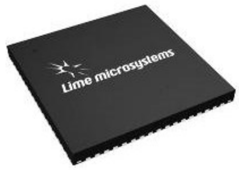

MyriadRF Board Control Implementation Using Zipper Board

- Description -

Rev: 1.0r02
Last modified: 11/10/2013

Contents

Revision History

Version 1.0r00 - 1.0r02
Started:12 Aug, 2013
Finished: 11 Oct, 2013
Initial version

1

Introduction

This document describes MyriadRF board control implementation using "Zipper" board. Document consists of a chapters that describes data exchange protocol between the computer and board, supported commands, detailed procedure of the drivers installation, firmware compilation, microcontroller programming and PC and communication via virtual COM port.

In this document all developments and installation procedures are done in the "Windows" environment.

2

Protocol

Protocol used for exchanging data between the computer and the board. This chapter describes the protocol version 1.

2.1 Packet structure

All data from/to board is transferring using fixed length (64 bytes) packets. Packets consist of constant size header (8 bytes length) and data field. Data field size can be calculated as Data_field_size = LMS_Ctrl_packet_size – Header_size.

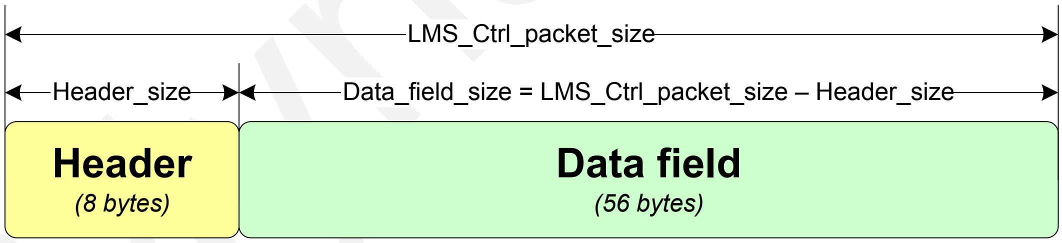

Figure 1 Packet structure

Header consist of several fields.

Table 1: Header structure

Field	Byte index	Description
Command	0	Command code
Status	1	Status
Data blocs	2	Data blocks in data field
Reserved	3-7	Reserved bytes for future use

2.2 Request – respond method

Personal computer's software must initiate command with sending request packet via control interface. When board receives request packet, board processes packet and form a respond packet. Depending on control interface, response packet may be send automatically after command execution (virtual COM port) or it may be required to read manually (USB endpoint).

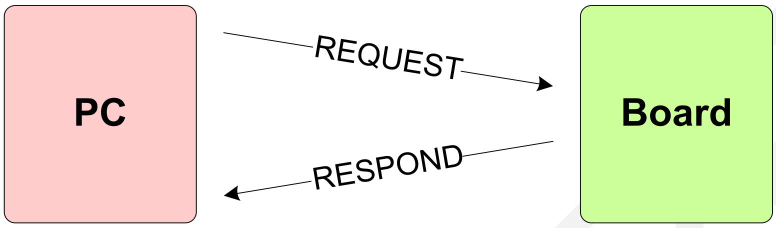

Figure 2 Request - respond mechanism

All response packets in the header returns last executed (or executing) command's code and its status in the header.

Table 1: Status values

Value	Value	Description
STATUS_COMPLETED_CMD	1	Command successfully executed.
STATUS_UNKNOWN_CMD	2	Unknown command.
STATUS_BUSY_CMD	3	Command execution not finished.
STATUS_MANY_BLOCKS_CMD	4	Too many blocks
STATUS_ERROR_CMD	5	Error occurred during command execution.

Packet's data field's structure depends on command. This is described in next chapter.

3

Commands

3.1 Zipper commands

Zipper board's supported commands are listed below.

Table 2: Zipper board's commands

Command	OP code	Description
CMD_GET_INFO	0x00	Returns some info about board and firmware
CMD_SI5351_WR	0x13	Writes data to SI5351 (clock generator) via I^2C
CMD_SI5351_RD	0x14	Reads data from SI5351 (clock generator) via I^2C
CMD_LMS_RST	0x20	Sets new LMS7002M chip's RESET mode (inactive, active, pulse)
CMD_LMS6002_WR	0x23	Writes data to LMS6002 chip via SPI
CMD_LMS6002_RD	0x24	Writes data to LMS6002 chip via SPI
CMS_LMS_LNA	0x2A	RF input selection
CMS_LMS_PA	0x2B	RF output selection
CMD_ADF4002 _WR	0x31	Writes data to ADF4002 (phase detector/frequency synthesizer) via SPI

Packet's data field's structure depends on command. Some commands data field may contain blocks (for WR, RD commands), some have static structure and some don't use data field at all. If data field consist of blocks it is important to set proper number of blocks in packet header.

