Interface between RTOS target and an Industrial System: Developing USB Digital I/O Module

Benjamin O’Brien

Embry-Riddle Aeronautical University

Graduate Research Project

3/22/13

Revision 0.1

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Signature** | **Date** |
| **Student** | Benjamin O’Brien |  |  |
| **GRP Advisor** | Dr. Andrew Kornecki |  |  |
| **MSE Coordinator / Department Chair** |  |  |  |
| **Reviewer** | Dr. Shou Pang |  |  |

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision** | **Description** |
| 3/22/13 | 0.1 | Initial Report Layout |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1 Introduction 6](#_Toc351719387)

[2 Objective 6](#_Toc351719388)

[3 Background 6](#_Toc351719389)

[3.1 Literary Exploration 6](#_Toc351719390)

[3.2 Existing Solutions 6](#_Toc351719391)

[3.2.1 EasyVeep, PSIM, Bradley Simulators, Sealevel USB Digital I/O device, etc 6](#_Toc351719392)

[3.2.2 Why Existing Solutions Won’t Work 6](#_Toc351719393)

[3.2.3 What was learned from Existing Solutions 6](#_Toc351719394)

[3.3 Existing Components Used 6](#_Toc351719395)

[4 Methodology 6](#_Toc351719396)

[4.1 Using EasyVeep 6](#_Toc351719397)

[4.1.1 Reversing EasyVeep Protocol 6](#_Toc351719398)

[4.1.2 Implementing Imitation EasyPort 6](#_Toc351719399)

[4.1.3 Problems with EasyVeep 6](#_Toc351719400)

[4.2 Creating MyEasyVeep 6](#_Toc351719401)

[4.2.1 Reuse of SWF Files 6](#_Toc351719402)

[4.2.2 Decompiling and Reversing SWF Files 6](#_Toc351719403)

[4.2.3 Interacting with SWFs programmatically 6](#_Toc351719404)

[4.2.4 Creating a Serial Protocol 6](#_Toc351719405)

[4.3 Creating MyEasyPort 6](#_Toc351719406)

[4.3.1 Platform Selection Iterations 6](#_Toc351719407)

[4.3.2 Circuit Design and Considerations 6](#_Toc351719408)

[4.4 Real Time Target Platform 6](#_Toc351719409)

[4.4.1 Using the AIM32-104 Digital I/O Card 7](#_Toc351719410)

[4.4.2 Implementing a VxWorks Interface 7](#_Toc351719411)

[4.5 Testing and Verification 7](#_Toc351719412)

[5 Budget and Scheduling 7](#_Toc351719413)

[5.1 Budget 7](#_Toc351719414)

[5.1.1 Projected Budget and Actual Budget 7](#_Toc351719415)

[5.2 Schedule 7](#_Toc351719416)

[5.2.1 Projected Schedule and Actual Schedule 7](#_Toc351719417)

[References 8](#_Toc351719418)

[Appendices 9](#_Toc351719419)

Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| GRP | Graduate Research Project |
| PLC | Programmable Logic Controller |
| RTS | Real-Time Systems |
| RTOS | Real-Time Operating System |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Introduction

# Object

# Background

The following sections include detailed descriptions of research that was performed during the proposal and initial development stages of the GRP. This section primarily documents the state of existing work in this field of study. This section also establishes the selected scope of work which is detailed in the Methodology section.

## Literary Exploration

Currently looking into more literary options and select those most pertinent to project.

## Existing Solutions

There are a number of existing simulation solutions for real time industrial control, but few come close to meeting the goal of this research project. Most available hardware and software solutions examined focus on training users to program and operate PLCs.

### Festo-Didatic EasyVeep

EasyVeep is advertised as a “graphical 2D process simulator with numerous attractive examples on PLC training.” EasyVeep is distributed freely and can be acquired easily. However, the software requires an external hardware interface, which is also produced by Festo-Didatic, called an EasyPort.

### Other Solutions (More on this later)

# Methodology

This section details the methods used and steps taken to complete the GRP. It is broken down into sections that coincide with the main work items that were experimented with during the research process. The items are presented in the chronological order in which they were researched.

## Using EasyVeep

As a software solution EasyVeep appears capable of meeting all the required functionality as discussed in Existing Solutions. In order to be used, a custom hardware device which replicates the behavior of Festo’s EasyPort is need. During in initial inspection of the manufacturer’s website, the documentation on the EasyPort was found to be inadequate in allowing complete replication of the EasyPort’s behavior. Eventually a document containing a protocol explanation was found after some broken links were fixed on Festo’s website. While waiting for the links to be repaired, a binary analysis of EasyVeep and its supporting binaries was performed.

### Reversing the EasyPort Protocol

The process of reversing the EasyPort protocol was a challenge. The extensive details on how the protocol was partially reversed are included in the appendix section Reversing EasyVeep. Primarily, the process consisted of static binary analysis using the freeware version of the Hex-Rays IDA interactive disassembler. Through examination of the files included with the EasyVeep installation, the library that facilitates serial communication with the EasyPort was located and analyzed both statically and dynamically.

To facilitated dynamic analysis, the com0com loopback driver was used to redirect, examine, and send information between various processes running on the local machine. This allowed for easy logging of the communication protocol as well as allowing fuzz testing of the library when attached to a debugger. Fuzzing was used when there was some ambiguity as to the content or context of the protocol. By attaching a small Python script to the other end of com0com loopback,

# Budget and Scheduling

## Budget

### Projected and Actual Project Budget

## Schedule

# References

# Appendices

## Reversing EasyVeep

In order to begin the reversing process, information regarding the format and structure of the EasyVeep executable needed to be obtained. The analysis began by simply launching EasyVeep. The user is greeted with a “Made with Macromedia” splash screen and visiting the Setup pane in EasyVeep confirms that the software relies heavily on what was previously known as Macromedia Flash Player (now an Adobe product). This was helpful information, and Flash, from previous experience, was known to be easy to decompile and reverse engineer in comparison to binaries produced with languages such as C and C++. However, pure Flash based programs cannot directly communicate with low level system devices such as serial ports; there are no documented functions in the official documentation for doing so. This indicates that the functionality of interest resided elsewhere. Analysis indicates that EasyVeep uses Flash only for running and displaying the simulated processes. Communication and protocol implementation is handled elsewhere.

The next step in the analysis was to examine the EasyVeep executable more closely. The assembly information was examined by viewing the executable properties. It was found to be an Authorware Runtime application. Authorware is primarily designed for developing learning management systems (LMS) in a visual manner (INSERT SOURCE). Given the description of the Authorware software, it did not appear that the Authorware application itself would have access to the serial port but further analysis was needed to eliminate the main executable as the targeted binary. To do so the Import Address Table (IAT) of the EasyVeep module was inspected.

The IAT serves as a table of virtual function pointers to various functions encapsulated in external libraries (<http://sandsprite.com/CodeStuff/Understanding_imports.html>). If the EasyVeep application were to perform the serial communication itself, it would likely be importing serial control functions from the Windows API. After dumping the IAT for EasyVeep.exe the 457 entries were checked and none appeared to have anything to do with serial communication. Additionally, the strings list was checked as well for anything related to serial communication. The strings table holds a list of all the ASCII strings used by a program and in this case, nothing related to the windows COM port file descriptor (\\COM%d) used when opening a serial port was found.

After the search for serial port related functions failed to turn up anything significant, the IAT was examined again for clues on where to look direct attention to next. Most imports were from the GDI32 library (Graphics Device Interface), USER32, and KERNEL32 libraries. The USER32 and GDI32 imports all were directed towards drawing and presenting the user interface while the KERNEL32 library functions were reading and traversing file directories and processes. The imports the proved to be of interest were those from the ole32 library. The ole32 library implements Object Linking and Embedding (OLE) which is primarily used for embedding and linking disparate data items, such as embedding a Flash movie into a Microsoft Word document. The OLE allows developers to create their own OLE Control Extensions (OCX) to extend functionality of existing user interfaces.

Inspection continued of the OLE functions. Using the freeware version of IDA interactive disassembler, breakpoints were set on all functions that loaded OLE objects. However, when launching the application through the debugger, exceptions were thrown and the application refused to load and hit the breakpoints. Instead of attempting to debug further, as bypassing anti-debugging measures is not within the extent of this project, a different approach was taken. EasyVeep was launched without a debugger attached and then once fully loaded IDA was used to attach the debugger. IDA provides a list of loaded modules and within that list a module named EasyPort.ocx was found. This appeared to be a promising target as it was named after the device that was being reverse engineered.

The EasyPort.ocx file was loaded into IDA for disassembly and debugging. Immediately upon checking the IAT, functions GetCommState, BuildCommDCBA, SetCommState, and PurgeComm appeared at the top of the list. A quick Google search showed that these were all part of the Windows Communication Functions API (<http://msdn.microsoft.com/en-us/library/windows/desktop/aa363194(v=vs.85).aspx)>.

The COM functions were break pointed and the debugger was attached to a running EasyVeep process. By stepping through the process it was found the EasyPort.ocx begins iterating over a range of integers, attempts to open a COM port of that integer, and then establish communications with an EasyPort device. The COM initialization code is shown below.



This code first builds the file description string for the COM port and then builds the DCB control settings for the serial device. It then tries to open the COM using CreateFileA and the commands immediately following that function call attempt to detect and report errors. A breakpoint was set after the third sprint call to examine the string that was used setting the DCB for the serial device. It was found that for each COM port, the above routine gets called twice. The first time a baud rate of 19200 is used and the second time a baud rate of 115200 is used. If communication fails on both baud rates, the next COM port is tried until no more remain. If communication success, the process continues and all ports found to have a COM device are reported to EasyVeep.



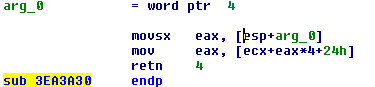
To determine if the COM port is attached to an EasyPort, the EasyPort.ocx sends the ascii string “setup0\r” and waits for a response. If a response is not sent in a valid time frame the setup string is sent again. In return, the software expects a string in the format of “setup%1d\r” to be returned where the value represented by %1d is between 1 and 4. If this succeeds, the string “DV\r” is sent. The expected string in return is in the format of “V=%1d.%2d\r”. Once this exchange is complete, the validity of the EasyPort is confirmed.

Upon getting a list of valid COM ports, EasyVeep will attempt to open each in order to determine what variation of EasyPort they are. It does this by sending the message in the form of “DEW%d.8” where %d corresponds to the number received from the EasyPort during the setup process. It appears that the length of time and manner in which the EasyPort responds to these requests effects the model type returned. Further analysis has not been made, as it was found during implementation that simply responding as fast as possible would cause the emulated EasyPort to be returned as the correct type.

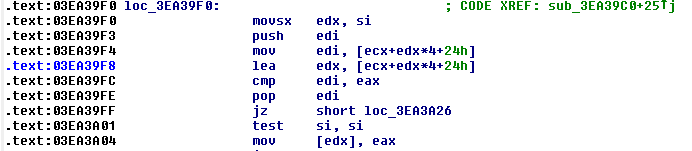
Further analysis was performed on the OCX object to determine what occurs after setup. It was recognized that the OCX was in the same format as a typical windows com object, so the virtual method table (vtable) was located in the .rdata section of the EasyPort.ocx disassembly (<http://blogs.msdn.com/b/oldnewthing/archive/2004/02/05/68017.aspx>). The vtable holds the name of the methods that can be called from a process using the OCX along with pointers to the associated functions. This allows plain text, expressive function names to be used in deciphering the cryptic disassembly.

The vtable contained many more methods than were necessary for meeting the project objective, so only those that were of concern were examined. The first function examined was “SetOutput”. The trace for set output eventually led to the sending of a command “MAW%d.%1d=%04X”, where MAW seemed to stand for Modify Output Word. The %d.%1d was similar to the “DEW%d.8” command found earlier, where the first digit corresponded to the module number, thus the format is Module#.Channel# = HexValue.

Determining how the input values were obtained was not as straight forward. The “GetInputWord” function did not directly lead to a certain serial command, but instead a small move instruction. The move instruction references an address based around the ecx register, which in C++ typically holds the address of the “this” variable. Therefore, at [eax\*4+0x24], the value of the most recent input is stored, where eax is assumed to be the module number.



Examining the code the is run when a serial data stream is received eventually led to a case where an almost identical, but reversed move was found. The same offset is being used to load the effective address of the field into edx and then the value in eax is moved into the address pointed to by edx. This code was found by tracing the execution for the case when the serial input is of the form “EW%1d.%1d=%4X\r”. Here EW denotes the value of the input word (German translation: Eingangswort) is the included four digit hex number.



With this knowledge, all the commands necessary to establish connection and interact with the base functions of EasyVeep are known. The cadance to initialize setup, along with the commands used to modify outputs, and query and update inputs are well documented. None of the other commands that a production EasyPort can interpret are necessary to perform the project objective.