**EB463**

# Mobile Phone Solution Teacher's notes

1. Introduction ....4
1.1 General introduction ....4
1.2 Teacher's Introduction ....4
1.2.1 Prerequisites ....4
2. Exercises and their structure ....5
2.1 Exercises ....5
2.1.1 RS232 Component Properties ....5
2.2 Further work ....6
2.3 Sample programs ....7
2.3.1 Audio examples ....8
2.3.2 SMS (text messaging) examples ....8
3. Further information ....9
3.1 E-blocks hardware ....9
3.2 EB-230 CD ROM ....9
3.3 GSM Information ....9
3.4 Software ....9
4. Contents ....10
4.1 What else is included? ....10
5. Setting up your mobile phone ....11
5.1 Positioning E-Blocks ....11
5.2 Connections - assuming a PIC16F877A is used ....11
5.3 Setting up the GSM modem ....11
5.4 Setting up the PICmicro ....12
5.4.1 Using the SPI and RS232 boards together ....12
6. Flowcode ....13
7. Using the mobile phone for teaching ....14
7.1 Motivational platform for learning microcontroller programming ....14
7.2 C and ASM users ....14
8. Exercise 1: A basic telephone ....16
8.1 Introduction ....16
8.2 Theory ....16
8.2.1 Commands ....16
8.2.2 ASCII characters ....16
8.2.3 Flowcode RS232 component - SendRS232char ....16
8.2.4 Strings ....18
8.3 Objectives ....19
8.4 Requirements ....19
8.5 Implementation ....20
8.5.1 Macro - Tx_Command ....20
8.5.2 Main program ....21
8.6 Learning outcome ....22
8.7 Further work ....22
9. Exercise 2: A simple 'State Machine' ....23
9.1 Introduction ....23
9.2 Theory ....23
9.2.1 State machine ....23
9.3 Objectives ....24
9.4 Requirements ....24
9.5 Implementation ....25
9.6 Learning outcome ....26
9.7 Further work ....26
10. Exercise 3: Modem responses ....27
10.1 Introduction ....27
10.2 Theory ....27
10.2.1 Flowcode LCD component ....27
10.3 Objectives ....28
10.4 Requirements ....28
10.5 Implementation ....29
10.5.1 Message characters ....29
10.5.2 Echoed characters ....29

10.5.3 Tests ........ 29
10.6 Learning outcome ........ 30
10.7 Further work ........ 30
11. Exercise 4: Listening to messages ........ 31
11.1 Introduction ........ 31
11.2 Theory ........ 31
11.2.1 Message reception ........ 31
11.3 Objectives ........ 32
11.4 Requirements ........ 32
11.5 Implementation ........ 33
11.5.1 Message detection ........ 33
11.5.2 Message interval timer ........ 33
11.5.3 IDLE state modifications ........ 34
11.5.4 RINGING state modifications ........ 35
11.6 Learning outcome ........ 36
11.7 Further work ........ 36
12. Exercise 5: Automatic call handling ........ 37
12.1 Introduction ........ 37
12.2 Theory ........ 37
12.3 Objectives ........ 37
12.4 Requirements ........ 37
12.5 Implementation ........ 38
12.6 Learning outcome ........ 40
12.7 Further work ........ 40
13. SMS introduction ........ 41
14. Exercise 6: Send a text message ........ 42
14.1 Introduction ........ 42
14.2 Theory ........ 42
14.2.1 Message format ........ 42
14.2.2 Send a message ........ 42
14.2.3 Rx_Message macro modifications ........ 43
14.2.4 Tx_Command macro modifications ........ 43
14.3 Objectives ........ 44
14.4 Requirements ........ 44
14.5 Implementation ........ 45
14.6 Learning outcome ........ 47
14.7 Further work ........ 47
15. Exercise 7: Receive a text message ........ 48
15.1 Introduction ........ 48
15.2 Theory ........ 49
15.3 Objectives ........ 49
15.4 Requirements ........ 50
15.5 Implementation ........ 51
15.6 Learning outcome ........ 52
15.7 Further work ........ 52
16. Exercise 8: Automatically respond to a text message ........ 53
16.1 Introduction ........ 53
16.2 Theory ........ 53
16.2.1 Message decoding ........ 53
16.2.2 Response transmission ........ 53
16.3 Objectives ........ 54
16.4 Requirements ........ 54
16.5 Implementation ........ 55
16.6 Learning outcome ........ 56
16.6 Learning outcome ........ 57
16.7 Further work ........ 57

# 1. Introduction

## 1.1 General introduction

The aim of this E-Blocks Solution kit is to provide a practical introduction to modern communications systems design and technology.

Learning is motivated by the rapid achievement of impressive results, and by the relevance of the results to current, popular culture. Successful completion of the course will provide an increased understanding of the operation of the underlying technology, illustrate the role of the microcontroller in modern devices, and be a basis for the development of additional applications.

By the end of this course the student will have been guided through programming examples resulting in the development of:

- a range of fully functional mobile phone devices.
- a range of text message controlled devices,

All the examples have the scope for further development and enhancement using the acquired Flowcode programming skills, additional documented Modem functions, and a little imagination.

Completion of all the exercises will require up to 20 hours of work – depending on how much information you give to students: see below.

## 1.2 Teacher's Introduction

These Teacher's notes are designed to introduce you to the concepts required to understand GSM modems and also to provide practical exercises with which to develop your skills as well as those of your students.

The course is structured into a number of sections that first take students through the basics of sending AT commands and creating a working telephone, then into intermediate topics such as text messaging and receiving data. The course also deals with some more advanced topics, including automated functions. Examples and suggested work are provided as a basis for developing demonstrations and practical activities for your students.

This course is carried out using Flowcode, a graphical programming language, which is supplied on CD ROM. Flowcode allows students to construct programs for microcontrollers at a systems level: it allows them to learn how the system works without getting bogged down in the detail of C or Assembly code. For communications this is particularly relevant – it allows students to see the sequence and content of message transfer in flow diagrams, and greatly facilitates learning.

### 1.2.1 Prerequisites

Students will need some programming experience to use this equipment. If students have already completed a course in C or Assembly code programming then they should immediately be comfortable with using Flowcode as a programming tool.

Students who have not undertaken a course in programming should work their way through the "Introduction to Microcontroller Programming" CD. This will take up to 50 hours depending on the level of student and their experience. A copy of the "Introduction to Microcontroller Programming" CD is included with the mobile phone solution.

## 2. Exercises and their structure

### 2.1 Exercises

A number of exercises have been provided to allow experience to be gained with both the assembled hardware and the Flowcode software.

The exercises have been divided into two main sections: One covering communication basics and audio topics, the other covering SMS topics. In each section the initial exercises have been made as simple as is practicable, while introducing the important issues relevant to further development. Later exercises build on the knowledge gained to provide examples of useful applications for the technology.

Exercises are split into two parts: the exercise itself and also a short section on implementation. You need to make a decision as to whether to give your students the implementation part of the exercise. This also gives you some opportunities for differentiation between different abilities of student. Without the implementation section students will also be required to plan their work more carefully, and it will take them longer to complete the tasks.

A Flowcode solution for each exercise is provided on the accompanying CD Rom; these can also be used to provide demonstrations of the technology and be the basis for further development.

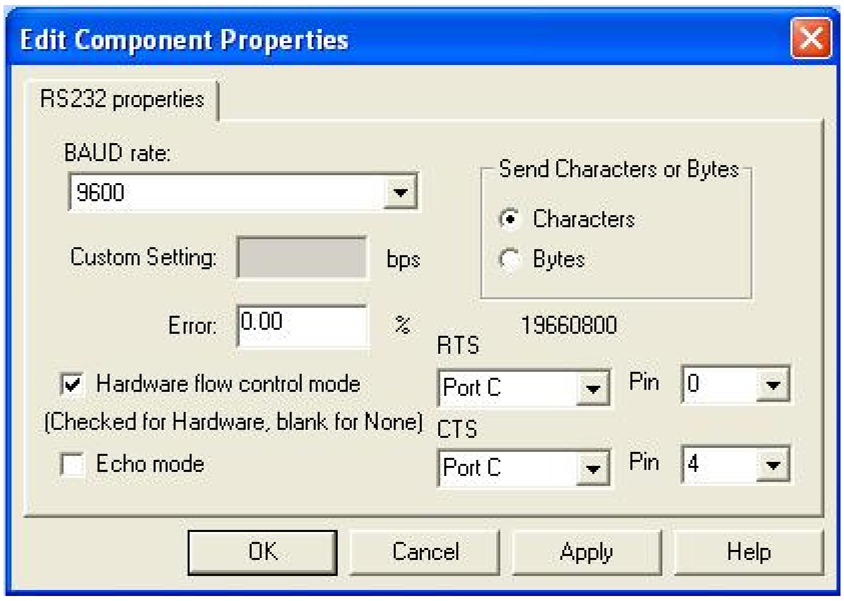
In addition to the general requirements for project planning and software writing, application development for the GSM modem is particularly dependent on two main topics:

- RS232 communications - the hardware and software considerations for transmission and reception of data.
- String manipulation - the creation and interpretation of sequences of text characters.

To assist with the management of the GSM modem communications, two software macros (subroutines) will be developed to enhance the functionality of the Flowcode RS232 component.

#### 2.1.1 RS232 Component Properties

The Flowcode RS232 component includes a properties section that allows the communication baud rate and flow control to be configured. The settings illustrated below should be used for all the exercises.


Edit Component Properties

RS232 properties

BAUD rate: 9600

Custom Setting: bps

Error: 0.00 %

19660800

Send Characters or Bytes: [✓ Characters] Bytes

[✓] Hardware flow control mode
(Checked for Hardware, blank for None)

[ ] Echo mode

RTS Port C Pin 0

CTS Port C Pin 4

BAUD rate
The number of data and formatting bits transmitted or received per second

Hardware flow control mode
This allows both the controller and the modem to suspend the transmission of data to them until they are ready to handle it. The RTS and CTS lines are controlled by the controller and modem respectively, and used to signal their ability to receive data.

## 2.2 Further work

The exercises are intended to introduce important topics, supply relevant information, and reinforce the learning process by the development of working hardware and software solutions. Example solutions for each exercise are supplied on the accompanying CR-ROM. In each case there is scope for improvement and further development and these, or the student's own solutions could be used as the starting point for further development and demonstration. Topics for further development are suggested at the end of each exercise.

The documentation for the GSM modem contains information regarding a range of features that are not utilized in the supplied exercises. With the experience gained from this course, the more advanced student should be capable of incorporating these features into future developments.

*Note:*
Some modem commands are part of the GSM specification and must be implemented fully, or partially, by all GSM devices. Others may be manufacturer or model specific and should be used with care if compatibility with other devices is required. The supplied documentation provides details of each command supported.

## 2.3 Sample programs

Several sample programs are available for use with this guide, each one representing a possible solution to one of the exercises. They are split into three groups

- The basics of communications between the controller and the GSM modem
- audio applications
- SMS text messaging.

The sample programs contain a number of Flowcode macros that provide assistance when completing the exercises. Sample programs are only available with the CD ROM EB230.

*Important!*
Most of the exercises require the availability of a donor mobile phone to provide connectivity. Some of the programs dial a pre-programmed number when operating. A dummy number is programmed into the software before shipping, but this number must be changed to the number of the donor phone before compiling the software.

### 2.3.1 Audio examples

**Tutorial program Phone_01.fcf**
This program shows how you can use Flowcode to send strings of characters using the RS232 component; specifically the AT commands required to control the GSM modem and replicate a simple telephone.

**Tutorial program Phone_02.fcf**
This is a more complex program that introduces a state machine model to improve the operation of the telephone.

**Tutorial program Phone_03.fcf**
The original state machine model lost the ability to answer an incoming call. This program displays the messages being generated by the modem during operation, and introduces methods required to handle them as the solution to the problem.

**Tutorial program Phone_04.fcf**
This program completes the operation of the basic telephone; including detection of the modem ringing.

**Tutorial program Phone_05.fcf**
The final audio-based example used a selection of modem messages to automate the call answer and hang-up processes to produce a remote listening or public address device.

### 2.3.2 SMS (text messaging) examples

**Tutorial program Phone_06.fcf**
This program shows how you can use Flowcode to send a simple text message at the touch of a button.

**Tutorial program Phone_07.fcf**
This is a more complex program that shows how you can use Flowcode to receive a simple text message, extract the massage from the accompanying information, and display the text on a screen.

**Tutorial program Phone_08.fcf**
The final program extracts data from a received text message and automatically replies with a response message

The final programs in each section form an aiming point for more advanced students - or you can use it as a group project where each student develops part of the program.

# 3. Further information

### 3.1 E-blocks hardware

Datasheets for all E-blocks printed circuit boards are available from:
http://www.matrixmultimedia.com

You will be provided with a small CD ROM marked 'ELSAM'. This includes a driver for your EB006 Multiprogrammer board.

### 3.2 EB-230 CD ROM

Your package contains a CD ROM marked EB230. This should include:

- A Word version of this document
- A PDF version of this document
- Example files to accompany this document.

### 3.3 GSM Information

This course is not designed around a specific GSM module, but one will be included in the package. There should also be an accompanying CD ROM with data specific to this GSM module, such as a technical datasheet and the module's AT command set.

### 3.4 Software

For each of the more complex E-blocks a programming strategy guide is available. This document gives an outline of how each E-block can be programmed for C or assembly code programmers. These strategy guides are available on our web site. Flowcode users should not need these documents.

