ECSE 421

System Design Document

Group #5

Max Chau
Ching-Wai Chee
Simon Foucher
Jean-Mikael Lassonde
Winston Lin
Mathieu Perreault
Logan Smyth
Philip Tang
Danny Wu

Table of Contents

1. Introduction

This document provides a high level architectural overview of the elevator system project, which controls three elevators servicing a twenty floor building. This document outlines both the hardware and software components which will be integrated in the final system, how they function individually and also how they will be integrated in the final embedded system.

The system is divided into two major components communicating via an SPI bus software interface channeled over an RS-232 serial data cable. The first physical component is mostly responsible for main control and decision making and will be implemented in hardware by a standard PC in a Linux environment. This component will handle data processing, decision making, will control the position of the virtual elevators, will display graphics and finally will handle serial communication with the second major component. This second major component will be implemented in a McGumps Microprocessor board. Its responsibility will be to capture all the user input via a PS/2 keyboard and transmit them to the central processing unit. Since the MSP will be simulating user input for the internal buttons of three elevators and the floor direction buttons on 20 floors, the system will also contain an LCD screen to tell the user which system is currently being emulated.

2. Design Overview

2.1. System Architecture

Figure 1 below depicts the overall system in the form of a deployment diagram. The system will be constructed from the following components:

- **Central Processing Station**
  This component monitors the overall elevator system and consists of an ordinary computer. It is the primary graphical and audio interface for the overall system and handles the majority of the elevator logic. Graphics will be shown on the attached ***Monitor***. Inputs specific for the monitoring system will be provided by the attached standard ***Keyboard*** and ***Mouse***.

    - **Application Logic**
      This custom application will be where the bulk of the software work is performed. The logic will handle incoming packets from the serial I/O drivers as well as requests from the user interface, and manage the virtual elevator locations. More information concerning this software component can be found in section 2.2.1.

- **Visual Interface**
  The user interface will display each elevator along with the status of the various panels and doors. The graphics will be displayed using standard OpenGL along with an interface to give floor calls and requests. More details are provided in section 2.2.3.

- **Serial I/O Drivers**
  This custom driver will interact with the ***Application Logic*** and is required to communicate over the serial line with the ***McGumps Microprocessor Board***. More information concerning the Central Processing Station serial I/O software component can be found in section 2.2.4.

- **McGumps Microprocessor Board**
  This component functions as the controls for the three elevators. Consisting of a MSP430F149 MCU chip and a MAX7128AE PLD, this control system will communicate with the ***Central Processing Station*** via a ***Serial Line***. It will accept the elevator inputs from an attached ***PS/2 Keyboard*** and provide simple graphical output from a directly attached ***Character LCD Display***.

  - **Serial I/O Drivers**
    These drivers will take care of the communication between the microprocessor board and the Central Processing Station via a serial line. More details about the implementation can be found in section 2.2.5.

  - **Keyboard I/O Drivers**
    The MSP will have basic driver set that will interrupt the processor whenever a key is pressed on the attached PS/2 keyboard, allowing the microprocessor software to interpret the request and transmit a packet to the Central system when needed. More information can be found in section 2.2.5.

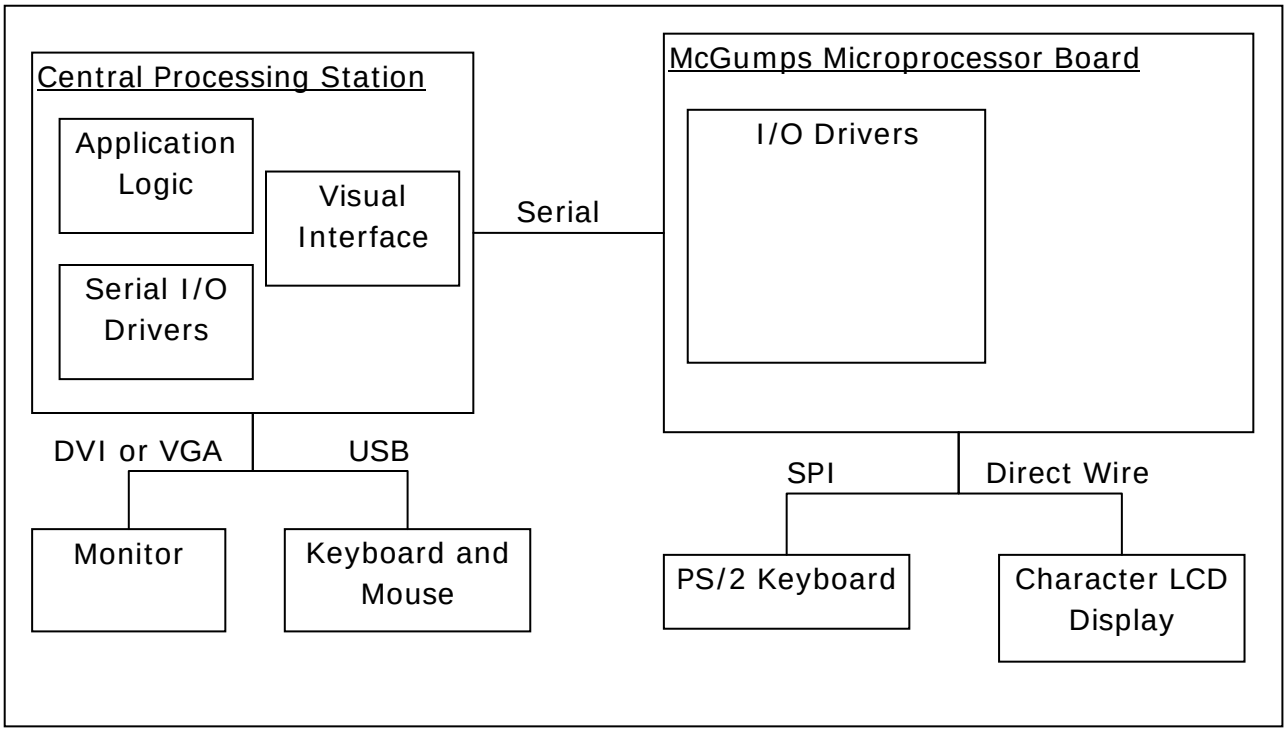

Figure 1: Deployment Diagram for the Elevator System

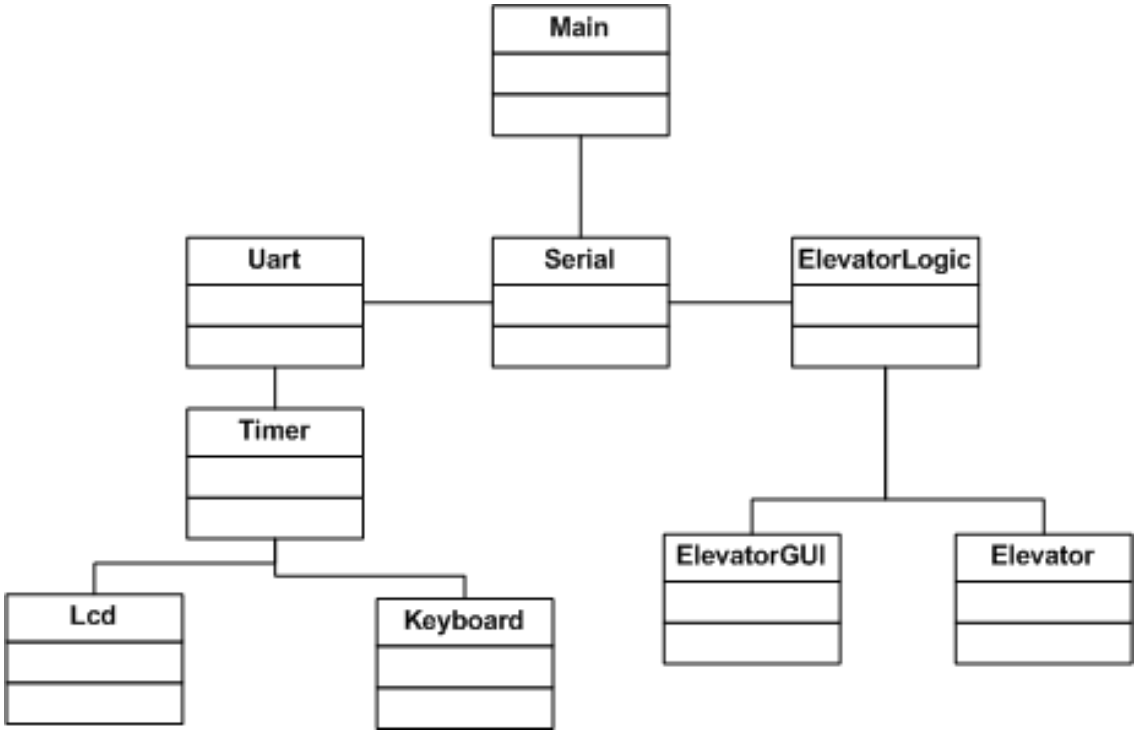

Figure 2: Implementation Diagram of the Elevator System

2.2. System Operation

2.2.1. Central Processing Station: Application Logic

The application logic on the Central Processing Station will behave according to Figure 3 (below). Upon starting the application on the Central Processing Station, the program logic will enter a stand-by state in which it will wait for input from the Microprocessor board. Once it receives data from the Microprocessor board (see section 2.2.4-5), the logic will processes the data and takes the proper decisions, according to the developed algorithm to manage the elevator system. For example, the application might receive a signal from the Microprocessor board emulating a user requesting an elevator for a specific floor. Once the logic receives this message, it will interpret it and dispatch an existing elevator in the system to the requested floor according to the elevator dispatching algorithm. It will then send the processed decision to the Microprocessor board, therefore committing the operation.

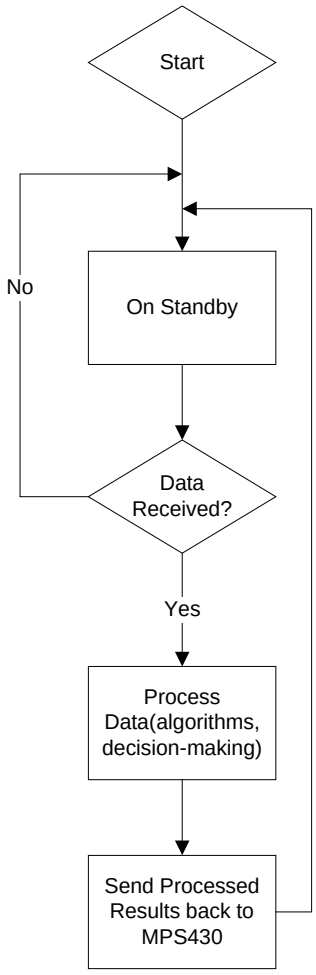

Figure 3: Central Processing Station Dataflow Diagram

In terms of programming paradigms, the application logic will function in its own thread, separate from the Visual Interface thread (see section 2.2.3). In order to ensure communication between the threads, the application will use shared memory and Elevator objects, which are described in section 2.2.3. Using the Boost library for C++ applications, a new thread will be spawned as soon as the application starts, and the application will then enter the stand-by mode, waiting for serial I/O, which is described in section 2.2.4.

