Product Update

Errata for eZ80F92 and eZ80F93 MCUs

Issues Related to eZ80F92 and eZ80F93 Microcontrollers

The errata listed in Table 1 highlights the issues and workarounds (if available) related to eZ80F92 and eZ80F93 microcontrollers.

Table 1. Errata to eZ80F92 and eZ80F93 Devices

No	Issue	Detailed Description
1	The infrared encoder/ decoder (endec) receiver misses bits when configured for low-data rates and the incoming signals are only 1.6 µs.	The infrared endec samples the incoming IR pulses, using the baud rate clock divided by 16. This sampling rate is not sufficient to capture the incoming pulses when they use a short-pulse format and low-data rates. This short pulse, 1.6 µs, is within IrDA specifications; however, not all transmitters use this particular signalling format. When the external transmitter is sending $^{3}/_{16}$ IR pulses, the endec on the eZ80F92 and eZ80F93 devices receive the data properly.
2	The real time clock (RTC) consumes excess current when the eZ80F92 and eZ80F93 MCUs are not in SLEEP mode.	When the eZ80F92 and eZ80F93 devices are not in SLEEP mode, the system clock drives the RTC's control and data registers. As a result, excess current is consumed through the RTC_V_{DD} pin that supplies power to the certain portion of the clock tree that drives the RTC registers. This current consumption is a function of operating frequency. Typical current consumption is 500 µA at 20 MHz.
3	Reading from Flash memory during ERASE or ROW PROGRAM operations.	The on-chip Flash memory controller automatically obstructs Flash memory Reads issued during ERASE and ROW PROGRAM operations only when configured for one or more Wait states. When the Flash memory controller is set for zero wait states, such Reads can return incorrect data and can cause corruption to the contents of Flash memory. **Workarounds** 1. Execution of row programming from Flash memory is not supported. However, before executing a Flash ERASE or ROW PROGRAM, configure Flash for one or more wait states. After the ERASE or ROW PROGRAM operation is completed, Flash memory can again be configured for zero wait states. 2. If executing a Flash ERASE from RAM or external memory, the user code should monitor either the PG_ERASE/MASS_ERASE flags in the FLASH_PGCTL register or the DONE flag in the FLASH_IRQ register. The user code waits for the ERASE or ROW PROGRAM operation to complete before proceeding.

Table 1. Errata to eZ80F92 and eZ80F93 Devices (Continued)

No	Issue	Detailed Description
4	A pulse on the SCL line when the I^2C bus is idle and the SDA line is held High causes the I^2C to lock.	A pulse on the SCL line prior to a START condition or after a STOP condition causes the I^2C bus to lock. This situation occurs regardless of the I^2C control register ENAB settings (I2C_CTL). If this situation occurs, an I^2C software reset does not unlock the I^2C bus. **Workarounds** 1. To prevent a lock from occurring: it is possible to completely disable the I^2C block prior to any bus activity using the Clock Peripheral Power-Down Register 1 (CLK_PPD1). Disable the I^2C block before setting ENAB in the I^2C Control register. 2. If a lock occurs after the SCL line is released, another device on the I^2C bus can issue a Stop by pulsing the SDA line. As a result, the I^2C should unlock. Another option is to initiate an overall SYSTEM RESET of the eZ80F91 device to reset the I^2C block.
5	RTC count errors of seconds, minutes, and hours can occur during eZ80F92 and eZ80F93 power-up.	The eZ80F92 and eZ80F93 MCUs each contain a private test mode register powered by V_{DD}. The test mode enables fast RTC counting (for production test) and is synchronously reset by the system clock. The test register is not tied down during reset. If this register powers up in a state that enables test mode, the RTC counters will start incrementing in fast mode until the register is reset by the first toggles of the system clock. **Workaround** During eZ80F92 and eZ80F93 power-up, gate off the RTC clock source and hold external RESET active (at least for three system clock periods) after V_{DD} ramps up to 3.3 V and the system clock source is stable.

Table 1. Errata to eZ80F92 and eZ80F93 Devices (Continued)

No	Issue	Detailed Description
6	GPIO edge-trigger interrupt mapping error	For edge triggered interrupts (Mode 6 and Mode 9), erroneous logic dependencies for the interrupt clearing logic exist on all port pins within each of the specific ports. To achieve proper interrupt clearing behavior for a particular port pin its mirror pin must be programmed in a similar manner. This affects how the designer utilizes GPIO alternate function pins with GPIO interrupt modalities of those port pins. The definition *mirrored pin* refers to any port in which for Port X, pin 0 is mirrored to pin 7, pin 1 is mirrored to pin 6, pin 2 is mirrored to pin 5, and pin 3 is mirrored to pin 4. **Note:** X is defined as Port B, C, or D. For example, if PB0 is programmed as an edge triggered interrupt, the logic dependency to clear the interrupt by writing to PB0_DR and protecting the actual PB0_DR register value from change comes from the mirror pin PB7 logic. This is an errata problem which causes erratic behavior problems. In the above example, the problem is that PB0_DR itself can be altered and might change the mode of operation for the port pin PB0. To correctly set up the logic dependencies, the mirrored pin must be placed in the same mode as its counterpart. As the functionally of these port pins need to be mirrored in order to correct the logic dependency, the alternate function assignments of these ports would not work correctly. To use the SPI alternate function modality (for example, SPI alternate function pins PB2, PB3, PB6, and PB7) you will not be able to use the mirror port pins PB5, PB4, PB1, and PB0 for Mode 6 and Mode 9 interrupt and vice versa. Any Port pin configured with Mode 6 or Mode 9 (an edge triggered interrupt) exhibits this behavior and affects the alternate function modality. The mirror mapping affects all Ports, specifically within the respective port pin pairs 0 and 7, 1 and 6, 2 and 5, and 3 and 4. **Note:** This design flaw in no way affects the IVECT address for the GPIO interrupts. **Workaround** Below is an example setup: PB0 Input, falling edge interrupt PB1 Input, dual edge interrupt PB2 Input, falling edge interrupt (continued)

Table 1. Errata to eZ80F92 and eZ80F93 Devices (Continued)

No	Issue	Detailed Description
		(continued from previous page) PB3 Input, falling or rising edge interrupt, depending on hardware configuration PB4 Input (not used) PB5 Input, falling edge interrupt PB6 Input, dual edge interrupt PB7 Input, falling edge interrupt: could be Rising, Dual or left floating (OPEN connection) Using the above example, if PB0 is Input, falling edge interrupt, then PB7 must be a separate input, EDGE MODE interrupt. In this example, PB7 is setup as a falling edge interrupt the same way as PB0. This additional separate input interrupt signal connected to PB7 could be rising, falling, dual, or you can leave it unconnected (OPEN) as well. You must not use PB7 for GPIO I/O or LEVEL sensitive interrupts. This same thought process is applicable to all Port bits pairs, specifically bits 0 and 7, 1 and 6, 2 and 5, and 3 and 4.

Table 1. Errata to eZ80F92 and eZ80F93 Devices (Continued)

No	Issue	Detailed Description

7	The UART is continually interrupting; the user cannot clear the interrupt.	The root cause of this issue has been duplicated with certain signal conditions after which a software instruction was executed to clear the receive FIFO. The signals involved were from bit 0 of the ISR, and the trigger counter with RXFIFO enabled and RXINTERRUPT disabled.
		Workarounds
		To prevent this error condition from occurring, you can perform one of the following two actions:
		(1)Do not enable the transmit or receive FIFO if it is not required. The Receive FIFO was the initial problem; however, both the TX and RX FIFOs are affected. Set bit 0 (FIFOEN) of the UART0_FCTL (0x0C2h) or UART1_FCTL (0x0D2h) registers to a value of zero.
		(2)If you are using either the transmit or receive FIFOs, mask off the following two bit locations to avoid changing the default bit value of zero. If bit 0 (FIFOEN) of the UART0_FCTL (0x0C2h) or UART1_FCTL (0x0D2h) registers is set to 1, then mask off bit 1 (CLRRxF) and bit 2 (CLRTxF) so that any Write accesses to the UART0_FCTL (0x0C2h) or UART1_FCTL (0x0D2h) registers will not alter this default zero value.
		(3)To correct this error condition when the UART Rx interrupt occurs but there is no Rx data detected, the user can clear the condition in software by putting the UART in loopback mode and then transmitting a single character. The following is an example of this workaround:
		`//====================workaround====================`
		`/*`
		`* Check for 'stuck' Fifo`
		`*/`
		`if( (Iir == SD_IIR_RX_INT) && ((Lsr & LSR_DR) == ) )`
		`{`
		`UINT32 Mcr;`
		
		`/*`
		`* To clear this condition, put the Uart in loopback`
		`* mode and send a character`
		`*/`
		`Mcr = BSP_RD32 ( pUART->Base|UART_REG_MCTL );`
		`BSP_WR32( pUart->Base|UART_REG_MCTL, Mcr | MCTL_LOOP);`
		`BSP_WR32( pUart->Base|UART_REG_THR, 'Z' );`
		`/*`
		`* Wait for the character to hit the Rx fifo`
		`*/`
		`Lsr = BSP_RD32( pUart->Base | UART_REG_LSR );`
		`while( !(Lsr & LSR_DR) )`
		`{`
		`Lsr = BSP_RD32( pUart->Base | UART_REG_LSR );`
		`}`
		(continued)

Table 1. Errata to eZ80F92 and eZ80F93 Devices (Continued)

No	Issue	Detailed Description

		(continued)
		`/*`
		`* Ignore any data trapped in the Rx fifo`
		`*/`
		`while( Lsr & LSR_DR )`
		`{`
		`Lsr = BSP_RD32( pUart->Base | UART_REG_RBR );`
		`Lsr = BSP_RD32( pUart->Base | UART_REG_LSR );`
		`}`
		
		`/*`
		`* Return to normal operation`
		`*/`
		`BSP_WR32( pUart->Base|UART_REG_MCTL, Mcr | MCTL_LOOP);`
		
		`/*`
		`* Record this event`
		`*/`
		`StuckRxCount++;`
		`}`
		
		`//==================normal handling==================`

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZiLOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZiLOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer