CS254 (Spring 2017): Lab Assignment 3

March 15-March 22, 2017

In this assignment, we will put together various parts that you designed in the last two assignments to implement a simple ATM controller.

The assignment has two parts. The first part relates to VHDL programming and mapping your design to the Digilent Atlys board. We will call this part the **frontend hardware**. This will be the physical device with which ATM users are going to interact. The second part relates to C programming, and interfacing to the controller on the Digilent Atlys board. This will serve as the back-end software that does checks of passwords, updates of account balances, issuing of instructions to the ATM controller to dispense money etc. We will call this part the **backend software**.

We first describe how an ATM user is supposed to interact with the hardware controller.

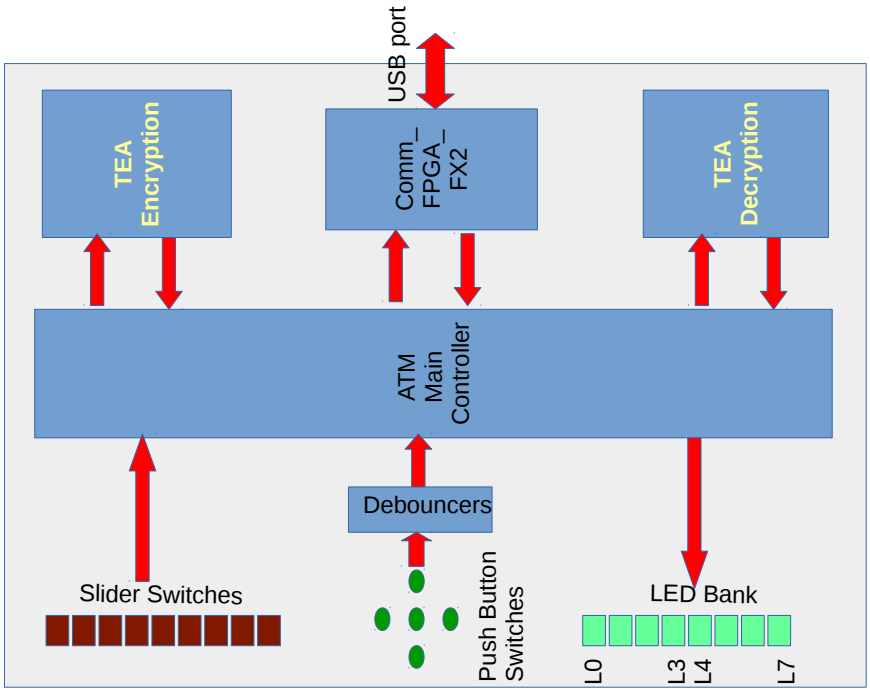

Figure 1: ATM controller

Interaction Specification: Please refer to Fig. 1 to understand the interaction specification clearly. For purposes of the following discussion, let T denote a suitably defined time period, say the time taken by

a counter to count from 0 to N in your VHDL design, where N should be a `generic` (recall `generic` in VHDL) parameter, than can be suitably set.

- The user presses a push-button `start` on the board to start using the ATM.
- The user uses the eight slider switches on the board to input data (ID, PIN, request) to the ATM in chunks of one byte. A push-button `next_data_in` is used to separate one input byte from the next, and to indicate to the ATM that an input byte is ready to be read in.
- The user presses a push-buttom `done` on the board to indicate that (s)he is done interacting with the ATM.
- LEDs $L0$ through $L7$ are used for displaying encoded messages to the user. The encoding of messages is explained below.
  - $L0$ through $L7$ turned off: Frontend controller in Ready state.
  - $L0$ blinking with inter-blink gap of T, $L4$ through $L7$ turned off: Frontend controller in Get_User_Input state. In this state, LEDs $L1$ through $L3$ are used display the number of input bytes (0 through 7) have read so far from the slider switches. A total of 8 bytes must be read in from the slider switches, with a push-button `next_data_in` being pressed to separate one input byte from the next (like we did in Assignment 1 involving encryption and decryption). After each press of `next_data_in`, LEDs $L1$ through $L3$ should indicate (in binary) how many bytes of the user input have been read in so far, so that the user doesn't have to remember this.

    In all cases, the first two bytes constitute the user ID, and the next two bytes constitute the PIN. Depending on the user type (normal or admin), the interpretation of the next four bytes are different.
    - If the user is a normal user (wants to withdraw cash from the ATM), the fifth byte indicates in binary the number of Rs. 2000 notes (s)he wants to withdraw. Similarly, the sixth, seventh and eighth bytes indicate the number of Rs. 1000, Rs. 500 and Rs. 100 notes, respectively, the user wants to withdraw.
    - If the user is an admin user (wants to load cash in the ATM), the fifth byte indicates in binary the number of Rs. 2000 notes (s)he wants to keep in the ATM. Similarly, the sixth, seventh and eighth bytes indicate the number of Rs. 1000, Rs. 500 and Rs. 100 notes, respectively, the user wants to keep in the ATM. An admin user can remove notes from an ATM or load notes in an ATM, regardless of the count of notes in the ATM at the point. Thus, the fifth to eighth bytes input by an admin user give the actual number of notes in the ATM after the operation.
  - $L0$ and $L1$ *blinking together* with inter-blink gap of T, all other LEDs turned off: Frontend controller in Communicating_With_Backend state.
  - $L0$, $L1$ and $L2$ *blinking together* for a total duration of at least $5T$ (longer, if needed by your design, is fine) with inter-blink gap of T, all other LEDs turned off: Frontend controller in Loading_Cash state.
  - $L0$, $L1$, $L2$ and $L3$ *blinking together* for a total duration of at least $5T$ (longer, if needed by your design, is fine) with inter-blink gap of T: Frontend controller in Dispensing_Cash state. In this state, LEDs $L4$ through $L7$ are used to visually render dispensing of cash and exception conditions. Specifically, we have the following three situations:

* *Sufficient balance in user's account, sufficient cash in ATM*: If k notes of Rs. 2000 are to be dispensed, LED $L4$ should blink k times with a gap of $2T$ between consecutive blinks. A similar visual rendering should happen for dispensation of notes of Rs. 1000 (LED $L5$ should blink), Rs. 500 (LED $L6$ should blink) and Rs. 100 (LED $L7$ should blink). Only one type of note should be dispensed at a time, and there should be a gap of $2T$ between dispensation of notes of different denominations. Furthermore, notes of higher denomination should be dispensed before those of lower denomination.
  As an example, if two notes of Rs. 2000, two notes of Rs. 500 and one note of Rs. 100 is to be dispensed, the sequence of blinking of LEDs $L4$ through $L7$ should be as follows: $L4$ blinks twice followed by $L6$ blinks twice followed by $L7$ blinks once, with a gap of $2T$ between consecutive blnks.
* *Insufficient balance in user's account:* LEDs $L4$ through $L7$ must *blink together* 3 times with an inter-blink gap of T.
* *Sufficient balance in user's account, insufficient cash in ATM:* LEDs $L4$ through $l7$ must *blink together* 6 times with an inter-blink gap of T.

Frontend Controller Specification: The controller is made of the following communicating components:

- Eight slider switches to input a byte of data at a time
- Four push-button switches: `reset`, `start`, `next_data_in` and `done`.

  The `reset` switch is pressed once to reset the front-end controller before it starts operating. The `start` switch is pressed once by a user when (s)he wants to start using the ATM. The `next_data_in` switch is pressed once between every consecutive input byte read from the slider switches, as described above. The `done` switch is pressed once after a user is done using the ATM.
- *Debouncers:* Each push-button switch needs a debouncer, like in Lab Assignment 1.
- *Encrypter module:* This is the same TEA encrypter that we designed in Assignment 1. It takes as input 64 bits of plaintext data and generates 64 bit of ciphertext data that are to be sent over the communication channels to the backend software (see below for how this communication must happen).
- *Decrypter module:* This is the same TEA decrypter that we designed in Assignment 1. It takes as input 64 bits of ciphertext data read from the backend software over the communication channels (see below for how this communication must happen) and generates 64 bit of plaintext data.
- *Communication module:* This is the `comm_fpga_fx2` VHDL module we referred to in Lab Assignment 2. You are not required to design this module but use it in your design to communicate with the USB controller on the FPGA board. Please see the problem statement handout for Lab Assignment 2 to find out where this VHDL module can be found.
- *ATM controller module:* This is the heart of the controller. It consists of four essential components, which must be present in your design; you may need to add additional components to suit your design objectives.

  The essential components are as follows:

  – Four registers named `n2000, n1000, n500, n100`, one each to store the number of Rs. 2000, Rs. 1000, Rs. 500 and Rs. 100 notes in the ATM at any point of time.

- *Timer:* This is a counter that increments from 0 to $N-1$ (the `generic` parameter referrred to earlier), and outputs a pulse of duration one clock period when the count reaches $N-1$. This is used to implement the blinks of LEDs with inter-blink gaps of T, etc.
- *Sequencer:* This keeps track of the state of the controller, and makes it transition from one state to another as required. The controller has five primary states: Ready, Get_User_Input, Communicating_With_Backend, Loading_Cash and Dispensing_Cash. Transitions between these primary states are described below. In each state, the sequencer must also make the LEDs turn on/off/blink as specified in the user interaction specification.
  * On pressing `reset` once, the controller goes from any state to the Ready state, and fills 0 in the register `n2000, n1000, n500, n100`.
  * In Ready state, if the `start` button is pressed once, the controller goes to the Get_User_Input state.
  * In Get_User_Input state, after all 8 bytes of input data have been read in from the slider switches, the controller goes to Communicating_With_Backend state.
  * In Communicating_With_Backend state, the following sequence of actions happen. The 8 bytes of data read in from the slider switches are first encrypted using the Encrypter module. The 64 bits of ciphertext thus generated are then sent to the backend software through the Communication module. The sequencer then waits to receive 64 bits of ciphertext response from the backend software through the Communication module. Details of the communication protocol between the frontend controller and the backend software are described below. Finally, the ciphertext response is decrypted using the Decoder module. Depending on the plaintext response thus obtained, the controller goes to one of Loading_Cash, Dispensing_Cash or Ready states.
  * If the response from the backend software indicates that the user ID and PIN could not be validated, the controller must go directly to the Ready state.
  * In Loading_Cash state, the decrypted response is used to update the registers `n2000, n1000, n500, n100`. After the updates of the registers are done, if `done` is pressed once, the controller returns to the Ready state.
  * In Dispensing_Cash state, the decrypted response and registers are checked to see which of the following three cases hold:
    - *Sufficient balance in user's account, sufficient cash in ATM*
    - *Insufficient balance in user's account*
    - *Sufficient balance in user's account, insufficient cash in ATM*

    Depending on which case holds, LEDs should be turned on/off/made to blink as described in the user interaction specification above.
    After the indications are done with the LEDs, if `done` is pressed once, the controller returns to the Ready state.

Backend Software Specification: On the host PC/laptop, you will be given a spreadsheet in .csv format that contains the following columns:
ID (values from 0-65535), a hash of the PIN (values from 0-65535), admin privilege (either 0 or 1), and Account Balance (values from 0 to $2^{32}-1$).

Your program must do the following in an infinite loop:

- Poll (i.e. read) channel 0 every 1 second to see if $0x01$ or $0x02$ is available on this channel. Once $0x01$ or $0x02$ is read on channel 0, the program must check whether the same value is read for three

consecutive reads on channel 0, each separated by 1 second. If not, the program must go back to polling channel 0 afresh.

- If $0x01$ or $0x02$ is read for three consecutive times on channel 0, the program must read channels 1 through 8 to read in the encrypted input from the frontend controller.
- The encrypted input must then be decrypted by a software implementation of the TEA decryption algorithm to figure out the 8 plaintext bytes. The user ID (read from channels 1-2), PIN (read from channels 3-4) and the four bytes of cash details (read from channels 5-8, with channel 5 giving the least significant byte and channel 8 giving the most significant byte) must then be identified.
- The PIN must be hashed by computing a cyclic left shift of the two PIN bytes by 11 bit positions.
- The user ID is then checked in the .csv file to identify the user's account. The hash of the PIN is checked, and if both match, your program must print `Valid user found`. If the user is also an admin, the program must print `User has admin privileges`
- If the user is not an admin user, the cash request details read from channels 5 thorugh 8 are then checked to see if the user has sufficient balance in his/her account. If so, the program must send the cash request details in encrypted form back to the frontend controller over channels 10 through 17 (pad with most signficant 0s to make it 64 bits, and use the TEA encryption algorithm in your program). In this case, the program must also send $0x01$ on channel 9 to the frontend controller to indicate that there is sufficient balance in the user's account, and update the user's account balance accordingly (only if code received on channel 0 was $0x01$). If the user doesn't have sufficient balance in his/her account, the program must send $0x02$ on channel 9 to the frontend controller. In this case, channels 10 through 17 must have values 0 on each of them.
- If the user is an admin user, the program must send $0x03$ on channel 9 and the cash upload request in encrypted form is sent to the front-end controller over channels 10 through 17 (pad with most significant 0s to make it 64 bits and use TEA encryption algorithm).
- If the user's ID and PIN cannot be validated from the .csv file, the program must send $0x4$ on channel 9.

Communication protocol:

- Channel 0 is used to send a code from frontend controller to the backend software.

  Codes are: $0x00$: no communication, $0x01$: user inputs ready on channels 1 through 8 and adequate cash available in ATM, $0x02$: user inputs ready on channels 1 through 8 but adequate cash not available in ATM, $0x03$: received response from backend.

- Channel 9 is used to send a code from backend software to frontend controller.

  Codes are: $0x00$: no communication, $0x01$: normal user validated and there are sufficient funds in balance, $0x02$: normal user validated but there are insufficient funds in balance, $0x03$: admin user validated, $0x04$: user not validated (ID/PIN didn't match)