firing rules for down-stream threads, and the time spent in other RTOS functions, including the maintenance of internal data structures.

Figure 3 also shows the relative overhead when selected DARK++ features are turned off. Data was averaged over three separate power stage control algorithms. It is clear that careful design of micro-kernel features can bring the overhead of dataflow applications to an acceptable level.

DARK++ supports four distinct feature configurations with progressively fewer features built-in. DARK++ is most capable in its **multi-threaded, preemptive configuration**. Here, all features are available, and context switching between threads is governed by priorities. For applications where priorities are not used to dynamically manage the scheduling of threads, the **multi-threaded, non-preemptive** configuration can be used instead. Here, DARK++ implements a more efficient form of cooperative context switching, allowing each thread to run through one complete data processing cycle before switching to another thread. In this configuration, context switching is only performed when a thread waits for new data to arrive. This reduces the number of context switches and also simplifies the internal maintenance tasks that must be performed by the kernel.

Alternatively, one can further reduce the overhead by using a single-threaded kernel configuration. In single-threaded mode, each ECO, or thread body, treated in a manner similar to a single procedure. Each thread's "procedure" is then invoked to execute one input/process/output cycle for that dataflow component. As a result, context switching between threads is reduced to a basic procedure call mechanism, and no thread is interrupted. In the **single-threaded, dynamically scheduled** configuration, firing rules are used to determine which thread(s) are ready for execution, and the procedure call mechanism is used in place of context switching. Alternatively, the **single-threaded, statically scheduled** configuration eliminates the

overhead of firing rules and uses a fixed order of thread firing specified at compile time by the application designer.

In any of these four configurations, the application designer can choose to use fixed-size FIFO queues for data channels, or to use speedier single-slot mailboxes. Figure 3 shows the progressive gains in performance achieved by eliminating the overhead from these DARK++ features. In practice, careful design of power stage control algorithms will allow them to operate correctly with the leanest, most efficient DARK++ configuration, although other DARK++ capabilities are available for more demanding applications. For applications that permit mailbox usage, this results in 89% less overhead than the multi-threaded, preemptive DARK++ configuration using queued data channels, an 89.3% improvement over μ C/OS-II, and a 98% improvement over VDK++.

Note that all of the configuration changes to the operating features inside DARK++ are completely transparent to the application. No source code modification is needed, either in the application's DFG or within any ECO class definition. Instead, the source code underlying the micro-kernel's API uses conditional compilation to selectively include or exclude critical code regions based on the configuration selected. The configuration can then be specified using a preprocessor symbol definition on the command line or in a makefile.

5. TRANSPARENT DISTRIBUTED COMMUNICATION

Because dataflow components interact only via data channels, distributing an application is simplified. The power electronics control systems under consideration use a DSP-based controller connected to a fiber optic communications network using a control-oriented protocol called PESNet [9]. The network uses a dual ring topology similar in many respects to the Fiber Distributed Data Interface (FDDI). Like FDDI, PESNet uses the two counter-rotating rings to achieve fault tolerance and automatic recovery from single-point network failures. While FDDI uses a token ring model, however, PESNet uses a single-packet format with live packets simultaneously in transit on all links. PESNet has been inspired by other low-level motion control and device control protocols. By providing a standardized communication infrastructure for PEBB-based devices with

