Multiple Image Support on JN51xx Application Note

JN-AN-1228

Revision 1.1

11-Nov-2016

Contents

About this Application Note

The NXP JN516x wireless microcontrollers are capable of running multiple applications in internal Flash memory. This Application Note demonstrates two methods that can be used to improve production tests and calibration, or simply run one of two stored applications based upon a DIO value at start-up.

This document makes reference to test applications, **BootTestApp** and **ValidateImage**, which are supplied in the Application Note package.

Organisation

This manual consists of four chapters, as follows:

- Chapter 1 introduces the concepts for using multiple JN51xx images.
- Chapter 2 describes the approach of using the Application ID of the image to select the image to run.
- Chapter 3 details the approach of using application headers to select the image to run.
- Chapter 4 lists the compatible NXP hardware and software, and describes the use of BeyondStudio to build an application

Related Documents

JN-UG-3087	JN516x Integrated Peripherals API User Guide
JN-AN-1003	Boot Loader Operation Application Note

Support Resources

To access online support resources such as SDKs, Application Notes and User Guides, visit the Wireless Connectivity area of the NXP web site:

www.nxp.com/products/interface-and-connectivity/wireless-connectivity

All NXP resources referred to in this manual can be found at the above address, unless otherwise stated.

Trademarks

1 Multiple Applications in JN51xx Internal Flash

The JN516x internal Flash memory is divided into 32Kbyte sectors. The JN516x bootloader checks for valid images that start in any of the sectors. If one is found, the Flash is remapped so that the sector in which the image begins is designated as Sector 0 starting at address 0, and any subsequent sectors occupied by the image are numbered contiguously (1, 2,…). Any sectors not occupied by the image are not remapped. After the remapping, the image is run.

Here we demonstrate two methods for switching between applications in internal Flash memory.

1.1 Switching between Applications Using the App ID

The first approach is to use a function in the bootloader called `VLoadBootImage(uint16 u16AppID)` that can be used to execute applications that are compiled with a particular Application ID. The advantage of this approach is that no internal Flash memory is reprogrammed when switching between the applications and both applications are still valid in Flash and can be restarted later. This approach is described further in Chapter 2.

1.2 Invalidating Application Headers

The second approach invalidates (by setting to 0xFF) the headers of the applications that are not required. This means that only the application with a valid header will run. Once this application has run and is no longer required (e.g. a production test or factory calibration application), the header for the main application can be validated (header reprogrammed with valid data) while the current application is invalidated (header reprogrammed to zero), and the device is restarted. Without erasing the sector, it is then no longer possible to run the original test application. This approach is described further in Chapter 3.

1.3 Limitations

When starting from power-up or reset, one image must have a valid header and an Application ID of zero. Otherwise, the bootloader will check for valid images in external Flash memory and reprogram the internal Flash memory if it finds one. Failing this, the bootloader will enter programming mode.

2 Using the Application ID

This chapter describes how an Application ID can be used to dictate which application the JN516x bootloader will run from multiple applications stored in JN516x internal Flash memory.

2.1 Allowing Different Application IDs

In the JN516x bootloader, there is a function with the following prototype:

`VLoadBootImage(uint16 u16AppID)`

This function can be called to execute an image with a specified Application ID (`u16AppID`). The function is not in a fixed location, but depends on the bootloader version that is running. For the current production JN516x devices, the bootloader version number and the address of the function in boot Flash are given below.

Device	Firmware Version (Hex)	Address (Hex)
JN5161, JN5164, JN5168	0x00080006	0x09E3
JN5169	0x000B0002	0x1A76

In order to build images with different Application IDs, a change is required to the **AppBuildStart.ld** file that is located in the following folder within the JN516x Software Developer's Kit (SDK) installation:

C:\NXP\bstudio_nxp\sdk\<*SdkVersion*>\Chip\<*ChipVersion*>

The change is to define a variable called `_application_id` and then replace `SHORT(0x0000)` with `SHORT(_application_id)` as shown below. Updated versions of these files are include in the Application Note package.

```
/* Set minimum heap size unless it is already set from App_Stack_Size.ld */
_minimum_heap_size = DEFINED(_minimum_heap_size) ? _minimum_heap_size : 3350;
_application_id = DEFINED(_application_id) ? _application_id : 0;

_SwConfig =  (DEFINED(g_bSWConf_Debug)) |
(DEFINED(g_bSWConf_AltDebugPort)<<1);

MEMORY
{
        flash  : ORIGIN = 0x00080000, LENGTH = 0x80000
        ram    : ORIGIN = 0x0400004c, LENGTH = 0x07fb4
}

/* Now building all code at once, so include vectors. MAC address is
embedded
   in build for now */
SECTIONS
{
        .version ABSOLUTE(ORIGIN(flash)-4):
        {
        /*
        000b - Chip Type 5169
        03   - 32K Ram
        0f   - 512K Flash
```



```
          */
LONG(0x0f03000b)
} > flash

.bir ABSOLUTE(ORIGIN(flash)):
{
     _flash_start = ABSOLUTE(.);
     _flash_beg = ABSOLUTE(.);
/* Magic number */
LONG(0x12345678)
LONG(0x11223344)
LONG(0x55667788)

/* Configuration A (32K, DRE, 16MHz) */
BYTE(0x08)

/* Configuration B (version 1) */
BYTE(0x01)

/* Application ID */
SHORT(_application_id)
} > flash

.flashheader :
{
/* Encryption IV */
LONG(0x00000000)
LONG(0x00000000)
LONG(0x00000000)
SHORT(0x0000)
```

2.2 Specifying an Application ID

Note: This has already been done in the test application.

To change the Application ID in an image, modify the application makefile to define `_application_id` when linking:

```
# Define the _application_id for the correct value when linking
LDFLAGS += -Wl,--defsym,_application_id=$(APP_ID)
```

To define the function, modify the makefile to create a function name that points to the correct address in the bootloader.

```
#Define the vLoadBootImage function call in the Bootloader
ifeq ($(JENNIC_CHIP),JN5168)
LDFLAGS += -Wl,--defsym,vLoadBootImage=0x000009e3
endif
ifeq ($(JENNIC_CHIP),JN5164)
LDFLAGS += -Wl,--defsym,vLoadBootImage=0x000009e3
endif
ifeq ($(JENNIC_CHIP),JN5161)
LDFLAGS += -Wl,--defsym,vLoadBootImage=0x000009e3
endif
ifeq ($(JENNIC_CHIP),JN5169)
LDFLAGS += -Wl,--defsym,vLoadBootImage=0x00001a76
endif
```

2.3 Building and Running the Demonstration

To run the test application, **BootTestApp**, supplied in this Application Note, you must modify the **AppBuildStart.ld** file as described in Sections 2.1 and 2.2.

The rest of this section describes how to build and load a JN516x image containing two instances of the test application with different Application IDs, 0 and 1. On start-up of the JN516x device, the bootloader will always run the application with an Application ID of 0. Once running, the test application will itself switch execution to the application with an Application ID of 1.

2.3.1 Details of the Test Application

BootTestApp is a simple test application that:

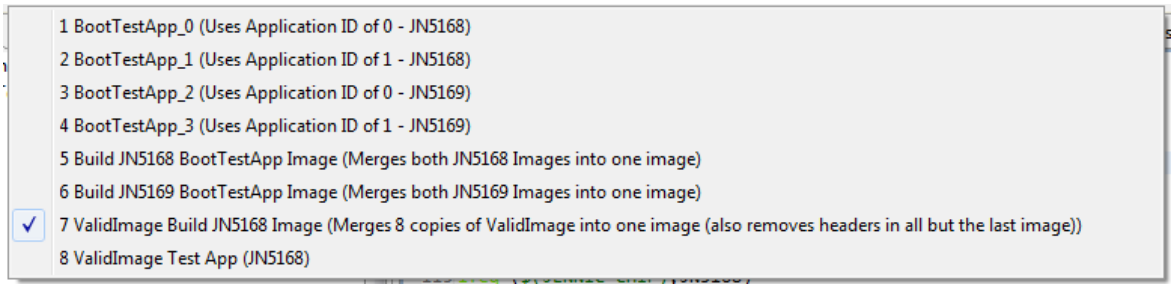
1. Prints out text indicating which version of the code is running (derived from the Application ID)
2. Adds a delay (to make the text readable)
3. Calls **vBootImage()** with the other Application ID to restart with the corresponding application.

