Product Update

Errata for eZ80Acclaim!® eZ80F91 Silicon

eZ80F91 MCU with Part Numbers and Date Codes 0317 and Later

The errata listed in Table 1 highlights the issues and workarounds (if available) for Zilog's eZ80F91 product with package part numbers eZ80F91AZ050SG, eZ80F91AZ050EG, eZ80F91NA050SG, and eZ80F91NA050EG illustrating date codes of 0317 and later, indicating assembly no earlier than the 17th week of the year 2003. Data in this document is Preliminary Only.

Table 1. Errata to the eZ80F91 Device with Date Codes 0317 and Later

No	Issue	Detailed Description
1	The real time clock (RTC) consumes excess current when the eZ80F91 device is not in SLEEP mode.	When the eZ80F91 device is not in SLEEP mode, the system clock drives the RTC control and data registers. As a result, excess current is consumed through the RTC_V_{DD} pin, which supplies power to the portion of the clock tree that drives the RTC registers. This current consumption is a function of the operating frequency. The faster the frequency, the more power is consumed. **Workaround** To prevent the excess current consumption, design with a battery-charging circuit for the RTC_V_{DD} pin. As a result, the charge will remain on the RTC battery while power is supplied to the chip.
2	WP pin functionality.	Assertion of the WP pin protects the Flash Memory boot block from memory Writes. It also protects all blocks and the information page from Page Erases, Mass Erases, and I/O Writes. **Workarounds** The following two options are available: (1) Do not assert the WP pin in your design. (2) De-assert the WP pin when you are executing page erases and I/O Writes to blocks other than Block 0 or executing a page erase or I/O Write of the Information Page.Then re-assert the pin when you are finished, for example, jumper block header.
3	When the RTC oscillator is disabled, RTC_X_{IN} pin leakage can exceed the 10 µA specification.	The RTC_X_{IN} pin can leak to a potential higher than the approximate 1.2V due to the internal 10 MΩ resistor. This can cause the input leakage current to exceed the specification. **Workarounds** To prevent this excess leakage, you can perform one of the following two actions, first being the recommended fix: (1) Leave the RTC_X_{IN} pin floating, if the oscillator is disabled. (2) Ground the RTC_X_{IN} pin, if the oscillator is disabled and not used at all.
4	Superseded by errata number 7.	This issue related to RTC time loss was determined to be the same as errata No. 7. To avoid potential confusion, it is no longer detailed here.

Table 1. Errata to the eZ80F91 Device with Date Codes 0317 and Later (Continued)

No	Issue	Detailed Description
5	The Test Data Input (TDI) pin does not meet the 1149.1 boundary scan specification.	**Warning:** TDI is an I/O pin without a pull-up resistor. NORMAL mode, following RESET, configures TDI as an input. An off-chip pull-up resistor is recommended. **Workaround** Add a 10 KΩ pull-up resistor to the TDI pin.
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 where for Port X, pin 0 is mirrored to pin 7, pin 1 is mirrored to pin 6, pin 2 is mirrored to pin 5, pin 3 is mirrored to pin 4. **Note:** X is defined as Port A, 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. (continued)

Table 1. Errata to the eZ80F91 Device with Date Codes 0317 and Later (Continued)

No	Issue	Detailed Description
		(continued from previous page) **Workaround** Below is an example setup: PB0 Input, falling edge interrupt PB1 Input, dual edge interrupt PB2 Input, falling edge interrupt 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 – This could be Rising, Dual, or even 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.
7	The RTC loses large amounts of time at spurious intervals. This loss of time may vary between software compiler runs.	The RTC clock divider counter can potentially be reset anytime the eZ80® internal address bits [7:0] transition to or through the hex value 0xED while the internal I/O WRITE_ENABLE (IOWR) signal is asserted (Active Low). **Workaround** This RTC problem can be triggered by any eZ80 CPU instruction that causes an I/O Write operation (such as OUT0) to any I/O mapped register or location. If the last byte of the I/O Write operation resides at the address location xxxxEC, the next available address is then 0xED and the I/O WRITE_ENABLE signal may deactivate during the address transition. If this does happen, the RTC clock divider counter gets reset. To avoid this problem you have to shift these instructions down with a NOP instruction to avoid the last byte of the I/O instruction landing on the 0xEC address. There are 17 instructions that can be affected: (continued)

