Controller Interface Device Software Documentation

Final Report

Semptember 2006

KLK-236

NIATT Report Number N06-22

Prepared for

OFFICE OF UNIVERSITY RESEARCH AND EDUCATION

U.S. DEPARTMENT OF TRANSPORTATION

Prepared by

National Institute for Advanced Transportation Technology

University of Idaho

Brian Johnson, Richard W. Wall, Michael Kyte, Darcy Bullock, Zhen Li, Eugene Bordenkircher, Sanjeev Giri

Table of Contents

INTRODUCTION

This report is divided into three main sections. The first section describes the CID software suite documentation system. The second section contains a paper presented at the 2004 IEEE Intelligent Transportation Systems Conference describing the development of software interfaces between the CID and CORSIM, VISSIM and Simtraffic. An interface to Paramics was also developed after that paper was presented. The third main section presents the development of an automated version of the suitcase tester. Papers describing this were presented at the 2005 Transportation Research Board meeting and the 2006 IEEE Intelligent Transportation Systems Conference. The paper from the 2006 conference is included since it has more detail about the software development.

In addition to the activities described above, there were some minor changes made to the USB descriptor tables. The original tables worked adequately with Windows 98, 2000, and ME. However, Windows XP performs stricter checking and there were some minor errors in the tables. Those were corrected. In addition, there was a modification to the USB driver to increase the number of CID's allowed in one simulation

PART 1: CID SOFTWARE DOCUMENTATION

The CID software documentation was done primarily using Doxygen, a free software documentation system that can be downloaded at http://www.doxygen.org.

About Doxygen

Doxygen is developed under Linux and Mac OS X, but is set-up to be highly portable. As a result, it runs on most other Unix flavors as well as Windows.

Doxygen is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), and to some extent PHP, C#, and D.

1. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.
2. A user can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. You can also visualize the relations between the various elements by means of including dependency

graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

3. Doxygen can be abused for creating normal documentation.

Doxygen in CID Software Documentation

The source code for each of the main CID software tools were run through Doxygen, producing a HTML document of each of the tools. The information in these web pages is not suitable for meaningful presentation in a report format. Instead, they are all available through a web page: http://www.ece.uidaho.edu/ee/power/brian/CIDSoft/.

A sample view of the main window is shown in Figure 1.

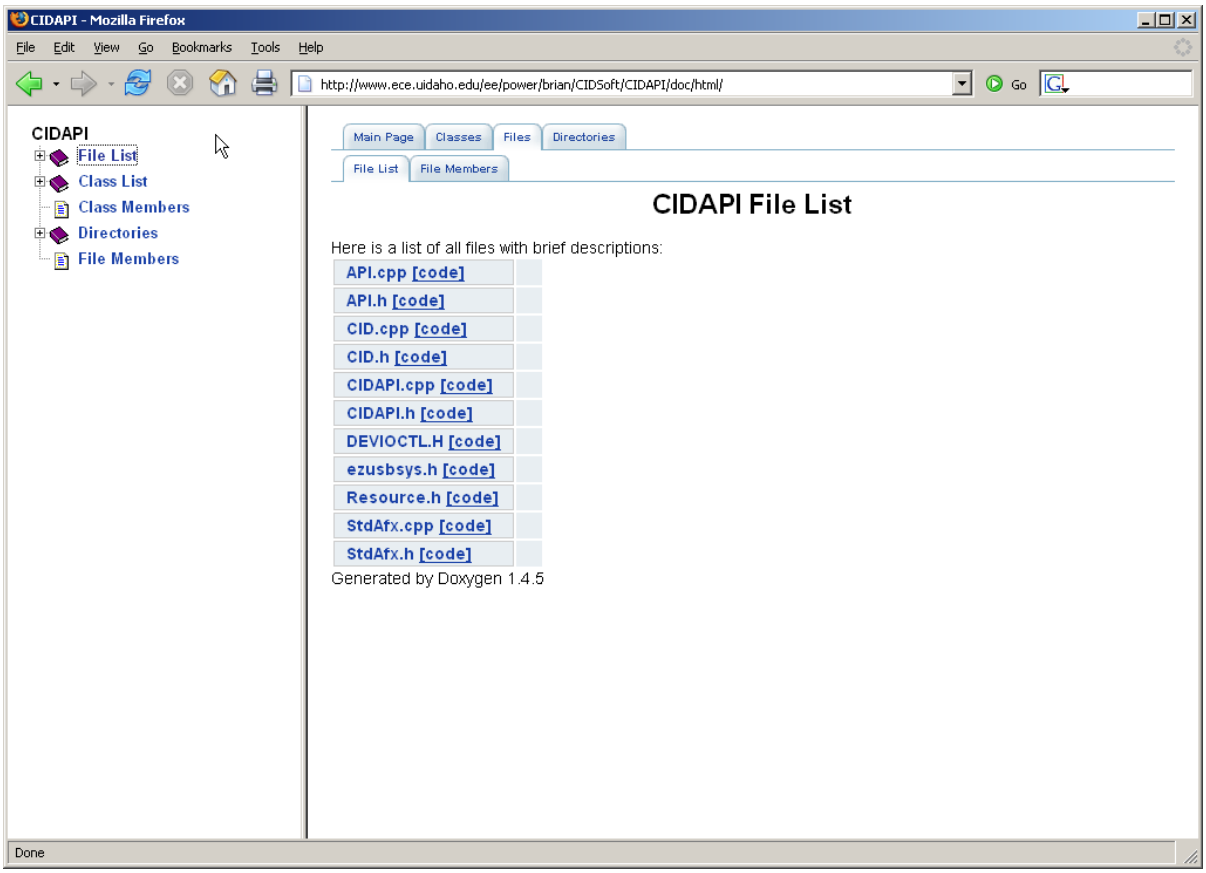

Figure 1: View of main web page for the CID API, with the file list tab selected.

The following software tools were documented in this fashion:

1. CID API
2. Corsim Datafile Generator
3. Corsim Interface
4. Corsim Version 5.1 Interface
5. VISSIM Interface
6. VISSIM Configuration Tool
7. SimTraffic Interface
8. Hardware tester
9. Suitcase tester
10. Paramics Configuration Tool
11. Paramics Interface

Since the automated suitcase tester described in part 3 of this document was not finalized at the time the report was written, it is not available in this section at present.

Unified Modeling Language and Diagrams

However, Doxygen does not have a very effective tool for creating Unified Modeling Language (UML) diagrams. As a result, these diagrams were made separately.

Introduction:

The UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction. There are various types of UML diagrams among which class diagram is the most widely used perspective. Class diagrams are used in nearly all Object Oriented software designs.

Software Tool:

For this project, IBM's Rational Rose Software Architect version 6.0 was used to generate the class diagrams.

Table 1: SDLC Projects and Their Class Diagram Files

CID projects	Class Diagram file
CIDAPI	CIDAPI.pdf
CID Hardware Tester	HardwareTester.pdf
CID Suitcase Tester	SuitCaseTester.pdf
CORSIM (pre-version 5.1) Interface	Interface.pdf
CORSIM version 5.1 or higher Interface	Interface51.pdf
CORSIM Generator	Generator.pdf
CID VISSIM Configuration Tool	VISSIM-Configuraton-Tool.pdf
VISSIM File Generator	VISSIM.pdf
CID SimTraffic Interface	SimTrafficCID.pdf
CID Paramics Interface	Paramics-InterfaceSoftare.pdf
CID Paramics Configuration Tool	Paramics-Configuration-Tool.pdf

Class Diagrams:

Classes are composed of three things: a name, attributes, and operations. Below is an example of a class.

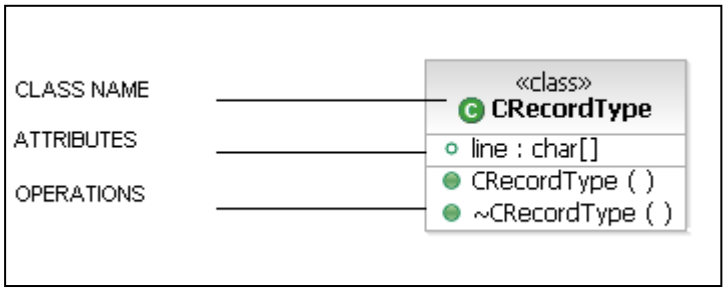

Figure 1.0: Example of a class diagram.

Class Attribute/Operation Symbols:

The symbols before the attribute or an operation indicate if the attribute/operation is public, protected, or private. Following is a diagram that shows the relationship between the attribute/operation symbols and type.

Visibility level	Icon for attribute	Icon for operation	Text symbol	Description
Private			-	Only classes in the same container can see and use the classes.
Protected			#	Only classes in the same container or a descendent of the container can see and use the classes.
Public			+	Any class that can see the container can also see and use the classes.
Package			~	Only classes within the same package as the container can see and use the classes.

Figure 1.1.1: Symbols and visibility level.

Class Relationships:

Class diagrams also display relationships such as containment, inheritance, associations, and others.