The following text describes the details of all current realization supported commands.

3.1.1 CMD_GET_INFO

To get information about protocol, device type and firmware from board you need to send packet with CMD_GET_INFO command in the header.

Request packet's data field: don't care.

Table 3: Response packet's data field

Field	Byte index	Description
FW_ver	0	Firmware version
Dev_type	1	Device type
Protocol_ver	2	Protocol version
Reserved	3 – (Data_field_size – 1)	Reserved bytes for future use

3.1.2 CMD_SI5351_WR

To write data to SI5351 (clock generator) via I2C you need to send packet with CMD_SI5351_WR command and one or more address – data pairs (blocks) to USB microcontroller. Address field length is 8 bits (1 byte).

Table 4: Request packet's data field

Block index	Field	Byte index	Description
0	Reg_addr	0	Register address
	Reg_data	1	Register data
1	Reg_addr	2	Register address
	Reg_data	3	Register data
n	Reg_addr	n*2	Register address
	Reg_data	n*2+1	Register data

Response packet's data field: don't care.
If SI5351 will not respond ACK, status will be STATUS_ERROR_CMD.

3.1.3 CMD_SI5351_RD

To read data from SI5351 (clock generator) via I2C you need to send packet with CMD_SI5351_RD command to USB microcontroller and one or more addresses (blocks) you want to read from. Address field length is 8 bits (1 byte).

Table 5: Request packet's data field

Block index	Field	Byte index	Description
0	Reg_n_addr	0	Register address
1	Reg_n_addr	1	Register address
n	Reg_n_addr	n	Register address

Table 6: Response packet's data field

Block index	Field	Byte index	Description
0	Reg_addr	0	Register address
	Reg_data	1	Register data
1	Reg_addr	2	Register address
	Reg_data	3	Register data
n	Reg_addr	n*2	Register address
	Reg_data	n*2+1	Register data

Response packet's data field: don't care.
If SI5351 will not respond ACK, status will be STATUS_ERROR_CMD.

3.1.4 CMD_LMS_RST

It is possible to deactivate, activate or make pulse on LMS chip's reset line. For this, you need to send packet with CMD_LMS_RST. First byte (index 0) in data field indicates reset line's mode.

Table 7: Request packet's data field

Field	Byte index	Description
LMS_RST_MODE	0	Reset line mode

Table 8: LMS_RST_MODE meanings

LMS_RST_MODE	Description
0	LMS_RST_DEACTIVATE
1	LMS_RST_ACTIVATE
2	LMS_RST_PULSE

Response packet’s data field: don’t care.

3.1.5 CMD_LMS6002_WR

To write data to LMS6002 chip via SPI you need to send packet with CMD_LMS6002_WR command and one or more address – data pairs (blocks) to USB microcontroller. Address and data field lengths are 8 bits (1 byte). MSB bit in address (R/W bit) is ignored, so it is possible to use same address for write/read operations (like in datasheet). LMS6002 SPI addresses are described in “SPI Register Map” document [1].

Table 9: Request packet's data field

Block index	Field	Byte index	Description
0	Reg_addr	0	Register address
	Reg_data	1	Register data
1	Reg_addr	2	Register address
	Reg_data	3	Register data
n	Reg_addr	n*2	Register address
	Reg_data	n*2+1	Register data

Response packet’s data field: don’t care.

3.1.6 CMD_LMS6002_RD

To read data from LMS6002 chip via SPI you need to send packet with CMD_LMS6002_RD command to USB microcontroller and one or more addresses (blocks) you want to read from. Address field length is 8 bit (1 byte). MSB bit in address (R/W bit) is ignored.
LMS6002 SPI addresses are described in “SPI Register Map” document [1].

