Freescale Semiconductor
Application Note

Document Number:AN4674
Rev 1, 06/2013

Qorivva Boot Assist Module Application

by: Mong Sim

Contents

1 Introduction

The boot loader software is a prevailing technology for modern System on Chip (SoC) application development. It provides some level of initialization to the system and bridges portability to the application program. In many cases, its role is invaluable. However, the boot loader does have its set of problems. It takes up resources and requires special skills to build it. The boot loader is also susceptible to corruption due to electrical surges, bad application programs, and so on, that can render the boot loader inoperable and cause the system to fail.

Freescale, however, provides its automobile customers an alternative to custom boot loader software— the Boot Assist Module (BAM). The BAM provides its users two interfaces — the Enhanced Serial Communication Interface (ESCI) and the Controller Area Network interface (CAN) — to upload application program using the BAM protocols. To embellish the BAM further, these two interfaces are equipped with the ability to detect fixed baud rate and automatic baud rate.

The BAM protocol is unique: it provides a simple but secure protocol, with password protection to ensure access rights to its users. The passwords are classified as Public and FLASH. The Public password option provides a fail-safe mechanism in case the FLASH password is forgotten. Although the BAM protocol does not provide any form of data integrity checks, it echoes back every byte that it has received to the user, for parity check.

The BAM requires the user to send data in a certain format defined by the BAM protocol. Whether it operates in fixed baud rate mode or automatic baud rate detection mode, the uploading application program must not violate the BAM protocol. This application note provides instructions for creating an application program that is capable of uploading application program to the SoC via the BAM ESCI and CAN interface in either fixed baud rate mode or automatic baud rate detection mode.

2 BAM Prerequisite

The BAM module is available in the MPC55XX and MPC56XX Freescale Qorivva devices. The BAM module supports the BAM protocol over the Enhanced Serial Communication and the Controller Area Network interfaces. However, in the MPC57XX, the BAF (Boot Assists Flash) provide a similar functionality as BAM. These Qorivva devices are tailored to target different market segments; therefore, these devices are configured differently. Although the BAM protocol remains the same across different devices, the BAM module initialization and memory map vary across devices. The BAM chapter of your Qorrivva device will provide information that complements this application note.

3 Programming Language Prerequisite

In this application note, the software examples are illustrated in Power Architecture assembly, C and C++ programming language. Readers are required to have working knowledge of these programming languages. The example programs are grouped into two types. The first type is the Target side program, which is a target specific program to be uploaded by the Host application and executes in the Target system. The second type is the Host side program, which is an implementation of the BAM protocol used to upload device specific application program to the Target system, the Qorivva device. All of these software examples require various hardware and software tools. Here is a list of the hardware and software tools used in this application note.

Software

- Microsoft Visual C++ Express 2010 microsoft.com
- Green Hills MULTI Compiler for Power PC ghs.com

Hardware

- XPC56XX Evaluations Motherboard with XPC563M144QFP Mini Module (EVB, The Target system)
- 12 VDC Power Supply for the EVB
- RS-232 cable
- Green Hill Probe for debugging
- Desktop PC with serial interface or any USB to serial cable
- SYSTEC USB CAN tool
- USB Cable to the CAN Tool
- DB9 Female to Female Connector to connect CAN Tool to EVB CAN port A
- Personal Computer with XP Operating System (The Host System)

4 BAM Protocol

The BAM provides two different protocols for downloading application program via the ESCI (some Qorivva Devices use FlexLinD UART mode instead of ESCI) or the CAN interfaces (MPC5746 does not supports BAF over CAN protocol). These two interfaces support both the fixed baud rate mode and automatic baud rate detection mode (Not all Qorivva devices support automatic baud rate detection).

Let us see how these protocols are formatted, what the fields are and how we can use the protocols to upload application program to the Qorivva device.

NOTE

To select BAM protocol over eSCI or CAN by simply using the correct interface cable to connect to that interface and start sending the BAM formatted data to that interface.

4.1 BAM Protocol for Enhanced Serial Communication Interface (ESCI)

The BAM Protocol for the ESCI is a simple, yet secure, protocol. Although the BAM does not provide data integrity checks, it echoes back every byte received to the sender. The sender is requested to ensure the echoed byte is correct before sending the next byte out. If the echoed byte is a mismatch, the sender must terminate the transfer, reset the Qorivva device and restart the transfer.

4.1.1 ESCI Fixed Baud Rate

The BAM protocol for ESCI fixed baud rate format, shown in Table 1, consists of a Password field, a Start Address field, a Data Size field and a Data field. The Qorivva devices support both BookE (Classic Power Architecture fixed length code) and VLE (Variable Length Encoding). Specifying the type of code, Book or VLE, will be explained in the Data Size section.

Table 1. BAM Protocol for ESCI Fixed Baud Rate Format

Password	Start Address	Data Size	Data

4.1.2 ESCI Automatic Baud Rate Detection

The BAM protocol for ESCI automatic baud rate detection, shown in Table 2, has an additional Synchronization Byte field before the Password field. This Synchronization Byte is used by the BAM hardware to calculate the unknown incoming data baud rate transmitted by the Host system. The BAM uses this calculated baud rate and apply to the ESCI for subsequent data communication.

Table 2. BAM Protocol for ESCI Automatic Baud Rate Format

Sync Byte	Password	Start Address	Data Size	Data

An additional pull down resistor is also required between the signal pin “EVTO” and “GND.” After installing this pull down resistor, reset the EVB. Use a debugger to verify if the SIU_RSR [ABR] bit is set at address 0xC3F900F for MPC5634M. If you are using a different device, please refer to your device Reference Manual for correct setting.

4.1.3 ESCI Synchronization Byte Field: 0x00

The Synchronization Byte must be used in automatic baud rate detection. The BAM requires the host to send a synchronization byte of 0x00 prior to sending the Password. The BAM uses the Synchronization Byte to detect the incoming data baud rate.

The BAM will not echo the Synchronization Byte.

4.1.4 ESCI Password Field: 8 Bytes

The FLASH password is eight bytes long and is programmed by the factory into the internal shadow flash of the device. The Public password, however, resides within the BAM and cannot be changed — user can only change the FLASH password. A valid password must be always programmed in the shadow flash, regardless of which boot mode is used. For a password to be valid, none of its four 16-bit half words must equal to 0x0000 or 0xFFFF. Please see the specific device reference manual for more detailed information.

The BAM module will receive and echo the password transmitted by the Host system and compare it with its FLASH or Public password. If the password is a mismatch, the BAM module will terminate the transaction. If the echoed byte is mismatched, the sender must terminate the transfer, reset the Qorivva device and restart the transfer.

NOTE

If the FLASH password is corrupted due to the reprogramming of the shadow FLASH, the Qorivva device will be locked and all access to the Qorivva device will be denied.

4.1.5 ESCI Start Address Field: 32-Bits Word

The Start Address Field serves two purposes. First, it is the starting address at which the BAM will store the application program from the host. Second, the BAM will jump to the memory location specified by this Start Address Field and relinquish control to the code after the upload is completed.

4.1.6 ESCI Data Size Field: 32-Bits Word

The Data Size Field tells the BAM the size of application program in bytes. In addition, the most significant bit of this field (VLE mode bit), if set, tells the BAM that the application program is in variable length encoded (VLE). If the MSB of this field is clear, the application program is in BookE format.

4.1.7 ESCI Data Field: The Application Program

The Data Field contains application program to be uploaded by the Host system to the Qorivva device via the BAM. The size of the application program is defined by the Data Size Field.

5 BAM Protocol for Controller Area Network Interface (CAN)

The BAM protocols for ESCI and CAN are similar if you examine how the data is arranged. The only difference is the format of the hardware interface which is used. The ESCI is byte oriented whereas the CAN interface is block oriented. Instead of sending data byte by byte using the ESCI, you can send a block of eight bytes of data over the CAN interface.

The BAM only implements standard frame transmission over CAN.

5.1 CAN Automatic Baud Rate Protocol Detection Message Format

The BAM protocol for CAN Automatic Baud Rate detection Message Format, shown in Table 3, is used when CAN Automatic Baud Rate detection mode is configured. This format consists of a message ID field and a message size field.

