Freescale Semiconductor
Application Note

Document Number: AN4641
Rev. 0, 12/2012

Enabling Android devices to use Freescale RF4CE BlackBox for Final Applications

Angel Corona
Abisai Negrete
Guadalajara applications group

Contents

1 Introduction

Higher demand of smart devices has integrated them into our everyday lives. Given the proliferation of these smart devices, technologies such as RF4CE may be included to provide consumers a richer experience while maintaining low power consumption and therefore battery life.

The i.MX family of devices offers a powerful yet low cost solution for multimedia devices supporting different Operating Systems such as Linux or Android. These devices could be easily interfaced with a dedicated MCU to manage other tasks like an RF4CE network in this case.

This application note demonstrates how to develop an RF4CE application using a Freescale 802.15.4 MC1323x device and the i.MX53 SABRE Tablet reference design running Android OS. Users must be familiar with the ZigBee RF4CE Specification available from www.zigbee.org and be familiar with at least the basic operation of the Freescale ZigBee/802.15.4 devices. See the appropriate data sheet or reference manual as needed for more information available at www.freescale.com/zigbee.

NOTE

The actual implementation of this specific example described in Section 5, "Example Application Overview," was developed using the Freescale i.mx53 SABRE reference design and MC13233C device, although any other similar Freescale device could be used.

2 BeeStack Consumer (RF4CE) BlackBox

The BeeStack Consumer BlackBox is an embedded application built on the BeeStack Consumer Network platform. The BeeStack Consumer BlackBox offers access to all the BeeStack Consumer Control Network features over a UART or an I2C interface. This allows BeeStack Consumer Network connectivity to be added to any system with limited modifications, because only a serial port is needed.

The BeeStack Consumer Network is a software networking layer that sits on top of the IEEE 802.15.4 MAC and PHY layers. It is designed for Wireless Personal Area Networks (WPANs) and conveys information over short distances among the participants in the network. It enables small, power efficient, inexpensive solutions to be implemented for a wide range of applications. Some key characteristics of a BeeStack Consumer network are:

- An over the air data rate of 250 kbit/s in the 2.4 GHz band
- Three independent communication channels in the 2.4 GHz band
- Two network node types: controller node and target node
- Channel agility mechanism
- Provides robustness and ease of use
- Includes essential functionality to build and support a CE network

NOTE

For further details about the RF4CE capabilities and features please refer to the RF4CE specification document from ZigBee Alliance or read the Freescale BeeStack Consumer Reference Manual.

This application note covers the I2C interface with which the i.MX53 device communicates with the MC1323x device. For further details about I2C packet structure please refer to the BlackBox reference manual.

2.1 HW connections

The i.MX53 SABRE Tablet reference design has the proper connections to communicate via I2C with the on-board MC1323x device, thus allowing the i.mx MPU to monitor and control the RF4CE network via I2C commands. This does not limit the usage of any different MCU/MPU because any other device could be used as long as it has an I2C, SPI, or UART module available.

NOTE

The design uses the I2C2 port from the i.MX53 processor. The bus is shared by SGTL5000, MAG3110, touch keys, and MC1323X devices.

Figure 1 is a block diagram of the necessary connections between the host MPU/MCU and the MC1323x when using the BlackBox I2C interface.

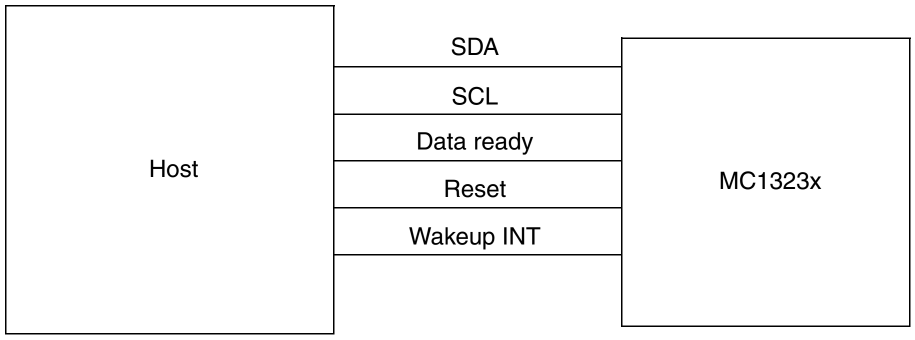


Figure 1. Basic connections between the host and the I2C slave

- **SDA and SCL —** I2C Data and clock signals as required by I2C communication. These signals are needed to send and receive commands and data to and from the MC1323x slave device.
- **Data ready —** GPIO used to indicate the host application that data is ready to be read from slave (MC1323x) such as events, responses or request confirms. This connection might be optional when using UART communication, but it becomes necessary when using I2C because the host MCU/MPU should know when the slave has any data to transmit so it can provide the bus clock to it.
- **Reset (optional) —** GPIO connected to RESET signal of MC1323x in case it is useful/necessary to reset the device. It is recommendable to have this line connected, otherwise there would not be any other way to reset the device unless you power off the whole system or the user implements a way to execute a software reset in the BlackBox application.
- **Wakeup INT (Optional) —** The BlackBox application does not need this signal, but it is necessary for sleeping devices. If low power is desired, a GPIO to wake up the MC1323x may become necessary.

Figure 2 is an abstract of the tablet's main board schematic illustrating the actual pins and their names connected to the on-board MC1323x device. For further details about connections in the SABRE Tablet design, please refer to the main board schematics.

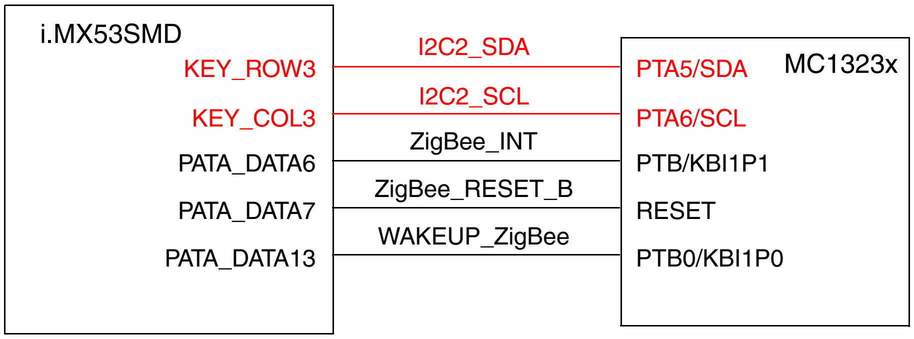


Figure 2. Actual connections between the i.MX53 and the on-board MC1323x device in SABRE Tablet reference design

2.2 Low level Driver

When developing an application on top of Android OS (10.3.2 BSP), users may need to develop a low-level driver to access the GPIOs involved in the BlackBox communication such as UART, SPI or I2C. This driver's development is completely up to the user and could be done in several ways.

