Application Note

Using the PCF8584 I²C Controller with the eZ80Acclaim!™ MCU

AN015701-0703

ZiLOG Worldwide Headquarters • 532 Race Street • San Jose, CA 95126
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
532 Race Street
San Jose, CA 95126
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 iv
List of Tables v
Abstract 1
eZ80Acclaim!™ Flash Microcontrollers Overview 1
eZ80Acclaim!™ Features 2
Discussion 3
Theory of Operation 3
Hardware Connections 3
eZ80F91 CSXn Register Settings 5
Software Design 6
Conclusion 7
Appendix A—Source Code 8
bbboot.s 9
initInts.s 19
isr.s 19
main.c 22
sio.c 25
funcs.c 27
i2c.c 29
init.c 45
types.h 48
ez80RegDefs.h 48
pcf8584.h 57
ez80_i2c.h 58
tb_vars.h 60
bb_cfg.h 63
Notes 66

List of Figures

Figure 1. Simple I2C Interface Block Diagram 1
Figure 2. I2C Schematic Diagram 5

List of Tables

Table 1. SN74LVC138AD-to-eZ80F91 Connections 3
Table 2. Hardware Connections 4

Abstract

This Application Note demonstrates how to connect Philips Semiconductor's PCF8584 I2C controller to ZiLOG's eZ80F91 Flash MCU. In this discussion, commands are executed through a simple menu-driven interface via a terminal window connected to the UART block of the eZ80F91 device. The source code files associated with this Application Note (see AN0157-SC01.zip) demonstrate how to initialize the I2C and transfer data between the PCF8584 and the eZ80F91, which is used as both master and slave. These basic functions can be used to generate an eZ80F91 I2C application device driver. The code sets up the eZ80F91 I2C for interrupt-driven control of the transfer, and uses polling to control the PCF8584 controller for I2C transfers. For details about the PCF8584 controller, please refer to the PCF8584 I2C Bus Controller Product Specification (pcf8584.pdf) on the Philips Semiconductor website.

The software applications described in this Application Note are written using ZiLOG's ZDSII IDE Development Tool.

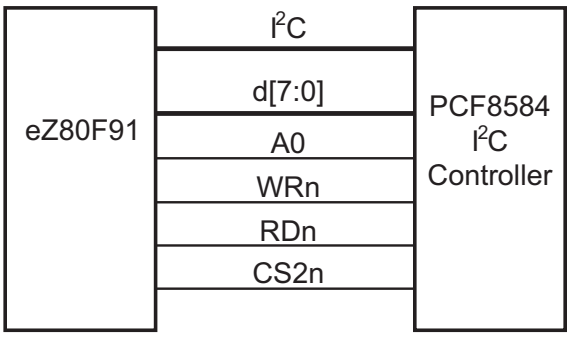

Figure 1. Simple I2C Interface Block Diagram

eZ80Acclaim!™ Flash Microcontrollers Overview

eZ80Acclaim!™ on-chip Flash Microcontrollers are an exceptional value for customers designing high-performance, 8-bit MCU-based systems. With speeds up to 50MHz and an on-chip Ethernet MAC (eZ80F91 only), designers have the performance necessary to execute complex applications quickly and efficiently. Combining Flash and SRAM, eZ80Acclaim!™ devices provide the memory required to implement communication protocol stacks and achieve flexibility when performing in-system updates of application firmware.

The eZ80Acclaim!™ Flash MCU can operate in full 24-bit linear mode addressing 16MB without a Memory Management Unit. Additionally, support for the Z80-compatible mode allows Z80/Z180 customers to execute legacy code within multiple

64KB memory blocks with minimum modification. With an external bus supporting eZ80®, Z80, Intel and Motorola bus modes and a rich set of serial communications peripherals, designers have several options when interfacing to external devices.

Some of the many applications suitable for eZ80Acclaim!™ devices include vending machines, point-of-sale terminals, security systems, automation, communications, industrial control and facility monitoring, and remote control.

eZ80Acclaim!™ Features

- Single-cycle instruction fetch, high-performance, pipelined eZ80® CPU core
- 10/100 BaseT Ethernet Media Access Controller with Media-Independent Interface (MII)
- 256KB Flash memory and 16KB SRAM (8KB user, 8KB EMAC)
- Low power features including SLEEP mode, HALT mode, and selective peripheral power-down control
- Two UARTs with independent baud rate generators
- SPI with independent clock rate generator
- I^2C with independent clock rate generator
- IrDA-compliant infrared encoder/decoder
- Glueless external peripheral interface with 4 Chip Selects, individual Wait State generators, and an external WAIT input pin—supports Z80-, Intel-, and Motorola-style buses
- Fixed-priority vectored interrupts (both internal and external) and interrupt controller
- Real-time clock with on-chip 32KHz oscillator, selectable 50/60Hz input, and separate V_{DD} pin for battery backup
- Four 16-bit Counter/Timers with prescalers and direct input/output drive
- Watch-Dog Timer with internal oscillator clocking option
- 32 bits of General-Purpose I/O
- OCI™ and ZDI debug interfaces
- 1149.1-compliant JTAG
- 144-pin LQFP package
- 3.0–3.6V supply voltage with 5V tolerant inputs

Discussion

This section contains the bus mode description, design, and implementation of the PCF8584 I2C controller connected to the eZ80F91 MCU.

Theory of Operation

The PCF8584 I2C controller can be connected to the eZ80F91 microcontroller with no glue logic. The PCF8584 INT pin is connected to GPIO Port B5, but is never used by the software. The main goal of this Application Note is to provide an example of how to program the eZ80F91 I2C block. The PCF8584 device can detect whether or not it is connected to an Intel- or Motorola-type bus. Only the eZ80F91 device's Intel Bus mode can be used with the PCF8584, because in both the Intel and Motorola bus modes, CSn always falls first. Therefore, the PCF8584 assumes an Intel-like interface.

Hardware Connections

For this application example, CS2n is subdivided into 8 peripheral Chip Selects via a Texas Instruments SN74LVC138AD 3- to 8-line decoder (DigiKey PN 296-8459-5-ND). This decoder uses the three binary select inputs and three enable inputs to select one of the eight outputs. One of these outputs (Y1n Pin 14) is connected to the PCF8584 device. Table 1 shows how the SN74LVC138AD device is connected to the eZ80F91 MCU.

Table 1. SN74LVC138AD-to-eZ80F91 Connections

SN74LVC138AD Pin	eZ80F91 Pin	Description
A0 Pin 1	A13 Pin 18	Address bit 13 on the eZ80F91
A1 Pin 2	A14 Pin 19	Address bit 14 on the eZ80F91 MCU
A2 Pin 3	A15 Pin 20	Address bit 15 on the eZ80F91 MCU
G2A Pin 4	CS2n Pin 35	Chip select 2 on the eZ80F91 MCU
G2B Pin 5 and GND Pin 7	NC	VDD Ground
G1 Pin 6 and VCC Pin 16	NC	VCC 3.3V
Y1n Pin 14	NC	Connect to the PCF8584 Pin 17
Y0n, Y2n through Y7n	NC	Chip selects used for other peripherals

The CSWSG CS2n is set to Intel mode with an address range of `800000h` to `80FFFFh`. The PCF8584 Chip Select address range is `802000h` to `803FFFh`, but only `802000h` to `802001h` is actually used.

Table 2 shows the minimum connections required to interface the PCF8584 device to the eZ80F91 MCU. Each of the signals for the PCF8584 device are described following Table 2.

Table 2. Hardware Connections

PCF8584 Pin	eZ80F91 Pin	Description
CLK	NA	Connect a 4 MHz clock oscillator to this pin (PCLK).
Reset/Strb	GPIO	Use GPIO pin as hardware reset.
CSn	CS2n	Intel Mode Chip Select.
RDn	RDn	Read Control.
WRn	WRn	Write Control.
A0	A0	Address Bus.
IACK	NC	No Connect.
SCL	SCL	I2C SCL.
SDA	SDA	I2C SDA.
D[7:0]	D[7:0]	Data Bus.
INTn	GPIO PORTB5	Use GPIO pin to detect I2C Interrupts.

CLK. The PCF8584 Pin 1 signal requires a clock source to be running for proper operation and for the reset to work properly (see section 6.10 of the PCF8584 product specification). The clock source in this example is called out on port PCLK (see the schematic diagram in Figure 2). The PCLK signal port is a 4-pin clock oscillator (DigiKey PN XC259-ND).

Reset/Strb. The PCF8584 Pin 19 signal is connected to the eZ80F91 MCU's SCL Pin 110.

CSn. The PCF8584 Pin 17 signal is connected to the SN74LVC138AD 3- to 8-line decoder Pin 14 discussed above.

RDn/DTACK. The PCF8584 Pin 16 signal is a buffered version of the eZ80F91 MCU's Pin 51 RDn signal. This signal is labelled CTRL2 in the schematic and the buffer used is a SN74LVC244ADWR buffer (DigiKey PN 296-1229-1-ND). This signal is also used to drive the direction pin on the Data Bus buffer.

WRn/R_W. The PCF8584 Pin 18 signal is a buffered version of the eZ80F91 MCU's Pin 52 WRn signal. This signal is labelled CTRL3 in the schematic and the buffer used is a SN74LVC244ADWR buffer (DigiKey PN 296-1229-1-ND).

A0. The PCF8584 Pin 6 signal is a buffered version of the eZ80F91 MCU's Pin 1 A0 signal. This signal is labelled A0 in the schematic and the buffer used is a SN74LVC244ADWR buffer (DigiKey PN 296-1229-1-ND).

IACKn. The PCF8584 Pin 4 signal is not connected.

SCL. The PCF8584 Pin 3 signal is connected to the eZ80F91 MCU's SCL Pin 110. This pin represents the I2C clock input for serial data transfers.This signal requires a 4.7K pull-up to 5.0V.

SDA. The PCF8584 Pin 2 signal is connected to the eZ80F91 MCU's SCL Pin 109. This pin represents the bidirectional I2C serial data line.This signal requires a 4.7K pull-up to 5.0V.

DataBus DB[0:7]. The PCF8584 Pin 7, Pin 8, Pin 9, Pin 11, Pin 12, Pin 13, Pin 14, and Pin 15 signals are a buffered version of the eZ80F91 MCU's Data Bus pins 39-46. These signals are labelled D0–D7 in the schematic and the buffer used is a SN74LVC245ADWR buffer (DigiKey PN 296-1232-1-ND).

INTn. The PCF8584 Pin 5 signal is connected to the eZ80F91 MCU GPIO Port B5 Pin 105.This pin is used to detect I2C interrupts.

eZ80F91 CSXn Register Settings

To set up the memory CSXn registers for proper operation, the following registers are loaded:

```
CS2n_LBR = 0x80h  // Represents the lower bound on the CS2n
CS2n_UBR = 0x80h  // Represents the upper bound on the CS2n
CS2n_CTL = 0x08h  // This bit enables CS2n only in MEM mode
CS2n_BMC = 0x87h  // Sets intel mode with 7 clock cycles
                  // between each state.
```

For details about the PCF8584 connection to the eZ80F91, see the schematic diagram in Figure 2.

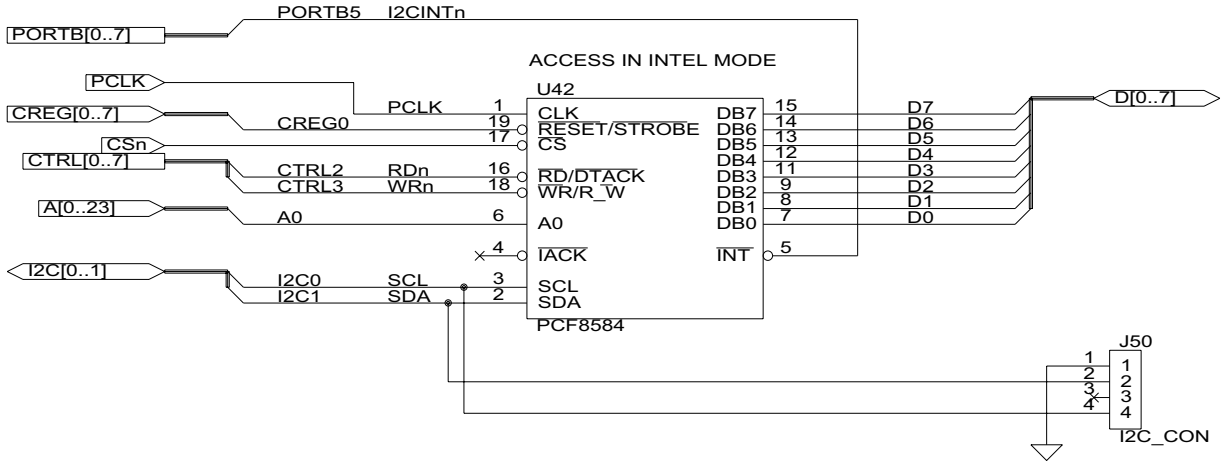

Figure 2. I2C Schematic Diagram

Software Design

Examples follow for each of the four types of transfers listed below.

- eZ80F91 as a Master that sends a packet to the PCF8584
- eZ80F91 as a Master that receives a packet from the PCF8584
- eZ80F91 as a Slave that sends data to the PCF8584
- eZ80F91 as a Slave that receives data from the PCF8584

The eZ80F91 I2C software is interrupt-driven, and the Interrupt Service Routine controls data transfer. The PCF8584 software uses polling to control the transfers. Because parameters are not passed to the ISR, global variables are used to specify the transfer. A define file, I2C_PKTSZ, sets the number of bytes to transfer.

Each packet is governed by a Start condition, which is followed by the Slave address/transfer direction bit, then the data bytes `01h` to I2C_PKTSZ. The software checks the data as it is received by the eZ80F91 and the PCF8584 to validate the transfers.

The global parameters are:

I2C_SLV_ADDR	The slave to be accessed.
I2C_NBYTES2XFER	The number of data bytes to transfer.
I2C_NBYTESXFERRED	The current byte count. Initialized to 0.
I2C_MODE	Sets the direction (LSB bit of first byte of packet).

The main eZ80F91 I2C functions are:

EZ80_MAS_TX	This function sets the eZ80F91 as the I2C master and transmits I2C_PKTSZ bytes to the PCF8584.
EZ80_MAS_RX	This function sets the eZ80F91 as the I2C master and receives I2C_PKTSZ bytes from the PCF8584.
PCF8584_MAS(MODE)	This function sets the PCF8584 as the master and transmits I2C_PKTSZ bytes to the eZ80F91 when mode is set to I2C_WR. When the mode is I2C_RD the PCF8584 receives I2C_PKTSZ bytes from the eZ80F91.
EZ80_I2C_ISR	This is the eZ80F91 I2C Interrupt Service Routine. It handles the data transfers of the eZ80F91 I2C based on the status from the I2C Status Register.

The support I2C routines are:

PERIPHERAL_RESET	Asserts a hardware reset to the PCF8584.
PCF8584_I2C_INIT	This functions initializes the PCF8584.
EZ80_I2C_INIT	This function initializes the eZ80F91 I2C.
EZ80_I2C_RESET	This function asserts a software reset to the eZ80F91 I2C.

For details about each of the above routines, refer to the Source Code listings in Appendix A on page 8.

Conclusion

This Application Note discusses the hardware and software interfaces of the eZ80F91 MCU signal to an I2C device. The eZ80F91 MCU software sets up a basic operation for both master and slave data transfers. The hardware interface is a simple addition to existing designs with minimum impact to the memory map. The software provided demonstrates the basis for a great start into the realm of the I2C world by showing that the eZ80F91 MCU can communicate either as a Master or Slave device on the I2C bus for any of your design applications.

Appendix A—Source Code

Source files for this Application Note can be found in the AN0157-SC01.zip file that is a companion to this document (please see www.zilog.com). These files are described below and listed in this section for easy reference.

bbboot.s	This module contains the startup code for the eZ80F91. It sets up the reset vectors, NMI interrupt, the interrupt vector table; initializes the Chip Selects and Wait State Generator registers; and calls the main program.
initInts.s	This routine initializes the I Register to point to the interrupt vector table and enables interrupts.
isr.s	This module contains the interrupt service routines for the NMI interrupt, the spurious interrupts, timer1 interrupt, and the I2C interrupt.
main.c	The main routine calls an init to setup the hardware and displays the I2C test menu selections through UART0.
sio.c	This module contains the code to set up UART0 as the STDIN and STDOUT I/O serial port.
funcs.c	This module contains support functions for the PCF8584 I2C controller and the eZ80F91 timer1. Timer1 is used for time delays and for software time-outs in polling loops to avoid any infinite loops in the software.
i2c.c	This module contains the support functions for the PCF8584 and eZ80® I2C and the 'C' level eZ80® interrupt support routine to handle the eZ80® interrupt driven I2C transfers. Once the eZ80® I2C is initialized the ISR handles the rest of the transfer.
init.c	This routine initializes the hardware and software.

The following *include* files are also listed in this section.

types.h	This file includes the typedefs for the different variable types.
ez80RegDefs.h	This file contains the defines for the eZ80® registers.
pcf8584.h	This file contains the defines for the PCF8584 I2C controller.
ez80_i2c.h	This file contains the defines for the eZ80® I2C.
tb_vars.h	This file defines the global variables and specifies the function prototypes.
bb_cfg.h	This file contains the hardware test platform dependent definitions.

bbboot.s

```
;*********************************************************************
;* Author: DP Consulting, Inc.
;* Source: bbboot.s
;* Description: Startup file for ZDS tools
;*
;* Rev   Who   Date     Description
;* 1.0   DPC   06/05/03 initial release
;*
;*    specify reset vectors
;*    build interrupt table
;*    initialize the stack pointers
;*    initialize the chip selects
;*    initialize sram - move constant data to sram (BSS section data)
;*    call main
;*
;*********************************************************************/

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

   .NOLIST
   .include “ez80f91.inc”
   .LIST

   .sect “bss”               ; In case no-one else names it
   .ref     .BSS_BASE        ; start of BSS
   .ref     .BSS_LENGTH      ; length of BSS
   .ref     .TOSPL           ; SP long

;*********************************************************************/
;
; define restart vectors
;
;*********************************************************************/

   .def     _c_int0
   .def     _spur_rst
   .extern  _main

;
; system reset vector - rst0
```

```
;
   .assume  ADL=0
   define   .rst0 ,    space=ROM, org=0000h
   segment  .rst0

   jp.lil   _c_int0

;
; rst 1
;
   define   .rst1 ,    space=ROM, org=0008h
   segment  .rst1
   ld       a,1
   jp       _spur_rst

;
; rst 2
;
   define   .rst2 ,    space=ROM, org=0010h
   segment  .rst2
   ld       a,2
   jp       _spur_rst

;
; rst 3
;
   define   .rst3 ,    space=ROM, org=0018h
   segment  .rst3
   ld       a,3
   jp       _spur_rst

;
; rst 4
;
   define   .rst4 ,    space=ROM, org=0020h
   segment  .rst4
   ld       a,4
   jp       _spur_rst

;
; rst 5
;
   define   .rst5 ,    space=ROM, org=0028h
   segment  .rst5
```

```
   ld       a,5
   jp       _spur_rst

;
; rst 6
;
   define   .rst6 ,    space=ROM, org=0030h
   segment  .rst6
   ld       a,6
   jp       _spur_rst

;
; rst 7
;
   define   .rst38,    space=ROM, org=0038h
   segment  .rst38
   ld       a,7

_spur_rst:
   halt
   jr       $

;
; nmi interrupt
;
   define   .nmitbl,    space=ROM, org=0066h
   segment  .nmitbl
   .extern  _nmi_int
   jp.lil   _nmi_int

;*****************************************************************/
;
; interrupt vector table
;
;*****************************************************************/
   define   .ivectbl,   space=ROM, org=0200h
   segment  .ivectbl

;
; define interrupt service routines
;
   .extern  _spur_int
   .extern  _tmr1_int
   .extern  _i2c_int
```

```
;
;  Note: IVec Table must be placed on a 512 byte boundary
;
   dl    _spur_int  ; 00h unused              200
   dl    _spur_int  ; 04h unused              204
   dl    _spur_int  ; 08h unused              208
   dl    _spur_int  ; 0ch unused              20c
   dl    _spur_int  ; 10h unused              210
   dl    _spur_int  ; 14h unused              214
   dl    _spur_int  ; 18h unused              218
   dl    _spur_int  ; 1ch unused              21c
   dl    _spur_int  ; 20h unused              220
   dl    _spur_int  ; 24h unused              224
   dl    _spur_int  ; 28h unused              228
   dl    _spur_int  ; 2ch unused              22c
   dl    _spur_int  ; 30h unused              230
   dl    _spur_int  ; 34h unused              234
   dl    _spur_int  ; 38h unused              238
   dl    _spur_int  ; 3ch unused              23c
   dl    _spur_int  ; 40h emac_rx             240
   dl    _spur_int  ; 44h emac_tx             244
   dl    _spur_int  ; 48h emac_sys            248
   dl    _spur_int  ; 4ch pll                 24c
   dl    _spur_int  ; 50h flash               250
   dl    _spur_int  ; 54h tim0                254
   dl    _tmr1_int  ; 58h tim1                258
   dl    _spur_int  ; 5ch tim2                25c
   dl    _spur_int  ; 60h tim3                260
   dl    _spur_int  ; 64h unused              264
   dl    _spur_int  ; 68h unused              268
   dl    _spur_int  ; 6ch rtc                 26c
   dl    _spur_int  ; 70h uart0               270
   dl    _spur_int  ; 74h uart1               274
   dl    _i2c_int   ; 78h i2c                 278
   dl    _spur_int  ; 7ch spi                 27c
   dl    _spur_int  ; 80h pa0                 280
   dl    _spur_int  ; 84h pa1                 284
   dl    _spur_int  ; 88h pa2                 288
   dl    _spur_int  ; 8ch pa3                 28c
   dl    _spur_int  ; 90h pa4                 290
   dl    _spur_int  ; 94h pa5                 294
   dl    _spur_int  ; 98h pa6                 298
   dl    _spur_int  ; 9ch pa7                 29c
```

```
   dl    _spur_int  ; a0h pb0                  2a0
   dl    _spur_int  ; a4h pb1                  2a4
   dl    _spur_int  ; a8h pb2                  2a8
   dl    _spur_int  ; ach pb3                  2ac
   dl    _spur_int  ; b0h pb4                  2b0
   dl    _spur_int  ; b4h pb5                  2b4
   dl    _spur_int  ; b8h pb6                  2b8
   dl    _spur_int  ; bch pb7                  2bc
   dl    _spur_int  ; c0h pc0                  2c0
   dl    _spur_int  ; c4h pc1                  2c4
   dl    _spur_int  ; c8h pc2                  2c8
   dl    _spur_int  ; cch pc3                  2cc
   dl    _spur_int  ; d0h pc4                  2d0
   dl    _spur_int  ; d4h pc5                  2d4
   dl    _spur_int  ; d8h pc6                  2d8
   dl    _spur_int  ; dch pc7                  2dc
   dl    _spur_int  ; e0h pd0                  2e0
   dl    _spur_int  ; e4h pd1                  2e4
   dl    _spur_int  ; e8h pd2                  2e8
   dl    _spur_int  ; ech pd3                  2ec
   dl    _spur_int  ; f0h pd4                  2f0
   dl    _spur_int  ; f4h pd5                  2f4
   dl    _spur_int  ; fch pd7                  2fc

   .sect    “code”
   .ref     _nmi_int            ; nmi interrupt handler

   .assume  ADL=1

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

 ;
 ; f91_bb:
 ;
 ; ext flash - 2mx8
 ; ext sram  - 2mx8
 ;
 ; cs0 - ExtFlash must be on   1M boundary (address depends on the
 ;                cs_lbr - see the 139 U22)
 ; cs1 - ExtSram  must be on 512K boundary (address depends on which
 ;                srams are stuffed on the board - see the 139 U22)
 ;
```

```
 ;
 ;   CHIP           Size         CS0             CS1           SPL
 ;                           LBR     UBR     LBR     UBR
 ;
 ;
 ; L92 Sram          2M      0x00    0x1F      Disable       0x1FFFFF
 ; L92 Flash         2M      0x00    0x1F    0x20    0x3F    0X3FFFFF
 ; L92 IntFlash      0K
 ;
 ; F91 Sram          2M      0x00    0x1F      Disable       0x1FFFFF
 ; F91 Flash         2M      0x00    0x1F    0x20    0x3F    0X3FFFFF
 ; F91 IntFlash    256K      0x40    0x5f    0x60    0x7F    0X7FFFFF
 ;
 ; F92 Sram          2M      0x00    0x1F      Disable       0x1FFFFF
 ; F92 Flash         2M      0x00    0x1F    0x20    0x3F    0X3FFFFF
 ; F92 IntFlash    128K      0x40    0x5f    0x60    0x7F    0X7FFFFF
 ;
 ; F93 Sram          2M      0x00    0x1F      Disable       0x1FFFFF
 ; F93 Flash         2M      0x00    0x1F    0x20    0x3F    0X3FFFFF
 ; F93 IntFlash     64K      0x40    0x5f    0x60    0x7F    0X7FFFFF
 ;
 ;
 ;

.TOSPS      .equ  ffffh

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

_c_int0:
   di
   ld.sis   sp,.TOSPS        ; Setup SPS
   ld.lil   sp,.TOSPL        ; Setup SPL

;
; set madl - this causes adl state to be stacked when an interrupt
occurs
;            C interrupt routines expect the adl bits to be stacked.
   stmix

;
; memory map
;
```

```
;
; csn[0] set jumper to external flash for Flash, IntFlash options
;        set jumper to external Sram for Sram option

   .ifdef Flash
      ld       a,00h
      out0     (CS0_LBR),a
      ld       a,01fh
      out0     (CS0_UBR),a
      ld       a,068h
      out0     (CS0_CTL),a
   .endif
   .ifdef IntFlash
      .ifdef ez80F91
         ld       a,040h
         out0     (CS0_LBR),a
         ld       a,05Fh
         out0     (CS0_UBR),a
         ld       a,068h
         out0     (CS0_CTL),a
      .endif
      .ifdef ez80F92
         ld       a,040h
         out0     (CS0_LBR),a
         ld       a,05Fh
         out0     (CS0_UBR),a
         ld       a,068h
         out0     (CS0_CTL),a
      .endif
      .ifdef ez80F93
         ld       a,040h
         out0     (CS0_LBR),a
         ld       a,05Fh
         out0     (CS0_UBR),a
         ld       a,068h
         out0     (CS0_CTL),a
      .endif
      .ifdef ez80L92
         ld       a,000h
         out0     (CS0_LBR),a
         ld       a,000h
         out0     (CS0_UBR),a
         ld       a,000h
         out0     (CS0_CTL),a
```

```
      .endif
   .endif
   .ifdef Sram
      ld       a,000h
      out0     (CS0_LBR),a
      ld       a,01fh
      out0     (CS0_UBR),a
      ld       a,48h
      out0     (CS0_CTL),a
   .endif
;
; csn[1] set jumper to external Sram for Flash, IntFlash options
;        set jumper to external Flash for Sram option
;
 ; cs1 = ext sram
   .ifdef Flash
      ld       a,020h
      out0     (CS1_LBR),a
      ld       a,03fh
      out0     (CS1_UBR),a
      ld       a,48h
      out0     (CS1_CTL),a
   .endif
   .ifdef IntFlash
      ld       a,060h
      out0     (CS1_LBR),a
      ld       a,07fh
      out0     (CS1_UBR),a
      ld       a,48h
      out0     (CS1_CTL),a
   .endif
   .ifdef Sram
      ld       a,0h       ; disable cs1
      out0     (CS1_CTL),a
   .endif
;
; csn[2]
; External I/O - memory space
; A15:13   Device
;  000     SCC
;  001     I2C
;  010     CF_MEM
;  011     CF_IO1
;  100     CF_IO2
```

```
;  101     EXP_CS
;  110     CREG
;  111     SREG
;
 ; setup cs2 for peripheral I/O intel bus mode at 0x80
   ld       a,080h
   out0     (CS2_LBR),a
   ld       a,080h
   out0     (CS2_UBR),a ;
   ld       a,008h
   out0     (CS2_CTL),a ;
   ld       a,087h
   out0     (CS2_BMC),a ;
;
; csn[3] - set to 0x90_0000 - 0x91_ffff (128K for lcd controller/lcd
buffer)
;        - use intel mode with 4 clks/state, memory mode
;
;
   ld       a,090h
   out0     (CS3_LBR),a
   ld       a,091h
   out0     (CS3_UBR),a
   ld       a,008h
   out0     (CS3_CTL),a
   ld       a,084h
   out0     (CS3_BMC),a ;

;
; Initialize the .BSS section to zero
;
.if   .INITBSS
   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]=src=.bss
   ld bc,.BSS_LENGTH   ; [bc]=byte cnt
   ld (hl),0           ; clear first byte
   dec   bc            ; 1st byte's taken care of
   jr z,_c_bss_done    ; Just 1 byte ...
   ld de,.BSS_BASE+1   ; [de]=dst=.bss+1
```

```
   ldir                ;
.endif                 ; .INITBSS

_c_bss_done:

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

   ld hl,.DATA_LENGTH    ; Check for non-zero length
   ld bc,0
   or a,a
   sbc   hl,bc
   jr z,_c_init_done     ; .DATA is zero-length ...
$$:
   ld hl,.DATA_COPY      ; [hl]=src=.data_copy
   ld de,.DATA_BASE      ; [de]=dst=.data
   ld bc,.DATA_LENGTH    ; [bc]= data length
   ldir                  ; Copy the data
.endif

_c_init_done:

   di
 ; place null argv[0] on stack
   ld hl,0               ; hl=NULL
   push hl               ; argv[0] = NULL

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

 ; set de=argc=0
   ld de,0               ; argc==0

   call  _main           ; de=argc, hl=&argv[0]
   pop   af              ; clean the stack (null argv[0]
   ret.l                 ; return with ADL=0
```

initInts.s

```
;*****************************************************************/
;* Author: DP Consulting, Inc.    Dick Pado
;* Source: init_ints.s
;* Description: setup/ enable interrupts
;*
;* Rev   Date     Description
;* 1.0   06/05/03 initial release
;*
;*****************************************************************/

   .NOLIST
   .include “ez80f91.inc”
   .LIST

   .assume  ADL=1
   .sect    “code”

;*****************************************************************/
;
; _init_ints - init ez80 i register, enable interrupts
;
;*****************************************************************/

   .def     _init_ints

_init_ints:
   im       2         ;interrupt mode 2
   ld       hl,0002h  ; ivectbl at 0x000200
;  ld       i,hl      ; opcode = ed c7
   db       0edh
   db       0c7h
   ei                 ; enable interrupts
   ret
```

isr.s

```
;*****************************************************************/
```

```
;* Author: DP Consulting, Inc.    Dick Pado
;* Source: isr.s
;* Description: Interrupt Service Routines
;*
;* Rev   Who   Date     Description
;* 1.0   DPC   11/15/02 initial release
;* 1.0   DPC    3/19/03 support F91 timers
;*****************************************************************/

   .extern  _tmr1_cnt
   .extern  _ez80_i2c_reset
   .extern  _pcf8584_i2c_reset
   .extern  _nmi_flag
   .extern  _printf

   .NOLIST
   .include “ez80f91.inc”
   .LIST

   .sect    “STRSECT”
_spurStr:   db “Spurious Interrupt”, 0ah, 0
_nmiStr:    db “NMI Interrupt”, 0ah, 0

   .sect    “code”
   .assume  ADL=1

   .def     _spur_int
_spur_int:
   exx
   ex       af,af’
   push     iy
   ld       bc,_spurStr
   push     bc
   call     (_printf)
   pop      iy

spur_L1:
   jr       spur_L1

   exx
   ex       af,af’
   pop      iy
   ei
   reti.l
```

```
   .def     _tmr1_int
_tmr1_int:

   exx
   ex       af,af'

 ; clear interrupt
   in0      a,(TMR1_IIR)

 ; increment tmier1 count
   ld       bc,(_tmr1_cnt)
   inc.s    bc
   ld       a,c
   ld       (_tmr1_cnt+0),a
   ld       a,b
   ld       (_tmr1_cnt+1),a
   nop

   exx
   ex       af,af'
   ei
   reti.l

   .def     _nmi_int
_nmi_int:

   exx
   ex       af,af'
   push     iy

 ; print message
   ld       bc,_nmiStr
   push     bc
   call     (_printf+0)
   pop      iy

 ; set nmi flag
   ld       a,1
   ld       (_nmi_flag),a

   exx
   ex       af,af'
```

```
   pop      iy
   ei
   reti.l

   .def     _i2c_int
   .extern  _ez80_i2c_isr

_i2c_int:

   push     ix
   push     af
   push     bc
   push     de
   push     hl
   push     ix
   push     iy
   call     (_ez80_i2c_isr)
   pop      iy
   pop      ix
   pop      hl
   pop      de
   pop      bc
   pop      af
   pop      ix
   ei
   reti.l
```

main.c

```
/*****************************************************************************
**
** Author: DP Consulting, Inc.    Dick Pado
**
** Source: main.c
**
** Description: F91_BB I2C AppNote
**
** Rev   Who   Date     Description
**
** 1.0   DPC   06/05/03 initial release
**
*****************************************************************************/
```

```
#define  MAIN

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <ez80f91.h>
#include “types.h”
#include “ez80RegDefs.h”
#include “tb_vars.h”
#include “ez80_i2c.h”
#include “bb_cfg.h”

extern   int   main(void);

int main() {
   int         testNum;
   int         choice;
   int         errRet;

   // h/w setup
   init();

   repeat       = OFF;

   // clear buffer and variable from previous testing
   kb_flush();
   inbuf[0] = ‘\0’;

   do {
      printf(
“\n\t************************************************\n”);
     printf(   “\t*      F91 Engineering Development Board      *\n”);
     printf(   “\t*      I2C Application Note                    *\n”);
      printf(
“\t************************************************\n”);

      printf(“\n”);
      printf(“ x. Exit\n”);
      printf(“ 1. reset ez80 i2c\n”);
      printf(“ 2. EZ80    Master tx\n”);
      printf(“ 3. PCF8584 Master tx\n”);
      printf(“ 4. EZ80    Master rx\n”);
      printf(“ 5. PCF8584 Master rx\n”);
```

```
printf("99. Set Test Options\n");
printf("\n\tEnter Selection: ");

printf("Choose a test number or 'x' to exit: ");
gets(inbuf);

if(inbuf[0] == 'x')
{
   printf("Exiting . . .\n");
   printf("End of Testing Program . . .\n");
   break;
}

testNum = atoi(inbuf);

//clear buffer:
kb_flush();

do {
   switch(testNum) {
   case 0: break;
   case 1: ez80_i2c_reset(); break;
   case 2: errRet = ez80_mas_tx();
           if (repeat==OFF)
              printf("eZ80 Master Tx Test %s\n",errRet ?
                     "Failed" : "Passed");
           break;
   case 3: errRet = pcf8584_mas(I2C_WR);
           if (repeat==OFF)
              printf("PCF8584 Master Tx Test %s\n",errRet ?
                     "Failed" : "Passed");
           break;
   case 4: errRet = ez80_mas_rx();
           if (repeat==OFF)
              printf("eZ80 Master Rx Test %s\n",errRet ?
                     "Failed" : "Passed");
           break;
   case 5: errRet = pcf8584_mas(I2C_RD);
           if (repeat==OFF)
              printf(" PCF8584Master Rx Test %s\n",errRet ?
                     "Failed" : "Passed");
           break;
   case 99:
      do
```

```
            {
               printf("Set Test Options\n");
               printf("1. repeat    = %s\n", repeat ? "ON" : "OFF");
               printf("2. stopOnErr = %s\n", stopOnErr ? "ON" : "OFF");
               printf("q. Quit\n Enter Selection: ");
               gets(inbuf);

               choice = atoi(inbuf);
               kb_flush();

               if(choice == 0)
               {
                  printf("Exiting sub_menu\n");
                  //clear variable and buffer:
                  inbuf[0] = '\0';
                  break;
               }

               if(choice == 1)
                  repeat = (repeat == OFF) ? ON : OFF;
               if(choice == 2)
                  stopOnErr = (stopOnErr+1)%2;
            } while(choice !=0);
            break;
         default:
           printf("You must choose a valid option or Zero to quit.\n");
            break;
         }
      } while (repeat==ON && !kbhit() && testNum<6);
      if (kbhit()) {
         kb_flush();
      }
   } while(inbuf[0] != 'x');
   return 0;
}
```

