

CS339 PROJECT:  
SIM CARD I/O SNIFFER WITH A Z8 ENCORE

By  
Jeffrey Karrels

SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCE  
AT  
THE GEORGE WASHINGTON UNIVERSITY  
WASHINGTON D.C.  
MAY 2006

© Copyright by Jeffrey Karrels, 2006

# Abstract

This project provides a means to sniff the communications between a subscriber identity module SIM card and a mobile device. Originally this project was meant to be more of a *Z8 in the middle* project where the Zilog Z8 Encore! was directly between the mobile device and the SIM card. Due to voltage differences, poor resolution of the timers for the speed needed, and lack of computing speed, I decided rather to just sit on the lines between the mobile equipment and the SIM. This still provides the data that I was wanting to obtain, it is just not as robust as the original plan.

Also included into this paper is the addition of a GPS module. It is not part of my Final project, but the GSM board I am using has support for the GPS module, so I decided to buy one, try it out, and describe it's uses.

# Chapter 1

## Overview

### 1.1 Introduction

This project was a lot of fun and a very good learning experience. The goal of this project was to be able to record, and perhaps inject, data transferred on the I/O line of a Subscriber Identity Module (SIM) card. A SIM card is what a Global System for Mobile Communications (GSM) network uses to store the identity of a Mobile device among many other things. This SIM is normally found underneath the battery of a GSM phone, and is specific to a service account. There is a specification for general communication between the GSM mobile equipment and the SIM card, but there are also some vendor specific commands that are not publicly specified. So the interesting thing here is to learn what the interaction is between the two entities.

### 1.2 Status

The original proposal for this project consisted of more of a Z8 in the middle type of approach to the problem. This proved to be a difficult task. Having said that, I never got around to the command injection part of my original proposal. It took a good deal of time to get the module up and running. Then the remainder of my time

was spent trying to figure out how to record the data from the SIM I/O line, and how that data was formatted. I hope this document will show how simplistic it would be to integrate a GSM module into a new or existing project as a source of wireless communication. Also I hope this will be a good starting point for a continuation of the original design.

Where the project status is the following:

1. GSM module is in working status. That is, it is able to function as a working cellular phone. (There are connections for a speaker and a microphone, but that was not in the scope of this project)
2. The Z8 Encore is connected to the Clock, Reset, and I/O lines of the GSM module.
3. The Z8 is able to record to RAM and output on its UART/RS232 the recorded transmissions from the SIM and GSM module interface.

# Chapter 2

## Project Specification

The ISO file 7816 [6] defines the specifications of an identification card, id est *smart card*. At the base level, a SIM card is and operates to this specification. Not in this specification however are GSM vendor specific functionalities. These specifications are hard to obtain. I ended up finding a leaked specification [7] on the web for an old model of a SchlumbergerSema SIM card. The goal of this project is to be able to listen and learn to transmissions between the ME and SIM that cannot be found in [6].

### 2.1 SIM Card

The SIM card that I used for this project was from the Cingular network and was a Axalto Simera Classic 3.3. It turns out that [2] defines 3 classes of SIM cards. There exists a Class A, B, and C SIM card which operate at 5v, 3v. and 1.75v respectively. It looks as though most current models of SIM cards being released are that of the type 3v class. Also card readers for both smart cards and SIM cards are adapted for interfacing all classes. A card reader will attempt to reset the card at 1.75v, and if no response will move up and try 3v, and finally 5v.

## 2.2 GSM module

I actually was not aware of these classes of SIM cards at the beginning of this project. But it turns out my lack of knowledge on this did not prove to be a problem. The GSM module I chose operates with only 1 class of SIM cards, the 3v class. Which turned out ok considering the SIM I had was of that type.

## 2.3 Interface

The project goal is to be able to sniff the I/O line of the SIM card. So the project should be able to capture the data coming across the I/O line in a matter that it can be parsed and made sense of. This will entail recording the data as well as transferring it to a personal computer so that it can be further used. Using the GSM module will aid in this task due to it has an external SIM card. So no additional hardware will need to be made in order to sit on the SIM communications lines.

# Chapter 3

## Implementation and Construction

Implementation of the project specification consists of a couple of milestones.

1. Getting the GSM module up and running in *standard* mode (Section 3.1)
  - (a) Power supply
  - (b) SIM connections
  - (c) Antenna connection
  - (d) UART connection
2. Connecting the Z8 to the Trizium
  - (a) SIM Clock
  - (b) SIM Reset
  - (c) SIM I/O
  - (d) Ground
3. Recording SIM I/O data
4. Transferring Capture data to Host machine

5. Parsing Data on Host machine

### 3.1 Trizium - Standard Operation

This section describes the process from the out of a box GSM module to a working module that is on the network. The hardware module used in this project was a Telit Trizium module. Section A.2 defines the specifics of the module. The following sub sections will cover powering, connecting, and interfacing the Trizium.