Association relationships

In UML visualization class diagrams, an association is a structural relationship that indicates that objects of one classifier (such as a class and interface) are connected and can navigate to objects of another classifier. Following is an example showing association relationship between two classes.

Java source code

OwnedClass.Java:

```
public class OwnedClass {

      // <<class body>>
}
```

OwnerClass.Java:

```
public class OwnerClass {
private OwnedClass associatedClass;
public OwnerClass otherClass;

      // <<class body>>
}
```

UML visualization

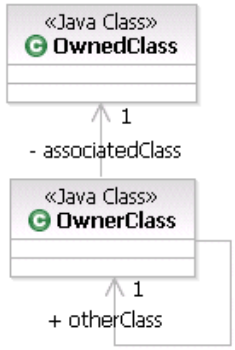

Figure 1.2.1: Example of an association relationship.

Dependency relationships

In UML visualization class diagrams, a dependency relationship indicates that a change to one class (the supplier) might cause a change in the other class (the client). The

supplier is independent because a change in the consumer does not affect the supplier. Following is an example showing dependency relationship between two classes.

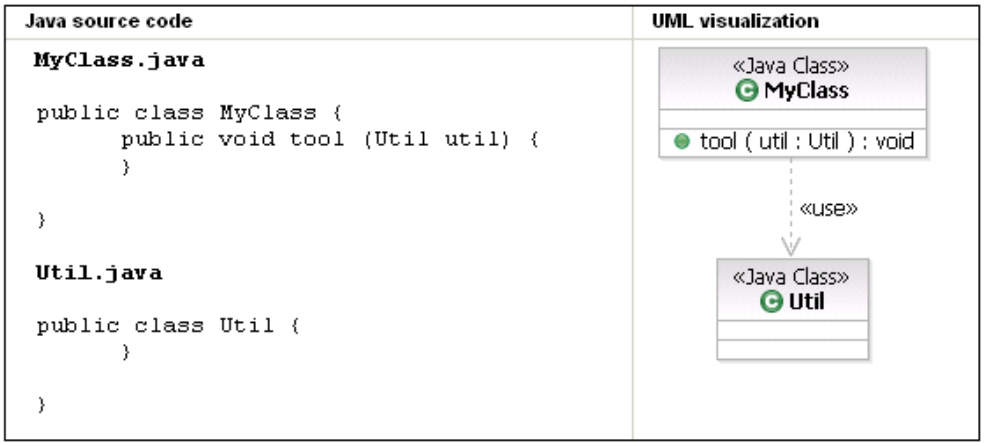

Figure 1.2.2: Example of a dependency relationship.

Extends relationships

In UML visualization class diagrams, an extends relationship (also called an inheritance or an is-a relationship) implies that a specialized (child) class is based on a general (parent) class. Following is an example showing extends relationship between two classes.

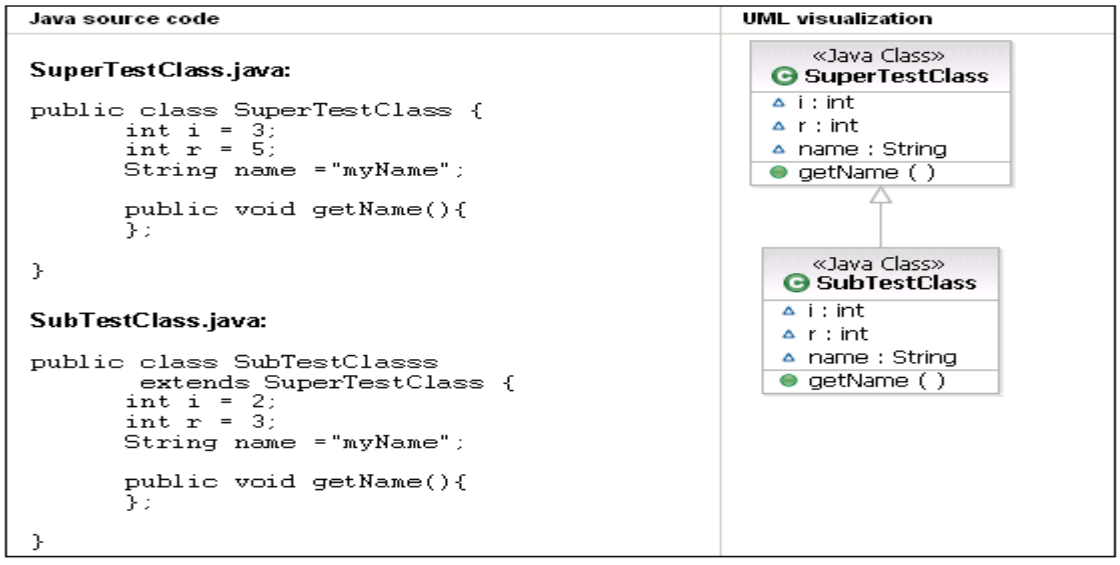

Figure 1.2.3: Example of an extends relationship.

Implements relationships

In UML visualization class diagrams, an implements relationship exists between two classes when one of them must implement, or realize, the behavior specified by the other. Following is an example showing implements relationship between two classes.

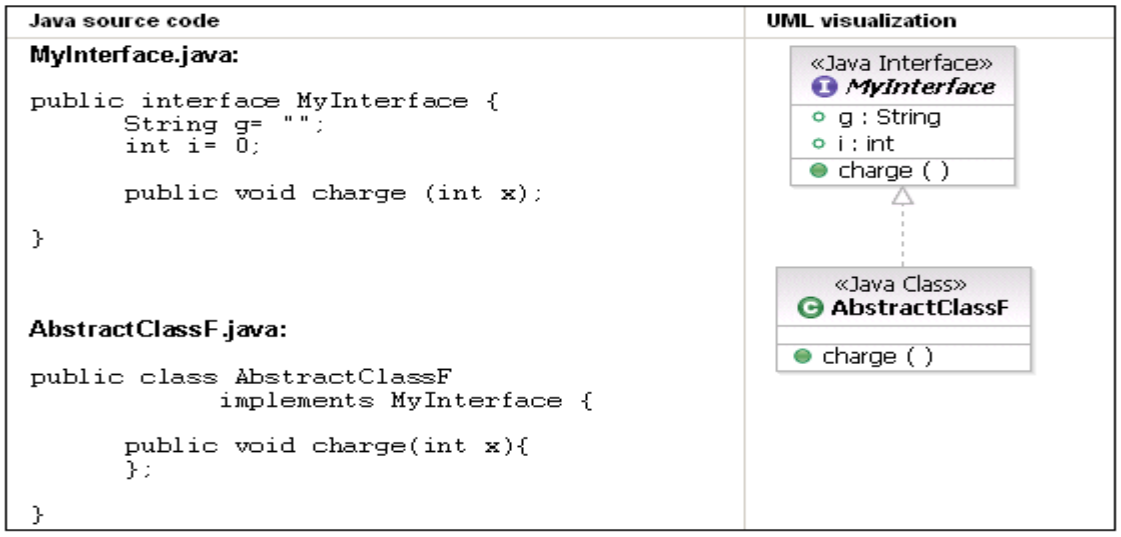

Figure 1.2.4: Example of an implements relationship.

Owned element association relationships

In UML visualization class diagrams, an owned element association relationship is a type of association that dictates ownership. Following is an example showing owned element association relationship between two classes.

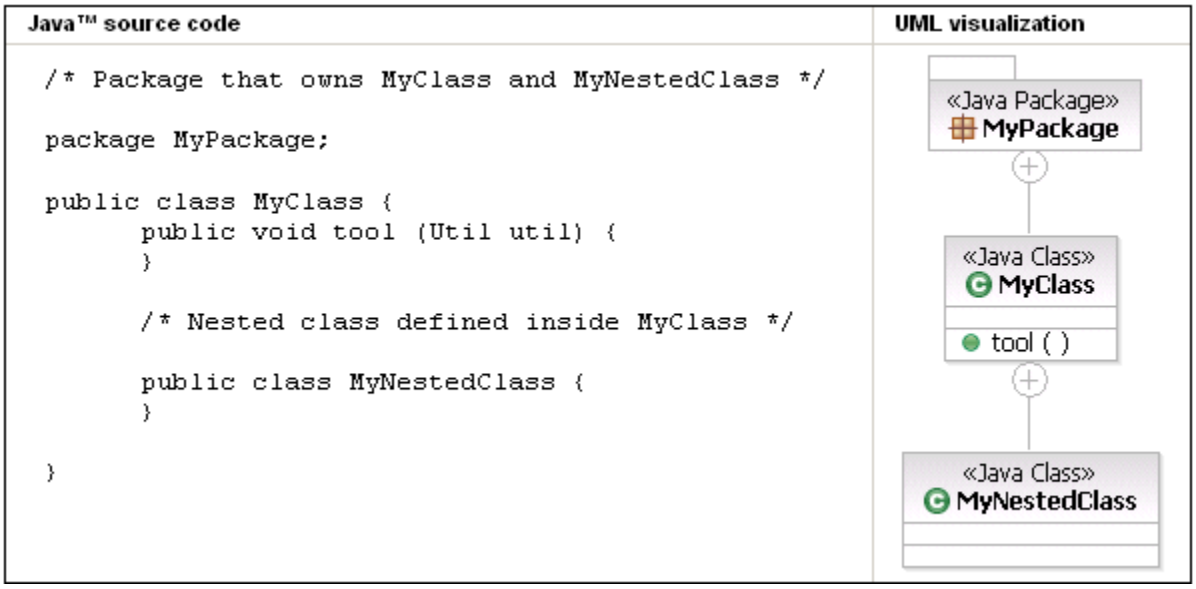

Figure 1.2.5: Example of an owned element association relationship.

The full UML diagram for any of the CID suite software tools is too detailed to fit on a single page. A PDF copy of the diagram for each of the tools in CID suite are available at the URL listed above for the Doxygen documentation. With the PDF files it is possible to zoom in and concentrate on the parts of the diagram of interest.

PART 2: HARDWARE-IN-THE-LOOP REAL-TIME SIMULATION INTERFACE SOFTWARE DESIGN

This section of the report consists of a paper presented at the 2004 IEEE Intelligent Transportation Systems Conference.

Abstract— For years, traffic engineers have used traffic simulation software to develop, model, and test signal timing plans. However, before timing plans can be implemented in the field, they must be fine-tuned in an actual traffic controller operating under actual intersection conditions. Testing a signal timing plan in the field can cause minor or even major traffic disruption, creating delay and frustration for motorists and pedestrians alike. Real-time hardware-in-the-loop traffic simulation (HILS) can test timing plans in the office or lab rather than out in the field. The Controller Interface Device (CID) is the key component of the real-time simulation system. This paper provides an overview of real-time hardware-in-the-loop simulation and then discusses CID hardware design. Finally, HILS interface software design for three different simulation models (CORISM, VISIM, SimTraffic) will be explained.

Introduction

Transportation engineers often use computer models to evaluate or test how well a traffic facility will operate under a given traffic demand. Depending on the complexity of the system under study, the engineer may use either a macroscopic deterministic model to evaluate the performance of a fixed-time signalized intersection or a stochastic microscopic simulation model (CORSIM [1], VISSIM [2], or SimTraffic [3]) to test the operation of actuated signalized intersections during over-saturated conditions. But one of the most serious limitations of these simulation models is that their signal controller emulator sub-models do not include many of the features present in today's traffic controller. The continuing changes to control algorithms as well as the need for manufacturers to maintain propriety precludes simulation software model developers from including these features into their models. For example, the FHWA TSIS/CORSIM model [1] uses actuated traffic control technology from the 1980's and does not allow the user to specify the advanced control algorithms that have been developed in recent years.

HILS [4] and the controller interface device that makes HILS possible were introduced to the traffic industry during the last 7 years. With HILS, engineers can test a signal-timing plan using

an actual traffic controller, in real time, in the convenience and security of the office or laboratory.

The following sections of this paper will first introduce concepts about HILS and the second-generation controller interface device (CID II). Then HILS interface software design for the three different simulation models (CORSIM, VISSIM, and SimTraffic) will be explained.

Hardware-in-the-loop Simulation (HILS)

The basic idea of the HILS technique is very simple. In order to test a piece of hardware, a special simulation model is used to generate test information, which "fools" the hardware into behaving as though it is operating in a real environment.

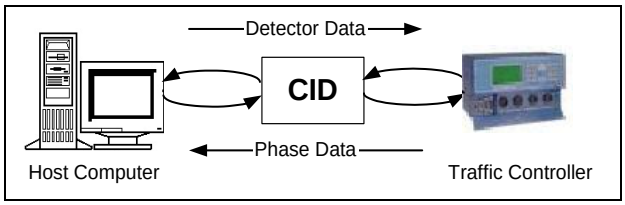

Fig. 1. CID system architecture.

As shown in Figure 1, the CID provides a link between a host computer and an actual NEMA [5, 6], type 170 [7], or 2070 [7] traffic controller. This link between software and hardware is known as real-time, hardware-in-the-loop. A real traffic signal controller replaces the simulation program's traffic controller component or the internal controller emulation logic. It allows a computer to communicate with traffic controller hardware by allowing the simulation models to send detector actuations to the control device and to read phase indications back from the control device. Several first generation CIDs have been developed for the traffic industry in recent years. For example:

- LSU-CID [8], which was developed by Louisiana State University, uses the RS232 serial port as the communication interface between the personal computer and CID hardware.
- REL-CID [9], which was developed by the Texas Transportation Institute, uses a parallel interface to communicate with the computer.
- Naztec TS2 test box, which was developed by Trafficware Corporation [3] and Naztec [10], uses the RS232 serial port as the communication interface between the personal computer and CID hardware.
- Other first generation CIDs were developed by Eagle [11], the University of Minnesota [12], Gardner Systems [13], and Innovative Transportation Concepts [2].

CIDs that use either RS-232 (standard serial communication) or another low speed communication interface are known as first generation CIDs. This communication speed cannot fully meet real time communication requirement. The RS-232 communication speed, less than 1 Mb/s, is not sufficient for the proposed HILS system. The modern NEMA TS2 controller has over 200 functions. Every controller function needs to use one CID input or output channel. Some first generation CIDs only implemented 32 or fewer input/output channels. Testing a traffic network with multiple intersections requires multiple CIDs, and the RS-232 communication interface limits the number of the CIDs that can be used in a HILS system.

CID Hardware Introduction

The second generation CID (CID II) overcomes the first generation CID limitations. The CID II uses a USB as its data communication interface, allowing high-speed, real-time communication of a larger data set. It has 64 I/O connections, which allow the user to test up to 128 traffic controller functions, on up to 40 controllers at a time.

The CID II is built around a microcontroller with a smart USB interface to reduce firmware load. The inputs and outputs use a 24/12-Volt interface. They can be used to detect or to activate traffic controllers' functions.

The CID II uses a motherboard-daughterboard design. It has one motherboard, one power board, one display board, one microcontroller board, two input boards, and two output boards. Each daughterboard has an edge connector for insertion in the motherboard. The motherboard-daughterboard structure inside the CID case is shown in Fig. 2. [14].

A cable connects the CID and the traffic controller through the A, B, C, and D connectors of the NEMA traffic controllers or the C1 connector of type-170 controllers. The NEMA traffic controller has a 24-Volt interface and the type-170 and 2070 controllers have a 12-Volt interface. The CID needs to be compatible with these two different voltage interfaces, so a special circuit is used to accommodate these differences. Both interfaces place detector calls by shorting the detector input to ground during the call. For phase indications, a grounded signal means the indication is on. Otherwise the indication is off.

To conduct HILS, traffic simulation models must exchange information with traffic controllers. The traffic simulation models can't directly talk with CID hardware. Interface software between

simulation models and CID must be designed. The implementation of the interface software for three different simulation models will be introduced in the following section.

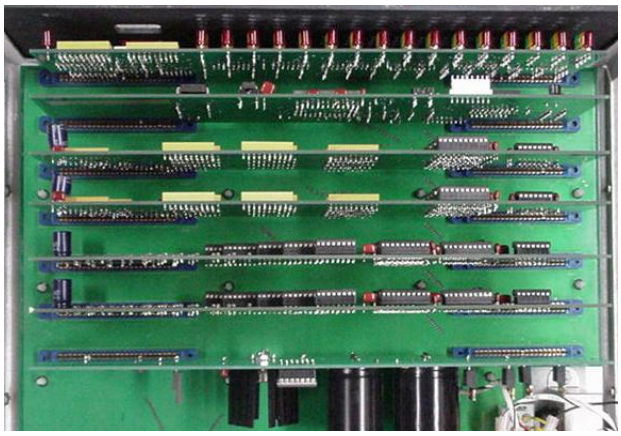

Fig. 2. CID motherboard and daughterboard structure.

Interface Software Design

One of the major issues in the design of the CID software was that it must run in real time, much slower than standard practice in which the simulation running time is limited only by the speed of the computer. With HILS, the clock of the simulation must slow down to match the real-time clock of the traffic controller. That goal has been achieved with the CID II software. Here's how it works:

The traffic simulation model simulates traffic flow based on the traffic demand, street geometry, and signal control plans for a given set of intersections. When vehicles approach a signalized intersection in the simulation, the simulation models send a signal to the controller that vehicles have been detected. The controller reacts to these signals as it would react to real detector actuations, and sends signal indications that it will make in response to those actuations back to the simulation module.

CORSIM Interface Software Design

CORSIM real time simulation is possible because of the Run-Time Extension (RTE) module [1] provided in CORSIM, a mechanism that can provide linkage between an external application and the CORSIM simulation model. The RTE provides for a "plug-in" for different pieces of code at model run-time. The RTE feature also allows users to change parts of the CORSIM simulation

algorithms, to examine and change most of the internal CORSIM data structures, and to have users' own code called at preset times during each cycle of the simulation. Because of the RTE capability, researchers can develop and study different signal control algorithms without adding modifications to the CORSIM code. This capability has also paved the way for HILS. Fig. 3 illustrates this process.

