MagicConfig Handbook
Version 1.0

This page is intentionally left blank.

Revision History		
Version	**Date**	**Updates**
1.0	070516	Document public release.

1 Overview

MagicConfig is a common configuration system to ease configuration and controlling the behavior of our development boards. A common PC utility is used to manage the interface called MCF, on the board itself a configuration MCU handling the tasks. The same code base is used on all platforms and only the application layer differs for making board specific tasks. This document describes the general functions of MagicConfig and how its handled, for board specific functions and commands please refer to documentation on each board and/or quick help by calling MCF utility with option 'MCF -?:<targetname>'. System can consist of a network of devices e.g. Bitfire with Atmosfire module. In such case the only connection you need is to the Bitfire board, you will then reach the Atmosfire module trough its I2C bridge. Several boards can be connected in addition such as application boards etc.

2 Interfaces

As of today 2 interfaces are supported UART (COM) and I2C, UART is used as a PC entry point and communication between modules are supported by I2C.

2.1 UART

The default interface for connecting a PC is UART connection trough a RS232 port and you do not have to think about setting any special mode to have working link. A magic packet is sent to the UART in question and the configuration MCU will listen the data coming on the port, if the magic packet is recognized it will lock up the UART port and communicate with the utility. After the task is finished the port can close again and leave the serial channel for the user again.

2.2 I2C

The purpose of the I2C interface is to enable a stack of modules to communicate, this means that several modules can be connected to the same 2-wire network. If the user hook up the MCF utility to a UART equipped node in the network this can act as a bridge and you will have access to all the nodes from a single connection e.g. Bitfire connection reaches the Atmosfire configuration trough the bridge.

3 Software upgrade

We have learned that housekeeping is a simple task but yet tricky to get it perfect first time. MagicConfig firmware is based on two parts, the boot monitor and the application. Application part which handles board specific tasks can be uploaded trough the same utility without the need to solder in connectors and attach programmers for the different configuration MCUs. This enables us to add functionality of the housekeepers when we get good ideas and feedback, you can download it in your board already lying on your desk.

4 Resources

It’s powerful and generic but uses very limited resources, this means that we can use very low cost MCU’s. In Bitfire and Atmosfire we are using ATMega48, this device has 4k of flash and 512bytes of RAM. The boot monitor and interface allocates 2kB and leaves 2kB for application covering most needs in the housekeeping task.

The configuration utility is written in Python interpreting language. With our written consent you are free to use the code base for both the configuration utility and firmware for your own housekeeping needs.

5 Basic Functionality

The protocol is based on plain ASCII with no echo and CR terminated, there are number of commands that are specific to the boot monitor, some of them are only available in freeze mode (application stopped) such as flash write, format etc. Others are available even if the application is running, such as status, version etc. Interface calls are first parsed in the boot monitor and then sent to the application for further parsing if no command match is found. When the MCU is started there are three checks made before application code can execute.

1. First a EEPROM boot code is checked, if non-zero it will lock in boot monitor mode with response code 0x01. The boot code is set to 0xff when you freeze the application, this is done before you prepare for uploading new firmware. Use ‘unleash’ command to set this code to 0x00.

2. Next step is to check the application code checksum against the stored value in EEPROM. If these do not match it will also lock in boot monitor, now with response code 0x02.

3. If a particular hardware condition is met for the given board it will force into boot monitor mode. This can be useful if application firmware has erroneous code that kill the interface engine somehow, response code is 0x03.

If none of the above mentioned conditions are fulfilled the application code will start. The interface however is still run trough the boot monitor and messages are delivered trough a shared struct with semaphores between the two applications.

All communications are based on 38400bps 8N1. The magic packet sent triggering the UART lock and enabling the command parser is case sensitive, it should be preceded by a 0x1A char to sync the receive buffer.

Magic Packet: “GreetingZ_MrMega,ReveaL_YourselF%760913%#”

6 Boot Monitor global commands

These are commands that leave response even when the application is running. These are mainly identification and traffic control commands. See I2C bridge for response variation on I2C bridge equipped units. On UART interface message buffer synchronization is done by sending 0x1A (Ctrl+Z), this resets the message pointer and makes sure interface is in sync. It is recommended to send this sync at the beginning of each command string.

Command: "freeze\r"
Response: None
Desc: Freezes application and resets the MCU, this will reset peripherals and disable the application code. Command clears the checksum value in EEPROM and sets the EEPROM boot code to 0xff.

Command: "status\r"
Response: Status object
Desc: This command returns a status object with the format:
[A|B]-[NN]-[VVVVVVVV]-[AA]-[FWID]-[BC]
Example: "A-01-0100001A-02-2233-00"
[A|B] – Exec mode (A)pplication or (B)oot
[NN] – Equipment ID
[VVVVVVVV] – Product ID
[AA] – Adress
[FWID] – Application CRC
[BC] – Boot response code

Command: "version\r"
Response: Version string
Desc: Returns version string of the bootloader.

Command: "crc HHHH\r"
Response: "OK\r\n"
Desc: Writes checksum to EEPROM register, this checksum is verified against application sector contents during system startup.

Command: "wpid HHHHHHHH\r"
Response: "OK\r\n" or "Err\r\n"
Desc: Writes product ID, this command is prohibited when the MSB byte differs from 0xFF, this secures that product ID is only writable one time in production phase.

Command: “read HHN”
Response: Read object or “Err\r\n”:
Desc: This commands is used to read the application flash contents, read object:
r[PP][N][DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD][CC]
Example: “r01001020304050607080910111213141516FF”
PP – Flash page number
N – Flash row number
DD – Data
CC – 8-bit checksum

Command: “quit\r”
Response: None
Desc: This command is *only available if the interface supported is UART*. This closes the tunnel and tri-states the TXD pin releasing the RS232 port.

Command: “bind HHHHHHHH AA”
Response: “OK\r\n” or None
Desc: This command is *only available if the interface supported is I2C*. This binds a unit with a certain product ID to a given network address. I2C bridge always reside on address 0x02.

7 Boot Monitor flash commands

The flash commands are only available when the node is not executing application code. When uploading new firmware it is not possible to execute application code as it is wiped during this phase. To force the node into boot use the ‘freeze’ command.

Command: “format\r”
Response: “OK\r\n”
Desc: This wipes the whole application sector preparing the device for firmware upload.

Command: “lHHNDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDCC”
Response: “OK\r\n” or “Err\r\n”
Desc: Loads a row in a page of flash memory, when a complete page has been loaded ‘flash’ command is used to program that information into the device.

HH – Flash page number
N – Flash row number
DD – Data
CC – Checksum

Command: “flash\r”
Response: “OK\r\n” or “Err\r\n”
Desc: This command initiates the programming sequence and stores the data loaded with the ‘l’ command. It keeps track of the page used last used for loading data, this is the page that will be targeted for the ‘flash’ command.

Command: “unleash\r”
Response: None
Desc: Resets the boot code in EEPROM enabling the device to start application code and restarts the MCU.

8 I2C bridge commands

These commands are implemented in the application parser but as the bridge is a standard component that will be used for all firmwares that target boards equipped with I2C bridge (e.g. expansion capable) therefore these are documented in this manual as well.

The bridge component is quite straight forward. A few commands are added to handle the flow control and destination settings and the packets are forwarded as is with addition of an origin address code. The added commands are as follows.

Command: “#open HH\r”
Response: “OK\r\n” or “Err\r\n”
Desc: This command opens the I2C tunnel with outgoing destination address HH, address 0x00 is a public call address and all units will listen to that address. While the tunnel is open all commands are ignored if they do not start with a ‘#’.

Command: “#close\r”
Response: “OK\r\n” or “Err\r\n”
Desc: Closes the I2C communication tunnel and return control to the command parser.

8.1 Behaviour while I2C tunnel is open

Data command packets are forwarded from UART to I2C and from I2C to UART. All packets coming back trough the tunnel are translated and looks as if they where sent to the local device with one difference. Origin address is added so for example a status packet could look like this: “05:A-02-02000010-05-56FB-00”, 05 at the beginning tells witch network address actually sent the packet.

Issuing a new “#open HH” command while the tunnel is open will change the destination address for outgoing packets.

Using address 0x00 is a public call and all members despite their address setting will respond. This is especially useful when doing discovery and auto configuration.

