E BLOCKS 2

GSM Communications

CP2832

MATRIX

www.matrixtsl.com

CP2832

Comunicaciones GSM

Guía del instructor

Contenido

Sobre este curso

Objetivos:
El objetivo de E-blocks2 Mobile Phone Solution es proporcionar una introducción práctica al diseño y la tecnología de los sistemas de comunicación modernos. Al final de este curso, el estudiante habrá sido guiado a través de ejemplos de programación que resultan en el desarrollo de:

- una gama de dispositivos de telefonía móvil totalmente funcionales
- una gama de dispositivos controlados por mensajes de texto,

Utilización del kit de telefonía móvil para la enseñanza

He aquí dos posibles enfoques:

- como plataforma de motivación para la enseñanza de la programación de microcontroladores
  Los estudiantes modernos quieren aprender electrónica en un contexto moderno. Como la mayoría tiene teléfono móvil y pasa mucho tiempo utilizándolo, tiene sentido utilizar el teléfono móvil como mecanismo para impartir ese aprendizaje.
- El teléfono móvil puede utilizarse para introducir temas relacionados con la programación de microcontroladores. Por ejemplo, se puede crear un archivo de ejemplo que envíe un mensaje de texto a través del teléfono móvil. A continuación, puede introducir el concepto de convertidor A/D para telemetría y pedir a los alumnos que desarrollen programas que examinen cómo funcionan los convertidores A/D dentro del teléfono móvil.
  No hemos previsto específicamente este uso, pero debería ser fácilmente capaz de desarrollar algunas variantes de los ejemplos que hemos proporcionado para adaptarse a sus propósitos aquí.como plataforma para enseñar sistemas de comunicaciones
- El curso está diseñado en torno a este enfoque: enseñar cómo funcionan los sistemas de comunicación. Paso a paso, los alumnos desarrollan un teléfono móvil totalmente funcional con todas las capacidades de voz y mensajería de texto.

Resultados:

El curso está muy estructurado y los resultados del aprendizaje se dividen en tres grupos:

Resultados de la programación:

- Control por teclado.
- Control por LCD.
- Protocolo y programación RS232.
- Construcción y deconstrucción de cadenas en comunicaciones.
- Utilización de máquinas de estado en el control de sistemas electrónicos.

Resultados de las comunicaciones:

- Comunicaciones RS232 y protocolos de handshaking.
- Representación ASCII de los caracteres en los mensajes.
- Estructura y protocolos de comandos AT.
- Envío y recepción de mensajes de texto.
- Control del módem y mensajería.
- Comprensión de la intensidad de la señal.

Gestión de proyectos y resultados de desarrollo

- Utilización de diagramas de flujo y diagramas de estado en la planificación de sistemas.
- El enfoque modular en la construcción de sistemas electrónicos.

Para más información

´ **Hardware E-blocks2**

Las fichas técnicas de todas las tarjetas E-blocks2 están disponibles en el sitio web de Matrix: www.matrixtsl.com

- CP2832 GSM Archivos de Recursos.zip también contiene:
- Una versión PDF de este documento
- Archivos de ejemplo que acompañan a este documento.

Información GSM

´ Este curso no está diseñado en torno a un módulo GSM específico, pero se incluirá uno en el paquete.

Software

Programación Flowcode:

Los estudiantes deben familiarizarse con los conceptos básicos del lenguaje leyendo las secciones del archivo de ayuda sobre la adición de iconos y componentes, y trabajando con los primeros tutoriales para adquirir experiencia sobre el funcionamiento de Flowcode.
También pueden utilizar el curso **"Introducción a la programación de microcontroladores".** Está diseñado para llevar a los usuarios desde lo más básico hasta temas bastante avanzados, y es un recurso útil.

Programación en C o ensamblador:

Para cada uno de los E-blocks2 más complejos, existe una guía de estrategia de programación disponible en nuestro sitio web. Éstas dan un esquema de cómo cada E-blocks2 puede ser programado en código C o ensamblador.
Si programa en C o ensamblador, puede beneficiarse de uno de nuestros cursos en CD-ROM para la pro-gramación de microcontroladores PIC.

Importante
La mayoría de los ejercicios requieren la disponibilidad de un teléfono móvil donante para proporcionar conectividad. Algunos de los programas marcan un número preprogramado al funcionar. Se programa un número ficticio en el software antes de su envío, pero este número debe cambiarse por el número del teléfono del donante antes de compilar el software.

Section	Notes for instructors	Timing (minutes)
3.4 Flowcode	Flowcode is one of the world's most advanced graphical programming languages for microcontrollers. The great advantage of Flowcode is that it allows those with little to no programming experience to create control programs for complex electronic systems in minutes. In this situation, it provides the simplest and quickest way to program the microcontroller used in the Mobile Phone kit. (The programs can be written in C or in Assembler, where students have expertise in those languages.) This section gives instructions on how to install and register a copy of the Flowcode program, and points out that Flowcode solutions to all exercises are provided on the accompanying CD-ROM.	15
3.5 Testing your mobile phone and E-blocks2	Each E-blocks2 board can be tested individually using test routines available from the Matrix website. Similarly, a sample program is available to allow testing of the assembled Mobile Phone kit.	10 - 25
4. Exercise 1 – A basic telephone		
4.1 Introduction	This is the first of a series of practical assignments using Flowcode to control the microcontroller responsible for managing communication with the modem. This exercise focuses on dialling a pre-loaded number, answering an incoming call, and terminating a connection using buttons on the keypad module. The introduction looks at the three AT commands needed in this application and how to incorporate them into a dialling string using ASCII code. At the heart of the application is the RS232 Flowcode component. The student is taken through configuring the corresponding Flowcode RS232 component, and its SendRS23Char function, used to send individual characters to the modem. Tx_Command, the first of two important macros, used repeatedly throughout these exercises, is discussed and then constructed. It is used to transmit a series of characters contained in the string variable COMMAND – hence its name. The 'What to do' section suggests three keypad keys for use in dialling a number, answering a call and then terminating it. The significance of the 'RS232Receive' function is explained. Students develop the Flowcode program from the flowchart provided A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
4.2 Objective		
4.3 Requirements		
4.4 The Flowcode program in detail		
4.5 What to do		

Section	Notes for instructors	Timing (minutes)
5. Exercise 2 – A simple 'State Machine'		
5.1 Introduction	The previous program suffered from a number of drawbacks. The user was allowed to interfere incorrectly with the functioning of the mobile phone, causing the modem to behave unpredictably. In this exercise, the telephone functionality is improved by creating a state machine, which limits such actions. The system sits in one of three states, and has limited options open to it, depending on its present state. These are described in the introduction. The 'What to do' section describes how to modify the program developed in exercise 1, to create the three states of the state machine, called 'Idle', 'Ringing' and 'Connected'. It is assumed that students know how to: • create variables using the Variables Manager; • add a keypad as an input device, and configure its properties; • add a component macro and select the keypad component; • add a RS232 device and configure its properties; • add a component macro and select the RS232 component; • call the RS232 component 'ReceiveChar' macro; • call the keypad 'GetAscii' macro; • create a macro; • use a Decision box to test the value of a variable; • set up a While loop based on the value of a variable; • create a String variable and initialise it; • use Calculation functions to initialise and increment a variable' In addition, students will need to know how to import the Tx_Command macro, developed in the previous program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
5.2 Objective		
5.3 Requirements		
5.4 The Flowcode program in detail		
5.5 What to do		
5.6 Further work		

Section	Notes for instructors	Timing (minutes)
6. Exercise 3 – Modem Responses		
6.1 Introduction 6.2 Objective 6.3 Requirements 6.4 The Flowcode program in detail 6.5 What to do 6.6 Further work	The first two exercises involved one-way communication with the modem only. Though characters were received from the modem, these were used only to prevent the modem transmit buffer from filling and thus blocking further communication. The program developed in this exercise displays the messages received from the modem. The introduction lists three types of transmission from the modem. The program distinguishes between these, and displays them on different rows of the LCD module. It also gives details of how to configure the LCD component in Flowcode. The 'Flowcode program in detail' section describes how to modify the previous program by creating a variable 'rx_char' and testing it to filter appropriate characters to display on the LCD. It also describes the modifications to the Tx_Command macro so that it displays the echoed characters on the appropriate line of the LCD. In addition to the prior knowledge assumed for exercise 2, it is assumed that students know how to: add a LCD display as an output device, and configure its properties; add a component macro and select the LCD display component call the LCD display 'Start' macro; call the LCD display 'Clear' macro; call the LCD display 'Cursor' macro; call the LCD display 'PrintAscii' macro A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30

Section	Notes for instructors	Timing (minutes)
7. Exercise 4 – Listening to messages		
7.1 Introduction 7.2 Objective 7.3 Requirements 7.4 The Flowcode program in detail 7.5 What to do 7.6 Further work	This exercise develops a macro to detect the presence of the 'RING' message, and moves the system between state 0 (IDLE) and state 1 (RINGING) automatically, as a result. The introduction identifies the steps needed to do this. The Rx_Message macro reads each character received from the modem until a <CR> is detected. Then it inserts a '0' in the Rx_Buffer to indicate that the message is ended. The program checks to see that the buffer is not overflowing by making sure that the string index 'Rx_index' does not exceed the buffer size. When complete, indicated by receipt of a <CR>, the saved string is examined in the main program. There, the compare$ function looks for the 'RING' message, and on finding it, moves the system to the RINGING state. Otherwise, it waits for a keypad '0' press to dial a number and make a call. The 'What to do' section lists the steps involved in creating the program, and shows, in two diagrams, the modifications needed to the previous. There are no new Flowcode functions in this program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
8. Exercise 5 – Automatic call handling		
8.1 Introduction 8.2 Objective 8.3 Requirements 8.4 The Flowcode program in detail 8.5 What to do	In this exercise the advantages of having a microcontroller in the system are exploited by developing a phone that answers incoming calls automatically and hangs-up when a call is terminated. This facility depends on detecting the 'NO CARRIER' message when a call is terminated remotely. Then, the controller sends the ATH message and moves directly to the IDLE state. The Tx_Command macro is modified by adding a loop to receive echoed characters and test for the <LF> character. The 'Flowcode program in detail' section lists the changes needed to the previous program, and these are then shown in the diagram in the 'What to do' section. There are no new Flowcode functions in this program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30

Section	Notes for instructors	Timing (minutes)
7. Exercise 4 – Listening to messages		
7.1 Introduction	This exercise develops a macro to detect the presence of the 'RING' message, and moves the system between state 0 (IDLE) and state 1 (RINGING) automatically, as a result. The introduction identifies the steps needed to do this. The Rx_Message macro reads each character received from the modem until a <CR> is detected. Then it inserts a '0' in the Rx_Buffer to indicate that the message is ended. The program checks to see that the buffer is not overflowing by making sure that the string index 'Rx_index' does not exceed the buffer size. When complete, indicated by receipt of a <CR>, the saved string is examined in the main program. There, the compare$ function looks for the 'RING' message, and on finding it, moves the system to the RINGING state. Otherwise, it waits for a keypad '0' press to dial a number and make a call. The 'What to do' section lists the steps involved in creating the program, and shows, in two diagrams, the modifications needed to the previous. There are no new Flowcode functions in this program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
7.2 Objective		
7.3 Requirements		
7.4 The Flowcode program in detail		
7.5 What to do		
7.6 Further work		
8. Exercise 5 – Automatic call handling		
8.1 Introduction	In this exercise the advantages of having a microcontroller in the system are exploited by developing a phone that answers incoming calls automatically and hangs-up when a call is terminated. This facility depends on detecting the 'NO CARRIER' message when a call is terminated remotely. Then, the controller sends the ATH message and moves directly to the IDLE state. The Tx_Command macro is modified by adding a loop to receive echoed characters and test for the <LF> character. The 'Flowcode program in detail' section lists the changes needed to the previous program, and these are then shown in the diagram in the 'What to do' section. There are no new Flowcode functions in this program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
8.2 Objective		
8.3 Requirements		
8.4 The Flowcode program in detail		
8.5 What to do		
8.6 Further work		

Section	Notes for instructors	Timing (minutes)
9. Exercise 6 – Send a text message		
9.1 Introduction	SMS text messaging involves a further set of AT commands. The remaining exercises explore their use. The introduction describes two optional message formats, and chooses the Text format for these exercises. It looks at the AT command involved and the modem response. The text message begins when the modem has sent a '> ' sequence. It is terminated using <CTRL-Z> as the end-of-file marker. The Rx_Message macro must be modified. At the moment, it looks for the <CR> character to indicate the end of a message. It must now be modified to use a character, set in the main program, and stored in the variable 'term_ch', to detect completion. This involves adding another decision box, and loop to the existing macro. The Tx_Command macro must also be modified. The macro modified in exercise 4 transmits a <CR> character at the end of each command string. Instead we want to add <CR> only to the last section. To do this, a byte variable (Send_CR) is used to control transmission of the <CR> character. When this is set to '0', the <CR> character is not sent, and the program execution jumps to a Connection point at the end of the macro. The value of Send_CR is set before executing the Tx_Command macro. The 'Flowcode program in detail' section describes the program structure, both in words, and through a flowchart. The 'What to do' section lists the steps needed to create the program. In addition to the knowledge assumed for earlier exercises, it is assumed that students know how to: add connection points to a Flowcode program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
9.2 Objective		
9.3 Requirements		
9.4 The Flowcode program in detail		
9.5 What to do		
9.6 Further work		

Section	Notes for instructors	Timing (minutes)
10. Exercise 7 – Receive a text message		
10.1 Introduction 10.2 Objective 10.3 Requirements 10.4 The Flowcode program in detail 10.5 What to do 10.6 Further work	This exercise creates a program to receive a simple text message, extract it from the accompanying information, and display it on a screen. As is pointed out in the 'Further work' section, this could form the basis of a remote control device if linked to a sensor and actuator of some form. It takes the simplified approach of relaying the contents of the text message to the LCD module directly, though there are more complex options, described in the introduction, involving saving to the SIM memory. The introduction goes on to explain the structure of a typical text message, beginning with the string '+CMT', and to show how it can be split into its components by specifying different delimiters. The next two sections show how to develop the program, helped by a flowchart detailing the program structure. There are no new Flowcode functions in this program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
11. Exercise 8 – Automatically respond to a text message		
11.1 Introduction 11.2 Objective 11.3 Requirements 11.4 The Flowcode program in detail 11.5 What to do 11.6 Further work	In this exercise the two previous programs are combined to create a useful application, capable of reading an incoming text message, and creating and sending a reply message. Again, this program can be modified to create a remote control device. The introduction gives an overview of the stages involved in splitting up the received message into relevant parts, and then replying to the message. The resulting program, though based on the two previous ones is relatively long, and so four diagrams are provided, giving detailed information about the required program. Both the Tx_Command and the Rx_Message macros are used within the main program. There are no new Flowcode functions in this program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30

Flowcode solutions to exercises

These solutions describe the Flowcode programs contained on the accompanying CD-ROM. The purpose is to allow instructors to supply students with either parts of, or outlines of, the programs so that the students can complete them.

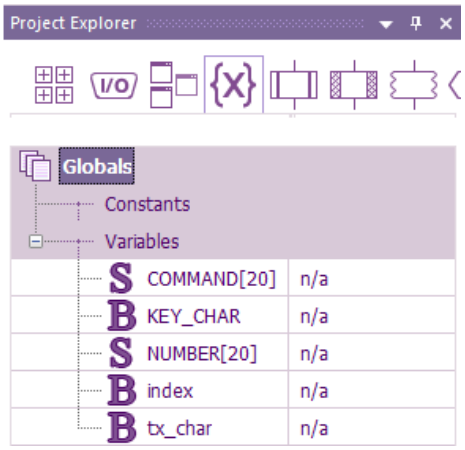
Two macros are used repeatedly throughout the exercises:

‘Tx Command’ – transmit a character;
‘Rx Message’ – receive a character.

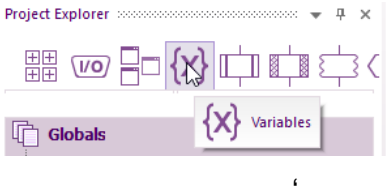
These macros are shown on the following pages.

Thereafter, the solutions omit the instruction to ‘Open the Properties box’ for clarity.

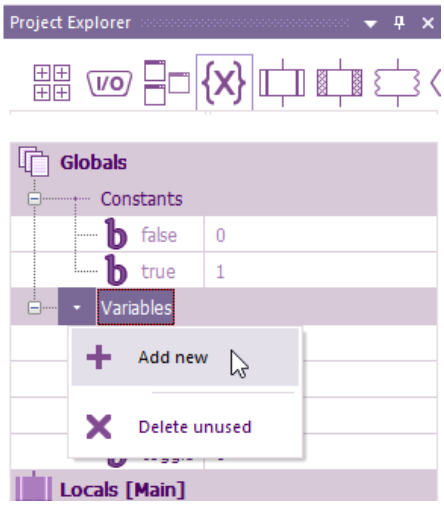
Icons that are left blank have been configured in previous programs.

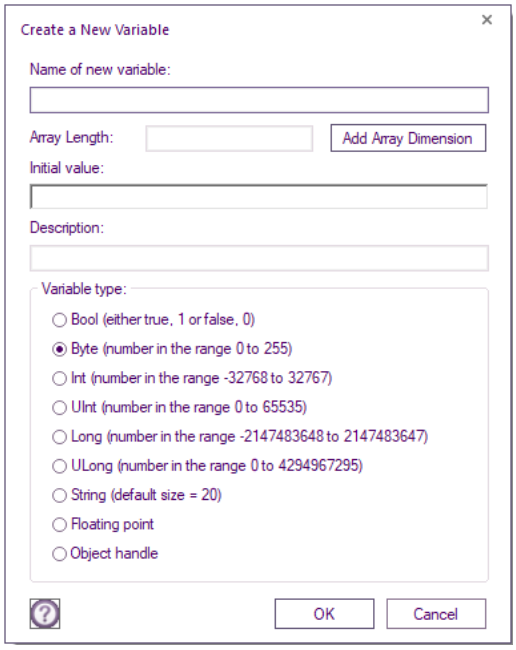


There are no instructions to create specific variables. However, it should be clear from the configuration details when a particular variable is needed. Variables are created either by clicking on the ‘Variables...’ button inside the Properties box, or by clicking the Variables icon, within the ‘Project Explorer’:



To create a new variable, click on the ‘Add New Variable...’ option:

