Introduction to Pre-Boot Loader
Supported by QorIQ Processors

FTF-NET-F0152

Zhongcai Zhou | Application Engineer

A P R . 2 0 1 4
Introduction

• What does Pre-Boot Loader (PBL) do?
  – Device configuration
  – Initialization before the core fetches instructions

• History
  – Before PBL, how did we do these?
  – MPC8548
    • Device configuration
      • Pins strapping
    • Initialization
      • I2C Boot sequencer

• Improvements
  – Greatly reduce the number of pin strapping
  – Expand the sources of boot sequencer from I2C only to I2C, eSPI, eSDHC, NAND flash, NOR flash
PBL in the Power Up Sequence

/poreset

/hreset

NAND preload
(if RCW from NAND)

RCW load

PLL lock

PBI load

cfg_rcw_src
sampling

clock switching
/hreset release

core0 executes
Description of PBL Functionality

• PBL RCW phase
  – PBL runs with SYSCLK

  – Device samples cfg_rcw_src at the rising edge of poreset_b
    ▪ Determine the source of the RCW

  – PBL fetches the RCW from the source, configure the device
    ▪ It checks the format of RCW
    ▪ It is written to DCFG_RCWSR0-15

  – PLLs start to work and lock according to the ratios specified in RCW

  – Error reports to DCFG_RSTRQPBLSR
Description of PBL Functionality (continued)

• PBL PBI phase
  – PBL switches to platform clock
  – PBL checks RCW[PBI_SRC]
    ▪ If PBI is disabled, then PBL is done
    ▪ If PBI is enabled, proceed to fetch PBI data from the source
  – PBL finishes the PBI, release the core0 to fetch instruction if RCW[BOOT_HO] is 0
    ▪ The boot code location is specified in RCW[BOOT_LOC] for Power Architecture based device
  – Error reports to DCFG_RSTRQPBL
  – For both RCW and PBI phase, if there is any error, the boot stops and /RESET_REQ is asserted
# CFG_RCW_SRC for Device with eLBC

**cfg_rcw_src pin strapping**

<table>
<thead>
<tr>
<th>Functional Signals</th>
<th>Reset Configuration Name</th>
<th>Value (Binary)</th>
<th>RCW Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>LGPL0/LFCLE, LGPL1/LFALE, LGPL2/LOE/LFRE, LGPL3/LFWP, LGPL5</td>
<td><strong>cfg_rcw_src[0:4]</strong></td>
<td>0_0000</td>
<td>I²C1 normal addressing (supports ROMs up to 256 bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_0001</td>
<td>I²C1 extended addressing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_0010</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_0011</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_0100</td>
<td>SPI 16-bit addressing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_0101</td>
<td>SPI 24-bit addressing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_0110</td>
<td>eSDHC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_0111</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_1000</td>
<td>eLBC FCM (NAND flash, 8-bit small page)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_1001</td>
<td>eLBC FCM (NAND flash, 8-bit large page)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_1010</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_1011</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_1100</td>
<td>eLBC GPCM (NOR flash, 8-bit)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_1101</td>
<td>eLBC GPCM (NOR flash, 16-bit)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_1110</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0_1111</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1_0000 - 1_1011</td>
<td>Hard-coded RCW options (See Hard Coded RCW Options, for more information.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1_1100 - 1_1111</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
RCW for Device with eLBC

- 512-bits (64-bytes)
  - Bits related to PBL or booting
  - RCW[192:195] -- PBI_SRC

  0000 I²C1 normal addressing (up to 256 byte ROMs)
  0001 I²C1 extended addressing
  0100 SPI 16-bit addressing
  0101 SPI 24-bit addressing
  0110 SD/MMC
  1000 eLBC FCM 8-bit small page NAND Flash
  1001 eLBC FCM 8-bit large page NAND Flash
  1100 eLBC GPCM 8-bit
  1101 eLBC GPCM 16-bit
  1111 disabled
RCW for Device with eLBC (continued)