NOTE

The Book "Linux Device Drivers" by Alessandro Rubini and Jonathan Corbet is a good guide when developing a new driver. You can find this document on Freescale's website.

In this specific design, some changes were done to the Android BSP (10.3.2) in order to define some pins as GPIOs and use them for the different signals needed in Blackbox communication (SeeSection 2.1, "HW connections"). Such changes are listed below:

1. Define the ports in `Myandroid/arch/arm/mach-mx5/mx53_smd.c`

```
#define MX53_SMD_ZIGBEE_INT(1*32 + 6)/* GPIO2_6 */
#define MX53_SMD_ZIGBEE_RESET_B(1*32 + 7)/* GPIO2_7 */
#define MX53_SMD_WAKEUP_ZIGBEE(1*32 + 13)/* GPIO2_13 */
```

2. Such definitions should be included in `iomux_v3_cfg_t mx53_smd_pads structure.`

```
static iomux_v3_cfg_t mx53_smd_pads[] = {

/* ZigBee_INT*/
MX53_PAD_PATA_DATA6__GPIO2_6,
/* ZigBee_RESET_B */
MX53_PAD_PATA_DATA7__GPIO2_7,
        /* WAKEUP_ZigBee */
MX53_PAD_PATA_DATA13__GPIO2_13,
/* I2C2 */
MX53_PAD_KEY_COL3__I2C2_SCL,
MX53_PAD_KEY_ROW3__I2C2_SDA,
}
```

3. Add initialization information to I2C port of i.MX.

```
static struct i2c_board_info mxc_i2c1_board_info[] __initdata = {

…
…
        {
        .type = "MC1323",
        .addr = 0x76,      /*Shifted MC1323´s address*/
        },
}
```

4. Create a new directory called MC1323 under myandroid/kernel_imx/drivers
5. Copy MC1323 files (driver source code and Makefile attached)
6. Modify myandroid/kernel_imx/drivers/Makefile

```
From
         obj-y i2c/   /media
to
         obj-y i2c/   /media  MC1323/
```

7. Compile Kernel driver

NOTE

The driver simply receives/send I2C commands and control the GPIOs described in Chapter 2.1 – HW connections. The BlackBox API implementation is done in the Android application.

Please refer to Appendix B, “Compiling the kernel with the MC1323x drivers,” for further details about how to include the attached driver and Makefile to the kernel.

2.3 JNI and driver

The Java Native Interface (JNI) is a programming framework that enables Java code to be called by the C-language GPIO/UART/SPI/I2C drivers. In this example, the JNI is used in order to call java functions form the GPIO drivers described in Section 2.2, “Low level Driver.”

NOTE

Please refer to the attached zip file containing the driver source file for further details about its implementation.

3 Controlling an RF4CE node using BlackBox

3.1 Starting the Node

To start using the MC1323x as an RF4CE node, you first must follow some necessary steps such as starting the network layer, set the node capabilities, etc. Table 1 shows the proper commands in order that should be executed before performing any other RF4CE command.

Table 1. Starting procedure for a RF4CE BlackBox Node

Commands in order	I2C data
ZTC-WriteExtAddr.Request	02 A3 DB 08 00 AA AA AD FF CF CC EE AA 65
RF4CE_NWK_Reset.Request	02 D0 00 01 00 01 D0
RF4CE_NWK_SetNodeCapabilities.Request	02 D4 04 01 00 08 D9
RF4CE_NLME_Start.Request	02 D0 01 00 00 D1
ZRCProfile_PushButtonPairOrig.Request	02 D6 00 10 00 FF FF FF FF FF 00 01 01 01 01 01 01 05 00 E8 03 D7

NOTE

Please refer to the BeeStack Consumer Application User’s guide – Chapter 4.3 for further details about the starting procedure (BSCONNAUG.pdf), available from the Freescale website.

3.1.1 ZTC-WriteExtAddr.Request

Before starting the network on the RF4CE Controller device, you should write the MAC address to the device unless it is predefined in the Firmware. For instance:

```
ZTC-WriteExtAddr.Request  02 A3 DB 08 00 AA AA AD FF CF CC EE AA 65

       Sync    [1 byte ]  = 02
       OpGroup [1 byte ]  = A3
       OpCode  [1 byte ]  = DB
       Length  [2 bytes]  = 08 00
       Address [8 bytes]  = AA EE CC CF FF AD AA AA
       CRC     [1 byte ]  = 65
```

3.1.2 RF4CE_NWK_Reset.Request

The network layer should be reset and all the entries from the pairing table will be removed unless the SetDefaultNIB value is set to false. When SetDefaultNIB is set to false, the node will retrieve all paired devices’ data from NVM.

```
RF4CE_NLME_Reset.Request 02 D0 00 01 00 01 D0

       StartOfFrame  [1 byte ] = 02
```

```
        Header        [2 bytes] = D0 00
        PayloadLength [2 bytes] = 00 01
        SetDefaultNIB [1 byte ] = 01 (true)
        Checksum      [1 byte ] = D0
```

3.1.3 RF4CE_NWK_SetNodeCapabilities.Request

The node type (target/controller) should be specified along other settings such as security enabled or power source. The nwkcNodeCapabilities bit fields are described in the RF4CE specification as Table 2 shows.

Table 2. nwkcNodeCapabilities bit fields

Bits				
0	**1**	**2**	**3**	**4–7**
Node Type	Power Source	Security capable	Channel norm. capable	Reserved

```
RF4CE_NWK_SetNodeCapabilities.Request 02 D4 04 01 00 08 D9

        StartOfFrame        [1 byte ] = 02
        Header              [2 bytes] = D4 04
        PayloadLength       [2 bytes] = 00 01
        nwkcNodeCapabilities [1 byte ] = 08 (Controller without security)
        Checksum            [1 byte ] = D9
```

3.1.4 RF4CE_NLME_Start.Request

Start the network layer and thus the node. After this point, the node will be able to send any other RF4CE command or perform any action such as pair with more devices.

```
RF4CE_NLME_Start.Request 02 D0 01 00 00 D1

        StartOfFrame  [1 byte ] = 02
        Header        [2 bytes] = D0 01
        PayloadLength [2 bytes] = 00 00
        Checksum      [1 byte ] = D1
```

3.2 Pairing with devices

Once started, the controller device is now ready to start pairing with other devices. Notice that the pairing process evaluates whether a target device could be paired or not. Whether it can will depend on each node’s capabilities and supported profiles. Refer to the RF4CE specification for further details.

