EC535
May 2013

RFID Transaction Module
Team: Embedded Bologna

Ian Bablewski: icb@bu.edu
Calvin Flegal: calvin.flegal@gmail.com
Bryant Moquist: bmoquist@bu.edu

Abstract:
This project implements an RFID transaction module that identifies users, reads balances, conducts transactions, and adds value to the current balance. Users interact with the module and receive information using buttons and an LCD screen. The charge amount, add new value amount, and mode are determined by three buttons interfaced via kernel module.

Introduction:
The use of embedded and large-scale computer systems to manage critical and peripheral tasks in commercial, industrial, and pedestrian applications frequently requires a means for users to interact with a given system. Radio frequency identification technology (RFID technology) offers a low-power, flexible solution at a reasonable price. The technology uses passive or active tags that can store data and be read or written to by RFID modules. The technology's applications range from tracking an automobile's production status in an assembly line to identifying livestock and wild animals. One recent example is Los Angeles's new ExpressPark program, which intends to leverage the power of RFID technology in conjunction with wireless sensor networks to create an intelligent parking system. The system will not only notify users of available parking spaces via a mobile app but also augment the city's ability to price parking spaces based upon demand. The use of RFID technology and its future potential to address a vast array of problems underlie our group's choice to explore this technology for the capstone project of this course.

Our project is an RFID transaction module capable of identifying individual RFID tag users, reading the tag's available monetary balance, and conducting transactions that add or deduct value from the current balance. The system uses a Parallax RFID module and tags, a Gumstix Verdex with expansion boards, buttons, and a Samsung LCD. Users can select three modes of system operation and adjust charge/add-value amounts using three buttons interfaced with the Gumstix board via kernel module. A user space program reads the device driver and sends commands to the RFID module using a state machine. Tags are both read and written with the RFID module depending on the type of transaction selected. The LCD screen is updated simultaneously to offer the user feedback on the status of his or her transaction.

The system successfully implements the operations described above and meets the primary objectives of our project namely to create a working RFID transaction module. Further improvements beyond the scope of this project could include more robustly packaging the components and integrating multiple transaction modules across a network.

Design Flow:

Task	Responsible party	Detailed description
Project proposal presentation	Ian Bablewski	Wrote the presentation and presented the project to the class.
Researching UART protocol and device drivers	Bryant Moquist Calvin Flegal	We initially considered implementing our own UART kernel module and spent a significant amount of time working through complex examples (e.g. “LDT - Linux Driver Template"). Eventually, we found that we could use already written Linux libraries and ttyS2 with the appropriate initialization settings to communicate with our device.
Learning to send commands to the RFID module, diagnosing hardware issues, and scoping the output signals	Calvin Flegal Bryant Moquist	The RFID commands involve a mix of ASCII, decimal, and hexadecimal characters. This made interacting via command line somewhat tricky as escape characters must be used occasionally to get each part of the command in the correct form. Using an oscilloscope, we captured the bits sent on the Gumstix transmit and receive lines. We discovered that we received a faulty RFID module as it could not transmit to the Gumstix after successfully receiving a command. We spoke with a Parallax engineer who agreed with our assessment and sent a replacement module.
Designing the RFID/Gumstix interface, state machine, and code	Bryant Moquist Calvin Flegal	With our new working module, we designed a program with command line options and debugging flags. Driven by a state machine, the program identifies tags and value, updates balances, and makes charges to the balance. Eventually, we integrated a kernel module to adjust settings based upon button presses.
Kernel module for button interface	Bryant Moquist Calvin Flegal	We designed a basic kernel module so that the user space program can read the button states.
Qt code and LCD operation	Ian Bablewski	Researched the Qt interface, worked through sample applications, and built a Qt program to display information about the RFID transaction unit.
System integration and debugging	Ian Bablewski Calvin Flegal Bryant Moquist	We integrated the UART initialization settings and RFID module state machine into the Qt code. We then tested and debugged the system until it was fully functional.
Boot load of kernel	Calvin Flegal	Experimented the Gumstix startup and

module and user space code		implemented a successful start of the entire RFID transaction unit on power up of the Gumstix board.
Project progress report and demonstration presentation	Bryant Moquist	Wrote the progress report and the final project demonstration presentation.
Final Report	Bryant Moquist Calvin Flegal Ian Bablewski	Wrote a comprehensive report of the project.

