Multicommunication Type
Identifying Debugging Probe

Group 1619 - Hardware Security

Team members:
Austin Funes
Cheng Guo
Sommy Okwuosah

Team Advisor:
Dr. Chandy

Graduate Advisor:
Sara Tehranipoor
**Summary:**
The objective of our project is to design and build a probe that will be able to interface with devices that have any type of communication port. This probe should be able to detect what type of communication protocol the device it is connected to is using, and also, act as a multi-purpose debugger for a wide variety of communication protocols.

**Background:**
Given today’s societal dependence on devices that have embedded systems in them, it is paramount that these devices are protected from malicious usage. Technology today is headed towards a new era known as the Internet of Things (IoT), and with this kind of intercommunication across devices leakage of information could be detrimental and even fatal.

There are a lot of devices today that still have open communication port headers that served as test ports in the design and testing phase during manufacturing. These can provide attackers access to the device’s firmware which could be flashed with modifications made to the functionality of the device.

Future goals of our project would be to determine ways in which these vulnerable devices can be protected from attackers. This would require further analysis of the communication signals being transmitted and how to make them harder to access. This way, the functionality and internal makeup of the device is kept proprietary and has less chances to be exploited.

**Communication protocols:**
Since our probe is designed to adapt to any input device, it should meet the requirements of different communication protocols. For our initial design, we plan on focusing on two protocols that are very commonly used: RS-232 and JTAG. We intend to make our design able to be expanded to include other protocols like SPI and I2C. In order to effectively design this probe, it is important to understand exactly how each protocol functions. In doing so, we can better understand their differences and use those differences to enable the probe to tell one protocol from another and relay that information back to us.

RS-232 is a standard serial communication protocol. It formally defines the signals connecting between a data terminal equipment, DTE, and a data circuit-terminating equipment, DCE. Our probe is able to take the responsibility of the DCE to not only identify all transmission line properties (Ground, Data in, Data out, Clock, etc) but also receive and transmit data with DTE part. Notice that it is different from typical logic voltage levels used by integrated circuits. The RS-232 standard signals are either in the range of +3 to +15 volts or the range −3 to −15 volts with respect to the "Common Ground" (GND) pin; consequently, the range between −3 to +3 volts is not a valid RS-232 level. For data transmission lines (TxD, RxD and their secondary channel equivalents) logic one is defined as a negative voltage, the signal condition is called "mark". Logic zero is positive and the signal condition is termed "space". For transmission rate, RS-232 is intended for bit rates lower than 20,000bits per second, which is relatively low
compared to the other three protocols, JTAG, SPI and I²C. Thus, Rs-232 can be easily differed from other protocols by taking the advantage of its different voltage level and frequency data acquired by comparators and FPGA. After figuring out what the protocol is, we can move on to identify all detailed functions of transmission lines and connect a fully reliable duplex communication. As fig.1 indicates, DCD, DSR, RI, CTS, RxD signals will be sent by our probe in the future design. We will consider and solve for the sequence and signal flow graph of these lines communicate with each other in order to first guess or accurately identify the line. For example, we could send an asserted CTS signal, or positive (+3V to +15V) to all possible lines to tell the device that our probe is ready to receive data. And the line that actually receives data is identified to be TxD line. By applying this method, we would be able to identify all lines.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Typical purpose</th>
<th>Abbreviation</th>
<th>Origin</th>
<th>DB-25 pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Terminal Ready</td>
<td>DTE is ready to receive, initiate, or continue a call.</td>
<td>DTR</td>
<td>DTE</td>
<td>20</td>
</tr>
<tr>
<td>Data Carrier Detect</td>
<td>DCE is receiving a carrier from a remote DCE.</td>
<td>DCD</td>
<td>DCE</td>
<td>8</td>
</tr>
<tr>
<td>Data Set Ready</td>
<td>DCE is ready to receive commands or data.</td>
<td>DSR</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Ring Indicator</td>
<td>DCE has detected an incoming ring signal on the telephone line.</td>
<td>RI</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>Request To Send</td>
<td>DTE requests the DCE prepare to transmit data.</td>
<td>RTS</td>
<td>DTE</td>
<td>4</td>
</tr>
<tr>
<td>Ready To Receive</td>
<td>DTE is ready to receive data from DCE. If in use, RTS is assumed to be always asserted.</td>
<td>RTR</td>
<td>DCE</td>
<td>4</td>
</tr>
<tr>
<td>Clear To Send</td>
<td>DCE is ready to accept data from the DTE.</td>
<td>CTS</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Transmitted Data</td>
<td>Carries data from DTE to DCE.</td>
<td>TxD</td>
<td>DTE</td>
<td>2</td>
</tr>
<tr>
<td>Received Data</td>
<td>Carries data from DCE to DTE.</td>
<td>RxD</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Common Ground</td>
<td>Zero voltage reference for all of the above.</td>
<td>GND</td>
<td>common</td>
<td>7</td>
</tr>
<tr>
<td>Protective Ground</td>
<td>Connected to chassis ground.</td>
<td>PG</td>
<td>common</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig.1 Commonly Used RS-232 Signals

