CS254 (Spring 2017) Project
Network of ATM controllers

Apr 5-Apr 28, 2017

In your final CS254 project, which accounts for 70% marks, you will be building on the simple ATM controller that you designed as part of Lab Assignment 3. In a sense, if you have done your previous lab assignments diligently and got things right, you already have a head-start to claim your 70% marks. If you haven't been able to make things work exactly the way you wanted in the earlier assignments, this is your chance to make them work, build on top of it, and claim your 70% marks. Hopefully, all those last-minute changes to code and design that we inevitably receive for every assignment will be put to good use in this project.

To avoid confusion, we will call ATM controllers designed for CS254 projects as *Advanced ATM Controllers* or AdvATMC. The ATM controllers designed as part of Lab Assignment 3 will be called *Simple ATM Controolers* or SimpATMC.

This document describes the overall functionality of AdvATMC. Unlike in the design for SimpATMC, where everything, including the communication protocol, was specified down to a very low level of detail, the design specification for AdvATMC will be given at a higher level. This is a design project, and it is up to you to design the details (e.g. of communication protocols) to make the overall system work. Clearly, we do not expect much similarities between the way in which the lower level details are worked out for different groups. Similarities, beyond the reasonable, will be ground for suspicion of plagiarism.

Network of AdvATMCs belonging to different banks: In Lab Assignment 3, we had a frontend controller communicating over a USB port with a backend software. There was a single spreadsheet accessed by the backend software, and this spreadsheet contained information about all (including admin) account holders. Effectively, we had only one bank, and information about all account holders was available in one spreadsheet accessible to the backend software.

In your project, different AdvATMCs must be considered to belong to different banks. Each frontend controller is associated with a bank, called its *home bank*, and it communicates over a USB port with a backend software that runs on a laptop/PC of the home bank. The backend software therefore only has access to a spreadsheet containing information about accounts in the home bank of the AdvATMC.

To keep things simple, we will assume that there is *only one AdvATMC per bank*. Each such AdvATMC is assumed to be connected via a dedicated USB link to the backend laptop/host PC of its home bank. The laptop/host PC is assumed to run the backend software, and is also assumed to have a spreadsheet (.csv file) containing information of all account holders in the home bank. AdvATMCs of different banks could, of course, be running concurrently. These AdvATMCs are assumed to be connected through ethernet links to an ethernet switch, as shown in Fig. 1. This gives rise to a network of AdvATMCs, each connected via a dedicated USB link to its home-bank backend (see Fig. 1). For purposes of simplicity of the backend

software design, we will assume that every pair of distinct AdvATMCs connected on the network belong to different home banks. This allows us to circumvent concurrency related issues that can arise when two different AdvATMCs belonging to the same home bank try to access/update account holder information in the same spreadsheet/database.

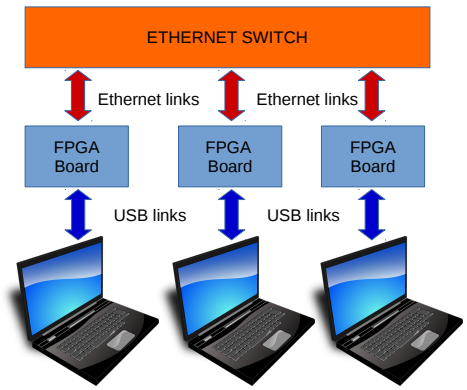

Figure 1: Network of ATM controllers

High-level user interaction specification of* AdvATMC*: This is largely the same as that of SimpATMC. The only exception is that a non-admin user directly specifies how much money (s)he wishes to withdraw (like in the normal ATMs we use), instead of specifying how many notes of what denomination (s)he wishes to withdraw (as was the case in SimpATMC). Your AdvATMC must dispense notes, if possible, that adds up to the amount requested, and this calculation must be done within the ATM controller Admin users however still continue to specify how many notes of what denomination are being loaded in the ATM.

The user IDs and PINs are assumed to be 16-bits wide, as in the case of Lab Assignment 3, and the total withdrawal amount requested by a non-admin user is assumed to be representable in 32 bits ($2^{32} - 1$ is quite a lot of cash!) The feedback to the user and mimicing of note dispensation using blink codes of LEDs are assumed to be exactly identical to what we had for SimpATMC in Lab Assignment 3.

A user with an account in a specific bank is allowed to withdraw cash from any AdvATMC on the network.

Bank identities and user IDs: Each bank is assumed to have a unique 5-bit bank id. You must reserve the least significant 5 bits of the (16-bit) user ID for the bank id. Thus, while each user ID continues to be

16-bits (as in Lab Assignment 3), user IDs that match in their least significant 5 bits correspond to users with home accounts in the same bank.

The 5-bit bank ID must be an input that can be fed to the back-end software when it is run. The same bank ID must also be stored in a special register of the front-end controller. Your VHDL code for the front-end controller must allow this register to be loaded with a 5-bit bank ID, read from the five least significant slider input switches, on pressing a special push button `load_bank_id` (not supposed to be accessible to users) when the sequencer of the front-end controller is in the Ready state.

User PINs are assumed to be hashed in the backend software by circularly left-shifting the 32-bit plaintext PIN by the decimal equivalent of the unique bank ID. Thus, the spreadsheets of different banks store PINs using different hash functions.

Communication related details: All communication over USB links and over ethernet must be encrypted using TEA with a pre-specified key that all banks agree upon. In Lab Assignment 3, we had used encrypted communication over the USB link only. In your project, you must use encrypted communication over the ethernet links as well.

The protocol for communication between an AdvATMC front-end controller, and the corresponding back-end of the same bank can be identical to what we had in Lab Assignment 3 (this will allow you to re-use code from your Lab Assignment 3).