The main body of the code is shown below:

```
#if IMAGE_ID==0
vPrintf("Running Boot Image 0\n\n");
AppID = 1;
#elif  IMAGE_ID==1
vPrintf("Running Boot Image 1\n\n");
AppID = 0;
#else
#pragma message "* Application ID not defined *"
#endif

// Add a delay
for (u32ctr=0;u32ctr<500000; u32ctr++);

// Run the other application
vLoadBootImage(AppID);

// should never get here
while(1);
```


2.3.2 Building the Applications

The application can be built with an Application ID of 0 or 1. The application can also be built for either the JN5168 or JN5169 device. Combinations of these build options are listed as options 1 to 4 in the BeyondStudio project (shown below).



Therefore build two applications for the same chip (see Section 4.2), one with an Application ID of 0 and one with an Application ID of 1.

2.3.3 Merging the Applications into a Single Image

You have produced two different binary files (or images) that can be individually programmed into the JN516x internal Flash memory. However, in this case you need to combine the two images into a single image to be programmed into Flash memory. To do this, you can use a utility called **ImageMerge** that is included with this Application Note package. When using **ImageMerge**, specify `–a <filename> to –h <filename>` to define the order of the images (with `-a` specifying the first image) in the final output file. The final output file is specified with `–o`. For example:

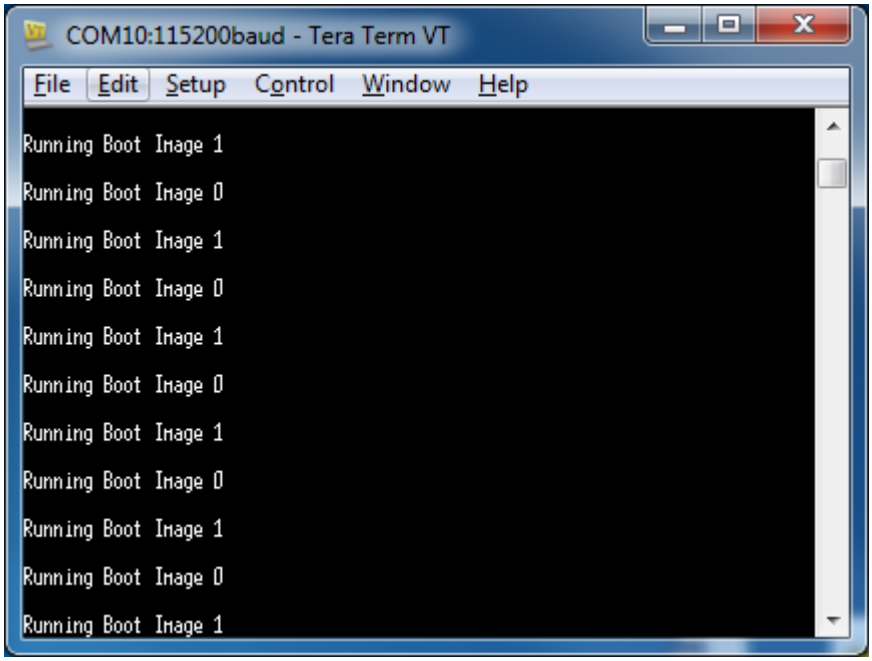
```
ImageMerge -a BootTestApp_JN5168_ID0.bin -b BootTestApp_JN5168_ID1.bin
-o BootTestApp_JN5168_ID0and1.bin
```

This utility can be called by using build option 5 (for JN5168) or option 6 (for JN5169) in the BeyondStudio project to create the final output file (**BootTestApp_JN5168_ID0and1.bin**).

2.3.4 Loading and Running the Image

The image can be programmed into the JN516x device using the Flash programmer within BeyondStudio, as describe in the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*. This requires a serial connection to be established between the development PC and a board hosting the target JN516x device. You must also set up a terminal window (such as Tera Term) associate with this serial connection on the PC, in order to display messages received from the JN516x device.

Once loaded, the application will automatically run and the following output will be displayed in the terminal window:



3 Using Application Header Validation

This chapter describes how application headers can be used to dictate which application the JN516x bootloader will run from multiple applications stored in JN516x internal Flash memory.

3.1 Header Format

Each application built for the JN516x devices has a header that provides details about the image to the bootloader, including:

- Where the entry points for **AppColdStart()** and **AppWarmStart()** are located
- The size of the application in Flash
- The size of the RAM segment that contains pre-initialised variables (these are copied by the bootloader into RAM)
- The size of the BSS segment (this is an area of RAM that needs to be pre-initialised to zero - the bootloader will do this initialisation before executing the application)
- The Application ID
- "Magic Bytes" that define the header as being valid, invalid or blank – this value comprises four bytes as follows:
  - 0xFF is used to indicate a blank header corresponding to an application instance that has not yet been run - an unused application
  - 0x00 is used to indicate an invalid header corresponding to an application instance that cannot be run – an already used application
  - A pre-defined value is used to indicate a valid header corresponding to the application instance that is currently running - the active application

The supplied **ValidateImage** application tests these headers at the beginning of each Flash sector to assess which sectors contain unused images.

Once the bootloader has established that an unused image is available in JN516x internal Flash memory, it will remap the addresses in Flash if necessary (to ensure that when running the application, the address map starts from zero, as this is how the image will have been built). If an image uses multiple sectors then all the sectors used are remapped so that the image appears contiguously from address zero in the memory map.

3.2 Building and Running the Demonstration

3.2.1 Details of the Test Application

ValidateImage is a simple application that is duplicated in all eight Flash sectors, but only one of these eight application instances will be active at any one time. The functionality of the application is to:

1. Check each sector to find the valid (active) image. When it is found, display the application configuration that is in the header.
2. Find all unused images (with their header's magic bytes set to 0xFF).
3. If one exists then select the one in the highest sector, validate its image (set its magic bytes to a valid value) and invalidate the current image (set its magic bytes to 0x00).
4. Restart the device with the newly activated application and repeat until all images have been used, and then stop with the message "Complete".

The current application can be invalidated using the function:

```
void vInvalidateCurrentImage(void)
```

This function will set the magic bytes of the current application to zero. It does not matter which sector the application was stored in, as the Flash will have been remapped so that the application starts at the first sector. This means that programming the first sixteen bytes to zero invalidates the header. Sixteen bytes are selected because the **bAHI_FullFlashProgram()** function requires a multiple of 16 bytes.

To validate an image, the following function can be used:

```
void vValidateImage(uint32 u32Address)
```

This function will program the correct magic bytes to the start of the application given by the address `u32Address`. It assumes that that the header is currently set to 0xFF.

The program is based around a FOR loop that calls the **vDumpFlash()** function for each of the sector. **vDumpFlash()** accepts the following parameters:

- **uint32** *u32Sector*: The number of the sector being checked
- **uint32** *u32Remap*: The current value of the Remap register to establish where the actual program is located

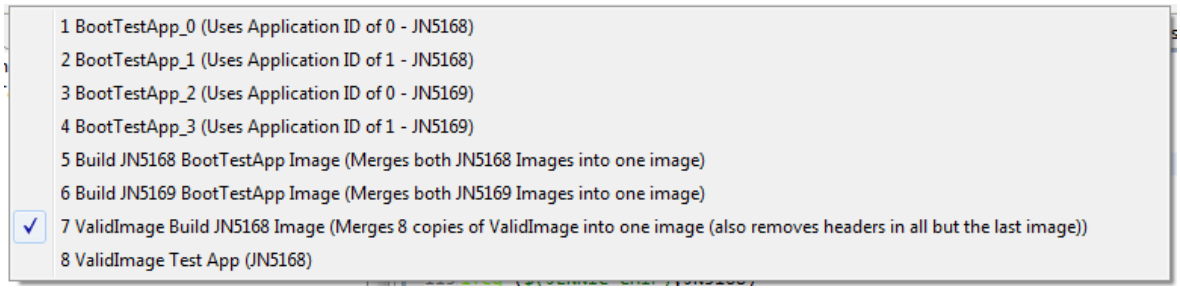
The function performs the following operations.

1. Check that the 12 magic bytes match the expected values. If so, display the information that is in the header.
2. Detect if the header is all zeros (already used and cannot be reprogrammed) or all ones (blank and so the image can be validated)
3. Store the sector and address of the next highest blank header (to be used later to decide validate that image).

Once the FOR loop has completed, a check is performed to see if all eight sectors have been covered. If so, the “Complete” message is displayed. Otherwise, the next highest blank header is validated, the current image is invalidated and application execution restarted.

3.2.2 Building the Application

The **ValidateImage** test application can be built for the JN5168 device using build option 8 in the BeyondStudio project (shown below).



Therefore build the application for the JN5168 device (see Section 4.2).

3.2.3 Merging Multiple Applications into a Single Image

You have now created a binary (with a valid header). To test this application, you need eight copies of the application with each aligned to a different Flash sector.

To do this, you can use the utility **ImageMerge** that is included within this Application Note package. When using **ImageMerge**, specify `–a <filename> to –h <filename>` to define the order of the images (with `-a` specifying the first image) in the final output file. The final output file is specified with `–o`.

You should set the headers’ magic bytes to 0xFF in the first seven images and force the only valid image to be in the eighth sector. You can use the `–p <sector>` option of **ImageMerge** to set the magic bytes to 0xFF in the specified sector. In this case, use the `–s` option to strip four bytes out of the beginning of each image in the final binary.

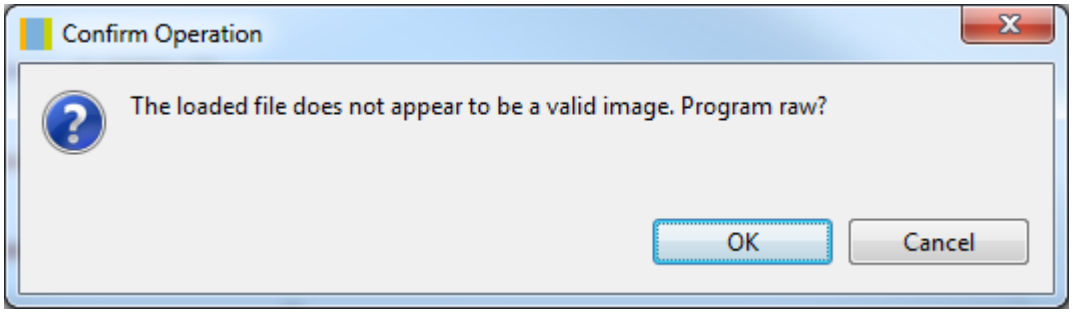
The build command is:

```
ImageMerge
-a ValidImage_TestApp_JN5168.bin
-b ValidImage_TestApp_JN5168.bin
-c ValidImage_TestApp_JN5168.bin
-d ValidImage_TestApp_JN5168.bin
-e ValidImage_TestApp_JN5168.bin
-f ValidImage_TestApp_JN5168.bin
-g ValidImage_TestApp_JN5168.bin
-h ValidImage_TestApp_JN5168.bin
-o ValidImage_TestApp_JN5168_Joined.bin
-p 0 -p 1 -p 2 -p 3 -p 4 -p 5 –p 6 –s
```

This can be achieved using option 7 in the BeyondStudio project.

3.2.4 Loading and Running the Image

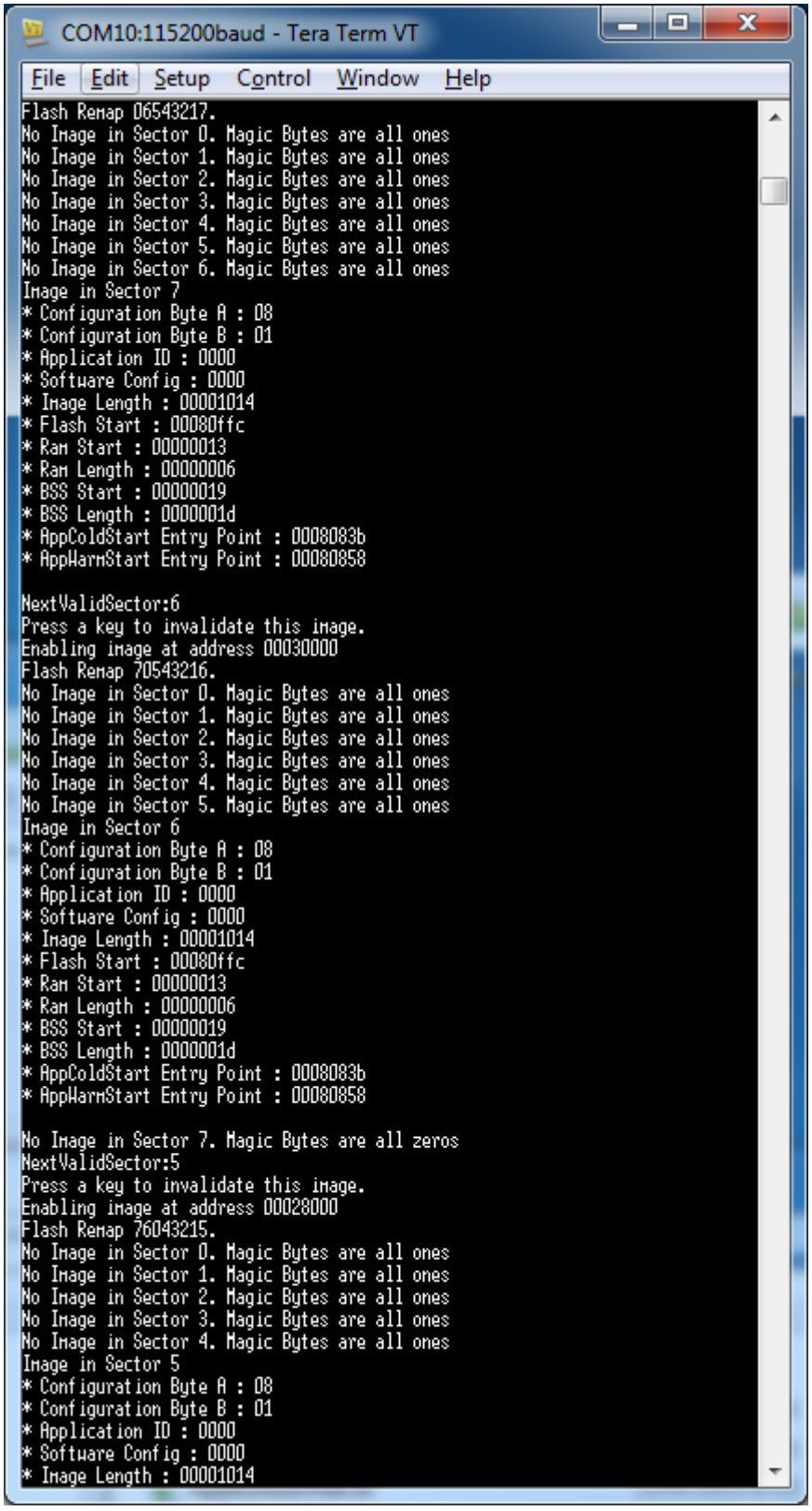
The final image (**ValidImage_TestApp_JN5168_Joined.bin**) can be programmed using the Flash programmer within BeyondStudio as describe in the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*. However, the following warning will be displayed:



This warning is because the image does not have the first four bytes (that indicate which processor the image was built for) or a valid header at the beginning of the image. The “program raw?” message is asking if you wish to program the image raw (i.e. without stripping off any bytes). In this case you do, so click **OK**.

The above requires a serial connection between the development PC and a board hosting the target JN516x device. You must also set up a terminal window (such as Tera Term) associate with this serial connection on the PC, in order to display messages received from the JN516x device.

Once loaded, the application will automatically run and the following output will be displayed in the terminal window:


COM10:115200baud - Tera Term VT
File Edit Setup Control Window Help
Flash Remap 06543217.
No Image in Sector 0. Magic Bytes are all ones
No Image in Sector 1. Magic Bytes are all ones
No Image in Sector 2. Magic Bytes are all ones
No Image in Sector 3. Magic Bytes are all ones
No Image in Sector 4. Magic Bytes are all ones
No Image in Sector 5. Magic Bytes are all ones
No Image in Sector 6. Magic Bytes are all ones
Image in Sector 7
* Configuration Byte A : 08
* Configuration Byte B : 01
* Application ID : 0000
* Software Config : 0000
* Image Length : 00001014
* Flash Start : 00080ffc
* Ram Start : 00000013
* Ram Length : 00000006
* BSS Start : 00000019
* BSS Length : 0000001d
* AppColdStart Entry Point : 0008083b
* AppWarmStart Entry Point : 00080858
NextValidSector:6
Press a key to invalidate this image.
Enabling image at address 00030000
Flash Remap 70543216.
No Image in Sector 0. Magic Bytes are all ones
No Image in Sector 1. Magic Bytes are all ones
No Image in Sector 2. Magic Bytes are all ones
No Image in Sector 3. Magic Bytes are all ones
No Image in Sector 4. Magic Bytes are all ones
No Image in Sector 5. Magic Bytes are all ones
Image in Sector 6
* Configuration Byte A : 08
* Configuration Byte B : 01
* Application ID : 0000
* Software Config : 0000
* Image Length : 00001014
* Flash Start : 00080ffc
* Ram Start : 00000013
* Ram Length : 00000006
* BSS Start : 00000019
* BSS Length : 0000001d
* AppColdStart Entry Point : 0008083b
* AppWarmStart Entry Point : 00080858
No Image in Sector 7. Magic Bytes are all zeros
NextValidSector:5
Press a key to invalidate this image.
Enabling image at address 00028000
Flash Remap 76043215.
No Image in Sector 0. Magic Bytes are all ones
No Image in Sector 1. Magic Bytes are all ones
No Image in Sector 2. Magic Bytes are all ones
No Image in Sector 3. Magic Bytes are all ones
No Image in Sector 4. Magic Bytes are all ones
Image in Sector 5
* Configuration Byte A : 08
* Configuration Byte B : 01
* Application ID : 0000
* Software Config : 0000
* Image Length : 00001014

4 Miscellaneous Topics

4.1 Application Compatibility

The software supplied with this Application Note allows stand-alone applications to be built and outputs the results to a terminal window. These applications have been designed for use with the following hardware and software:

Product Type	Part Number	Version	Supported Chips
Evaluation Kit	JN516x-EK001 JN516x-EK004	-	JN5161 JN5164 JN5168 JN5169
SDK Libraries	JN-SW-4163 JN-SW-4168	V1307 or higher V1279 or higher	JN5161 JN5164 JN5168 JN5169
BeyondStudio for NXP	JN-SW-4141	V1308	JN5161 JN5164 JN5168 JN5169

4.2 Importing and Building in BeyondStudio

To build an application and load it into a JN516x-based module:

1. Start the BeyondStudio platform and import the relevant project files (**.project** and **.cproject**) as follows:
2. In BeyondStudio, follow the menu path **File>Import** to display the **Import** dialogue box.
    a. In the dialogue box, expand **General**, select **Existing Projects into Workspace** and click **Next**.
    b. Enable **Select root directory**, browse to the **Application** directory and click **OK**.
    c. In the **Projects** box, select the project to be imported and click **Finish**.
3. Build the application. To do this, ensure that the project is highlighted in the left panel of BeyondStudio and use the drop-down list associated with the hammer icon in the toolbar to select the relevant build configuration – once selected, the application will automatically build. The binary file will be created in the **Build** directory, the resulting filename indicating the chip type (e.g. **JN5168**) for which the application was built.
4. Load the resulting binary file into the module. You can do this directly from within BeyondStudio.

For more information, refer to the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*.

Revision History

Version	Date	Description
1.0	8-Feb-2016	First release
1.1	11-Nov-2016	Document re-worked for public release

Important Notice

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

NXP Semiconductors

For the contact details of your local NXP office or distributor, refer to:

www.nxp.com

