Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Design of Hybrid Systems With Real-Time and FPGA Targets

Jeanne Sullivan Falcon*

**National Instruments, Austin, TX 78759*
USA (Tel: 512-683-6618; e-mail: Jeannie.falcon@ni.com).

Abstract: Hybrid control systems integrate discrete logic combined with physical system dynamics. Discrete logic can be represented in a graphical, data flow language using a UML compliant statechart model. The physical system dynamics can also be integrated in the hybrid system by using the same data flow language. The resulting system can execute on a real-time processor or a system which includes both a processor and a field programmable gate array (FPGA). A case study is presented for a cruise control application.

1. INTRODUCTION TO HYBRID CONTROL SYSTEMS

Hybrid control systems integrate discrete logic combined with physical system dynamics. Antsaklis and Koutsoukos (Antsaklis *et. al.*, 2002) discuss the use of hybrid control systems in manufacturing, communication networks, auto-pilot design, engine control, traffic control and chemical process control. They also discuss how hybrid control is employed in a wide variety of embedded control systems that interacts with the physical world. Hespanha and Liberzon (Hespanha *et. al.*) also examined applications of hybrid control systems and focused on reconfigurable systems, fault correction systems, and certain classes of parameter-adaptive systems.

A simple example of a hybrid control system is given in (Antsaklis *et. al.*, 2002) as a manufacturing plant with multiple machines connected via an automated material handling system. Parts are processed on separate machines but the processing only begins when digital sensors indicate the delivery of the parts to the machines. So, the complete system is described by combination of the event-driven dynamics of parts moving between machines and the time-driven dynamics of the processes within the machines.

Discrete logic can be represented using a graphical approach to event based programming called a statechart. The physical system dynamics can be fully described using a graphical simulation diagram. The combination of these two computational models results in a complete hybrid control system simulation or real-time implementation.

2. STATECHART REPRESENTATION OF DISCRETE LOGIC OR CONTROL

Statecharts were proposed by David Harel (1987) as a more advanced method for describing reactive systems and state machines than state transition diagrams or “state diagrams.” State diagrams are directed graphs containing nodes for states and arrows for transitions between states. The transition arrows are labeled with triggering events and guarding conditions. A drawback of state diagrams they do not incorporate hierarchy or modularity. Thus, they can become quite large and unmanageable for complex systems.

Harel (1987) proposed statecharts as a means to simplify the visual description of complex discrete event systems. The statechart approach developed by Harel allows for modularity, clustering, orthogonality of states (or concurrency) and refinement. It is also possible to “zoom” within a statechart to explore different levels of abstraction.

The statechart concept for discrete event systems was incorporated into the Unified Modeling Language (UML) specification developed by the Object Management Group (OMG) consortium (2007). The OMG consortium is an international, open membership, not-for-profit computer industry consortium. The LabVIEW Statechart Module is compatible with the UML specification for statecharts (OMG 2007).

A program created with one or more statecharts can run on desktop PCs running Windows or it can run natively on desktops converted into dedicated real-time targets. In addition, the statechart can run on a wide variety of embedded targets including distributed programmable automation controllers and custom microprocessor based designs through C code generation. In addition, statecharts can run on field programmable gate array (FPGA) targets such as PXI/PCI FPGA devices or on the FPGA within a distributed programmable automation controller.

3. DYNAMIC SYSTEM SIMULATION INTEGRATION WITH GRAPHICAL PROGRAMMING

The LabVIEW Control Design and Simulation Module includes tools for linear systems analysis, control system and estimator design, numerical simulation, and control implementation (National Instruments, 2007).

The module allows continuous-time or discrete-time systems to be represented in standard block diagram form. Both feedback and feedforward signal paths can be represented with arrows on the wires within the block diagram. Hierarchy in the block diagram may be used to represent the dynamics of different subsystems. Also, the same simulation program that has been developed for offline simulation may be easily configured as a real-time simulation or control system. Parallel simulation loops, with adjustable execution priority, may also be employed to create multi-rate systems. Additionally, textual programming may be incorporated into the simulation loop directly so that a combined textual and graphical approach may be used to describe the physical system. Finally, a simulation loop may be placed within a larger graphical program. This capability facilitates the creation of a hybrid or a complex intelligent system.

4. CASE STUDY: CRUISE CONTROL SIMULATION

The front panel for a Cruise Control User Interface program written in LabVIEW is shown in Fig. 1. This example allows the user to test drive a hybrid control system simulation that combines discrete logic for the user interface with a continuous-time simulation of the vehicle dynamics, road profile and PID cruise control system. When the user runs the program, the vehicle can be driven manually with desired acceleration and braking commanded using sliders on the front panel. As the vehicle goes over the simulated road, the speed of the vehicle is shown slowing down when it goes up an incline and then speeding up when it goes downhill.

