17ELC018 Coursework – Task 3

Thomas Finch (B417345)

Themba Kaonga (B422757)

Contributions

Thomas Finch B417345

Task 1

- Designed and completed the class and sequence diagrams
- Wrote the appropriate documentation accompanying the diagrams

Task 2

- Developed the device driver and hardware abstraction classes
- Developed the Timer class implementation

Task 3

- Redesigned the Timer class and developed the new DoorButton class
- Developed the motors speed control through the use of pulse width modulation and a mark space ratio
- Developed the application layer classes
- Developed the program list solution enabling the user to add new stages and programs through the modification of a char array

Themba Kaonga B422757

Task 1

- Designed and completed the activity and use case diagrams
- Wrote and formatted the Task 1 report

Task 2

- Co-developed the code demonstration class
- Wrote the report description for the source files
- Wrote demonstration instructions
- Commented the code for each class

Task 3

- Re-designed the Class diagram to match the added/redesigned classes
- Re-designed the Sequence diagrams to show all possible scenarios from specification
- Wrote the design documentation comments and class descriptions for Task 3 report

System Requirements

- The system design should be generic such that different wash programs can be added.
- System should take user input and set user-selected program.
- System should take user input on “Accept button” to start the selected program.
- The system should handle extra “Accept” presses as advance inputs.
- System should not start if door is not closed.
- System should suspend program and alert user if door input occurs during program execution.
- If the door is then reclosed during suspension, system will resume.
- System should take user input on “Cancel button” to interrupt and pause the selected program.
- While paused, the system should resume program on “Accept” input and reset the machine on a “Cancel” input.
- System should poll all inputs at an interval faster than a possible user input can occur.

Class Descriptions

There are 14 classes which integrate together to form the software program for the Bytronix washing machine hardware. These classes encompass 3 separate layers of development, the highest layer representing the software application components, which the user will interface with, the middle.

A brief description of each classes’ operation is below, categorised under their respective levels abstraction:

Layer 1: Application (AL) Classes

- WashingMachine: Factory class that instantiates the inputs, outputs, program management, control, low level hardware interface objects to run the application.
- SystemOperator: Main class to facilitate programs based on the truth tables drawn from different combinations of user input to the washing machine hardware.
- Program: Configures the array of stages for each wash program and provides program status information to classes with access
- ProgramManager : When a program is called, the ProgramManager fills the program array with its stages
- Stage: This class contains the characteristics of each program stage and calls output objects functions when the stage runs.

Layer 2: Hardware Abstraction (HAL) Classes

- Button : Gets Boolean status of buttons
- ButtonLatch : Sets Boolean status of button latch circuit
- DoorButton : Gets Boolean status of door buttons
- Timer : Halts the application for a specified amount of time
- SevenSegmentDisplay : Sets display output based on user input value
- Motor : Toggles the operation & direction of the motor
- Buzzer : Toggles the operation of the buzzer