#### 3.1.1 Power Supply

The first task in getting the Trizium up to an operating state is to work out it's power issues. The Trizium requires a 3.4v to 4.2v power supply. The specification [4] also states that a max current draw of 2A. In order to meet those requirements a Linear Technologies *LT1528* adjustable voltage regulator was used. Section B.1.1 defines this component and how it was configured in order to provide the specified power. Figure 3.1 shows the configuration of the circuit in order to provide the voltage of 3.8v.

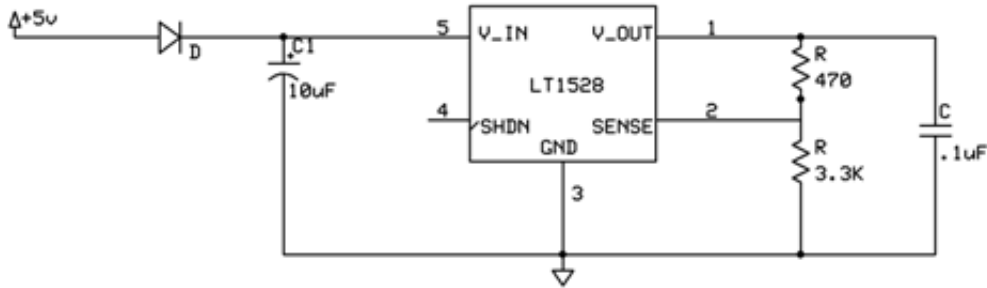


Figure 3.1: LT1528 circuit for 3.8v Voltage Regulator

### 3.1.2 UART - RS232 Conversion

Once the power requirements are fulfilled, the next step is to worry about the UART conversion. The UART on the Trizium board was 3v logic. I originally had thought, and could not get this working, with the 3.3v UART on the Z8. I later found out that it was a bad ground wire between the Z8 and the Trizium. So the Trizium UART can be directly connected to the Z8 Trizium and all will be fine. For a basic standard operation though it is pretty easy to setup a RS232 conversion and talk to the Trizium via a desktop machine over RS232.

#### MAX3232

The first RS232 translator I used was the *MAX3232* described in section B.2.1. The circuit design for this setup can also be seen in Figure 3.2. This was an easy connection, just power the *MAX3232* with the desired power of the UART being interfaced with, connect the Receive and Transmit lines, and it was working.

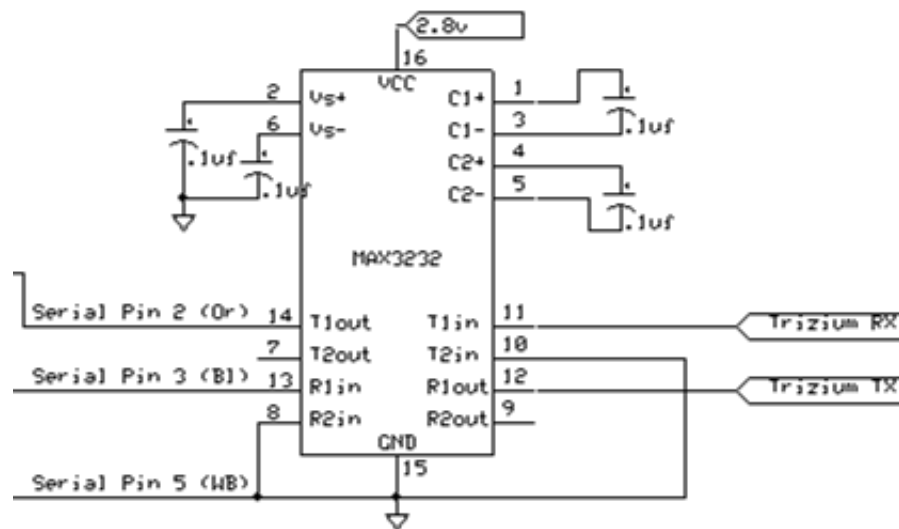


Figure 3.2: MAX3232 Conversion

## **XN04312**

In order to learn a little more, I also implemented a level shifter that was in [4]. This consisted of a NPN-PNP transistor circuit. Section B.2.2 describes the IC used in this circuit. Figure 3.3 shows the circuit.

### **Spark Fun's RS232 Converter**

Finally after I got the level shifter working, I purchased a pre-made shifter that used the same IC and circuit described in the previous section with addition of some LEDs for Receive and Transmission status. Sparkfun's version of this can be seen in section B.2.3.

### **3.1.3 SIM connections**

The Trizium uses an external SIM card connector opposed to a built-in SIM reader found in many devices. This was one of the main reasons why I chose this model. This means that one has to wire the SIM card into the Trizium board. But this also means that wiring the connections in order to tap on to the wires for sniffing purposes becomes much easier. In order to connect the SIM, a SIM housing needs to be used. Again, Spark Fun takes care of this with their SIM card break-out board detailed in Section A.3.1. This takes the necessary wires from a SIM card and puts them to a header. It also contains a switch that lets the Trizium know if a SIM card is inserted or not. Figure 3.3 shows the SIM connections to the Trizium.

## 3.2 Z8 Connection

### 3.2.1 UART

In order to get the recorded data out of the Trizium and onto a host machine, the Z8 UART and RS232 converter are used. The standard Serial cable and DB-9 Connector to Port P1 was used for this with UART 0 on the Z8 Encore.

### 3.2.2 SIM Clock, Reset, Input Output

As Section 3.1.3 describes, the tapping onto the SIM clock, reset, and I/O lines is a fairly simple task in this setup. A simple Y cable setup was used in which the SIM breakout board had two wires coming out of it. One going to the Trizium and the other going to the Z8 for recording purposes.

## 3.3 Software Setup

### 3.3.1 GPIO Ports

After the physical connections are made, one must configure the setup in order to perform the capture. This was where the majority of my work was performed. Obviously the pins are setup as input. But what to do with them was the question at hand.

#### **TIMER capture input**

The first thing I tried was to setup the Trizium SIM clock as a Timer input in Capture/Compare mode. Then on the interrupt for the clock going high I could determine the clock speed as well as what the other pin statuses were. This drew a couple of problems after I tried this. After I put the SIM clock on a Oscilloscope

I found out that the clock was running at 3.25Mhz. The 3GPP specification [1] describes the startup clock frequency as 3.57Mhz. The resolution of the Z8 timer could not calculate a frequency that high with any precision. So if the Capture timer was not accurate, there was no point in using it.

### **Interrupt Driven**

The next thing I tried was to simply setup the SIM clock sniff line as a interrupt driven GPIO. This seemed to be perfect. The ISO 7816-3 [6] describes how to measure the data on the I/O line as a function of the clock. Bits on the I/O line are separated by time where time is defined by an Elementary Time Unit (ETU). The initial ETU for the line is  $1ETU = \frac{372}{f}$  where  $f = 3.25Mhz$ . So for every 372 clock cycles I should be able to read the I/O line and receive valid bit. This looked to work fine, however when I went to parse the data, I could not make sense of it. I seemed to be missing bits while recording data.

### **Polling**

After the failures of previous sections I switched to a simple polling design. In this design, in the main while loop of the program I just polled the status of the I/O line and wrote it to RAM. I realized that in previous sections, one of my problems was that I was outputting to the host machine during times in which I was also trying to record the SIM data. Printing to the UART for data transfer proved to be very computationally consuming. So my new design polled the I/O line, wrote it to RAM, and when I the SIM was not doing anything, the Z8 could transfer its data to the host machine. In order to cut down on the amount of data stored in RAM, I later changed the program to only sample the I/O line on HIGH CLOCK transitions. When doing

this, I found that I could sample the I/O line 6 times for 1 ETU. The other thing that I needed to make sure of was that the Z8 could perform those 6 captures for ETU for both when the I/O line was 1 and 0. Meaning that capturing a 1 should take the same amount of time as capturing a 0. Chapter 4 describes this in more detail.

### 3.3.2 Z8-Host

Listing 3.1 shows the C code for the *SW1* push button. This code simply dumps the data recorded in RAM to the UART 0 so that it can be further parsed on a host machine.

---

Listing 3.1: Outputting Capture data to Host

---

```
char myCHAR;
address = START_ADDRESS;

while( address < CURRENT_ADDRESS){
    myCHAR = SRAMADDR( address );
    printf ( '\n0x%3X-0x%2X' , address , myCHAR);
    address++;
}
```

---

### 3.3.3 Trizium-Host

This section will go over some of the basic AT commands that can be issued to the Trizium. [3] describes all of the AT commands that the Trizium supports while [5] describes some examples of how to use the commands.

## Baud Setting

The first thing that can be done with the Trizium is to setup the baud rate of the UART. The module default is 38400. Listing 3.2 shows a sample of the dialog between the host and the Trizium in order to set the baud rate.

Listing 3.2: Set Baud Rate

---

```
//See if the connection works
> AT<CR>
> OK

//Set the BAUD rate to 38400, 8N1
> AT+IPR=38400<CR>
> OK
```

---

## Band Selection

The module also defaults to European networks. This means that in North America, upon initial startup of this module, it will not connect to a network. This is due to the band needing to be changed. Listing 3.3 shows a sample dialog of setting the band of the module to the 1900Mhz Band.

Listing 3.3: Set Band

---

```
> AT+#BND=1<CR>
> OK
```

---

## Text Message

The interval of which the status light on the Trizium blinks shows the status of a connection. Also there are some AT commands defined in [5] to check on the network status of the module. Once the module is up on a network a fun test to verify the

operation is to send a text message. Listing 3.4 shows a sample dialog on sending a text message out.

Listing 3.4: SMS Message

---

```
//Set up for text message
> AT+CMGF=1<CR>
> OK

//Set destination address
>AT+CMGS="7035555555" <CR>

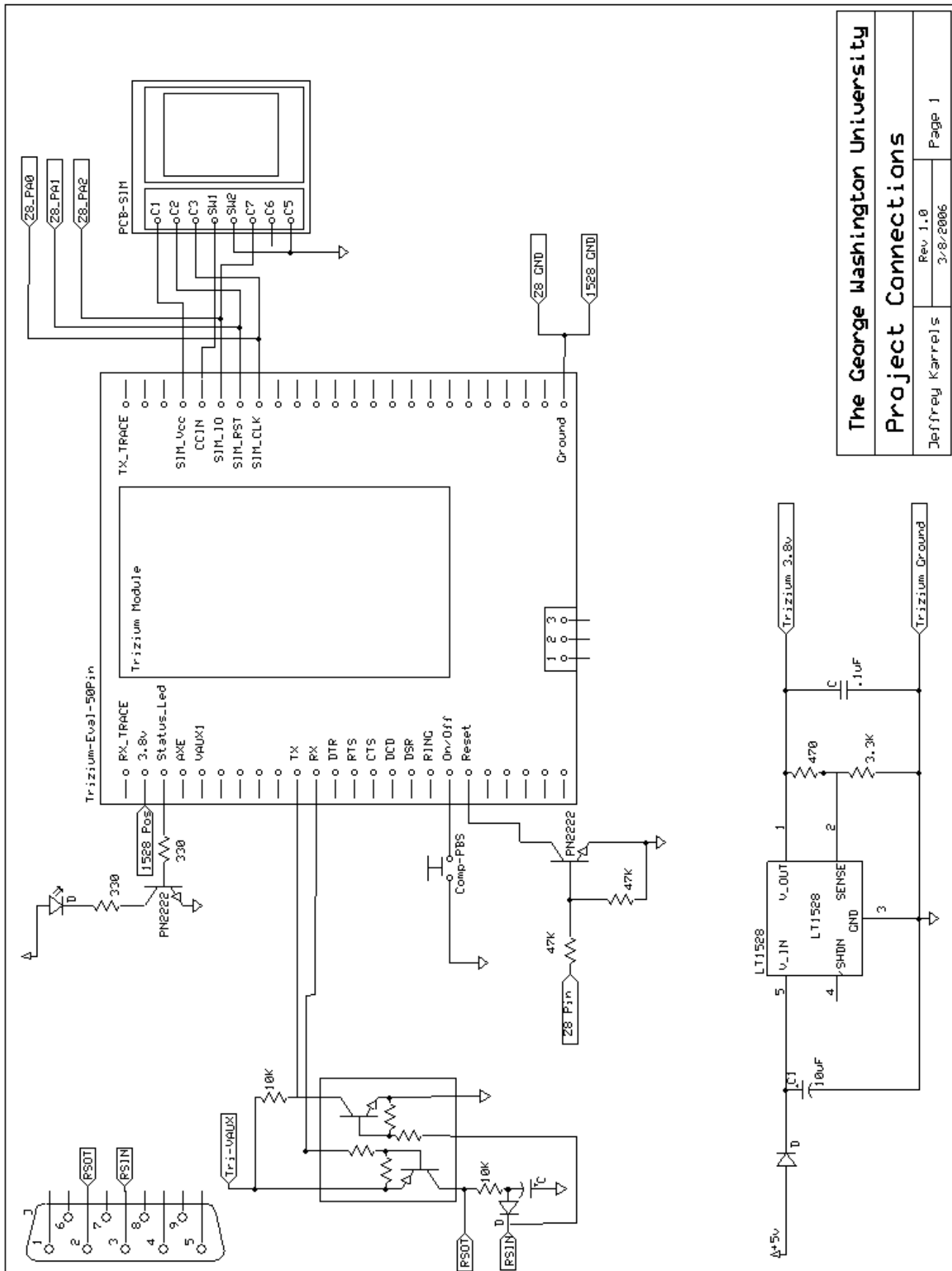
//Wait for '>' character
>

//Enter text message and end with CTRL-F
This is a test message<CTRL-F>
```

---

### 3.3.4 Trizium-Z8

As previously mentioned, it is a simple task to setup the Trizium UART to talk to the Z8. This was implemented, but later found out to be changing the values of the polling of the I/O line. So the method described in Section 3.1.2 was used.



# Chapter 4

## Findings

Section 3.3.1 described the process of obtaining data from the SIM card. This chapter will focus more on what that data was. It took a great deal of time to parse this data. After all, the data I was reading showed that there was 6 clock cycles per ETU while the specification stated that there was supposed to be 372. I never did get a chance to record the clock and I/O on an oscilloscope to verify its speed coming across, but I am confident that I was capturing the data in a manner that allowed for accurate parsing. I think that I just probably could not sample quick enough to get the 372 clock cycles. I am confident in the data due to [6] Section 6.4.1 stating that another method in determining the ETU is to use the initial character. The initial character defined in Section 4.2.1 can be used to measure the ETU. A third of the time between the first two falling edges in the Initial character can be used as 1 ETU. The following sections will describe the data that was found.

### 4.1 RESET

The first thing the Trizium does to the SIM card is go through a RESET. This is in order to setup the interface between the two devices. ISO 7816-3 [6] section 5

describes the RESET. This followed specification.

## 4.2 Answer to Reset (ATR)

After a RESET (*Section 4.1*) the card responds with an answer to reset (ATR). This answer gives some basic information about the SIM card.

### 4.2.1 Initial Character

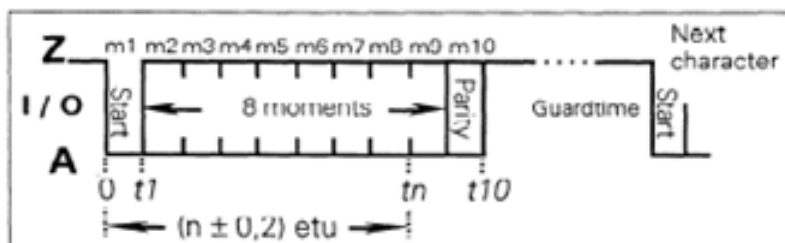


Figure 4.1: ATR Initial Character

The first byte in the ATR is none other than the initial character. Section 6.4.1 of [6] defines the Initial Character. Figure 4.1 shows the layout of this data. This can be one of two values. This is in order to setup the convention code of the all following bytes of data. That is, if a low state means logical 1 or 0 as well as where the MSB is located. The initial character that I read from the card was for the direct convention. The following is the hand parsing of the Initial Character.

```
// ATR
// Initial Character
// -----
// Character AZZAZZZAAZ sets up the direct
// convention where state Z codes value 1 and moment
// m2 conveys the least significant bit (LSB first). When
// decoded by direct convention, the conveyed byte is
```

```

// equal to 3B.
//
// |----- M1 Start Bit
// ||----- M2
// |||----- M3
// ||||----- M4
// |||||----- M5
// |||||----- M6
// |||||----- M7
// |||||----- M8
// |||||----- M9
// |||||----- M10 Parity
// *****
// 0110111001
// 0001110111 //0x3B (Start bit and Parity bit static, M2-M9 backwards)

//DATA
0x40D - 0xF8 //1111 1000 1
0x40E - 0x1F //000 11111 0//Start bit
0x40F - 0xFE //1 111111 0 11
0x410 - 0x07 //00000 111 0
0x411 - 0xFF //111 11111 1
0x412 - 0xFE //1 111111 0 11
0x413 - 0x00 //00000 000 0
0x414 - 0x3F //00 111111 01
0x415 - 0xFF //11111111 1

```

## 4.2.2 T0

T0 follows the initial character. T0 is defined in Section 6.4.2 of [6] and tells the presence of the bytes to follow. The following is a hand parsing of T0.

```

// ATR
// T0
// -----
//
// |----- M1 Start Bit
// ||----- M2 B8 TD(1)
// |||----- M3 B7 TC(1)

```

```

//  ||||----- M4 B6 TB(1)
//  |||||----- M5 B5 TA(1) Present
//  |||||----- M6 B4 |
//  |||||----- M7 B3 ||-Number of historical Bytes present [6]
//  |||||----- M8 B2 ||
//  |||||----- M9 B1 |
//  *****-- M10 Parity
//  0000101101 Direct Convention
//  0011010001 Pre-Transformation

```

```

0x477 - 0xC0 //1100 0000 0
0x478 - 0x03 //0000 0011 0
0x479 - 0xFF //1111 1111 1
0x47A - 0xC0 //1100 0000 10
0x47B - 0xFC //1111 1100 1
0x47C - 0x00 //0000 0000 0
0x47D - 0x00 //0000 0000 00
0x47E - 0xFF //1111 1111 1
0x47F - 0xFF //1111 1111 11

```

### 4.2.3 Further bytes

The rest of the ATR is defined in [6], and that document can be referenced for those details. Figure 4.2 displays an example of ATR. This table was found in [7] and although the data is not the same as what I have captured, the format is identical.

## 4.3 APDU

Once the ATR is taken care of, the ME and the SIM interface through application protocol data units (APDUs). Figure 4.3 defines the format of an input APDU. The *Cingular.txt* that accompanies this document contains a hand parsed example of the ATR with some APDUs being interchanged amongst the ME and SIM. That is all that I will describe in this document though.

Byte	Value	Description
TS	3B	Value for the initial character with the direct convention A(ZZAZZZAA)Z
T0	16	<ul style="list-style-type: none"> <li>• 1 means that TA1 is transmitted</li> <li>• 6 means that there are 6 historical characters</li> </ul>
TA1	11	Value used before personalization (9600 bauds at 3.57MHz) <ul style="list-style-type: none"> <li>• 1 means that F=312</li> <li>• 1 means that D=16</li> </ul>
	94	Value used after personalization (55800 bauds at 3.57MHz) <ul style="list-style-type: none"> <li>• 9 means that F=512</li> <li>• 4 means that D=8</li> </ul>
HC1	9C	Component code
HC2	01	Hardmask code
HC3	01	Hardmask version
HC4	00	Softmask code
HC5	80	Softmask version
HC6	XX	Additional character to know which functionality is enable/disable during personalization

Figure 4.2: Sample ATR table from [7]

Reader	Cla	Ins	P1	P2	P3		Data	
Card						Ack		SW1 SW2
Length (byte)	1	1	1	1	1	1	Lgth	2

Figure 4.3: APDU format from [7]

# Chapter 5

## Conclusion

### 5.1 Retrospective

When I was looking for a project idea, my first idea was to build a tracking device. That would have consisted of a GPS and GSM module with the Z8 interfacing the two together. I ended up actually buying a GPS module the other week and am glad I did not do that as my project. That interfacing of the two is fairly simple. It took a little under a day to figure out how to use the GPS module to get a location to SMS message through the GSM module. Both connections could be done through the UARTS, and the GPS ran off of a standard 3.3v.

In retrospective, I think I just did not have the clock cycles to do what I wanted. The SIM card operating at 3.25Mhz was just too much. I really wanted to get a lot further than I actually did, but I guess that is a pretty common saying. There is nothing that I would really change about what I did. I feel as though there was a very broad range of tasks that needed to be accomplished in order to get where I ended up. From building actual hardware to trying to debug with an oscilloscope. I am not going to say if I had more time... I think I started early enough on the project. I had my hardware to work with starting with the second class of the semester.

In general, I am happy with my outcome. I have taken something that I knew absolutely nothing about, and achieved my main objective to a degree. If I had to start this project from what I know now though, I would have chosen a faster microprocessor or perhaps attempted a FPGA implementation of this. Also, I started by creating a fairly large complex program in order to capture the data. I was getting data, but was unable to make any sense of it. I later realized that I need to slim down my code because I was missing data due to the speed in which it was coming across the line. I can say, that I have a far greater respect for 3.25Mhz.

## 5.2 Further work

I tried many things to accomplish further parsing of the IO data on the Z8. Basically this would consist of only storing the 8 bits of a byte in RAM instead of the entire sample. I believe that it does not have the computing power to do those tasks though. I was more comfortable with the Z8 just capturing all of the data and outputting it. This would be more versatile in case of a SIM that did something different than the two that I tried. This way one could write a parse routine that would parse the captured data and see exactly what happened on the line. So the next step I would take would be to write a parse routine on a desktop machine.

The second issue I had was the size of the RAM on the Z8. I am thinking that either some external memory needs to be used, or the implementation of a ring buffer or something similar in order for continuous capture of I/O data.