System Design:
Our system includes the Gumstix Verdex with expansion boards, a Samsung LCD, a Parallax RFID module with tags, three buttons, a battery pack, and a power cord for the Gumstix. The RFID module requires a power range between 4.5 – 6.0 volts so we included a separate power supply for the module and provided a common ground with the Gumstix power supply level.

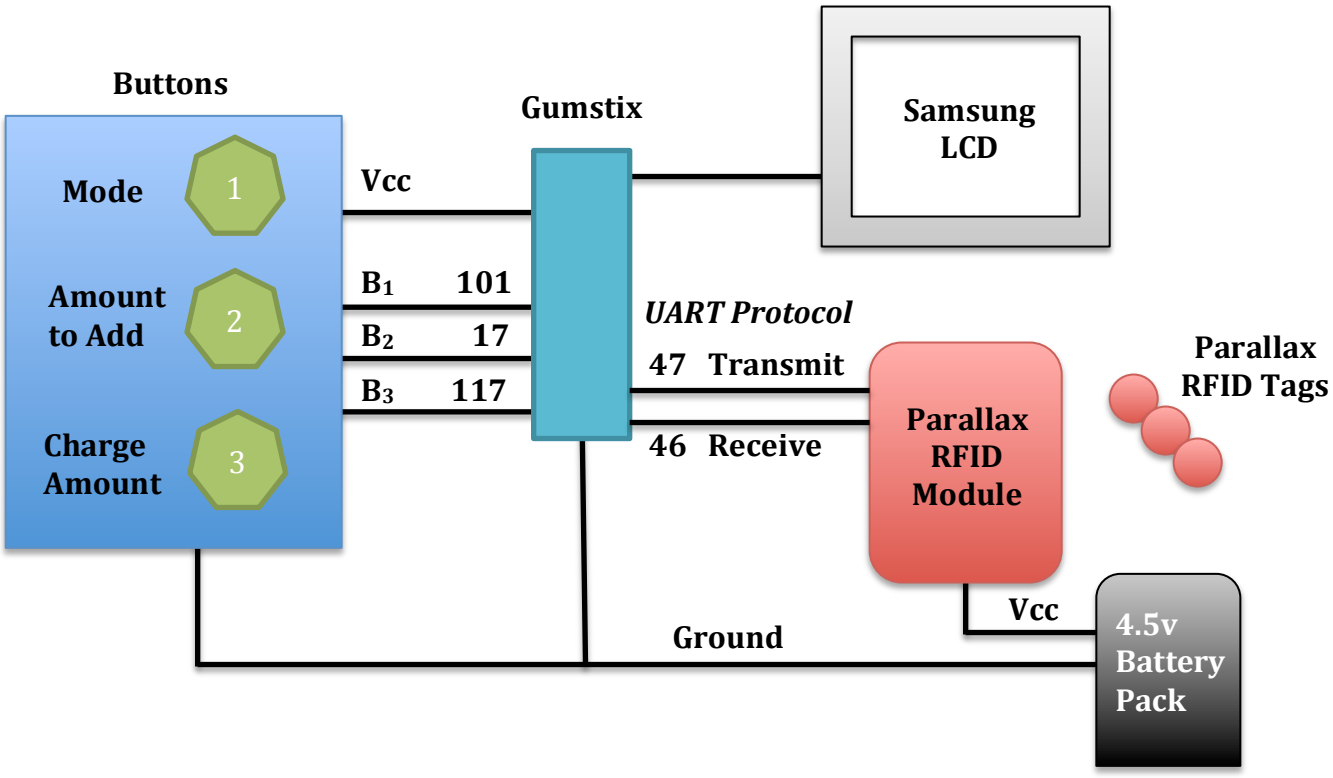
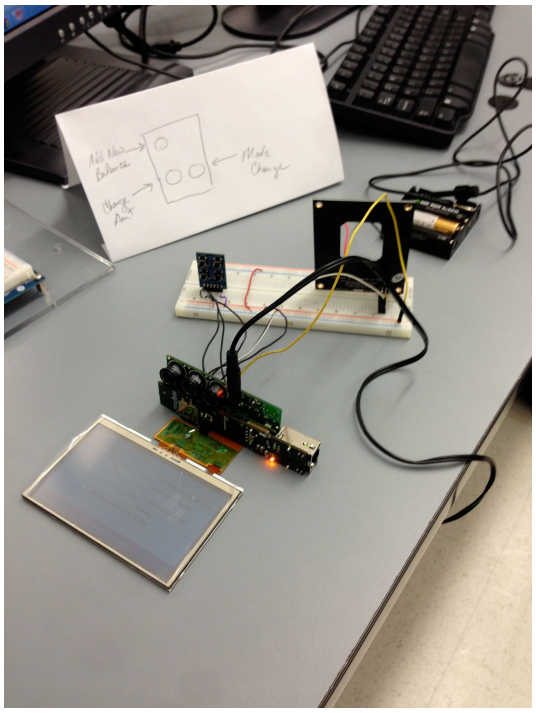
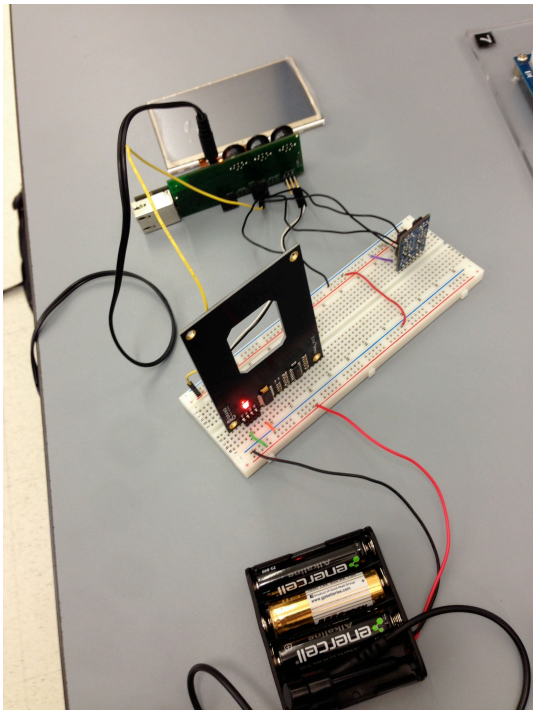
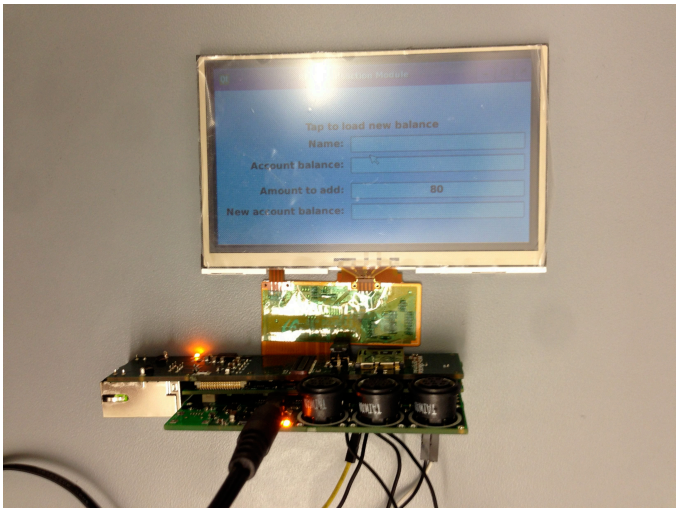


Diagram: System layout of the RFID module (above). 46, 47, 101, 17, and 117 are GPIO pins on the Gumstix board.