Table 10: Request packet's data field

Block index	Field	Byte index	Description
0	Reg_addr	0	Register address
1	Reg_addr	1	Register address
n	Reg_addr	n	Register address

Table 11: Response packet's data field

Block index	Field	Byte index	Description
0	Reg_addr	0	Register address
	Reg_data	1	Register data
1	Reg_addr	2	Register address
	Reg_data	3	Register data
n	Reg_addr	n*2	Register address
	Reg_data	n*2+1	Register data

3.1.7 CMD_LMS_LNA

To select RF input you need to send packet with CMD_LMS_LNA command to USB microcontroller and required data. This command controls RF inputs MUX. First byte (index

0) in data field indicates RF input selection. Only two least significant bits of LMS_LNA are used.

Table 12: Request packet's data field

Field	Byte index	Description
LMS_LNA	0	RF input selection

Table 13: LMS_LNA meaning

LMS_LNA[1] (V1) (GPIO1)	LMS_LNA[0] (V0) (GPIO0)	Description
0	0	RXIN3
0	1	RXIN2
1	0	NC
1	1	RXIN1

3.1.8 CMD_LMS_PA

To select RF output you need to send packet with CMD_LMS_PA command to USB microcontroller and required data. This command controls RF outputs MUX. First byte (index 0) in data field indicates RF output selection. Only one least significant bit of LMS_PA is used.

Table 14: Request packet's data field

Field	Byte index	Description
LMS_PA	0	RF output selection

Table 15: LMS_PA meanings

LMS_PA[0] (V0) (GPIO0)	Description
0	TXOUT1
1	TXOUT2

3.1.9 CMD_ADF4002_WR

To control ADF4002 (phase detector/frequency synthesizer) chip using USB microcontroller you need to send packet with CMD_ADF4002_WR command and then required data. Each block consist of 3 bytes of data (MSB first). Each block's two least significant bits are control bits and these bits selects one of four ADF4002 destination latches.

Table 16: Request packet's data field

Block index	Field	Byte index	Description
0	Data_MSB	0	Data (MSB)
	Data	1	Data
	Data_LSB	2	Data (LSB)
1	Data_MSB	3	Data (MSB)
	Data	4	Data
	Data_LSB	5	Data (LSB)
n	Data_MSB	n*3	Data (MSB)
	Data	n*3+1	Data
	Data_LSB	n*3+2	Data (LSB)

Response packet's data field: don't care.

3.2 Commands usage examples

In this chapter are shown some commands usage examples, packets.

3.2.1 Sending reset pulse

Lets say we want to make reset pulse for LMS chip. So, we need to send this packet to USB microcontroller:

```
0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

In this packet first header byte (0x20) says that it is “LMS chip’s RESET mode” command (CMD_LMS7002_RST). Data field’s first byte (index 0) (0x02) indicates what mode to use (LMS7002_RST_PULSE).

After that microcontroller will respond:

```
0x20 0x01 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

In this packet first header byte (0x20) says that “LMS7002M chip’s RESET mode” (CMD_LMS7002_RST) command executed. Second header byte (0x01) indicates status of executed command (STATUS_COMPLETED_CMD).

4

Microcontroller’s firmware

Microcontroller’s firmware implements described protocol, commands, USB, I2C, SPI and all required functions using hardware and software capabilities. Firmware use LUFA library (Lightweight USB Framework for AVRs, formerly known as MyUSB) [2].

Source can be compiled using GCC tools.

4.1 Firmware compilation

Firstly, you have to install WinAVR [3]. If you need to customize firmware, you can modify source before compilation. There are two methods of compiling project: using console or using “Programmer’s notepad”.

4.1.1 Firmware compilation using console

Open console and make active project directory “Zipper\firmware project”.

This can be done by typing “cmd” in Run field and pressing “Enter”.

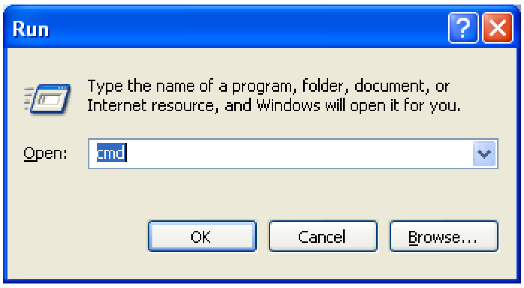

Figure 3 Starting console

You will see black window, where you can type some commands. Then you have to set active directory by using “cd” command „cd path_to_the_project_folder“. For example „cd c:\projects\Zipper\firmware project “.

In “Windows 7” there is simpler way to set active directory. Press “Shift” key on the keyboard and simultaneously press right mouse button on project’s folder (“firmware source”). In context menu select „Open command window here“.