The Joint Test Action Group (JTAG) is an electronics industry association formed in 1985 for developing a method of verifying designs and testing printed circuit boards after manufacture. The clock frequency is typically 10 to 100MHz, which is much faster in comparison to the other three protocols. Thus, if our probe detects sampled signals in this range, it may “guess” the protocol to be JTAG. Depending on the version of JTAG, two, four, or five pins are added. Also due to the widely usage of JTAG in various types of device, existing connectors/headers may have 5, 6, 7, 10, 14, 20 and other number of pins. To sum up, four essential pins and one optional pin are critical difficulties for our probe’s function of identification. They are TDI (Test Data In), TDO (Test Data Out), TCK (Test Clock), TMS (Test Mode Select) and TRST (Test Reset) optional. Based on tests data till now, we found that all transmissions lines are set to remain constant value by company when connecting to unknown device (our probe) due to data security. Our project, under this circumstance, will only cover cases where TDI, TDO, TCK are open to receive and transmit or setting information are provided to activate JTAG devices. When we are able to receive and sample signals, we can
decode them and use the method we mentioned in the RS-232 part to determine all pins and eventually read and write freely.

![Diagram](image)

**Fig.2** TMS, TCK and TDI are from device and TDO is from our probe

The Serial Peripheral Interface (SPI) bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. The interface was developed by Motorola and has become a de facto standard. Typical applications include Secure Digital cards and liquid crystal displays. The latest version of SPI has a clock frequency up to few MHz, which is easily distinguishable from JTAG after the SCLK signal is sampled. Like JTAG, SCLK (Clock), MOSI (Data In) and SS (Slave Select) are transmitted by SPI master chip (devices) and only MISO (Data Out) signal is sent from SPI Slave chip (our probe). Since our probe is the only slave chip, we can ignore the SS line in most cases. The identification process is as same as before. Firstly, CLK can be detected if our probe sees a several MHz signal continually shifting between ‘1’ (logic high) and ‘0’ (logic low). Secondly, MOSI can be detected if our probe sees an irregular data string (e.g. 0110001). Finally, MISO can be found by sending ‘trying’ signal from our probe to device line by line. Whenever our probe receives data responding to the desired outcome of “trying” signal, it can be determined that the line sent “trying” signal is MISO line.

![Diagram](image)

**Fig.3** SCLK, MOSI & SS are from device and MISO is from our probe
I²C (Inter-Integrated Circuit), pronounced I-squared-C, is a multi-master, multi-slave, single-ended, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors). It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers. Common I²C bus speeds are the 100 kbit/s standard mode and the 10 kbit/s low-speed mode, but arbitrarily low clock frequencies are also allowed. Recent revisions of I²C can host more nodes and run at faster speeds (400 kbit/s Fast mode, 1 Mbit/s Fast mode plus or Fm+, and 3.4 Mbit/s High Speed mode). These speeds are more widely used on embedded systems than on PCs. The frequency range of I²C coincides with both RS-232 in low-speed mode and SPI in High Speed mode. So it will be difficult for our probe to detect I²C through clock frequency. However, I²C has one big difference from the other three protocols, which is its transmission lines are either SCL (Clock) or SDA (Data). Note that SDA allows bidirectional communication by four potential modes of operation for a given bus device. In our design, we will only use master transmit (device is sending data to probe) and slave receive (probe is receiving data from device) nodes.

**Solution:**

For our project, we chose to create a probe that essentially works as a debugger. With this in mind, we have to further analyze commercial debuggers to see exactly what type of information is accessible via the communication ports. We are specifically working with the JTAG flyswatter and the Keil Ulink2. These debuggers will be useful for preliminary testing of simple motion sensors. The data we get from the testing will be beneficial in gaining insight as to how the signals we receive from the devices we test will behave.

Since our goal is to be able to connect to any device without prior knowledge of its pin-out, we may not know where the specific signals we need are located. This means we will need to have a large number of input wires to our devices undergoing testing so as to account for multiple potential locations for the signals. Understanding that reading from the device will be done in a very short amount of time, as well as the need for a large number of input and output pins (coming to and from the device), we chose to use an Field Programmable Gate Array

![Fig.4 I²C only has SDA and SCL](image-url)
(FPGA) board to read the signals coming from the device. FPGAs typically have a great number of general purpose input output (GPIO) pins and also have really fast clocks. These features make them ideal for our task.

Once the signals have been read, there is more processing that needs to be done in order to identify the type of signal that was received. Although the FPGA could handle this type of processing, we do not necessarily need these calculations to be performed at high speed, also, using the hardware descriptive language (HDL) that gets implemented unto the FPGA is generally harder to debug. For these reasons, we chose to go with using software and specifically, a C-based microcontroller to perform all the processing that does not need to occur at high speed.

These two devices, the FPGA and microcontroller, make up the bulk of our probe. However, we realized that having these two separate devices would require more time spent debugging because we would have to find a way to seamlessly have the two work together. We foresaw potential issues and design challenges with this plan such as; difficulty in getting information to and from each of them since they operated at different speeds, increased surface area and more materials being used (wires and extra components), and also the cost of each device was considerable. As a result, we decided to go with the Zybo Zynq FPGA board since it has an embedded microcontroller. This feature was especially attractive as it gives us the ability to be able to seamlessly communicate between the two. The Zynq board is also considerably smaller than an average FPGA which reduces our overall surface area and prevents our probe from being too bulky.

**Design Choices:**

Our design of the probe makes the use of multiplexers and demultiplexers that are connected to the device directly in order to reduce the total amount of hardware needed in the physical part of our probe. These multiplexers are controlled by the FPGA in order to keep track of which wire is currently being read which will ultimately result in a more simplified code. Furthermore, if we use the FPGA to control the multiplexers we can keep track of the voltage levels being input into the device which would give the probe more information when attempting to detect what the pin is of the communication protocol.

The demultiplexers are used to carefully input signals into the device at a variety of voltage levels. If the voltage being input is too low then the device will not be able to read the input. However, if the voltage is too high the probe risks burning out the device that we are working on rendering it useless. By using a demultiplexer the probe can input a signal at the lowest level, check the other wires if it results in data being output from the device, and if no output from the wire is seen then it will select another pin on the device to input to.

The comparator chip that we are using is the LM393 serves two purposes: detecting the voltage of the pin and its capability of being configured as a level shifter. The voltage of the pin is critical to identifying the communication protocol and pin in the protocol because as discussed
earlier different protocols operate at different voltages. Thus, if a certain voltage is seen the probe can immediately identify the protocol and pin from this information alone. Secondly, the FPGA will only be able to read the data being input into it if it is at a certain voltage which is why a resistor, capacitor and power source will be added to the output of the LM393 in order to level shift the voltage to 3.3V which is the voltage that the FPGA will need to read the data.

The Zybo Zynq board we felt was the ideal choice for our probe because of the high frequency it can operate at in addition to containing an integrated microcontroller. The high frequency of the FPGA component of the Zynq board is crucial to our probe because it will need to sample clocks being output from the device. If it cannot sample the clock fast enough then the probe will not be able to accurately determine if it is a clock pin or not. Furthermore the microcontroller component of the Zynq will be critical to the probe as a whole because it will be reading the data input into the FPGA and attempt to predict the communication protocol and the pin currently selected. Doing this analysis of the data in the microcontroller will be ideal because C will be a lot more friendly to code in than VHDL.