3.2.1 ZRCProfile_PushButtonPairOrig.Request

Start a pushbutton pairing process in the controller. The target should start it within 30 seconds to be successful.

```
ZRCProfile_PushButtonPairOrig.Request 02 D6 00 10 00 FF FF FF FF FF 00 01 01 01 01 01 01
                                      05 00 E8 03 D6

StartOfFrame                             [1 byte ] = 02
Header                                   [2 bytes] = D6 00
PayloadLength                            [2 bytes] = 00 10
RecipPanId                               [2 bytes] = FF FF
RecipShortAddress                        [2 bytes] = FF FF
RecipDeviceType                          [1 byte ] = FF
OrigAppCapabilities_UserStringSpecified  [1 byte ] = 00 (UserStringNotIncludedInFrame)
OrigAppCapabilities_NoOfSupportedDeviceTypes [1 byte ] = 01 (OneDeviceTypeInDeviceTypeList)
OrigAppCapabilities_NoOfSupportedProfiles [1 byte ] = 01 (OneSupportedProfilesInProfileIdList)
OrigDevTypeList                          [1 byte ] = 01
               OrigDevTypeList[0] = 01
OrigProfileIdList                        [1 byte ] = 01
               OrigProfileIdList[0] = 01
DiscProfileIdListSize                    [1 byte ] = 01
DiscProfileIdList                        [1 byte ] = 01
               DiscProfileIdList[0] = 01
KeyExTransferCount                       [1 byte ] = 05
RequestAppAcceptToPair                   [1 byte ] = 00 (FALSE)
TimeToWaitAppAcceptToPair                [2 bytes] = 03 E8

Checksum                                 [1 byte ] = D7
```

3.3 Saving and retrieving pairing information

Although the RF4CE nodes already save some important data such as network address in the pairing table, it is only the minimum required to establish a communication with such node again. It is up to the application to save other data such as device type or user string if the application requires doing so.

This specific example saves all paired nodes information such as device type, vendor ID or user string, in order to retrieve it in case the node is started with SetDefaultNIB set to false. This will allow the application to not only recover the communication with all the already-paired devices, but also some specific information such as device type or Vendor String, which are not normally stored in the RF4CE pairing table.

An easy way to store application specific data is to use `OutputStreamWriter` in java. Something similar to:

```
public void StoreData(){
    try {
            FileOutputStream fOut = openFileOutput("DevicesDataFile",MODE_WORLD_READABLE);
            OutputStreamWriter osw = new OutputStreamWriter(fOut);
            osw.write(PairedDeviceType);
            // …
            // append any other data you would like to store
            // …
            // …

             osw.close();
             fOut.close();

            } catch (IOException e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
            }
 }
```

To retrieve the store data simply open the saved file ad use `InputStreamReader` instead.

3.4 Sending ZRC commands

Once paired, it may become necessary to send ZRC commands to other devices. For this, a ZRCProfile_Command.Request will need to be used.

3.4.1 ZRCProfile_Command.Request

Send a command to the target device. If successful, the connection has been properly set and the application will work as expected. If there is any error, please verify the devices paired table index and the rest of the parameters are correct.

```
ZRCProfile_Command.Request 02 DD 00 0C 00 00 01 01 05 00 84 05 00 00 15 00 00 40

       StartOfFrame  [1 byte ] = 02
       Header        [2 bytes] = DD 00
       PayloadLength [2 bytes] = 00 0C
       PairingRef    [1 byte ] = 00 (first device in pairing table)
       CommandCode   [1 byte ] = 01 (gZRC_CmdCode_UserCtrlPressed_c)
       Command       [1 byte ] = 01
       VendorId      [2 bytes] = 00 05
       TxOptions     [1 byte ] = 84  (vendor specific, unicast, ack without security)
       DataLength    [1 byte ] = 05
       Data          [5 bytes] = 00 00 15 00 00
                      Data[0] = 00
                      Data[1] = 00
                      Data[2] = 15
                      Data[3] = 00
                      Data[4] = 00
       Checksum      [1 byte ] = 40
```

NOTE

This application is intended to demonstrate how to send commands to the RF4CE paired devices. Custom payloads and commands are application specific and its implementation is up to the user.

3.5 Handling responses

Every BlackBox command request will generate a response from the slave. This response contains the status of the operation or important data such as the paired device's information. It is up to the application to read such responses and handle them properly.

Because the I2C communication is asynchronous, the host will need a way to know there is some data ready to be read from the MC1323x slave. This is done by monitoring the state of Data Ready PIN (see Section 2.1, "HW connections").

Figure 3 shows a basic flow chart illustration how the responses are handled. Notice that in this specific example there is not any timeout or mechanism preventing the activity block when waiting for a response. Using threading or implementing a timeout could be an improvement, but its implementation is application specific.

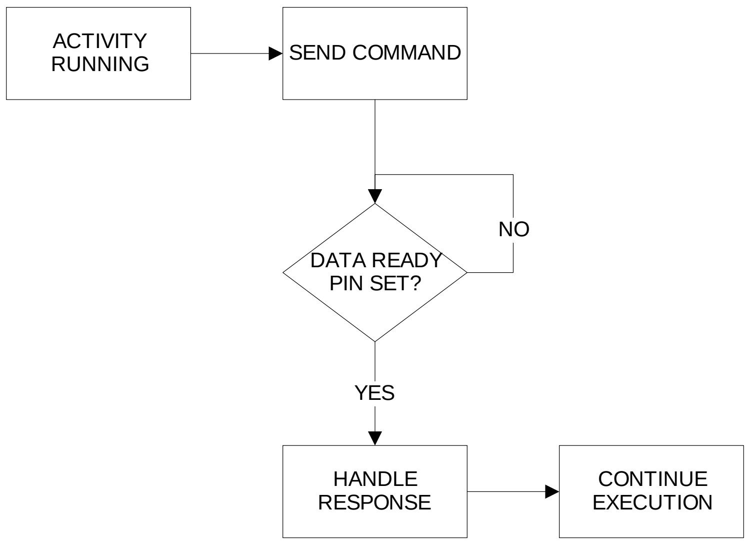


Figure 3. BlackBox response handling flow chart

4 Graphic User Interface (GUI)

Users may take advantage of using a powerful MPU such as the i.MX53 **and develop a graphic user interface** to manage the RF4CE node, thus making it user-friendly. Having an android application to control and monitor an RF4CE network will enrich the user's experience and make the solution usable in several devices by simply installing the application.