Photographs of the working RFID transaction module.

Research:
Our initial research focused on understanding the RFID module's UART serial communication protocol. Specifically, the communications interface must be configured at 9600 bps, 8 data bits, no parity, 1 stop bit and least significant big first or "8N1" for short. We then proceeded to see how we could configure the Gumstix to communicate with those specifications.

Accessible example tutorials on how to implement UART on the Gumstix/Linux were surprisingly difficult to find. We even initially thought that we might need to implement our own UART kernel module. We found an open source device driver

kernel template by Constantine Shulyupin that implemented UART. The code was very instructive but also quite complex.

In the course of our research, we also found IBM's Command Reference manual for stty. With this, we realized we could configure an already implemented UART from the command line in user space and communicate with the RFID module via ttyS2.

Finally, we found Dave Hyland's open-source code, "sertest.c", which provided a clear, well-commented example of how to use UART on the Gumstix. The commands used to properly set and open UART communication in Hyland's example served as a key foundation of our system's code.

Command line debugging and hardware diagnosis:
After finding the IBM reference and reading the relevant parts of the ARM/Xscale reference manuals, we worked to set-up STUART (standard UART) on the Gumstix accessed via etc/ttyS2 from the command line. We set up ttyS2 with the proper flags using the stty tool and the list of flags provided through the IBM Infocenter web application. Then, we redirected echo commands to ttyS2. Initially, we could not get the RFID module to respond to our commands, so we connected an oscilloscope to the transmit line of the Gumstix.

The protocol of the RFID reader requires all commands to be preceded by "!RW" in ASCII, followed by a command opcode in hexadecimal, and then an address in decimal. The module has a two color LED that changes from green to red upon receiving a command and will remain red for several seconds while looking for an RFID tag. Eventually, we managed to get the RFID module to look for a tag with the following command line entry: "echo –e "\x21\x52\x52\x01\x20" > ttyS2". The –e flag allows for hexadecimal entry. The first three bytes are "!RW" in hexadecimal format. The next byte is the opcode for reading, and the final byte is hexadecimal for 32. This command tells the device to read address 32 of the tag, which is the tag's serial number.

The RFID module successfully received the above command as its LED turned red and even responded to tags placed near the module by flipping the LED back to green. However, once we attached an oscilloscope to the output of the module, we realized that the module transmitted no data back to the Gumstix. To find a solution, we contacted Parallax, the manufacturer, and an engineer at the company confirmed that we likely had a faulty module. The company promptly sent a replacement module, which properly responded to commands.

User space program:

UART initialization:

Our user program initializes UART serial protocol with the termios.h library by setting up a termios struct entitled attr, a 9600 baud rate, and a path to /dev/ttyS2 in the form of c-string. We open a serial port called gPortFd with the device path allowing reading and writing and using non-blocking mode. Once the port is open we turn off the non-blocking and use tcgetattr(gPortFd, attr) to get the parameters associated with the terminal referred to by the file descriptor gPortFd and store them in the termios structure referenced by attr. We then enter raw mode by calling cfmakeraw(&attr), which allows input to be available character by character. From here, we disable the modem control lines, enable the receive, and set the baud rate. Then, we set attr, the termios structure, to discard any queued input.

Qt set-up and description:

Qt is a cross-platform widget-based application framework for developing graphical user interfaces. Our application called for a multi-paged GUI that allowed the user to interact with different pages depending upon the mode of the application. Our first page is a welcome page that alerts the user that the program is starting and after a brief time period switches to the next page based upon the application's current mode. Our other pages include a read balance page, a payment page, and an add-value page. The read balance page displays the customer's name and current balance information. The payment page displays the customer's name, prior balance, deduction amount, and new balance. Last, the add-value page displays the customer's name, prior balance, amount to add, and the new balance.

Our GUI uses the QTFormLayout with integrated QLineEdit and QLabel widgets. The widgets map into the QTFormLayout's adaptive platform, which displays the application's information. The LCD screen's widget layout is updated with a finite state machine with input from the button module. Global variables are passed with a similar method to the proper labels in order to paint and clear the widget inputs.