9 The MCF Utility

The MCF utility is more or less self documented, place the utility with its supporting files in a bin catalogue available in you path and call it from a command prompt. For now it’s a command line utility, it might be translated into tool with a GUI later on but for now we keep our options open. This section is brief guide on how the basic usage works, this documentation will not cover all details because the utility itself is a living document and will be enhanced continuously and therefore quickly outdate this documentation.

9.1 Usage

You can send any number of commands to a specific target with one call. The basic usage is:

```
MCF [switches/options] [<target> <command0> … [commandN]]
```

Switches and options are always placed first and always begin with a '-' char.

Default the utility will scan the first 100 COM-ports and look for a MagicConfig boot monitor on all available ports, it will stop its search as soon as it finds a positive response. This means it will send MagicPackets on all free ports until it finds the correct one. If you do not want this or have several targets you can use '-c' option to specify the com-port used.

Units on the bridge that are unbinded to a network address need to be binded before firmware upload can take place please use the '–bind' option to do this. Binded addresses are stored in EEPROM and are only valid for native I2C hooked devices.

'-list' switch gives you an overview of what devices can be found locally and connected to the bridge if available, and in what status each device are in.

Sending basic commands uses a lookup scheme to find the correct target. Basic usage to send commands are:

1. Using the equipment ID in name format or parts of it, this example sends reset to the first found Atmosfire module on the bridge:

```
C:\>MCF atmosfire reset
C:\>MCF atmo reset
```

2. Using the product ID number

```
C:\>MCF 0x0200001A reset
```

3. Using the network address:

```
C:\>MCF 0x05 reset
```

Sending several commands are also permitted this example changes memory mapping and resets the Atmosfire:

```
C:\>MCF atmos cfg(boot=flash) reset
```

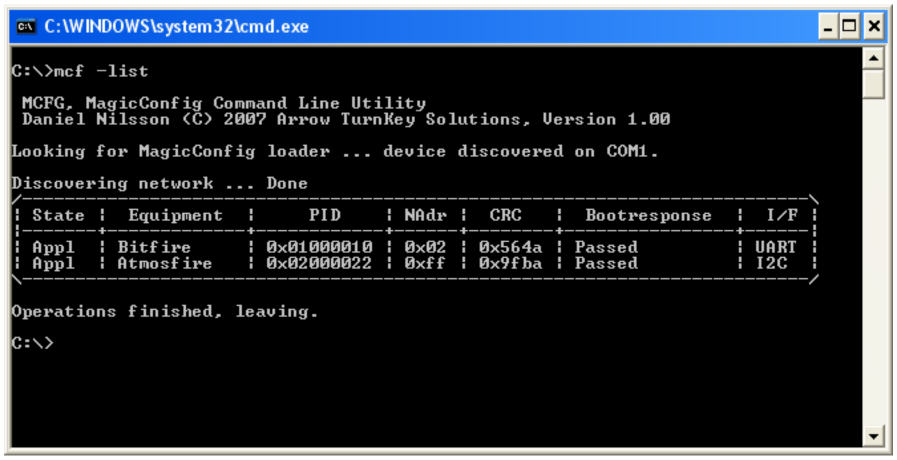
For more information on application specific commands use the "?:<target>" to discover your options:

```
C:\>MCF -?:atmosfire
C:\>MCF -?:bitfire
```

Use the ‘-l:<file>’ and ‘-d’ options to upload new firmware. The latest firmware can be downloaded from our support page.

