Hardware and Software Considerations for Implementing Hardware-in-the-Loop Traffic Simulation

Richard B. Wells, MRCI/NIATT
John Fisher, MRCI
Ying Zhou, MRCI
Brian K. Johnson, MRCI/NIATT
Michael Kyte, NIATT
University of Idaho
Moscow, ID 83844-0901
USA
rwells@ece.uidaho.edu
jcf@ieee.org
b.k.johnson@ieee.org
mkyte@uidaho.edu

Abstract – **Digital computer simulation provides an important tool for the study of complex systems. When the complexity of the problem is too large to warrant an analytical solution, simulation is the only option to analyze system configurations or operational modes prior to their implementation in the field. Not all components in a complex system can be modelled in adequate detail in computer simulations: for example, only simple, generic models of traffic controllers are available. Real-time hardware-in-the-loop simulation allows real traffic controllers to interact with computer simulations to improve accuracy. This paper provides an overview of real-time simulation and then discusses hardware and software constraints to implementing a controller interface device (the NIATT CID II) for real-time hardware-in-the-loop simulation.**

I. INTRODUCTION

Digital computer simulation provides an important tool for the study of complex systems. When the complexity of the problem is too large to warrant an analytical solution, simulation is the only option to analyze system configurations or operational modes prior to implementation in the field. Dramatic improvements in computer equipment have provided the ability to model large systems with relatively short simulation run-times, with some classes of simulation running in less time than the time period they are simulating.

However, not all components in a complex system can be modelled in adequate detail in computer simulations. This is the result of increasing complexity of devices such as traffic controllers or protective relays in power systems. These devices are often small embedded computer systems that are capable of performing multiple functions, and are often user programmable.

Although the basic algorithms used are often well known, the details of the implementation in the controller can have significant impact in the response to changes in systems conditions. These manufacturer-specific details are generally proprietary and are thus not available to incorporate into the models available in computer simulation. As a result, the computer simulation may use generic models to represent the controller. These are adequate for developing a general simulation of a system, but there can be significant errors when compared to field results.

Real-time hardware-in-the-loop simulation uses a computer simulation to model the bulk of the system. However, the control hardware is interfaced to the computer simulation and receives inputs from the simulation, makes control decisions based on those inputs, and sends commands to activate device models in the simulation. There are several key challenges to implementing successful hardware-in-the-loop simulation in a cost-effective manner. In general, either a specialized computer is needed to run the simulation and interface to the traffic controller, or an auxiliary interface device is needed between the computer and the control device [1]. It is also possible to create a network of control devices connected to single computer simulation [2].

This paper provides an overview of real-time simulation and then discusses hardware and software constraints to implementing a controller interface device (NIATT CID II) for real-time hardware-in-the-loop simulation. This is followed with a description of a specific controller interface device.

II. REAL-TIME SIMULATION

Simulation tools may be classified as ‘real-time’ or ‘off-line’. Real-time simulation tools generate results in synchronism with a real-time clock. This means that calculations for each simulation timestep have to be completed within the same corresponding interval of real-world time. Real-time simulation tools can be based on physical analog models or they can be implemented digitally, often with special purpose computer architectures. Off-line microscopic traffic simulation packages include: TSIS/CORSIM [3], VISSIM [4], SimTraffic [5] and Paramics [6]. Each of these tools has models to represent the traffic system components, which can then be assembled into a system model. These tools are the end result of years of development. Among the models available are generic models for traffic controllers that can be set up to implement specific signal plans. Each tool also has a graphical user interface to allow easy display of the simulation results. The solution engine in each tool allows relatively large traffic networks to be modeled with fixed simulation time-steps generally ranging from 100 milliseconds to 1 second.

Real-time simulation of a traffic network or any other type of system is only necessary if there is a need for the simulation to coordinate with something in the real world. Real-time simulation tools are generally used for some form of hardware-in-the-loop simulation, where some form of control hardware such as a traffic controller or a protective relay for an electric power system is interfaced to the computer to interact with the computer simulation. The control hardware takes measurements of operating conditions from the computer simulation, acts on that information as it would actual measurements in the field and then sends a control signal back to the computer simulation (in place of

controlling hardware in the field).

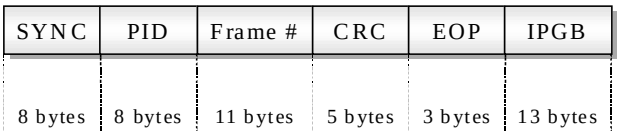
Timing is a critical issue in real-time hardware-in-the-loop simulation. First, the computer simulation must be able to consistently run at real-time speeds, i.e. it should take 10 seconds to simulate 10 seconds of system time. If the simulation takes 4 hours to simulate 10 seconds of system time, it will not be possible. The operating systems commonly used to run the simulation packages can create difficulties in this regard. If the computer system is loaded heavily performing several tasks in parallel, the simulation could take significantly longer to run. In some cases, special purpose computing platforms with limited real-time operating systems (non-multitasking operating systems) will be required. In addition, if the simulation runs faster than real-time, the simulation must be slowed down to run in real time.

Second, the communication between the computer running the simulation and control device must be fast enough to allow real-time simulation to be maintained. In many cases, a special interface will be needed between the computer and the control device since the control device may not use communication protocols that are compatible with standard computer interfaces. This interface device will communicate with the controller through the controller's normal cable interface. This side of the communication will generally be sufficient for real-time simulation since the control device normally communicates this way in the field. A key concern is achieving sufficient data transfer rates between the computer and the interface device. There are many high-speed data communication protocols available, but, if possible this communication should take place using standard PC communication instead of requiring additional hardware for the PC as well (adding a second interface). For example, most newer PC's are shipped with Universal Serial Bus (USB) ports. If USB communication is capable of performing real-time simulation, this would preclude the need to add capability to use faster industrial communication protocols. This communication can create a significant bottleneck, especially in cases where a large number of control devices are to be connected to one computer simulation.

III. HARDWARE CONSTRAINTS

Hardware-in-the-loop operation imposes the constraint of real-time data processing on the system. This has in its turn constraints that must be met by the system, which affect or are affected by the computer's operating system (OS), the CID II's embedded controller, and the simulation network. In this section we briefly describe these considerations.

The first hardware constraint imposed on the system arises in conjunction with the OS. Ideally a real-time processing task is best suited to run under a real-time-executive (RTE) operating system. RTE's differ from conventional multi-tasking OS's in terms of scheduling algorithms [7], synchronization of various system resources [8], and consistency control schemes for memory resources accessed by concurrent processes [9]. In contrast, a conventional OS, e.g. Windows, lacks real-time support mechanisms. For example, scheduling is often carried out using a run-time slice algorithm, which simply assigns to each executing process some particular average percentage of time. It thus falls on the real-time application to manage its own real-time characteristics and, additionally, frequently places constraints on the number of active applications multi-tasked "concurrently" in the system. In some cases of older OS's (e.g. Windows 3.1), multitasking is "cooperative," a term that means the application program itself is required to check a message queue and "voluntarily" give up return control of the system to the OS. "Uncooperative" applications could prevent the user from switching applications or even from accessing the system itself. In the application described in this paper, responsibility for software management rests with the CORSIM program itself and the user is required to restrict the total application task load and/or properly assign execution priorities among multiple tasks.



SYNC: sync field
PID: packet identifier
Frame #: frame number
CRC: cyclic redundancy check for the frame number
EOP: end of packet
IPGB: inter-packet guard band

Fig. 1: USB Start of Frame Packet

Timing consistency is crucial to the accuracy of the hardware-in-loop simulation. In particular, the fundamental time reference is maintained by the CORSIM software, and CID II timing must remain synchronized to this reference. To do this, a periodic timing signal must be transmitted from the PC to the CID II network. In our application, this task is accomplished as part of the CID II network communication protocol. Providing a consistent timing reference for the system favors the use of a synchronous communication protocol (whereas most computer peripherals employ an asynchronous parallel protocol, e.g. IEEE 1284 or IEEE 488, or an asynchronous serial protocol, i.e. RS-232) [10]. In synchronous protocols, the unit of information transfer is called a frame. At the physical layer, the interface hardware provides a SP with frame synchronization and error control. For this application, since it was desirable to select a standard protocol, we use the Universal Serial Bus (USB) protocol [11].

