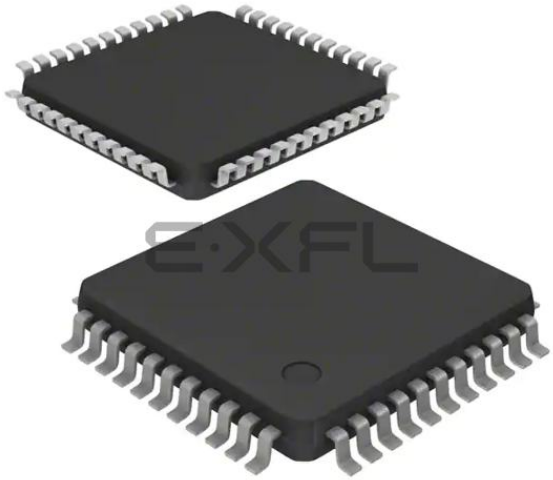

Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	eZ8
Core Size	8-Bit
Speed	24MHz
Connectivity	DALI, DMX, I²C, LINbus, SPI, UART/USART, USB
Peripherals	DMA, LVD, POR, PWM, WDT
Number of I/O	36
Program Memory Size	64KB (64K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	3.75K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	A/D 10x12b; D/A 1x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	44-LQFP
Supplier Device Package	44-LQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/zilog/z8f6481an024xk2246

9.4.4. Interrupt Request 3 Register

The Interrupt Request 3 (IRQ3) Register, shown in Table 55, stores interrupt requests for both vectored and polled interrupts. When a request is presented to the interrupt controller, the corresponding bit in the IRQ3 Register becomes 1. If interrupts are globally enabled (vectored interrupts), the interrupt controller passes an interrupt request to the eZ8 CPU. If interrupts are globally disabled (polled interrupts), the eZ8 CPU can read the Interrupt Request 3 Register to determine if any interrupt requests are pending.

Table 55. Interrupt Request 3 Register (IRQ3)

Bit	7	6	5	4	3	2	1	0
Field	AESI	MCTI	U1RXI	U1TXI	PC3I/ DMA3I	PC2I/ DMA2I	PC1I	PC0I
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Address	FC9h							

Bit	Description
[7] AESI	**AES Interrupt Request** 0: No interrupt request is pending for the AES. 1: An interrupt request from the AES is awaiting service.
[6] MCTI	**Multi-Channel timer Interrupt Request** 0: No interrupt request is pending for multi-channel timer. 1: An interrupt request from multi-channel timer is awaiting service.
[5] U1RXI	**UART 1 Receiver Interrupt Request** 0: No interrupt request is pending for the UART 1 receiver. 1: An interrupt request from the UART 1 receiver is awaiting service.
[4] U1TXI	**UART 1 Transmitter Interrupt Request** 0: No interrupt request is pending for the UART 1 transmitter. 1: An interrupt request from the UART 1 transmitter is awaiting service.
[3:2] PC*x*I/DMA*x*I	**Port C*x* or DMA *x* Interrupt Request** 0: No interrupt request is pending for GPIO Port C*x* or DMA *x*. 1: An interrupt request from GPIO Port C*x* or DMA *x* is awaiting service; *x* indicates the specific GPIO Port C bit or DMA number (3–2).
[1:0] PC*x*I	**Port C Pin *x* Interrupt Request** 0: No interrupt request is pending for GPIO Port C pin *x*. 1: An interrupt request from GPIO Port C pin *x* is awaiting service; *x* indicates the specific GPIO Port C pin number (1–0).

10.1.7. Timer 0–2 Noise Filter Control Registers

The Timer 0–2 Noise Filter Control (T*x*NFC) registers, shown in Table 87, enable and disable the Timer Noise Filter and set noise filter control.

Table 87. Timer 0–2 Noise Filter Control Registers (T*x*NFC)

Bit	7	6	5	4	3	2	1	0
Field	NFCTL				NFCON	Reserved		
Reset	0	0	0	0	0	0	0	0
R/W	W				W	R		
Address	T0NFC @ F2Ch, T1NFC @ F2Dh, T2NFC @ F2Eh							

Note: *x* references bits in the range [2:0].

Bit	Description
[7:4] NFCTL	**Noise Filter Control** This field controls the delay and noise rejection characteristics of the Noise Filter. The wider the counter the more delay that is introduced by the filter and the wider the noise event that will be filtered. 0000: The Noise Filter is disabled. Received inputs bypass the filter 0001: 2-bit up/down counter 0010: 3-bit up/down counter 0011: 4-bit up/down counter 0100: 5-bit up/down counter 0101: 6-bit up/down counter 0110: 7-bit up/down counter 0111: 8-bit up/down counter 1000: 9-bit up/down counter 1001: 10-bit up/down counter 1010: 11-bit up/down counter 1011–1111: Reserved.
[3] NFCON	**Noise Filter Connection** 0: Noise Filter connects to Timer(x) and filters timer Input 0 (T*x*IN ORed with Event System Timer(x) Input 0). 1: Noise Filter is reassigned to Timer(x+1) and filters Event System Timer(x+1) Input 1. Timer(x) Input 0 effectively bypasses the Noise Filter. In the case of Timer 2, its Noise Filter connects to Timer 0 Input 1. With this selection, the Noise Filter is reassigned to the designated timer and uses its timer clock.
[2:0]	**Reserved** This bit is reserved and must be programmed to 0.

