Initial Design Report

DEVEMB

ResolveSOFT

Fatih Mehmet DOĞU

Hayri Çağlayan ERDENER

Adem HALİMOĞLU

Ulaş TUTAK

December 4, 2006

Contents

Chapter 1

Introduction

1.1 Purpose of the System

In DEVEMB project, we are going to develop a software development environment, compiler and emulator for a specific embedded system card which is PIC demo board that is designed for CEng 336 Embedded Systems course. By using DEVEMB, users will be able to compile, debug and test their programs in the virtual card emulated by software. Since, testing and simulation software is not integrated, testing is very difficult. Therefore, DEVEMB will be the software which integrates the simulation and testing.

In this project we are developing software which will be used in industry and education. Therefore, our customers will be students, engineers and technicians. Our product has to work correct and fast. The interface has to be easy to understand and by using our product one should develop his/her program rapidly. Also, our product should include all the necessary tools in order to program 16F877 PICmicro microcontroller.

DEVEMB is an integrated tool set for the development of embedded applications for Microchip's PIC 16F877 microcontrollers. DEVEMB runs as a 32-bit application on Windows, is easy to use and includes software components for fast application development debugging. DEVEMB serves as a software emulator, compiler and development environment for CEng Card which is used in CEng336 course. Users will be able to compile, upload and debug their programs to the card. Also they will be able to test their programs

in the virtual card emulated by software.

1.2 Design Goals

1.2.1 Extensibility

Although our project is in a specific case and conditions are restricted, (as we can see from data flow diagrams) our system is designed so that it can be extended in particular case. As we will generate a basic high level compiler for example we can compile simple hex files. By extending it PIC processor can generate more complex codes. Also the emulator that we make for using DEVEMB can be similar to the PICmicro microcontroller, and the user can do all kinds of work from the screen. This will help CEng336 classes mostly.

1.2.2 Robustness

The system should be able to manage invalid user inputs or inconsistent conditions. It provides error checking to ensure the right input format and returns errors and warnings to the user.

1.2.3 Reliability

The system should produce the expected output for a valid input at all times. Any unexpected work or error will be reported immediately, and the system prints errors to the screen for further applications in order to protect them.

1.2.4 Functionality

The system should function according to the requirements specified in Requirements Analysis Report.

1.2.5 Usability

The project should be user friendly. The goal is to provide the user an easy- to- use interface. The design is chosen due to the familiarity of most users with this kind of interface. It consists of simple interface and

the user who is familiar with 16F877 PICmicro microcontroller can use the system efficiently and it also provides the user more simple features to understand. The project provides user/system interaction. The user is placed in a familiar environment, which eases the general use of the application.

1.3 Overview

This report explains the design of our application in detail and provides an overview of functionality and implementation of our program. Throughout this document the following subsections will be stated: UML Diagrams and Dynamic View. In the UML Diagrams sections, the reader can find Use-Cases, Class Diagrams, Class Descriptions and Sequential Diagrams. Some of these diagrams are explained more briefly with descriptions. In the Dynamic View section, the actions that the user able to do are explained.

Chapter 2

Behavioral Description of the Design

2.1 Simulation Module

2.1.1 Use-Case: Simulation

Simulate Program

User can load a program to simulator, configure the simulator, run or debug the program and interact with the simulator.

Preconditions There must be a compiled program or a binary file.

Trigger User start the simulator by clicking a button from the user interface.

Basic Course of Events user should start the simulator. User opens a file. User configures the simulator. User starts the simulation. Finally, either the program ends normally or user halts it.

Postconditions Output files must be created. Output files must be saved and closed.

Configure Simulator User can configure the clock, and stimuli.

Preconditions s: In addition to the

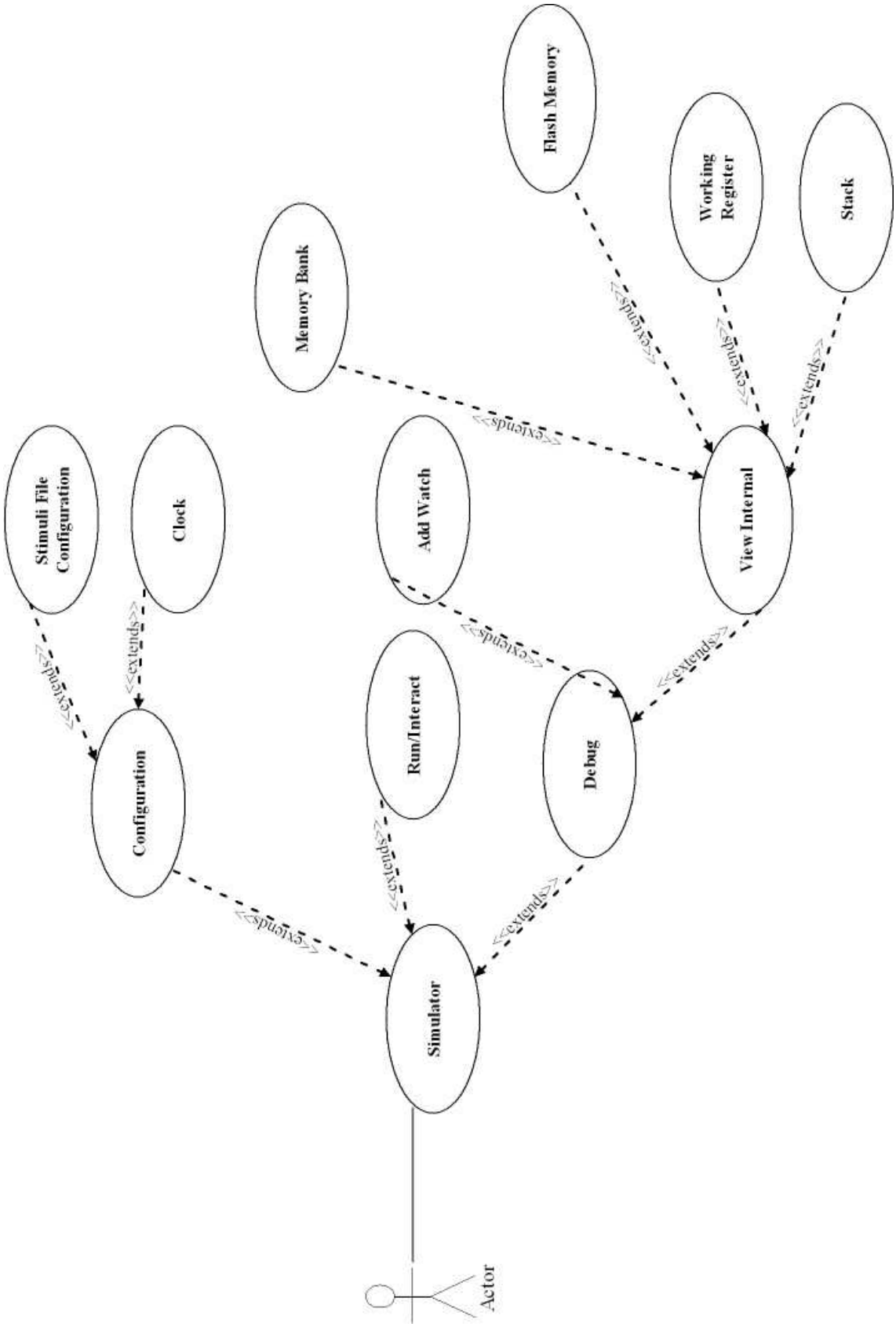


Figure 2.1: Use-Case: Simulation

Preconditions s explained in simulate program, user must entered to simulation mode but simulation must not be started.

Trigger User can click load stimuli button or changes the clocking values from the interface

Basic Course of Events User should select configure mode. Configuration pop-up appears. User changes the settings or values that s/he wants. User can click the OL or CANCEL buttons to finish the configuration mode.

Postconditions Changes must be saved to a configuration file.

Run Simulation This use case shows the interaction of user with the simulator after starting the simulation.

Preconditions Simulation mode

Trigger User Interface

Basic Course of Events User starts the simulation. After starting simulation user can click buttons. Simulation ends normally or user halts it. Simulator shows the changes on the LEDs, LCD, etc.

Postcondition:

Stimulus files must be created in this phase.

Sequence Diagrams

Activity Diagram

2.1.2 Debugging the Code

Use-Case

User can trace the program view contents of the memory.

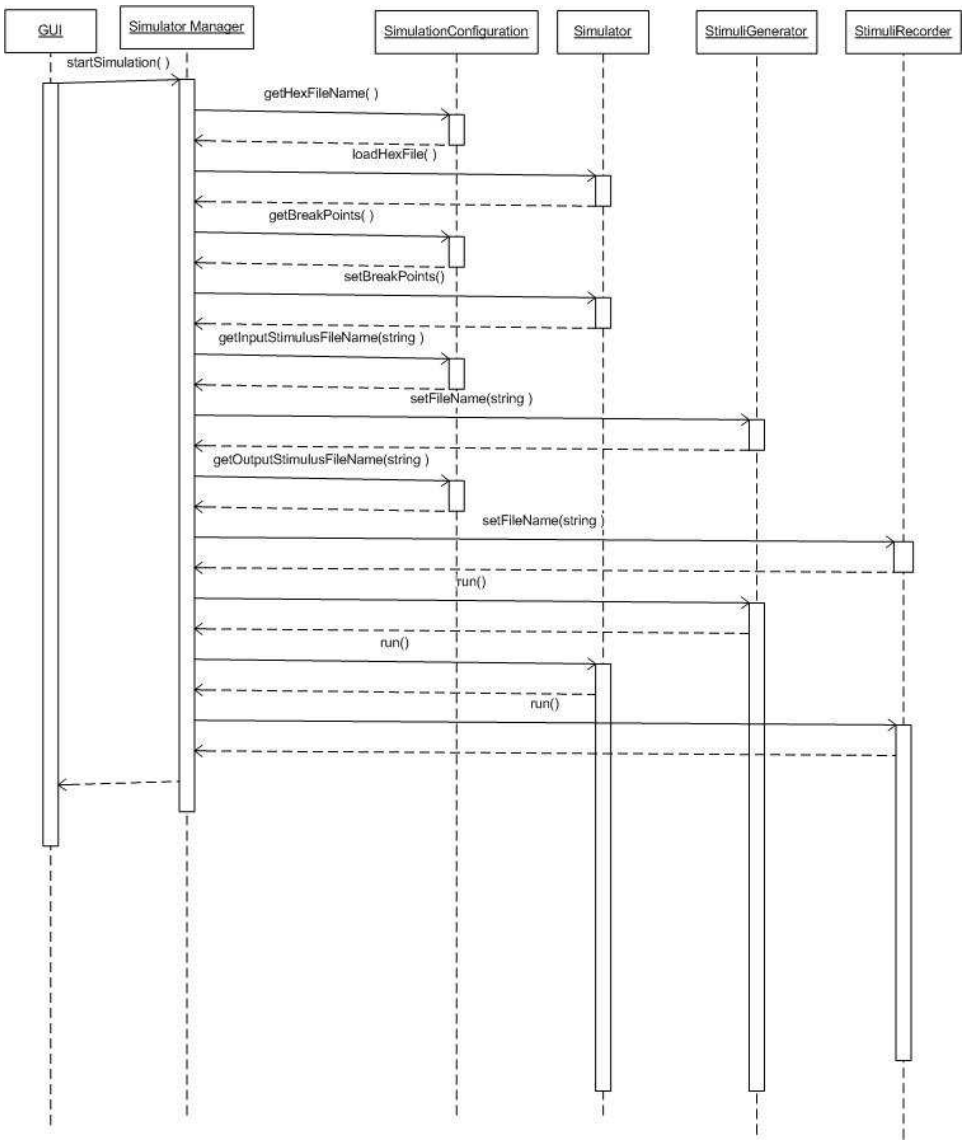


Figure 2.2: Simulation Setup

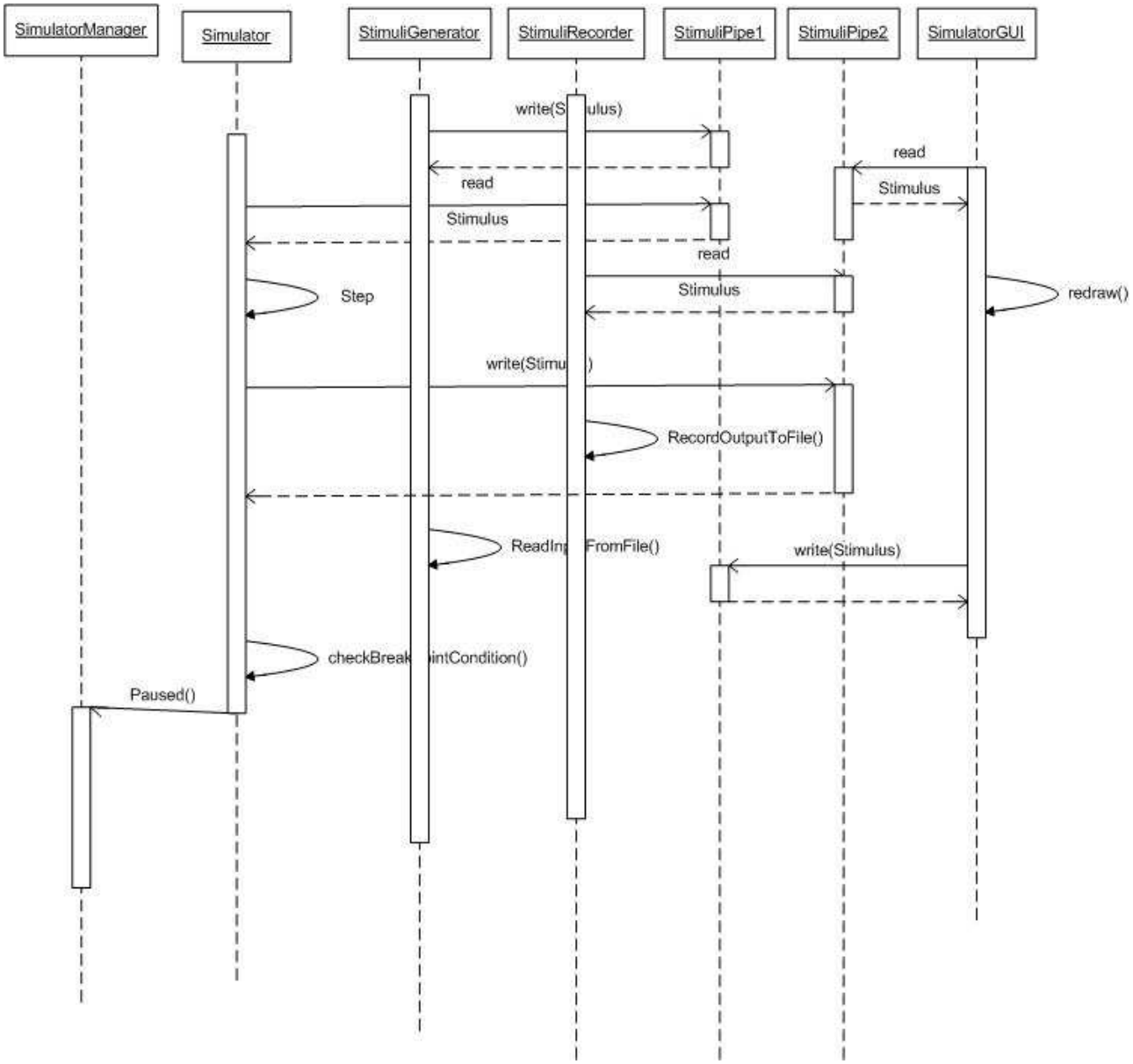


Figure 2.3: Simulation Cycle 1

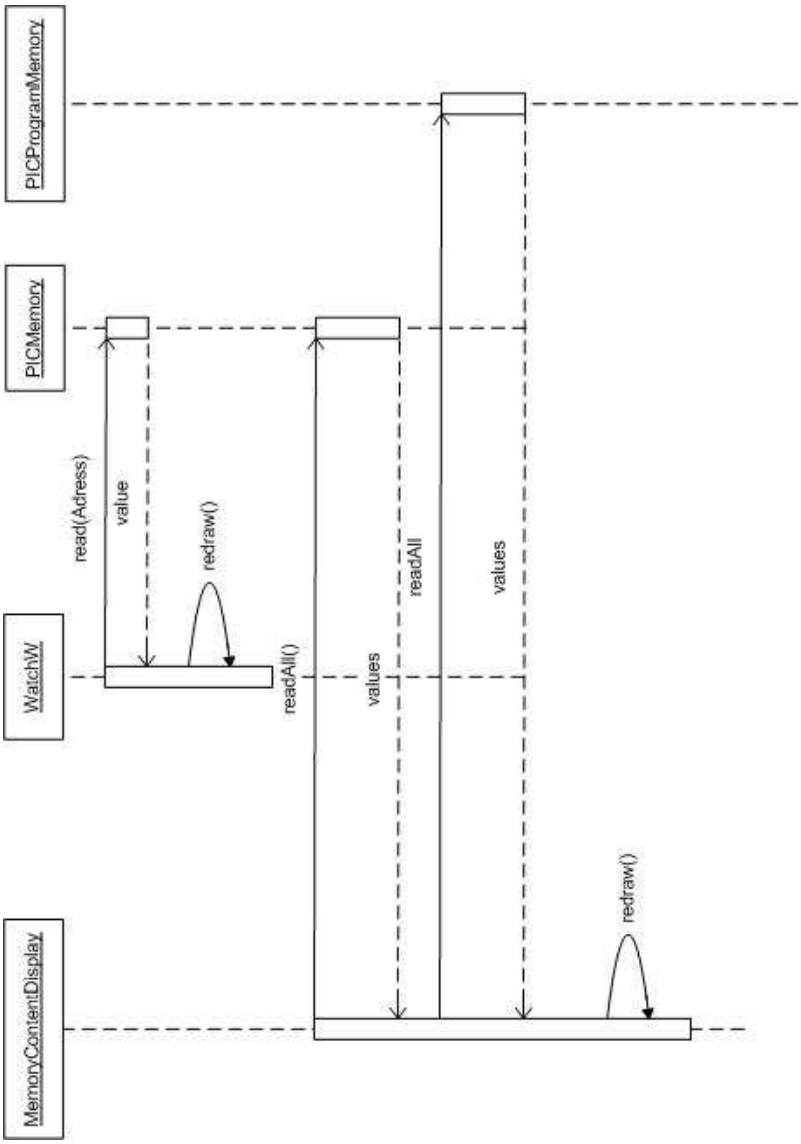


Figure 2.4: Simulation Cycle 2

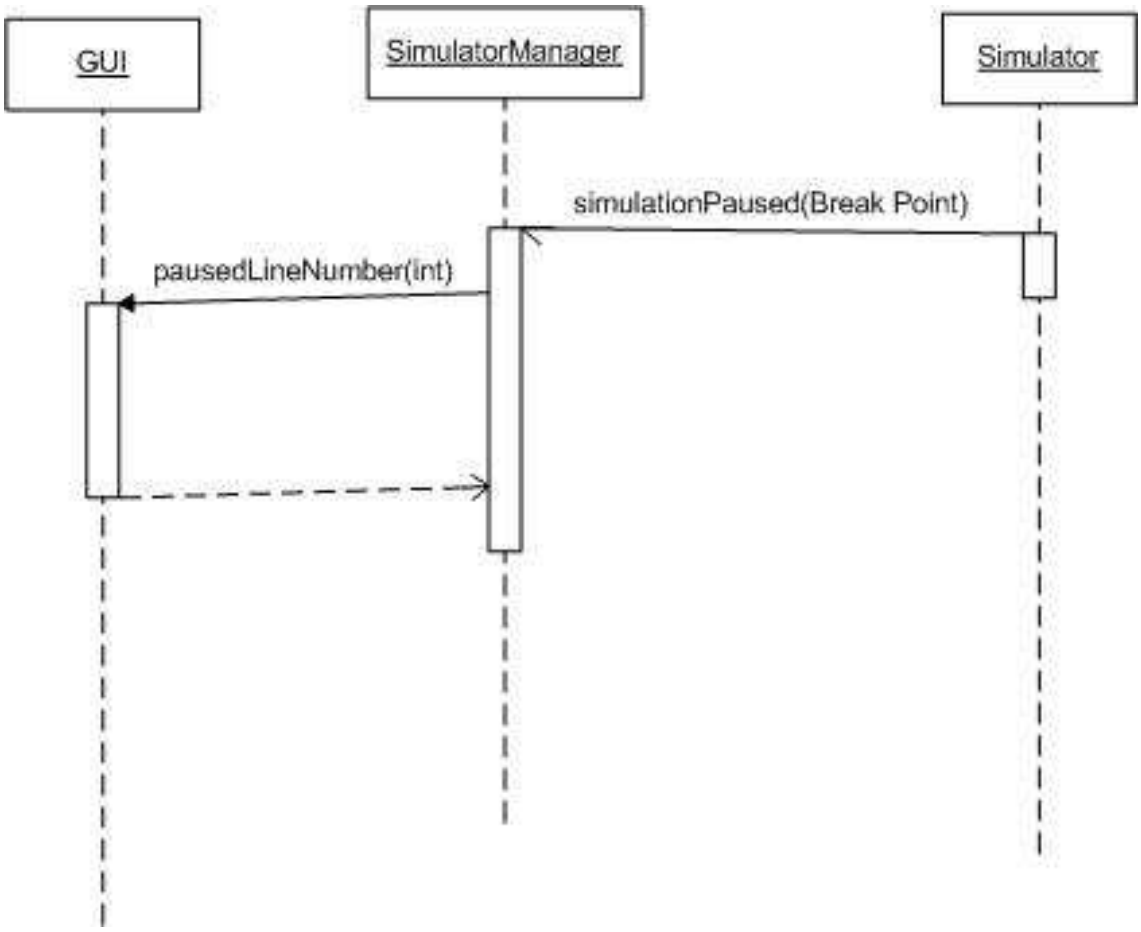


Figure 2.5: Simulation Breakpoint Reached

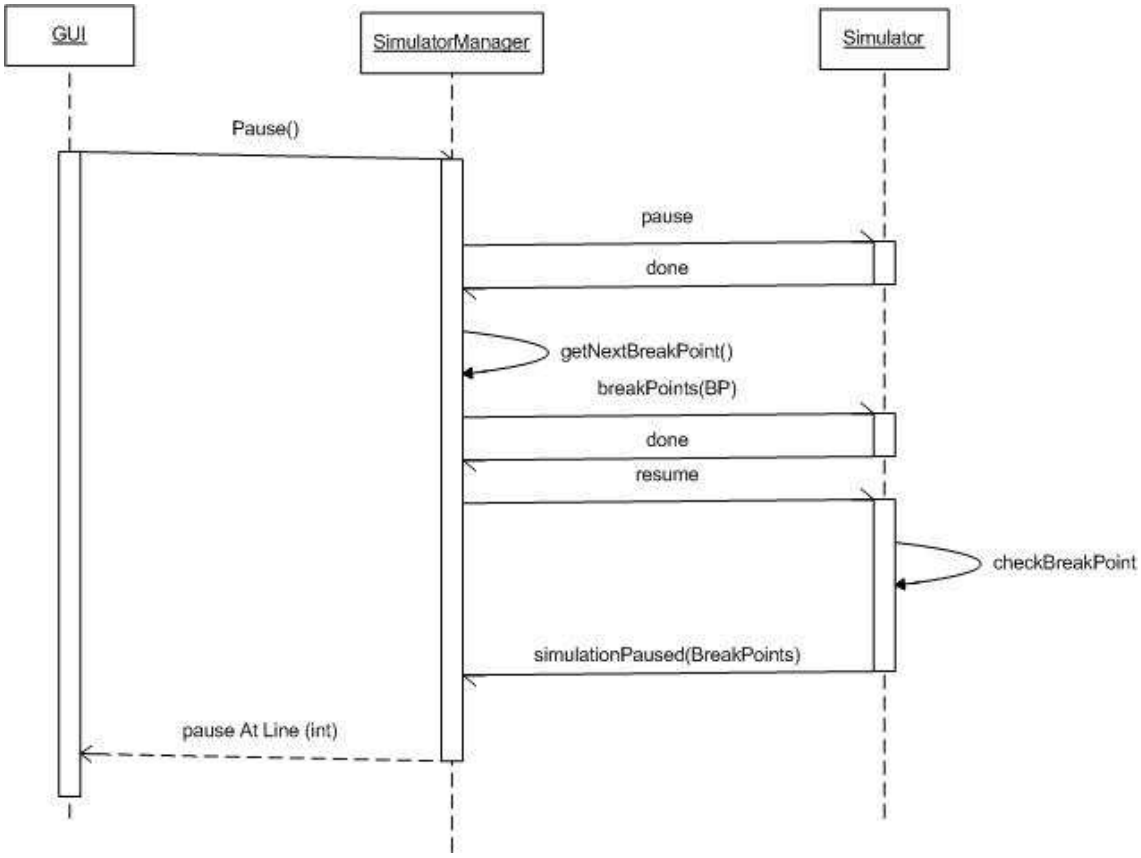


Figure 2.6: Simulation Pause

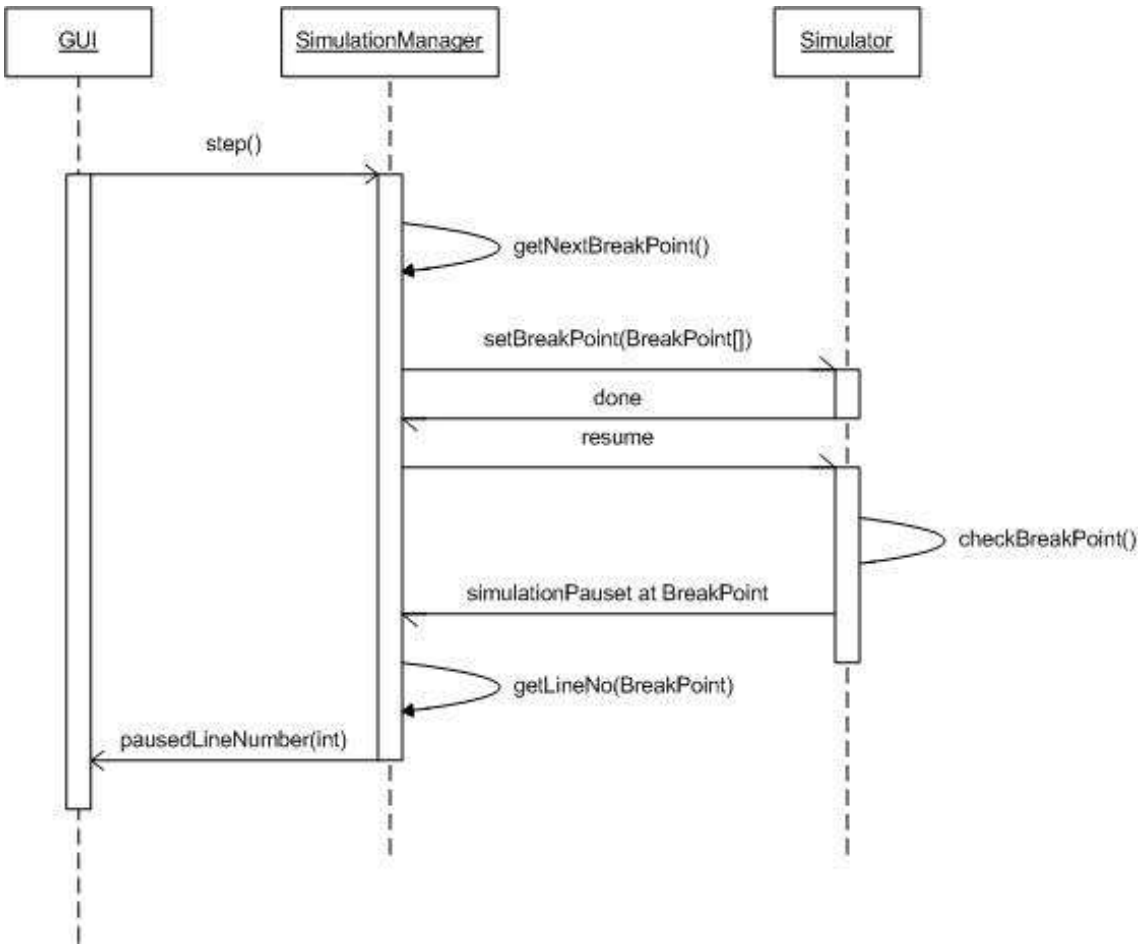


Figure 2.7: Simulation Step

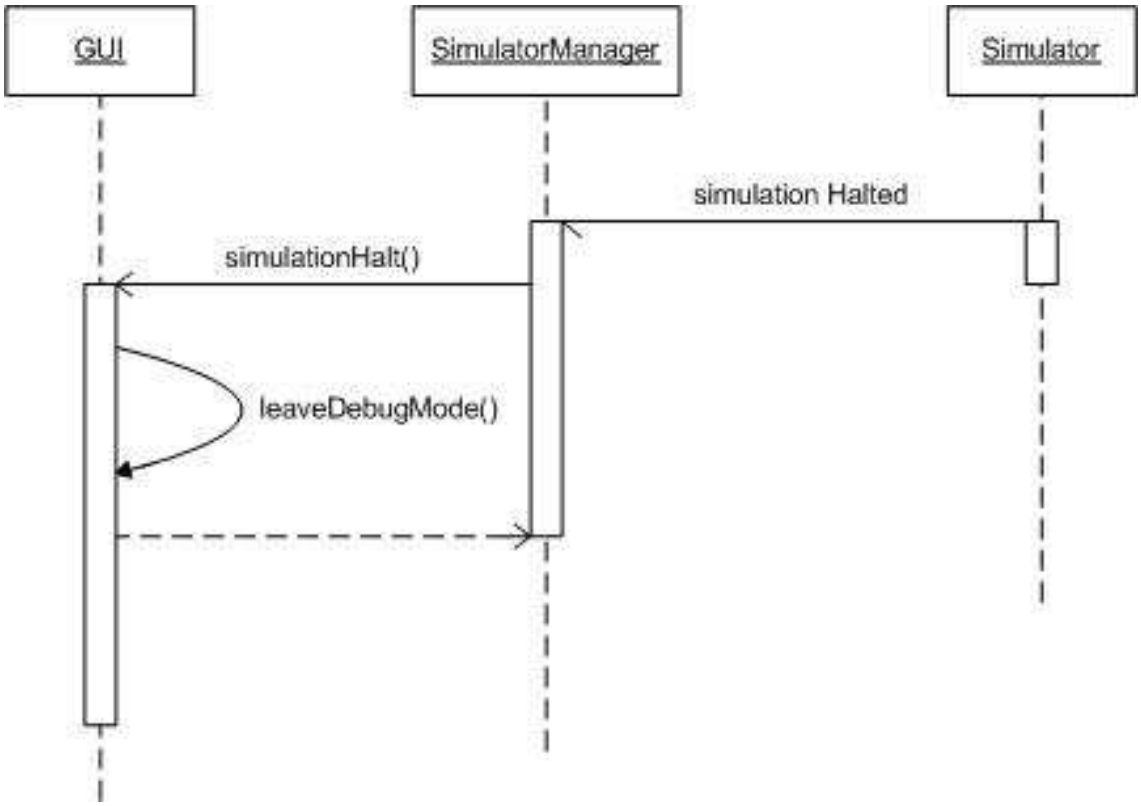


Figure 2.8: Simulation Halt