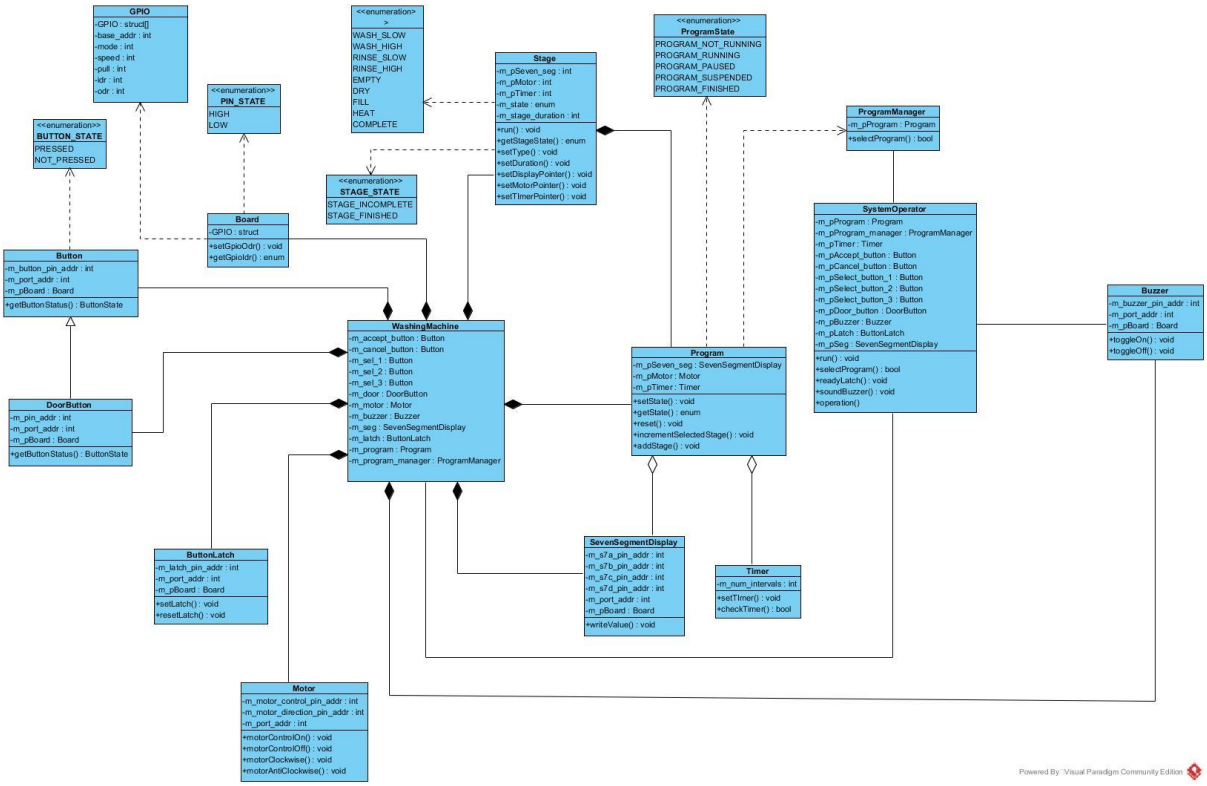
Layer 3: Device Driver (DDL) Classes

- Board : Configures the direct GPIO access to the target hardware
- GPIO: Database that holds pointers to GPIO port registers

Design Documentation

Class Diagram

The application design was changed to more accurately follow the principles of Object Oriented Programming (OOP) development. The authors found that the changes explained below reduced the development time of the application code layer.



The WashingMachine class is represented in Fig 1 as having composite of all other classes (except Stage). This was designed so only a single object would be created for the inputs and outputs, and those objects are passed to the SystemOperator for operation. The advantages of this approach are encapsulation of object data and hardware resource efficiency (as multiple objects are not created for a single input/output).

The entire function and operation of the Timer class has been revised from a hardware clock timer, to currently use HAL delay in the STMicroelectronics firmware. This has led to simplification of the of the timer class implementation throughout the code.

The initial Program class has split into Program & ProgramManager classes.

The user input buttons have been more accurately modelled in an object-oriented fashion as opposed to the poorly cohesive “UserInput” class of the initial design.

Button status is now queried by polling the Button objects contrary to the interrupt handling design formed in Task 1. This simplified the application layer, as user input is modelled as a simple truth table based on which different object methods are called.

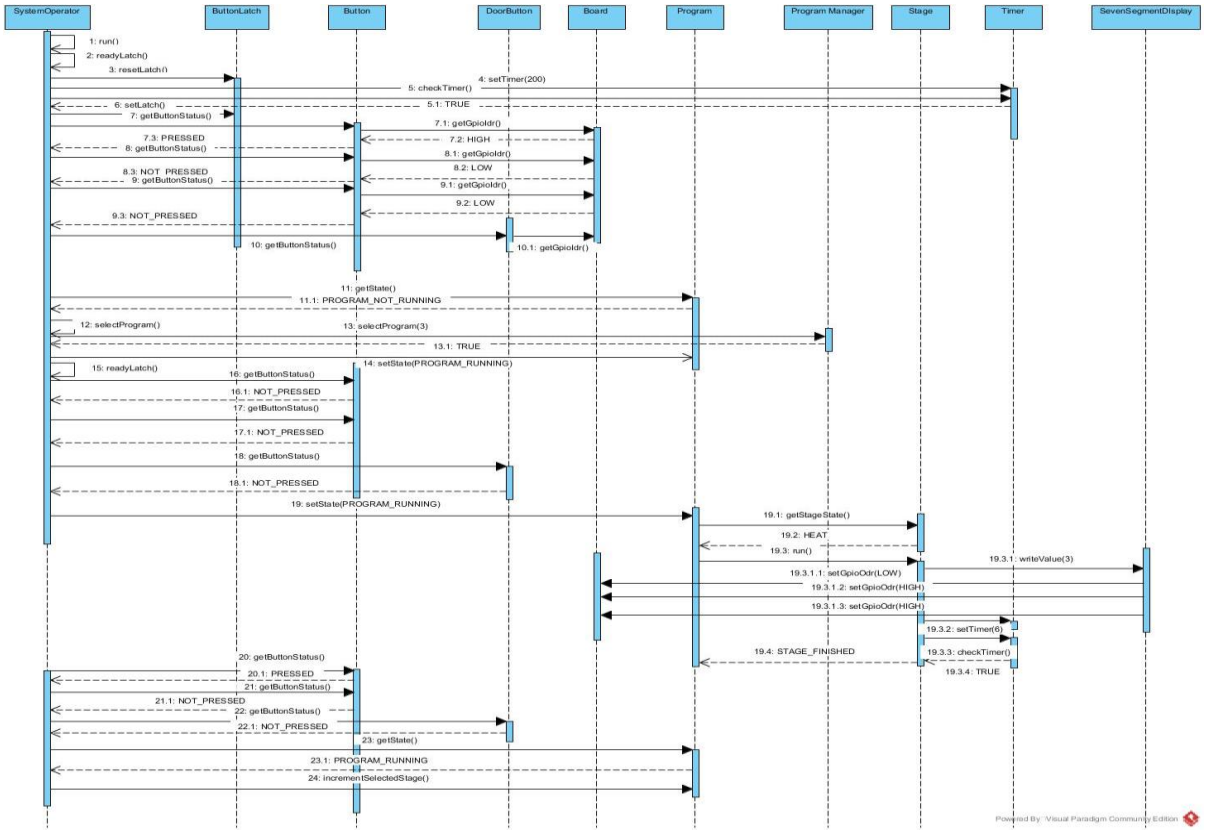
The output class has been removed as it the hardware outputs had significantly different operations breaking the OO guidelines for cohesion, and are now modelled separately via the Buzzer, Motor and SevenSegmentDisplay & classes.

Sequence 1: Run and Advance

The SystemOperator function runs the main application in step 1. Function calls (FC) 2-6 show the button latch preparation process. Steps 3-6 are called inside the readyLatch() function; this process is not shown again in these diagrams for brevity's sake.

Steps 7-9.3 are the sequence to check the Boolean values of the three program select buttons. The three Button objects call the Board object's function to get the value and then reply to the SystemOperator using the enumerated BUTTON_STATE value. This process is shortened for future Button user input in this sequence diagram. Steps 10 & 11 are to check that the door has not been opened and that a program is not running respectively.

Based on successful returns, the select program runs and sets the program to the run sequence, which begins at Step 16. Steps 16-18.1 check that the user hasn't interrupted this sequence by pressing accept, cancel or door button and that the program has been set to the running state.



Program state is set again to running and the current stage runs in Step 19, once the timer has elapsed, the stage state is set to FINISHED.

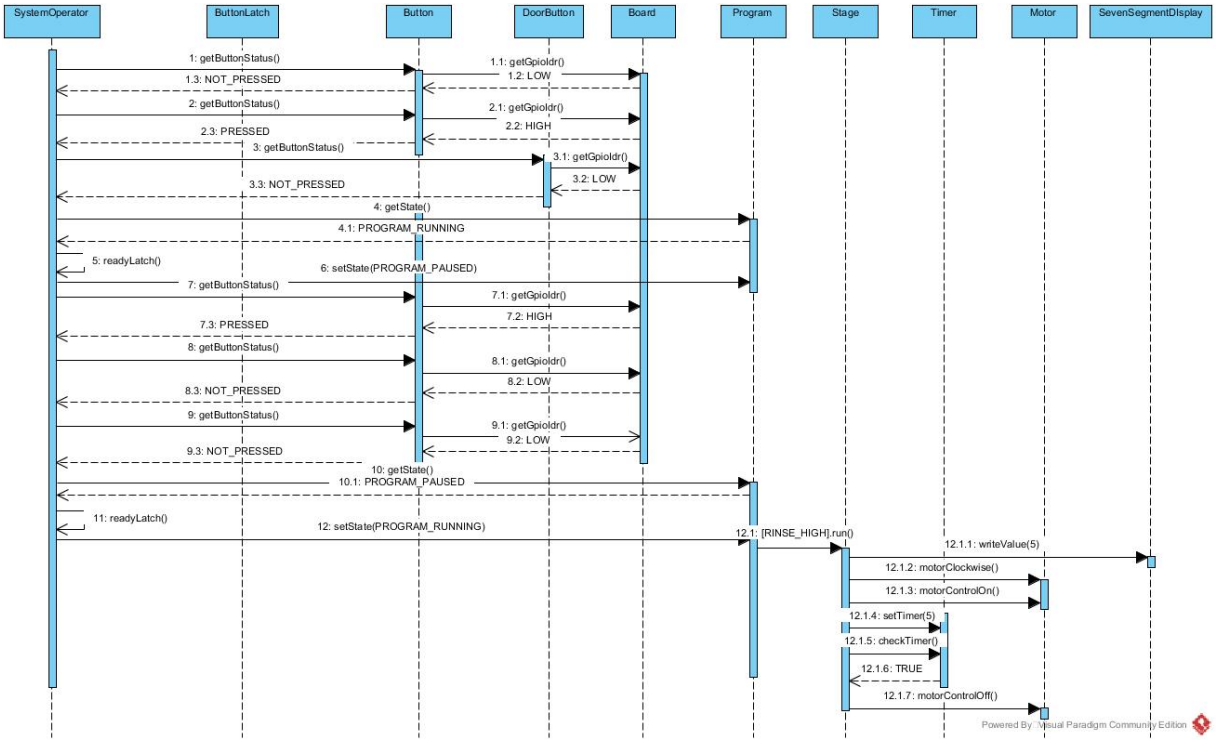
Finally, the advance sequence runs in Steps 20-24.