Table 3. CAN Automatic Baud Rate Message Format

MSG ID	MSG Size

5.2 CAN Message Format

The BAM Protocol for CAN Message Format, shown in Table 4, includes a Message ID field, a Message Size field and a Data field. This CAN Message Format is used to encapsulate the Password Message, the Start Address and Data Size Message and the Data Message. There are a total of seven message IDs. These IDs are described in detail in the Message ID CAN Message ID: 0x00, 0x01,0x02,0x03,0x11,0x12 and 0x13.

Table 4. CAN Fixed Baud Rate Message Format

MSG ID	MSG Size	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

5.3 CAN Message ID: 0x00, 0x01,0x02,0x03,0x11,0x12 and 0x13

The BAM protocol for CAN has seven unique Message IDs. Six IDs serves as three pairs for transmit and echo and one Message ID is used to inform the Qorivva device that a Synchronization Message is encapsulated for CAN automatic baud rate detection. Please see the Message IDs as tabulated in Table 5.

Table 5. BAM Protocol for CAN Message ID

ID	Description
0x00	Automatic Baud Rate Detection Message ID send by the host
0x11	Password Message ID send by the host
0x01	Password Message ID echo by the BAM
0x12	Start Address and Data Size Message ID send by the host
0x02	Start Address and Data Size Message ID echo by the BAM
0x13	Data Message ID send by the host
0x03	Data Message ID echo by the BAM

5.4 CAN Synchronization Message

The Synchronization Message, shown in Table 6, must be used in CAN Automatic Baud Rate detection. The BAM requires the host to send a Synchronization Message with a message ID of 0x00 and a message size of 0x00 prior to sending the Password Message. The BAM uses this message to detect the incoming data baud rate.

Table 6. BAM Protocol for CAN Synchronization Message

MSG ID	MSG Size

The Qorivva Device will not echo the Synchronization message.

5.5 CAN Password Message

Please refer to ESCI Password Field: 8 Bytes for FLASH and Public Password.

The Password Message, shown in Table 7, has a message ID of 0x11 and a message size of eight follows by the eight bytes of password. For this example, the password is “FEEDFACECAFEBEEF.”

Table 7. BAM Protocol for CAN Password Message

MSG ID	MSG Size	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x11	0x08	0xFE	0xED	0xFA	0xCE	0xCA	0xFE	0xBE	0xEF
		PASSWORD							

Table 8. Qorivva Device Echoes CAN Password Message

MSG ID	MSG Size	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	0x08	0xFE	0xED	0xFA	0xCE	0xCA	0xFA	0xBE	0xEF
		PASSWORD							

The Qorivva Device will echo the Password Message, shown in Table 8, with an ID of 0x01.

5.6 CAN Start Address and Data Size Message

The Start Address and Data Size Message, shown in Table 9, has a message ID of 0x12 and a message size of eight followed by the Start Address on the first four bytes and the Data Size on the lower four bytes. For this example, the Start Address is 0x40000000 and the Data Size is 0x1400.

Table 9. BAM Protocol for CAN Start Address and Data Size Message

MSG ID	MSG Size	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x12	0x08	0x40	0x00	0x00	0x00	0x00	0x00	0x14	0x00
		Address				VLE + Data Size			

In addition, the most significant bit of Date Size (MSB of Byte 4, the VLE mode bit) if set, tells the BAM that the application program is in variable length encoded (VLE). If the VLE mode bit is clear, the application is in BookE format. For this example, the application program is in BookE format.

Table 10. Qorivva Device Echoes Start Address and Data Size Message

MSG ID	MSG Size	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x02	0x08	0x40	0x00	0x00	0x00	0x00	0x00	0x14	0x00
		Address				VLE + Data Size			

The Qorivva Device will echo the Start Address and Data Size Message, shown in Table 10, with an ID of 0x02.

5.7 CAN Data Message

The Data Message, shown in Table 11, has a message ID of 0x13 and a message size of eight bytes or less is a target specific application program that you want to send to the Target system. The size of the application program is defined in the Start Address and Data Size message in the Date Size.

The Host application must send multiple data messages (If the defined data size is greater than eight bytes) up to the size defined in the Data Size field.

Table 11. BAM Protocol for CAN Data Message

MSG ID	MSG Size	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x13	0x08	Qorivva Application Code							

Table 12. Qorivva Device Echoes Data Message

MSG ID	MSG Size	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x03	0x08	Qorivva Application Code							

The Qorivva Device will echo the Data Message, shown in Table 12, with an ID of 0x03.

6 BAM Qorivva Device Specific Application Program

By now, you are familiar with the BAM Protocol formats over ESCI and CAN interfaces. We still need a BAM Qorivva Device specific application program to be transferred to the Target system. This device specific application program must have the following features as appended so that the uploaded application program can be successfully executed in the Target system. Here are the features.

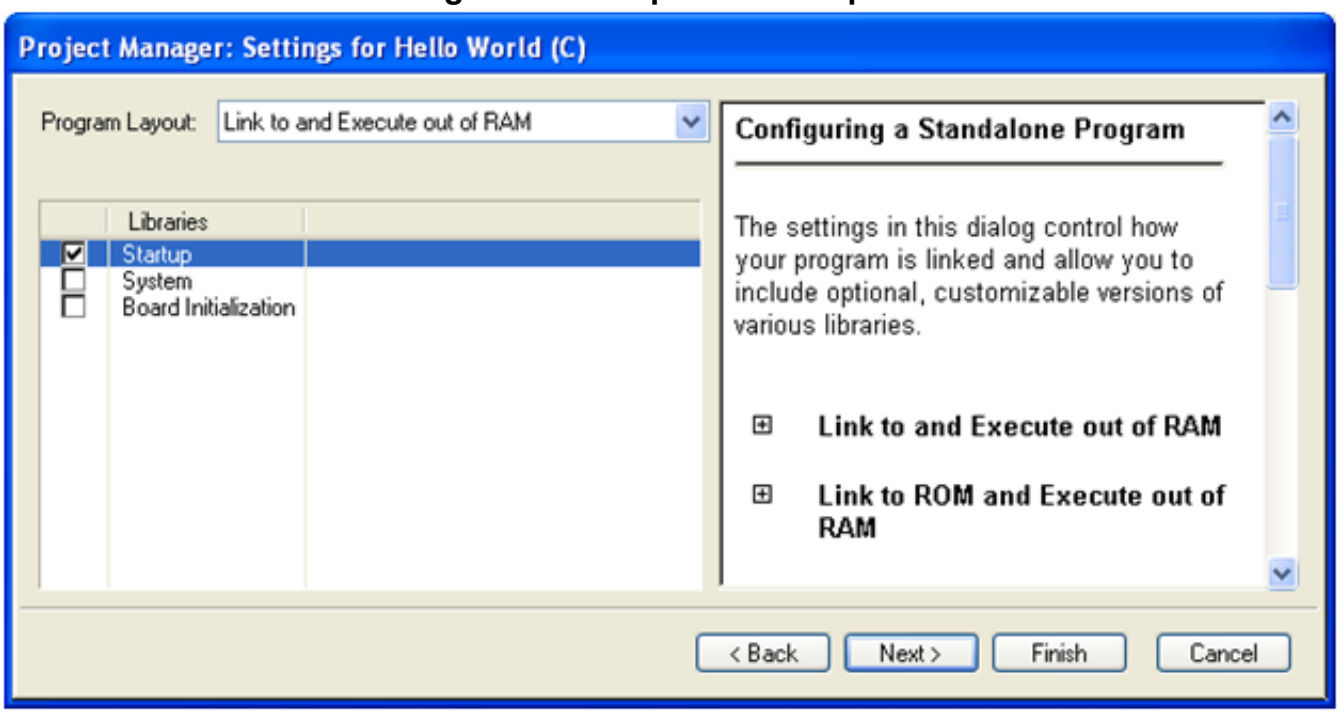
- The application program must execute from RAM
- The application program must NOT initialize RAM space occupied by the application program (this is like erasing the application program)
- The application program must initialize all the RAM space allocated for data and stack immediately after the BAM relinquished control to the application program.
- The application program image must be contiguous – any space in between the fragments of codes must be filled with bytes. The bytes content can be of any value.
- The application program must execute from the beginning of the RAM image.

This application note explains how to build the BAM Qorivva Device Specific Application Program using Green Hills MULTI Assembly and C programming language.

