Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Obsolete
Core Processor	ARM926EJ-S
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	454MHz
Co-Processors/DSP	Data; DCP
RAM Controllers	DRAM
Graphics Acceleration	No
Display & Interface Controllers	LCD, Touchscreen
Ethernet	-
SATA	-
USB	USB 2.0 + PHY (1)
Voltage - I/O	2.0V, 2.5V, 2.7V, 3.0V, 3.3V
Operating Temperature	-40°C ~ 85°C (TA)
Security Features	Cryptography, Hardware ID
Package / Case	128-LQFP
Supplier Device Package	128-LQFP (14x14)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mcimx233cag4b

Table 5-189. HW_ICOLL_INTERRUPT84 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:5	**RSRVD1**	RO	0x0	Always write zeroes to this bitfield.
4	**ENFIQ**	RW	0x0	Set this to 1 to steer this interrupt to the non-vectored FIQ line. When set to 0 the interrupt will pass through the main IRQ FSM and priority logic. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
3	**SOFTIRQ**	RW	0x0	Set this bit to one to force a software interrupt. NO_INTERRUPT = 0x0 turn off the software interrupt request. FORCE_INTERRUPT = 0x1 force a software interrupt
2	**ENABLE**	RW	0x0	Enable the interrupt bit through the collector. DISABLE = 0x0 Disable ENABLE = 0x1 Enable
1:0	**PRIORITY**	RW	0x0	Set the priority level for this interrupt, 0x3 is highest, 0x0 is lowest (weakest). LEVEL0 = 0x0 level 0, lowest or weakest priority LEVEL1 = 0x1 level 1 LEVEL2 = 0x2 level 2 LEVEL3 = 0x3 level 3, highest or strongest priority

DESCRIPTION:

This register provides a mechanism to specify the priority associated with an interrupt bit. In addition, this register controls the enable and software generated interrupt. WARNING: Modifying the priority of an enabled interrupt may result in undefined behavior. You should always disable an interrupt prior to changing its priority.

EXAMPLE:

```
HW_ICOLL_INTERRUPT84_SET(0,0x00000001);
```

5.4.95 Interrupt Collector Interrupt Register 85 Description

This register provides a mechanism to specify the priority level for an interrupt source. It also provides an enable and software interrupt for each one, as well as security designation.

HW_ICOLL_INTERRUPT85	0x670
HW_ICOLL_INTERRUPT85_SET	0x674
HW_ICOLL_INTERRUPT85_CLR	0x678
HW_ICOLL_INTERRUPT85_TOG	0x67C

Table 5-190. HW_ICOLL_INTERRUPT85

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSRVD1																											ENFIQ	SOFTIRQ	ENABLE	PRIORITY	

EXAMPLE:

```
pCurCmd = (hw_apbh_chn_cmd_t *) HW_APBH_CHn_CURCMDAR_RD(1); // read the whole register, since there
is only one field
pCurCmd = (hw_apbh_chn_cmd_t *) BF_RDn(APBH_CHn_CURCMDAR, 1, CMD_ADDR); // or, use multi-register
bitfield read macro
pCurCmd = (hw_apbh_chn_cmd_t *) HW_APBH_CHn_CURCMDAR(1).CMD_ADDR; // or, assign from bitfield of
indexed register's struct
```

10.5.13 APBH DMA Channel 1 Next Command Address Register Description

The APBH DMA Channel 1 Next Command Address register contains the address of the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to 1 in the DMA command word to process command lists.

HW_APBH_CH1_NXTCMDAR 0x0C0

Table 10-28. HW_APBH_CH1_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 10-29. HW_APBH_CH1_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for channel 1.

DESCRIPTION:

APBH DMA Channel 1 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 1 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
HW_APBH_CHn_NXTCMDAR_WR(1, (reg32_t) pCommandTwoStructure); // write the entire register, since
there is only one field
BF_WRn(APBH_CHn_NXTCMDAR, 1, (reg32_t) pCommandTwoStructure); // or, use multi-register bitfield
write macro
HW_APBH_CHn_NXTCMDAR(1).CMD_ADDR = (reg32_t) pCommandTwoStructure; // or, assign to bitfield of
indexed register's struct
```

10.5.14 APBH DMA Channel 1 Command Register Description

The APBH DMA Channel 1 command register specifies the cycle to perform for the current command chain item.

HW_APBH_CH1_CMD 0x0D0

DESCRIPTION:

APBX DMA Channel 12 is controlled by a variable sized command structure. This register points to the command structure currently being executed.

EXAMPLE:

```
Empty example.
```

11.5.91 APBX DMA Channel 12 Next Command Address Register Description

The APBX DMA Channel 12 next command address register points to the next multiword command to be executed. Commands are threaded on the command address. Set CHAIN to one to process command lists.

HW_APBX_CH12_NXTCMDAR 0x650

Table 11-184. HW_APBX_CH12_NXTCMDAR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
CMD_ADDR																															

Table 11-185. HW_APBX_CH12_NXTCMDAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**CMD_ADDR**	RW	0x00000000	Pointer to next command structure for Channel 12.

DESCRIPTION:

APBX DMA Channel 12 is controlled by a variable sized command structure. Software loads this register with the address of the first command structure to process and increments the Channel 12 semaphore to start processing. This register points to the next command structure to be executed when the current command is completed.

EXAMPLE:

```
Empty example.
```

11.5.92 APBX DMA Channel 12 Command Register Description

The APBX DMA Channel 12 command register specifies the cycle to perform for the current command chain item.

HW_APBX_CH12_CMD 0x660

If a different LOWPOWER_CONTROL bit is set to 1 while in one of the low-power modes, or on clearing of the original bit to 0, the memory controller exits the current low-power mode. There will be at least a 15 cycle delay before the memory controller is fully operational or enters the new low-power mode.

NOTE: There is a deadlock possibility that exists when using the manual low-power mode entry. If a read cycle from the ARM core occurs to the DRAM when a manual low-power mode is active, the ARM cycle does not complete. There is no other device within the SOC that can deactivate the low-power mode. Thus, the system will be deadlocked. The same can occur with multiple write cycles that will fill the two-command deep write buffer of the memory controller.

12.2.6.5 Register Programming

The low-power modes of the memory controller are controlled through the LOWPOWER_CONTROL and LOWPOWER_AUTO_ENABLE bit fields in HW_DRAM_CTL16. These five-bit bit fields each contain one bit for controlling each low-power mode. The LOWPOWER_CONTROL bit field enables the associated low-power mode, and the LOWPOWER_AUTO_ENABLE bit field sets the entry method into that mode as manual or automatic. Table 12-1 shows the relationship between the five bits of the lowpower_control and lowpower_auto_enable bit fields and the various low-power modes.

Table 12-1. Low-Power Mode Bit Fields

Low-Power Mode	Enable	Entry
Memory Power-Down (Mode 1)	LOWPOWER_CONTROL [4] =1	LOWPOWER_AUTO_ENABLE [4] • 0 = Manual • 1 = Automatic
Memory Power-Down with Memory Clock Gating (Mode 2)	LOWPOWER_CONTROL [3] =1	LOWPOWER_AUTO_ENABLE [3] • 0 = Manual • 1 = Automatic
Memory Self-Refresh (Mode 3)	LOWPOWER_CONTROL [2] =1	LOWPOWER_AUTO_ENABLE [2] • 0 = Manual • 1 = Automatic
Memory Self-Refresh with Memory Clock Gating (Mode 4)	LOWPOWER_CONTROL [1] =1	LOWPOWER_AUTO_ENABLE [1] • 0 = Manual • 1 = Automatic
Memory Self-Refresh with Memory and Controller Clock Gating (Mode 5)	LOWPOWER_CONTROL [0] =1	LOWPOWER_AUTO_ENABLE [0] • 0 = Manual • 1 = Automatic

When a LOWPOWER_CONTROL bit field bit is set to 1 by the user, the memory controller checks the LOWPOWER_AUTO_ENABLE bit field.

- If the associated bit in the LOWPOWER_AUTO_ENABLE bit field is set to 1, then the memory controller watches the associated counter for expiration, and then enters that low-power mode.Table 12-2 shows the correlation between the low-power modes and the counters that control each mode’s automatic entry.

Table 14-6. HW_ECC8_STATUS1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**STATUS_PAYLOAD7**	RO	0xc	Count of symbols in error during processing of payload area 7. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
27:24	**STATUS_PAYLOAD6**	RO	0xc	Count of symbols in error during processing of payload area 6. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
23:20	**STATUS_PAYLOAD5**	RO	0xc	Count of symbols in error during processing of payload area 5. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
19:16	**STATUS_PAYLOAD4**	RO	0xc	Count of symbols in error during processing of payload area 4. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.
15:12	**STATUS_PAYLOAD3**	RO	0xc	Count of symbols in error during processing of payload area 3. 0xF indicates uncorrectable. NO_ERRORS = 0x0 No errors occurred. ONE_CORRECTABLE = 0x1 One correctable error occurred. TWO_CORRECTABLE = 0x2 Two correctable errors occurred. THREE_CORRECTABLE = 0x3 Three correctable errors occurred. FOUR_CORRECTABLE = 0x4 Four correctable errors occurred. FIVE_CORRECTABLE = 0x5 Five correctable errors occurred. SIX_CORRECTABLE = 0x6 Six correctable errors occurred. SEVEN_CORRECTABLE = 0x7 Seven correctable errors occurred. EIGHT_CORRECTABLE = 0x8 Eight correctable errors occurred. NOT_CHECKED = 0xC This block was not examined by the ECC8. UNCORRECTABLE = 0xE Errors occurred that were uncorrectable. ALL_ONES = 0xF All bits are 1.

detected within a block, the BCH hardware activates the error correction logic to determine where bit errors have occurred and ultimately correct them in the data buffer in system memory. After an entire flash page has been read and corrected, the BCH will signal an interrupt to the CPU.

Figure 15-2 indicates how data read from the GPMI is operated on within the BCH hardware. As the BCH receives data from the GPMI (top row) it is written to memory by the BCH's Bus Interface Unit (BIU) (second row). For blocks requiring correction, the KES logic will be activated after the entire block has been received. Once the error locator polynomial has been computed, the corrections are determined by the Chien Search and fed back to the BIU, which performs a read/modify/write operation on the buffer in memory to correct the data.

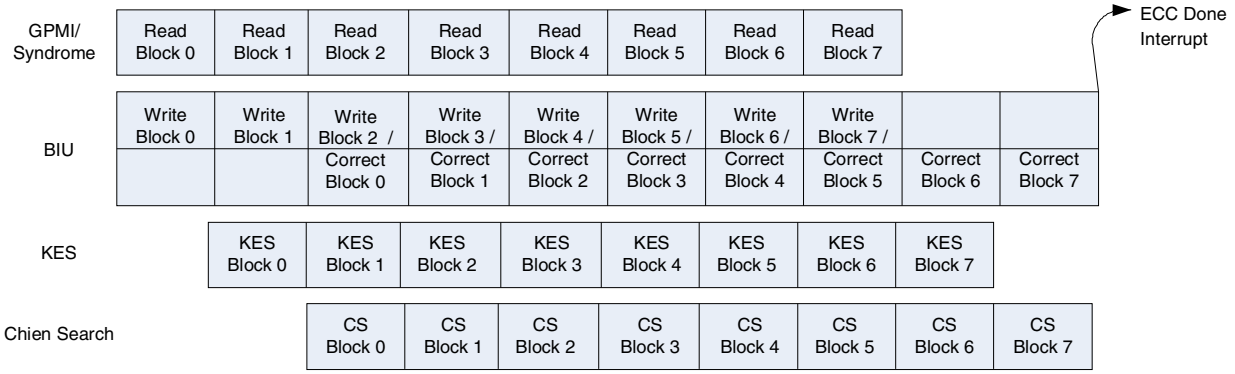

Figure 15-2. Block Pipeline while Reading Flash

15.2.1 BCH Limitations and Assumptions

- The BCH is programmable to support 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20 bit error correction. ECC0 is supported as a passthrough, non-correcting mode.
- Data block sizes must be a multiple of 4 bytes and must be 4-byte aligned in system memory.
- The BCH supports a programmable number of metadata/auxiliary data bytes, from 0 to 255.
- Metadata will be written at the beginning of the flash page to facilitate fast access for filesystem operations.
- Metadata may be treated as an independent block for ECC purposes or combined with the first data block to conserve bits in the flash.
- The BCH does not support a partial page write.
- Flash read operations can read the entire page or the first block on the page.
- The BCH also supports a memory-to-memory mode of operation that does not require the use of DMA or the GPMI.

15.2.2 Flash Page Layout

The BCH supports a fully programmable flash page layout, versus the hardwired modes supported in the former ECC8 engine. The BCH maintains 4 independent layout registers that can describe four completely different NAND devices or layouts. When the BCH initiates an operation, it selects one of the lay-

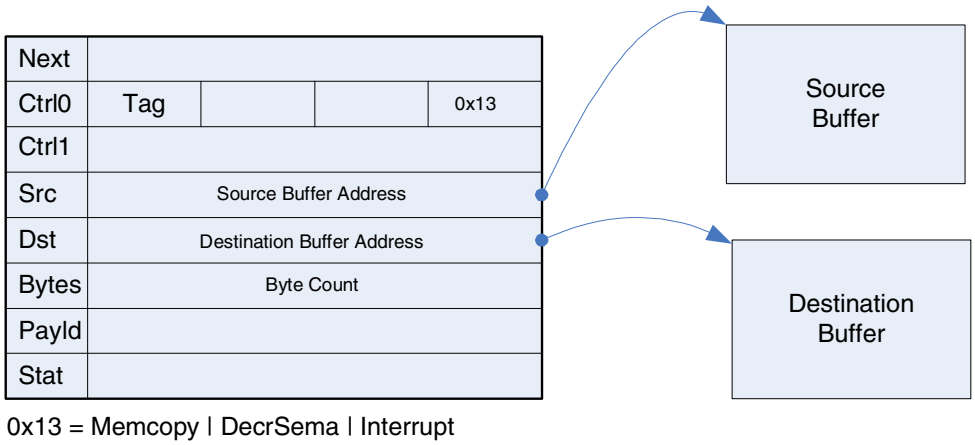

Figure 16-6. Basic Memory Copy Operation