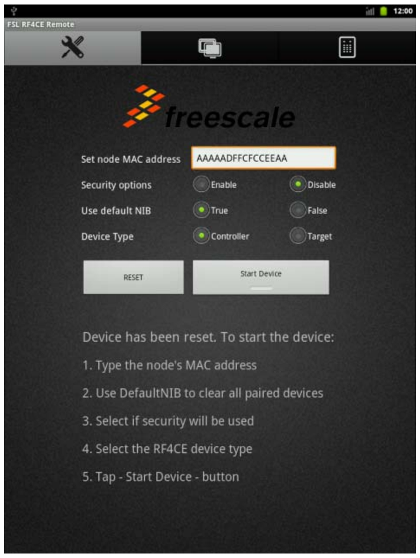
Please refer to Android developer's forum for further details about developing your own GUI. There is a lot of documentation and examples to get familiar with the app development (try http://developer.android.com).

5 Example Application Overview

This application example was developed using Android 2.3 SDK (10.3.2 BSP), although it should work with newer versions. The graphic interface has three different tabs in which users may configure the device, pair/unpair remote devices, and send RF4CE commands to them.

5.1 Tab 1 — Device configuration

In the settings tab, users can initialize or reset the device. They are able to choose between enabling and disabling security in RF4CE messages, clear or keep the already-paired devices (pairing table) and choose the type of device you want the device to be, for example an RF4CE controller.



Set node MAC address: AAAAADFFCFCCEEAA

Security options: Enable [✓ Disable]

Use default NIB: [✓ True] False

Device Type: [✓ Controller] Target

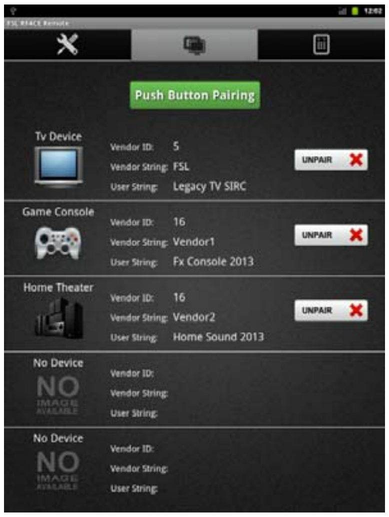
Device has been reset. To start the device:

1. Type the node's MAC address
2. Use DefaultNIB to clear all paired devices
3. Select if security will be used
4. Select the RF4CE device type
5. Tap - Start Device - button

Figure 4. Example Application Tab 1 – Device Configuration

5.2 Tab 2 — List of paired devices

Once started, you are able to pair with devices with a simple push of a button. All that is needed to start the pushbutton pairing process in both the controller (tablet) and the target devices you want to be paired with such as TV, STB, DVD, etc. RF4CE already defines the type of possible devices a node can operate as.



Tv Device
Vendor ID: 5
Vendor String: FSL
User String: Legacy TV SIRC

Game Console
Vendor ID: 16
Vendor String: Vendor1
User String: Fx Console 2013

Home Theater
Vendor ID: 16
Vendor String: Vendor2
User String: Home Sound 2013

No Device
Vendor ID:
Vendor String:
User String:

No Device
Vendor ID:
Vendor String:
User String:

Figure 5. Example Application Tab 2 – List of paired devices

5.3 Tab 3 — Remote control

Once the devices are paired, the Tab 3 allows users to start sending commands to each of the paired devices. Users are able to choose between all the devices the controller was paired with. Back in Tab 2, users can unpair a device they no longer want to communicate with.

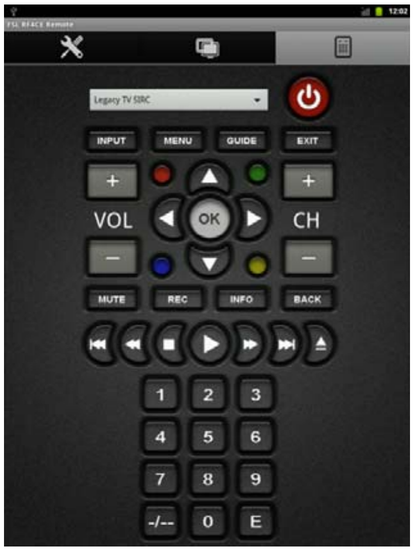


Figure 6. Example Application Tab 3 – Remote control

With this application example, users will have a very simple, yet powerful universal remote control for RF4CE devices in a smart device such as a tablet, a smart phone, or any device with a RF4CE-capable device.

NOTE

This application is intended to demonstrate how to send commands to the RF4CE paired devices. Custom payloads and commands are application specific and its implementation is up to the user.

6 Additional resources

To develop and debug any application for the MC1323x device, you should have the proper Codewarrior IDE version according to the codebase you want to use. To check what version needs to be used, please refer to the RF4CE codebase release notes included in Freescale installation folder (i.e. C:\Program Files\Freescale) under System Requirements.

NOTE

Please refer to the BeeKit User’s Guide for further details about generating, exporting and loading BeeKit applications.

Please note that both RF4CE and ZigBee stack are bigger than 32KB, thus a special edition of Codewarrior will not be enough. If you want to use either stack, you will need to have the proper license that allow you to compile and load more than 32KB of code. The evaluation edition of Codewarrior does not limit you in code size, but it is only available for 30 days.

Appendix A Generating the RF4CE BlackBox for MC1323x

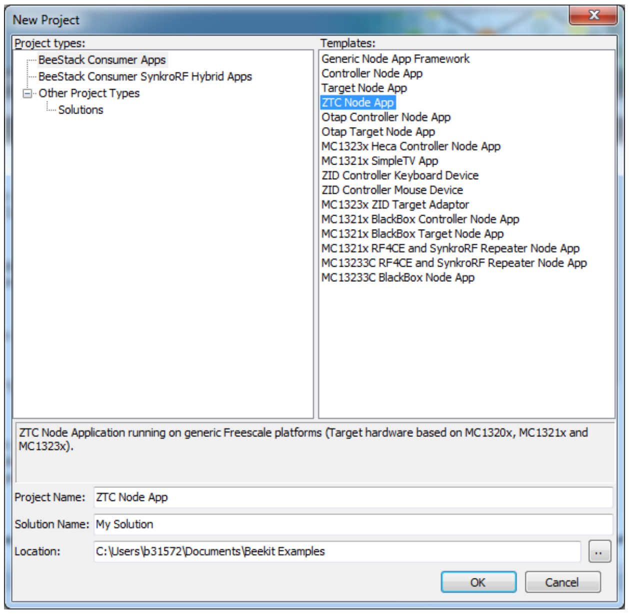
The BlackBox application could be easily generated using BeeKit Wireless Connectivity tool and selecting the ZTC Application. However, some settings need to be changed in order to use such application as a BlackBox.

