REINFORCING THE DESIGN FOUNDATION OF ASYNCHRONOUS SERIAL DATA COMMUNICATIONS USING LOGIC AND PROTOCOLS ANALYZERS

Dario Schor, Witold Kinsner, and Kathryn Marcynuk

Department of Electrical and Computer Engineering
University of Manitoba, Winnipeg, MB, Canada, R3T 5V6
and Telecommunications Research Laboratories, TRILabs
{dschor|kinsner}@ece.umanitoba.ca, ummarcy@cc.umanitoba.ca

Abstract

The foundation behind asynchronous serial data communications in microprocessor-based systems is generally taught through the theoretical timing diagrams and implementation of a protocol in a laboratory setting. Although students can extract the necessary information from the timing diagram to program a selected microprocessor, they face a number of challenges during the implementation because of the lack of tools to debug and observe the output of the microprocessor incrementally. More specifically, students cannot apply some of the acquired debugging skills like the use of breakpoints or oscilloscopes because (i) programming breakpoints can confirm the logic state of a signal and sequence of events, but not the timing of events, (ii) oscilloscopes can only capture portions of timing signals, and (iii) the signals captured are not digitized, thus displaying uncertainty in noisy environments. Once the programming task is completed, the protocol is verified by transmitting a known message, with the expectation that it will be received at the other end of the serial transmission link - an approach (all-or-nothing) that can be very frustrating during a lab session. This paper presents the use of a logic/protocol analyzer to enhance learning of asynchronous serial data communications by capturing and visualizing the real timing diagrams from a laboratory unit. The use of the Saleae Logic Analyzer provides students with a visual representation of the waveforms at every stage of their design and establishes a very clear link between the timing diagrams discussed in a class and their actual implementations in the lab.

Keywords: Logic/protocol analyzers; microprocessor interfacing; data communications.

1. INTRODUCTION

Over the last few decades, the computer industry has found innovative ways of making smaller and faster electronic devices. Although this trend provides consumer with more powerful tools, it challenges educators to find new alternatives to teach the fundamental concepts that drive the new technologies. Traditional teaching methods such as lectures, visual

aids, and reading assignments help students understand the theoretical concepts that must then be implemented in a laboratory setting in order to fully appreciate all the intricate details and practical considerations of the designs.

One of the fundamental topics in computer engineering that challenges educators is the concept of asynchronous serial data communications in microprocessor-based systems. This refers to the transmission of data from one data acquisition system to another (or many others) over a wired or wireless channel. In general terms, the data transmission process involves (i) acquiring analog signal from transducers, (ii) performing some signal conditioning (i.e., signal filtering and amplification), (iii) converting the signal into the digital domain, (iv) encoding/encrypting the data, (v) applying mechanisms for error detection and correction, and finally (vi) modulating the coded data for transmission [Kins10a, Kins10b]. At the receiving end, another data acquisition system performs the same routines in reverse order to recover the original signal. In asynchronous serial communications, bits are transmitted one at a time in short frames that include overhead (start bit, stop bit, and optional parity) using either an external clock (i.e., as seen in the Serial Peripheral Interface protocol) or encode the clock as part of the transmission (i.e., Manchester encoding).

Asynchronous serial data communications is generally taught through the theoretical timing diagrams and implementation of a protocol in a laboratory setting. In this format, there is an inherited disconnect between the theory (abstract and invisible) and practice (concrete and visible) because students can read and understand the concepts behind a timing diagram and they have to programming background to design the software used by the microprocessors, however, they lack experience and tools to link the theory and practice. Although logic and protocol analyzers have been used in research and development labs for many years, their cost prevented their migration to undergraduate teaching labs. Today, they are very inexpensive, portable, and robust so that not only teaching labs can afford them, but they could be acquired by individual students and hobbyist to experiment at home.

This paper presents the use of a logic/protocol analyzer to

help bridge the gap between the abstract concepts taught in a lecture and the practical applications experienced in the laboratory, thus visualizing the invisible. Examples and observations from an undergraduate course and capstone projects are presented to show the impact of using logic/protocol analyzers in teaching.

2. CURRENT TEACHING TECHNIQUES AND CHALLENGES

2.1. Theoretical Background

Timing diagrams are the most common method to teach students how to design asynchronous serial data communications between microprocessor-based systems. These diagrams appear in textbooks, component datasheets, and white papers as means of describing the sequence of events visually. Arrows help identify the causal behaviour of the system denoting the interdependencies between signals in a system. For example, Fig. 1 shows a timing diagram for a simplified RS232 character oriented protocol for communicating between two microprocessors. The transmitting microprocessor initiates the sequence with a high-to-low transition of the ready-to-send (RTS) line. The receiver then acknowledges the communication with a high-to-low transition of the clear-to-send (CTS) line. The transmitter sends a single character and waits for a signal from the receiver indicating that its buffer is full and is ready to terminate the communication. Once it receives the signal, it forces RTS to a logic high and terminates the transfer.

Fig. 1: Causal timing diagram for the simplified RS232 character oriented protocol [Kins10b]. The character is formatted using a start bit, stop bit, and parity not shown in this diagram.

2.2. Practical Experience

In order to implement the timing diagram shown in Fig. 1, students must dissect the diagram to extract the sequence of events for each processor. Continuing with the aforementioned example, the sequence for the transmitter consists of five parts that (i) initiates the communication, (ii) waits for a response from the receiver, (iii) send the data, (iv) waits for

the signal from the receiver and (v) terminates the transmission. Similarly, the receiver would implement all the tasks in the reverse order.

Experienced students spend time planning their designs and can avoid many issues during implementation. However, for novice students, the implementation is a difficult challenge that requires them to both learn the protocol, understand the hardware and software involved, and to identify the causes of malfunction.

When implementing the protocol in an undergraduate class, novice students face a number of challenges because they have difficulties in applying the same debugging techniques to real-time systems that they have learned throughout their educational process. A real-time system responds to external and internal events faster than the time constraints established for the system. Real time may be hard (strict, without any tolerance), or soft (with some tolerance) [Kins10a]. Some of these challenges are described next.

2.2.1. Breakpoints

A breakpoint is a signal that tells the debugger running on a microprocessor to temporarily enter a suspended state of execution in order to read the state of internal variables and processor registers [Volv11]. Breakpoints are often used in debugging software as a means to verify the operation of a system incrementally by stepping through specific lines in the software [Fitz10]. In real-time systems, breakpoints must be used with caution because time sensitive operations cannot be interrupted without altering the behaviour of the system. For example, including a breakpoint in the data portion of the RS232 protocol from Fig. 1 can cause the receiver to timeout and exit its operation because no data arrived at the receiver. This would indeed affect the flow of the program and prevent a student from stepping through the code.

2.2.2. Oscilloscopes

Having experienced breakpoints in real-time systems, students usually try using an oscilloscope to visually debug their system. They place a probe on the RTS, CTS, and DATA lines and monitor the transmission. The results are captured in Fig. 2 using a Tektronix TDS 1012B oscilloscope. To capture this image, a student needs to run the software and hold one hand on the trigger for the scope in order to pause the live capture when the image is being displayed. A number of attempts may be required to capture the portion of the signal that is of interest to the student (specially for fast processors).

Once the image is captured, the first thing the student notices is that the signal is inverted because of the use of the RS232 protocol, and thus the idle state is actually low, so students must invert the timing diagram in their mind in order to compare the results to those from the figure. Then, as the student zooms into the picture, it becomes very difficult to identify each character and its start bit, stop bits, and parity

bit(s). Students can spend a large portion of their time debugging the system trying to identify the characters for each run of the software. This can lead to students not completing the laboratory on time and/or focusing on reading characters instead of the core objectives for the lab.

Fig. 2: Oscilloscope screen capture showing 'HELLO USER' being transmitted between two machines using the simplified RS232 protocol from Fig. 1.

2.2.3. Uncertainty and Noise

In noisy environments, it is even more difficult to use oscilloscopes as described in Sec. 2.2.2. The signals captured can be distorted by different classes of noise and thus can be very difficult to interpret. For example, white noise would add many spikes to the signals, leading to some uncertainty depending on the sampling rate of the scope. In contrast, pink or brown noise can distort the entire shape of the signal and make a series of bits be shifted with respect to the DC voltage. Thus, correlated noise can complicate reading values from the signals and interpreting what each bit is.

2.3. Verification

When the student feels comfortable with their implementation of the protocol, the verification is generally an all-or-nothing approach. Without proper tools to debug portions of the software as described in the previous sections, the student is forced to send a known message and hope that it gets received at the other microprocessor. If it is received, then the protocol works and the exercise is complete. However, if it does not work, the debugging cycle restarts and students must resort to a combination of the aforementioned techniques to identify the problem. This can be a frustrating task for students who are already pressed for time to complete a laboratory in a short amount of time.

3. INTRODUCING LOGIC/PROTOCOL ANALYZERS

A logic analyzer is a hardware debugging tool used to visualize digital logic signals versus time [Volv11]. It samples and captures a signal, digitizes it using thresholds, and outputs the digital waveforms to a screen. The protocol analyzer is an extension for the common logic analyzers; it automatically decodes preprogrammed protocols and identifies important information by extracting the data from a packet or frame being transmitted. Logic analyzers often have a large number of inputs channels (i.e., 8, 32, 64, and so on) to monitor signals both in real-time and/or to capture them for post-analysis. These devices have been used since the 1970s, but it was not until recently that the price and size of the machines made it possible to incorporate these elements into the education setting [ScWM78, ChMS96, John05].

In an educational setting, a logic analyzer can be used as a complementary tool to aid in the debugging of protocols and asynchronous serial communication between microprocessors. The capture of physical waveforms provides a direct link between the abstract timing diagrams introduced in lectures and the practical implementation that takes place in a laboratory setting. Thus, making the signals visible to the students in order to visually identify problems and detect irregularities between the intended design and the operation of the system without affecting the flow of the real-time system.

For example, Fig. 3 shows the screen capture of a logic/protocol analyzer for the same transmission appearing in Figs. 1 and 2. The first channel shows the RTS signal that initiates the transmission and the second channel shows the DATA being transmitted. Zooming into the data, the device samples and clearly indicates which bits are high or low during the transmission and extracts the data from each character frame. In contrast to the oscilloscope, the idle state is fixed in the protocol configuration and thus makes it very easy to read the diagram and compare it directly to the theoretical timing diagrams.

Additional features of the logic analyzer allow the user to extract metrics about the transmission and measure any interval on the screen to ensure the delays match those specified in datasheets for each device used.

Complementing the oscilloscope captures with a logic analyzer empowers students to debug and analyze their system independently and efficiently. It is very easy to implement part of the protocol (i.e., the handshaking prior to the transmission), test it and visualize it. Then, incrementally add more components to simultaneously build and verify the operation of the system. In noisy environments, oscilloscopes can help identify areas where bits may have been flipped and thus enable students to analytically compensate for those errors to verify the operation of the system.

Fig. 3: Screen capture of Saleae Logic Analyzer software showing 'HELLO USER' being transmitted between two machines using the simplified RS232 protocol from Fig. 1.

4. EXPERIENCE FROM STUDENTS AT THE UNIVERSITY OF MANITOBA

In 2010, the Department of Electrical and Computer Engineering, University of Manitoba purchased 25 Saleae Logic/Protocol Analyzers for undergraduate and graduate student use. The Saleae Logic Analyzer is an 8-channel USB device that captures information on a workstation for post-analysis. The software supports many common protocols and provides an API for users to extend the use for other applications. The number of samples recorded is limited by the amount of memory on the computer, thus allowing up to a 20MHz sampling rate.

