# Application Note

## Understanding the eZ80 Interrupt Structure and Initializing Interrupts in C and Assembly

AN010003-1101

ZiLOG Worldwide Headquarters • 910 E. Hamilton Avenue • Campbell, CA  95008
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.zilog.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

**ZiLOG Worldwide Headquarters**
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com



## Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

## Document Disclaimer

## *Table of Contents*

## *List of Figures*

## *List of Tables*

**Acknowledgements**

Application and Support Engineers

Mark Thissen

System and Code Development

Mark Thissen

## Introduction

This application note iss a reference on embedded software for programmers to understand, initialize and use interrupts on the eZ80 embedded web server. This application note, together with other eZ80 tools assists the programmer, either new to or already familiar with ZiLOG programming, to use eZ80 interrupts. The projects described in this application note pertain to the eZ80 Evaluation board with the Realtek Ethernet controller. This application note contains tables with setup information to enable the evaluation board that uses the Crystal LAN Ethernet controller as well. The user must become familiar with the settings and set up the memory map appropriately for whichever evaluation board is in use.

This application note addresses the following topics:

- Discussion of interrupt structure, assembly language initialization, and C language initialization
- Application examples for both languages

Also included are source code files for timer interrupts for both Assembly and C project files, schematics, and a customer feedback form.

## Discussion

### Interrupt Structure

The eZ80 family of devices are capable of 128 vectored priority interrupts from both internal and external sources and one non-maskable interrupt (NMI). The eZ80190 specifically is capable of providing only 43 of these vectored priority interrupts. The eZ80190 does not support Z80 interrupt modes IM0, IM1, or IM2 because of the non-availability of the INT pin. All priority vectors are returned on the eZ80190 internal vector bus (IVECT7:0). The eZ80190 supports NMI (Non-Maskable Interrupt) and 43 IO interrupts. NMI has the highest priority and, as it's name implies, can not be masked. NMI comes from a hardware pin and always vectors to address 0x000066. Here you can code a jump vector into memory to service the NMI request. This application note does not go into additional detail of the NMI. For more information on NMI, see the eZ80 User's Manual. The vectors and sources for the 43 I/O interrupts are listed in order of priority in Table 1. "Vector Table Setup" on page 5 lists the 43 internal sources, in order of their priority, along with an explanation of the vector location in memory and their respective setup in assembly language programming.

### How the eZ80 Fetches an Internal Interrupt Vector

The following events happen when a vectored interrupt occurs:

- The 8-bit vector (Table 1) is read from the internal Vector bus (IVECT7:0)
- IEF1 and IEF2 (Interrupt Enable Flag) are reset to 0

What happens next depends on the chosen operating mode. See Table 2.

Table 1. I/O Interrupts

| Vector | Source | Vector | Source | Vector | Source | Vector | Source |
|---|---|---|---|---|---|---|---|
| 00h | MACC | 1Ch | Port A3 | 38h | Port C1 | 54h | Port D7 |
| 02h | DMA0 | 1Eh | Port A4 | 3Ah | Port C2 | Vectors 56h through 126h are Reserved and must be coded to point to a null interrupt vector. | |
| 04h | DMA1 | 20h | Port A5 | 3Ch | Port C3 | | |
| 06h | PRT0 | 22h | Port A6 | 3Eh | Port C4 | | |
| 08h | PRT1 | 24h | Port A7 | 40h | Port C5 | | |
| 0Ah | PRT2 | 26h | Port B0 | 42h | Port C6 | | |
| 0Ch | PRT3 | 28h | Port B1 | 44h | Port C7 | | |
| 0Eh | PRT4 | 2Ah | Port B2 | 46h | Port D0 | | |
| 10h | PRT5 | 2Ch | Port B3 | 48h | Port D1 | | |
| 12h | UZI0 | 2Eh | Port B4 | 4Ah | Port D2 | | |
| 14h | UZI1 | 30h | Port B5 | 4Ch | Port D3 | | |
| 16h | Port A0 | 32h | Port B6 | 4Eh | Port D4 | | |
| 18h | Port A1 | 34h | Port B7 | 50h | Port D5 | | |
| 1Ah | Port A2 | 36h | Port C0 | 52h | Port D6 | | |

Terms and Definitions for Modes and Operations

- I[7:0] = The contents of the interrupt vector register (I)
- IVECT[7:0] = The contents of the eZ80's internal vector bus
- MBASE = A programmable offset used in virtual Z80 mode
- PC(15:0) or PC(23:0) = The short or long contents of the program counter

- SPL = Stack Pointer Long (24bit),SPS = Stack Pointer Short (16bit)
- ADL = Address/Data Long,  MADL = Mixed ADL

Table 2. Vectored Interrupt Operation

| Current Memory Mode | MADL Control Bit | ADL Mode Bit | Operation |
|---|---|---|---|
| Z80 Mode | 0 | 0 | The Starting program counter is: {MBASE[1], PC(15:0)[2]}<br>• Push the 2-byte return address, PC(15:0), onto the stack, {MBASE,SPS[3]}.<br>• The ADL[4] Mode bit remains cleared to 0.<br>• Interrupt Vector Address is: {MBASE, I[7:0][5], IVECT[7:0][6]}. Therefore, PC(15:0)←{MBASE, I[7:0], IVECT[7:0]}<br>• The final program counter (PC [15:0]) is therefore: {MBASE, I[7:0], IVECT[7:0]}<br>• The Interrupt Service Routine must end with RETI. |
| ADL Mode | 0 | 1 | The Starting program counter is: PC(23:0).<br>• Push the 3-byte return address, PC(23:0), onto the SPL[7] stack.<br>• The ADL Mode bit remains set to 1.<br>• Interrupt Vector Address is: {00h, I[7:0], IVECT[7:0]}. Therefore, PC(23:0←{00h, I[7:0], IVECT[7:0]}<br>• The final program counter (PC [23:0]) is therefore: {00h, I[7:0], IVECT[7:0]}<br>• The Interrupt Service Routine must end with RETI. |

Notes:
1. MBASE = A programmable offset used in Z80 mode.
2. PC(15:0) or PC(23:0) = The short or long contents of the program counter.
3. SPS = Stack Pointer Short (16-bit)
4. ADL = Address/Data Long.
5. I[7:0] = The contents of the interrupt vector register (I).
6. IVECT[7:0] = The contents of the eZ80's internal ector bus.
7. SPL = Stack Pointer Long (24-bit).
8. MADL = Mixed ADL.

Table 2. Vectored Interrupt Operation (Continued)

| Current Memory Mode | MADL Control Bit | ADL Mode Bit | Operation |
|---|---|---|---|
| Z80 Mode | 1 | 0 | The Starting program counter is: (MBASE, PC(15:00)}<br>• Push the 2-byte return address, PC(15:0), onto the SPL stack.<br>• Push a 20h byte onto the SPL stack, indicating interrupting from Z80 mode (because MADL - 1 and ADL = 0).<br>• Set the ADL bit to 1.<br>• Interrupt Vector Address is: {00h, 1[7:0], IVECT[7:0]}<br>• The final progrma counter (PC [23:0]) is therefore: {00h, I[7:0], IVECT[7:0]}<br>• The Interrupt Service Routine muss end with RETI.L. |
| ADL Mode | 1 | 1 | The Starting program counter is: PC(23:0).<br>• Push the 3-byte return address, PC(23:0), onto the SPL stack.<br>• Push a 03 byte onto the SPL stack, indicating an interrupt from ADL mode (because MADL[8] = 1 and ADL = 1).<br>• The ADL Mode bit remains set to 1.<br>• Interrupt Vector Address is: {00h, I[7:0], IVECT[7:0]}. Therefore, PC(23:0)←{00h, I[7:0], IVECT[7:0]}<br>• The final program counter (PC [23:0]) is therefore: {00h, I[7:0], IVECT[7:0]}<br>• The Interrupt Service Routine must end with RETI.L. |

Notes:
1. MBASE = A programmable offset used in Z80 mode.
2. PC(15:0) or PC(23:0) = The short or long contents of the program counter.
3. SPS = Stack Pointer Short (16-bit)
4. ADL = Address/Data Long.
5. I[7:0] = The contents of the interrupt vector register (I).
6. IVECT[7:0] = The contents of the eZ80's internal ector bus.
7. SPL = Stack Pointer Long (24-bit).
8. MADL = Mixed ADL.

# *Assembly Language Initialization*

## Vector Table Setup

```
******************************************************************************************************

                                  Interrupt Vector Table

   .align256     ;This ZMASM Assembler Directive allows you to align your table on an even 256 byte
                 boundary
int_vect_tbl:    ;Label name representing 16 bit start of Interrupt Vector Table
   dw macc_vct   ;16 bit vector for Multiply Accumulate Engine (Vector of int_vect_tbl + 00)
   dw dma0_vct   ;16 bit vector for Direct Memory Access Controller0 (Vector of int_vect_tbl + 02)
   dw dma1_vct   ;16 bit vector for Direct Memory Access Controller1 (Vector of int_vect_tbl + 04)
   dw prt0_vct   ;16 bit vector for Programmable Reload Timer0 (Vector of int_vect_tbl + 06)
   dw prt1_vct   ;16 bit vector for Programmable Reload Timer1 (Vector of int_vect_tbl + 08
   dw prt2_vct   ;16 bit vector for Programmable Reload Timer2 (Vector of int_vect_tbl + 0A)
   dw prt3_vct   ;16 bit vector for Programmable Reload Timer3 (Vector of int_vect_tbl + 0C)
   dw prt4_vct   ;16 bit vector for Programmable Reload Timer4 (Vector of int_vect_tbl + 0E)
   dw prt5_vct   ;16 bit vector for Programmable Reload Timer5 (Vector of int_vect_tbl + 10)
   dw uzi0_vct   ;16 bit vector for Universal ZiLOG Interface0 (Vector of int_vect_tbl + 12)
   dw uzi1_vct   ;16 bit vector for Universal ZiLOG Interface1 (Vector of int_vect_tbl + 14)
   dw pta0_vct   ;16 bit vector for PortA bit0 (Vector of int_vect_tbl + 16)
   dw pta1_vct   ;16 bit vector for PortA bit1 (Vector of int_vect_tbl + 18)
   dw pta2_vct   ;16 bit vector for PortA bit2 (Vector of int_vect_tbl + 1A)
   dw pta3_vct   ;16 bit vector for PortA bit3 (Vector of int_vect_tbl + 1C)
   dw pta4_vct   ;16 bit vector for PortA bit4 (Vector of int_vect_tbl + 1E)
   dw pta5_vct   ;16 bit vector for PortA bit5 (Vector of int_vect_tbl + 20)
   dw pta6_vct   ;16 bit vector for PortA bit6 (Vector of int_vect_tbl + 22)
   dw pta7_vct   ;16 bit vector for PortA bit7 (Vector of int_vect_tbl + 24)
   dw ptb0_vct   ;16 bit vector for PortB bit0 (Vector of int_vect_tbl + 26)
   dw ptb1_vct   ;16 bit vector for PortB bit1 (Vector of int_vect_tbl + 28)
   dw ptb2_vct   ;16 bit vector for PortB bit2 (Vector of int_vect_tbl + 2A)
   dw ptb3_vct   ;16 bit vector for PortB bit3 (Vector of int_vect_tbl + 2C)
   dw ptb4_vct   ;16 bit vector for PortB bit4 (Vector of int_vect_tbl + 2E)
   dw ptb5_vct   ;16 bit vector for PortB bit5 (Vector of int_vect_tbl + 30)
```

