Middle East Technical University
Department of Computer Engineering

CENG 491

Computer Engineering Design I

2006-2007

SimSys Corporation

Initial Design Report

PIDE

Emulator and Development Environment for

CEng Embedded System Card

04.12.2006

Table of Contents

1. Introduction

1.1 Purpose of the Document

This document is prepared to supply an initial design for the PIDE Project.

This report should be considered as an intermediate outcome of the design process. The work done and results are included in this document in a formal way. Since design process consists of modelling the system, the report contains diagrams and models of the current system. Design process is still going on.Hence,all the diagrams and models are subject to change.

The report consist of two parts. In the Modelling of the System part, static and dynamic components of the system are represented. In the System and Project Specifications part, standards related to project implementation and various system components are introduced.

1.2. Project Description

As the technology evolves, the embedded systems start to find wide area of usage. In most of the devices that people use daily, there exists a core logic which is mostly an embedded microcontroller or microprocessor with some external storage. Besides, those integrated devices also let the implementation and testing of various new controller ideas very easily. This popularity of Embedded Systems is a little overshadowed by the difficulty in developing embedded software due to the lack of a well fitted development environment and pre-testing it on a special independent system prepared just for testing purposes.

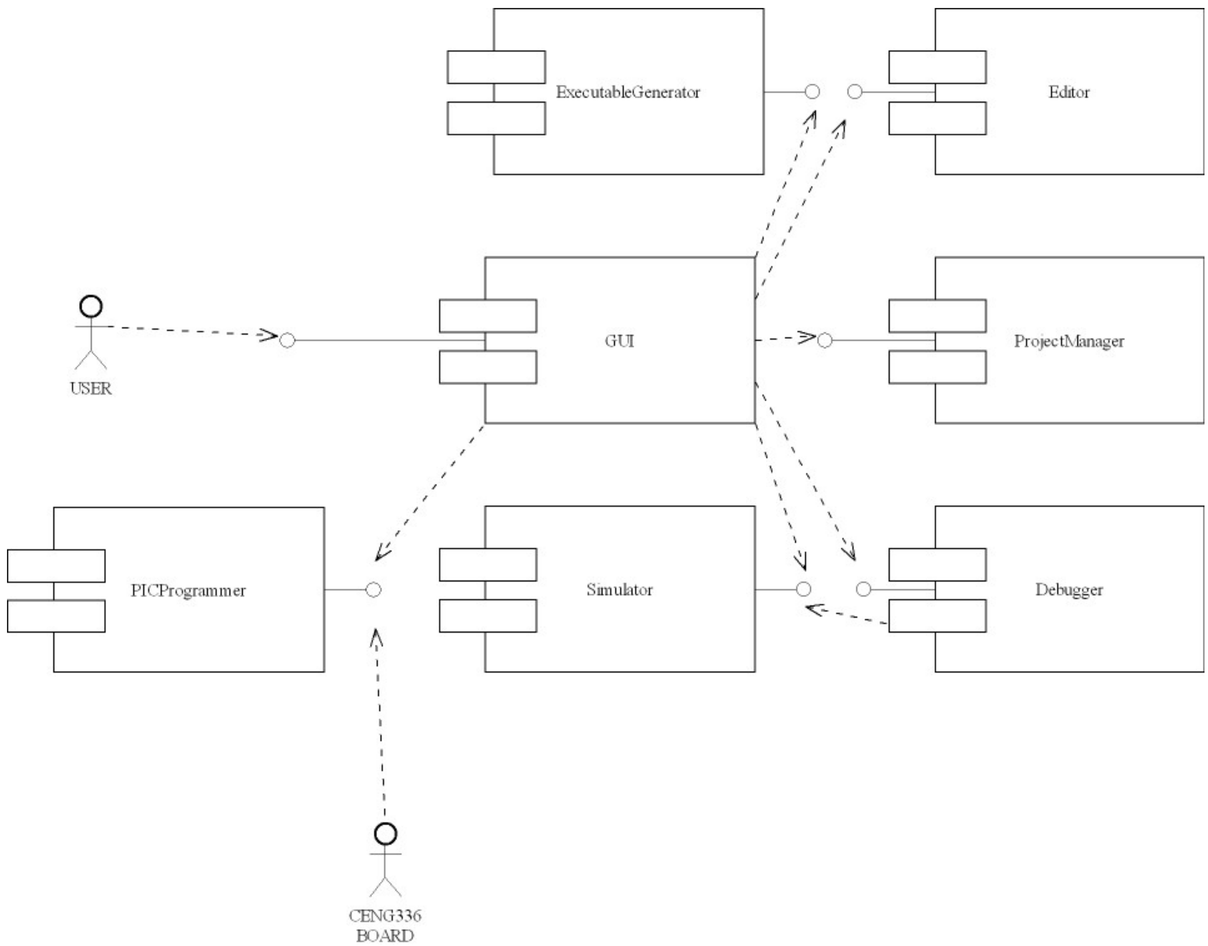
An example to the above discussion exists for the CEng336 Embedded Systems course. Among the course contents, development of embedded software and testing on a test board is of primary importance. However, obviously a standalone testing environment that will simulate exactly the same features with high accuracy would greatly simplify the testing procedure.

As a solution to the problem stated above, SimSys Corporation will develop an emulator and development environment for the card used in Ceng336 Embedded Systems course. Considering such a development and simulation environment, the system will support various types of microcontrollers, communicate through various interface standards such as parallel, serial or USB and accommodate some display interfaces such as LCD or LED driving structures. Users will have the chance of compiling their programs and they can test and debug it on the virtual card emulated by the software.

For such a development and simulation environment design project, the implementation areas are unlimited just as the fact that the implementation areas of the embedded systems are unlimited. As a result, such a system, which will simplify the development and testing process, will find great interest from the embedded systems developers. Together with the Ceng336 Card, this software will be useful for computer engineers, electrical engineers, high school students and everyone interested in PIC programming.

2. System Architecture

In the figure below, the major components of the PIDE is given.



3. Modeling

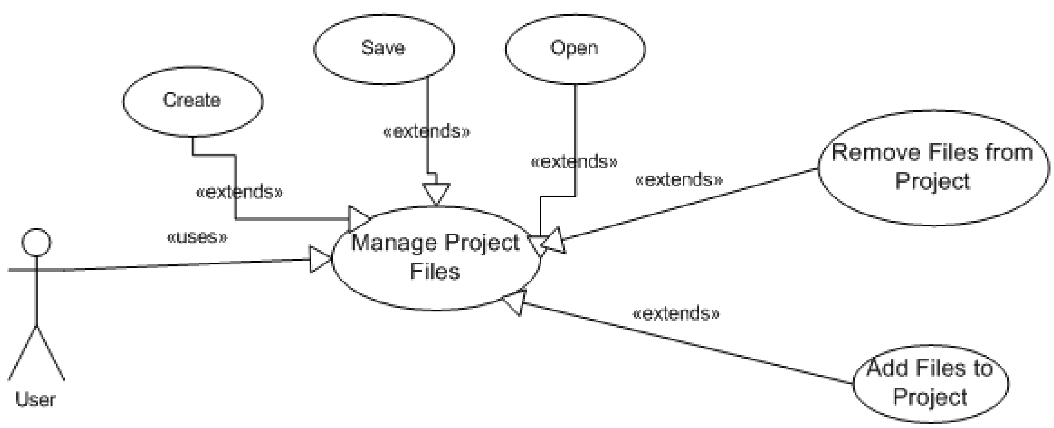
3.1. Scenario Based Model – Use Case Diagrams

The use cases of the system describe the interaction between the system and the user from the user's point of view. This schematic is important to define the capabilities that are given to the user and his/her possible choices. There is no timing relationship existing in this diagram; however that information is given in the sequence diagrams, since these use cases are only to present the alternative paths that can be followed.

3.1.1. Manage Project

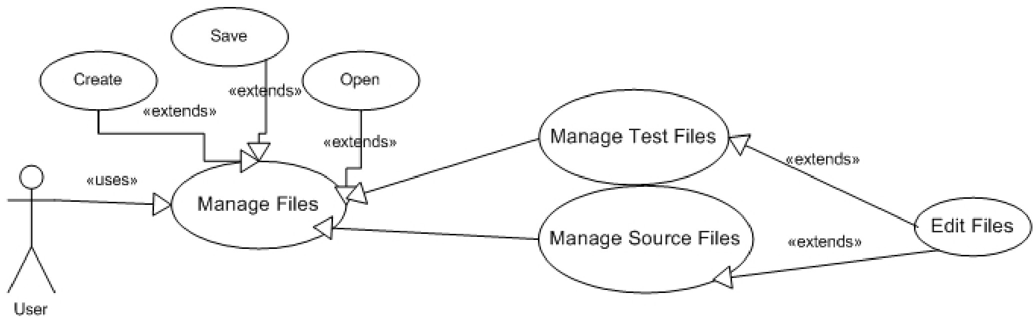
Managing a project is in fact handling of files within a project. Creation of new files, adding existing files to the project, removing files from the project are the possible tasks that can be performed in this use case. The files that are mentioned here may be of various types. The alternatives for file types are ASM++ source files, ASM source files and test bench files.

MANAGE PROJECT FILES:



3.1.2. Manage Files

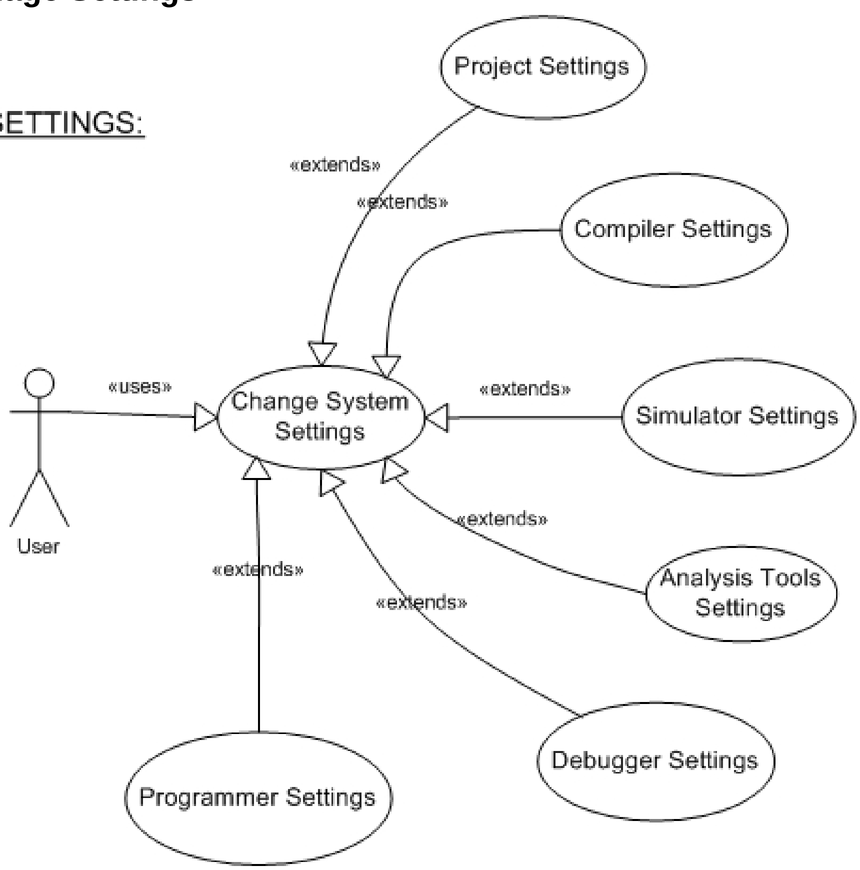
MANAGE FILES:



The user may select to manage the files using PIDE. Here, files may be created, saved, opened. These files are the source files and test bench files. The source files are the ASM++ files or ASM files. The test bench file contains the input timing information for the peripherals.

3.1.3. Manage Settings

CHANGE SETTINGS:



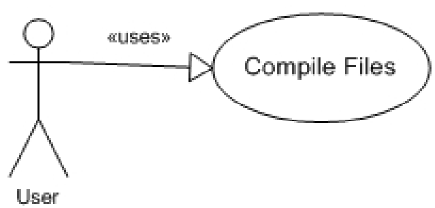
This use case defines the interaction of the user with the system to manage the settings of various internal modules of the software. Here, by means of graphical dialog windows, the user will be able to modify the system settings. This use case is in fact composed of a number of independent use cases. These are setting the project settings, compiler settings, simulator settings, debugger settings, analysis settings and finally the programmer settings. The first ones are self explanatory; however the last two require some elaboration.

Analysis settings are the specification of signals that are to be saved for later investigation. Here, some probes are inserted to the system, where the logic levels or voltages on those nodes are saved. Those saved waveform graphics can later be viewed via the analysis tool.

Programmer settings are about the programming interface of the board. Here, the parallel port selection can be performed and other choices about device programming can be made.

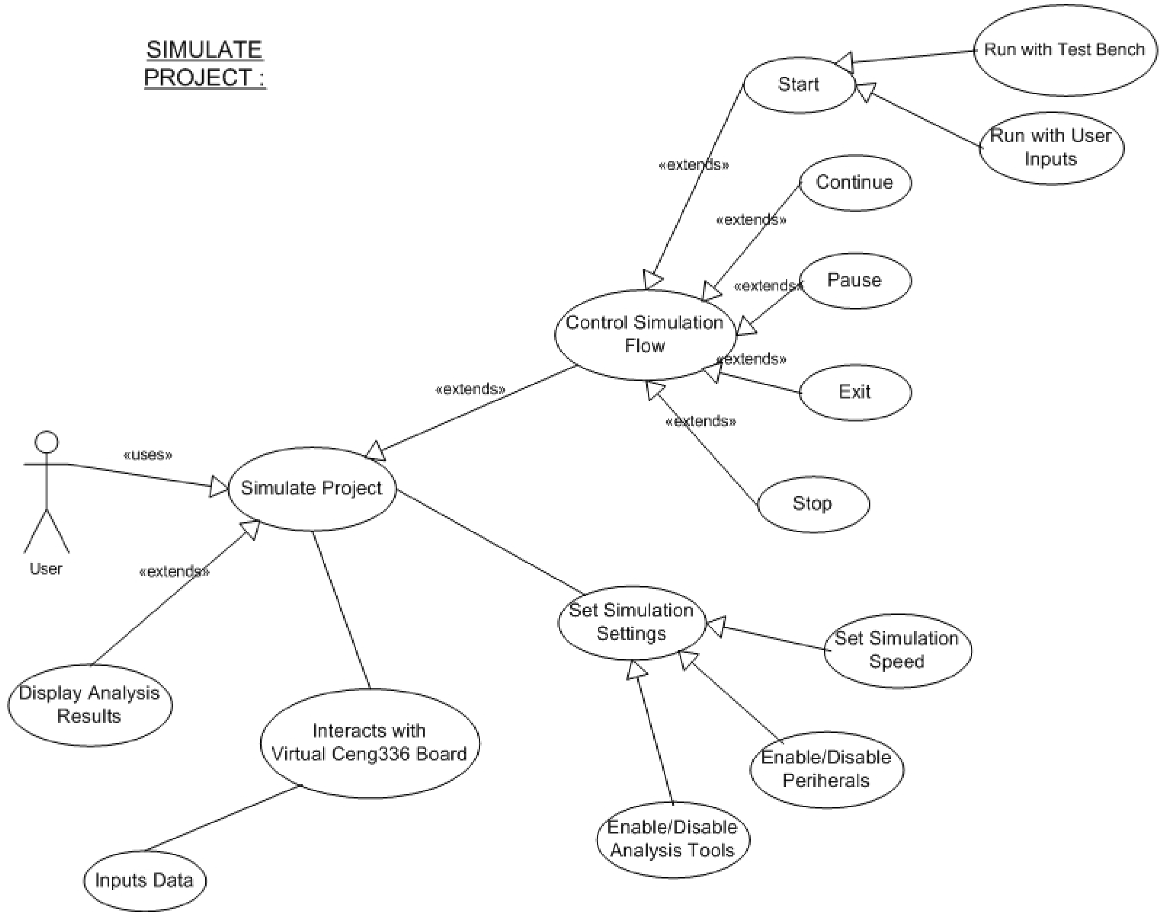
3.1.4. Compile Project

COMPILER:



The use case with the compile system is very straightforward. The user just requests a compile operation from the system. All syntax checking, parsing, linking and conversions are performed transparently to the user. The results are displayed in the output pane of the user interface.

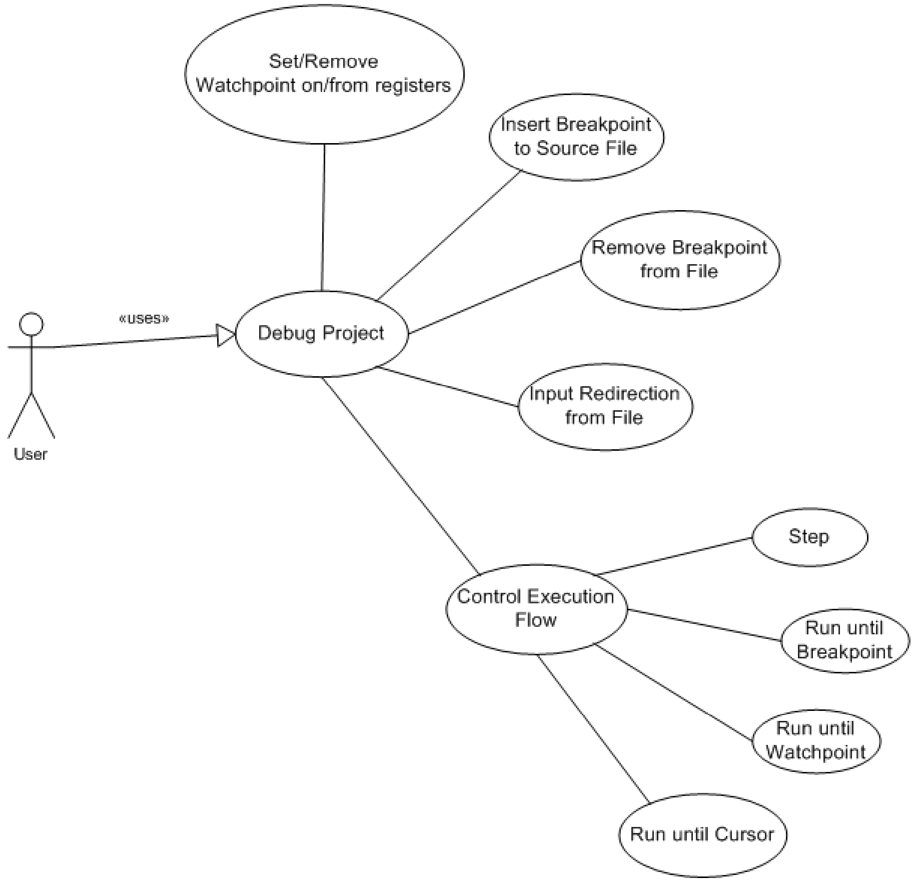
3.1.5. Simulate Project



In the simulation use case, the user will ask the system to run according to the specified inputs. The inputs may be provided by the user either real time by means of the graphical user interface which is exactly the same as the layout of the board, or some files that specifies some sequence of data to the input devices. These special files are called test bench files and have their special file format.

Simulation system has some special features. One of them is the enable/disable mechanism of the peripherals on the evaluation board. Another one is the selectable run speed. This feature will make the user much more comfortable in simulation of high frequency systems. For instance, in order to observe a signal toggling at 100 KHz, the system may be configured to run in 5us steps.

3.1.6. Debug Project

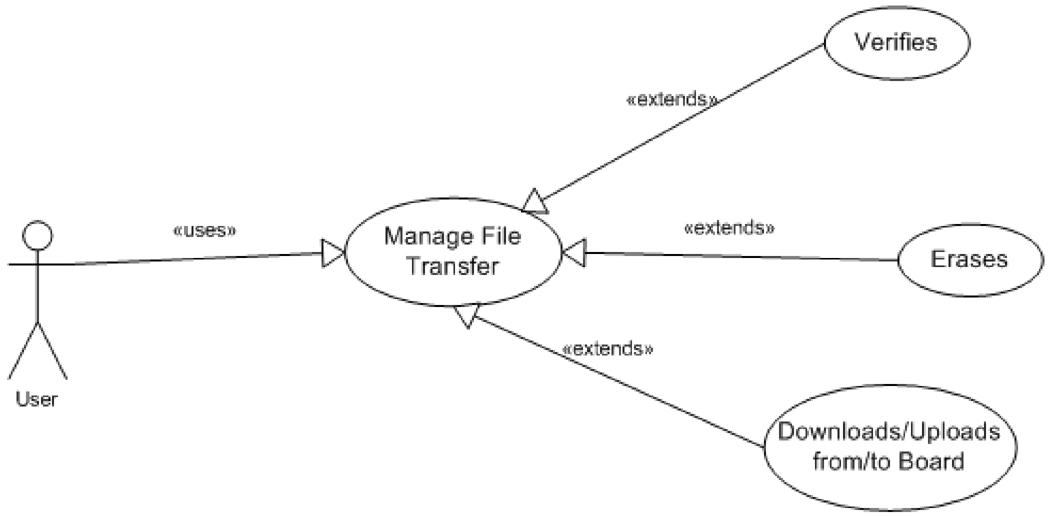


Debugging a project is to concentrate on the flow of the program on some specific parts of the source code. Debugging a project internally requires the project to be compiled and if current system is in not compiled state, then automatically the compile routine is invoked. Critical concepts for the debugger are the breakpoints and watch points.

Breakpoints are identifiers on some source code lines that state that the execution of the program will continue until that point and will halt there. The internal state of the system will be completely visible to the user, together with the contents of the registers. The execution flow will continue with some special events from the user such as a "step" command.

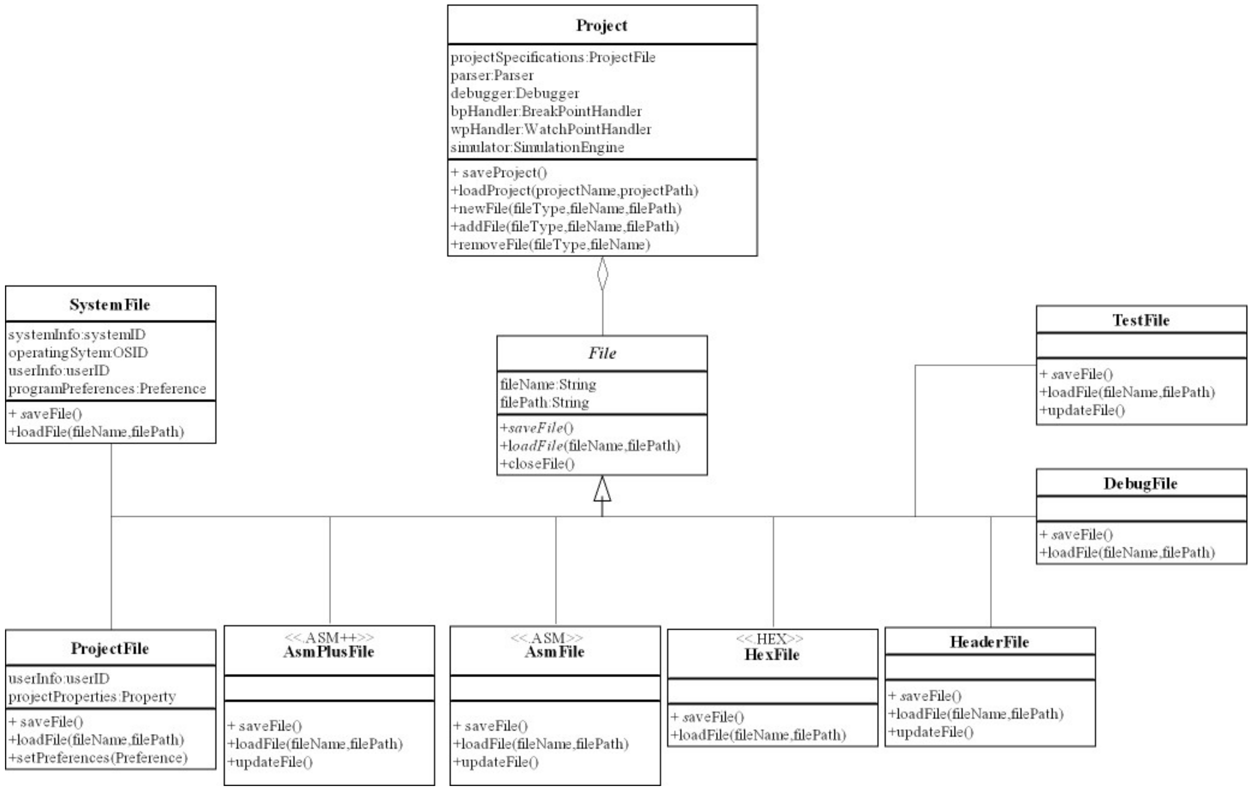
Watch points are identifiers attached to registers. These watch points are triggered when the value in the register is modified. The execution of the program halts at this point. Resuming is based on the same procedure as the one in breakpoints.

3.1.7. Manage File Transfer



Once the simulation is performed and the required results observed in the system, the user will upload the hex file to the microcontroller on the board to verify the operation physically. The user may also request to see the source of the program in currently residing in the microcontroller or may request a verification to check whether the uploaded program is consistent with the one in hand. The user may also want to clear the contents of the memory in the controller to be on the safe side and to start everything from scratch.

3.2. Class diagrams



Project

	Attribute Name	Type		Description
Attributes	projectName	string		The name of the project.
	projectPath	string		The path of the project on the disk.
	specifications	ProjectFile		The specifications of the project.
	compiler	Compiler		The compiler module.
	debugger	Debugger		The debugger module.
	bpHandler	BreakPointHandler		The breakpoint handler.
	wpHandler	WatchPointHandler		The watchpoint handler.
	simulator	SimulationEngine		The simulator module.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	saveProject()	void	void	Saves the project.
	loadProject()	void	projectName, projectPath	Loads the project.
	newFile()	void	fileType,	Creates a new file and adds it to

			fileName, filePath	the project.
	addFile()	void	fileType, fileName, filePath	Adds an existing file to the project.
	removeFile()	void	fileType, fileName,	Removes a file from the project.

File

Attributes	**Attribute Name**	**Type**		**Description**
	fileType	int		The type of the file, i.e. asm, hex, test, etc.
	fileName	string		The name of the file.
	filePath	string		The path of the file on the disk.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.
	close()	void	void	Closes the file.

SystemFile :: File

Attributes	**Attribute Name**	**Type**		**Description**
	systemInfo	systemID		Information about the system.
	operatingSystem	OSID		The type of the operating system.
	userInfo	userID		Information about the user.
	preferences	Preference		The preferences of the user.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.

ProjectFile :: File

Attributes	**Attribute Name**	**Type**		**Description**
	userInfo	userID		Information about the user.
	properties	Property		The properties of the project.
Methods	**Method Name**	**Return**	**Arguments**	**Description**

	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.

AsmPlusFile :: File

	Method Name	Return	Arguments	Description
Methods	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.
	update()	void	void	Updates the file with the current changes.

AsmFile :: File

	Method Name	Return	Arguments	Description
Methods	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.
	update()	void	void	Updates the file with the current changes.

HexFile :: File

	Method Name	Return	Arguments	Description
Methods	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.

HeaderFile :: File

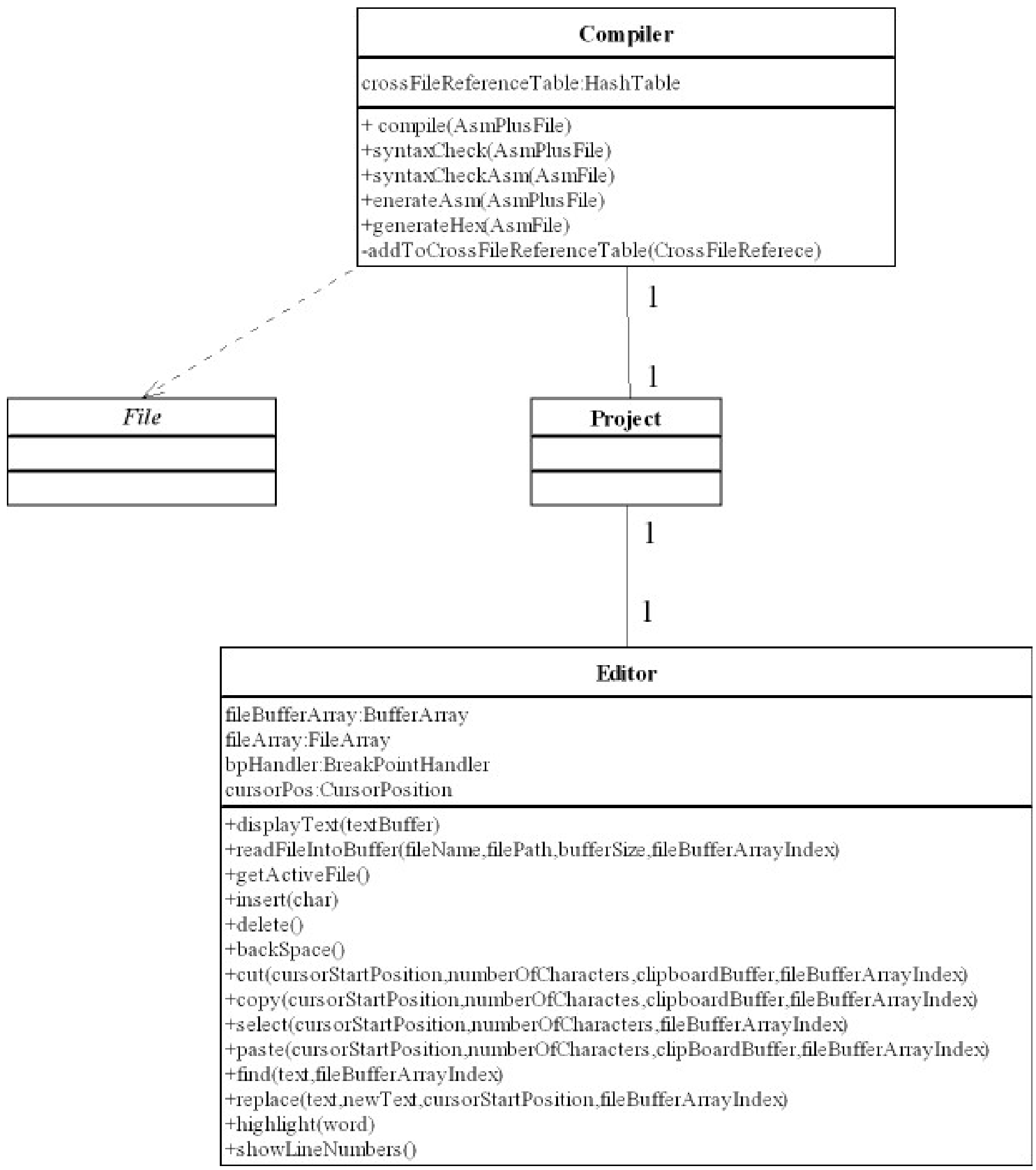
	Method Name	Return	Arguments	Description
Methods	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.
	update()	void	void	Updates the file with the current changes.

DebugFile :: File

	Method Name	Return	Arguments	Description
Methods	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.

TestFile :: File

	Method Name	Return	Arguments	Description
Methods	save()	void	void	Saves the file.
	load()	void	fileName, filePath	Loads the file.
	update()	void	void	Updates the file with the current changes.



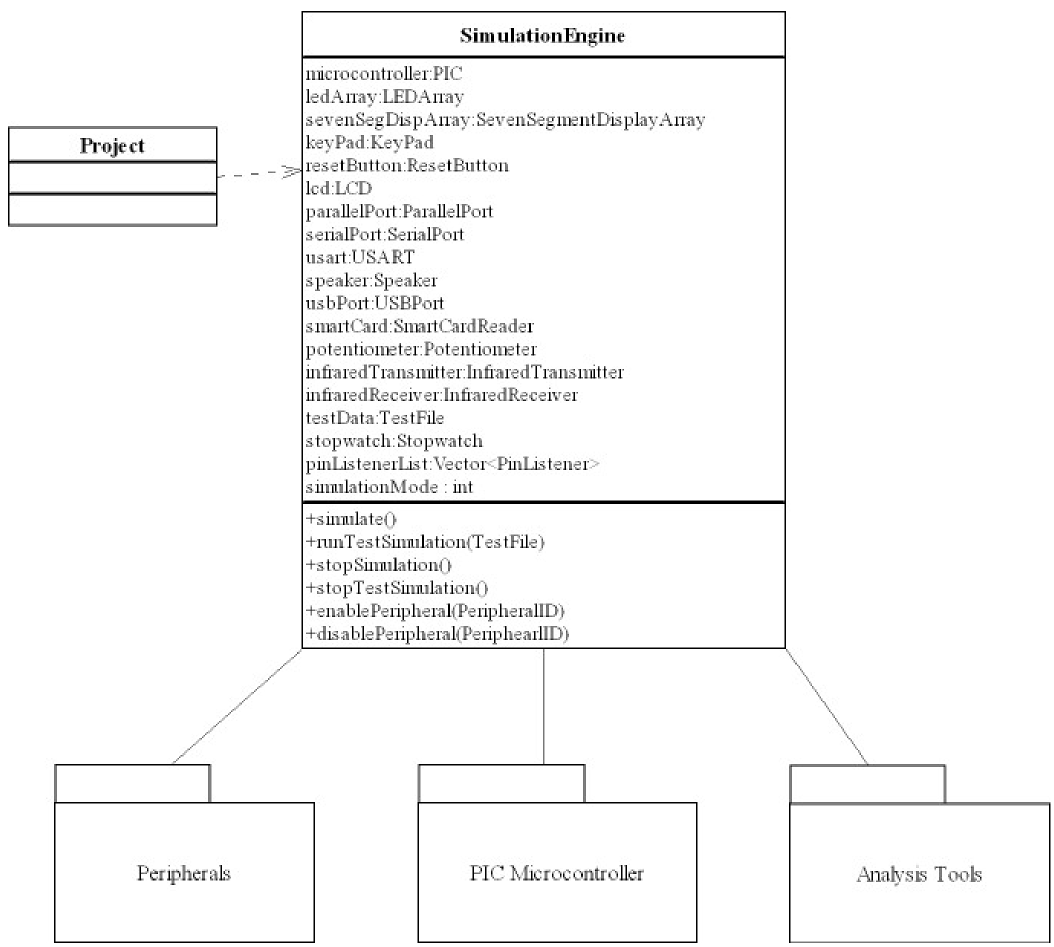
Editor

	Attribute Name	**Type**		**Description**
Attributes	fileBufferArray	BufferArray		The contents of the currently opened files.
	fileArray	fileArray		The files those are currently open.
	bpHandler	BreakPointHandler		The breakpoint handler.
	cursorPos	CursorPosition		Current position of the cursor
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	displayText()	void	textBuffer	Displays the text in the buffer.
	readFileIntoBuf()	void	fileName, filePath, bufferSize, fileBufferArrayIndex	Reads the file into the specified buffer.
	getActiveFile()	fileName	void	Returns the name of the active file.
	insert()	void	char	Inserts the given char to the Buffer.
	delete()	void	void	Deletes the selected items from the Buffer.
	backspace()	void	void	Deletes the last character in the Buffer.
	select()	void	cursorStartPosition, numOfCharacters, fileBufferArrayIndex	Selects *numOfCharacters* characters starting from the *cursorStartPosition*.
	cut()	void	cursorStartPosition, numOfCharacters, clipboardBuffer, fileBufferArrayIndex	Puts the selected item into the clipboard buffer.
	copy()	void	cursorStartPosition, numOfCharacters, clipboardBuffer, fileBufferArrayIndex	Copies the selected item into the clipboard buffer.
	paste()	void	cursorStartPosition, numberOfCharacters, clipBoardBuffer, fileBufferArrayIndex	Pastes the last item in the clipboard buffer.
	find()	void	text, cursorStartPosition, fileBufferArrayIndex	Find *text* in the file.
	replace()	void	text, newText, cursorStartPosition, fileBufferArrayIndex	Find *text* in the file and replace with *newText*.
	highlight()	void	word	Highlights the *word*.

	showLineNums()	void	void	Shows the line numbers.