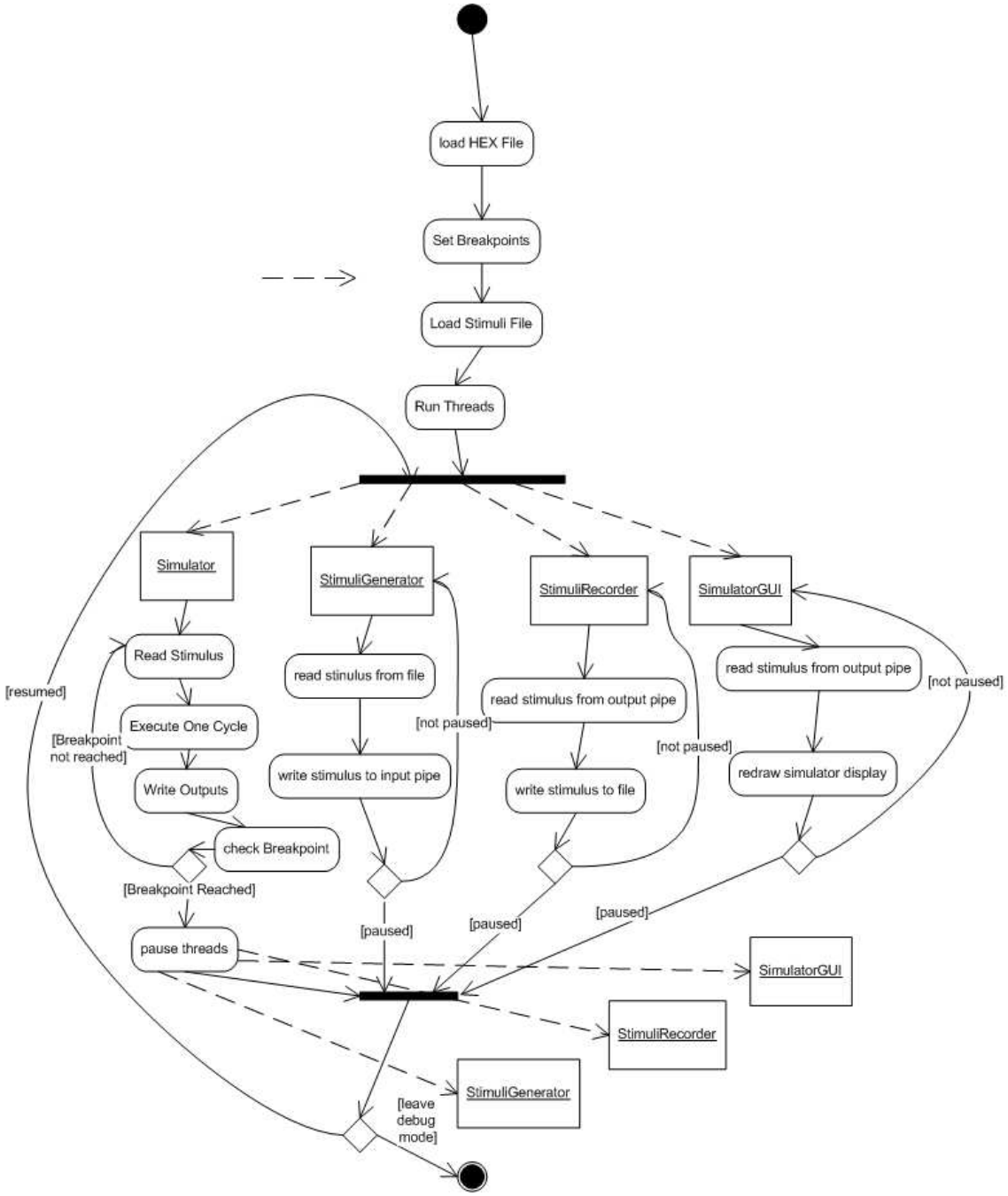


Figure 2.9: Simulator Activity Diagram

Preconditions There must be a program which is compiled for debugging or binary file.

Trigger User start the debugger by clicking a button from the user interface.

Basic Course of Events User selects the memory location that s/he wants to see the content. User puts the breakpoints if there is. User traces the program. Debugger shows the contents of the selected memory locations. Debugger ends normally or user halts it.