USB provides a 12 Mbps serial channel divided into frames of 1 msec each. Frame synchronization is provided by a 48-bit start of frame (SOF) packet broadcast from the PC once per msec. The USB SOF packet structure is shown in Fig. 1. The CID II USB interface hardware detects SOF, and SOF detect is used to establish the fundamental timing reference within the CID II. All CID II operations are constrained to execute within a time interval of 1 msec and then wait for the next SOF detect.

Link management and flow control are provided in the second layer of the interface protocol (data link layer). We

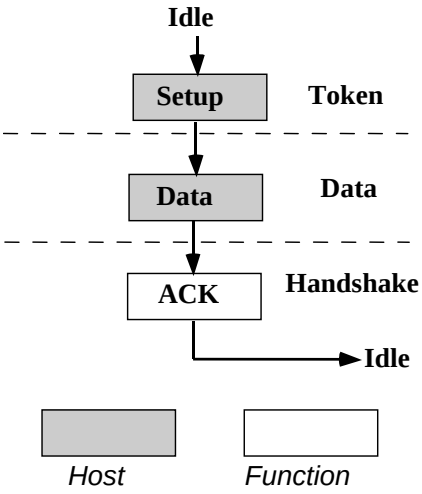


Fig: 2: Control Transfer SETUP Transaction

use two of the four available transfer types available in USB. The control transfer is required of all USB devices and is executed when a peripheral is attached to the USB. This transfer is used to setup and identify the USB peripheral to the PC and establish its operating characteristics. It consists of a setup transaction, a data transaction, and an acknowledgement transaction as shown in Fig. 2.

Actual hardware-in-the-loop simulation operation uses USB's isochronous transfer mode [11]. This transfer mode guarantees bounded data transfer latency, guaranteed bus access, lower transaction overhead in the frame, and can be implemented with relatively simple firmware in the CID II controller. Two frame formats are employed in our protocol, an Out Format (OUT) for data transfers from the PC to the CID II and an In Format (IN) for data transfers from the CID II to the PC.

The OUT format is shown in Fig. 3. Each CID II is allocated 73 bytes of data plus 10 bytes of transaction overhead. The transaction overhead consists of 2 bytes of sync, 2 bytes of packet ID (PID), 2 bytes for device addressing and a CRC5 check field, 2 bytes for a CRC16

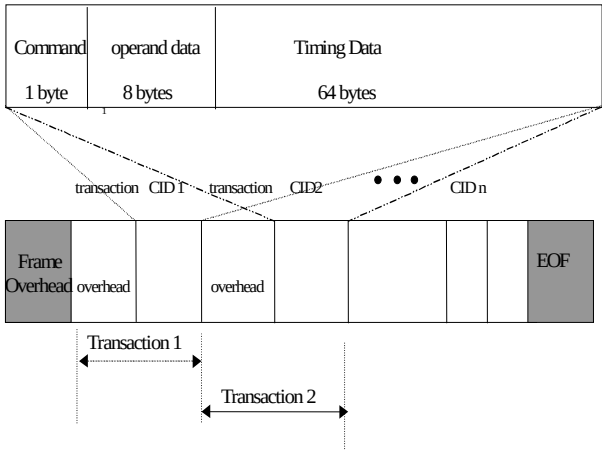


Fig. 3: USB Frame Structure (OUT) for CID II Application

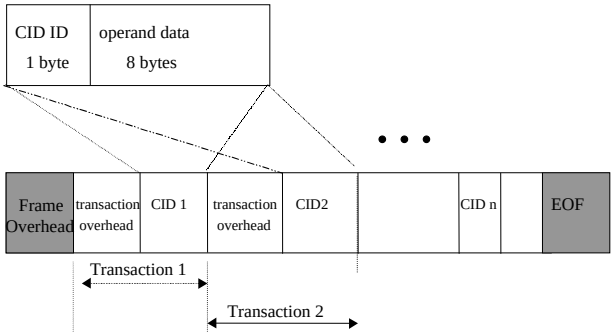


Fig. 4: USB Frame Structure (IN) for CID II Application

data check field, and 2 bytes of guard interval between transaction packets. The data packet, shown in Fig. 3, includes a command op code for the CID II, an 8 byte operand field, and a 64 byte traffic sensor timing data field. These last two fields contain data from CORSIM that determine the traffic sensor input signals CID II supplies to the traffic controller (TC). These fields merit some explanation, which will be provided below after we discuss the IN field. The IN format is shown in Fig. 4. This format is used to return phase information from the TC to the CORSIM program.

Referring again to Fig. 3, the CID II simulates the binary-valued inputs for up to 64 traffic sensors to the TC. The time-width of each sensor signal is specified by CORSIM with 1 msec timing resolution. Specification of the pulse width requires 2 bytes of timing data, providing a maximum pulse width of 65.535 seconds. With a 64 byte field available to each CID II, our protocol allows new sensor commands to be issued for only up to 32 sensors during any one USB frame. This is acceptable because it is extremely atypical for a CORSIM simulation to require changes to one-half or more of a TC's sensor signals at any one time. The 8 byte operand field is used to specify which sensor outputs are receiving new actuation data in an OUT transaction. Timing data is packed into the timing data field such that the lowest numbered activated sensor receives the first 2 bytes of timing data, the next lowest sensor receives the next 2 bytes, etc.

When a sensor is activated, CID II stores the commanded pulse width for that sensor in internal memory in a "milliseconds remaining" (MSR) format. At every SOF detect, every non-zero MSR is decremented; the sensor output to the TC is de-asserted when this count reaches zero.

Using this protocol scheme, a large network of controllers can be accommodated using a dedicated USB network, as illustrated in Fig. 5. The principal limitation on the size of the network will be set by the execution-time required by the CORSIM software in its simulation computations and managing the data buffering to and from the CID II network.

One last hardware constraint is worthy of note. Using real-time simulation involving actual TC's inevitably means that hardware-in-loop simulations require a total simulation time equivalent to the real traffic flow time in the field. Consequently, it is conceivable that these simulations may take several hours to complete for complex traffic corridors. This brings with it a requirement that the CID II be robust in

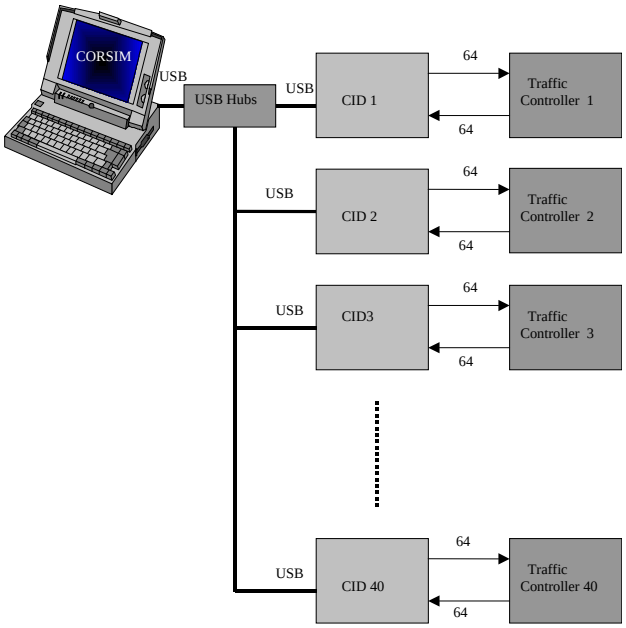


Fig. 5: NIATT CID II System Topology

the face of short-term power line dropouts. The CID II power supply has been designed to be able to withstand a line dropout of up to 1 second in duration, which is comparable to the ability of a typical PC's line cycle dropout immunity. The largest power dissipation element of the CID II is its LED display panel, which provides the operator with visual feedback of the progress of the simulation. The CID II includes in its design a power dropout detection circuit that the CID II's microcontroller monitors. When a power dropout is detected, the CID II conserves power by turning off all its LED indicators, which nearly doubles the dropout immunity of the CID II for the same operating efficiency in its power supply.

