Technical Note on *New MCM* program

using USB-RS485 FTDI converter cable

June 29th 2012

This short document describes Software and hardware aspect of New program for MCM which can be used for MCM communication on Linux operating system

GMRT-NCRA
Pune

Concern GMRT Team :

R Balasubramanian
Raju Uprade
Charu Kanade
C. Satheesh
Naresh Sisodiya
Mahadev Mishal

Table of Content

Introduction

Monitor and Control Modules (MCM) are the final part of the Telemetry System. They are single cards distributed at all the remote antennas and inside the CEB, doing the interface to all the settable GMRT sub units. The typical monitor points are LO levels, power supply voltage and switch status of Front End, Intermediary Frequency and Baseband systems.

Typically a MCM scans some specified analog channels, digitizes the signals, and stores the data into its internal memory. It can also output a 16-bit word for various control applications (control lines of external multiplexers, power switching, electromechanical devices control...).

Each MCM is connected to an Antenna Based Computer (ABC) with a RS-485 shared serial link, using the 9th bit multi-drop protocol (explained in next chapter). The basic functions of the MCM software is to accept a command from ABC, to execute it, and to transmit the answer to ABC. The list of available commands is also detailed further.

1.1. Linux PC / MCM communication :

The MCM / ABC connection can be replaced by a MCM / PC link, so that any MCM can be used directly from a PC, without passing through the whole Telemetry chain. Any point accessible from a MCM can thus be directly controlled, monitored and tested for maintenance or evolution purposes.

We have a Linux based mcmcom program which uses a serial device driver called mcm-driver for MCM communication using 9th bit protocol. This Linux based is running successfully in Lab as well as Antenna base.

1.2 The RS-485 multi-drop protocol (9th bit protocol) :

The Linux PC is following the same protocol with MCMs than any ABC does. So the PC can be connected to many MCMs through a shared RS-485 serial link. That means that data will be transferred on the same lines regardless to which MCM they are associated with. This paragraph explains how the “9th bit” protocol, also called “multi-drop” protocol, organizes this communication.

In this configuration, the PC is the communication host. MCMs cannot spontaneously write to the PC, they only answer the host whenever they are requested. The PC talks to only one MCM at a time, others remaining dormant. By default the PC port is in reading mode, it just switches to its writing mode only for the time it writes to a MCM (cf. hardware paragraph).

Two kind of bytes can be transmitted from the PC: the address bytes and the data bytes. In asynchronous serial communication, a byte is sent with a start bit first, then the 8 bits, an optional parity bit, and a stop bit. In this application, the 9th parity bit is set to '1' when the byte is an MCM address, and to '0' otherwise.

The PC sends a packet of bytes, among which the first byte is an address one. Originally all MCMs are in a state where they can only read an address byte. If a MCM recognizes its address, then it reads the following data bytes, processes the command and answers to the PC.

1.3 Problem statement :

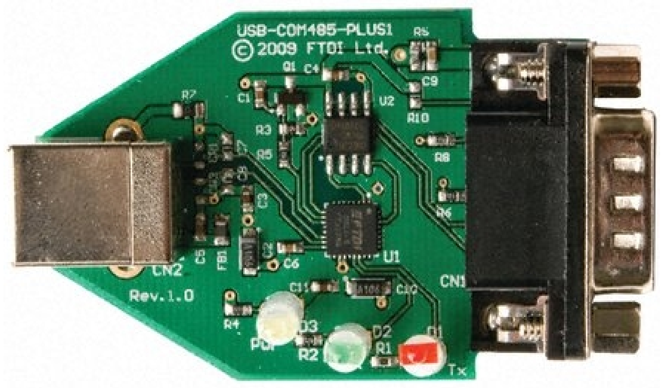
As new laptop don't have serial port, so we cannot use mcmcom program on laptop for field testing.We were in need to find a suitable hardware and software solution so that laptop can be used for field testing. Intially we purchased USB-To-Serial port product of Prolific Technology, these were normal serial port which supports even, odd and none parity only.As we are using RS485 multidrop protocol, We needed the mark and space parity to specify address byte and data bytes but that wasn't possible using simple USB-To-Serial port through application program. We changed the PL2303 device driver to suit our need.We were successful in transmitting data from PC to MCM but eventually all hardware device failed.

Technical Solution

2.1 Hardware device :

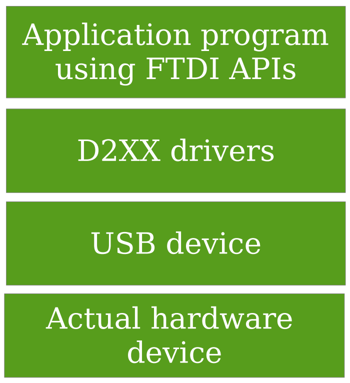
As all our low cost hardware device failed, we searched for standard USB-To-Serial device which will have support for mark and space parity.we found out product from FTDI meets out requirement. After going through the technical data sheet we finalized two products from FTDI.

USB to RS485 converter cable



USB to RS485 Adapter Module

2.2 Application Program :



Software Architecture

As both product has support for mark and space parity, we do not need to go into the device driver level. An application program solved our purpose.

Figure shows the software architecture for the New MCM program. On the top is the application program which communicated to the D2XX drivers using FTDI APIs. D2XX drivers allow direct access to the USB device through a DLL. Application software can access the USB device through a series of DLL function calls. The functions available are listed in the D2XX Programmer's Guide document.

FTDI provides ***libftdi*** library which we need to install on our PC to complie the program.

Using the FTDI APIs we were able to write an application program which is used for MCM communication.

List of commands available in the MCM program :

0	set_mcm_address
1	nul_cmd
2	set_idle_mode
3	set_scan_mode
4	set_an_mask
5	set_dmask_16b
6	set_dmask_32b
7	read_an_mask
8	read_dmask_16b
9	read_dmask_32b
10	read_version
11	read_mode
12	reboot
13	FE_control_old
14	feed_data_monitoring
15	FE_control_new
16	fe_box_mon
17	common_box_mon
18	set_dmask_64b
19	read_dmask_64b
20	set_mean_mode
21	SET_LO
22	SET_FE
23	SET_IF
24	EXIT

Testing in ABR lab and Receiver room :

ABR lab has been given USB-RS485 device number "*FTUN7KZE*" for their lab testing. While testing in ABR lab and Receiver room we encountered two problems.

- New MCM program was working fine with Telemetry laptop in both places but it was not working with ABR lab laptop as well as with receiver room machine. So we adjusted the delay between address and data bytes to 4000 microseconds after which basic communication started working.
- While trying to Set digital 16,32 and 64 bit, we found out that we have to reverse the byte order in order to set the digital mask. We modified the New MCM program code to have the reverse byte order logic. With the modified code, we were able to set 155, 325, 255 MHz LO.

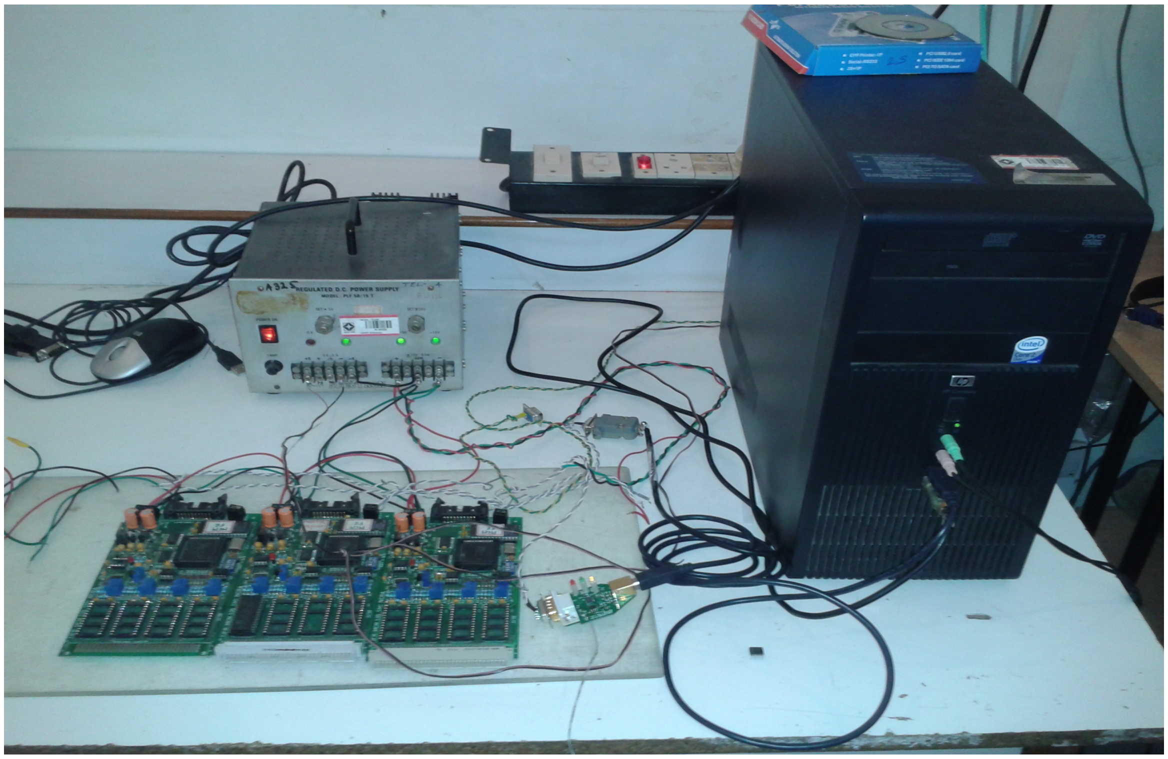
Testing in Front end lab:

Front end lab has been given USB-RS485 device number ***FTSERRS1*** for their lab testing. We installed the New MCM program using USB-Rs-485 device on old laptop. We successfully tested the program in Front end lab with their hardware setup. We were able to set bandwidth, noise,swap,RF and attenuation.