Chapter 11. Multi-Channel Timer

The Multi-Channel timer has a 16-bit up/down counter and a 4-channel Capture/Compare/PWM channel array. This timer provides multiple synchronous Capture/Compare/PWM channels based on a single timer. The Multi-Channel Timer features include:

- 16-bit up/down timer counter with programmable prescale
- Selectable clock source (system clock or external input pin)
- Count Modulo and Count up/down counter modes
- Four independent capture/compare channels which reference the common timer
- Channel modes:
  - One-Shot Compare Mode
  - Continuous Compare Mode
  - PWM Output Mode
  - Capture Mode
- Event System and external input pin for timer input
- DMA request source

11.1. Architecture

Figure 21 shows the Multi-Channel Timer architecture.

13.5.13. Real-Time Clock Alarm Control Register

The RTC_ACTRL Register, shown in Table 118, contains control bits for the Real-Time Clock. This register is cleared by a Power-On Reset (POR).

Table 118. Real-Time Clock Alarm Control Register (RTC_ACTRL)

Bit	7	6	5	4	3	2	1	0
Field when MODE=0	Reserved			ADOM_EN	ADOW_EN	AHRS_EN	AMIN_EN	ASEC_EN
Field when MODE=1	Reserved			ADOM_EN	Reserved	AHRS_EN	AMIN_EN	ASEC_EN
Power-On Reset	0	0	0	0	0	0	0	0
CPU Access	R	R	R	R/W	R/W	R/W	R/W	R/W
Address	F3Ch							

Note: X=undefined; R=read-only; R/W=read/write

Bit	Description
Calendar Mode Operation (MODE=0)	
[7:5]	**Reserved** These bits are reserved and must be programmed to 000.
[4] ADOM_EN	**Alarm Day Of The Month Enable** 0: The day-of-the-month alarm is disabled. 1: The day-of-the-month alarm is enabled.
[3] ADOW_EN	**Alarm Day Of The Week Enable** 0: The day-of-the-week alarm is disabled. 1: The day-of-the-week alarm is enabled.
[2] AHRS_EN	**Alarm Hours Enable** 0: The hours alarm is disabled. 1: The hours alarm is enabled.
[1] AMIN_EN	**Alarm Minutes Enable** 0: The minutes alarm is disabled. 1: The minutes alarm is enabled.
[0] ASEC_EN	**Alarm Seconds Enable** 0: The seconds alarm is disabled. 1: The seconds alarm is enabled.

b. If Multiprocessor Mode is not enabled, then enable parity (if appropriate), and select either even or odd parity.

5. Check the RDA bit in the UART-LDD Status 0 Register to determine if the Receive Data Register contains a valid data byte (indicated by a 1). If RDA is set to 1 to indicate available data, continue to Step 6. If the Receive Data Register is empty (indicated by a 0), continue to monitor the RDA bit that is awaiting reception of the valid data.
6. Read data from the UART-LDD Receive Data Register. If operating in Multiprocessor (9-bit) Mode, further actions may be required depending on the Multiprocessor Mode bits MPMD[1:0].
7. Return to Step 5 to receive additional data.

14.1.5. Receiving Data Using the Interrupt-Driven Method

The UART-LDD Receiver interrupt indicates the availability of new data (as well as error conditions). Observe the following steps to configure the UART-LDD receiver for interrupt-driven operation:

1. Write to the UART-LDD Baud Rate High and Low Byte registers to set the appropriate baud rate.
2. Enable the UART-LDD pin functions by configuring the associated GPIO port pins for alternate function operation.
3. Execute a DI instruction to disable interrupts.
4. Write to the Interrupt Control registers to enable the UART-LDD Receiver interrupt and set the appropriate priority.
5. Clear the UART-LDD Receiver interrupt in the applicable Interrupt Request Register.
6. Write to the UART-LDD Control 1 Register to enable Multiprocessor (9-bit) Mode functions, if appropriate.
   a. Set the Multiprocessor Mode Select (MPEN) bit to enable Multiprocessor Mode.
   b. Set the Multiprocessor Mode Bits, MPMD[1:0] to select the appropriate address matching scheme.
   c. Configure the UART-LDD to interrupt on received data and errors or errors only (interrupt on errors only is unlikely to be useful for Z8 Encore! devices without a DMA block).
7. Write the device address to the Address Compare Register (automatic Multiprocessor modes only).
8. Write to the UART-LDD Control 0 Register to:
   a. Set the receive enable (REN) bit to enable the UART-LDD for data reception.

MPRX=1, a new frame begins. If the address of this new frame is different from the UART-LDD's address, then MPMD[0] must be set to 1 by software, causing the UART-LDD interrupts to go inactive until the next address byte. If the new frame's address matches the UART-LDD's address, then the data in the new frame is also processed.

The second scheme is enabled by setting MPMD[1:0] to 10b and writing the UART-LDD's address into the UART-LDD Address Compare Register. This mode introduces more hardware control, interrupting only on frames that match the UART-LDD's address. When an incoming address byte does not match the UART-LDD's address, it is ignored. All successive data bytes in this frame are also ignored. When a matching address byte occurs, an interrupt is issued and further interrupts occur on each successive data byte. The first data byte in the frame has NEWFRM=1 in the UART-LDD Status 1 Register. When the next address byte occurs, the hardware compares it to the UART-LDD's address. If there is a match, the interrupt occurs and the NEWFRM bit is set to the first byte of the new frame. If there is no match, the UART-LDD ignores all incoming bytes until the next address match.

The third scheme is enabled by setting MPMD[1:0] to 11b and by writing the UART-LDD's address into the UART-LDD Address Compare Register. This mode is identical to the second scheme, except that there are no interrupts on address bytes. The first data byte of each frame remains accompanied by a NEWFRM assertion.

14.1.10. LIN Protocol Mode

The Local Interconnect Network (LIN) protocol, as supported by the UART-LDD module, is defined in Revision 2.0 of the LIN Specification Package. The LIN protocol specification covers all aspects of transferring information between LIN master and slave devices using *message frames*, including error detection and recovery, SLEEP Mode and wake up from SLEEP Mode. The UART-LDD hardware in LIN Mode provides character transfers to support the LIN protocol including break transmission and detection, wake-up transmission and detection and slave autobauding. Part of the error detection of the LIN protocol is for both master and slave devices to monitor their receive data when transmitting. If the receive and transmit data streams do not match, the UART-LDD asserts the PLE bit (i.e., the physical layer error bit in the Status 0 Register). The *message frame* time-out aspect of the protocol depends on software requiring the use of an additional general-purpose timer. The LIN Mode of the UART-LDD does not provide any hardware support for computing/verifying the checksum field or verifying the contents of the identifier field. These fields are treated as data and are not interpreted by hardware. The checksum calculation/verification can easily be implemented in software via the Add with Carry (ADC) instruction.

The LIN bus contains a single Master and one or more slaves. The LIN master is responsible for transmitting the message frame header which consists of the Break, Synch and Identifier fields. Either the master or one of the slaves transmits the associated *response* section of the message which consists of data characters followed by a checksum character.

Chapter 15. Enhanced Serial Peripheral Interface

The Enhanced Serial Peripheral Interface (ESPI) supports the Serial Peripheral Interface (SPI) and Inter-IC Sound (I^2S). ESPI includes the following features:

- Full-duplex, synchronous, character-oriented communication
- Four-wire interface ($\overline{SS}$, SCK, MOSI and MISO)
- Transmit and receive buffer registers to enable high throughput
- Master Mode transfer rates up to a maximum of one-half the system clock frequency
- Slave Mode transfer rates up to a maximum of one-eighth the system clock frequency
- Error detection
- Dedicated Programmable Baud Rate Generator
- Data transfer control via polling, interrupt or DMA

15.1. Architecture

The ESPI is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface (serial clock, transmit data, receive data and slave select). The ESPI block consists of a shift register, data buffer register, a baud rate (clock) generator, control/status registers and a control state machine. Transmit and receive transfers are in synch because there is a single shift register for both transmitting and receiving data. Figure 37 shows a diagram of the ESPI block.

loses arbitration during the transmission of a data byte, it releases the SDA line and waits for the next stop or start condition.

The master detects a loss of arbitration when a 1 is transmitted but a 0 is received from the bus in the same bit-time. This loss occurs if more than one master is simultaneously accessing the bus. Loss of arbitration occurs during the address phase (two or more Masters accessing different slaves) or during the data phase, when the masters are attempting to write different data to the same slave.

When a master loses arbitration, the software is informed by means of the Arbitration Lost interrupt. The software can repeat the same transaction at a later time.

A special case can occur when a slave transaction starts just before the software attempts to start a new master transaction by setting the start bit. In this case, the state machine enters its slave states before the start bit is set and as a result the I^2C controller will not arbitrate. If a slave address match occurs and the I^2C controller receives/transmits data, the start bit is cleared and an Arbitration Lost interrupt is asserted. The software can minimize the chance of this instance occurring by checking the BUSY bit in the I2CSTATE Register before initiating a master transaction. If a slave address match does not occur, the Arbitration Lost interrupt will not occur and the start bit will not be cleared. The I^2C controller will initiate the master transaction after the I^2C bus is no longer busy.