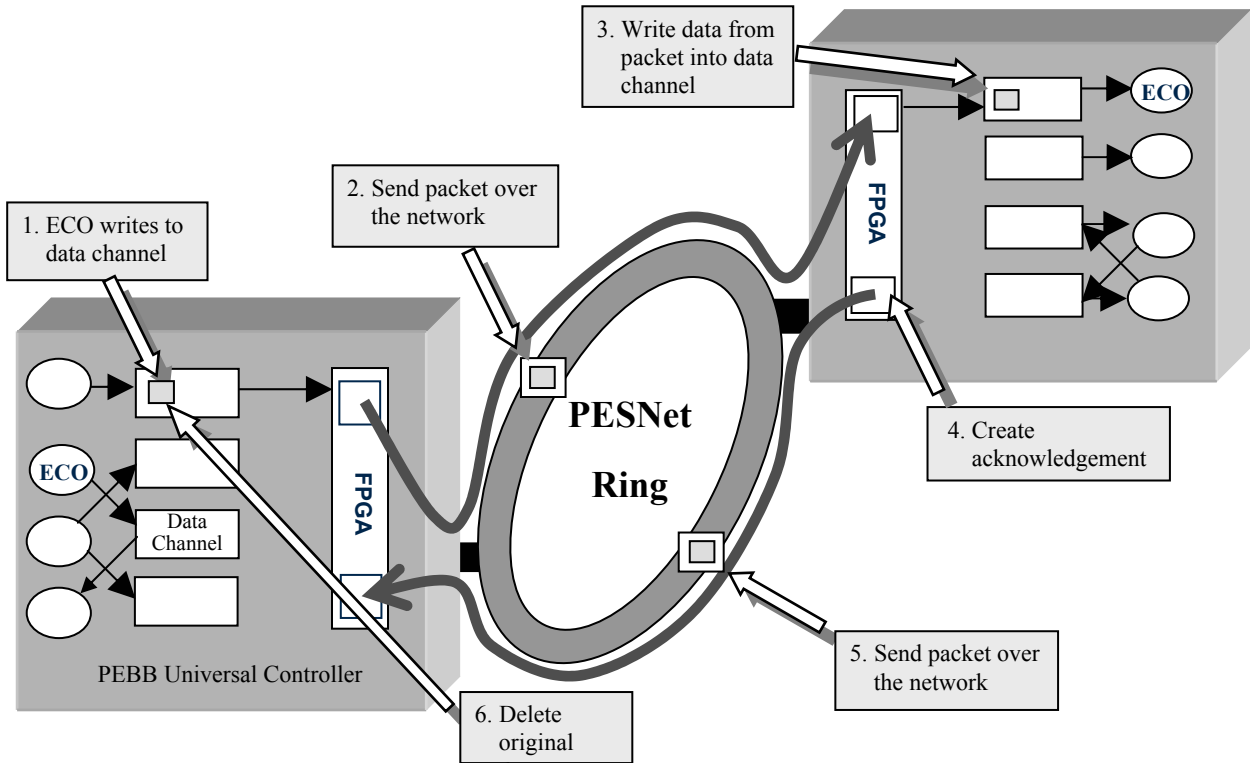


Figure 4: Transparently mapping data channel messages over the network.

basic fault tolerance built-in, it provides a clean solution to the board interconnect issues that must be resolved for plug-compatible power electronics devices.

Using PESNet as a base, we have designed the necessary support for DARK++ to transparently map data channels across the communications network. In a dataflow design, neither the sender nor the receiver directly addresses the other. Instead, the sender writes to a data channel, and the receiver reads from it. To support transparent messaging across the network, two data channels are created: one on the sender's processor and one on the receiver's. When the sender writes a message, DARK++ bundles it into a network packet and sends it to the alternate processor. On the receiving end, the data is unpacked into the receiver's incoming data channel where it is ready for processing. The micro-kernel on the receiving end then sends an acknowledgement back. Once the acknowledgement is received, the original data can then be removed from the sender's data channel. Figure 4 depicts this strategy. In effect, this approach splits one data channel in the original DFG into two, and places the micro-kernel itself as a transparent forwarding agent between the two. Data channels mapped over PESNet in this fashion are also similar in spirit to interrupt-driven data channels, with network messages playing the role of the corresponding external triggering events.

While mapping simple data channels across a communications network is easy, it is also all that is required to support distributed execution of dataflow applications. This is a result of the fact that dataflow components communicate only through data channels, and do not interact in any other way.

6. PROCESSOR ALLOCATION FOR DATAFLOW APPLICATIONS

Much research has been done on the issues of distribution and real-time scheduling of dataflow applications [5, 6, 13]. Al-

though computing an optimal solution for distributed scheduling is NP-complete, many useful heuristics have been described. Using the topological constraints of an application's dataflow interconnections, the relative running time of each dataflow component in the system, and the time taken to send a network-based message between two distributed components, an efficient algorithm for automatically allocating dataflow components to processors while respecting real-time deadlines can be created. We have implemented and tested an algorithm based on Kim and Browne's linear clustering heuristic [5], which partitions the dataflow graph for the entire application into smaller subgraphs that will be more efficient if allocated to the same processor. Each such subgraph is then allocated to an available processor using a load balancing heuristic.

The result is a fully automated approach to distributing an application written on top of DARK++. This algorithm computes a static allocation and it can be run as part of the application's startup phase. The same library of ECO subclasses and the complete application DFG can be provided as part of the startup code on each processor. As the system is powered up, the heuristic processor allocation algorithm is executed to determine where each process in the DFG will live. At this point, each processor need only start up the threads (ECOs) that it has been allocated. Data channels between threads allocated to the same processor are implemented in-memory; data channels between threads allocated to different processors are mapped across the network as discussed in Section 5.

The result is a self-configuring application that, upon discovering how many processors are available on the control network during startup, can automatically compute its own processor allocation to adapt to the resources available. We are currently in the process of demonstrating a multi-processor control algorithm for a boost rectifier in our test environment.