NOTE

For further details about BeeKit and how to generate solutions based in any of Freescale codebases, please refer to the BeeKit User's guide included in the documentation folder.

By default, the ZTC application exposes MAC and RF4CE layers functionality through a serial communication interface to a host system. To create an RF4CE Black Box application, the ZTC application needs to be configured to expose only the RF4CE layer functionality using the following steps:

1. Select the "ZTC Node App" project from BeeStack Consumer Codebase.


New Project
Project types:
BeeStack Consumer Apps
BeeStack Consumer SynkroRF Hybrid Apps
Other Project Types
Solutions
Templates:
Generic Node App Framework
Controller Node App
Target Node App
ZTC Node App
Otap Controller Node App
Otap Target Node App
MC1323x Heca Controller Node App
MC1321x SimpleTV App
ZID Controller Keyboard Device
ZID Controller Mouse Device
MC1323x ZID Target Adaptor
MC1321x BlackBox Controller Node App
MC1321x BlackBox Target Node App
MC1321x RF4CE and SynkroRF Repeater Node App
MC13233C RF4CE and SynkroRF Repeater Node App
MC13233C BlackBox Node App
OK
Cancel

ZTC Node Application running on generic Freescale platforms (Target hardware based on MC1320x, MC1321x and MC1323x).

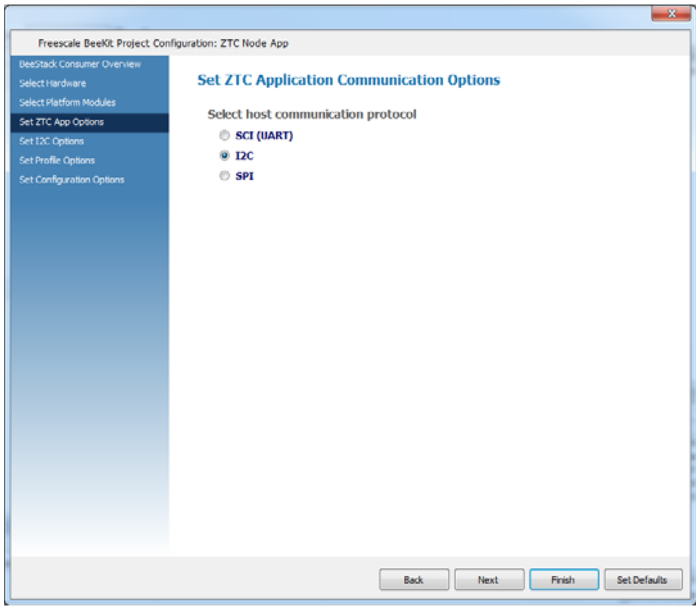
Project Name: ZTC Node App

Solution Name: My Solution

Location: C:\Users\b31572\Documents\Beekit Examples

Figure 7. Selecting RF4CE ZTC Node App

2. Select the host communication protocol. For this demo the I2C needs to be enabled. In the next step, leave the I2C slave address as default (0x76). Click next.



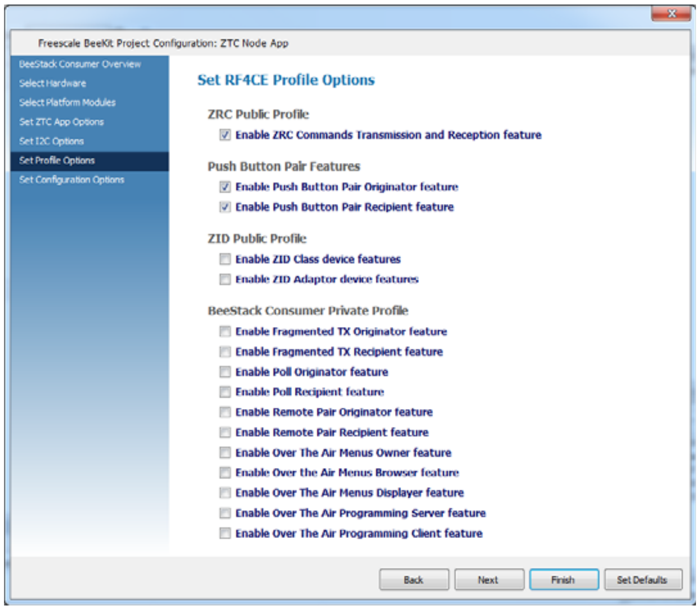
Set ZTC Application Communication Options

Select host communication protocol

- SCI (UART)
- [✓ I2C]
- SPI

Figure 8. Selecting the I2C BlackBox communication protocol

3. For each profile (ZRC Profile, BeeStack Consumer Private Profile) enable the desired features. Also, enable the Push button pair originator/recipient and the ZRC commands transmission and reception features. By default, these profile features are disabled, because the ZTC Node App is used to test (by default) the MAC and RF4CE primitives. Other features may be enabled if desired.



Set RF4CE Profile Options

ZRC Public Profile

- [✓] Enable ZRC Commands Transmission and Reception feature

Push Button Pair Features

- [✓] Enable Push Button Pair Originator feature
- [✓] Enable Push Button Pair Recipient feature

ZID Public Profile

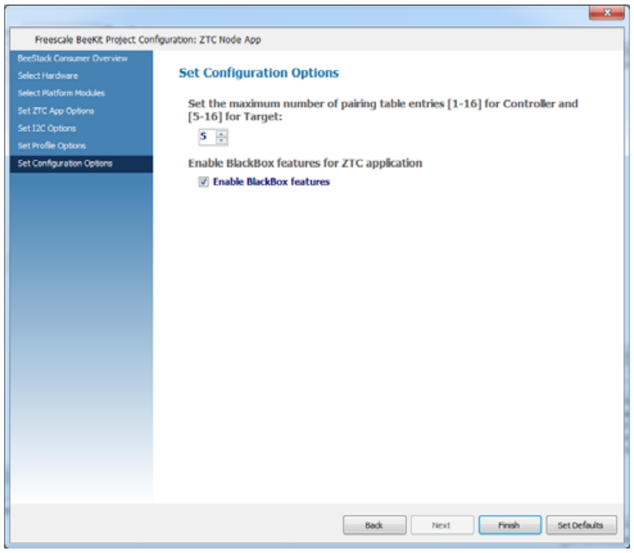
- [ ] Enable ZID Class device features
- [ ] Enable ZID Adaptor device features

BeeStack Consumer Private Profile

- [ ] Enable Fragmented TX Originator feature
- [ ] Enable Fragmented TX Recipient feature
- [ ] Enable Poll Originator feature
- [ ] Enable Poll Recipient feature
- [ ] Enable Remote Pair Originator feature
- [ ] Enable Remote Pair Recipient feature
- [ ] Enable Over The Air Menus Owner feature
- [ ] Enable Over the Air Menus Browser feature
- [ ] Enable Over The Air Menus Displayer feature
- [ ] Enable Over The Air Programming Server feature
- [ ] Enable Over The Air Programming Client feature

Figure 9. Enabling ZRC commands and profile specific features

4. Enable the BlackBox features for ZTC application by checking the box. Also, select the airing table entries for the RF4CE device.



Set Configuration Options

Set the maximum number of pairing table entries [1-16] for Controller and [5-16] for Target: 5

Enable BlackBox features for ZTC application

✓ Enable BlackBox features

Figure 10. Enabling BlackBox features

Once finished and validated, the solution is now ready to be exported to an IDE (Codewarrior for the appropriate MC1323x device) in order to compile it and load it to the device. The only chance that might be needed to use it for the Android application developed for this demo, would be the pin definitions for interfacing the i.MX53 and the MC1323x.

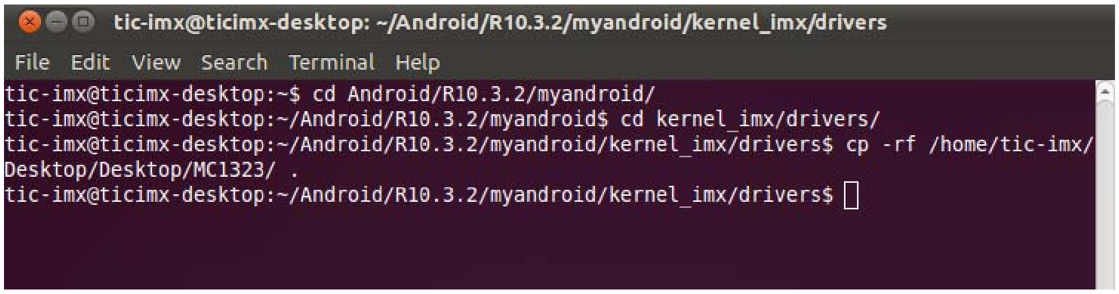
In this demo, only the Data Ready pin used (See Chapter 2.1 – Hardware connections) changed and it should be defined in the IIC_Interface.h file of the BlackBox firmware. The application example uses PTB1 pin for such purpose. This is done as follows:

```
#ifndef gIIC_TxDataAvailablePortDataReg_c
  #define gIIC_TxDataAvailablePortDataReg_c     PTBD
#endif
#ifndef gIIC_TxDataAvailablePortDDirReg_c
  #define gIIC_TxDataAvailablePortDDirReg_c     PTBDD
#endif
#ifndef gIIC_TxDataAvailablePinMask_c
  #define gIIC_TxDataAvailablePinMask_c         0x02
#endif
```

Appendix B Compiling the kernel with the MC1323x drivers

Included in the zip file attached to this Application Note are the driver source file (MC1323.c) and Make File. These files need to be added to the kernel image in order to use them in the Android application. This is achieved by following the next steps:

1. Copy the MC1323 file to the Android drivers folder. This folder is located in myandroid/kernel_imx/drivers. Please just notice that the android folder (myandroid in this case) might have a different name and could also bee in a different location (Android/R10.3.2 folder in this case).


tic-imx@ticimx-desktop: ~/Android/R10.3.2/myandroid/kernel_imx/drivers
File Edit View Search Terminal Help

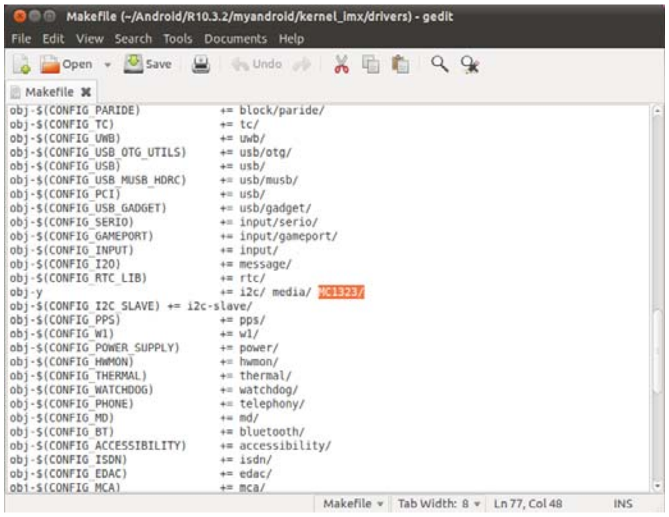
```
tic-imx@ticimx-desktop:~$ cd Android/R10.3.2/myandroid/
tic-imx@ticimx-desktop:~/Android/R10.3.2/myandroid$ cd kernel_imx/drivers/
tic-imx@ticimx-desktop:~/Android/R10.3.2/myandroid/kernel_imx/drivers$ cp -rf /home/tic-imx/
Desktop/Desktop/MC1323/ .
tic-imx@ticimx-desktop:~/Android/R10.3.2/myandroid/kernel_imx/drivers$
```

Figure 11. Copying driver source file to Android drivers folder

NOTE

The location where this files where unzipped might be different than the one shown in the figure. In this case, the files were located in /home/tic_imx/Desktop/Desktop/MC1323 folder.

2. Once copied, the Make File should be edited to include such folder in the objects to be compiled. To edit the file, simply type '*gedit Makefile*' in the terminal.


Makefile (~/Android/R10.3.2/myandroid/kernel_imx/drivers) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
Makefile
Makefile
Tab Width: 8
Ln 77, Col 48
INS

```
obj-$(CONFIG_PARIDE)            += block/paride/
obj-$(CONFIG_TC)                += tc/
obj-$(CONFIG_UWB)               += uwb/
obj-$(CONFIG_USB_OTG_UTILS)     += usb/otg/
obj-$(CONFIG_USB)               += usb/
obj-$(CONFIG_USB_MUSB_HDRC)     += usb/musb/
obj-$(CONFIG_PCI)               += usb/
obj-$(CONFIG_USB_GADGET)        += usb/gadget/
obj-$(CONFIG_SERIO)             += input/serio/
obj-$(CONFIG_GAMEPORT)          += input/gameport/
obj-$(CONFIG_INPUT)             += input/
obj-$(CONFIG_I2O)               += message/
obj-$(CONFIG_RTC_LIB)           += rtc/
obj-y                           += i2c/ media/ MC1323/
obj-$(CONFIG_I2C_SLAVE) += i2c-slave/
obj-$(CONFIG_PPS)               += pps/
obj-$(CONFIG_W1)                += w1/
obj-$(CONFIG_POWER_SUPPLY)      += power/
obj-$(CONFIG_HWMON)             += hwmon/
obj-$(CONFIG_THERMAL)           += thermal/
obj-$(CONFIG_WATCHDOG)          += watchdog/
obj-$(CONFIG_PHONE)             += telephony/
obj-$(CONFIG_MD)                += md/
obj-$(CONFIG_BT)                += bluetooth/
obj-$(CONFIG_ACCESSIBILITY)     += accessibility/
obj-$(CONFIG_ISDN)              += isdn/
obj-$(CONFIG_EDAC)              += edac/
obj-$(CONFIG_MCA)               += mca/
```

Figure 12. Adding MC1323 folder to Makefile

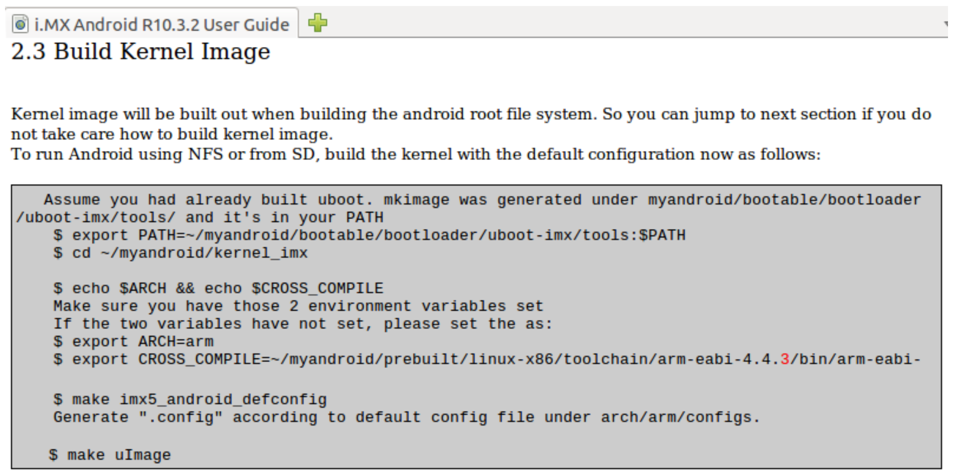
In this example, the MC1323 was added to the I2C folder, but it can be done in any obj –y entry.

3. Once done, and after selecting the parent of the current directory (type ‘*cd* ..’ in terminal), the kernel is ready to be compiled. For this, users may refer to the Android User’s guide available in Freescale website, where this process is described in detail.

NOTE

This demo uses the Android 10.3.2, and the corresponding User’s guide could be downloaded from SABRE Tablet documentation website under IMX5X_R10_32_ANDROID_DOCSBUNDLE.

Figure 13 shows the user’s guide chapter about how to build the kernel image, where the environment variables needed are shown.



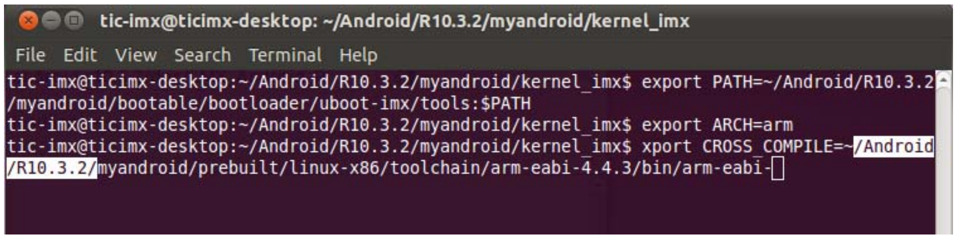
2.3 Build Kernel Image

Kernel image will be built out when building the android root file system. So you can jump to next section if you do not take care how to build kernel image.
To run Android using NFS or from SD, build the kernel with the default configuration now as follows:

```
   Assume you had already built uboot. mkimage was generated under myandroid/bootable/bootloader
/uboot-imx/tools/ and it's in your PATH
    $ export PATH=~/myandroid/bootable/bootloader/uboot-imx/tools:$PATH
    $ cd ~/myandroid/kernel_imx

    $ echo $ARCH && echo $CROSS_COMPILE
    Make sure you have those 2 environment variables set
    If the two variables have not set, please set the as:
    $ export ARCH=arm
    $ export CROSS_COMPILE=~/myandroid/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-

    $ make imx5_android_defconfig
    Generate ".config" according to default config file under arch/arm/configs.

   $ make uImage
```

