Software Formalization

Year: 2019 Semester: Fall Team: 19 Project: Digitopoly

Creation Date: September 29th 2019 Last Modified: October 4th, 2019

Author: Niraj Menon Email: menon18@purdue.edu

Assignment Evaluation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| **Assignment-Specific Items** | | | | |
| **Third Party Software** |  | x2 |  |  |
| **Description of Components** |  | X3 |  |  |
| **Testing Plan** |  | x3 |  |  |
| **Software Component Diagram** |  | x4 |  |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** |  | x2 |  |  |
| **Formatting and Citations** |  | x1 |  |  |
| **Figures and Graphs** |  | x2 |  |  |
| **Technical Writing Style** |  | x3 |  |  |
| **Total Score** |  | | |  |

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

*Relevant overall comments about the paper will be included here*

1.0 Utilization of Third Party Software

For using the embedded microcontrollers in our project, we are not utilizing any software other than the regular firmware provided to us by STM32 and using their SystemWorkbench software in order to code, run, test and flash the firmware onto the microcontroller.

Our project utilizes the following commonly available software:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | License | Description | Use |
| Raspbian | GPL v2 | A free operating system based on Debian optimized for the Raspberry Pi hardware [1] | Allows us to program the Raspberry Pi as a functional computer and install necessary libraries |
| node.js | MIT | Web server software | Used to serve the web app that displays when the RPi starts up. |
| Electron Framework | MIT | node.js framework for developing cross-platform apps using pure HTML/CSS/JS [2] | Renders the graphics needed to display the game on the monitor |
| raspi-serial | MIT | node.js library for utilizing the COM port on the Raspberry Pi | Allows us direct access to the serial port from within node.js in order to communicate with the STM using less code. |
| jQuery | MIT | JavaScript library used to provide more powerful animation control | Animates the different cards on the screen with fewer lines of code |

2.0 Description of Software Components

A comprehensive list of our high-level functions includes, but is not limited to:

* **On the STM32F407VG-DISC1 (Main board micro)**: The firmware installed on the microcontroller will have the following:
  + **STM-to-Pi communication functions**: In order to manipulate objects on the screen, the Pi needs to be told what to do by the STM32. On the STM side of things, we have functions that send information to the Pi using a UART transmission function. Some of these functions include:
    - **pi\_click():** simulates a mouse click on the Pi
    - **pi\_scroll(direction):** simulates a scroll up/down on the Pi
    - **pi\_buy(propertyName):** display a “buy” on the screen
    - **pi\_sell(propertyName):** display a “sell” on the screen
    - **pi\_trade(propertyName, player1, player2):** display a “trade” on the screen
    - **pi\_poweroff()** - STM received a power button click, shutdown the Pi
    - **pi\_move(spaces)** - the mechanical movement of pieces have started, update display with new position determined by spaces.
    - **pi\_showChance(id)** - Draw a Chance card on screen
    - **pi\_showCC(id)** - Draw a Community Chest card on screen
  + **Monopoly game functions**:
    - **buy(propertyObject):** add a property object to the current player's inventory and deduct funds
    - **sell(propertyObject):** remove a property object from the current player's inventory and add funds