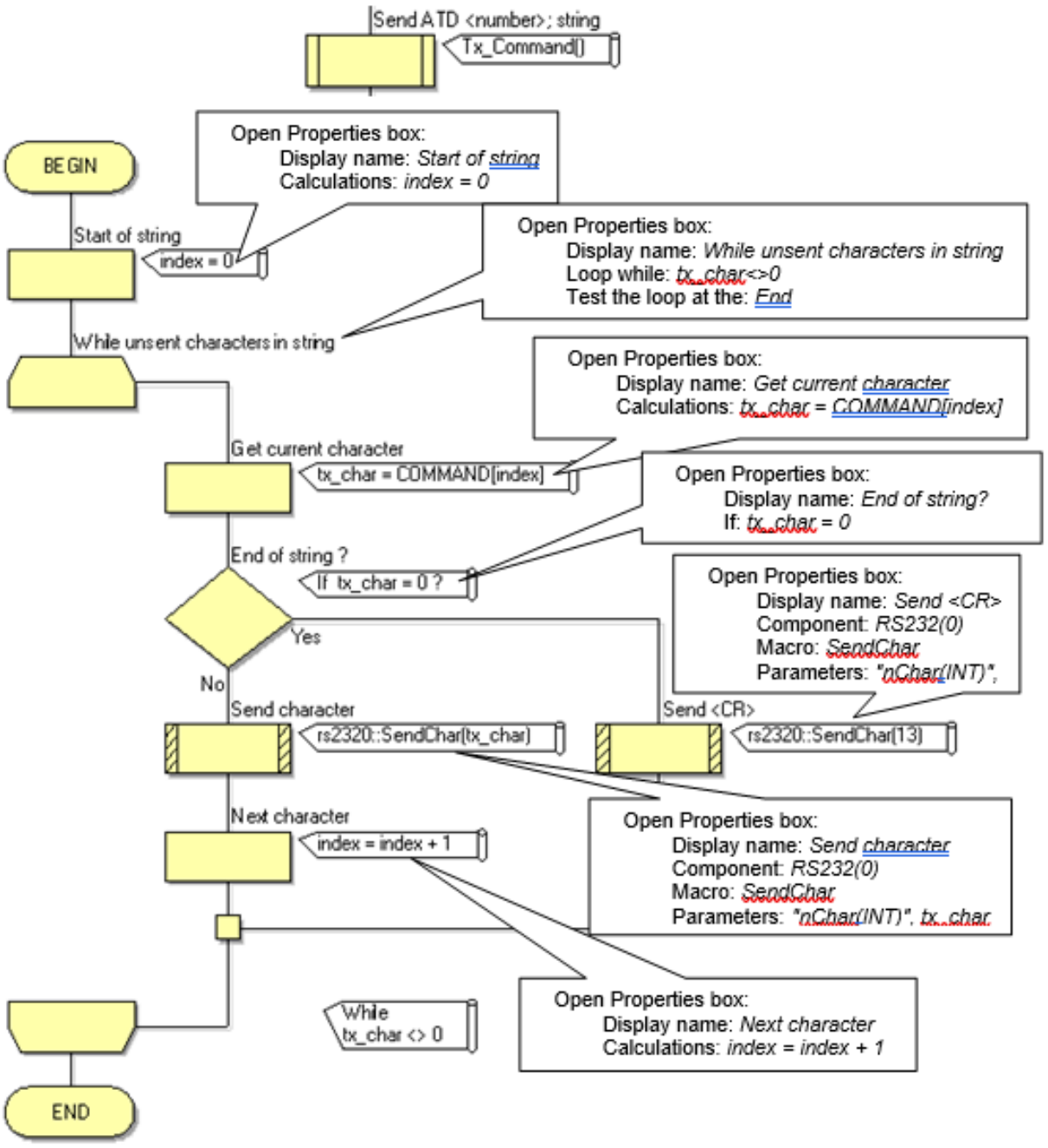




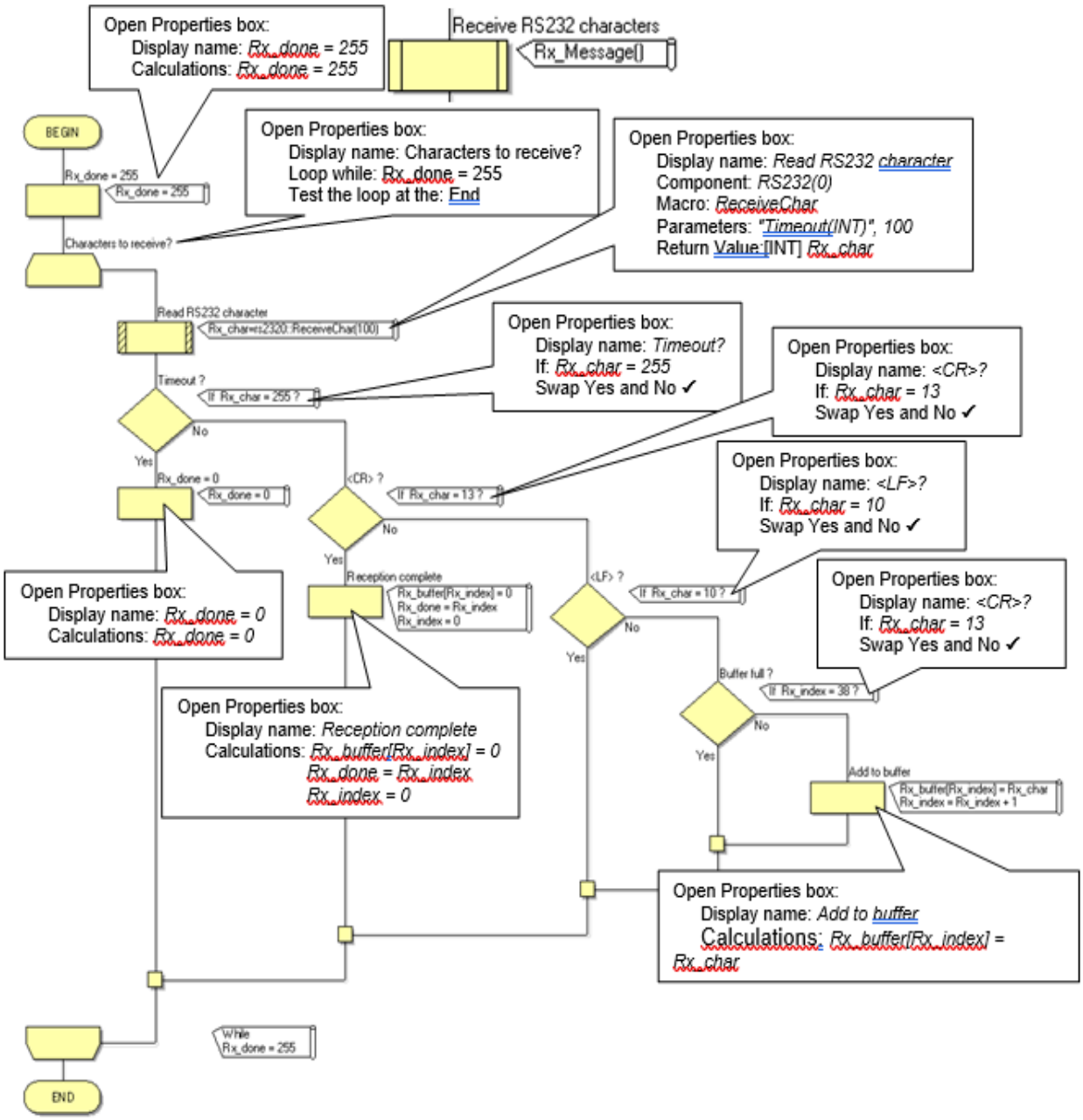
this then opens the window on the left
The name of the new variable can be typed in, and the variable type can then be selected.

Flowcode solutions to exercises

Macro 1 – transmit a command string and add a <CR>



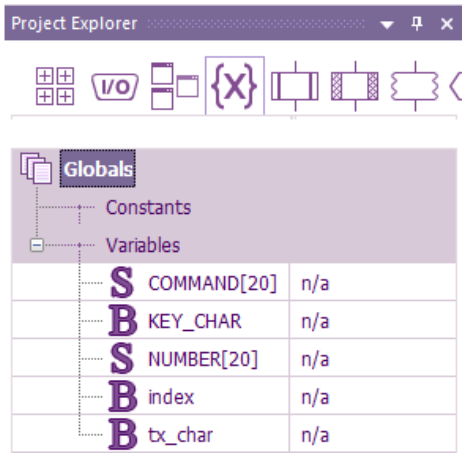
Macro 2 – build a string of received characters.



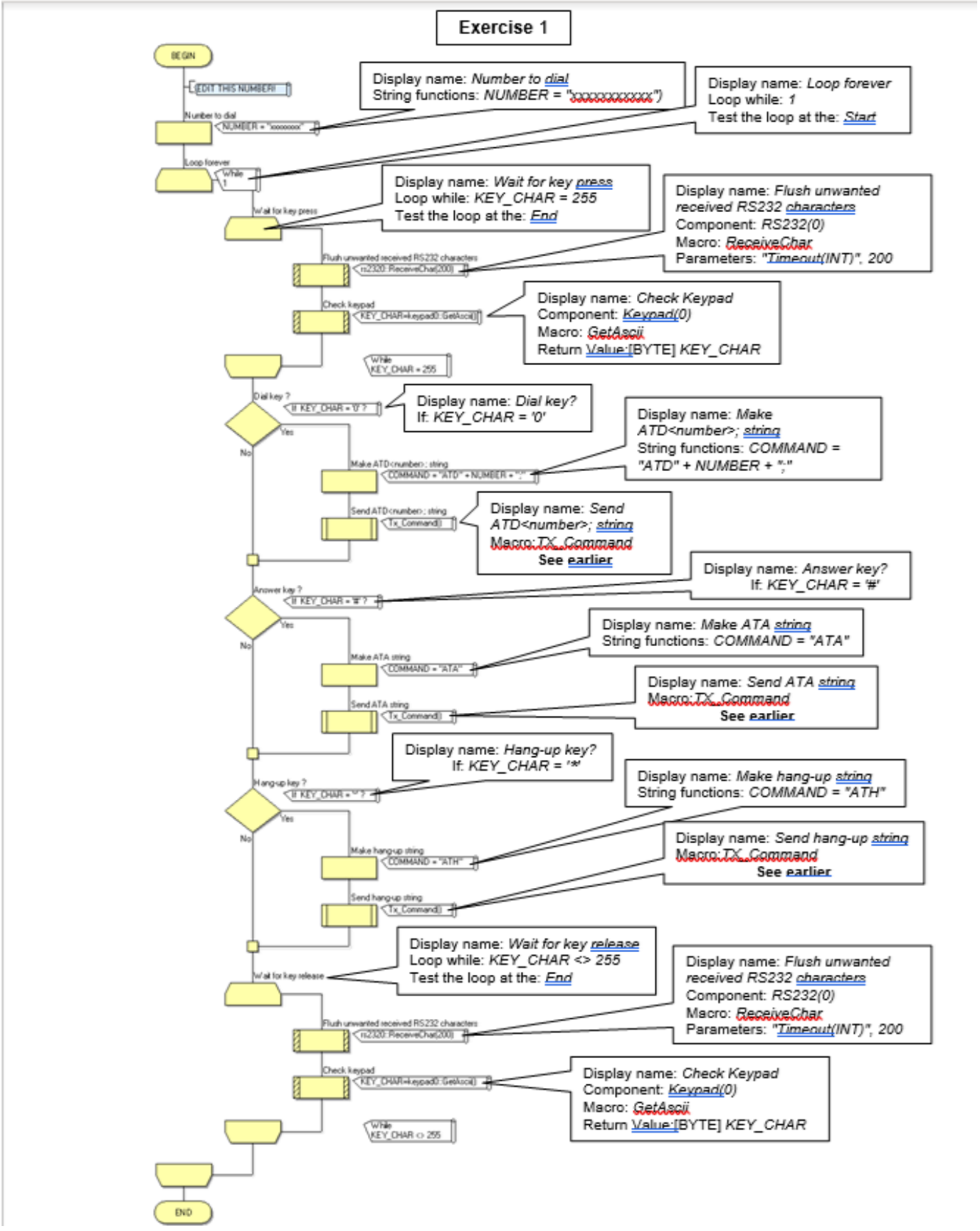
Exercise 1:

shows how to use Flowcode to send strings of characters, using the RS232 component, to transmit AT commands to control the GSM modem and replicate a simple telephone.

The program uses the following variables:



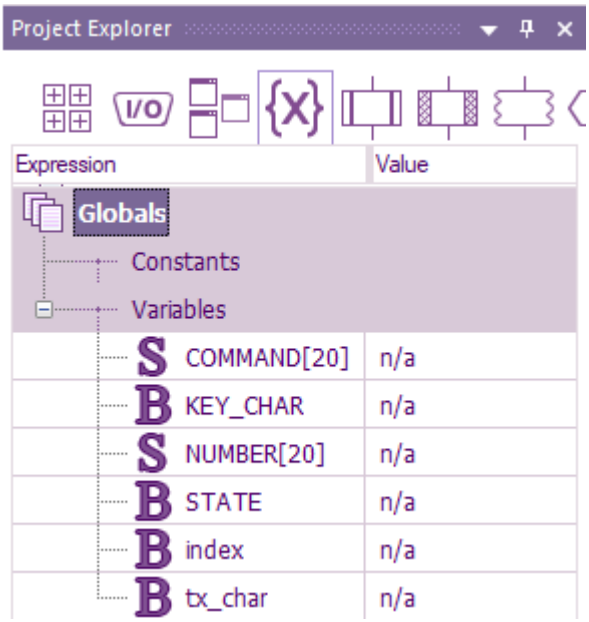
The structure of the program is detailed in the following diagram on the next page:


Exercise 1
BEGIN
EDIT THIS NUMBER!!
Number to dial
NUMBER = "xxxxxxxx"
Loop forever
While 1
Wait for key press
Flush unwanted received RS232 characters
rs232i0 ReceiveChar(200)
Check keypad
KEY_CHAR=Keypad0 GetAscii()
While KEY_CHAR = 255
Dial key ?
If KEY_CHAR = '0' ?
Yes
No
Make ATD<number>; string
COMMAND = "ATD" + NUMBER + ";"
Send ATD<number>; string
Tx_Command()
Answer key ?
If KEY_CHAR = '#' ?
Make ATA string
COMMAND = "ATA"
Send ATA string
Tx_Command()
Hang-up key ?
If KEY_CHAR = '*' ?
Make hang-up string
COMMAND = "ATH"
Send hang-up string
Tx_Command()
Wait for key release
While KEY_CHAR <> 255
END
Display name: Number to dial
String functions: NUMBER = "xxxxxxxxxx")
Display name: Loop forever
Loop while: 1
Test the loop at the: Start
Display name: Wait for key press
Loop while: KEY_CHAR = 255
Test the loop at the: End
Display name: Flush unwanted received RS232 characters
Component: RS232(0)
Macro: ReceiveChar
Parameters: "Timeout(INT)", 200
Display name: Check Keypad
Component: Keypad(0)
Macro: GetAscii
Return Value:[BYTE] KEY_CHAR
Display name: Dial key?
If: KEY_CHAR = '0'
Display name: Make ATD<number>; string
String functions: COMMAND = "ATD" + NUMBER + ";"
Display name: Send ATD<number>; string
Macro:TX_Command
See earlier
Display name: Answer key?
If: KEY_CHAR = '#'
Display name: Make ATA string
String functions: COMMAND = "ATA"
Display name: Send ATA string
Macro:TX_Command
See earlier
Display name: Hang-up key?
If: KEY_CHAR = '*'
Display name: Make hang-up string
String functions: COMMAND = "ATH"
Display name: Send hang-up string
Macro:TX_Command
See earlier
Display name: Wait for key release
Loop while: KEY_CHAR <> 255
Test the loop at the: End
Display name: Flush unwanted received RS232 characters
Component: RS232(0)
Macro: ReceiveChar
Parameters: "Timeout(INT)", 200
Display name: Check Keypad
Component: Keypad(0)
Macro: GetAscii
Return Value:[BYTE] KEY_CHAR

Exercise 2:

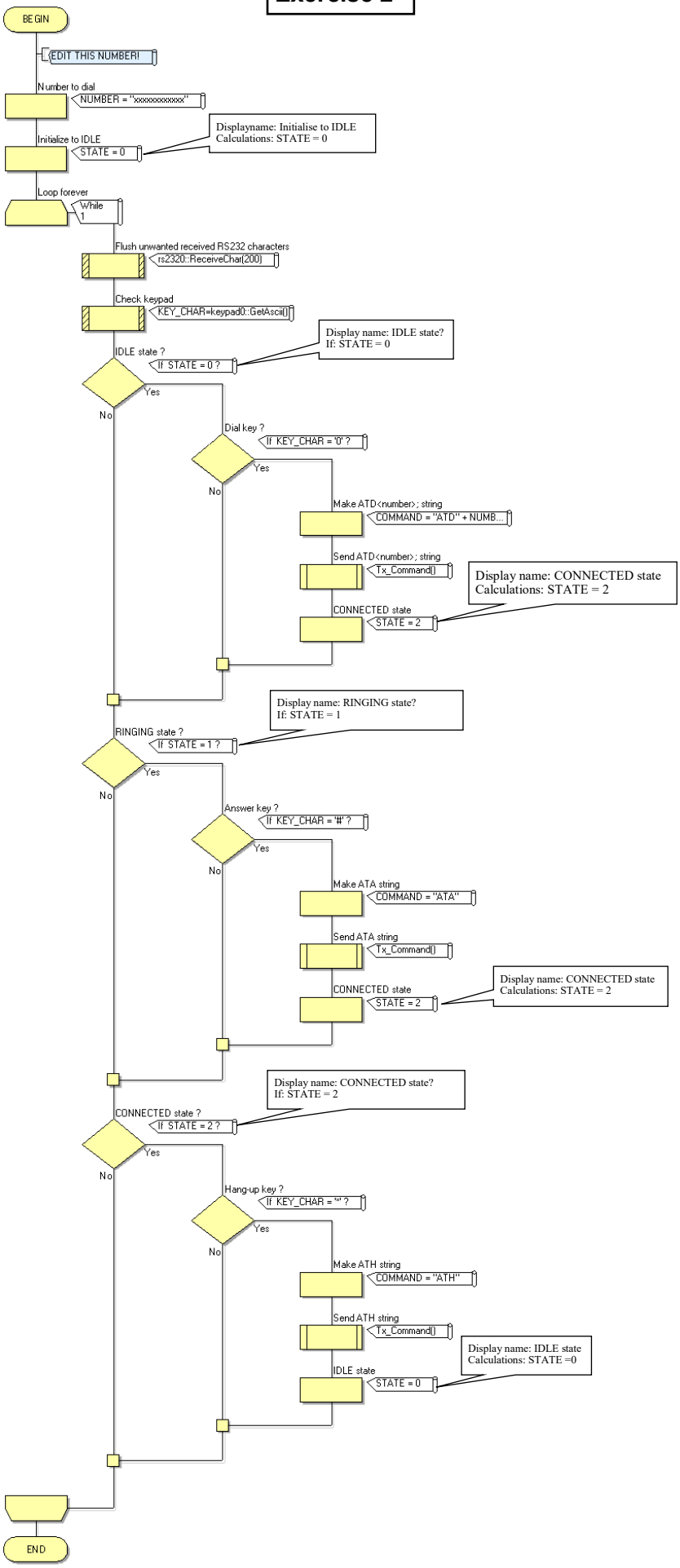
is a more complex program that introduces a state machine model to improve the operation of the telephone.

The program uses the following variables:



The structure of the program is detailed in the following diagram on the next page:

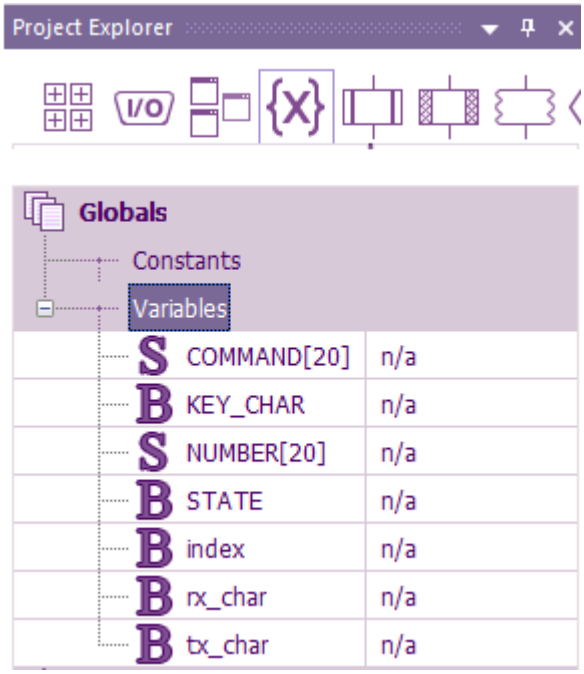
Exercise 2


BEGIN
EDIT THIS NUMBER!
Number to dial
NUMBER = "xxxxxxxxxxx"
Initialize to IDLE
STATE = 0
Displayname: Initialise to IDLE
Calculations: STATE = 0
Loop forever
While 1
Flush unwanted received RS232 characters
rs2320::ReceiveChar(200)
Check keypad
KEY_CHAR=keypad0::GetAscii()
IDLE state ?
If STATE = 0 ?
Display name: IDLE state?
If: STATE = 0
Yes
No
Dial key ?
If KEY_CHAR = '0' ?
Yes
No
Make ATD<number>; string
COMMAND = "ATD" + NUMB...
Send ATD<number>; string
Tx_Command()
CONNECTED state
STATE = 2
Display name: CONNECTED state
Calculations: STATE = 2
RINGING state ?
If STATE = 1 ?
Display name: RINGING state?
If: STATE = 1
Yes
No
Answer key ?
If KEY_CHAR = '#' ?
Yes
No
Make ATA string
COMMAND = "ATA"
Send ATA string
Tx_Command()
CONNECTED state
STATE = 2
Display name: CONNECTED state
Calculations: STATE = 2
CONNECTED state ?
If STATE = 2 ?
Display name: CONNECTED state?
If: STATE = 2
Yes
No
Hang-up key ?
If KEY_CHAR = '*' ?
Yes
No
Make ATH string
COMMAND = "ATH"
Send ATH string
Tx_Command()
IDLE state
STATE = 0
Display name: IDLE state
Calculations: STATE =0
END

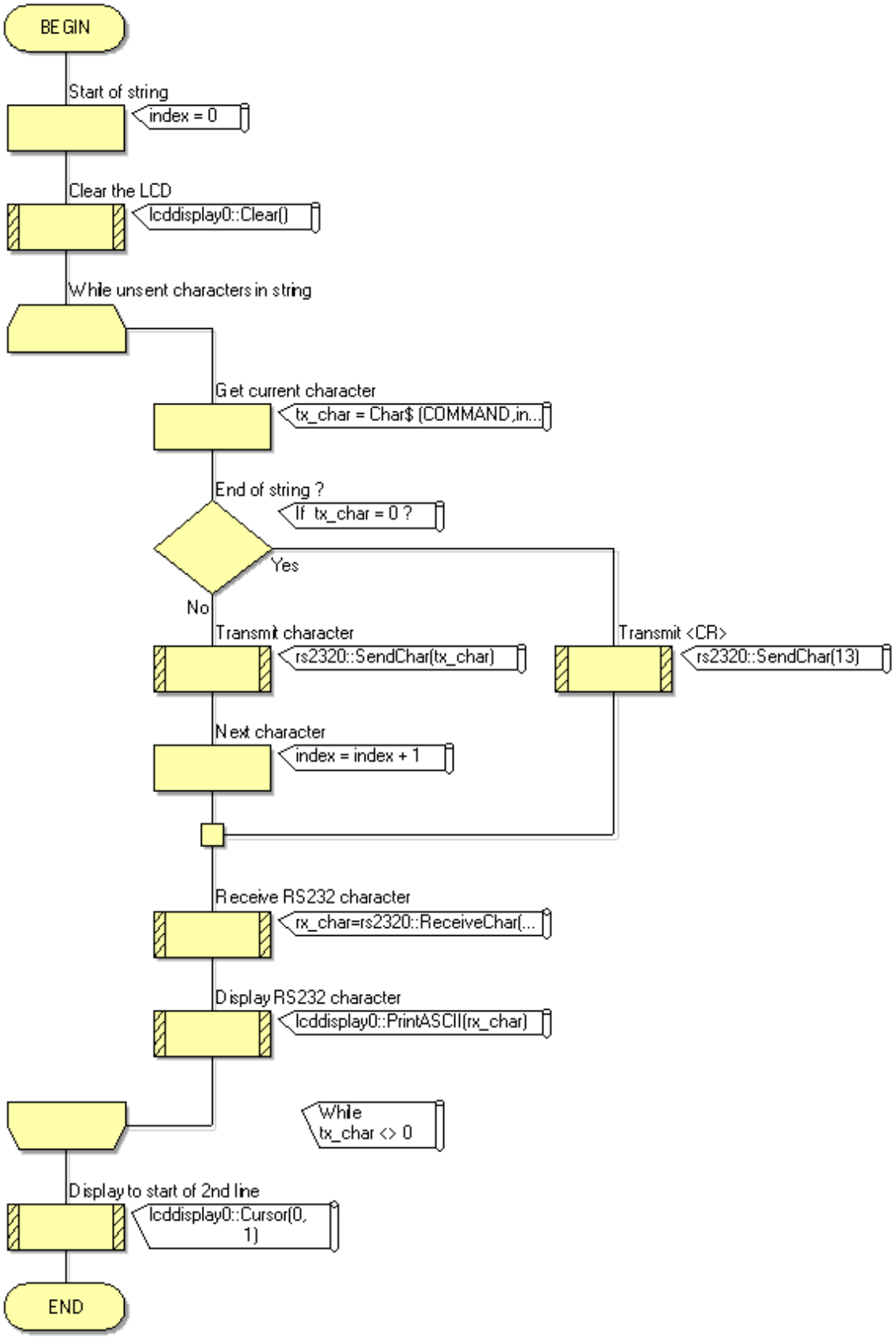
Exercise 3:

displays messages generated by the modem during operation, and introduces methods required to handle them including the ability to answer an incoming call.