**Signal Flow:**

![Diagram of the signal flow of the probe](image-url)
All the pins of the device will be connected to multiplexers of the device. Which pin is selected is controlled by the Zynq board. Once a pin has been selected the data will be compared to a range of possible voltage levels: 1.8, 3.3, 5, and 15 volts and level shift the voltage to 3.3V. Once it has been level shifted the data will be sent into a priority encoder in order to select the leftmost bit. This is necessary because the comparison occurs in parallel but only one of the outputs is needed. Once the data has been received from the priority encoder is received the Zynq board will attempt to predict the communication protocol and pin. If no data is seen from the pin then the Zynq board will output data into another set of level shifters in order to account for a wide range of feasible configurations. Once the data has been level shifted the Zynq board will decide which voltage to input into the device by controlling a multiplexer and select the pin to input to on the device. The Zynq board will then attempt to read data from the other pins of the device and again attempt to determine the protocol and pin while this data is being input. This repeats for every wire of the device. If our probe operates as intended then the protocol and pinout will be determined by the time the probe has finished reading the last pin.

**Analysis of Components:**

MAX306CPI+-ND (16:1 Multiplexer) - Supports +/- 4.5V to +/-20V which accommodates the minimum and maximum voltages that we expect from the communication protocols that we will be trying to identify.

LM393 - Supports up to 36 volts and can be configured to also be used as a level shifter using the configuration seen in figure 6.

![Comparator chip configured to also act as a level shifter](image)

Fig. 6 Comparator chip configured to also act as a level shifter
Zybo Zynq Board - FPGA with internal clock speeds of up to 450 MHz and also has a built in microcontroller enabling the probe to use C code as well. Contains 6 PMOD connectors each containing 6 pins that can be configured as GPIO pins as well.

TC74HC4051APF-ND (8:1 Multiplexer/Demultiplexer) - Supports up to 15 volts accommodates minimum and maximum voltages of protocols

**Budget:**
1x 16:1 Multiplexer - 8.70$ each
3x 8:1 Multiplexer/Demultiplexer - 0.74$ each
8x LM393 - 0.45$ each
1x Zybo Zynq Board - 132$ each

Total - 146.52$

**Project Phases:**

Design - Consist of planning what the probe should consist of and how it should operate.

Preliminary Testing - This testing includes simulations of the hardware aspect of the design and seeing if the outputs are what we expect from it. Also we intend to connect devices to an oscilloscope in order to see what the signals would look like in order to have a better idea of what we’re looking for when we are coding the microcontroller to interpret the signals.

Assembly - This phase consists of assembling the hardware and coding the FPGA and microcontroller according to our design.

Testing/Debugging - We will connect devices to our probe and see if it works. Since it most likely will not we will see where bugs and issues come up and debug the probe accordingly. We expect most of the issues to come up with the software aspect since determining the protocol and what each pin is specifically can be complex.

Attempt to retrieve flash from chip on device and write to flash - Time permitting we would like to further expand the project and allow it to be a universal debugger because once the protocol and pinout is determined we can attempt to emulate the operation of other debuggers.
Finalize project and documentation- To finalize the project we would aim to solder the components to a pcb design and document how the probe should be used in order to have any person use the probe to identify the communication protocol being used by the device.

Timeline:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>11/11/2015</td>
</tr>
<tr>
<td>Preliminary Testing</td>
<td>1/29/2016</td>
</tr>
<tr>
<td>Assembly</td>
<td>2/12/2016</td>
</tr>
<tr>
<td>Testing and Debugging</td>
<td>2/26/2016</td>
</tr>
<tr>
<td>Retrieve flash from chip on device</td>
<td>3/11/2016</td>
</tr>
<tr>
<td>Write to flash memory</td>
<td>3/11/2016</td>
</tr>
<tr>
<td>Finalize project and documentation</td>
<td>4/1/2016</td>
</tr>
</tbody>
</table>