    - **trade(propertyObject, player1, player2):** transfer property object from player1 to player2, and deduct/add funds where required.
    - **banker(action, funds, player):** Performs all money related operations - adding or deducting from player.
    - **move():** Upon receipt of the dice roll, will determine the set of moves required to move from one place to another, and sets player properties based on the final destination (eg. if they pass Go, the banker is called to add $200 to the current player’s account, or if they end up in jail, the player’s personal properties are edited to indicate that they cannot move anywhere unless a double roll is detected.)
    - **toJail():** Adds the ‘in jail’ property to the current player. Only executed if the move function determines that the player must go to jail.
    - **drawChance():** Using a random number generator function (math.h/rand() in C) a random number fed into a formula will determine which Chance card to draw and display to the user. The ID of the card is sent to the Pi using pi\_showChance (ID).
    - **drawCC():** Using a random number generator function (math.h/rand() in C) a random number fed into a formula will determine which Community Chest card to draw and display to the user. The ID of the card is sent to the Pi using pi\_showCC (ID).
    - **awaitRoll():** sends a ‘heartbeat’ to the dice micros, which ‘wakes’ them up and has them record and analyze orientation and acceleration data, figure out their final orientations and transmit it (still need to figure out if we can do this simultaneously or we have to do this consecutively).
  + **Mechanical operation functions:**
    - **moveXY (x, y):** Top level function for controlling motors in order to move the electromagnet to another position by using a stepping algorithm.
    - **grabPiece()**: Turns the electromagnet on
    - **dropPiece()**: Turns the electromagnet off
  + **User input handling functions:**
    - **powerOff():** Interrupt triggered function occurring when the user presses the power button. Calls the Pi handler which causes the Pi to shut down, then by itself will save necessary game data and enter an infinite for loop.
    - **click()/scroll():** Interrupt triggered function occurring when the user scrolls using the rotary encoder or presses buttons to click things on the screen. The action performed will call the necessary associated Pi function.
  + **Peripheral perspective (not functions, but is a list of hardware modules on the STM32 we would utilize):**
    - USART1 - Talks to the Pi
    - USART2 - Talks to the dice (1 master, 2 slaves configuration)
    - GPIO OUT/PWM - Motor controller square wave
    - GPIO OUT - Electromagnet control line to transistor switch
    - GPIO IN - Rotary encoders from the motors and user input
    - GPIO IN - Power button to turn off
* **On the BlueNRG 2 die micro:** 
  + **BlueNRG-to-STM communication functions:** These functions will only run as part of an interrupt triggered by a UART transmission from the STM in order to save power on the dice. Since the purpose of the dice is to simply transmit orientation, these will be the few functions coded into the micro:
    - **sendOneOrientation():** Record and analyze orientation and acceleration data, then figure out the final orientation and transmit it.
* **heartBeat():** Primarily, this is a debugging function, but we thought it was important to retain this function in the end to provide a way for the STM to be able to check that the dice is still available for contact. At some point we plan to use this to transmit battery level information to STM and show it on the Pi display.