Sequence Diagrams

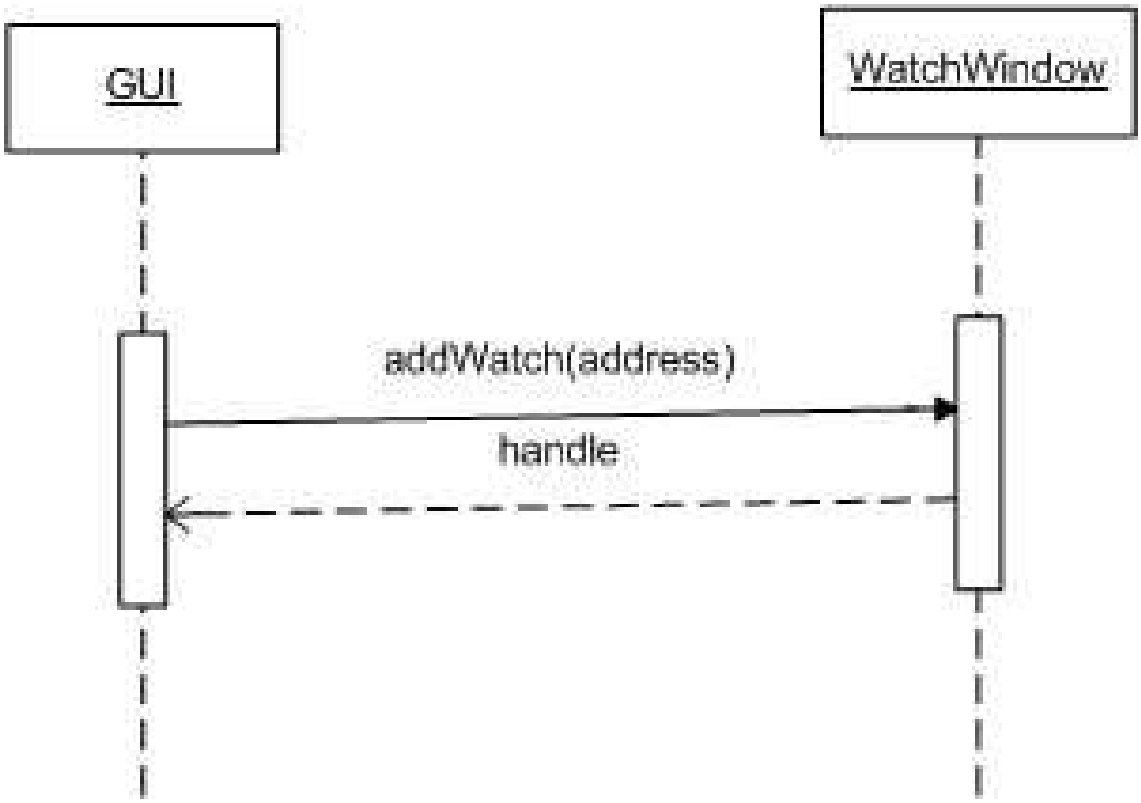


Figure 2.10: Debug Add Watch

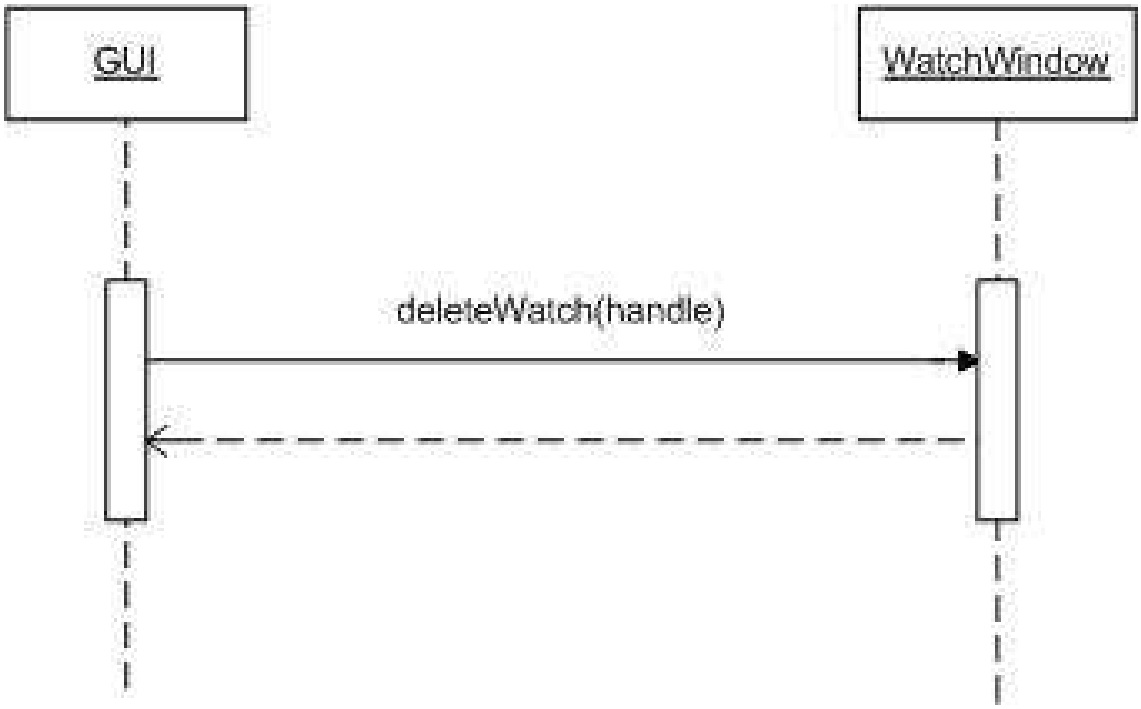


Figure 2.11: Debug Delete Watch

Activity Diagram

2.2 Compilation Module

2.2.1 Use-Case: Compilation

User may chose to generate pic machine codes and debugging information from C and Assembly source files opened in the editor component of the user interface or generate assembly code from pic machine instructions.

Compile a C Program

User can compile a program to generate output file(s) in a specified format.

Preconditions There must exist a C source file.

Trigger User starts the compiler by clicking a button or selecting a menu item labeled appropriately from the user interface.

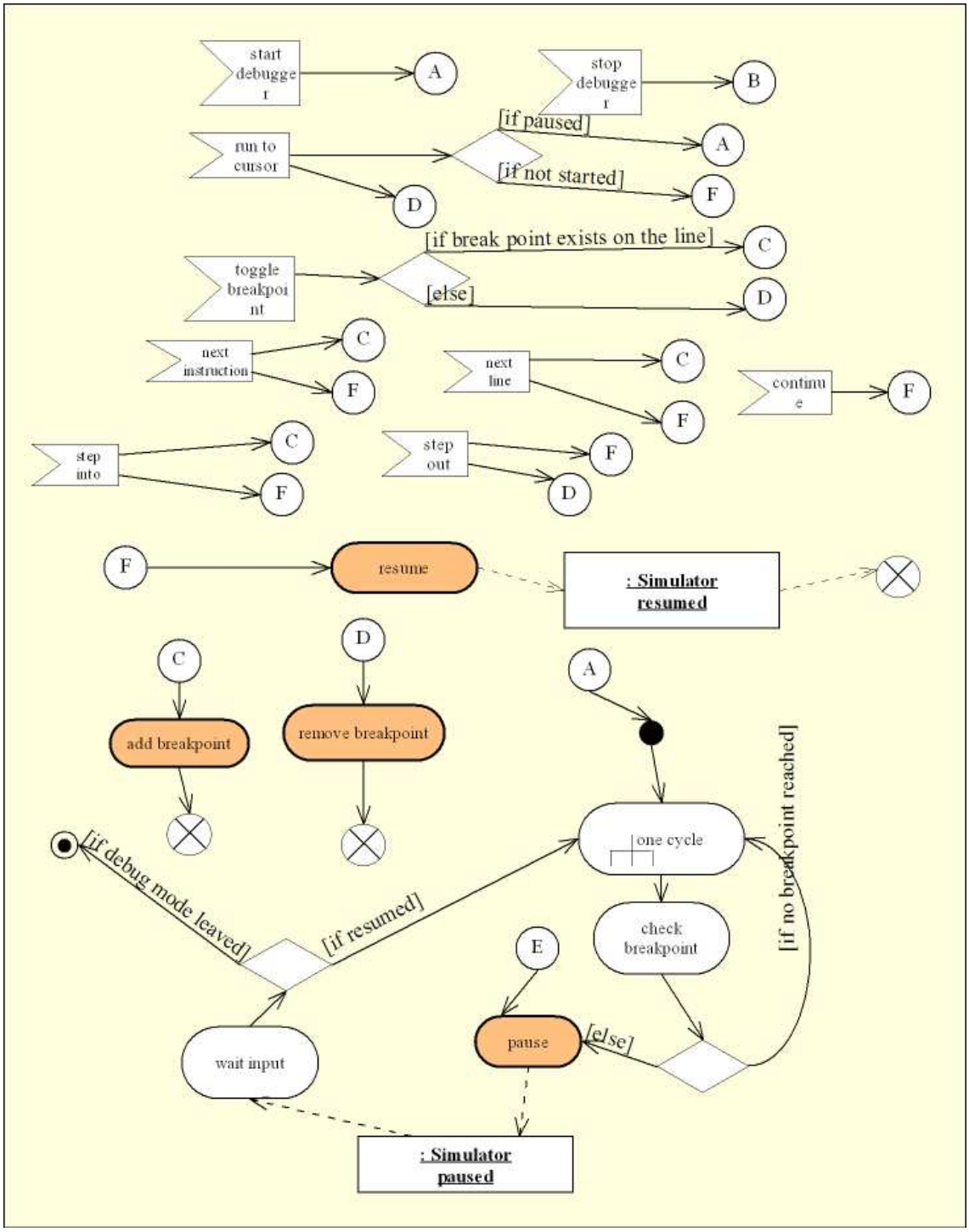


Figure 2.12: Debug-mode Activity Diagram

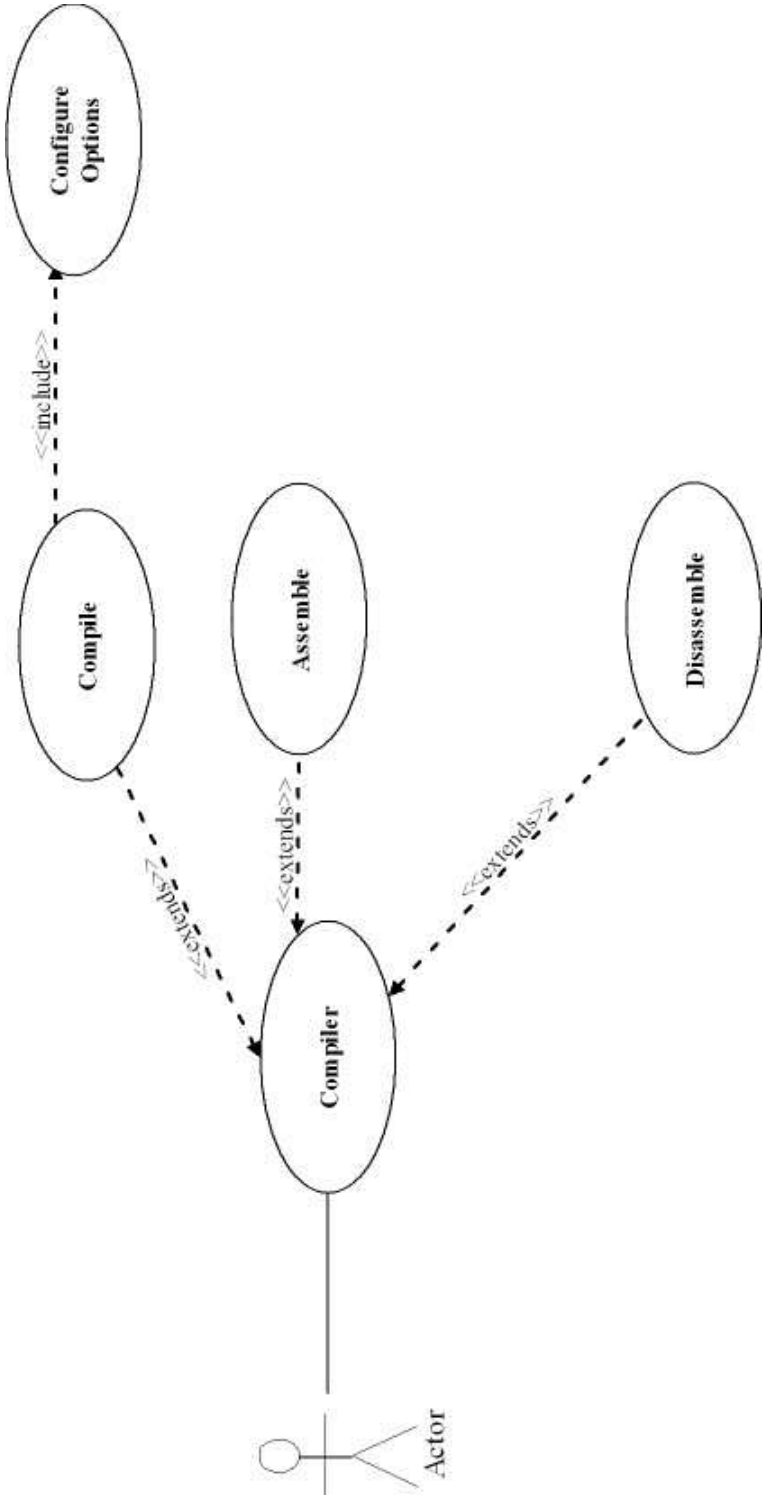


Figure 2.13: Use-Case: Compilation

Basic Course of Events User starts the compilation. Compiler program processes the source file. The compiler might generate warnings. It either generates error messages and quits without completion of the compilation or completes the compilation and reports success. Any messages generated by the compiler are listed in a window.

Postcondition:

Output files, e.g. relocatable object code (.o), assembler source(.asm), assembler listing (.lst), executable for debugging/simulation file (.cod), executable for programming the processor (.hex), depending on the compilation options and the level of the compilation reached when the compiler program halts.

Assemble an Assembly Program

User can assemble an .asm file into one or more of the output formats mentioned above.

Preconditions s: There must exist an assembly source file.

Trigger User starts the assembler by clicking a button or selecting a menu item labeled appropriately from the user interface.

Basic Course of Events Basic course of events are the same as compilation use-case.

Postconditions

Postconditions are the same as compilation use-case.

Disassemble an Executable

User can assemble an .hex file into a .asm file.

Preconditions s: There must exist an .hex file.

Trigger User starts the dis-assembler by clicking a button or selecting a menu item labeled appropriately from the user interface and selecting an executable hex file from a file dialog. Unlike compile and assemble

use-cases in dis-assemble use-case the input file need not be open at the time of processing, as opening a hex file in an editor makes little sense.

Basic Course of Events User starts the dis-assembly. Dis-assembler program processes the hex file. It either generates error messages (e.g. if invalid op-codes are present in the input file) and quits without completion of the dis-assembly or completes the dis-assembly and reports success. Any messages generated by the dis-assembler are listed in a window.

2.3 Program Loading Module

2.3.1 Use-Case: Load Program

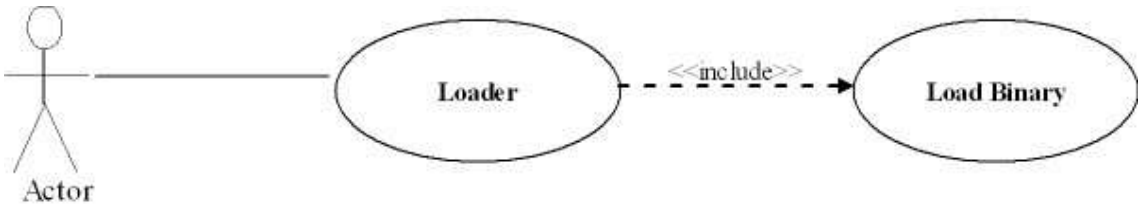


Figure 2.14: Use-Case: Load Program

User can assemble an .hex file into a .asm file.

Preconditions s: There must exist an .hex file.

Trigger User starts the dis-assembler by clicking a button or selecting a menu item labeled appropriately from the user interface and selecting an executable hex file from a file dialog. Unlike compile and assemble use-cases in dis-assemble use-case the input file need not be open at the time of processing, as opening a hex file in an editor makes little sense.

Basic Course of Events User starts the dis-assembly. Dis-assembler program processes the hex file. It either generates error messages (e.g. if invalid op-codes are present in the input file) and quits without completion of the dis-assembly or completes the dis-assembly and reports success. Any messages generated by the dis-assembler are listed in a window.

2.3.2 Sequence Diagrams

Load Executable to Board

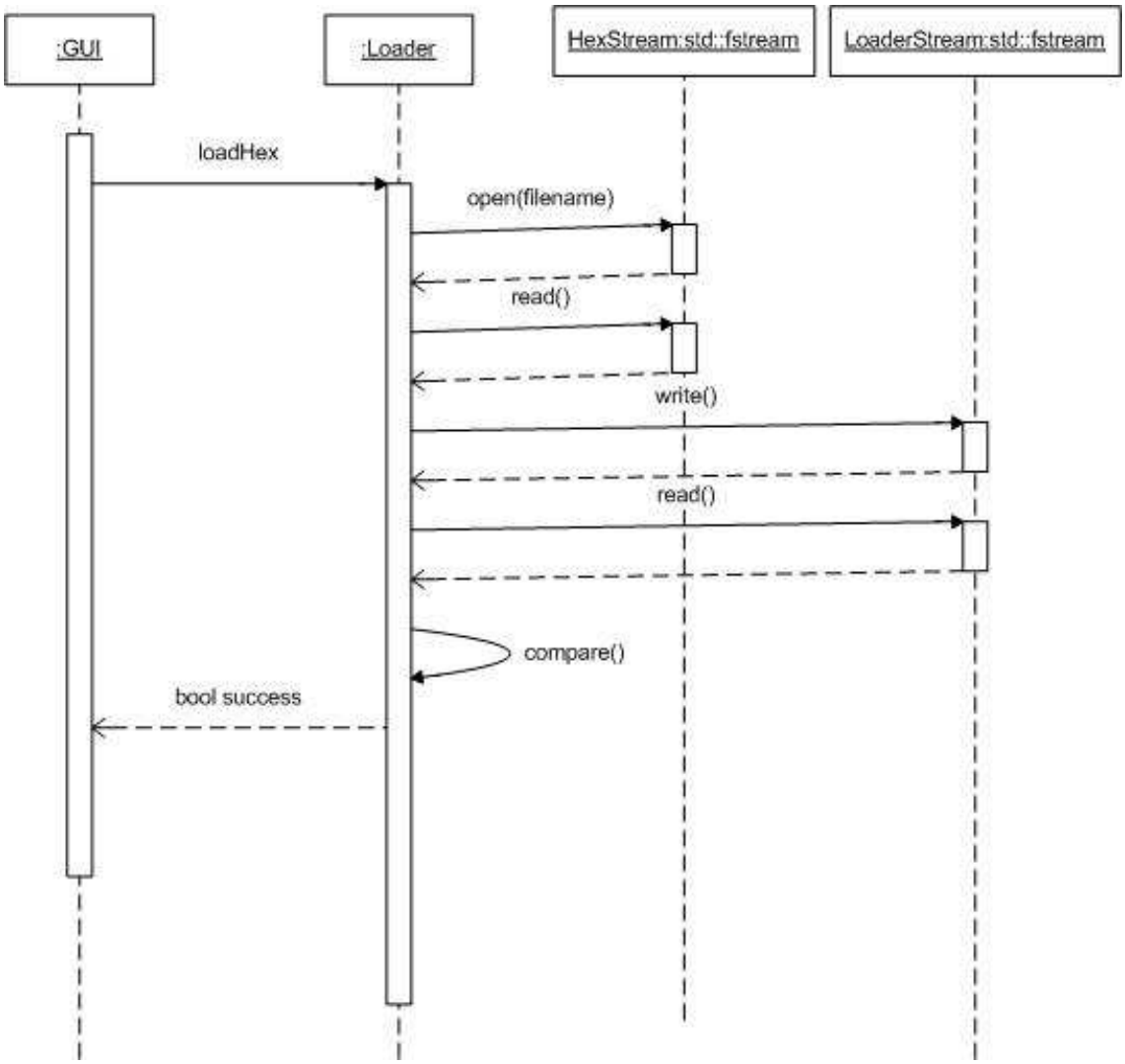


Figure 2.15: Load Executable to Board

2.4 Project Management Module

2.4.1 Use-Case: Project Manager

Preconditions DEVEMB must be executed.

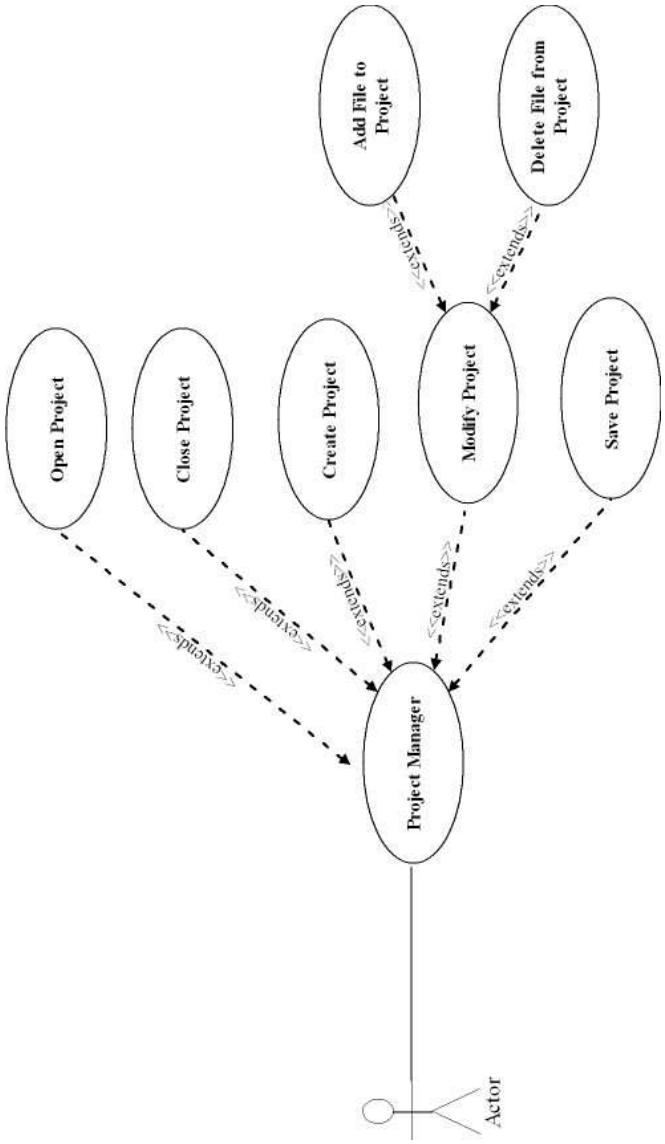


Figure 2.16: Use-Case: Project Manager

Trigger User start managing the project by clicking a button from the user interface and also can see the files of the project from the user menu.

Basic Course of Events user should opened the project. user should close the project. user should create the project. user should modify the project. user should save the project.

Postcondition:

Project should be closed 2.1) Modifying a Project

Preconditions A project must be opened

Trigger User starts modifying the project from the user menu.

Basic Course of Events User should add a file to the project. User should delete a file to the project. Postcondition:

New configuration must be set.

2.4.2 Sequence Diagrams

Close Project

Open Project

Save Project

Add a File to Project

Remove a File from Project

2.5 File Management Module

2.5.1 Use-Case: File Manager

Preconditions DEVEMB must be executed.

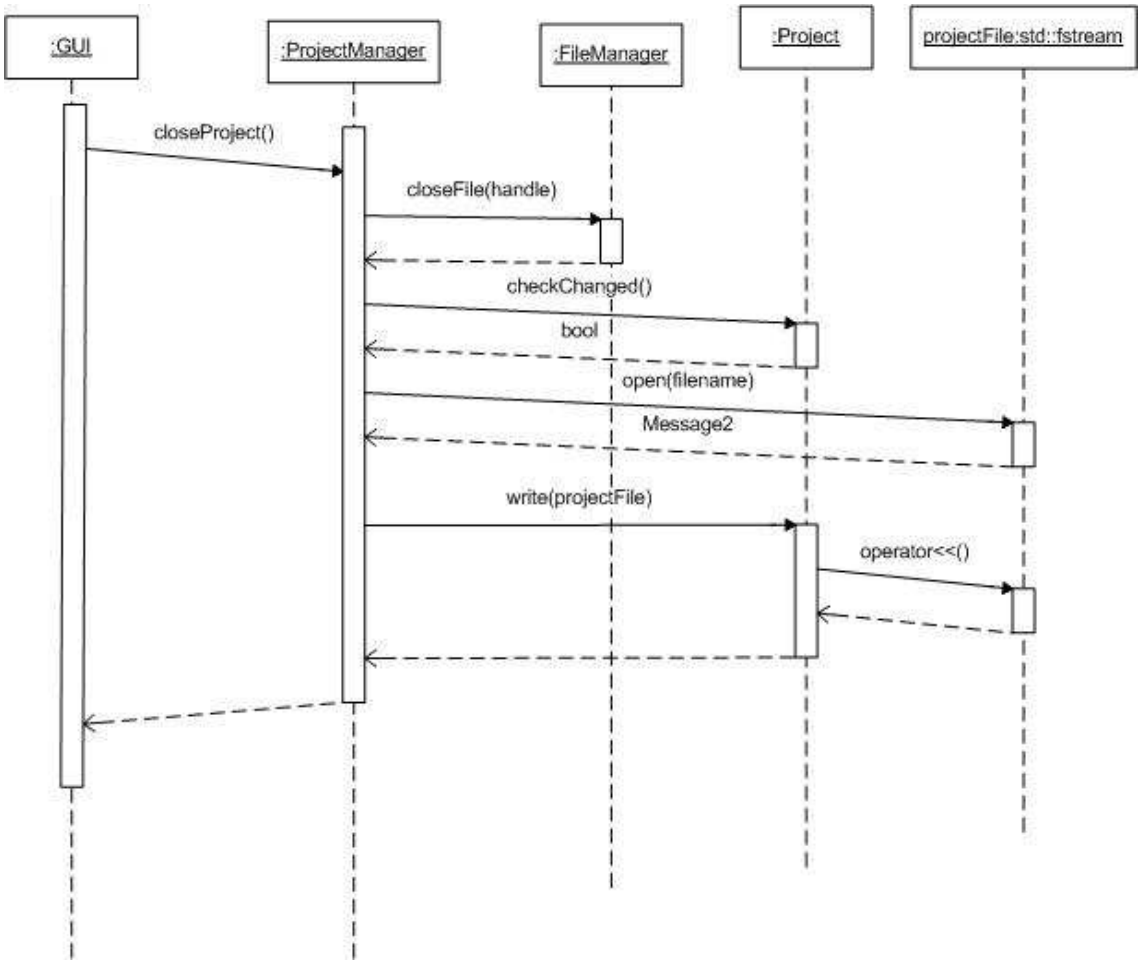


Figure 2.17: Close Project

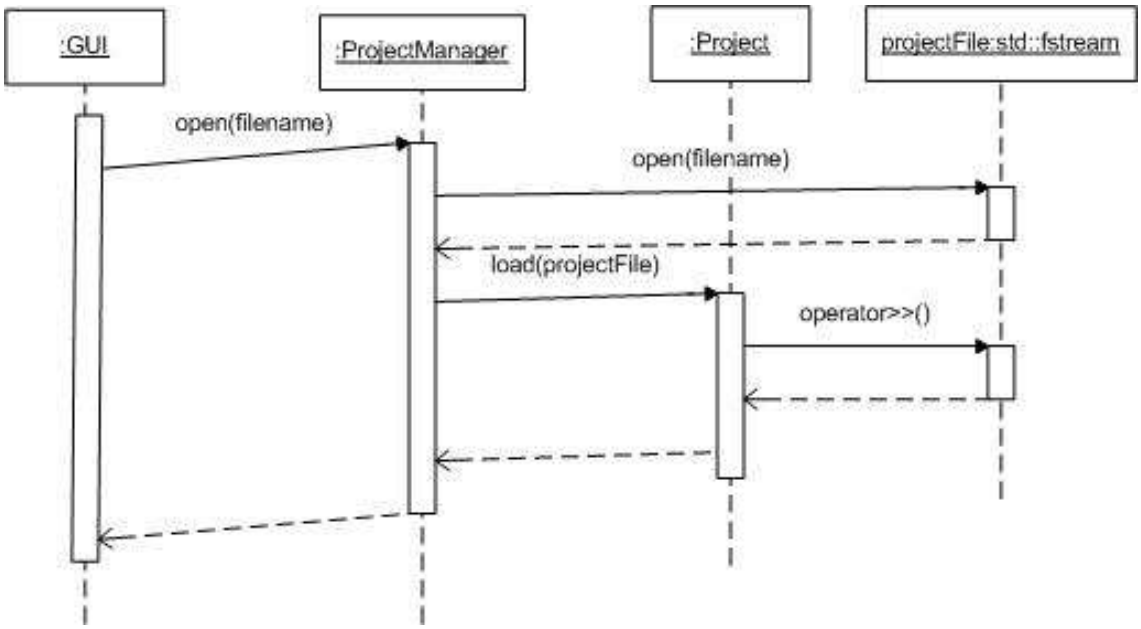


Figure 2.18: Open Project

Trigger User start managing the files by clicking a button from the user interface.

Basic Course of Events user should; edit a file. create a file close a file save a file.

Postcondition: Files should be closed

2.5.2 Sequence Diagrams

Close File

New File

Open File

Save File

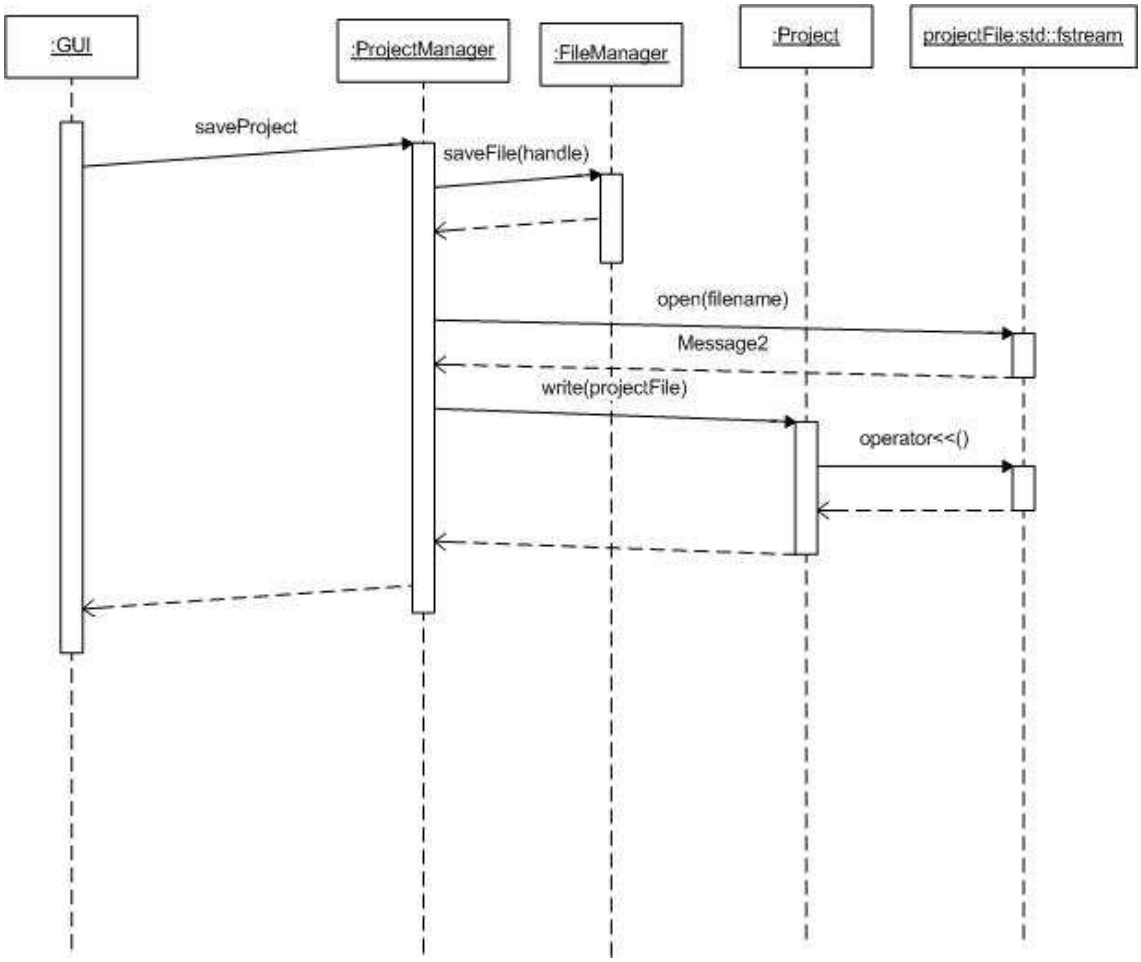


Figure 2.19: Save Project

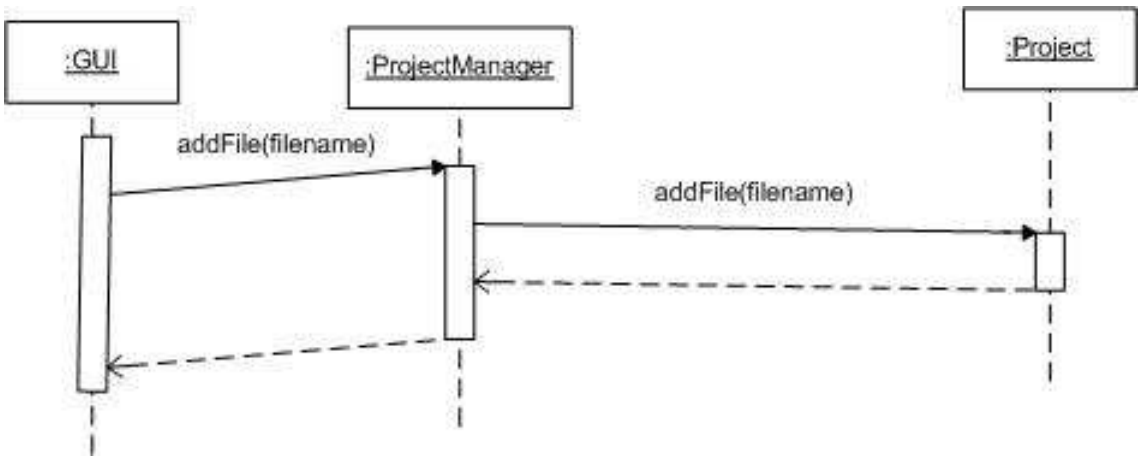


Figure 2.20: Add a File to Project

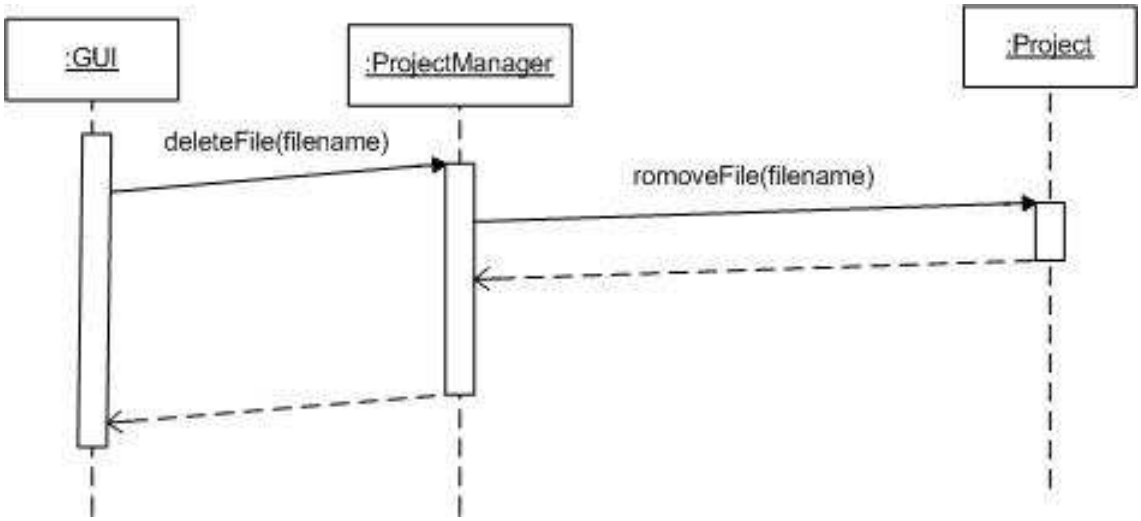


Figure 2.21: Remove a File from Project

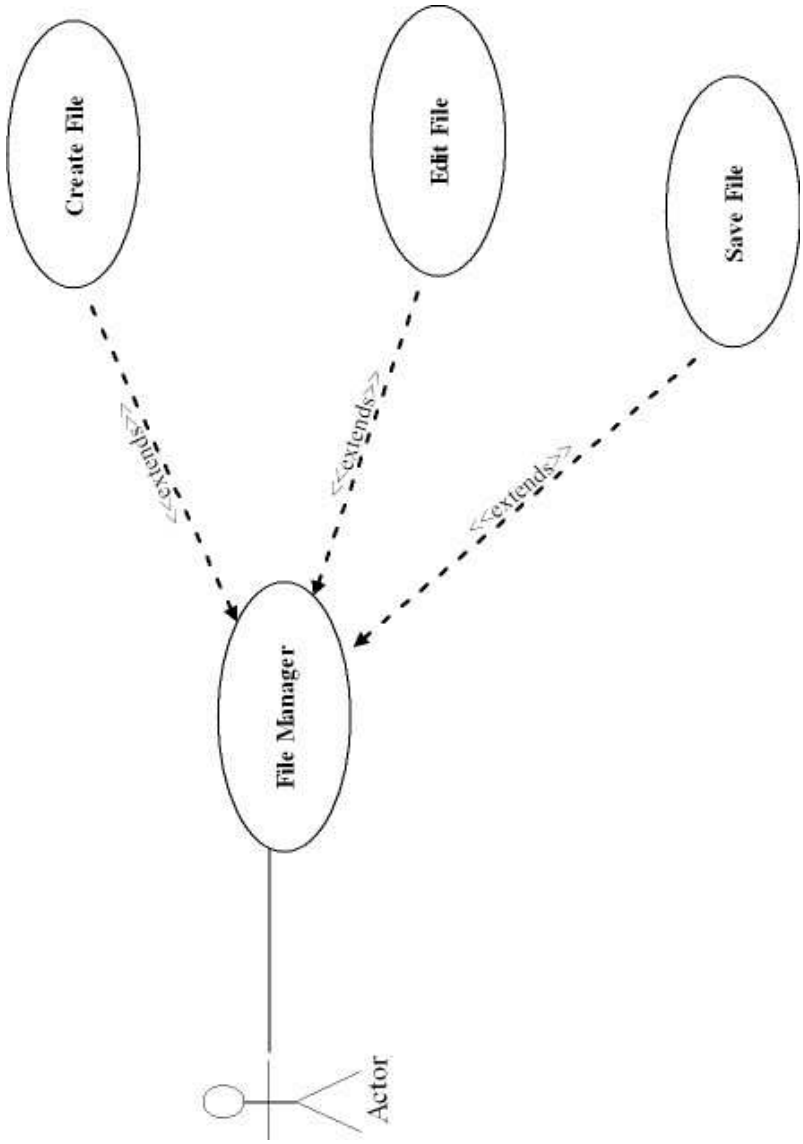


Figure 2.22: Use-Case: File Manager

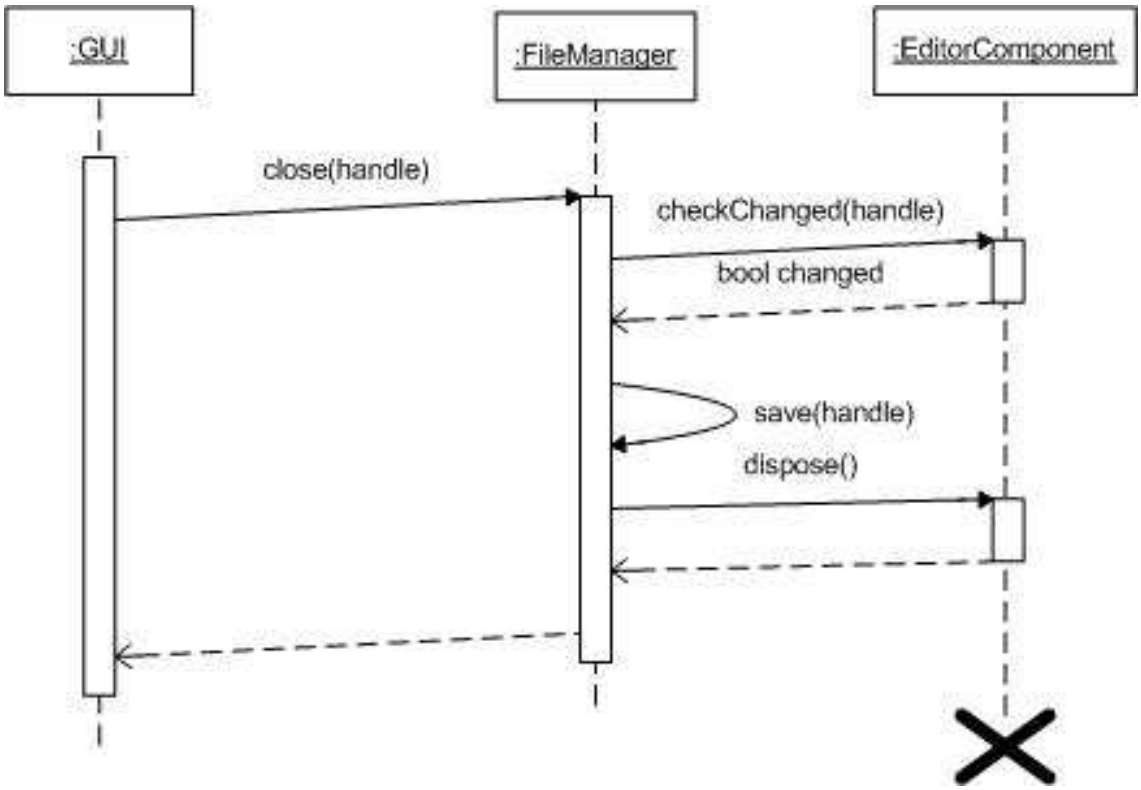


Figure 2.23: Close File

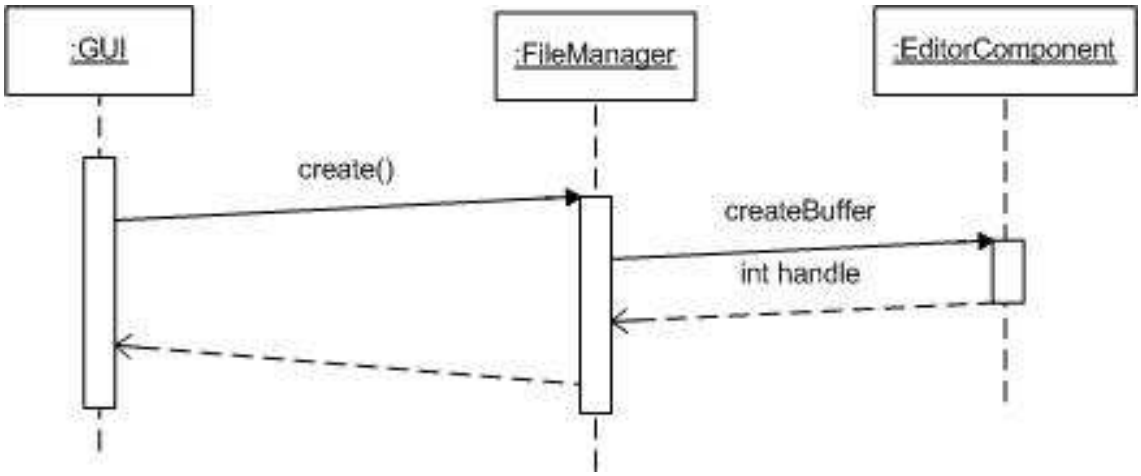


Figure 2.24: File

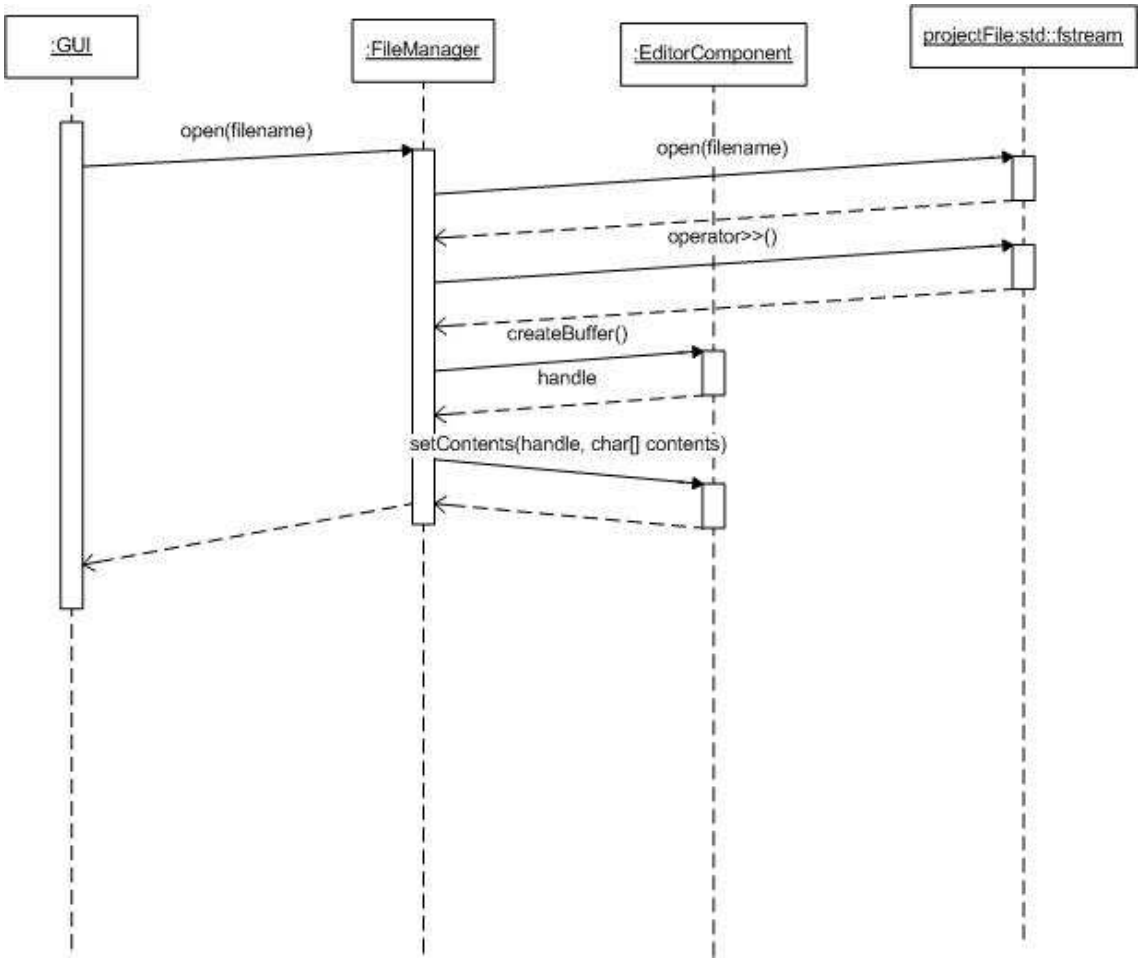


Figure 2.25: Open File

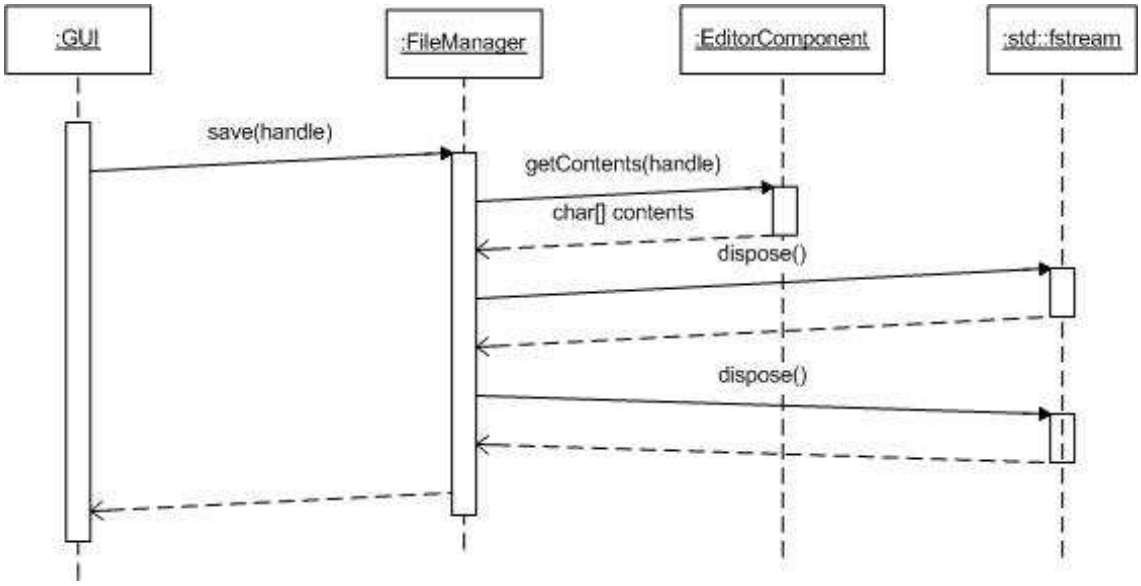


Figure 2.26: Save File

Chapter 3

Static Structure

3.1 Classes

3.1.1 GUI Class

This class represents graphical user interface and thus very important class. This contains menu and tool bars and is an external structure for the software. It triggers most of the events.

Method Name	Type	Arguments	Method Description
pauseLineNumber(int)	Public:void	void	It pause the simulation at given line number.
leaveDebugMode()	Public:void	void	It ends the debug mode.
simulationHalt()	Public:void	void	It ends the simulation.

Table 3.1: GUI class

3.1.2 ProjectManager Class

This class manages the project. It has methods for closing, deleting, opening the files in a project. It also adds a new file to a project.

3.1.3 FileManager Class

This class manages the file operations. It has methods that handles file operations such as: Opening a file Closing a file Adding a file to a project Creating a file in a project Saving a file

Method Name	Type	Arguments	Method Description
closeProject()	Public:void	void	It closes the currently opened project.
deleteFile(fileName)	Public:void	void	It deletes the file whose name is fileName from the project.
open(fileName)	Public:void	void	It opens the file whose name is fileName from the currently opened project.
addFile(fileName)	Public:void	void	It adds a file whose name is fileName to the currently opened project.

Table 3.2: ProjectManager Class

Method Name	Type	Arguments	Method Description
closeFile(Handle)	Public:void	void	It closes the file whose id is Handle.
addFile(fileName)	Public:void	void	It adds a file whose name is fileName.
close(Handle)	Public:void	void	It closes the currently opened file.
save(Handle)	Public:void	void	It saves the currently opened file.
create()	Public:void	void	It creates a new file in the project.
open(fileName)	Public:void	void	It opens the file whose name is fileName.
saveFile(Handle)	Public:void	void	It saves the file whose id is Handle.

Table 3.3: FileManager Class

3.1.4 EditorComponent Class

This class represents the text editor of the software. It has methods to handle text editor operations. It checks whether the contents of a code segment in the editor changed. Also, it saves the contents of the editor. It first checks whether a file is saved or not before closing the file.

Method Name	Type	Arguments	Method Description
checkChanged(Handle)	Public:void	bool	It checks whether there is any change or not in the file whose id is Handle and it returns Boolean.
dispose()	Public:void	void	After checking process finish, it closes the file.
createBuffer()	Public:void	void	It creates a buffer to write new text.
setContents(Handle, char[], contents)	Public:void	void	It gets the saved files contents whose id is Handle and sets this contents to the buffer.
getContents(Handle)	Public:void	void	It gets the contents of the buffer, whose id is Handle, to save.

Table 3.4: EditorComponent Class

3.1.5 WatchWindow Class

It handles the operations for displaying the contents of a given memory address. Also, it closes the previous and opens the new one.

Method Name	Type	Arguments	Method Description
addWatch(address)	Public:void	void	It adds a specified memory address to display its content.
deleteWatch(Handle)	Public:void	void	It deletes an added address previously whose id is Handle.

Table 3.5: WatchWindow

3.1.6 Project Class

This class handles the operations of project that is developed. It opens a project and handles the operations of the project files. When a project is closed it checks whether the files in the project is saved or not. It removes files from the project or adds files to the project.

Method Name	Type	Arguments	Method Description
checkChanged()	Public:void	bool	It checks whether there is any change or not in the project that currently opened and it returns Boolean.
write(projectFile)	Public:void	void	It saves the currently opened file of the project.
removeFile(fileName)	Public:void	void	It deletes the file whose name is fileName from the project.
addFile(fileName)	Public:void	void	It adds the file whose name is fileName to the currently opened project.
load(projectFile)	Public:void	void	It loads a project which is named as projectFile.

