6.111 FA2017 Final Project Proposal

Aneesh Agrawal and Paige Studer

Overview

Dorm rooms, mailboxes, meal swipes - MIT students have many valuable possessions they need to keep secure in a hostile world. MIT has recently rolled out the Duo 2 factor authentication system to increase security, but at the cost of increased friction due to the requirement of a smartphone or USB Yubikey device. The goal of our project is to revolutionize 2FA at MIT by accepting MIT ID cards as a convenient alternative second factor. To illustrate this, we will create a secure “joke database” for MIT students. If a student is able to complete a login, then he or she will be rewarded with the display of a randomized joke. The system will first ask for a username and password, and then ask for an MIT ID card to be tapped before determining whether or not the user is authorized to see a joke from the database. This project will utilize a basic analog circuit to read the MIT ID via passive RFID, as well as RS-232 (emulated over microUSB) serial communication to transfer data from a computer to and from the Nexys4 board. An overarching finite-state machine will control the authentication and authorization flow, determining when to ask for a username, password, and ID tap, and validating whether or not a user is authorized based on modules and custom databases we create. Finally, all of our information will be displayed on a monitor via XVGA resolution using the 2D graphics techniques of sprite and color maps.

Design

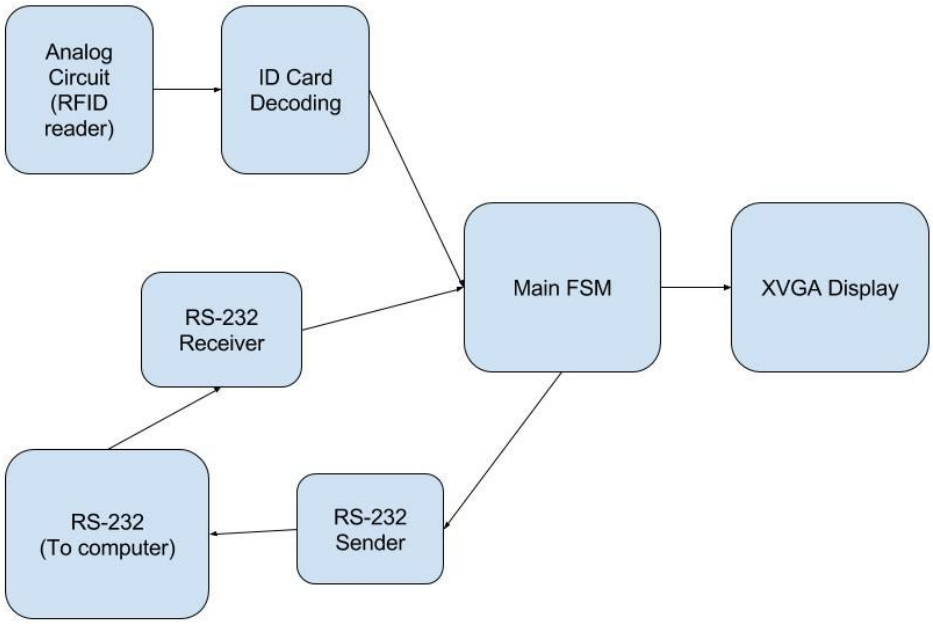

Figure 1: Overall (simplified) Block Diagram

The system has 4 main parts: an ID card reading pipeline, RS-232 input and output modules, the main control FSM, and an XVGA rendering subsystem.

ID Card Pipeline

This part of the design is responsible for actually reading MIT ID cards and deciphering the RFID data on them. This is a one-way pipeline, where data flows continuously and in one direction (although the main FSM will be able to enable/disable this segment of the circuit completely to conserve power when not in use). Most of these modules will have no local state but simply transform their inputs in a combinatorial way, buffering them for the next stage to avoid glitches. This cuts down complexity by eliminating feedback paths and minimizing local state, and makes each module easy to test in isolation.

This part of the circuit will require both analog and digital subsections to read and decode the RFID code on the data. Starting with a 125Khz square wave carrier generated by the FPGA, an analog circuit will be responsible for boosting and emitting the RFID signal, as well as reading, detecting and filtering the input signal. After passing through an ADC, the data

stream will pass through a phase-shift keying decoder, a command format recognizer, a module that reverses the proprietary FlexSecure protection, and a BCD convertor to turn the binary data into decimal data suitable for display.