6.1 Building a RAM Application for BAM

Green Hills Compiler is used to illustrate this example. Create a “Standalone RAM” project using the project wizard with “Startup Libraries” as shown in Figure 1.

Figure 1. Startup Libraries Option



Once the project is created, modify the program to provide a simple blinking LED function. Then compile and run the program and make sure it execute correctly.

6.2 Modify Linker File

Next, we are going to modify the linker file. Open the “standalone_ram.ld” linker file and modify the following section, shown in Figure 3 and append two markers at the end of the linker file.

In the “MEMORY” section make the following changes.

```
//  64KB of internal SRAM starting at 0x40000000
    dram_rsvd1      : ORIGIN = 0x40000000, LENGTH = 0
    dram_reset      : ORIGIN = .,          LENGTH = 0
    dram_memory     : ORIGIN = .,          LENGTH = 64K-8K-16
    heap_and_stack  : ORIGIN = .,          LENGTH = 8K
    dram_rsvd2      : ORIGIN = .,          LENGTH = 16
```

Figure 2.

In the “DEFAULTS” section make the following changes.

```
DEFAULTS {
    stack_reserve = 4K
    heap_reserve  = 4K
}
```

Figure 3. Memory Map

End of the linker file is appended by these two markers, shown in Figure 4. These two markers will provide the start address and the ending address where the SRAM needs to be initialized.

```
__ram_image_heap = ENDADDR (heap_and_stack);
__ram_image_end  = ENDADDR (.bss);
```

Figure 4. Memory Region Markers

After you have done all the above, compile the program and make sure it still executes as expected. If not, back track and correct the error and repeat the process.

6.3 Modify the C Runtime File

Finally, we need to add a minimum initialization code to the CRT file “crt0.ppc” (C RunTime) resident in the “tgt\libstartup” directory so that the RAM Application can execute successfully via the BAM upload. Open the “crt0.ppc” file with an editor and locate the code fragment in this file as appended in Figure 5.

Once you have located the code fragment, insert the initialization code listed in Figure 6 between the “addic” opcode and the macro “#endif /* PPC64 */” flow control statement.

```
; initialize RAM if we're running from flash
#if defined(__PPC64__)
      Lis      r11,       %highesta(__ghs_board_memory_init)
      ori      r11, r11, %highera(__ghs_board_memory_init)
      sldi     r11, r11, 32
      oris     r11, r11, %hiadj(__ghs_board_memory_init)
      addic.   r11, r11, %lo(__ghs_board_memory_init)
#else
      lis     r11, %hiadj(__ghs_board_memory_init)
      addic.   r11, r11, %lo(__ghs_board_memory_init)
#endif /* PPC64 */
```

Figure 5. Code Fragment in crt0.ppc

```
; This inclusion disables the watchdogs and adds system initialization
;
;*******************************
; Disable SWT and Core Watchdog
;*******************************
        //software watchdog off
        lis     r12,      SWT_CR@h
        ori     r12, r12, SWT_CR@l
        lwz     r11, 0(r12)
        clrrwi  r11, r11, 1
        stw     r11, 0(r12)

        //core watchdog off
        li      r6,0x00
        mtspr   340,r6

;*******************************
; init SRAM
;*******************************

        lis     r30,0x0000   // initializes ram space after the
        lis     r31,0x0000   // application program image
        lis     r11,0x4000
        ori     r11,r11,%lo(__ram_image_end)  #see linker file
sram_init:
        stmw    r30,0(r11)
        addi    r11,r11,8
        andi.   r12,r11,0xFFF0
        bne     sram_init
```

Figure 6. Initialization Code

The initialization code disables the core and software watchdogs, enable the BTB (Branch Target Buffer), initializes the portion of internal SRAM not occupied by the RAM Application code. We also need to define the “SWT_CR” at the beginning of the “crt0.ppc” file as shown in Figure 7.

```
SWT_CR  .equ    0xfff38000      # SWT control register
```

Figure 7. Software Watchdog Control Register

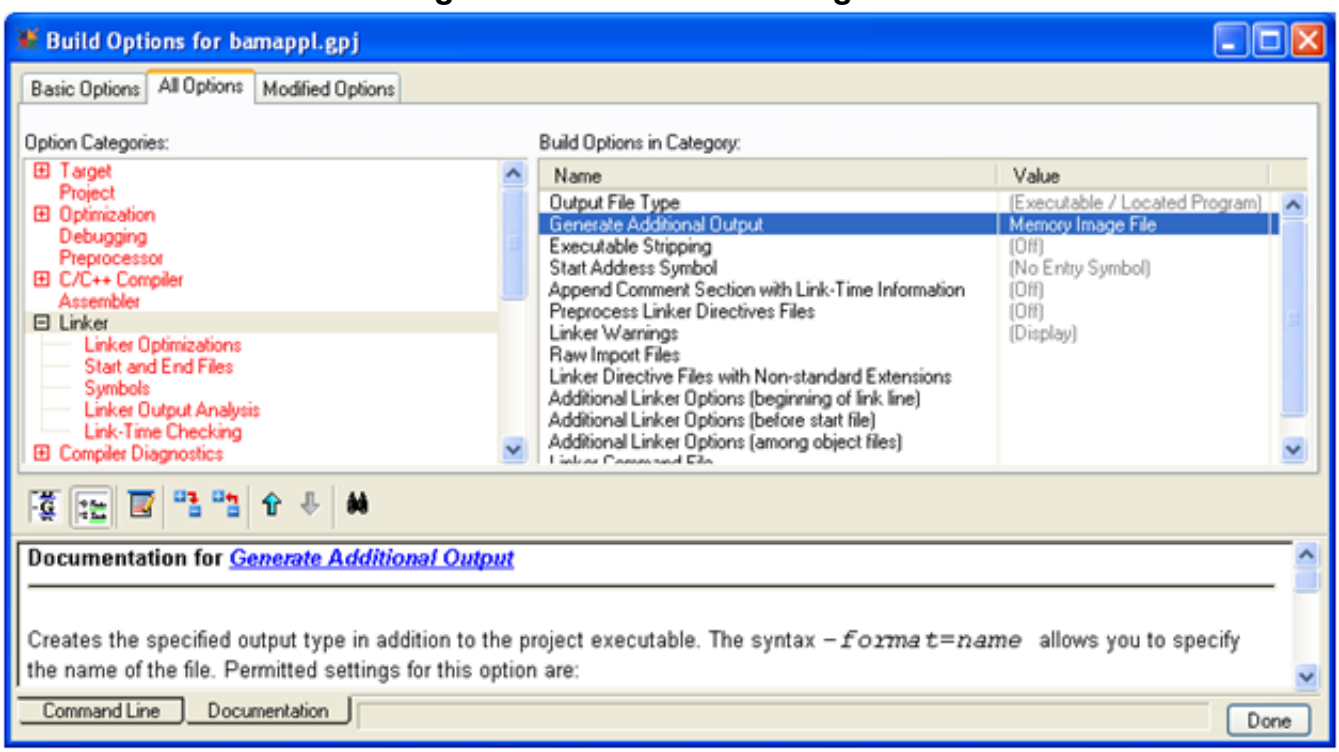
6.4 Generate image file

A common question is if we can use the S-Record (ASCII-Hex-Space format) or ELF (Executable and Linkable Format) file instead of a ram image file. It is possible to use, however, using S-Record or ELF files requires more complex software to convert the application program into a contiguous RAM image file. Application program in S-Record and ELF file formats are stored in sections and the sections may not be contiguous.

We are almost ready to test if the RAM Application works over BAM. Before we do that, we need the compiler to help us generate a memory image of the RAM Application. Let me show you how to configure the compiler to produce a memory image of your application program.

Go to the Project Window and right click at the “bamappl.prj: Program” and choose the “Set Build Options.” A window will appear, click on the “Build Options in Category:” and choose the “Generate Additional Output” and change its “Value” to “Memory Image File” as shown in Figure 8. When you have done that, click the “Done” button on the bottom right to close the window. Recompile the project and the compiler will produce an additional file namely, “bamappl.mem.”

Figure 8. Generate RAM Image File



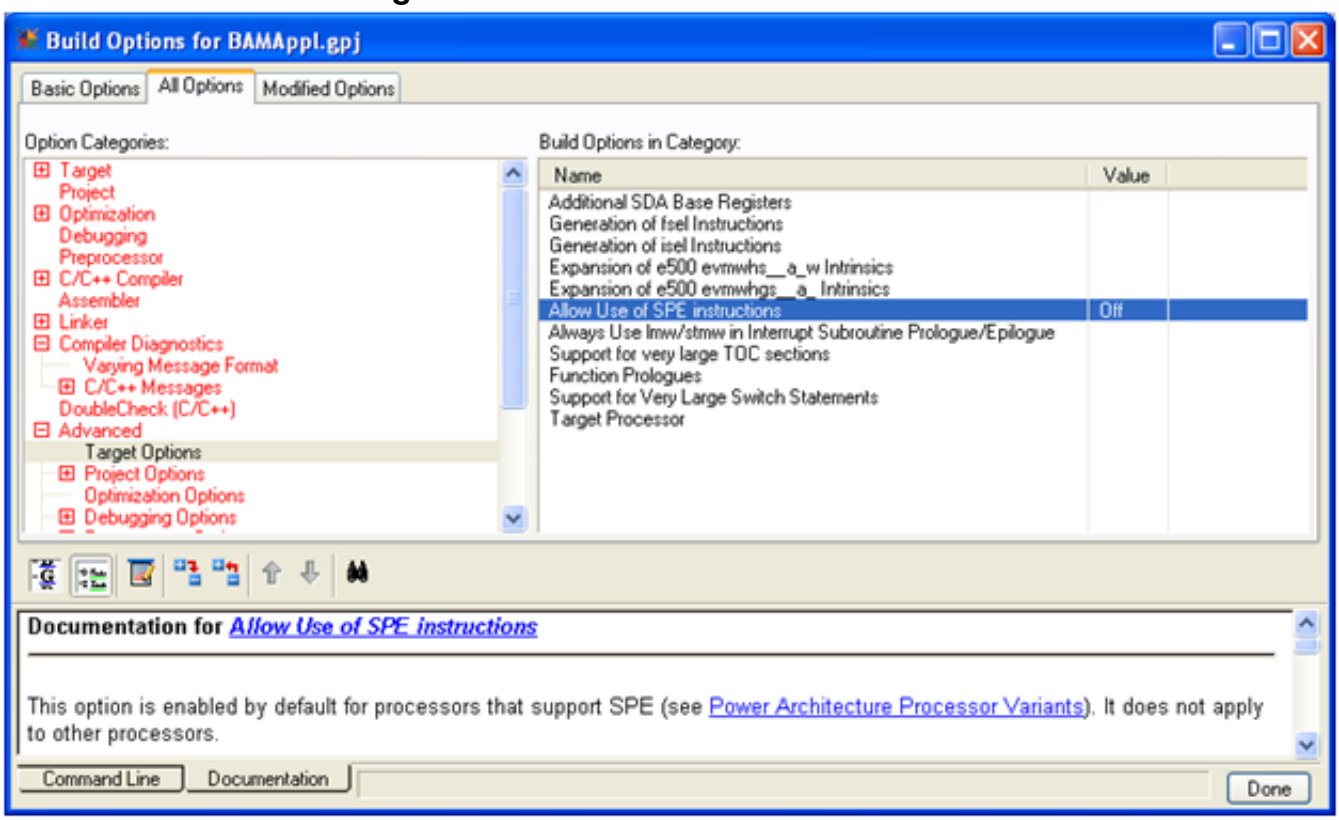
Some compilers like the Green Hills MULTI generate SPE code by default. If this is the case, you have two options to ensure that your RAM Application works (Most of Qorivva devices support SPE, please refer to your device RM). One, enable your device SPE feature with the appended code as shown in Figure 9.

Two, configure your compiler not to generate SPE codes as shown in Figure 10.

Figure 9. Enable SPE

```
//*****************************
// Enable SPE
//*****************************
        mfmsr r6
        oris  r6, r6, 0x0200
        mtmsr r6
```

Figure 10. Disable SPE Code Generation



Now we are ready to upload this application via the BAM to our Target system. If everything works, we should see four LEDs blinking in binary counting order. Please see Appendix1 for the source code of this application program. For a copy of this project, please contact your Freescale Sales Representative.

7 BAM Host Specific Application Program

At this stage, you have learned the BAM protocols over ESCI and CAN interfaces. You also learned how to create a BAM Qorivva Specific Application Program and how to generate image file using the compiler. What we are short of is a delivery system program, the BAM Host Specific Application Program.

The BAM Host Specific Application Program is an application program running in the host system that is capable of uploading the Qorivva Device Specific application program to the Target system (Qorivva device) via the ESCI or CAN interface using the BAM protocol.

Before we talk about the minimum implementation of the BAM Host Specific Application Program, let me show you how the Host system and the Target system data transaction sequences over the ESCI and the CAN interfaces work so that we can picture what is the minimum implementation we need for the BAM Host Specific Application Program.

These two sequences are given in two ladder diagrams, one for the ESCI and one for the CAN respectively.

7.1 ESCI and Host Data Transaction Sequence

Figure 11. ESCI Ladder Diagram

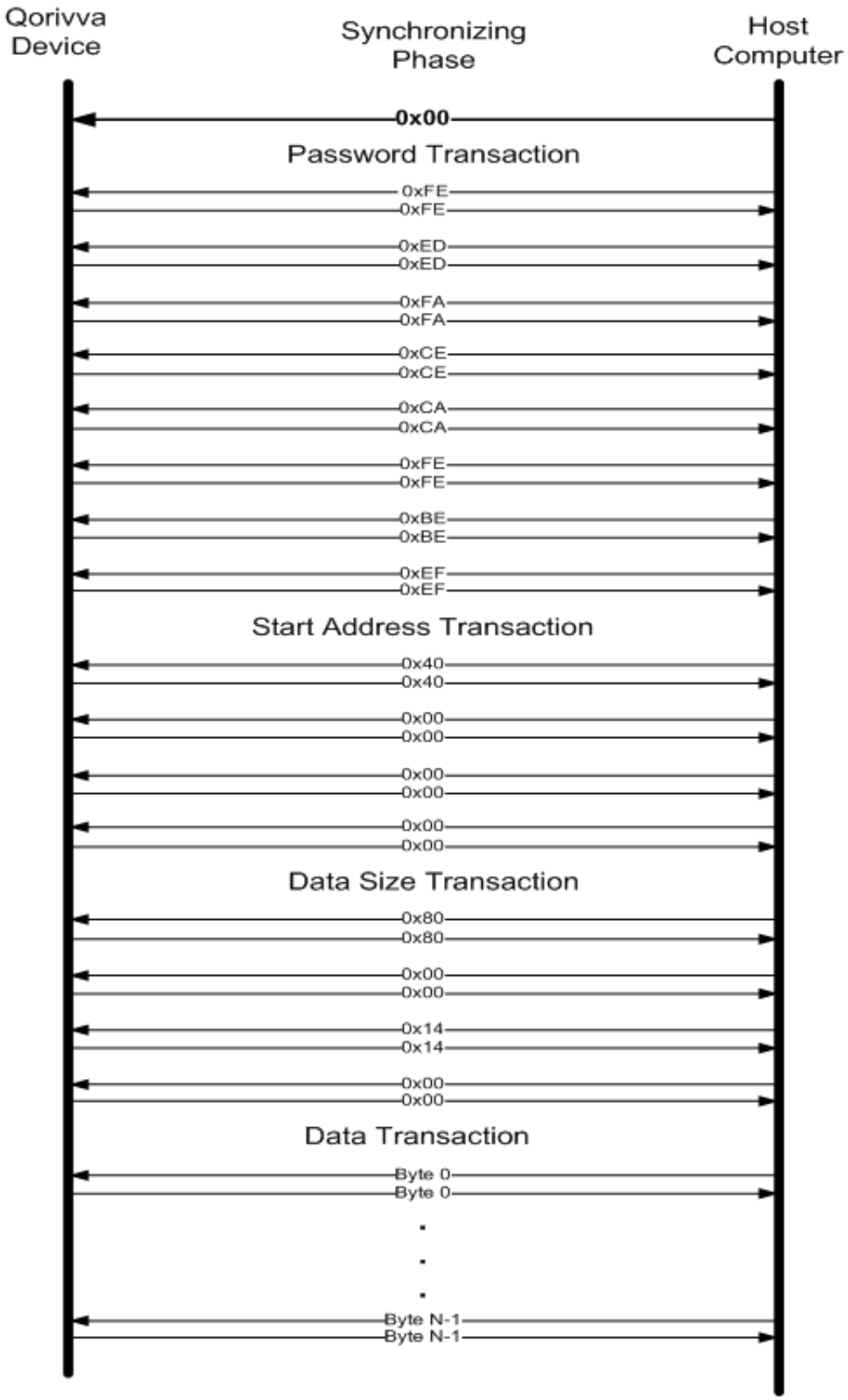

Qorivva Device
Synchronizing Phase
Host Computer
0x00
Password Transaction
0xFE
0xFE
0xED
0xED
0xFA
0xFA
0xCE
0xCE
0xCA
0xCA
0xFE
0xFE
0xBE
0xBE
0xEF
0xEF
Start Address Transaction
0x40
0x40
0x00
0x00
0x00
0x00
0x00
0x00
Data Size Transaction
0x80
0x80
0x00
0x00
0x14
0x14
0x00
0x00
Data Transaction
Byte 0
Byte 0
Byte N-1
Byte N-1

Figure 11 shows the transaction sequence between the Target system ESCI and the Host system RS232 port with the following parameters and a detail explanation in Table 13.

Table 13. ESCI and Host Transaction Sequence

Description	Value	Remarks
Synchronization Byte	0x00	Host sends this byte only if automatic baud rate detection is configured (See 4.1.3). If fixed baud rate is configured, user must NOT send this byte. The Qorivva Device will NOT echo this byte.
Password	0xFE, 0xED, 0xFA, 0xCE, 0xCA, 0xFE, 0xBE and 0xEF	After the Synchronization Byte (Only in automatic baud rate detection mode), the Host will transmit the password byte by byte starting with 0xFE and end with 0xEF. The Qorivva Device will echo every byte it received from the Host.
Start Address	0x40, 0x00, 0x00 and 0x00	After the Password, the Host will transmit the Start Address starting with 0x40 and end with 0x00 (The Start Address is 0x40000000). The Qorivva Device will echo every byte it received from the Host.
Data Size	0x80, 0x00, 0x14 and 0x00	After the Start Address, the Host will transmit the Data Size starting with 0x80 and end with 0x00 (The Data Size is 0x80001400). The most significant bit set on the Data Size indicates that the application program is in VLE mode. The size of the application program is 0x1400 or 5120 bytes. The Qorivva Device will echo every byte it received from the Host.
Data	Byte 0…Byte N-1	After the Data Size, the Host will transmit the Data, application program. The size of the Data is defined in the Data Size field. The Qorivva Device will echo every byte it received from the Host. If the Qorivva Device has received the number of bytes as defined in the Data Size Field, the BAM will relinquish control to the application program by executing a jump command to the memory location defined by the Start Address Field.

7.2 CAN and Host Data Transaction Sequence

Figure 12. CAN Ladder Diagram

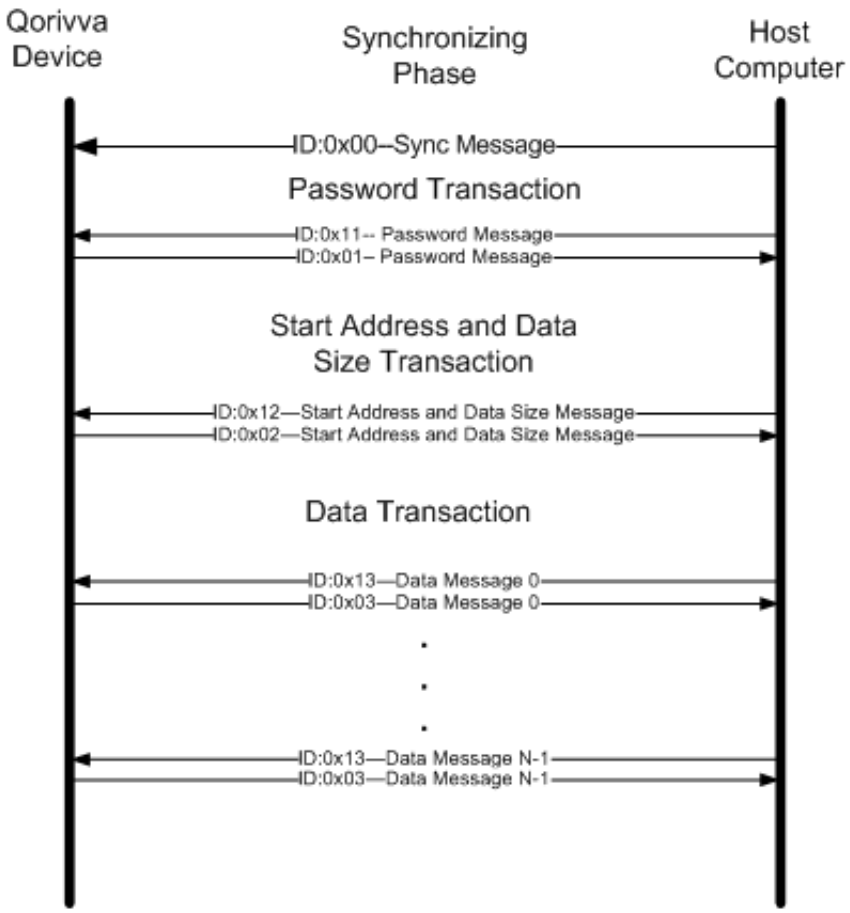


Figure 12 shows the transaction sequence between the CAN and a CAN tool connected to the Host computer with the following parameters and a detail explanation in Table 14.

Table 14. CAN and Host Transaction Sequence

Description	ID	Size	Data	Remarks
Synchronization Message	0x00	0x00	-	Host transmits this Synchronization Message only if automatic baud rate detection is configured (See 4.2.4). If fixed baud rate is configured, user must NOT send this Message. The Qorivva Device will not echo this Message.
Password Message	0x11	0x08	0xFE, 0xED, 0xFA, 0xCE, 0xCA, 0xFE, 0xBE and 0xEF	After the Synchronization Message (Only in

Table continues on the next page...

Table 14. CAN and Host Transaction Sequence (continued)

Description	ID	Size	Data	Remarks
				automatic baud rate detection mode), the Host will transmit the Password Message to the Qorivva Device.
Device Echoes	0x01	0x08	0xFE, 0xED, 0xFA, 0xCE, 0xCA, 0xFE, 0xBE and 0xEF	The Qorivva Device will echo the Message it received from the Host.
Start Address and Data Size Message	0x12	0x08	0x40, 0x00, 0x00, 0x00, 0x80, 0x00, 0x14 and 0x00	After the Password Message, the Host will transmit the Start Address and Data Size Message to the Qorivva Device (Start Address: 0x40000000, Data Size: 0x1400 and VLE mode).
Device Echoes	0x02	0x08	0x40, 0x00, 0x00, 0x00, 0x80, 0x00, 0x14 and 0x00	The Qorivva Device will echo the Message it received from the Host.
Data Message	0x13	0x08	Byte 0…Byte 7	After the Start Address and Data Size Message, the Host will transmit the Data of size defined in the Data Size field.
Device Echoes	0x03	0x08	Byte 0…Byte 7	The Qorivva Device will echo the Message it received from the Host. This process will repeat step 6 and 7 until all the data is transmitted to and received from the Qorivva Device. If the Qorivva Device has received the number of bytes as defined in the Data Size Field, the BAM will relinquish control to the application program by executing a jump command to the memory location defined by the Start Address Field.

7.3 Minimum Implementation of the Host Program

If we analyze these two ladder diagrams Figure 11 and Figure 12, it is clear that we only need three software routines to perform the actual transfer of application program to the Target system. Additionally, we need other software constructs and components to build this application. Let us look at the appended code fragment as shown in Figure 13.

In automatic baud rate detection mode, the WriteChar routine sends a synchronization byte to the BAM. This routine is appropriate because it only transmits a byte. The WriteVerifyChar routine transmits the password, start address, and data size. It reads the data from the RAM and verifies what it sends with the echoed byte, to ensure data integrity. If there is any disparity, this method will terminate the session immediately. Finally, WriteVerifyFileChar, this routine is similar to the WriteVerifyChar method with the exception that it reads the data from a file.

The routines shown in Figure 13 are applicable for transferring application program to the Target system via ESCI. If you modify these routines so that it can load one to eight bytes of data, and encapsulates the data in a CAN message, you have a delivery system over the CAN interface.

The BAM Host Specific Application Program has been created, using the above models, the Enhanced Serial Loader (eSL) using Visual C++ Express 2010. The eSL is a BAM specific application program on the Host system that understands the BAM Protocol. The eSL implements BAM protocol over ESCI, two CAN tools (iTAS and SysTech) and CAN tool using the EVB CAN port (Software is implemented using Green Hills MULTI). The eSL also implements Qorivva BAM emulation for testing BAM Host application. We will use the eSL as our Host program for this application note.

```
//Auto baud rate detection for the BAM
if (autobaudrate)
{
WriteChar(0);
Delay(0x0FFFFFF);
}
// Send password to BAM
status = WriteVerifyChar(password, 8, verbosity, "State 1: Password ");
if (status != 0)
{
return status;
}
// Send start address to BAM
status = WriteVerifyChar(address, 4, verbosity, "State 2: Address ");
if (status != 0)
{
return status;
}
// The number of data bytes to be sent to BAM
status = WriteVerifyChar(byte_count, 4,verbosity,"State 3: Byte Count ");
if (status != 0)
{
return status;
}
// Send the SRAM image file to the BAM
status = WriteVerifyFileChar(SRAM_FNAME,verbosity,"State 4: SRAM Image");
if (status != 0)
{
return status;
}
```

Figure 13. Transfer SRAM Image Routine

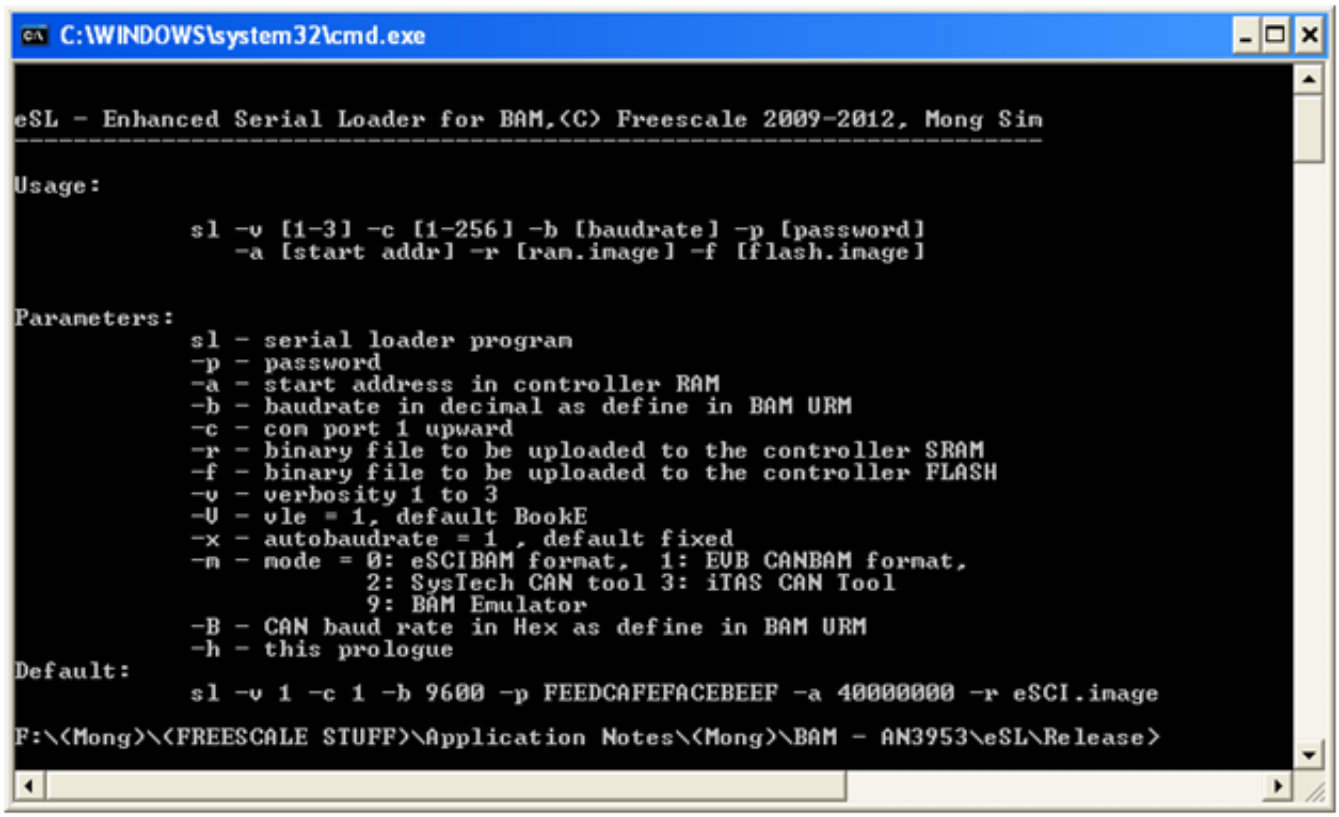


Figure 14. Enhanced Serial Loader Command Line Options

Figure 14 shows the command line option of eSL. For more information on eSL and a copy of eSL, please contact your Freescale Sales Representative.

7.4 Test the BAM Specific Applications

Before we test our implementations, we need to know how to setup the system for test. Here are the steps you need for different configuration, please see Table 15. This example uses the MPC5634M device.

Table 15. Test Setup

Configuration	Description	Remarks
ESCI	Connect the RS232 to the Host COM port (or USB port if you are using a USB-to-Serial cable) and Target system ESCI 0. Connect power to the EVB	All ESCI configurations BAM only supports ESCI 0
ESCI Fixed Baud Rate	The BAM ESCI baud rate for the MPC5634M is calculated using the following equation: Baud Rate = fsys / 832	The EVB used in this application note has an 8Mhz crystal, hence the baud rate is 9600 with one start bit and one stop bit.

Table continues on the next page...

Table 15. Test Setup (continued)

Configuration	Description	Remarks
Rate ESCI Automatic Baud	An additional pull down resistor is also required between the signal pin “EVTO” and “GND.” After installing this pull down resistor, reset the EVB. Use a debugger to verify if the SIU_RSR [ABR] bit is set.	Please see Table 5 for maximum and minimum baud rate supported.
eSL for ESCI	eSL –c 4 –b 9600 –p FEEDFACECAFEBEER – a 40000000 – r bamappl.mem eSL –x 1 –c 4 –p FEEDFACECAFEBEER – a 40000000 – r bamappl.mem	Starts eSL for ESCI fixed baud rate Starts eSL for ESCI automatic baud rate
CAN	Connect the CAN Tool to Host system USB port and the Target system CAN A port. Connect power to the EVB	All CAN configurations BAM only supports CAN A.
CAN Fixed Baud Rate	The BAM CAN baud rate for the MPC5634M is calculated using the following equation: Baud Rate = fsys / 40	The fixed baud rate for the CAN is 200bits/s.Refer to Table 6 for more detail.
CAN Automatic Baud Rate	An additional pull down resistor is also required between the signal pin “EVTO” and “GND.” After installing this pull down resistor, reset the EVB. Use a debugger to verify if the SIU_RSR [ABR] bit is set.	Refer to Table 5 for maximum and minimum baud rate supported.
eSL for CAN	eSL –c 4 –B 0307 –p FEEDFACECAFEBEER – a 40000000 – r bamappl.mem eSL –x 1 –c 4 –p FEEDFACECAFEBEER – a 40000000 – r bamappl.mem	Starts eSL for CAN fixed baud rate. The – B is the SysTech CAN bit rate setting. This will set the bit rate 200Kbit/s. Starts eSL for CAN automatic baud rate.

Table 16. Maximum and Minimum Detectable Baud Rates

fsys = fxtal [MHz]	Max baud rate for CAN (fsys/8)[bit/s]	Min CAN baud rate (fsys/(25 * 256) [bit/s]	Max baud rate for SCI (fsys/160) [bit/s]	Min baud rate for SCI (fsys/(16*2^16) [bit/s]
8	1M	1250	50K	7.6

On CAN fixed baud rate mode, the BAM configures the Target system (Qorivva device, MPC5634M) baud rate to 200Kb/s based on the equation provided in Table 16. This is correct; however, the actual configuration under the hood is a little more complex. It is important to know this so that you can try to use the same setting to setup your CAN tool if possible for better reception. The BAM divides the system clock frequency by 4 by setting the Prescaler Division Factor in the CAN control register to achieve a 2 MHz CAN clock. This 2 MHz clock is further divided down to 200Kb/s as shown in Table 17.

Table 17. CAN Bit Timing for Fixed Baud Rate

SYNC_SEG	TIME SEGMENT 1	TIME SEGMENT 2
1 Time Quanta	7 Time Quanta	2 Time Quanta
	PROSEG=4+1 PSEG1=1+1	PSEG2=1+1

After you have setup the systems based on the configuration in Table 15, open a command Window in the Host system. This application is built to run on Microsoft XP Windows Operating System. and Window 7. Click on the “Start” button on bottom left of the status bar and choose the option “Run…” A window will pop up and type in “cmd” and enter. A command window will appear. Start eSL with the following parameter as shown in Table 15. Power on the EVB and choose “2) Transfer SRAM Image Now!”

Figure 15 shows eSL has successfully uploaded the RAM Application to the EVB via ESCI interface and Figure 16 shows the upload via the CAN interface.

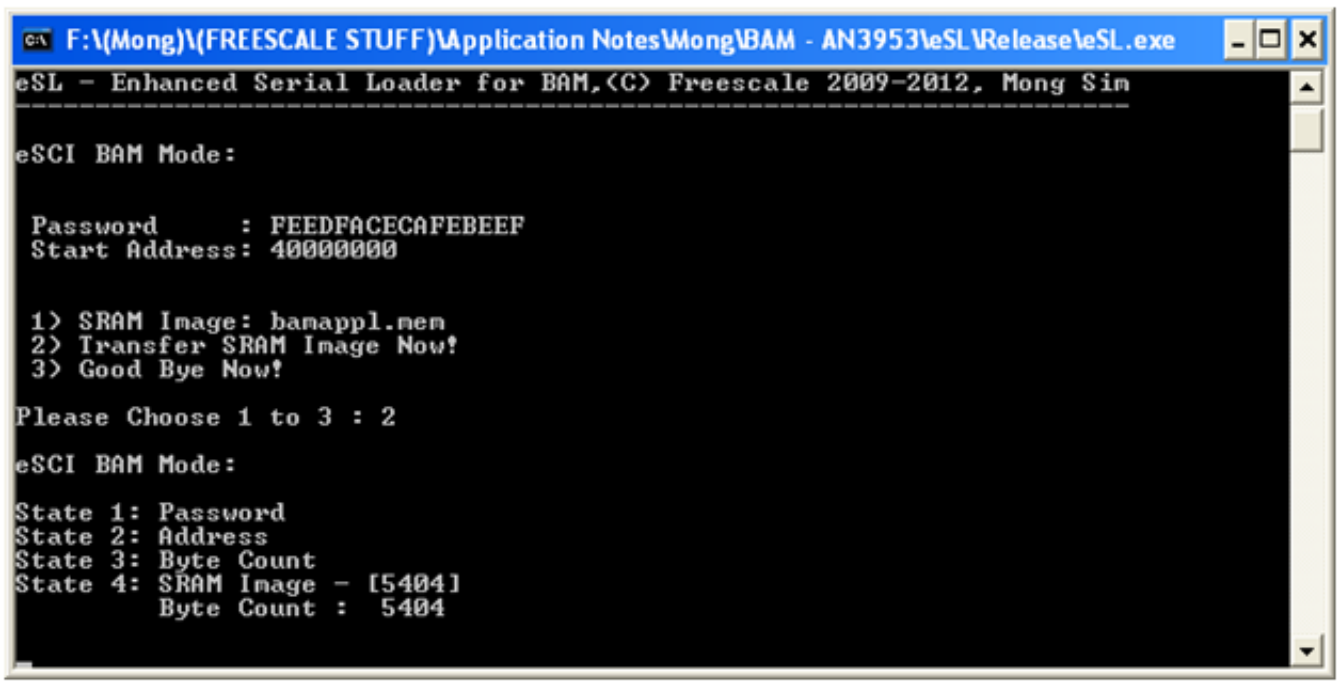


Figure 15. Uploading the RAM Application to the EVB via the ESCI

NOTE

If you are using a USB-to-Serial cable, the data transfer will not go any faster even if you increase the baud rate of the transfer (In Automatic Baud Rate Detection mode). This is due to the slow USB host polling rate. Depending on the Operating System that you are using, it can be as low as 8ms or 125Hz. There are ways to overcome this problem. One of the ways to overcome this problem is to transfer a block of data equal to the size of the USB buffer and read back the echoed block and compare for data integrity.

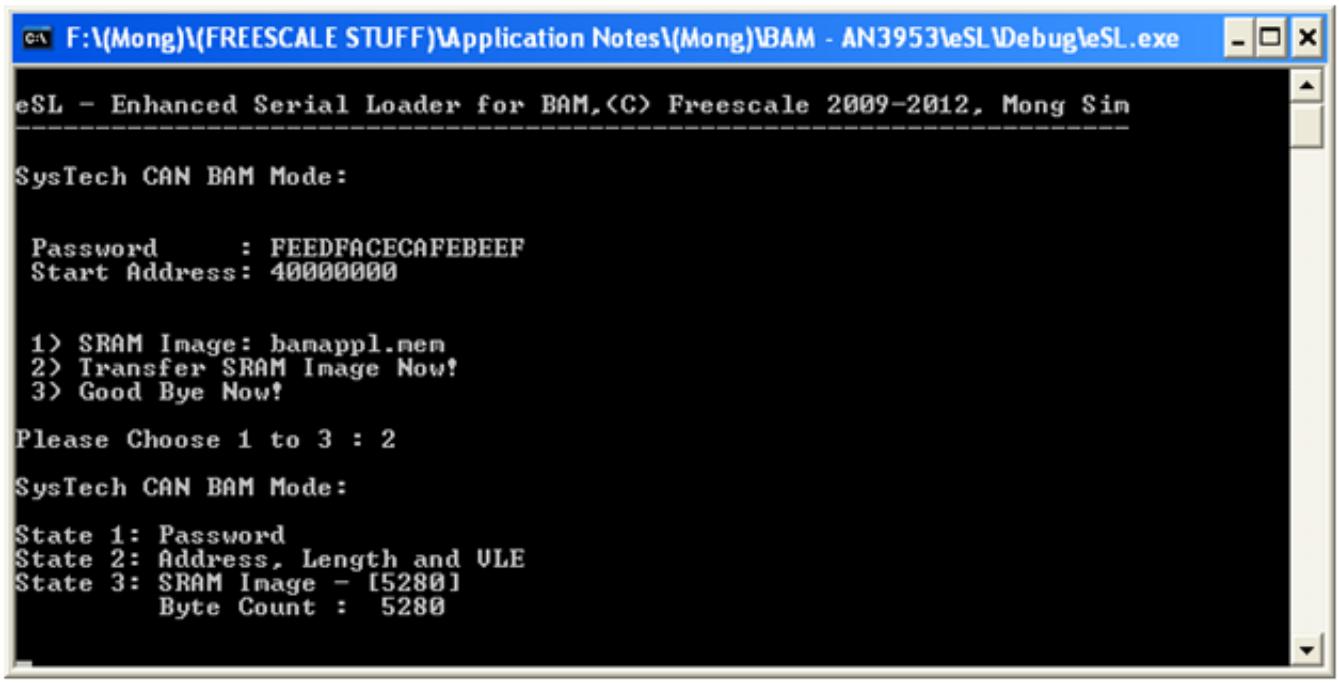


Figure 16. Uploading the RAM Application to the EVB via the CAN

8 Summary

Not all Qorivva devices support automatic baud rate detection and CAN interface. Also, in the MPC57XX Qorivva devices, a similar module known as the Boot Assist Flash (BAF) also provides the same BAM protocol for ESCI and CAN. Some Qorivva devices use FlexLinD UART mode instead of ESCI.

If you have any question, please contact your local FAE or visit our website at freescale.com for more information.

9 Appendix 1: BAM Qorivva Specific Application Source Code

MAIN.C

```
/*
 * LICENSE:
 * Copyright (c) 2012 Freescale Semiconductor
 *
 * Permission is hereby granted, free of charge, to any person
 * obtaining a copy of this software and associated documentation
 * files (the "Software"), to deal in the Software without
 * restriction, including without limitation the rights to use,
 * copy, modify, merge, publish, distribute, sublicense, and/or
 * sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following
 * conditions:
 *
 * The above copyright notice and this permission notice
 * shall be included in all copies or substantial portions
 * of the Software.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
```

```
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 *
 * Composed By: Sim, Mong Tee
 * Dated      : July 25, 2012
 * Compiler   : Green Hill Multi
 *
 * Objective  : This application program is intended for Freescale customer
 *              who one way or the other would like to tap the power of the
 *              BAM feature presents in the Freescale PPC products line
 *
 * Addendum   : This microcontroller specific program to be used
 *              in conjunction with the Serial Loader (VC++ Express) written
 *              by me. Application note for this application can be found at
 *              Freescale website or you can request from your area FAE
 *
 */

#include "boardport.h"

//-------------------------------------------------------------------------
// MAIN
//-------------------------------------------------------------------------
void main(void)
{
    uint32_t stmElapseTime      = 0;
    uint32_t stmStartTime       = 0;
    uint8_t  cntLED             = 0;

    STM_init(STM_CLOCK);                //setup the clock for the system time module
    LED_init();                         //init the GPIO that drives the LED

    while(1)
    {
        while(stmElapseTime > (0.25*STM_sec))   // set a quarter second
        {
            stmStartTime = STM_read();         // start new time for next
            LED_count(cntLED++);               // LEDs count
            stmElapseTime = 0;
            cntLED &= 0x0F;// we only have 4 LEDs
        }

        stmElapseTime = STM_elapse(stmStartTime);
    }

}
```

MLED.C

```
#include "boardport.h"

#ifdef __cplusplus
extern "C" {
#endif
//-------------------------------------------------------------------------
// LED_init function
//-------------------------------------------------------------------------
void LED_init(void)
{

    SIU.PCR[PLED0].R = GPIO_OP;              // init all the GPIO
    SIU.PCR[PLED1].R = GPIO_OP;             // that are connected to the
    SIU.PCR[PLED2].R = GPIO_OP;             // LEDs on the EVB
    SIU.PCR[PLED3].R = GPIO_OP;
```

```
    SIU.GPDO[PLED0].R = 1;
    SIU.GPDO[PLED1].R = 1;
    SIU.GPDO[PLED2].R = 1;
    SIU.GPDO[PLED3].R = 1;
}


//-----------------------------------------------------------------------
// LED_count function
//-----------------------------------------------------------------------
void LED_count(uint8_t state)
{
    SIU.GPDO[PLED0].B.PDO = (state & 0x01) ? 0 : 1;   // On or Off individual
    SIU.GPDO[PLED1].B.PDO = (state & 0x02) ? 0 : 1;   // based on the value in
    SIU.GPDO[PLED2].B.PDO = (state & 0x04) ? 0 : 1;   // state
    SIU.GPDO[PLED3].B.PDO = (state & 0x08) ? 0 : 1;
}

…


#ifdef __cplusplus
}
#endif

STM.C
#include "boardport.h"

#ifdef __cplusplus
extern "C" {
#endif



//-----------------------------------------------------------------------
// STM_init
//-----------------------------------------------------------------------
void STM_init(uint32_t STM_CR)
{
    STM_CR = 0x00000001 | (STM_CR << 8);    //do not stop in debug mode
  //STM_CR = 0x00000003 | (STM_CR << 8);    //stop in debug mode
    STM.CR.R = STM_CR;
}


//-----------------------------------------------------------------------
// STM_read
//-----------------------------------------------------------------------
uint32_t STM_read(void)
{
    return STM.CNT.R;   //read current counter value
}


//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// STM_elapse - Non-block function
//-----------------------------------------------------------------------
uint32_t STM_elapse(uint32_t BeginTime)
{
    uint32_t CurrTime  = 0;
    uint32_t DiffTime  = 0;
    CurrTime = STM.CNT.R;
    if (BeginTime > CurrTime)
    {
        DiffTime = (0xFFFFFFFF - BeginTime) + CurrTime; // If overflow
    }
    else
    {
```

```
        DiffTime = CurrTime - BeginTime;                 // within range
    }

    return DiffTime;                                    // return the elapse time

}

…

#ifdef __cplusplus
}
#endif

BOARDPORT.H
#ifndef __BOARDPORT_H_
#define __BOARDPORT_H_

#include <string.h>

//-----------------------------------------------------------------------------
// System Clock setting resident in init.s

#define MHz             1000000
#define SYSCRYSTAL      8*MHz
#define SYSCLK          SYSCRYSTAL

#define MPC5634M_EVB    1
#define MPC5674F_EVB    0
#define MPC5643L_EVB    0

#define ESCI_PORT       0
#define CAN_PORT        0
#define PIT_MOD         0


#include "typedefs.h"
#include "typedefs_UINT8.h"

#if MONACO_EVB
#include "MPC5634M_MLQB80.h"
#endif

#include "stm.h"
#include "mled.h"

#if ESCI_PORT
#include "serial.h"
#endif

#if CAN_PORT
#include "can.h"
#endif

#if PIT_MOD
#include "pit.h"
#endif


#endif // __BOARDPORT_H_

MLED.H
#ifndef __MLED_H__
#define __MLED_H__

#include <stdint.h>

#if MPC5643L_EVB

#define GPIO_OP 0x028c  //as output
#define GPIO_IP 0x018c  //as input
```

```
    #define PLED052
    #define PLED153
    #define PLED254
    #define PLED355

#endif

#if MPC5674F_EVB

#define GPIO_OP 0x028c  //as output
#define GPIO_IP 0x018c  //as input

    #define PLED0179
    #define PLED1180
    #define PLED2181
    #define PLED3182

#endif



#if MPC5634M_EVB

#define GPIO_OP 0x028c  //as output
#define GPIO_IP 0x018c  //as input

    #define PLED0188
    #define PLED1189
    #define PLED2190
    #define PLED3191

#endif


#define DLED0       0
#define DLED1       1
#define DLED2       2
#define DLED3       3

#define DLED_ON     1
#define DLED_OFF    2


void    LED_init    (void);
void    LED_count   (uint8_t state);
void    LED_toggle  (uint8_t whichLED);
void    LED_onoroff (uint8_t whichLED,uint8_t ledState);

#endif


STM.H
#ifndef __STM_H__
#define __STM_H__

#include <stdint.h>
//---------------------------------------------------------------------------
// STM API
//---------------------------------------------------------------------------
#define STM_CLOCK               SYSCLK
#define STM_START               1
#define STM_ELAPSE              0
#define STM_us                  1
#define STM_ms                  1000
#define STM_sec                 1000000

#define STM_VECT200             200
#define STM_VECT201             201
```

```
#define STM_IRQ0                0
#define STM_IRQ1                1
#define STM_IRQ2                2
#define STM_IRQ3                3

void     STM_init               (uint32_t STM_CR);
uint32_t STM_read               (void);
uint32_t STM_elapse             (uint32_t BeginTime);
void     STM_delay              (uint32_t delay);

void     STM_clearCCR           (uint32_t ch);
void     STM_clearAllCCR        (void);
void     STM_setCCR             (uint32_t ch);
void     STM_setAllCCR          (void);
void     STM_setCMP             (uint32_t ch,uint32_t ticks);
void     STM_setAllCMP          (uint32_t ch0,uint32_t ch1,uint32_t ch2,uint32_t ch3);
void     STM_setIRQ             (uint32_t ch);
void     STM_setAllIRQ          (void);

void     STM_initISR0           (uint32_t nextTrigger0);
void     STM_initISR1           (uint32_t nextTrigger1);
void     STM_initISR2           (uint32_t nextTrigger2);
void     STM_initISR3           (uint32_t nextTrigger3);
void     STM_initISR123         (uint32_t nextTrigger1,uint32_t nextTrigger2,uint32_t
nextTrigger3);

#endif                  // _STM_H_
```

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4674
Revision 1, 06/2013