sio.c

```
/*****************************************************************
** Author: DP Consulting, Inc.    Dick Pado
** Source: sio.c
** Description: Routines to support the serial devices on the
** breadboard
```

```
**
** Rev   Date     Description
** 1.0   11/15/02 initial release
**
**********************************************************************/

#include <stdio.h>
#include <string.h>
#include <ez80f91.h>
#include “types.h”
#include “ez80RegDefs.h”
#include “tb_vars.h”

//************ UART0 control **********************

int kbhit(void) {
   return(UART0_LSR & RxDataRdy);
}

void kb_flush()
{
   while(kbhit())
      getch();
}

int getch(void)
{
   int ch;

   // wait for data character available
   while (!(UART0_LSR & RxDataRdy));
   ch = UART0_RBR;

   if ((ch == ‘\r’ || ch == (‘\r’|0x80)) && uart0Mode==ASCII)
      ch = ‘\n’;

   // echo character back
   if (uart0Echo)
      putch(ch);
   return(ch);
}

// pass the character to be xmitted
void putch(int c)
```

```
{
   // if lf, then send cr + lf
   if (c == '\n' && uart0Mode==ASCII) {
      // wait for tx holding reg empty
      while (!(UART0_LSR & TxRdy));
      UART0_THR = '\r';
   }
   // wait for tx holding reg empty
   while (!(UART0_LSR & TxRdy));
   UART0_THR = c;
}
```

funcs.c

```
/*************************************************************************
** Author: DP Consulting, Inc.    Dick Pado
** Source: funcs.c
** Description: I2C support Routines
**
** Rev   Date     Description
** 1.0   11/15/02 initial release
**
*************************************************************************/

#include "types.h"
#include <ez80f91.h>
#include "ez80RegDefs.h"
#include "pcf8584.h"
#include "tb_vars.h"

//************************************************************************
//
// rd_i2c_creg - pcf8584 control register read
//
// input parameters: ctrl specifies the register to be read.
//
//************************************************************************

u_char rd_i2c_dreg(u_char ctrl) {
      I2C_CREG = ctrl;
      return(I2C_DREG);
}
```

```
//*********************************************************************
//
// wr_i2c_creg - pcf8584 control register write
//
// input parameters: ctrl specifies the register
//                   data specifies the data to be written to the desired
reg
//
//*********************************************************************

void wr_i2c_creg(u_char ctrl, u_char data) {
      I2C_CREG = ctrl;
      I2C_DREG = data;
}

//*********************************************************************
//
// startTimer - starts timer0 for the desired time delay
//
// input parameters: toh - specifies the timer high value
//                   tol - specifies the timer low  value
//
//*********************************************************************

void startTimer(u_char toh, u_char tol) {

   // Stop timer first
   TMR0_CTL  = TmrSelSys | TmrClkDiv16 | Tmr1Shot | TmrDis;
   // Disable interrupts for timer 0
   TMR0_IER  = 0;
   // set timeout
   TMR0_RR_H = toh;
   TMR0_RR_L = tol;
   TMR0_CTL  = TmrSelSys | TmrClkDiv16 | Tmr1Shot | TmrLoad;
   // Wait until timeout flag is cleared
   while(TMR0_TO);
   // start timer
   TMR0_CTL  = TmrSelSys | TmrClkDiv16 | Tmr1Shot | TmrEn | TmrLoad;

}

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

```
// timeDelay - delays 'numTenths' * .1 seconds
//
//*********************************************************************

void timeDelay(int numTenths) {
   int   i,j;

   for (j=0; j<=numTenths; j++) {
      // 1 second delay
      for (i=0; i<5; i++) {
         startTimer(TO20MS>>8,TO20MS%256);
         while(!(TMR0_TO));
      }
   }
}
```

i2c.c

```
//*********************************************************************
** Author: DP Consulting, Inc.    Dick Pado
** Source: i2c.c
** Description: Routines to test the eZ80 I2C
**
** Rev   Date     Description
** 1.0   06/05/03 initial release
**
*********************************************************************/

#include <stdio.h>
#include <ez80f91.h>
#include "types.h"
#include "ez80RegDefs.h"
#include "pcf8584.h"
#include "ez80_i2c.h"
#include "tb_vars.h"