- While testing in Front end lab, Mr. Anil Raut suggested to write a script to disable the default USB driver at the time of plugging in the USB to RS 485 converter module to PC/LAPTOP (as required by program).
- We made the suggested changes in the mcmtest.c program,It looks for the default drivers and if drivers are loaded, program removes the drivers. Now there is no need to manually unload the default drivers.
- Modified MCM program has been installed in Front end lab and ABR lab,and it's working fine.

Snapshots :

Snapshot shows both hardware device plugged into the PC USB port and on the other side connected to multiple MCMs.



Snapshot below showing the output terminal when we run the program :

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ TIME TO ENTER NEW COMMAND $$$$$$$$$$$$$$$$$$$$$$$$$$$$
0: set_mcm_address
1 : nul_cmd
2 : set_idle_mode
3 : set_scan_mode
4: set_an_mask
5 : set_dmask_16b
6 : set_dmask_32b
7: read_an_mask
8: read_dmask_16b
9: read_dmask_32b
10: read_version
11: read_mode
12: reboot
13: FE_control_old
14: feed_data_monitoring
15: FE_control_new
```

```
16: fe_box_mon
17: common_box_mon
18: set_dmask_64b
19: read_dmask_64b
20: set_mean_mode
21: SET_LO
22: SET_FE
23: SET_IF
24: EXIT

Enter New Command from the list==>1
nul_cmd
Mcm addr 2
 Device opened successfully
 Device SET timeout ok
FT_SetRTS(0)
FT_SetBaudRate(0)
SET FT_SetDataCharacteristics(0)
SET FT_SetDataCharacteristics(0)
FT_ClearRTS(0)
######### RESPONSE FROM THE SYSTEM###########
MCM ADDRESS IS=>2
PACKET LENGTH IS=>10
MCM RESPONSE BUFFER [0] = 0x00
MCM RESPONSE BUFFER [1] = 0x00
MCM RESPONSE BUFFER [2] = 0x00
MCM RESPONSE BUFFER [3] = 0x01
MCM RESPONSE BUFFER [4] = 0x04
MCM RESPONSE BUFFER [5] = 0x00
MCM RESPONSE BUFFER [6] = 0x04
MCM RESPONSE BUFFER [7] = 0x00
MCM RESPONSE BUFFER [8] = 0xED

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ TIME TO ENTER NEW COMMAND $$$$$$$$$$$$$$$$$$$$$$$$
```

SOP :

1. The MCM communication program is in Hello/MCM directory.

2. Each USB-RS485 cable has a unique serial number,which is hardcoded in the program, So if in future we use a different USB-RS485 cable,we have to recompile the program with new unique serial number of the device.

3. Kindly note you have to be root in order to use this program as it uses USB device drivers.

4. Now enter into the directory and give ./mcmtest.

Future enhancement :

This version 1 software will go through lots of enhancement like, We will try to develop a GUI for this New MCM program.

References :

- D2XX programmers guide by FTDI
- Linux device drivers
- USB host architecture

Appendix :

For ABR lab testing command number 20. SET_LO has been implemented in the new MCM program. SET_LO command ask for two frequency LO1 and LO2 as input then the program internally generates the proper 32 and 64 bit hex arguments for the LO1 and LO2. After generating the proper arguments, program calls the *set_dmask_32 and set_dmask_64* commands to set the LO1 and LO2 frequency.

Snapshot of command 21:

```
Enter command number =>21
ENTER <LO1>  <LO2> : 155 680
lo1= 155, lo2= 680
04F1A 040E5
: 2
##################################################
 32 MASK bit 01000 09000
##################################################
32 HEX bit Argument generated 0 1 0 9
##################################################
64 MASK bit040E5 0C0E5 2FFF AFFF
##################################################
64 bit HEX Argument generated e 4 e c ff 2f ff af

 :2
##################################################
64 MASK bit0168 8168 1001 9001
##################################################
64 bit HEX Argument generated 68 1 68 81 1 10 1 90
##################################################
 32 MASK bit 6005 E015
##################################################
32 HEX bit Argument generated 5 60 15 e0
:3
##################################################
 32 MASK bit 01000 09000
##################################################
32 HEX bit Argument generated 0 1 0 9
##################################################
64 MASK bit40C6 C0C6 2FD3 AFD3
##################################################
64 bit HEX Argument generated c6 40 c6 c0 d3 2f d3 af

:3
##################################################
64 MASK bit0258 8258 1008 9008
##################################################
64 bit HEX Argument generated 58 2 58 82 8 10 8 90
##################################################
 32 MASK bit 6005 E015
##################################################
32 HEX bit Argument generated 5 60 15 e0
```