Compiler

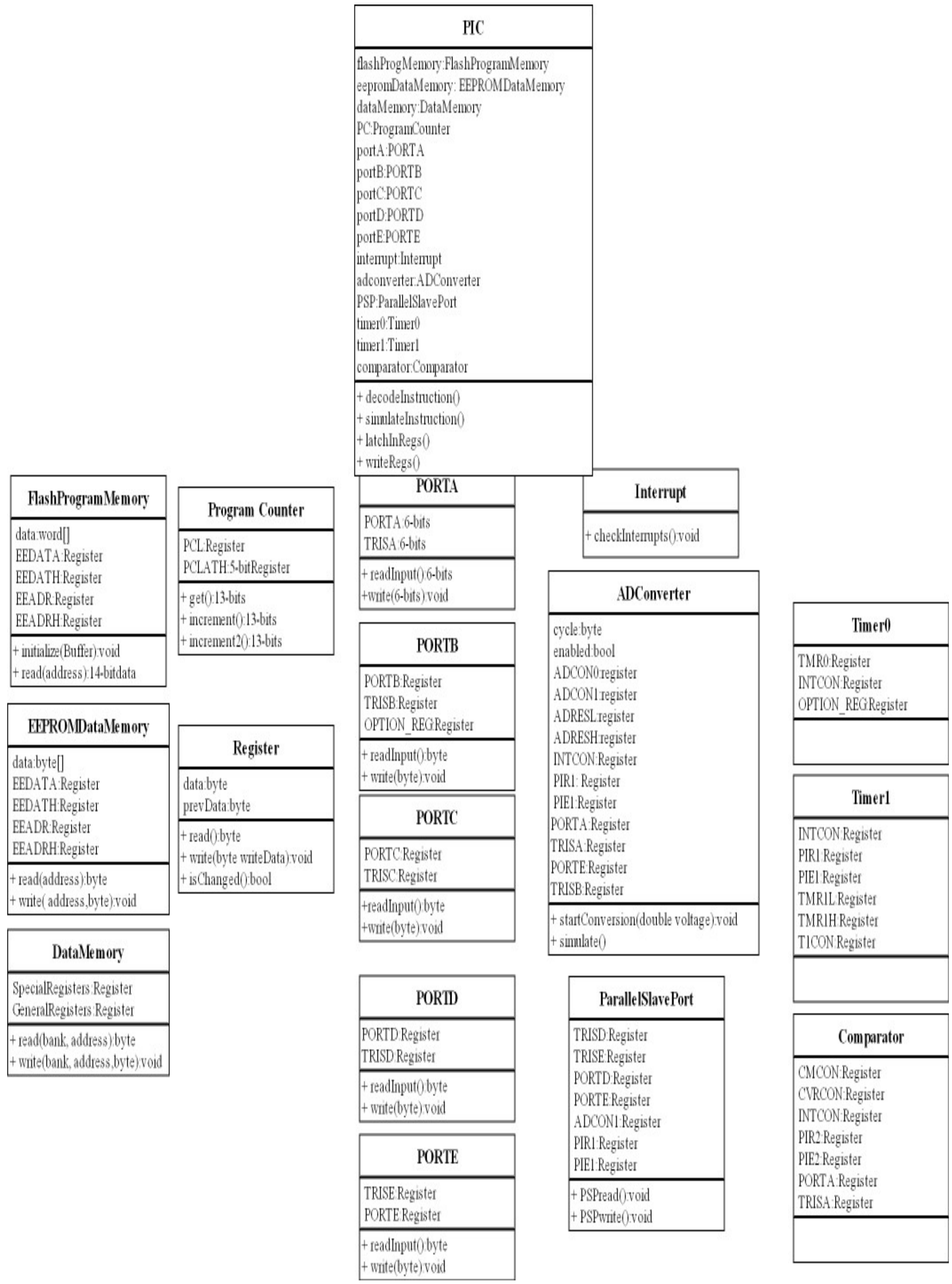
Attributes	**Attribute Name**	**Type**		**Description**
	crossFileReference-Table	Hash Table		The mapping between the source file and the hex file.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	compile()	void	AsmPlusFile	Starts the compilation process.
	syntaxCheck()	void	AsmPlusFile	Checks the syntax of the AsmPlusFile.
	syntaxCheckAsm()	void	AsmFile	Checks the syntax of the AsmFile.
	generateAsm()	void	AsmPlusFile	Generates an AsmFile from the AsmPlusFile.
	generateHex()	void	AsmFile	Generates a HexFile from the AsmFile.
	addToCrossFile-ReferenceTable()	void	CrossFileReference	Adds the CrossFileReference entry to the CrossFileReferenceTable.



Simulation Engine

Attributes	**Attribute Name**	**Type**		**Description**
	microcontroller	PIC		PIC microcontroller.
	ledArray	LEDArray		LED array on the board.
	sevenSegDispArray	SevenSegmentDisplayArray		7segment display array on the board.
	keyPad	KeyPad		Keypad on the board.
	resetButton	ResetButton		Reset button on the board.
	lcd	LCD		LCD display on the board.
	parallelPort	ParallelPort		Parallel port on the board.
	serialPort	SerialPort		Serial port on the board.
	usart	USART		USART module on the board.
	speaker	Speaker		Speaker on the board.
	usbPort	USBPort		USB port on the board.
	smartCard	SmartCard		Smart card reader on the board.
	potentiometer	Potentiometer		The analog input POT on the board.
	infraredTransmitter	InfraredTransmitter		Infrared-transmitter on the board.
	infraredReceiver	InfraredReceiver		Infrared-receiver on the board.
	testData	TestFile		Test bench data for simulation.
	stopwatch	Stopwatch		Stopwatch to keep the time during simulation.
	pinListenerList	Vector<PinListener>		Pin listener to keep the logic values of the pins.
	simulationMode	int		The mode of the simulation.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	simulate()	void	void	Makes the simulation.
	runTestSimulation()	void	TestFile	Makes the test bench simulation.
	stopSimulation()	void	void	Stop the simulation.
	stopTestSimulation()	void	void	Stop the test bench simulation.
	enablePeripheral()	void	PeripheralID	Enables the peripheral in the simulation

	disablePeripheral()	void	PeripheralID	Disables the peripheral in the simulation



PIC

	Attribute Name	Type		Description
Attributes	flashProgMemory	FlashProgramMemory		Flash program memory
	eepromDataMemory	EEPROMDataMemory		EEPROM data memory
	dataMemory	DataMemory		Data memory
	pc	ProgramCounter		Program Counter
	portA	PORTA		PORT A of the PIC
	portB	PORTB		PORT B of the PIC
	portC	PORTC		PORT C of the PIC
	portD	PORTD		PORT D of the PIC
	portE	PORTE		PORT E of the PIC
	interrupt	Interrupt		Interrupt module of the PIC
	adConverter	ADConverter		Analog-to-Digital Converter
	psp	ParallelSlavePort		Parallel Slave Port
	timer0	Timer0		Timer 0 of the PIC
	timer1	Timer1		Timer 1 of the PIC
	comparator	Comparator		Comparator of the PIC
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	decodeInstruction()	void	void	Decodes the next instruction.
	simulateInstruction()	void	void	Simulates the next instruction.
	latchInRegs()	void	void	Latch in the register values before the execution of a step.
	writeRegs()	void	void	Write the updated values of the registers after the execution of a step.

FlashProgramMemory

	Attribute Name	Type	Description
Attributes	data	word[]	The content array of the memory.
	EEDATA	Register	EEDATA register
	EEDATH	Register	EEDATH register
	EEADR	Register	EEADR register

	EEADRH	Register		EEADRH register
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	initialize()	void	Buffer	Initializes the memory.
	read()	14bit-data	Address	Read the data at the *Adress*.

EEPROMDataMemory

Attributes	**Attribute Name**	**Type**		**Description**
	data	byte[]		The content array of the memory.
	EEDATA	Register		EEDATA register
	EEDATH	Register		EEDATH register
	EEADR	Register		EEADR register
	EEADRH	Register		EEADRH register
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	read()	byte	Address	Reads the byte at the *Adress*.
	write()	void	Address, byte	Writes the *byte* to the *Adress*.

DataMemory

Attributes	**Attribute Name**	**Type**		**Description**
	specialRegisters	Register		The special registers in Data Memory.
	generalRegisters	Register		The general registers in Data Memory.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	read()	byte	Bank, Address	Reads the byte at the *Adress* on *Bank*.
	write()	void	Bank, Address, byte	Writes the *byte* to the *Adress* on *Bank*.

ProgramCounter

Attributes	**Attribute Name**	**Type**	**Description**
	PCL	Register	PCL Register in the PIC
	PCLATH	5bit-Register	PCLATH Register in the PIC

Methods	**Method Name**	**Return**	**Arguments**	**Description**
	get()	13bit	void	Gets the current value of the program counter.
	increment()	13bit	void	Increments the value of the program counter.
	increment2()	13bit	void	Increments the value of the program counter by 2.

Register

Attributes	**Attribute Name**	**Type**		**Description**
	data	byte		The content of the register
	prevData	byte		Previous content of the register
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	read()	byte	void	Reads the data in the register.
	write()	void	byte	Writes the *byte* into the register.
	isChanged()	bool	void	Returns true if the content of the register has been changed, returns false otherwise.

PORTA

Attributes	**Attribute Name**	**Type**		**Description**
	PORTA	6-bit data		The content of the Port register
	TRISA	6-bit data		The data direction Register
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	readInput()	6-bit data	void	Reads the input data in the port.
	write()	void	6-bit data	Writes the *6-bit data* into the port.

PORTB

Attributes	**Attribute Name**	**Type**	**Description**
	PORTB	Register	The content of the Port register
	TRISB	Register	The data direction Register

Methods	Method Name	Return	Arguments	Description
	readInput()	byte	void	Reads the input data in the port.
	write()	void	byte	Writes the *byte* into the port.

PORTC

Attributes	Attribute Name	Type		Description
	PORTC	Register		The content of the Port register
	TRISC	Register		The data direction Register
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	readInput()	byte	void	Reads the input data in the port.
	write()	void	byte	Writes the *byte* into the port.

PORTD

Attributes	Attribute Name	Type		Description
	PORTD	Register		The content of the Port register
	TRISD	Register		The data direction Register
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	readInput()	byte	void	Reads the input data in the port.
	write()	void	byte	Writes the *byte* into the port.

PORTE

Attributes	Attribute Name	Type		Description
	PORTE	Register		The content of the Port register
	TRISE	Register		The data direction Register
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	readInput()	byte	void	Reads the input data in the port.
	write()	void	byte	Writes the *byte* into the port.

ADConverter

	Attribute Name	Type		Description
Attributes	cycle	byte		AD conversion cycle
	enabled	bool		If AD conversion is enabled
	ADCON0	Register		ADCON0 Register
	ADCON1	Register		ADCON1 Register
	ADRESL	Register		ADRESL Register
	ADRESH	Register		ADRESH Register
	INTCON	Register		INTCON Register
	PIR1	Register		PIR1 Register
	PIE1	Register		PIE1 Register
	PORTA	Register		Local copy of PORTA
	PORTE	Register		Local copy of PORTE
	TRISA	Register		Local copy of TRISA
	TRISE	Register		Local copy of TRISE
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	startConversion()	void	double	Starts the AD conversion of the given analog voltage.
	simulate()	void	void	Simulates the AD conversion.

Interrupt

Methods	Method Name	Return	Arguments	Description
	checkInterrupts()	void	void	Checks if there are interrupts.

ParallelSlavePort

Attributes	Attribute Name	Type	Description
	PORTD	Register	Local copy of PORTD
	PORTE	Register	Local copy of PORTE
	TRISD	Register	Local copy of TRISD
	TRISE	Register	Local copy of TRISE

	ADCON1	Register		Local copy of ADCON1
	PIR1	Register		Local copy of PIR1
	PIE1	Register		Local copy of PIE1
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	pspRead()	void	void	Read the data.
	pspWrite()	void	void	Write the data.

Timer0

Attributes	**Attribute Name**	**Type**	**Description**
	TIMER0	Register	Local copy of TIMER0
	INTCON	Register	Local copy of INTCON
	OPTION_REG	Register	Local copy of OPTION_REG

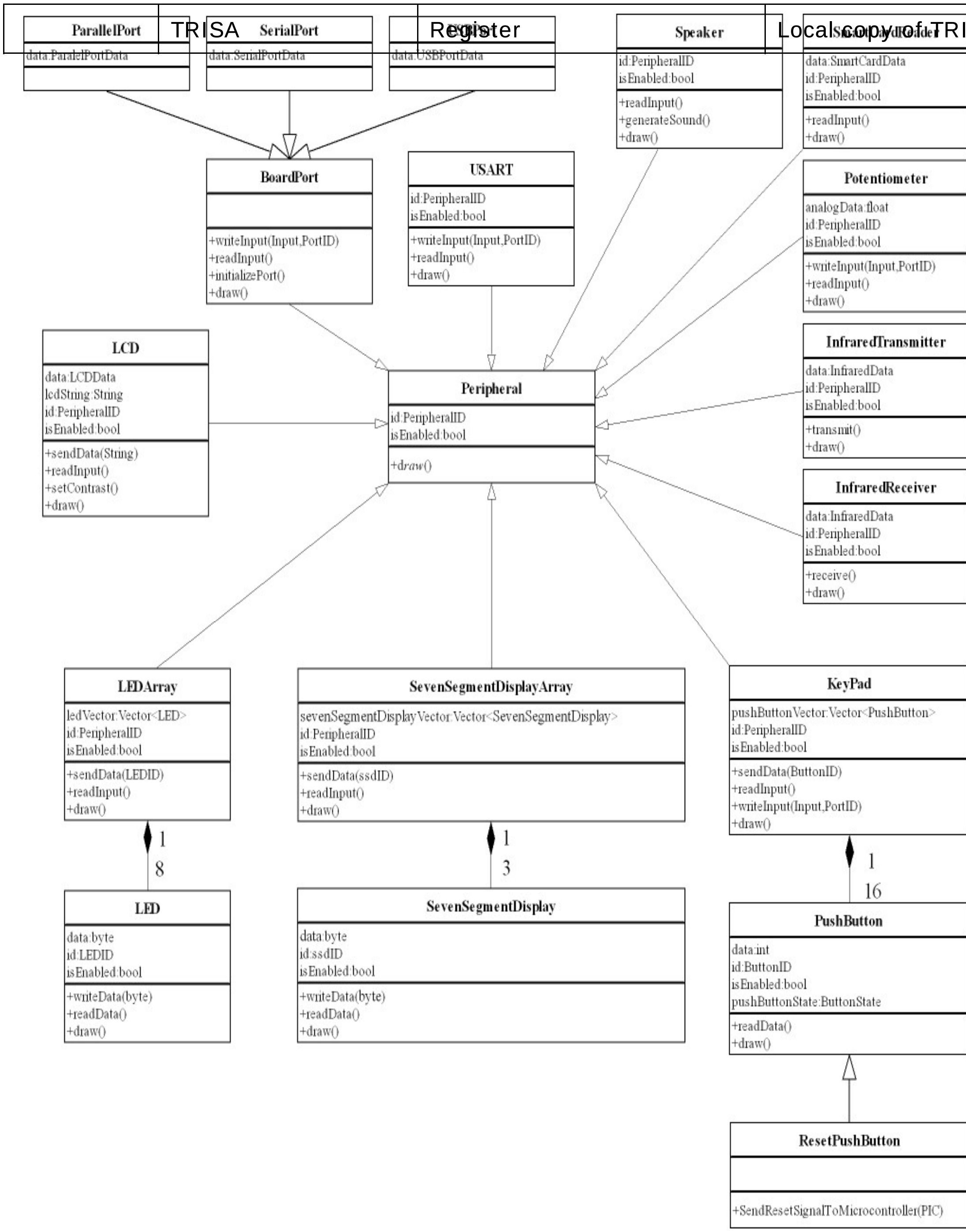
Timer1

Attributes	**Attribute Name**	**Type**	**Description**
	INTCON	Register	Local copy of INTCON
	PIR1	Register	Local copy of PIR1
	PIE1	Register	Local copy of PIE1
	TMR1L	Register	Local copy of TMR1L
	TMR1H	Register	Local copy of TMR1H
	T1CON	Register	Local copy of T1CON

Comparator

Attributes	**Attribute Name**	**Type**	**Description**
	CMCON	Register	Local copy of CMCON
	CVRCON	Register	Local copy of CVRCON
	INTCON	Register	Local copy of INTCON
	PIR2	Register	Local copy of PIR2
	PIE2	Register	Local copy of PIE2
	PORTA	Register	Local copy of PORTA

	TRISA	Register	Local copy of TRISA



Peripheral

Attributes	Attribute Name	Type	Description
	id	PeripheralID	ID of the peripheral

	isEnabled	bool		if the peripheral is enabled
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	draw()	void	void	Draws the peripheral

LEDArray :: Peripheral

Attributes	**Attribute Name**	**Type**		**Description**
	ledVector	Vector<LED>		The vector of 8 LEDs
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	sendData()	void	ledID	Sends data to the LED with ledID.
	readInput()	void	void	Reads the input.
	draw()	void	void	Draws this peripheral.

LED

Attributes	**Attribute Name**	**Type**		**Description**
	id	ledID		The ID of this LED
	ledData	byte		The data of this LED
	isEnabled	bool		if this LED is enabled
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	writeData()	void	byte	Writes the data to this LED.
	readData()	byte	void	Reads the input.
	draw()	void	void	Draws this LED.

SevenSegmentDisplayArray :: Peripheral

Attributes	**Attribute Name**	**Type**		**Description**
	sevenSegment-DisplayVector	Vector<SevenSegmentDisplay>		The vector of 3 seven segment displays.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	sendData()	void	ssdID	Sends data to the SSD with ssdID.
	readInput()	void	void	Reads the input.

	draw()	void	void	Draws this peripheral.

SevenSegmentDisplay

Attributes	**Attribute Name**	**Type**		**Description**
	id	ssdID		The ID of this SSD.
	ssdData	byte		The data of this SSD.
	isEnabled	bool		if this SSD is enabled
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	writeData()	void	byte	Writes the data to this SSD.
	readData()	byte	void	Reads the input.
	draw()	void	void	Draws this SSD.

KeyPad :: Peripheral

Attributes	**Attribute Name**	**Type**		**Description**
	pushButtonVector	Vector<PushButton>		The vector of 16 push buttons.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	sendData()	void	buttonID	Sends data to the push button with buttonID.
	readInput()	void	void	Reads the input.
	writeInput()	void	Data, portID	Sends the input *data* to the port with *portID.*
	draw()	void	void	Draws this peripheral.

PushButton

Attributes	**Attribute Name**	**Type**		**Description**
	id	buttonID		The ID of this push button.
	buttonData	int		The data of this button.
	isEnabled	bool		if this button is enabled
	state	int		The state of this button
Methods	**Method Name**	**Return**	**Arguments**	**Description**

	readData()	int	void	Reads the input.
	draw()	void	void	Draws this push button.

ResetButton :: PushButton

Methods	Method Name	Return	Arguments	Description
	sendResetSignalToPIC()	void	void	Sends RESET signal to the PIC.

LCD :: Peripheral

Attributes	Attribute Name	Type		Description
	data	LCDData		The data of the LCD.
	lcdString	string		The string on the LCD.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	sendData()	void	string	Sends data to the LCD.
	readInput()	void	void	Reads the input.
	setContrast()	void	float	Changes the contrast of the LCD to the given value.
	draw()	void	void	Draws this peripheral.

BoardPort :: Peripheral

Methods	Method Name	Return	Arguments	Description
	writeInput()	void	Data	Writes data to the Port.
	readInput()	void	void	Reads the input.
	initialize()	void	void	Initializes the Port.

ParallelPort :: BoardPort

Attributes	Attribute Name	Type		Description
	data	ParallelPortData		The data of the port.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	draw()	void	void	Draws this peripheral.

SerialPort :: BoardPort

Attributes	**Attribute Name**	**Type**		**Description**
	data	SerialPortData		The data of the port.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	draw()	void	void	Draws this peripheral.

USBPort :: BoardPort

Attributes	**Attribute Name**	**Type**		**Description**
	data	USBPortData		The data of the port.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	draw()	void	void	Draws this peripheral.

USART :: Peripheral

Methods	**Method Name**	**Return**	**Arguments**	**Description**
	writeInput()	void	void	Writes data.
	readInput()	void	void	Reads the input.
	draw()	void	void	Draws this peripheral.

Speaker :: Peripheral

Methods	**Method Name**	**Return**	**Arguments**	**Description**
	readInput()	void	void	Reads the input.
	generateSound()	void	Data	Generates sound according to the given input.
	draw()	void	void	Draws this peripheral.

SmartCardReader :: Peripheral

Attributes	**Attribute Name**	**Type**		**Description**
	data	SmartCardData		The data of the smart card.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	readInput()	void	void	Reads the input.
	draw()	void	void	Draws this peripheral.

Potentiometer :: Peripheral

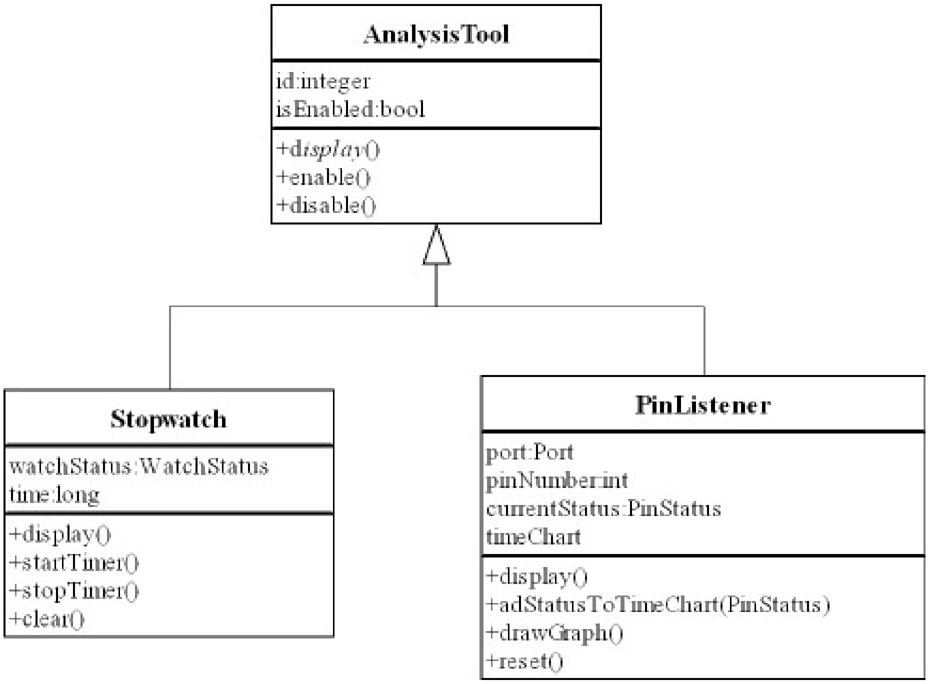
Attributes	**Attribute Name**	**Type**		**Description**
	analogData	float		The analog voltage value of the potentiometer.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	writeInput()	void	Data, PortID	Writes data to the Port with PortID.
	readInput()	float	void	Reads the input.
	draw()	void	void	Draws this peripheral.

InfraredTransmitter :: Peripheral

Attributes	**Attribute Name**	**Type**		**Description**
	data	InfraredData		The data of the infrared transmitter.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	transmit()	void	void	Transmits the data.
	draw()	void	void	Draws this peripheral.

InfraredReceiver :: Peripheral

Attributes	**Attribute Name**	**Type**		**Description**
	data	InfraredData		The data of the infrared receiver.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	receive()	void	void	Receives the data.
	draw()	void	void	Draws this peripheral.



AnalysisTool

Attributes	**Attribute Name**	**Type**		**Description**
	id	int		The ID of the analysis tool.
	isEnabled	bool		If the analysis tool is enabled.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	enable()	void	void	Enables the analysis tool.
	disable()	void	void	Disables the analysis tool.
	display()	void	void	Displays the analysis tool.

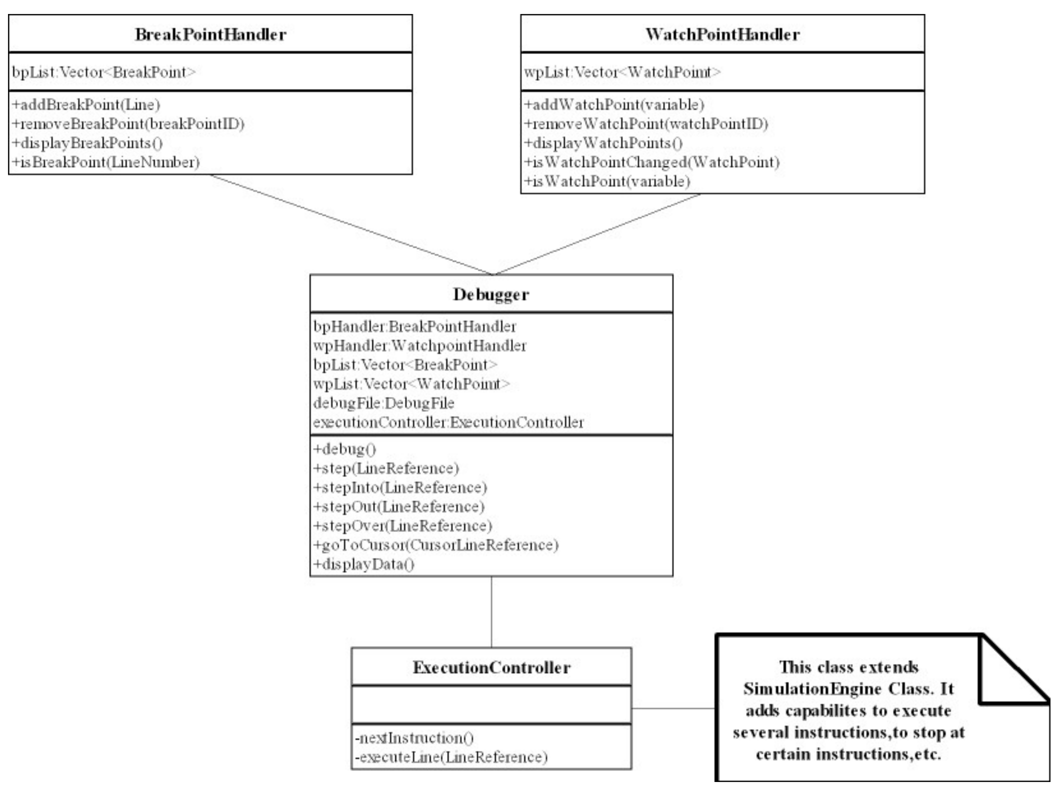
StopWatch :: AnalysisTool

Attributes	**Attribute Name**	**Type**		**Description**
	status	int		The stop watch status
	time	long		The time passed during execution
Methods	**Method Name**	**Return**	**Arguments**	**Description**

	startTimer()	void	void	Starts the timer.
	stopTimer()	void	void	Stops the timer.
	clear()	void	void	Resets the timer.

PinListener :: AnalysisTool

Attributes	**Attribute Name**	**Type**		**Description**
	port	Port		The Port that the pin belongs to.
	pinNumber	int		The pin number on the Port.
	status	int		Current status of the pin.
	timeChart	TimeChart		The time chart to display the pin value with respect to time.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	addStatusToTime-Chart()	void	int	Adds the given status to the timeChart.
	drawGraph()	void	void	Draws the timeChart graph.
	reset()	void	void	Resets the pin listener.



Debugger

Attributes	**Attribute Name**	**Type**		**Description**
	bpHandler	BreakPointHandler		The breakpoint handler.
	wpHandler	WatchPointHandler		The watchpoint handler.
	bpList	Vector<BreakPoint>		The list of the breakpoints.
	wpList	Vector<WatchPoint>		The list of the watchpoints.
	debugFile	DebugFile		The file used during debugging process.
	executionController	ExecutionController		The simulator used during debugging process.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	debug()	void	void	Starts the debugging process.
	step()	void	LineReference	Executes one step.
	stepInto()	void	LineReference	Steps into the next block.
	stepOut()	void	LineReference	Steps out of the current block.
	stepOver()	void	LineReference	Steps over the next block.
	gotoCursor()	void	cursorPosition	Executes upto the cursor position.
	displayData()	void	void	Displays the debug data.

ExecutionController :: SimulationEngine

Methods	**Method Name**	**Return**	**Arguments**	**Description**
	nextInstruction()	void	void	Executes the next instruction.
	executeLine()	void	LineReference	Executes one line in the AsmPlusFile.

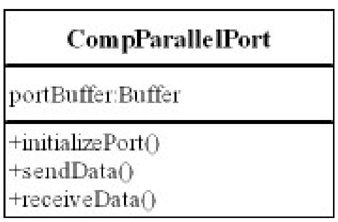
BreakPointHandler

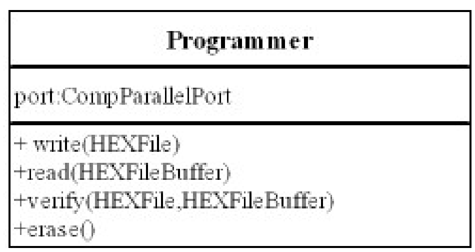
Attributes	**Attribute Name**	**Type**		**Description**
	bpList	Vector<BreakPoint>		The list of the breakpoints.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	addBreakPoint()	void	LineNumber	Adds a break point to the given line.
	removeBreakPoint()	void	bpID	Removes the break point with *bpID.*

	displayBreakPoints()	void	void	Displays the breakpoints on the editor.
	isBreakPoint()	bool	LineNumber	Returns true if there exists a breakpoint on the line with *lineNumber.*

WatchPointHandler

Attributes	**Attribute Name**	**Type**		**Description**
	wpList	Vector<WatchPoint>		The list of the watchpoints.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	addWatchPoint()	void	variable	Adds a watch point to the given *variable*.
	removeWatchPoint()	void	wpID	Removes the watch point with *wpID.*
	displayWatchPoints()	void	void	Displays the watch points.
	isWatchPoint()	bool	variable	Returns true if there exists a watch point associated with the *variable.*
	isWatchPoint-Changed()	bool	wpID	Returns true if the variable associated with *wpID* is changed.





Programmer

Attributes	**Attribute Name**	**Type**		**Description**
	port	CompParallelPort		The parallel port of the computer to be used for reading/writing programs to the PIC.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	write()	void	HexFile	Writes the hex file to the PIC.
	read()	void	HexFileBuffer	Reads the program on the PIC.
	verify()	void	HexFile, HexFileBuffer	Compares the program on the PIC with the one on the buffer and verifies.

	erase()	void	void	Erases the program on the PIC.

CompParallelPort

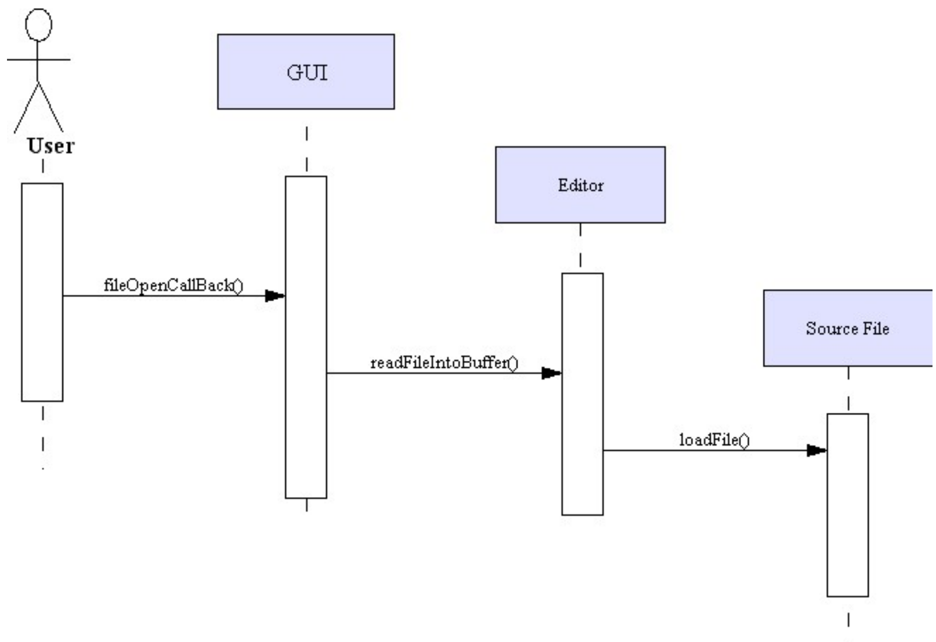
Attributes	**Attribute Name**	**Type**		**Description**
	portBuffer	Buffer		The buffer to be used for the parallel port of the computer.
Methods	**Method Name**	**Return**	**Arguments**	**Description**
	initialize()	void	void	Initializes the port.
	sendData()	void	void	Sends the data in the buffer to the port.
	receiveData()	void	void	Receives the data from the port into the buffer.

3.3. Sequence diagrams

3.3.1. Editor Module

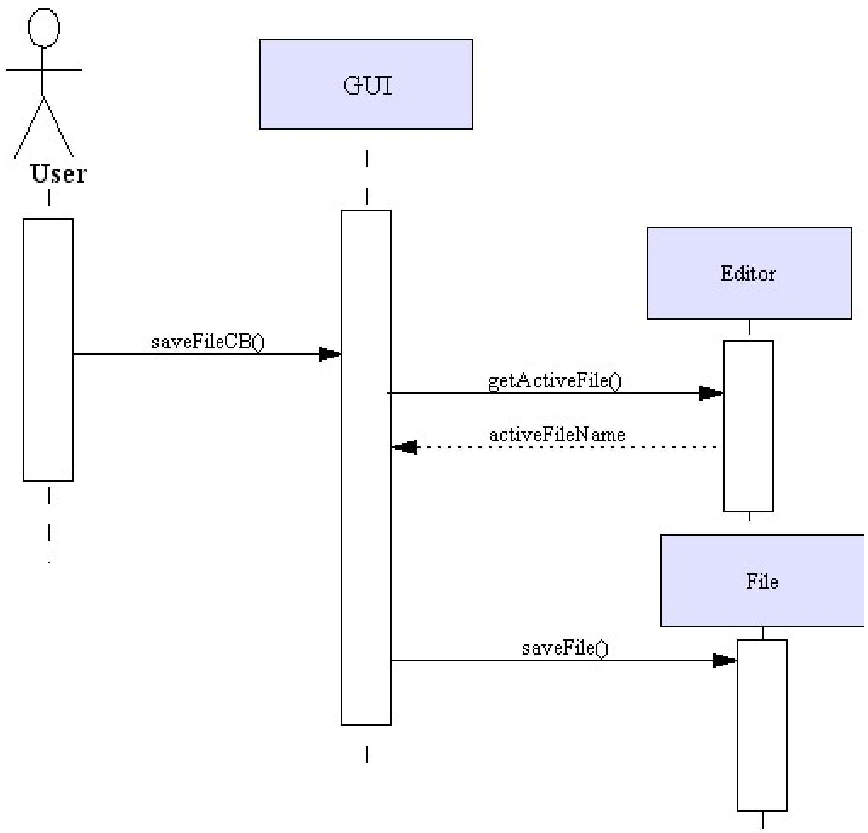
The Editor Module consists of several sub-modules.

Open File Module



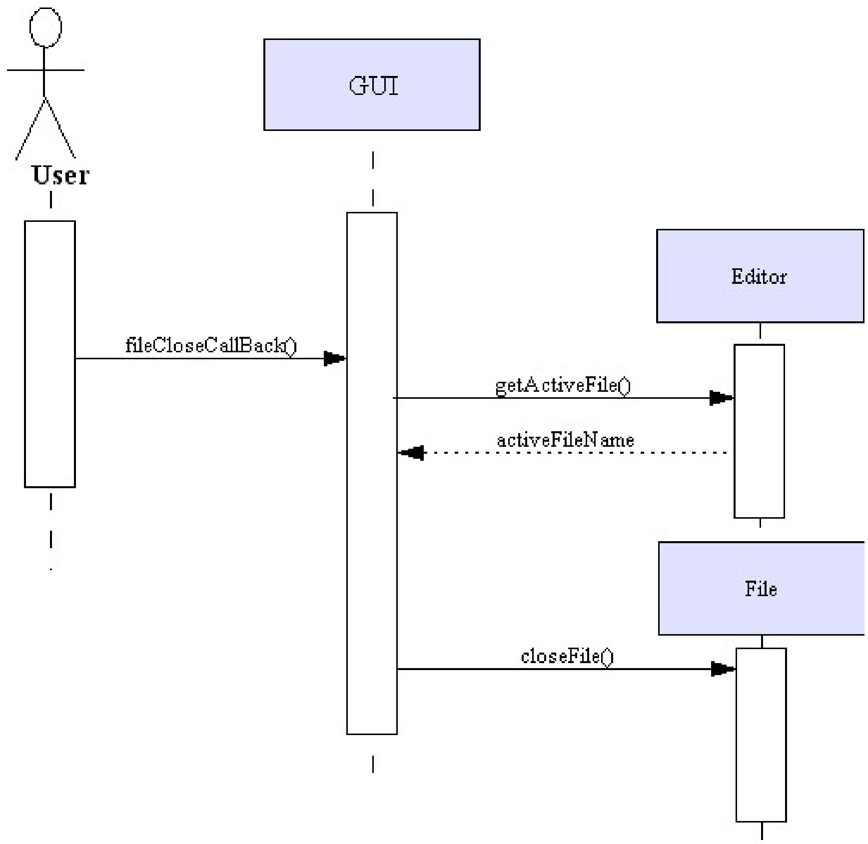
When the user selects "Open File" from the menu or from the screen, the fileOpenCallBack () function of GUI is called and this calls the clickedFile () method of Project object which returns the fileName of the clicked file. As soon as GUI gets the name of the file, it invokes the readFileIntoBuffer () method of the Editor object which reads the text data of the file into its buffer array. For this purpose, this function invokes the loadFile () method of the SourceFile object, which was created back in the GUI initialization. Once the file is read into the buffer the editor calls the displayText () function and it prints the buffer content into the monitor.

Save File Module



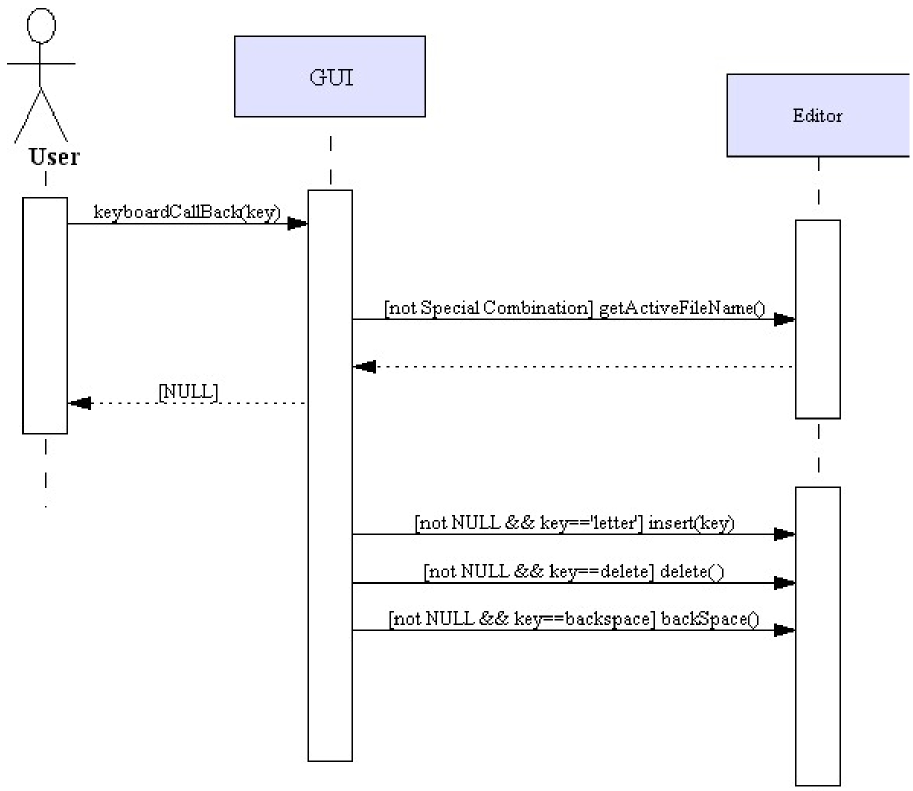
When the user selects "Save File" from the menu or from the screen, the fileSaveCallBack() function of GUI is called and this calls the getActiveFile() method of Editor object which returns the activeFileName of the current file. As soon as GUI gets the name of the file, it invokes the saveFile() method of the File object which saves the text.

Close File Module



When the user selects "Close File" from the menu or from the screen, the fileCloseCallBack() function of GUI is called and this calls the getActiveFile() method of Editor object which returns the activeFileName of the current file. As soon as GUI gets the name of the file, it invokes the closeFile() method of the File object which saves the text.

Edit File Module



When the user presses a key from the keyboard, the keyboardCallBack() method of GUI is called and it checks the parameter.

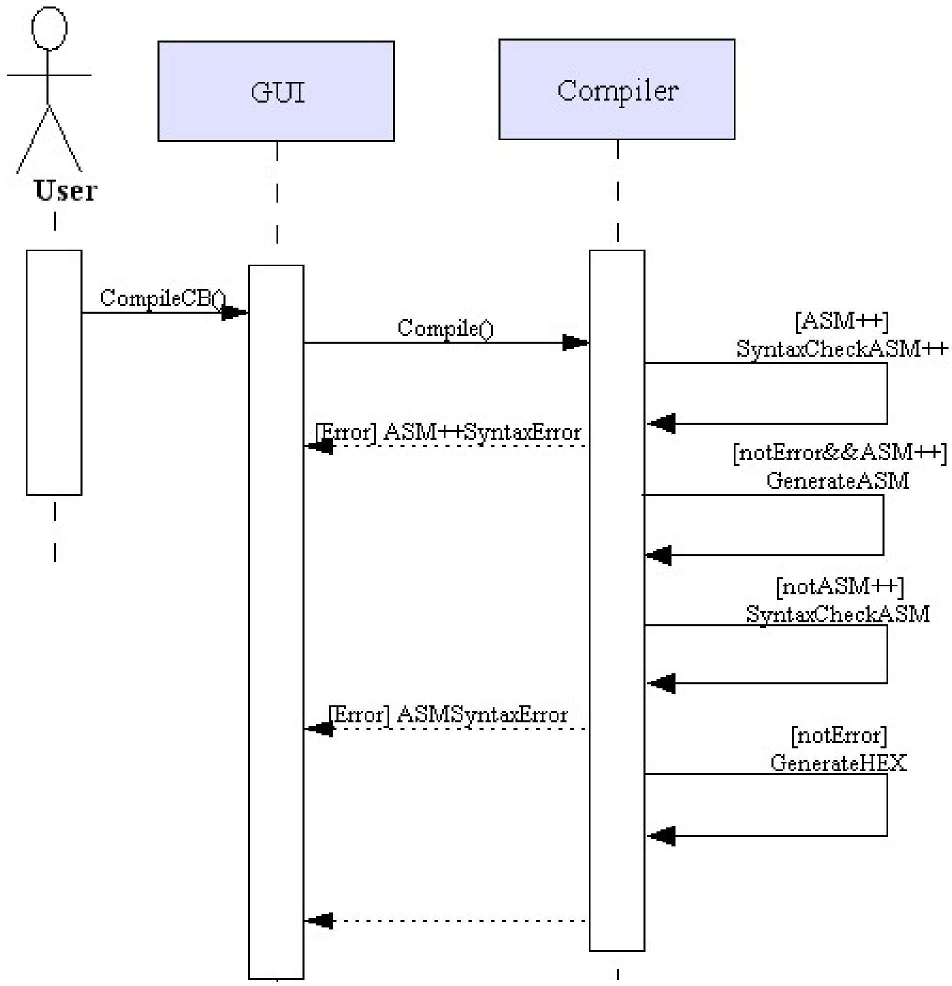
a. If the entered key is a special key combination it handles in itself and returns to the user without entering the Editor class.

b. If the entered key is not a special combination the getActiveFileName() method of the Editor. If the Editor returns a valid value i.e. if there exist a current active file open, GUI calls three different methods of the Editor object:

1. If the entered key is a single letter the insert(key) method,
2. If the entered key is the "Delete" key the delete() method,
3. If the entered key is the "Backspace" key the backSpace() method

If the Editor returns a null value, the GUI goes into idle state.

3.3.2. Compile Module



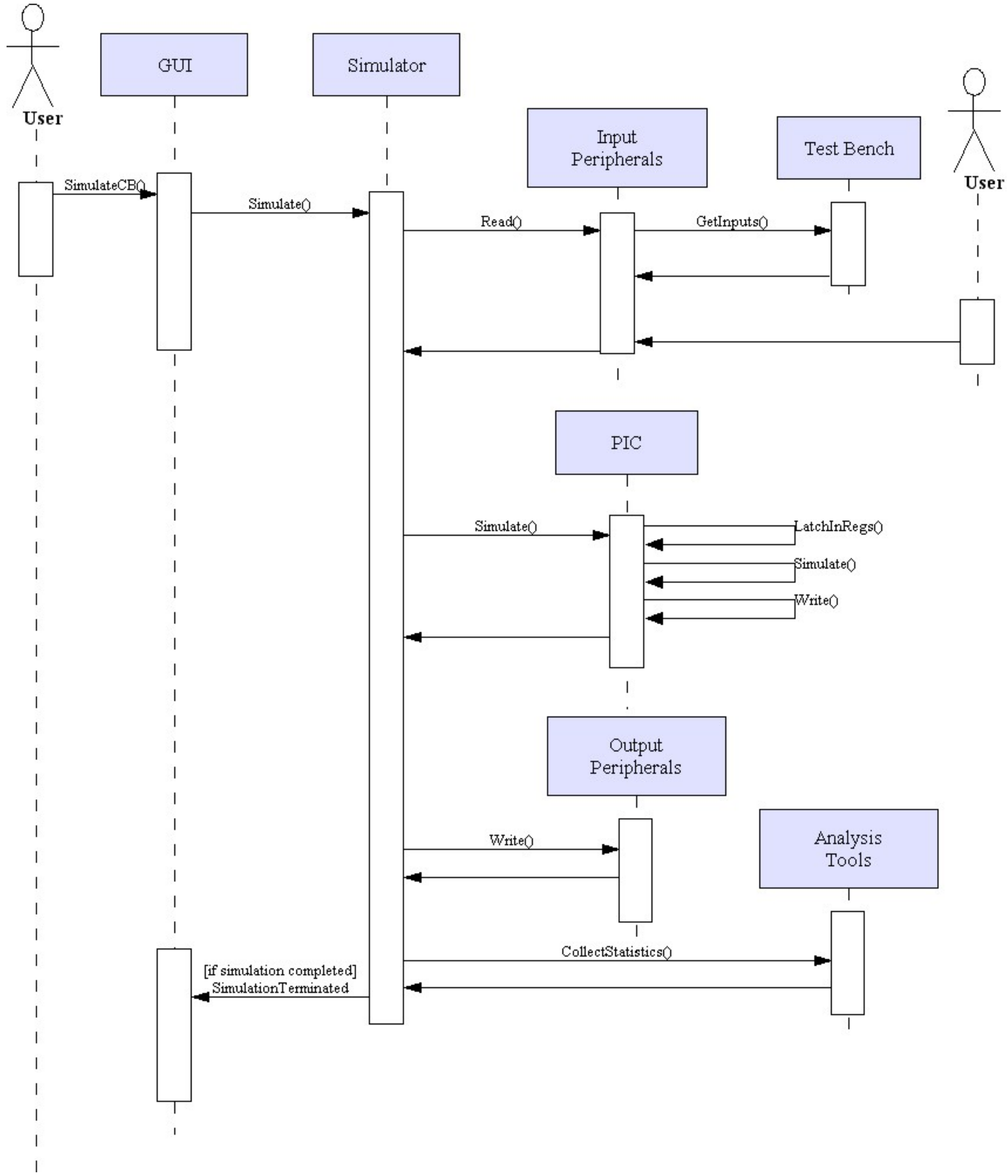
When the user presses the "Compile" button in the main window, the Project class, which is running in the background since the invocation of the program, calls the compileCallBack() function. This function calls the compile() function of the GUI object and creates a Compiler object, by using its constructor. The Compiler object checks the type of the file which is requested to be compiled:

1. If the file is an ASM++ file, it calls the syntaxCheckASM++() method, which checks if there is an error or not syntactically or not. If there are any errors it reports the error to its caller GUI and terminates immediately. If there are no errors, it generates an ASM file.

2. If the file is an ASM file, it invokes generateHEX() method which creates the HEX file correspondent of the ASM file.

Finally the Compiler object returns a successful termination to the GUI object.

3.3.3. Simulate Module



When user starts a simulation using GUI of PIDE, a message is sent to GUI class which calls simulate() function of the Simulator module. simulate() function runs until simulation is stopped. In the sequence diagram, operations after simulate() method simulates one clock cycle of the board. In other words, simulation runs in descrete time intervals. During simulation, simulate() keeps simulating the clock cycles.

Simulator module can be run in two different modes. First of them is the direct interaction with the user and the other is using a test file. In either case Simulator send Read message to all enabled input peripherals to see if there is an input from a source. Peripherals update their data accordingly and return. Simulator then sends simulate() message to PIC which first calls latchInRegs function to take a snapshot of the current registers before simulating a cycle of the PIC. After saving registers, PIC simulates its modules and saves last data with write() function. After simulating the PIC, Simulator now passes PIC's last state to output peripherals and simulates them. At the end of the cycle, analysis tools such as pin listeners update data with collectStatistics() method. Simulation is stopped by user's request.

3.3.4. Debugger Module

When the user presses the "Debug" button in the main window, the Project class, which is running in the background since the invocation of the program, calls the debugCallBack() function. This function calls the debug() function of the GUI object and creates a Debugger object, by using its constructor. The Debugger object calls the nextInstruction() method immediately which creates an ExecutionControl object. ExecutionControl object calls two functions:

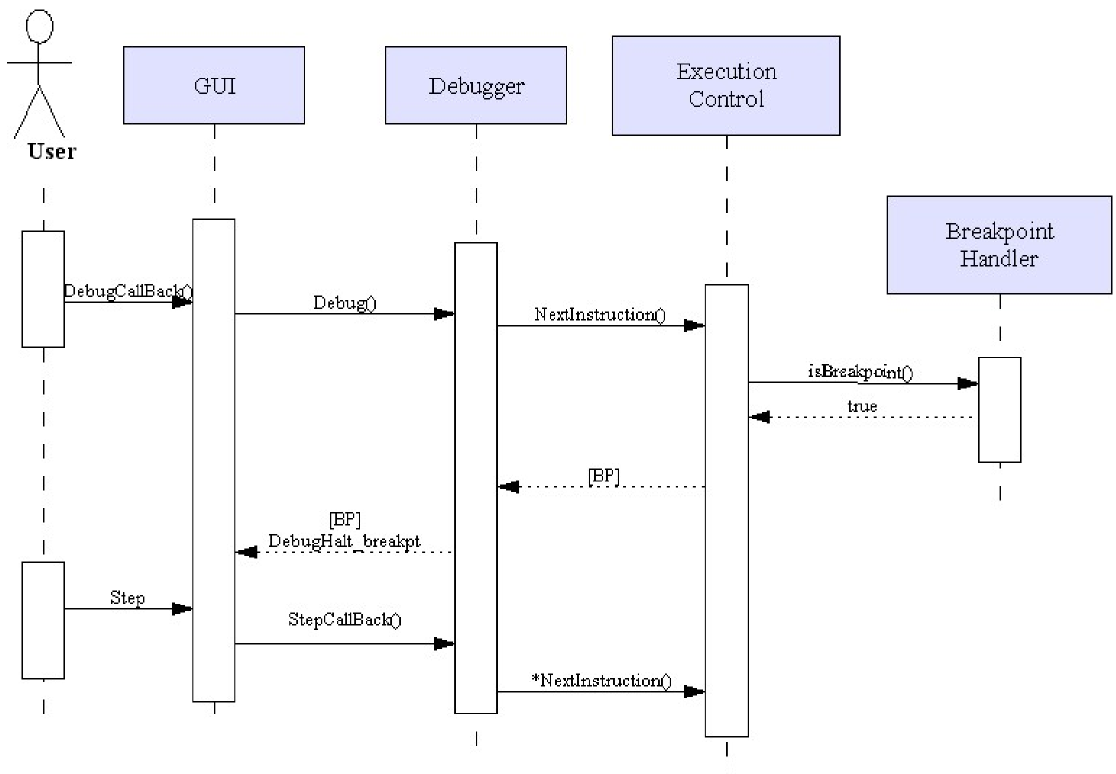
1. isBreakpoint() method of the BreakpointHandler object which was created by the Project object before. It returns "true" or "false".

2. isWatchpoint() method of the WatchPointHandler object which was created by the Project object before. It returns "true" or "false".

It takes three branches into execution according to the return values of these functions:

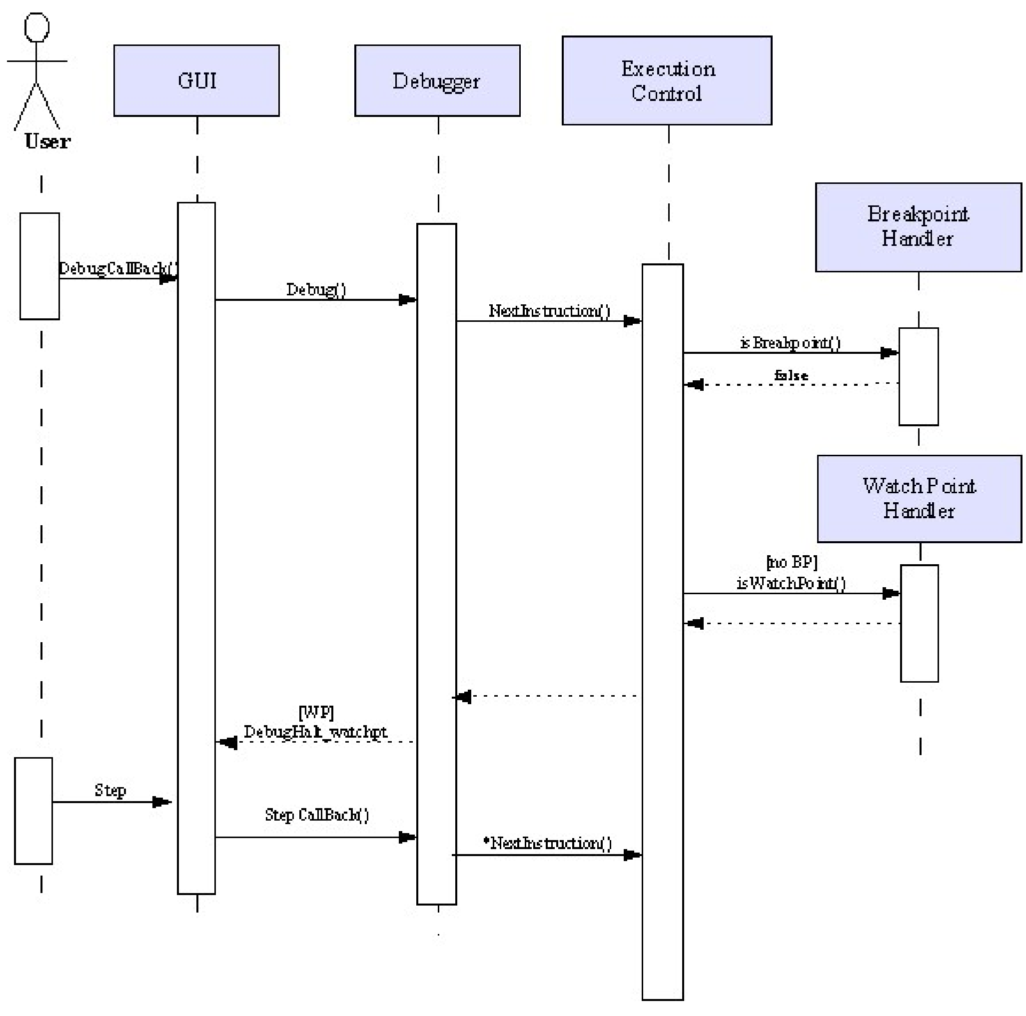
Debug:Breakpoint

If isBreakpoint() returns true, Execution Control returns the breakpoint information to the Debugger and it returns debugHaltBreakpoint to the GUI and at the end, GUI prompts the user to step into the next instruction. When user steps into the next instruction, the stepCallBack() method is invoked and the Debugger continues with the next instruction



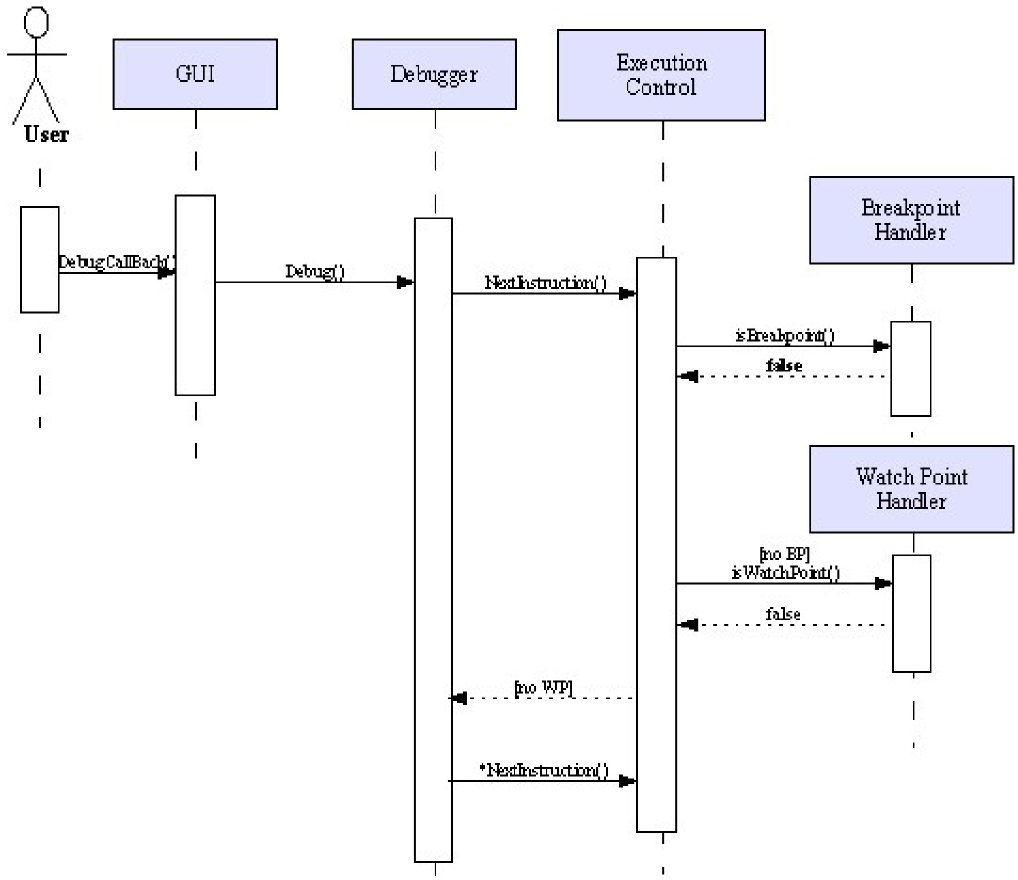
Debug:No Breakpoint but Watchpoint

If isBreakpoint() returns false and isWatchpoint() returns true, the Execution Control returns the watchpoint information to the Debugger and it returns debugHaltWatchpoint to the GUI and at the end, GUI prompts the user to step into the next instruction. When user steps into the next instruction, the stepCallBack() method is invoked and the Debugger continues with the next instruction.



Debug: No breakpoint and no watchpoint

If isBreakpoint() returns false and isWatchpoint() returns, the Debugger steps into the next instruction without any prompt.

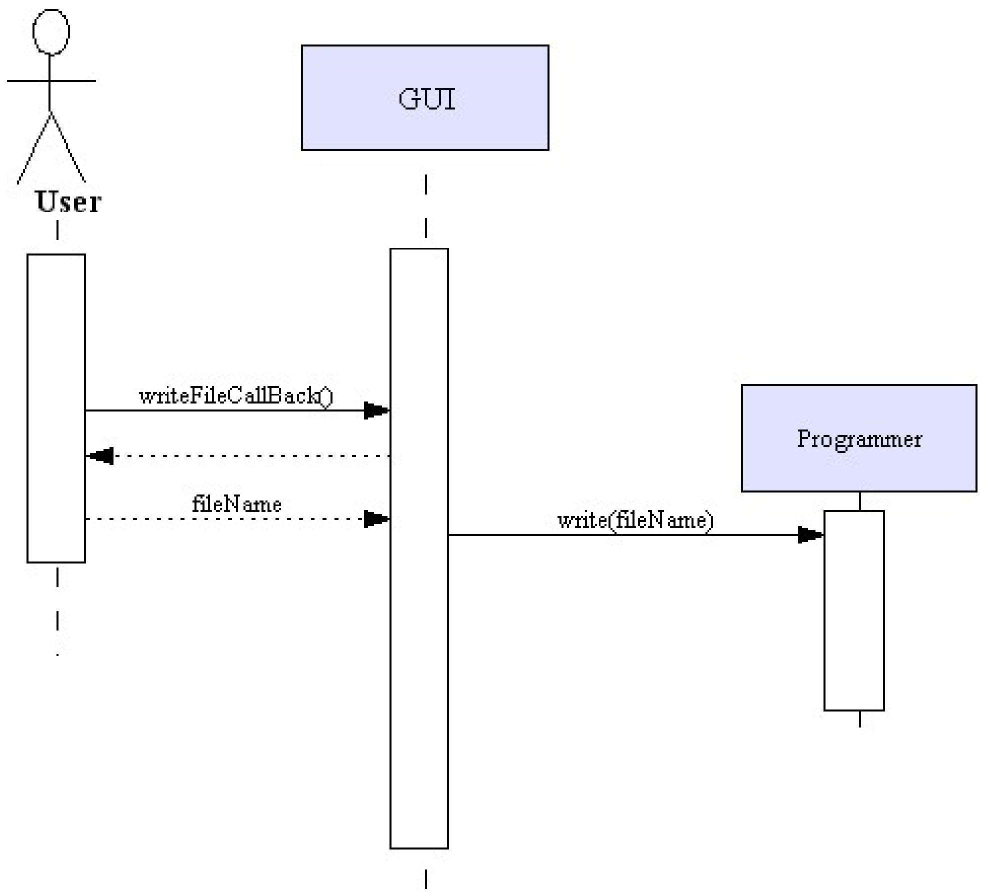


.

3.3.5. PIC Programmer Module

The Programmer Module consists of three sub-modules.

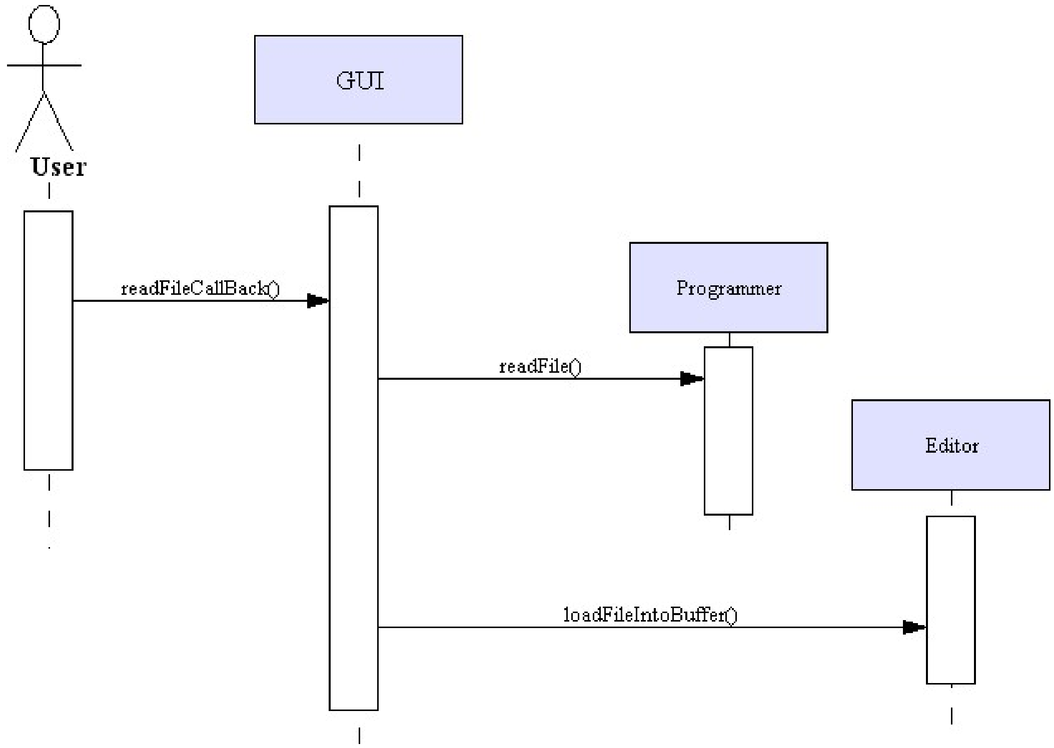
Write



When the user selects "Write File" from the menu, the writeCallBack() function of the GUI is called and it prompts the user to choose the file to be downloaded.

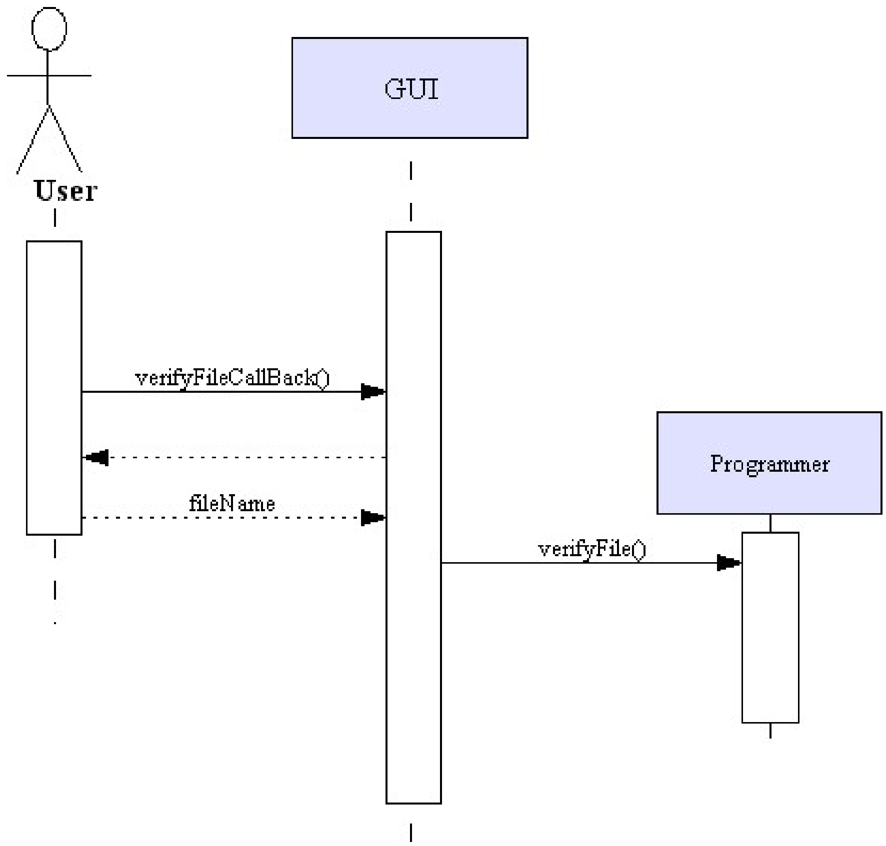
The user specifies the file path. Then the module programs the PIC using parallel port.

Read



When the user selects "Read File" from the menu, the readCallBack() function of the GUI is called. Then the file is uploaded by calling the uploadFile() method of the Programmer module. The uploaded file is read to the editor buffer by calling the loadFileIntoBuffer() method of the Editor object.

Verify



When the user selects "Verify File" from the menu, the verifyFileCallBack() function of the GUI is called and it prompts the user to select the file to be verified. The user specifies the path. Then the module compares the files by calling the verifyFile() function of the Editor module.

4. Graphical User Interface Design

Below in Figure 4.1, the GUI of the PIDE program, showing the menus, toolbars and the status bar can be found.

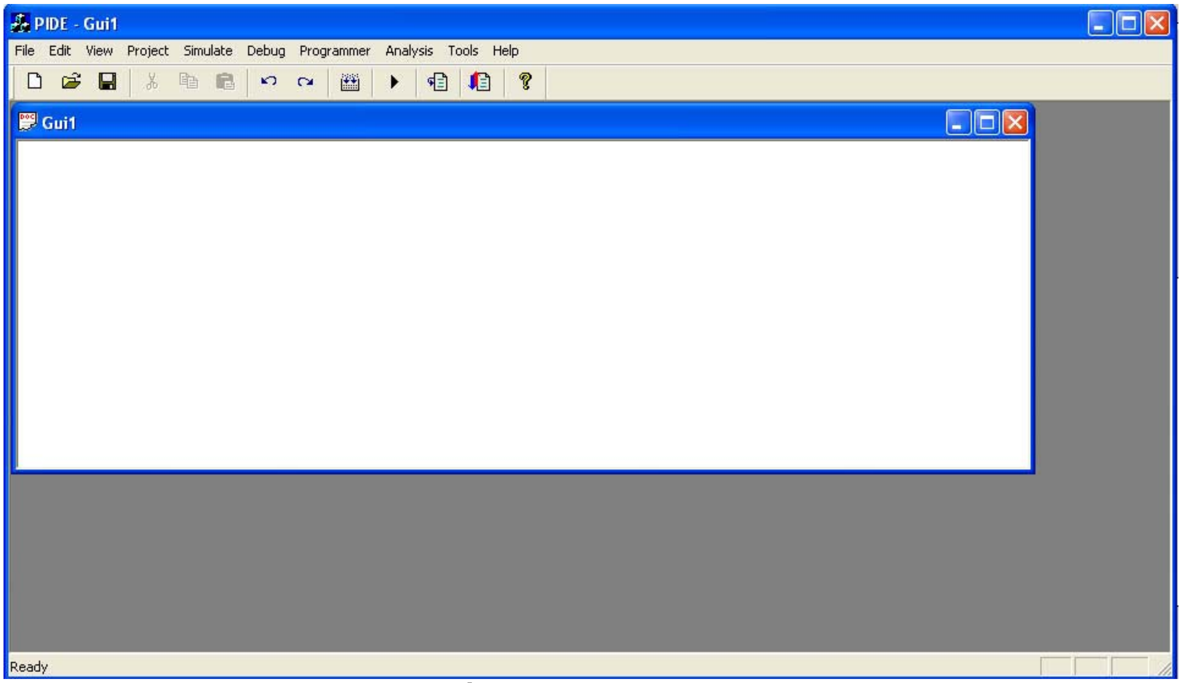


Figure 4.1

Figure 4.1 shows the case when there is no active project. When the user creates/opens a project, the workspace view will also be present on the left hand side. The program will be able to handle multiple opened files.

In Figure 4.2, the menu bar of the PIDE is shown. The menu items will be explained in detail in the following sections.

Figure 4.2

In Figure 4.3, the toolbar of the PIDE is shown. Here exist shortcuts of the frequently used operations in the menu bar.

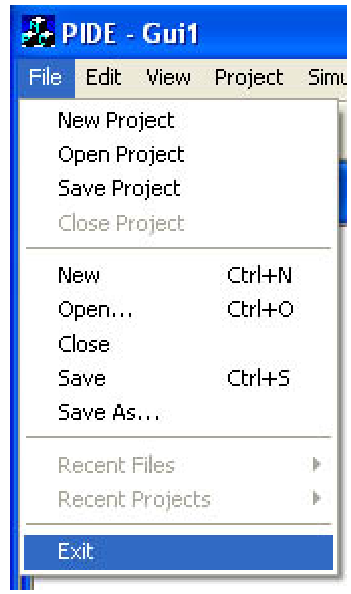
Figure 4.3

MENUS

There are *File*, *Edit*, *View*, *Project*, *Simulate*, *Debug*, *Programmer*, *Analysis*, *Tools* and *Help* menus in the PIDE program. The operation of each menu item is described below.

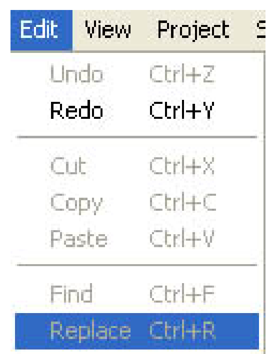
FILE MENU

New Project	Create a new project.
Open Project	Open an existing project.
Save Project	Save the current project.
Close Project	Close the current project.
New	Create a new file.
Open…	Open an existing file.
Close	Close the current file.
Save	Save the current file.
Save As…	Save the current file with a different name or save to a different place.
Recent Files	Shows the most recently used files,
Recent Projects	Shows the most recently used projects,
Exit	Quit from the program.



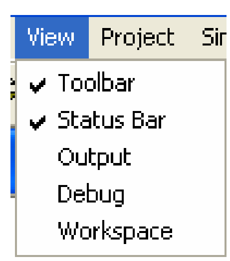
EDIT MENU

Undo	Undo the last action.
Redo	Redo the last undo action.
Cut	Cut the selected item.
Copy	Copy the selected item.
Paste	Paste the last cut or copied item.
Find	Find a given word in the current file.
Replace	Replace the given word with another word.



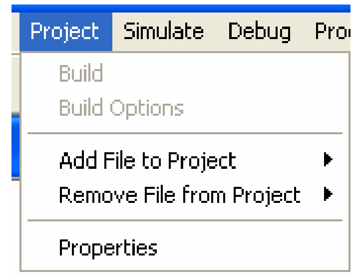
VIEW MENU

Toolbar	Show/Hide the toolbar.
Status Bar	Show/Hide the status bar.
Output	Show/Hide the output view.
Debug	Show/Hide the debug windows.



Workspace	Show/Hide the workspace view.

PROJECT MENU

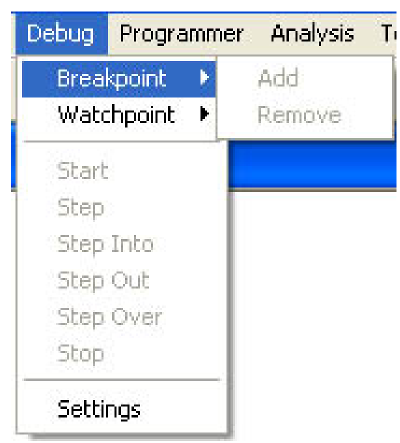


Build	Build the current project.
Build Options	Change the build options.
Add File to Project	Add a new file to the current project.
Remove File from Project	Remove a file from the current project.
Properties	Change the project properties.

SIMULATE MENU

Start	Start the simulation.
Pause	Pause the simulation.
Continue	Continue the simulation.
Stop	Stop the simulation.
Settings	Change the simulation settings.

DEBUG MENU

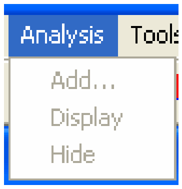


Breakpoint	Add or Remove breakpoints.
Watchpoint	Add or Remove watchpoints.
Start	Start the debugging process.
Step	Execute one step.
Step Into	Step into the next block.
Step Out	Step out of the current block.
Step Over	Step over the next block.
Stop	Stop the debugging process.
Settings	Change the debug settings.

PROGRAMMER MENU

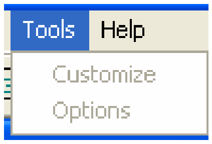
Write	Write the current program onto the PIC.
Read	Read the program in the PIC.
Verify	Verify if the program is written correctly onto the PIC.
Erase	Erase the program written in the PIC.
Settings	Change the programmer settings.

ANALYSIS MENU



Add	Add a new analysis tool.
Display	Display the analysis results.
Hide	Hide the analysis results.

TOOLS MENU



Customize	Customize the program settings.
Options	Change the program options.

HELP MENU

About PIDE…	Show brief information about the program.
Contents	Show the help contents.
Index	Show the help index.
Search	Search a help topic in the help contents.

5. Components to be Simulated

5.1. PIC MCU

The PIC microcontroller instruction set contains 35 basic instructions. All of those basic instructions are single word, i.e. 14 bits. They last finite durations, read from some specific registers and update some other specific registers. Therefore, simulations of all 35 instructions are independent and atomic.

5.1.1. Memory

The memory system of the MCU is composed of FLASH program memory, the RAM Data Memory and the EEPROM Data Memory.

FLASH Program Memory

The program to be uploaded is stored in the Flash Program memory, which has 8KB storage. Each instruction is 14 bits wide. Since the program counter is 13 bits wide, 2^{13} = 8K-words can be addressed in the Flash program memory.

Paging

The FLASH program memory consists of four pages. The address ranges of those four pages are given below.

Page Number	Start Address	End Address
Page 0	0005h	07FFh
Page 1	0800h	0FFFh
Page 2	1000h	17FFh
Page 3	1800h	1FFFh

As a result of the paging system of the program memory, the operations of the jump instructions require special attention. The CALL/GOTO instructions take 11bit arguments, addressing only 2KB of the memory. Actually, the MSB 2 bits of the address are taken from PCLATH<4:3>. Therefore, when a subroutine in another page is to be called, first the PCLATH<4:3> bits should be set accordingly, and then the low order 11 bits should be given to the CALL instruction.

Registers

Register	**Usage**
EEDATA	Data
EEDATH	Data
EEADR	Address LSBs
EEADRH	Address MSBs (0000h-1FFFh)
EECON1	controls
EECON2	controls

Read and Write Operations

Data read operation from the FLASH memory is performed as single word read and data write operation is performed as four word block write.

Read

1. Write address to EEADRH and EEADR
2. Set EEPGD
3. Set RD
4. Wait for 2 cycles idle (those statements are ignored)
5. Read from EEDATH and EEDAT

Write

A write operation to the FLASH program memory can only be performed if not write-protected mode is selected, as defined in device configuration word bits WRT<1:0>

Data is written in four word blocks, where a block is four words with sequential addresses. These four words are identified by EEADR<1:0> bits.

Load ALL 4 buffer registers with order 00-01-10-11:

1. Write address to EEADRH and EEADR
2. Write data to EEDATH and EEDATA
3. Set EEPGD
4. Write 55h to EECON2
5. Write AAh to EECON2
6. Set WR
7. Wait for 2 cycles
8. When last one is written, data is transferred from buffers to FLASH.
9. Then, processor waits for 4ms for the write to be completed.

RAM Data Memory

The RAM data memory is 512B, containing the special purpose registers and general purpose registers (368B).

Bank System

The RAM Data memory is comprised of 4 banks. Bank selection is performed by means of RP1 (Status<5>) and RP2 (Status<6>) bits.

128 Bytes/Bank (128 = 0x7F)

General Purpose Registers

Special Purpose Registers

An important note should be added here. Since the central processing unit of the PIC microcontroller has a very limited RISC architecture core, it has no special registers in it. Also the memory read/write speed is the same as the registers inside the CPU. As a result, the memory of the PIC is used just as registers. Therefore, Microchip refers the memory words as registers and in this report from this point forward, the data memory words will be referred as registers.

The distribution of the data memory space is given in the figure. As can be seen from the above distribution of the registers, the first portions of all four banks are reserved for special purpose registers and the rest for general purpose registers. The bitwise explanation of the special purpose registers are given in the PIC 16F877 datasheet by Microchip.

A careful examination of the above data memory address space gives us why Microchip defines the data memory “up to 368 Bytes”. The General Purpose Registers are totally 96+80+16+80+16+80 = 368 Bytes.

Among the special purpose registers, the some registers are of special interest. Those registers are STATUS, OPTION, and INTCON. STATUS register is controlled to switch between the banks, Time-out, Power-down modes and carry/borrow control of arithmetic operations. OPTION register is used to enable PORTB internal weak pull ups, Interrupt enabling, timer source and edge and prescale selections. INTCON register is used to configure the interrupts in the system. Global, peripheral, timer, external, portB interrupts are enabled and the flags are read/cleared from this register. There are also PIE1, PIR1, PIE2, PIR2 registers for enabling peripheral interrupts and their flags.

The PCON register contains the flags for different types of reset operations such as power-on reset, watchdog reset external reset and brown-out reset.

FIGURE 2-3: PIC16F876A/877A REGISTER FILE MAP

	File Address		File Address		File Address		File Address
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD(1)	08h	TRISD(1)	88h		108h		188h
PORTE(1)	09h	TRISE(1)	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved(2)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Reserved(2)	18Fh
T1CON	10h		90h	General Purpose Register 16 Bytes	110h	General Purpose Register 16 Bytes	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch	CMCON	9Ch		11Ch		19Ch
CCP2CON	1Dh	CVRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h
			EFh		16Fh		1EFh
		accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h - 7Fh	1F0h
	7Fh		FFh		17Fh		1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

■ Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876A.
2: These registers are reserved; maintain these registers clear.

Indirect Addressing

Indirect addressing is accomplished by means of INDF virtual register name. When INDF is used as the target address, actually the address pointed by the FSR

(File Select Register) register is accessed. 8 bits in FSR register and 1 IRP bit give 9 bits to address the overall 2KByte data memory (000h – 1FFh).

EEPROM Data Memory

The EEPROM data memory has 256Bytes of storage and is the non-volatile data storage system.

Register	**Data Memory**
EEDATA	8 bit data
EEADR	Address (00h-FFh)
EECON1	Controls
EECON2	Controls
PIR2	flags

Data read operation from the EEPROM memory is performed as single byte read and data write operation is performed as single byte write. The EEPROM data memory is not directly addressed, but is accessed indirectly via special registers.

EECON1 Register Contents

EEPGD=0 Data
EEPGD=1 Program
RD read , can only be set by user; reset by hardware
WR write , can only be set by user; reset by hardware
WREN write enable
WRERR write error when there's a MCLR or WDT reset

PIR2 Register Contents

EEIF Write complete interrupt flag

Read Operation

1. Write address to EEADR
2. Clear EEPGD
3. Set RD
4. Next cycle, data is ready at EEDATA, so next instruction can read it

Write

WR inhibited from being set if WREN is cleared

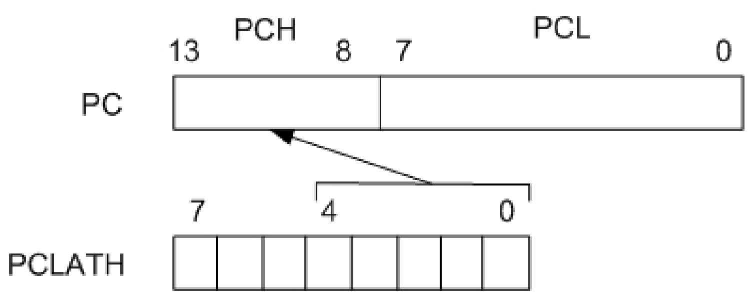
1. Write address to EEADR.
2. Write 8-bit data EEDATA
3. Clear EEPGD
4. Set WREN

5. Disable interrupts (if enabled).
6. Execute the special five instruction sequence:
    i. Write 55h to EECON2 in two steps (first to W, then to EECON2)
    ii. Write AAh to EECON2 in two steps (first to W, then to EECON2)
    iii. Set the WR bit
7. Enable interrupts (if using interrupts).
8. Clear the WREN.
9. At the completion of the write cycle, the WR bit is cleared and the EEIF interrupt flag bit is set. (EEIF must be cleared by firmware.)

Program Counter

The program counter (PC) of the Microcontroller is a part of the data memory. The value inside the PC shows the next instruction to be executed in the program memory. The PC 13 bits, and is held in two registers.

- 8 LSBs (<7:0>) are in PCL register, readable and writable.
- MSB 5 bits (<12:8>) are copied from PCLATH register (<4:0>) on a “write to PC” instruction such as “ADDWF PCL”.



PC Stack

Related to the PC, the stack is of primary importance. Stack is used to store the current value of the PC in case of a subroutine/function call, to be able to proceed with normal operation upon return. The user cannot access (i.e. read or modify) the stack.

- Stack is 8 PC words (13 bits) deep.
- Stack pointer is not readable / writable
- Stack is circular, i.e. a 9th write overwrites stack address 0.

5.1.2. PORTS

There are five ports on the microcontroller. These ports are used for various purposes, but mostly for digital I/O. The names of the ports and the number of pins on each are PORTA (6), PORTB (8), PORTC (8), PORTD (8) and PORTE (3).

Port Name	Pins	Connected Modules

PORTA	5	Digital Input/Output A/D Converter(default) Comparator SPI Timer0
PORTB	8	Digital Input/Output External Interrupt Weak internal pull-up Interrupt on Change
PORTC	8	Digital Input/Output Timer1 PWM 1-2 SPI I2C USART
PORTD	8	Digital Input/Output Parallel Slave Port
PORTE	3	Digital I/O A/D Converter Parallel Slave Port

Interrupts

There are 15 sources of interrupts in the system. Therefore, that number of interrupt vectors will be used to select the address to be jumped onto in case of an interrupt. Among the most important interrupt vectors, the reset vector of the system resides in the address 0000h and the external interrupt vector in 0004h.

5.1.3. Parallel Slave Port

Parallel Slave Port registers and usage

Set TRISE<2:0> for inputs

ADCON1<3:0> for digital I/O not analog I/O

Write with WR low and CS low, when any one becomes high, IBF flag is set, and PSPIF interrupt flag is set

Read PORTD to clear IBF

If a second write before read, IBOV is set

Read with RD low and CS low, OBF is cleared, when any one becomes high, PSPIF interrupt flag is set, OBF low until data is written

5.1.4. Analog to Digital Converter

The conversion of an analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low-voltage reference input that is software selectable to some combination of VDD, VSS, RA2 or RA3.

The ADRESH and ADRESL registers contain the 10-bit result of the A/D conversion. When the A/D conversion is complete, the result is loaded into this A/D Result register pair, the GO/DONE bit (ADCON0<2>) is cleared and the A/D interrupt flag bit ADIF is set. The block diagram of the A/D module is shown in Figure 11-1.

Clearing the GO/DONE bit during a conversion will abort the current conversion. The A/D input pins must be configured as input pins via the TRIS register to be used as analog inputs.

INTCON	Interrupt Enable
PIR1	Interrupt flag
PIE1	Interrupt enable
ADRESH	Conversion Result MSBs (or LSBs)
ADRESL	Conversion Result LSBs (or MSBs)
ADCON0	Analog input channel selection Conversion clock selection Conversion flag A/D enable
ADCON1	AD port configuration Result format selection
TRISA	Pin directions
PORTA	Analog input port
TRISE	Pin directions
PORTE	Analog input port

5.1.5. Other Features of the MCU

Timer0, 8Bit timer/counter with 8Bit prescaler

Timer1, 16Bit timer/counter with prescaler

Timer2

Capture-Compare-PWM Modules

SSP, Synchronous Serial Port

SPI, serial Peripheral Interface

I2C

USART, Universal Synchronous / Asynchronous Receiver Transmitter (9-bit)

BOR, Brown Out Reset

Analog Comparator Module

WDT, Watchdog Timer

Sleep Mode

5.2. Peripherals

The CEng 336 board is a complete evaluation board that contains various devices on it. These devices can be classified into two with respect to their usage, input devices and output devices. The list of the peripherals on the board are given below with their brief explanations.

5.2.1. Input Peripherals

Parallel Port

Parallel port (LPT) is the port that is used for programming the microcontroller on the evaluation board. This port can be used for parallel communication, such as PSP mode, or for serial communication, either synchronous or asynchronous.

Serial Port

Serial port connection, i.e. RS232, is used for asynchronous serial data transfer between other devices and the microcontroller.

USB Port

The USB port is a high speed serial communications interface. For PIC applications, in fact the speed of the USB port is very high, however since in the recent PCs, the serial communications port is being replaced with the USB ports, the controller should be able to communicate using this protocol.

Smart Card Reader

Smart card reader provides extra storage capability to the system. Since the storage capacity of the EEPROM on the MCU is limited, some extra storage may be necessary. The addressing and read/write operation of the reader should be modeled in the system.

Infrared Transmitter and Receiver

Infrared communication is included on the board to be used for special purpose applications. The system is internally analog and requires special modelling.

Keypad

There are 16 pushbuttons on the evaluation board. The pushbuttons are active high buttons, pulled low during normal operation.

Reset Pushbutton

The reset pushbutton, being active low, is directly connected to the reset of the microcontroller. An MCLR signal is asserted with this input.

5.2.2. Output Peripherals

Led Array

A light emmiting diode (LED) is nothing but a semiconductor device that emits light when given logic high value.

Seven Segment Display Array

A collection of LEDs, arranged in a format that will enable the display of alphanumeric characters is called a seven segment display. On the CEng336 board, there are three of those devices, forming an array.

LCD

Using light emmiting diodes for displaying data is clearly not the best method. Seven segment displays improve the user interface a little but still, it is very old fashioned. Newest systems always include some LCD components as the interface. These devices latch in the data entered, decode the characters and display them on their screen. Moving the cursor on the LCD and deleting are some special operations available on most of the off-the-shelf LCD modules.

Speaker

A speaker is a source of accoustic waves. The input signal is analog and the frequency/intensity of the accoustiv waves is determined by the input waveform characteristics.

6. Language Specifications

6.1. ASM++ Language Format

A Simple Language

We have decided to define a new language which is simply an improvement on assembly language, including some new keyword definitions or introducing some high definition language concepts such as function calls or variable definitions. The name of the language is ASM++ (ASM plus plus).

General

ASM++ is not case sensitive. Upper-case letters and lower-case letters are not considered to be distinct in all tokens, including reserved words. White space (space character, tab character and end-of-line) serves to separate tokens; otherwise, it is ignored. No token can extend past end-of-line. Spaces may not appear inside any token except character and string literals. A comment begins with two forward slashes or a semicolon as it is default for assembly language and extends to end of line, as in C++.

Identifiers

Identifiers start with a letter and contain letters and digits. An identifier must fit on a single line and its first 20 characters are significant.

Reserved Words

The following keywords are reserved in ASM++:

adda1a2	suba1a2	addwa1a2	suba1a2	swapa1a2
iorwa1a2	andwa1a2	xorwa1a2	mova1a2	if
else	then	for	function	begin
end	define	var	array	

Literals

An integer literal consists of a sequence of one or more digits in decimal or hexadecimal format.
A character literal is a single character enclosed by a pair of apostrophes (sometimes called "single quotes".) Examples include 'A', 'x', and '''. A character literal is distinct from a string literal of length one.
There is nothing like string literal.

Other Tokens (delimiters and operators)

: ; , ( ) & | as one character
! < = > '
!= >= <= // as two characters
and the end-of-file character

Macros

Macros are introduced by declarations of the form

define ID number

Variables

Variables are introduced by declarations of the form

var ID, ID, ..., ID

For example:
var a 0x121

Arrays

Arrays are introduced by declarations of the form

```
array name(address, length)
```

For example:

```
array a(0x5510,10)
```

Expressions

For binary operators, both operands must be the same type. Similarly, for assignment compatibility, both the left and right sides must have the same type.

Short Circuiting

Logical operators and and or use short-circuit evaluation.
This means that as soon as the truth value can be determined, evaluation stops. For example, if the first operand of an and evaluates **false**, the expression will evaluate **false** no matter what the second operand is, so the second operand is not even evaluated. If the first operand of an or evaluates **true**, the second isn't evaluated either.

Statements

Assignment statement

("=" is the assignment operator). For example

```
var a 0x121
a = 0x1C4
```

If statement

```
define MAX 100
define MIN 0
..............
..............
if x > MAX then
   goto hede
```

```
else if x < MIN then
   goto hodo
```

Loop Statement

The compiler will support while and for loops.

Example:

```
while(hede)
begin
................
................
end
```

Function Definitions

The compiler will be supplying the function calls. They can be limitedly nested which are defined as follows:

```
function func_name(parameter1, parameter2)
{
        ...........
        ............
}
```

The user will be provided a bunch of library functions for use.

Comments

The comments are specified by a semicolon or two forward slashes. It will be covering the whole line it is put at.

6.2. Test Bench (.test) File Format

During the simulation of a source file, the user will want to enter various inputs to the system. The input devices on the board are communication ports, keypad, pushbuttons and pots. Using a test bench file, the user can state the exact time instants that the inputs from these devices will be modified, e.g. a reset signal may be asserted for a period. Test bench files will release the burden of entering the inputs to peripherals at correct instants. This is especially useful in the case of high frequency input requirements.

Test bench file can control the system inputs in two different modes. In the Peripheral mode, the user may control the timing of the inputs to the peripheral devices. Alternatively, in the PIC mode, the user may choose to directly access the pins of the microcontroller. The mode selection is performed by <ModeName> tag. A test file may contain only one mode selection tag.

The format of the test bench files is given below.

```
timescale <time unit>

<PIC>
        #<time> PORT<Port Name>.PIN< Pin No> = <Expression1>
        #<time> PORT<Port Name> = <expression2>


        always #<time> PORT<Port Name>.PIN<Pin No> = <expression1>
        always #<time> PORT<Port Name> = <expression2>

        #<time> $finish
```

```
timescale <time unit>

<PERIPHERAL>
        #<time> <DeviceName>.PIN<Pin No> = <expression3>
        #<time> <DeviceName> = <Expression4>

        always #<time> <DeviceName>.PIN<Pin No> = <expression3>
        always #<time> <DeviceName> = <Expression4>

        #<time> $finish
```

Indentation is not important, since the parser ignores white spaces. The instructions are not case-sensitive.

The language for the Peripheral and PIC modes are defined below.
For Peripheral Mode:

```
<Expression3> = 0 | 1 | <DeviceName>.PIN<Pin No>
                       | ~<DeviceName>.PIN<Pin No>


<Expression4> = <word>    | <DeviceName>  + <CONST>
                               | <DeviceName>  - <CONST>

<Device Name> = LPT | RS232 | USB | Keypad | Reset
```

PIC Mode:

```
<Expression1> = 0 | 1 | PORT<Port Name>.PIN<Pin No>
                       | ~PORT<Port Name>.PIN<Pin No>


<Expression2> = <byte>    | PORT<Port Name> + <CONST>
                          | PORT<Port Name> - <CONST>
                          | PORT<Port Name>

Port Name = PORTA | PORTB | PORTC | PORTD | PORTE
```

<u>Example Files</u>

For PIC Mode:

```
timescale <1ms>

<PIC>
      #0 PORTA = 0
      #0 PORTB = 0

      always #10 PORTA.2 = ~PORTA.2
      always #100 PORTB = PORTB + 1

#<1000> $finish
```

For Peripheral Mode:

```
<PERIPHERAL>
      #0 Keypad = 0
      #0 Reset = 1
      #5 Reset = 0

      #10 Keypad.PIN5 = 0
      #10 Keypad.PIN2 = ~Keypad.PIN2
      always #100 Keypad.PIN3 = ~Keypad.PIN3

#<1000> $finish
```

7. File Formats

7.1. Project File Format

PIDE is desgined to be able to create projects and save workspaces for a better IDE experience. PIDE saves all necessary information in a file <project_name>.pde to recreate a previously used workspace. "pde" is the PIDE project save file extension. Each project has a pde file under its project folder. Below are the specifications and format of the project file. Since not all the desgin specifications are final, the file specifications and format is subject to change with high possibility.

Project Description in Project File

Project files include a project description section at the begining. It includes version of PIDE, name of the project, user/corporate name, creation and last modification dates of the project and description of the project if available. Each description is leaded by a keyword and followed by a new line. Project description can span several lines with project description token (#) at the beginning of each line. Below is an example of the project description section.

```
#PIDE 1.0- PIC Integrated Development Enviroment with ASM++
#Project_Name= Heat Sensor
#Creator= e1347061
#Created@ 2/12/2006 13:29:06
#Modified@ 2/12/2006 13:45:33
#Description= Ceng336 odevi icin yazdigimiz bir isi sensoru
```

Other Files in Project File

Project file holds trace of all files included in the project. These files may be ASM++ source files, ASM files, HEX files, debug files and test files. Each file is defined with its type and path name. The lines preceeding types of the files begin with file type token (>) and file paths are saved after "FILE=" keyword. Below is an example of files.

```
>ASM++
FILE= ./source/heat sensor.asm++
>ASMHEADER
FILE= ./myLib/a2dcalculate.ah
>ASMHEADER
FILE= ./d2acalculate.ah
>TESTFILE
FILE= ./testcase1.test
>DEBUGFILE
FILE= ./heat sensor.dbg
```

Workspace in ProjectFiles

Project file saves last snapshot of the workspace. When user opens an existing project, GUI will be modified according to these settings. This section begins with WORKSPACE_BEGIN keyword and ends with WORKSPACE_END keyword. Between the keywords states of all the views and windows are saved.View properties, i.e. visibility of toolbars, shortcuts, etc. are leaded with "VIEW_" tag and window properties, i.e. subwindows which were open just before leaving workspace, are leaded with "WINDOW_" tag. Editor windows are special cases since they require additional information like the file they are editing. There is an editors section in the workspace between "WINDOW_EDITOR_LIST_BEGIN" keyword and "WINDOW_EDITOR_LIST_END" keyword. In this section a mode tag is followed by a file path.Below is an example of workspace.

```
WORKSPACE_BEGIN
VIEW_TOOLBAR_DEBUG= OFF
VIEW_BUTTON_DEBUG_STEP= ON
...
(removed)
...
WINDOW_EDITOR_LIST_BEGIN
FULL= NONE
FLOATING= ./source/heat sensor.asm++
MINIMIZED= ./testcase1.test
WINDOW_EDITOR_LIST_END
...
removed
...
WINDOW_BUTTOM_CONSOLE= TABBED
WINDOW_BUTTOM_LOG= ON
WINDOW_SIDE_WATCHPOINT= TABBED
WINDOW_SIDE_REGISTERS= ON
WORKSPACE_END
```

7.2. Debug File Format

Debug files hold data of the source and binary executable files that will be used in debugging process. Debugger needs watchpoints and breakpoints to halt execution. Watchpoints are held as register adresses and breakpoints as line number of some source file. Debug file holds existing watchpoint and breakpoint locations in a file <project_name>.dbg. Below are the specifications and format of the debug file.

Cross Mappings of the Line Numbers for Breakpoints

Breakpoints are defined using source files. These lines should be mapped to corresponding lower level file lines.Breakpoints may be lying in different files so each files line number is seperated from another. Breakpoint section begins with BREAKPOINT_BEGIN keyword and ends with BREAKPOINT_END keyword. After BREAKPOINT_BEGIN keyword, the path of the file to which source file line numbers are mapped is saved. This file is usually a generated asm file with file name <project_name>_g.asm. Each source file's breakpoint data is listed under its path name leaded with its source type. After each file, END_OF_BP_LIST keyword is used to indicate the source file has no other breakpoints. Each breakpoint is indicated with a >BP tag followed by line number of the associated source file and mapped line number.Other mappings simply don't have any tags. Below is an example of breakpoint section.

```
BREAKPOINT_BEGIN DEST= ./heat sensor_g.asm
ASMFILE= ./source/heat sensor.asm++
4 1
5 2
...
...
11 11
>BP 12 14
13 16
...
...
45 50
>BP 46 55
...
...
81 90
END_OF_BP_LIST
ASMHEADER= ./myLib/a2dcalculate.ah
1 91
2 94
3 95
>BP 4 98
...
...
>BP 19 122
...
...
>BP 24 130
...
...
END_OF_BP_LIST
BREAKPOINT_END
```

Register Adresses for Watchpoints

Watchpoints are defined using registers of the microcontroller. They are mapped to a real address value in the PIC and debugger halts whenever a register referenced by a watchpoint is altered. Debugger receive line number information to continue debugging process from simulator. Watchpoint section begins with WATCHPOINT_BEGIN keyword and ends with WATCHPOINT_END keyword. Each watchpoint is indicated with a >WP tag followed by register address of PIC in hexadecimal format. Some special registers are indicated with descriptive labels such as stack registers. Below is an example of watchpoint section.

```
WATCHPOINT_BEGIN
>WP 0x0101
>WP STACK1
>WP W
>WP STATUS
WATCHPOINT_END
```

8. Coding Standarts

8.1. Coding Conventions

To increase maintainability of the source code, all project members will obey the coding standarts described below

Inside the class scope, attributes and method declarations should be followed with method definitions. Attributes and method declarations shall be logically grouped using appropriate comments.

Class attributes should be private. All attributes must have its own getter and setter methods implemented.

8.2. Naming Conventions

Naming conventions will be as Java naming conventions.

Class names will be as descriptive as possible and initial letters of each word and abrreviation letters will be capitalized. Example: Class, ClassName, CClass, ClassC etc.

Method names and Class attributes always start with small letters. Each word or abrreviation letter after the first word or abbreviation letter will be start with capital letters. Example: var, varP, varPoint, iPoint, varFirstSecond, varFS, method(), methodName(), mName(), methodN() etc.

8.3. Comments

Comment conventions will be as Java commenting conventions.

At the beginning of each file, there will be a descriptive comment which must include file name, creator, creation time, last edit date.

Classes, attributes and methods should be leaded with descriptive comments.

Class comments should describe functionality of the class and may include special notes if any. The comment should have @author <author name> line in the end.

Attribute comments should be brief as much as possible.

Method comments should describe behavior and aim of the method. All parameters should be described using @param tag and return values should be described with @return. The comment should have @author <author name> line in the end. Local variables should be described inside the method.

8.4. Indentation

Indentation conventions will be as Eclipse Java Indentation conventions.
Scope defining curly braces should be put in a new line and indented to the same vertical line. Example:

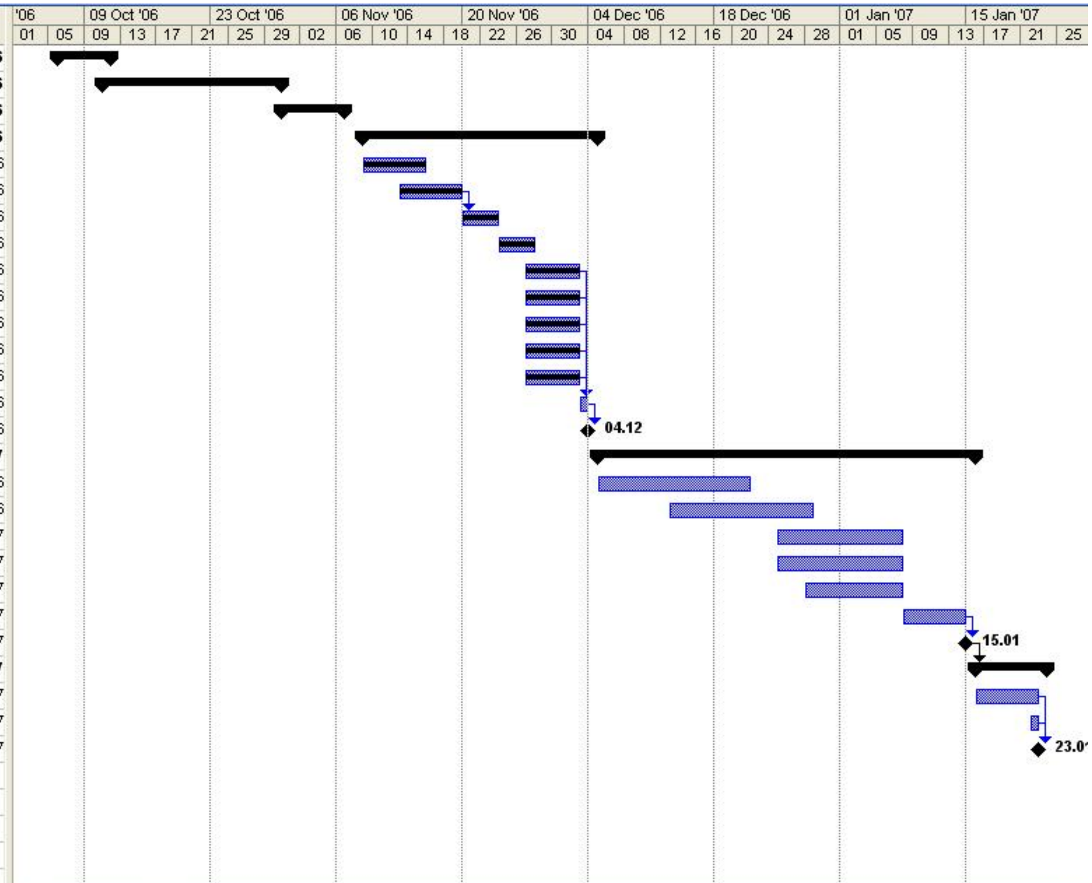
```
Class Class1
{
        void method ()
        {
                if ( var1 == var2 )
                {
                        if ( var2 == var3 )
                        {
                                ...
                        }
                }
        }
}
```

To increase readibility, there should be white spaces before and after any names, operators, etc. Example:

```
var = 3 + ( var1 + var2 * var3 / method() );
```

9. Gantt Chart

ID	Indicators	Task Name	Start	Finish
1		**Project Proposal**	**Fri 06.10.06**	**Wed 11.10.06**
4		**Literature Survey**	**Wed 11.10.06**	**Mon 30.10.06**
9		**Requirement Analysis**	**Tue 31.10.06**	**Mon 06.11.06**
18		**Initial Design Report**	**Thu 09.11.06**	**Mon 04.12.06**
19	✓	Use cases	Thu 09.11.06	Wed 15.11.06
20	✓	Defining Structures and Hierarchies	Mon 13.11.06	Sun 19.11.06
21	✓	Preparation of Class Relationship Diagra	Mon 20.11.06	Thu 23.11.06
22	✓	Preparation of State Transition Diagrams	Fri 24.11.06	Mon 27.11.06
23	✓	System Design	Mon 27.11.06	Sat 02.12.06
24	✓	Object Design	Mon 27.11.06	Sat 02.12.06
25	✓	Human Interface Design	Mon 27.11.06	Sat 02.12.06
26	✓	Data Management Design	Mon 27.11.06	Sat 02.12.06
27	✓	Task Management Design	Mon 27.11.06	Sat 02.12.06
28		Initial Design Report Preparation	Sun 03.12.06	Sun 03.12.06
29		Initial Design Report Deadline	Mon 04.12.06	Mon 04.12.06
30		**Final Design Report**	**Tue 05.12.06**	**Mon 15.01.07**
31		Revising Use cases	Tue 05.12.06	Thu 21.12.06
32		Preparation of Class Diagrams	Wed 13.12.06	Thu 28.12.06
33		Architectural Design	Mon 25.12.06	Sun 07.01.07
34		Graphical User Interface Design	Mon 25.12.06	Sun 07.01.07
35		Sequence Diagrams and Timing Relation:	Thu 28.12.06	Sun 07.01.07
36		FinalDesign Report Preparation	Mon 08.01.07	Sun 14.01.07
37		Final Design Report Deadline	Mon 15.01.07	Mon 15.01.07
38		**Prototype**	**Tue 16.01.07**	**Tue 23.01.07**
39		Prototype Preparation	Tue 16.01.07	Mon 22.01.07
40		Presentation Preparation	Mon 22.01.07	Mon 22.01.07
41		Demonstration Deadline	Tue 23.01.07	Tue 23.01.07



Ready