Sequence 2: Pause and Resume

Steps 1-18 from Sequence 1 which describe the run event sequence have not included in the following sequence diagrams for brevity.

In this sequence, steps 1-6 show that the Cancel button has been pressed and the program is running, therefore the program is set to the PROGRAM_PAUSED state.

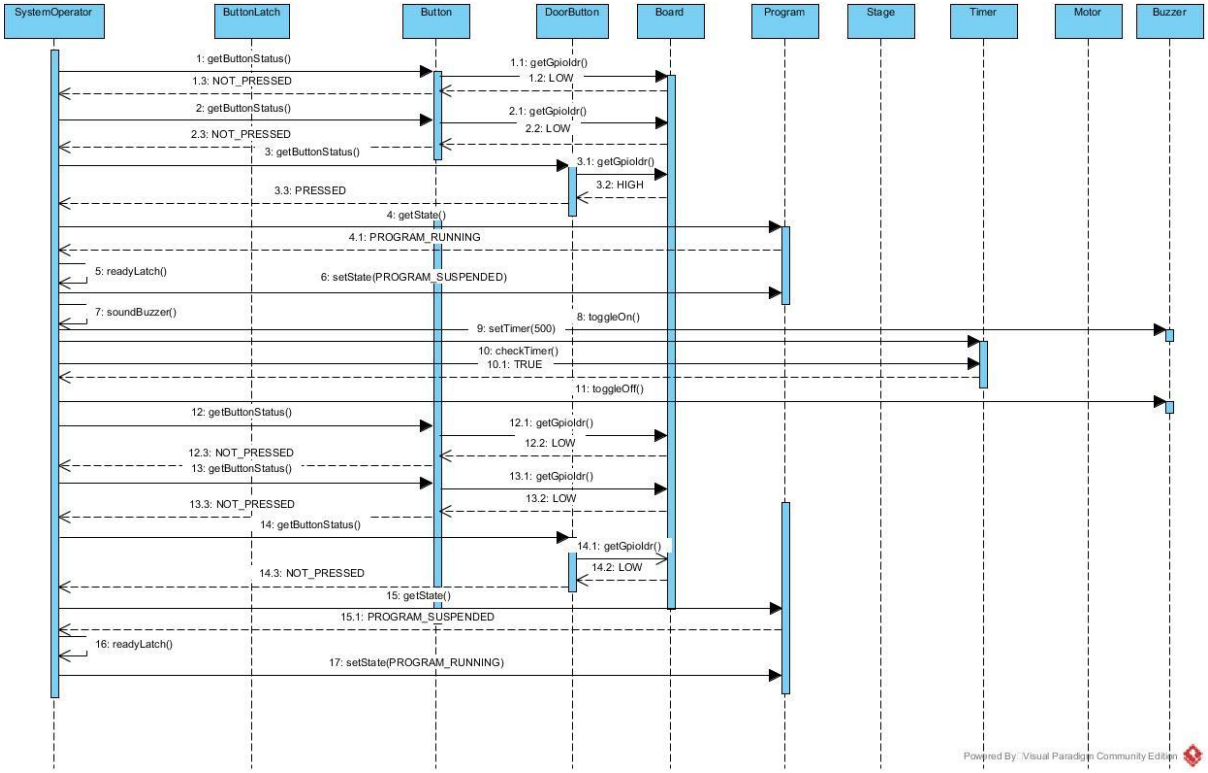
Then in the Resume sequence, steps 7-10.1 show that the Accept button has been pressed and the program is in a paused state, therefore the program is set back to the PROGRAM_RUNNING state. Steps 12.1-12.1.7 show an example of the RINSE_HIGH wash program state operating the motor object.



Sequence 3: Suspend and Resume

Steps 1-4.1 show that the door has been opened and the program was in the PROGRAM_RUNNING state, therefore the program is set to PROGRAM_SUSPENDED in step 6.

The buzzer is briefly sounded to alert the user of the opened door in steps 7-11.



Excluding the part of the sequence which polls the Button/DoorButton objects for the current program state, previously shown processes have been shortened.

Conclusion

Having reached the end of the development cycle for this software program, verification and validation can occur. Review and test of the application's final design demonstrates that the initial system requirements set at the beginning of development have been quantitively met.

Furthermore, the qualitative requirements of the customer regarding abstraction methodology in the development process have been considered to be met throughout the application code.