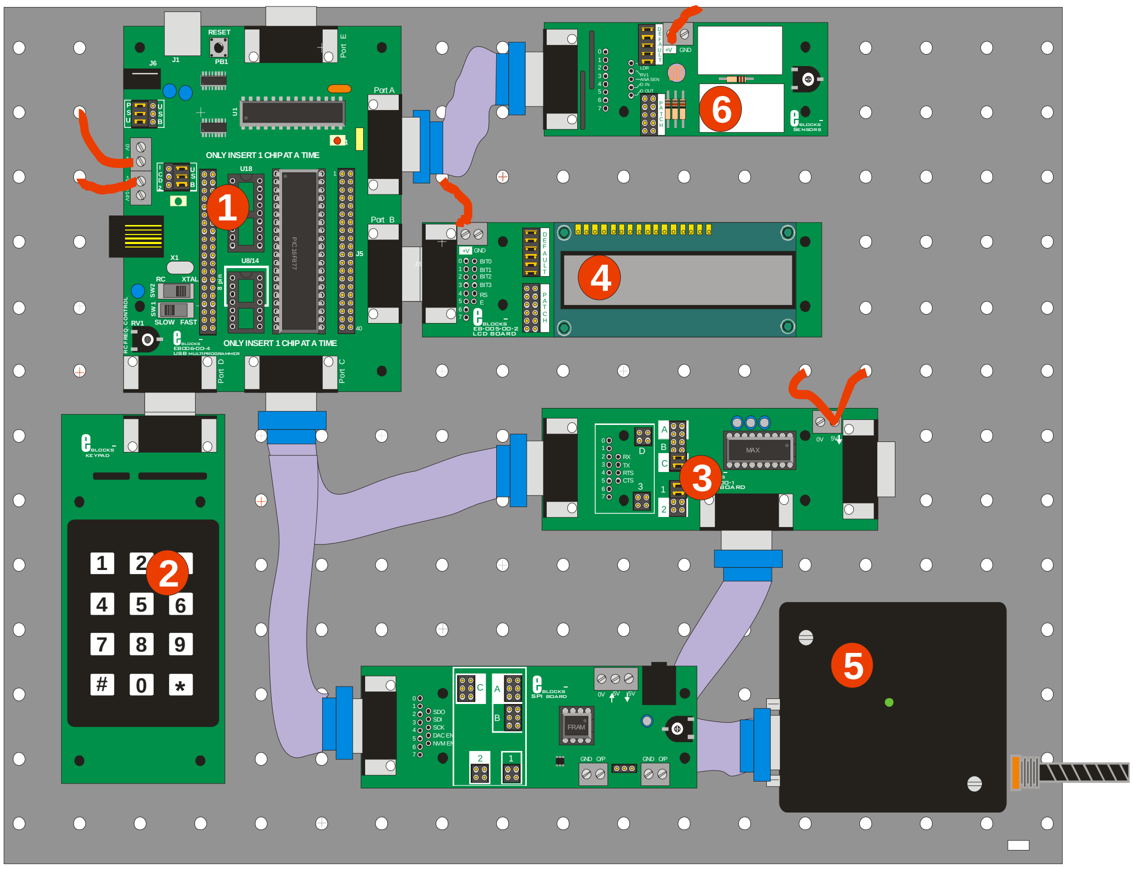
To be able to use Flowcode you will need to familiarise yourself with the basics. Read the sections in the help file on adding icons, and components, and work through the first couple of tutorials just to get the hang of how Flowcode works.

You can also use the "An Introduction to Microcontroller Programming" CD, which is included in this solution. It is designed to take users from the absolute basics through to quite advanced topics, and is a useful resource.

If you are programming in C or assembly then you may benefit from one of our CD ROM courses for programming PICmicro microcontrollers.

## 4. Contents

The Mobile phone solution can be seen here in graphical format:



(1) The PICmicro multiprogrammer with 5 ports - A to E.
(2) The keypad E-Block is connected to port D
(3) The RS232 E-Block is connected to port C
(4) The LC-Display is connected to port B
(5) The GSM terminal unit is connected to the Modem port of the RS232 E-Block
(6) The Sensor E-Block is connected to port A
(7) The SPI / DAC E-Block is also connected to port C using a splitter cable

Please refer to the individual E-Blocks datasheets for further details of connection and configuration. Boards 3, 4, 6 and 7 require a 5V power source to be connected through from the +V outputs on the Multiprogrammer.

### 4.1 What else is included?

With your solution you will receive a check sheet showing all the parts of your package. This will show you what E-blocks are included and will also show you what additional items are in the package. Please make sure you have all these items.

# 5. Setting up your mobile phone

## 5.1 Positioning E-Blocks

The mobile phone uses the following core E-blocks: Multiprogrammer, keypad, LCD Display, RS232 board, and a GSM terminal unit (modem). Please refer to E-Blocks User Guide for general details about securing and connecting E-Blocks. If your mobile phone system is not pre-wired then use the images in this document as a rough guide to where to place the components on the backplane.

## 5.2 Connections - assuming a PIC16F877A is used

LCD display (EB-005) on Port B
RS232 board (EB-015) on Port C via a ribbon cable connector
Keypad (EB-014) on Port D
GSM terminal unit connected to the modem connection on the RS232 board via a ribbon cable connector

Sensor board (EB-003) on Port A
SPI board on Port C with the Dual IDC splitter cable

*Note:*
The LCD display, RS232 board, SPI board and Sensors board require connecting to +V supply on the multiprogrammer.

*Note:*
If you are using an Atmel / ARM multiprogrammer then the arrangement of E-Blocks will change. Note also that you will need to use the patch system for the SPI and the RS232 board to ensure that the USART RX and TX pins connect to the appropriate pins on the RS232 and SPI board respectively. Please refer to the E-Blocks datasheets for details.

## 5.3 Setting up the GSM modem

The GSM modem is the heart of the mobile phone system. It can send and receive signals via the aerial, and can communicate to other systems using RS232 communication protocols.

*IMPORTANT!*
You need an active SIM card.

The GSM modem requires a mobile phone SIM card with credit on it. Just like any other mobile phone the GSM modem needs a SIM card to work. You will need to purchase a SIM card from the same place that you would purchase one for a normal mobile phone.

*Note:*
Contract SIM cards will not work due to data encryption, so you will need a "pay-as-you-go" SIM card.

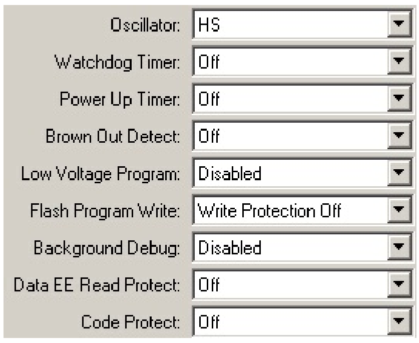
You will also need to ensure that there is sufficient credit on the SIM card for the messages you will be sending. SIM card credits can be topped up in the same way as topping up a normal mobile phone.

## 5.4 Setting up the PICmicro

The experiments in this course are designed for use with a PIC16F877A PICmicro chip on the multiprogrammer board, although a number of other devices could potentially be used.

A crystal clock is used with a frequency of 19.6608MHz as this allows the required BAUD rate of 9600 to be easily generated. Within the set up software - PPP - you need to specify HS oscillation mode. All other features in the configuration word need to be disabled or turned off.

The required configuration word in the PPP utility is shown below.



| | |
|---|---|
| Oscillator: | HS |
| Watchdog Timer: | Off |
| Power Up Timer: | Off |
| Brown Out Detect: | Off |
| Low Voltage Program: | Disabled |
| Flash Program Write: | Write Protection Off |
| Background Debug: | Disabled |
| Data EE Read Protect: | Off |
| Code Protect: | Off |

An RS232 E-Block is required. This needs to be attached to Port C of the '877A as this contains the USART module within this PICmicro.

The mobile phone includes an SPI board. The SPI board needs to be connected to Port C as this also contains the SPI communication module within the 16F877A.

As both the RS232 and SPI need to be on Port C the equipment is shipped with an EB635 Dual IDC splitter cable that allows you to connect both the RS232 board and the SPI board to Port C.

### 5.4.1 Using the SPI and RS232 boards together

The default DAC enable and NVM (Non Volatile Memory) enable outputs are set to the same pins as the RS232 uses to transmit and receive. To avoid potential problems you are advised to change the pins to others not used by the RS232 board, such as pins C1 and C2. These can be changed on the SPI components property page. You can use the jumpers in block 2 to enable the patch system, and wires to connect the DAC EN and NVM EN to the chosen pins.

***Note:***
Whilst the SPI board is connected the patch system links are not made.

For further information see the section on Configuring the PICmicro in the Flowcode help file.

For details on PICmicro devices please refer to the appropriate device datasheet from Microchip.

# 6. Flowcode

If you are using Flowcode then it will need to be installed onto your local computer. To do this you have to follow the instructions inside the DVD case.

You will need to register your copy of Flowcode within 30 days of installing it. To do this simply click the visit website button when starting the software and fill in your details. You will then receive a registration code within a few days, which will permanently unlock the software.

Sample Flowcode mobile phone programs are available on the CD supplied with the mobile phone solution. The CD is labelled with the part number EB230.

## 6.1 Testing your mobile phone and E-blocks

The mobile phone can be tested by downloading the simple telephone program as detailed in Exercise 1 below. This requires Flowcode and the sample Tutorial program Phone_01.fcf.

Each individual E-block has a separate test routine that you can carry out if you feel the E-block is not working correctly. The test routines are based on PICmicro code and are available from www.matrixmultimedia.com. The associated technical datasheet describes what the test routine does.

# 7. Using the mobile phone for teaching

You have two choices here:

### 7.1 Motivational platform for learning microcontroller programming

Modern students are no longer satisfied by simply programming an LED to come on or off, or making counter sequences. These students want to learn electronics in a modern context and want to learn how the devices they use everyday work. As most students have mobile phones, and spend a large amount of time using them, it makes sense to use a mobile phone as a basis for learning electronics.

You can use the mobile phone as a way of introducing topics within microcontroller programming: for example you could set up an example file that sends a text message using the mobile phone. You could then introduce the concept of an A/D converter for telemetry use and then ask the students to develop programs that teach them how A/D converters inside the mobile phone works. We do not specifically cater for this use – but you should easily be able to develop some variants of the examples we have provided which will suit your purposes here.

### 7.2 Using the mobile phone to teach communications systems

It is more likely that you will want to follow our course which is designed to teach students how communication systems are built. The course is well structured and has several learning outcomes that can be split into three groups:

Programming outcomes:
- Keypad control
- LCD control
- RS232 protocol and programming
- String construction and deconstruction in communications
- The use of state machines in controlling electronic systems

Communications outcomes:
- RS232 communications and handshaking protocols
- ASCII representation of characters in messages
- AT command structure and command protocols used in telecommunications
- Sending and receiving text messages in mobile phone systems
- Modem control and messaging
- Understanding signal strength

Project management and development outcomes
- The use of flowcharts and state diagrams in planning systems
- How electronic systems are developed from scratch
- The modular approach to building electronic systems

This course will teach students the basic systems theory that most communications systems are based on. Students undertaking the full course will gain a tremendous amount of satisfaction from the course and will have developed a fully functioning mobile telephone with full voice and text messaging capabilities.

### 7.2 C and ASM users

The mobile phone can be used as a motivational tool for C and Assembly users too, however you may need to develop some core routines for your students that will get them started.

Note that you can also use Flowcode for C and Assembly users in the first part of their course: the programming strategy for microcontrollers is the same independent of which language is used. Flowcode allows students to learn programming strategy without getting bogged down in problems of syntax and these skills are easily transferred to C and assembly later.

C and assembly users may also want to get involved in understanding mobile telephony in more detail. In particular they will be interested in AT command structures, how they are used in mobile phones, and how they are implemented using C programming. There is some information on AT commands below, but these students are likely to want to use the GSM terminal datasheet for further work such as SIM card operations and embedded phone systems.

# 8. Exercise 1: A basic telephone

## 8.1 Introduction

In this exercise a simple telephone will be developed that is capable of dialling a pre-loaded number, answering an incoming call, and terminating a connection (hanging-up). The functions will be manually controlled using individual buttons on the keypad module.

## 8.2 Theory

A telephone must be able to perform three basic control functions:

1. Make a connection to another selected telephone.
2. Accept a connection from another telephone.
3. Disconnect from a remote telephone.

### 8.2.1 Commands

The GSM Modem allows these functions to be performed through the provision of three simple AT commands:

| | |
|---|---|
| • ATD<number>; | (Dial) |
| • ATA | (Answer) |
| • ATH | (Hang-up) |

The commands are transmitted to the modem via the RS232 board. The information is transmitted one character at a time using codes from the ASCII character set.

### 8.2.2 ASCII characters

The ASCII character set uses individual numeric values to represent each of the alphabetic characters (upper and lower case), numeric digits, punctuation marks, and control codes.

Useful ASCII codes are:

| | | |
|---|---|---|
| <LF> | 10 | Line Feed |
| <CR> | 13 | Carriage Return |
| <CTRL-Z> | 26 | End Of File |
| 0 - 9 | 48 - 57 | |
| A - Z | 65 - 90 | |
| a - z | 97 - 122 | |

### 8.2.3 Flowcode RS232 component - SendRS232char

The Flowcode RS232 component must be loaded into the program in order to gain access to the RS232 functions.

One of the functions provided by the RS232 component is SendRS232char. This allows the transmission of a single character from the RS232 port.

All the modem commands consist of multiple characters. Each character in a command can be transmitted with an individual use of the SendRS232char function, but some commands consist of a large number of characters, and the result would be an excessively complex program. A useful starting point for this, and many other programs, is the development of a macro (subroutine) that allows groups of characters (strings) to be transmitted using a single command. In addition, all modem commands must be completed with the 'Carriage Return' character <CR>. This can be difficult to include in a string using normal, printable characters, but the macro can be written to add the character at the end of each sequence.

### 8.2.4 Strings

A string consists of a series of individual byte values at adjacent addresses, referenced with a single name.

- The values of the individual bytes in a string usually represent ASCII characters.
- A string is terminated with a byte set to the numeric value 0
- The Flowcode string manipulation functions operate on the entire array contents between the start address and the first 0 value.
- A string can also be treated as an array of bytes, allowing the individual locations to be accessed using an index pointer

An example of string addition illustrates the structure of a string and the way they can be manipulated:

**Name="Matrix"**

| Name | M | a | t | r | i | x | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| Name[] | 77 | 97 | 116 | 114 | 105 | 120 | 0 | | | | | | | | | | | | | |

Numeric calculations can include the individual contents of an array. The array is referenced using its name with the individual element referenced by an index value in square brackets.

Name[2] contains the character 't' which has the ASCII numeric value 116

**Name = Name + " Multimedia"**

(note the single space character before the "M")

| Name | M | a | t | r | i | x | | M | u | l | t | i | m | e | d | i | a | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| Name[] | 77 | 97 | 116 | 114 | 105 | 120 | 32 | 77 | 117 | 108 | 116 | 105 | 109 | 101 | 100 | 105 | 97 | 0 | | |

**Name** contains "Matrix Multimedia"

The 0 that terminated the original "MATRIX" string (index 6) is used as the start point for the addition; being overwritten in the process. The 0 at the end of the " MULTIMEDIA" string (index 17) becomes the terminator for the new compound string when the two are added.

A byte variable can be set to most ASCII character values by surrounding the character with single quotes

Char = 'A' is equivalent to: Char = 65

Most non-printable characters (<CR>, <LF> etc.) can only be read and written using their numeric values.

## 8.3 Objectives

- Establish an RS232 communication link between the microcontroller and the modem.
- Develop a macro to allow command strings to be sent to the modem.
- Use some of the microcontroller inputs to control the transmission of AT commands to the modem.
- Develop a Flowcode program that will cause the modem to behave as a telephone.

## 8.4 Requirements

- A multi-programmer board attached to a PC running Flowcode
- An RS232 board
- A keypad
- A GSM modem with an active SIM card and an audio headset attached
- A working telephone

## 8.5 Implementation

The main program should contain a string variable called NUMBER. A logic block, initializing NUMBER to contain the number of mobile phone being used in conjunction with this exercise, should be placed at the top of the main program; making it easy to locate and edit..

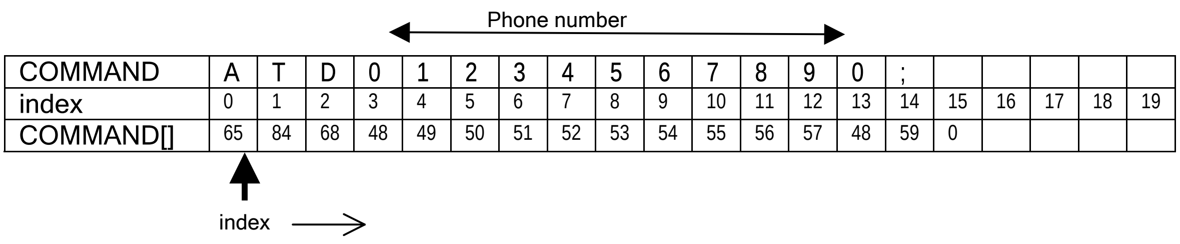
### 8.5.1 Macro - Tx_Command

The first part of the program to be developed should be a macro called Tx_Command. The main program should contain a string variable called COMMAND. When the macro is executed it will transfer each character in COMMAND to the SendRS232char function until it encounters a character with the value 0. When the 0 is reached the macro will transfer the value 13 (<CR>) to the SendRS232char function and return to the main program.

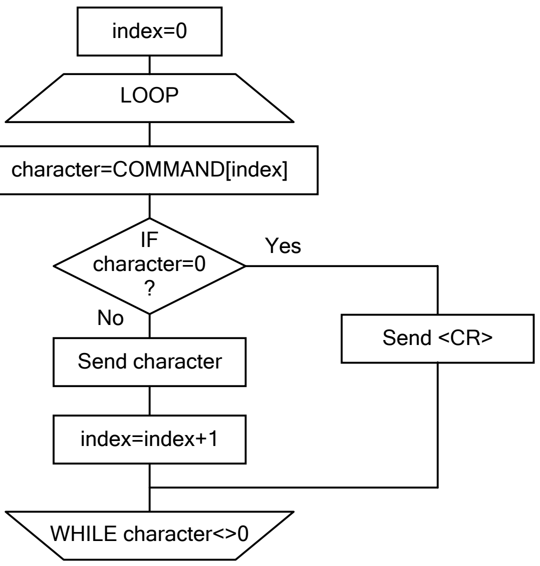
The command string to be transmitted must be loaded into COMMAND before executing the macro. The ATA and ATH commands are simple, but the ATD command must have the NUMBER string and a semi-colon added to it. E.g.

COMMAND="ATD"+NUMBER+';'

An example of a command string and the general operation of the Tx_Command macro is given below.


Phone number
index

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COMMAND | A | T | D | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | ; | | | | | |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| COMMAND[] | 65 | 84 | 68 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 48 | 59 | 0 | | | | |



### 8.5.2 Main program

This exercise requires the transmission of three message strings to be controlled by three buttons on the keypad.

Suggested key assignments:

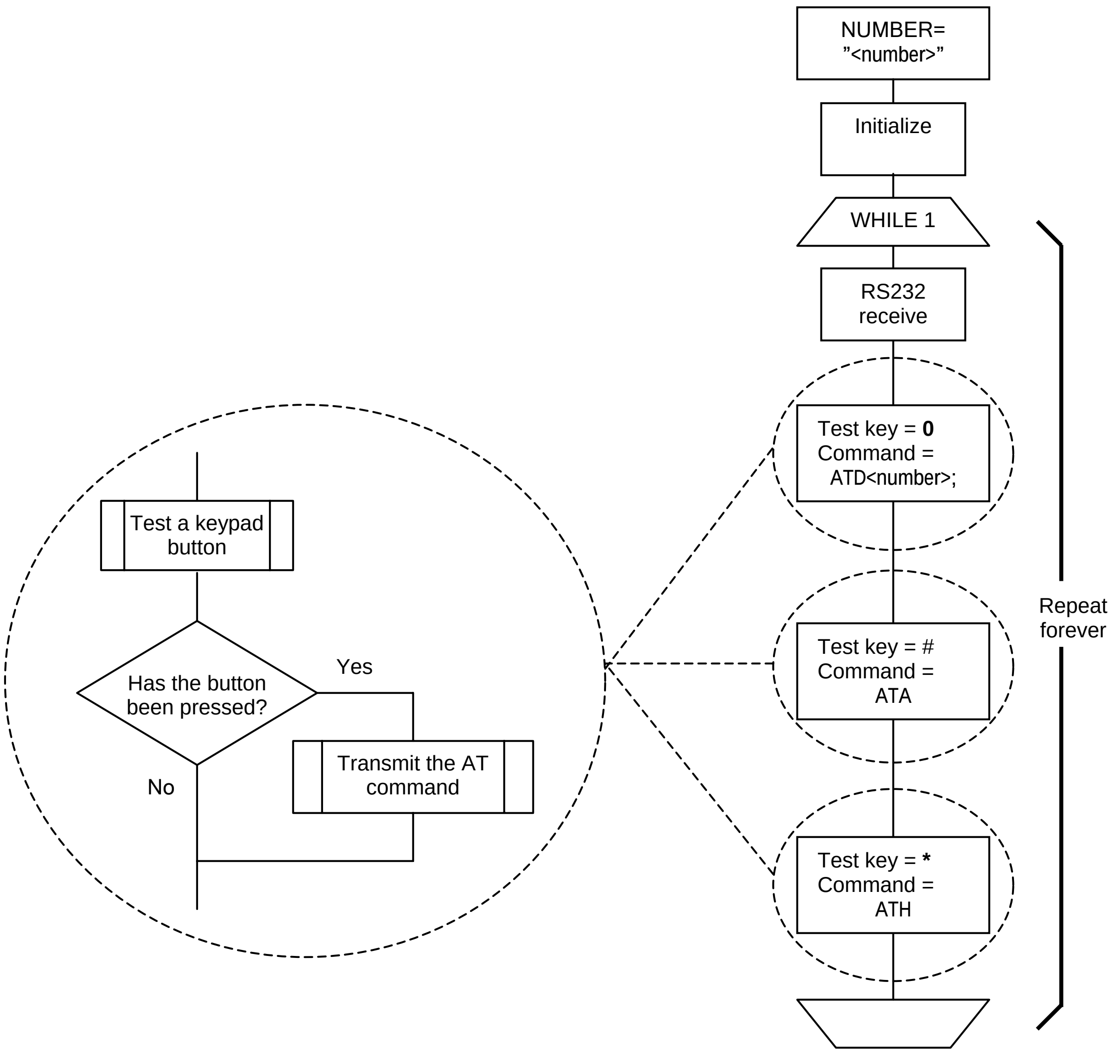
| Key | Function | Action |
|---|---|---|
| 0 | Dial number | Send the ATD<number>; command |
| # | Answer call | Send the ATA command |
| * | Hang-up | Send the ATH command |

The program should consist of three copies of the code section listed below, running in a continuous loop. Each copy should be configured to test one of the assigned keys and to transmit the associated AT command.

**Note:**

The 'RS232 receive' function is required because the modem transmits response messages during operation, and if these are not accepted, the modem transmit buffer will become full, preventing further communication.

Characters being transmitted by the modem can be discarded in this exercise.


NUMBER= "<number>"
Initialize
WHILE 1
RS232 receive
Test key = 0 Command = ATD<number>;
Test key = # Command = ATA
Test key = * Command = ATH
Repeat forever
Test a keypad button
Has the button been pressed?
Yes
No
Transmit the AT command

Telephone control structure

## 8.6 Learning outcome

On completion of this exercise, the student should be able to:

- Identify the role of each component in the system
- Configure and control the RS232 component.
- Configure and control the keypad component.
- Develop, debug, and download a Flowcode program.
- Create a Flowcode macro.
- Understand string storage and manipulation.
- Understand the transmission of AT commands.
- Understand the basic functions of a telephone.

## 8.7 Further work

The program can currently only dial a single, pre-loaded number. Expand the program to allocate different phone numbers to more of the numeric keys on the keypad module; creating a useful 'speed dial' phone.

# 9. Exercise 2: A simple 'State Machine'

## 9.1 Introduction

In the previous exercise a simple telephone application was developed. Although the resulting system was functional, it suffered from a number of drawbacks; one of the main ones being the ability to press the wrong button at the wrong time and cause the modem to behave unpredictably.

e.g.

Pressing the dialling key when a call is already in progress causes the modem to loose the existing connection and fail to obtain a new connection.

In this exercise the telephone functionality will be improved by introducing a state machine to monitor the current condition of the system and prevent inappropriate actions from being taken.

## 9.2 Theory

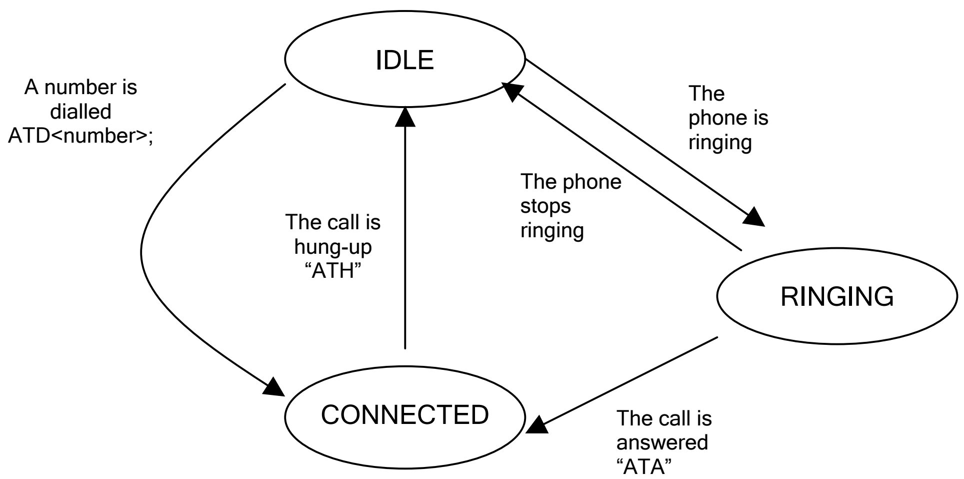
The operation of a telephone, as with most other devices, can be broken down into a set of states. Each state describes one of the conditions in which a device can exist. The current state can be changed, in a pre-defined way, by the effect of external or internal influences.

### 9.2.1 State machine

The telephone example can de defined by three simple states:

- IDLE - No calls are connected. The phone can detect incoming calls and will allow outgoing calls to be dialled
- RINGING - An incoming call is detected. The phone can allow the call to be answered.
- CONNECTED - An outgoing call has been dialled, or an incoming call has been answered. The phone can allow the call to be terminated.

It is not necessary to implement every possible path between states. In this case there is no direct path from CONNECTED to RINGING


IDLE
A number is dialled ATD<number>;
The phone is ringing
The phone stops ringing
The call is hung-up "ATH"
RINGING
CONNECTED
The call is answered "ATA"

Telephone state diagram

Each state allows a single action to be initiated:

| State | Action | Control | Command |
|---|---|---|---|
| IDLE | Dial a number | 1 key | ATD<number>; |
| RINGING | Answer a call | # key | ATA |
| CONNECTED | Hang-up | * key | ATH |

## 9.3 Objectives

- Develop a state machine that improves the functionality of the phone application from example 1.

Note:
In this example the ability to answer a call is temporarily lost. An improved function will be developed in the next exercise.

## 9.4 Requirements

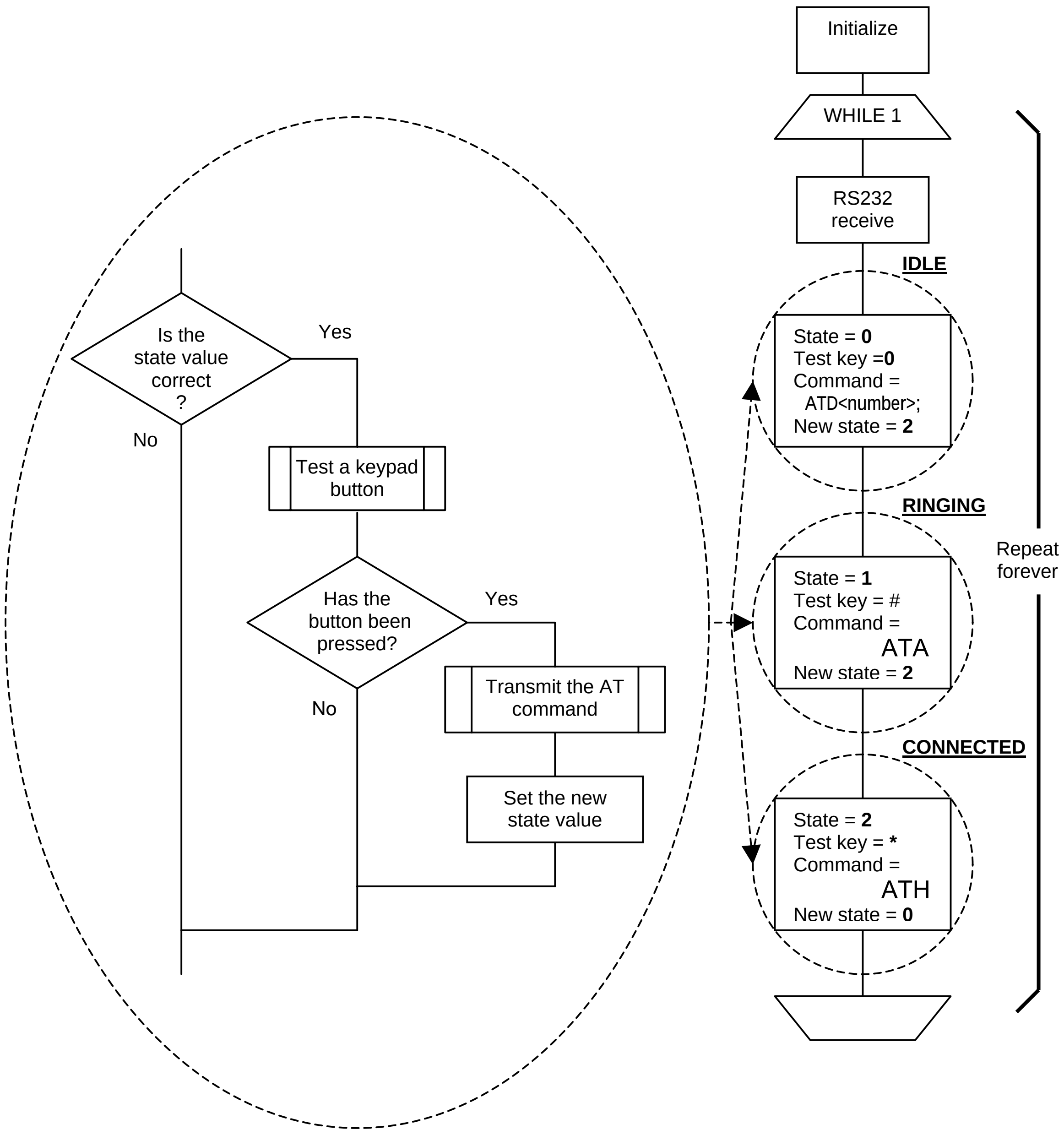
- A multi-programmer board attached to a PC running Flowcode
- An RS232 board
- A keypad
- A GSM modem with an active SIM card and an audio headset attached
- A working telephone
- A working solution to exercise 1, or a copy of Phone_01.fcf.

## 9.5 Implementation

Starting with the solution to exercise 1

- Add a state variable and initialize it to 0
- Use the state variable to decide whether to test each key
- Update the state variable if a command is transmitted

Suggested state values
0 = IDLE
1 = RINGING
2 = CONNECTED



Telephone control structure

### 9.6 Learning outcome

This exercise has concentrated on the structure of the software and the solution to some practical problems. In addition, the loss of the call answering function appears to be a backward step. But the work carried out forms the basis of the next exercise, and begins to demonstrate the advantages of structured code.

The lessons learned should include:

- The advantages of structured code
- The ability to define and implement simple state machines.

### 9.7 Further work

Work out the requirements for allowing the system to detect an incoming call.
Check the modem data sheet for possible solutions to the problem.

# 10. Exercise 3: Modem responses

## 10.1 Introduction

In the first two exercises communication with the modem has essentially been in one direction. Reception of characters from the modem has only been included in the programs to prevent the modem transmit buffer from filling and blocking further communication.

The program to be developed in this exercise will provide a display of the messages being transmitted by the modem, and allow subsequent software to be developed that can make use of them.

## 10.2 Theory

The modem provides a lot of useful information about its condition and the condition of any connection it is making. The ability to read and understand these messages will improve the functionality of any device developed to use them, and allow full automation of call connection and monitoring applications.

Modem messages are transmitted to the controller as strings of ASCII characters; similar to the commands being sent to it.

The program from the pervious exercise can be modified to display the information being transmitted by the modem during normal operation.

Characters can be transmitted by the modem for one of three main reasons:

1. **Echoed characters**: Every character transmitted to the modem is immediately transmitted back (echoed) to allow the controller to check the integrity of the communications.
2. **Response message**: The execution of each command will produce a confirmation response. This can be a simple "OK" message, the requested data, or an error message.
3. **Unsolicited messages**: The modem can transmit messages that are not responses to controller commands, but indicate changes in the condition of the modem or a connected call.

The typical response to a command that has been carried out correctly, and is not required to supply any data is:

<CR><LF>OK<CR><LF>

The LCD will be used to display the modem responses. To avoid confusion the messages will be split into two groups. Echoed characters will be displayed on the top line, and all other messages on the bottom line.

### 10.2.1 Flowcode LCD component

- The LCD component must be loaded into the program before it can be used.
- The LCD Start function must be executed before sending any other LCD commands.
- The LCD Clear function is used to ensure that the display area is blank and the cursor is at the start of the top line.
- Each character sent to the LCD is displayed at the current position. The cursor then moves one position to the right.
- The cursor can be sent to any position, on either line, using the Cursor function.
- The cursor is set to be invisible.
- The PrintASCII function can be used to send one character at a time.

### 10.3 Objectives

- Use the LCD component
- Receive and display the characters transmitted by the modem
- Categorise the characters.
- Develop a strategy to extract messages from the stream of characters and use them for their intended purpose.

### 10.4 Requirements

- A multi-programmer board attached to a PC running Flowcode
- An RS232 board
- A keypad
- A GSM modem with an active SIM card and an audio headset attached
- A working telephone
- A working solution to exercise 1, or a copy of Phone_02.fcf.

## 10.5 Implementation

### 10.5.1 Message characters

The ReceiveRS232char function that is already in use in the main program loop can be used as the source of the displayed characters received from the modem.

Add a variable called 'rx_char' to the program and use it to receive the value returned by ReceiveRS232char. - A value of 255 indicates that no character was received before the function timed out, so this value should not be sent to the LCD; any other value represents a received character and should be sent.

The LCD functions must be used to Start and Clear the LCD, and set the cursor position to the start of the bottom line, before entering the main program loop. This will allow all message characters to be displayed on the bottom line of the LCD. The echoed characters will be handled by the Tx_command macro and displayed on the top line of the LCD.

### 10.5.2 Echoed characters

The Tx_Command macro is a useful place to detect echoed characters. Use the LCD Clear function to clear the display before entering the main loop within the macro. This will clear the display and place the cursor at the start of the top line each time a command is sent to the modem.

Use the ReceiveRS232char function within the macro loop to receive one character from the RS232 port for every one sent, and use the LCD PrintASCII function to display it.

Use the LCD Cursor function to place the cursor at the start of the bottom line when all the characters have been sent, allowing the responses to be displayed on the bottom line.

### 10.5.3 Tests

1. Reset the controller board

    - Use the 0 key on the keypad to dial the pre-loaded number of the donor phone.
    - Take note of the echo characters on the top line of the display and the message characters on the bottom line - the extra blank characters are <LF> and <CR>, which cannot be displayed properly.
    - Terminate the call remotely by hanging up the donor phone.
    - Note the new message added to the bottom line of the display. It cannot be displayed fully but should be "NO CARRIER" - the space between NO and CARRIER is a genuine space, not a <LF>.
    - Hang up the call locally by pressing the * key
    - Note the new set of display characters.

2. Reset the controller board

    - Dial the modem from the donor phone.
    - Note the messages being produced on the bottom line of the display.

3. Reset the controller board

    - Turn the modem off and on again.
    - Note the message filling the bottom line of the display. This is an unsolicited message that is transmitted when the modem powers-up and could cause problems with other message checking functions.

## 10.6 Learning outcome

- The purpose of messages transmitted by the modem.
- The content and format of modem messages.
- A strategy for handling modem messages and making use of their contents.
- Use of the LCD component

## 10.7 Further work

Operation of the modem is enhanced by correct reception, interpretation and reaction to the information being sent by it to the controller.

Interpretation required the separation of echoed characters from message characters, and the separation of messages from the surrounding control characters.

- Develop a routine to flush all unexpected modem characters before starting the main program.
- Develop the Tx_Command macro to remove all echoed characters received by the RS232 port, leaving only message characters to be dealt with by the main program.

# 11. Exercise 4: Listening to messages

## 11.1 Introduction

The previous exercises left a significant gap in the functionality of the phone - the inability to answer an incoming call. This exercise will address this problem by developing a macro to detect the presence and persistence of the ‘RING’ message as the indication of an incoming call, moving the system between state 0 (IDLE) and state 1 (RINGING) automatically.

## 11.2 Theory

When the modem detects an incoming call it transmits the message “RING” at regular intervals (approximately 2 seconds). If the controller is able to detect this character sequence being received by the RS232 port, it can automatically control transitions between the IDLE and RINGING states. Both the IDLE and RINGING states must ‘listen’ for the “RING” message and perform the following actions:

IDLE

“RING” message detected:

- Initialize a timer with a period greater that the expected time between “RING” messages.
- Move the system to the RINGING state

RINGING

“RING” message detected:

- Re-initialize the timer with a period greater that the expected interval between “RING” messages - preventing a time-out while the messages are being received.

Timer timed out:

- Move the system back to the IDLE state

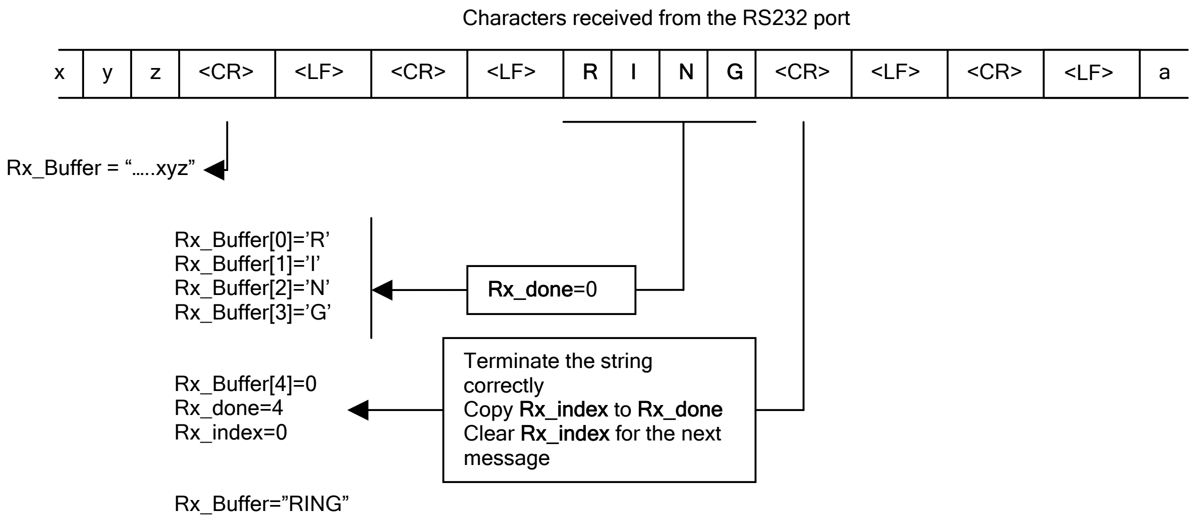
Call answered ( * key pressed):

- Send the ATA message to the modem
- Move the system to the CONNECTED state

### 11.2.1 Message reception

The RING message, and any other message, can be detected by developing a macro that saves incoming characters into a string variable (Rx_Buffer). This approach relies on the fact that the Tx_Command macro is now removing all echoed characters from the incoming data stream.