#define I2C_PKTSZ 10    // sets the size of the i2c transfers

//*********************************************************************
//
// peripheral_reset - assert the peripheral reset - hardware reset to
pcf8584
//
```

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

void peripheral_reset(void) {
   I2C_SET_RESET;
   timeDelay(1);
   I2C_CLR_RESET;

   // initialize pcf8584 I2C controller
   pcf8584_i2c_init(PCF8584_SLVADDR);
}

//*********************************************************************
//
// pcf8584_i2c_init - initialize the pcf8584 i2c
//
// input parameter: slvAddr - the pcf8584 i2c address
//
//*********************************************************************

void pcf8584_i2c_init(u_char slvAddr) {

   // program sequence for initial setup
   wr_i2c_creg(I2C_IOWN_ADDR, slvAddr);
   wr_i2c_creg(I2C_IIVEC, 0x00);
   wr_i2c_creg(I2C_ICLKREG, I2C_CLKSEL_90 | I2C_CLKFREQ_3);
   I2C_CREG = I2C_CTLSTAT;

   startTimer(TO20MS>>8,TO20MS%256);
   while(!TMR0_TO);
}

//*********************************************************************
//
// ez80_i2c_init - initialize the ez80 i2c
//
// input parameter: slvAddr - the ez80 i2c address
//
//*********************************************************************

void ez80_i2c_init( u_char slvAddr) {
   // disable i2c
   I2C_CTL     = 0;
   // set the clock
   I2C_CCR     = EZ80_CLKDATA;
```

```
   // set 7 bit slave address
   I2C_XSAR    = 0;
   I2C_SAR     = (slvAddr<<1) | EZ80_CALLADDR_ENA;
}

//*********************************************************************
//
// ez80_i2c_reset - software reset
//
//*********************************************************************

void ez80_i2c_reset() {

   // reset ez80 i2c
   I2C_SRR = 0xff;

   // delay
   startTimer(TO20MS>>8,TO20MS%256);
   while(!TMR0_TO);

   I2C_CTL  = 0;
}

//*********************************************************************
//
// ez80_mas_tx - ez80 i2c in master mode sends packet to pcf8584.
//
// ez80 i2c handles by the ez80_i2c_isr.
// pcf8584 i2c is polled.
//
//*********************************************************************

u_short ez80_mas_tx(void) {
   int         i;
   int         nbytes = 0;
   int         stat;
   u_char      mode;
   u_char      data;
   int         retCode = 0;
   u_char      busy = 1;

   // set slave receive / slave txmit mode
   mode = I2C_WR;
```

```
   i2c_error = 0;

   // reset to ez80 i2c
   if (repeat==OFF)
      ez80_i2c_reset();
   // reset/init the pcf8584 i2c controller
   pcf8584_i2c_init(PCF8584_SLVADDR);

   // setup data for ez80 i2c isr
   i2c_slv_addr = PCF8584_SLVADDR;
   i2c_nbytes2xfer = I2C_PKTSZ;
   i2c_nbytesXferred = 0;
   i2c_mode = I2C_WR;

   nmi_flag = 0;

   // init ez80 i2c
   ez80_i2c_init(EZ80_SLVADDR);

   // set start condition - this will put the i2c into master mode
   // and a start interrupt will occur 0x08
   // The i2c_int isr will transfer the slave address and the data bytes
   //
   I2C_CTL  = EZ80_ENAB | EZ80_STA | EZ80_IEN;

   while(busy && nmi_flag==0) {
      // wait for pcf8584 interrupt
      startTimer(TO20MS>>8,TO20MS%256);
      i = 0;
      while(((stat=I2C_SREG) & S1_PIN) && nmi_flag==0) {
         if (TMR0_TO) {
            i++;
            if (i>I2C_TO_DELAY) {
               printf("Timeout Error1 in ez80_master_tx waiting for
S1_pin negation\n");
               printf("Press NMI PB to exit\n");
               while(!nmi_flag);
            } else {
               startTimer(TO20MS>>8,TO20MS%256);
            }
         }
      }
      // process the interrupt
      if (stat & S1_AAS) {
```

```
         // read address, clears interrupt
         data = I2C_DREG;
         // dummy read - don’tknow why but it seemed to make it work
         I2C_DREG;
         // check slave receive / slave txmit mode
         if (mode != (data & I2C_RD)) {
            printf(“8584 Error - wrong mode exp %x got
%x\n”,mode,data&I2C_RD);
            retCode = 1;
            busy = 0;
         }
         // check address
         if ((data>>1)!= PCF8584_SLVADDR) {
            printf(“8584 Error: slave address mismatch\n”);
            retCode = 1;
            busy = 0;
         }
      } else if (stat & S1_BER) {

         // clear interrupt
         I2C_CREG = I2C_CTLSTAT;
         printf(“8584 Error: bus error\n”);
         retCode = 1;
         busy = 0;
      } else if (stat & S1_LAB) {
         printf(“master Lost Arbitration - unexpected status\n”);
         retCode = 1;
         busy = 0;
      } else if (stat&S1_STS) {
         busy = 0;
      } else {
         if (mode&I2C_RD) {
            // slave tranmit mode
            nbytes++;
            // slave tx mode - check LRB
            if (stat&S1_AD0_LRB) {
               busy = 0;
            } else {
            }
            // write byte to data reg, clears interrupt
            I2C_DREG = nbytes;
         } else {
            // slave receive mode
            nbytes++;
```

```
            // read data, clears interrupt
            data = I2C_DREG;
            // check data
            if (data!=nbytes) {
               retCode = 1;
               printf("rx error on byte %d  got %x\n",nbytes,data);
            }
         }
      }
   }
   return retCode;
}

//*********************************************************************
//
// ez80_mas_rx - ez80 i2c in master mode to receive packet from pcf8584
//
// pcf8584 sends an incrementing pattern to the ez80 i2c.
// Data pattern checked in the ez80_i2c_isr.
// pcf8584 i2c is polled.
//
//*********************************************************************

u_short ez80_mas_rx(void) {
   int         i;
   int         nbytes = 0;
   int         stat;
   u_char      mode = 0;
   u_char      data;
   int         retCode = 0;
   u_char      busy = 1;

   i2c_error = 0;

   // reset/init the pcf8584 i2c controller
   pcf8584_i2c_init(PCF8584_SLVADDR);

   // setup data for ez80 i2c isr
   i2c_slv_addr = PCF8584_SLVADDR;
   i2c_nbytes2xfer = I2C_PKTSZ;
   i2c_nbytesXferred = 0;
   i2c_mode = I2C_RD;

   nmi_flag = 0;
```

```
   // s/w reset to ez80 i2c
   if (repeat==OFF)
      ez80_i2c_reset();
   ez80_i2c_init(EZ80_SLVADDR);

   // set start condition - this will put the i2c into master mode
   // and and send start interrupt will occur
   // The i2c_int isr will transfer the slave address and the data bytes
   //
   I2C_CTL  = EZ80_ENAB | EZ80_STA | EZ80_IEN;

   // wait for AAS bit set
   // NOTE: aas bit gets set before pin goes low: stat=84
   //
   startTimer(TO20MS>>8,TO20MS%256);
   i = 0;
   while((!((stat=I2C_SREG) & S1_AAS)) && nmi_flag==0) {
      if (TMR0_TO) {
         i++;
         if (i>I2C_TO_DELAY) {
           printf("Timeout Error1 in ez80_master_rx waiting for S1_AAS
assertion\n");
            return 2;
         } else {
            startTimer(TO20MS>>8,TO20MS%256);
         }
      }
   }
   while(busy && !nmi_flag) {
      // wait for pcf8584 interrupt
      startTimer(TO20MS>>8,TO20MS%256);
      i = 0;
      while(((stat=I2C_SREG) & S1_PIN) && nmi_flag==0) {
         if (TMR0_TO) {
            i++;
            if (i>I2C_TO_DELAY) {
               printf("Timeout Error2 in ez80_master_rx waiting for
S1_PIN negation\n");
               return 2;
            } else {
               startTimer(TO20MS>>8,TO20MS%256);
            }
         }
```

```
}
// process the interrupt
if (stat & S1_AAS) {
   // read address, does not clear interrupt in slave receive mode
    data = I2C_DREG;
    // set slave receive / slave txmit mode
    mode = data & I2C_RD;
    // check address - Note this doesn't account for genearal
    // call addresses
    if ((data>>1)!= PCF8584_SLVADDR)
       retCode = 1;
    if (mode & I2C_RD) {
       // load first byte to send
       nbytes++;
       I2C_DREG = nbytes;
    }
} else if (stat & S1_BER) {
    // clear interrupt
    I2C_CREG = I2C_CTLSTAT;
    // exit loop
    retCode = 1;
    busy = 0;
} else if (stat & S1_LAB) {
    printf("master Lost Arbitration - unexpected status\n");
    retCode = 1;
    busy = 0;
} else if (stat&S1_STS) {
    // clear interrupt
    I2C_CREG = I2C_CTLSTAT;
    // check if all bytes received
    if (nbytes!=i2c_nbytes2xfer)
       retCode = 1;
    // exit loop
    busy = 0;
} else {
    //
    // handle data transfers here
    //
    if (mode&I2C_RD) {
       // slave tx mode - check LRB
       if (stat&S1_AD0_LRB) {
          // clear interrupt
          I2C_CREG = I2C_CTLSTAT;
          // exit loop
```

```
               busy = 0;
            } else {
               nbytes++;
               // write byte to data reg, this clears interrupt
               I2C_DREG = nbytes;
            }
         } else {
            // slave receive mode
            nbytes++;
            // read data, clears interrupt
            data = I2C_DREG;
            // check data
            if (data!=nbytes) {
               retCode = 1;
               printf("rx error on byte %d  got %x\n",nbytes,data);
            }
            // if stop condition, then exit loop, done
            if (stat&S1_STS) {
               // check if all bytes received
               if (nbytes!=i2c_nbytes2xfer)
                  retCode = 1;
               busy = 0;
            }
         }
      }
   }
   // return i2c_error or'ed with retCode
   return (retCode | i2c_error);
}

//*****************************************************************
//
// pcf8584_mas
//
// input parameter: mode - specifies read or write
//                        - write is master tx
//                        - read  is master rx
//
//*****************************************************************

u_short pcf8584_mas(int mode) {
   int         i;
   int         nbytes = 0;
   int         busy = 1;
```

```
   int         retCode = 0;
   u_char      data;
   u_char      stat;
   u_char      firstRead = 1;

   pcf8584_i2c_init(PCF8584_SLVADDR);

   // reset ez80 i2c
   I2C_SRR = 0xff;
   ez80_i2c_init(EZ80_SLVADDR);
   i2c_error = 0;
   i2c_nbytes2xfer = I2C_PKTSZ;
   i2c_nbytesXferred = 0;

   nmi_flag = 0;

   // Enable the i2c, enable interrupts
   I2C_CTL  = EZ80_ENAB | EZ80_ACK | EZ80_IEN;

   I2C_DREG = (EZ80_SLVADDR<<1) | mode;

   startTimer(TO20MS>>8,TO20MS%256);
   i = 0;
   while(!(I2C_SREG & S1_BBN) && !nmi_flag) {
      if (TMR0_TO) {
         i++;
         if (i>I2C_TO_DELAY) {
           printf("Timeout Error1 in pcf8584_master waiting for S1_BBN
assertion\n");
         } else {
            startTimer(TO20MS>>8,TO20MS%256);
         }
      }
   }
   I2C_CREG = I2C_CTLSTAT | S1_STA;

   // polling loop
   while(busy && !nmi_flag) {
      startTimer(TO20MS>>8,TO20MS%256);
      i = 0;
               // process the interrupt
      while(((stat=I2C_SREG) & S1_PIN) && nmi_flag==0) {
         if (TMR0_TO) {
            i++;
```

```
            if (i>I2C_TO_DELAY) {
               printf("Timeout Error2 in pcf8584_master waiting for
S1_PIN negation\n");
            } else {
               startTimer(TO20MS>>8,TO20MS%256);
            }
         }
      }
      if (stat & S1_AAS) {
         retCode = 1;
         busy = 0;
      } else if (stat & S1_BER) {
         // clear interrupt
         I2C_CREG = I2C_CTLSTAT;
         retCode = 1;
         busy = 0;
      } else if (stat & S1_LAB) {
         printf("8584 Error: master Lost Arbitration - unexpected
status\n");
         retCode = 1;
         busy = 0;
      } else if (stat&S1_STS) {
        printf("8584 Error: Stop condition detected in master mode\n");
         retCode = 1;
         busy = 0;
      } else {
         //
         // handle data transfers here
         //
         if (mode&I2C_RD) {
            // increment to next byte to read
            nbytes++;
            // master receive mode
            // check if last byte to read
            if (stat & S1_AD0_LRB) {
               // got nack - send stop condition
               // clear interrupt, send stop condition
               I2C_CREG = I2C_CTLSTAT | S1_STO;
               busy = 0;
            } else {
               if (nbytes==i2c_nbytes2xfer-1) {
                  // disable auto ack - want to nack last byte to end
transfer
                  I2C_CREG = S1_ES0;
```

```
            }
         }
         // read data, clear interrupt
         data = I2C_DREG;
         if (!firstRead) {
            // check data for incrementing pattern
            if (data!=nbytes) {
               retCode = 1;
               printf("8584 Error: rx data exp %d  got
%x\n",nbytes,data);
            }
         } else {
            firstRead = 0;
            nbytes--;
         }
      } else {
         // master tx mode
         if (nbytes==i2c_nbytes2xfer) {
            // clear interrupt, send stop condition
            I2C_CREG = I2C_CTLSTAT | S1_STO;
            busy = 0;
         } else {
            nbytes++;
            // write data - incrementing pattern
            I2C_DREG = nbytes;
         }
      }
   }
   return(i2c_error | retCode);
}

//*************************************************************
//
// ez80_i2c_isr - ez80 i2c interrupt service routine
//
// read the status to determine the cause of the interrupt
// handle the interrupt
//
// The following global parameters must be set prior to starting
transfer:
//
//  i2c_slv_addr      - the slave address
//  i2c_nbytes2xfer   - the size of the i2c transfer
```

```
//  i2c_nbytesXferred - the number of bytes actually transferred (init
to 0)
//  i2c_mode          - sets the direction of the transfer (read or
write)
//
//  The define I2C_PKTSZ holds the number of bytes to transfer
//
//  The data packet is an incrementing pattern 1 to I2C_PKTSZ
//
//  When the ez80 I2C is receiving data, the data is checked for the
correct
//  data
//
//*****************************************************************

void ez80_i2c_isr(void) {
   u_char      stat;
   u_char      data;

   stat = I2C_SR;

   switch(stat) {
   case EZ80_STARTCOND:
      // send slave address
      I2C_DR = (i2c_slv_addr<<1) | i2c_mode;
      // clear interrupt flag
      I2C_CTL &= ~EZ80_IFLG;
      break;
   case EZ80_MWADDRACK:
      if ((i2c_mode&I2C_RD)==I2C_WR) {
         i2c_nbytesXferred++;
         // send data - incrementing pattern starting at 1 to I2C_PKTSZ
         I2C_DR = i2c_nbytesXferred;
         // clear interrupt flag
         I2C_CTL &= ~EZ80_IFLG;
      } else {
         printf("Got EZ80_MWADDRACK in master receive mode - Press NMI
PB to exit\n");
         while(nmi_flag==0);
         // clear interrupt flag
         I2C_CTL &= ~EZ80_IFLG;
      }
      break;
   case EZ80_MWADDRNACK:
```

```
      // clear interrupt flag, set stop condition
      I2C_CTL &= (I2C_CTL & ~EZ80_IFLG) | EZ80_STP;
      break;
   case EZ80_MRADDRACK:
      if (i2c_mode&I2C_RD) {
         if (i2c_nbytes2xfer>1) {
            // clear interrupt flag, set ACK
            I2C_CTL = (I2C_CTL & (~EZ80_IFLG)) | EZ80_ACK;
         } else {
            // clear interrupt flag
            I2C_CTL &= ~EZ80_IFLG;
         }
      } else {
         printf("ERROR: Got EZ80_MRADDRACK in master transmit mode -
press NMI PB to exit\n");
         while(nmi_flag==0);
         // clear interrupt flag
         I2C_CTL &= ~EZ80_IFLG;
      }
      break;
   case EZ80_MRADDRNACK:
      // clear interrupt flag, set stop condition
      I2C_CTL = (I2C_CTL & ~EZ80_IFLG) | EZ80_STP;
      break;
   case EZ80_MTXDATAACK:
      if (i2c_nbytesXferred==i2c_nbytes2xfer) {
         // clear interrupt flag, set stop condition
         I2C_CTL = (I2C_CTL & ~EZ80_IFLG) | EZ80_STP;
      } else {
         i2c_nbytesXferred++;
         // send data - incrementing pattern starting at 1 to I2C_PKTSZ
         I2C_DR = i2c_nbytesXferred;
         // clear interrupt flag
         I2C_CTL &= ~EZ80_IFLG;
      }
      break;
   case EZ80_MTXDATANACK:
      // write out dummy byte
      I2C_DR = i2c_nbytesXferred;
      printf("mas tx data nack on byte %d, send
stop\n",i2c_nbytesXferred);
      // clear interrupt flag, set stop condition
      I2C_CTL = (I2C_CTL & ~EZ80_IFLG) | EZ80_STP;
      break;
```

```
   case EZ80_MRXDATAACK:
      i2c_nbytesXferred++;
      // read the data
      data = I2C_DR;
      // check data
      if (data!=i2c_nbytesXferred) {
         i2c_error = 1;
        printf("rx error on byte %d  got %x\n",i2c_nbytesXferred,data);
      }
      // if only 1 more byte, negate ACK
      if (i2c_nbytesXferred==i2c_nbytes2xfer-1) {
         // clear interrupt flag, neagate ACK (don't send ack on last
byte)
         I2C_CTL = I2C_CTL & ~(EZ80_IFLG|EZ80_ACK);
      } else if (i2c_nbytesXferred==i2c_nbytes2xfer) {
         printf("Should have not acked last byte - press NMI PB to
exit\n");
         while(nmi_flag==0);
         // clear interrupt flag
         I2C_CTL &= ~EZ80_IFLG;
      } else {
         // clear interrupt flag
         I2C_CTL &= ~EZ80_IFLG;
      }
      break;
   case EZ80_MRXDATANACK:
      i2c_nbytesXferred++;
      // read the data
      data = I2C_DR;
      // check data
      if (data!=i2c_nbytesXferred) {
         i2c_error = 1;
        printf("rx error on byte %d  got %x\n",i2c_nbytesXferred,data);
      }
      if (i2c_nbytesXferred==i2c_nbytes2xfer) {
      // clear interrupt flag, set stop condition
         I2C_CTL = (I2C_CTL & ~EZ80_IFLG) | EZ80_STP;
      }
      break;
   case EZ80_SWADDRACK:
   case EZ80_SWADDRACKLARB:
      // clear interrupt flag
      I2C_CTL &= ~EZ80_IFLG;
      break;
```

```
   case EZ80_SRXADDRDATAACK:
      i2c_nbytesXferred++;
      // read the data
      data = I2C_DR;
      // check data
      if (data!=i2c_nbytesXferred) {
         i2c_error = 1;
         printf("slave rx error on byte %d  got
%x\n",i2c_nbytesXferred,data);
      }
      // clear interrupt flag
      I2C_CTL &= ~EZ80_IFLG;
      break;
   case EZ80_SRXADDRDATANACK:
      i2c_nbytesXferred++;
      // read the data
      data = I2C_DR;
      // check data
      if (data!=i2c_nbytesXferred) {
         i2c_error = 1;
         printf("slave rx error on byte %d  got
%x\n",i2c_nbytesXferred,data);
      }
      i2c_error = 1;
      printf("slave rx data with nack, should have sent ack - press NMI
PB to exit\n");
      while(nmi_flag==0);
      // clear interrupt flag
      I2C_CTL &= ~EZ80_IFLG;
      break;
   case EZ80_SRXSTOPREPSTART:
      // clear interrupt flag
      if (i2c_nbytes2xfer!=i2c_nbytesXferred) {
         printf("detected stop condition, transferred %d bytes, exp
%d\n",
               i2c_nbytesXferred, i2c_nbytes2xfer);
         i2c_error = 1;
      }
      I2C_CTL &= ~EZ80_IFLG;
      break;
   case EZ80_SRXADDRRDACK:    // 0xa8
   case EZ80_STXDATAACK:      // 0xb8
      // send byte back to master
      i2c_nbytesXferred++;
```

```
      // send data - incrementing pattern starting at 1 to I2C_PKTSZ
      I2C_DR = i2c_nbytesXferred;
      // clear interrupt flag
      I2C_CTL &= ~EZ80_IFLG;
      break;
   case EZ80_STXDATANACK:
   case EZ80_STXLASTDATAACK:
      // clear interrupt flag, clear ACK
      I2C_CTL = (I2C_CTL & ~(EZ80_IFLG|EZ80_ACK));
      break;
      case EZ80_BUSERR:
         printf("EZ80 I2C Buss Error - press NMI PB to exit\n");
         while(nmi_flag==0);
         break;
   default:
      printf("default case: ez80 I2C status=%x - press NMI PB to
exit\n",
              I2C_SR);
      while(nmi_flag==0);
      // clear interrupt flag
      I2C_CTL &= ~EZ80_IFLG;
      break;
   }
}
```