However, the protocol for communication between the individual AdvATMCs over the ethernet is for you to design and implement. The only requirement we have is that information shuttling over the network must be encrypted. Needless to say, you must describe in detail (with justificaiton) your communication protocol for communication over the ethernet in your final project report.

When a user comes and enters her user ID, PIN and cash details at an AdvATMC, the front-end controller must first check whether the user is an account holder in its home bank. If so, the front-end controller must directly communicate with the backend software via the dedicated USB link, and serve the user accordingly. However, if the front-end controller detects that the user ID doesn't belong to its home bank, it must then forward a suitable request (which you have to design) to other AdvATMCs on the ethernet network. If any AdvATMC connected to the ethernet switch finds that the user ID in the forwarded request belongs to its home bank, it must process the forwarded request (containing user ID, PIN and cash details, all encrypted, of course) using the dedicated USB link to its home bank backend, and forward the response back to the AdvATMC where the user presented her request to begin with. On receiving this response, the AdvATMC serving the user must either present cash to the user or display an appropriate message, as appropriate.

Note that while the communication protocol over the USB link was specified to the last detail in Lab Assignment 3, and is going to be used unchanged in your project, the details of the communication protocol over the ethernet links is not completely specified in this problem statement. It is part of your responsibility to design and implement a suitable protocol, document it, and demonstrate its use in the overall system.

A few additional points: A few points might be worth noting when designing your advanced ATM controller:

- Unlike in real life, we will assume that withdrawing money from an account in bank 1, using an ATM of bank 2, does not result in an additional charge being levied to the user. So no extra complications or heartburns here!

- There can be changes in central bank policies that affect ATMs of all banks uniformly. These changes

can be of two types:

- Restrictions related to disbural of specific types of notes: This restricts the maximum count of notes of a specific denomination that can be disbursed to a user during each withdrawal. For example, the restriction could be that no Rs. 1000 note can be disbursed, and not more than five Rs. 500 notes can be disbursed in one withdrawal.
- Restrictions related to withdrawal limits: This restricts the maximum amount of money that a user can withdraw from her/his account in a single withdrawal. Usually such restrictions apply to daily withdrawals, but to keep things simple, we will assume that such restrictions apply per withdrawal. As an example, the restriction could be that no user can withdraw more than 90% of the total balance in her/his account or Rs. 20000 (whichever is lower) in one withdrawal.

Restrictions of the second type above can be implemented by changes to the backend software (we did something similar in Lab Assignment 3). Of course, the backend software of all banks must incorporate the required changes.

Restrictions on disbursal of notes of specific denominations must be communicated to each front-end controller from the backend software of the home bank when the system starts. Thus, all restrictions are effected by modifying, re-compiling and re-executing the backend software. The restrictions on disbursal of notes of specific denominations are conveyed to the front-end as the system starts. The other restrictions are enforced on a per transaction basis.

- An admin user of a particular bank can load cash using only a front-end controller of the same bank.

Bonus points for caching account information in* AdvATMC*: The front-end controller of an AdvATMC can cache the account information of a fixed number (at least 4) of home bank account holders, when a withdrawal is made by such an account holder from the AdvATMC. Groups implementing this correctly will get an additional 10 (out of 70) marks as bonus

To illustrate caching better, suppose an account holder of bank 1 withdraws some money from an AdvATMC of bank 1. The front-end controller of the AdvATMC must then store information about this account holder in its cache (replacing information about account holders who withdrew cash earlier, if needed). Subsequently, even if the (USB) link to the backend controller of the home bank is broken (detected by waiting for response from the backend for a certain timeout period, and obtaining no response), and if the same account holder comes to withdraw additional cash from the same AdvATMC, the front-end controller can dispense cash to the user using the cached information. Thus, the front-end controller must store all information necessary to authenticate a user and determine the amount of cash that must be disbursed to the user, even if the (USB) link to the home bank backend is broken, if the information about this account holder is present in its cache. Once the (USB) link is restored, the front-end controller must sync up with the backend, and ensure that all updates to the cached information are communicated to the home bank's backend, before any further transactions are allowed.

Note that this was not implemented as part of Lab Assignment 3, and you will need to design special protocols to detect that the USB link is broken, and that it has been restored.

Please use the "Least Recently Used" (LRU) policy to replace account holder information from the front-end controller's cache, when a fresh account holder's information is to be stored, and the cache is already used. You may read about LRU at
`https://en.wikipedia.org/wiki/Cache_replacement_policies`.

Evaluation: For evaluating your projects, we will burn $n(\geq 2)$ FPGA boards with your VHDL code, and connect them to an ethernet switch via ethernet links. We will also use n laptops, each running a copy of your backend software, and each connected to one of the FPGA boards.

We will then store a unique 5-bit bank ID to each in the front-end controllers (using the `load_bank_id` button, as described earlier), and use the same bank IDs as input parameters of the corresponding home bank backend software. We will also have n separate spreadsheets (.csv files) containing information about account holders in the n banks in the corresponding laptops.

Cash will then be loaded in each of the front-end controllers using the same protocol we used in Lab Assignment 3. Note, however, the credentials (user ID and PIN) of an admin user in one bank may be different from that in another bank.

Finally, we will try to withdraw cash from both the front-end controllers. For this, we will try to withdraw cash from a home bank account, as well as from a remote bank account.

To test caching (if you have implemented it), we will withdraw cash from one front-end controller, then physically disconnect the USB link to its corresponding home bank laptop, and try to withdraw cash again from the same account. After a few transactions, we will re-connect the USB link, and examine if the backend software correctly updates the different account holders' information.