There is a shared memory interface between CORSIM and the CID interface software (RTE). When a CORSIM simulation is initiated, the shared memory will also automatically be established. By using this interface, the CID interface software can exchange data with CORSIM in real-time. The shared memory format is very strict. The software should be able to follow the format to read and update the correct information. When CORSIM simulates a traffic network, vehicles, vehicle movement, phases, and detector information will be updated during each time step. This information is recorded in the shared memory. For every time step, the real-time interface software collects all detector information by reading the shared memory. The CID interface software can update vehicle movement phases by writing updated information to the shared memory.

CORSIM is a time step driven simulation tool. All computations and data transfer must be done within the current time step. This means that all CORSIM object states must be updated and the RTE external software sub-module must complete its execution within this time step. The CID interface software is a CORSIM RTE external software sub-module that has two software sub-modules (shown in the Fig. 4). The phase data update software sub-module reads all phase data back from the CID to update the phase colors for all external-control nodes. The detector data update software sub-module reads all detector data from the shared memory and writes all detector data to the traffic controller.

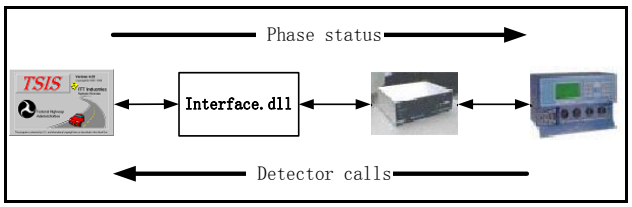

Fig. 3. CORSIM real-time HILS.

The CID phase data update software module has two functions:

- *Read phase data from CID.* The phase data originates from the traffic controller. The CID hardware converts the controller's analog phase signal to digital data and sends it via the USB port to the host computer using the interface software. The software module stores the latest phase data in the CID input buffer.
- *Write phase data to the shared memory.* The CID phase data update software module reads all phase data from the CID input buffer and uses it to update the CORSIM shared memory. CORSIM then accesses the same shared memory and uses the newest phase data to update the phases for the external control nodes.

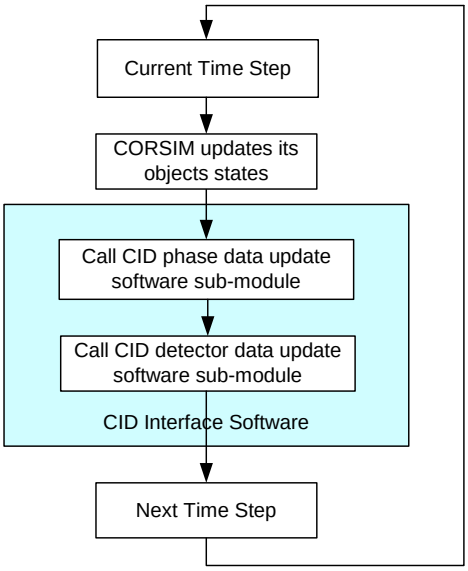

Fig. 4. RTE architecture.

The CID detector data update software module has three functions:

- *Read detector data from the CORSIM shared memory*. After CORISM finishes updating its object states, all current detector data are copied to the shared memory. The software module reads the latest detector data from the shared memory.
- *Format the CID output buffer.* After the software module gets the latest detector data from the shared memory, the software module filters the detector data by the CID ID number and pin number according to the detector to which the data belongs. The software module formats the output buffer with the detector status (On/Off) and timing data.
- *Send the detector data to the CID.* The CID detector data update software module reads the CID output buffer according to the CID ID number, so the detector data can be sent to the correct traffic controller.

The following issues were considered in the interface software design:

- *USB communication initialization.* The CID uses the USB port as a communication interface. A custom USB device driver manages the data flow between the CID and the computer. The CID real-time interface software needs to initialize the USB communication. After initialization, the CID real-time interface application software gets a handle for each CID device. The software uses this handle to communicate with each CID.
- *Traffic network initialization.* The interface software should be able to read the traffic network input file. From this file, the interface software should be able to get the number of links, nodes, detectors, and vehicles position. In addition, the interface software should be able to derive other information from the known information. It should be able to read in the configuration information to get CID hardware settings, including:
  - The traffic network links, nodes, and detectors.
  - The detector position and relations between nodes and detector.
  - Node control type (CID external control or CORSIM internal control).
  - Node control algorithm (fixed-time or actuated).
  - Detector station number. This detector station number includes the CID ID and CID pin number.
- Configuration *information.* The CID interface software needs configuration information before it is ready to run. This information should be generated automatically. The needed configuration information includes:
  - The CID identification number that each external-controlled node is connected to.
  - The phase number that each detector will call.
  - The CID pin number that each phase number corresponds to.
  - The CID-controlled node number.
  - The controller type that is connected to a CID.
  - The phase pattern used by the externally-controlled node.
- *Error handling.* Error handling is a very important part of the interface software. This module can guarantee that the interface software is running smoothly. The following kinds of errors need to be handled:
  - USB communication errors (run time error handling).
  - Configuration information data format errors.
  - Traffic network description file format errors.
  - USB driver initialization errors.

VISSIM Interface Software Design

The VISSIM model was developed at the University of Karlsruhe, Germany during the early 1970s. Commercial distribution of VISSIM began in 1993 by PTV Transworld AG, which continues to distribute and maintain VISSIM today.

The VISSIM model consists of two primary components: the simulator and the signal state generator (SSG). The simulator generates traffic. The SSG is separated from the simulator. It is where the signal control logic resides. Here, the user has the ability to define the signal control logic and thus emulate any type of control logic found in a signal controller manufacturer's firmware. The SSG permits the user to analyze the impacts of signal operations including, but not limited to: fixed time, actuated, adaptive, transit signal priority, and ramp metering. The SSG reads detector information from the simulator at every time step. Based on the detector information, the SSG decides the status of the signal display during the subsequent time step.

VISSIM HILS combines the traffic flow models of VISSIM with the use of an actual traffic controller operating in real time. VISSIM generates and keeps track of the movements of individual vehicles and vehicle detector calls, while the traffic controller (linked through the CID) uses these detector calls as it would in the real world. The controller passes back phase data as VISSIM continues to model the movement of individual vehicles.

VISSIM uses the Dynamic Data Exchange (DDE) method for exchanging data with SSG. VISSIM is the DDE server and the SSG (CID interface software) is the DDE client. The server, VISSIM, can engage in several conversations at the same time with client, CID interface software. While the simulation model is running, detector states are updated by VISSIM. The interface software sends these detector states to the CID via the USB interface. The CID transforms these digital signals to pulse signals and sends them to the traffic controller. Similarly, after the detector information has been sent to the traffic controller, the controller updates the signal phases. The phase information is then sent to the CID. The CID transforms the pulse signals to digital signals and sends them to the computer via the USB port. The phase states that the SSG reads via the USB interface and are updated in the shared memory structure, so VISSIM can determine which traffic streams should be permitted to move. Fig. 5 illustrates this process.

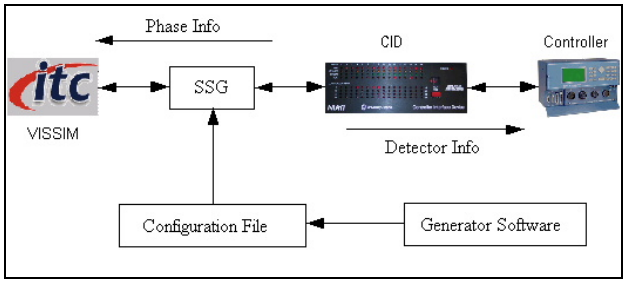

Fig. 5. VISSIM real-time HILS.

VISSIM is a time step driven simulation tool. All computations and data transfer must be done within the current time step. This means that all detector and phase information must be exchanged between interface software and VISSIM within the current time step through DDE communication. One intersection in VISSIM is controlled by one interface software module. The CID interface software has two software sub-modules. The phase data update software sub-module reads all phase data back from the CID to update the phase colors for the external-control intersection. The detector data update software sub-module reads all detector data from VISSIM and writes all detector data to the traffic controller.

The CID phase data update software module has two functions:

- *Read phase data from CID*. The phase data originates from the traffic controller. The CID hardware converts the controller's analog phase signal to digital data and sends it via the USB port to the phase data update module.
- *Write phase data to VISSIM*. After the CID phase data update software module receives phase data from the CID, it will immediately parse the phase and get a different phase color and send the phase color information to VISSIM using DDE communication.

The CID detector data update software module has two functions:

- *Read detector data from the VISSIM through DDE*. The detector information includes detector status and vehicle, which activates the detector, speed information.
- *Send the detector data to the CID*. There is a mapping relationship between the detector and the CID channel. The CID detector data update software module uses mapping to activate the CID channels.

The following issues were considered in the interface software design:

- *USB communication initialization*. VISSIM initializes an instance of the interface software for every externally-controlled intersection. Every instance of interface software needs to initialize the USB communication before it starts. After initialization, it gets the USB handle

for a CID device. The software uses this handle to communicate with each CID.

- *Configuration information.* The CID interface software needs configuration information before it is ready to run. This information should be generated automatically. The needed configuration information includes:
  - The CID identification number that each external-controlled node is connected to.
  - The phase number that each detector will call.
  - The CID pin number that each phase number corresponds to.
  - The CID-controlled node number.
  - The controller type that is connected to a CID.
  - The detector length.
- *DDE communication.* VISSIM and interface software participating in DDE communication are said to be engaged in a DDE conversation. The interface software, which initiates the conversation, is the DDE client application; VISSIM, which responds to the client, is the DDE server application. An application can engage in several conversations at the same time, acting as the client in some and as the server in others. By using this DDE client-server architecture, VISSIM can talk with multiple instances of interface software in real-time. For every time step, the real-time interface software collects all detector information from VISSIM using DDE. The CID interface software can update vehicle movement phases by writing updated information to VISSIM using DDE.
- *Error handling.* The VISSIM interface software error handling function is similar to CORSIM's.

SimTraffic Interface Software Design

SimTraffic is a traffic micro-simulation model developed by Trafficware Company. [3] SimTraffic can model signalized and unsignalized intersections, and freeway sections with cars, trucks, pedestrians, and busses. SimTraffic is a time step driven simulation tool. Its time step length is 100 ms. All computations and data transfer must be completed within the current time step. While the SimTraffic HILS is running, SimTraffic updates detector states and the interface software reads those detector states and sends them to the CID via the USB interface. The CID transforms these digital signals to pulse signals and sends them to the traffic controller. Similarly, after detector information has been sent to the traffic controller, the controller updates the signal phases. The phase information is then sent to the CID. The CID transforms the pulse signals to digital signals and sends them to the computer via the USB port. Fig. 6 illustrates this process.

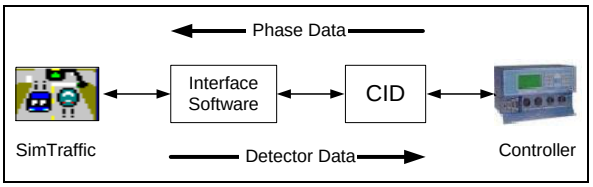

Fig. 6. SimTraffic real-time HILS.

SimTraffic employs memory-mapped files (MMFs) [15] to communicate with the CID interface software. MMF offers a unique memory management feature that allows applications to access dynamic memory through pointers. With this capability, SimTraffic can map a view of the shared data to a specific range of addresses within interface software process address space. MMF uses a server and client structure. The CID interface software acts as a server that creates a named file-mapping object (shared data) during initialization period. The SimTraffic acts as a client that is started after the server. SimTraffic will open a file-mapping object created by the server and map a view to this object to obtain a pointer for local use.

The SimTraffic real-time HILS interface software (shown in Fig. 7) is a software package that can generate the CID configuration data and also control real-time simulation. The CID configuration information includes “Controller Type” and “CID ID.” Fig. 8 shows how this configuration information is collected.

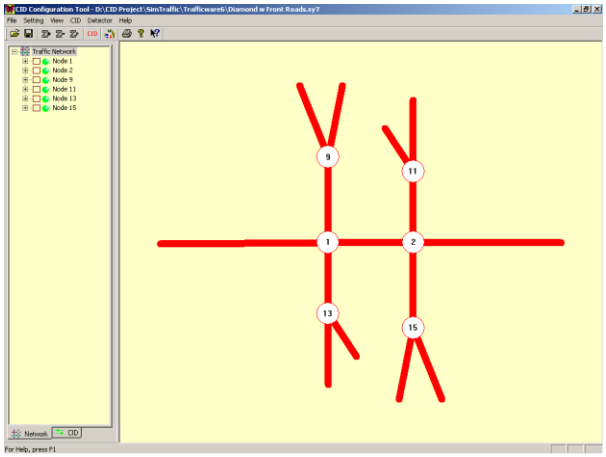

Fig. 7. SimTraffic HILS interface.

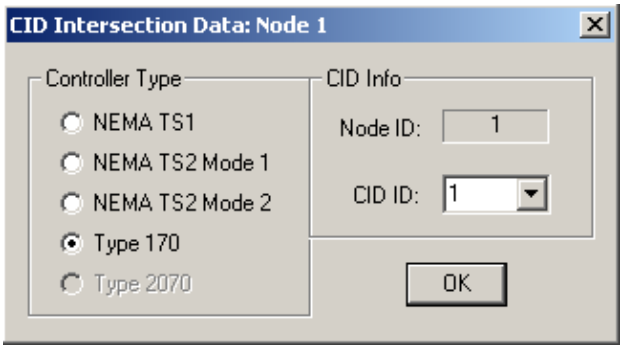

Fig. 8. HILS configuration data input

In order to match the SimTraffic time step, the CID interface software establishes an interrupt. The interrupt service routine is executed every 100 ms. The interrupt service routine reads and writes the phase and detector data from and to shared memory. The interrupt service routine has two software modules:

- *Phase data update software module*. This software module has two functions: (1) Read phase data from CID. (2) Update phase data to the shared memory.
- *Detector data update software module.* This software has three functions: (1) Read detector data from the SimTraffic shared memory. (2) Format the CID output buffer. (3) Send the detector data to the CID.

The issues like: USB communication initialization, configuration information, shared memory interface, and error handling were considered in the SimTraffic HILS interface software design. The implementation of these functions is similar to the CORSIM CID interface software.

Conclusion

The CID II system design includes hardware, firmware, and CID interface software that enables the operation of a real-time HILS. This CID solves the problems that existed with the first generation CID by increasing CID I/O channels and using the USB instead of the serial port as the communication interface. This CID can interface with the most common traffic controllers used in the U.S. market and also the most widely used traffic simulation models.

References

[1] ITT (1995-2003). TRAFED User's Guide. Actuated Controller Coordination. TSIS 5.1 Online Help. Traffic Software Integrated System. Version 5.1, Build 9. ITT Industries, Systems Division.

[2] ITC. (2003). Innovative Transportation Concepts Inc. 1128 NE 2nd St., Ste. 204 Corvallis, OR 97330. http://www.itc-world.com/. Accessed Nov. 01, 2003.

[3] Trafficware Corporation, Synchro Software User Manual. Web site: http://www.trafficware.com/. Accessed October 5, 2003.

[4] Bullock, D., B. Johnson, R. Wells, M. Kyte, and Z. Li, "Hardware-in-the-loop simulation," Transportation Research Part C: Emerging Technologies, Vol. 12, Issue 1, pp. 73-89, February 2004.

[5] National Electrical Manufacturers Association."Traffic Control Systems. NEMA Standards" Publication No. TS 1-1989. Washington, D.C., 1989.

[6] National Electrical Manufacturers Association. "Traffic Controller Assemblies. NEMA Standards" Publication No. TS 2-1992. Washington, D.C., 1992.

[7] CalTrans (1997). Transportation Electrical Equipment Specifications (TEES). California Department of Transportation. Sacramento, CA. Web site: http://www.dot.ca.gov/. Accessed October. 2, 2003.

[8] Bullock, D., and A. Catarella. "A Real-Time Simulation Environment for Evaluating Traffic Signal Systems". Paper presented at the 77th Annual Transportation. Research Board Meeting, Washington D.C., January 1998.

[9] Engelbrecht, R.J., K.N. Balke, S.P. Venglar, S.R. Sunkari. "Recent Applications of Hardware-in-the-Loop Traffic Simulation." Compendium of Papers of the 70th Annual Meeting of the Institute of Transportation Engineers (CD-ROM), Institute of Transportation Engineers, Washington D.C., 2000.

[10] NAZTEC Inc, http://www.naztec.com/index.htm. Accessed October 5, 2003.

[11] EAGLE Traffic Control Systems, A Business Unit of Siemens Energy & Automation, Inc. http://www.eagletcs.com/.

[12] University of Minnesota. (2001). Civil Engineering Department. http://www.ce.umn.edu/.

[13] Gardner Transportation Systems Business Unit. (2003).Web site: http://www.gardnersys.com/newsite/index2.html.Accessed December 5, 2003.

[14] Zhou, Y. and R. Wells, (2000). "A USB Compatible Controller Interface Device" Masters' degree thesis. University of Idaho. 2000.

[15] Randy Kath. Managing Memory-Mapped Files in Win32. Microsoft Developer Network Technology Group. Feb. 9, 1993. http://msdn.microsoft.com/. Accessed Feb. 15, 2004.

PART 3: HARDWARE AND SOFTWARE DESIGN OF AN AUTOMATED TESTING TOOL FOR TRAFFIC CONTROLLERS

This section of the report consists of a paper presented at the 2006 IEEE Intelligent Transportation Systems Conference.

Abstract — Traffic engineers and technicians are faced with the challenge of testing traffic controllers to ensure that they comply with the state requirements prior to deployment. Currently the test is being done manually using manual "suitcase testers." These manual based tests can be tedious and time consuming and come with many limitations such as the inability to document the results and hard to replicate field cases. This paper introduces development of an automated test tool and a new script language (CIDScript) for testing traffic signal controller functionality. The automation test tool can be used to replace or supplement manual testing.