Table 1. Errata to the eZ80F91 Device with Date Codes 0317 and Later (Continued)

No	Issue	Detailed Description

		(continued from previous page)
		`out0 (n),a`
		`outd2r` `oti2r` `out(n),a`
		`otdm` `otim` `outd`
		`otdmr` `otimr` `outd2`
		`otdr` `otir` `outi`
		`otdrx` `otirx` `outi2`
		`out (bc),a`

		You must iterate through the following 4-step sequence until all the I/O instructions are located at a safer address location. The steps in the sequence below use the OUT0 instruction as an example. This instruction is 3 bytes long. Therefore, the starting address for an OUT0 to cause the RTC problem would be 3 bytes back from address `xxxxEC` = `xxxxEA`. The following assembly code snippet might look like this:
		
		`ld A, 02`
		`OUT0 (%4D), A`
		
		After the linker process, assume that the same code snippet can be found in the hex file search as follows:
		
		`0003E8 3E ; This is the ld A,02 instruction.`
		`0003E9 02`
		`0003EA ED ; This is the OUT0 (%4D), A instruction.`
		`0003EB 39`
		`0003EC 4D`
		`0003ED 3E ; This is the start of another LD instruction.`
		
		To move the OUT0 last byte away from address 0x0003EC, perform the following sequence:
		(1) Post-process the complete software build so that the addresses for the executable code are resolved.
		(2) Search for the first OUT0 instruction that starts with address location `xxxxEA`.
		(3) If the instruction traces back from the hex address through the map file to the C or .asm function, return to the source code and add a NOP instruction just before the OUT0 instruction. If an assembly source code, just add a NOP instruction. If C source code, just add the in-line ASM statement `_asm("\tNOP\n");`.
		(4) Recompile, link, and re-inspect the hex output file, starting at step 2.

Table 1. Errata to the eZ80F91 Device with Date Codes 0317 and Later (Continued)

No	Issue	Detailed Description

8	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 (CLR-RxF) 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, you 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 the eZ80F91 Device with Date Codes 0317 and Later (Continued)

No	Issue	Detailed Description

		(continued from previous page)
		`/*`
		`* 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==================`

9	The timer misses timer End-of-Count (EOC).	The root cause for the eZ80F91 timer to lose track of timer EOC has been traced to an internal signal within the timer block that is preventing timer EOC. This issue affects only the timer functions of the eZ80F91 MCU. No other eZ80Acclaim!® devices are affected. The timer EOC is blocked when three conditions are met during the execution of user software, as follows. (1) The lower 8 bits of the internal eZ80þaddress bus are equal to the specific timer control register (TMRx_CTL) where x = 0, 1, 2, or 3. The internal data bus value is odd, meaning that bit 0 is equal to 1. (2) The specific timer block in which the control register address identified in the first condition above has timed out (generated a time-out EOC). If all of the above internal conditions are met, the timer EOC is missed. How the internal data bus is driven depends on the current instruction. For the `LD A,n` instruction, the internal data bus is driven with immediate data. If the `LD A,01` instruction is placed at an address where the lower 8 bits are equal to the timer control register address 01, the HoldTimerEnable signal is asserted as the 01 is being fetched. If the timer is about to time out, the timer EOC is missed. If the fetched data value is an even number, the problem is not visible, and the timer EOC is not missed. (continued)

Table 1. Errata to the eZ80F91 Device with Date Codes 0317 and Later (Continued)

No	Issue	Detailed Description

		(continued from previous page)
		Additionally, a timer configured for SINGLE PASS mode will fail to stop and instead will reload and continue counting after an occurrence above three conditions. If the conditions do not occur at the subsequent EOC, the timer will correctly stop.
		All timer functions that utilize the EOC to determine their result are affected. The input capture, event count, and output capture work correctly since they do not rely on the timer EOC signal.
		Workaround
		To prevent this error from occurring (and the only way to guarantee that you do not miss any timer EOC), use two timers programmed to run in synchronization so that their time-out EOC occur at the same time. The key element is to ensure that these timer EOC occur at the same time and on the same interval.
		As a result, you are guaranteed that, if the first timer has met all three conditions, the second timer will not have met the first condition, and therefore does not miss an EOC, and vice versa. Any other method that results in removing one of the three conditions above will ensure that the you will not miss any timer EOC, but the task becomes painful and user-code-dependent due to selectable wait states, etc.
		For example, you can determine missed timer EOC using timer 3 code for timer 3 usage and look for out instructions that provide an odd value on the data bus. Once this code has been located, the you can move the timer code by using a NOP instruction.
		Assume you use a timer 3 and has written the following ASM code segments:
		`;******** Init Timer3 ********************`
		` ld hl,250 * (CPU_Freq / 1000) / 4000`
		` out0 (TMR3_RR_L),l`
		` out0 (TMR3_RR_H),h`
		` ld a,%07`
		` out0 (TMR3_CTL),a;Timer0 Enable`
		` ld a,%01`
		` out0 (TMR3_IER),a;Timer0 Int. Enable`
		` ei`
		` jp testproc`
		`...`
		`testproc:`
		` ld a,%01`
		`..`
		(continued)

Table 1. Errata to the eZ80F91 Device with Date Codes 0317 and Later (Continued)

No	Issue	Detailed Description

		(continued from previous page)
		The assembly listing for this code could be similar to the following:
		`A 40 ;******** Init Timer3`
		`***************`
		`000084 21350C00 A 41 ld hl,250 * (CPU_Freq / 10`
		`000088 ED2977 A 42 out0 (TMR3_RR_L),l`
		`00008B ED2178 A 43 out0 (TMR3_RR_H),h`
		`00008E 3E07 A 44 ld a,%07`
		`000090 ED3974 A 45 out0 (TMR3_CTL),a`
		`000093 3E01 A 46 ld a,%01`
		`000095 ED3975 A 47 out0 (TMR3_IER),a`
		`000098 FB A 48 ei`
		`000099 C3 72 04 00 A 49 jp testproc`
		`...`
		`...`
		`000471 A 68 testproc:`
		`000471 3E01 A 69 ld a,%01`
		
		Two key items to be aware of are:
		(1) The address in the software is 0471, which is close to TMR3_CTL(0x0074h).
		(2) The LD a, %01 instruction in bringing in an odd data byte. One solution in this case is for you is to insert a NOP instruction at address 000471, thereby pushing the testproc statement down 1 byte to the following:
		
		`000471 A 68 testproc:`
		`000471 00 A 69 NOP`
		`000472 3E01 A 70 ld a,%01`

10	The infrared encoder/ decoder (endec) receiver misses bits when configured for low data rates and the incoming signals are only 1.6 µs.	The endec samples the incoming IR pulses, using the baud rate clock divided by 16. This sampling rate can be insufficient 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 eZ80F91 device receive the data properly.
11	GPIO Port B and Port C pins draw current when input voltage exceeds one diode drop above the supply voltage.	For the 5 V input-tolerant GPIO pins, when the input voltage exceeds approximately 4.0 V (for a 3.3 V supply voltage), current is drawn by the input pin. **Workaround** For GPIO pins that toggle at low frequencies, a 10 KΩ resistor can be placed between the GPIO pin and the external driver. This placement limits the current into the pin to about 150 µA. For higher frequencies, a 1KΩ resistor should be used; the resulting input current to the pin is limited to about 1.5 mA.

Table 1. Errata to the eZ80F91 Device with Date Codes 0317 and Later (Continued)

No	Issue	Detailed Description
12	A pulse on the Serial Clock (SCL) line while the I^2C bus is idle and the Serial Data (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 to lock up. This situation occurs irrespective of the I^2C Control Register ENAB settings (I2C_CTL). When this situation occurs, an I^2C Software Reset does not unlock the I^2C bus. **Workarounds** To prevent lock-up, 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) before setting ENAB in the I^2C Control Register. In the event that lock-up of the I^2C block occurs, then after the SCL line is released, another device on the bus can issue a STOP condition 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.
13	Disable of PLL when selected as system clock source.	If the PLL is selected as the system clock source, the PLL can be disabled by the eZ80 CPU. As a result, the system clock stops. The eZ80F91 will not respond to RESET pin assertion without a system clock. If the system clock source is disabled, the eZ80F91 can only be reset by a WDT reset, a POR reset, or a RTC SLP_WAKE.

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