CS254 (Spring 2018): Lab Assignment 6 Instructions

March 13, 2018

In this lab, we will build on the previous lab assignment and add some extra features to what you have already implemented in the previous lab.

Specifically, the C program is required to do the following:

[H1] It reads a table in the format already given from a file named “network.txt”.

[H2] Starting from $i = 0$ to $i = 63$,

[H2.1] It reads from channel i the encrypted co-ordinates of a railway signaling controller and decrypts it.

[H2.2] It re-encrypts the co-ordinates received on channel $2i$ and sends it back on channel $2i + 1$ to figure out if this really came from a signaling controller (or if it was some junk values read on the channel).

[H2.3] It then waits to hear a special 32-bit encrypted acknowledgement, say `Ack1`, from the signaling controller (FPGA) on channel $2i$.

[H2.4a] If what it obtains from channel $2i$ decrypts to `Ack1`, it knows that the coordinates that came on channel $2i$ indeed came from a genuine controller.

[H2.4b] Otherwise, it reads channel $2i$ again after 5 seconds and check if the decryption of what it read on channel $2i$ is indeed `Ack1`

[H2.5] If it fails to receive the encrypted `Ack1` on channel $2i$, it repeats the polling with the next value of i or with $i = 0$ if i was already 63.

[H2.6] Otherwise, i.e. if it successfully received the encrypted `Ack1` on channel $2i$, it remembers in its local memory the association between the co-ordinates and the channel number. Subsequently, it always communicates with the controller at the saved co-ordinates using channels $2i$ (for listening) and $2i + 1$ (for writing).

[H3 It then sends its own special 32-bit encrypted reverse acknowledgement, say `Ack2`, to the signaling controller on channel $2i$. You are free to design what this reverse acknowledgement should be (another of your favourite sequence of 32 bits).

[H4] The host computer now consults the table it read to figure out the 8 bits per direction it needs to send to the railway signaling controller. This requires searching the table for the coordinates of the controller and putting together all of the required information as 8 bytes.

[H5] The host computer then encrypts the first 4 bytes and sends them to the railway signaling controller on channel $2i + 1$.

- It waits for the encrypted acknowledgment `Ack1` from the controller on channel $2i$. If it doesn't receive this acknowledgment within 256 seconds, it goes back to step H2.

[H8] On receiving the `Ack1`, it encrypts the last 4 bytes and sends these to the railway signaling controller over channel $2i+1$.

[H9] It again waits for the encrypted acknowledgment `Ack1` from the controller on channel $2i$. If it doesn't receive this acknowledgment within 256 seconds, it goes back to step H2.

[H10] Once the above cycle is over, it sends the encrypted acknowledgment `Ack2` on channel $2i$.

[H11] Then the host computer waits for 32 seconds and re-starts the entire process.

You are also required to modify `cksum_rtl.vhdl` so that it does the following:

[C0] It uses a specific pair of channels $2i$ and $2i+1$ for communicating with the host computer. You can choose any i from 0 to 63. This should be hard-coded as constants in your VHDL file.

[C1] It constructs 32 bits out of the 8 bits of its coordinates, encrypts them and sends them on channel $2i$.

[C2] It listens on channel $2i+1$ to receive 32 encrypted bits, decrypts them and checks whether its coordinates are embedded in the derypted message. The embedding of the 8-bit co-ordinates in the 32-bit message should be exactly the same as was used to construct the 32-bit message from the 8-bit coordinates. If it doesn't receive its co-ordinates back within 256 seconds, it goes back to step C1.

[C3] If it receives its co-ordinates back on channel $2i+1$, it then sends the special encrypted acknowledgment `Ack1` on channel $2i$ and listens on channel 1 until it gets the special encrypted acknowledgment `Ack2` from the host computer. If it doesn't receive this acknowledgment within 256 seconds, it goes back to step C1.

[C4] After receiving the encrypted `Ack2` from the host computer, it receives the next 32 bits on channel 1, decrypts it and re-constructs the information about tracks in 4 directions passing through this junction.

[C5] It then sends the encrypted acknowledgment `Ack1` back to the host computer on channel 0.

[C6] Next, it receives the remaining 32 bits on channel 1, decrypts it and re-constructs the information about tracks in the remaining 4 directions passing through this junction.

[C7] It again sends the encrypted acknowledgment `Ack1` back to the host computer on channel 0.

[C8] Next, it waits to receive the encrypted acknowledgment `Ack2` from the host computer on channel 1. If it doesn't receive this within 256 seconds, it goes back to step C1.

[C9] It next reads the eight slider switches and interprets them as follows.

  - If switch j (in 0 through 7) is set to 1, it means there is a train waiting to come to the junction where the controller is location from direction i. Otherwise, no train is waiting to come to the junction from direction i.

[C10] It then displays the signal in all eight directions using the LEDs as follows:

- If there is a train waiting to come from a direction, and no train waiting to come from the other direction, it sets the signal for that direction to green for 3 seconds and the signal in the opposite direction to red for 3 seconds.
- If there are trains waiting to come from opposite directions, it sets the signal to green for the direction encoded using a higher binary encoding (use the same encoding as was used earlier for directions) to green for 1 second, followed by amber for 1 second, followed by red for 1 second. The signal in the opposite direction should similarly be set to red for the entire duration.
- If there are no trains waiting to come from a direction, it should always set the signal to red for that direction.

[C10] After displaying signals for all the 8 directions, it waits for 8s and repeats the entire cycle.