init.c

```
/*****************************************************************
** Author: DP Consulting, Inc.    Dick Pado
** Source: init.c
** Description: Initialize the eZ80 registers
**
** Rev   Date     Description
** 1.0   06/05/03 initial release
*****************************************************************/

#include <stdio.h>
#include <ez80f91.h>
#include "types.h"
#include "ez80RegDefs.h"
#include "tb_vars.h"
#include "bb_cfg.h"
```

```
void init(void) {
   int i;
   // disable interrupts
   IDIS;

   //initialize test variables:
   repeat    = OFF;
   stopOnErr = ON;

   // initialize uart0
   // /RTS bit needs to be high on port D2 for RS232 to work
   // using reset condition for now.
   UART0_LCTL  = DLABit;
   UART0_BRG_H = BAUD57600>>8;
   UART0_BRG_L = BAUD57600;
   UART0_LCTL  = Char8Bits;
   // Enable fifos
   UART0_FCTL  = EnableFifo;
   // set default mode to ascii
   uart0Mode   = ASCII;
   uart0Echo   = 1;

   // initialize the internal RAM control
   RAM_CTL = RAM_EN;

   // initialize the internal RAM address
   RAM_ADDR_U = INTERNAL_RAM_BASE;

   // initialize the CREG
   CREG  = 0x00;
   timeDelay(1);
   CREG  = 0x80;
   timeDelay(1);
   CREG |= 0x82;
   timeDelay(1);
   CREG  = INIT_CREG;

   // setup PortD gpio as uart0
   // all pins controlled by alternate function
   // pd0 = txd1 OUTPUT
   // pd1 = rxd1 INPUT
   // pd2 = rts1 OUTPUT
   // pd3 = cts1 INPUT
```

```
// pd4 = dtr1 OUTPUT
// pd5 = dsr1 INPUT
// pd6 = dcd1 INPUT
// pd7 = ri1  INPUT
PD_ALT2 = INIT_PD_ALT2;
PD_ALT1 = INIT_PD_ALT1;
PD_DDR  = INIT_PD_DDR;
PD_DR   = INIT_PD_DR;

// setup PortC gpio as uart1
// all pins controlled by alternate function
// pc0 = txd1 OUTPUT
// pc1 = rxd1 INPUT
// pc2 = rts1 OUTPUT
// pc3 = cts1 INPUT
// pc4 = dtr1 OUTPUT
// pc5 = dsr1 INPUT
// pc6 = dcd1 INPUT
// pc7 = ri1  INPUT
PC_ALT2 = INIT_PC_ALT2;
PC_ALT1 = INIT_PC_ALT1;
PC_DDR  = INIT_PC_DDR;
PC_DR   = INIT_PC_DR;

// setup PortB gpio as open drain, ddr=out
// enable SPI ports and t4_out
// drive ss pin since not pulled-up on board
PB_ALT2 = INIT_PB_ALT2;
PB_ALT1 = INIT_PB_ALT1;
PB_DDR  = INIT_PB_DDR;
PB_DR   = INIT_PB_DR;

// port a - all outputs
PA_ALT2 = INIT_PA_ALT2;
PA_ALT1 = INIT_PA_ALT1;
PA_DDR  = INIT_PA_DDR;
PA_DR   = INIT_PA_DR;

// assert hardware reset to pcf8584
peripheral_reset();

// ez80 i2c reset - s/w reset
ez80_i2c_reset();
```

```
   // enable interrupts
   init_ints();
}
```

types.h

```
/*************************************************************************
** Author: DP Consulting, Inc.    Dick Pado
** Source: types.h
** Description: different types defined
**
** Rev   Date     Description
** 1.0   06/03/03 initial release
**
*************************************************************************/

# ifndef TYPES_H
# define TYPES_H

typedef  unsigned int   u_int ;
typedef  unsigned char  u_char ;
typedef  unsigned char  byte ;
typedef  unsigned char  bool ;
typedef  unsigned short u_short ;
typedef  unsigned long  u_long ;

# ifndef TRUE
# define TRUE  1
# define FALSE 0
# endif

# endif
```

ez80RegDefs.h

```
/*************************************************************************
**
** Author: DP Consulting, Inc.    Dick Pado
**
** Source: ez80RegDefs.h
```

```
**
** Description: Defines for the eZ80
**
** Rev   Date     Description
**
** 1.0   06/03/03 initial release
**
*****************************************************************/

#ifndef EZ80REG_DEF
#define EZ80REG_DEF

//
// define word accesses to 16 bit EMAC regs
//
// ZDS_COMPILER BUG can't define 16 bit internal i/o regs
//#define ISFRW volatile unsigned short __INTIO *
//#define ISFRW volatile unsigned short __INTIO *

/* COMPILER BUG - get internal error 900
#define EMAC_CTLD       *((volatile unsigned short __INTIO *)0x3C)
#define EMAC_TPTV       *((volatile unsigned short __INTIO *)0x2B)
#define EMAC_MAXF       *((volatile unsigned short __INTIO *)0x30)
#define EMAC_CTLD       *((volatile unsigned short __INTIO *)0x3C)
#define EMAC_TLBP       *((volatile unsigned short __INTIO *)0x42)
#define EMAC_RHBP       *((volatile unsigned short __INTIO *)0x47)
#define EMAC_RRP        *((volatile unsigned short __INTIO *)0x49)
#define EMAC_PRSD       *((volatile unsigned short __INTIO *)0x4E)
#define EMAC_RWP        *((volatile unsigned short __INTIO *)0x51)
#define EMAC_TRP        *((volatile unsigned short __INTIO *)0x53)
#define EMAC_BLKSLFT    *((volatile unsigned short __INTIO *)0x55)
#define EMAC_FDATA      *((volatile unsigned short __INTIO *)0x57)
*/

//*************************************************************************
// WDT
//*************************************************************************
// wdt control reg bit defs
#define WDT_EN          0x80
#define WDT_DIS         0x00
#define WDT_NMIOUT      0x40
#define WDT_RSTOUT      0x00
#define WDT_RSTFLAG     0x20
```

```
// bit 4
#define WDT_NMIFLAG     0x10
// bits 3:2
#define WDT_SYSCLK      0x00
#define WDT_INTRCOSC    0x08
#define WDT_RTCCLK      0x04

// bits 1:0
#define WDT_TO27        0x00
#define WDT_TO25        0x01
#define WDT_TO22        0x02
#define WDT_TO18        0x03

//*************************************************************************
// Internal RAM
//*************************************************************************
#define RAM_EN          0x80
#define RAM_DIS         0x00
#define ERAM_EN         0x40     // emac ram

//*************************************************************************
// IrDA
//*************************************************************************
#define IRDA_LOOPBACK   4
#define IRDA_RXEN       2
#define IRDA_EN         1
#define IRDA_TXEN       0
#define IRDA_DIS        0

//*************************************************************************
// RTC
//*************************************************************************
#define RTC_ALARM          0x80
#define RTC_IEN            0x40
#define RTC_IDIS           0x00
#define RTC_BCD            0x20
#define RTC_PWRLINE        0x10
#define RTC_XTAL           0x00
#define RTC_PWR60          0x00
#define RTC_PWR50          0x08
#define RTC_SLP_WAKE       0x02
#define RTC_UNLOCK         0x01
#define RTC_LOCK           0x00
```

```
// RtcACtrl bit defs
#define RTC_ADOW_EN        0x08
#define RTC_AHRS_EN        0x04
#define RTC_AMIN_EN        0x02
#define RTC_ASEC_EN        0x01

//*************************************************************************
// Internal FLASH
//*************************************************************************
#define FLASH_KEY_CODE     (u_short) 0x49b6          // Flash key codes
// flash control reg
#define FLASH_WAIT_STATES(n)  (n<<5)
#define FLASH_EN           0x08
// flash interrupt control reg
#define FLASH_DONE_IEN     0x80
#define FLASH_ERR_IEN      0x40
#define FLASH_DONE         0x20
#define FLASH_WR_VIO       0x08
#define FLASH_RP_TMO       0x04
#define FLASH_PG_VIO       0x02
#define FLASH_MASS_VIO     0x01
#define FLASH_NO_ERRS      0x00
// flash page reg
#define FLASH_MAIN_MEM     0x00
#define FLASH_INFO_MEM     0x80
// flash progrm control reg
#define FLASH_ROW_PGM      0x04
#define FLASH_PG_ERASE     0x02
#define FLASH_MASS_ERASE   0x01

#define FLASH_ERASE_VALUE  0xff

//*************************************************************************
// GPIO
//*************************************************************************

#define GPIOALT2_TOTEMPOLE_OUT      0x00
#define GPIOALT1_TOTEMPOLE_OUT      0x00
#define GPIODDR_TOTEMPOLE_OUT       0x00
#define GPIODATA_TOTEMPOLE_OUT      0xFF
//For port pins as inputs                (MODE 2)
#define GPIOALT2_INPUT_ONLY         0x00
#define GPIOALT1_INPUT_ONLY         0x00
#define GPIODDR_INPUT_ONLY          0xFF
```

```
#define GPIODATA_INPUT_ONLY         0xFF
//For O/D i/o                           (MODE 3)
#define GPIOALT2_OPENDRAIN_INOUT    0x00
#define GPIOALT1_OPENDRAIN_INOUT    0xFF
#define GPIODDR_OPENDRAIN_INOUT     0x00
#define GPIODATA_OPENDRAIN_INOUT    0xFF
//For open source I/O                   (MODE 4)
#define GPIOALT2_OPENSOURCE_INOUT   0x00
#define GPIOALT1_OPENSOURCE_INOUT   0xFF
#define GPIODDR_OPENSOURCE_INOUT    0xFF
#define GPIODATA_OPENSOURCE_INOUT   0xFF
//For Dual Edge Interrupt               (MODES 5/6)
#define GPIOALT2_DUALEDGEINTR       0xFF
#define GPIOALT1_DUALEDGEINTR       0x00
#define GPIODDR_DUALEDGEINTR        0x00
#define GPIODATA_DUALEDGEINTR       0xFF
//For Alternate Function Select         (MODE 7)
#define GPIOALT2_ALTERNATE_FUNC     0xFF
#define GPIOALT1_ALTERNATE_FUNC     0x00
#define GPIODDR_ALTERNATE_FUNC      0xFF
#define GPIODATA_ALTERNATE_FUNC     0xFF
//For interrupt active low/high level   (MODE 8)
#define GPIOALT2_INTRLEVEL          0xFF
#define GPIOALT1_INTRLEVEL          0xFF
#define GPIODDR_INTRLEVEL           0x00
#define GPIODATA_INTRLEVEL          0xFF
//For interrupt active low/high edge    (MODE 9)
#define GPIOALT2_INTREDGE           0xFF
#define GPIOALT1_INTREDGE           0xFF
#define GPIODDR_INTREDGE            0xFF
#define GPIODATA_INTREDGE           0xFF
//-----------------------------------------------
//Keep track of reserved modes, if any
#define GPIOMODE_RESERVED       -1
//Modes as per the H/W specification numbering
#define GPIOMODE_1                  1
#define GPIOMODE_2                  2
#define GPIOMODE_3                  3
#define GPIOMODE_4                  4
#define GPIOMODE_5          GPIOMODE_RESERVED       //5 is a reserved
mode
#define GPIOMODE_6                  6
#define GPIOMODE_7                  7
#define GPIOMODE_8                  8
```

```
#define GPIOMODE_9                  9

#define GPIO_MIN_MODE         GPIOMODE_1
#define GPIO_MAX_MODE         GPIOMODE_9
//-------------------------------------------------
//Also define descriptive mode names for ease of reading code
//Mode 5 is reserved and skipped here
#define GPIOMODE_TOTEMPOLE_OUTPUT       GPIOMODE_1
#define GPIOMODE_INPUT_ONLY             GPIOMODE_2
#define GPIOMODE_OPENDRAIN_INOUT        GPIOMODE_3
#define GPIOMODE_OPENSOURCE_INOUT       GPIOMODE_4
#define GPIOMODE_DUALEDGE_INTERRUPT     GPIOMODE_6
#define GPIOMODE_ALTERNATE_FUNCTION     GPIOMODE_7
#define GPIOMODE_LEVEL_INTERRUPT        GPIOMODE_8
#define GPIOMODE_EDGE_INTERRUPT         GPIOMODE_9
//-------------------------------------------------
#define GPIOERROR_ILLEGALPORT       -1
#define GPIOERROR_ILLEGALMODE       -2
//-------------------------------------------------
#define BIT0                        0x01
#define BIT1                        0x02
#define BIT2                        0x04
#define BIT3                        0x08
#define BIT4                        0x10
#define BIT5                        0x20
#define BIT6                        0x40
#define BIT7                        0x80

//*****************************************************************
//** Uarts
//*****************************************************************
//Bit definitions for Line status register “ULSRn”
#define RxDataRdy                   0x01
#define RxOverrun                   0x02
#define ParityError                 0x04
#define MD_ADDR_MARK                0x04
#define FramingError                0x08
#define BreakIndic                  0x10
#define TxRdy                       0x20
#define TxEmpty                     0x40
#define ErrInFIFO                   0x80
//Bit definitions for Modem Control Register
#define MultiDropMode               0x20     // f92,f93,f1 only
#define MCR_Loop                    0x10
```

```
#define MCR_Out2                    0x08
#define MCR_Out1                    0x04
#define ReqToSend                   0x02
#define DataTermRdy                 0x01
//Bit definitions for Modem status register “UMSRn”
#define CTSChanged                  0x01
#define DSRChanged                  0x02
#define RIDetected                  0x04
#define DCDChanged                  0x08
#define ClearToSnd                  0x10
#define RTSLoopBack                 0x10
#define DataSetRdy                  0x20
#define DTRLoopBack                 0x20
#define Out1LoopBack                0x40
#define Out2LoopBack                0x80
//Bit definitions for Interrupt Enable Register “UIERn”
#define RxRdyIntEnbl                0x01
#define TxEmptyIntEnbl              0x02
#define LineStatusIntEnbl           0x04
#define ModemStatusIntEnbl          0x08
//Bit definitions for LineControl Reg “ULCRn”
#define DLABit                      0x80
#define BreakBit                    0x40
#define XmitParityErr               0x20
#define EvenParity                  0x10
#define ParityEnable                0x08
// DPC 10/20/02
#define OddParity                   0x00
#define TwoStopBits                 0x04
// Define some character framing ULCR field constants for convenience
//Two bit field D1..D0 in ULCR for Char length
#define Char5Bits                   0x00
#define Char6Bits                   0x01
#define Char7Bits                   0x02
#define Char8Bits                   0x03

//Bit definitions for FIFO Control Register “UFCRn”
#define EnableFifo                  0x01
#define ClearRxFifo                 0x02
#define ClearTxFifo                 0x04
#define FifoTrigLev1                0x00
#define FifoTrigLev4                0x40
#define FifoTrigLev8                0x80
#define FifoTrigLev14               0xC0
```

```
//-------------------------------------------------------------------
----------
//Error codes
#ifndef UART_ERRORBASEINDEX
#define UART_ERRORBASEINDEX         0
//#pragma message UART_ERRORBASEINDEX not defined - defaulting to 0

//*******************************************************************
//** SPI
//*******************************************************************
//Bit definitions for SPI status register
#define SpiMODF                     0x10
#define SpiWCOL                     0x40
#define SpiSPIF                     0x80
//Bit definitions for SPI Control Reg
#define SpiIRQ_EN                   0x80
#define SpiSPI_EN                   0x20
#define SpiMASTER_EN                0x10
#define SpiCPOL                     0x08
#define SpiCPHA                     0x04
#endif

#define UARTERROR_ILLEGALBITRATE    UART_ERRORBASEINDEX - 1
#define UARTERROR_ILLEGALCHANNEL    UART_ERRORBASEINDEX - 2

//*******************************************************************
// Timers F91
//*******************************************************************
// TmrCtl
#define TmrBrk       0x80
#define TmrSelEcr    0x60 //new bits ECx tmr1 or tmr2 Rising edge src
#define TmrSelEcf    0x40 //new bits ECx tmr1 or tmr2 Falling edge src
#define TmrSelRtc    0x20 //new bits RTC src
#define TmrSelSys    0x00 //new bits System Clk src
#define TmrClkDiv256 0x18
#define TmrClkDiv64  0x10
#define TmrClkDiv16  0x08
#define TmrClkDiv4   0x00
#define TmrCont      0x04
#define Tmr1Shot     0x00
#define TmrLoad      0x02
#define TmrEn        0x01
#define TmrDis       0x00
```

```
// TMR_IIR Timer Identification Register
#define TmrTO        0x01
#define TMR0_TO      (TMR0_IIR & TmrTO)

// TMR_IER Timer Interrupt enable register
#define TmrIEN       0x01

//*****************************************************************************
// RTC
//*****************************************************************************
#define RTC_ALARM_INTEN 0x40
#define RTC_SLP_WAKE    0x02
#define RTC_ASEC_EN     0x01

//*****************************************************************************
// PLL
//*****************************************************************************
//
//PLL_CTL0
//
#define CPC100UA        0x00
#define CPC500UA        0x40
#define CPC1MA          0x80
#define CPC1P5MA        0xc0
#define LC8CYC20NS      0x00
#define LC16CYC20NS     0x04
#define LC8CYC400NS     0x08
#define LC16CYC400NS    0x0c
#define CLKISEXTOSC     0x00
#define CLKISPLL        0x01
#define CLKISRTC        0x02
//
// PLL_CTL1
//
#define PLLISLOCKED     0x20
#define PLLLOCK         0x10
#define PLLUNLOCK       0x08
#define PLLLOCKIE       0x04
#define PLLUNLOCKIE     0x02
#define PLLEN           0x01


#endif
```

pcf8584.h

```
/*********************************************************************
**
** Author: DP Consulting, Inc.    Dick Pado
** Source: pcf8584.h
** Description: Support for PCF8584 I2C Controller
**
** Rev   Date     Description
** 1.0   06/05/03 initial release
**
*********************************************************************/

#define S1_PIN    0x80  // pending interrupt not wr pin=1 clears status
bits
#define S1_ES0     0x40  // 0=serial i/f off, 1=serial i/f on
#define S1_ES1     0x20  // reg addressing
#define S1_ES2     0x10  // reg addressing
#define S1_ENI     0x08  // 1=enable interrupts
#define S1_STA     0x04  // start/repeated start/stop control
#define S1_STO     0x02  // start/repeated start/stop control
#define S1_ACK     0x01  // acknowledge 1=send ack automatically
                         //             0=last master receive byte
#define S1_STS     0x20  // set in slv/rec mode when stop detected
#define S1_BER     0x10  // set when MisPlaced start or stop detected
#define S1_AD0_LRB 0x08  // if AAS=0, Last Received Bit
                         // if AAS=1, AD0=1 if general call address
                         // if AAS=1, AD0=0 if own slave address
#define S1_AAS     0x04  // set=1 if addr=own_addr or addr=general call
#define S1_LAB     0x02  // Lost Arbitration Bit
#define S1_BBN     0x01  // Bus Busy active low 0=bus busy

// internal register addressing
// S1_ES0=0
#define I2C_RESET_STAT  S1_PIN
#define I2C_IOWN_ADDR   (I2C_RESET_STAT | 0x00)
#define I2C_IIVEC       (I2C_RESET_STAT | S1_ES2)
#define I2C_ICLKREG     (I2C_RESET_STAT | S1_ES1)     // S1_ES0=1
#define I2C_CTLSTAT     (I2C_RESET_STAT | S1_ES0 | S1_ACK)
#define I2C_CTLSRX      (I2C_RESET_STAT | S1_ES0)
#define I2C_RSTAT       S1_ES0
#define I2C_RDREG       S1_ES0
#define I2C_RIVEC       (S1_ES0 | S1_ES2)
```

```
// status reg - s1 RO (a0=1)
// clock reg bit definition
#define I2C_CLKFREQ_3   0x00        // 3    MHz
#define I2C_CLKFREQ_4   0x10        // 4.43 MHz
#define I2C_CLKFREQ_6   0x14        // 6    MHz
#define I2C_CLKFREQ_8   0x18        // 8    MHz
#define I2C_CLKFREQ_12  0x1c        // 12   MHz

#define I2C_CLKSEL_90   0x00        // 90   KHz
#define I2C_CLKSEL_45   0x01        // 45   KHz
#define I2C_CLKSEL_11   0x02        // 11   KHz
#define I2C_CLKSEL_1p5  0x03        // 1.5  KHz
```