16.2.5.2. Master Address-Only Transactions

It is sometimes preferable to perform an address-only transaction to determine if a particular slave device is able to respond. This transaction can be performed by monitoring the ACKV bit in the I2CSTATE Register after the address has been written to the I2CDATA Register and the start bit has been set. After the ACKV bit is set, the ACK bit in the I2CSTATE Register determines if the slave is able to communicate. The stop bit must be set in the I2CCTL Register to terminate the transaction without transferring data. For a 10-bit slave address, if the first address byte is acknowledged, the second address byte should also be sent to determine if the preferred slave is responding.

Another approach is to set both the stop and start bits (for sending a 7-bit address). After both bits have been cleared (7-bit address has been sent and transaction is complete), the ACK bit can be read to determine if the slave has acknowledged. For a 10-bit slave, set the stop bit after the second TDRE interrupt (which indicates that the second address byte is being sent).

16.2.5.3. Master Transaction Diagrams

In the following transaction diagrams, the shaded regions indicate the data that is transferred from the master to the slave, and the unshaded regions indicate the data that is transferred from the slave to the master. The transaction field labels are defined as follows:

S Start

W Write

Slave 10-Bit Address Recognition Mode. If IRM=0 during the address phase and the controller is configured for MASTER/SLAVE or SLAVE 10-BIT ADDRESS Mode, the hardware detects a match to the 10-bit slave address defined in the I2CMODE and I2CSLVAD registers and generates the slave address match interrupt (the SAM bit=1 in the I2CISTAT Register). The I^2C controller automatically responds during the Acknowledge phase with the value in the NAK bit of the I2CCTL Register.

16.2.6.2. General Call and Start Byte Address Recognition

If GCE=1 and IRM=0 during the address phase and the controller is configured for master/slave or slave in either 7- or 10-bit address modes, the hardware detects a match to the General Call Address or the start byte and generates the slave address match interrupt. A General Call Address is a 7-bit address of all zeroes, with the R/$\overline{W}$ bit=0. A start byte is a 7-bit address of all zeroes, with the R/$\overline{W}$ bit=1. The SAM and GCA bits are set in the I2CISTAT Register. The RD bit in the I2CISTAT Register distinguishes a General Call Address from a start byte which is cleared to 0 for a General Call Address). For a General Call Address, the I^2C controller automatically responds during the address acknowledge phase with the value in the NAK bit of the I2CCTL Register. If the software is set to process the data bytes associated with the GCA bit, the IRM bit can optionally be set following the SAM interrupt to allow the software to examine each received data byte before deciding to set or clear the NAK bit. A start byte will not be acknowledged – a requirement of the I^2C specification.

16.2.6.3. Software Address Recognition

To disable hardware address recognition, the IRM bit must be set to 1 prior to the reception of the address byte(s). When IRM=1, each received byte generates a receive interrupt (RDRF=1 in the I2CISTAT Register). The software must examine each byte and determine whether to set or clear the NAK bit. The slave holds SCL Low during the acknowledge phase until the software responds by writing to the I2CCTL Register. The value written to the NAK bit is used by the controller to drive the I^2C bus, then releasing the SCL. The SAM and GCA bits are not set when IRM=1 during the address phase, but the RD bit is updated based on the first address byte.

16.2.6.4. Slave Transaction Diagrams

In the following transaction diagrams, the shaded regions indicate data transferred from the master to the slave and the unshaded regions indicate the data transferred from the slave to the master. The transaction field labels are defined as follows:

S	Start
W	Write
A	Acknowledge
A	Not Acknowledge
P	Stop

Table 161. I2CSTATE_H

State Encoding	State Name	State Description
0000	Idle	I^2C bus is idle or I^2C controller is disabled.
0001	Slave Start	I^2C controller has received a start condition.
0010	Slave Bystander	Address did not match; ignore remainder of transaction.
0011	Slave Wait	Waiting for stop or restart condition after sending a Not Acknowledge instruction.
0100	Master Stop2	Master completing stop condition (SCL=1, SDA=1).
0101	Master Start/Restart	MASTER Mode sending start condition (SCL=1, SDA=0).
0110	Master Stop1	Master initiating stop condition (SCL=1, SDA=0).
0111	Master Wait	Master received a Not Acknowledge instruction, waiting for software to assert stop or start control bits.
1000	Slave Transmit Data	Nine substates, one for each data bit and one for the Acknowledge.
1001	Slave Receive Data	Nine substates, one for each data bit and one for the Acknowledge.
1010	Slave Receive Addr1	Slave receiving first address byte (7- and 10-bit addressing) Nine substates, one for each address bit and one for the Acknowledge.
1011	Slave Receive Addr2	Slave receiving second address byte (10-bit addressing) nine substates, one for each address bit and one for the Acknowledge.
1100	Master Transmit Data	Nine substates, one for each data bit and one for the Acknowledge.
1101	Master Receive Data	Nine substates, one for each data bit and one for the Acknowledge.
1110	Master Transmit Addr1	Master sending first address byte (7- and 10-bit addressing) nine substates, one for each address bit and one for the Acknowledge.
1111	Master Transmit Addr2	Master sending second address byte (10-bit addressing) nine substates, one for each address bit and one for the Acknowledge.

Table 162. I2CSTATE_L

State I2CSTATE_H	Substate I2CSTATE_L	Substate Name	State Description
0000–0100	0000	–	There are no substates for these I2CSTATE_H values.
0110–0111	0000	–	There are no substates for these I2CSTATE_H values.
0101	0000	Master Start	Initiating a new transaction
	0001	Master Restart	Master is ending one transaction and starting a new one without letting the bus go idle.

Bit	Description (Continued)
[1] HSNAK	**EP0 Handshake NA** HSNAK is automatically set when a Setup token arrives. Software clears HSNAK by writing a 1 to it. 0: Do not send a NAK handshake. 1: Send a NAK handshake for every packet in the Status stage.
[0] STALL	**EP0 Stall** STALL is automatically cleared when a Setup token arrives. 0: Do not send a STALL handshake. 1: Send a STALL handshake for any IN or OUT token during the data or handshake phases of the control transfer.

17.3.19. USB IN 0–3 Byte Count Subregisters

The USB IN 0–3 Byte Count subregisters, shown in Table 188, contain the USB IN endpoint byte counts.

Table 188. USB IN 0–3 Byte Count Subregisters (USBIxBC)

Bit	7	6	5	4	3	2	1	0
Field	Reserved	BC						
Reset	0	0	0	0	0	0	0	0
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	R
Address	If USBSA = 35h, 37h, 39h, 3Bh in the USB Subaddress Register, accessible through the USB Subdata Register							

Bit	Description
[7]	**Reserved** This bit is reserved and must be programmed to 0.
[6:0] BC	**IN Byte Count** 00–40: After loading the IN endpoint x buffer memory, software should write BC with the number of bytes loaded. Writing to the USBIxBC Register arms the IN endpoint x by setting BUSY in USBIxCS. When the host sends an IN token for IN endpoint x and BUSY is set, the USB Module will transmit a BC length data packet. 41-7F: Reserved.

16.2.7. DMA Control of I2C Transactions – see page 326

17.2.10. USB Module Interrupts and DMA – see page 355

20.2.1. AES Operation and DMA – see page 426

21.2.8. ADC Interrupts and DMA – see page 450

22.2.4. DAC Interrupt and DMA – see page 468

18.2.4. Transfer Types

Three transfer types provide Register Bus bandwidth-sharing options, and are selected with the BURST bit in the DMA Control Register. If no DMA requests are asserted, the CPU has 100% of the Register Bus bandwidth.

18.2.4.1. Block (BURST=00)

The DMA Controller can be configured to transfer data blocks by selecting BURST=00. It will transfer the entire transfer length as long as the DMA request is asserted.

The CPU will not execute instructions while the DMA Controller is transferring the block. The DMA Controller will pause to allow CPU execution, but only if the DMA requestor does not continue to assert DMA requests during the block transfer. Care should be taken using block transfer if CPU response time is critical.

18.2.4.2. Burst4 (BURST=01)

The DMA Controller can be configured to limit data transfer length to bursts of 4 transfers by selecting BURST=01. After 4 consecutive DMA transfers, the CPU is allowed to execute an instruction; i.e., one CPU instruction is interleaved with a burst of 4 DMA transfers.

If the requesting DMA does not require all four transfers and deasserts a DMA request, CPU instruction execution will occur after the last required transfer.

18.2.4.3. Single (BURST=10)

The DMA Controller can be configured to limit data transfer length to a single transfer by selecting BURST=10. After a DMA transfer, the CPU will execute one instruction; i.e., one CPU instruction is interleaved with each DMA transfer).

18.2.5. Direct Operation

18.2.5.1. DMA Setup with Autoincrement

The DMA Subregister selection and status registers have an autoincrement that allows the DMA Controller to be set up without modifying the DMASA subregister address in the DMAxSA Register. To enable autoincrementing, set the AUTOINC bit in the DMA Control Register. DMASA is autoincremented whenever DMA*x*SD is accessed (i.e., read or written) while DMAx is not active. This autoincrement allows for convenient channel setup, because software can sequentially write to the DMA channel subregisters without

20.2.6. Decrypt Key Derivation

The Round 10 (R[10]) expanded encryption key can be used as the decryption key for decrypting data sent using the encryption key. This decryption key can be derived and made available for retrieval by performing a decrypt key derivation using MODE=11. For this operation, real or dummy plain text data can be used. When the operation is completed, the derived 16-byte R[10] decryption key can be read from the AESDATA Register and stored for use as a decryption key. The first key byte read is the most significant byte (associated with s(0,0) in Figure 63 – see page 425).

