Ense 352 – Fall 2018 – Term Project

Revision 4: 2018-12-04 17:06:58 -0600 (9b4fd67)

Handed Out: 2018-11-11

Due: ~~Mon 2018-12-03 by 16:00~~ ~~Wed 2018-12-05 by 23:55~~ Thu 2018-12-06 by 16:00

The project for this year is to write, *in assembly language*, a "Whack-a-mole" game. You will target your code to run on the enel384 board, which carries the VLdiscovery cortex-m3 board used in our class.

This youtube video gives an introduction to the basics of the game, which challenges you attain your best possible speed by increasing your reaction time. There are a few differences:

- In our case the human interface will consist of 4 pushbuttons and 4 corresponding LEDs, instead of five "moles" emerging from holes.
- You will press the button corresponding to the lit LED, as fast as possible. Using a big mallet, *however soft*, will destroy your board and will require you to build/obtain a new one.
- You'll notice in the video, several moles can emerge at once. For our project, *only one* LED will turn on at any given time. This simplification was suggested by your lab instructor, Trevor Douglas, so you can thank him for making your life easier.

You will be writing code in cortex-m3 assembly language. For best marks follow my assembly code examples given during the lectures, particularly the assembly language questions. In addition, I've posted a file `how-to-comment-assembly.pdf` which shows how to document an assembly language function.

The Use Cases

These use cases attempt to capture the system requirements from a user's perspective. As the course proceeds we will be discussing these in class, and it's possible some may be changed as a result of those discussions.

UC1 Turning on the system.

1. The user performs a system boot by pressing the reset button
2. The system enters Use Case 2 (UC2): Waiting for Player

UC2 Waiting for Player

1. The system goes into the startup routine. This is an LED pattern indicating that no game is in progress and the system is waiting for a player to start. For example you could have the four LEDs cycling back and forth at 1Hz. This continues without stopping until:
2. The user presses any of the four buttons. The system enters Normal Game Play (UC3).

UC3 Normal Game Play.

1. A fixed wait time elapses: `PrelimWait`. (Notice on the video it appears to be a few tenths of a second.)
2. The game turns on a randomly selected LED. The game starts the `ReactTime` timer. (The player has to respond before this expires.) The shorter this time, the more challenging the game.
3. The user presses the corresponding button before `ReactTime` expires.
4. The `ReactTime` value is reduced by a certain amount to prepare for the next cycle. (So each cycle gets a bit harder.)
5. The system goes back to step 1. After `NumCycles` of these successful loops complete, the game enters End Success (UC4). `NumCycles` may be, for instance, 16. (On the video it's 44, which might be too high)

UC3 Alternate Path: `ReactTime` expires.

1. During UC3 step 2 the user fails to press the correct button before `ReactTime` expires. Or the user presses an incorrect button
2. The game enters End Failure (UC5).

UC4 End Success. The user has won the game.

1. The game displays the "winning" signal, which is some sort of LED pattern indicating the person won. This signal is displayed for time `WinningSignalTime`.
2. After displaying this signal, the game displays the user's proficiency level. This display remains visible for 1 minute, after which the game returns to UC2.

UC5 End Failure. The user has lost the game.

1. The system displays the "losing" signal, which is a flashing display, **in binary**, of the number of successful cycles completed. Since this is a 4-LED display you can display any number from **one** to **fifteen**. This flashing signal is displayed for time `LosingSignalTime`. If the cycles completed are less than 1 or greater than 15, then you will have to decide how you'll deal with that, and document it in your `readme.txt` file.
2. After displaying this signal, the game returns to UC2, Waiting for Player.

Deliverables

Submit your solution via urcourses in a single zipfile. Place the solution to the questions in a subfolder named `proj`. Include all assembly files (**.s**), as well as the project files (**.uvprojx**) and (**.uvoptx**). Also include an ASCII text file `readme.txt` which describes

1. What the game is
2. How to play
3. Any information about problems encountered, features you failed to implement, extra features you implemented beyond the basic requirements, possible future expansion, etc.
4. How the user can adjust the game parameters, including:
   (a) `PrelimWait`: at the beginning of a cycle, the time to wait before lighting a LED
   (b) `ReactTime`, the time allowed for the user to press the correct button to avoid terminating the game.
   (c) The number of cycles in a game: `NumCycles`
   (d) values of `WinningSignalTime` and `LosingSignalTime`.

Do not include any generated files.