To activate the cruise control, the user can click the *on* button in the cruise control section of the front panel. This will set the cruise control state to *idle*. Then, the user can click on the *set* button to provide the setpoint speed for the cruise control system. The accelerator slider will then begin moving on its own to indicate the control action of the cruise control system. There will be less variation of speed than the manual control system (the driver) as the vehicle goes uphill and downhill. If the user wants to disable the cruise control system and switch to manual control, he or she can press the *on* button again to turn the cruise control off. The user can press the *resume* button in the cruise control section of the front panel to bring the cruise control immediately back to the last setpoint for speed and to the *on* state.

The screenshot in Fig. 2 shows the top-level block diagram for the Cruise Control User Interface program. It shows a continuous-time simulation loop with both hierarchy for different physical subsystems (cruise control, road, and vehicle) as well as the Run Statechart program to call the Cruise Control Logic.

The Cruise Control Logic statechart diagram can be viewed by right-clicking on the Run Statechart program and then selecting the "Open Statechart" option. The screenshot in Fig. 3 shows the statechart diagram for this example. It includes an Initial state as well as four other states: Off, Idle, On, and Set.

5. HYBRID SYSTEM DEPLOYMENT

An FPGA may be used in conjunction with a microprocessor to form a heterogenous computational device. Discrete logic tasks that must run at high speeds may be described by statecharts and deployed to the FPGA.

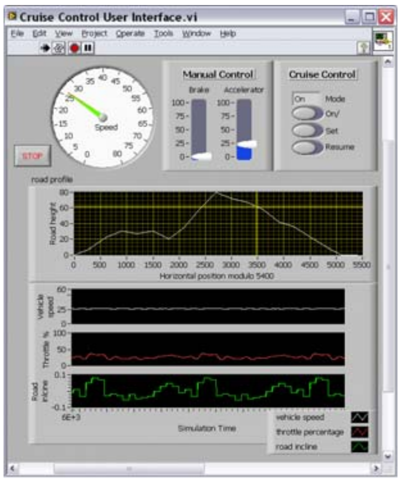

Fig. 1: Front panel for a Cruise Control program

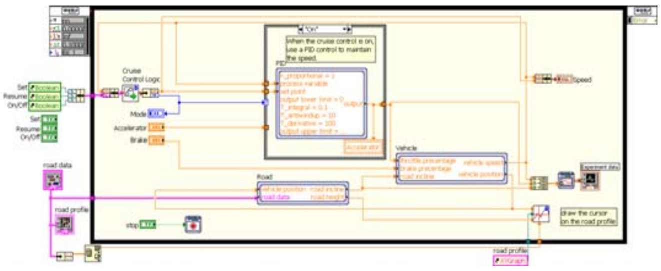
Cruise Control Logic
Mode
Accelerator
Brake
Set
Resume
On/Off
When the cruise control is on, use a PID control to maintain the speed.
Vehicle
Road
road data
road profile
stop
Speed
draw the cursor on the road profile

Fig. 2: Top level block diagram for Cruise Control

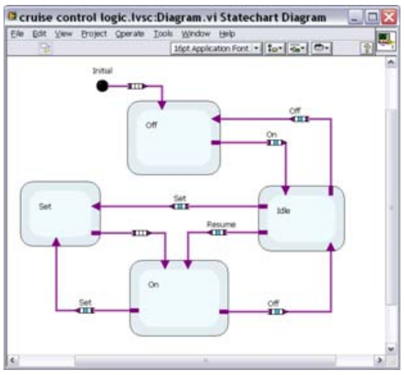

Fig. 3: Statechart used for Cruise Control program.

REFERENCES

Antsaklis, P. and X. Koutsoukos, X. (2002). "Hybrid Systems Control," http://www.nd.edu/~pantsakl/elecpubs/256.pdf

Douglass, B. (1999). "UML Statecharts," *Embedded.com*, http://www.embedded.com/1999/9901/9901feat1.htm

Harel, D. (1987). "Statecharts: a visual formalism for complex systems," *Science of Computer Programming.* Vol. 8, p. 231.

Hespanha, J., and Liberzon, D., Abstract for "Tutorial On Logic-Based Control: Switched Control Systems," http://med.ee.nd.edu/MED10/pdf/abstract.pdf

Object Management Group (2007). "Unified Modeling Language™ - UML® Resource Page" http://www.uml.org/

National Instruments (2007). "Embedded System Design," http://www.ni.com/embedded