Whenever MODE is written to 11, KEYLD is cleared. The decryption key will be derived only if the encryption key was loaded while MODE=11 which sets KEYLD=1. After a decrypt key derivation is completed, the decryption key is available to be read. When any other confidentiality mode is selected, the decryption key can no longer be read without again setting MODE=11, loading the encryption key, and starting/completing the decrypt key derivation.

The following example outlines a procedure for deriving and retrieving the decryption key.

1. Write the AESCTRL Register as follows: AES_EN=1, MODE=11 (KEYGEN), DECRYPT=0, AUTODIS=1.
2. Write the AESKEY Register with the encryption key.
3. Write real or dummy data to the AESDATA Register, or use DMA.
4. Set the START/BUSY bit in the AESSTAT Register, or use auto-start by setting AUTODIS=0 in Step 1.
5. Poll the START/BUSY bit, or use an interrupt.
6. Read the R[10] key from the AESDATA Register and store it.
7. Select another MODE.

20.3. AES Register Definitions

The AES accelerator is accessed through the registers listed in Table 228. The remainder of this chapter describes each of these registers.

20.3.5. AES Status Register

TheAES Status Register is shown in Table 233.

Table 233. AES Status Register (AESSTAT)

Bit	7	6	5	4	3	2	1	0
Field	START/ BUSY	Reserved			ERROR	IVLD	KEYLD	DATALD
Reset	0	0	0	0	0	0	0	0
R/W	R/W1*	R	R	R	R	R	R	R
Address	FBBh							

Note: *R/W1=Writing a 1 clears this bit.

Bit	Description
[7] START/ BUSY	**AES Start/Busy Status** 0: AES is idle or the requested encryption/decryption operation is complete. 1: Write 1 to start encryption/decryption, will remain 1 (Busy) till operation is complete and clears when finished.
[6:4]	**Reserved** These bits are reserved and must be programmed to 000.
[3] ERROR	**ERROR Status** 0: No error occurred during processing. 1: Error occurred during processing.
[2] IVLD	**Initialization Vector Load Status** 0: Initialization vector not fully loaded. 1: Initialization vector fully loaded.
[1] KEYLD	**Key Load Status** 0: Key not fully loaded. 1: Key fully loaded.
[0] DATALD	**Data Load Status** 0: Data not fully loaded. 1: Data fully loaded.

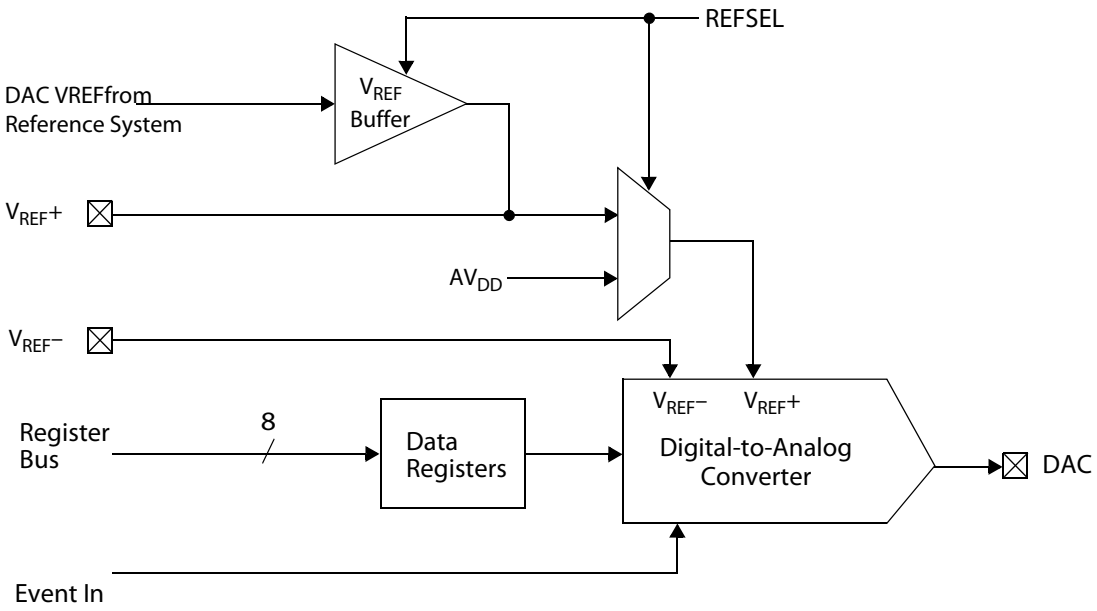

Figure 78. Digital-to-Analog Converter Block Diagram

22.2. Operation

The DAC is enabled by setting the DAC bit in the PWRCTL1 Register, which is described in the Low-Power Modes chapter on page 50. The DAC converts the digital input, DACDH and DACDL, in the DAC data registers, DACD_H and DACD_L, to an analog output level.