- RCW[196:200] -- BOOT_LOC
  0_0000 PCIe1
  0_0001 PCIe2
  0_0010 PCIe3
  0_1000 sRIO1
  0_1001 sRIO2
  1_0000 Memory complex 1
  1_0001 Memory complex 2
  1_0100 Interleaved memory complexes
  1_1000 eLBC FCM 8-bit small page NAND Flash
  1_1001 eLBC FCM 8-bit large page NAND Flash
  1_1100 eLBC GPCM 8-tb
  1_1101 eLBC GPCM 16-bit
Devices with IFC

- CFG_RCW_SRC[0:8] (based on T4240)

  It needs more bits to specify the IFC NOR/NAND options
  NOR: port size/address shift/AVD
  NAND: port size/page/block/BBI/ECC
  This is due to the difference between IFC and eLBC

- If RCW/PBI is not from IFC, but BOOT_LOC is from IFC, then
  RCW[IFC_MODE] determines the IFC configuration
Restriction on CFG_RCW_SRC, PBI_SRC, BOOT_LOC

- RCW and pre-boot initialization data must be loaded from the same non-volatile memory device
  - In the design, PBI_SRC is ignored. PBI is always loaded from CFG_RCW_SRC
- At most, only one given IFC option can be used for CFG_RCW_SRC, PBI_SRC, and BOOT_LOC

<table>
<thead>
<tr>
<th>CFG_RCW_SRC</th>
<th>PBI_SRC</th>
<th>BOOT_LOC</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2C</td>
<td>I2C</td>
<td>NAND or NOR</td>
<td>OK</td>
</tr>
<tr>
<td>eSPI</td>
<td>eSPI</td>
<td>NAND or NOR</td>
<td>OK</td>
</tr>
<tr>
<td>eSDHC</td>
<td>eSDHC</td>
<td>NAND or NOR</td>
<td>OK</td>
</tr>
<tr>
<td>NAND</td>
<td>NAND</td>
<td>NOR</td>
<td>No</td>
</tr>
<tr>
<td>I2C</td>
<td>eSPI</td>
<td>NAND or NOR</td>
<td>eSPI is ignored</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No error</td>
</tr>
</tbody>
</table>
PBL checks the correctness of data structure

- If not detecting preamble 0xA5A5, it reports error code 0x70
- If ACS=1, it reports 0x71 (ACS is logic 1 during RCW)
- If BYTE_CNT!=000000, it reports 0x72(Byte count is not 64 for RCW)
- If SYS_ADDR!=0x0e0100, it reports 0x74(Addr is in protected space)
  - SYS_ADDR for RCW points to DCFG_RCWSR0

For new Layerscape devices, DCFG_RCWSR0 address is different from Power based device
### Data Format: PBI Format

**Address/Data Pair, Write (Exception: PBL Command)**

<table>
<thead>
<tr>
<th>First Pre-Boot Initialization Command (optional)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS</td>
<td>BYTE_CNT</td>
<td>CONT=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[23-16]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[15-8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[7-0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.......................</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE N-1 (up to 63)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Second Pre-Boot Initialization Command (optional)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS</td>
<td>BYTE_CNT</td>
<td>CONT=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[23-16]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[15-8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[7-0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.......................</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE N-1 (up to 63)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
End of PBL Data Structure

<table>
<thead>
<tr>
<th>End Command (required, special CRC Check command with CONT=0)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS=0, BYTE_CNT = 00100 (4 bytes), CONT=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[23:16] = 0x13^2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[15:8] = 0x80^2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_ADDR[7:0] = 0x40^2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **CONT=0, end command**
- **PBL reports 0x79(Invalid End command error) if it detects the followings:**
  - BYTE_CNT != 4
  - ACS==1
  - For Power Architecture based device, SYS_ADDR!=0x138040
ACS - Alternate Configuration Space

• If ACS=0, it access CCSR address space (16 Mbytes)

• If ACS=1, it access Alternate Configuration Space (16 Mbytes)
  - ALTCBARH, ALTCBARL : Defines base address
  - ALTCAR registers: Target

• By changing the alternate configuration space base address
  PBL can access the whole physical address space

4.4.6 Alternate configuration attribute register (LCC_ALTCAR)

Address: 0h base + 18h offset = 18h

<table>
<thead>
<tr>
<th>Bit</th>
<th>EN</th>
<th>-</th>
<th>TRGT_ID</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Reset</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Reset</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
CRC Check Command

• For Power Architecture based device, the last 8 bytes must be:
  0x08_138040 + 4-byte CRC
  0x08 means: ACS=0, BYTE_CNT=4, CONT=0
  0x138040 is “CRC check” command

• PBL command
  - SYS_ADDR equals to PBL CCSR address space, PBL treats this command entry as a PBL internal command.

  0x138000 is the PBL address space for Power Architecture based devices

For new Layerscape device, the CCSR address for PBL is different.
## All PBL Commands

<table>
<thead>
<tr>
<th>Command Name</th>
<th>SYS_ADDR (Power Architecture)</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flush</td>
<td>0x13_8000</td>
<td></td>
<td>Chases the previous write with a read from the same address.</td>
</tr>
<tr>
<td>CRC check</td>
<td>0x13_8040</td>
<td>Next 4 bytes are CRC value</td>
<td>Checks CRC for data blocks</td>
</tr>
<tr>
<td>Jump</td>
<td>0x13_8080</td>
<td>Target data address</td>
<td>PBL reads next data from the jump address</td>
</tr>
<tr>
<td>Wait</td>
<td>0x13_80C0</td>
<td>Number of PBL clock(platform clock/2)</td>
<td>Wait a number of cycles before reading next data</td>
</tr>
</tbody>
</table>
PBL Error Code

- PBL checks the correctness of PBL data format. It reports to DCFG_RSTRQPBLSR for any error
  - Preamble/Header errors
  - End command errors, including CRC error
  - Interface errors from I2C, eSPI, eSDHC, eLBC/IFC
- The register can be read out from JTAG to help debug if the device does not boot.
## PBL Error Code (continued)

<table>
<thead>
<tr>
<th>Error Code[0:6]</th>
<th>Error Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x01</td>
<td>I²C timeout while waiting for I2CSR[MIF] to assert</td>
</tr>
<tr>
<td>0x02</td>
<td>I²C lost arbitration</td>
</tr>
<tr>
<td>0x03</td>
<td>I²C did not receive ACK during address transmit</td>
</tr>
<tr>
<td>0x04</td>
<td>I²C SCL signal was stuck low at POR</td>
</tr>
<tr>
<td>0x05-0x0F</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x10</td>
<td>eSPI timeout while waiting for SPIE[DON] to assert</td>
</tr>
<tr>
<td>0x11-0x20</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x21</td>
<td>eLBC detects error in LTESR register, user can scan out LTESR upon receiving this error.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> In cases where LTESR fields BM, PAR, WP, or CS are set, the PBL will indicate a data transfer error from memory devices (encoding 0x78).</td>
</tr>
<tr>
<td>0x22</td>
<td>eLBC timeout error</td>
</tr>
<tr>
<td>0x23-0x3F</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x40</td>
<td>eSDHC: Err_Reset_1</td>
</tr>
<tr>
<td></td>
<td>Indicates that eSDHC has not completed its soft-reset sequence. SYSCTL:RSTA (offset: 0x02C, mask: 0x0100_0000) did not clear within 1 second. It should clear when eSDHC completes its 'reset' sequence.</td>
</tr>
<tr>
<td>0x41</td>
<td>eSDHC: Err_Reset_2</td>
</tr>
<tr>
<td></td>
<td>Indicates that eSDHC has not completed its soft-reset sequence. SYSCTL:RSTA (offset: 0x02C, mask: 0x0100_0000) did not clear within 1 second. It should clear when eSDHC completes its 'reset' sequence.</td>
</tr>
</tbody>
</table>

Many, many eSDHC error entries
## PBL Error Code (continued)

<table>
<thead>
<tr>
<th>eSDHC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4F</td>
<td>eSDHC: Reserved</td>
</tr>
<tr>
<td>0x50-0x6F</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x70</td>
<td>No preamble is detected</td>
</tr>
<tr>
<td>0x71</td>
<td>ACS is logic 1 during RCW</td>
</tr>
<tr>
<td>0x72</td>
<td>Byte count is not 64 for RCW or 4 for PBL commands or 1/2/4/8/16/32/64 for pre-boot initialization commands</td>
</tr>
<tr>
<td>0x73</td>
<td>Misaligned address</td>
</tr>
<tr>
<td>0x74</td>
<td>Address is in protected space</td>
</tr>
<tr>
<td>0x75</td>
<td>CRC Error</td>
</tr>
<tr>
<td>0x76</td>
<td>Time out counter expires</td>
</tr>
<tr>
<td>0x77</td>
<td>Unrecognized PBL commands, including unrecognized offset and invalid parameter</td>
</tr>
<tr>
<td>0x78</td>
<td>Data Transfer error from memory devices</td>
</tr>
<tr>
<td>0x79</td>
<td>Invalid End command error</td>
</tr>
<tr>
<td>0x7A</td>
<td>Invalid RCW or pre-boot initialization word source encoding</td>
</tr>
<tr>
<td>0x7B-0x7F</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
ICM: Interface Control Module
What Does ICM (Interface Control Module) Do?

- Each interface has its own ICM (except eLBC GPCM or IFC NOR)

- It has a state machine to go through the register initialization

- Initiate the read process and polling the status until read data is back

- Passing the data to PBL CCL

- If there is any error, it reports the error back to PBL CCL

- All the register value will be reset back to their POR value when PBL has completed
ICM for NAND

- ICM works with power on reset machine and NAND controller to pre-fetch data from NAND device

reset FSM
insn NAND Ctrl
to pre-load data

ICM automatically
load more pages
if needed

NAND preload

ICM reads RCW
from internal buffer

PLL lock

PBI load

Preload boot code
from NAND from fresh page

cfg_rcw_src sampling NAND

PBL FSM starts

clock switching /hreset release

core0 executes
- PBL works with I2C1
- PBL supports I2C devices
  - Normal 16-bit Addressing
  - Extended 24-bit Addressing
- When standard I2C EEPROMs are used, it is assumed that each EEPROM holds 256 bytes of data. After 256 bytes have been received, the PBL will increment the slave address and access the next EEPROM. A maximum of eight standard I2C EEPROMs may be used with the PBL
- Extended I2C EEPROMs have variable sizes. However, the I2C controller increments the slave address after 64 Kbytes have been received
• SPI_CS0 is used

• Only SPI Mode 0 is used
  eSPI memory device must also support mode 0

• A value of 0x03 is used for the read command. The SPI slave device must decode 0x03 as a read command
How Is the Speed of I2C, eSPI and eSDHC determined during RCW and PBI?

- Target speed:
  - I2C: 100K
  - eSPI: 2Mhz
  - eSDHC: Identification process is 400Khz
    Normal phase is 25Mhz for SD, 20Mhz for MMC

- This is based on design assumption of
  SYSCLK 200MHz for RCW phase, CCB 2GHz for PBI phase
  The real speed is scaled according to the real SYSCLK/CCB clock

For example, SYSCLK 100Mhz, CCB=800Mhz
- I2C: RCW=100Mhz/200Mhz *100KHz=50KHz
- PBI =800Mhz/2000Mhz *100KHz=40Khz
PBL Application Examples

- ✔️ Errata Workaround
- ✔️ Boot from eSPI, eSDHC and NAND
Errata Workaround: Example

A-006559: Configuration is required for proper e500mc operation
Affects: GEN

Description: In order to ensure data integrity between the core and the platform (the core running asynchronously with the CoreNet platform), special internal registers must be set. These internal registers are part of the debug configuration register address space (DCSR).

Impact: Data and instruction corruption is possible if the workaround is not followed

Workaround: Set the internal register bits [16:19] for each core before bringing the cores out of reset. The internal register of Core0 is at offset 0x2_1008; the internal register of Core1 is at offset 0x2_1028; the internal register of Core2 is at offset 0x2_1048; and the internal register of Core3 is at offset 0x2_1068. This configuration may be done using PBI code.

An example for PBI initialization:

1. Write a value of 0x0000_0000 to ALTCBARH register at offset 0x10
2. Write a value of 0x0000_0000 to ALTCBARL register at offset 0x14
3. Write a value of 0x81d0_0000 to ALTCAR register at offset 0x18, this step set the target ID to DCSR
4. Write a value of 0x0000_f000 to the internal register at offset 0x2_1008, set configuration for Core0
5. Write a value of 0x0000_f000 to the internal register at offset 0x2_1028, set configuration for Core1
6. Write a value of 0x0000_f000 to the internal register at offset 0x2_1048, set configuration for Core2
7. Write a value of 0x0000_f000 to the internal register at offset 0x2_1068, set configuration for Core3

Fix plan: No plans to fix
QCS Tool (V3.0.4) – Steps to Implement Example 2

- File -> New -> QorIQ Configuration Project
- Fill the “project name”, click Next
- Choose Processor and Silicon Revision, click Next
- Select “PBL – Preboot Loader RCW configuration”
  Click Next
- Choose “Import configuration from an existing PBL file” *(if starting from scratch, choose “Use RCW Hard-coded configuration” as update field as needed)*
  Use “Browse” to select the file
Choose “File Format”, click Finish.
QCS Tool – Steps to Implement Example 2 (continued)

• Project Explorer->*project name* (pbl-demo)
• Components -> PBL:PBL
• On the right side, Choose the appropriate items to modify
  (1) Boot Configuration-> PBI_SRC: 0b1101 – eLBC GPCM 16b NOR flash
QCS Tool – Steps to Implement Example 2 (continued)

Click here
QCS Tool – Steps to Implement Example 2 (continued)

- Add CCSR.Data (plain 4-bytes) to PBI Data input
  - Offset: 0x00000

PBI Data definition. Use this property to add additional PBI commands into the generated PBI data. If a CRC calculation is desired in any place of the PBI data the “PBI_CRC_DATA” control string (after the CRC command header) could be used to let the tool calculate the CRC value, e.g. “09138040”
Add PBI Data

1. Offset 0x000014, Data 0x00000000, click add
   - Click
2. Offset 0x000018, Data 0x81d00000, click add
3. Choose ACS Data (4 Byte)
   - Offset 0x021008, Data 0x00000f00
4. Repeat (3) with offset 0x021028
5. Repeat (3) with offset 0x021048
6. Repeat (3) with offset 0x021068
7. Click Apply to save the data
Choose the Output Format

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBI_SRC [192-195]</td>
<td>0b0000 - I2C1 Normal Addressing</td>
</tr>
<tr>
<td>BOOT_LOC [196-200]</td>
<td>0b11101 - eLBC GPCM 16b NOR Flash</td>
</tr>
<tr>
<td>BOOT_HO [201]</td>
<td>0b0 - All cores except core 0 in hol...</td>
</tr>
<tr>
<td>SB_EN [202]</td>
<td>0b0 - Secure boot is not enabled</td>
</tr>
</tbody>
</table>

**Clocking Configuration**

**Memory and High-Speed I/O**

**General Purpose Information**

**Pin Multiplexing Configuration**

**Group A Pin Configuration**

**Group B Pin Configuration**

**PBI Data**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBI Data input</td>
<td>(string list)</td>
</tr>
</tbody>
</table>

**PBL Data**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Output Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>XXD Object Dump</td>
</tr>
</tbody>
</table>

- S Record
- Source Code
- Text Table RCW Format
- U-Boot Commands
- eDINK32 Commands
- Hex String
- RCW[0:511] CW JTAG Config
- RCW[0:511] Hex String
- Binary
Generate Output

1. Choose ProcessExpert.pe
2. Right click, choose “Generate Processor Expert Code”
3. Output workspace/project/Generated_code/PBL.pbl
PBL Application Boot from eSPI, eSDHC and NAND

- PBL-based devices
  - BOOT_LOC: no options for boot from eSPI or eSDHC
- Solution
  - Use PBL to write the image. The whole image is written with PBI
Steps to Produce PBL Image for U-boot for eSPI

1. After creating a QorIQ Configuration Project, the project appears in the Project Panel view.
2. Find “Import” tab in window "Component Inspector", select "Input Format" value to be "RCW[0:511] U-Boot CCSR Dump"
   - Copy/paste RCW data from u-boot to Input data
   - Use Browse if RCW is saved in a file
3. Change "fe8" in the second row to "580"
   This enables PBI from eSPI and BOOT_LOC is Memory Complex 1
4. Click “Import”
Steps to Produce PBL Image for U-boot for eSPI (continued)

Linux
1. make ARCH=powerpc CROSS_COMPILE=powerpc-
linux-gnu- P3041DS_SPIFLASH
2. xxd u-boot.bin > u-boot.xxd

QCS Tool
1. Click button in "Value" column of row "PBI Data input"
   "PBI Data input" view will appears
2. Paste commands in the right into text field
   Configure CPC as 1M SRAM, u-boot image will be written by PBI to CPC
3. Add u-boot image as PBI command
   - Select "ACS File (XXD Object Dump)"
   - Change Offset to "f80000"
   - Click “Browse" to select the file "u-boot.xxd" produced above
   - Click "Add", 
     Content of the "u-boot.xxd" will be pasted
4. Paste "09138000 00000000" and "091380c0 00000000" at the end.
   flush
   wait
5. Click "Apply".
Steps to Produce PBL Image for U-boot for eSPI (continued)

• Find and click "Generate Processor Expert Code" in menu "Project", after it finished, click "Generated_Code" in "Project Panel" window and "PBL1.pbl" appears, find the file "PBL1.pbl" in workspace of this project

• Linux
  xxd -r PBL1.pbl > u-boot.pbl

• Burn u-boot.pbl to eSPI to boot
  => loadb 1000000 (or tftp 100000 u-boot.pbl)
  => sf probe 0
  => sf erase 0 100000
  => sf write 1000000 0 $filesize
Introducing The QorIQ LS2 Family

Breakthrough, software-defined approach to advance the world’s new virtualized networks

**New, high-performance architecture built with ease-of-use in mind**
Groundbreaking, flexible architecture that abstracts hardware complexity and enables customers to focus their resources on innovation at the application level

**Optimized for software-defined networking applications**
Balanced integration of CPU performance with network I/O and C-programmable datapath acceleration that is right-sized (power/performance/cost) to deliver advanced SoC technology for the SDN era

**Extending the industry’s broadest portfolio of 64-bit multicore SoCs**
Built on the ARM® Cortex®-A57 architecture with integrated L2 switch enabling interconnect and peripherals to provide a complete system-on-chip solution
High performance cores with leading interconnect and memory bandwidth
- 8x ARM Cortex-A57 cores, 2.0GHz, 4MB L2 cache, w Neon SIMD
- 1MB L3 platform cache w/ECC
- 2x 64b DDR4 up to 2.4GT/s

A high performance datapath designed with software developers in mind
- New datapath hardware and abstracted acceleration that is called via standard Linux objects
- 40 Gbps Packet processing performance with 20Gbps acceleration (crypto, Pattern Match/RegEx, Data Compression)
- Management complex provides all init/setup/teardown tasks

Leading network I/O integration
- 8x1/10GbE + 8x1G, MACSec on up to 4x 1/10GbE
- Integrated L2 switching capability for cost savings
- 4 PCIe Gen3 controllers, 1 with SR-IOV support
- 2 x SATA 3.0, 2 x USB 3.0 with PHY

Unprecedented performance and ease of use for smarter, more capable networks
See the LS2 Family First in the Tech Lab!

4 new demos built on QorIQ LS2 processors:

- Performance Analysis Made Easy
- Leave the Packet Processing To Us
- Combining Ease of Use with Performance
- Tools for Every Step of Your Design