Figure 13. Build Kernel Image instructions from User’s Guide

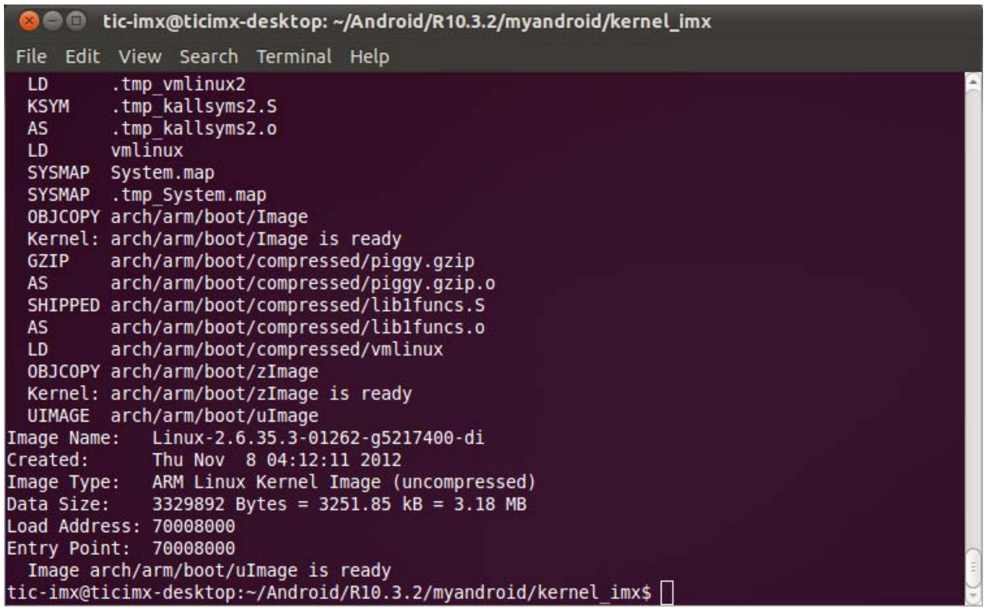
Remember to update the PATH wherever your ‘myandoird’ folder is located. In this specific case, it is located under /Android/R10.3.2/ folder, thus every PATH variable was updated as shown in next figure.


tic-imx@ticimx-desktop: ~/Android/R10.3.2/myandroid/kernel_imx
File Edit View Search Terminal Help

```
tic-imx@ticimx-desktop:~/Android/R10.3.2/myandroid/kernel_imx$ export PATH=~/Android/R10.3.2
/myandroid/bootable/bootloader/uboot-imx/tools:$PATH
tic-imx@ticimx-desktop:~/Android/R10.3.2/myandroid/kernel_imx$ export ARCH=arm
tic-imx@ticimx-desktop:~/Android/R10.3.2/myandroid/kernel_imx$ xport CROSS_COMPILE=~/Android
/R10.3.2/myandroid/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin/arm-eabi-
```

Figure 14. Environmental variables needed for compilation

4. Finally, simply use ‘make uImage’ and the new file including the recently-added drivers will be generated. Figure 15 illustrates how it should look like after finishing the process.


tic-imx@ticimx-desktop: ~/Android/R10.3.2/myandroid/kernel_imx
File Edit View Search Terminal Help

```
  LD      .tmp_vmlinux2
  KSYM    .tmp_kallsyms2.S
  AS      .tmp_kallsyms2.o
  LD      vmlinux
  SYSMAP  System.map
  SYSMAP  .tmp_System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  GZIP    arch/arm/boot/compressed/piggy.gzip
  AS      arch/arm/boot/compressed/piggy.gzip.o
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-2.6.35.3-01262-g5217400-di
Created:      Thu Nov  8 04:12:11 2012
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    3329892 Bytes = 3251.85 kB = 3.18 MB
Load Address: 70008000
Entry Point:  70008000
  Image arch/arm/boot/uImage is ready
tic-imx@ticimx-desktop:~/Android/R10.3.2/myandroid/kernel_imx$
```

Figure 15. Making uImage file

5. It might become necessary to change device node permissions in order to work properly. Therefore, after the uImage was compiled, users can simply use the following commands:

```
# cd myandroid/out/target/product/imx53_smd
# dd if=uramdisk.img of=ramdisk.img.gz skip=64 bs=1
# gunzip ramdisk.img.gz
# mkdir ramdisk; cd ramdisk
# cpio -i < ../ramdisk.img
# vim init.rc   (modify the init.rc, add the line "chmod 0666
/dev/MC1323x_dev")
# find . | cpio --create --format='newc' | gzip > ../ramdisk.img
# mkimage -A arm -O linux -T ramdisk -C none -a 0x70308000 -n "Android Root
Filesystem" -d ./ramdisk.img ./uramdisk.img
```

6. To copy and replace the new uImage, users may have several options. One is to use the manufacturing tool as described in the User’s guide - chapter 3.2.2 Download Images with MFG tool.

3.2.2 Download Images with MFG Tool

Take MX53 TABLET with eMMC boot as example:

Before using the MFG tool, please setup the board's to serial download mode:

- Copy the android images: u-boot.bin, uImage, uramdisk.img, system.img, recovery.img to Profiles/MX53 Linux Updata/OS Firmware/files/android.
- Run the MFGTool, plug in the usb cable and choose your profile

Notes: Find a new device and install the driver (the drivers\ directory)

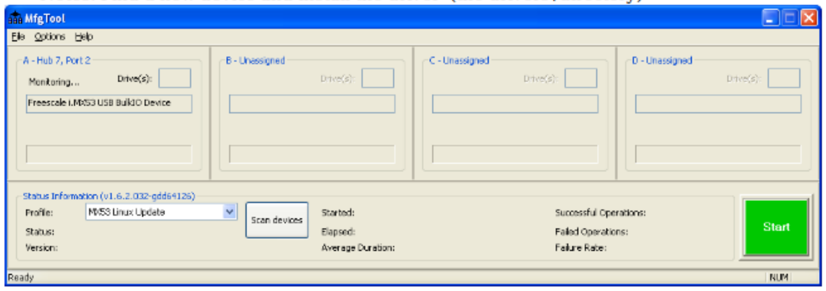


Figure 16. Manufacturing tool for downloading images

Another option is to use a dd utility as described in chapter 3.2.3 Download Images with dd utility.

3.2.3 Download Images with dd utility

The linux utility "dd" on Linux PC can be used to download the images into the MMC/SD card. Before downloading, make sure your MMC/SD card's partitions are created.
And all the partitions can be recognized by Linux PC. To download all the images into the card, please use the commands below:

```
    Download the uboot image:
    # sudo dd if=u-boot.bin of=/dev/sdx bs=1K skip=1 seek=1; sync
Or If you're using no padding uboot image:
    # sudo dd if=u-boot-no-padding.bin of=/dev/sdx bs=1K seek=1; sync
    Download the kernel image:
    # sudo dd if=uImage of=/dev/sdx bs=1M seek=1; sync
    Download the initramfs image:
    # sudo dd if=uramdisk.img of=/dev/sdx bs=1M seek=6; sync
    Download the android system root image:
    # sudo dd if=system.img of=/dev/sdx2; sync
    Download the android recovery image:
    # sudo dd if=recovery.img of=/dev/sdx4; sync
```

Figure 17. dd utilities for downloading images

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4641
Rev. 0
12/2012