RTU8 Compact Station

Getting Started Guide to SMS Funtionality

V. 1.00 / Nov 2010 / Doc 40030

1 Getting Started Guide to SMS Functionality

Today the mobile phone is truly every mans personal communicator and the most widely used standard of mobile phone is GSM.

GSM "bearer" services defined by the GSM governing body called the MoU, or "Memo of Understanding" defines an internationally accepted digital cellular telephony standard which has more than 300 GSM mobile networks in Europe, the US, Asia, Africa and Australia.

The "data over cellular" bearer services are part of the phase 2 implementation of GSM including something called SS 7 (Signalling System Number 7). A robust set of techniques or protocols designed to provide fast, efficient, reliable transfer and delivery of signalling information across the GSM network and to support both switched voice and non-voice applications. The SS 7 protocol enables extremely fast data connections and the SS 7.05 subset of the protocol defines SMS, or short message services, whereby text messages can bepassed to and between GSM mobiles.

SMS or Small Message Service allows you to pass text messages between GSM mobile phones and in the case of the RTU8, to send simple, but important messages/alarms from a remote/unmanned plant to your mobile phone, wherever you may be.

The RTU8 is able to send SMS messages, provided it is connected to a GSM modem, such as the Brodersen UCB-91. The RTU8 will automatically give the correct commands to the modem, manipulation of AT commands being unnecessary. Up to 10 different text messages can be sent to up to 50 different telephone numbers.

Please note that when the GSM modem sends an SMS message it does not go off hook and dial. The message is sent as a background task and the modem will not give any visible sign that the message is being sent. It is also good practice, on power up, to delay sending an SMS message, to give the GSM modem time to initialise. A 10 sec delay, on power up, before messages are sent, works fine.

Where and when these messages are sent is completely under the control of the user, or to be more accurate, the person who makes the simple configuration of the RTU8, at which we shall now look:

There are four pieces of information you need to give the RTU:

The phone number of the mobile phone to which the SMS message is to be sent.

The phone number of the bearer message centre, (the SMS telephone number) often not necessary.

The text of the message to be sent.

When to send the message.

The first three are simply entered via text fields in Ioexplorer and the forth is part of the local task defined with the B-CON programming tool.

To enter the data in Ioexplorer you must first create a project file by connecting the RTU8 to your PC, either directly via a null modem cable or through a modem. Once connected the RTU8 must be scanned by

Ioexplorer. Please refer to the RTU8 Manual for instructions on how to scan an outstation. Once this has been achieved, you can check by looking in the system log (bottom of screen in Ioexplorer) where it will say that a connection exists, highlight the RTU8 on the left of the screen and you will see four tabs appear on the right of the screen. (See fig 1.0)

Entering the Phone Numbers

Now select the config tab and double click on the "Tel. no." field. A box will appear into which you can enter the telephone numbers of the mobile phones to which messages will need to be sent, (See fig 1.1) This box will only appear if you have a connection to the RTU8.

Note that the fields are numbered 0 – 49, 50 in total, and make a note of which telephone number is in which field. You will need this information when you make your B-CON task. If you are entering international numbers prefix the number with a "+" and the call will automatically be routed through an International Service i.e. No need to use 001 etc.

Once you have entered all the numbers hit the "write" button and the numbers will be downloaded into the RTU8, it will take a few seconds, after which you can close this box.

Entering the SMS Telephone Number

Scroll down the config tab and double click on the "SMS Tel. no." field. A box will appear into which you can enter the telephone number. (See Fig 1.3)

This is the number of the SMS message centre though which SMS messages are sent. Often this number is not required and this field can be left blank. Please check with your GSM service provider.

Entering the Message Texts

Scroll down the config tab and double click on the "SMS Text" field. A box will appear into which you can enter the text of the ten different messages that can be sent, (See fig 1.4) Note that the fields are numbered 0 – 9, 10 in total, and make a note of which message is in which field. You will need this information when you make your B-CON task.

This box will only appear if you have a connection to the RTU8.

That completes the configuration of the SMS functions in Ioexplorer. We now need to make the B-CON task, which will control the sending of the SMS messages.

Please refer to the B-CON Getting Started Guide and Manual for instructions on how to use the B-CON editor and make B-CON tasks.

B-CON Configuration

The B-CON task running in the RTU8 needs to decide:

To whom a message should be sent

Which message should be sent

When to send a message

It also needs to monitor that a message has been successfully sent or not.