7. CONCLUSIONS

A dataflow approach to constructing embedded control software offers many advantages. This paper focuses on how DARK++ provides powerful support for transparent communication among distributed software elements, automatic partitioning of an application across the processing resources available, and self-configuration of control applications in multi-processor environments. This work is an example of computing technology that makes use of basic communications features to better support control applications in a specific application domain—power electronics.

ACKNOWLEDGMENTS

This work was supported primarily by the Office of Naval Research under Award Number N00015-01-1-0954 and secondarily by the ERC Program of the National Science Foundation under Award Number EEC-9731677.

REFERENCES

- [1] A. Ba-Razzouk, A. Pittet, A. Cheriti, and V. Rajagopalan, "SIMULINK Based Simulations of Power Electronic Systems," In *IEEE 4th Workshop on Computers in Power Electronics*, 1994, pp.105-108.
- [2] S. Bhakthavatsalam, *Measuring the Perceived Overhead Imposed by Object-Oriented Programming in a Real-time Embedded System*, MS Thesis, Dept. of Computer Science, Virginia Tech, 2002. Available on-line at: <http://web-cat.cs.vt.edu/PEBB/publications.php>.
- [3] I. Celanovic, N. Celanovic, I. Milosavljevic, D. Boroyevich, and R. Cooley, "A New Control Architecture for Future Distributed Power Electronics Systems," In *Proc. 31st Annual Power Electronics Specialists Conf., PESC 00*, Vol. 1, IEEE, 2000, pp. 113-118.
- [4] D.E. Culler, "Dataflow Architectures," In *Annual Review of Computer Science*, Vol. 1, Annual Reviews Inc., Palo Alto, CA, 1986.
- [5] A. Darte, Y. Robert, and F. Vivien, *Scheduling and Automatic Parallelization*, Birkhauser, Boston, MA, 2000.
- [6] R. Davoli, F. Tamburini, and L. Giachini, "Scheduling Data Flow Programs in Hard Real-Time Environments," In *Proc. Formal Techniques in Real Time and Fault Tolerant Systems*, Springer, Lecture Notes in Computer Science Vol. 1135, 1996, pp. 263-278.
- [7] dSPACE, Inc., "dSPACE - Solutions for Control", web page: <http://www.dspaceinc.com/>.
- [8] J. Francis and D. Borojevic, "Design of a Universal Controller for Distributed Control and Power Electronics Applications," CPES 2001 Power Electronics Seminar, Virginia Tech, Blacksburg, VA, 2001, pp.543-546. Available on-line at: http://web-cat.cs.vt.edu/PEBB/R94_Francis.pdf.
- [9] J. Francis, J. Guo, and S.H. Edwards. "Protocol Design of Dual Ring PESNet (DRPESNet)," CPES 2002 Power Electronics Seminar, Virginia Tech, Blacksburg, VA, 2002. Available on-line at: <http://web-cat.cs.vt.edu/PEBB/CPES02-Francis.pdf>.
- [10] P. Gray, R. Massara, and M. Hollier, "LabVIEW, a Tool for Modelling Complex Signal Processing Architectures," "In *IEE Colloquium on The Use of Systems Analysis and Modelling Tools: Experiences and Applications*, 1998, pp.3/1-3/6.
- [11] J. Guo, S.H. Edwards, and D. Borojevic, "Elementary Control Objects: Toward a Dataflow Architecture for Power Electronics Control Software," In *Proc. IEEE 33rd Annual Power Electronics Specialists Conf., PESC 02*, Vol. 4, IEEE, 2002, pp. 1705-1710.
- [12] J. Guo, S.H. Edwards, and D. Borojevic, "Implementing Dataflow-based Control Software for Power Electronics," In *Proc. IEEE 9th Workshop on Computers in Power Electronics (COMPEL)*, 2002.
- [13] Y. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, Vol. 31, No. 4, Dec. 1999, pp. 406-471.
- [14] The MathWorks, "The MathWorks – Real-Time Workshop", web page: <http://www.mathworks.com/products/rtw/>.
- [15] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.
- [16] K. Singh, *Design and Evaluation of an Embedded Real-time Micro-kernel*, MS Thesis, Dept. of Computer Science, Virginia Tech, 2002. Available on-line at: <http://web-cat.cs.vt.edu/PEBB/publications.php>.
- [17] J.D. Van Wyk and F.C. Lee, "Power Electronics Technology at the Dawn of the New Millennium—Status and Future," In *Proc. IEEE 30th Annual Power Electronics Specialists Conf.*, Vol. 1, IEEE, 1999, pp. 3–12.