Information gained from previous exercises indicates that all messages and responses start and finish with the <CR><LF> characters. One way to isolate message would be to save all received characters until a <CR> is detected; then make the saved string available to the main program.



- Use a byte variable (Rx_index) to index Rx_Buffer
- Use another byte variable (Rx_done) to inform the main program of the completion of the message by copying Rx_index to it when the <CR> is detected. Set it to zero in all other cases.
- Do not save the <LF> character.
- Do not save the <CR> character, but use it to indicate the end of the message.
- The <CR><LF><CR><LF> sequences will produce an additional message string when the second <CR> is received. As the macro does not save <CR> or <LF> characters the Rx_index value copied to Rx_done will be zero, so the main program will not detect any reception.

## 11.3 Objectives

- Develop a macro to receive individual modem messages and responses (not echoed characters).
- Recognise the 'RING' message as indication of an incoming call.
- Automatically adjust the program state on detection/loss of the 'RING' message
- Create a timing function that:
    - Maintains the RINGING state between consecutive 'RING' messages.
    - Times-out if the 'RING' messages are no longer being received.
    - Does not suspend operation of the program.

## 11.4 Requirements

- A multi-programmer board attached to a PC running Flowcode
- An RS232 board
- A keypad
- A GSM modem with an active SIM card and an audio headset attached.
- A working solution to exercise 2, or a copy of **Phone_03.fcf**

## 11.5 Implementation

### 11.5.1 Message detection

Develop a macro, Rx_Message, to perform the message detection function and use a non-zero value of the Rx_done variable to force the main program to test the message in Rx_Buffer.

Use the string manipulation function, Compare$, to detect the presence of 'RING' in Rx_Buffer, and use a byte variable, match, to store the result of the comparison.

```
match = Compare$("RING", Rx_Buffer, 1)
```

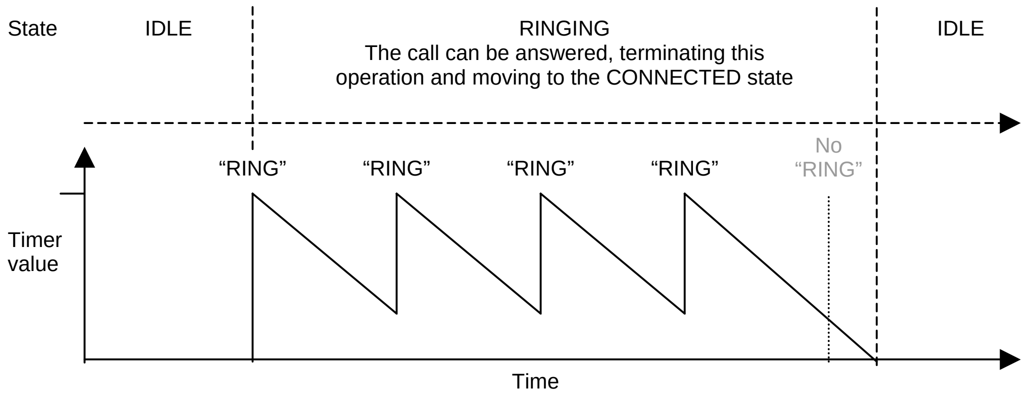
Note:
Compare$ returns a zero if the two strings match

Use Rx_Message to detect the "OK" responses to modem commands and confirm execution before changing the state variable.

### 11.5.2 Message interval timer

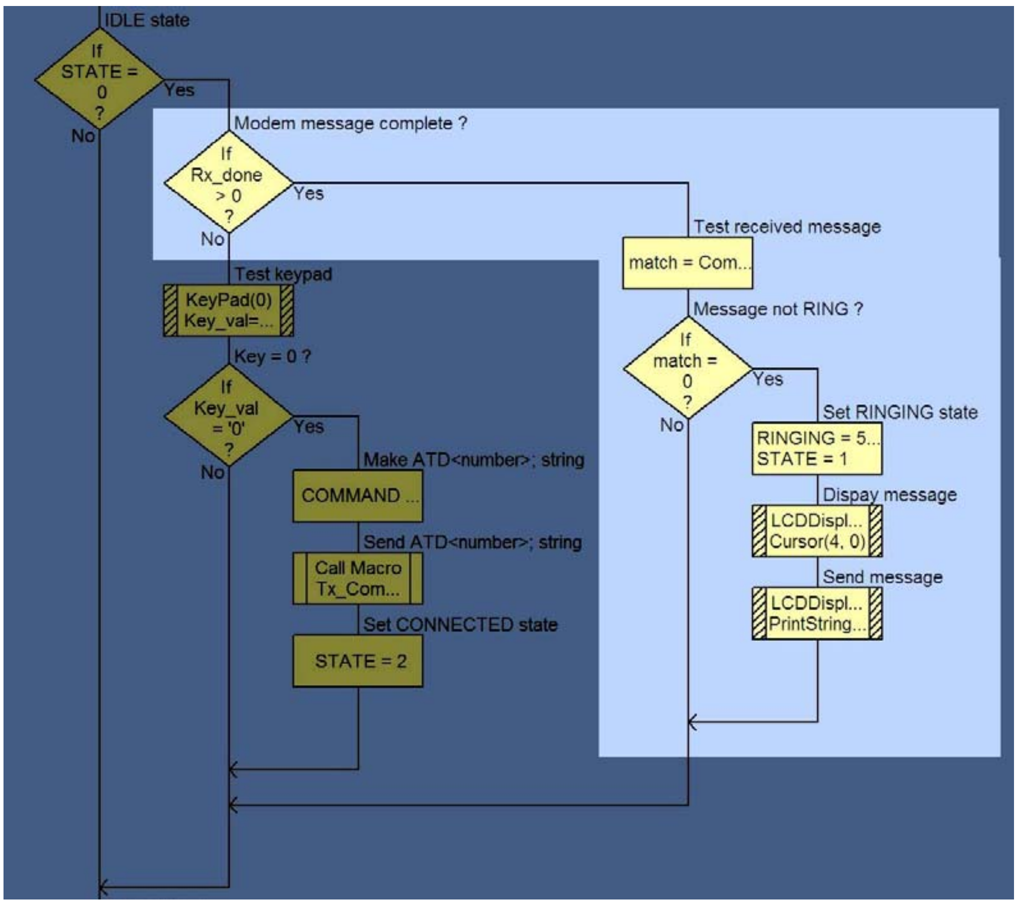
The Flowcode 'Delay' function allows accurate time delays to be introduced into programs, but suspends program operation during these periods. In this exercise it is necessary to maintain program operation in the period between consecutive 'RING' messages, allowing the RS232 port to be read and the * (answer) key to be tested.

To achieve this, the timer should be based on an integer variable that is set to a value (500) when a 'RING' message is detected in the IDLE or RINGING states. The timer is decreased by 1 every time the RINGING state code is executed, until it reaches zero - time-out. The timer will be prevented from reaching zero if regular 'RING' messages are received, maintaining the RINGING state and allowing the call to be answered. The timer period is not accurate, due to the effects of other parts of the program, but is suitable for this application.


State
IDLE
RINGING
The call can be answered, terminating this operation and moving to the CONNECTED state
IDLE
"RING"
"RING"
"RING"
"RING"
No "RING"
Timer value
Time

### 11.5.3 IDLE state modifications

The IDLE state code from exercise 2 can be modified to recognise the 'RING' message and perform the functions necessary to change to the RINGING state



Test for the 'RING' message

- Change the state to RINGING
- Start the interval timer
- Display a message on the LCD

The IDLE state code with modifications to detect the 'RING' message from the modem.

### 11.5.4 RINGING state modifications

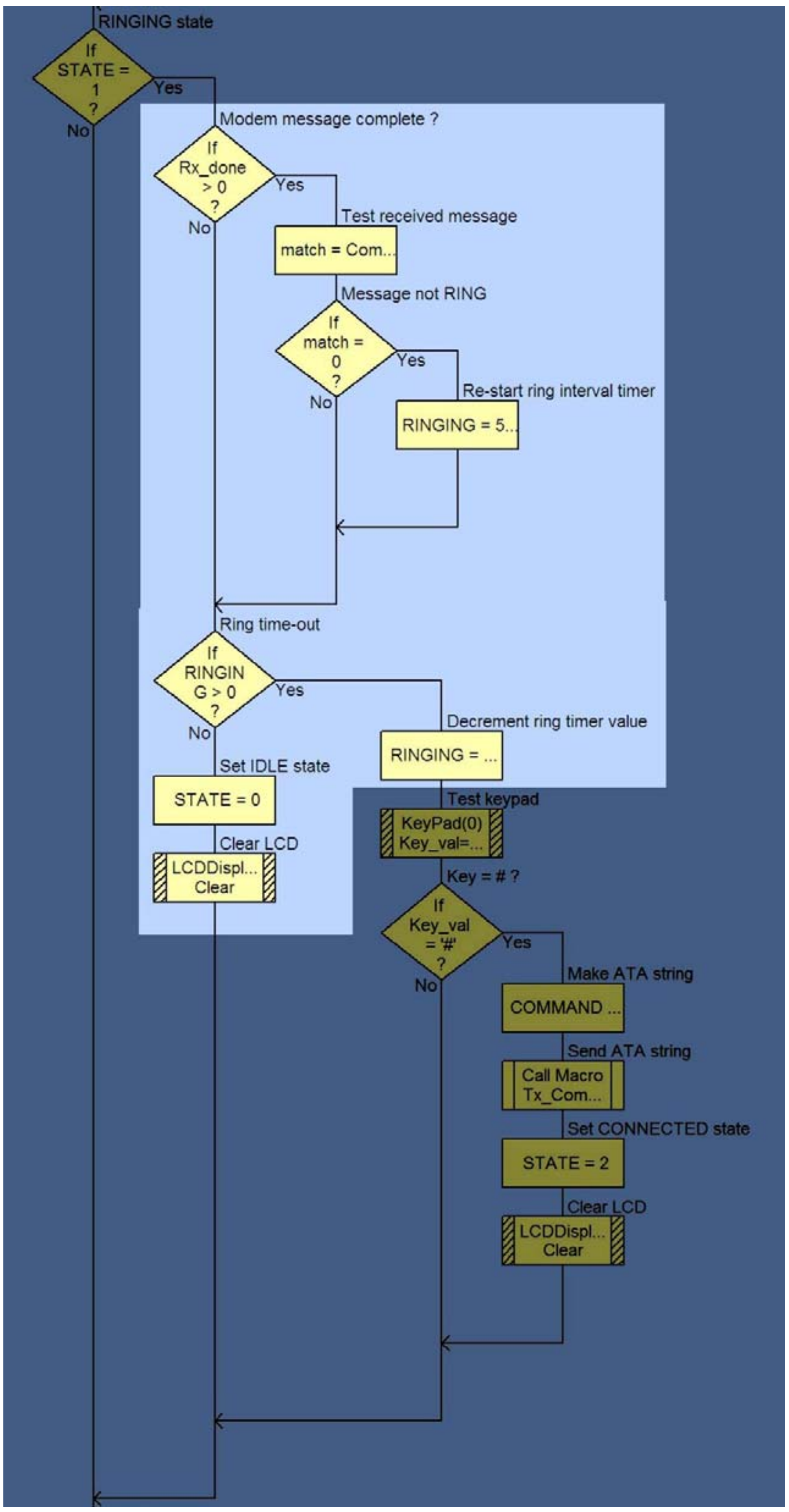
The RINGING state code from exercise 2 can be modified to detect the 'RING' message, manage the timer operation, and return to the IDLE state if a time-out occurs.

Test for a 'RING' message and re-initialize the timer if found

If the timer count has not reached zero (time-out), decrement the value and carry on

A 'RING' message was not received before the timer reached zero

- Return to the IDLE state.
- Clear the LCD



The IDLE state code with modifications to detect the 'RING' message from the modem and manage the interval timer

## 11.6 Learning outcome

This exercise should result in the development of a functional telephone that is structured to operate correctly in all circumstances. The student should now be capable of:

- Using existing code as the basis of further development.
- Implementing a range of program flow control mechanisms.
- Reading and identifying messages transmitted by the modem.

## 11.7 Further work

The main features missing from the telephone developed in this exercise are:

- Practical indication of an incoming call - the headset and LCD are insufficient.
- The ability to dial alternative numbers without editing the program.
- A display of call information.

Suggested improvements:

- Use the detection of the ‘RING’ message to create a more practical indication of an incoming call

- Modify the program to build a string of numbers entered from the keypad, and use the result with the ATD command to initiate a call.

- Display the number being dialled on the LCD and allow it to be edited before making the call.

- The modem can transmit several different messages indicating the state of a call connection (see the documentation). Test for some of these messages, in addition to ‘RING’, and make the program respond correctly.

  (Advanced)
- The modem can provide information on incoming calls (see the documentation). Send the appropriate AT commands to the modem, check for the expected responses and display the information on the LCD.