Introduction

The traffic signal controller is a complex device, and traffic engineers are able to program the controller to adapt to many different traffic environments. Before deploying these devices to the field, however, traffic engineers need to test the settings they have programmed and also test the controller's functions. Nowadays traffic engineers often use a device called Suitcase Tester [1] (shown in the Fig. 1) to evaluate or test how well a traffic signal controller will operate under a given signal timing setting. This test device has LEDs and mechanic switches. The LEDs are used to replicate the traffic controller outputs. The mechanic switches are used to activate the traffic controller input functions.

Fig. 1. Traditional NEMA traffic controller suitcase tester device.

In an age of rapid development in increasingly complex environments, automated testing is a more reliable means for achieving an acceptable, consistent level of test coverage in many fields of technology. However, even with the tools and technology available today, the majority of traffic controller testing in the traffic industry is done manually.

The Controller Interface Device (CID), which can be used to facilitate Hardware-in-the-loop simulation (HILS) [2], was introduced into the traffic industry over the last 8 years. In addition, a CID suitcase tester emulator [3] was introduced to provide a less traffic controller specific suitcase tester. The CID tools can also enable automated testing of traffic controller functionalities.

In the following sections, the CID system, traditional suitcase testers, and CID suitcase tester emulator will be described. The need for the design of the traffic controller automated test tool will be explained in subsequent sections.

CID System

The CID system was originally developed for HILS applications. The basic idea of the HILS technology is very simple. In order to test a piece of hardware, a special simulation model is used to generate test information, which is used to "fool" the hardware into thinking it is operating in a real field environment. By doing this, all information collected from the hardware testing will tell the user how it works without disrupting field operations.

The CID (shown in the middle of Fig. 2) provides a link between a host computer and a NEMA [4, 5], type 170 [6], or type 2070 [6] traffic controller that operates in real-time. When this link is used in the traffic simulation models, a physical traffic signal controller replaces the simulated traffic controller component or the simulation program's internal controller emulation logic. The CID allows a computer to communicate with traffic control hardware by allowing the simulation models to send detector actuations to the control device and to read phase indications back from the control device. The CID functions as a bridge between the electrical signals of the computer and those of the traffic signal controller.

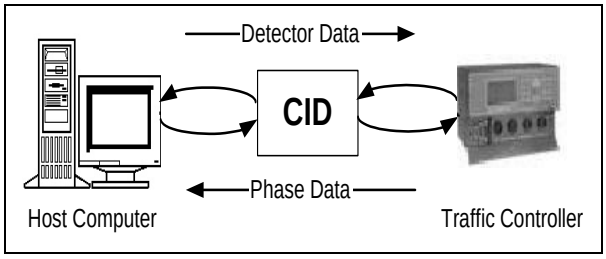

Fig. 2. CID system architecture.

CID II Hardware Design

The CID II is built around a microcontroller with a smart USB interface to reduce firmware load. The inputs and outputs are 24/12-Volt interface (24-Volt for NEMA traffic controller, 12-Volt for type-170, and 2070 traffic controllers). They can be used to detect or to activate traffic controllers' functions.

The CID II uses a motherboard-daughterboard design. It has one motherboard, one power board, one display board, one microcontroller board, two input boards, and two output boards. Each daughterboard has an edge connector for insertion in the motherboard. Fig. 3 shows the motherboard-daughterboard structure inside the CID case.

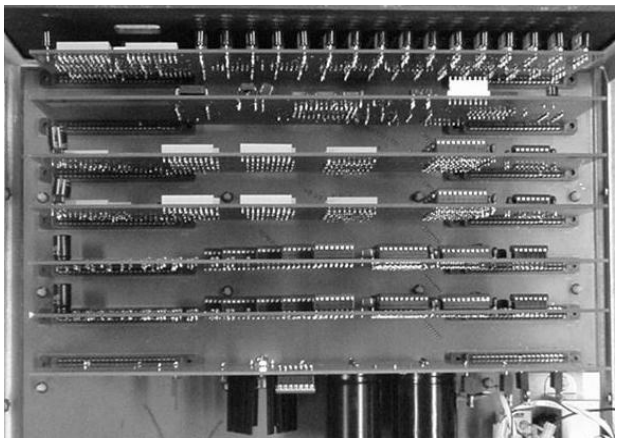

Fig. 3. CID Motherboard-daughter board.

Fig. 4 shows a block diagram of the hardware used on the microcontroller board. The 64 inputs and outputs and the display LEDs are multiplexed on an 8-bit data bus. The 3-to-8 decoder selects which bank of inputs, outputs, or display LEDs is being accessed; an address latch selects which bank of inputs, outputs, or LEDs is read or written.

A cable connects the CID and the traffic controller through the A, B, C, and D NEMA connectors or the C1 connector of type-170 controllers. Both interfaces place detector calls by shorting the detector input to ground during the call. For phase indications, a grounded signal means the indication is on. Otherwise the indication is off.

CID Firmware

The CID firmware consists of two main parts, (1) the USB enumeration and CID configurations for the USB protocol, and (2) the CID functionality, such as the number of detector signals and phase indications, the update rate, and the data resolution. Following two issues are considered in the firmware design:

(1) *Timing mechanism.* For the 64 detector signals, up to 32 simultaneously actuated detectors have timing capability. One datum of detector timing is saved in a two-byte data format.

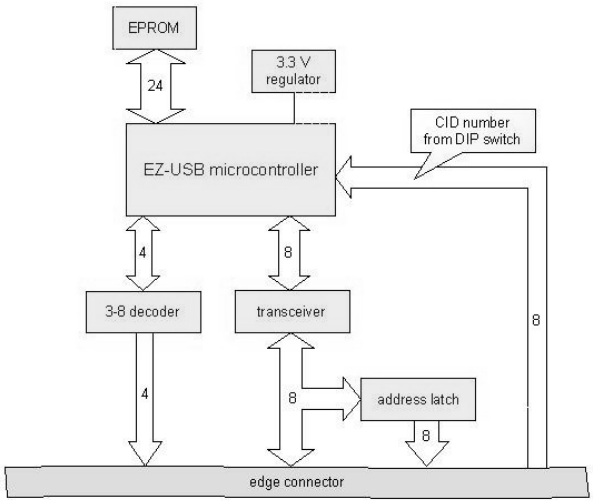

Fig. 4. CID micro-controller board structure.

(2) *Self test.* In order to test CID hardware functions, a self-test function is included. This self-test function tests 64 input and 64 output channels and 128 LED(s).

When a reset signal is issued from the USB host, the CID firmware initializes all hardware and the corresponding registers and variables in the microcontroller. The CID firmware operates in the isochronous USB data transfer mode. Its communication resolution is 1 ms [13].

Traffic Controller Suitcase Testers

Traditional suitcase tester [1]

Traffic engineers have traditionally used a suitcase tester to test and evaluate the operation of a traffic controller before it is placed in the field. A suitcase tester is a suitcase-like device containing switches and LEDs as shown in Fig. 1, that, when linked to a traffic controller, activates functions on the controller and monitors outputs from the controller.

A suitcase tester provides a controlled environment for verifying that the controller behaves as expected when individual functions are programmed. However, the traditional suitcase tester device can be time-consuming and inefficient to use, particularly when it is necessary to verify the operation of all of the controller's functions. In addition, each type of traffic controller on the market requires its own suitcase tester.

CID suitcase tester emulator

The suitcase tester emulator software [3] is a computerized version of the traditional suitcase tester. Just like the traditional suitcase tester device, the suitcase tester emulator can verify that the controller has been programmed correctly, and it can also troubleshoot problems with the controller. It goes beyond traditional suitcase testers in that it can test the most widely used functions of any NEMA or type-170 controller, rather than just one specific model.

In recent years, with the CID technology emerging in the traffic industry, several types of suitcase tester emulators have been developed to supplement shortcomings of the traditional suitcase tester. One class includes devices such as Eagle Traffic Control Systems [3] and Naztec [4] NEMA TS2 testers. These NEMA TS2 testers consist of two main parts: the tester device and the emulator software. They use the SDLC port to communicate with the traffic controller and the COM port to connect with the host computer. However, these test boxes come with certain limitations. The length of a detector call cannot be specified less than a time step (update period). The number of traffic controllers that they can test simultaneously is also limited.

Another type of suitcase tester emulators is the CID suitcase tester emulator software developed at the National Institute for Advanced Transportation Technology (NIATT) [3]. The software is a computerized version of the traditional suitcase tester. Just like the traditional suitcase tester, the suitcase tester emulator can verify that the controller has been programmed correctly, and it can also troubleshoot problems with the controller. It goes beyond traditional suitcase testers in that it can test the most commonly used functions of any NEMA or type-170 controller, rather than those of just one specific model. It can also test multiple controllers at the same time.

The NIATT CID suitcase tester emulator implements three user interfaces for use with three different controller types – NEMA TS1, NEMA TS2 (type-2), and type-170. Each interface includes two basic functions – input and output. The input functions are used to activate the controller's input functions. The output indications are used to monitor the traffic controller's outputs. Two types of buttons are implemented in the user interface. The output buttons are used to monitor the traffic controller output functions (such as phases). These output buttons emulate the traditional suitcase tester device's LEDs. The input buttons are used to activate traffic controller input functions such as activating detector calls. The input buttons emulate the

traditional suitcase tester device's mechanical switches, but with more functions than the mechanical switches are able to provide.

Automated Testing Tool Design

In this section, constraints of the existing testing tool will be introduced and the reasoning behind the development of the Automated Testing Tool will be discussed. The desired features for Automated Testing will be described, followed by a description of the resulting software design. The automated test tool is built around the CID hardware.

Why Automation Is Needed?

One of the most serious limitations of present traffic controller testing devices is that their operation cannot be automated and the test results cannot be recorded. In order to allow the traffic controller testing procedure to be automatic, automated test functions should be added to the suitcase tester.

Automation can be used to replace or supplement manual testing with a test program. Benefits to traffic engineers include increased test quality, reduced test time, repeatable test procedures, and reduced testing costs.

Current Testing Constraints

Manual operation

The mechanical switches in the traditional suitcase tester and input buttons in the CID suitcase tester emulator graphical user interface both require manual operation. To activate a detector, mechanic switches and input buttons are turned on or off manually. The switches and buttons operations cannot be automated.

Lack of intelligence in the tester

Human intervention is heavily involved in the traditional testing procedure. The person testing the controllers decides which buttons or switches are activated and the time sequence. However the timing is not exact. This limits use of algorithms in the testing procedure and results in inconsistent testing.

Test result cannot be recorded

The testing results from the traditional suitcase tester and CID suitcase tester emulator cannot be recorded electronically. The person conducting the test can write down results, but the information is limited. Without detailed test results, the in depth performance analysis cannot be conducted.

Hard to replicate the real-world cases

To replicate the real-world cases manually, the following factors must be guaranteed:

- Detector on/off timing accuracy
- Detector actuation sequence

It is very hard to emulate the detector on/off timing actuation sequence by using mechanical switches. Although CID suitcase tester emulator can set the detector on/off timing, it has no way to emulate special detector actuation sequence.

What Is The Automated Test Tool?

Automation [10] is a technology that allows software packages to expose their unique features to scripting tools and other applications. This technology is widely used in the industries for functions testing purpose. Automation uses the Component Object Model (COM) [11]. The CID automated test tool uses the automation technology and exposes CID functions through the VBScript programming language.

Exposing CID functions provides a way to manipulate an application's tools programmatically. This allows users to use VBScript programming language that automates repetitive tasks that might not have been anticipated. With automation, solution providers can use CID functions to build applications that target a specific task.

The objects an application or programming tool exposes are called ActiveX [12] objects. Applications and programming tools that access those objects are called ActiveX clients. The CID Automated Testing Tools is an ActiveX client, which is an application or programming tool that manipulates CID ActiveX objects. The CID objects exist in the same application. Clients can use existing objects, create new instances of objects, get and set properties, and invoke methods supported by the object.

Desired Features

The automated test software will integrate a script language (VBScript) and a set of CID commands. This new script language (CIDScript) is a set of traffic controller testing commands (definitions and instructions) and VBScript script language commands. Every programmed set of CIDScript can be called a "*strategy."* The strategy includes the specific sequence and programmed commands. The strategy runs in a control engine. This engine will execute the command that the user defined in the strategy. The CIDScript tells the engine what to do at each step to control the process.

CIDScript will be in four forms: Actions, Status,Condition, and Logs.

- Action commands do something in the testing process; for example, *write detector data to controller, read phase data back from controller.*
- Status commands set or get CID status; for example, *set CID channel status and get phase color.*
- Condition commands are questions that always have two possible answers, either yes or no (true or false). The answer determines the flow of logic and what happens next in the process.
- Logs commands record something in the testing process; for example, *timestamp for the CID channel's status.*

Automated Test Tool Software Design

The automated test software is designed to provide a script-based environment so that users can easily design and execute their own CIDScript to test their traffic controller to meet their objectives. Active Scripting [14] technology makes it possible to integrate scripting language (VBScript) in automated test software. The idea behind Active Scripting is to abstract the mechanics of scripting behind some COM interfaces so that it's easy to create the automated test side of a scripting environment. Active Scripting includes the script interpreter in automated test.

The Active Scripting architecture consists of a family of COM interfaces that defines a protocol for connecting a VBscript engine to automated test software. The VBscript engine is just a COM object that's capable of executing script code dynamically in response to either direct parsing or loading of script statements, explicit calls to the script engine's "IDispatch" interface, or outbound method calls (events) from the automated test software's objects [14]. The automated test software can expose its automation interfaces (CID) to the script engine's namespace,

allowing the automated test software‘s objects to be accessed as programmatic variables from within dynamically executed scripts.

Fig. 5 shows the basic architecture of Active Scripting used in the CID automated test software. The automated test software creates and initializes a scripting engine based on the CIDScript to be parsed, and connects automated test software to the engine via the “SetScriptSite” method. The automated test software can then feed the engine script text that it can execute, based on both the script content and the state of the engine. VBScript has two Active Scripting interfaces (“IActiveScript” and “IActiveScriptParse”), which are implemented to expose VBScript parsing engine to CID automated test application. The test software implemented another pair of interfaces (“IActiveScriptSite” and “IActiveScriptSiteWindow”) to access the automated test application's automation interfaces.

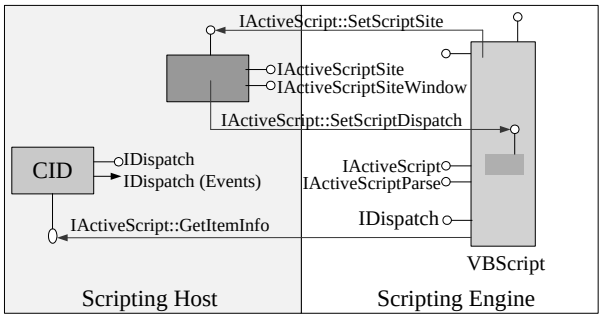

Fig. 5. Automated test software architecture.

For user’s convenience, the CID automated test software (shown in Fig. 6) was developed as an Integrated Development Environment (IDE) that integrates code editing, functions wizard, compilation, execution, and execution output recording functions. The IDE GUI has three components: The first is functions wizard, which can help users to write correct format code. The second is code editor, which allows input and editing VBScript scripts and CID commands. The third is result output, which lists the CID command execution results.

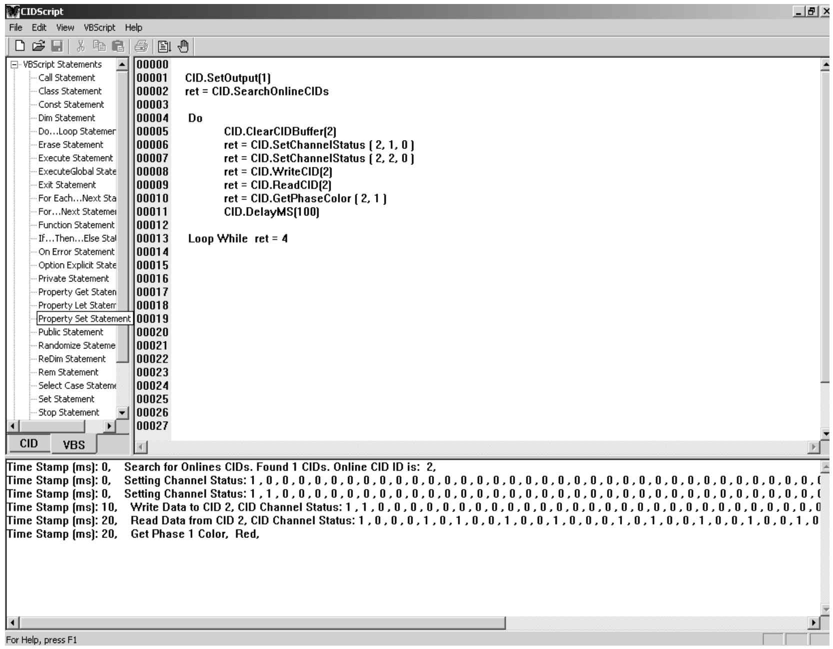

Fig. 6. Automated test tool software GUI.

Automated test tool function wizard

There are 9 CID commands and over 100 VBScript functions and statements. It is tedious to remember all these functions, commands, and statements format. The function wizard is designed to help users to write correct code. The function wizard is organized using two tabs (shown in Fig. 7). One tab is “CID,” which lists all CID commands. The other tab is “VBS”, which lists VBScript functions and statements.