This part of the design is fairly low risk as there is a wealth of information available on generic RFID protocols including sample circuit designs, and in particular there are 2-3 previous MIT student projects from various classes that have examined reading and decoding MIT ID card data in specific. We will be able to leverage and build on the work done in those classes to focus on the unique digital components of our design.

RS-232 Serial Input/Output Pipelines

To enable input of the username and password as the first factor, the emulated RS-232 interface on the Nexys4 will be used to interface with an external computer, where prompts for data and inputs will be displayed and entered in a serial terminal. As RS-232 is a full-duplex protocol, this will be enabled by two isolated independent pipelines that can operate at the same time, one that receives input and sends it to the main FSM, and a second that receives data from the main FSM and sends them out. These modules will be fairly similar to those used in Lab 5b for infrared communication, and require no additional hardware or resources (i.e. no BRAMs), making this a low risk part of the project. These modules will be parameterized along various dimensions (baud rate, data format (number of data, stop and parity bits, etc.) to enable ease of testing.

Main FSM

The main FSM keeps track of the state of the login flow. The user will be sequentially shown screens asking for their username, then for their password, then for an ID tap, and finally whether or not they are authorized or not authorized. The initial state will be asking for the username. Once a username has been entered, it will move onto asking for the password. Once the password has been entered, an ID tap will be asked for. Once all of these steps have been completed, we will check the pre-written database to see if the person is authorized. If so, we will reward the user with a random joke from the database. If the “escape” key is entered via the serial link at any time, or the user walks away while the system is waiting for input and a timeout expires, the FSM will reset the login flow, starting again at the asking for a username stage. This requires a joke database BROM and a valid Id database BROM, each of which should take at most 20Kbit for ~10 entries each.

The FSM is currently set up to be a 2 factor authentication system, where all of the information is entered together and checked simultaneously. As a stretch goal, the FSM could be changed into a 2 step verification system where each step would be checked before the next step could happen, enhancing the user experience although providing a small reduction in theoretical security. Another extension would be to enable line editing of serial inputs (i.e. usernames and passwords), allowing keys such as backspace, delete, and the arrow keys to work properly.

Renderer

The renderer subsystem is where information from the FSM is interpreted and used for display on a screen, with a pipeline that converts pixel addresses to pixel values. For each pixel address, first,the FSM state will be determined from the main FSM. Once the state is determined, we will access the pixel data from the appropriate spritemap ROM or the appropriate background ROM (various states will correspond to different backgrounds), depending on the layout of that particular state. We will then send the pixel value through the colormap to determine the actual pixel color data for that pixel, and output it for VGA display. This will use multiple block ROMs on the Nexys4; overestimated sizes are as followed. One background will be pure black (requires no space), and the authorized and unauthorized backgrounds will require ~1500Kbit each at full XVGA resolution with 4 colors available. Additionally, there is a spritemap which stores ~70 64x64 black and white character sprites + 1 200x200 4 color logo (MIT logo) which takes about 400Kbit. Finally, a colormap from 8bit color to 12bit color needs to store 2^8 = 256 values of (8 bits + 12 bits =) 20 bits each, or 5Kbit.

For a stretch goal, we would like to have a photo library of authorized persons. For example, if John Doe was an authorized user, a picture of his face would appear on the screen. Pictures take up more memory than sprites, so in this case we would need to use an SD card to store the images, modify the main FSM to load them on demand, and cache images in a BRAM as they are used, either one at a time or multiple using an LRU algorithm. Another possible extension is introducing double buffering to our display architecture to prevent screen tearing.

We have already used XVGA control signal generating modules so it is low risk, but the integration with various types of graphics BROMS and compositing behaviors (e.g. double buffering) is new territory and contributes risk.