The program uses the following variables:

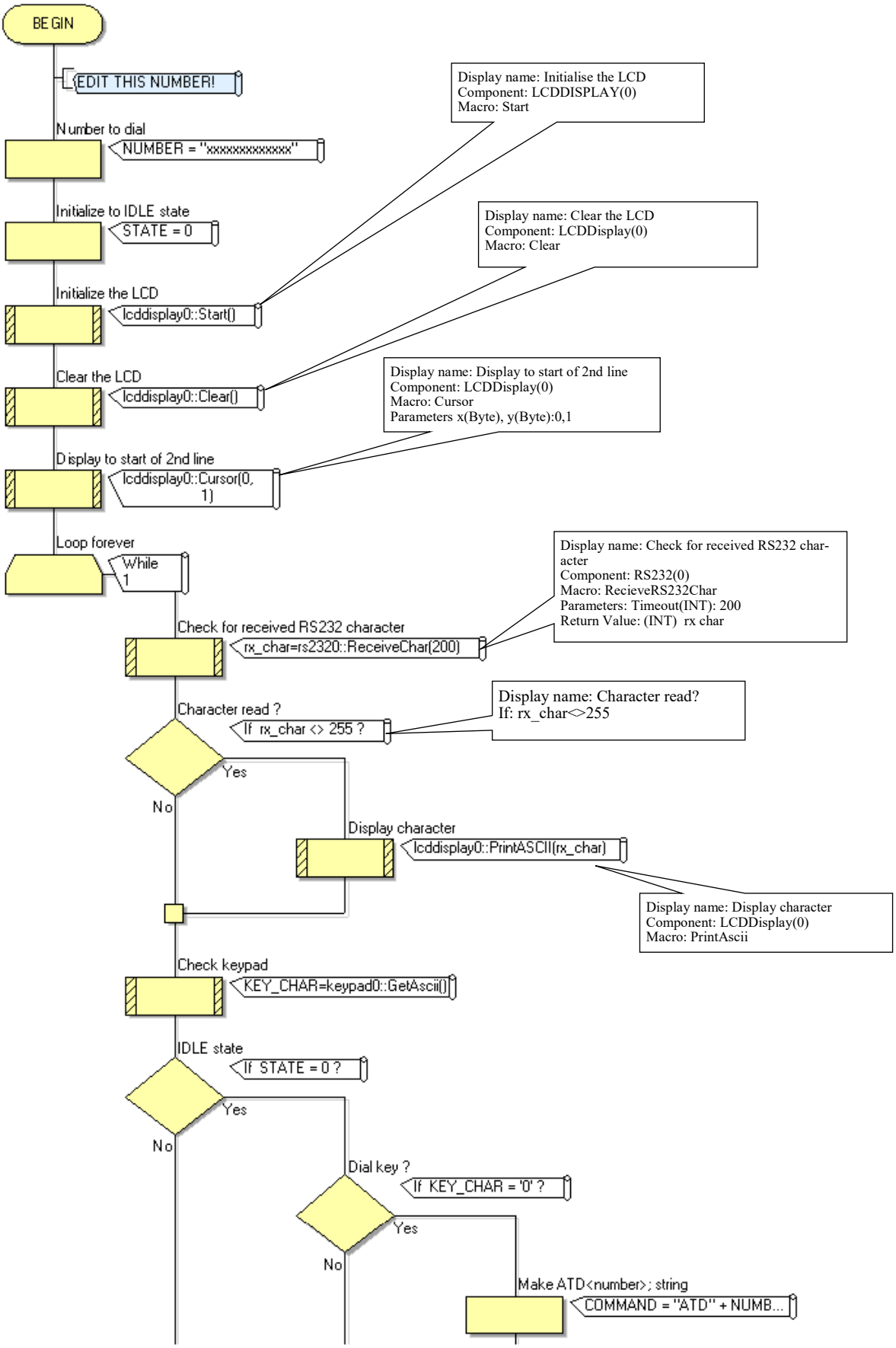


The modified Tx_Command macro, used in exercise 3, is shown below.



The structure of the program is detailed in the following diagram:

Exercise 3

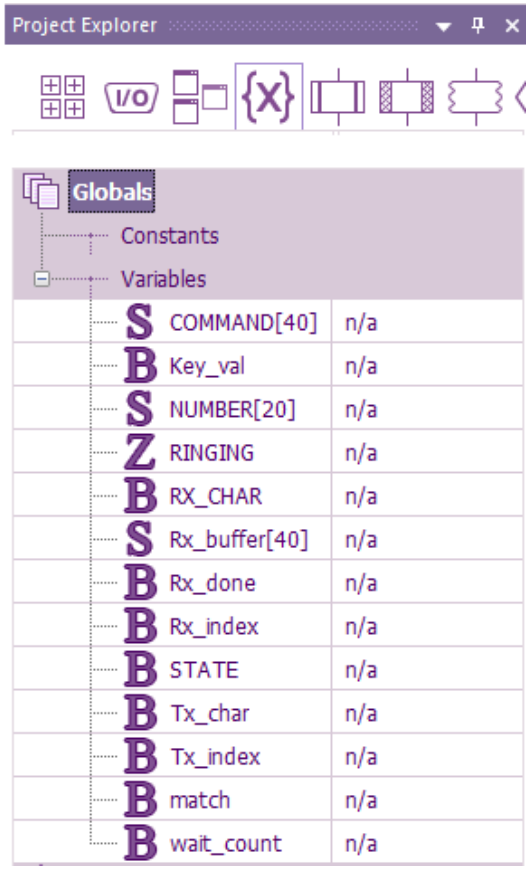


The remainder of the program is identical to that in exercise 2

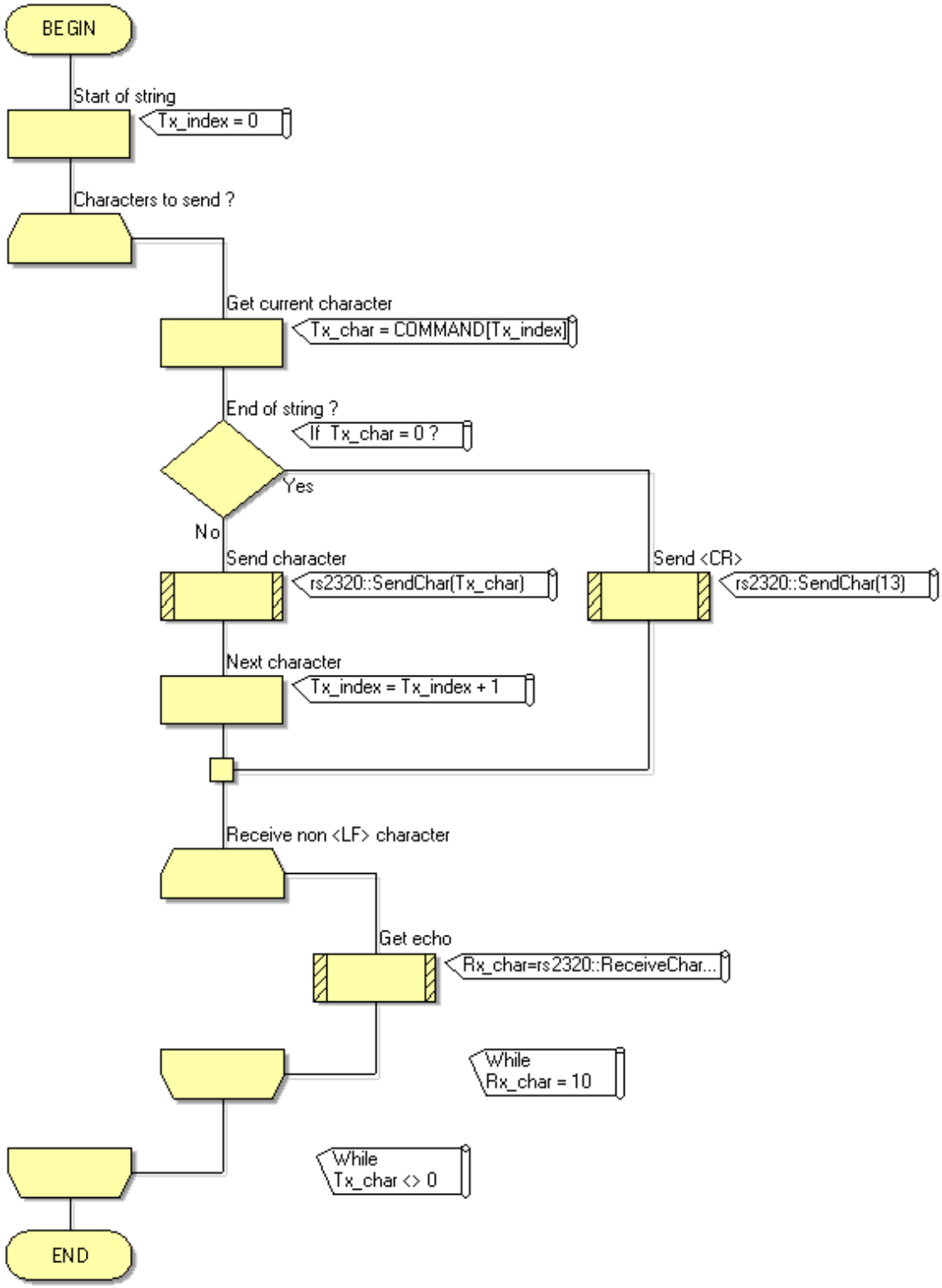
Exercise 4:

completes the operation of the basic telephone; including detection of the modem ringing.

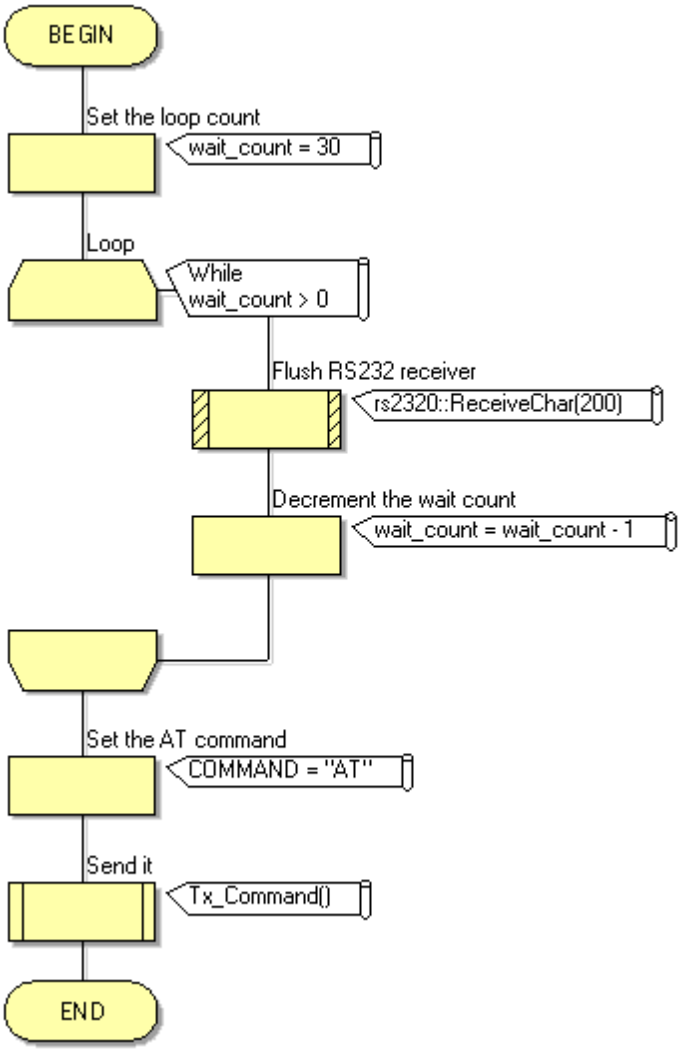
The program uses the following variables:



The modified Tx_Command macro, used in exercise 4, is shown below.

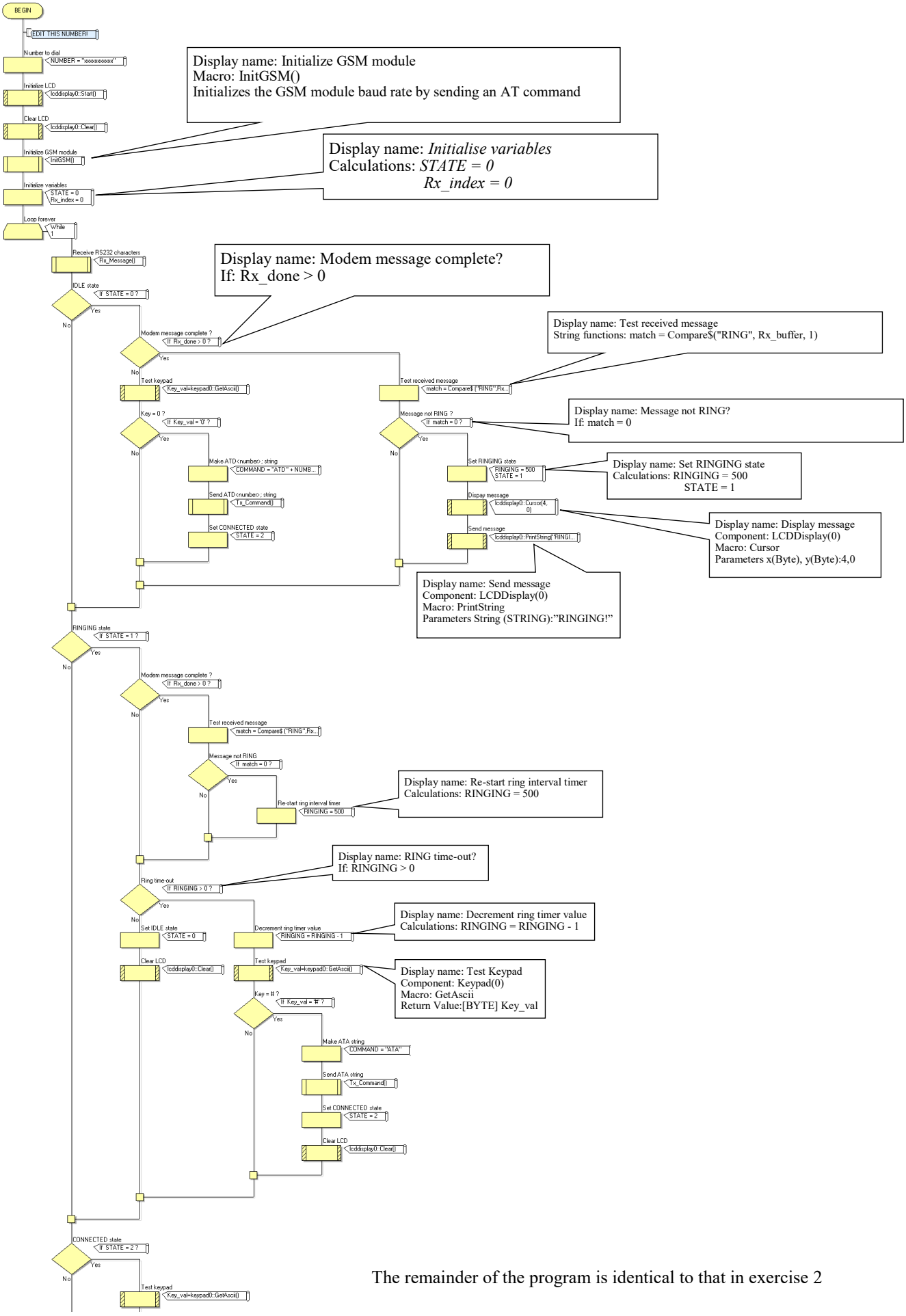


In previous exercises we have been issuing commands to the modem. In exercise 4 we are waiting for an incoming RING. Some modems have auto baud rate detection so we first need to send an AT command to the modem in order to initialize this to our working rate of 9600 baud. To do this we introduce a new macro InitGSM(). This macro also gives the modem a few seconds to start-up before issuing the AT command.



The structure of the main program is detailed in the following diagram on the next page:

Exercise 4

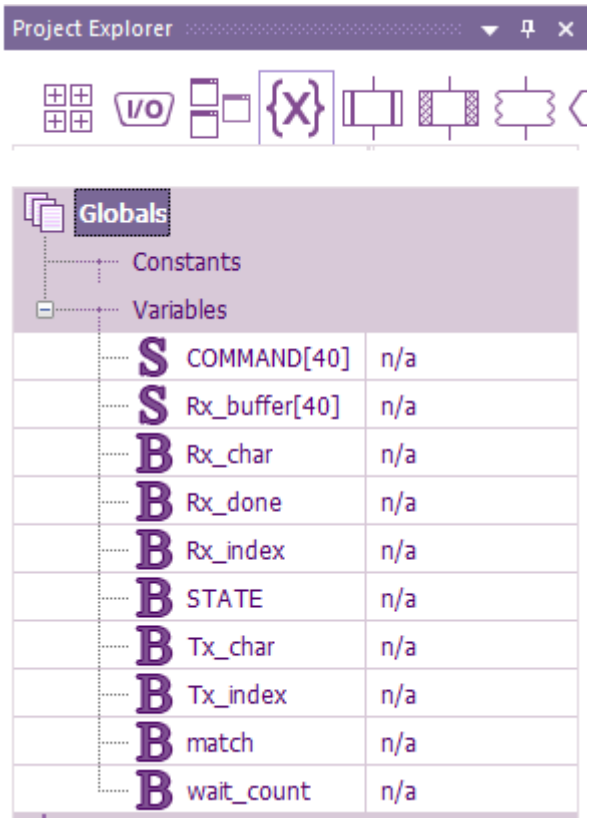


The remainder of the program is identical to that in exercise 2

Exercise 5:

uses a selection of modem messages to automate the call answer and hang-up processes, to produce a remote listening or public address device.

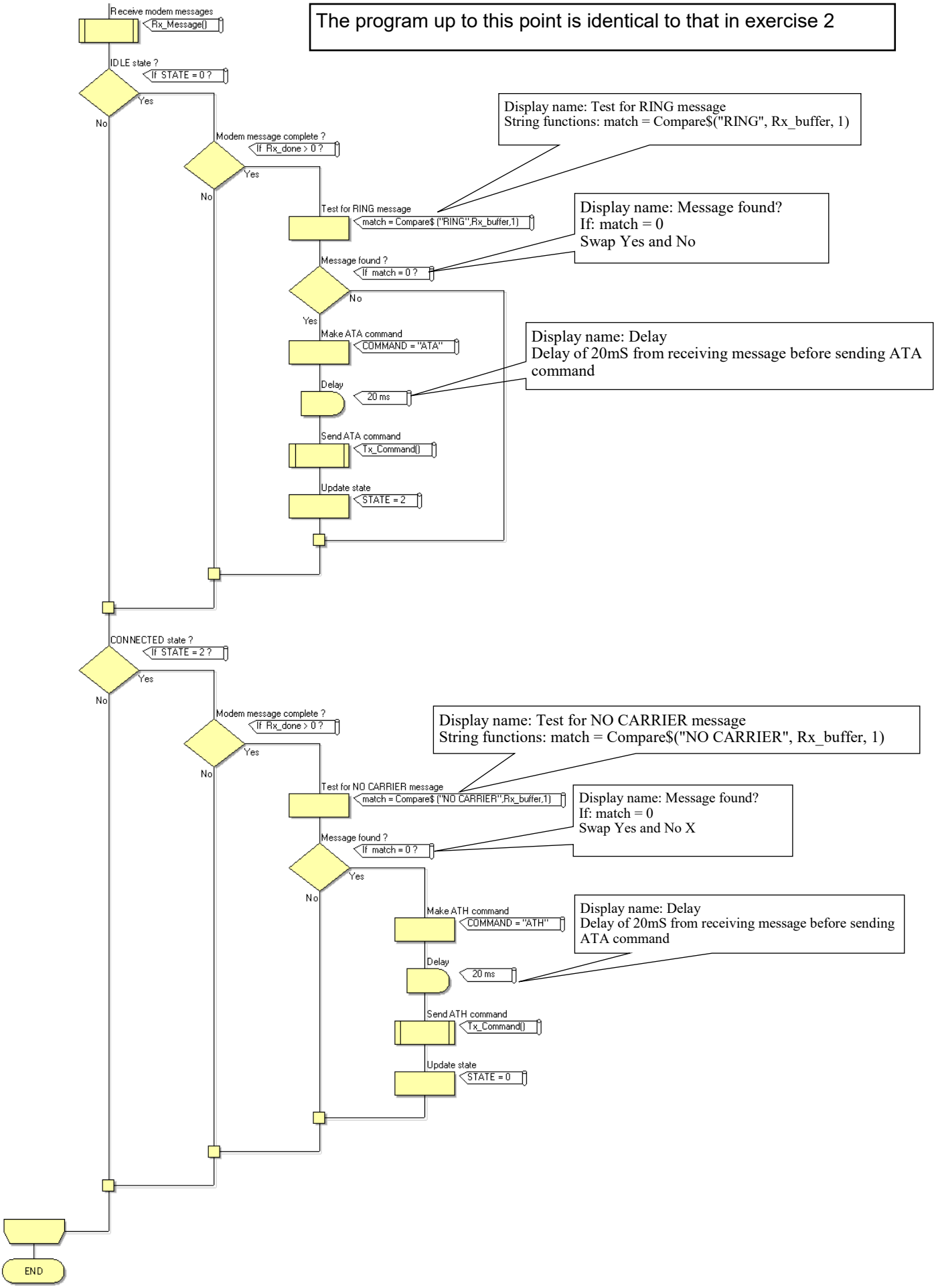
The program uses the following variables:



The structure of the program is detailed in the following diagram on the next page:

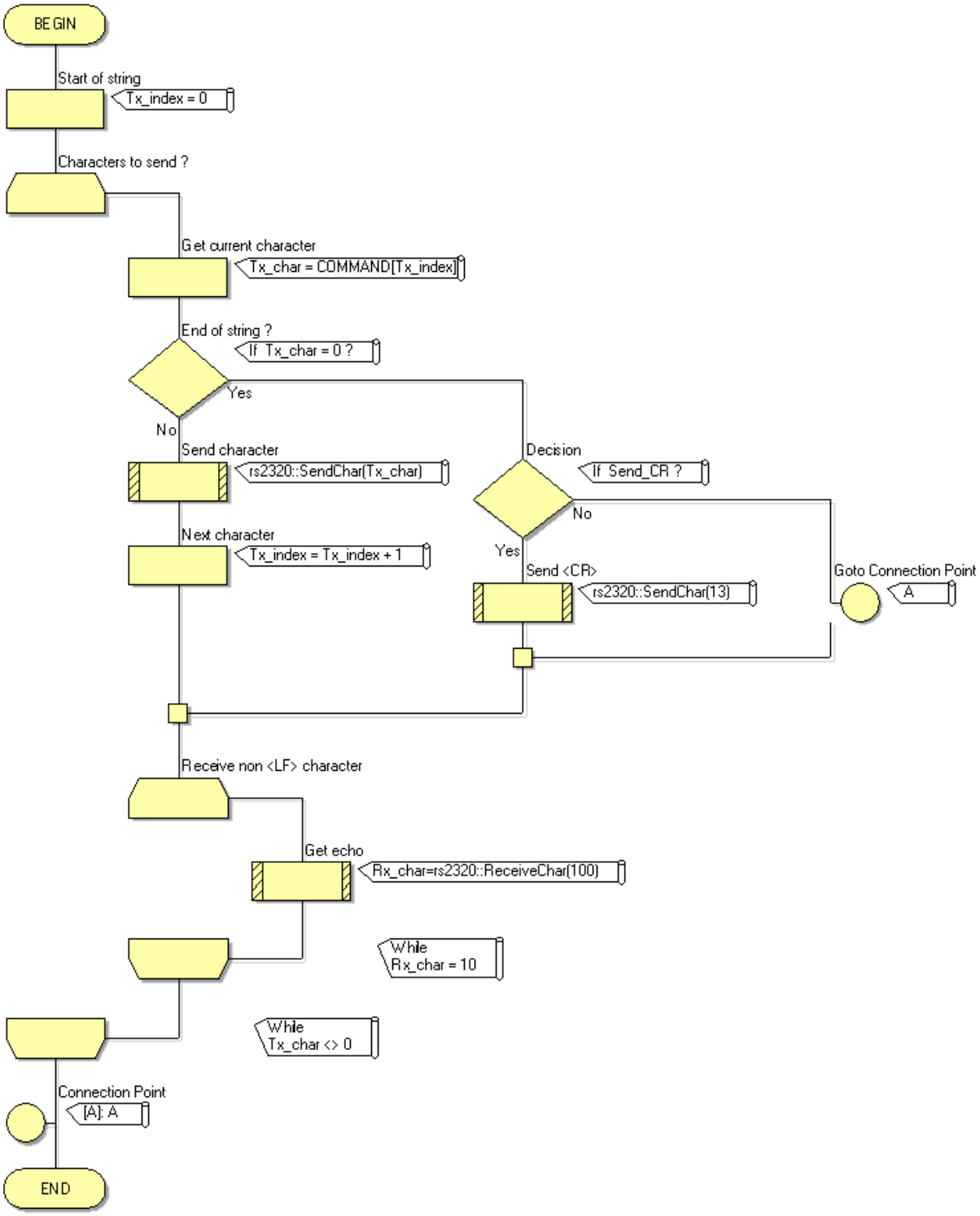
Exercise 5

The program up to this point is identical to that in exercise 2

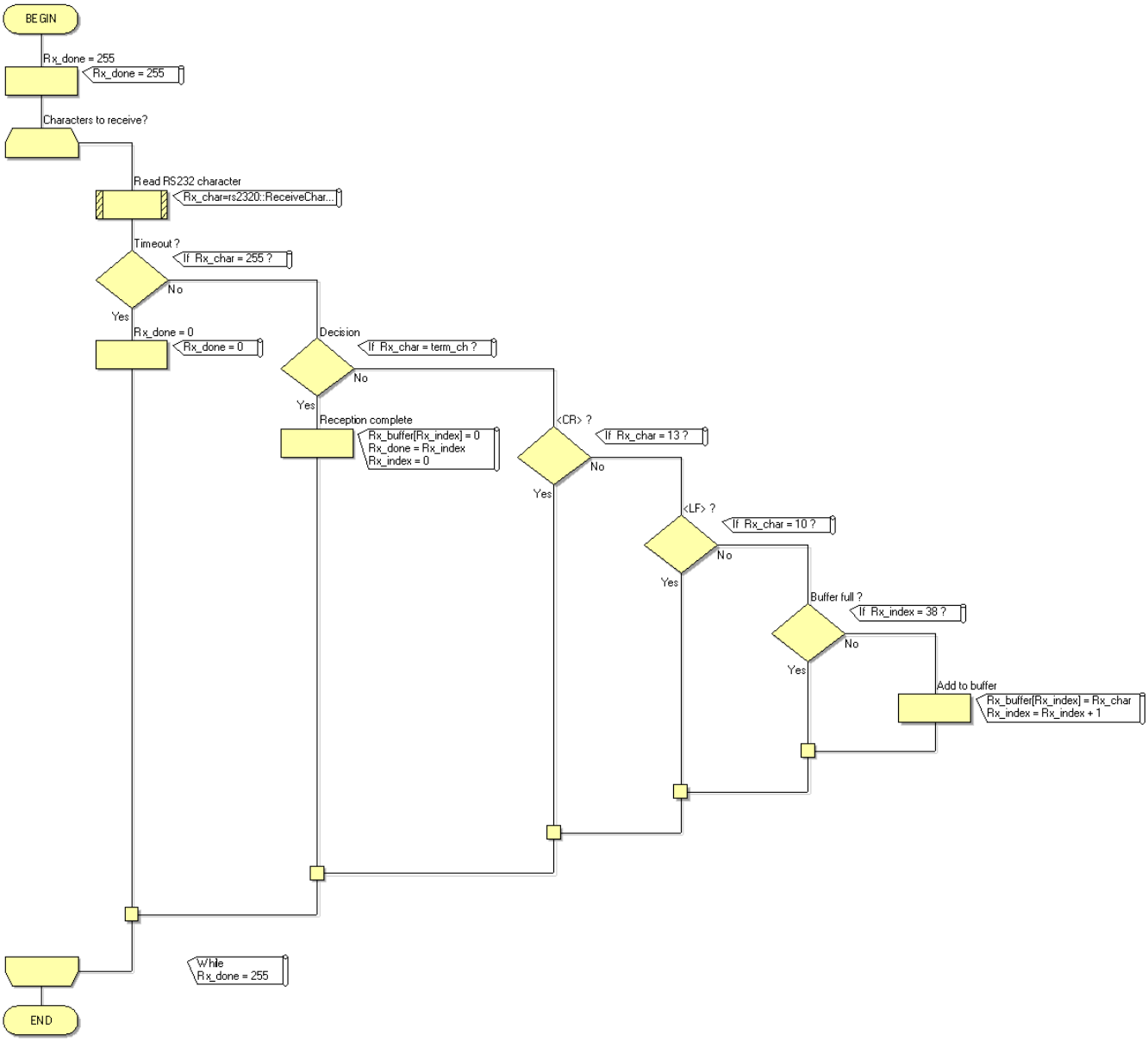


For the remaining exercises, which focus on the transmission of text messages, the two macros ‘Tx_Command’ and ‘Rx_message’ are modified slightly, as shown below.
In particular, the variable ‘index’ is replaced by two – ‘Tx_index’ and ‘Rx_index’.

Tx_Command



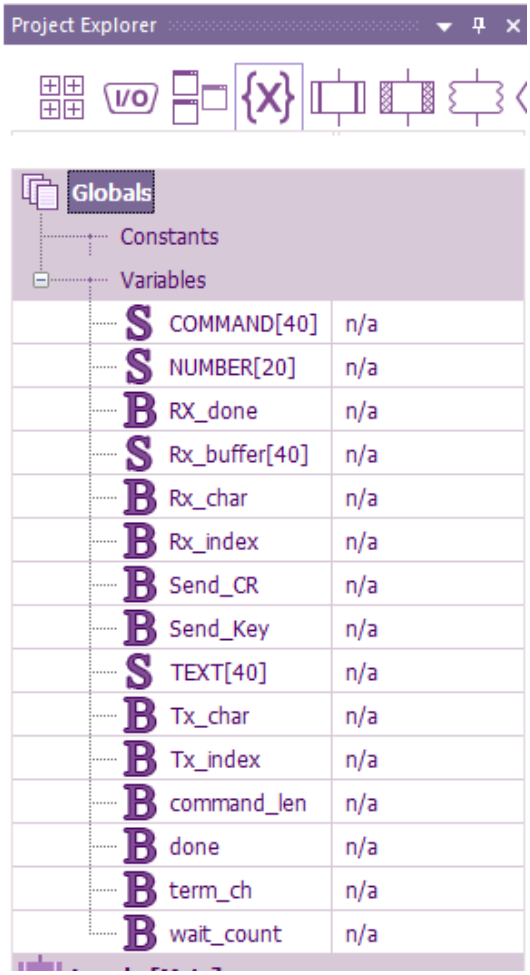
Rx_Message


BEGIN
Rx_done = 255
Rx_done = 255
Characters to receive?
Read RS232 character
Rx_char=rs2320::ReceiveChar...
Timeout ?
If Rx_char = 255 ?
No
Yes
Rx_done = 0
Rx_done = 0
Decision
If Rx_char = term_ch ?
No
Yes
Reception complete
Rx_buffer[Rx_index] = 0
Rx_done = Rx_index
Rx_index = 0
<CR> ?
If Rx_char = 13 ?
No
Yes
<LF> ?
If Rx_char = 10 ?
No
Yes
Buffer full ?
If Rx_index = 38 ?
No
Yes
Add to buffer
Rx_buffer[Rx_index] = Rx_char
Rx_index = Rx_index + 1
While
Rx_done = 255
END

Exercise 6:

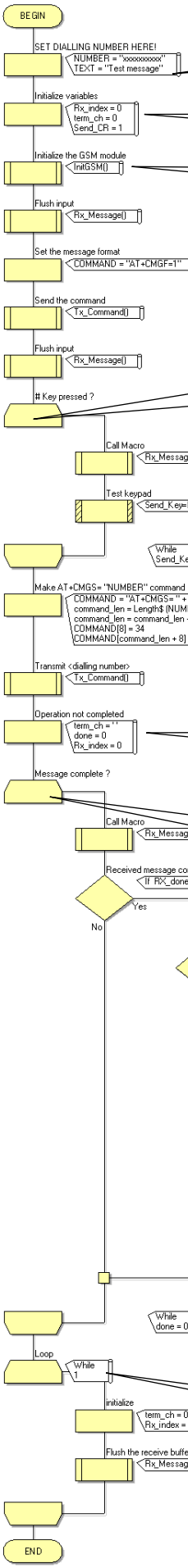
shows how to use Flowcode to send a simple text message at the touch of a button.

The program uses the following variables:



The structure of the program is detailed in the following diagram on the next page:

Exercise 6



Display name: SET DIALLING NUMBER HERE!
String functions: NUMBER = "xxxxxxxxxx"
TEXT = "Test message"

Display name: Initialize variables
Calculations: Rx_index = 0 term_char = 0

Display name: Initialize the GSM module
Macro: InitGSM()

Display name: Set the message format
Calculations: COMMAND = “AT+CMGF=1”

Display name: # Key pressed?
Loop while: Send_Key <> '#'
Test loop at the: End

Display name: Test Keypad
Component: Keypad(0)
Macro: GetAscii
Return Value:[BYTE] Send_Key

Display name: Make AT+CMGS=”NUMBER” command
String functions: COMMAND = "AT+CMGS= " + NUMBER + " "
command_len = Length$ (NUMBER)
command_len = command_len + 1
COMMAND[8] = 34
COMMAND[command_len + 8] = 34

Display name: Operation not complete
Calculations: term_ch = ' '
Rx_index = 0

Display name: Message complete?
Loop while: done = 0
Test the loop at the: End

Display name: Received message complete?
If: RX_done

Display name: Received message =>?
If: Rx_buffer[0] = '>'

Display name: String Manipulation
String functions: COMMAND = TEXT + " "
command_len = Length$(COMMAND)

Display name: Calculation
Calculations: Send_CR = 0
command_len = command_len - 1
COMMAND[command_len] = 26

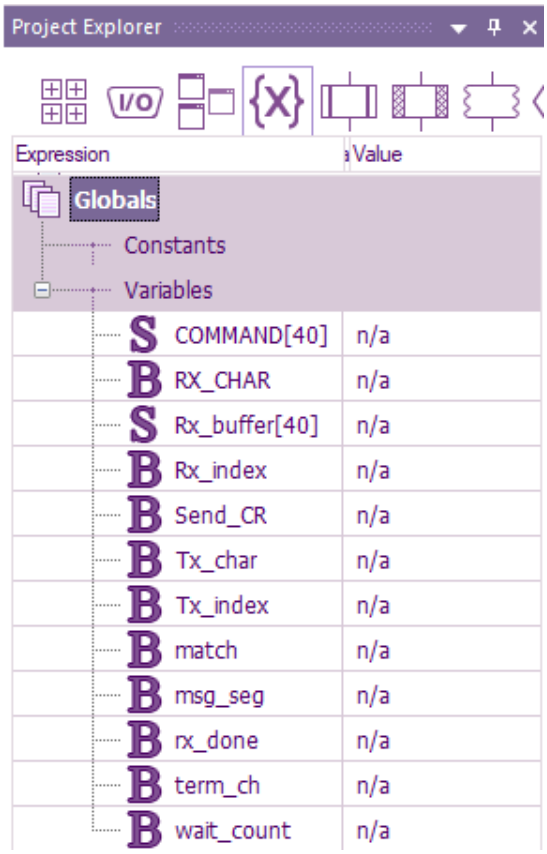
Display name: Operation complete
Calculations: done = 1

Loop whilst flushing the receive buffer

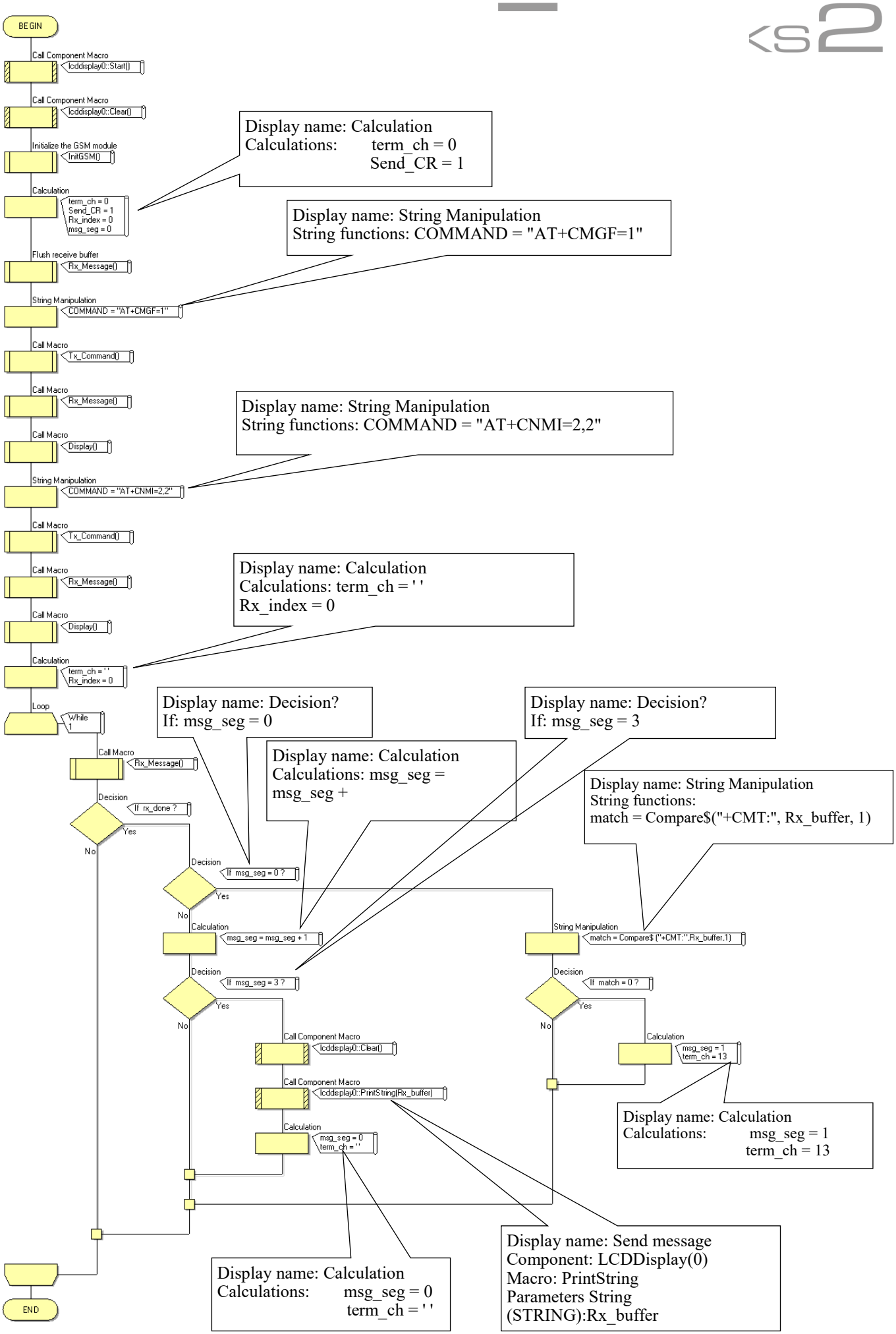
Exercise 7:

is a more complex program that uses Flowcode to receive a simple text message, extract it from the accompanying information, and display it on a screen.

The program uses the following variables:



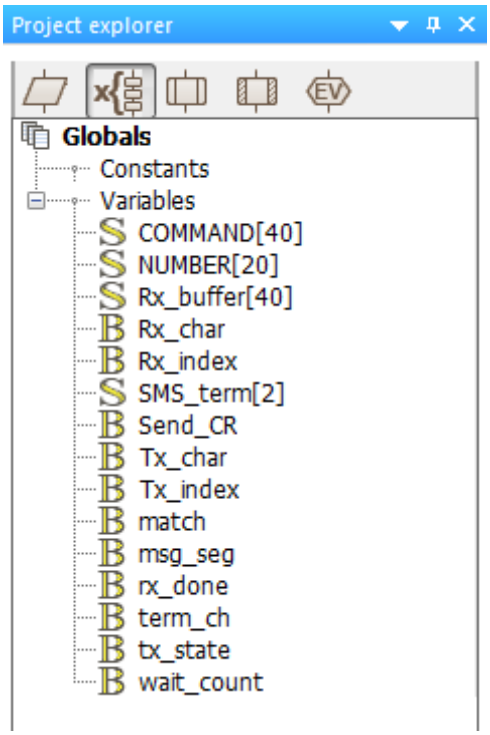
The structure of the program is detailed in the following diagram on the next page:


BEGIN
Call Component Macro
lcddisplay0::Start()
Call Component Macro
lcddisplay0::Clear()
Initialize the GSM module
InitGSM()
Calculation
term_ch = 0
Send_CR = 1
Rx_index = 0
msg_seg = 0
Display name: Calculation
Calculations: term_ch = 0
Send_CR = 1
Flush receive buffer
Rx_Message()
String Manipulation
COMMAND = "AT+CMGF=1"
Display name: String Manipulation
String functions: COMMAND = "AT+CMGF=1"
Call Macro
Tx_Command()
Call Macro
Rx_Message()
Call Macro
Display()
String Manipulation
COMMAND = "AT+CNMI=2,2"
Display name: String Manipulation
String functions: COMMAND = "AT+CNMI=2,2"
Call Macro
Tx_Command()
Call Macro
Rx_Message()
Call Macro
Display()
Calculation
term_ch = ''
Rx_index = 0
Display name: Calculation
Calculations: term_ch = ' '
Rx_index = 0
Loop
While 1
Call Macro
Rx_Message()
Decision
If rx_done ?
Yes
No
Display name: Decision?
If: msg_seg = 0
Decision
If msg_seg = 0 ?
Yes
No
Calculation
msg_seg = msg_seg + 1
Display name: Calculation
Calculations: msg_seg =
msg_seg +
Display name: Decision?
If: msg_seg = 3
Decision
If msg_seg = 3 ?
Yes
No
Call Component Macro
lcddisplay0::Clear()
Call Component Macro
lcddisplay0::PrintString(Rx_buffer)
Calculation
msg_seg = 0
term_ch = ''
Display name: Calculation
Calculations: msg_seg = 0
term_ch = ' '
Display name: Send message
Component: LCDDisplay(0)
Macro: PrintString
Parameters String
(STRING):Rx_buffer
String Manipulation
match = Compare$ ("+CMT:",Rx_buffer,1)
Display name: String Manipulation
String functions:
match = Compare$("+CMT:", Rx_buffer, 1)
Decision
If match = 0 ?
Yes
No
Calculation
msg_seg = 1
term_ch = 13
Display name: Calculation
Calculations: msg_seg = 1
term_ch = 13
END

Exercise 8:

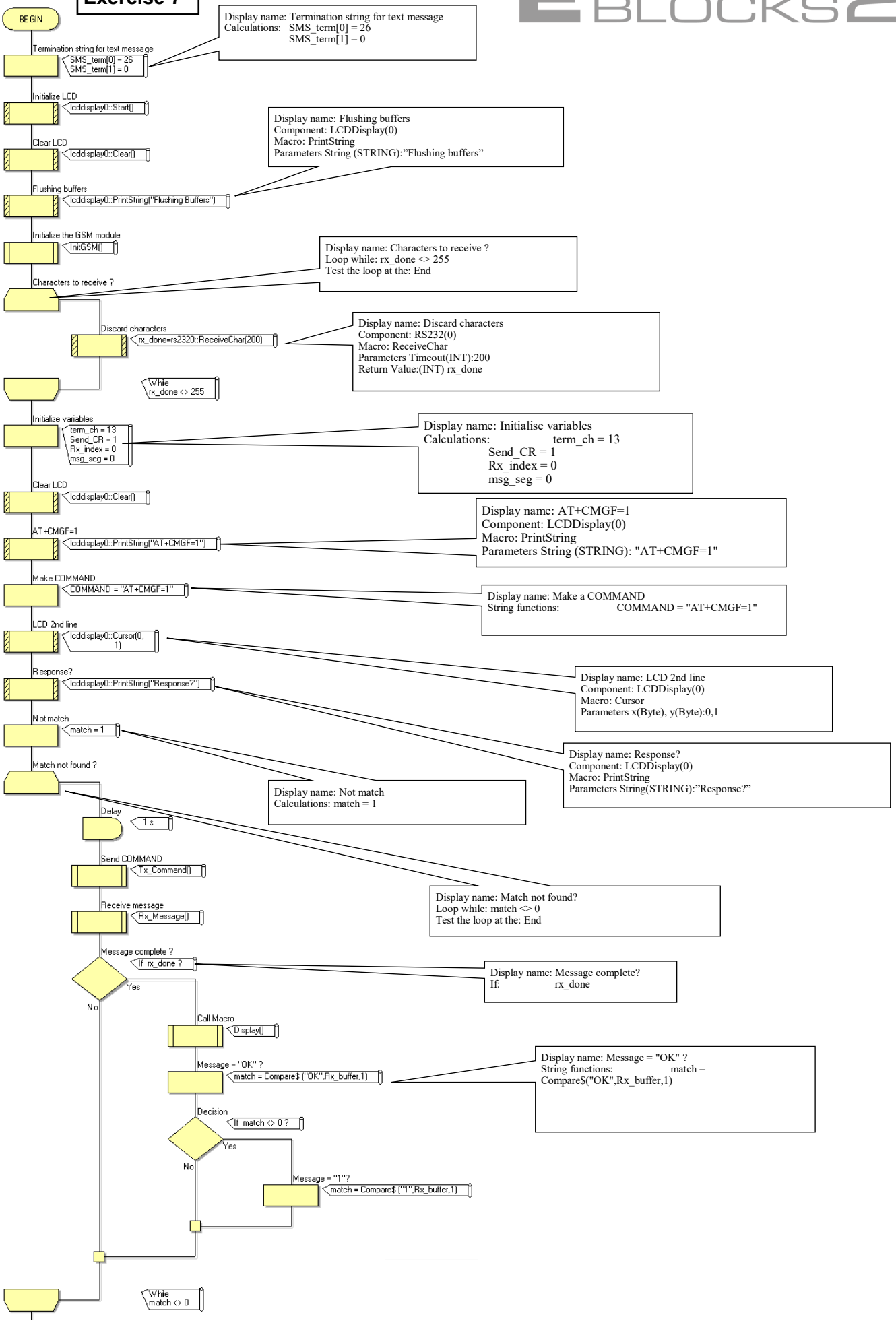
extracts data from a received text message and automatically replies with a response message.

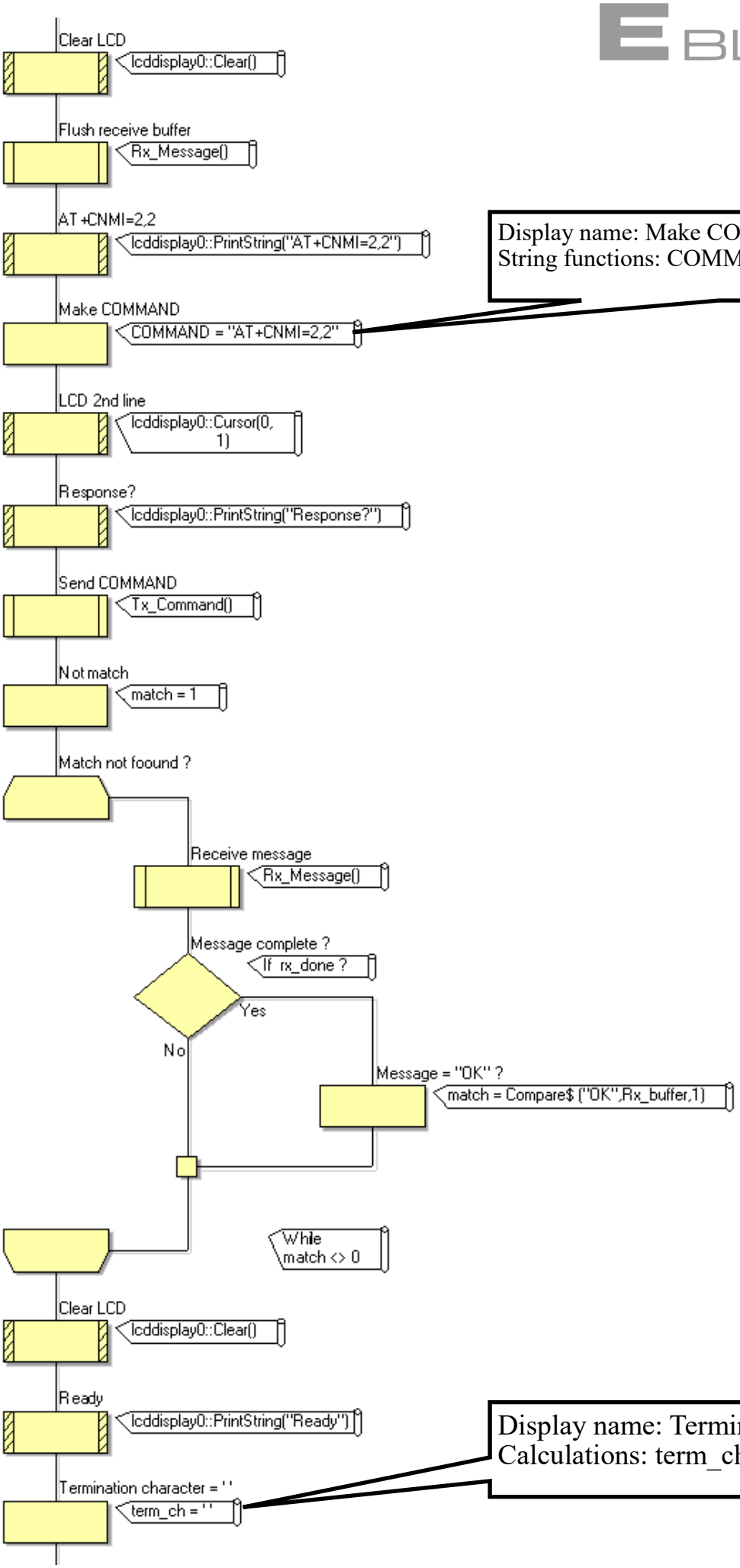
The program uses the following variables:



The structure of the program is detailed in the following diagrams:

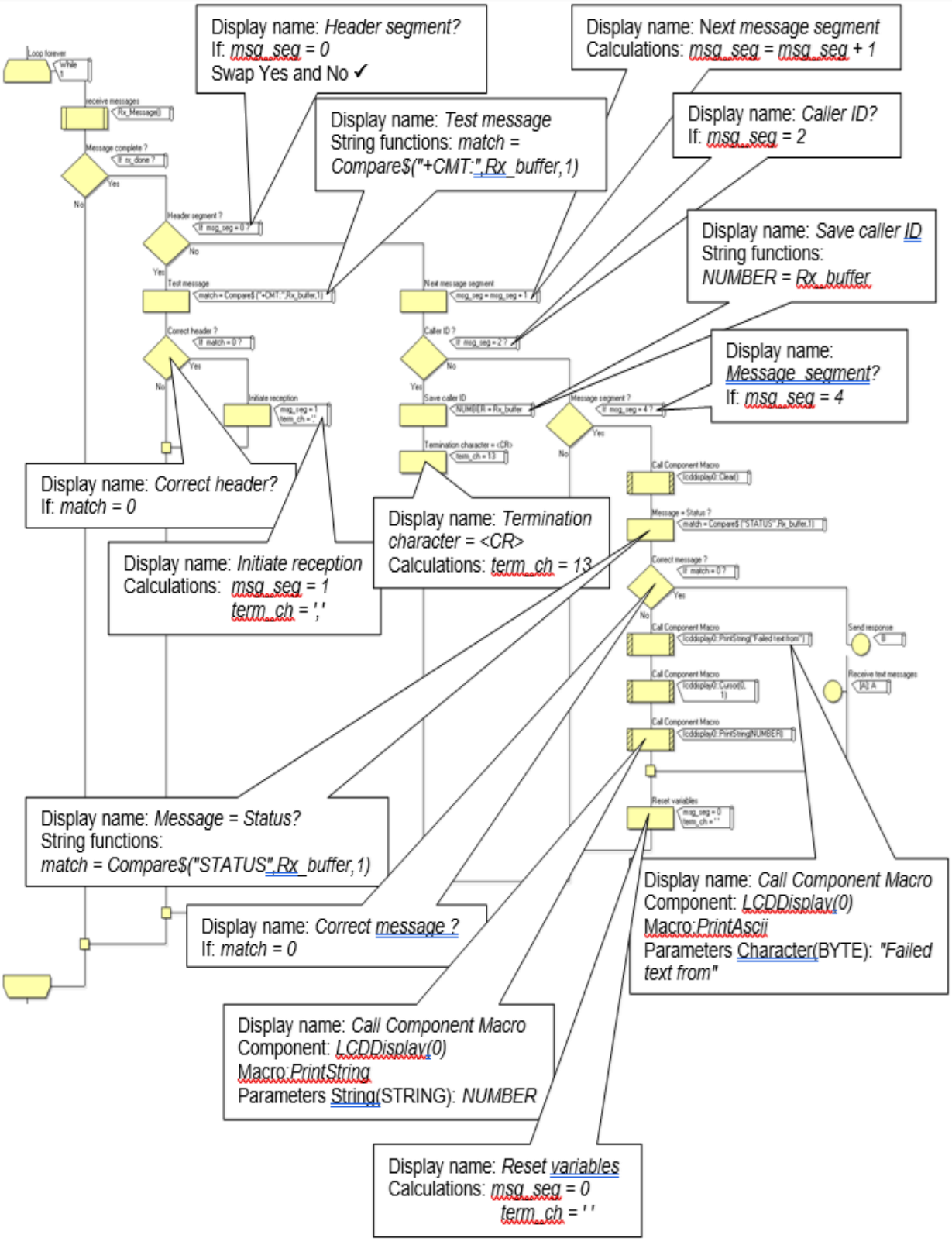
Exercise 7

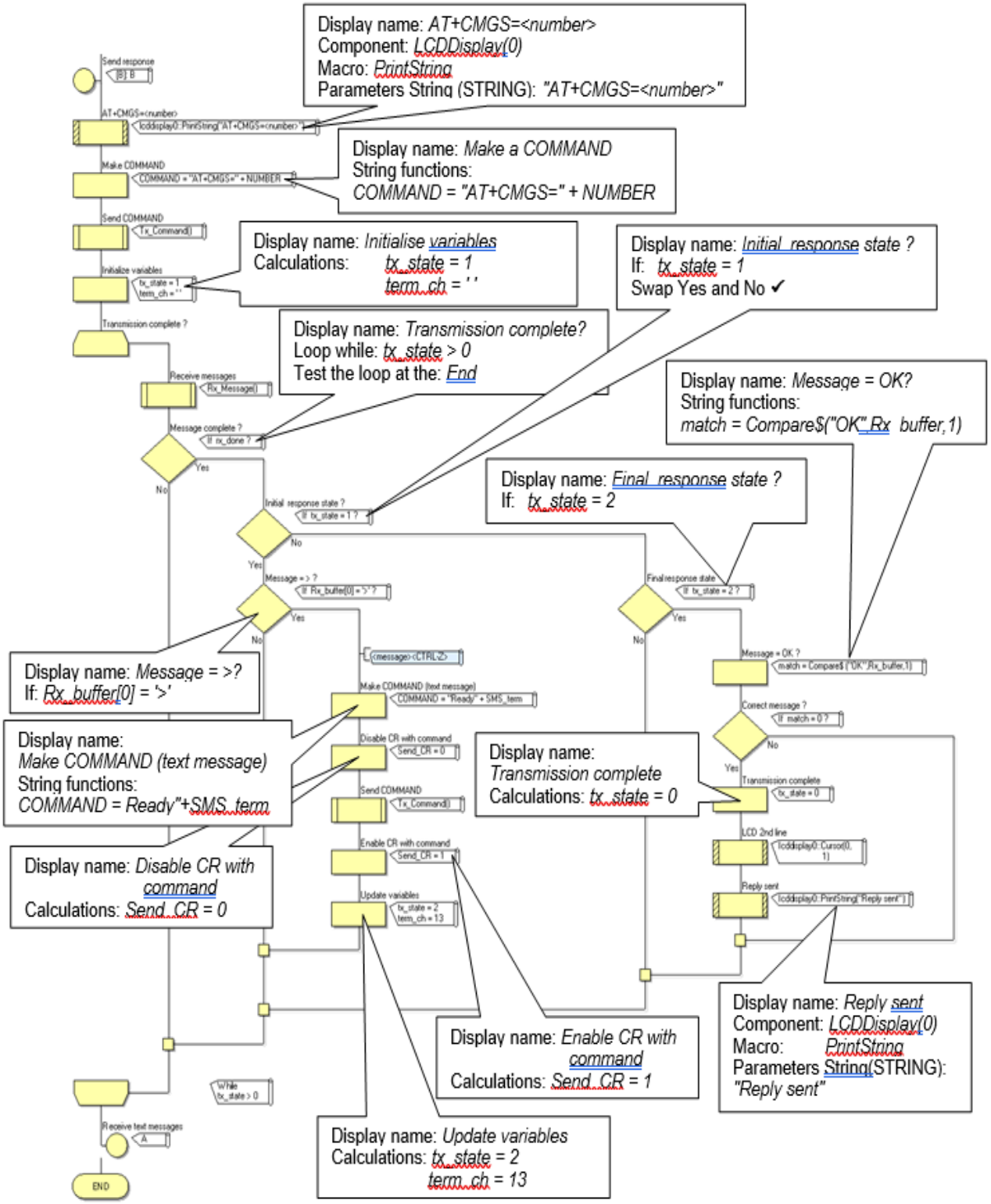

BEGIN
Termination string for text message
SMS_term[0] = 26
SMS_term[1] = 0
Display name: Termination string for text message
Calculations: SMS_term[0] = 26
SMS_term[1] = 0
Initialize LCD
lcddisplay0::Start()
Clear LCD
lcddisplay0::Clear()
Display name: Flushing buffers
Component: LCDDisplay(0)
Macro: PrintString
Parameters String (STRING):"Flushing buffers"
Flushing buffers
lcddisplay0::PrintString("Flushing Buffers")
Initialize the GSM module
InitGSM()
Display name: Characters to receive ?
Loop while: rx_done <> 255
Test the loop at the: End
Characters to receive ?
Discard characters
rx_done=rs2320::ReceiveChar(200)
Display name: Discard characters
Component: RS232(0)
Macro: ReceiveChar
Parameters Timeout(INT):200
Return Value:(INT) rx_done
While
rx_done <> 255
Initialize variables
term_ch = 13
Send_CR = 1
Rx_index = 0
msg_seg = 0
Display name: Initialise variables
Calculations: term_ch = 13
Send_CR = 1
Rx_index = 0
msg_seg = 0
Clear LCD
lcddisplay0::Clear()
AT +CMGF=1
lcddisplay0::PrintString("AT+CMGF=1")
Display name: AT+CMGF=1
Component: LCDDisplay(0)
Macro: PrintString
Parameters String (STRING): "AT+CMGF=1"
Make COMMAND
COMMAND = "AT+CMGF=1"
Display name: Make a COMMAND
String functions: COMMAND = "AT+CMGF=1"
LCD 2nd line
lcddisplay0::Cursor(0, 1)
Display name: LCD 2nd line
Component: LCDDisplay(0)
Macro: Cursor
Parameters x(Byte), y(Byte):0,1
Response?
lcddisplay0::PrintString("Response?")
Display name: Response?
Component: LCDDisplay(0)
Macro: PrintString
Parameters String(STRING):"Response?"
Not match
match = 1
Display name: Not match
Calculations: match = 1
Match not found ?
Delay
1 s
Send COMMAND
Tx_Command()
Receive message
Rx_Message()
Display name: Match not found?
Loop while: match <> 0
Test the loop at the: End
Message complete ?
If rx_done ?
Yes
No
Display name: Message complete?
If: rx_done
Call Macro
Display()
Message = "OK" ?
match = Compare$ ("OK",Rx_buffer,1)
Display name: Message = "OK" ?
String functions: match = Compare$("OK",Rx_buffer,1)
Decision
If match <> 0 ?
Yes
No
Message = "1"?
match = Compare$ ("1",Rx_buffer,1)
While
match <> 0



Display name: Make COMMAND
String functions: COMMAND = "AT+CNMI=2,2"

Display name: Termination character = ''
Calculations: term_ch = ''


Display name: Header segment?
If: msg_seg = 0
Swap Yes and No ✓
Display name: Next message segment
Calculations: msg_seg = msg_seg + 1
Display name: Test message
String functions: match = Compare$("+CMT:",Rx_buffer,1)
Display name: Caller ID?
If: msg_seg = 2
Display name: Save caller ID
String functions:
NUMBER = Rx_buffer
Display name: Message_segment?
If: msg_seg = 4
Display name: Correct header?
If: match = 0
Display name: Initiate reception
Calculations: msg_seg = 1
term_ch = ','
Display name: Termination character = <CR>
Calculations: term_ch = 13
Display name: Message = Status?
String functions:
match = Compare$("STATUS",Rx_buffer,1)
Display name: Correct message ?
If: match = 0
Display name: Call Component Macro
Component: LCDDisplay(0)
Macro:PrintAscii
Parameters Character(BYTE): "Failed text from"
Display name: Call Component Macro
Component: LCDDisplay(0)
Macro:PrintString
Parameters String(STRING): NUMBER
Display name: Reset variables
Calculations: msg_seg = 0
term_ch = ' '


Display name: AT+CMGS=<number>
Component: LCDDisplay(0)
Macro: PrintString
Parameters String (STRING): "AT+CMGS=<number>"
Display name: Make a COMMAND
String functions:
COMMAND = "AT+CMGS=" + NUMBER
Display name: Initialise variables
Calculations: tx_state = 1
term_ch = ' '
Display name: Initial_response state ?
If: tx_state = 1
Swap Yes and No ✔
Display name: Transmission complete?
Loop while: tx_state > 0
Test the loop at the: End
Display name: Message = OK?
String functions:
match = Compare$("OK",Rx_buffer,1)
Display name: Final_response state ?
If: tx_state = 2
Display name: Message = >?
If: Rx_buffer[0] = '>'
Display name:
Make COMMAND (text message)
String functions:
COMMAND = Ready"+SMS_term
Display name: Disable CR with command
Calculations: Send_CR = 0
Display name:
Transmission complete
Calculations: tx_state = 0
Display name: Enable CR with command
Calculations: Send_CR = 1
Display name: Reply sent
Component: LCDDisplay(0)
Macro: PrintString
Parameters String(STRING):
"Reply sent"
Display name: Update variables
Calculations: tx_state = 2
term_ch = 13
Send response
AT+CMGS=<number>
Make COMMAND
Send COMMAND
Initialize variables
Transmission complete ?
Receive messages
Message complete ?
Initial response state ?
Message = > ?
Make COMMAND (text message)
Disable CR with command
Send COMMAND
Enable CR with command
Update variables
While tx_state > 0
Receive text messages
END
Final response state
Message = OK ?
Correct message ?
Transmission complete
LCD 2nd line
Reply sent

The RS232 Protocol

RS-232 is a telecommunications standard dating from the 1960's, defined originally for use in teletype-writers and still in widespread use. For example, it is the basis for data transfer from a computer's 9-pin serial and 25-pin parallel ports.

It appears in a number of different forms, such as EIA/TIA232, RS-232D, V.24, V.28, X20, and X21. It is used in both asynchronous data transfer and synchronous links such as HDLC, Frame Relay and X.25.

Scope
It includes not only electrical specifications, and definitions of the signals used, but also pin outs for a range of connectors such as 9 and 25 pin D-type connectors and RJ45 connectors.

In its native form, logic voltage levels are -15 to -3V for a logic 1 (mark), and +3 to +15V for a logic 0 (space). TTL based RS232 makes use of an inverting level-converter IC to change from TTL voltage levels to those valid for RS232.

Jargon!

Devices which use serial cables for their communication are split into two categories, DCE (Data Communications Equipment) and DTE (Data Terminal Equipment.)
Data Communications Equipment includes devices such as an analogue modem, TA adapter (on an ISDN line), CSU/DSU (Channel Service Unit / Data Service Unit – a digital modem, in effect) etc., while Data Terminal Equipment is often a computer or router. Usually, the DCE device controls the flow of data between the DCE and the DTE by providing synchronisation signals or timing signals. The DTE device is also known as the data terminal, whereas the DCE device is the data set.

Confusion can arise over the pin descriptions TD (Transmit Data) and RD (Receive Data). In reality, both pins may 'transmit' data and 'receive' data at times, depending on whether they are located on the DTE or the DCE device. The solution is to look at these pins from the viewpoint of the DTE device. The DTE device transmits data on the TD line. When the DCE device receives this data, it receives it on the TD line as well! When the modem or CSU/DSU receives data from the outside world and sends it to the DTE, it sends it on the RD line because from the viewpoint of the DTE, the data is being received!

Signalling overview

Data is transmitted and received by the data terminal on pins 2 and 3, (TD and RD) respectively.

The Data Set Ready (DSR) and Data Terminal Ready (DTR) signals become active usually when the respective devices are powered up. They enable these devices to check each others status.

Data Carrier Detect (DCD) indicates that a good carrier is being received from a remote modem.

The Carrier Detect (CD) and the Ring Indicator (RI) lines are only useful in connections to a modem and telephone line.

CP2832

GSM Communications

Student Guide

Contents

1. Introduction to mobile telephony

GSM (Global System for Mobile communication) mobile phones are a means of connecting users to the telephone network.

On switch-on, the mobile phone searches for a suitable network. The network then keeps track of the mobile, so that it is able to send it incoming calls.

Base stations provide connection through a series of cells, usually pictured as interlocking hexagons. During a call, speech, in the form of digital data, is passed to and from the base station. At the same time both handset and the base station monitor the situation, judging whether or not a better cell is available. When not in a call, the mobile checks which base stations it can reach, and the network keeps track of the location of the mobile.

As the mobile moves around, it transfers from cell to cell. The network switches any calls, so that the changeover is seamless.

1.1 Features -

SIM (Subscriber Identity Module) card:

The SIM card is a small smartcard that stores the mobile phone number and the user's address book. It also contains several unique serial numbers for the phone and the user, authentication information, and details of the user's network and passwords.