Data can be right-justified or left-justified, as selected by the JUSTIFY bit in the DAC-CTL Register. If data is left-justified, 8-bit resolution can be achieved by writing only the Data High Register, DACD_H.

The data format can be either unsigned (binary) or signed (two's-complement), as selected by the DFORMAT bit in the DACCTL Register. As shown in Figure 79, for unsigned data, 000h represents V_{REF}–, 7FFh represents midrange, and FFFh represents V_{REF}+. The equation for determining the analog level with unsigned data can be calculated as:

$$\text{DAC Output} = (V_{REF}+ - V_{REF}-) \times (\text{data} \div 4095)$$

In this equation, *data* represents the value of {DACDH, DACDL}.

26.3.6. LCD Control 2 Register

The LCDCTL2 Register, shown in Table 272, provides memory status and control, interrupt control and control of the LCD mode and waveform type. Writes to this register take effect at the end of the current waveform.

Table 272. LCD Control 2 Register (LCDCTL2)

Bits	7	6	5	4	3	2	1	0
Field	MSTAT	IRQS	LCDMODE				DMMODE	
Reset	0	0	0	0	0	0	0	0
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Address	FB6h							

Bit	Description
[7] MSTAT	**LCD Memory Status** 0: LCD Display Memory Bank A is currently the source for the LCD Controller outputs. 1: LCD Display Memory Bank B is currently the source for the LCD Controller outputs.
[6] IRQS	**Interrupt Request Select** 0: Frame interrupt. For static, 1/2, and 1/4 duty, the interrupt occurs at 7/8 frame for type A waveforms (LCDMODE[2]=1) and 7/4 frame for type B waveforms (LCDMODE[2]=0). For 1/3 duty, the interrupt occurs at 5/6 frame for type A waveforms (LCDMODE[2]=1) and 5/3 frame for type B waveforms (LCDMODE[2]=0). 1: Blink Interrupt. When DMMODE=10, interrupt occurs when the LCD controller switches from LCD Memory Bank A to LCD Memory Bank B. When a blink mode is selected (BMODE = 01, 10, 11), interrupt occurs at the transition from displayed to blank.
[5:2] LCDMODE	**LCD Operating Mode** 0000: 1 common (COM0), 1/1 duty, static bias, static waveform. 0001: 2 commons (COM[1:0]), 1/2 duty, 1/2 bias, type A waveform. 0010: 2 commons (COM[1:0]), 1/2 duty, 1/2 bias, type B waveform. 0011: 2 commons (COM[1:0]), 1/2 duty, 1/3 bias, type A waveform. 0100: 2 commons (COM[1:0]), 1/2 duty, 1/3 bias, type B waveform. 0101: 3 commons (COM[2:0]), 1/3 duty, 1/2 bias, type A waveform. 0110: 3 commons (COM[2:0]), 1/3 duty, 1/2 bias, type B waveform. 0111: 3 commons (COM[2:0]), 1/3 duty, 1/3 bias, type A waveform. 1000: 3 commons (COM[2:0]), 1/3 duty, 1/3 bias, type B waveform. 1001: 4 commons (COM[3:0]), 1/4 duty, 1/2 bias, type A waveform. 1010: 4 commons (COM[3:0]), 1/4 duty, 1/2 bias, type B waveform. 1011: 4 commons (COM[3:0]), 1/4 duty, 1/3 bias, type A waveform. 1100: 4 commons (COM[3:0]), 1/4 duty, 1/3 bias, type B waveform. Others: Reserved.

munication will only work when using an external clock source. To operate in high-speed synchronous mode, simply Auto-Baud to the desired speed. The Auto-Baud generator will automatically run at the desired baud rate.

Slow bus rise times due to the pull-up resistor become a limiting factor when operating at high speeds. To compensate for slow rise times, the output driver can be configured to drive the line High. If the Transmit Drive (TXD) bit is set, the line will be driven both High and Low during transmission. The line starts being driven at the beginning of the start bit and stops being driven at the middle of the stop bit. If the Transmit Drive High (TXDH) bit is set, the line will be driven High until the input is High or until the center of the bit occurs, whichever occurs first. If both TXD and TXDH are set, the pin will be driven High for one clock period at the beginning of each 0-to-1 transition. An example of a high-speed synchronous interface is shown in Figure 101.

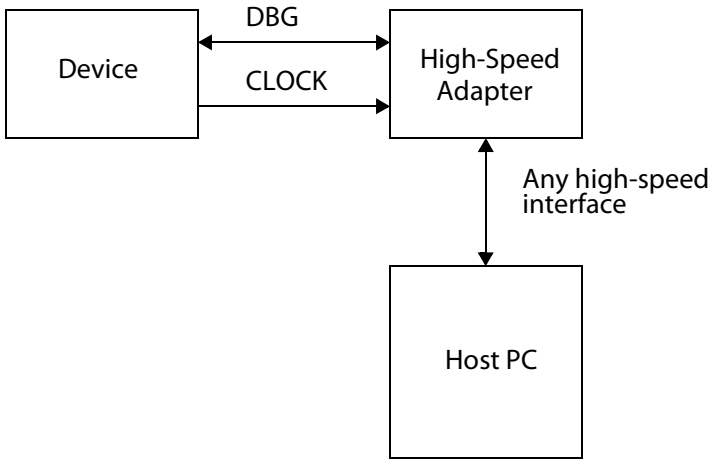

Figure 101. Synchronous Operation

30.2.6. OCD Serial Errors

The On-Chip Debugger can detect any of the following error conditions on the DBG pin:

- Serial Break (a minimum of ten continuous bits Low)
- Framing Error (the received stop bit is Low)
- Transmit Collision (OCD and host simultaneous transmission detected by the OCD)

When the OCD detects one of these errors, it aborts any command currently in progress, transmits a Serial Break 4096 system clock cycles long back to the host, and resets the Auto-Baud Detector/Generator. A Framing Error or Transmit Collision can be caused by the host sending a Serial Break to the OCD. Because of the open-drain nature of the inter-

Table 308. OCD Control Register (OCDCTL)

Bit	7	6	5	4	3	2	1	0
Field	DBGMODE	BRKEN	DBGACK	BRKLOOP	BRKPC	BRKZRO	Reserved	RST
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit	Description
[7] DBGMODE	**Debug Mode** Setting this bit to 1 causes the device to enter Debug Mode. When in Debug Mode, the eZ8 CPU stops fetching new instructions. Clearing this bit causes the eZ8 CPU to resume execution. This bit is automatically set when a BRK instruction is decoded and breakpoints are enabled. 0: The device is running (operating in Normal Mode). 1: The device is in Debug Mode.
[6] BRKEN	**Breakpoint Enable** This bit controls the behavior of BRK instruction (op code 00h). By default, breakpoints are disabled and the BRK instruction behaves like a NOP. If this bit is set to 1 and a BRK instruction is decoded, the OCD takes action depending upon the BRKLOOP bit. 0: BRK instruction is disabled. 1: BRK instruction is enabled.
[5] DBGACK	**Debug Acknowledge** This bit enables the debug acknowledge feature. If this bit is set to 1, then the OCD sends a Debug Acknowledge character (FFh) to the host when a breakpoint occurs. This bit automatically clears itself when an acknowledge character is sent. 0: Debug Acknowledge is disabled. 1: Debug Acknowledge is enabled.
[4] BRKLOOP	**Breakpoint Loop** This bit determines what action the OCD takes when a BRK instruction is decoded and breakpoints are enabled (BRKEN is 1). If this bit is 0, the DBGMODE bit is automatically set to 1 and the OCD enters Debug Mode. If BRKLOOP is set to 1, the eZ8 CPU loops on the BRK instruction. 0: BRK instruction sets DBGMODE to 1. 1: eZ8 CPU loops on BRK instruction.
[3] BRKPC	**Break when PC == OCDCNTR** If this bit is set to 1, then the OCDCNTR Register is used as a hardware breakpoint. When the program counter matches the value in the OCDCNTR Register, DBGMODE is automatically set to 1. If this bit is set, the OCDCNTR Register does not count when the CPU is running. 0: OCDCNTR is setup as counter. 1: OCDCNTR generates hardware break when PC == OCDCNTR.

Table 320. CPU Control Instructions (Continued)

Mnemonic	Operands	Instruction
STOP	–	Stop Mode
WDT	–	Watchdog Timer Refresh

Table 321. Load Instructions

Mnemonic	Operands	Instruction
CLR	dst	Clear
LD	dst, src	Load
LDC	dst, src	Load Constant to/from Program Memory
LDCI	dst, src	Load Constant to/from Program Memory and Autoincrement Addresses
LDE	dst, src	Load External Data to/from Data Memory
LDEI	dst, src	Load External Data to/from Data Memory and Autoincrement Addresses
LDWX	dst, src	Load Word using Extended Addressing
LDX	dst, src	Load using Extended Addressing
LEA	dst, X(src)	Load Effective Address
POP	dst	Pop
POPX	dst	Pop using Extended Addressing
PUSH	src	Push
PUSHX	src	Push using Extended Addressing

Table 322. Logical Instructions

Mnemonic	Operands	Instruction
AND	dst, src	Logical AND
ANDX	dst, src	Logical AND using Extended Addressing
COM	dst	Complement
OR	dst, src	Logical OR
ORX	dst, src	Logical OR using Extended Addressing
XOR	dst, src	Logical Exclusive OR
XORX	dst, src	Logical Exclusive OR using Extended Addressing