LabVIEW Based Embedded Design

Sadia Malik
Ram Rajagopal

Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712
malik@ece.utexas.edu
ram.rajagopal@ni.com

Abstract

LabVIEW is a graphical programming tool based on the dataflow language G. Recently, runtime support for a hard real time environment has become available for LabVIEW, which makes it an option for embedded system prototyping.

Due to its characteristics, the environment presents itself as an ideal tool for both the design and implementation of embedded software. In this project we study the design and implementation of embedded software by using G as the specification language and the LabVIEW RT real time platform. One of the main advantages of this approach is that the environment leads itself to a very smooth transition from design to implementation, allowing for powerful cosimulation strategies (e.g. hardware in the loop, runtime modeling).

In order to evaluate the effectiveness and possible improvements on G as an embedded software description language we prove that, under certain conditions and semantic restrictions, a non-terminating G program is strictly bounded in memory. We provide a mechanism to always determine a valid G schedule. The theory formalizes the current behavior of the G execution system, and provides insights in how to use G for embedded processing. Also we present an $O(N^3)$ algorithm for detecting non-determinism in a G program.

Finally, we implement a state-of-the-art embedded motion control algorithm using LabVIEW as the specification, simulation, and implementation tool.

1. Introduction

LabVIEW, developed by National Instruments, is a graphical programming environment based on a dataflow model. It was originally targeted towards the test, measurement, and automation industries.

In recent years there has been a tremendous growth in the embedded software systems market. It was motivated by, among other factors, the reduction in the cost of hardware and the need for fast portable solutions with short time to market. National Instruments developed LabVIEW RT (Real Time) to answer these demands.

The objective of this project is to develop a framework for using the LabVIEW RT software and hardware environment for embedded systems design. To define a consistent framework for embedded design, we prove that the underlying model of computation of LabVIEW, the G language, satisfies fundamental requirements for languages for embedded systems specification. As an example of the advantages of using the LabVIEW embedded environment for development, we implement an embedded motion controller using LabVIEW RT.

2. G in Embedded Design

There are many different definitions for embedded software, but an accepted one is a system, with extremely restricted user interface, that acts on infinite streams of data. The main desired requirements for specifying and executing an embedded program can be listed as [P95]:

[R1] The program specification should preferably be determinate, and therefore the outputs should be consistent to the inputs, regardless of execution details. Also, the program specification should be sample rate consistent and causal.

[R2] The scheduler should implement a complete execution of the G program, so if the program is non-terminating, it should not deadlock.

[R3] The scheduler should, if possible, execute a bounded G program in bounded memory.

These requirements are quite natural and express that a well-behaved program should be able to operate without hurdles in standard embedded environments. In this section we will present algorithms and restrictions that guarantee that the main requirements are met, thus allowing G to be transparently used in embedded software design.

In order to derive our algorithms and restrictions, we will assume that for every execution of the G program the front panel input values are fixed. This assumption is reasonable under the hypothesis that the program is running in an embedded environment. Due to space constraints, all of the proofs are presented in the appendix.

2.1 Determinism and Consistency

A G graph is always sample rate consistent because it is homogeneous. Also causality in G is guaranteed, because the semantics do not allow delayless feedback loops.

Due to the language semantics [AK98], non-determinism only arises in G when local or global (storage) variables are used as part of a diagram. Because of the fact that G allows multiple reads and writes to a single variable, race conditions may arise.

An efficient algorithm to identify non-determinate programs in G is given in Table 1. This algorithm is based on propositions 1 and 2 below. A flattened diagram is a G diagram where all higher level actors are decomposed into G's basic actors.

Proposition 1: If all actors in a flattened G diagram only read from storage variables, then the program is determinate.

Proposition 2: If an actor A writes to a storage variable, and an actor B reads or writes to the same storage variable, then this program can be determinate if and only if there is a directed path from actor A to actor B or vice versa.

The procedure for creating virtual edges for G structures is presented in the appendix. The "Find Non Determinism" algorithm is of order $O(|\text{actors}|^3)$ [CL90].

Find Non Determinism	

Flatten the G graph.
If (no storage variable write in diagram)
{ program is determinate;}
else
{ Create virtual edges for all G structures;
Run All pairs Shortest path
algorithm on graph [CL90];
For each storage variable S{
Select a vertice V_A that writes to
an instance of S;
	For every vertex V_k connected to an instance of S
	{
	If (dist(V_A,V_k) $= \infty$ and
	dist(V_k,V_A) $= \infty$)
	return Program is non-determinate;
	}
	} //for storage
	} //else
	return Program is determinate;

Table 1: Algorithm for identifying non-determinate G programs

2.2 Bounded Memory Execution

A scheduler that operates in G does not need to be concerned about bounded memory scheduling. G programs can only be either strictly bounded or unbounded in memory (proposition 3). The primary reasons are the homogeneity of the G graph and the syntax restrictions imposed by the language.

Proposition 3: G programs are either strictly bounded or unbounded in memory [P95]. Unbounded memory programs are defined by the use of *build array* actors inside *while* loops or having indexing enabled for *tunnels* in such loops. Therefore a scheduler can always satisfy requirement R3.

A simple requirement that would force every G program to be strictly bounded in memory is to force every *while* loop to have a maximum count.

2.3 Complete Execution

Because of the semantics of G, and the fact that it is homogeneous, any valid schedule guarantees complete execution of a determinate program. There are no possible deadlocks, unless they are introduced by the behavior of the actors (e.g. infinite while loop). Theorem 1 states that every determinate diagram has a valid execution schedule that satisfies requirement R3. If every loop has a maximum count, then no G program will ever deadlock.

Theorem 1: Given a determinate G diagram, in the sense of section 2.3.1, there is always a valid schedule that consists of a sequential Breadth First Search [CL90] order firing of every actor on the diagram.

2.4 G in LabVIEW RT

In many applications it maybe required that a certain periodic schedule is executed within a hard timed loop. For example, in a digital control application hard timing is extremely important to keep the system stable [DT97].

LabVIEW RT was developed to allow a hard timed execution of a G diagram. It is basically an execution kernel (RT Engine) that runs the G diagram under the PharLap real time operating system. The execution kernel is supported by a standard industrial PC (PXI) as well as by a custom made data acquistion board. The development is done on a host computer using a standard LabVIEW interface. The host computer is linked to the RT system for program download and user interface update.

3. LabVIEW RT Based Embedded Motion Control

Formally, we have shown that G is adequate for embedded design (section 2). In this section we discuss the implementation of an embedded motion controller based on G.

A typical motion control system [NI99] is presented in Figure 3. The host machine handles the user interface and higher-level executive routines. The trajectory generator outputs position vs. time data on the fly. This data is used by a PID (Proportional-Integral-Derivative) [DT97] control loop to drive the plant, that could be composed of several motors (axis).

In existing systems, the control loop is programmed on a custom board and can only be changed by rewriting the code. The key limitation of this approach is the lack of flexibility in terms

of algorithm and parameter ranges. Moreover, PID control limits the system performance, as it treats every axis independently.

Next generation motion control algorithms require increased flexibility in the implementation. Therefore, these advanced algorithms cannot be easily translated into embedded system firmware.

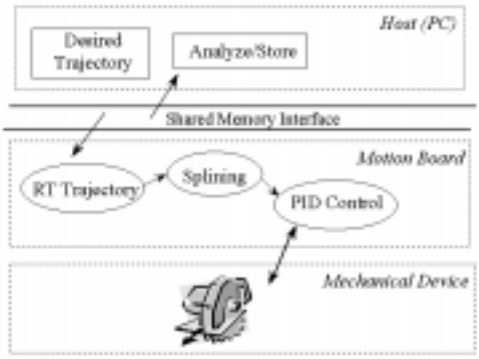

Figure 1: Current Motion Control System

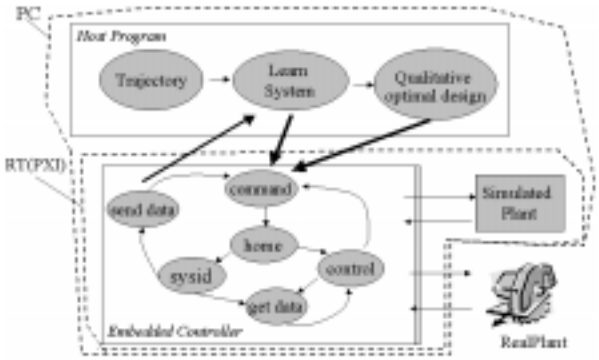

Figure 2: LabVIEW RT based System

3.1 LabVIEW based Motion Control Architecture

When compared with existing motion control architectures, our system provides an easy and flexible control environment. The new architecture (Figure 2) encapsulates a mechanism for the user to start with a completely unknown plant, identify it's parameters, design a control algorithm of his choice, and run it on real time hardware in one seamless system. This is a new paradigm in embedded motion control.

In the new architecture, the host is still in the PC, which includes system identification capability [DT97]. Thus a model of the plant is automatically generated based on a simple trajectory experiment. With a model of the plant, the whole range of control algorithms can be explored extending the capabilities of PID. We exemplified this by implementing an optimal control design routine in the host program.

The RT system itself runs the embedded controller. The controller is a state machine that can select between a PID algorithm for system identification and a multiple input multiple output

(MIMO) feedback algorithm [DT97], defined by the control design program in the host PC. In addition, the host communicates with the RT board using a TCP/IP based protocol. This enables the RT subsystem to be used anywhere in a distributed system and still be controlled by the host. This adds tremendous flexibility in realistic embedded motion control systems.

The embedded controller, as well as the host program, are completely implemented in G. The controller is implemented using a combination of Finite State Machine (FSM) and Dataflow programs [GL99]. For the end user, such an implementation allows for adaptability, as completely different control strategies could be programmed and easily added as a new state in the controller. The communication and input/output infrastructure would remain the same.

Another important characteristics of the system is that the structure is such that the real plant may be replaced by a simulated plant in software without requiring significant modifications. This technique is called 'Hardware in the loop' (HIL) [HL99] and is a very popular system verification and simulation technique. Traditional systems do not have such a well-defined and easy to use interface that can support HIL experiments, even for user defined algorithms.

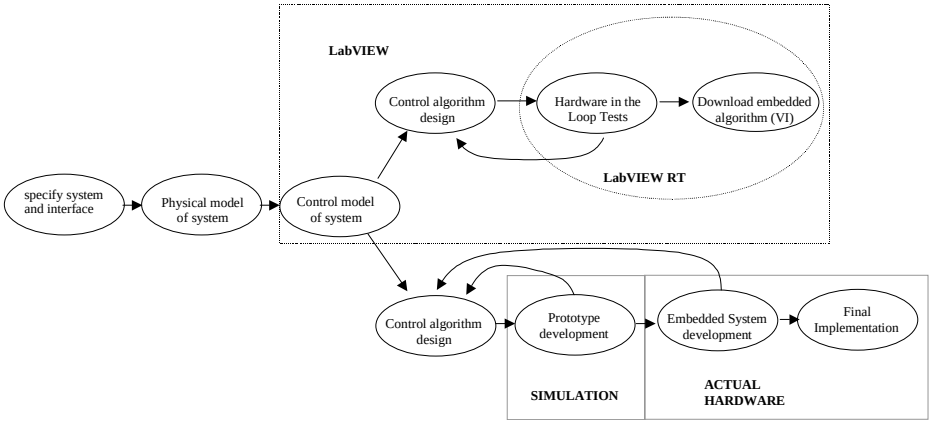

Figure 3: Control Design Cycle

Figure 3 compares the traditional design process with the new design process that becomes available as a result of our system architecture. This comparison clearly demonstrates the reduction in the development cost and cycle time due to the use of LabVIEW based design.

3.2 Typical Execution

In a typical scenario, the user would use the implemented system without any modifications. The user inputs a trajectory using a GUI window. This trajectory is converted to a set of points at defined time, using cubic spline interpolation and taking into consideration restrictions on velocity and acceleration.

The trajectory is sent to the RT board. The RT system will run an assumed PID algorithm to make the system follow the desired trajectory, using the previously defined points as set points. The actual path followed by the system and the desired trajectory are returned by the RT system, and used by the host system identification program to determine plant model parameters. The interactive state space control design program is then executed in the host, allowing the user to define the controller. Then the controller can be invoked at any time by the state machine, when commanded by the host computer.

The implemented control design routine is based on the Linear Quadratic Regulator (LQR) and optimal control principles [DT97]. The idea is to balance the conflicting requirements of minimum error (requiring large control signals) and minimum control effort (requiring smooth paths, that may deviate from desired path). In keeping with this simple concept, the user interface only includes sliders to choose the relative weighting between smoothness and accuracy.

Currently a two axis controller, that runs at approximately 1.7KHz, is supported by the system. Some applications require control loop frequencies around 4KHz [DT97]. Improvements in our driver infrastructure will enable our system to run at those frequencies.

3. Conclusion and Future Work

In this project we explored the application of LabVIEW RT for embedded software development. We showed that G, the underlying model of computation, satisfies the requirements for specifying embedded systems, under certain simple restrictions.

We developed an embedded motion control system as an application of LabVIEW RT for embedded design. The main advantage that LabVIEW RT presented was the reduction of the development cycle time, because of the common simulation and execution environment. Another important advantage was that G allowed us to easily combine different models of computation.

The implemented motion control system includes automatic plant modeling (system identification), MIMO control capabilities, and an innovative qualitative optimal control design program. Moreover, our architecture implements HIL techniques for replacing real plants with software simulated plants. The system is also very flexible, allowing the end user to access and replace control routines.

This work can be extended to include improvements in the qualitative control design methodology, that will allow for more generic control algorithm specifications. Moreover, the current system identification can be extended to identify systems when there are coupled axis.

4. References

[AK98] H. Andrade and S. Kovner, "Software Synthesis from Dataflow Models for Embedded Software Design in the G Programming Language and the LabVIEW Development Environment", *Proc. IEEE Asilomar Conference on Signals, Systems, and Computers*, Nov. 1998, pp. 1705-1710.

[CL90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw Hill Pub. Co., ISBN 0070131430, 1990.

[DT97] K. Dutton, S. Thompson, and B. Barraclough, *The Art of Control Engineering*, Addision-Wesley Longman, ISBN 0201175452, 1997.

[GL99] A. Girault, B. Lee, and E. A. Lee, "Hierarchical Finite State Machines with Multiple Concurrency Models", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, June 1999, pp 742-760.

[HL99] S. Han, M. Lee, and R.R Mohler, "Real-time implementation of a robust adaptive controller for a robotic manipulator based on digital signal processors", *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 29, no. 2, March 1999, pp 194-204.

[LM87] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing", *IEEE Transactions on Computers*, vol. C-36, no. 2, Feb. 1987.

[P95] T. M. Parks, "Bounded Scheduling of Process Networks", Technical Report UCB/ERL-95-105, University of California at Berkeley, Berkley, CA 94720, Dec. 1995.

[NI99] *Flex Motion Software Reference Manual,National Instruments, 1999 (www.ni.com/support).*