Implementation

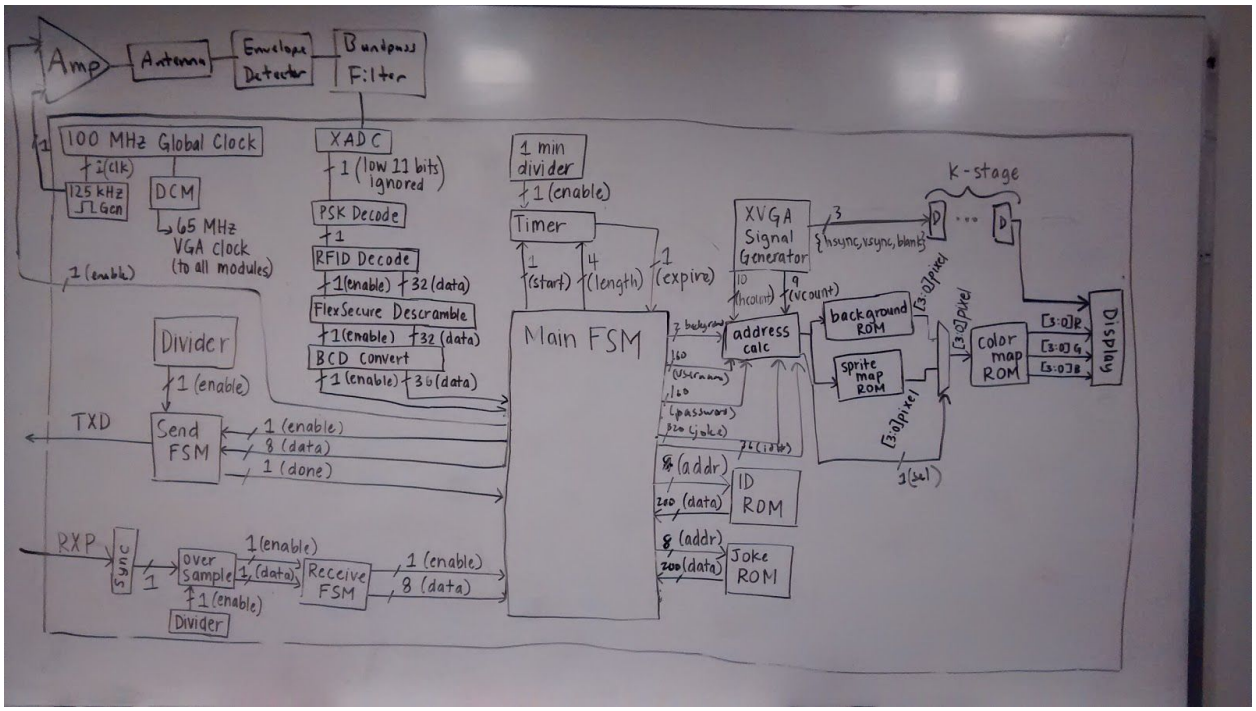

Figure 2: Complete Block Diagram

There are three main clock domains used: one is the real world/analog circuitry, which is clockless. The second is the global 100MHz FPGA clock, which is used to generate the other timing signals like the 125KHz RFID carrier and the VGA clock. The third and main clock is the 65MHz XVGA clock, which is needed to provide the VGA data at the right speed. To simplify the implementation, the 65MHz clock will be used for most modules on the FPGA, with the 100MHz clock only being used with the clock manager and to generate external signals to prevent the need for cross-domain synchronization on the FPGA.

ID Card Pipeline

RFID Carrier Frequency Generation

The MIT ID cards use a 125 KHz square wave carrier wave, which will be generated on the FPGA. This is a fairly low frequency signal, in fact too low to be synthesized by the clock management modules of the FPGA, and will be a written as a simple counting Verilog module.

This module will take the 100MHz global FPGA clock, a reset line, and an enable signal as inputs, and output the 125 KHz square wave as an output, which will be tied to a user I/O pin on the FPGA. This module will be parameterized for ease of testing, and will be accompanied by a simulation testbench; it requires no memories or external components.

Analog Circuitry

An analog circuit will be used to actually read the data embedded in MIT ID cards. These cards use passive RFID, in which the reader continuously emits a carrier frequency, and the card contains a small circuit which is powered by the carrier (i.e. contains no internal power source) and returns a backscattered signal, which can be read by monitoring the current draw of the antenna. The circuit will contain a few stages:

- Amplification: Use an LM18293 push-pull driver to amplify the square wave carrier to provide increased energy to allow for a reasonable reading range and enable powering the fully-passive card.
- Antenna: An RLC circuit is used to create the antenna. The LC components will be chosen for a resonant frequency of 125Khz, and a 1 ohm resistor will be used for an approximate Q factor of 10. This will require winding custom antenna loops out of enamel wire.
- Envelope detector and filtering: Assist demodulation of the RFID data by using an envelope detector, and add a band pass filter (low pass and high pass cascaded). The XADC is already fairly fast, but doing this step in the analog circuitry is simple and cuts down on complexity on the FPGA.

The LM18293 is chosen as the amplifier because it is already available in the lab; 125Khz is a fairly low frequency which is within the slew rate and response rate of this chip. Most of the other components are available as passive jelly-bean components in the lab, with the exception of the custom-wound wire loop inductive antenna.

ADC

The on-board Xilinx ADC (XADC) IP will be used to convert the 1 wire analog signal to a 1 bit digital signal. It requires an input in the range from 0 to 1 volts, which will require input clamping in the analog circuitry. The XADC has a sample rate of 1MSPS and 12-bit resolution, which is adequate to oversample the 125Khz signal by 8x and a much higher resolution than needed. This is fairly low risk as Vivado includes a wizard for this functionality and Mitchell's project has sample code for the XADC as well.

Decoding

There are a few transformations necessary to convert the raw RFID data to an MIT ID number. First, a PSK decoding module will be used, comparing successive bits and outputting one bit at a time, yielding a one on amplitude change and a zero otherwise. Next, a module will accumulate 224 bits into a buffer (the data width of the RFID data) and recognize the 172 fixed bits, extracting and outputting the 32 user-specific bits of each tap (along with a 1 wire enable line). Third, a descrambling module will be used to reverse the FlexSecure protection on the ID number, which is actually a fairly simple combinational circuit consisting of a few bit rearrangements and XORs with a hardcoded public key. Finally, a BCD module will be used to

convert the 32 bit binary number into the native 9 digit decimal format of MIT ID numbers, suitable for display via RS-232 or on XVGA.

The main component contributing risk is the analog circuit; previous projects in this space were analog-only, and our design will need to be able to reliable interface with an FPGA. The decoding is low-risk as the format is well-known and the FlexSecure protection was reverse-engineered in a student project over 10 years ago.

No BRAMs or external component are necessary for this section of the design.

RS-232 Serial Input/Output Pipelines

The RS-232 communication channels are full duplex, meaning input and output can be used simultaneously. This will be accomplished by independent dataflow pipelines for each I/O direction. The Nexys4 does not have a native RS-232 interface, but contains an FTDI chip to emulate RS-232 signals over the microUSB port, which will be mapped via the master XDC constraints file to pins on the Artix-7 FPGA, enabling serial communication with a laptop or desktop via USB. (On the computer side, a virtual serial port over USB will be used with freely available drivers from FTDI, along with a serial terminal such as Hyperterminal or picocom.)

RS-232 Input

RS-232 is a fairly simple serial protocol, and will be handled with a chain of input modules similar to those used for the Lab 5b infrared receiver. Using a divider module to generate the appropriate baud rate, oversampled by 8x, the synchronized RS-232 input (which is a one bit signal wire) will be sampled to watch for the start bit, then each data bit will be downsampled from the oversampled data, followed by parity and format checking to ensure packet validity (malformed or incomplete packets will be silently discarded.) The module will be parameterized for number of parity, stop and data bits. Each received character will be output on a wire bus (sized by the data bit parameter, which will be set to 8 for the actual implementation), along with an 1 bit enable line to signal new data. The enable line will be asserted for one cycle when new data is available, although the data byte will be buffered until the next valid data byte is read. This will be read directly by the main FSM.

RS-232 Output

A separate minor FSM will be used to enable sending characters over the RS-232 link. This module will use its own internal divider module to generate the baud rate (no need for a multiplier), and will accept 8 bits of data along with a 1 bit send signal as inputs. On each send request (each cycle where send is asserted), this FSM will start sending the data in RS-232 format at the baud rate, starting with stop bits, then the data bits in sequence, followed by parity bits and stop bits, where these values are also parameterized. When this FSM is finished sending, it will assert a 1 bit done signal for one cycle to signal to the main FSM that it is ready to accept another character.

No BRAMs or external memories are necessary for either sending or receiving data, and all of the required hardware is already built into the Nexys4 and lab stations (the FTDI chip on the board and the microUSB to USB cable to the computers.)

Main FSM

The main FSM will start by using the RS-232 output minor FSM to send the character prompt, sending one character at a time and waiting for each send to complete before moving to the next character. Next, it will read RS-232 input one character (8 bits) at a time, waiting for each input (each time enable is asserted), sending each character back for remote echo (via the same process), and saving each character into an internal buffer. This process will repeat for the password prompt. Then, the main FSM will prompt for the ID card tap, and wait for a valid ID number to be received by the ID card reader stack.
At this point, the main FSM will access a database to check validity of the credentials, and transition to either an authorized state, sending/display a random joke, or an unauthorized state.

The main FSM will continuously expose its current state and received data (username, password, ID number, joke) to the renderer so they can be display on the screen; for the RS-232 display, these will only be sent one time and in-order since the sent values will stay in the output of the RS-232 terminal.

Additional modules that the FSM must interact with are a timer module and audio. Once the user is asked for a password, and again for an ID tap, the timer module will be enabled and begin a countdown for input timeout, after which the FSM will reset in case the user walks away.

Renderer

For the display, we will be using XVGA which specifies a 1024x768 resolution, requiring a 65MHz clock at a screen refresh rate of 60Hz. The Nexys4 uses 4-bit color, so each pixel is a 12-bit RGB color value to display. Because most of our graphics will be letters and blocks, rather than images, we will initially use only 16 colors to save on memory. If there is memory to spare, or we need to upload images, then we can increase the colors available to 256 colors.

COE files will be created for our spritemap(s) and color maps. First, images will need to be converted to a .bmp format and then run through the provided MATLAB script to create the COE file. These COE files will then be saved to ROMs in Vivado.

The address calculator module will be a module that takes the information from the FSM and decides from what address in which ROM the current pixel data should by checking internal layout information. This will require numerous comparisons, additions/subtractions and multiplications depending on the complexity of our layouts and will need to pipelined to provide adequate time to fetch data from the ROM. This will take the 4-bit state indicator and additional data, and return locations of the pixel as well as whether we want a background pixel or a sprite pixel. The pixel chosen will then be sent through a color map to get its RGB values. The RGB values are then sent to the display.

Testing and Responsibilities

Most modules will have testbenches, some will be tested live only (aka not in simulation) The input and output chains can be given per-module testbenches and full live testing per chain, as well as the VGA rendering chain; the main FSM will be tested with the full system integration, using the hex display and the integrated logic analyzer for debugging.

Aneesh will be responsible for the ID card reading input chain, as well as the RS-232 serial input and output chains, due to previous experience with these technologies in previous classes and Lab 5b. Paige will be responsible for the graphics, rendering, and optional sound extension of the system with artistic control. The main FSM will be worked on jointly by both partners, although mainly worked on by Aneesh since it integrates directly with the inputs and outputs.

Week	Aneesh	Paige
Oct 30 - Nov 05	- Configure git repo - Generate clock IP module - Wire custom antenna - Build ID reader circuit	- Get basic display - Generate backgrounds - Begin working on sprites
Nov 06 - Nov 12	- Write ID card modules, test with reader circuit - Generate XADC IP modules	- Create all basic sprites - Write address calculator module
Nov 13 - Nov 19	- Write and test serial input and output modules	- Test display timing - Debug display
Nov 20 - Nov 26	- Write and smoke test main FSM module	-Help with main FSM module -Work on audio extension
Nov 27 - Dec 03	- Test interaction of ID card, serial and main FSM modules	- Work on picture extension
Dec 04 - Dec 11	- Trip to Hawaii	- Final testing and debugging
Dec 12 - Dec 14	Final Demo, Checkoff and Report	

Figure 3: Schedule

Resources

The card reader itself will be an analog circuit which requires certain special components beyond common passives (resistors, capacitors, diodes):

- LM18293 Push Pull Driver (used as an amp) - already in lab for 6.115 class
- Inductor loop for antenna - hand-made from enamel wire (have this in the lab as well)

Making the inductor loops will require a bit of skill, but no exotic components are required, making this low-risk in terms of supply chain requirements.

Total cost: $0

The other main constrained resource is on-board BRAM/BROM. The Nexys4 board has 4860Kbit available, of which we use ~3445Kbit as follows:

- 400Kbit for spritemap
- 2x 1500Kbit for 2 background maps (other background is pure black)
- 20Kbit for joke database
- 20Kbit for ID database
- 5Kbit for colormap

This fits comfortably within the available space with headroom and leaves enough space to allow for double buffering as well (with a few modifications). See individual design sections for details about these numbers (which are themselves overestimates).

Conclusion

This 2 factor authentication project will be an interesting and complex project. One of our biggest challenges will be to recognize whether or not a person is authorized, given that three different factors must be correct for this to happen, and all three of these components have not been previously taught. This project is applicable to MIT life as we already have DUO for our computers, we might as well have a similar process when we physically need to identify ourselves.