TDMA;
A mobile is logged onto only one cell at a time, but that cell may simultaneously be connected to several other mobiles. These connections all use Time Division Multiple Access (TDMA). The data stream from each mobile is chopped into small segments, lasting around 20ms. These are interleaved with segments from the other mobiles. In effect, they take turns to transmit their segments, all on the same frequency channel.

Frequency hopping:
Each GSM frequency band (900MHz, 1800MHz and 1900MHz) uses a number of uplink (mobile to base) and downlink (base to mobile) frequency pairs to transmit messages. To minimise the effects of interference, the mobile and the base frequency-hop (switch between frequency pairs) during a call.

Encryption and authentication:
The GSM standard uses two levels of security:

- the data is encrypted before transmission;
- the network authenticates (checks the identity of) the mobile when setting up a call.

The approach is similar for both levels. The SIM card stores a private 'key' (binary number). The network has a copy of this key. The key itself is never transmitted.
To authenticate a mobile, the network sends a random number to the handset. This is combined with the private key using an encryption algorithm, and the result is transmitted back to the network. At the same time, the network performs the same calculation, using its copy of the private key. If the two answers match, the mobile is authenticated. In the same way, the mobile and the network generate a cipher key using a different algorithm. The result is used to encrypt each packet of data.

Roaming:
Roaming allows a GSM phone user to make and receive calls using any GSM network, when abroad for instance. The handset always tries to find its home network first, but if that fails, it will then scan for other networks.

SMS:
Short Message Service (SMS) messages can be slow because of the limited bandwidth available to carry them. SMS delivery is a store-and-forward system, where the message is stored on the network, which then forwards it to the destination mobile, when it is accessible.

2. Overview of the Exercises

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 CD-ROM, and are available from your instructor. In each case there is scope for improvement and for further development. These, or the student's own solution, could be used as the starting point for discussion 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, more advanced student should be capable of incorporating these features into further developments.

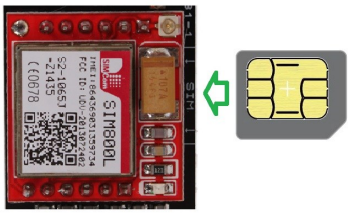
Note:
Some modem commands are part of the GSM specification and must be implemented fully, or partially, by all GSM devices. Others may be manufacturer or model specific and should be used with care if compatibility with other devices is required. The documentation supplied provides details of each supported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

3. Hardware configuration

3.1 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.

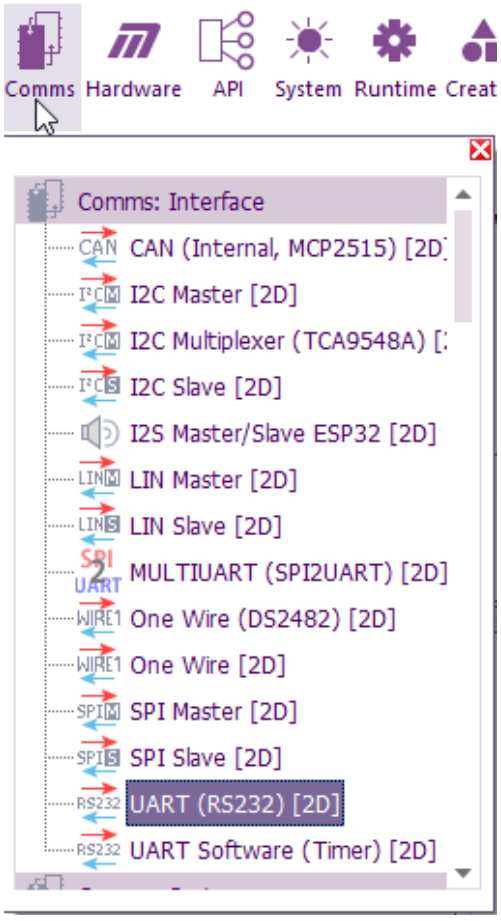
IMPORTANT!
You need an active SIM card. The GSM modem requires a mobile phone SIM card with credit on it. Just like any other mobile phone the GSM modem needs a SIM card in order to work. You will need to purchase a SIM card from the same place that you would purchase one for a normal mobile phone.



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.

To use Flowcode, version 8 or later, it must be installed onto your local computer.

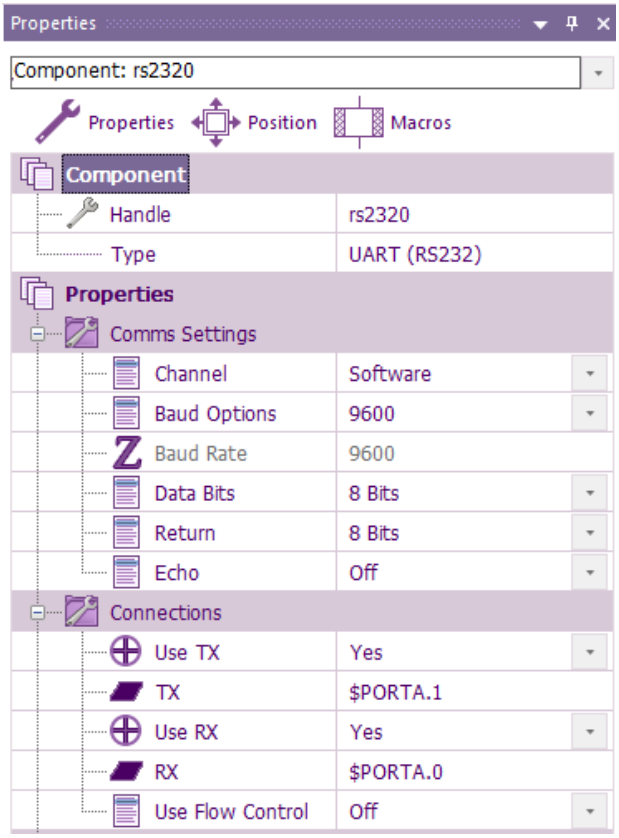
Sample Flowcode exercises in this tutorial are available on the CD-ROM (CD2130) supplied with the GSM Communications kit.



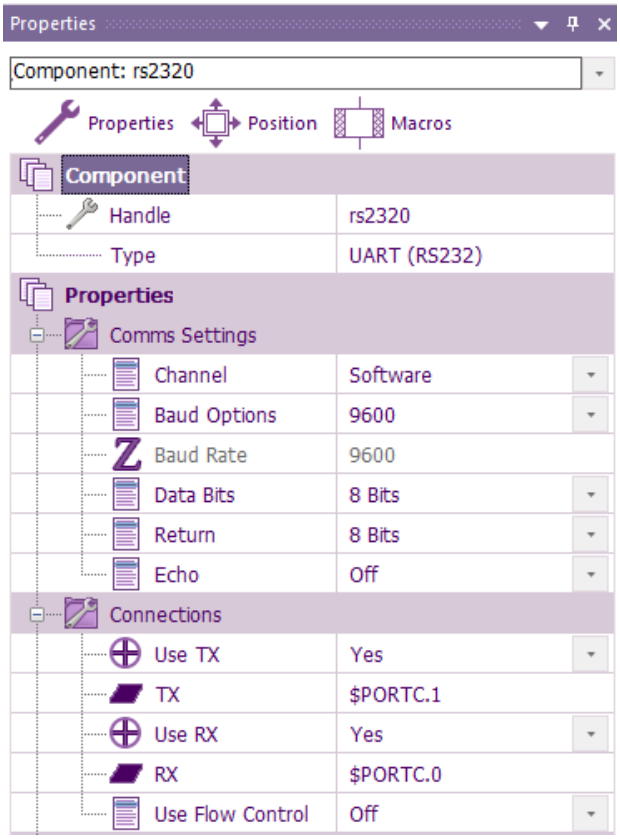
3.2 Configuring the RS232 Component

The Flowcode RS232 component can be found in the ‘Comms’ section of the Components Toolbar, as shown here.

The Flowcode RS232 component includes a Component Properties section that allows the communication baud rate to be configured. The settings illustrated here should be used for all the exercises.



BL0011



BL0055

3.2.1 BAUD rate

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

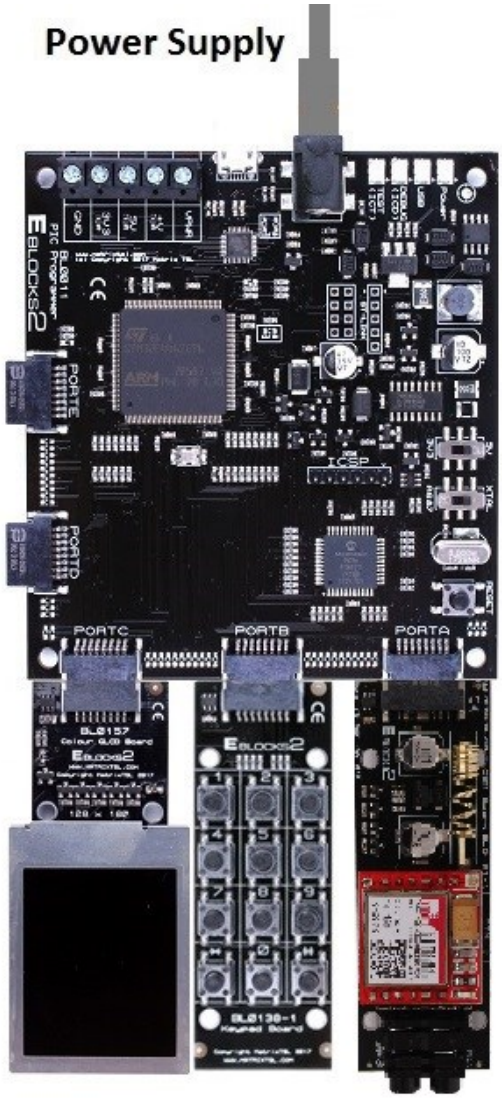
3.2 Configuring the microcontroller system

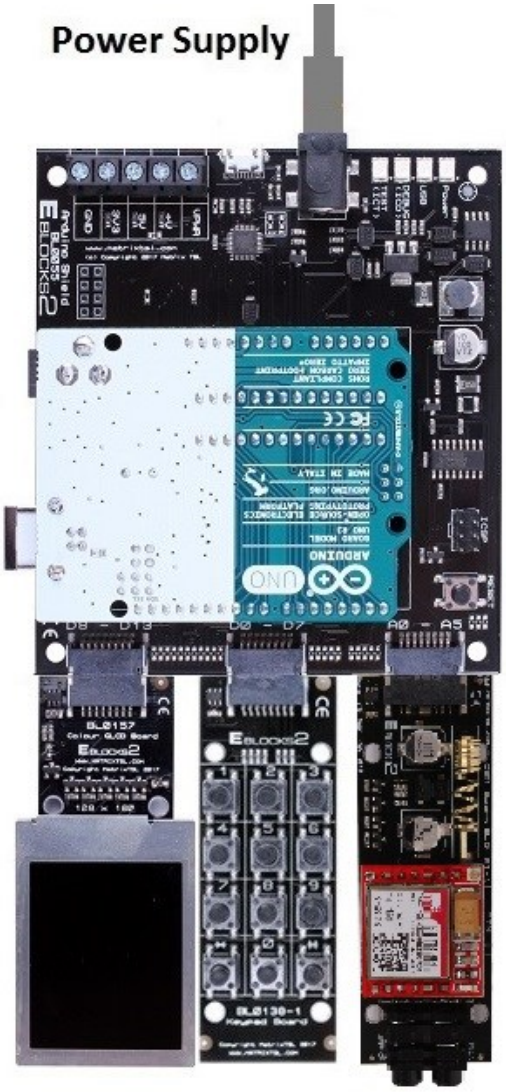
The Flowcode exercise examples for use with this curriculum target specific E-blocks2 processor and peripheral boards.

The processor boards are part number BL0011 for the PIC processor board and BL0055 for the Arduino Uno board. Using these as target devices for Flowcode will pre-configure all processor settings.

The E-blocks2 boards are to be attached to the processor board as shown in the table and images that follow.

Hardware configuration





PIC BL0011			Arduino BL0055		
Port A	Port B	Port C	A0-5 (C)	D0-7 (D)	D8-13 (B)
BL0131	BL0138	BL0157	BL0131	BL0138	BL0157

BL0131	GSM E-blocks2 board
BL0138	Keypad E-blocks2 board
BL0157	gLCD E-blocks2 board

4. Exercise 1: A basic telephone

4.1 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.

A telephone must be able to perform three basic control functions:
- Make a connection to another selected telephone.
- Accept a connection from another telephone.
- Disconnect from a remote telephone.

4.1.1 AT 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.

4.1.2 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	

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.

4.1.3 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
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 + “ Ltd”

(note the single space character before the “L”)

Name	M	a	t	r	i	x		L	t	d	
index	0	1	2	3	4	5	6	7	8	9	10
Name[ ]	77	97	116	114	105	120	32	76	116	100	0

Name contains “Matrix Ltd”

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 “ Ltd” string (index 10) becomes the terminator for the new compound string when the two are added.

4.2 Objectives

The objectives are to:

- establish a RS232 communication link between the microcontroller and the modem;
- develop a macro to allow command strings to be sent to the modem;
- use microcontroller inputs to control the transmission of AT commands to the modem;
- develop a Flowcode program that causes the modem to behave as a telephone.

In doing so, the learning outcomes are 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.

4.3 Requirements

A multi-programmer board attached to a PC running Flowcode
A keypad E-blocks2 board
A GSM modem E-blocks2 board with an active SIM card and an audio headset attached
A working telephone

4.4 The Flowcode program in detail

The main program contains a string variable called NUMBER. A logic block is placed at the top of the main program, making it easy to locate and edit. This initialises NUMBER, so that it contains the number of mobile phone used in conjunction with this exercise,

4.4.1 Flowcode RS232 component – SendChar

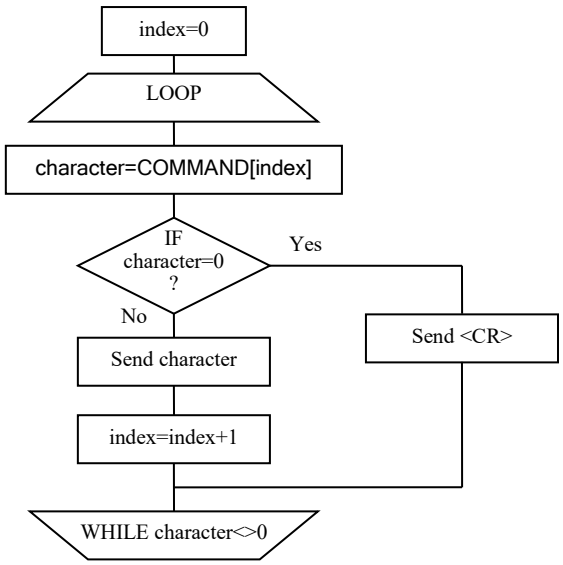
The Flowcode RS232 component must be loaded into the program in order to gain access to the RS232 functions. One of these functions is SendChar**,** which allows the transmission of a single character from the RS232 port.

All the modem commands consist of multiple characters. Each character in a command can be transmitted with an individual use of the SendChar 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.

4.4.2 Macro – Tx_Command

This macro is called Tx_Command. The main program contains a string variable called COMMAND. When the macro is executed, it transfers each character in COMMAND to the SendChar 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 SendChar function and return to the main program.

The following flowchart summarises this macro:



4.4.3 Using AT commands

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

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				

4.5 What to do

This exercise requires the transmission of three message strings, controlled by three buttons on the key-pad.

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 consists 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.

Using the flowchart as a guide, construct the Flowcode program by dragging appropriate icons onto the workspace from the icon toolbar. Then double-click on the icon to open the Properties dialogue box which allows the function to be configured.

In particular, double-click on the first String manipulation icon and enter the phone number of the phone you will call to test the program. This is done in the form:

NUMBER = "XXXXXXXXXXX"

where XXXXXXXXXXX represents the telephone number of the phone you will call.

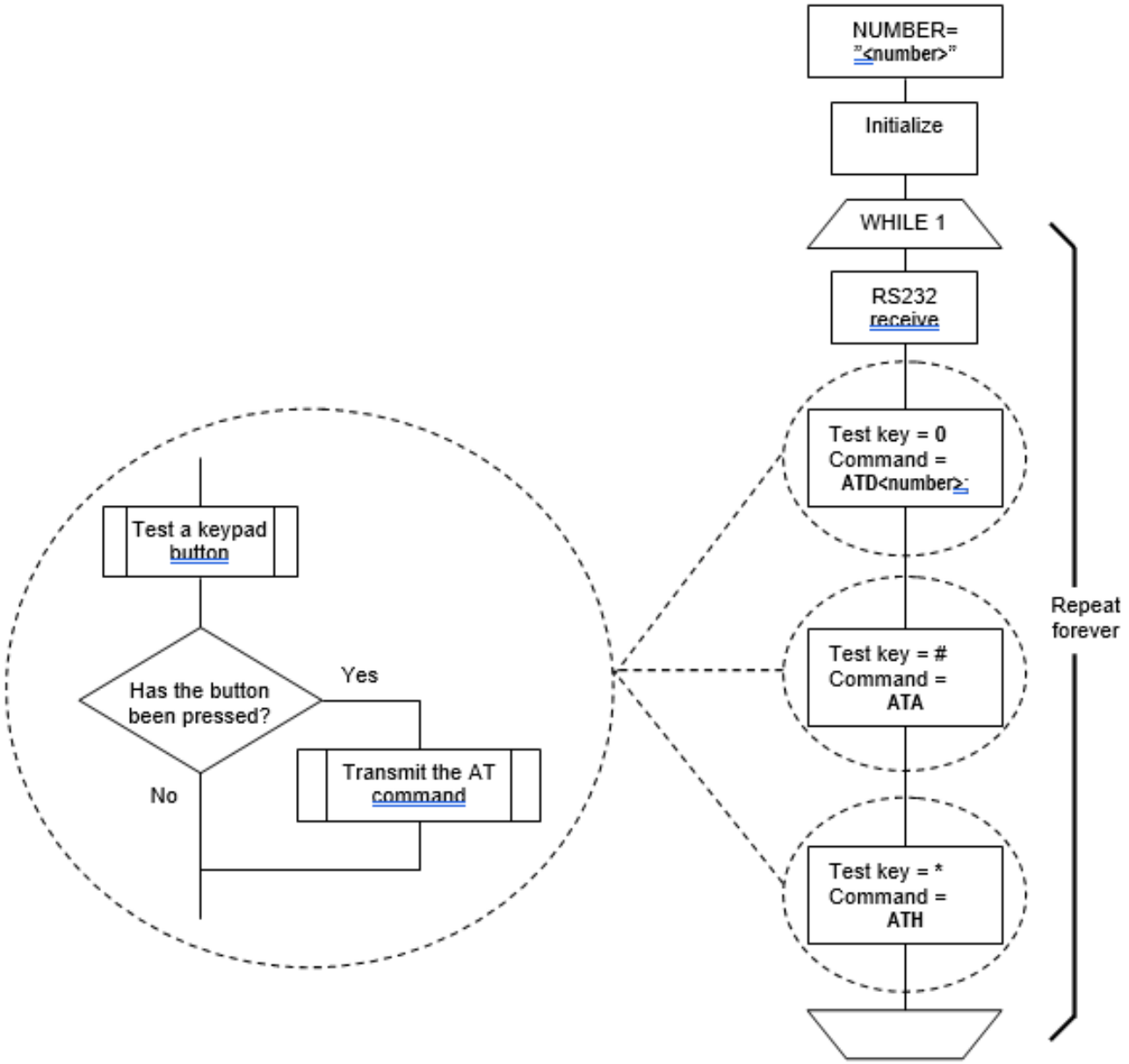
Note: The ‘RS232 receive’ function is required because the modem transmits response messages during operation, and if these are not accepted, the modem transmit buffer will become full, preventing further communication.

Characters being transmitted by the modem are discarded in this exercise.

Download the program to the microcontroller.

Test it by running the program and checking that each of the assigned keys works as expected.

Exercise 1: A basic telephone



Telephone control structure

4.6 Further work

The program can currently dial only 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.

5. Exercise 2: A simple 'State Machine'

5.1 Introduction

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

For example, pressing the dialling key when a call is already in progress causes the modem to lose the existing connection, and it fails 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.

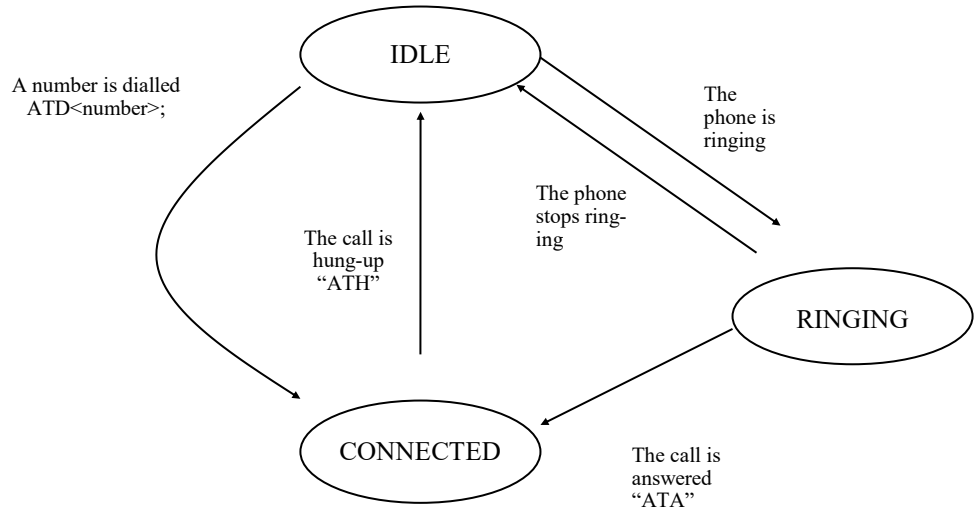
The operation of a telephone, as with most other devices, can be broken down into a set of states. Each state describes one of the conditions in which a device can exist. The current state can be changed, in a pre-defined way, by the effect of external or internal influences.

5.1.1 State machine

Telephone operation can be 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



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

5.2 Objectives

The objective is to develop a state machine that improves the functionality of the phone application developed in exercise 1.

In doing so, the learning outcomes are to:

- understand the advantages of structured code;
- define and implement simple state machines.