```
<code>

typedef struct _dcp_descriptor
{
    u32             *next;
    hw_dcp_packet1_t  ctrl0;
    hw_dcp_packet2_t  ctrl1;
    u32             *src,
                    *dst,
                     buf_size,
                    *payload,
                     stat;
}  DCP_DESCRIPTOR;

  DCP_DESCRIPTOR dcp1;
  u32 *srcbuffer, *dstbuffer;

  // set up control packet
  dcp1.next = 0;                  // single packet in chain
  dcp1.ctrl0.U = 0;               // clear ctrl0 field
  dcp1.ctrl0.B.ENABLE_MEMCOPY = 1; // enable memcopy
  dcp1.ctrl0.B.DECR_SEMAPHORE = 1; // decrement semaphore
  dcp1.ctrl0.B.INTERRUPT = 1;     // interrupt
  dcp1.ctrl1.U = 0;               // clear ctrl1
  dcp1.src = srcbuffer;           // source buffer
  dcp1.dst = dstbuffer;           // destination buffer
  dcp1.buf_size = 512;            // 512 bytes
  dcp1.payload = NULL;            // not required
  dcp1.status = 0;                // clear status

  // Enable channel 0
  HW_DCP_CHnCMDPTR_WR(0, dcp1);   // write packet address to pointer register
  HW_DCP_CHnSEMA_WR(0, 1);        // increment semaphore by 1

  // now wait for interrupt or poll
  // polling code
  while ( (HW_DCP_STAT_RD() & 0x01) == 0x00 );

  // now check/clear channel status
  if ( (HW_DCP_CHnSTAT_RD(0) & 0xFF) != 0 ) {
    // an error occurred
    HW_DCP_CHnSTAT_CLR(0, 0xff);
  }
  // clear interrupt register
  HW_DCP_STAT_CLR(1);

</code>
```

16.2.6.2 Basic Hash Operation Programming Example

To perform a basic hash operation, only a single descriptor is required. The DCP simply reads data from the source buffer and computes the hash value on the contents. This process is illustrated in Figure 16-7.

Table 16-36. HW_DCP_PACKET3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RO	0x0	Source Buffer Address Pointer. This value is the working value and will update as the operation proceeds.

DESCRIPTION:

This register shows the contents of the Source Address register from the packet being processed. When the CONSTANT_FILL bit in the Control 0 field is set, this field contains the data written to the destination buffer.

EXAMPLE:

```
Empty Example.
```

16.3.13 DCP Work Packet 4 Status Register Description

This register displays the values for the current work packet offset 0x10 (Destination Address) field.

HW_DCP_PACKET4 0x0C0

Table 16-37. HW_DCP_PACKET4

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ADDR																															

Table 16-38. HW_DCP_PACKET4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDR**	RO	0x0	Destination Buffer Address Pointer. This value is the working value and will update as the operation proceeds.

DESCRIPTION:

This register shows the contents of the Destination Address register from the packet being processed.

EXAMPLE:

```
Empty Example.
```

16.3.14 DCP Work Packet 5 Status Register Description

This register displays the values for the current work packet offset 0x14 (Buffer Size) field.

HW_DCP_PACKET5 0x0D0

Table 16-39. HW_DCP_PACKET5

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
COUNT																															

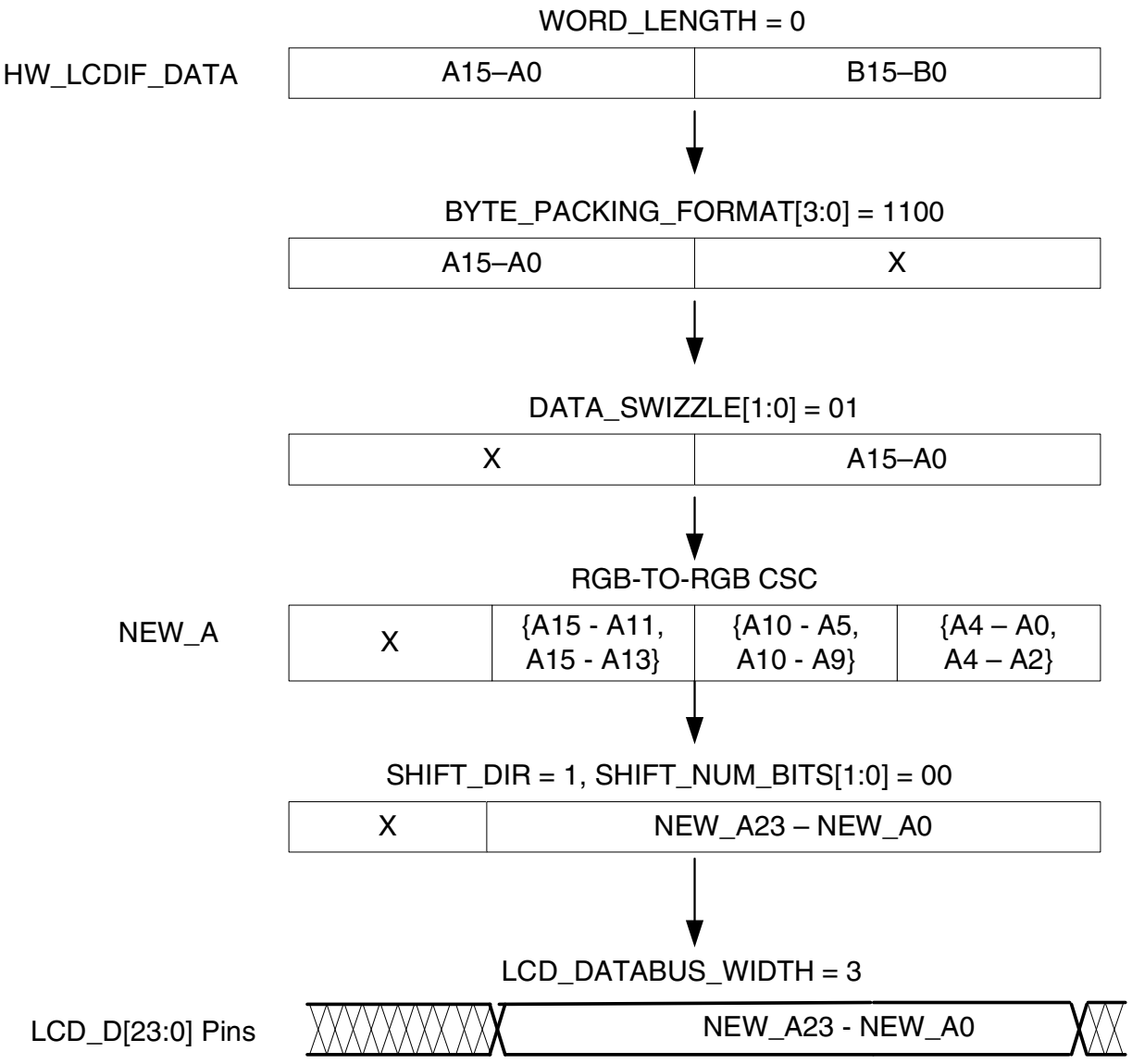

Figure 18-6. 16-Bit LCDIF Register Programming—Example B

18.2.3 LCDIF Interrupts

The LCDIF supports a number of interrupts to aid controlling and status reporting of the block. All the interrupts have individual mask bits for enabling or disabling each of them. They all get funneled through a single interrupt line connected to the interrupt collector (ICOLL). The following list describes the different interrupts supported by the LCDIF:

- Underflow interrupt is asserted when the clock domain crossing FIFO (TXFIFO) becomes empty but the block is in active display portion at that time. Software should take corrective action to make sure that this does not happen. This interrupt is of value only in DOTCLK and DVI modes.
- Overflow interrupt will be asserted in PIO mode if software writes to LFIFO while it is full.

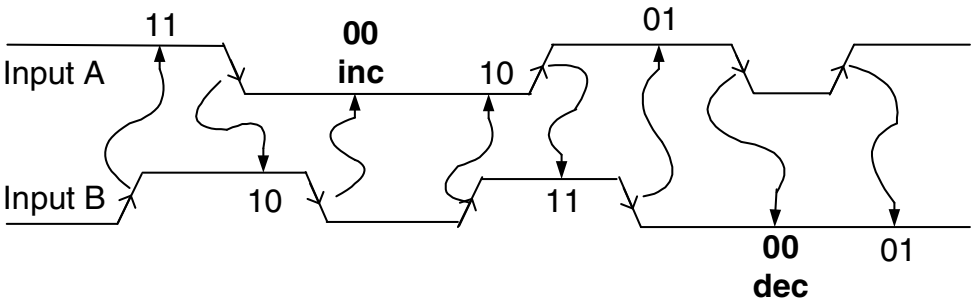

Figure 22-7. Rotary Decoding Mode—Input Transitions

Figure 22-7 shows that each detected edge causes a transition in the decoder state machine. Not all transitions are legal (see Table 22-2). For example, there is no legal way to transition directly from state 11 to 00 using normal inputs. In the cases where this occurs, the state machine goes to an alternate set of states and follows the input sequence until a valid sequence leading to state 00 is detected. No increment or decrement action is taken from the alternate state sequence.

Table 22-2. Rotary Decoder State Machine Transitions

CURRENT STATE	"INPUT" BA=00	"INPUT" BA=01	"INPUT" BA=10	"INPUT" BA=11
00	00	01	10	error
01	00, dec	01	error	11
10	00, inc	error	10	11
11	error	01	10	11

22.3.1 Testing the Rotary Decoder

To test the rotary decoder, select PWM1 and PWM2 as inputs to ROTARYA and ROTARYB. Since PWM1 and PWM2 can be started with known phase offsets and duty cycles, a continuous increment or decrement stream can be generated. Since PWM1 and PWM2 can be used as GPIO devices, the final part of the test is to generate and test a sequence of clockwise and counter-clockwise rotations to cover the entire state machine transitions, including the error conditions.

22.3.2 Behavior During Reset

A soft reset (SFTRST) can take multiple clock periods to complete, so do NOT set CLKGATE when setting SFTRST. The reset process gates the clocks automatically. See Section 39.3.10, "Correct Way to Soft Reset a Block," for additional information on using the SFTRST and CLKGATE bit fields.

22.4 Programmable Registers

The following registers describe the programming interface for the timers and the rotary decoder.

Table 22-20. HW_TIMROT_TIMCTRL3 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:20	**RSRVD2**	RO	0x0	Always write zeroes to this bit field.
19:16	**TEST_SIGNAL**	RW	0x0	Selects the source of the signal to be measured in duty cycle mode. NEVER_TICK = 0x0 Never tick. Freeze the count. PWM0 = 0x1 Input from PWM0. PWM1 = 0x2 Input from PWM1. PWM2 = 0x3 Input from PWM2. PWM3 = 0x4 Input from PWM3. PWM4 = 0x5 Input from PWM4. ROTARYA = 0x6 Input from Rotary A. ROTARYB = 0x7 Input from Rotary B. 32KHZ_XTAL = 0x8 Input from 32-kHz crystal. 8KHZ_XTAL = 0x9 Input from 8 kHz (divided from 32-kHz crystal). 4KHZ_XTAL = 0xA Input from 4 kHz (divided from 32-kHz crystal). 1KHZ_XTAL = 0xB Input from 1 kHz (divided from 32-kHz crystal). TICK_ALWAYS = 0xC Always tick.
15	**IRQ**	RW	0x0	This bit is set to one when Timer 3 decrements to zero. Write a zero to clear it or use Clear SCT mode.
14	**IRQ_EN**	RW	0x0	Set this bit to one to enable the generation of a CPU interrupt when the count reaches zero in normal counter mode.
13:11	**RSRVD1**	RO	0x0	Always write zeroes to this bit field.
10	**DUTY_VALID**	RO	0x0	This bit is set and cleared by the hardware. It is set only when in duty cycle measuring mode and the HW_TIMROT_TIMCOUNT3 has valid duty cycle data to be read. This register will be cleared if not in duty cycle mode or on writes to this register. In the case that it is written while in duty cycle mode, this bit will clear but will again be set at the appropriate time for reading the count register.
9	**DUTY_CYCLE**	RW	0x0	Set this bit to one to cause the timer to operate in duty cycle measuring mode.
8	**POLARITY**	RW	0x0	Set this bit to one to invert the input to the edge detector. 0: Positive edge detection. 1: Invert to negative edge detection.
7	**UPDATE**	RW	0x0	Set this bit to one to cause the running count to be written from the CPU at the same time a new fixed count register value is written.
6	**RELOAD**	RW	0x0	Set this bit to one to cause the timer to reload its current count from its fixed count value whenever the current count decrements to zero. When set to zero, the timer enters a mode that freezes at a count of zero. When the fixed count is zero, setting this bit to one causes a continuous reload of the fixed count register so that writting a non-zero value will start the timer.

Table 24-22. HW_PWM_PERIOD4 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
22:20	**CDIV**	RW	0x0	Clock divider ratio to apply to the crystal clock frequency (24.0 MHz) that times the PWM output signal. DIV_1 = 0x0 Divide by 1. DIV_2 = 0x1 Divide by 2. DIV_4 = 0x2 Divide by 4. DIV_8 = 0x3 Divide by 8. DIV_16 = 0x4 Divide by 16. DIV_64 = 0x5 Divide by 64. DIV_256 = 0x6 Divide by 256. DIV_1024 = 0x7 Divide by 1024.
19:18	**INACTIVE_STATE**	RW	0x0	The logical inactive state that is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Inactive state sets PWM output to high-impedance. 0 = 0x2 Inactive state sets PWM output to 0 (low). 1 = 0x3 Inactive state sets PWM output to 1 (high).
17:16	**ACTIVE_STATE**	RW	0x0	The logical active state is mapped to the PWM output signal. Note that the undefined state of 0x1 is mapped to high-impedance. HI_Z = 0x0 Active state sets PWM output to high-impedance. 0 = 0x2 Active state sets PWM output to 0 (low). 1 = 0x3 Active state sets PWM output to 1 (high).
15:0	**PERIOD**	RW	0x0	Number of divided XTAL clock cycles in the entire period of the PWM waveform, minus 1. For example, to obtain 6 clock cycles in the actual period, set this field to 5.

DESCRIPTION:

This register contains the programming paramters for multi-chip attachment mode, clock divider value, active high, low values and period for channel 4.

EXAMPLE:

```
HW_PWM_PERIODn_WR(4, 0x00000b1f); // Set up period and active/inactive output states
```

24.4.12 PWM Version Register Description

This register indicates the version of the block for debug purposes.

HW_PWM_VERSION 0x0b0

Table 24-23. HW_PWM_VERSION

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
MAJOR								MINOR								STEP															

dropped (decimated) to find the next sample at which to produce a filtered multibit sigma delta A/D value to send to the DMA. Whenever the whole number part (bits 23:16) is zero, then a sample is produced.

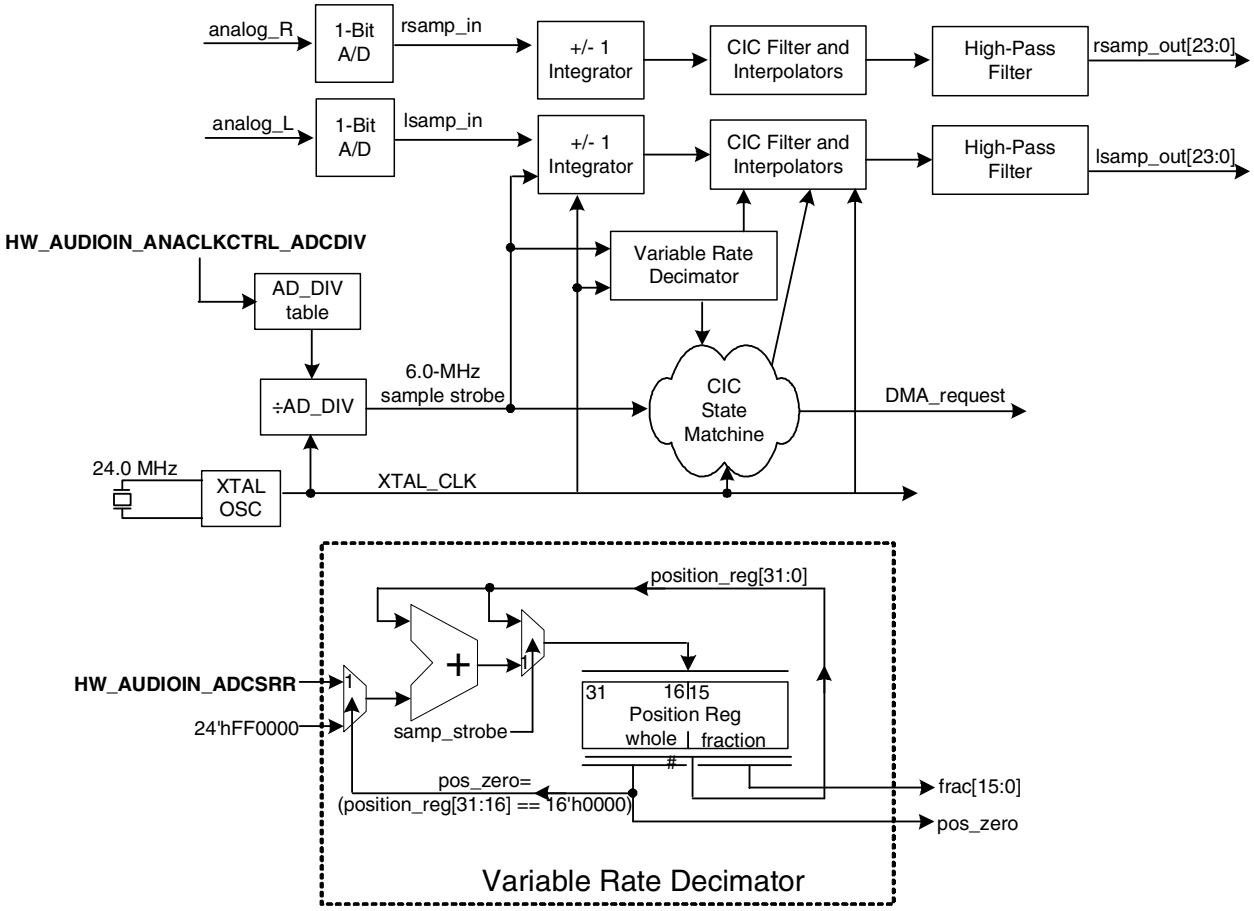

Figure 28-3. Variable-Rate A/D Converter

The range of values of the samples stored into the on-chip RAM is proportional to the square of the over-sample rate (OSR) used in the capture process. The larger the OSR, the longer period the integrators run in the ADC. As a result, the range of values seen for the same signal wave form captured at the same sample rate but with two different OSR will be different.

For example:

- An 8-kHz microphone captured at F_{ADC} = 6.0 MHz will be 36 times smaller than the values resulting from capturing the same source signal at F_{ADC} = 1.0 MHz.
- The peak range of values seen in a capture of a signal at 44.1 kHz with $Fanalog_{ADC}$ = 6.0 MHz is ±3200.
- The oversample ratio in this case is OSR= 136.054.
- Calculate a magnitude constant, K_{filter} for ADC's filter from this as $K_{filter} = OSR^2/Peak\ Value = (136.054)^2/3200 = 5.7846$.
- For any OSR in any sample rate, the peak value can be approximated by $Value_{peak} = OSR^2 / K_{filter}$.

shots them and starts a new conversion operation for all scheduled channels. Thus, one can set the schedule bits for four channels on the same clock edge. The channel with the largest channel number is converted last and has its interrupt status bit set last. If that channel is the only one of the four with an interrupt enable bit set, then it interrupts the ARM after all four channels have been converted, effectively ganging four channels together.

33.2.4 Delay Channels

To minimize the interrupt load on the ARM processor, four delay channels are provided. Each has an 11-bit counter that increments at 2 kHz. A delay channel can be kicked off either by an ARM store instruction or at the completion of a delay channel time-out. At time-out, each channel has the option of kicking off any combination of LRADC conversions, as well as any combination of delay channels.

NOTE: The DELAY fields in HW_LRADC_DELAY0, HW_LRADC_DELAY1, HW_LRADC_DELAY2, and HW_LRADC_DELAY3 must be non-zero; otherwise, the LRADC will not trigger the delay group. The ACCUMULATE bit in the appropriate channel register HW_LRADC_CHn must be set to 1 if NUM_SAMPLES is greater then 0; otherwise, the IRQs will not fire.

Consider the case of a touch-screen that requires 4x oversampling of its coordinate values. Further, suppose you wish to receive an oversampled X or Y coordinate approximately every 5 ms and that the oversampling should be spaced at 1-ms intervals.

- In the touch-screen, first select either X or Y drive, then set up the appropriate LRADC.
- In setting up the LRADC, clear the accumulator associated with it by setting the ACCUMULATE bit and set the NUM_SAMPLES field to 3 (4 samples before interrupt request).
- Next, set up two delay channels.
  - Delay Channel 1 is set to delay 1 ms (DELAY = 1, two ticks) and then kick the schedule bit for LRADC 4. Its LOOP_COUNT bit field is also set to 3, so that four kicks of LRADC 4 occur, each spaced by 1 ms.
  - Delay Channel 0 is set to delay 1 ms with LOOP_COUNT = 0, i.e., one time. Its TRIGER_DELAYS field is set to trigger Delay Channel 1 when it times out. The ISR routine kicks off Delay Channel 0 immediately before it does its return from interrupt. Another interrupt (LRADC4_IRQ) is asserted once the entire 4x oversample data capture is complete. A sample timeline for such a sequence is shown in Figure 33-3.

NOTE: If a delay group schedules channels to be sampled and a manual write to the schedule field in CTRL0 occurs while the block is discarding samples, the LRADC will switch to the new schedule and will not sample the channels that were previously scheduled. The time window for this to happen is very small and lasts only while the LRADC is discarding samples.

WARNING: The pad ESD protection limits the voltage on the LRADC0-LRADC6 inputs to VDDIO. The BATT and 5V inputs to the LRADC have built-in dividers to handle the higher voltages.

Table 33-28. HW_LRADC_DELAY0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
19:16	**TRIGGER_DELAYS**	RW	0x0	Setting a bit in this bit field to one causes the delay controller to trigger the corresponding delay channel. This trigger occurs when the delay count of this delay channel reaches zero. Note that all four delay channels can be triggered at the same time, including the one that issues the trigger. This can have the effect of automatically retriggering a delay channel.
15:11	**LOOP_COUNT**	RW	0x00	This bit field specifies the number of times this delay counter will count down and then trigger its designated targets. This is particularly useful for scheduling multiple samples of an LRADC channel set. If this field is set to 0x0, then exactly one delay loop will be generated with exactly one event triggering the target LRADC and/or delay channels. Note setting the loop count to 0x01 will yield two conversions.
10:0	**DELAY**	RW	0x000	This 11-bit field counts down to zero. At zero it triggers either a set of LRADC channel conversions or another delay channel, or both. It can trigger up to all eight LRADCs and all four delay channels in a single even. This counter operates on a 2KHz clock derived from crystal clock.

DESCRIPTION:

The LRADC Delay Channel provides control by which LRADC channels and delay channels (including itself) may be triggered. The triggering of the selected delay and LRADC channel(s) is delayed by the DELAY field value which counts down on a 2 kHz clock. It is possible to use delay channels chained together to configure dependent timing of channel conversions as in the example provided in introduction to this block. A delay channel may also be configured to trigger itself. In this case, it could be used to simultaneously trigger an LRADC channel, providing continuous acquisitions of the conversions executed, delayed by the value specified in the DELAY field. The delay channel is started by setting the KICK bit to one.

EXAMPLE:

```
HW_LRADC_DELAYn_WR(0, (BF_LRADC_DELAYn_TRIGGER_LRADCS(0x05) | // LRADC channel 0 and 2
BF_LRADC_DELAYn_KICK(1) | // Start the Delay channel
BF_LRADC_DELAYn_TRIGGER_DELAYS(0x1) | // restart delay channel 0 each time
BF_LRADC_DELAYn_DELAY(0x0E45) ) ); // delay 3653 periods of 2 kHz clock
// ... do other things until the triggered LRADC channels report an interrupt.
```

33.4.15 LRADC Scheduling Delay 1 Description

The LRADC scheduling delay 1 register controls one delay operation. At the end of this delay, this channel can trigger one or more LRADC channels or one or more Scheduling delay channels .

HW_LRADC_DELAY1	0x0E0
HW_LRADC_DELAY1_SET	0x0E4
HW_LRADC_DELAY1_CLR	0x0E8
HW_LRADC_DELAY1_TOG	0x0EC

- When the synchronized edge is recognized by the on-chip SJTAG controller, it pulls the DEBUG line back down clock to Q to pad after the rising edge of its 24-MHz clock. This is the critical timing mark that is detected in the FPGA/CPLD and used to time data in next phase.
- The timing mark phase ends when the on-chip SJTAG stops driving the serial JTAG wire low for one cycle.

34.2.3 Debugger Send TDI, Mode Phase

- During the first 24-MHz clock period of this phase, the FPGA/CPLD sends a one clock-wide signal that either tells the on-chip SJTAG that it is present and a JTAG clock begins, or it tells the on-chip SJTAG to do a JTAG reset operation to the ARM JTAG TAP controller.
- If a noise glitch falsely triggered the ASYNC Start Phase, then the on-chip SJTAG will treat it as a TAP controller reset in most cases.
- If the debugger is performing a JTAG clock cycle operation then, it next sends the state of the debugger TDI and MODE pins sequentially on the wire, i.e., one in each of the following two 24-MHz clocks. Notice that for this phase, the FPGA/CPLD knows the correct timing to change these three data elements on the wire because of the timing information it learned from the Timing Mark Phase.
- This phase ends after the FPGA/CPLD drives the serial wire low on the fourth 24-MHz clock of this phase.

To review, the first data element sent is the signal that distinguishes clock cycles from TAP reset cycles. The next two bits sent are the JTAG MODE and TDI bits from the debugger. Finally, the line is driven low and pulled down for half a 24-MHz clock and the driver is turned off and the pulldown left on. This phase ends when the half-clock pulldown is complete. The rising edge of the JTAG clock is sent to the ARM TAP controller during this phase.

34.2.4 i.MX23 Wait For Return Clock Phase

During this phase:

- The on-chip SJTAG controller waits for the ARM TAP controller to send back the return clock. This is an asynchronous event for both the on-chip TAP controller and the FPGA/CPLD controller.
- The on-chip controller drives the serial wire high for one 24-MHz clock period to tell the FPGA/CPLD that the variable length wait for return clock period is complete.
- This phase ends when the on-chip SJTAG detects the return clock going high and drives the serial high for one 24-MHz clock.

34.2.5 i.MX23 Sends TDO and Return Clock Timing Phase

During this phase:

- The on-chip SJTAG controller sends the value of the ARM TAP controller’s TDO signal on the wire for one full 24-MHz period that begins on the rising edge of the on-chip 24-MHz clock.
- This phase ends when the TDO value has been sent.

35.8.3.1 NAND Config Block

The primary NAND Config Block (NCB) resides on the NAND attached to GPMI_CE0. The NCB is the first sector in the first good block. In the single NAND configuration, a copy of the NCB also resides on the NAND—its location is immediately after NCB1 - 2nd search area. In the case of multiple NAND devices, copies of the NCB, LDLB, and DBBT are located on the first two NANDs. This case is shown in more detail in Section 35.8.3.4, “Firmware Layout on the NAND.”

The layout of first good page containing NCB is described in Figure 35-6.

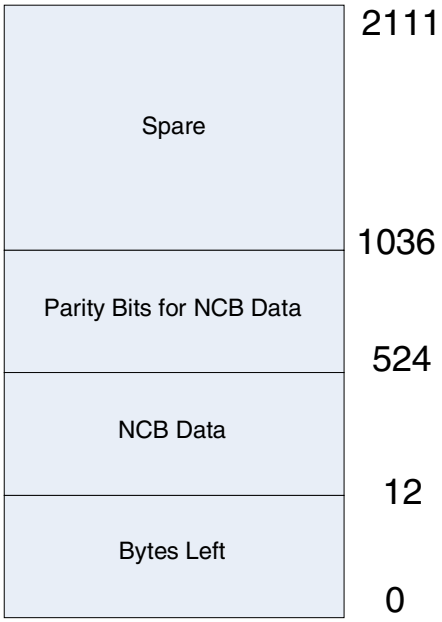

Figure 35-6. Layout of Boot Page Containing NCB

The NCB is located on the first good page of the NAND; minimum size of a page is 2112 bytes. The first 12 bytes of NCB page are reserved and left blank (all 0s), next 512 bytes are reserved for NCB data. The remaining 512 bytes are available for software ECC and the rest are all 0s. NCB is protected using SEC-DED Hamming codes.

35.8.3.2 Single Error Correct and Double Error Detect (SEC-DED) Hamming

For each 8 bits of data in 512 byte NCB, 5-bit parity is used. Each byte of parity area contains 5 parity bits (LSB) and 3 unused bits (MSB).

For each 8 bits of NCB data, parity is calculated and compared with corresponding parity bits read from the parity area of NCB page to detect errors and correct single error.

If errors are more than one then flag shall be raised against the block.

To determine a good NCB, all three fingerprints must match and the ECC must not fail.

Table 36-5. 169-Pin BGA Pin Definitions by Pin Name (continued)

Number	Pin Name	Group	Type	Description 1	Description 2	Description 3
A11	BATT	DCDC	A	Battery Input		
B11	DCDC_BATTERY	DCDC	A	DCDC Battery		
A13	DCDC_GND	DCDC	A	DCDC Ground		
B13	DCDC_LN1	DCDC	A	DCDC Inductor N 1		
A12	DCDC_LP	DCDC	A	DCDC Inductor P		
B12	DCDC_VDDA	DCDC	A	DCDC Analog Power		
D13	DCDC_VDDD	DCDC	A	DCDC Digital Core Power		
C13	DCDC_VDDIO	DCDC	A	DCDC I/O Power		
E12	DEBUG	SYSTEM	DIO	1-Wire Debug Port		
K13	EMI_A00	EMI	DIO	EMI Address 0		
K12	EMI_A01	EMI	DIO	EMI Address 1		
J13	EMI_A02	EMI	DIO	EMI Address 2		
K11	EMI_A03	EMI	DIO	EMI Address 3		
G10	EMI_A04	EMI	DIO	EMI Address 4		
G12	EMI_A05	EMI	DIO	EMI Address 5		
F12	EMI_A06	EMI	DIO	EMI Address 6		
G11	EMI_A07	EMI	DIO	EMI Address 7		
G13	EMI_A08	EMI	DIO	EMI Address 8		
F13	EMI_A09	EMI	DIO	EMI Address 9		
J12	EMI_A10	EMI	DIO	EMI Address 10		
H10	EMI_A11	EMI	DIO	EMI Address 11		
H13	EMI_A12	EMI	DIO	EMI Address 12		
L13	EMI_BA0	EMI	DIO	DRAM Bank Address 0		
L12	EMI_BA1	EMI	DIO	DRAM Bank Address 1		
M13	EMI_CASN	EMI	DIO	EMI Column Address Strobe#		
J10	EMI_CE0N	EMI	DIO	EMI Chip Enable 0#		
H12	EMI_CE1N	EMI	DIO	EMI Chip Enable 1#		
F11	EMI_CKE	EMI	DIO	EMI Clock Enable		
M6	EMI_CLK	EMI	DIO	EMI Clock		
N6	EMI_CLKN	EMI	DIO	EMI Clock#		
K10	EMI_D00	EMI	DIO	EMI Data 0		
M10	EMI_D01	EMI	DIO	EMI Data 1		
N11	EMI_D02	EMI	DIO	EMI Data 2		
N10	EMI_D03	EMI	DIO	EMI Data 3		
M11	EMI_D04	EMI	DIO	EMI Data 4		
K9	EMI_D05	EMI	DIO	EMI Data 5		
L11	EMI_D06	EMI	DIO	EMI Data 6		
L9	EMI_D07	EMI	DIO	EMI Data 7		
N9	EMI_D08	EMI	DIO	EMI Data 8		
N8	EMI_D09	EMI	DIO	EMI Data 9		
M8	EMI_D10	EMI	DIO	EMI Data 10		
K7	EMI_D11	EMI	DIO	EMI Data 11		
L7	EMI_D12	EMI	DIO	EMI Data 12		
K6	EMI_D13	EMI	DIO	EMI Data 13		
N7	EMI_D14	EMI	DIO	EMI Data 14		
M7	EMI_D15	EMI	DIO	EMI Data 15		

Table 36-7. 169-Pin BGA Ball Map

	1	2	3	4	5	6	7	8	9	10	11	12	13	
A	USB_DP	VSSA2	VAG	XTALI	XTALO	LINE1_INR	HP_VGND	LRADC0	VDD4P2	LRADC4	BATT	DCDC_LP	DCDC_GND	A
B	VDDS	USB_DM	PSWITCH	RTC_XTALI	LINE1_INL	MIC	HPL	HPR	LRADC1	LRADC5	DCDC_BATTERY	DCDC_VDDA	DCDC_LN1	B
C	SPEAKERP	VDDXTAL	PWM0	RTC_XTALO	I2C_SCL	I2C_SDA	VDDA1	VSSA1	LRADC2	VDDM	VDAC1	PWM4	DCDC_VDDIO	C
D	SPEAKERN	PWM1	LCD_D00	VSSA5	VDDD1/3	VSSD1/5	ROTARYA	VDDD1/3	LRADC3	SSP1_DETECT	PWM3	SSP1_SCK	DCDC_VDDD	D
E	LCD_D17	LCD_D01	LCD_D16	LCD_D02	LCD_D15	GPMI_RDY3	GPMI_WRN	ROTARYB	VSSD1/5	SSP1_DATA3	VDDIO33_1/2/3	DEBUG	VDD5V	E
F	LCD_D03	LCD_D14	LCD_D04	LCD_D13	LCD_D05	GPMI_RDY2	GPMI_WPN	GPMI_CE1N	SSP1_DATA1	SSP1_DATA2	EMI_CKE	EMI_A06	EMI_A09	F
G	LCD_D12	LCD_D06	VDDIO33_1/2/3	LCD_D11	LCD_D07	GPMI_RDY0	AUART1_CTS	GPMI_CE0N	SSP1_DATA0	EMI_A04	EMI_A07	EMI_A05	EMI_A08	G
H	LCD_D10	LCD_D08	LCD_D09	LCD_CS	LCD_ENABLE	GPMI_RDN	AUART1_RTS	AUART1_RX	SSP1_CMD	EMI_A11	PWM2	EMI_CE1N	EMI_A12	H
J	LCD_VSYNC	LCD_HSYNC	LCD_RS	LCD_WR	LCD_RESET	GPMI_RDY1	VSSIO_EMI1/2	AUART1_TX	VSSIO_EMI1/2	EMI_CE0N	EMI_WEN	EMI_A10	EMI_A02	J
K	LCD_DOTCK	GPMI_D00	GPMI_CLE	GPMI_ALE	GPMI_CE2N	EMI_D13	EMI_D11	EMI_DQS1	EMI_D05	EMI_D00	EMI_A03	EMI_A01	EMI_A00	K
L	GPMI_D01	GPMI_D02	GPMI_D03	GPMI_D05	GPMI_D04	VDDIO_EMIQ	EMI_D12	VDDIO_EMI1/2	EMI_D07	VDDIO_EMI1/2	EMI_D06	EMI_BA1	EMI_BA0	L
M	GPMI_D07	GPMI_D06	GPMI_D08	GPMI_D09	GPMI_D10	EMI_CLK	EMI_D15	EMI_D10	EMI_DQM1	EMI_D01	EMI_D04	EMI_DQM0	EMI_CASN	M
N	GPMI_D11	GPMI_D12	GPMI_D13	GPMI_D14	GPMI_D15	EMI_CLKN	EMI_D14	EMI_D09	EMI_D08	EMI_D03	EMI_D02	EMI_DQS0	EMI_RASN	N
	1	2	3	4	5	6	7	8	9	10	11	12	13	

Table 37-10. HW_PINCTRL_MUXSEL1 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
27:26	**BANK0_PIN29**	RW	0x3	Pin 69, AUART1_TX pin function selection: 00= auart1_tx; 01= reserved; 10= ssp1_d7; 11= GPIO.
25:24	**BANK0_PIN28**	RW	0x3	Pin 68, AUART1_RX pin function selection: 00= auart1_rx; 01= reserved; 10= ssp1_d6; 11= GPIO.
23:22	**BANK0_PIN27**	RW	0x3	Pin 67, AUART1_RTS pin function selection: 00= auart1_rts; 01= reserved; 10= ssp1_d5; 11= GPIO.
21:20	**BANK0_PIN26**	RW	0x3	Pin 66, AUART1_CTS pin function selection: 00= auart1_cts; 01= reserved; 10= ssp1_d4; 11= GPIO.
19:18	**BANK0_PIN25**	RW	0x3	Pin 60, GPMI_RDN pin function selection: 00= gpmi_rdn; 01= reserved; 10= reserved; 11= GPIO.
17:16	**BANK0_PIN24**	RW	0x3	Pin 65, GPMI_WRN pin function selection: 00= gpmi_wrn; 01= reserved; 10= ssp2_sck; 11= GPIO.
15:14	**BANK0_PIN23**	RW	0x3	Pin 64, GPMI_WPN pin function selection: 00= gpmi_wpn; 01= reserved; 10= reserved; 11= GPIO.
13:12	**BANK0_PIN22**	RW	0x3	Pin 63, GPMI_RDY3 pin function selection: 00= gpmi_ready3; 01= reserved; 10= reserved; 11= GPIO.
11:10	**BANK0_PIN21**	RW	0x3	Pin 62, GPMI_RDY2 pin function selection: 00= gpmi_ready2; 01= reserved; 10= reserved; 11= GPIO.
9:8	**BANK0_PIN20**	RW	0x3	Pin 43, GPMI_RDY1 pin function selection: 00= gpmi_ready1; 01= reserved; 10= ssp2_cmd; 11= GPIO.