# 12. Exercise 5: Automatic call handling

### 12.1 Introduction

The previous exercises have resulted in the development of an application that is analogous to an original, mechanically operated, telephone. In this exercise the advantages of having a microcontroller in the system will be exploited by developing a phone that automatically answers incoming calls and hangs-up correctly when the calls are terminated. The detection of specific modem messages will cause the controller to transmit the necessary AT command strings.

### 12.2 Theory

The controller can be programmed to respond to the detection of specific modem messages and automatically send the appropriate control commands back to the modem.

The controller should already be capable of detecting the 'RING' message, but only responds by displaying a message on the LCD and managing a timer. This requires manual intervention to transmit the 'ATA' command and answer a call. The controller could be re-programmed to send the 'ATA' command on first detection of the 'RING' message, while in the IDLE state, and move directly to the CONNECTED state, resulting in an instantaneous, automatic, answering function.

The modem sends the 'NO CARRIER' message when a call is terminated remotely. Detection of this message, while in the CONNECTED state, would allow the controller to send the ATH message and move directly to the IDLE state.

Starting with the solution to the previous exercise, it should be possible to modify the code in the following way:

- The IDLE state code should only respond to detection of the 'RING' message, not the dial key.
- The RINGING state code should answer the call immediately, not wait for the answer key or any subsequent 'RING' messages.
- The CONNECTED state code should hang-up the call on reception of a line fault message (NO CARRIER), not the HANG-UP key.

If the device gives no indication that it is answering an incoming call, and the supplied headset is replaced with a sensitive microphone circuit, the result could be used as a long-range bugging device.

Replacing the headset with a more powerful audio amplifier and speaker would allow the device to be used as a remote access public address system.

Note:
The sensitive microphone and powerful speaker should not be used at the same time due to problems with audio feedback.

### 12.3 Objectives

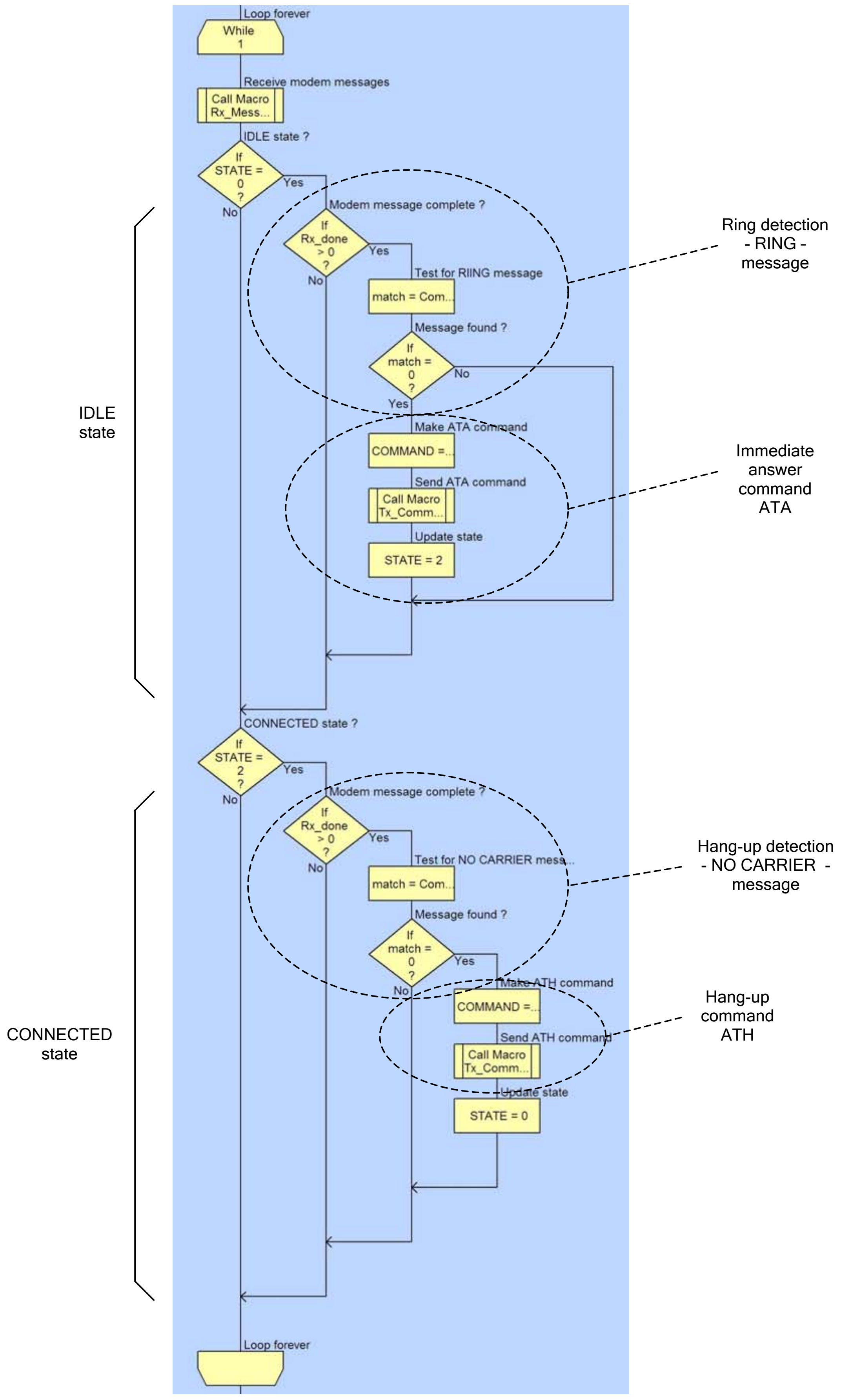
- Carry out modifications to the previously developed code to change the device function from a normal telephone to an auto-answering audio link.

### 12.4 Requirements

- A multi-programmer board attached to a PC running Flowcode
- An RS232 board
- A GSM modem with an active SIM card and an audio headset attached.
- A working solution to exercise 4, or a copy of Phone_04.fcf

## 12.5 Implementation

- The dialling key detection can be removed from the IDLE state as this phone only answers incoming calls.
- The whole of the RINGING state can be removed as the phone is answered immediately from the idle state
- The ATA command can be issued directly from the IDLE state when the first 'RING' message is detected.
- The hang-up key detection in the CONNECTED state is replaced by NO CARRIER' message detection.


Loop forever
While 1
Receive modem messages
Call Macro Rx_Mess...
IDLE state ?
If STATE = 0 ?
Yes
No
Modem message complete ?
If Rx_done > 0 ?
Yes
No
Test for RIING message
match = Com...
Message found ?
If match = 0 ?
No
Yes
Make ATA command
COMMAND = ...
Send ATA command
Call Macro Tx_Comm...
Update state
STATE = 2
CONNECTED state ?
If STATE = 2 ?
Yes
No
Modem message complete ?
If Rx_done > 0 ?
Yes
No
Test for NO CARRIER mess...
match = Com...
Message found ?
If match = 0 ?
Yes
No
Make ATH command
COMMAND = ...
Send ATH command
Call Macro Tx_Comm...
Update state
STATE = 0
Loop forever
IDLE state
CONNECTED state
Ring detection - RING - message
Immediate answer command ATA
Hang-up detection - NO CARRIER - message
Hang-up command ATH

### 12.6 Learning outcome

A good understanding of the previous exercises should allow this exercise to be completed quickly and neatly. Unnecessary code sections could be removed completely, or inhibited by decision blocks. The student should now be capable of:

- Recognising the function of individual blocks of code.
- Editing and re-arranging the sequences of code blocks to change the operation of a device.

### 12.7 Further work

- Add security to the system by detecting the caller ID and only answer if it matches one of a set of pre-loaded numbers (see the modem documentation).

## 13. SMS introduction

SMS "Short Message Service" text messaging has been an integral part of the GSM standard since it was first devised. A set of AT commands have been included in the GSM standards to allow messages to be sent, received, stored, recalled and formatted. The following exercises represent a brief introduction to text messaging functions, but result in the rapid development of powerful applications.

Due to the nature of text messaging it is necessary to introduce some of the complexities from the outset. All the solutions are based on the concepts developed in the audio section and should not pose significant problems to any one who successfully completed the audio exercises.

# 14. Exercise 6: Send a text message

## 14.1 Introduction

The text message functions of the GSM modem are handled in a similar way to voice calls, using AT commands, but with a different set of functions.

A number of options are available to control the way the modem handles and presents the messages. These will be configured in initialization code to achieve the required results

The aim of this exercise is to develop a Flowcode program that will perform the functions necessary to transmit a pre-loaded text message to a pre-loaded number, when an allocated keypad key is pressed.

## 14.2 Theory

### 14.2.1 Message format

Configuration of the modem is advisable before attempting to transmit or receive any messages.

The AT+CMGF command sets the format of the test messages and most of the modem responses.

Text messages can be delivered in compressed PDU (Protocol Data Unit) format. This format is efficient (7 bits per character), but can be difficult to decode into readable text. Text format uses a standard, 8-bit ASCII code for each message character; these can be transferred directly into string variables.

The AT command required to configure this option is Message Format command:

```
AT+CMGF=1
```

### 14.2.2 Send a message

In voice mode operation AT commands are used to establish and control a call connection, but the voice data is handled separately by the modem using additional audio circuitry.
In SMS text mode, all control and data information passes through the serial port in the form of AT commands, responses, and messages.

The AT command to send a text message is:

```
AT+CMGS=<”number”>
```

The modem responds by transmitting a '<CR><LF>> ' message when ready( note the space after the > character and the absence of a <CR> or <LF> character as termination) - this would indicate the start position of the text if the system was being controlled from a terminal screen and keyboard.

The message text can be transmitted to the modem after receipt of the '> ' characters, and the whole sequence is terminated with the <CTRL-Z> character - value 26: a historic “End Of File” marker.

This is the full sequence:

| | |
|---|---|
| • Transmit | AT+CMGS=<”number”><CR> |
| • Wait for | <CR><LF>><space> |
| • Transmit | <message> |
| • Transmit | <CTRL-Z><CR> |

The modem then compiles a message header, adds the <message> text, and transmits the resulting data to the SMS network.

### 14.2.3 Rx_Message macro modifications

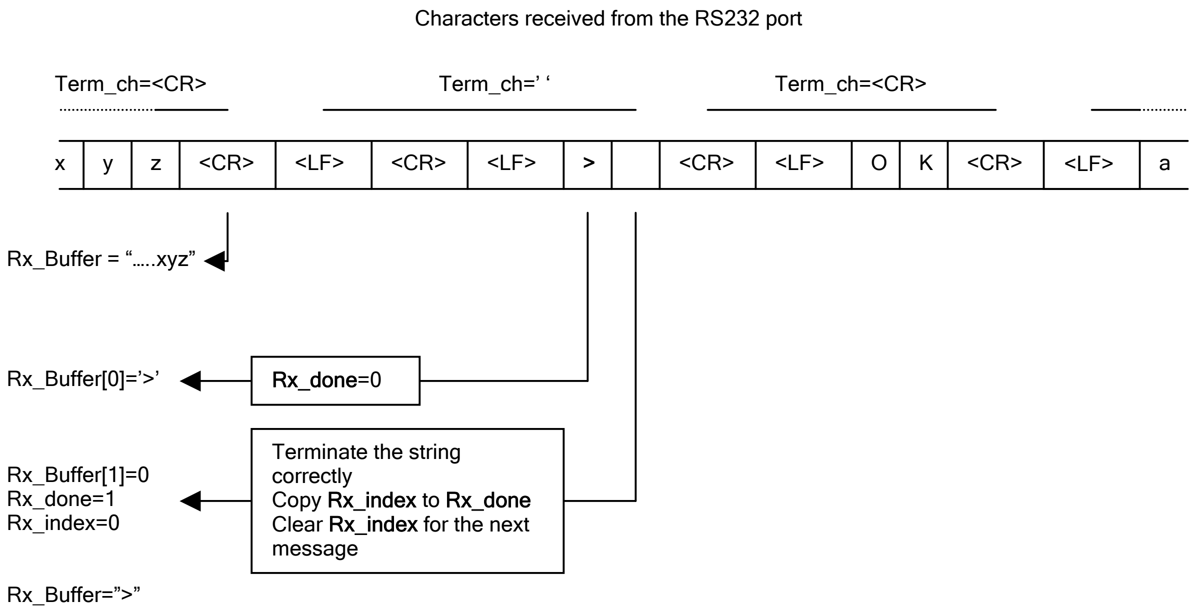
The range of message formats to be detected in SMS mode is greater than in audio mode. Detection of the "> " characters cannot be achieved with the existing Rx_Message macro due to the absence of a <CR> character. The Rx_Message macro must now be modified to use a character supplied by the main program to detect the completion of a message, or message segment.

Use a byte variable to hold the message termination character (Term_ch) and set the required value from the main program.

The macro should compare each received character with Term_ch, instead of <CR>, but perform the same message completion functions as previously developed.

As before, <CR> and <LF> must not be added to the Rx_Buffer string.

The value of Term_ch might need to be changed regularly, depending on the expected messages.



### 14.2.4 Tx_Command macro modifications

The original Tx_Command macro transmits a <CR> character at the end of each command string. There are now a number of requirements for strings to be transmitted without the automatic addition of <CR>. This will allow commands to be sent to the modem in sections with the <CR> only being added to the last section.

Add a byte variable (Send_CR) and use this to control the transmission of the <CR> character by the Tx_Command macro: 0=don't send.