Starting in the Fall of 2010, the logic/protocol analyzers were used in the Microprocessing Interfacing for Real-Time Systems (ECE 4240) course [Kins10a, Kins10b], the subsequent course Digital Systems Design 1 (ECE 3760) and for a small number of undergraduate capstone projects. The following sections describe some of the experiences from the students.

4.1. Laboratory Experience

The ECE 4240 laboratory is design to teach students how to interface an asynchronous data communication adapter (ACIA) to a 68000 microprocessor-based system to transfer data between two free-running computers at rates of up to 19.2 kbps. In previous years, students implemented the protocol and struggled as they debugged their system using many of the techniques described in Sec. 2.2. In particular, students were frustrated to find out that the use of breakpoints that they had grown accustomed to using throughout the course did not

help solve the problems. Instead, they had to manually edit the receiver code to remove the timeouts before running their software in order to be able to step through the software and debug the system. Furthermore, as evidenced by summited final lab reports, the students described the procedures but were unable to provide a clear picture showing that they were indeed able to get the system to work.

The introduction of logic analyzers helped students debug some of the problems. Most students used the oscilloscope for debugging incremental steps like the handshaking at the beginning of the protocol. Once that component was working, they incorporated the logic analyzer to visualize the data portion of the transmission. In cases where the received data stream did not match the sender, the logic analyzer helped confirm the message being transmitted, the frequency, and other components that ultimately helped students resolve the problem.

However, the real impact of the devices was in the visualization of the transmission and the possible room for experimentation. Part of the laboratory required students to test different pairs of transmitter/receiver speeds to gain an intuitive understanding of how the speed mismatch can affect the protocol. Normally, this causes the microprocessors to stall as handshaking transitions are missed due to the mismatched speeds, but, in this case, the students were able to capture one transmission and then change settings in the protocol analyzer to see how the data would be interpreted for different cases. This took into account start, stop, and parity bits, thus showing how a microprocessor may ignore invalid character frames.

Furthermore, in the final lab reports, the students were

required to include an annotated screen capture of the logic analyzer [Marc10]. This served as a way of closing the loop on the material by linking the result from the implementation back to the theoretical timing diagrams and served to reinforce the design foundation of asynchronous serial data communications.

In the laboratories for the following course, ECE 3760, students implemented serial communications protocols such as SPI, I²C, and CAN. Although annotated screen captures were not a requirement, some students identified situations where using protocol analyzers would help debug the real-time systems and applied their skills to write and verify their protocol implementation.

4.2. Capstone Project Experience

Throughout the course of the year, two groups of students found the logic analyzers to be useful tools in their capstone design projects for both wired and wireless applications.

In [GlSe11], the students programmed a 16-bit PIC microprocessor to read information from a 16-bit inclinometer using the SPI. The logic analyzer was used to interface the 8-bit asynchronous bus of the PIC to the inclinometer by manually controlling some of the slave select lines. Furthermore, in the development process, the protocol analyzer was used to identify malfunctions in the PIC that caused the clock of one of the asynchronous serial outputs to drift during transmissions.

In [AbSK11], the students interfaced wireless temperature sensors for large structures, such as buildings and bridges. During the early stages of the implementation, the students wrote custom software to test a simple communication between two wireless nodes and confirm the operation before integrating other components of their system. Although the system worked in the laboratory, the team was unable to receive data in an outdoor setting where the nodes would ultimately be deployed. The students used the protocol analyzer to confirm the data was being received properly and identified the start sequence as the culprit that prevented them from successfully receiving data outdoors. After extending the initial sequence with alternative ones and zeros, the receiver was able to synchronize the transmission and receive the data properly.

In both cases, the use of logic/protocol analyzers empowered students to debug their systems, identify the problems, and ultimately complete their projects.

4.3. Other Uses

In addition of using logic analyzers as debugging tools, testing has already began to incorporate their use into a graduate course on Chaos and Fractals Engineering that covers different classes of coloured noise. The objective is to generate synthetic coloured noise to insert into a transmission and see the effects on the digital data. The combination of logic analyzers and oscilloscope captures can aid in the practical

understanding of coloured noise and its effects on digital data transmissions.

The logic and protocol analyzers can also be used in the teaching-teachers workshops [KiBC11] and the visualization of signals for high school students during various outreach activities [KBSD11].

5. CONCLUDING REMARKS

Logic and protocol analyzers are valuable tools for learning about asynchronous serial data communications that help bridge the gap between theory and practice by visualizing otherwise invisible signals. Oscilloscope and logic analyzers together serve as powerful tools that empower students to independently test, experiment, and analyze a system.

Acknowledgments

This work was supported in part by the Department of Electrical and Computer Engineering, University of Manitoba.

References

- [AbSK11] Saif Aboud-McConnell, Matt Singer, and Crystal Kotlar, *Wireless Temperature Sensor System*. Undergraduate capstone project. Winnipeg, MB: Department of Electrical and Computer Engineering, University of Manitoba, 2011. 99 pp.
- [ChMS96] J. Sherwood Charlton, Jeffrey J. McFarland, and James D. Shuffield, "A microcomputer-based logic signal generator and recorder/analyzer for beginning digital electronics laboratory," in *Proc. of the American Society of Engineering Education Annual Conference, 1996*, pp. 151-156, 1996.
- [Fitz10] Sue Fitzgerald, Renee McCauley, Brian Hanks, Laurie Murphy, Beth Simon, and Carol Zander, "Debugging from the student perspective," in *IEEE Trans. on Education*, vol. 53, no. 3, pp. 390-396, August 2010.
- [GlSe11] Pawel Glowacki and Frank Serafin, *Wireless Sensor for Remote Power Line Monitoring*. Undergraduate capstone project. Winnipeg, MB: Department of Electrical and Computer Engineering, University of Manitoba, 2011. 83 pp.
- [John05] Mark C. Johnson, "Work in progress - Use of CPU prototyping and logic analyzers to cultivate troubleshooting skills," in *Proc. of the American Society of Engineering Education / Institute of Electrical and Electronic Engineers Frontiers in Education Conference*, session F3J, pp. 9-10, 2005.
- [KBSD11] Witold Kinsner, M. G. (Ron) Britton, Dario Schor, Arash Fazel Darbandi, Kris Goodmanson, Cody Friesen, and Emily Bashford, "Preliminary experience from the T-Sat project," in *Proc.*

of Canadian Engineering Education Conference, CEEC (St. John's, NL: June 6-8, 2011), 2011.

- [KiBC11] Witold Kinsner, M. G. (Ron) Britton, and Jeff Cieszecki, "Teaching-teachers: could I design, build, and test a tiny smart robot?," in *Proc. of Canadian Engineering Education Conference, CEEC* (St. John's, NL: June 6-8, 2011), 2011.
- [Kins10a] Witold Kinsner, *Microprocessing Interfacing for Real-Time Systems Course Notes*. Winnipeg, MB: Department of Electrical and Computer Engineering, University of Manitoba, 2010.
- [Kins10b] Witold Kinsner, *Microprocessing Interfacing for Real-Time Systems Laboratory Manual*. Winnipeg, MB: Department of Electrical and Computer Engineering, University of Manitoba, 2010.
- [Marc10] Kathryn Marcynuk, *Final Lab Report - Lab 4: Asynchronous Serial Data Communication with Hardware Handshaking*. Final Lab Report. Winnipeg, MB: Department of Electrical and Computer Engineering, University of Manitoba, 2010, 44 pp.
- [ScWM78] John A. Scharrer, Robert G. Wickliff, Jr., and William D. Martin, "Interactive logic state and timing analyses for tracking down problems in digital systems," in *Hewlett-Packard Journal*, vol. 29, no. 6, pp. 14-20, February 1978.
- [Volv11] Jonathan W. Volvano, *Embedded Microcomputer Systems - Real Time Interfacing (3rd ed.)*, Stamford, CT: Cengage Learning, 2011 {ISBN 1-111-42625-2} 793 pp.