Table 3.6: Project Class

3.1.7 ProjectFile:std:fstream Class

This object is an instance of fstream class. It handles the stream operations such as opening, reading from and writing to a file.

Method Name	Type	Arguments	Method Description
open(fileName)	Public:void	void	It opens the file that is named as fileName.
¡¡()	Public:void	void	writes the object given in the parameter to the stream
¿¿()	Public:void	void	reads the object given in the parameter from the stream

Table 3.7: projectFile:std::fstream Object

3.1.8 Loader Class

This class is for loading the program to the PIC. It loads hex files and also checks whether the loaded file is transferred successfully or not.

Method Name	Type	Arguments	Method Description
loadHex()	Public:void	bool	It starts the loading a hex file process and returns a Boolean whether it starts or not.
compare(Handle)	Public:void	void	It compares the file whose id is Handle whether it is loaded successfully or not.

Table 3.8: Loader Class

3.1.9 StimuliPipe1 Class

This class is for input stimuli. It has methods for reading and writing.

Method Name	Type	Arguments	Method Description
write(stimulus)	Public:void	void	It writes the stimulus that is produced by StimuliGenerator to a file.
read()	Public:Stimulus	void	It reads the stimulus from StimuliGenerator.

Table 3.9: StimuliPipe Class

3.1.10 SimulatorManager Class

This is a huge class that manages the simulation. It has operations for specifying the location of the next break point. It starts the simulation and can execute the program step by step. It returns the pausing location of the program.

Method Name	Type	Arguments	Method Description
getNextBreakpoint()	Public:void	void	It determines where the next breakpoint is.
pause()	Public:void	void	It stops the simulator while it is running as a result of the users request.
startSimulation()	Public:void	void	It starts the simulation.
step()	Public:void	void	It executes an instruction for each execution of it.
getLineNo(Breakpoint)	Public:void	void	It gets the line number of the breakpoint that is set before simulation.

Table 3.10: SimulatorManager Class

3.1.11 StimuliRecorder Class

This class is for output operations. It records the results of a run and specifies the output file name.

Method Name	Type	Arguments	Method Description
recordOutputToFile()	Public:void	void	
setFileName(fileName)	Public:void	void	It sets the fileName as the file name of the output stimulus file.
run()	Public:void	void	

Table 3.11: StimuliRecorder

3.1.12 Simulator Class

This class is also for simulation. It checks whether the conditional break is satisfied in the debugging execution of the program. It loads the hex file that is to be debugged. And set break points in the program that is to be debugged.

Method Name	Type	Arguments	Method Description
step()	Public:void	void	It gets the line number of the breakpoint that is set before simulation.
checkBreakpointCondition()	Public:void	void	It checks whether there is any breakpoint or not on the current line.
loadHexFile()	Public:void	void	It loads the HexFile to the simulator.
setBreakpoints(Breakpoint)	Public:void	void	It set a breakpoint a specified place.

Table 3.12: Simulator Class

3.1.13 PICMemory Class

This class is for PIC memory. It has methods that reads a given memory location and reading the memory locations.

Method Name	Type	Arguments	Method Description
read(address)	Public:void	void	It reads the content of the given address in the Pic memory.
readAll()	Public:void	void	It reads the contents of the whole Pic memory.

Table 3.13: PICMemory Class

3.1.14 StimuliGenerator Class

This class is for input operations. It reads input from the input stimulus and sets the input file name and starts giving input during the run of the program.

Method Name	Type	Arguments	Method Description
readInputFromFile()	Public:void	void	It reads the stimulus from the file.
setFileName(fileName)	Public:void	void	It sets the fileName as the file name of the currently generated stimulus file.
run()	Public:void	void	It runs the simulator according to specified configurations by the user.

Table 3.14: StimuliGenerator Class

3.1.15 StimuliPipe2 Class

This class is for output stimuli. It has methods for reading and writing.

Method Name	Type	Arguments	Method Description
read()	Public:void	void	It reads the stimulus created by simulator.
write(stimulus)	Public:void	void	It writes the stimulus created by simulator.

Table 3.15: stimuliPipe2:StimuliPipe object

3.2 SimulatorGUI Class

This class is for redrawing the interface for simulation. When it is executed it refreshed the simulation.

Method Name	Type	Arguments	Method Description
redraw()	Public:void	void	It draws the simulation on the screen.

Table 3.16: SimulatorGUI Class

3.2.1 MemoryContentDisplay Class

This is for displaying the contents of the memory locations. It has a method for refreshing the situation of the memory locations.

3.2.2 WatchWindow Class

This class is for displaying the contents of the so-called watch variables register.

3.2.3 PICProgramMemory Class

This class is for displaying the contents of the Program Memory. It has a method for refreshing the situation of the Program Memory.

Method Name	Type	Arguments	Method Description
read()	Public:void	int offset	It shows the contents of the memory locations.

Table 3.17: MemoryContentDisplay Class

Method Name	Type	Arguments	Method Description
redraw()	Public:void	void	display the contents of the watched meory locations on the screen

Table 3.18: WatchWindow Class

3.2.4 SimulationConfiguration Class

This class is for configuration of the simulation. It has methods to get hex file to be simulated, to get the break point locations, to get the name of the input stimulus file name and output stimulus file name.

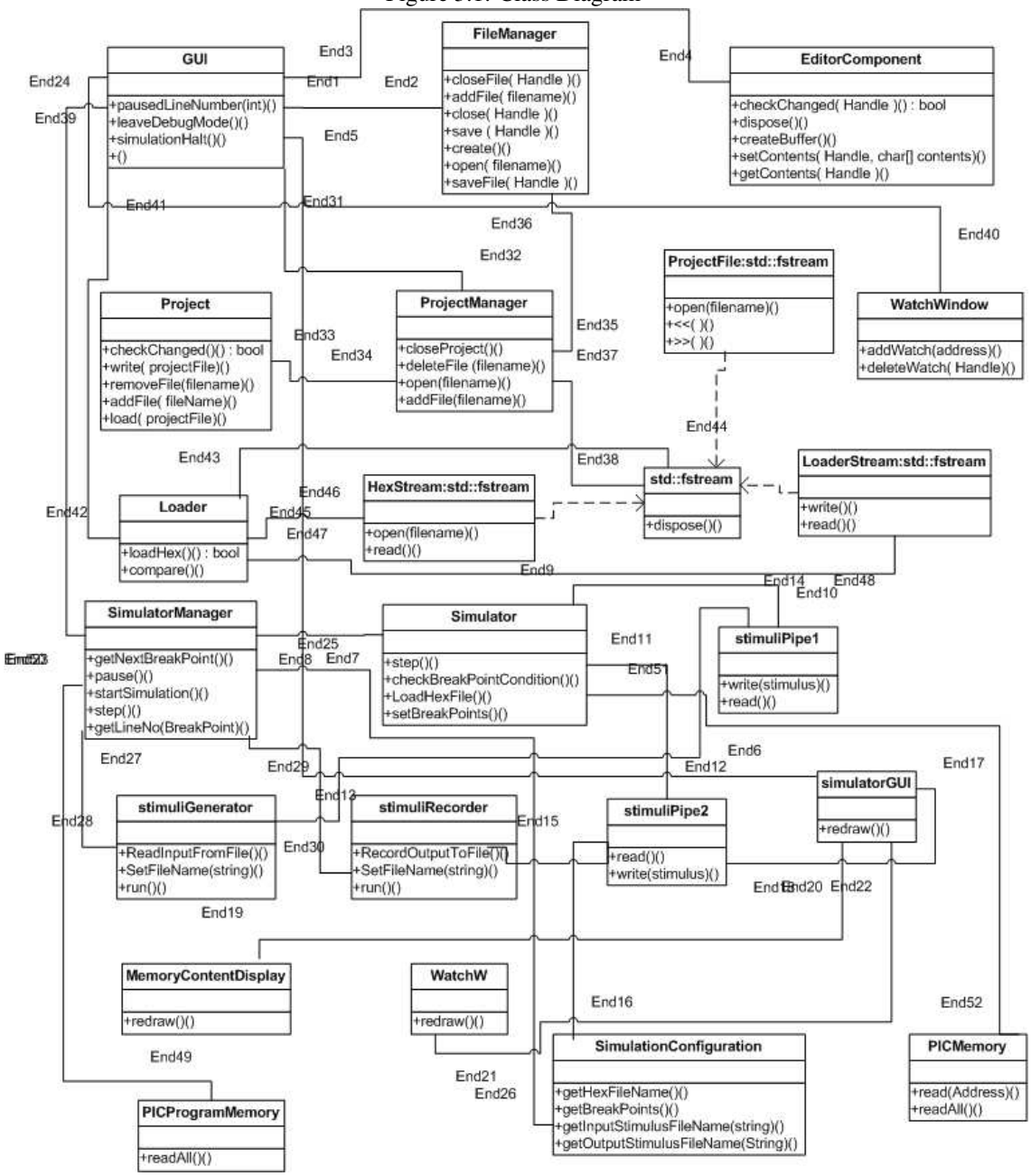
Method Name	Type	Arguments	Method Description
readAll()	Public:void	void	It reads and shows the contents of the program memory.

Table 3.19: PICProgramMemory Class

Method Name	Type	Arguments	Method Description
getHexFileName()	Public:void	void	It gets the file name which will be simulated.
getBreakpoints()	Public:void	void	It gets the breakpoints locations of the file that will be simulated.
getInputStimulusFileName(fileName)	Public:void	void	It sets the input stimulus file name as fileName.
getOutputStimulusFileName(fileName)	Public:void	void	It sets the output stimulus file name as fileName.

Table 3.20: SimulationConfiguration Class

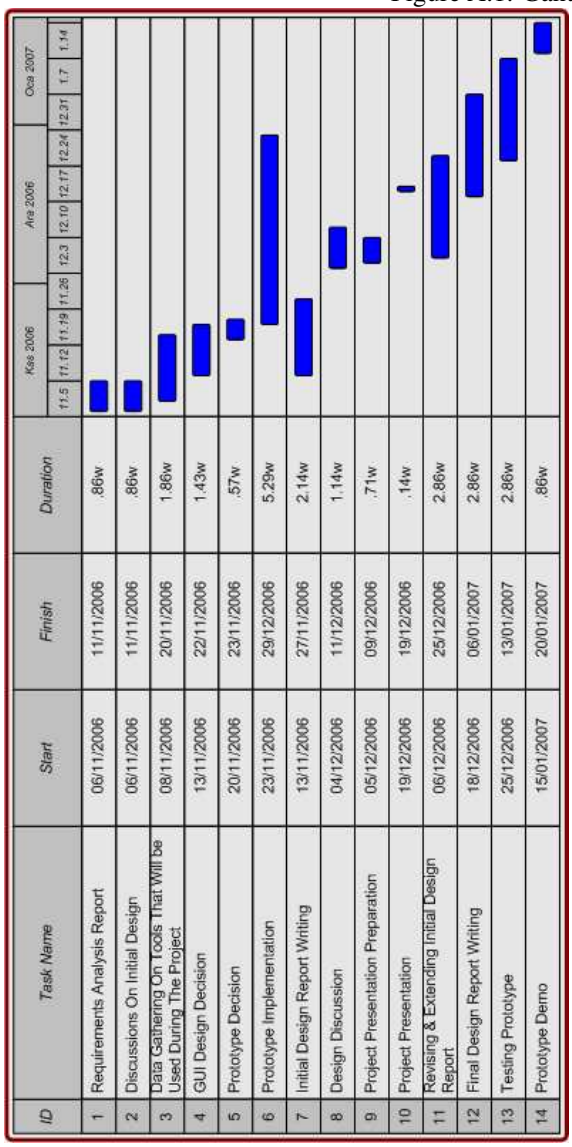
Figure 3.1: Class Diagram



Appendix A

GANTT CHART

Figure A.1: Gantt Chart Monthly



ID	Task Name	Start	Finish	Duration
1	Requirements Analysis Report	06/11/2006	11/11/2006	.86w
2	Discussions On Initial Design	06/11/2006	11/11/2006	.86w
3	Data Gathering On Tools That Will be Used During The Project	08/11/2006	20/11/2006	1.86w
4	GUI Design Decision	13/11/2006	22/11/2006	1.43w
5	Prototype Decision	20/11/2006	23/11/2006	.57w
6	Prototype Implementation	23/11/2006	29/12/2006	5.29w
7	Initial Design Report Writing	13/11/2006	27/11/2006	2.14w
8	Design Discussion	04/12/2006	11/12/2006	1.14w
9	Project Presentation Preparation	05/12/2006	09/12/2006	.71w
10	Project Presentation	19/12/2006	19/12/2006	.14w
11	Revising & Extending Initial Design Report	06/12/2006	25/12/2006	2.86w
12	Final Design Report Writing	18/12/2006	06/01/2007	2.86w
13	Testing Prototype	25/12/2006	13/01/2007	2.86w
14	Prototype Demo	15/01/2007	20/01/2007	.86w