Interrupt Vector Table

```
dw ptb6_vct  ;16 bit vector for PortB bit6 (Vector of int_vect_tbl + 32)
dw ptb7_vct  ;16 bit vector for PortB bit7 (Vector of int_vect_tbl + 34)
dw ptc0_vct  ;16 bit vector for PortC bit0 (Vector of int_vect_tbl + 36)
dw ptc1_vct  ;16 bit vector for PortC bit1 (Vector of int_vect_tbl + 38)
dw ptc2_vct  ;16 bit vector for PortC bit2 (Vector of int_vect_tbl + 3A)
dw ptc3_vct  ;16 bit vector for PortC bit3 (Vector of int_vect_tbl + 3C)
dw ptc4_vct  ;16 bit vector for PortC bit4 (Vector of int_vect_tbl + 3E)
dw ptc5_vct  ;16 bit vector for PortC bit5 (Vector of int_vect_tbl + 40)
dw ptc6_vct  ;16 bit vector for PortC bit6 (Vector of int_vect_tbl + 42)
dw ptc7_vct  ;16 bit vector for PortC bit7 (Vector of int_vect_tbl + 44)
dw ptd0_vct  ;16 bit vector for PortD bit0 (Vector of int_vect_tbl + 46)
dw ptd1_vct  ;16 bit vector for PortD bit1 (Vector of int_vect_tbl + 48)
dw ptd2_vct  ;16 bit vector for PortD bit2 (Vector of int_vect_tbl + 4A)
dw ptd3_vct  ;16 bit vector for PortD bit3 (Vector of int_vect_tbl + 4C)
dw ptd4_vct  ;16 bit vector for PortD bit4 (Vector of int_vect_tbl + 4E)
dw ptd5_vct  ;16 bit vector for PortD bit5 (Vector of int_vect_tbl + 50)
dw ptd6_vct  ;16 bit vector for PortD bit6 (Vector of int_vect_tbl + 52)
dw ptd7_vct  ;16 bit vector for PortD bit7 (Vector of int_vect_tbl + 54)
dw null_isr  ;16 bit null vectors (RESERVED) for (int_vect_tbl + 56 through 126)
```

**********************************************************************************************************

Vector Locations 0xxx56 through 0xxxFE are reserved and must be coded in the table to point to a null interrupt routine as follows:

```
dw null_isr     ;16 bit vector pointing to an ISR labeled “null_isr:”
```

And somewhere in the code in 16-bit space:

```
null_isr:
                ei          ;re-enable interrupts
                reti        ;return from interrupt
```

You can use an interrupt jump table in Assembly language as well as in C. The previous example does not use this technique. Later, when we look at a routine to set this up this example in C, we use a routine that uses a jump table to allow us to locate the actual ISR anywhere in the 24-bit memory map. In the previous example, the ISR must be located in 16-bit memory space. Take a brief look below at how to setup an interrupt jump table in assembly. The included Assembly example code uses the jump table technique.

```
                         Interrupt Jump Table
int_vect_tbl:            ;Label name representing 16 bit start of Interrupt Vector Table
     dw jump_tbl  + 0    ;16 bit vector for Multiply Accumulate Engine (Vector of int_vect_tbl + 00)
     dw jump_tbl  + 5    ;16 bit vector for Direct Memory Access Controller0 (Vector of int_vect_tbl +
                         02)
     dw jump_tbl  + A    ;16 bit vector for Direct Memory Access Controller1 (Vector of int_vect_tbl +
                         04)
     dw jump_tbl  + F    ;16 bit vector for Programmable Reload Timer0 (Vector of int_vect_tbl + 06)
     dw jump_tbl  + 14   ;16 bit vector for Programmable Reload Timer1 (Vector of int_vect_tbl + 08)
     dw jump_tbl  + 19   ;16 bit vector for Programmable Reload Timer2 (Vector of int_vect_tbl + 0A)
     dw jump_tbl  + 1E   ;16 bit vector for Programmable Reload Timer3 (Vector of int_vect_tbl + 0C)
     dw jump_tbl  + 23   ;16 bit vector for Programmable Reload Timer4 (Vector of int_vect_tbl + 0E)
     dw jump_tbl  + 28   ;16 bit vector for Programmable Reload Timer5 (Vector of int_vect_tbl + 10)
```

Assemble the table as in the example above. These parameters could all point to a single null vector routine if you are not using the particular interrupts.

At the address of jump_tbl in the 16-bit space, code another table with the Op Code of jump.lil (5BC3) and the 24-bit vector corresponding to the actual ISR (5 bytes total).

```
jump_tbl:
    jp.lil Lable_Name_of_ISR    ;This assembles the appropriate two byte Op Code
                                plus 3-byte vector to the appropriate 24-bit
                                memory location. Jump_tbl + 0
    jp.lil Lable_Name_of_ISR    ;jump_tbl  + 5
    jp.lil Lable_Name_of_ISR    ;jump_tbl  + A
    jp.lil Lable_Name_of_ISR    ;jump_tbl  + F
```

This table may contain only one entry (null_isr for example). It may contain multiple entries depending on how many ISRs you use.

### Initializing the Interrupt Vector Register (I)

The interrupt vector register (I) is used to point to the high byte (A15-A8) of the 16 bit address of the interrupt vector table. For Assembly language, the I register can only be loaded from the accumulator as follows:

```
ld a, HIGH int_vect_tbl    ;Load High Byte of interrupt vector table address into the
                           Accumulator
ld i,a                     ;Load the Interrupt Vector Register with the High Byte of the
                           Interrupt Vector Table
```

The demonstration software that runs from ZDS/ZDI uses the Interrupt Jump Table technique.To make sure that your ISRs can reside anywhere within the 24-bit memory map perform the following steps.

In the main assembly file (see Figure 1)

1. Cut the statement " segment code_data", line 226 of the code.
2. Paste it below one, two, or all three of the ISRs.

This action allows the ISRs to be assembled in 16-bit space as opposed to higher than address F0000h or in 20-bit space. No matter where the ISR is located, the interrupt jump table locates it correctly.

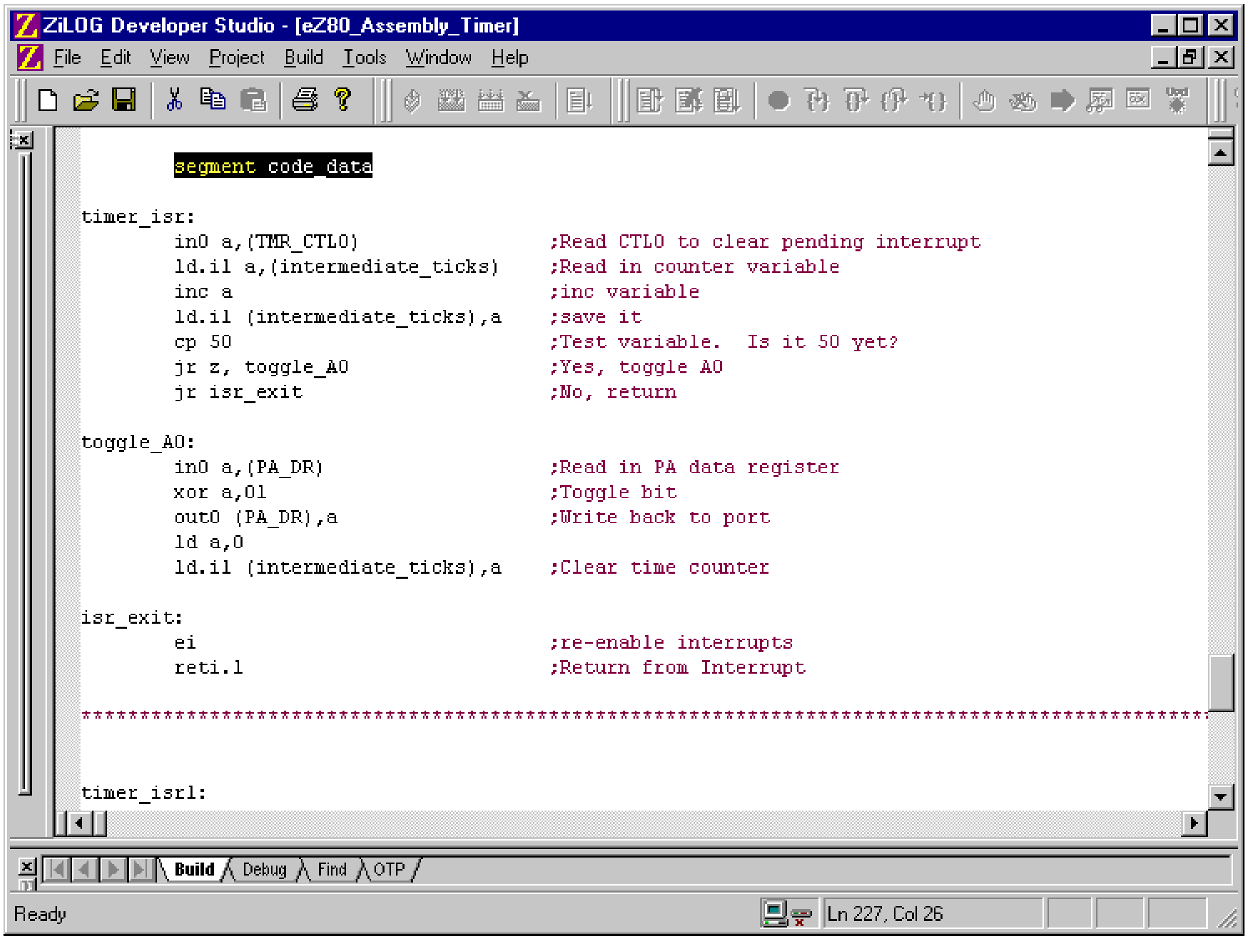


Figure 1. Main Assembly File

## Initialization in C

### Initializing the Interrupt Vector Register (I) in C

In this section we define the various interrupt tables to reside in a certain predetermined space (see "Definitions" on page 11). The _asm() C function allows the programmer to directly insert assembly code from the C source file into the assembly file. The \t inserts a tab character into the assembly file being generated. The \t can also represent a space or physical tab.