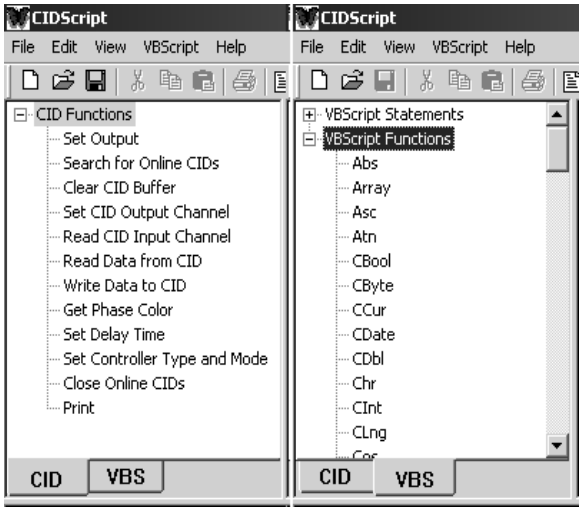

Fig. 7. Function Wizard.

When one double-clicks on the function tree item, a dialog box or function template will be present to users. The dialog box presents all the function parameters and limits the parameters input range.

Automated test script editor

The automated test script editor is designed to input and edit scripts. It implements basic edit functions, such as: Cut, Copy, and Paste (shown in Fig. 7).

Automated test tool output

The results from the execution of each CIDScript command are displayed in the output window. The contents displayed in the output window can be saved in text format. All CIDScript command execution outputs have three parts: the time stamp; the CIDScript command name; and the CIDScript command execution result (shown in Fig. 7).

CIDScript CID commands Introduction

CIDScript script language is a combination of VBScript language and CID commands. VBScript language introduction can be found at Microsoft MSDN web site [16]. A total of 12 CID commands were implemented.

- *Set output sign command* - This CID command is used to set output sign (ON/OFF) for CID commands. If this output sign is set to ON, the following CID commands execution results will be displayed in the CID output window. If this output sign is set to OFF, the following

results from executing CID commands will not be displayed in the CID output window.

- *Search for online CIDs command* - This CID command is used to search for all CIDs that are connected to the host computer.
- *Clear CID buffer command* – It clears all CID output buffers.
- *Set CID output channel command* - Every CID has 64 output channels. This CID command sets CID output channel status.
- *Read data from CID command* -This CID command is used to read data from CID (traffic controller). The data includes 16 phase data and some auxiliary data. This CID command has one parameter – CID ID.
- *Write data to CID command* - This CID command is used to send data to a CID. The data to be sent to the CID includes the status of the 64 CID output channels.
- *Get phase color command* - This CID command parses phase color from the phase data.
- *Set delay time command* - This CID command is used to set period of time for pausing CID script execution.
- *Set controller type and mode command* - This CID command is used to set controller type and mode.
- *Close online CIDs command* - This CIDScript command usually is called after all other CIDScript commands finish. It enumerates the CID link list to archive CID objects. It closes every CID object USB handle and deletes CID objects and removes the CID link list. It has no return value.
- *Print text command* - This CIDScript command prints the text as its parameter to output windows.
- *Read CID input channel status command* - Every CID object has an input data buffer for 64 input channels' status bits. This CIDScript command reads individual channel's status. When an input channel status bit is 0, the channel is inactive. When an input channel status bit is 1, the channel is active.

Controller Automated Testing Sample

A sample script was designed to show the automated test tool functions. In this section, the sample scripts test environment, sample scripts, and sample scripts test result will be discussed.

Test Environment

The sample test was conducted using an Econolite NEMA TS2 traffic controller and a CID II. The phase sequence was set to NEMA dual ring. The maximum green, minimum green, yellow,

and all red times for all phases were set to 20 seconds, 5 seconds, 3 seconds and 2 seconds, respectively.

Sample Automated Test CIDScript

Following scripts sample will record the phase color on the phase 1 to 8 color (red, yellow, and green) termination points. The sample script will first place detector calls on detectors 1 to 8. It then records the phase color every 100 milliseconds until the phase 8 color changes to green. Following shows the sample scripts:

```
Dim CIDID, Phase, Delay_period
Phase = 0
CIDID = 2
Delay_period = 50


CID.SetOutput(1)
CID.SearchOnlineCIDs
ret = CID.SetControllerTypeMode ( CIDID, 1, 0 )
CID.ClearCIDBuffer(CIDID)

For counter = 1 To 8 Step 1
    ret = CID.SetChannelStatus ( CIDID, counter, 0 )
Next
ret = CID.WriteCID(CIDID)

Do
      ret = CID.ReadCID(CIDID)
          For Phase = 1 To 8 Step 1
          ret = CID.GetPhaseColor (CIDID, Phase )
     Next
    CID.DelayMS(Delay_period )
 Loop While  ret = 1
```

Automated Test Result

A total of 7253 CID command outputs were recorded during 75.516 seconds of test execution time. The test results showed that there is a difference between the actual duration of the phases and the controller setting. While the green time was set to 20 seconds, the actual durations were 19.969 seconds, 20.063 seconds, and 19.969 seconds for phases 1 and 5, 2 and 6, and 3 and 7, respectively. While such differences are not operationally significant, the test demonstrated the ability of the automated testing tool to measure and report phase duration with a degree of accuracy. This feature could be useful in comparing phase duration under different controller setting alternatives or detector configurations.

Conclusion

In order to overcome limitations of existing traffic controller test technologies, an automated test tool has been developed. This tool gives traffic engineers maximum flexibility to test traffic controllers. In order to let users automate the testing process, the automated testing tool incorporates a new script language (CIDScript) to control the traffic controller testing process. With this capability, the software can set timed or delayed output pulses. It can wait for specific event responses within certain periods of time and log time stamped results. The functionalities and capabilities of the automated test tool were described. Sample CIDscripts test demonstrated the usage of the automated test tool.

References

[1] McCain Traffic Supply. http://www.mccaintraffic.com/home.html. Accessed May 05, 2004.

[2] Bullock, D., B. Johnson, R. Wells, M. Kyte, and Z. Li, "Hardware-in-the-loop simulation," Transportation Research Part C: Emerging Technologies. Vol. 12, Issue 1, pp. 73-89, February 2004.

[3] Li, Z.; Kyte, M.; Johnson, B., “A Controller Interface Device Based Suitcase Tester,” IECON ’02: The 28th Annual Conference of the IEEE Industrial Electronics Society. Vol. 4, pp. 3107-3111, Nov. 5-8, 2002.

[4] National Electrical Manufacturers Association. Traffic Control Systems. NEMA Standards Publication, No. TS 1-1989. Washington, D.C., 1989.

[5] National Electrical Manufacturers Association. Traffic Controller Assemblies. NEMA Standards Publication No. TS 2-1992. Washington, D.C., 1992.

[6] California Department of Transportation, Transportation Electrical Equipment Specifications (TEES). Sacramento, CA, March 1997.

[7] EAGLE Traffic Control Systems, A Business Unit of Siemens Energy & Automation, Inc. http://www.eagletcs.com/.

[8] NAZTEC Inc, http://www.naztec.com/index.htm.

[9] Synchronous Data Link Control and Derivatives. Cisco Documentation. Accessed in May, 2004. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/sdlcetc.

[10] The Microsoft Developer Network. Overview of Automation. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/automat/htm/chap1_3r1q.asp.

[11] The Microsoft Developer Network. Component Object Model. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/componentobjectmodelanchor.asp.

[12] The Microsoft Developer Network. About ActiveX Controls. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/componentobjectmodelanchor.asp.

[13] Richard B. Wells, John Fisher, Ying Zhou, Brian K. Johnson, Michael Kyte. "Hardware and Software Considerations for Implementing Hardware-in-the-Loop Traffic Simulation". IECON'01: The 27th Annual Conference of the IEEE Industrial Electronics Society. Vol. 3, pp. 1915-1919, Nov. 29 – Dec. 2, 2001.

[14] Don Box. Say Goodbye to Macro Envy with Active Scripting. February 1997 issue of Microsoft Interactive Developer.

[15] National Electrical Manufacturers Association. National Transportation Communications for ITS Protocol (NTCIP) Object Definitions for Actuated Traffic Signal Controller Units. NEMA Standards Publication No. TS3.5-1996. Washington, D.C., 1996.

[16] Texas Department of Transportation, Traffic Operations Division. Departmental Specification TO-4048: Full-Actuated Solid State Controller Unit (TS-2) with Diamond, TBC, Preempt, and Closed Loop Operation; and Base-Mounted TS 2 Cabinet Assembly. Revision 5-99. Austin, TX, 1999.

[17] California Department of Transportation, Transportation Electrical Equipment Specifications.

[18] R. Engelbrecht, "Using Hardware-in-the-Loop Traffic Simulation to Evaluate Traffic Signal Controller Features," IECON'01: The 27th Annual Conference of the IEEE Industrial Electronics Society. Vol. 3, pp. 1920-1925, Nov. 29-Dec. 2, 2001.

PROJECT TEAM

Management

Brian K. Johnson, Co-Principle Investigator

Richard W. Wall, Co-Principle Investigator

Technical Advisors

Darcy Bullock, Consultant (Purdue University)

Mike Kyte, NIATT Director

Graduate Research Assistants

Eugene Bordenkircher, Computer Engineering

Sanjeev Giri, Computer Engineering

Zhen Li, Computer Engineering