ez80_i2c.h

```
/*************************************************************************
**
** Author: DP Consulting, Inc.    Dick Pado
** Source: ez80_i2c.h
** Description: SUpport fot the eZ80 I2C
**
** Rev   Date     Description
** 1.0   11/15/02 initial release
**
*************************************************************************/

#ifndef EZ80_I2C
#define EZ80_I2C

// control reg
#define EZ80_IEN     0x80
#define EZ80_ENAB    0x40
#define EZ80_STA     0x20
#define EZ80_STP     0x10
#define EZ80_IFLG    0x08
#define EZ80_ACK     0x04

// slave address reg
#define EZ80_CALLADDR_ENA  0x01
#define EZ80_CALLADDR_DIS  0x00
```

```
// status reg
#define EZ80_BUSERR           0x00
#define EZ80_STARTCOND        0x08
#define EZ80_REPSTART         0x10
#define EZ80_MWADDRACK        0x18
#define EZ80_MWADDRNACK       0x20
#define EZ80_MTXDATAACK       0x28
#define EZ80_MTXDATANACK      0x30
#define EZ80_ARBLOST          0x38
#define EZ80_MRADDRACK        0x40
#define EZ80_MRADDRNACK       0x48
#define EZ80_MRXDATAACK       0x50
#define EZ80_MRXDATANACK      0x58
#define EZ80_SWADDRACK        0x60
#define EZ80_SWADDRACKLARB    0x68
#define EZ80_SGCADDRACK       0x70
#define EZ80_SGCADDRACKLARB   0x78
#define EZ80_SRXADDRDATAACK   0x80
#define EZ80_SRXADDRDATANACK  0x88
#define EZ80_SRXGCADDRDATAACK 0x90
#define EZ80_SRXGCADDRDATANACK   0x98
#define EZ80_SRXSTOPREPSTART  0xa0
#define EZ80_SRXADDRRDACK     0xa8
#define EZ80_SRXADDRRDLARB    0xb0
#define EZ80_STXDATAACK       0xb8
#define EZ80_STXDATANACK      0xc0
#define EZ80_STXLASTDATAACK   0xc8
#define EZ80_MWADDR2ACK       0xd0
#define EZ80_MWADDR2NACK      0xd8
#define EZ80_NOSTAT           0xf8

// clock register
#ifdef CLK20
#define EZ80_CLKDATA    ((5<<3) | 2)   // fsamp=5.0 MHz, fscl=250 KHz
#endif
#ifdef CLK50
#define EZ80_CLKDATA    ((2<<3) | 3)   // fsamp=6.25  MHz, fscl=208.33
KHz
#endif

#define I2C_WR    0
#define I2C_RD    1

#define I2C_TO_DELAY    50
```

```
#endif
```

tb_vars.h

```
/*********************************************************************
**
** Author: DP Consulting, Inc.    Dick Pado
** Source: tb_vars.h
** Description: Prototypes, defines, and variables for the eZ80F91
** Module
**
** Rev   Date     Description
** 1.0   06/03/03 initial release
**
*********************************************************************/

#include “types.h”
#include “bb_cfg.h”

#ifndef TB_VARS
#define TB_VARS

#define BUFLEN    64
#define NUMPKTS   6
#define OFF       0
#define ON        1

// function prototypes
extern   void          putch(int);
extern   int           getch(void);
extern   int           kbhit(void);
extern   void          kb_flush(void);

// define uart baud rate, spi clk freq,
//        timer period for pclk (peripheral clock)
#ifdef CLK50
   #define SYSCLKFREQ         50000000L
   #define FLASH_WSTATES      1
#else
   #ifdef CLK48
      #define SYSCLKFREQ      48000000L
```

```
      #define FLASH_WSTATES   1
   #else
      #ifdef CLK40
         #define SYSCLKFREQ      40000000L
         #define FLASH_WSTATES   1
      #else
         #ifdef CLK20
            #define SYSCLKFREQ      20000000L
            #define FLASH_WSTATES   1
         #else
            #define SYSCLKFREQ      10000000L
            #define FLASH_WSTATES   1
         #endif
      #endif
   #endif
#endif

#define BAUD2400     ((SYSCLKFREQ + 8L *   2400L) / 16L /   2400L)
#define BAUD9600     ((SYSCLKFREQ + 8L *   9600L) / 16L /   9600L)
#define BAUD19200    ((SYSCLKFREQ + 8L *  19200L) / 16L /  19200L)
#define BAUD57600    ((SYSCLKFREQ + 8L *  57600L) / 16L /  57600L)
#define BAUD115200   ((SYSCLKFREQ + 8L * 115200L) / 16L / 115200L)
// CLK*1/10 sec / 256 clkdiv
#define TMR3_100MS   (u_short)((SYSCLKFREQ + 1280L) / 256L / 10L)
#define TO20MS       (u_short)((SYSCLKFREQ / 1000L  *  20L + 8L) / 16L)

#define TMR4_RELOAD_CNT 1

// define serial tranfer modes
#define ASCII     1
#define BINARY    0

#ifdef MAIN
         int                 errno;

         u_char              i2c_mode;
         u_char              i2c_slv_addr;
         u_char              i2c_nbytes2xfer;
         u_char              i2c_nbytesXferred;
         u_char              i2c_error;
         u_short             tmr1_cnt;
         int                 uart0Echo;          // 1=echo characters
back
         u_char              uart0Mode;
```

```
        char                inbuf[BUFLEN];
        int                 repeat;
        int                 stopOnErr;
        int                 nmi_flag;

#else
extern  u_char              i2c_mode;
extern  u_char              i2c_slv_addr;
extern  u_char              i2c_nbytes2xfer;
extern  u_char              i2c_nbytesXferred;
extern  u_char              i2c_error;
extern  u_short             tmr1_cnt;
extern  int                uart0Echo;       // 1=echo characters back
extern  u_char              uart0Mode;
extern  char                inbuf[];
extern  int                 repeat;
extern  int                 stopOnErr;
extern  int                 nmi_flag;

#endif

extern  void    init(void);
extern  void    startTimer(u_char, u_char);

extern  void    ez80_i2c_reset(void);
extern  void    ez80_i2c_isr(void);
extern  void    ez80_i2c_init(u_char);
extern  void    peripheral_reset(void);
extern  void    pcf8584_i2c_init(u_char);
extern  u_short ez80_mas_tx(void);
extern  u_short ez80_mas_rx(void);
extern  u_short pcf8584_mas(int);
extern  void    timeDelay(int);
extern  void    tmr3_isr(void);
extern  void    init_ints(void);
extern  void    wr_i2c_creg(u_char, u_char);
extern  u_char  rd_i2c_dreg(u_char);

#define IEN     asm("EI")
#define IDIS    asm("DI")

#endif
```

bb_cfg.h

```
/*********************************************************************
**
** Author: DP Consulting, Inc.
** Source: bb_cfg.h
** Description: Defines for the F91_bb Memory Map, GPIO mapping
**
** Rev   Date     Description
** 1.0   06/05/03 initial release
**
*********************************************************************/

#ifndef F91_BB_CFG
#define F91_BB_CFG

// define chip selects
//
#define INTERNAL_RAM_BASE     0xa0     // int  8k ram 0xa0e000-0xa0ffff
                                       // emac 8k ram 0xa0c000-0xa0dfff
#define IRAM_OFFSET          0xe000
//
// cs2 is decoded into 8 chip selects of 8k space each thru a 138
//    cs2+0x0000 - cs2+1fff   => scc
//    cs2+0x2000 - cs2+3fff   => i2c
//    cs2+0x4000 - cs2+5fff   => cfmem
//    cs2+0x6000 - cs2+7fff   => cfio1
//    cs2+0x8000 - cs2+9fff   => cfio2
//    cs2+0xa000 - cs2+bfff   => exp_cs
//    cs2+0xc000 - cs2+dfff   => creg
//    cs2+0xe000 - cs2+ffff   => sreg
//
#define IOCS2_TEST   (*((volatile u_char*)0x802000))
#define IOCS         0x800000
// control and status regs on f91_bb
#define CREG         (*((volatile u_char*)(IOCS + 0xC000)))
#define SREG         (*((volatile u_char*)(IOCS + 0xE000)))
#define INIT_CREG    0xef
// creg bit definitions
//
//   bit      description
//    7        cf_reset
//    6        spi slvseln
```

```
//    5        rs232 ring indicator
//    4        phy isolate (when using phy internal to fpga)
//    3        irda shutdown
//    2        s/w ide
//    1        cf_csel pin
//    0        s/w rstn
//

// GPIO definitions
// pcf8584 i2c
#define I2C_SET_RESET         CREG  &= ~0x01
#define I2C_CLR_RESET         CREG  |=  0x01

// Timeout values for 20MS time out loops
#define TOHALFSEC    25
#define TO1SEC       50
#define TO2SEC       100
#define TO4SEC       200
#define TO10SEC      500
#define TO60SEC      3000

/*********************************************************************
// memory map f91_bb board
// cs2 - peripheral chip select
/*********************************************************************

// PCF8584
#define I2CA0        (IOCS + 0x2000)
#define I2CA1        (IOCS + 0x2001)
#define I2C_CREG     (*((volatile u_char*) I2CA1))
#define I2C_DREG     (*((volatile u_char*) I2CA0))
#define I2C_SREG     I2C_CREG

#define PCF8584_SLVADDR 0x55
#define EZ80_SLVADDR    0x15

// define F91_BB GPIO's
//
// portD - all bits    used for uart0 rs232 port     a = alternate
function
//       - bits[1:0]   used for IrDA
//       - bits[4,2:0] used for RS485
//
// portC - all bits    used for modem                a
```

```
//       - bits[4,2:0] used for RS485
//
// portB - bit7        used for SPI mosi            a
//       - bit6        used for SPI miso            a
//       - bit5        used for I2C IREQn           i = input
//       - bit4        Spare                        i
//       - bit3        used for SPI sck             a
//       - bit2        used for SPI SSn             o
//       - bit1        used for SCC IREQn           i
//       - bit0        used for CF  IREQ            i
//
// portA - all outputs
//
#define INIT_PA_ALT2    0x00
#define INIT_PA_ALT1    0x00
#define INIT_PA_DDR     0x00
#define INIT_PA_DR      0x00
#define INIT_PB_ALT2    0xc8
#define INIT_PB_ALT1    0x00
#define INIT_PB_DDR     0xfb
#define INIT_PB_DR      0x04
#define INIT_PC_ALT2    0xff
#define INIT_PC_ALT1    0x00
#define INIT_PC_DDR     0xff
#define INIT_PC_DR      0x00
#define INIT_PD_ALT2    0xff
#define INIT_PD_ALT1    0x00
#define INIT_PD_DDR     0xff
#define INIT_PD_DR      0x00

#endif
```

Notes