**Note:** The code samples in this section are color-coded. Green denotes offset comment and blue is for C proprietary language. Black denotes user code.

```
_asm(\tld a, 0xxx) ;Load High Byte of interrupt vector table address into the Accumulator
_asm(\tld i,a)     ;Load the Interrupt Vector Register with the High Byte of the Interrupt Vector Table
```

And again, somewhere in the code

:

```
#pragma interrupt
void isr_null(void)
                {
                }
```

This code becomes the null interrupt vector and gives the program a place to vector to and return from if a spurious interrupt occurs. The #pragma interrupt ensures the interrupt routine in the assembly file is terminated with ei followed by reti.

The following is the sethandler function prototype:

```
void sethandler(void (*)(void), unsigned char);
```

The sethandler function takes two arguments:

- The address of the actual interrupt service routine
- The unsigned character, vector

The sethandler function returns nothing.

The following function is the sethandler:

```
void sethandler(void (*handler)(void), unsigned char vector)
{
        void** ptr;
        ptr=(void*)(INTERRUPT_JUMP_TABLE+vector/2*5);
        /* vector 0 / 2 * 5 = 0 + interrupt jump table E100 = E100
           vector 2 / 2 * 5 = 5 + interrupt jump table E100 = E105
           vector 4 / 2 * 5 = A + interrupt jump table E100 = E10A

           point vector to the jump table by physically writing the
           vector into the jump table into the vector table*/
        *((unsigned short*)(INTERRUPT_TABLE+vector))=ptr;
        /* Therefore, this is what memory would look like starting at
           the interrupt table E000:
           00 E1 05 E1 0A E1 0F E1 14 E1..........*/
        /* Write the jp.lil Op Code in big endian format to the jump table */
        *((unsigned short*)ptr)=0xc35b;
        /* Increment the pointer by two*/
        ptr=(void**)(INTERRUPT_JUMP_TABLE+vector/2*5+2);
        /* Write the address of the isr handler into the jump table */
        *ptr=handler;
}
```

### Definitions

INTERRUPT_TABLE, INTERRUPT_JUMP_TABLE, and vector are all predefined in the code somewhere:

Example:

```
#define INTERRUPT_TABLE                                      0x00e000
#define INTERRUPT_JUMP_TABLE                                 0x00e100
#define VECTOR_TIMER0                                        0x06
#define VECTOR_TIMER1                                        0x08
```

It is here that we define the INTERRUPT_TABLE to be E000h (eZ80190 internal RAM), the INTERRUPT_JUMP_TABLE to be E100h (eZ80190 internal RAM), and the VECTOR_TIMER0 to be 06h. Also program the “I” register with 0xE0.

The eZ80190’s internal RAM locates both an interrupt vector table (0xE000) and an interrupt jump table (0xE100). The Interrupt Vector table must be aligned on an even 256-byte boundary. Upon receiving an interrupt, the eZ80 reads the appropriate 16-bit vector from the interrupt_table. The eZ80 vectors to the interrupt_jump_table and reads and executes a 24-bit jump to any memory location in the 24-bit map.

## *Application Example*

### eZ80190 C Timer Interrupt Routine

This program runs from ZDS/ZDI with the eZ80 Evaluation board.  It uses the eZ80190 running at 40MHz, and initializes two Timers, TMR0 and TMR1, to interrupt every 10mS. Timer0 uses a period counter, and a time high and time low register to implement a Modulated PWM routine on PA0 and PA2.  PA2 is implemented as the inverse of PA0. The period time of the modulated waveform is 100mS and modulates through 9 iterations from 1/10 on and 9/10 off to 9/10 on and 1/10 off. One complete iteration of the modulated routine looks as follows, with PA2 being exactly the opposite. See Figure 2.

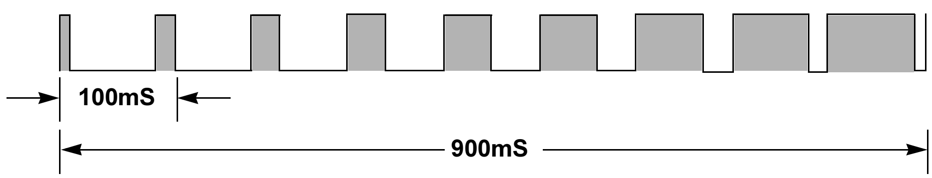


Figure 2. PA0 Timing Waveform for C Timer Interrupt Timing Example

Timer1 uses a 10X Counter to toggle portA1 every 100mS. The initial settings for the eZ80 and the memory map are outlined below.  Two versions of initial settings with memory map variations have been used to demonstrate that no matter where the ISR's are located (be it 16 bit space or 24 bit space) that the jump table can be used to properly set the vector locations.

#### C Project Tools

- C Compiler  >>  eZ80CC1.01
- ZDS  >>  3.65Beta,     Tab Setting >>  C Files = 4, .s files = 8
- EZ8019000100ZCO  >>  eZ80 Evaluation Board with Realtek Ethernet Controller

#### List of Files in C Project

- eZ80boot.s
- main.c
- time_pwm.c
- interrupts.c

C project <Dependencies>

- ez80def.h

- ez80.h
- interrupts.h

C Memory Maps

Figure 3 illustrates the initial and secondary memory map settings.

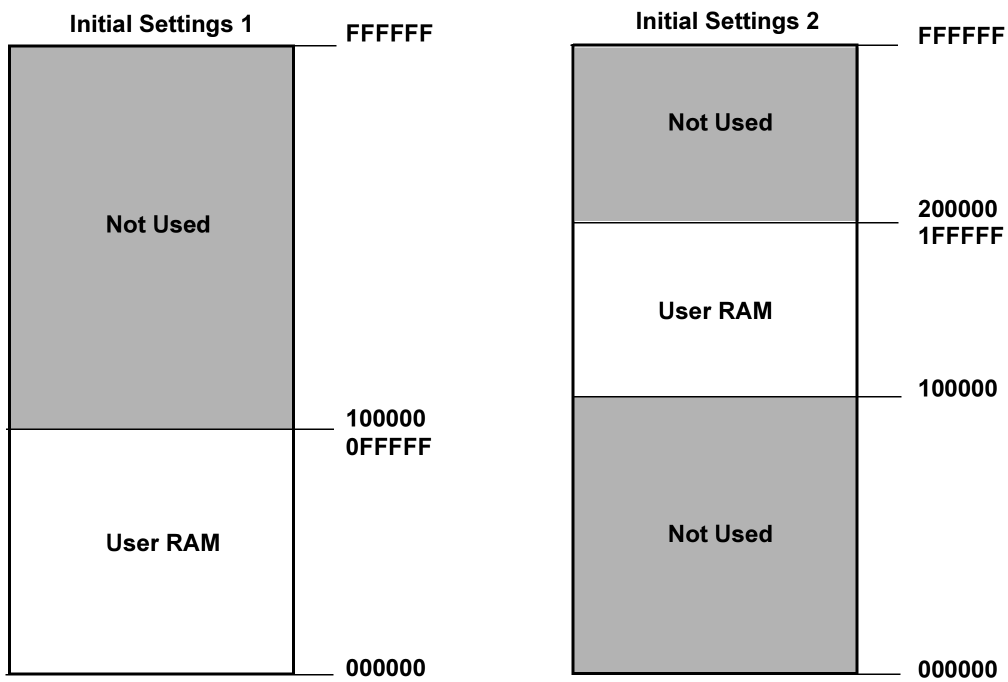


Figure 3. Memory Maps

**Note:** When using the eZ80 Evaluation Kit with the Realtek Ethernet Controller, it is no longer possible to split the two 512K RAM's because the RAM's are now enabled with CS1 only. When using the Realtek board, please set CS1, the PC and the Stack pointers appropriately. See Table 3 for a list of settings that work with the Realtek board for both the C and Assembly projects. The following two sets of settings can also be used.

C Project 1 Initial Settings

- SPL `0FFFFFh`
- SPS `FFFFh`
- PC `0000h`
- CS0 Not used, all zeros
- CS1 Lower Bound = 0, Upper Bound = 0F, Control Register = 28

- CS2 Not used, all zeros
- CS3 Not used, all zeros

C Project 1 LINKER Settings

- EXTIO `0000h` to `FFFFh`
- INTIO `0000h` to `00FFh`
- ROM `000000h` to `0FFFFFh`

C Project 2 Initial Settings

- SPL `1FFFFFh`
- SPS `FFFFh`
- PC `100000h`
- CS0 Not used, all zeros
- CS1 Lower Bound = 10, Upper Bound = 1F, Control Register = 28
- CS2 Not used, all zeros
- CS3 Not used, all zeros

C Project 2 LINKER Settings

- EXTIO `0000h` to `FFFFh`
- INTIO `0000h` to `00FFh`
- ROM `100000h` to `1FFFFFh`

Table 3. Realtek Evaluation Board Initial and Linker Settings

| C and Assembly Initial Settings with Realtek Ethernet Controller | | Linker Settings | |
|---|---|---|---|
| SPS | FFFFh | EXTIO | 0000 to FFFFh |
| SPL | 0FFFFFh | INTIO | 0000 to 00FFh |
| PC | 0000h | ROM | 000000 to 0FFFFFh |
| CS0 | Not used, all zeros | | |
| CS1 | Lower Bound = 00h, Upper Bound - 0Fh, Control = 28h | | |
| CS2 | Not used, all zeros | | |
| CS3 | Not used, all zeros | | |

## eZ80 Assembly Timer Interrupt Routine

This routine runs from ZDS/ZDI using the eZ80 evaluation board and initializes Timers 0 and 1 to interrupt every 10mS from an eZ80 running at 40MHz. Timer0's ISR outputs a pulse on PortA0 every 10mS*50 or 0.5s.