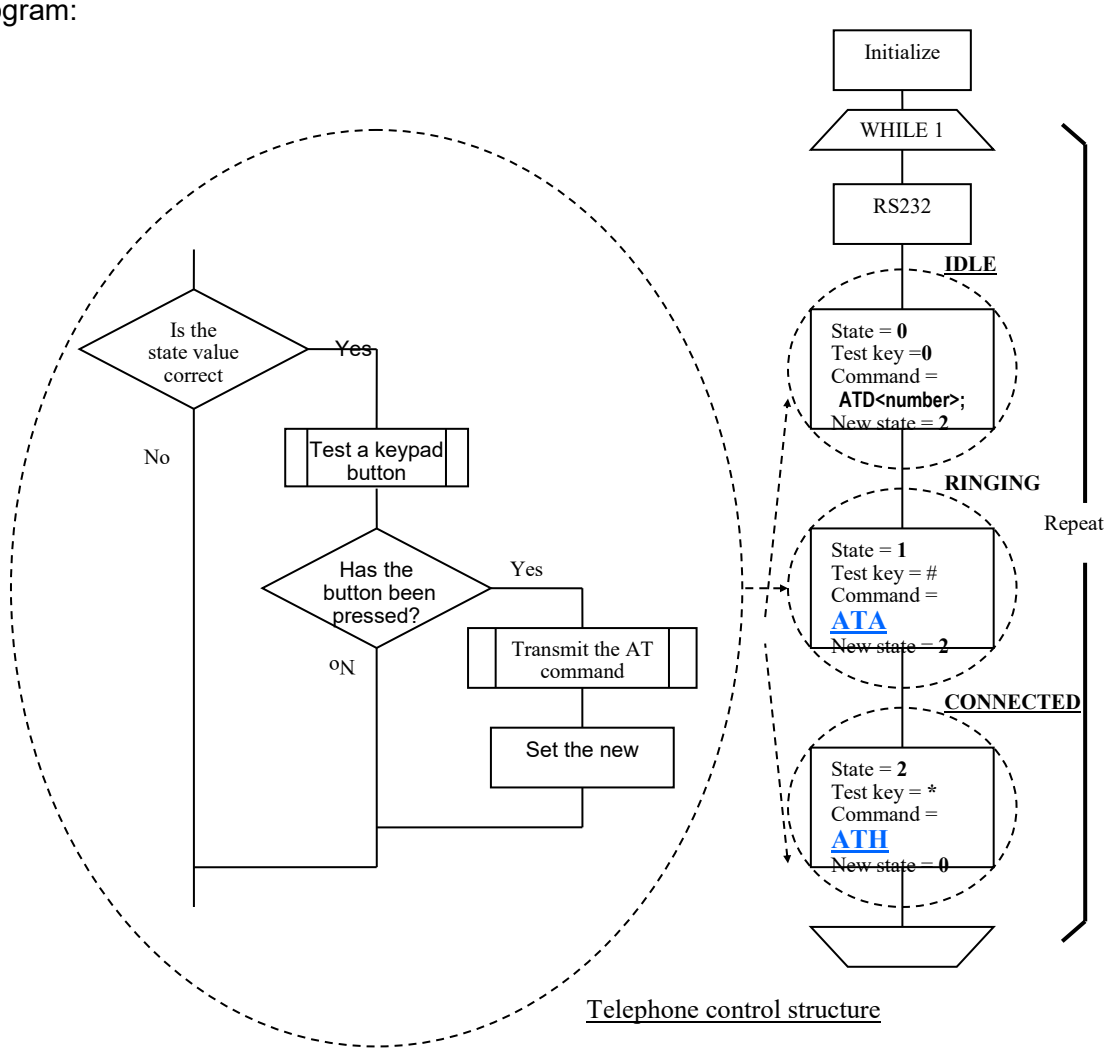
Note:
In this exercise, the ability to answer a call is temporarily lost. An improved version is developed in the next exercise.
This exercise concentrates on the structure of the software and the solution to some practical problems. The loss of the call answering function may appear to be a backward step, but the work carried out forms the basis of the next exercise, and demonstrates the advantages of structured code.

5.3 Requirements

- A multi-programmer board attached to a PC running Flowcode
- A keypad E-blocks2 board
- A GSM modem E-blocks2 board 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.fcfx.**

5.4 The Flowcode program in detail

The following diagram sets out the proposed structure of the program:



Telephone control structure

5.5 What to do

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

Download it and test it in the same way as for exercise 1.

5.6 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.

6. Exercise 3: Modem responses

6.1 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 developed in this exercise provides a display of the messages transmitted by the modem, and generates software that can make use of them subsequently.

6.1.1 Displaying messages

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 improves the functionality of any device developed to use them, and allows full automation of call connection and monitoring applications.

Messages are transmitted to the controller as strings of ASCII characters; similar to the commands being sent to it. The program from the previous 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.

6.1.2 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 any other LCD commands.

The LCD Clear function ensures that the display area is blank and that the cursor is at the start of the top line.

Each character sent to the LCD is displayed at the current cursor 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.

6.2 Objectives

The objective is to develop a Flowcode program that will receive and display the characters transmitted by the modem, using the LCD component.

In doing so, the learning outcomes are to:
- recognise the content and format of modem messages;
- identify the meaning and purpose of messages transmitted by the modem;
- develop a strategy to extract messages from the stream of characters and use them for their intended purpose.

6.3 Requirements

A multi-programmer board attached to a PC running Flowcode
A keypad E-blocks2 board
A LCD E-blocks2 board
A GSM modem E-blocks2 board with an active SIM card and an audio headset attached
A working telephone
A working solution to exercise 2, or a copy of Phone_02.fcfx.

6.4 The Flowcode program in detail

6.4.1 Message characters

The ReceiveChar 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.

To do this, add a variable called ‘rx_char’ to the program and use it to receive the value returned by ReceiveChar. 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.

6.4.2 Echoed characters – modified Tx_Command macro

The Tx_Command macro is a useful place to detect echoed characters. The LCD Clear function clears the display and places the cursor at the start of the top line each time a command is sent to the modem.

The ReceiveChar function is used within the macro loop to receive each character, in turn, from the RS232 port, and the LCD PrintAscii function then displays it.

The LCD Cursor function places 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.

6.5 What to do

Modify the program for exercise 2 by following the steps outlined above.

The Tx_Command macro needs to be modified, again as outlined above, by adding icons to clear the LCD module, near the beginning of the macro, adding a ReceiveChar function and a LCD PrintAscii function at the end of the loop, and adding a LCD Cursor function just after the loop.

Then test the program as follows:

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.

Reset the controller board

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

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.

6.6 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 requires 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.

7. Exercise 4: Listening to messages

7.1 Introduction

The previous exercises left a significant gap in the functionality of the phone - the inability to answer an incoming call. This exercise addresses 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.

7.1.1 Recognising an incoming call

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, 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, to prevent a time-out while the messages are being received.

Timer timed out:
- move the system back to the IDLE state.

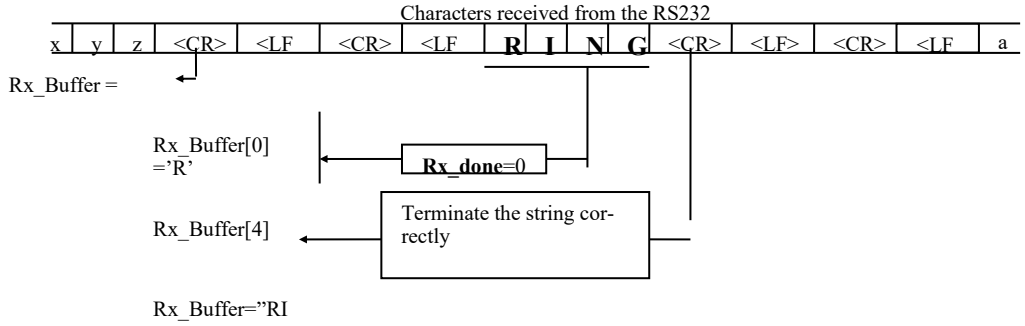
Call answered (* key pressed):
- send the ATA message to the modem;
- move the system to the CONNECTED state.

7.1.2 Message reception

The RING message, and any other message, can be detected by developing a macro that saves incoming characters as 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.

As pointed out earlier, all messages and responses start and finish with the <CR><LF> characters. One way to isolate the message would be to save all received characters until a <CR> is detected and then make the saved string available to the main program.

This process is illustrated in the following diagram:



7.2 Objectives

This exercise results in the development of a functional telephone that is structured to operate correctly in all circumstances. This involves:

- developing a macro to receive individual modem messages and responses (but not echoed characters).
- recognising the 'RING' message as an indication of an incoming call.
- automatically adjusting the program state on detection/loss of the 'RING' message
- creating 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.

7.3 Requirements

- A multi-programmer board attached to a PC running Flowcode
- A keypad E-blocks2 board
- A LCD E-blocks2 board
- A GSM modem E-blocks2 board with an active SIM card and an audio headset attached.
- A working solution to exercise 2, or a copy of **Phone_03.fcfx**

7.4 The Flowcode program in detail

The program will:

- 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, (and set it to zero in all other cases;)
  discard the <LF> character;
  discard 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.

7.5 What to do

7.5.1 Message detection

Create a macro, Rx_Message, to perform the message detection function.

Use a non-zero value of the Rx_done variable to force the main program to test the message in Rx_Buffer.

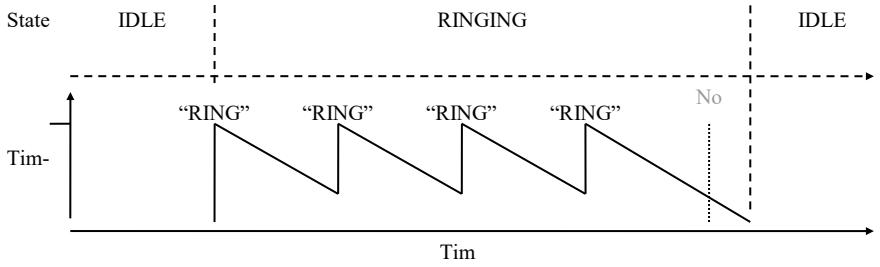
Use the string manipulation function, Compare$, to detect the presence of ‘RING’ in Rx_Buffer, and use a byte variable, match, to store the result of the comparison, i.e. match = Compare$("RING", Rx_Buffer, 1)
Note: The function Compare$ returns a zero if the two strings match

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

7.5.2 Message interval timer

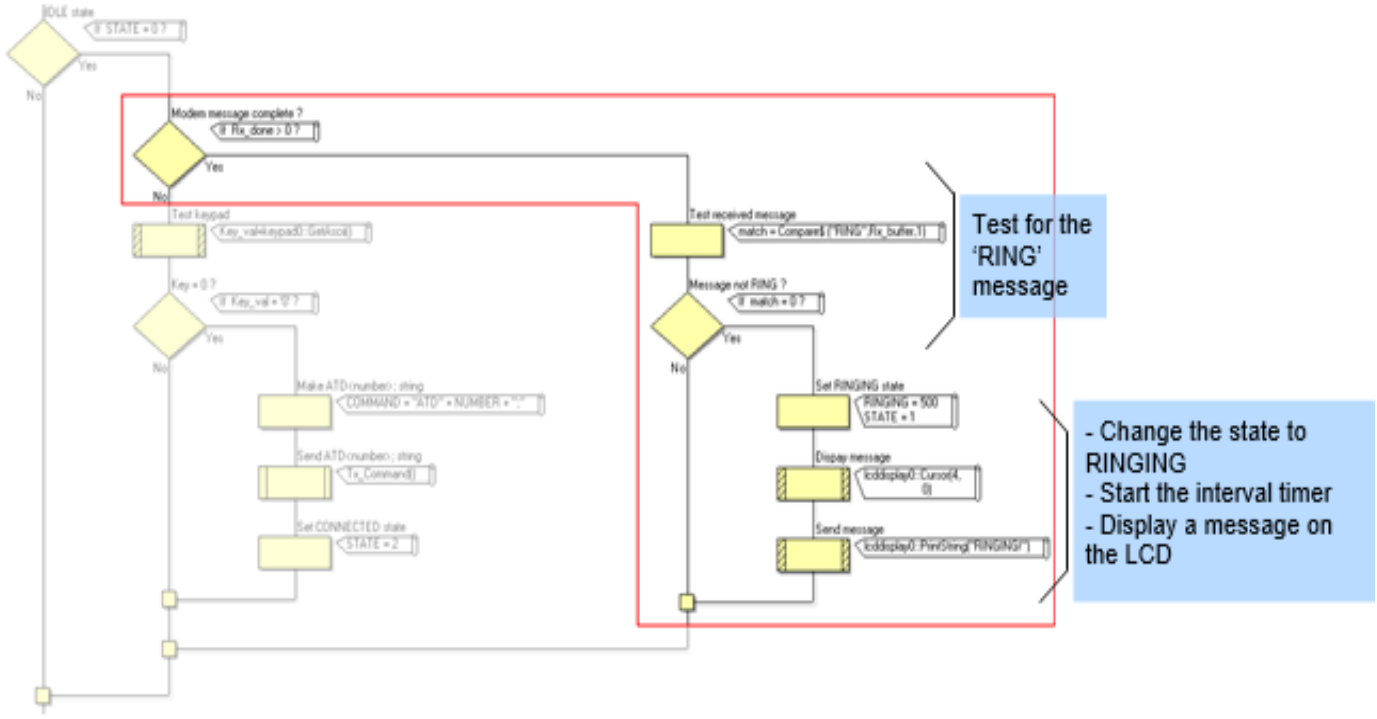
The Flowcode ‘Delay’ function allows accurate time delays to be introduced into programs, but suspends program operation during these periods. In this exercise it is necessary to maintain program operation in the period between consecutive ‘RING’ messages, allowing the RS232 port to be read and the *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



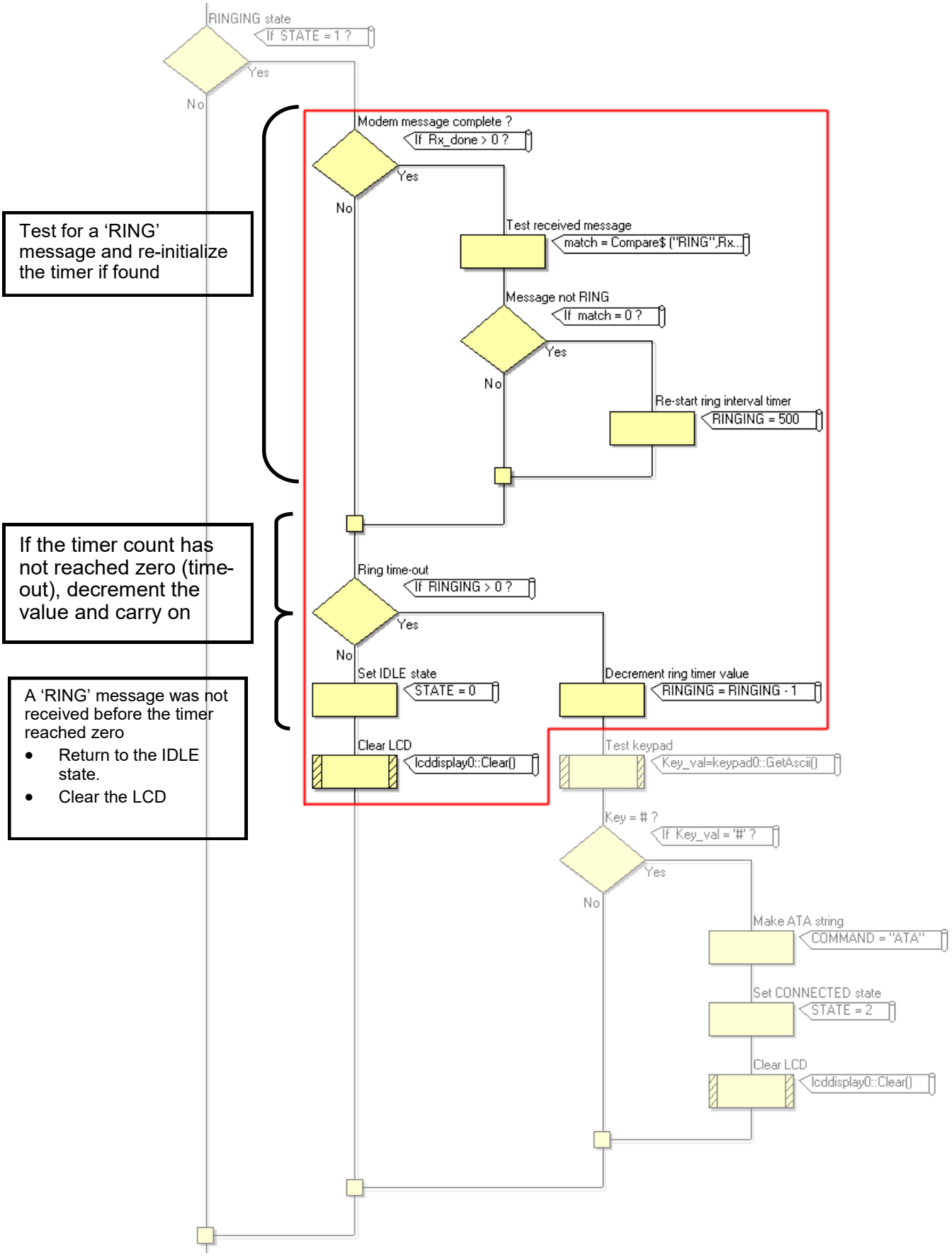
7.5.3 IDLE state modifications

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:



7.5.4 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. The changes needed are shown below:



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

8. Exercise 5: Automatic call handling

8.1 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.

8.1.1 Automating the process

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

The controller is already programmed to detect the 'RING' message, but responds only 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 is possible to modify the code in the following way:

- The IDLE state code responds only to detection of the 'RING' message, not the dial key.
- The RINGING state code answers the call immediately, without waiting for the answer key, or any subsequent 'RING' messages.
- The CONNECTED state code hangs-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.

8.2 Objectives

This exercise results in changing the device function from a normal telephone to an auto-answering audio link.

This is achieved by:

- recognising the function of individual blocks of code;
- editing and re-arranging the sequences of code blocks to modify the device operation.

8.3 Requirements

- A multi-programmer board attached to a PC running Flowcode
- A GSM modem E-blocks2 board with an active SIM card and an audio headset attached.
- A working solution to exercise 4, or a copy of **Phone_04.fcfx**

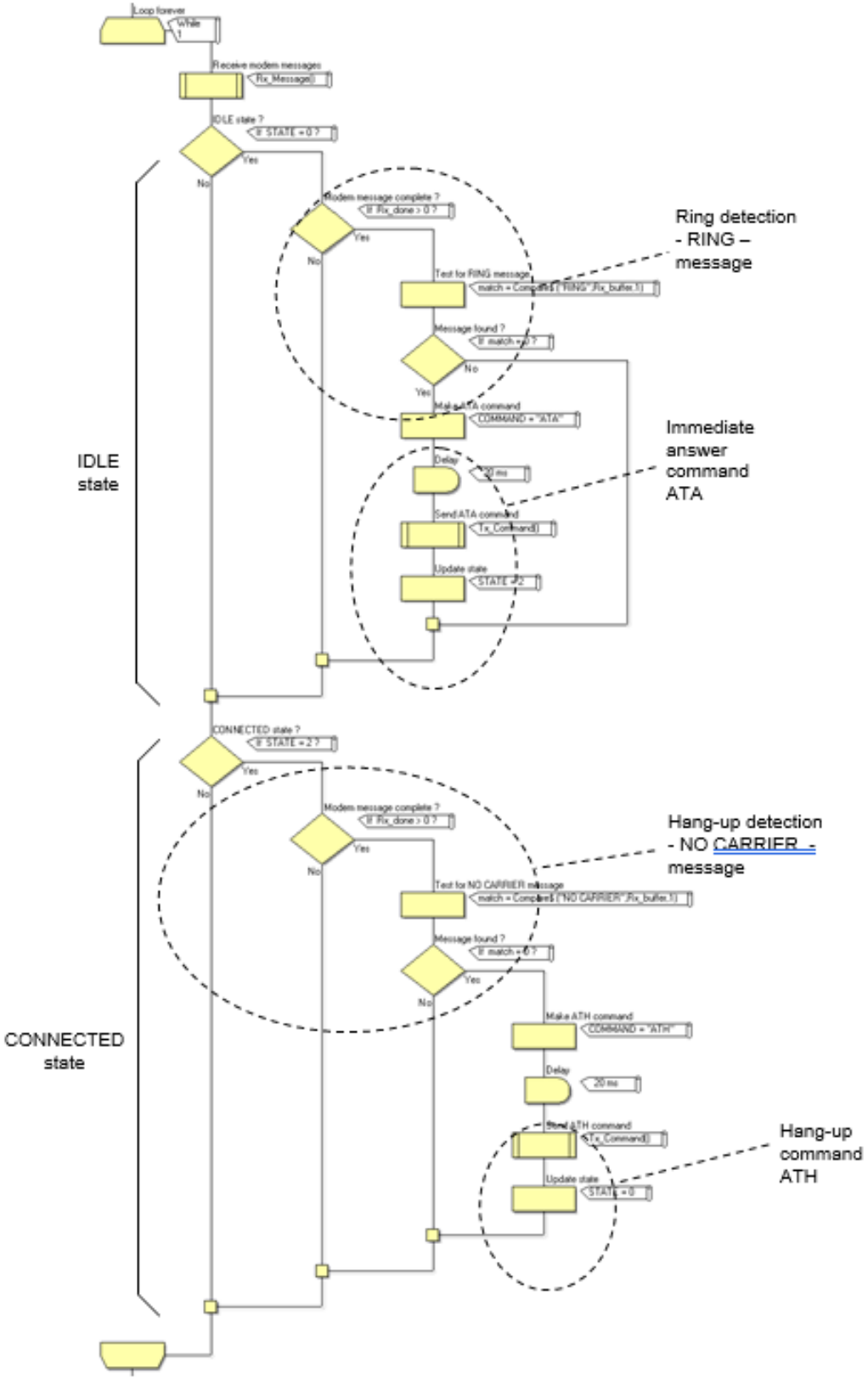
8.4 The Flowcode program in detail

The program from exercise 4 is modified as follows:

- the dialling key detection is removed from the IDLE state as this phone answers only incoming calls;
- the whole of the RINGING state is removed as the phone is answered immediately from the idle state;
- the ATA command is 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.

8.5 What to do

Modify the program from exercise 4 as described above, and as shown in the following diagram:



8.6 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).