IV. SOFTWARE MODIFICATIONS

There several software modifications that are also required to run hardware-in-the-loop simulation. First of all, there has to be some means for pulling data from the simulation as it runs and simultaneously writing the response of the traffic controller into the simulation as commands to the simulated

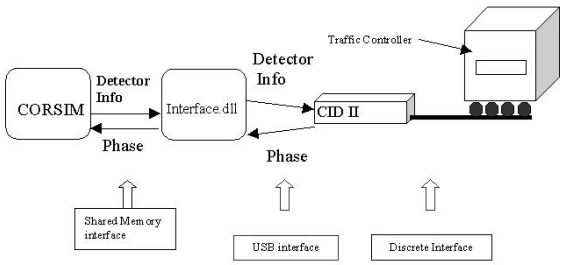


Fig. 6: Software interface between the CID II and CORSIM

traffic signals. In addition, the simulation data file needs to be modified to identify which intersections will access the CID II to communicate with the physical traffic controller rather than the simulated traffic controller. The user could modify the file by hand; however, it is more efficient to develop a graphical interface to simplify this task for the program user.

CORSIM exchanges data via a shared memory. To perform real-time simulation, the CORSIM application is modified to call two dynamic link library (DLL) modules (CORSIM.dll and interface.dll). These two dynamic link library modules exchange data with the CID by using the shared memory structure. Detector states are updated by CORSIM, and the interface.dll reads those detector states and sends them to the CID II via the USB interface. Similarly, the phase states that the interface.dll reads via the USB interface are updated in the shared memory structure so CORSIM can act on them.

Finally, a software driver is needed to interface the OS with the USB bus. Operating systems that bundle systems with a USB driver will meet the minimum requirements described in section III. In addition, microcontroller vendors who sell USB capable microcontrollers distribute their own drivers. Fig. 6 shows the role of the software in the CID II.

The controller interface device also enables the creation of additional software tools beyond the hardware-in-the-loop simulation. A software version of a suitcase tester can also be implemented. The suitcase tester software application allows the user the ability to test each traffic controller input and verify its response to a variety of programmed conditions.

V. HARDWARE IMPLEMENTATION

A. System Overview

The CID II is a microcontroller-based system that connects to a PC using the Universal Serial Bus (USB) interface. Physically, a motherboard/daughterboard arrangement is used to facilitate easy assembly and part replacement. Seven daughterboards are used: microcontroller, display, CID II input (2), CID II output (2), and power supply.

Because of the large number of signals being routed and the spacing limitations of the target manufacturing process, discrete logic chips were used for all digital logic functions instead of programmable logic.

A PC running Microsoft Windows can perform send and receive operations for up to 40 CID IIs in one second. Thus, a CORSIM simulation with a 1 Hz update rate could employ 40 CID II's as shown in Fig. 5.

B. Microcontroller

The microcontroller used is Cypress Semiconductors' popular EZ-USB 8051. This processor runs at 24 MHz with a four-clock instruction cycle, and contains approximately 6 KB of available on-chip RAM in our configuration. Code is stored on an external 16 KB EPROM.

Communication with the input and output boards is accomplished over an 8-bit data bus and enable/disable lines. Both input boards and both output boards are identical; the

motherboard routes the proper enable/disable lines to the proper boards.

C. Traffic Controller Interface

The CID II uses discrete connectors and custom cables to interface with a variety of traffic controller models with up to 64 inputs and 64 outputs. Traffic controller inputs are read every 1 ms, when the USB start-of-frame interrupt is received.

Traffic controller inputs are set when the CID II receives new controller input data from the PC. Each set of input data tells what bits should be turned on/off, as well as the duration for which a given data bit is on or off. Including timing data allows the PC to send data at a manageable rate, and keeps pulse lengths deterministic.

D. PC Interface

The CID II communicates with the PC over the USB interface. The USB specification defines several communication modes. The CID II uses the "isochronous" mode, designed for data that is time-critical but may be lost without catastrophe (e.g. digital audio data to a USB speaker).

The CID II can receive a command from the PC every 1 ms. Traffic controller outputs are read every 1 ms and sent to the PC to be read at its leisure.