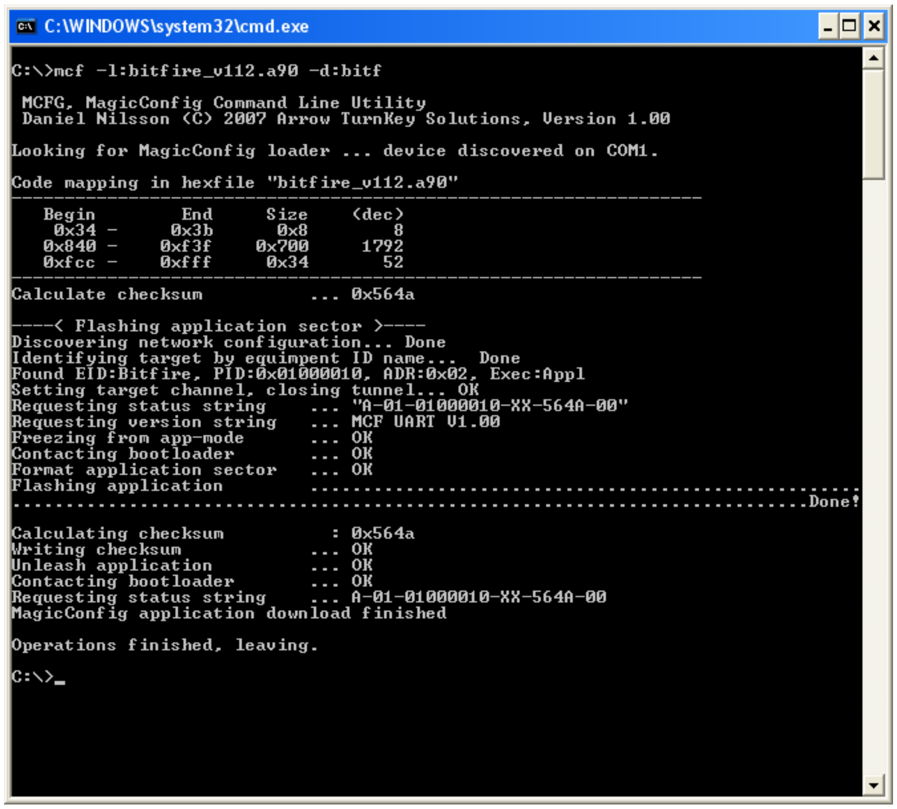
9.2 Screenshots

This is a examples on how it can look like.



C:\WINDOWS\system32\cmd.exe

```
C:\>mcf -list

 MCFG, MagicConfig Command Line Utility
 Daniel Nilsson (C) 2007 Arrow TurnKey Solutions, Version 1.00

Looking for MagicConfig loader ... device discovered on COM1.

Discovering network ... Done
/--------------------------------------------------------------------------\
¦ State ¦  Equipment  ¦     PID    ¦ NAdr ¦  CRC   ¦  Bootresponse   ¦ I/F ¦
¦-------+-------------+------------+------+--------+-----------------+-----¦
¦ Appl  ¦ Bitfire     ¦ 0x01000010 ¦ 0x02 ¦ 0x564a ¦ Passed          ¦ UART¦
¦ Appl  ¦ Atmosfire   ¦ 0x02000022 ¦ 0xff ¦ 0x9fba ¦ Passed          ¦ I2C ¦
\--------------------------------------------------------------------------/

Operations finished, leaving.

C:\>
```



C:\WINDOWS\system32\cmd.exe

```
C:\>mcf -l:bitfire_v112.a90 -d:bitf

 MCFG, MagicConfig Command Line Utility
 Daniel Nilsson (C) 2007 Arrow TurnKey Solutions, Version 1.00

Looking for MagicConfig loader ... device discovered on COM1.

Code mapping in hexfile "bitfire_v112.a90"
-----------------------------------------------------------------
   Begin        End      Size     (dec)
    0x34 -      0x3b      0x8         8
   0x840 -     0xf3f     0x700      1792
   0xfcc -     0xfff      0x34        52
-----------------------------------------------------------------
Calculate checksum           ... 0x564a

----< Flashing application sector >----
Discovering network configuration... Done
Identifying target by equimpent ID name...   Done
Found EID:Bitfire, PID:0x01000010, ADR:0x02, Exec:Appl
Setting target channel, closing tunnel... OK
Requesting status string     ... "A-01-01000010-XX-564A-00"
Requesting version string    ... MCF UART V1.00
Freezing from app-mode       ... OK
Contacting bootloader        ... OK
Format application sector    ... OK
Flashing application         ...................................................
...........................................................................Done!

Calculating checksum           : 0x564a
Writing checksum             ... OK
Unleash application          ... OK
Contacting bootloader        ... OK
Requesting status string     ... A-01-01000010-XX-564A-00
MagicConfig application download finished

Operations finished, leaving.

C:\>_
```

This page is intentionally left blank.

Notes: