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	Active
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	-10°C ~ 70°C (TA)
Security Features	Cryptography, Hardware ID
Package / Case	128-LQFP
Supplier Device Package	128-LQFP (14x14)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mcimx233dag4c

The BCH sits on the AXI fabric with close coupling to both the GPMI and external memory controller.

See Chapter 15, “20-BIT Correcting ECC Accelerator (BCH).”

1.2.13 Data Co-Processor (DCP)—Memory Copy, Crypto, and Color-Space Converter

The i.MX23 SOC contains a data co-processor consisting of four virtual channels. Each channel is essentially a memory-to-memory copy engine. The linked list control structure can be used to move byte-aligned blocks of data from a source to a destination. In the process of copying from one place to another, the DCP can be programmed to encrypt or decrypt the block using AES-128 in one of several chaining modes. An SHA-1 hash can be calculated as part of the memory-copy operation.

See Chapter 16, “Data Co-Processor (DCP),” for more information.

1.2.14 Mixed Signal Audio Subsystem

The i.MX23 contains an integrated high-quality mixed signal audio subsystem, including high-quality sigma delta D/A and A/D converters, as shown in Figure 1-4.

The chip includes a low-noise headphone driver that allows it to directly drive low-impedance (16Ω) headphones. The direct drive, or “capless” mode, removes the need for large expensive DC blocking capacitors in the headphone circuit. The headphone power amplifier can detect headphone shorts and report them via the interrupt collector. A digitally programmable master volume control allows user control of the headphone volume. Use of the headphone amplifier volume control is recommended as the digital control may reduce SNR performance. Annoying clicks and pops are eliminated by zero-crossing updates in the volume/mute circuits and by headphone driver startup and shutdown circuits.

The microphone circuit has a mono-to-stereo programmable gain pre-amp and an optional microphone bias generator.

Also integrated is a class A-B mono speaker amplifier which must be powered from a sufficiently high-enough current 4.2V source such as the battery. The speaker amplifier can support up to 2W rms of output assuming a 4.2V supply and a 4Ω speaker load.

These features are described in Chapter 28, “AUDIOIN/ADC,” and Chapter 29, “AUDIOOUT/DAC.”

Table 4-9. HW_CLKCTRL_HBUS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:30	**RSRVD4**	RO	0x0	Reserved
29	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28	**DCP_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on DCP activity. 0 = Run at the programmed CLK_H frequency.
27	**PXP_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on PXP activity. 0 = Run at the programmed CLK_H frequency.
26	**APBHDMA_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on APBH DMA activity. 0 = Run at the programmed CLK_H frequency.
25	**APBXDMA_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on APBX DMA activity. 0 = Run at the programmed CLK_H frequency.
24	**TRAFFIC_JAM_AS_ENABLE**	RW	0x0	Enable auto-slow mode when less than three masters are trying to use the AHB. More than three active masters will engage the default mode. 0 = Run at the programmed CLK_H frequency.
23	**TRAFFIC_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on AHB master activity. 0 = Run at the programmed CLK_H frequency.
22	**CPU_DATA_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on with CPU Data access to AHB. 0 = Run at the programmed CLK_H frequency.
21	**CPU_INSTR_AS_ENABLE**	RW	0x0	Enable auto-slow mode based on with CPU Instruction access to AHB. 0 = Run at the programmed CLK_H frequency.
20	**AUTO_SLOW_MODE**	RW	0x0	Enable CLK_H auto-slow mode. When this is set, then CLK_H will run at the slow rate until one of the fast mode events has occurred. Note: The AUTO_SLOW_MODE bit must be cleared before writing to the SLOW_DIV bitfield.
19	**RSRVD2**	RO	0x0	Reserved
18:16	**SLOW_DIV**	RW	0x0	Slow mode divide ratio. Sets the ratio of CLK_H fast rate to the slow rate. Note: The AUTO_SLOW_MODE bit must be cleared before writing to the SLOW_DIV bitfield. BY1 = 0x0 Slow mode divide ratio = 1 BY2 = 0x1 Slow mode divide ratio = 2 BY4 = 0x2 Slow mode divide ratio = 4 BY8 = 0x3 Slow mode divide ratio = 8 BY16 = 0x4 Slow mode divide ratio = 16 BY32 = 0x5 Slow mode divide ratio = 32
15:6	**RSRVD1**	RO	0x0	Reserved
5	**DIV_FRAC_EN**	RW	0x0	1 = Enable fractional divide. 0 = Enable integer divide.
4:0	**DIV**	RW	0x01	CLK_P-to-CLK_H divide ratio. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0.

DESCRIPTION:

This register controls the clock divider that generates the CLK_H, the clock used by the AHB and APBH buses, when HW_CLKCTRL_EMI_SYNC_MODE_EN = 0.

Note: Do not write register space when busy bit(s) are high.

Table 4-14. HW_CLKCTRL_PIX

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
CLKGATE	RSRVD2	BUSY	RSRVD1																DIV_FRAC_EN	DIV											

Table 4-15. HW_CLKCTRL_PIX Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**CLKGATE**	RW	0x1	CLK_PIX Gate. If set to 1, CLK_PIX is gated off. 0: CLK_PIX is not gated. When this bit is modified, or when it is high, the DIV field should not change its value. The DIV field can change ONLY when this clock gate bit field is low.
30	**RSRVD2**	RO	0x0	Always set to zero (0).
29	**BUSY**	RO	0x0	This read-only bit field returns a one when the clock divider is busy transfering a new divider value across clock domains.
28:13	**RSRVD1**	RO	0x0	Always set to zero (0).
12	**DIV_FRAC_EN**	RW	0x0	Reserved - Always set to zero (0).
11:0	**DIV**	RW	0x1	The Pixel clock frequency is determined by dividing the selected reference clock (ref_xtal or ref_pix) by the value in this bit field. This field can be programmed with a new value only when CLKGATE = 0. NOTE: The divider is set to divide by 1 at power-on reset. Do NOT divide by 0. Do not divide by more than 255.

DESCRIPTION:

This register controls the divider that generates the PIX (LCDIF) clock.

Note: Do not write register space when busy bit(s) are high.

EXAMPLE:

```
HW_CLKCTRL_PIX_WR(BF_CLKCTRL_PIX_DIV(40));
```

4.8.8 Synchronous Serial Port Clock Control Register Description

The SSP control register provides control for SSP clock generation.

HW_CLKCTRL_SSP 0x070

Table 5-193. HW_ICOLL_INTERRUPT86 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_INTERRUPT86_SET(0,0x00000001);
```

5.4.97 Interrupt Collector Interrupt Register 87 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_INTERRUPT87	0x690
HW_ICOLL_INTERRUPT87_SET	0x694
HW_ICOLL_INTERRUPT87_CLR	0x698
HW_ICOLL_INTERRUPT87_TOG	0x69C

Table 5-194. HW_ICOLL_INTERRUPT87

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	

Table 6-3. HW_DIGCTL_CTRL Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**RSVD3**	RO	0x0	Reserved.
30	**XTAL24M_GATE**	RW	0x0	If set to 1, disable the Digital Control Microseconds counter, STRB1MHZ. If set to 0, enable the Digital Control Microseconds counter..
29	**TRAP_IRQ**	RW	0x0	This bit is set when an AHB access occurs to the range defined by the TRAP_ADDR registers below and the trap function is enabled with the TRAP_ENABLE bit.
28:27	**RSVD2**	RO	0x0	Reserved.
26	**CACHE_BIST_TMODE**	RW	0x0	Set this bit to enable the Cache BIST test mode.
25	**LCD_BIST_CLKEN**	RW	0x0	Set this bit to enable the LCD memory BIST clock.
24	**LCD_BIST_START**	RW	0x0	Set this bit to start the LCD memory BIST.
23	**DCP_BIST_CLKEN**	RW	0x0	Set this bit to enable the DCP memory BIST clock.
22	**DCP_BIST_START**	RW	0x0	Set this bit to start the DCP memory BIST.
21	**ARM_BIST_CLKEN**	RW	0x0	Set this bit to enable the ARM BIST clock.
20	**USB_TESTMODE**	RW	0x0	Set this bit to get into USB test mode.
19	**ANALOG_TESTMODE**	RW	0x0	Set this bit to get into analog test mode.
18	**DIGITAL_TESTMODE**	RW	0x0	Set this bit to get into digital test mode.
17	**ARM_BIST_START**	RW	0x0	Set this bit to start the ARM cache BIST controller.
16	**UART_LOOPBACK**	RW	0x0	Set this bit to loop the two AUARTs back on themselves in a null modem configuration (as well as connect AUART1 to DUART). NORMAL = 0x0 No loopback. LOOPIT = 0x1 Internally connect AUART1 TX to AUART2 RX and DUART RX, also connect AUART2 TX to AUART1 RX (note that DUART TX is unaffected).
15	**SAIF_LOOPBACK**	RW	0x0	Set this bit to loop SAIF1 to SAIF2 and SAIF2 to SAIF1. To use SAIF loopback, configure one SAIF for transmit and the other for receive. Because this bit connects SAIF1 output to SAIF2 input and SAIF2 output to SAIF1 input, it does not matter which of the two ports is configured for TX and the other for RX. Either configuration will produce an internal TX to RX loopback. Note that SAIF_CLKMST_SEL is ignored when loopback is enabled. NORMAL = 0x0 No loopback. LOOPIT = 0x1 Loop SAIF1 and SAIF2 back to each other.

Table 8-36. HW_USBCTRL_USBSTS Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
4	**SEI**	RW	0x0	System Error. This bit is not used in this implementation and will always be set to 0.
3	**FRI**	RW	0x0	Frame List Rollover. The Host Controller sets this bit to a 1 when the Frame List Index rolls over from its maximum value to 0. The exact value at which the rollover occurs depends on the frame list size. For example. If the frame list size (as programmed in the Frame List Size field of the USBCMD register) is 1024, the Frame Index Register rolls over every time FRINDEX [13] toggles. Similarly, if the size is 512, the Host Controller sets this bit to a 1 every time FHINDEX [12] toggles. Only used by the host controller.
2	**PCI**	RW	0x0	Port Change Detect. The Host Controller sets this bit to a 1 when on any port a Connect Status occurs, a Port Enable/Disable Change occurs, or the Force Port Resume bit is set as the result of a J-K transition on the suspended port. The Device Controller sets this bit to a 1 when the port controller enters the full or high-speed operational state. When the port controller exits the full or highspeed operation states due to Reset or Suspend events, the notification mechanisms are the USB Reset Received bit and the DCSuspend bits respectively. This bit is not EHCI compatible.
1	**UEI**	RW	0x0	USB Error Interrupt (USBERRINT). When completion of a USB transaction results in an error condition, this bit is set by the Host/Device Controller. This bit is set along with the USBINT bit, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Section 4.15.1 in the EHCI specification for a complete list of host error interrupt conditions. The device controller detects resume signaling only.
0	**UI**	RW	0x0	USB Interrupt (USBINT). This bit is set by the Host/Device Controller when the cause of an interrupt is a completion of a USB transaction where the Transfer Descriptor (TD) has an interrupt on complete (IOC) bit set. This bit is also set by the Host/Device Controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes.

DESCRIPTION:

status

EXAMPLE:

```
Empty Example.
```

Table 11-33. HW_APBX_CH1_CMD Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**CHAIN**	RO	0x0	A value of one indicates that another command is chained onto the end of the current command structure. At the completion of the current command, this channel will follow the pointer in HW_APBX_CH1_CMDAR to find the next command.
1:0	**COMMAND**	RO	0x00	This bitfield indicates the type of current command: 00- NO DMA TRANSFER 01- write transfers, i.e. data sent from the APBX device (APB PIO Read) to the system memory (AHB master write). 10- read transfer 11- reserved NO_DMA_XFER = 0x0 Perform any requested PIO word transfers but terminate command before any DMA transfer. DMA_WRITE = 0x1 Perform any requested PIO word transfers and then perform a DMA transfer from the peripheral for the specified number of bytes. DMA_READ = 0x2 Perform any requested PIO word transfers and then perform a DMA transfer to the peripheral for the specified number of bytes.

DESCRIPTION:

The command register controls the overall operation of each DMA command for this channel. It includes the number of bytes to transfer to or from the device, the number of APB PIO command words included with this command structure, whether to interrupt at command completion, whether to chain an additional command to the end of this one and whether this transfer is a read or write DMA transfer.

EXAMPLE:

```
Empty Example.
```

11.5.16 APBX DMA Channel 1 Buffer Address Register Description

The APBX DMA Channel 1 buffer address register contains a pointer to the data buffer for the transfer. For immediate forms, the data is taken from this register. This is a byte address which means transfers can start on any byte boundary.

HW_APBX_CH1_BAR 0x1A0

Table 11-34. HW_APBX_CH1_BAR

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
ADDRESS																															

Table 11-35. HW_APBX_CH1_BAR Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:0	**ADDRESS**	RO	0x00000000	Address of system memory buffer to be read or written over the AHB bus.

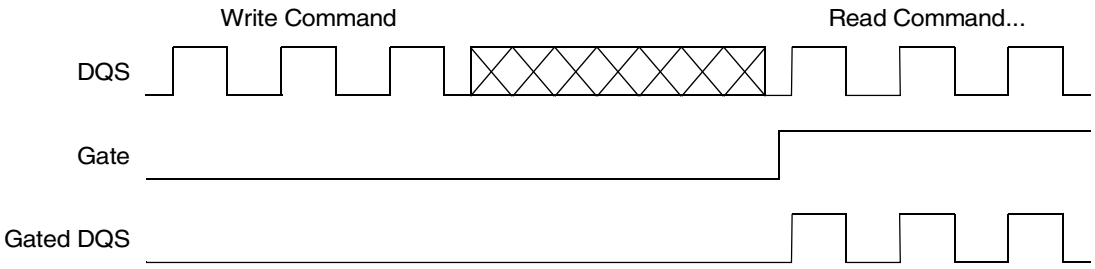


Figure 12-7. DQS Gating

The timing of when to start gating the DQS depends on the design itself, the flight time of the clock to memory, and the flight time of the data/DQS to the memory controller, as follows:

- If the round trip time is between ½ cycle and 1½ cycles, program the *caslat_lin* parameter equal to the *caslat* parameter.
- If the round trip time is less than ½ cycle, program the *caslat_lin* parameter one value less (which translates to ½ cycle) than the *caslat* parameter to open the gate ½ cycle sooner.
- If the round trip time is longer than 1½ cycles, program the *caslat_lin* parameter one value more (which translates to ½ cycle) than the *caslat* parameter to open the gate ½ cycle later.

In addition, the *caslat_lin_gate* parameter controls the opening of the gating signal. Nominally, *caslat_lin_gate* should have the same value as the *caslat_lin* parameter. However, to accommodate the skew of the memory devices, it may be necessary to open the gate a 1/2-cycle sooner or later. Adjusting the value of *caslat_lin_gate* modifies the gate opening by this factor.

There is a requirement that the DQS signals must be known and low when the memory controller is not driving. Because of the large variance in access times for the mobile devices, the gate for the DQS received by the memory controller must be active for longer than the period of time that the memory drives the DQS. Maintaining the DQS bus low when neither the memory controller nor the memory is driving ensures a clean DQS received by the memory controller.

12.2.3.2 mDDR Read Data Timing Registers

When using an mDDR external DRAM device, control of the read data timing is provided through multiple registers, as shown in Figure 12-8. First, the HW_DRAM_CTL04_DLL_BYPASS_MODE selects whether the DCC DLL circuitry is enabled or bypassed. Programming a 1 into this register disables the DLL auto-sync functionality and instead uses a fixed delay-chain select point programmed into the HW_DRAM_CTL19_DLL_DQS_DELAY_BYPASS1 and 0 bit fields. Programming a 0 into the DLL_BYPASS_MODE field enables the DLL auto-sync mode, utilizing the HW_DRAM_CTL18_DLL_DQS_DELAY_BYPASS1 and 0 values to define the percentage of the clock period of delay to add to the DQS inputs before being used as data capture controls.

The BYPASS_MODE or control bit is set based on the desired EMI_CLK frequency. At frequencies above 80 MHz, the BYPASS_MODE should be disabled, allowing the DLL to auto-sync. Frequencies below this point show enable the BYPASS_MODE.

Table 13-23. HW_GPMI_STAT Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
2	**DEV2_ERROR**	RO	0x0	0= No error condition present on NAND Device 2. 1= An Error has occurred on NAND Device 2 (Timeout or compare failure, depending on COMMAND_MODE).
1	**DEV1_ERROR**	RO	0x0	0= No error condition present on NAND Device 1. 1= An Error has occurred on NAND Device 1 (Timeout or compare failure, depending on COMMAND_MODE).
0	**DEV0_ERROR**	RO	0x0	0= No error condition present on NAND Device 0. 1= An Error has occurred on NAND Device 0 (Timeout or compare failure, depending on COMMAND_MODE).

DESCRIPTION:

The GPMI control and status register provides a read back path for various operational states of the GPMI controller.

EXAMPLE:

```
No Example.
```

13.4.12 GPMI Debug Information Register Description

The GPMI debug information register provides a read back path for diagnostics to determine the current operating state of the GPMI controller.

HW_GPMI_DEBUG 0x0C0

Table 13-24. HW_GPMI_DEBUG

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
READY3	**READY2**	**READY1**	**READY0**	**WAIT_FOR_READY_END3**	**WAIT_FOR_READY_END2**	**WAIT_FOR_READY_END1**	**WAIT_FOR_READY_END0**	**SENSE3**	**SENSE2**	**SENSE1**	**SENSE0**	**DMAREQ3**	**DMAREQ2**	**DMAREQ1**	**DMAREQ0**	**CMD_END**				**UDMA_STATE**				**BUSY**	**PIN_STATE**			**MAIN_STATE**			

Table 13-25. HW_GPMI_DEBUG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31	**READY3**	RO	0x0	Read-only view of Ready Line 3.
30	**READY2**	RO	0x0	Read-only view of Ready Line 2.
29	**READY1**	RO	0x0	Read-only view of Ready Line 1.
28	**READY0**	RO	0x0	Read-only view of Ready Line 0.

Table 14-7. HW_ECC8_DEBUG0

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							KES_DEBUG_SYNDROME_SYMBOL									KES_DEBUG_SHIFT_SYND	KES_DEBUG_PAYLOAD_FLAG	KES_DEBUG_MODE4K	KES_DEBUG_KICK	KES_STANDALONE	KES_DEBUG_STEP	KES_DEBUG_STALL	BM_KES_TEST_BYPASS	RSRVD0				DEBUG_REG_SELECT			

Table 14-8. HW_ECC8_DEBUG0 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:25	**RSRVD1**	RO	0x0	Reserved.
24:16	**KES_DEBUG_SYNDROME_SYMBOL**	RW	0x0	The 9-bit value in this bit field will be shifted into the syndrome register array at the input of the KES engine whenever HW_ECC8_DEBUG0_KES_DEBUG_SHIFT_SYND is toggled. NORMAL = 0x0 Bus master address generator for synd_gen writes operates normally. TEST_MODE = 0x1 Bus master address generator always addresses last four bytes in auxiliary block.
15	**KES_DEBUG_SHIFT_SYND**	RW	0x0	Toggling this bit causes the value in HW_ECC8_DEBUG0_KES_SYNDROME_SYMBOL to be shifted into the syndrome register array at the input to the KES engine. After shifting in 16 symbols, one can kick off both KES and CF cycles by toggling HW_ECC8_DEBUG0_KES_DEBUG_KICK. Be sure to set KES_ECC8_DEBUG0_KES_STANDALONE mode to 1 before kicking.
14	**KES_DEBUG_PAYLOAD_FLAG**	RW	0x0	When running the standalone debug mode on the error calculator, the state of this bit is presented to the KES engine as the input payload flag. DATA = 0x1 Payload is set for 512 byte data block. AUX = 0x1 Payload is set for 65 or 19 byte auxiliary block.
13	**KES_DEBUG_MODE4K**	RW	0x0	When running the standalone debug mode on the error calculator, the state of this bit is presented to the KES engine as the input mode (4K or 2K pages). 4k = 0x1 Mode is set for 4K NAND pages. 2k = 0x1 Mode is set for 2K NAND pages.
12	**KES_DEBUG_KICK**	RW	0x0	Toggling causes KES engine FSM to start as if kicked by the bus master. This allows standalone testing of the KES and Chien Search engines. Be sure to set KES_ECC8_DEBUG0_KES_STANDALONE mode to 1 before kicking.

Table 14-20. HW_ECC8_VERSION Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:24	**MAJOR**	RO	0x01	Fixed read-only value reflecting the MAJOR field of the RTL version.
23:16	**MINOR**	RO	0x0	Fixed read-only value reflecting the MINOR field of the RTL version.
15:0	**STEP**	RO	0x0000	Fixed read-only value reflecting the stepping of the RTL version.

DESCRIPTION:

This register indicates the RTL version in use.

EXAMPLE:

```
if (HW_ECC8_VERSION.B.MAJOR != 1) Error();
```

ECC8 Block v1.1, Revision 2.5

15.4.1.1 DMA Structure Code Example

The following code sample illustrates the coding for one write transaction involving 4096 bytes of data payload (eight 512-byte blocks) and 10 bytes of auxiliary payload (also referred to as *metadata*) to a 4K NAND page sitting on GPMI CS2.

```
//------------------------------------------------------------------------
// generic DMA/GPMI/ECC descriptor struct, order sensitive!
//------------------------------------------------------------------------
typedef struct {
  // DMA related fields
  unsigned int dma_nxtcmdar;
  unsigned int dma_cmd;
  unsigned int dma_bar;

  // GPMI related fields
  unsigned int gpmi_ctrl0;
  unsigned int gpmi_compare;
  unsigned int gpmi_eccctrl;
  unsigned int gpmi_ecccount;
  unsigned int gpmi_data_ptr;
  unsigned int gpmi_aux_ptr;
} GENERIC_DESCRIPTOR;

//------------------------------------------------------------------------
// allocate 9 descriptors for doing a NAND ECC Write
//------------------------------------------------------------------------
GENERIC_DESCRIPTOR write[9];

//------------------------------------------------------------------------
// DMA descriptor pointer to handle error conditions from psense checks
//------------------------------------------------------------------------
unsigned int * dma_error_handler;

//------------------------------------------------------------------------
// 8 byte NAND command and address buffer
// any alignment is ok, it is read by the GPMI DMA
//   byte 0 is write setup command
//   bytes 1-5 is the NAND address
//   byte 6 is write execute command
//   byte 7 is status command
//------------------------------------------------------------------------
unsigned char nand_cmd_addr_buffer[8];

//------------------------------------------------------------------------
// 4096 byte payload buffer used for reads or writes
// needs to be word aligned
//------------------------------------------------------------------------
unsigned int write_payload_buffer[(4096/4)];

//------------------------------------------------------------------------
// 65 byte meta-data to be written to NAND
// needs to be word aligned
//------------------------------------------------------------------------
unsigned int write_aux_buffer[65];

//------------------------------------------------------------------------
// Descriptor 1: issue NAND write setup command (CLE/ALE)
//------------------------------------------------------------------------
write[0].dma_nxtcmdar = &write[1];                          // point to the next descriptor

write[0].dma_cmd = BF_APBH_CHn_CMD_XFER_COUNT    (1 + 5)|   // 1 byte command, 5 byte address
                   BF_APBH_CHn_CMD_CMDWORDS      (3)    |   // send 3 words to the GPMI
                   BF_APBH_CHn_CMD_WAIT4ENDCMD   (1)    |   // wait for command to finish before continuing
                   BF_APBH_CHn_CMD_SEMAPHORE     (0)    |
                   BF_APBH_CHn_CMD_NANDWAIT4READY(0)    |
                   BF_APBH_CHn_CMD_NANDLOCK      (1)    |   // prevent other DMA channels from taking over
                   BF_APBH_CHn_CMD_IRQONCMPLT    (0)    |
                   BF_APBH_CHn_CMD_CHAIN         (1)    |   // follow chain to next command
                   BV_FLD(APBH_CHn_CMD, COMMAND, DMA_READ); // read data from DMA, write to NAND

write[0].dma_bar = &nand_cmd_addr_buffer;                  // byte 0 write setup, bytes 1 - 5 NAND address

// 3 words sent to the GPMI
write[0].gpmi_ctrl0 = BV_FLD(GPMI_CTRL0, COMMAND_MODE, WRITE)    | // write to the NAND
                      BV_FLD(GPMI_CTRL0, WORD_LENGTH,  8_BIT)    |
                      BV_FLD(GPMI_CTRL0, LOCK_CS,      ENABLED)  |
                      BF_GPMI_CTRL0_CS                 (2)       | // must correspond to NAND CS used
                      BV_FLD(GPMI_CTRL0, ADDRESS,      NAND_CLE) |
                      BF_GPMI_CTRL0_ADDRESS_INCREMENT  (1)       | // send command and address
                      BF_GPMI_CTRL0_XFER_COUNT         (1 + 5);    // 1 byte command, 5 byte address

write[0].gpmi_compare = NULL;                               // field not used but necessary to set eccctrl
```

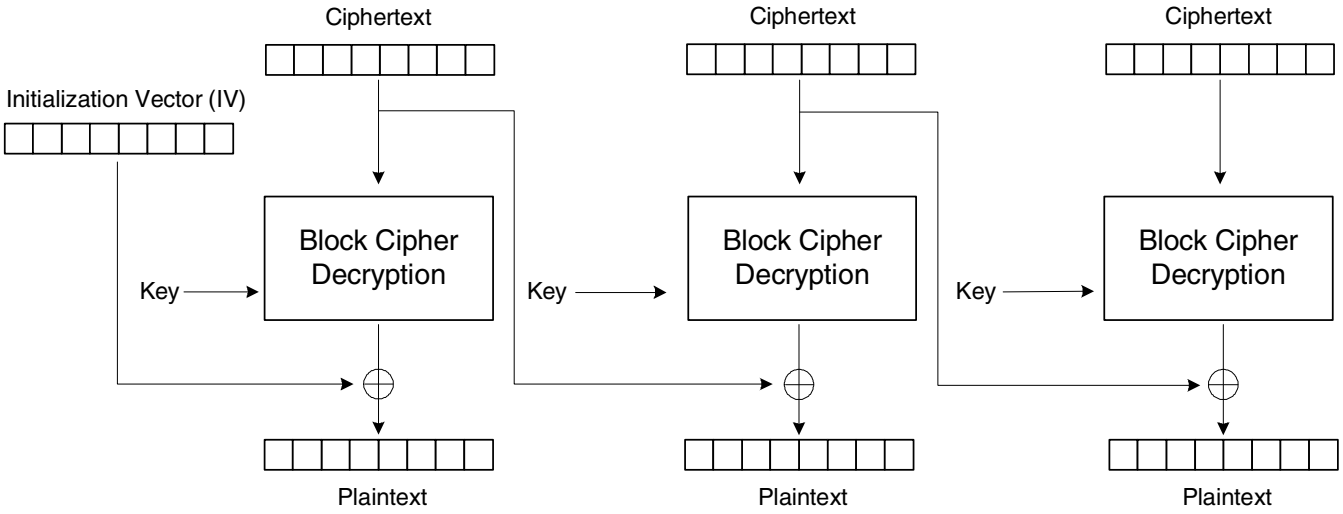


Figure 16-3. Cipher Block Chaining (CBC) Mode Decryption

16.2.3 Hashing

The hashing module implements the SHA-1 hashing algorithm and a modified CRC-32 checksum algorithm. These algorithms produce a signature for a block of data that can be used to determine whether the data is intact.

The CRC-32 algorithm implements a 32-bit CRC algorithm similar to the one used by Ethernet and many other protocols. The CRC differs from the Unix cksum() function in three ways:

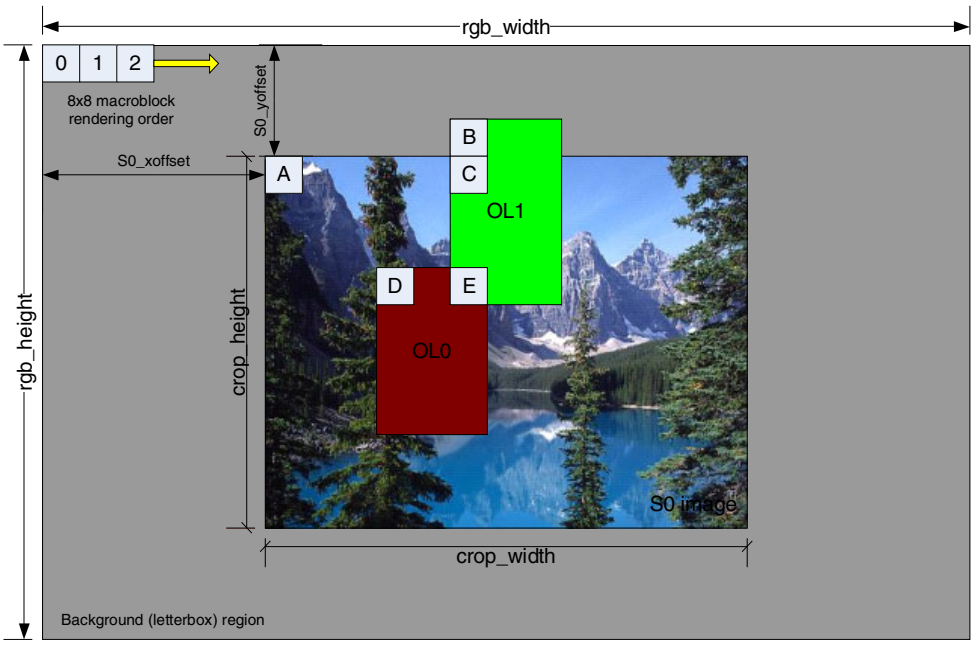
- The CRC is initialized as 0xFFFFFFFF instead of 0x00000000.
- Logic pads zeros to a 32-bit boundary for trailing bytes.
- Logic does not post-pend the file length.

The SHA-1 block implements a 160-bit hashing algorithm that operates on 512-bit (64-byte) blocks as defined by US FIPS PUB 180-1 in 1995. The purpose of the hashing module is to generate a unique signature for a block of data that can be used to validate the integrity of the data by comparing the resulting “digest” with the original digest.

Results from hash operations are written to the beginning of the payload for the descriptor. The DCP also has the ability to check the resulting hash against a value in the payload and issue an interrupt if a mismatch occurs.

16.2.4 Managing DCP Channel Arbitration and Performance

The DCP can have four channels compete for DCP resources to complete their operations. Depending on the situation, critical operations may need to be prioritized above less important operations to ensure smooth system operation. To help software achieve this goal, the DCP implements a programmable arbi-



0,1,2: Pixel blocks rendered with the background color. The numbering and arrow indicate the order of macroblock rendering.
A: S0 Image rendered
B: Overlay 1 blended with background
C: Overlay 1 blended with S0 image
D: Overlay 0 blended with S0 image
E: Overlay 0 blended with S0 image (OL0 takes precedence over OL1)

Figure 17-3. Pixel Pipeline (PXP) Macro Blocks

It is important to understand how the PXP renders each output macroblock to properly understand how it accomplishes cropping, letterboxing, and overlay blending. The following sections will provide more details on these operations.

The PXP also has the ability to rotate/flip images for cases when the pixel scan order is not in the traditional left-to-right/top-to-bottom raster scan (landscape raster). This can occur when a handheld device with a traditional landscape scan is rotated into a portrait orientation (in which the scan order is now bottom-to-top/left-to-right or vice versa) or when a cell phone oriented display (portrait raster) is rotated into a landscape orientation for viewing videos. In these cases, the PXP still renders the image in scan-order format (as sent to the device), but it will traverse the input images based on the transformations required.

The following sections detail each of the PXP's functional capabilities.

17.2.1 Pixel Handling

All pixels are internally represented as 24-bit RGB values with an 8-bit alpha value at all stages in the PXP after the colorspace converter. Input pixels are converted into this format using the following rules:

- 32-bit ARGB8888 pixels are read directly with no conversion for both the S0 and overlay images.
- 32-bit RGB888 pixels are assumed to have an alpha value of 0xFF (full opaque).
- 16-bit RGB565 and RGB555 values are expanded into the corresponding 24-bit colorspace and assigned an alpha value of 0xFF (opaque). The expansion process replicates the upper pixel bits

EXAMPLE:

```
Empty example.
```

19.4.2 TV Encoder Configuration Register Description

This is configuration register of the TV Encoder Block

HW_TVENC_CONFIG	0x010
HW_TVENC_CONFIG_SET	0x014
HW_TVENC_CONFIG_CLR	0x018
HW_TVENC_CONFIG_TOG	0x01C

Table 19-3. HW_TVENC_CONFIG

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
RSRVD5				DEFAULT_PICFORM	YDEL_ADJ			RSRVD4	RSRVD3	ADD_YPBPR_PED	PAL_SHAPE	NO_PED	COLOR_BAR_EN	YGAIN_SEL		CGAIN		CLK_PHS		RSRVD2	FSYNC_ENBL	FSYNC_PHS	HSYNC_PHS	VSYNC_PHS	SYNC_MODE			RSRVD1	ENCD_MODE		

Table 19-4. HW_TVENC_CONFIG Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
31:28	**RSRVD5**	RO	0x0	Always write zeroes to this bit field.
27	**DEFAULT_PICFORM**	RW	0x1	Permits use of a set of default parameters, tailored to the mode defined by T_ENCD_MODE, to be used in place of the values in the LINEx registers.
26:24	**YDEL_ADJ**	RW	0x4	Delays luma versus chroma for composite output. Luma lags chroma by YDEL_ADJ-4 cycles of 27MHz clock. For example, if YDEL_ADJ=0, the luma leads by 4 cycles. And if YDEL_ADJ=7, the luma lags by 3 cycles.
23	**RSRVD4**	RO	0x0	Always write zeroes to this bit field.
22	**RSRVD3**	RO	0x0	Enables Svideo output on DAC-B and DAC-D, not available on HuaShan.
21	**ADD_YPBPR_PED**	RW	0x0	Permits the insertion of a black pedestal when sync is inserted on one or more component signals.
20	**PAL_SHAPE**	RW	0x0	Set to impose a 250nS edge shape as required by PAL, otherwise the steeper edges as specified by NTSC are used.
19	**NO_PED**	RW	0x0	Can be set to prevent insertion of a black pedestal as required by NTSC-J.
18	**COLOR_BAR_EN**	RW	0x0	Enable insertion of internally generated color bars.
17:16	**YGAIN_SEL**	RW	0x0	Controls the luma gain: 00 : NTSC 01 : PAL 1x : no gain

- When an address is sent, each device in the system compares the first seven bits after a start condition with its address.
- If they match, the device considers itself addressed by the master.

Data transfer with acknowledge is obligatory.

- The transmitter must release the I2C_SDA line during the acknowledge pulse.
- The receiver must then pull the data line low, so that it remains stable low during the high period of the acknowledge clock pulse.
- A receiver that has been addressed is obliged to generate an acknowledge after each byte of data has been received.
- A slave device can terminate a transfer by withholding its acknowledgement.

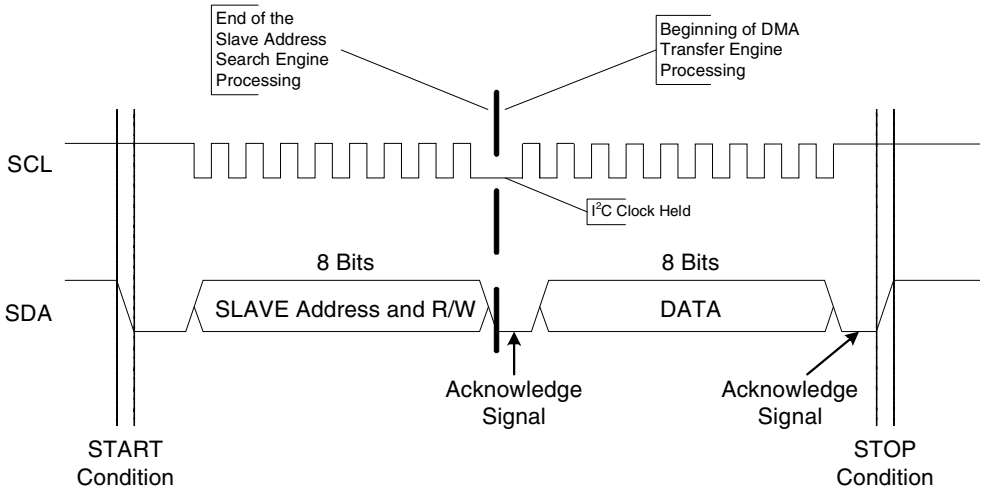


Figure 25-2. I²C Data and Clock Timing

The clock is generated by the master, according to parameters set in the HW_I2C_TIMINGn register. This register also provides programmable timing for capturing received data, as well as for changing the transmitted data bit.

For received words: 1) If the FIFOs are enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO; 2) if the FIFOs are not enabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO). The received data bytes (up to 4) are read by performing reads from the 32-bit DATA register. The status information can be read by a read of the UART Status register.

The Overrun Error bit is set to 1 if data is received and the receive FIFO is already full. This is cleared to 0 once there is an empty space in the FIFO and a new character can be written to it. The Break Error bit is set to 1 if a break condition was detected, indicating that the received data input was held LOW for longer than a full-word transmission time (defined as start, data, parity and stop bits). In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state), and the next valid start bit is received. When the Parity Error bit is set to 1, it indicates that the parity of the received data character does not match the parity selected as defined by bits 2 and 7 of the LCR_H register. In FIFO mode, this error is associated with the character at the top of the FIFO. When the Framing Error bit is set to 1, it indicates that the received character did not have a valid stop bit (a valid stop bit is 1). In FIFO mode, this error is associated with the character at the top of the FIFO.

EXAMPLE:

```
No Example.
```

26.4.8 UART Status Register Description

The UART Status Register contains the various flags and receive status. If the status is read from this register, then the status information for break, framing and parity corresponds to the data character read from the UART Data Register prior to reading the UART Status Register. The status information for overrun is set immediately when an overrun condition occurs.

HW_UARTAPP_STAT 0x070

Table 26-16. HW_UARTAPP_STAT

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
PRESENT	HISPEED	BUSY	CTS	TXFE	RXFF	TXFF	RXFE	RXBYTE_INVALID				OERR	BERR	PERR	FERR	RXCOUNT															

Table 32-19. HW_POWER_DCDC4P2 Bit Field Descriptions

BITS	LABEL	RW	RESET	DEFINITION
12:8	**BO**	RW	0x0	Brownout voltage in 25mV steps for the DCDC_4P2 pin. 0b00000 : 3.6V .. 0b11111 : 4.375V
7:5	**RSRVD1**	RO	0x0	Empty Description.
4:0	**CMPTRIP**	RW	0x18	Sets the trip point for the comparison between the DCDC_4P2 and BATTERY pin. When the comparator output is high then, the switching converter may use the DCDC_4P2 pin as the source for the switching converter, otherwise it will use the DCDC_BATT pin. 0b00000 DCDC_4P2 pin >= 0.85 * BATTERY pin 0b00001 DCDC_4P2 pin >= 0.86 * BATTERY pin 0b11000 DCDC_4P2 pin >= BATTERY pin (default) 0b11111 DCDC_4P2 pin >= 1.05 * BATTERY pin

DESCRIPTION:

Empty Description.

EXAMPLE:

```
Empty Example.
```

32.11.10 DC-DC Miscellaneous Register Description

This register contains controls that may need to be adjusted to optimize DC-DC converter performance using the battery voltage information

HW_POWER_MISC 0x090

Table 32-20. HW_POWER_MISC

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
RSRVD2																									FREQSEL			RSRVD1	DELAY_TIMING	TEST	SEL_PLLCLK

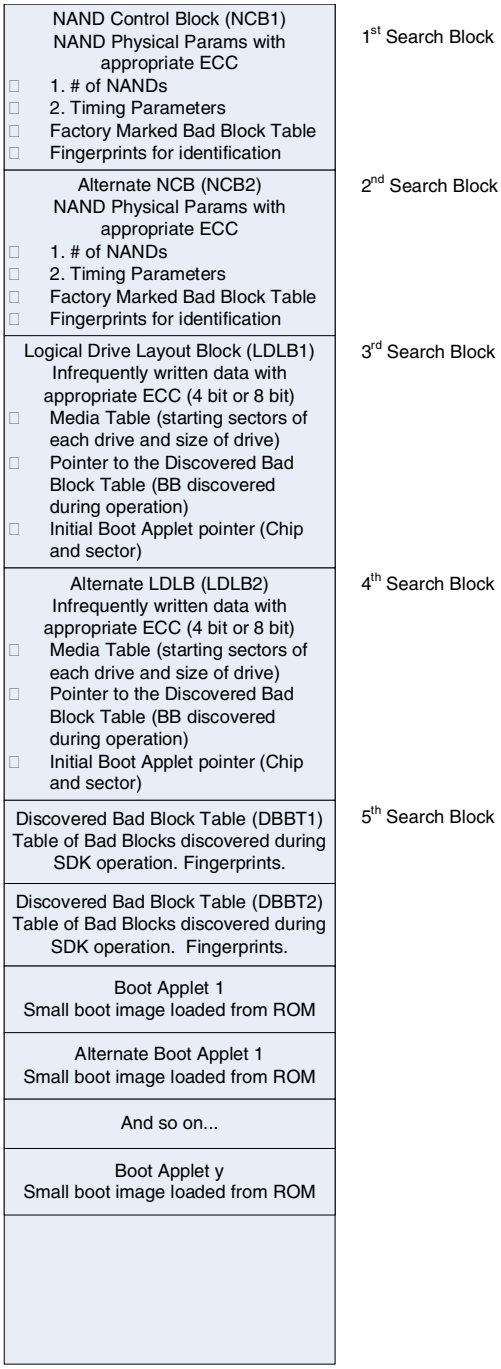


Figure 35-5. Expected NAND Layout

Search areas are defined as 64 pages * efSearchSize. They are defined in such a way that there is at least one extra block to hedge against a Boot Control Block (BCB) going bad during operation. If a Boot Control Block goes bad during operation, the data is copied from the original BCB to the extra BCB, and the original BCB is written with zeros. Since we rely upon both the ECC being correct and the fingerprints, overwriting the BCB with zeros should invalidate all the data. The search algorithm will search an entire search area before looking for the backup or secondary Boot Control Block.

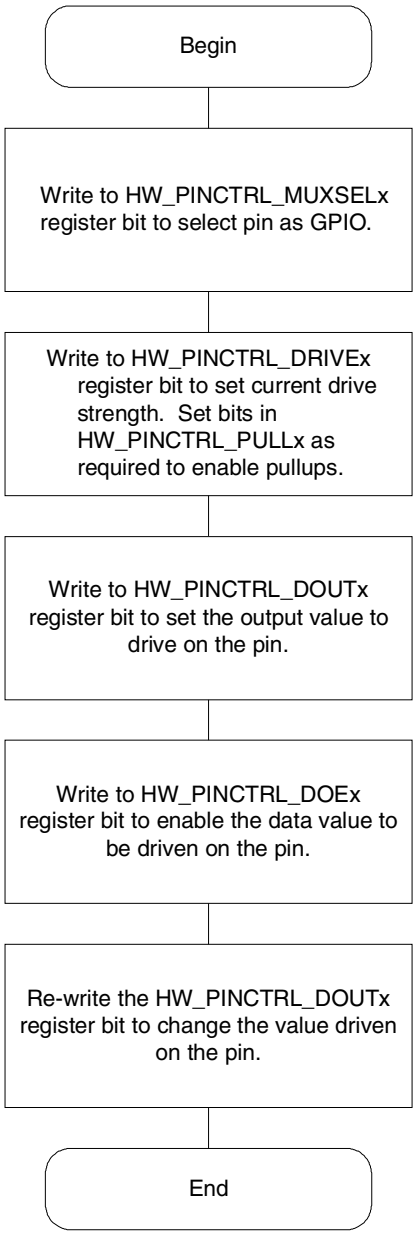


Figure 37-2. GPIO Output Setup Flowchart

37.2.3.2 Input Operation

Any (non-EMI high speed) digital pin may be used as a GPIO input by programming its HW_PINCTRL_MUXSELx field to 3 to enable GPIO mode, programming its HW_PINCTRL_DOEx field to 0 to disable output, and then reading from the HW_PINCTRL_DINx register, as shown in Figure 37-3. Note that because of clock synchronization issues, the logic levels read from the HW_PINCTRL_DINx registers are delayed from the pins by two APBX clock cycles.