Selecting the telephone number, to which the message is to be sent, is simply a matter of setting a value in a register. The value will correspond to the number of the relevant field in the list of telephone numbers we entered via Ioexplorer. The first number in the list being 0 and the last 49.

The register is bm3, a byte marker into which we need to place a byte constant.

If we wish to select the first number in the list we would use the command:

mov bc0 bm3 (move a byte constant of 0 to byte marker 3)

and for the second number in the list:

mov bc1 bm3

Similarly selecting the message to be sent, is simply a matter of setting a value in a register. The value will correspond to the number of the relevant field in the list of messages we entered via Ioexplorer. The first number in the list being 0 and the last 9.

The register is bm19, a byte marker into which we need to place a byte constant.

If we wish to select the first message in the list we would use the command:

mov bc0 bm19 (move a byte constant of 0 to byte marker 19)

and for the second number in the list:

mov bc1 bm19

When to send the message needs to be decided by the logic within the B-CON task and we shall look at an example below, but how to trigger the sending of the message we shall look at now.

To instigate the sending of an SMS message we need to write to the Command register, bm2, however this register is not exclusively used for triggering SMS messages, but is also used to make dialup connections and hang-ups.

The Command Register bm2 has four states:

0 – Standby

1 – Dial

2 – Hangup

3 – Send SMS

Thus to send an SMS message we would use the command:

mov bc3 bm2 (move a byte constant of 3 to byte marker 2)

This value only needs to remain in the command register (bm2) for one scan, when it will by picked up by the firmware and actioned. It is therefore good practice to clear the command register at the start of each scan and the following can be placed at the top of your B-CON instructions:

mov bc0 bm2

This will set the Command register to standby, ready for the next action.

Thus to send an SMS message we need only to use three lines of B-CON instructions:

mov bc0 bm3

mov bc0 bm19

mov bc3 bm2

However, in practice we need to do a bit more than this, we have to decide when to send a message and also control the sending of multiple messages. Let's make a simple task.

Example

Assume we have a simple requirement. We have two digital inputs, 0 and 1, and one analogue input, 0. If both digital inputs are activated we need to send message 0 to telephone number 0 and if the analogue input goes higher than 50% (the analogue input has a twelve bit resolution, 4095, therefore 50% is 2048) we need to send message 1 to telephone number 1.

Our control logic for monitoring the digital inputs will be:

```
    ld      i0.0        (load input 0)
    and     i0.1        (and input 1)
    jmpcn   label1      (jump if false)
    mov     c1 m40.0    (set a marker bit(chosen at random))
label1:                 (jump label)
    ld      i0.0        (load input 0)
    and     i0.1        (and input 1)
    jmpc    label2      (jump if true)
    mov     c0 m40.0    (reset the marker bit)
label2:                 (jump label)
```

The above will set the marker bit true when both digital inputs are active and reset the marker bit false when either digital input is not active. The marker bit will be used for triggering the sending of the SMS message.

Our control logic for monitoring the analogue input will be:

```
    ld      wi2000      (load analogue input 0)
    gt      wc2048      (greater than 2048?)
    jmpcn   label3      (jump if false)
    mov     c1 m40.1    (set a marker bit(chosen at random))
label3:                 (jump label)
    ld      wi2000      (load analogue input 0)
    gt      wc2048      (greater than 2048?)
    jmpc    label4      (jump if true)
```

```
    mov     c0 m40.1    (reset the marker bit)
label4:                 (jump label)
```

The above will set the marker bit high when the analogue input is greater than 2048 and reset the marker when the analogue input is equal to or less than 2048. The marker bit will be used for triggering the sending of the SMS message.

We now have the logic to decide when to send our SMS messages, the triggers being the marker bits m40.0 and m40.1. Lets now use these bits to send the SMS messages.

We could use:

```
    ld      m40.0       (load marker bit)
    jmpcn   label5      (jump if false)
    mov     bc0 bm3     (select telephone number 0)
    mov     bc0 bm19    (select message number 0)
    mov     bc3 bm2     (send SMS)
label5: (jump label)
```

This will send message 0 to telephone number 0, but it will also keep sending it as long as the marker bit is high, effectively for as long as the two digital inputs are active. Hundreds of messages could be sent! This is clearly not what we want. We only require one message to be sent when both digital inputs go active, in other words to detect the leading edge.

This can be achieved by using an interlock bit, set at the same time as the Cammand Register is set. The next time the instructions are executed the AND gate will not be true, the interlock bit being true, but negated in the AND gate and the instructions sending the SMS message will be jumped over. When the alarm has cleared the interlock bit will be reset:

```
    ld      m40.0       (load marker bit)
    andn    m40.2       (and the interlock bit negated)
    jmpcn   label6      (jump if false)
    mov     bc0 bm3     (select telephone number 0)
    mov     bc0 bm19    (select message number 0)
    mov     bc3 bm2     (send SMS)
    mov     c1 m40.2    (set interlock bit)
    jmp     label7      (jump)
label6:     (jump label)
    ld      m40.0       (load marker bit)
    jmpc    label7      (jump if true)
    mov     c0 m40.2    (reset interlock bit)
label7:     (jump label)
    ld      m40.1       (load marker bit)
```

```
andn      m40.3      (and the interlock bit negated)
jmpcn     label8     (jump if false)
mov       bc1 bm3    (select telephone number 0)
mov       bc1 bm19   (select message number 0)
mov       bc3 bm2    (send SMS)
mov       c1 m40.3   (set interlock bit)
jmp       label9     (jump)
label8:              (jump label)
ld        m40.1      (load marker bit)
jmpc      label9     (jump if true)
mov       c0 m40.3   (reset interlock bit)
label9:              (jump label)
```

The above will fulfil our requirement, except for one problem.

Lets look at the situation when both alarms are generated almost together, both digital inputs are activated and one second later the analogue input goes higher than 50%. The above instructions will first send the digital alarm to the GSM modem and then almost immediately send the analogue alarm. When the GSM modem receives the command to send the analogue alarm it will not have yet send the digital alarm it received one second earlier. It just has not had the time! What will happen is that the analogue alarm will overwrite the digital alarm and the digital alarm will not be sent.

This is clearly not acceptable. Therefore within our B-CON task we must ensure that when one message is sent, further messages are not sent to the GSM modem, until the modem acknowledges the success or not of sending the first.

When the modem has finished sending an SMS message it will return one of two states into marker bits m20.4 and m40.7 which can be explained thus:

m20.4 set true - The SMS message has been successfully sent. This means that the GSM bearer service has accepted the message for onward transmission. It does guarantee that the message has been received by the mobile phone concerned, the phone may be turned off! m20.4 once set true will remain so until the next SMS message is sent, when it will be cleared down.

m20.7 set true – The sending of the SMS message has failed, the message has not been sent. This could be because the GSM cell is busy, down etc. Altogether 4 attempts will be made to send the SMS message, with a 60 second delay between each attempt, after which m20.7 will be set true. This bit could obviously be used to re-send the SMS message.

Let us then use these bits to ensure that messages are sent one after the other, without loosing any messages.

We can do this by setting a second, common, interlock bit which when set will make a jump over the send instructions until being reset when the SMS message has been sent or failed.

```
ld  m40.4       (load second interlock bit)
    jmpc        label10     (jump if true)
    ld          m40.0       (load marker bit)
    andn        m40.2       (and the interlock bit negated)
    jmpcn       label6      (jump if false)
    mov         bc0 bm3     (select telephone number 0)
    mov         bc0 bm19    (select message number 0)
    mov         bc3 bm2     (send SMS)
    mov         c1 m40.2    (set interlock bit)
    mov         c1 m40.4    (set second interlock bit)
    jmp         label11     (jump)
label6:                     (jump label)
    ld          m40.0       (load marker bit)
    jmpc        label7      (jump if true)
    mov         c0 m40.2    (reset interlock bit)
label7:                     (jump label)
    ld          m40.1       (load marker bit)
    andn        m40.3       (and the interlock bit negated)
    jmpcn       label8      (jump if false)
    mov         bc1 bm3     (select telephone number 0)
    mov         bc1 bm19    (select message number 0)
    mov         bc3 bm2     (send SMS)
    mov         c1 m40.3    (set interlock bit)
    mov         c1 m40.4    (set second interlock bit)
    jmp         label11     (jump)
label8:                     (jump label)
    ld          m40.1       (load marker bit)
    jmpc l      abel9       (jump if true)
    mov         c0 m40.3    (reset interlock bit)
    label9:                 (jump label)
label10:
    ld          m20.4       (load Message Sent bit)
    or          m20.7       (or Message Failed bit)
```

```
    jmpcn     label11      (jump if false)
    mov       c0 m40.4     (reset second interlock bit)
label11:                   (jump label)
```

Please note that when you have set the second interlock bit it is import that you unconditionally jump over the reset instructions (at label10) to label11. The reason being that at this point in time either m20.4 or m20.7 will be set true, from the last sending.

We also need to introduce a delay, on power up, of around 10secs. This gives the GSM modem time to initialise, before we start to send SMS messages. A one shot 10 second delay can created by the following B-Con instructions (assuming the scan time is set to 100msecs):

```
    ld        m152.0       (load interlock)
    jmpc l    abel4a       (jump if interlock is set)
    ld        wm50         (any free marker word)
    add       wc1          (increment by 1 each scan (100 msec))
    st        wm50         (store counter)
    ld        wm50         (load counter)
    lt        wm100        (less then 100? 100 x scan (100msecs)
    jmpc      label17      (jump if yes)
    mov       c1 m152.0    (set interlock when counter > 100)
label4a:
```

Thus 10 seconds after power up the counter will reach 100 and the jump, over the SMS send commands, will not be made. Likewise as the interlock is never reset, this jump will only ever happen once, after power up.

Our complete B-CON task will thus be:

```
main:
    mov       c0 bm2       (reset Command Register)
    ld        i0.0         (load input 0)
    and       i0.1         (and input 1)
    jmpcn     label1       (jump if false)
    mov       c1           m40.0 (set a marker bit(chosen at random))
label1:                    (jump label)
    ld        i0.0         (load input 0)
    and       i0.1         (and input 1)
    jmpc      label2       (jump if true)
    mov       c0 m40.0     (reset the marker bit)
label2:                    (jump label)
```

```
    ld      wi2000      (load analogue input 0)
    gt      wc2048      (greater than 2048?)
    jmpcn   label3      (jump if false)
    mov     c1 m40.1    (set a marker bit(chosen at random))
label3:                 (jump label)
    ld      wi2000      (load analogue input 0)

    gt      wc2048      (greater than 2048?)
    jmpc    label4      (jump if true)
    mov     c0 m40.1    (reset the marker bit)
label4:                 (jump label)
    ld      m152.0      (load interlock)
    jmpc    label4a     (jump if interlock is set)
    ld      wm50        (any free marker word)
    add     wc1         (increment by 1 each scan (100 msec))
    st      wm50        (store counter)
    ld      wm50        (load counter)
    lt      wm100       (less then 100? 100 x scan (100msecs)
    jmpc    label17     (jump if yes)
    mov     c1 m152.0   (set interlock when counter > 100)
label4a:
    ld      m40.4       (load second interlock bit)
    jmpc    label10     (jump if true)
    ld      m40.0       (load marker bit)
    andn    m40.2       (and the interlock bit negated)
    jmpcn   label6      (jump if false)
    mov     bc0 bm3     (select telephone number 0)
    mov     bc0 bm19    (select message number 0)
    mov     bc3 bm2     (send SMS)
    mov     c1 m40.2    (set interlock bit)
    mov     c1 m40.4    (set second interlock bit)
    jmp     label11     (jump)
label6:                 (jump label)
```

```
    ld          m40.0       (load marker bit)
    jmpc l      abel7       (jump if true)
    mov         c0 m40.2    (reset interlock bit)
label7:                     (jump label)
                ld m40.1    (load marker bit)
    andn        m40.3       (and the interlock bit negated)
    jmpcn       label8      (jump if false)
    mov         bc1 bm3     (select telephone number 0)
    mov         bc1 bm19    (select message number 0)
    mov         bc3 bm2     (send SMS)
    mov         c1 m40.3    (set interlock bit)
    mov         c1 m40.4    (set second interlock bit)
    jmp         label11     (jump)
label8:                     (jump label)
    ld          m40.1       (load marker bit)
    jmpc        label9      (jump if true)
    mov         c0 m40.3    (reset interlock bit)
label9:                     (jump label)
label10:
    ld          m20.4       (load Message Sent bit)
    or          m20.7       (or Message Failed bit)
    jmpcn       label11     (jump if false)
    mov         c0 m40.4    (reset second interlock bit)
label11:                    (jump label)
ep
```

If you cut and paste these instructions into a B-CON editor please ensure that the comments in brackets are prefixed with a forward slash (/).

The above may not be exactly what you need for your application but it will, hopefully, give you head start in implementing SMS functions into an RTU8.