State machine:

Our program begins in a state where it regularly sends a command to the RFID module to read the address of the unique serial number associated with a token. Once the token is read, the serial number's identification is processed by the program and matched to names stored in a struct. After outputting the name, it advances to the next state where the balance is read from a different address on the tag. After reading the balance, the next state is determined by the current mode setting. Three modes are available: read serial number, make a payment, and add value. In the case of reading a serial number, the transaction complete state follows. In the case of make a payment or add value, the respective states adjust the current balance by adding or deducting value, and the new balance is written to the tag. The process then moves to the transaction complete state where the balance is read again from the tag to confirm a successful write. A three second sleep is implemented in transaction complete to ensure that users do not conduct accidental

transactions. Last, the state is reset to the initial state where the RFID module polls for a serial number.

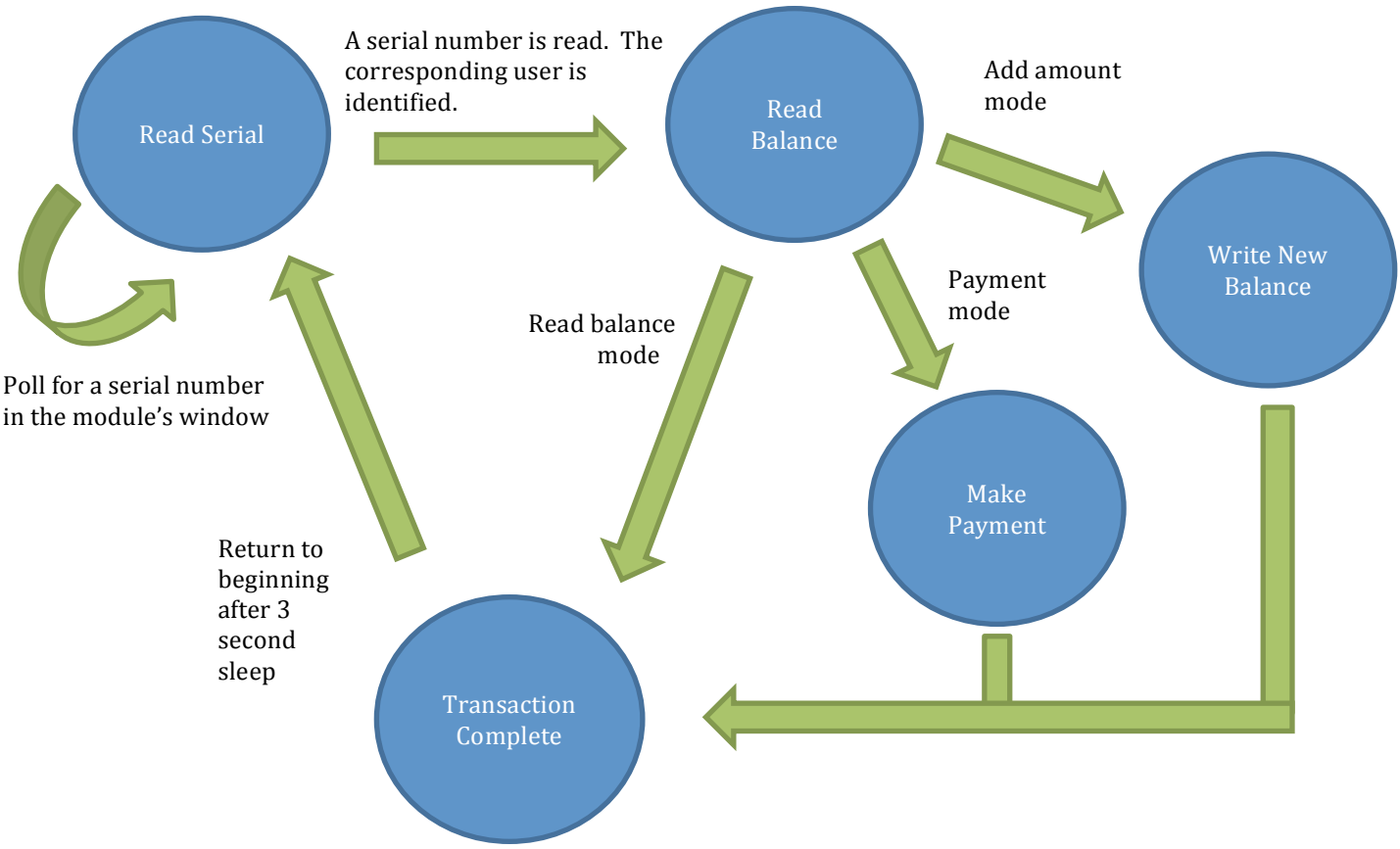


Diagram: The user space program state machine that sends commands to the RFID module.

Reading kernel module:
We regularly check the state of the three buttons by opening a file pointer to our kernel module in "/dev/kRFID" and using sscanf to get three values that denote the mode setting, charge setting, and new value setting. The charge setting and new value setting are multiplied by 5 and 20, respectively, to set the charge and add-value amounts used in the user space program.

Safeguards:
We limit the highest balance on a token to 1000 and do not allow for negative values. If either case is encountered during the program, either 1000 or 0 is written to the tag depending upon the scenario. We also implement a sleep at the end of the transaction complete state to ensure that users do not accidentally conduct multiple transactions by holding their tag in front of the module for too long of a time.

Integration:
Our RFID application and Qt GUI application were developed simultaneously. To integrate the programs, we ported the RFID module code to C++ classes embedded in our existing Qt application code.

Kernel module:
For our system to be operational without command line input, we implemented a kernel module that interfaced three buttons with our system. The buttons cycle through three modes, ten payment settings, and ten add-value settings. The kernel module only required a read function so our user space program could regularly get button state information.