The prevent confusion in a multiple-CID II environment, data sent to the PC is tagged with a number identifying the CID II that sent the data. This number is set with a bank of printed circuit board mounted dual inline package (DIP) IC switches.

E. Human Interface

To facilitate debugging, a display is included. 128 LEDs show the status of all inputs and outputs. Two 7-segment displays show the CID II identification number. A self-test button is also provided.

F. Self-test Function

The self-test feature uses a loopback cable to connect each CID II input to the correspondingly-numbered CID II output. The test first turns on all the display LEDs for visual inspection. It then performs a set of input/output checks to find faulty inputs or outputs. After the test is completed, the position of faulty I/O pairs is displayed on the display LEDs.

VI. CONCLUSION

Digital computer simulation provides an important tool for the study of complex systems. When the complexity of the problem is too large to warrant an analytical solution, simulation is the only option to analyze system configurations or operational modes prior to their implementation in the field. Not all components in a complex system can be modelled in adequate detail in computer simulations, for example only simple, generic models of traffic controllers are available. Real-time hardware-in-the-loop simulation allows real traffic controllers to interact with computer simulations to improve accuracy. This paper provides an overview of real-time simulation and then discusses hardware and software constraints to implementing a controller interface device (the NIATT CID II) for real-time hardware-in-the-loop simulation.

VII. ACKNOWLEDGMENT

The authors gratefully acknowledge the contributions from Zhen Li, Darcy Bullock, Peter Kohl, Jeremiah J. Remus, James Richards, Eugene Bordenkircher, Cody Miller, Kenton Veeder, Tricia Veeder, Ivan Anderson, Dan Gordon, Darin McKee, Geoff Biedler, and Mike Adams.

U.S. Department of Transportation University Transportation Centers Grant DTRS98-G-0027 supported Work on this project. This work was performed through the University of Idaho National Institute for Advanced Transportation Technology and the University of Idaho Microelectronics Research and Communications Institute.

VIII. REFERENCES

[1] Bullock, D., T. Urbanik, "Hardware-In-The-Loop Evaluation of Traffic Signal Systems," *Proceedings of the 2000 IEE Conference on Road Transport Information and Control*. April 4-6, 2000.

[2] Engelbrecht, R., C. Poe, and K. Balke, 1999, "Development of a Distributed Hardware-In-The-Loop Simulation System for Transportation Networks," *Transportation Research Board Annual Meeting*. National Research Council, Washington, DC, Preprint #990599.

[3] ITT Systems & Sciences Corporation. *CORSIM User's Manua*l. Version 1.04. FHWA, U.S. Department of Transportation, March 1998.

[4] PTV Planung Transport Verkehr AG. *VISSIM User Manua*l. Version 3.00. Karlsruhe, Germany, Jan. 2000.

[5] Trafficware. *SimTraffic User's Guid*e. Berkeley, CA, 1998.

[6] Quadstone Limited. *Paramics System Overview*, Edinburgh Scotland

[7] K. Ramamritham and J.A. Stankovic, "Scheduling algorithms and operating systems support for real-time systems," *Proc. IEEE*, vol. 82, no. 1, Jan., 1994, pp. 55-67.

[8] N. Suri, M.M. Hugue, and C.J. Walter, "Synchronization issues in real-time systems," *Proc. IEEE*, vol. 82, no. 1, Jan., 1994, pp. 41-54.

[9] P.S. Yu, K.-L. Wu, K.-J. Lin and S.H. Son, "On real-time databases: concurrency control and scheduling," *Proc. IEEE*, vol. 82, no. 1, Jan., 1994, pp. 140-157.

[10] *Reference Data for Engineers: Radio, Electronics, Computer, and Communications*, 7th ed., E.C. Jordan (ed.), Indianapolis, IN: Howard W. Sams, 1986, Chap. 26, pp. 10-19.

[11] Compaq, Digital Equipment Corp., IBM PC Co., Intel, Microsoft, NEC, and Northern Telecom, "USB Specification 1.0," rev. 1.0, Jan. 19, 1996, http://www.usb.org