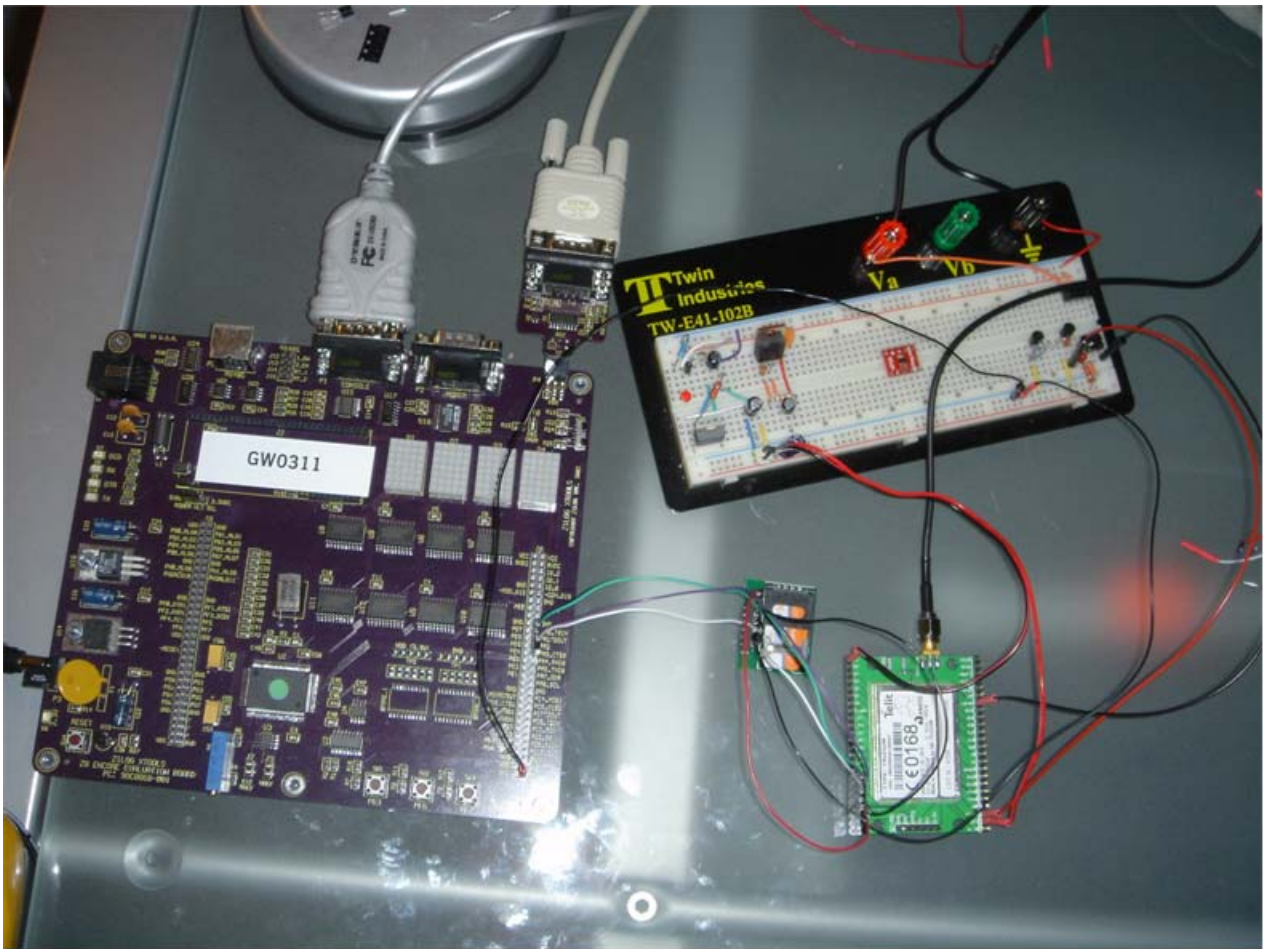


Figure 5.1: Complete Project Photo

# Appendix A

## Main Hardware Components

### A.1 Z8 Encore!

Item	Vendor	Part Number	Qty	Price
DEV KIT FOR Z8 ENCORE	Digi-Key	269-3125-ND <sup>1</sup>	1	\$49.95

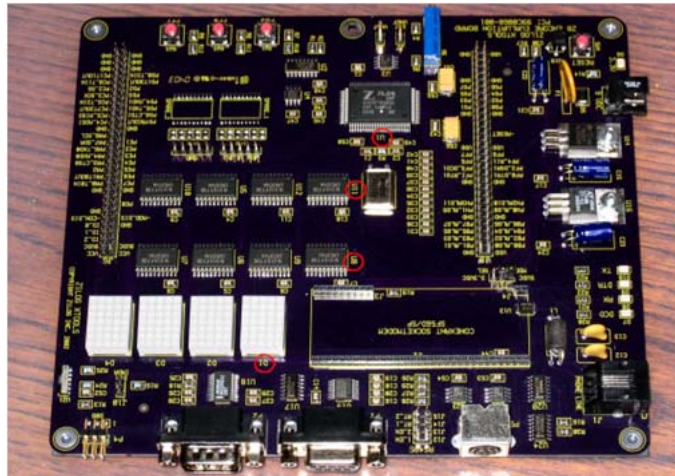


Figure A.1: Zilog Z8 Encore! Demo Board

## A.2 GSM module

### A.2.1 Demo Board

Item	Vendor	Part Number	Qty	Price
Trizium Evaluation Board	Spark Fun <sup>2</sup>	Trizium-Eval-50Pin	1	\$119.95

The Trizium Evaluation Board from Spark Fun Electronics pulls the necessary pins from the Trizium module to header rows. This Trizium module has 3 bands of operation. The {900, 1800, 1900}Mhz bands. The 900 and 1800 Mhz bands are for European use, but the 1900Mhz band works with the North American city networks. I specify city due to the rural areas using a lower 850Mhz.



Figure A.2: Spark Fun Electronic's Trizium Evaluation Board

### A.2.2 Antenna

Item	Vendor	Part Number	Qty	Price
Tri-band GPRS Cellular Antenna SMA	Spark Fun	ANT-Triband	1	\$12.95

## A.3 SIM card

### A.3.1 Breakout board

Item	Vendor	Part Number	Qty	Price
Breakout Board for SIM Cards	Spark Fun	PCB-SIM	1	\$14.95



Figure A.3: Tri-Band GSM antenna



Figure A.4: SIM card Breakout Board

## A.4 GPS module

### A.4.1 Demo Board

Item	Vendor	Part Number	Qty	Price
20 Channel ET-301 SiRF III Engine SMD	Spark Fun	GPS-ET301	1	\$64.95

Section A.2 describes the GSM evaluation board. This component also supports the addition of a GPS module on the bottom side of the board. The ET301 solders directly on to this board.

### A.4.2 Combo Antenna

Item	Vendor	Part Number	Qty	Price
Covert Strip Tri-Band Antenna	Spark Fun	ANT-CS	1	\$39.95



Figure A.5: 20 Channel ET-301

If the GPS module is desired, a GPS antenna also needs to be used. Spark Fun stocks an antenna that has a dual purpose. GSM and GPS.



Figure A.6: Covert GPS and GSM antenna

# Appendix B

## Supporting Hardware

### B.1 Voltage Regulators

#### B.1.1 LT1528

Item	Vendor	Part Number	Qty	Price
Linear LT1528	Digi-Key	LT1528CT-ND	1	\$6.50

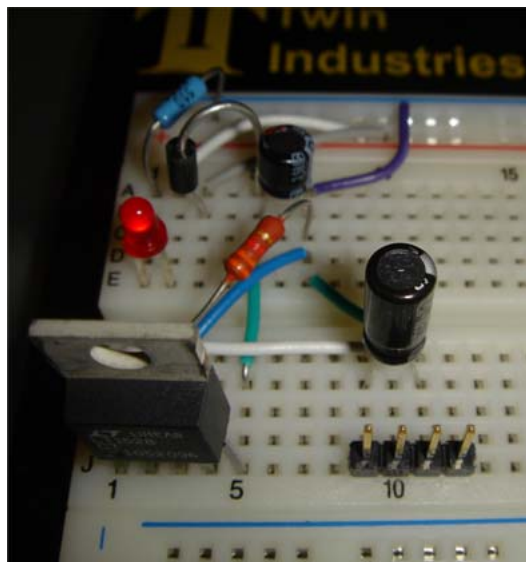


Figure B.1: 3.8v Voltage Regulator Circuit

### B.1.2 LM317

Item	Vendor	Part Number	Qty	Price
Linear LM317	Digi-Key	LM317LZ	1	\$0.70

## B.2 Level Shifters

### B.2.1 MAX 232

Item	Vendor	Part Number	Qty	Price
Maxim MAX3232	Digi-Key	MAX3232ECPE-ND	1	\$5.04

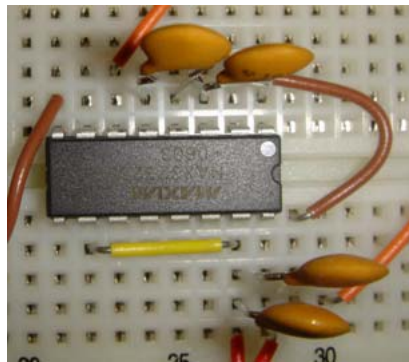


Figure B.2: MAX3232 on the Breadboard

### B.2.2 XN04312

Item	Vendor	Part Number	Qty	Price
Transistor Array NPN/PNP w/ Built-In Resistors	Spark Fun	IC-TransArray	1	\$1.95

### B.2.3 SparkFun's Premade XN04312 level shifter

Item	Vendor	Part Number	Qty	Price
RS232 Shifter SMD	Spark Fun	PType-Shifter-SMD	1	\$9.95

### B.2.4 Breadboard

Item	Vendor	Part Number	Qty	Price
Breadboard SMT adapter	smt-adapter.com	10TS05-D3-SMT-S	1	\$5.00



Figure B.3: Spark Fun's RS232 Converter



Figure B.4: SIM card and SMT Breadboard adapter with XN04312

# Bibliography

- [1] 3GPP TS 11.11, *Specification of the subscriber identity module - mobile equipment (sim - me) interface*, 3rd Generation Partnership Project, v8.13.0 ed., 06 2005.
- [2] ETSI, *Ts 102.221*, ETSI, v7.1.0 ed., 05 2005.
- [3] Telit Communications Group, *Telit gm682 and trizium family at commands description*, DAI Telecom S.p.A., 80264st10013a rev. 1 ed., 05 2005.
- [4] ———, *Telit trizium hardware user guide*, DAI Telecom S.p.A., 1vv0300659 rev. issue 3 ed., 08 2005.
- [5] ———, *Telit trizium software user guide*, DAI Telecom S.p.A., 1vv0300660 rev. issue 2 ed., 07 2005.
- [6] ISO, *Iso/iec 7816*, ISO, e ed., 05 1997.
- [7] SchlumbergerSema, *Simera card user guide*, SchlumbergerSema, 2001.