Total Wave form period = 1s. Timer1's ISR toggles portA1 every 10mS*100 or 1second. Total Wave Form Period = 2 Seconds. See Figure 5.

### Assembly Project Tools

- ZDS >> 3.65Beta , Tab Setting >> 8
- EZ8019000100ZCO >> eZ80 Evaluation Board with Realtek Ethernet Controller

### List of Files in Project

- eZ80_assembly_timer.asm
- ez80.inc

### Assembly Memory Map

Figure 4 illustrates the initial and secondary memory map settings.

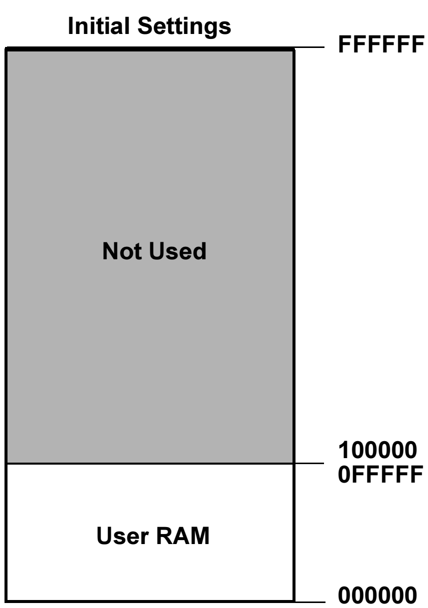


Figure 4. Assembly Memory Map

Assembly Project Initial Settings

- SPL `0FFFFFh`
- SPS `FFFFh`
- PC `0000h`
- CS0 Not used, all zeros
- CS1 Lower Bound = 0, Upper Bound = 0F, Control Register = 28
- CS2 Not used, all zeros
- CS3 Not used, all zeros

LINKER Settings

- EXTIO `0000h` to `FFFFh`
- INTIO `0000h` to `00FFh`
- ROM `000000h` to `0FFFFFh`

Assembly Project Output

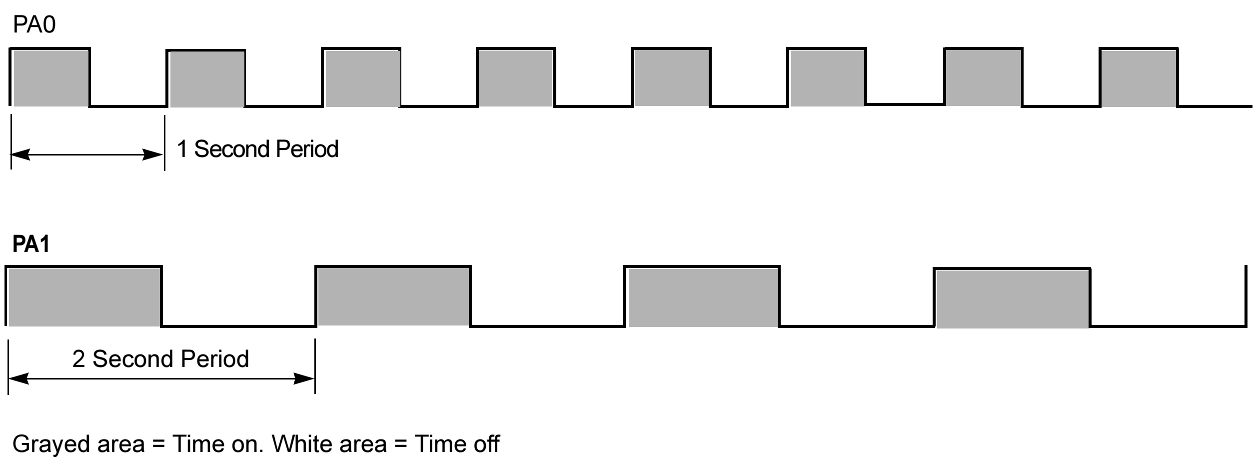


Figure 5. Timer Interrupt Routine

## EZ80190 Webserver Evaluation Board Jumper Settings

Table 4. Evaluation Board Jumper Settings for All Projects—
Realtek Ethernet Controller Board

| Jumper # | ON | OFF | Don’t Care |
|---|---|---|---|
| J1 | 1-2 | | |
| J2 | X | | |
| J3 | X | | |
| J4 | | X | |
| J5 | 2-3 | | |
| J7 | X | | |
| J10 | 2-3 | | |
| J11 | X | | |

Table 5. Evaluation Board Jumper Settings for all Projects
Crystal LAN Ethernet Controller Board

| Jumper # | ON | OFF | Don't Care |
|---|---|---|---|
| J1 | X | | |
| J2 | X | | |
| J3 | 1-2 | | |
| J4 | X | | |
| J5 | X | | |
| J7 | 1-2 | | |
| J8 | | X | |
| J9 | | X | |

**Note:** For a Crystal LAN board that has been programmed with a Flash Loader and a application running from Flash, you may have to remove J8 to get the ZDS project started. Keep in mind that to run the Flash Loader or application programmed into Flash, that the J8 jumper must be replaced.

## *Summary*

This application note together with the software resources provided, ZDS, ZPAK, and the eZ80 evaluation board, provide a clean example of how programmers can set up and initialize interrupts in both C and Assembly environments. These are only simple examples and can be embellished upon depending on the application needs. This application note is meant as a basis to understand how the interrupt structure works and how to initialize, setup, and locate interrupts tables and ISR's. If you are having difficulty locating something in memory, consult the .map file. The .map file is one of the surest ways to discover if you are locating code or data where you think you are within the available memory space.

## References

- Source Code Listing
- Appendix B – eZ80190 Evaluation Board Schematics

## Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form besides the intended application, must be approved through a license agreement between both parties. ZiLOG is not responsible for any code(s) used beyond the intended application. Contact your local ZiLOG Sales Office to obtain necessary license agreements.