9. Exercise 6: Send a text message

9.1 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 has been included in the GSM standards to allow messages to be sent, received, stored, recalled and formatted. The remaining 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 anyone who successfully completed the audio exercises.

9.1.1. SMS 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

9.1.2 Message format

It is best to configure the modem before attempting to transmit or receive any messages. The AT+CMGF command sets the format of the text messages and most of the modem responses.
Text messages can be delivered in compressed PDU (Protocol Data Unit) format, or Text format.

- PDU 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, and these can be transferred directly into string variables.
  The AT command required to configure this option is Message Format command: AT+CMGF=1

9.1.3 Send a message

In voice mode, 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.

Here is the full sequence:

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

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

9.1.4 Rx_Message macro modifications

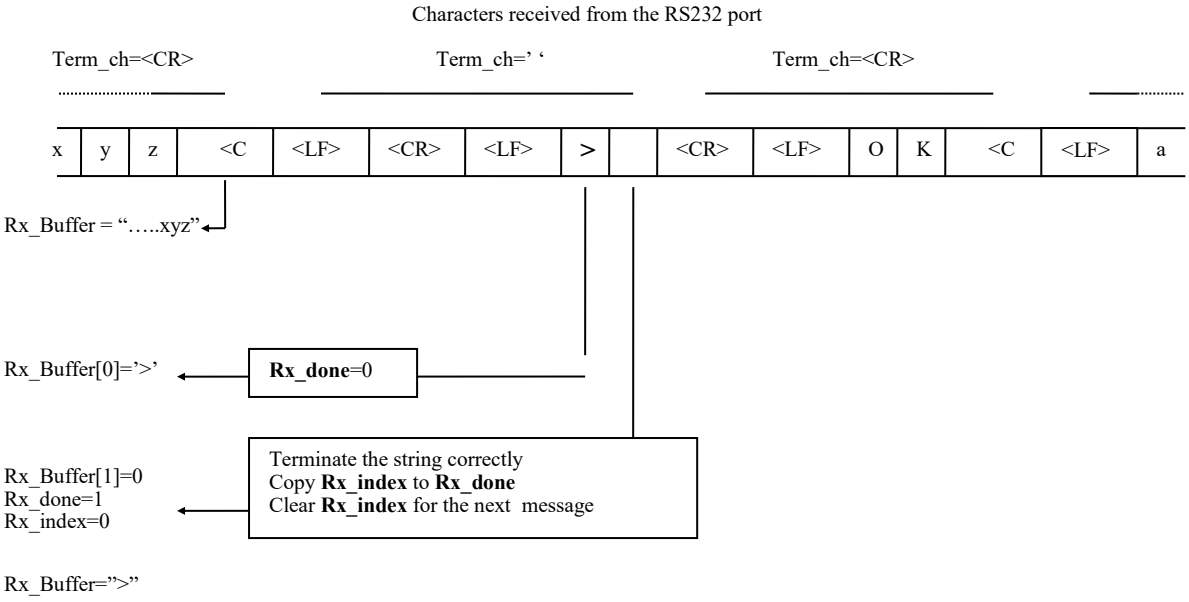
The range of message formats to be detected in SMS mode is greater than in audio mode. Detection of the "> " characters cannot be achieved with the existing Rx_Message macro due to the absence of a <CR> character. The Rx_Message macro must now be modified to use a character supplied by the main program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



9.1.5 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.

9.2 Objectives

The aim is to develop a Flowcode program that will transmit a pre-loaded text message to a pre-loaded number, when an allocated keypad key is pressed.

9.3 Requirements

- A multi-programmer board attached to a PC running Flowcode
- A keypad E-blocks2 board
- A GSM modem E-blocks2 board 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)

9.4 The Flowcode program in detail

The steps involved in this program are:

- 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.

The Flowcode program transmits the AT+CMGS="**<number>**" command when the '#' key on the keypad is pressed.

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

It is not possible to include the **"** character in a string so this character is transmitted independently, before and after transmission of the **<number>** string.

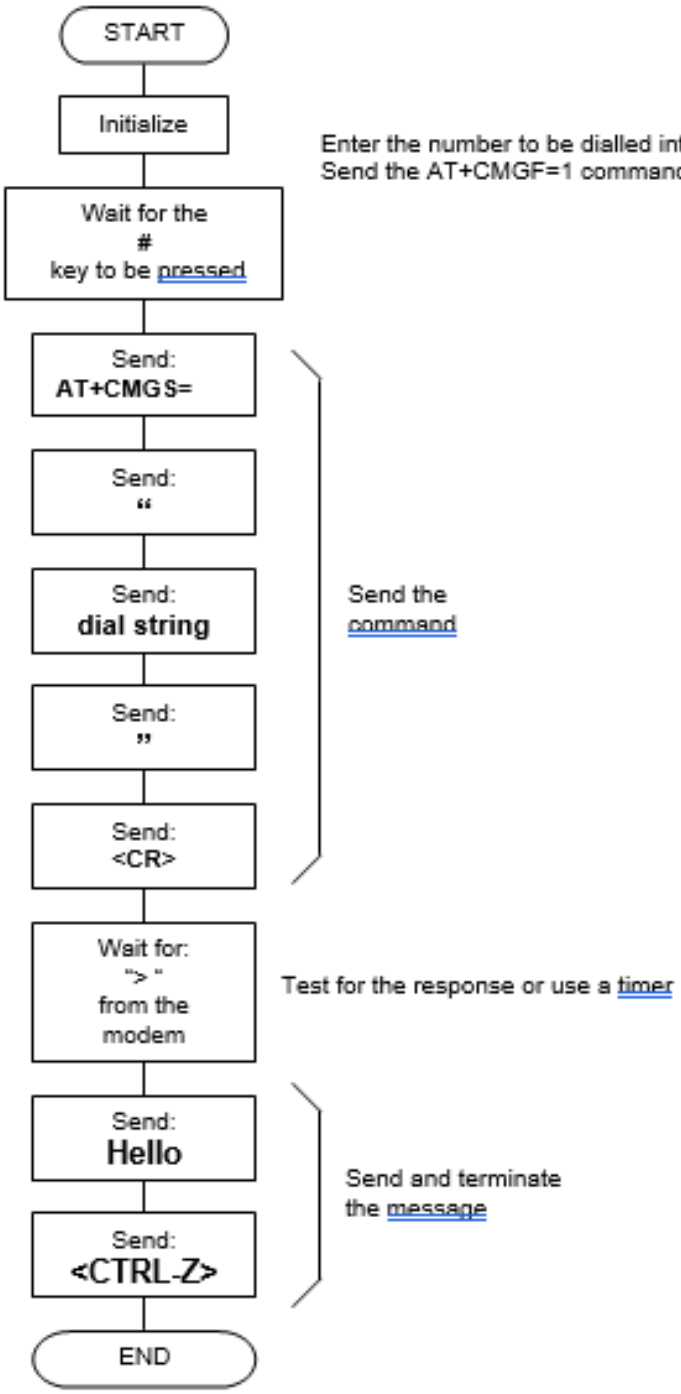
It is important to wait for the modem to generate the "> " response before sending the text message and so the code allows for this. The new Rx_Message macro detects the space after the > character and tests the Rx_Buffer for the > character.

Note:
The Compare$ function does not work reliably with single character strings and so a Calculation function is used to compare Rx_Buffer[0] with '>' directly.

When the modem is ready to receive the text message, the message should be sent and terminated with a <CTRL-Z> character (26). You need to set the Send_CR variable to 0 before transmission to prevent the <CR> character from being added after <CTRL-Z>.

On completion, the program terminates to prevent accidental transmission of further messages.

The general program structure is in the next diagram.



9.5 What to do

- As outlined earlier:
  - modify the Tx_Command macro;
  - modify the Rx_Message macro.

- Insert a String function icon, and configure the contents with the number of the mobile phone to be called, and the text message to be sent.
- Set Text mode, using the AT+CMGF=1 command.
- Create a loop to wait until the ‘#’ key is pressed.
- Transmit the mobile phone number to be called.
- Check to see if that mobile phone answers, by waiting for the ‘>’ message.
- Then transmit the text message, and then terminate the call.

9.6 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.

10. Exercise 7: Receive a text message

10.1 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, to signify detection, and text message itself will be used.

10.1.1 Filtering the incoming message

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.

A byte variable (SEGMENT) controls 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 increments the SEGMENT value and proceeds to the detection of the caller ID, date, and time. This information is not required for this exercise, and so setting the delimiter character to <CR> will retrieve all the characters in a single operation.

Detection of the first <CR> character allows 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 is stored as a string in ‘Rx_Buffer’, allowing it to be displayed, tested, or manipulated, as the application requires. In this case it is displayed.

Example:

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

<CR><LF>+CMT:

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

(1) (2) (3)

Key:

1. Calling **Rx_Message** with the delimiter set to ‘ ‘ (space) causes the function to set a value in **Rx_done** when the +CMT: segment of the message is received.

   +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>** then sets a value in **Rx_done** when the caller ID, date, and time characters are received. The information is not required in this example and can be ignored

3. Calling **Rx_Message** again with the delimiter set to **<CR>** sets a value in **Rx_done** when the text section of the message is received.

   This is 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.

10.2 Objectives

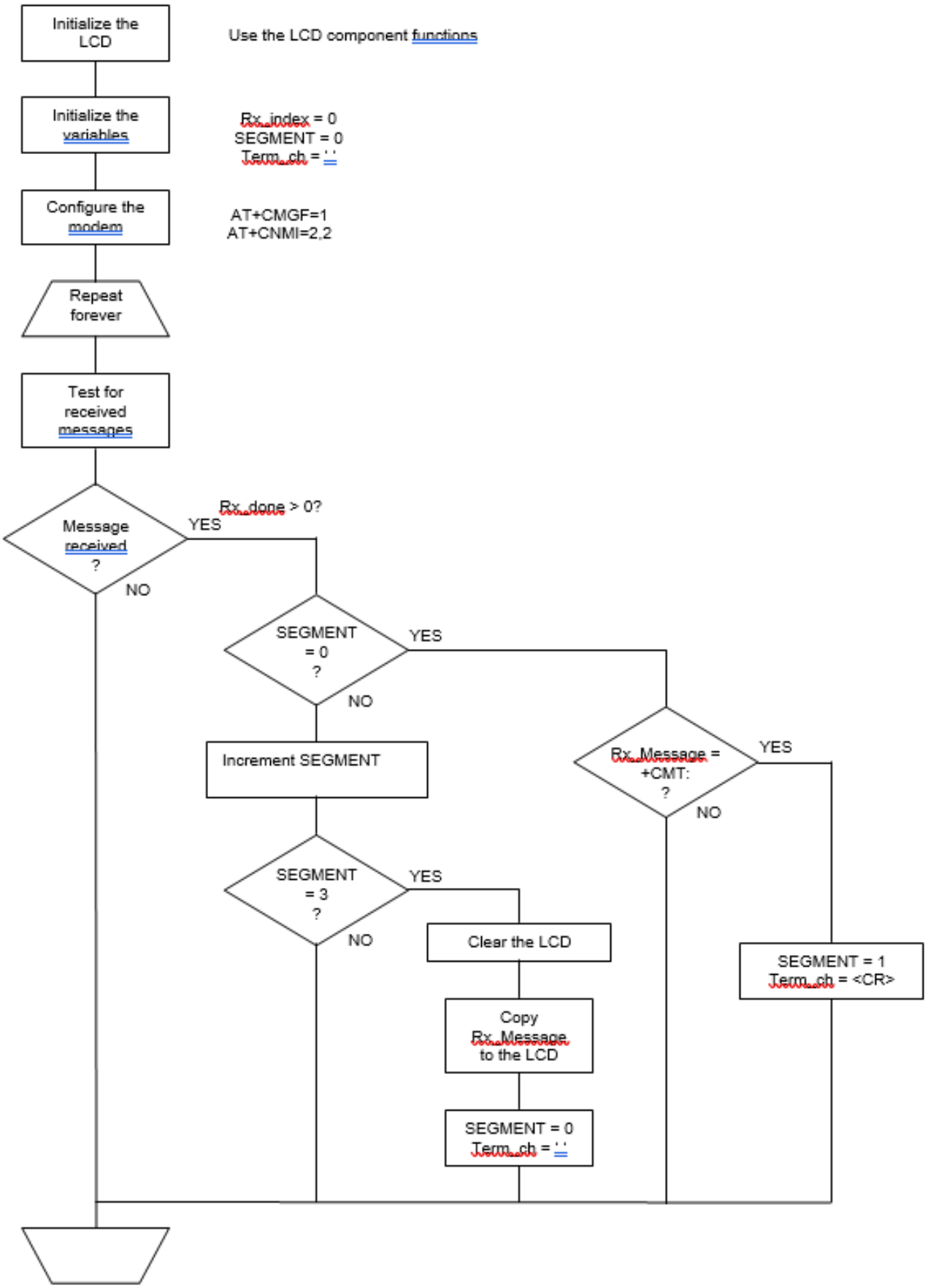
This exercise creates a more complex program to receive a simple text message, extract it from the accompanying information, and display it on a screen.

10.3 Requirements

- A multi-programmer board attached to a PC running Flowcode.
- A keypad E-blocks2 board
- A LCD E-blocks2 board
- A GSM modem E-blocks2 board 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.

10.4 The Flowcode program in detail

The following diagram shows the stages of the program developed in this exercise:



10.5 What to do

Build the program for this exercise, using the flowchart given above as a guide.

The steps involved are:

- Initialise and clear the LCD unit.
- Initialise the modem for the correct text mode operation.
- Check whether a text message has been received.
- If so, check which segment of the header has been isolated.
- When the message itself is received, display it on the LCD module.

Download it to the microcontroller and test it by sending a text message.

10.6 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 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 these extensions 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.

11. Exercise 8: Automatically respond to a text message

This is the most complex of all the exercises and is a test of both the text message reception and transmission requirements, and illustrates their potential 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

11.1 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 are combined to create a useful application, capable of reading an incoming text message, and creating and sending a reply message.

11.1.1 Message handling

A text message reception program can be developed to detect the message header (+CMT:) 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 formatting a reply message.

11.1.2 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.
A typical example is:

+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 completed, 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.

11.1.3 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.

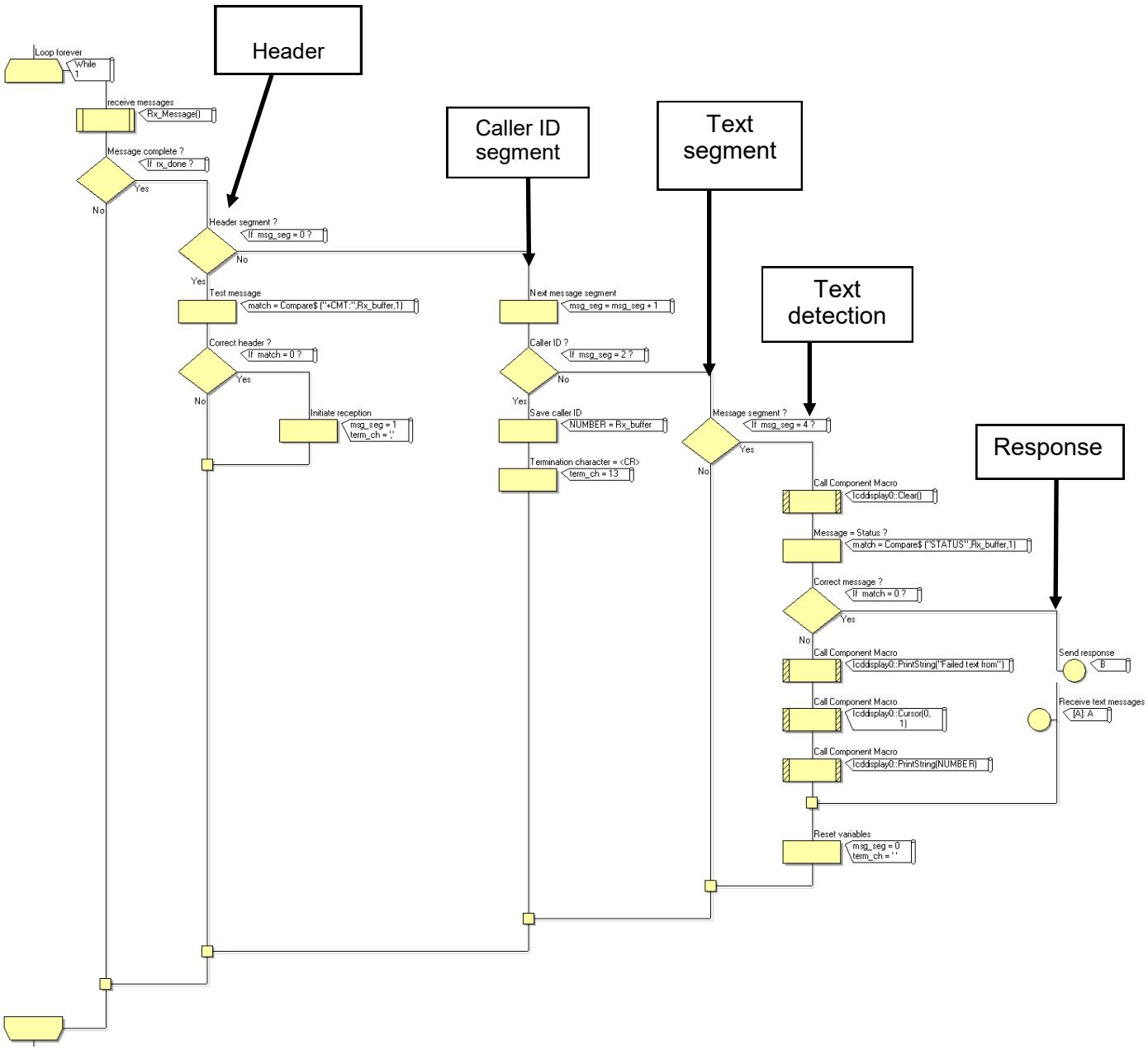
11.2 Objectives

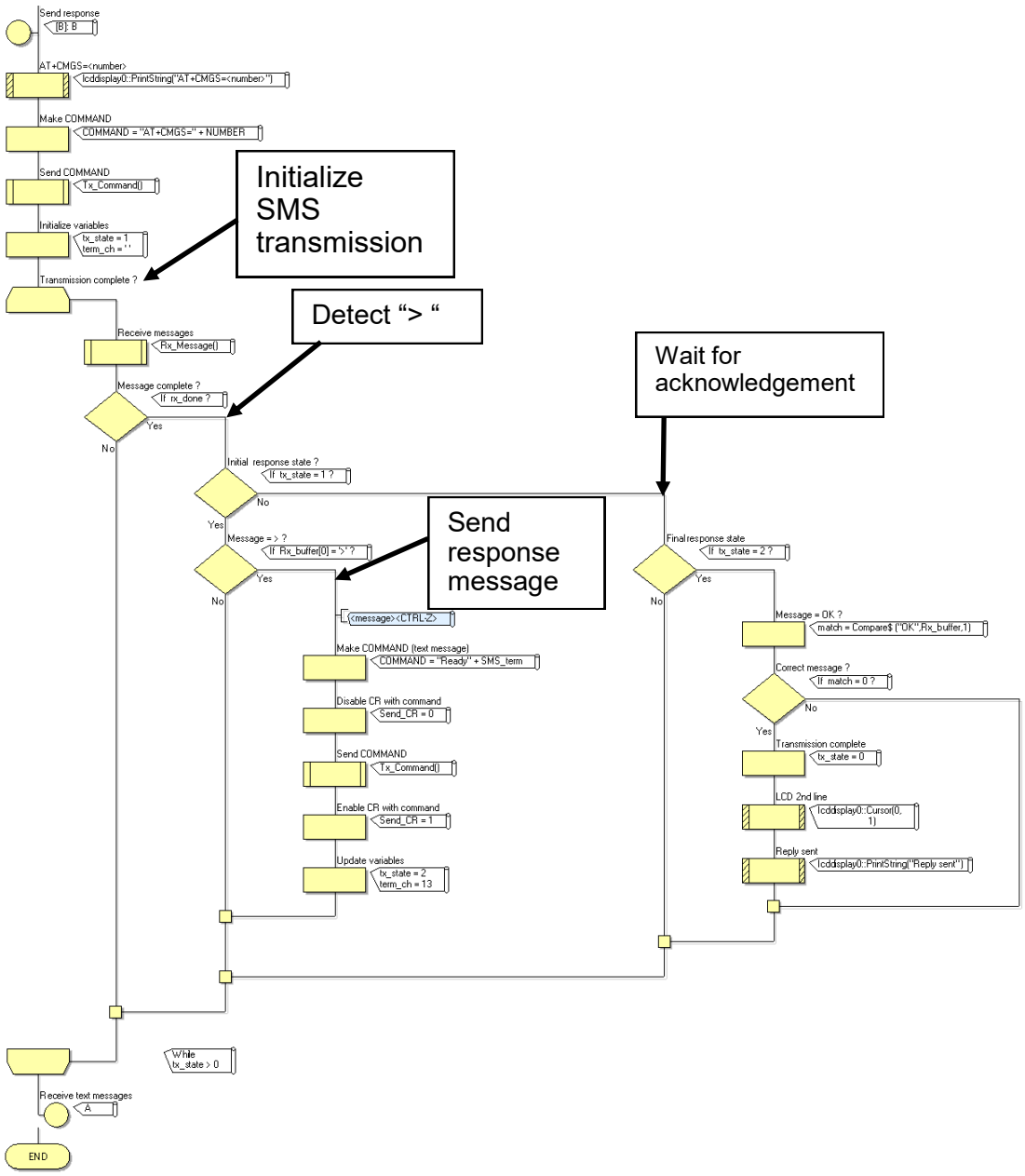
- Receive an incoming text message.
- Extract and interpret sections of the data stream.
- Use message and system data to generate and transmit a response message.

11.3 Requirements

- A multi-programmer board attached to a PC running Flowcode.
- A keypad E-blocks2 board
- A LCD E-blocks2 board
- A GSM modem E-blocks2 board 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.

11.4 The Flowcode program in detail





11.5 What to do

Build the program shown in the two diagrams above.

Download it to the microcontroller in the usual way.

Test it by sending a text message to the Mobile Phone kit and see if there is a response.

11.6 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 (and hence save money!)
- Increase the number of messages that can be identified and used to perform specific functions.

- 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.