2.2.2. Central Processing Station: Data processing, Decision Making and operations of virtual elevators

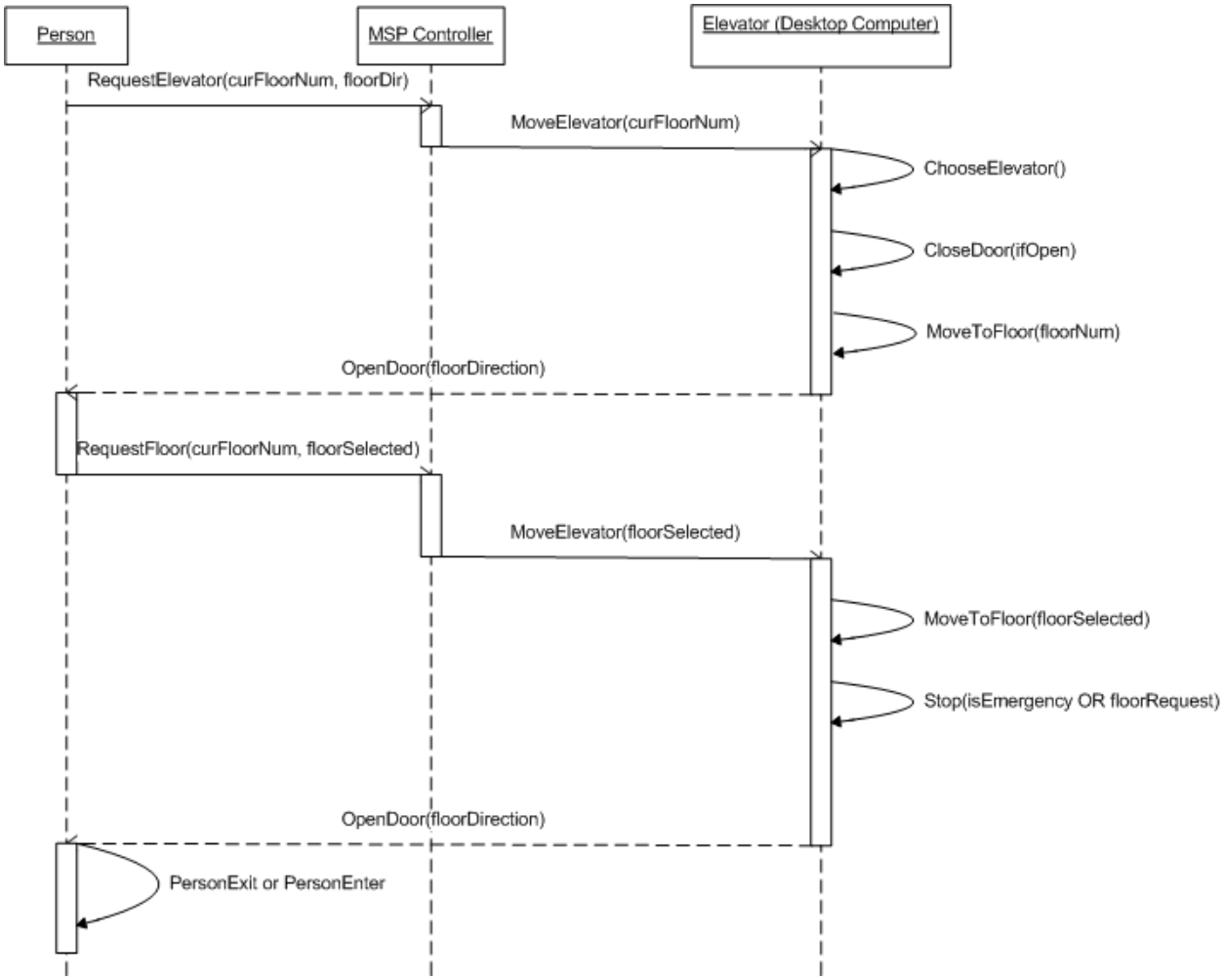

Figure 4: UML Communication Sequence Diagram

2.2.3. Central Processing Station: OpenGL Graphical User Interface (GUI)

The Graphical User Interface (GUI) will visually represent what is happening in the elevator simulation system. It will be the focus of an observer wanting to know the current state of the system. The application itself will consist of a 1024x768 pixel window with OpenGL graphics inside. Every time an event occurs in the program logic (see below), the OpenGL GUI will display the appropriate graphics. The end goal is a fully intuitive simulation environment.

Programming Implementation

In terms of programming paradigms, every elevator in the system is represented by a C++ Elevator object. Many functions can be called on each Elevator object

present in the system (a default of 3 elevators are present). For example, every frame (1/60th of a second), the method draw() is called on each Elevator object present in the system.

Software Library Used

The *freeglut* library is used throughout the visual interface to display OpenGL graphics. This library supports user input (mouse + key input) as well as superimposed pop-up menus. This library is supported on the Windows, Linux and Mac platforms and can be found on the web.[1]

Elevator Class Reference

Instance Variables

int floor : variable used to keep track of which floor the Elevator object is standing at.

int numfloors : variable indicating how many floors are travelled in this specific Elevator (default value in this implementation is 20).

bool *buttons : array of Boolean variables keeping track of the state of buttons on the Elevator control panel.

float position: Floating point variable keeping track of the precise position of the Elevator object.

bool jammed: Boolean variable keeping track of the Jammed/Not Jammed state of this Elevator objects.

int direction: a variable indicating the current movement direction of the elevator: up, down or idle

Class Functions

Program Logic ➔ Visual Interface (OpenGL)

moveToFloor(int): Function which will move the targeted Elevator object to the floor specified by the integer parameter.

openDoor(), closeDoor(): Functions which will open or close the doors of the targeted Elevator object.

[1] http://freeglut.sourceforge.net/

colorButton(int, bool): Function which will color the specified button on the targeted Elevator's control panel depending on the Boolean parameter passed as second argument.

stopAndJam(): Function which will stop the targeted Elevator and assert the variable used to indicate the jammed state of this Elevator.

Visual Interface (OpenGL) ➔ Program Logic

currentPosition(): Function which will return the floating-point current position of the Elevator object for the program logic to use.

areDoorsClosed(): Function which will return a Boolean indicating whether the doors are opened (true) or closed (false).

currentDirection(): Function which will return an integer indicating the current direction of travel of the elevator

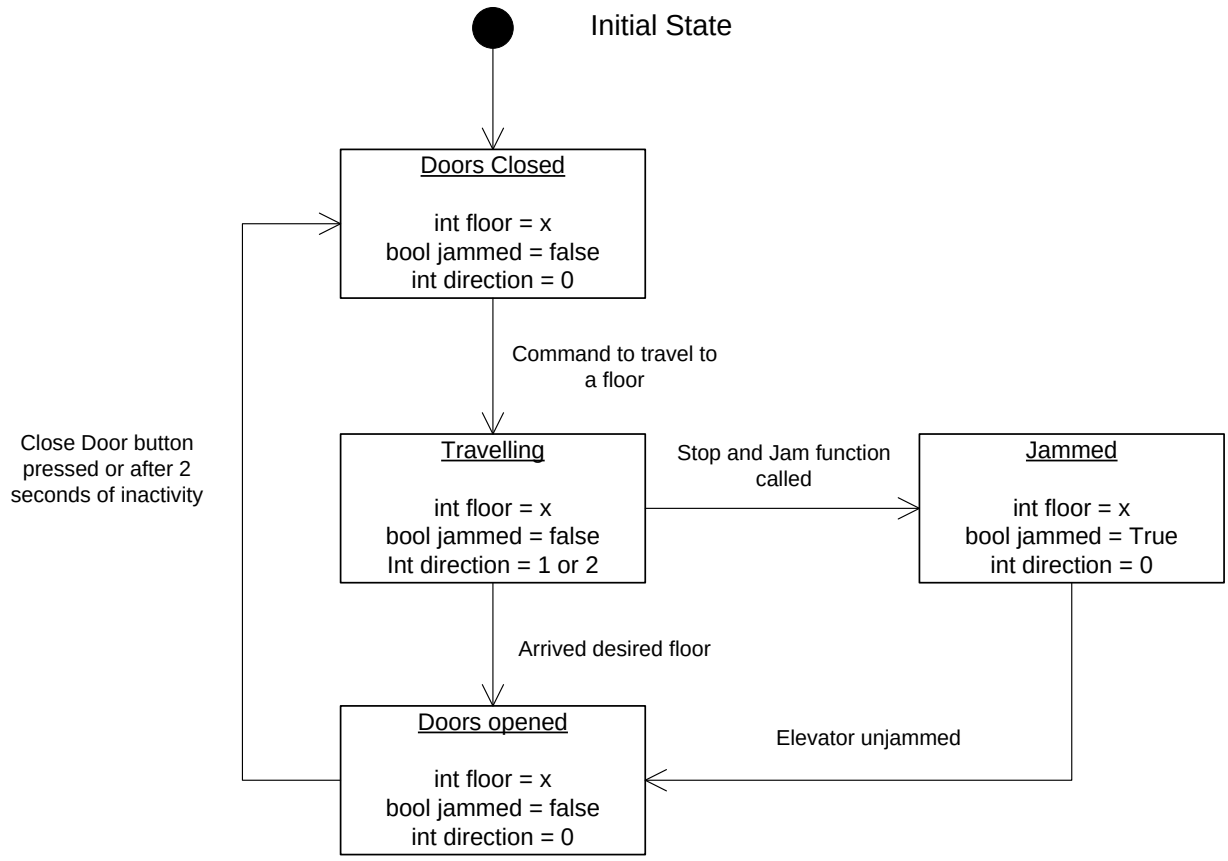

Figure 5: Elevator Class State Diagram

2.2.4. Central Processing Station: Serial I/O Drivers

The serial interface on the central processing station will be continuously monitored to receive any status updates provided by the McGumps Microprocessor board. Any button pressed from within an elevator will update the buttons boolean array of that specific Elevator object in order to reflect a button pressed. Once a status update is received, the elevator operations algorithm is run to re-compute the job of each elevator. In order to update the tasks of the elevators, instructions are sent back to the McGumps board over the serial line to refresh the jobs of each elevator.

In terms of programming paradigms, the application logic is going to monitor the Linux /dev/ttyS0 device handle for possible data transmissions coming from the serial line connected to the McGumps. Using the Boost serial I/O library for C++, this process will run in a separate thread and handle asynchronous events without disrupting the operations of the central processing block. Once such a change is detected (handled by internal OS Interrupt), the serial driver is going to put the received data in a special queue, which the main program logic is going to poll continuously.

2.2.5. McGumps Microprocessor Board: I/O Drivers

Programming Implementation

The main loop of the system puts the processor to sleep until an input is sensed via an interrupt on either the system's serial bus, or via a keyboard input. When input is sense, the system will process the request and forward it to the central system over serial.

The system will also keep an FSM to keep track of which elevator and which floor are currently being emulated, so the source address of the packets can be set properly for the central system to process.

IO Reference

Functions

Visual Interface (OpenGL) ➔ Program Logic

keyboard_get_char(): Function which will return the a character read in from the PS/2 keyboard.

serial_tx(packet*): Function which will return transmit the packet struct given as an argument over the serial interface.

serial_rx(packet*): Function which will copy a packet from the packet queue into the packet structure passed as an argument, or return false if no such packet exists.

lcd_printf(char*, …): Function which will output characters onto the display.

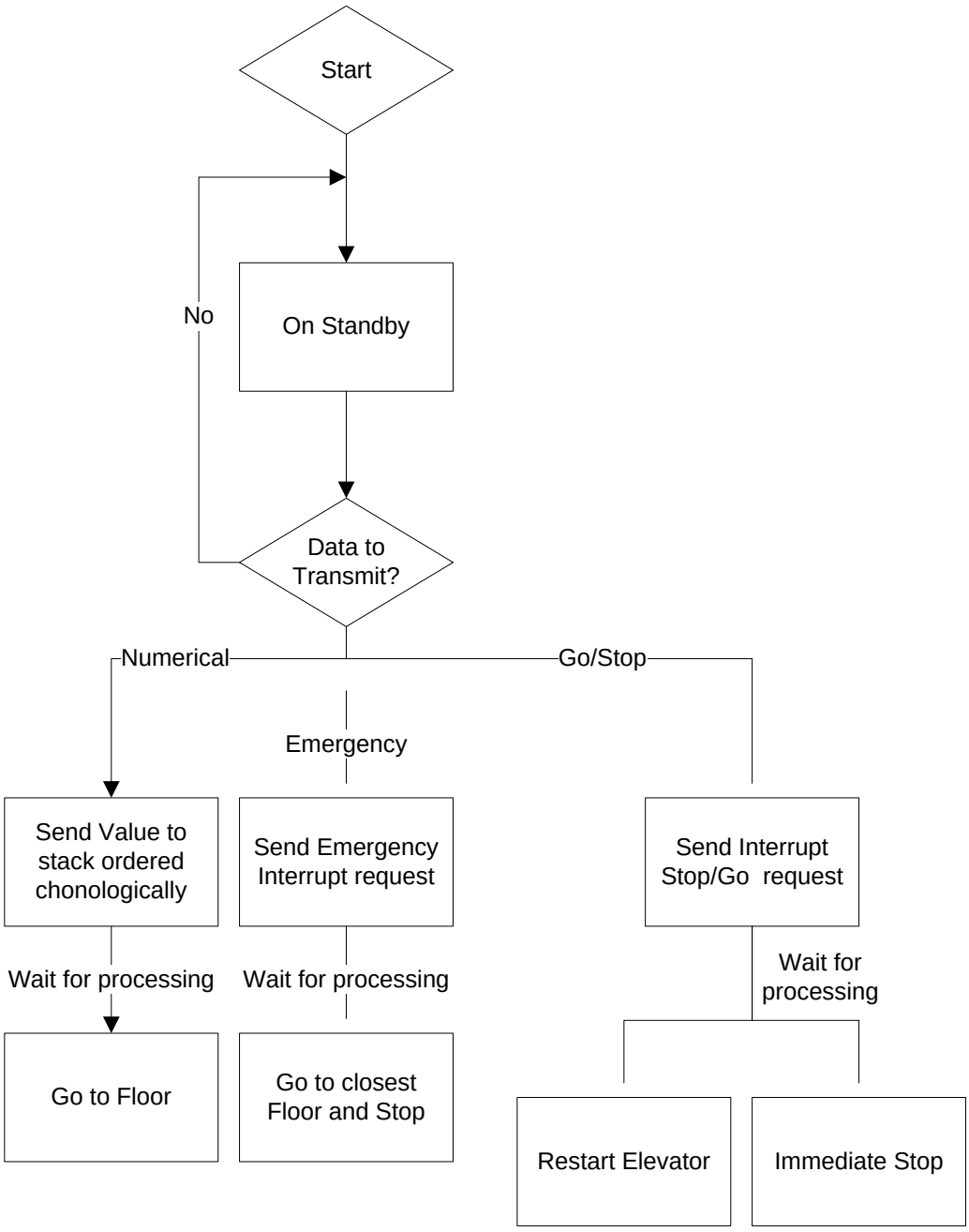

Figure 6: Microprocessor Board dataflow

3. Requirements Traceability Matrix

ID	Functional Requirement	Implementation
T1	*There is one up/down signal per floor. Whenever pressed, the microcontroller responds by sending an elevator to that location with the intention of going in the direction signaled.*	Button low level drivers implemented in the McGumps Microprocessor board (See Section 2.2.5). The event is handled in the central processing station via a serial I/O driver (See Section 2.2.4)
T2	*Position feedback is sent by every elevator such that the control system is always aware of all the elevator positions.*	Since the elevator is actually virtual, its position will be generated by a sub system of the central processing station, responsible to create, operate and maintain the virtual elevators (See Section 2.2.2). The elevator position feedback is also sent to the GUI which displays it on the monitor (See section 2.2.3). The communication is handled implicitly via a global variable elevator Object accessible by all systems.
T3	*Any given elevator spans all the floors of the building, such that any floor is accessible from all the others.*	This is implemented at a low level in the Data Processing unit (see Section 2.2.1) by means of variable boundaries. It is also reflected in the I/O drivers (See Section 2.2.5): there are 20 valid floor buttons which can be pressed by the user), as well as in the GUI (See Section 2.2.3 the virtual building displayed is 20 floors high)
T4	*Each elevator is equipped with a number button board; one button representing one floor.*	Implemented by the PS/2 keyboard connected to the McGumps Microprocessor board (See Section 2.2.5)
T5	*Each elevator is equipped with an EMERGENCY button.*	Implemented by the PS/2 keyboard connected to the McGumps Microprocessor board (See Section 2.1: System Architecture for Hardware and Section 2.2.5 for software drivers)
T6	*Each elevator is equipped with an OPEN and CLOSE door buttons.*	Implemented by the PS/2 keyboard connected to the McGumps Microprocessor board (See Section 2.1: System Architecture for Hardware and Section 2.2.5 for software drivers)
T7	*A certain priority of commands will be maintained in the system.*	Implemented in the Data Processing and decision making software component (See Section 2.2.2)
T8	*A 3D visual interface will serve as visual support for the Elevator System.*	Implemented in the Graphical User Interface component (see Section 2.2.3)

ID	Error Detection Functional Requirement	Implementation
T1	*Very high speed recovery.*	Implemented in the Data Processing and decision making software component (See Section 2.2.2)
T2	*Emergency Timeout*	Implemented in the Data Processing and decision making software component (See Section 2.2.2)