## Document Disclaimer

# Source Code

## eZ80 Timer Interrupts—Assembly Project Files

### eZ80. Inc.

The following code is contained in the file eZ80.inc.

```
*****************************************************************************
*                        PORTA                                         *
*****************************************************************************
PA_DR        equ    %96
PA_DDR       equ    %97
PA_ALT1      equ    %98
PA_ALT2      equ    %99

*****************************************************************************
*                        PORTB                                         *
****************************************************************************
PB_DR        equ    %9A
PB_DDR       equ    %9B
PB_ALT1      equ    %9C
PB_ALT2      equ    %9D
*****************************************************************************
*
*                        PORTC                                         *
****************************************************************************
PC_DR        equ    %9E
PC_DDR       equ    %9F
PC_ALT1      equ    %A0
PC_ALT2      equ    %A1
*****************************************************************************
*                        PORTD                                         *
*****************************************************************************
PD_DR        equ    %A2
PD_DDR       equ    %A3
PD_ALT1      equ    %A4
PD_ALT2      equ    %A5
****************************************************************************
*                        UART0                                         *
****************************************************************************

UART_RBR0    equ    %C0
UART_THR0    equ    %C0
BRG_DLRL0    equ    %C0
BRG_DLRH0    equ    %C1
UART_IER0    equ    %C1
UART_IIR0    equ    %C2
```

```
UART_FCTL0  equu   %C2
UART_LCTL0  equ   %C3
UART_MCTL0  equ   %C4
UART_LSR0   equ   %C5
UART_MSR0   equ   %C6
UART_SPR0   equ   %C7
*****************************************************************************
*                       UART1                                          *
****************************************************************************
BRG_DLRL1   equ   %D0
BRG_DLRH1   equ   %D1
UART_RBR1   equ   %D0
UART_THR1   equ   %D0
UART_IER1   equ   %D1
UART_IIR1   equ   %D2
UART_FCTL1  equ   %D2
UART_LCTL1  equ   %D3
UART_MCTL1  equ   %D4
UART_LSR1   equ   %D5
UART_MSR1   equ   %D6
UART_SPR1   equ   %D7
*****************************************************************************
*                       UZI CONTROL                              *
*****************************************************************************
UZI_CTL0    equ   %CF
UZI_CTL1    equ   %DF
*****************************************************************************
*                       I2C0                                     *
*****************************************************************************
I2C_SAR0    equ   %C8
I2C_XSAR0   equ   %C9
I2C_DR0     equ   %CA
I2C_CTL0    equ   %CB
I2C_SR0     equ   %CC
I2C_CCR0    equ   %CC
I2C_SRR0    equ   %CD

*****************************************************************************
*                       I2C1                                     *
*****************************************************************************
I2C_SAR1    equ   %D8
I2C_XSAR1   equ   %D9
I2C_DR1     equ   %DA
I2C_CTL1    equ   %DB
I2C_SR1     equ   %DC
I2C_CCR1    equ   %DC
I2C_SRR1    equ   %DD
*****************************************************************************
```

```
*                        SPI0                                 *
*******************************************************************************
SPI_CTL0     equ    %B6
SPI_SR0      equu   %B7
SPI_RBR0     equ    %B8
SPI_TSR0     equ    %B8
*******************************************************************************
*                        SPI1                                 *
*******************************************************************************
SPI_CTL1     equ    %BA
SPI_SR1      equ    %BB
SPI_RBR1     equ    %BC
SPI_TSR1     equ    %BC
*******************************************************************************
                         TIMER0                                     *
*******************************************************************************
TMR_CTL0     equ    %80
TMR_DRL0     equ    %81
TMR_DRH0     equ    %82
TMR_RRL0     equ    %81
TMR_RRH0     equ    %82
*******************************************************************************
*                        TIMER1                                     *
*******************************************************************************
TMR_CTL1     equ    %83
TMR_DRL1     equ    %84
TMR_DRH1     equ    %85
TMR_RRL1     equ    %84
TMR_RRH1     equ    %85


*******************************************************************************
*                        TIMER2                                     *
*******************************************************************************
TMR_CTL2     equ    %86
TMR_DRL2     equ    %87
TMR_DRH2     equ    %88
TMR_RRL2     equ    %87
TMR_RRH2     equ    %88
*******************************************************************************
*                        TIMER3                                     *
*******************************************************************************
TMR_CTL3     equ    %89
TMR_DRL3     equ    %8A
TMR_DRH3     equ    %8B
TMR_RRL3     equ    %8A
TMR_RRH3     equ    %8B
```

```
****************************************************************************
*                        TIMER4                                           *
****************************************************************************
TMR_CTL4     equ   %8C
TMR_DRL4     equ   %8D
TMR_DRH4     equ   %8E
TMR_RRL4     equ   %8D
TMR_RRH4     equ   %8E
****************************************************************************
*                        TIMER5                                           *
****************************************************************************
TMR_CTL5     equ   %8F
TMR_DRL5     equ   %90
TMR_DRH5     equ   %91
TMR_RRL5     equ   %90
TMR_RRH5     equ   %91
****************************************************************************
*                        WDT                                        *
****************************************************************************
WDT_CTL      equ   %93
WDT_RR       equ   %94
****************************************************************************
*                  Chip Select & WSG                                *
***************************************************************************
CS_LBR0      equ   %A8
CS_UBR0      equ   %A9
CS_CTL0      equ   %AA
CS_LBR1      equ   %AB
CS_UBR1      equ   %AC
CS_CTL1      equ   %AD
CS_LBR2      equ   %AE
CS_UBR2      equ   %AF
CS_CTL2      equ   %B0
CS_LBR3      equ   %B1
CS_UBR3      equ   %B2
CS_CTL3      equ   %B3
****************************************************************************
*                  RAM CONTROL                                      *
****************************************************************************
RAM_CTL0     equ   %B4
RAM_CTL1     equ   %B5
****************************************************************************
*                        DMA                                        *
****************************************************************************
DMA_SARL0    equ   %EE
DMA_SARM0    equ   %EF
DMA_SARH0    equ   %F0
DMA_DARL0    equ   %F1
```

```
DMA_DARM0   equ   %F2
DMA_DARH0   equ   %F3
DMA_BCL0    equ   %F4
DMA_BCH0    equ   %F5
DMA_CTL0    equ   %F6

DMA_SARL1   equ   %F7
DMA_SARM1   equ   %F8
DMA_SARH1   equ   %F9
DMA_DARL1   equ   %FA
DMA_DARM1   equ   %FB
DMA_DARH1   equ   %FC
DMA_BCL1    equ   %FD
DMA_BCH1    equ   %FE
DMA_CTL1    equ   %FF
******************************************************************************
*                       MACC                                    *
******************************************************************************
MAC_XSTART  equ   %E0
MAC_XEND    equ   %E1
MAC_XRELOAD equ   %E2
MAC_LENGTH  equ   %E3
MAC_YSTART  equ   %E4
MAC_YEND    equ   %E5
MAC_YRELOAD equ   %E6
MAC_CTL     equ   %E7
MAC_AC0     equ   %E8
MAC_AC1     equ   %E9
MAC_AC2     equ   %EA
MAC_AC3     equ   %EB
MAC_AC4     equ   %EC
MAC_SR      equ   %ED
******************************************************************************
******************************************************************************
```