Our read function prepares a string with each value separated by a space. After the buffer is built, it is copied to the user space after which we increment the file offset pointer.

Boot-up:
To mount our kernel module and run our user space program before the login, we wrote a bash script and placed it in /etc/init.d/. Once the script was written and executable, we added a symlink to the script in the /etc/rc.5 folder with a low priority so that it wouldn't preempt other modules. Our script had a nice fail-safe because one of its lines sets an environmental variable to an SD card path. If we need to reach login, we can simply remove the SD card, which throws an error, and allows the system to continue to the login prompt.

Summary:
Our RFID transaction module conducts identification of tags, reads balances, conducts transactions, and adds value to the current balance. Information is displayed on an LCD screen and users are able to interact with the module with three buttons interfaced with the main program via a kernel module. The project met our initial design objectives and included additional features such as adding value to the current balance. Further enhancements beyond the scope of this project could include more robustly packaging the components and integrating multiple transaction modules across a network.

References:

ASCII table:
http://www.asciitable.com/

Gumstix Wiki:
http://docwiki.Gumstix.org/index.php?title=Gumstix_UARTs

IBM Infocenter:
http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp?topic=%2Fcom.ibm.aix.cmds%2Fdoc%2Faixcmds5%2Fstty.htm

IBM Command Reference for stty:
http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds5/stty.htm

LA ExpressPark Program:
http://www.laexpresspark.org

"LDT - Linux Driver Template" by Constantine Shulyupin <const@makelinux.net>
https://github.com/makelinux/ldt/blob/master/ldt.c

Qt Project Documentation:
Qt-project.org/doc/

"sertest.c" by Dave Hylands <dhylands@gmail.com>. Implements a sample program for accessing the serial port. http://svn.hylands.org/host/sertest/sertest.c

"The latest UART driver model for porting serial devices as standard Lnux serial ports" by Jim Chen <jim.chen@hytec-electronics.co.uk>. June 14, 2011.
www.aps.anl.gov/epics/meetings/2011-06/sys/data/46/UART_device_driver_model.pptm

Wikipedia entry on RFID:
http://en.wikipedia.org/wiki/Radio-frequency_identification