If everything is ok, you will work in your project directory. Project compilation can be done by typing “make” and pressing “Enter” key. If you were to make serious changes to the source before compilation need to execute the command “make clean”.

If there are no errors, project will be compiled as shown in Figure 4. Some warning could be displayed. Compiled firmware file will be placed in project directory as file with *.hex extension.

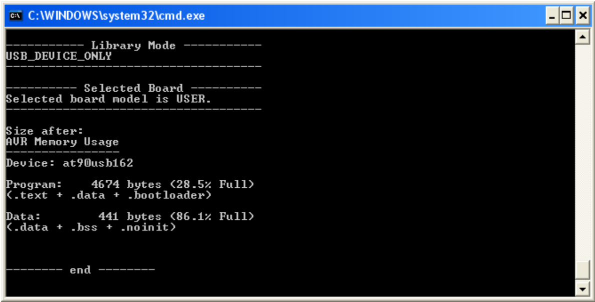

Figure 4 Console window after successful compilation

4.1.2 Firmware compilation using “Programmer’s notepad”

Firstly start “Programmers notepad” (“Start -> WinAVR -> Programmers Notepad [WinAVR]”). This program usually installed along with WinAVR. Open project file by clicking. “File -> Open Project(s)…” and selecting “zipper.pnproj” which is located in folder “firmware project”.

Press “Tools -> [WinAVR] Make Clean” and then “Tools -> [WinAVR] Make All”. All information will be printed in the “Output” section of “Programmer’s notepad”.

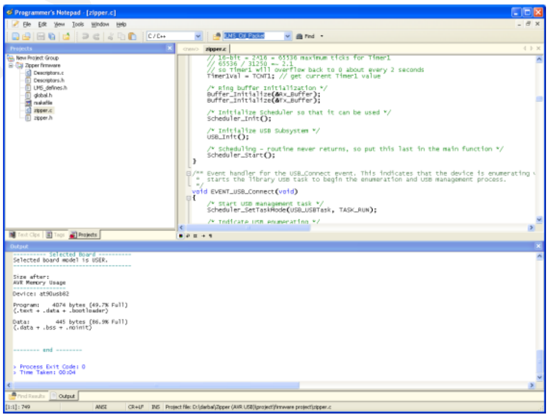

Figure 5 “Programmer’s notepad” window after successful project compilation

4.2 Programming AVR microcontroller

Microcontroller can implement various of functions that depends on programmed firmware.

After successful compilation firmware file “zipper.hex” will be created in the project directory. This firmware file must be programmed into AVR microcontroller. This can be done by programming microcontroller via USB interface without any additional tools (programmers, debuggers).

Firstly, install “Atmel’s” tool “FLIP” (Flexible In-System Programmer) [4].

Then boot AVR microcontroller into bootloader mode. This is done through the following steps:

- plug your board to USB port
- pressing the RESET and HWB buttons
- release the RESET button
- release the HWB button

After this procedure you may need to install drivers from “FLIP” installation directory (in our case “C:\Program Files\Atmel\Flip 3.4.7\usb”). This is described in section “Installing bootloader’s drivers”.

From this point there are several methods of programming microcontroller. Choose the most suitable programming method. These methods are described below.

4.2.1 Programming USB microcontroller using “FLIP” application

Start FLIP application (Start -> Flip 3.4.7 -> Flip 3.4.7).

Under “Device”, click “Select” and then choose the “AT90USB162” microcontroller.

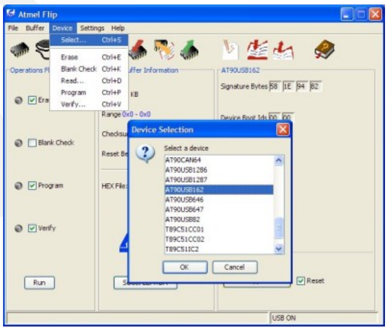

Figure 6 Microcontroller selection

Then press “Settings -> Communication -> USB” and another program window will pop up. In this window press “Open” button.

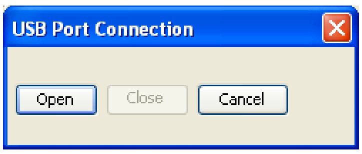

Figure 7 Opening USB port connection

If all the steps have been completed successfully all controls in FLIP's window will become active and in the status bar (on the bottom right) will be printed "USB ON".

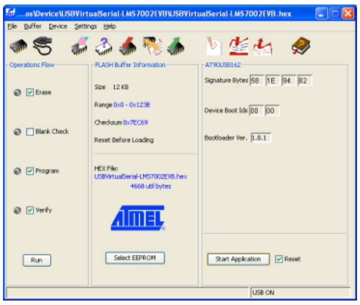

Figure 8 Software successfully connected to the microcontroller

Press "File ->Load HEX File…" and load compiled firmware (hex file) from project directory.

Now select the "Erase", "Program", and "Verify" check boxes and click "Run" to program the board.

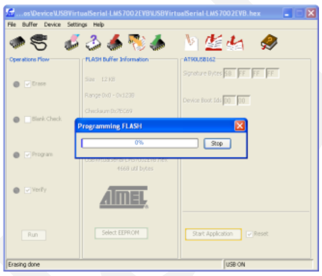

Figure 9 Programming microcontroller

If all was successful, the lights should be green. This means that the microcontroller programming was successful.

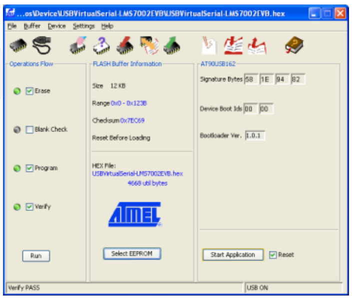

Figure 10 Successful programming

Now press "Start Application" button to start your custom firmware.

After this procedure you may need to install drivers for serial device. This is described in section "Installing serial drivers (Windows XP)".

4.2.2 Programming USB microcontroller using console

Another method to program the USB microcontroller is to use console command “make program”. After this command successful execution, microcontroller will be programmed and restarted.

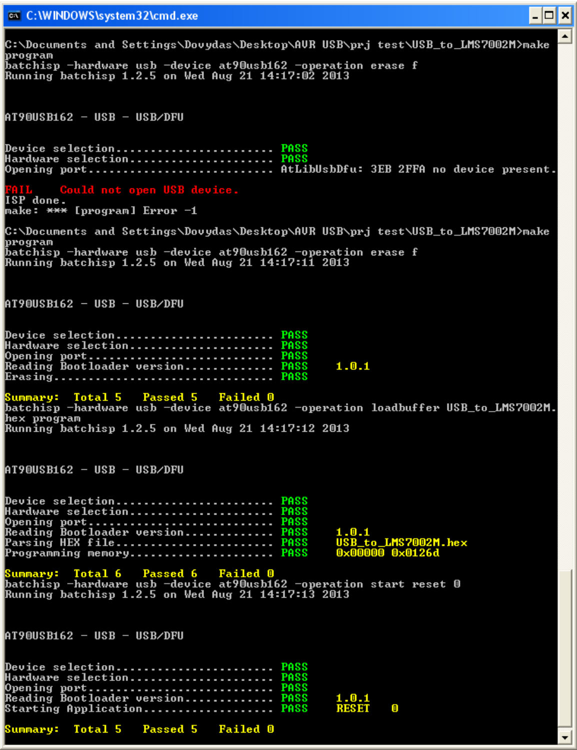

Figure 11 microcontroller programming from console

4.2.3 Programming USB microcontroller using “Programmer’s notepad”

Also it is possible to program microcontroller from “Programmer’s notepad” environment by pressing “Tools -> Program”.

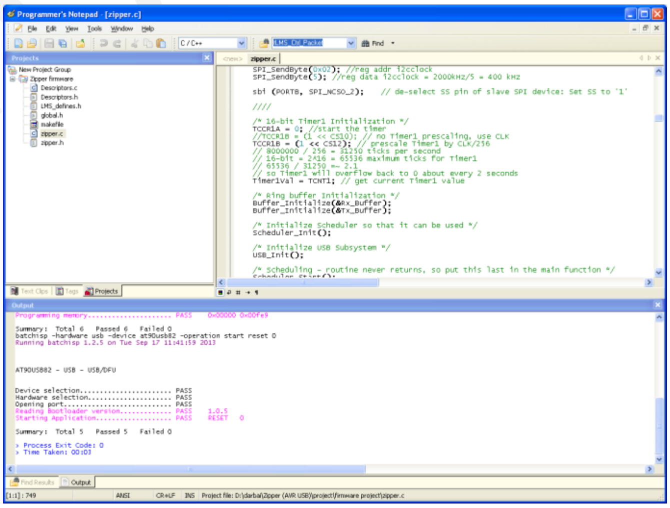

Figure 12 microcontroller’s programming from “Programmer’s notepad” environment

4.3 Drivers installation

This section describes how to install drivers.

4.3.1 Installing serial drivers (Windows XP)

You may need to be logged in as Administrator to perform the following.

Plug in your USB board (You may need to be logged in as Administrator to perform the following) to the free USB port on your Windows machine. Windows' *New Hardware Wizard* should appear. You should proceed by telling Windows to install its own Virtual Serial port drivers. So, after installation procedure begins, DO NOT let Windows search as it will not find anything.

Figure 13 Windows driver installation, Step 1

Next, You will want to install from a specific location.

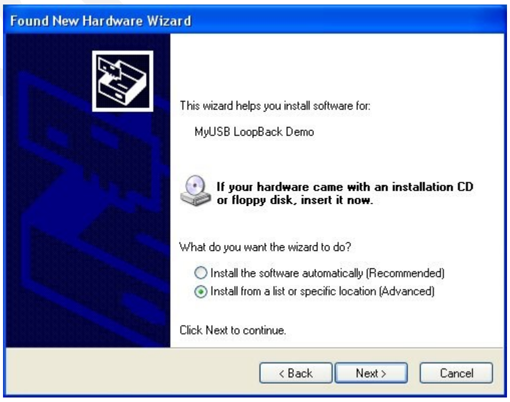

Figure 14 Windows driver installation, Step 2

Next, you need to browse for driver folder, which can be found in folder "Zipper\drivers\LMS VCP drivers". Drivers should work fine for Windows XP, Windows Vista and Windows 7.

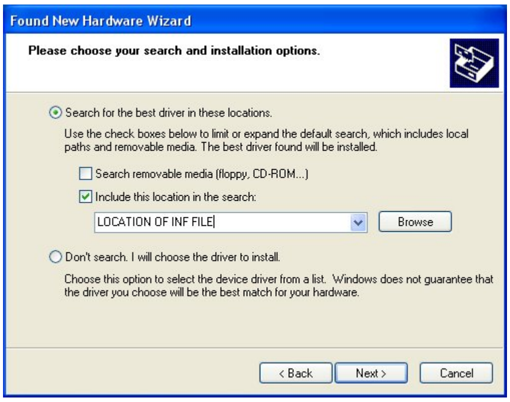

Figure 15 Windows driver installation, Step 3

Windows should proceed to install drivers. Enumeration process should start now. If everything is successful unplug and then plug in your device again (or RESET the device) to be able to use it.

4.3.2 Installing bootloader's drivers

Same procedure as in previous section except that the driver location is from flip directory (in our case "C:\Program Files\Atmel\Flip 3.4.7\usb").

5

PC and board communication via virtual COM port

USB microcontroller emulates COM port and in system it will be seen as virtual COM port. To control LMS6002 chip you need to write and read the commands and data using standard COM port driver.

5.1 COM port settings

The software must open COM port using those settings:

Table 17: COM port settings

Setting	Description
Port	Determine to what port USB controller is connected (see 2.2 Determining Serial Port)
Baud Rate	Baud rate defines SPI bus clock frequency: 9600 baud means 4Mhz SPI clock 14400 baud means 2Mhz SPI clock 19200 baud means 1MHz SPI clock 38400 baud means 500kHz SPI clock 57600 baud means 250kHz SPI clock 115200 baud means 125kHz SPI clock
Data	8 bit
Parity	None
Stop	1 bit
Flow Control	None

5.2 Determining serial port

After enumeration (USB term meaning *"connect and establish communication with"*) Windows will assign to your USB Virtual Serial device a serial port – **COM?**.
Right-Click on **My Computer**, then click **Properties**, then the **Hardware** tab, then **Device Manager**, then find "LMS Virtual COM Port" under "Ports (COM & LPT)". Note that on this system it has enumerated as "COM7".

Figure 16 Finding Enumerated USB Port

6

References

1. Lime Microsystems, LMS6002 – MULTI-BAND, MULTI-STANDARD MIMO RF TRANSCEIVER IC (SPI Register Map).
2. Dean Camera, Lufa. Link: http://www.fourwalledcubicle.com/LUFA.php
3. Winavr. Link: http://winavr.sourceforge.net
4. Atmel, Flip. Link: http://www.atmel.com/tools/FLIP.aspx