3.0 Testing Plan

Each of the hardware components in our project is hooked up to either the main STM32F4 micro, or the BlueNRG-2 dice micro. As a result, most of the testing for these components simply relied on writing code more than actual wiring of circuits beyond the necessary connections between the components and the micro, as well as power and ground.

**Components tested:**

* **Raspberry Pi programming:**
  + The very first thing that the Raspberry Pi should do is to be able to load the node.js app we developed after the white background desktop loads. We configured this by turning the Pi into a [4] kiosk-mode one. At the OS level, this is all the installed Raspbian has to do, apart from providing the necessary setup for the node.js app to run.
  + We will test the time it takes to go from cold boot all the way to the starting app screen. to make sure it doesn’t take too long. If there are issues, that may indicate issues with the chip or a corrupted SD card.
  + The app itself should be able to navigate between menus when provided instructions in JSON by the STM over UART. While debugging, we will use the mouse and keyboard to manually navigate the app and debug code onboard when the Pi has already been placed onto the board.
* **Display:** 
  + Very simple test of plugging in power and display data cables. If the Raspberry Pi screen loads clearly with no flickering, the test is successful.
* **STM32F4 UART:**
  + The UART was configured to transmit at 115200 Baud, 8 data bits, 1 stop bit and no parity or hardware flow control.
  + We had the following tests for our UART, namely:
    - Transmission to:
      * PuTTY on our lab computer via the RS-232 FTDI USB-to-Serial converter
      * UART on the Pi (via serial console on /dev/ttyAMA0)
      * UART on the Pi (via JavaScript serial port wrapper on node.js)
    - Reception from:
      * PuTTY on our lab computer via the RS-232 FTDI USB-to-Serial converter
      * UART on the Pi (via JavaScript serial port wrapper on node.js)
  + We would debug the transmissions using the Mixed Signal Oscilloscope on our lab bench to be sure that the signal coming out of the UART TX/RX lines weren’t wrong, and that would tell us if the UART was working properly or not.
* **Raspberry Pi UART:**
  + The UART was also configured to transmit at 115200 Baud, 8 data bits, 1 stop bit and no parity or hardware flow control.
  + We had the following tests for our UART, namely:
    - Transmission to:
      * PuTTY on our lab computer via the RS-232 FTDI USB-to-Serial converter
      * UART on the STM, which would echo back whatever we typed on the Pi. We thought this was a better test because we were reassured that the data was transmitting just fine with no noise issues.
    - Reception from:
      * PuTTY on our lab computer via the RS-232 FTDI USB-to-Serial converter
      * UART on the STM (repeated messages from the STM via serial console on /dev/ttyAMA0)
      * UART on the Pi (repeated messages from the STM via JavaScript serial port wrapper on node.js)
  + We would debug the transmissions using the Mixed Signal Oscilloscope on our lab bench to be sure that the signal coming out of the UART TX/RX lines weren’t wrong, and that would tell us if the UART was working properly or not.
* **BlueNRG UART:**
  + We found that transmission via the Bluetooth module was getting garbled at high baud rates, for which we still have not identified the issue.
  + For now, the UART is configured to transmit at 9600 Baud, 8 data bits, 1 stop bit and no parity or hardware flow control.
  + We had the following tests for our UART, namely:
    - Transmission via Bluetooth to:
      * PuTTY on our lab computer via the RS-232 FTDI USB-to-Serial converter
      * UART on the STM, which would echo back whatever we typed on the Pi. We thought this was a better test because we were reassured that the data was transmitting just fine with no noise issues.
    - Reception from:
      * PuTTY on our lab computer
      * UART on the STM (repeated messages from the STM via serial console on /dev/ttyAMA0)
      * UART on the Pi (repeated messages from the STM via JavaScript serial port wrapper on node.js)
* **IMU Unit:**
  + We wrote tests for checking:
    - The initial state of the IMU and whether it was giving out a consistent value for orientation and a zero value for acceleration while not being moved around.
    - We wrote tests to simulate slow and fast movements of the IMU in all three directions (X-axis, Y-axis, Z-axis) in order to watch the angular rate and acceleration values change. [5] Slow movements in a specific direction will result in a small positive value for acceleration and vice versa. [3] The angular velocities read at a simulated crash of the IMU falling to the floor (like throwing dice) should reflect the expected directions of the X, Y and Z axes in different orientations.

We will also be consolidating these functions into a full-code-coverage verification suite for the firmware and software we develop. The suite will include functions from verification of pin outputs after basic pin toggling to verification of complex results from high-level functions.

4.0 Sources Cited:

[1] “Welcome to Raspbian,” *Raspbian*. [Online]. Available: https://www.raspbian.org/. [Accessed: 29-Sep-2019].

[2] Electron, “electron/electron,” *GitHub*, 29-Sep-2019. [Online]. Available: https://github.com/electron/electron. [Accessed: 29-Sep-2019].

[3] A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications, *Starlino Electronics,* 29-Dec-2019. [Online]. Available: <http://www.starlino.com/imu_guide.html>. [Accessed: 29-Sep-2019].

[4] Setup a Raspberry Pi to run a Web Browser in Kiosk Mode, *Die Antwort,* Oct-2017. [Online]. Available: [https://die-antwort.eu/techblog/2017-12-setup-raspberry-pi-for-kiosk-mode/.](https://die-antwort.eu/techblog/2017-12-setup-raspberry-pi-for-kiosk-mode/) [Accessed: 13-Sep-2019].

[5] Manon Kok, Jeroen D. Hol and Thomas B. Sch¨on (2017), ”Using Inertial Sensors for Position and Orientation Estimation”, *Foundations and Trends in Signal Processing: Vol. 11: No. 1-2, pp 1-153*. Available: <http://dx.doi.org/10.1561/2000000094>. [Accessed: 13-Sep-2019].

Appendix 1: Software Component Diagram ([uncropped diagram here if not clear](https://drive.google.com/open?id=1XxtEC2q5Oadla4zulYXKXdgdIDR62IyI))