### eZ80_Assembly_Timer.asm

The following code is contained in the file `eZ80_Assembly_Timer.asm`.

```
******************************************************************************
******************************************************************************
*  eZ80 Assembly Timer0 Interrupt Routine Written by Mark Thissen 8/6/01.
*
* This routine runs from ZDS/ZDI using the eZ80 evaluation board and intializes
Timers 0 and 1 to interrupt
* every 10mS from an eZ80 running at 40MHz.  It then outputs a pulse on PortA0
every 10mS*50 or 0.5s.
* Total Wave form period = 1s. Timer1 toggles portA1 every 10mS*100 or 1second.
Total Wave Form Period =
```

```
* 2 Seconds.
*
*     ZDS   >>  3.65Beta
*     Tab Setting >> 8
*
*     List of Files in Project
*
*     eZ80_assembly_timer.asm
*     ez80.inc
*
*******************************************************************************
*              Memory Map
****************************************************************************
*     Initial Settings
*
*     _______________________ FFFFFF
*     |               |
*     |               |
*     |               |
*     |///////////////|
*     |///NOT USED ///|
*     |///////////////|
*     |               |
*     |               |
*     |_______________|__ 100000
*     |               | 0FFFFF
*     |      USER RAM |
*     |               |
*     |_______________|__ 000000
*
*
****************************************************************************
*
*
* Project Initial Settings:
*     SPL   0FFFFFh
*     SPS   FFFFh
*     PC    0000h
*     CS0   Not used, all zeros
*     CS1   Lower Bound = 0, Upper Bound = 0F, Control Register = 28
*     CS2   Not used, all zeros
*     CS3   Not used, all zeros
* LINKER Settings
*     EXTIO 0000 to FFFFh
*     INTIO   0000 to 00FFh
*     ROM   000000 to 0FFFFFh
******************************************************************************
      include "ez80.inc"
```

```
      org 0 ;eZ80 Reset Vector

      .assume adl=1
      define code_data, space = ROM, org = %f0000     ;Control Section starting
                                                      ;at F0000h
      define code_data0, space = ROM, org = %100      ;Control Section starting
                                                      ;at 100h
      define nmi_loop, space = ROM, org = %66         ;Control Section starting
                                                      ;at 66h for NMI
      globals on

      jp.lil start                                    ;Jump to start (0x000100)

      segment code_data0
start:
      di                                              ;Ensure Interrupts Disabled
      ld.lil sp,%FFFFF                                ;Init Stack Pointer
      stmix                                           ;Use eZ80 mix mode

;Note,  All internal registers pertinent to program control must be initialized in the
;code when running from ROM.  ZDS Initial Settings will no longer apply.

      ld a, HIGH int_vect_tbl            ;Get high byte of interrupt vector
                                         ;table address
      ld i,a                             ;Program the Interrupt Vector Register
                                         ;(I)
      ld a,0
      ld.lil (intermediate_ticks),a      ;Initialize interrupt counter
      ld.lil (intermediate_ticks1),a     ;Initialize interrupt counter1

********************************************************************************
*                        Port A Initialization
********************************************************************************
      ld a,%00
      out0 (PA_DDR),a                    ;Make PA0 output
      out0 (PA_ALT2),a                   ;No special Functions
      out0 (PA_ALT1),a
********************************************************************************
*                         Timer init and Enable
********************************************************************************
      ld a, 0
      out0 (TMR_CTL0),a
      out0 (TMR_CTL1),a
      ld a, 61h
      out0 (TMR_RRH0),a       ;Timer High byte reload constant
      out0 (TMR_RRH1),a       ;Timer High byte reload constant
      ld a,0A8h
```

```
        out0 (TMR_RRL0),a            ;Timer Low byte reload constant
        out0 (TMR_RRL1),a            ;Timer Low byte reload constant
        ld a,5Eh
        out0 (TMR_CTL0),a            ;Multi-pass mode, Clock/16, Interrupt Enable
        out0 (TMR_CTL1),a            ;Multi-pass mode, Clock/16, Interrupt Enable
        in0 a,(TMR_CTL0)
        or a,01
        out0 (TMR_CTL0),a            ;Start Timer0
        in0 a,(TMR_CTL1)
        or a,01
        out0 (TMR_CTL1),a            ;Start Timer1

        ei                           ;Enable Interrupts

loop:
        jr loop                      ;Sit here and loop and wait for interrupt

        segment nmi_loop  ;NMI control Section (66h)

nmi_isr:
        nop                          ;No Operation
        retn                         ;Return from Non-maskable Interrupt

        segment code_data0

************************************************************************************
*                        Vector Tables and ISR's
************************************************************************************

        .align 256          ;This ZMASM Assembler Directive allows you to
                            ;align your table on an even 256 byte boundary

int_vect_tbl:               ;Label name of Interrupt Vector Table

        dw jump_tbl + 0     ;16 bit vector for Multiply Accumulate Engine
                            ;(Vector of int_vect_tbl + 00)
        dw jump_tbl + 0     ;16 bit vector for Direct Memory Access Controller0
                            ;(Vector of int_vect_tbl + 02)
        dw jump_tbl + 0     ;16 bit vector for Direct Memory Access Controller1
                            ;(Vector of int_vect_tbl + 04)
        dw jump_tbl + 5     ;16 bit vector for Programmable Reload Timer0
                            ;(Vector of int_vect_tbl + 06)
        dw jump_tbl + 10    ;16 bit vector for Programmable Reload Timer1
                            ;(Vector of int_vect_tbl + 08)
        dw jump_tbl + 0     ;16 bit vector for Programmable Reload Timer2
                            ;Vector of int_vect_tbl + 0A)
        dw jump_tbl + 0     ;16 bit vector for Programmable Reload Timer3
                            ;(Vector of int_vect_tbl + 0C)
```

```
dw jump_tbl + 0    ;16 bit vector for Programmable Reload
                   ;Timer4 (Vector of int_vect_tbl + 0E)
dw jump_tbl + 0    ;16 bit vector for Programmable Reload
                   ;Timer5 (Vector of int_vect_tbl + 10)
dw jump_tbl + 0    ;16 bit vector for Universal ZiLOG
                   ;Interface0 (Vector of int_vect_tbl + 12)
dw jump_tbl + 0    ;16 bit vector for Universal ZiLOG
                   ;Interface1(Vector of int_vect_tbl + 14)
dw jump_tbl + 0    ;16 bit vector for PortA bit0 (Vector of
                   ;int_vect_tbl + 16)
dw jump_tbl + 0    ;16 bit vector for PortA bit1 (Vector of
                   ;int_vect_tbl + 18)
dw jump_tbl + 0    ;16 bit vector for PortA bit2 (Vector of
                   ;int_vect_tbl + 1A)
dw jump_tbl + 0    ;16 bit vector for PortA bit3 (Vector of
                   ;int_vect_tbl + 1C)
dw jump_tbl + 0    ;16 bit vector for PortA bit4 (Vector of
                   ;int_vect_tbl + 1E)
dw jump_tbl + 0    ;16 bit vector for PortA bit5 (Vector of
                   ;int_vect_tbl + 20)
dw jump_tbl + 0    ;16 bit vector for PortA bit6 (Vector of
                   ;int_vect_tbl + 22)
dw jump_tbl + 0    ;16 bit vector for PortA bit7 (Vector of
                   ;int_vect_tbl + 24)
dw jump_tbl + 0    ;16 bit vector for PortB bit0 (Vector of
                   ;int_vect_tbl + 26)
dw jump_tbl + 0    ;16 bit vector for PortB bit1 (Vector of
                   ;int_vect_tbl + 28)
dw jump_tbl + 0    ;16 bit vector for PortB bit2 (Vector of
                   ;int_vect_tbl + 2A)
dw jump_tbl + 0    ;16 bit vector for PortB bit3 (Vector of
                   ;int_vect_tbl + 2C)
dw jump_tbl + 0    ;16 bit vector for PortB bit4 (Vector of
                   ;int_vect_tbl + 2E)
dw jump_tbl + 0    ;16 bit vector for PortB bit5 (Vector of
                   ;int_vect_tbl + 30)
dw jump_tbl + 0    ;16 bit vector for PortB bit6 (Vector of
                   ;int_vect_tbl + 32)
dw jump_tbl + 0    ;16 bit vector for PortB bit7 (Vector of
                   ;int_vect_tbl + 34)
dw jump_tbl + 0    ;16 bit vector for PortC bit0 (Vector of
                   ;int_vect_tbl + 36)
dw jump_tbl + 0    ;16 bit vector for PortC bit1 (Vector of
                   ;int_vect_tbl + 38)
dw jump_tbl + 0    ;16 bit vector for PortC bit2 (Vector of
                   ;int_vect_tbl + 3A)
dw jump_tbl + 0    ;16 bit vector for PortC bit3 (Vector of
                   ;int_vect_tbl + 3C)
```

```
dw jump_tbl + 0   ;16 bit vector for PortC bit4 (Vector of
                  ;int_vect_tbl + 3E)
dw jump_tbl + 0   ;16 bit vector for PortC bit5 (Vector of
                  ;int_vect_tbl + 40)
dw jump_tbl + 0   ;16 bit vector for PortC bit6 (Vector of
                  ;int_vect_tbl + 42)
dw jump_tbl + 0   ;16 bit vector for PortC bit7 (Vector of
                  ;int_vect_tbl + 44)
dw jump_tbl + 0   ;16 bit vector for PortD bit0 (Vector of
                  ;int_vect_tbl + 46)
dw jump_tbl + 0   ;16 bit vector for PortD bit1 (Vector of
                  ;int_vect_tbl + 48)
dw jump_tbl + 0   ;16 bit vector for PortD bit2 (Vector of
                  ;int_vect_tbl + 4A)
dw jump_tbl + 0   ;16 bit vector for PortD bit3 (Vector of
                  ;int_vect_tbl + 4C)
dw jump_tbl + 0   ;16 bit vector for PortD bit4 (Vector of
                  ;int_vect_tbl + 4E)
dw jump_tbl + 0   ;16 bit vector for PortD bit5 (Vector of
                  ;int_vect_tbl + 50)
dw jump_tbl + 0   ;16 bit vector for PortD bit6 (Vector of
                  ;int_vect_tbl + 52)
dw jump_tbl + 0   ;16 bit vector for PortD bit7 (Vector of
                  ;int_vect_tbl + 54)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 56)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 58)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 5A)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 5C)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 5E)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 60)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 62)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 64)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 66)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 68)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 6A)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 6C)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 6E)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 70)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 72)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 74)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 76)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 78)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 7A)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 7C)
dw jump_tbl + 0   ;16 bit null vectors (RESERVED) for (int_vect_tbl + 7E)

*******************************************************************************
```

```
jump_tbl:

      jp.lil null_isr         ;Jump to Null ISR
      jp.lil timer_isr  ;Jump to Timer0 ISR
      jp.lil timer_isr1 ;Jump to Timer1 ISR

******************************************************************************
******************************
*                  These ISR's reside in 20 bit space as defined by segment
code_data
******************************************************************************
segment code_data

timer_isr:
      in0 a,(TMR_CTL0)              ;Read CTL0 to clear pending interrupt
      ld.il a,(intermediate_ticks)  ;Read in counter variable
      inc a ;inc variable
      ld.il (intermediate_ticks),a  ;save it
      cp 50                         ;Test variable.  Is it 50 yet?
      jr z, toggle_A0               ;Yes, toggle A0
      jr isr_exit                   ;No, return

toggle_A0:
      in0 a,(PA_DR)                 ;Read in PA data register
      xor a,01                      ;Toggle bit
      out0 (PA_DR),a                ;Write back to port
      ld a,0
      ld.il (intermediate_ticks),a  ;Clear time counter

isr_exit:
      ei;re-enable interrupts
      reti.l;Return from Interrupt

******************************************************************************
timer_isr1:
      in0 a,(TMR_CTL1)              ;Read CTL1 to clear pending interrupt
      ld.il a,(intermediate_ticks1) ;Read in counter variable
      inc a                         ;inc variable
      ld.il (intermediate_ticks1),a ;save it
      cp 100                        ;Test variable.  Is it 100 yet?
      jr z, toggle_A1               ;Yes, toggle A1
      jr isr1_exit                  ;No, return

toggle_A1:
      in0 a,(PA_DR)                 ;Read in PA data register
      xor a,02                      ;Toggle bit
      out0 (PA_DR),a                ;Write back to port
      ld a,0
```

```
      ld.il (intermediate_ticks1),a ;Clear time counter
isr1_exit:
      ei                            ;re-enable interrupts
      reti.l                        ;Return from Interrupt


*******************************************************************************
null_isr:

      ei                            ;re-enable interrupts
      reti.l                        ;return from interrupt

******************************************************************************
intermediate_ticks: db [1]0         ;Timer count variable
intermediate_ticks1: db [1]0        ;Timer1 count variable
*******************************************************************************
      end                           ;End of Assembly
```

## eZ80 Timer Interrupts—C Project Files

### eZ80_boot.s

The following code is contained in the file `eZ80_boot.s`.

```
;*************************************************************************
;*    ez80Boot: C Runtime Startup
;*    Copyright (c) ZiLOG, 1999
;*************************************************************************
      define      .nbss,      space=ROM
;*************************************************************************
      .sect ".nbss"           ; In case no-one else names it
;*************************************************************************
.INITSIM    .equ    0         ; Using simulator?

      .if .INITSIM
      .assume                 ADL=0
      define                  .simstart,SPACE=ROM, org=0
      segment                 .simstart
      jp.lil                  _c_int0
      .endif

      define                  .startup,   space=ROM

      .sect                   ".startup"  ; This should be placed properly
      .def                    _c_int0
      .def                    __exit
      .ref                    _main
      .ref                    .BSS_BASE,.BSS_LENGTH
      .ref                    .TOSPS
      .ref                    .TOSPL

.INITBSS    .equ    1         ;Zero the .bss section ?
.INITCOPY   .equ              1           ;Copy the initialized tabels?

      .assume                 ADL=1

;*********************************
; Program entry point
;*********************************

_c_int0:
      ld.sis                  sp,.TOSPS   ; Setup SPS
      ld.lil                  sp,.TOSPL   ; Setup SPL
      call.il _c_int1                     ; Call the init with ADL=1
__exit:
      jr    $                             ; ?
```

```
        .assume ADL=1

_c_int1:
        .if    .INITBSS

;------ Initialize the .BSS section to zero

        ld                  hl,.BSS_LENGTH    ; Check for non-zero length
        ld                  bc,0              ; *
        or                  a,a               ; *
        sbc                 hl,bc             ; *
        jr                  z,_c_bss_done     ; .BSS is zero-length ...
$$:
        ld                  hl,.BSS_BASE      ; [hl]=.bss
        ld                  bc,.BSS_LENGTH
        ld                  (hl),0
        dec                 bc                ; 1st byte's taken care of
        ld                  hl,0
        sbc                 hl,bc
        jr                  z,_c_bss_done     ; Just 1 byte ...
        ld                  hl,.BSS_BASE      ; reset hl
        ld                  de,.BSS_BASE+1    ; [de]=.bss+1
        ldir
_c_bss_done:

        .endif                                ; .INITBSS

        .if                       .INITCOPY   ; Copy Initialized tabels
        .ref     .DATA_BASE       ; Address of initialized data section
        .ref     .DATA_COPY       ; Address of initialized data section copy
        .ref     .DATA_LENGTH     ; Length of initialized data sectrion

;------ Copy the initialized data section
        ld                        hl,.DATA_LENGTH   ; Check for non-zero length
        ld                        bc,0        ; *
        or                        a,a         ; *
        sbc                       hl,bc       ; *
        jr                        z,_c_data_done    ; .DATA is zero-length ...
$$:
        ld                        hl,.DATA_COPY     ; [hl]=.data_copy
        ld                        de,.DATA_BASE     ; [de]=.data
        ld                        bc,.DATA_LENGTH   ; [bc]= data length
loaddata:                                           ; Load 64k at a time.
        add hl, bc                                  ;
        push hl          ;
        or a, a
        sbc hl,bc
```

```
      ldir                          ; Copy the data section
      push hl                       ; load next address to bc
      pop  bc                       ;
      pop hl                        ; load end address to hl
      or a,a                        ; reset cary flag
      sbc hl,bc                     ; check if done transfer
      jr z, _c_data_done      ;
      push bc
      push hl
      pop bc
      pop hl
      jr loaddata

_c_data_done:
      .endif

;------ main()

      ld                  hl,0        ; hl=NULL
      push                hl          ; argv[0] = NULL
      ld                  ix,0
      add                 ix,sp       ; ix=&argv[0]
      push                ix          ; &argv[0]

      pop                 hl
      ld                  de,0        ; argc==0
      call  _             main        ; main()
      pop                 af          ; clean the stack
      ret.l                           ; return with ADL=0
```

### eZ80def.h

The following code is contained in the file `ez80def.h`.

```
/*****************************************************************
 * ez80def.h
 *
 *****************************************************************/

#ifndef     _EZ80DEF_H
#define     _EZ80DEF_H

#include <ez80.h>

#define     di()                        _asm("\tdi");
#define     ei()                        _asm("\tei");

#define     VECTOR_TIMER0               0x06
#define VECTOR_TIMER1                   0x08
```

```
#define     VECTOR_UART0                  0x12
#define     VECTOR_UART1                  0x14

#define     RAM_CTL0          (*((__INTIO volatile unsigned char *)0xb4))
#define     RAM_CTL1          (*((__INTIO volatile unsigned char *)0xb5))

#define TMR_CTL0  (*((__INTIO volatile unsigned char *)0x80))
#define TMR_DRL0  (*((__INTIO volatile unsigned char *)0x81))
#define TMR_DRH0  (*((__INTIO volatile unsigned char *)0x82))
#define TMR_RRL0  (*((__INTIO volatile unsigned char *)0x81))
#define TMR_RRH0  (*((__INTIO volatile unsigned char *)0x82))

#define TMR_CTL1  (*((__INTIO volatile unsigned char *)0x83))
#define TMR_DRL1  (*((__INTIO volatile unsigned char *)0x84))
#define TMR_DRH1  (*((__INTIO volatile unsigned char *)0x85))
#define TMR_RRL1  (*((__INTIO volatile unsigned char *)0x84))
#define TMR_RRH1  (*((__INTIO volatile unsigned char *)0x85))

#define     PA_DR       (*((__INTIO volatile unsigned char *)0x96))
#define     PA_DDR      (*((__INTIO volatile unsigned char *)0x97))
#define     PA_ALT1     (*((__INTIO volatile unsigned char *)0x98))
#define     PA_ALT2     (*((__INTIO volatile unsigned char *)0x99))

#define     PB_DR       (*((__INTIO volatile unsigned char *)0x9a))
#define     PB_DDR      (*((__INTIO volatile unsigned char *)0x9b))
#define     PB_ALT1     (*((__INTIO volatile unsigned char *)0x9c))
#define     PB_ALT2     (*((__INTIO volatile unsigned char *)0x9d))

#define     PC_DR       (*((__INTIO volatile unsigned char *)0x9e))
#define     PC_DDR      (*((__INTIO volatile unsigned char *)0x9f))
#define     PC_ALT1     (*((__INTIO volatile unsigned char *)0xa0))
#define     PC_ALT2     (*((__INTIO volatile unsigned char *)0xa1))

#define     PD_DR       (*((__INTIO volatile unsigned char *)0xa2))
#define     PD_DDR      (*((__INTIO volatile unsigned char *)0xa3))
#define     PD_ALT1     (*((__INTIO volatile unsigned char *)0xa4))
#define     PD_ALT2     (*((__INTIO volatile unsigned char *)0xa5))

#define     UART_RBR0   (*((__INTIO volatile unsigned char *)0xc0))
#define     UART_THR0   (*((__INTIO volatile unsigned char *)0xc0))
#define     BRG_DLRL0   (*((__INTIO volatile unsigned char *)0xc0))
#define     BRG_DLRH0   (*((__INTIO volatile unsigned char *)0xc1))
#define     UART_IER0   (*((__INTIO volatile unsigned char *)0xc1))
#define     UART_IIR0   (*((__INTIO volatile unsigned char *)0xc2))
#define     UART_FCTL0  (*((__INTIO volatile unsigned char *)0xc2))
#define     UART_LCTL0  (*((__INTIO volatile unsigned char *)0xc3))
#define     UART_MCTL0  (*((__INTIO volatile unsigned char *)0xc4))
#define     UART_LSR0   (*((__INTIO volatile unsigned char *)0xc5))
```

```
#define     UART_MSR0    (*((__INTIO volatile unsigned char *)0xc6))
#define     UART_SPR0    (*((__INTIO volatile unsigned char *)0xc7))
#define     I2C_SAR0     (*((__INTIO volatile unsigned char *)0xc8))
#define     I2C_XSAR0    (*((__INTIO volatile unsigned char *)0xc9))
#define     I2C_DR0      (*((__INTIO volatile unsigned char *)0xca))
#define     I2C_CTL0     (*((__INTIO volatile unsigned char *)0xcb))
#define     I2C_SR0      (*((__INTIO volatile unsigned char *)0xcc))
#define     I2C_SRR0     (*((__INTIO volatile unsigned char *)0xcd))
#define     UZI_CTL0     (*((__INTIO volatile unsigned char *)0xcf))

#define     UART_RBR1    (*((__INTIO volatile unsigned char *)0xd0))
#define     UART_THR1    (*((__INTIO volatile unsigned char *)0xd0))
#define     BRG_DLRL1    (*((__INTIO volatile unsigned char *)0xd0))
#define     BRG_DLRH1    (*((__INTIO volatile unsigned char *)0xd1))
#define     UART_IER1    (*((__INTIO volatile unsigned char *)0xd1))
#define     UART_IIR1    (*((__INTIO volatile unsigned char *)0xd2))
#define     UART_FCTL1   (*((__INTIO volatile unsigned char *)0xd2))
#define     UART_LCTL1   (*((__INTIO volatile unsigned char *)0xd3))
#define     UART_MCTL1   (*((__INTIO volatile unsigned char *)0xd4))
#define     UART_LSR1    (*((__INTIO volatile unsigned char *)0xd5))
#define     UART_MSR1    (*((__INTIO volatile unsigned char *)0xd6))
#define     UART_SPR1    (*((__INTIO volatile unsigned char *)0xd7))
#define     I2C_SAR1     (*((__INTIO volatile unsigned char *)0xd8))
#define     I2C_XSAR1    (*((__INTIO volatile unsigned char *)0xd9))
#define     I2C_DR1      (*((__INTIO volatile unsigned char *)0xda))
#define     I2C_CTL1     (*((__INTIO volatile unsigned char *)0xdb))
#define     I2C_SR1      (*((__INTIO volatile unsigned char *)0xdc))
#define     I2C_SRR1     (*((__INTIO volatile unsigned char *)0xdd))
#define     UZI_CTL1     (*((__INTIO volatile unsigned char *)0xdf))

#endif
******************************************************************************
```

### interrupts.c

The following code is contained in the file `interrupts.c`.\

```
/*****************************************************************
 * interrupts.c
 *
 * These are interrupt routines for the eZ80
 *
 * Since interrupt table and interrupt routines must be within
 * first 64k boundary, the following allows interrupt
 * routines to be located anywhere within the 16MB address space.
 *
 * To do this, two tables are setup in the on-board sram which
 * is at 00E000 to 00FFFF
 *
```

```
 * The first table is the interrupt vector table. It lies
 * at 00E000 to 00E0FF (or wherever INTERRUPT_TABLE is defined
 * to point to in interrupts.h) and contains vectors into the next
 * interrupt "jump" table.
 *
 * The second table lies from 00E100 to 00E37f (or wherever
 * INTERRUPT_JUMP_TABLE points to in interrupts.h). It is a table
 * of jp.lil instructions to the 24 bit address of the actual interrupt
 * routine. Each entry is 5 bytes. The first two bytes are the opcode
 * 0x5b, 0xc3 which is the jp.lil pneumonic. The next three bytes are
 * the 24 bit address of the interrupt routine.
 *
 * The sethandler function will automatically place an isr routine
 * in the interrupt "jump" table.
 *
 ****************************************************************/

#include    "interrupts.h"
extern void isr_null(void);
extern void _asm(char *);

#pragma interrupt
void isr_null(void)
      {
      }

/****************************************************************
 * This function sets up the interrupts tables on the eZ80. It will
 * initialize each vector in the interrupt vector table to point to
 * its corresponding entry in the interrupt "jump" table. It
 * calls the sethandler function and initializes all the 128 eZ80
 * interrupt vectors to point to isr_null.
 *
 * Lastly, it initializes the eZ80's "i" register with the high
 * byte of the interrupt vector table or in this case, "E0"
 *
 ****************************************************************/
void init_interrupts(void)
{
      int i;

      //initialize all interrupt vectors to null isr
      for(i=0; i < 256; i+=2)
      {
            sethandler(&isr_null,i);
      }

      _asm("\tld a,%e0\n\tld i,a");
```

```
}

/**************************************************************
 * This function places an interrupt handler in the interrupt
 * "jump" table.
 *
 * You only need to pass it the actual interrupt vector number
 * along with the ISR handler address. It Will compute the offset
 * into the interrupt jump table and set it accordingly.
 *
 **************************************************************/

 void sethandler(void (*handler)(void), unsigned char vector)
 {
      void** ptr;

      ptr=(void*)(INTERRUPT_JUMP_TABLE+vector/2*5);
      /* vector 0 / 2 * 5 = 0 + interrupt jump table E100 = E100
         vector 2 / 2 * 5 = 5 + interrupt jump table E100 = E105
         vector 4 / 2 * 5 = A + interrupt jump table E100 = E10A

         point vector to the jump table by physically writing the
         vector into the jump table into the vector table*/

      *((unsigned short*)(INTERRUPT_TABLE+vector))=ptr;

        /* Therefore, this is what memory would look like starting at
           the interrupt table E000:
           00 E1 05 E1 0A E1 0F E1 14 E1...........*/

      /* Write the jp.lil opcode in big endian format to the jump table */
      *((unsigned short*)ptr)=0xc35b;

      /* Increment the pointer by two*/
      ptr=(void**)(INTERRUPT_JUMP_TABLE+vector/2*5+2);

      /* Write the address of the isr handler into the jump table */
      *ptr=handler;

}

/**************************************************************/
```

**interrupts.h**

The following code is contained in the files `interrupts.h`.

```
/**************************************************************
```

```
 * interrupts.h
 *
 ****************************************************************/

#ifndef                 _INTERRUPTS_H
#define                 _INTERRUPTS_H

#define                 INTERRUPT_TABLE               0x00e000
#define                 INTERRUPT_JUMP_TABLE    0x00e100

#define                 ei()  _asm("\tei");
#define                 di()  _asm("\tdi");

void sethandler(void (*)(void), unsigned char);
void init_interrupts(void);

#endif                  /* _INTERRUPTS_H */
```

### Main.c

The following code is contained in the file `Main.c`.

```
/*************************************************************************
*           eZ80190 C Timer Interrupt Routine by Mark Thissen 8/7/01
*
*  This program is set up to run from ZDS/ZDI with the eZ80 Evaluation board.
* It utilizes the eZ80190 running at 40MHz., and initializes two Timers,
* TMR0 and TMR1, to interrupt every 10mS.  Timer0 utilizes a period counter,
* and a time high and time low register to implement a Modulated PWM routine
* on PA0 and PA2.  PA2 is simply implemented as the inverse of PA0.  The period
* time of the modulated waveform is 100mS and modulates through 9 iterations
* from 1/10 on and 9/10 off to 9/10 on and 1/10 off.
* The initial settings for the eZ80 and the memory map are outlined below.
* Two versions of initial settings with memory map variations have been
* utilized to demonstrate that no matter where the ISR's are located
* (be it 16 bit space or 24 bit space) that the jump table can be used
* to properly set the vector locations.
*
*     C Compiler  >>  eZ80CC1.01
*     ZDS               >>  3.65Beta
*     Tab Setting >>  C Files = 4, .s files = 8
*
*     List of Files in Project
*
*     eZ80boot.s
*     main.c
*     time_pwm.c
*     interrupts.c
*     <Dependencies>
```

```
*     ez80def.h
*     ez80.h
*     interrupts.h
*
**************************************************************************
*
*                             Memory Maps
*
**************************************************************************
C Memory Maps

Initial Settings1                          Initial Settings2
   _______________________ FFFFFF          __________________________ FFFFFF
  |                      |                    |                     |
  |                      |                    |                     |
  |                      |                    | //// NOT USED /////|
  |                      | 20000              |                     |
  |         |            | 1FFFFF             |                     |
  | //// NOT USED /////  |                    |_____________________|__200000
  |                      |                    |                     | 1FFFFF
  |                      |                    |                     |
  |                      |                    |      USER RAM       |
  |                      |                    |                     |
  |                      |                    |_____________________|__100000
  |______________________|__ 100000           |                     | 0FFFFF
  |                      |   0FFFFF    |      |                     |
  |                      |                    |                     |
  |     USER RAM         |                    |  //// NOT USED ////|
  |                      |                    |                     |
  |                      |                    |                     |
  |                      |                    |                     |
  |______________________|__ 000000    |____________________________|__ 000000


**************************************************************************
*
*
* Project Initial Settings1:
*     SPL         0FFFFFh
*     SPS         FFFFh
*     PC          0000h
*     CS0         Not used, all zeros
*     CS1         Lower Bound = 0, Upper Bound = 0F, Control Register = 28
*     CS2         Not used, all zeros
*     CS3         Not used, all zeros
* LINKER Settings
*     EXTIO       0000 to FFFFh
```

```
*     INTIO        0000 to 00FFh
*     ROM          000000 to 0FFFFFh
*
****************************************************************************
*
* Project Initial Settings2:
*     SPL          1FFFFFh
*     SPS          FFFFh
*     PC           100000h
*     CS0          Not used, all zeros
*     CS1          Lower Bound = 10, Upper Bound = 1F, Control Register = 28
*     CS2          Not used, all zeros
*     CS3          Not used, all zeros
* LINKER Settings
*     EXTIO        0000 to FFFFh
*     INTIO        0000 to 00FFh
*     ROM          100000 to 1FFFFFh
***************************************************************************/


#include    "ez80def.h"
#include    "interrupts.h"
void init_timer(void);
void init_timer1(void);

int main(void)
{
      unsigned int i,k,y,z;
      RAM_CTL0=0xc0;           //enable on-chip sram
      RAM_CTL1=0x00;
      PA_DDR &= 0x78;          //make PA0 - PA2 outputs
      PA_ALT2 &= 0x78;
      PA_ALT1 &= 0x78;

      init_interrupts();
      ei();
      init_timer();
      init_timer1();

      for(i=0;i<256;i++)
            {
            for(k=0;k<256;k++)            // This is one long loop, almost
45min.
                  {
                  for(y=0;y<256;y++)
                        {
                        for(z=0;z<100;z++);
                        }
```

```
                  }
            }
}
```

### time_PWM.c

The following code is contained in the file `time_PWM.c`.

```
/*****************************************************************
 * time_pwm.c
 *
 * This will setup timer0 to output a modulated PWM waveform
 * on PA0 and it's inverse on PA2.  The Routine goes through 9
 * iterations of the period from 1/10 (10mS) on, to 9/10 (90mS)
 * off through 9/10 (90mS) on to 1/10 (10mS) off.
 * PA1 toggles under interrupt service from Timer1 every 100mS.
 *
 *
 *****************************************************************/

#include "ez80def.h"
void timer0_on_off (void);
extern void isr_timer1(void);
extern void isr_timer0(void);
extern void sethandler(void (*)(void), unsigned char);
unsigned char intermediate_ticks;
unsigned char period, t_lo, t_hi;


/*****************************************************************
 * This will initialize timer0 to interrupt every 10ms
 *
 * 16 bit time constant is not big enough for 100ms interrupts,
 * so we will use additional intermediate counter to count
 * every 10 ticks.
 *****************************************************************/

void init_timer(void)
{
      sethandler(&isr_timer0,VECTOR_TIMER0);  /*Pass the address of
                                                                   isr_timer0
and the vector number
                                                                    to be
placed into the interrupt jump table */
      PA_DR = 0;
      period = 9;
      t_hi = 0;
      t_lo = 0;
```

```
      TMR_CTL0 = 0x00;
      TMR_RRH0 = 0x61;  //setup timer to interrupt every 10ms
      TMR_RRL0 = 0xa8;
      TMR_CTL0 = 0x5e;  //timer0 = multipass, /16, interrupt enable
      TMR_CTL0 |= 0x01; //enable timer

}
/
*****************************************************************************/

void init_timer1(void)
{
      intermediate_ticks=0x00;
      sethandler(&isr_timer1,VECTOR_TIMER1);  /*Pass the address of
                                                                isr_timer1
and the vector number
                                                                 to be
placed into the interrupt jump table */
      TMR_CTL1 = 0x00;
      TMR_RRH1 = 0x61;  //setup timer to interrupt every 10ms
      TMR_RRL1 = 0xa8;
      TMR_CTL1 = 0x5e;  //timer0 = multipass, /16, interrupt enable
      TMR_CTL1 |= 0x01; //enable timer

}

/****************************************************************
 *                 These Timer ISR's get called every 10ms.
 ****************************************************************/

#pragma interrupt
void isr_timer0(void)
{
      unsigned char temp;

      if (t_lo == 0)    //Switch case every 100mS
      {
      switch (period)
            {
            case 1: t_hi = 9;
                        t_lo = 1;
                        timer0_on_off();
                        break;
            case 2: t_hi = 8;
                        t_lo = 2;
                        timer0_on_off();
                        break;
            case 3: t_hi = 7;
```

```
                        t_lo = 3;
                        timer0_on_off();
                        break;
            case 4: t_hi = 6;
                        t_lo = 4;
                        timer0_on_off();
                        break;
            case 5: t_hi = 5;
                        t_lo = 5;
                        timer0_on_off();
                        break;
            case 6: t_hi = 4;
                        t_lo = 6;
                        timer0_on_off();
                        break;
            case 7: t_hi = 3;
                        t_lo = 7;
                        timer0_on_off();
                        break;
            case 8: t_hi = 2;
                        t_lo = 8;
                        timer0_on_off();
                        break;
            case 9: t_hi = 1;
                        t_lo = 9;
                        timer0_on_off();
                        break;
             }
          }
            else
            {
            timer0_on_off();
         }
      temp=TMR_CTL0;          //read to clear pending int
}

/*****************************************************************/

#pragma interrupt
void isr_timer1(void)
{
      unsigned char temp;
      temp=TMR_CTL1;          //read to clear pending int

      intermediate_ticks++;
      if(intermediate_ticks >= 10)
      {
            intermediate_ticks=0;
```

```
            PA_DR ^= 0x02;    //toggle PA1 every 100mS
      }
}

/******************************************************************/
void timer0_on_off (void)
{
      if (t_hi != 0)
      {
      PA_DR |= 0x01;    //Make PA0 High
      PA_DR &= 0xFB;    //Make PA2 Low
      t_hi --;
      }
      else
      {
      PA_DR &= 0xFE;    //Make PA0 Low
      PA_DR |= 0x04;    //Make PA2 High
      t_lo --;
      }

      if    (t_lo == 0) //End of Period?
       {
       period --;       //Next Case
      if(period <= 0)   //End of Case Modulation (900mS)?
            {
            period = 9;
          }
       }
}
```

## *Customer Feedback Form*

If you experience any problems while operating this product, or if you note any inaccuracies while reading this Application Note, please copy and complete this form, then mail or fax it to ZiLOG (see *Return Information*, below). We also welcome your suggestions!

| |
|---|
| eZ80190 Webserver Date Code |
| Serial # or Board Fab #/Rev. # |
| Software Version |
| Document Number |
| Host Computer Description/Type |

Customer Information

| | |
|---|---|
| Name | Country |
| Company | Phone |
| Address | Fax |
| City/State/Zip | E-Mail |

Return Information

ZiLOG
System Test/Customer Support
910 E. Hamilton Avenue, Suite 110, MS 4–3
Campbell, CA 95008
Fax: (408) 558-8536
ZILOG World Wide Customer Support Center

Problem Description or Suggestion

Provide a complete description of the problem or your suggestion. If you are reporting a specific problem, include all steps leading up to the occurrence of the problem. Attach additional pages as nec