Each command, or command segment, should be copied into the COMMAND string, and the value of Send_CR set before executing the Tx_Command macro.

### 14.3 Objectives

- Initialize the modem for correct text mode operation.
- Initiate a text message transmission sequence when a key is pressed.
- Wait for the required modem response.
- Send and terminate the message text.

### 14.4 Requirements

- A multi-programmer board attached to a PC running Flowcode
- An RS232 board
- A keypad
- A GSM modem with an active SIM card (audio headset not required)
- A mobile phone capable of receiving text messages (the dialling number of this phone must be entered into the AT+CMGS command string in the Flowcode software)

## 14.5 Implementation

The Flowcode program for this exercise should transmit the AT+CMGS="<number>" command when the '#' key on the keypad is pressed.

The number to be dialled, <number>, can be entered into a string at the top of the program to make it easier to locate and change.

Note:
It is not possible to include the " character in a string so this character should be transmitted independently; before and after transmission of the <number> string.

It is important to wait for the modem to generate the "> " response before sending the text message so the code must allow for this.

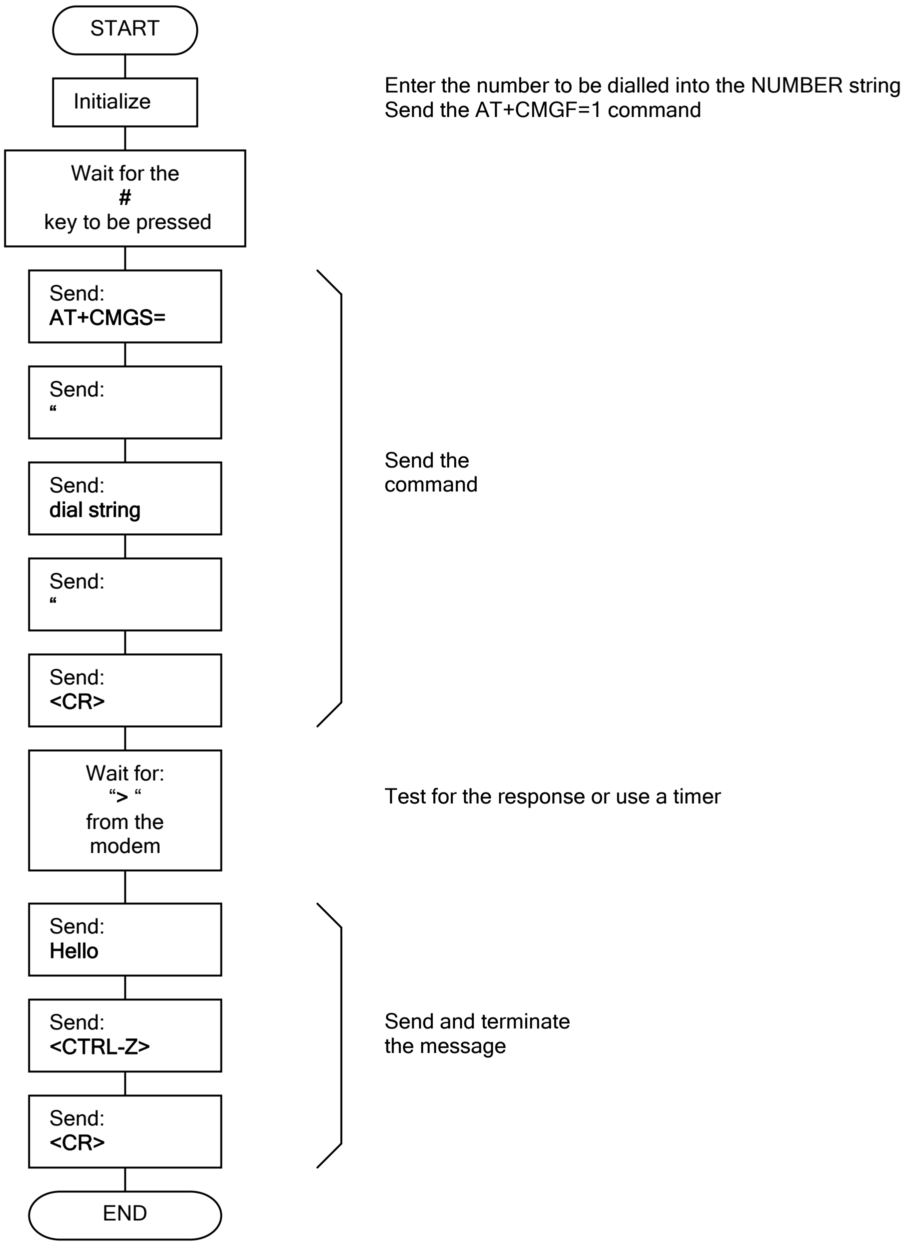
Use the new Rx_Message macro to detect the space after the > character and test the Rx_Buffer for the > character.

Note:
The Compare$ function does not appear to work reliably with single character strings so compare Rx_Buffer[0] with '>' directly using calculation functions.

When the modem is ready to receive the text message, the message should be sent and terminated with a <CTRL-Z> character (26).

On completion the program should terminate to prevent accidental transmission of further messages.

The general program structure is given below.


START
Initialize
Wait for the # key to be pressed
Send: AT+CMGS=
Send: "
Send: dial string
Send: "
Send: <CR>
Wait for: "> " from the modem
Send: Hello
Send: <CTRL-Z>
Send: <CR>
END
Enter the number to be dialled into the NUMBER string
Send the AT+CMGF=1 command
Send the command
Test for the response or use a timer
Send and terminate the message

## 14.6 Learning outcome

The program developed in this exercise performs the relatively complex task of initiating and transmitting an SMS text message. Completion should provide an understanding of:

- Event sequencing for text message transmission.
- String data formatting and transmission

## 14.7 Further work

The program developed in this exercise allows the transmission of a single SMS text message when a keypad key is pressed.

- Develop the program to loop continuously and monitor for a range of input events.
- Transmit different messages for each event.
- Include variable data in the transmitted messages

***Note:***
A solution for this exercise allows for the development of many applications in the field of remote telemetry. The system can be programmed to transmit meaningful messages when triggered by a range of events, including: digital input conditions; analogue input levels; or timer values. The messages can contain up to 140 characters and include variable text content and real data values - this is mainly an exercise in string manipulation.

The temperature sensor provided with this solution can be used to simulate a range of industrial and domestic applications, and generate text message alarms when the detected temperature reaches preset levels. Alternatively, the numeric values from the sensor readings can be transmitted at regular intervals to provide a remote temperature log over time.

## 15. Exercise 7: Receive a text message

### 15.1 Introduction

A text message is received as a stream of characters that include large amounts of data in addition to the original text. The modem can be configured to respond in a number of ways to the reception of a text message.

Some of the possible responses are:

- Save the text message to the SIM card memory (the message can be found by reading some of the modem parameters at a later date).
- Save the text message to the SIM card memory and transmit an unsolicited message to the controller indicating its location.
- Transmit the text message to the controller immediately and not save it to SIM card memory.

The following examples will configure the modem to produce the latter response as this provides immediate results and will not fill the SIM card memory. The AT command required to achieve this is the New Message Indication command:

```
AT+CNMI=2,2
```

A message received from the modem will be of the following form:

```
+CMT: "+441234567890",,"07/02/27,09:14:15+00"<CR><LF>Hello<CR><LF>
```

| | |
|---|---|
| +CMT: | Message header |
| "+441234567890" | Caller ID |
| 07/02/27 | Date |
| 09:14:15+00 | Time + local time zone offset |
| <CR><LF> | Message header termination |
| Hello | Text message |
| <CR><LF> | Text message termination |

All of this information is potentially useful and can be extracted from the data stream using the improved RX_Message macro from the previous exercise.

For this exercise only the message header and text message will be used - the header for detection and the message for display purposes.

## 15.2 Theory

The incoming message can be split into segments using the RX_Message macro. Correct selection of the termination character will allow the required information to be isolated and extracted. A simple state machine can be used to count the message segments, control the operation of the program, and select the appropriate termination character.

Create a byte variable (SEGMENT) to control the message segment detection function.

Detection of the message header can be achieved by setting the RX_Message terminator character to ' ' (the space character). When the macro indicates a message reception the buffer string can be compared with "+CMT:".

If the +CMT: header is detected, the program can increment the SEGMENT value and progress to detection of the caller ID, date, and time. The information is not required for this exercise, so setting the delimiter character to <CR> will retrieve all the characters in a single operation.

Detection of the first <CR> character will allow the SEGMENT variable to be incremented, indicating that the following data will be the original text message.

Detection of the second <CR> character will indicate completion of the message reception, allowing the SEGMENT value to be reset to the header detection value and the delimiter character to be reset to ' '. The original text message will be stored as a string in Rx_Buffer, allowing it to be displayed, tested, or manipulated, as the application requires (displayed in this case).

**Example:**

The message generated by the modem to deliver a received text message is:

<CR><LF>+CMT: "+441234567890",,"07/02/27,09:14:15+00"<CR><LF>Hello<CR><LF>

① ② ③

1. Calling **Rx_Message** with the delimiter set to ' ' (space) will cause the function to set a value in **Rx_done** following reception of the +CMT: segment of the message. +CMT: should also be available in the **Rx_Buffer** string to confirm reception of the correct message.

2. Calling **Rx_Message** with the delimiter set to **<CR>** will then set a value in **Rx_done** following reception of the caller ID, date, and time characters. The information is not required in this example and can be ignored

3. Calling **Rx_Message** again with the delimiter set to **<CR>** will set a value in **Rx_done** following reception of the text section of the message. This will be available in the Rx_Buffer string for comparison and/or display.

**Note:**
The delimiter characters <CR> and <LF> are never included in the receive buffer.

## 15.3 Objectives

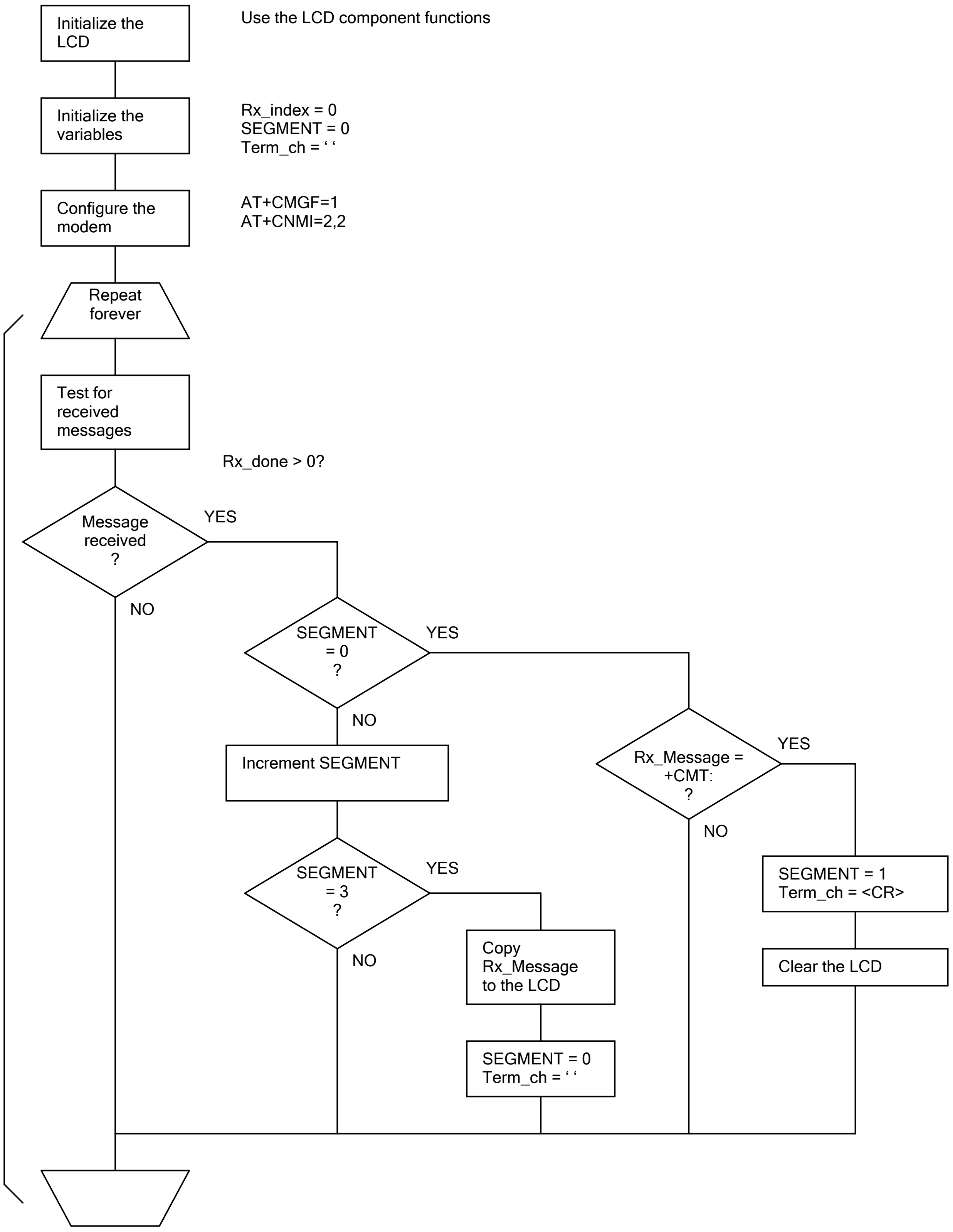
- Understand the format of the data stream sent by the modem on receipt of a text message.
- Receive the data stream and split it into its component parts.
- Recognize and confirm the message header.

- Extract and display the text message.

## 15.4 Requirements

- A multi-programmer board attached to a PC running Flowcode.
- An RS232 board.
- A keypad.
- A GSM modem with an active SIM card (audio headset not required).
- A mobile phone capable of receiving text messages (the dialling number of this phone must be entered into the AT+CMGS command string in the Flowcode software.

## 15.5 Implementation



## 15.6 Learning outcome

Completion of this difficult exercise should provide a thorough understanding of:

- Text message delivery formats
- Advanced string handling concepts

## 15.7 Further work

The program developed in the exercise is able to identify an incoming text message report stream, read it, and display the message contents on the display.

Develop the software to read the read the contents of the message and perform a range of functions dependent on the message received:-

- Turn outputs on and off
- Display different messages
- Read and display input values

*Note:*
A solution to this exercise represents a practical remote control application. The development of string detection and interpretation code will allow the system to respond to commands and data contained within the received text message, including the ability to control attached devices.

# 16. Exercise 8: Automatically respond to a text message

## 16.1 Introduction

The programs developed in the previous two exercises have individually demonstrated the transmission and reception of SMS text messages. In this exercise the two previous programs will be combined to create a useful application that is capable of reading an incoming text message, and creating and sending a reply message.

## 16.2 Theory

A text message reception program can be developed to detect the message header of a text message indication stream (+CMTI:) and extract both the caller ID and the message content. The message contents can be used to control an operation, and the caller ID information can be used in the formatting of a reply message.

### 16.2.1 Message decoding

The RX_Message macro can be used to identify the message header, extract and save the caller ID (including quotation marks), and extract the message contents.

```
+CMT: "+441234567890",,"07/02/27,09:14:15+00"<CR><LF><message><CR><LF>
```

If the message contents are recognized, a message sending function can be executed, using the stored caller ID number, to generate a response message.

Using the RX_Message macro:

- Set the delimiter to ' ' to isolate the message header - test the results against "+CMT:".
- Set the delimiter to ',' to isolate the caller ID - save the results to a NUMBER string - including the speech marks.
- Set the delimiter to <CR> to isolate the time and date characters - discard the results.
- Leave the delimiter as <CR> to isolate the message contents

When reception of the text message is complete, use the Compare$ function to test the received message in the Rx_Buffer string against a sample string. Use "Status" as the sample string and perform the comparison with case matching disabled.

Use a match between the two strings to initiate a message transmission sequence.

### 16.2.2 Response transmission

Code similar to that in Exercise 6 can be used to transmit a response message:

- There is no need to detect a transmit key as the code will only be executed when a message transmission is required.
- The number to be dialled is already stored, along with the surrounding speech marks, in the NUMBER string.
- The message transmission code should return operation to the message reception code when the transmission is completed

Send the message "Ready" in response to the "Status" message.

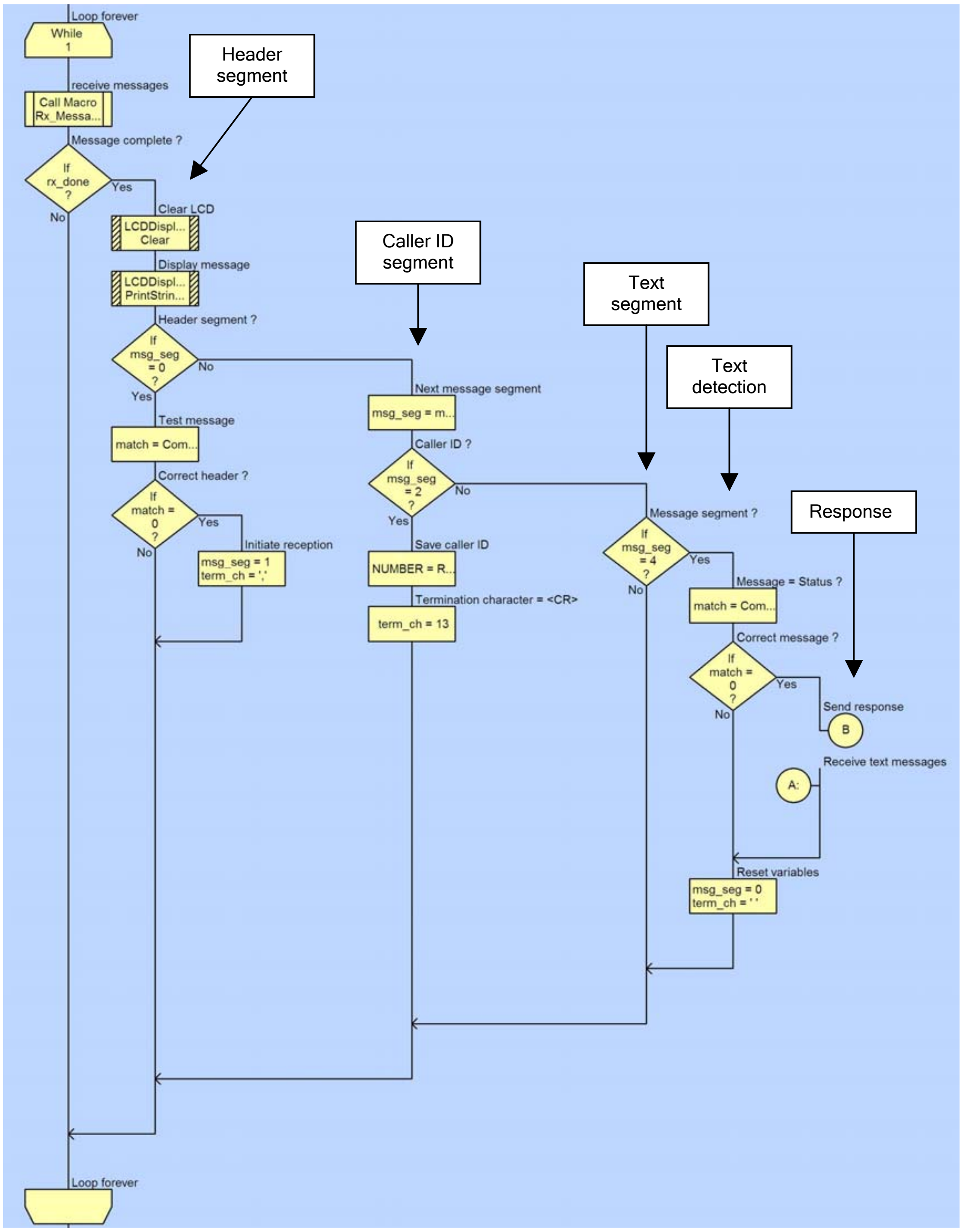
### 16.3 Objectives

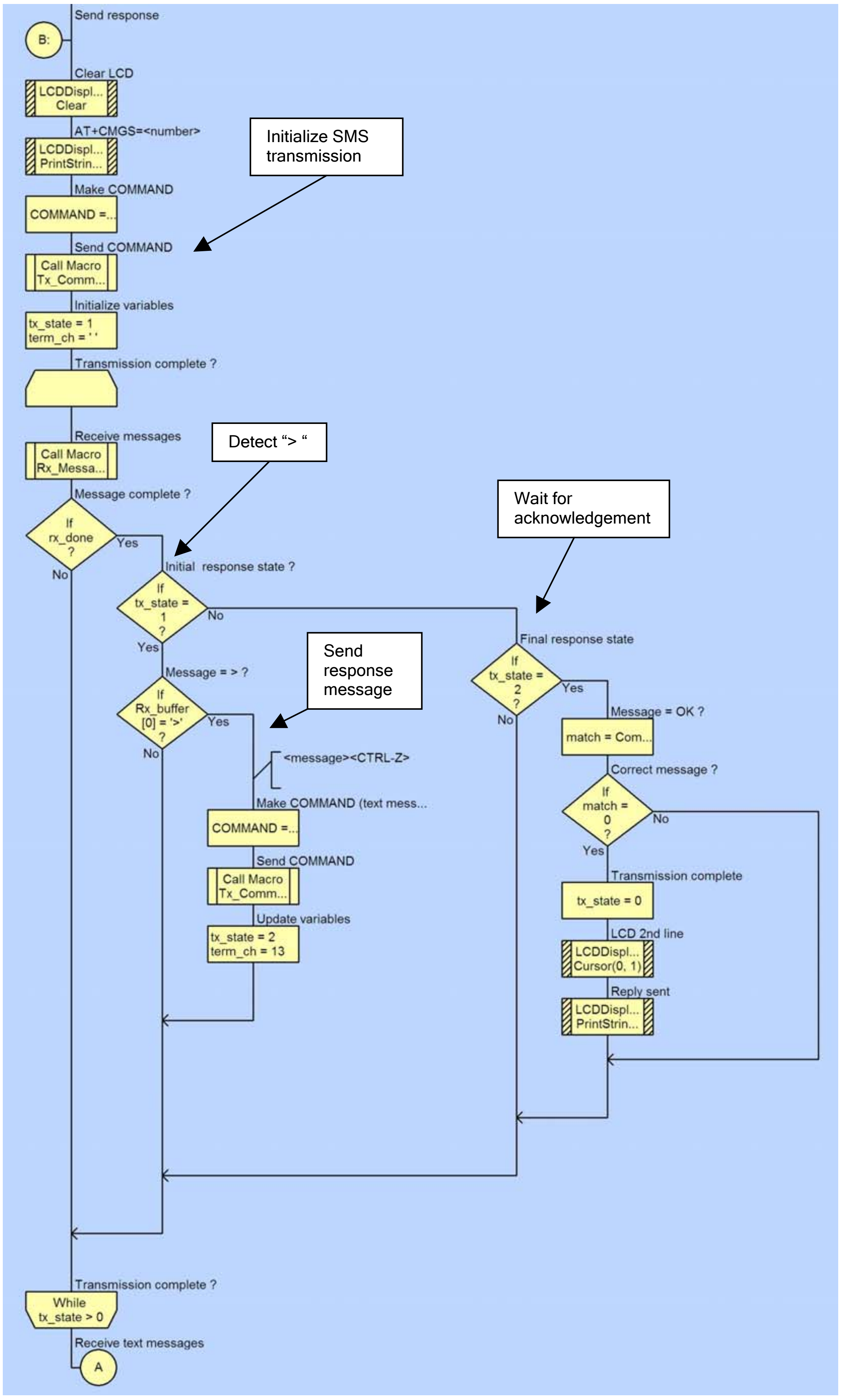
- Receive an incoming text message.
- Extract and interpret sections of the data stream.
- Use message and system data to generate and transmit a response message.

### 16.4 Requirements

- A multi-programmer board attached to a PC running Flowcode.
- An RS232 board.
- A keypad.
- A GSM modem with an active SIM card (audio headset not required).
- A mobile phone capable of receiving text messages (the dialling number of this phone must be entered into the AT+CMGS command string in the Flowcode software.

## 16.5 Implementation




Send response
B:
Clear LCD
LCDDispl... Clear
AT+CMGS=<number>
LCDDispl... PrintStrin...
Initialize SMS transmission
Make COMMAND
COMMAND =...
Send COMMAND
Call Macro Tx_Comm...
Initialize variables
tx_state = 1 term_ch = ' '
Transmission complete ?
Receive messages
Call Macro Rx_Messa...
Detect "> "
Message complete ?
If rx_done ?
Yes
No
Wait for acknowledgement
Initial response state ?
If tx_state = 1 ?
No
Yes
Message = > ?
If Rx_buffer [0] = '>' ?
Yes
No
Send response message
<message><CTRL-Z>
Make COMMAND (text mess...
COMMAND =...
Send COMMAND
Call Macro Tx_Comm...
Update variables
tx_state = 2 term_ch = 13
Final response state
If tx_state = 2 ?
Yes
No
Message = OK ?
match = Com...
Correct message ?
If match = 0 ?
No
Yes
Transmission complete
tx_state = 0
LCD 2nd line
LCDDispl... Cursor(0, 1)
Reply sent
LCDDispl... PrintStrin...
Transmission complete ?
While tx_state > 0
Receive text messages
A

## 16.6 Learning outcome

This is the most complex of all the exercises and is a test of both the text message reception and transmission requirements, and of their potential use in practical applications.

The completed exercise represents a remote control and telemetry application that could be used in conjunction with a wide range of additional sensors and actuators, providing global access to the functions provided

## 16.7 Further work

- Add security to the system by checking the caller ID against a pre-loaded list of approved numbers. Only react to recognized numbers (save money).
- Increase the number of messages that can be identified and used to perform specific functions.
- Include the ability to handle numeric data in both the received and transmitted messages.
- Develop an alarm system that can send a message, including specific information, when triggered. Allow the alarm to be remotely reset.
- Use the LCD to display the progress of message reception and transmission.
- Make sure that the system can recover from any error conditions (unexpected responses from the modem etc.)

***Note:***
A solution to this exercise represents a complete remote telemetry application. Commands and data can be sent to the remote device in addition to the retrieval of data. In this trivial example the 'Status' command simply triggered the 'Ready' response. With appropriate string manipulation code, a range of commands and data can be used to control the remote application as well as retrieving data.

A possible application would be a burglar alarm system that could be activated, deactivated, and configured. The alarm would send a message containing details of any trigger events (zones triggered etc.) and allow remote resetting.