HOLT INTEGRATED CIRCUITS INC.

AN-145

December 2017

ARINC 429 / SPI Demonstration board
HI-3585, HI-3598/HI-3599 Applications Note

INTRODUCTION

The Holt HI-3598 and HI-3599 are silicon gate CMOS ICs for interfacing eight ARINC 429 receive buses to a high-speed Serial Peripheral Interface (SPI) enabled microcontroller.

The Holt HI-3585 is a single-chip ARINC 429 Serial Peripheral Interface (SPI) IC incorporating a single ARINC 429 receive channel and transmitter.

These devices interface to a host CPU via the 4-wire SPI. Op codes are used to control the flow of information between the host and the HI-3585, HI-3598 or HI-3599.

An analog line driver and receiver are included on the HI-3585 to allow the user to connect directly to an ARINC bus. Automatic label recognition, transmit and receiver protocol options and 32-word data buffering FIFOs are included in the HI-3585.

Automatic label recognition, a 4-word data buffer and on-chip analog line receivers are available on each HI-3598 and Hi-3599.

Complete data sheets describing each device can be found at the Holt web-site www.holtic.com.

This application note describes a simple interface design between the HI-3585, the HI-3598 or HI-3599 and a Freescale MC9S12XD64 microcontroller. The C code routines are shown as examples of how a user may program the Holt devices by setting up control registers, transmitting internal/external self test data, polling the status register, and retrieving received ARINC 429 data from each of the enabled receivers.

HARDWARE DESIGN

An example circuit for the HI-3585 and HI-3598 configuration is shown in Figure 1.

The HI-3585 and HI-3598 are connected to the MC9S12XD64 as shown in the following two tables.

MC9S12XD64	HI-3585
$\overline{SS0}$	$\overline{CS}$
SCK0	SCK
MOSI0	SI
MISO0	SO
PA0	TFLAG
PA1	RFLAG
PA2	RESET

MC9S12XD64	HI-3598
$\overline{SS1}$	$\overline{CS}$
SCK1	SCK
MOSI1	SI
MISO1	SO
PAD00	FLAG
PAD01	FLAG1
PA4	FLAG2
PA5	FLAG3
PA6	FLAG4
PA7	FLAG5
PB0	FLAG6
PB1	FLAG7
PB2	FLAG8
PB3	RESET

SOFTWARE DESIGN

All C code was written and compiled with Special Edition CodeWarrior for S12(X) V5.0 software that can be found at www.Freescale.com. For more information about the microcontroller MC9S12XD64 and Code Warrior software visit www.Freescale.com

The software was downloaded to the MC9S12XD64 using a P&E Microcomputer Systems USB BDM Multilink cable which can be found at www.PEMicro.com (USB-ML-12 cable required only).

The SPI demo board was configured to use the SPI ports with the SPI data clocked on the rising edge of SCK and the data changing on the falling edge of SCK. See the Freescale microcontroller data sheet for a full description of the SPI control registers.

The MC9S12XD64 SPI control register 1, SPICR1, is set as follows for the design examples in this application note:

SPIE (bit 7): SPI interrupts are disabled
SPE (bit 6): SPI enabled, port pins are dedicated to SPI functions
SPTIE (bit 5): SPI Transmitter empty flag disabled
MSTR (bit 4): SPI is in master mode
CPOL (bit 3): Active-high clocks selected. In idle state SCK is low
CPHA (bit 2): Sampling of odd data occurs at odd edges (1, 3, 5 etc.) of the SCK clock
SSOE (bit 1): Slave select output is enable
LSBFE(bit 0): Data is transferred most significant bit first

(For a full description of SPICR1, please see SPIDemoR1.2.c, attached and MC9S12XD64.h, V2.24, which is available with Code Warrior S12(X) V5.0.)

The following short subroutine was written for the examples in order to simplify instructions. This routine enables SPI data transfers between the MC9S12XD64 and either the HI-3585 or HI-3598.

txrx8bits_5(x,x): transfers 8 bit read/write data for HI-3585
txrx8bits_*(x,x): transfers 8 bit read/write data for HI-3598

A more detailed explanation of this subroutine may be found in the attached example program, SPIDemoR1.c.

An expanded header file, SPIDemoR1.h, written to accompany the C program, is also attached. The header contains definitions for status flags, SPI pins, MR pins and LEDs.

The source code for both the C file (SPIDemoR1.c) and header file (SPIDemoR1.h) listed in this applications note are also available in electronic format by contacting the Holt Sales Department at sales@holtic.com.

BASIC OPERATION

Control Word (HI-3598 example)

The control word is a 16-bit register used to configure the device. The control word register bits, CR15-CR0, are loaded from SPI with opcode 0xN4 where 'N' specifies the channel number in hex for HI-3598 and 0x10 for HI-3585.

The following code loads the control word in to the HI-3598.

```
// clear SPI status register
dummy = SPI1SR;
// writing opcode to Data Reg starts SPI xfer
SPI1DR = (j << 4) + 0x04;
while (!SPI1SR_SPIF) {;}
// read Rx data in Data Reg to reset SPIF
dummy = SPI1DR;

// write upper Control Register
SPI1DR = (char)(ControlReg_8 >> 8);
while (!SPI1SR_SPIF) {;}
dummy = SPI1DR;

// write lower Control Register
SPI1DR = (char)(ControlReg_8 & 0xFF);
while (!SPI1SR_SPIF) {;}
dummy = SPI1DR;
```

Load Transmitter FIFO (HI-3585 example)

The transmitter FIFO is loaded with op code 0x0E for the HI-3585. The most significant bit of data must follow the last op code bit.

The following code loads one word in the transmitter FIFO:

```
// send op code (ignore returned data byte)
dummy = txrx8bits_5(0x0E,1);
// send MS byte (ignore returned data byte)
dummy = txrx8bits_5((char)(TxBusWord_5[i]>>24),1);
// send next byte (ignore returned data byte)
dummy =
txrx8bits_5((char)((TxBusWord_5[i]>>16)&0xFF),1);
// send next byte (ignore returned data byte)
dummy =txrx8bits_5((char)((TxBusWord_5[i]>>8)&
0xFF),1);
// send LS byte (ignore returned data byte)
dummy = txrx8bits_5((char)(TxBusWord_5[i] &
0xFF),1);
```

Enable Transmit (HI-3585 example)

If control word bit 14 = 0, TFLAG will be low when the TX FIFO contains at least one word. There are two ways to transmit data with the HI-3585. If control word bit 13 = 1, then transmission begins as soon as data is available in the TX FIFO. If control word bit 13 = 0, op code 0x12 must be written to HI-3585 in order to enable transmission. In SPIDemoR1.c, data transmission is immediate because CR13 = 1.

The following code demonstrates how the single op code is necessary to initiate transmission.

```
// enable ARINC bus transmit, return only after
// completion
txOpCode_5 (0x12,1);
```

Unload RX FIFO (HI-3585 and HI-3598 example)

For HI-3585, if control word bit 15 = 0, RFLAG will be low when the RX FIFO contains at least one valid ARINC word. Op code 0x08 will retrieve one word at a time.

The following code uses op code 0x08 to retrieve one word and to transfer that word, 8 bits at a time, into the variable rxdata.

```
// send op code (ignore returned data byte)
rxdata = txrx8bits_5(0x08,1);

// send dummy data
// receive and left-justify most signif. byte

j = txrx8bits_5(0x00,1);
rxdata = (j << 24);

// send dummy data
// receive, left-shift then OR next byte
j = txrx8bits_5(0x00,1);
rxdata = rxdata | (j << 16);

// send dummy data
// receive, left-shift then OR next byte
j = txrx8bits_5(0x00,1);
rxdata = rxdata | (j << 8);

// send dummy data
// receive and OR the least signif. byte
j = txrx8bits_5(0x00,1);
rxdata = rxdata | j ;
```

For HI-3598, if control word bit 1 = 0, FLAG will be high when the RX FIFO contains at least one valid ARINC word. Op code 0xN3 will retrieve one word at a time - where 'N' is the channel number.

The following code uses op code 0xN3 to retrieve one word and to transfer that word, 8 bits at a time, into the variable rxdata.

```
// send op code (ignore returned data byte)
rxdata = txrx8bits_8((k << 4) + 0x03,1);

// send dummy data
// receive and left-justify most signif. byte
j = txrx8bits_8(0x00,1);
rxdata = (j << 24);

// send dummy data
// receive, left-shift then OR next byte
j = txrx8bits_8(0x00,1);
rxdata = rxdata | (j << 16);

// send dummy data
// receive, left-shift then OR next byte
j = txrx8bits_8(0x00,1);
rxdata = rxdata | (j << 8);

// send dummy data
// receive and OR the least signif. byte
j = txrx8bits_8(0x00,1);
rxdata = rxdata | j ;
```

LABELS

The HI-3585 has user-programmable label recognition for any combination of the 256 possible labels. There are five options to program labels. Labels can all be set (op code (0x03), reset (op code 0x02), all 256 may be programed in any combination (op code 0x06) or individual labels can be set or reset at a time (op code 0x04 to reset, op code 0x05 to set). To verify the 256 labels are set/reset op code 0x0D is used to read back all 256 labels. The labels will not be reset after having been read.

The following code examples show how each op code is used and how program arrays can keep track of the label memory.

Reset all HI-3585 ARINC label selections

```
// reset all label bits in HI-358x device
txOpCode_5(02,1);
// reset all bits all 32-bytes of global LabelArray[]
for (i=0; i<32; i++) {
  LabelArray_5[i] = 0;}
```

Set all HI-3585 ARINC label selections

```
// set all label bits in HI-358x device
txOpCode_5(03,1);
// set all bits all 32-bytes of global LabelArray[]
for (i=0; i<32; i++) {
  LabelArray_5[i] = 0xFF;}
```

Copy HI-3585 ARINC label selections from LabelArray[ ] to HI-3585 ie. Write 256 Label memory to HI-3585

```
// send op code (ignore returned data byte)
dummy = txrx8bits_5(0x06,1);

// send 32 bytes of ARINC label data
for (i=31; i>=0; I--) {
  // send 1 byte of label data, ignore returned data
  //byte
dummy = txrx8bits_5(LabelArray_5[i],1);
}
```

Modify one label, either set or reset

```
// send op code (ignore returned data byte)
if (label_value == 1) i = txrx8bits_5(0x05,1);
// "set" op code
else i = txrx8bits_5(0x04,1);
// "reset" op code
// send 1 byte label address, ignore returned data
byte
i = txrx8bits_5(label_num,1);
```

Verify match: HI-3585 ARINC label selections to LabelArray[ ] ie. Read 256 Label memory from HI-3585

```
// send op code (ignore returned data byte)
rxbyte = txrx8bits_5(0x0D,1);

j = 0xffff;
// starting at high end, read 8-bit increments of
// ARINC label data
for (i=31; i>=0; i--) {
// send dummy data, read 1 byte of label data
rxbyte = txrx8bits_5(0,1);
// check for mismatch
if (rxbyte != LabelArray_5[i]) {
```

The HI-3598 has user-programmable label recognition for up to 16 different labels per channel. Op code oxN1, where 'N' is the channel number, is used to write 16 labels to a specific channel. Op code 0xN2, where 'N' is the channel number, is used to read the 16 programmed labels. The labels will not be reset after having been read.

The following code examples show how each op code is used and how program arrays can keep track of the label memory.

Load label values from LabelArrayCh[ ][ ] to HI-3598 label memory

```
// send op code (ignore returned data byte)
dummy = txrx8bits_8((j<< 4) + 0x01,1);

// send 16 bytes of ARINC label data
for (i=15; i>=0; I--) {
  // send 1 byte of label data,
  dummy = txrx8bits_8(LabelArrayCh_8[j-1][i],1);
}
```

Read the contents of HI-3598 label memory and verify matches to LabelArrayCh[ ][ ]

```
// send op code (ignore returned data byte)
rxbyte = txrx8bits_8((k << 4) + 0x02,1);

j = 0xffff;
// starting at high end, read 8-bit increments of
// ARINC label data
for (i=15; i>=0; i--) {
 // send dummy data, read 1 byte of label data
 rxbyte = txrx8bits_8(0,1);
 // check for mismatch
 if (rxbyte != LabelArrayCh_8[k-1][i]) j=0x0000;}
```

STATUS REGISTER

The HI-3585 has an 8-bit status register which can be polled to determine the status of the receiver FIFO and transmitter FIFO. Op code 0x0A is used to retrieve the status register bits.

The following code is an example of how to use op code 0x0A and store the contents in a variable to be used throughout the program.

```
// send op code (ignore returned data byte)
rxdata = txrx8bits_5(0x0A,1);
// send dummy data / receive Status Reg byte
rxdata = txrx8bits_5(0x00,1);
```

EXAMPLE PROGRAMS

```
/*************************************************
*
*       Copyright (C) 2009 Holt Integrated Circuits, Inc.
*       All Rights Reserved
*
* Filename:     SPIDemoR1.h
* Revision:     1.0
* Description: SPIDemo Header File
* Notes:        Used in Project SPIDemoR1.mcp
*
*************************************************/

/* include peripheral declarations */
#include <mc9s12xd64.h>

/* define value for LED's when on and off */
#define ON 0
#define OFF 1

/* define value for switches when up (not pressed) and down (pressed) */
#define UP 1
#define DOWN 0

/* define LED's */
#define LED1 PORTB_PB4
#define LED2 PORTB_PB5
#define LED3 PORTB_PB6
#define LED4 PORTB_PB7

/* define chip select outputs for the 3 SPI ports */
#define SPI1_nSS PTP_PTP3
#define SPI0_nSS PTM_PTM3

/* define 3585 device status flags */
#define TFLAG PORTA_PA0
#define RFLAG PORTA_PA1
#define MR3585    PORTA_PA2

/* define 3598 device status flags */
#define FLAG1 PT01AD1_PT1AD11
#define FLAG2 PORTA_PA4
#define FLAG3 PORTA_PA5
#define FLAG4 PORTA_PA6
#define FLAG5 PORTA_PA7
#define FLAG6 PORTB_PB0
#define FLAG7 PORTB_PB1
#define FLAG8 PORTB_PB2

/* define 3598 and 3599 device status flags */
#define FLAG  PT01AD1_PT1AD10
#define MR3598  PORTB_PB3

#define MARK  PORTA_PA3
```

The HI-3598 has a 16-bit status register which can be polled to determine the status of the receiver FIFOs. Op code 0x06 is used to retrieve the status register bits.

The following code is an example of how to use op code 0x06 and store the contents in a variable to be used throughout the program.

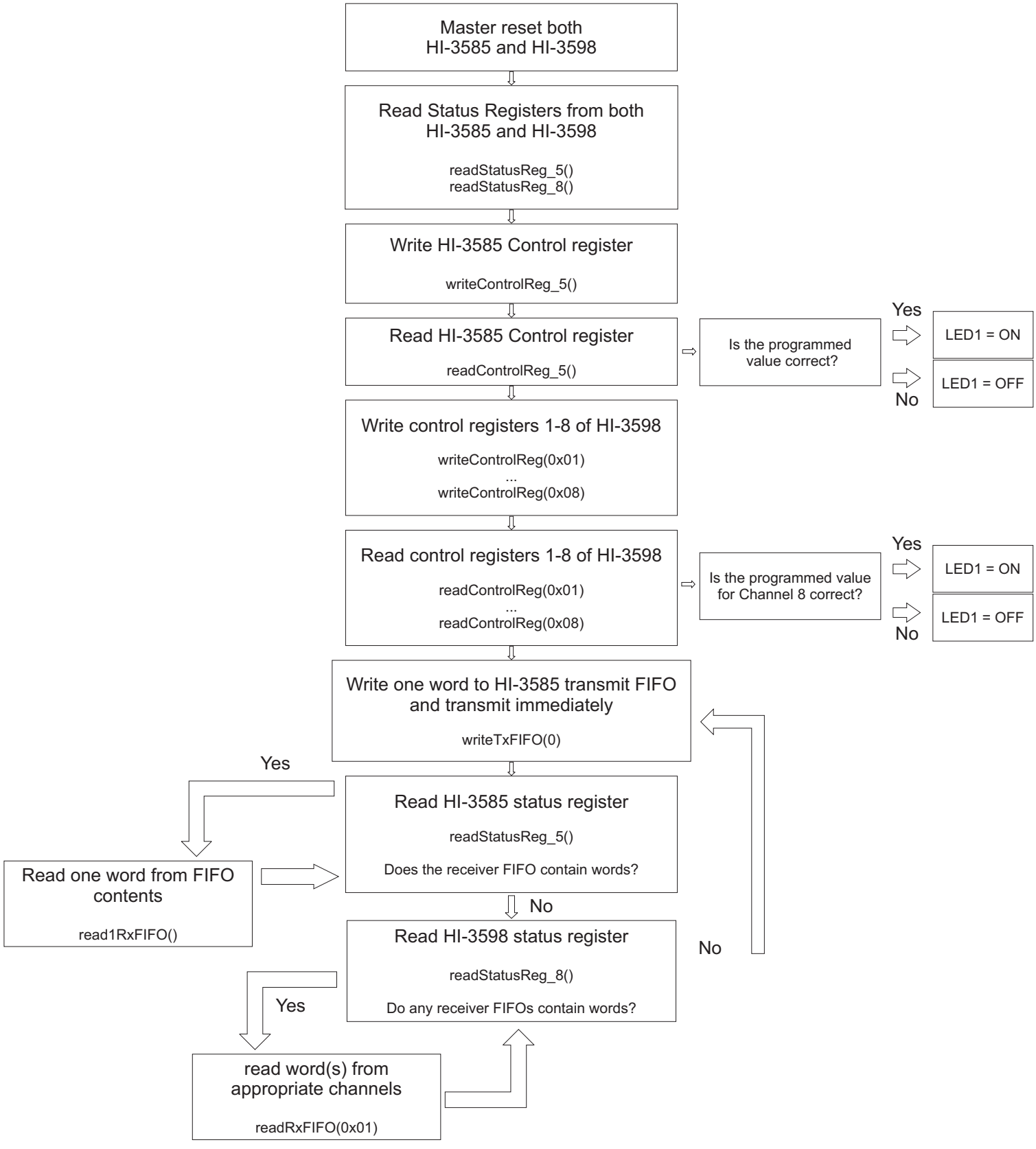
```
// send op code (ignore returned data byte)
rxdata = txrx8bits_8(0x06,1);
// send dummy data / receive Status Reg byte
rxdata = txrx8bits_8(0x00,1);
// send dummy data / receive Status Reg byte
rxdata = txrx8bits_8(0x00,1);
```

Additional Information

Information on the Freescale microcontroller can be found by searching the device type number at www.Freescale.com.

EXAMPLE PROGRAMS continued

The following flow chart describes the execution for the following SPIDemoR1.c program.



EXAMPLE PROGRAMS continued

```
/*****************************************************************************
*
*       Copyright (C) 2010 Holt Integrated Circuits, Inc.
*       All Rights Reserved
*
* Filename:     SPIDemoR1.2.c
* Revision:     1.2
*
* Description: Test Routines for Holt HI-3585, -3598 & -3599
*               ARINC 429 ICs with SPI Interface
*
* Notes:        Uses the Freescale MC9S12XD64.
*                                                 Created using CodeWarrior V5.9.0 for S12X.
*               Programmed using P&E USB Multilink (USB-ML-12)
*
*****************************************************************************/

#include <hidef.h>      /* common defines and macros */
#include "derivative.h"      /* derivative-specific definitions */

#include "SPIDemoR1.h"        // Include the demo board declarations

/* Global Variables */

unsigned long RxBusWord_5[32], TxBusWord_5[32];
unsigned long RxBusWord_8[8][4], TxBusWord_8[32], StatusWord_8;

/* Initialize Control Registers */
unsigned short ControlReg_5 = 0x2800 ; //auto start, labels no reverse
unsigned short ControlReg_8 = 0x0220;  //labels no reverse


//=================================================================
/* Initial selections for active ARINC word labels
   Each byte defines 8 label addresses, LS bit = lowest addr 0 of 7
                                        MS bit = highest addr 7 of 7
     [n]   = array pointer value,
   xxx-yyy = corresponding ARINC label address range

   Example:
   [3] denotes LabelArray[3] which controls label addr 24-31 decimal.
   Bit 0 controls label addr 24, Bit 1 controls label addr 25, etc. */

unsigned char LabelArray_5[32] = {

// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
// 000-007 008-015 016-023 024-031 032-039 040-047 048-055 056-063
     0x00,   0x00,   0X00,   0X00,   0X00,   0X00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
// 064-071 072-079 080-087 088-095 096-103 104-111 112-119 120-127
     0X00,   0X00,   0X00,   0X00,   0X00,   0X00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [16]    [17]    [18]    [19]    [20]    [21]    [22]    [23]
// 128-135 136-143 144-151 152-159 160-167 168-175 176-183 184-191
     0X00,   0X00,   0X00,   0X00,   0X00,   0X00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [24]    [25]    [26]    [27]    [28]    [29]    [30]    [31]
// 192-119 200-207 208-215 216-223 224-231 232-239 240-247 248-255
     0X00,   0X00,   0X00,   0X00,   0X00,   0X00,   0x00,   0x00
// ---------------------------------------------------------------
//
};

// label array for HI-3598 or HI-3599
//=================================================================
/* Initial selections for active ARINC word labels

     [n][k]   = n: channel; k: label position,

   Example:
   [2][5] denotes channel 2, label position 5 */
```

```
unsigned char LabelArrayCh_8 [8][16] = {

{  // channel 1
// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
},

//  channel 2[16] =
{

// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
},

// channel 3[16] =
{

// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
},

// channel 4[16] =
{

// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
},

// channel 5[16] =
{

// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
},

// channel 6[16] =
{

// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
},

// channel 7[16] =
{
```

```
// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
},

// channel 8[16] =
{
// ---------------------------------------------------------------
//   [0]     [1]     [2]     [3]     [4]     [5]     [6]     [7]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------
//   [8]     [9]     [10]    [11]    [12]    [13]    [14]    [15]
     0x00,   0x00,   0X00,   0X00,   0X00,   0x00,   0X00,   0X00,
// ---------------------------------------------------------------

}
}
;

/* ---------------------------------------------------------------------
/  Initialization Function. Is called by main() after reset
/  ---------------------------------------------------------------------

Argument(s):  none

     Return:  nothing

     Action:  initializes processor configuration registers */

void PeriphInit(void)
{

    DDRB_DDRB0 = 0;           // Port B[0..2] input (FLAG6-8)
    DDRB_DDRB1 = 0;
    DDRB_DDRB2 = 0;
    DDRB_DDRB3 = 1;           // Port B[3] output MR3598
    DDRB_DDRB4 = 1;           // Port B[4..7] output (LED1-LED4)
    DDRB_DDRB5 = 1;
    DDRB_DDRB6 = 1;
    DDRB_DDRB7 = 1;

    LED1 = OFF;                                    // Turn Off LEDs
    LED2 = OFF;
    LED3 = OFF;
    LED4 = ON;                // Leave LED4 as an indicator

    ATD1DIEN1_IEN0 =1;
    ATD1DIEN1_IEN1 =1;

    DDRA_DDRA0 = 0;           // Port A[0..1] input (TFLAG & RFLAG)
    DDRA_DDRA1 = 0;
    DDRA_DDRA2 = 1;           // Port A[2] output (MR3585)
    DDRA_DDRA3 = 1;           // Port A[3] output (MARK)
    DDRA_DDRA4 = 0;           // Port A[4..7] input (Flags for 3598)
    DDRA_DDRA5 = 0;
    DDRA_DDRA6 = 0;
    DDRA_DDRA7 = 0;
    DDR1AD1_DDR1AD10 =0;
    DDR1AD1_DDR1AD11  =0;

    PUCR_PUPAE = 1;           // Turn on pullups for TFLAG & RFLAG
    PUCR_PUPBE = 1;
    PER1AD1_PER1AD10=1;
    PER1AD1_PER1AD11=1;

    MR3585 = 0;
    MR3585 = 0;
    MARK = 0;

    DBGC1_BDM = 1;             // BKGD enable
    FSEC_SEC = 10;             // unsecure device
```

```
/* ---------------------------------------
PLL init for 4MHz osc ---> 80MHz bus (max freq)
load synth register = 9 as (2(9+1)) x 4MHz = 80MHz  */
SYNR = 10;
// wait for PLL LOCK flag to go high
while (!(CRGFLG & 0x08)) {
  ;
}
// set PLLSEL bit in Clock Select register
CLKSEL = 0x80;

/* ------------------------------------------------------------------
The Freescale DEMO9S112XDT152 Demo Board has 3 SPI ports.
Each has pins muxed with other features or general purpose I/O.
Some SPI port assignments are nonstandard since the board uses
the 80-QFP package with reduced pin count.

------------------------------------------------------------------
SPI0 - All 4 pins are available. For the 80-QFP package, SPI0
only appears on Port M [5:2]. The Module Routing Register MODRR
bit 4 must be set to make the SPI appear on Port M. The SPI0
Control Register SPE enable bit must be set.
              Signal    U1 Pin       J1
               Name     80PQFP      Pin
              ------    ------     ------
               SCK0       70         21
               /SS0       72         23
              MOSI0       71         17
              MISO0       73         19

The next 5 lines ONLY apply if using SPI0...    */
DDRM_DDRM3 = 1;       // make Port M[3] output (SPI0 /SS)
SPI0_nSS = 1;         // default SPI0 /SS state = high
MODRR_MODRR4 = 1;     // Route SPI0 to Port M[5:2] by setting MODRR[4]

/*----------------------------------------------------------------
SPI1 - All pins are available, appearing on Port P[3:0] when SPI1
Control Register SPE enable bit is set. Two SPI1 I/O pins are used
for Freescale demo board pushbuttons SW1 and SW2. If using SPI1,
SW1 and SW2 should be disconnected from the microprocessor pins, and
could be be reassigned to spare I/O if needed. Remove "User Jumpers"
1 & 2 to disconnect pushbuttons. Suggestion: Edit the header file to
comment-out SW1 and SW2 symbols, so the complier doesn't accidentally
create conflicts if SW1 or SW2 references persist in C program.
              Signal    U1 Pin       J1
               Name     80PQFP      Pin
              ------    ------     ------
               SCK1        2         30
               /SS1        1         32
              MOSI1        3         11
              MISO1        4          9

The next 2 lines ONLY apply if using SPI1. ..   */
DDRP_DDRP3 = 1;     // make Port P[3] output (SPI1 /SS)
SPI1_nSS = 1;       // default SPI1 /SS state = high

/*----------------------------------------------------------------
SPI2 - When using the 80-pin PQFP, 3 of 4 pins are available on
Port P bits 4,5 & 7 when SPI2 Control Register SPE bit is asserted..
              Signal    U1 Pin       J1
               Name     80PQFP      Pin
              ------    ------     ------
               SCK2       78          6
               /SS2       --          --   not avail in 80-PQFP
              MOSI2       79         36
              MISO2       80         34

NOTE: When using the 80-QFP package, SPI2 lacks an /SS output. The
/SS output is not essential in many SPI applications, but is needed
for HI-3585 interface. A bit-banged general purpose output can provide
the /SS function BUT this will require modifications to the 8-bit
SPI transfer functions that use hardware "auto /SS" control.

(no SPI2 initialization provided due to the above compatibility issue)
```

```
    ------------------------------------------------------------------- */

    // used for 3585
    SPI0CR1 = SPI0CR1_SPE_MASK|SPI0CR1_SSOE_MASK;
    // enable SPI0 mode fault
    SPI0CR2 = SPI0CR2_MODFEN_MASK;
    // set baud rate at 5.0 Mbps  (80MHz / 8)
    SPI0BR = 0x05;
    // read status (the write has no effect)
    SPI0SR = SPI0SR;
    // toggle MSTR bit to idle SPI0 in Master State
    SPI0CR1 = SPI0CR1_SPE_MASK|SPI0CR1_SSOE_MASK|SPI0CR1_MSTR_MASK;

    //used for 3598 or 3599
    SPI1CR1 = SPI1CR1_SPE_MASK|SPI1CR1_SSOE_MASK;
    // enable SPI0 mode fault
    SPI1CR2 = SPI1CR2_MODFEN_MASK;
    // set baud rate at 5.0 Mbps  (80MHz / 8)
    SPI1BR = 0x05;
    // read status (the write has no effect)
    SPI1SR = SPI1SR;
    // toggle MSTR bit to idle SPI0 in Master State
    SPI1CR1 = SPI1CR1_SPE_MASK|SPI1CR1_SSOE_MASK|SPI1CR1_MSTR_MASK;


}   /* PeriphInit() */



/* ---------------------------------------------------------------------
/  Send 8-Bit Op Code without Data
/  ---------------------------------------------------------------------

Argument(s):  txbyte, return_when_done

     Return:  nothing

     Action:  This function sends 8 bits using SPI0 (_5) and SPI1 (_8).
              This function can only be used with op codes
              that DO NOT have an associated Data Field:

              Examples for HI-3585
              OP CODE   ACTION
               0x01     Master Reset
               0x02     Reset/disable all label selections
               0x03     Set/enable all label selections
               0x11     Reset the Transmit FIFO
               0x12     Enable ARINC bus transmission

              If return_when_done is True, the function waits for transfer
              completion before returning, which may be needed for back-to-
              back commands. If return_when_done is False, the function
              returns immediately after initiating the transfer.

Example Use:  txOpCode_5(0x01,0); // apply MR, return immediately */

void txOpCode_5 (unsigned char txbyte, unsigned char return_when_done) {

  unsigned char dummy;

  dummy = SPI0SR;          // clear SPI status register

  SPI0DR = txbyte;         // write Data Register to begin transfer

  if (return_when_done) {  // optional wait for SPIF flag
    while (!SPI0SR_SPIF) {
      ;
    }
  }
  dummy = SPI0DR;           // clear the SPIF bit SPI0SR_SPIF

}    /* txOpCode */

void txOpCode_8 (unsigned char txbyte, unsigned char return_when_done) {

  unsigned char dummy;
```

```
  dummy = SPI1SR;          // clear SPI status register

  SPI1DR = txbyte;         // write Data Register to begin transfer

  if (return_when_done) {  // optional wait for SPIF flag
    while (!SPI1SR_SPIF) {
      ;
    }
  }
  dummy = SPI1DR;           // clear the SPIF bit SPI0SR_SPIF

}

/* ------------------------------------------------------------------
/  SPI0 8-Bit Send Data / Receive Data Function
/  ------------------------------------------------------------------

Argument(s):  txbyte, return_when_done

     Return:  rxbyte

     Action:  Using SPI0/SPI1, this function sends txbyte and returns rxbyte
              as part of a chained multi-byte transfer. The calling
              program controls the /SS chip select instead of using the
              automatic /SS option available in the Freescale SPI port.
              This permits simple chaining of op code commands and Tx/Rx
              data as a series of 8-bit read/writes followed by /SS
              deassertion by the calling program.

              If return_when_done is True, the function waits for transfer
              completion before returning, which may be needed for back-to-
              back commands. If return_when_done is False, the function
              returns immediately after initiating the transfer.

Example Call: rcv_byte = txrx8bits_5(0xFF,1) // sends data 0xFF then returns
                                             // data when xfer is done  */

unsigned char txrx8bits_5 (unsigned char txbyte, unsigned char return_when_done) {

  unsigned char rxbyte;

  rxbyte = SPI0SR;         // clear SPI status register

  SPI0DR = txbyte;         // write Data Register to begin transfer

  if (return_when_done) {  // optional wait for SPIF flag
    while (!SPI0SR_SPIF);
  }
  // get received data byte from Data Register
  return rxbyte = SPI0DR;

}    /* txrx8bits */

unsigned char txrx8bits_8 (unsigned char txbyte, unsigned char return_when_done) {

  unsigned char rxbyte;

  rxbyte = SPI1SR;         // clear SPI status register

  SPI1DR = txbyte;         // write Data Register to begin transfer

  if (return_when_done) {  // optional wait for SPIF flag
    while (!SPI1SR_SPIF);
  }
  // get received data byte from Data Register
  return rxbyte = SPI1DR;

}

/* ------------------------------------------------------------------
/  Write HI-3585 Control Register Function
/  ------------------------------------------------------------------

Argument(s):  none
```

```
     Return:  nothing

     Action:  Using SPI0, this function transmits op code 0x10 plus
              16-bits of CR data using three 8-bit SPI transfers.
              The global variable ControlReg is loaded to Control reg.

Example call: ControlReg = 0x1234;  // set value to be loaded
              writeControlReg();    // SPI transfer     */

void writeControlReg_5 (void) {

  unsigned char dummy;

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;
  dummy = SPI0SR;       // clear SPI status register
  SPI0DR = 0x10;        // writing opcode to Data Reg starts SPI xfer
  while (!SPI0SR_SPIF) {
    ;
  }
  dummy = SPI0DR;       // read Rx data in Data Reg to reset SPIF
  //-------------------------------------------
  SPI0DR = (char)(ControlReg_5 >> 8); // write upper Control Register
  while (!SPI0SR_SPIF) {
    ;
  }
  dummy = SPI0DR;       // read Rx data in Data Reg to reset SPIF
  //-------------------------------------------
  SPI0DR = (char)(ControlReg_5 & 0xFF); // write lower Control Register
  while (!SPI0SR_SPIF) {
    ;
  }
  dummy = SPI0DR;       // read Rx data in Data Reg to reset SPIF
  //===========================================
  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;

} /* writeControlReg() */

/* ---------------------------------------------------------------------
/  Write HI-3598 or HI-3599 Control Register Function
/  ---------------------------------------------------------------------

Argument(s):  Channel number

     Return:  nothing

     Action:  Using SPI1, this function transmits op code 0xn4 plus
              16-bits of CR data using three 8-bit SPI transfers, where
              n is the channel number

Example call: ControlReg = 0x1234;    // set value to be loaded
              writeControlReg(0x04);  // SPI transfer to channel 4    */

void writeControlReg_8 (unsigned short j) {

  unsigned char dummy;

  // disable auto /SS output, reset /SS Output Enable
  SPI1CR1 = SPI1CR1 & ~SPI1CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 & ~SPI1CR2_MODFEN_MASK;
  // assert the SPI1 /SS strobe
  SPI1_nSS = 0;

  dummy = SPI1SR;       // clear SPI status register
  SPI1DR = (j << 4) + 0x04;;        // writing opcode to Data Reg starts SPI xfer
```

```
  while (!SPI1SR_SPIF) {
    ;
  }
  dummy = SPI1DR;       // read Rx data in Data Reg to reset SPIF
  //-------------------------------------------
  SPI1DR = (char)(ControlReg_8 >> 8); // write upper Control Register
  while (!SPI1SR_SPIF) {
    ;
  }
  dummy = SPI1DR;       // read Rx data in Data Reg to reset SPIF
  //-------------------------------------------
  SPI1DR = (char)(ControlReg_8 & 0xFF); // write lower Control Register
  while (!SPI1SR_SPIF) {
    ;
  }
  dummy = SPI1DR;       // read Rx data in Data Reg to reset SPIF
  //===========================================
  // de-assert the SPI1 /SS strobe
  SPI1_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI1CR1 = SPI1CR1 | SPI1CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 | SPI1CR2_MODFEN_MASK;

} /* writeControlReg() */


/* ---------------------------------------------------------------------
/  Read HI-3585 Control Register Function
/  ---------------------------------------------------------------------

Argument(s):  none

     Return:  16-bit Control Register value

     Action:  Using SPI0, this function transmits op code 0x0B plus
              16-bits of dummy data using three 8-bit SPI transfers.
              The last 2 transfers receive Control Register data bytes
              that are combined to yield the 16-bit value returned */

unsigned short readControlReg_5 (void) {

  unsigned short rxword;

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;
  //-------------------------------------------
  rxword = SPI0SR;        // clear SPI status register
  SPI0DR = 0x0B;          // writing opcode to Data Reg starts SPI xfer
  while (!SPI0SR_SPIF);   // wait for SPIF flag assertion
  rxword = SPI0DR;        // read/ignore Rx data in Data Reg, resets SPIF
  //-------------------------------------------
  SPI0DR = 0;             // send dummy data, receive upper Control Reg
  while (!SPI0SR_SPIF) {   // wait for SPIF flag assertion
  }
  rxword = (SPI0DR<<8);   // read upper Control Reg byte in Data Reg
  //-------------------------------------------
  SPI0DR = 0;             // send dummy data, receive lower Control Reg
  while (!SPI0SR_SPIF);   // wait for SPIF flag assertion
  rxword = rxword|SPI0DR; // read lower Control Reg byte in Data Reg
  //-------------------------------------------
  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;
  //-------------------------------------------
  return rxword;

} /* readControlReg() */
```

```
/* ---------------------------------------------------------------
/  Read HI-3598 or HI-3599 Control Register Function
/  ---------------------------------------------------------------

Argument(s):  j = channel number

     Return:  16-bit Control Register value

     Action:  Using SPI1, this function transmits op code 0xn5 plus
              16-bits of dummy data using three 8-bit SPI transfers.
              The last 2 transfers receive Control Register data bytes
              that are combined to yield the 16-bit value returned */

unsigned short readControlReg_8 (unsigned short j) {

  unsigned short  rxword;

  // disable auto /SS output, reset /SS Output Enable
  SPI1CR1 = SPI1CR1 & ~SPI1CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 & ~SPI1CR2_MODFEN_MASK;
  // assert the SPI1 /SS strobe
  SPI1_nSS = 0;
  //-------------------------------------------
  rxword = SPI1SR;        // clear SPI status register

  SPI1DR = (j << 4) + 0x05;          // writing opcode to Data Reg starts SPI xfer
  while (!SPI1SR_SPIF);   // wait for SPIF flag assertion
  rxword = SPI1DR;        // read/ignore Rx data in Data Reg, resets SPIF
  //-------------------------------------------
  SPI1DR = 0;             // send dummy data, receive upper Control Reg
  while (!SPI1SR_SPIF) {   // wait for SPIF flag assertion
  }
  rxword = (SPI1DR<<8);   // read upper Control Reg byte in Data Reg
  //-------------------------------------------
  SPI1DR = 0;             // send dummy data, receive lower Control Reg
  while (!SPI1SR_SPIF);   // wait for SPIF flag assertion
  rxword = rxword|SPI1DR; // read lower Control Reg byte in Data Reg
  //-------------------------------------------
  // de-assert the SPI1 /SS strobe
  SPI1_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI1CR1 = SPI1CR1 | SPI1CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 | SPI1CR2_MODFEN_MASK;
  //-------------------------------------------
  return rxword;

} /* readControlReg() */

/* ---------------------------------------------------------------
/  Read HI-3585 Status Register Function
/  ---------------------------------------------------------------

Argument(s):  none

     Return:  8-bit Status Register data

     Action:  Using SPI0, this function transmits op code 0x0A plus
              1 byte of dummy data using two 8-bit SPI transfers.
              The received byte from SPI transfer #2 is returned  */

unsigned char readStatusReg_5 (void) {

  unsigned char rxdata;

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;
  // send op code (ignore returned data byte)
  rxdata = txrx8bits_5(0x0A,1);
```

```
  // send dummy data / receive Status Reg byte
  rxdata = txrx8bits_5(0xFF,1);
  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;
  return rxdata;

} /* readStatusReg() */

/* ---------------------------------------------------------------------
/  Read HI-3598 or HI-3599 Status Register Function
/  ---------------------------------------------------------------------

Argument(s):  none

     Return:  8-bit Status Register data

     Action:  Using SPI1, this function transmits op code 0x06 plus
              1 byte of dummy data using two 8-bit SPI transfers.
              The received byte from SPI transfer #2 is returned  */

unsigned char readStatusReg_8 (void) {

  unsigned char rxdata;

  // disable auto /SS output, reset /SS Output Enable
  SPI1CR1 = SPI1CR1 & ~SPI1CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 & ~SPI1CR2_MODFEN_MASK;
  // assert the SPI1 /SS strobe
  SPI1_nSS = 0;
  // send op code (ignore returned data byte)
  rxdata = txrx8bits_8(0x06,1);
  // send dummy data / receive Status Reg byte
  rxdata = txrx8bits_8(0x00,1);
  // send dummy data / receive Status Reg byte
  rxdata = txrx8bits_8(0x00,1);
  // de-assert the SPI1 /SS strobe
  SPI1_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI1CR1 = SPI1CR1 | SPI1CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 | SPI1CR2_MODFEN_MASK;
  return rxdata;

} /* readStatusReg() */



/* ---------------------------------------------------------------------
/  Write HI-3585 ACLK Divisor Function
/  ---------------------------------------------------------------------

Argument(s):  divisor (only decimal 1,2,4,8 or 10 is valid)

     Return:  none

     Action:  Using SPI0, this function transmits op code 0x07 plus
              1 byte Divisor data using two 8-bit SPI transfers.

Example Use:  writeACLKdiv(10); // writes ACLK div-by-10    */

void writeACLKdiv (unsigned char divisor) {

  unsigned char rxdata;

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;
```

```
  // send op code (ignore returned data byte)
  rxdata = txrx8bits_5(0x07,1);
  // send 8-bit divisor (ignore returned data byte)
  rxdata = txrx8bits_5(divisor,1);
  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;

} /* writeACLKdiv() */


/* ---------------------------------------------------------------------
/  Read HI-3585 ACLK Divisor Function
/  ---------------------------------------------------------------------

Argument(s):  none

     Return:  8-bit Status Register data

     Action:  Using SPI0, this function transmits op code 0x0C plus
              1 byte of dummy data using two 8-bit SPI transfers.
              The received byte from SPI transfer #2 is returned */

unsigned char readACLKdiv (void) {

  unsigned char rxdata;

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;
  // send op code (ignore returned data byte)
  rxdata = txrx8bits_5(0x0C,1);
  // send dummy data / receive Status Reg byte
  rxdata = txrx8bits_5(0x00,1);
  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;
  return rxdata;

} /* readACLKdiv() */


/* ---------------------------------------------------------------------
/  Read HI-3585 next RxFIFO Word Function
/  ---------------------------------------------------------------------

Argument(s):  none

     Return:  next 32-bit ARINC word in RxFIFO

     Action:  Using SPI0, this function transmits op code 0x08 plus
              4 bytes of dummy data using five 8-bit SPI transfers.
              The received bytes from SPI transfers #2-5 are merged
              to yield the 32-bit ARINC word that is returned

unsigned long read1RxFIFO (void) {

    unsigned long j, rxdata;  // long = 32 bits

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;
  // send op code (ignore returned data byte)
```

```
  rxdata = txrx8bits_5(0x08,1);
  // send dummy data / receive and left-justify most signif. byte
  j = txrx8bits_5(0x00,1);
       rxdata = ( j << 24);
  // send dummy data / receive, left-shift and OR next byte
  j = txrx8bits_5(0x00,1);
       rxdata = rxdata | (j << 16);
  // send dummy data / receive, left-shift and OR next byte
  j = txrx8bits_5(0x00,1);
       rxdata = rxdata | (j << 8);
  // send dummy data / receive and OR the least signif. byte
  j = txrx8bits_5(0x00,1);
       rxdata = rxdata | j;
  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // 0nable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;
  return rxdata;

} /* read1RxFIFO() */


/* ---------------------------------------------------------------------
/  Read HI-3598 or HI-3599 RxFIFO Word Function
/  ---------------------------------------------------------------------

Argument(s):  channel number

     Return:  next 32-bit ARINC word in RxFIFO

     Action:  Using SPI1, this function transmits op code 0xn8 plus
              4 bytes of dummy data using five 8-bit SPI transfers.
              The received bytes from SPI transfers #2-5 are merged
              to yield the 32-bit ARINC word that is returned

unsigned long readRxFIFO (unsigned short k) {

  unsigned long j, rxdata;

  // disable auto /SS output, reset /SS Output Enable
  SPI1CR1 = SPI1CR1 & ~SPI1CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 & ~SPI1CR2_MODFEN_MASK;
  // assert the SPI1 /SS strobe
  SPI1_nSS = 0;
  // send op code (ignore returned data byte)
  rxdata = txrx8bits_8((k << 4) + 0x03,1);
  // read up to 32 ARINC 32-bit data words
  // send dummy data / receive and left-justify most signif. byte
  j = txrx8bits_8(0x00,1);
  rxdata = (j << 24);
  // send dummy data / receive, left-shift then OR next byte
  j = txrx8bits_8(0x00,1);
  rxdata = rxdata | (j << 16);

  // send dummy data / receive, left-shift then OR next byte
  j = txrx8bits_8(0x00,1);
  rxdata = rxdata | (j << 8);

  // send dummy data / receive and OR the least signif. byte
  j = txrx8bits_8(0x00,1);
  rxdata = rxdata | j ;

  // de-assert the SPI1 /SS strobe
  SPI1_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI1CR1 = SPI1CR1 | SPI1CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 | SPI1CR2_MODFEN_MASK;

  return rxdata;

} /* readRxFIFO() */
```

```
/* ---------------------------------------------------------------
/  Write HI-3585 TxFIFO Function
/  ---------------------------------------------------------------

Argument(s):  words_to_send (number of words to load, range 1 to 32)

     Return:  none

     Action:  Using SPI0, this function transmits op code 0x0E. Then
              one ARINC word is written using four 8-bit transfers.
              The returned data from the 4 transfers is ignored. The
              word count is decremented

Example Use:  writeTxFIFO(11); // writes 11 words to TxFIFO from the
                               // 32-element array TxBusWord[i], sending
                               //  words TxBusWord[0] - TxBusWord[10] */

void writeTxFIFO_5 (unsigned char words_to_send) {

  unsigned char dummy, i;
  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;
  // send op code (ignore returned data byte)
  dummy = txrx8bits_5(0x0E,1);

  for (i=0; i<words_to_send; i++) {


    // send MS byte (ignore returned data byte)
    dummy = txrx8bits_5((char)(TxBusWord_5[i]>>24),1);
    // send next byte (ignore returned data byte)
    dummy = txrx8bits_5((char)((TxBusWord_5[i]>>16) & 0xFF),1);
    // send next byte (ignore returned data byte)
    dummy = txrx8bits_5((char)((TxBusWord_5[i]>>8) & 0xFF),1);
    // send LS byte (ignore returned data byte)
    dummy = txrx8bits_5((char)(TxBusWord_5[i] & 0xFF),1);

  }
  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;

} /* writeTxFIFO() */



/* ---------------------------------------------------------------
/  Write HI-3598 or HI-3599 TxFIFO Function
/  ---------------------------------------------------------------

Argument(s):  none

     Return:  none

     Action:  Using SPI1, this function transmits op code 0x09. Then
              one ARINC word is written using four 8-bit transfers.
              The returned data from the 4 transfers is ignored. The
              word count is decremented

Example Use:  writeTxFIFO(); // writes a words to self test register from the
                               // 32-element array TxBusWord_8[i]  */

void writeTxFIFO_8low (void) {

  unsigned char dummy, i;
  i=0;
  // disable auto /SS output, reset /SS Output Enable
```

```
  SPI1CR1 = SPI1CR1 & ~SPI1CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 & ~SPI1CR2_MODFEN_MASK;
  // assert the SPI1 /SS strobe
  SPI1_nSS = 0;
  // send op code (ignore returned data byte)
  dummy = txrx8bits_8(0x09,1);


    // send MS byte (ignore returned data byte)
    dummy = txrx8bits_8((char)(TxBusWord_8[i]>>24),1);
    // send next byte (ignore returned data byte)
    dummy = txrx8bits_8((char)((TxBusWord_8[i]>>16) & 0xFF),1);
    // send next byte (ignore returned data byte)
    dummy = txrx8bits_8((char)((TxBusWord_8[i]>>8) & 0xFF),1);
    // send LS byte (ignore returned data byte)
    dummy = txrx8bits_8((char)(TxBusWord_8[i] & 0xFF),1);


  // de-assert the SPI1 /SS strobe
  SPI1_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI1CR1 = SPI1CR1 | SPI1CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 | SPI1CR2_MODFEN_MASK;

} /* writeTxFIFO() */

/* ---------------------------------------------------------------------
/  Write HI-3598 or HI-3599 TxFIFO Function
/  ---------------------------------------------------------------------

Argument(s):  none

     Return:  none

     Action:  Using SPI1, this function transmits op code 0x08. Then
              one ARINC word is written using four 8-bit transfers.
              The returned data from the 4 transfers is ignored. The
              word count is decremented

Example Use:  writeTxFIFO(11); // writes a word to self test register from the
                               // 32-element array TxBusWord_8[i]   */

void writeTxFIFO_8high (void) {

  unsigned char dummy, i;
  i=0;
  // disable auto /SS output, reset /SS Output Enable
  SPI1CR1 = SPI1CR1 & ~SPI1CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 & ~SPI1CR2_MODFEN_MASK;
  // assert the SPI1 /SS strobe
  SPI1_nSS = 0;
  // send op code (ignore returned data byte)
  dummy = txrx8bits_8(0x08,1);


    // send MS byte (ignore returned data byte)
    dummy = txrx8bits_8((char)(TxBusWord_8[i]>>24),1);
    // send next byte (ignore returned data byte)
    dummy = txrx8bits_8((char)((TxBusWord_8[i]>>16) & 0xFF),1);
    // send next byte (ignore returned data byte)
    dummy = txrx8bits_8((char)((TxBusWord_8[i]>>8) & 0xFF),1);
    // send LS byte (ignore returned data byte)
    dummy = txrx8bits_8((char)(TxBusWord_8[i] & 0xFF),1);


  // de-assert the SPI1 /SS strobe
  SPI1_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI1CR1 = SPI1CR1 | SPI1CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 | SPI1CR2_MODFEN_MASK;
} /* writeTxFIFO() */
```

```
/* ---------------------------------------------------------------
/  Reset all HI-3585 ARINC label selections
/  ---------------------------------------------------------------

Argument(s):  none

     Return:  none

     Action:  Using SPI0, this function transmits op code 0x02. This
              op code resets all 256 ARINC labels. This function then
              updates the global LabelArray[n] by setting all bits to
              permit status tracking.                               */

void resetAllLabels_5 (void ) {

  unsigned char i;

  // reset all label bits in HI-358x device
  txOpCode_5(2,1);
  // reset all bits all 32-bytes of global LabelArray[]
  for (i=0; i<32; i++) {
    LabelArray_5[i] = 0;
  }
} /* resetAllLabels */

/* ---------------------------------------------------------------
/  Set all HI-3585 ARINC label selections
/  ---------------------------------------------------------------

Argument(s):  none

     Return:  none

     Action:  Using SPI0, this function transmits op code 0x03. This
              op code sets all 256 ARINC labels. This function then
              updates the global LabelArray[n] by setting all bits to
              permit status tracking.                               */

void setAllLabels_5 (void ) {

  unsigned char i;

  // set all label bits in HI-358x device
  txOpCode_5(3,1);
  // set all bits all 32-bytes of global LabelArray[]
  for (i=0; i<32; i++) {
    LabelArray_5[i] = 0xFF;
  }
} /* setAllLabels */


/* ---------------------------------------------------------------
/  Copy HI-3585 ARINC label selections from LabelArray[] to HI-3585
/  ---------------------------------------------------------------

Argument(s):  none

     Return:  none

     Action:  Using SPI0, this function transmits op code 0x06. This
              function then copies the global LabelArray[n] to HI-358x
              label memory so labels are enabled/disabled to match the
              program's LabelArray[].                               */

void copyAllLabels_5 (void ) {

  unsigned char dummy;
  signed char i;

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
```

```
  SPI0_nSS = 0;

  // send op code (ignore returned data byte)
  dummy = txrx8bits_5(0x06,1);

  // send 32 bytes of ARINC label data
  for (i=31; i>=0; i--) {
    // send 1 byte of label data, ignore returned data byte
    dummy = txrx8bits_5(LabelArray_5[i],1);
  }

  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;
}  /* copyAllLabels */


/* ---------------------------------------------------------------------
/  Verify match: HI-3585 ARINC label selections to LabelArray[]
/  ---------------------------------------------------------------------

Argument(s):  none

     Return:  0xFFFF if LabelArray[n] fully matches HI-358x label memory

              Failing array pointer "n" value 0x0 to 0xFF if a mismatch
              is found. Match testing starts at 0xFF and works toward 0.
              Other mismatches beyond the first failing 8-bit label range
              may exist but are not reported.

     Action:  Using SPI0, this function transmits op code 0x0D. This
              function then compares all 256 bits in HI-358x label
              memory (8 bits at a time) to the corresponding element
              in the program's LabelArray[n]. Return value indicates
              whether or not mismatch is detected.                */

unsigned short checkAllLabels_5 (void) {

  unsigned char rxbyte;
  signed char i;
  unsigned short j;

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;

  // send op code (ignore returned data byte)
  rxbyte = txrx8bits_5(0x0D,1);

  j = 0xffff;
  // starting at high end, read 8-bit increments of ARINC label data
  for (i=31; i>=0; i--) {
    // send dummy data, read 1 byte of label data
    rxbyte = txrx8bits_5(0,1);
    if (rxbyte != LabelArray_5[i]) j=0x0000;
  }

  // no errors, de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;
  // return the "no fail" value
  return j;
}  /* checkAllLabels */


/* ---------------------------------------------------------------------
```

```
/  Set/Reset a single HI-3585 ARINC label, update LabelArray[]
/  -------------------------------------------------------------------

Argument(s):  8-bit label_num (range: 0-0xFF (255)
              label_value, "1" if set, "0" if reset

     Return:  none

     Action:  Using SPI0, this function transmits op code 0x05 or
              0x04 to specify label set or label reset. The function
              then sends the label number, completing the HI-358x
              tranaction.

              The program's LabelArray[n] is updated: A single bit
              is set or reset to match HI-358x label memory.    */

void modifyLabel_5(unsigned char label_num, char label_value) {

  unsigned short i, j;

  // disable auto /SS output, reset /SS Output Enable
  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;
  // assert the SPI0 /SS strobe
  SPI0_nSS = 0;

  // send op code (ignore returned data byte)
  if (label_value == 1) i = txrx8bits_5(0x05,1); // "set" op code
       else i = txrx8bits_5(0x04,1);               // "reset" op code
  // send 1 byte label address, ignore returned data byte
  i = txrx8bits_5(label_num,1);

  // de-assert the SPI0 /SS strobe
  SPI0_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;
  // enable auto /SS output, set SPI0 Mode Fault
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;

  // now update processor LabelArray[]
  // here "i" is the array pointer
  i = label_num >> 3;
  // "j" is an 8-bit mask with one bit set (1,2,4,8,16,32,64 or 128)
  j = 1 << (label_num & 0x7);
  if (label_value == 1) LabelArray_5[i] = LabelArray_5[i] | j; // set bit
  else LabelArray_5[i] = LabelArray_5[i] & ~j;  // reset bit

} /* modifyLabel */


/* -------------------------------------------------------------------
/  Copy HI-3598 or HI-3599 ARINC label selections from LabelArray[][]
/  -------------------------------------------------------------------

Argument(s):  j = channel number

     Return:  none

     Action:  Using SPI1, this function transmits op code 0xj1. This
              function then copies the global LabelArrayCh_8[j][n] to HI-3598/99
              label memory so labels are enabled/disabled to match the
              program's LabelArrayCh_8[][].                          */

void copyAllLabels_8 (unsigned short j ) {

  unsigned char dummy;
  signed char i;

  // disable auto /SS output, reset /SS Output Enable
  SPI1CR1 = SPI1CR1 & ~SPI1CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 & ~SPI1CR2_MODFEN_MASK;
  // assert the SPI1 /SS strobe
  SPI1_nSS = 0;
```

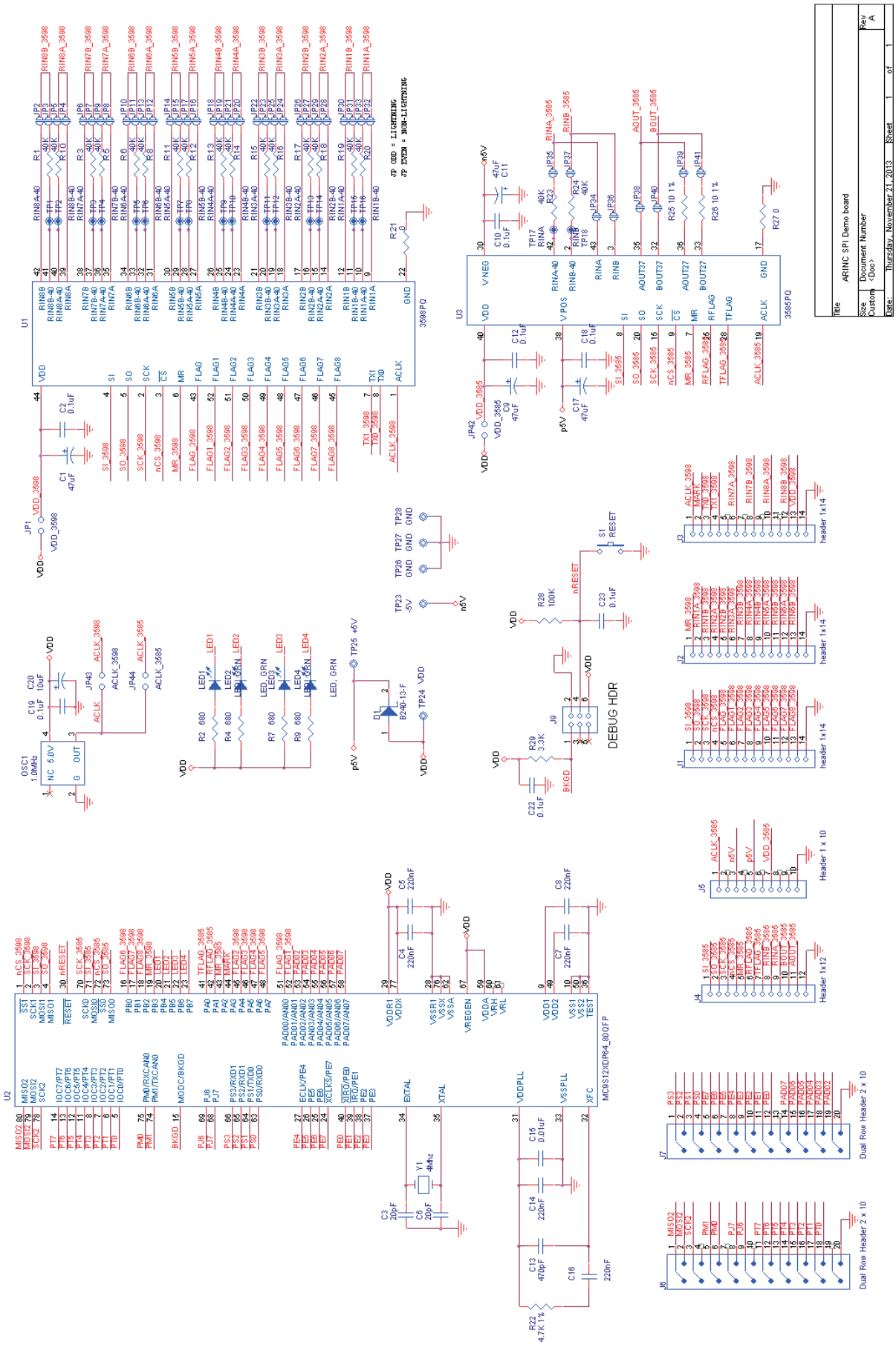
```
  // send op code (ignore returned data byte)

  dummy = txrx8bits_8((j<< 4) + 0x01,1);

  // send 16 bytes of ARINC label data
  for (i=15; i>=0; i--) {
    // send 1 byte of label data, ignore returned data byte
    dummy = txrx8bits_8(LabelArrayCh_8[j-1][i],1);
  }

  // de-assert the SPI1 /SS strobe
  SPI1_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI1CR1 = SPI1CR1 | SPI1CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 | SPI1CR2_MODFEN_MASK;
}  /* copyAllLabels */


/* ---------------------------------------------------------------------
/  Verify match: HI-3598 of HI-3599 ARINC label selections to LabelArray[][]
/  ---------------------------------------------------------------------

Argument(s):  channel number

     Return:  0xFFFF if LabelArray[][n] fully matches HI-3598/9 label memory

     Action:                    */

unsigned short checkAllLabels_8 (unsigned short k) {

  unsigned char rxbyte;
  signed char i;
  unsigned short j;

  // disable auto /SS output, reset /SS Output Enable
  SPI1CR1 = SPI1CR1 & ~SPI1CR1_SSOE_MASK;
  // disable auto /SS output, reset SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 & ~SPI1CR2_MODFEN_MASK;
  // assert the SPI1 /SS strobe
  SPI1_nSS = 0;

  // send op code (ignore returned data byte)

  rxbyte = txrx8bits_8((k << 4) + 0x02,1);

  j = 0xffff;
  // starting at high end, read 8-bit increments of ARINC label data
  for (i=15; i>=0; i--) {
    // send dummy data, read 1 byte of label data
    rxbyte = txrx8bits_8(0,1);
    if (rxbyte != LabelArrayCh_8[k-1][i]) j=0x0000;
  }

  // no errors, de-assert the SPI1 /SS strobe
  SPI1_nSS = 1;
  // enable auto /SS output, set /SS Output Enable
  SPI1CR1 = SPI1CR1 | SPI1CR1_SSOE_MASK;
  // enable auto /SS output, set SPI1 Mode Fault
  SPI1CR2 = SPI1CR2 | SPI1CR2_MODFEN_MASK;
  // return the "no fail" value
  return j;
}  /* checkAllLabels */



// ---------------------------------------------------------------------------
// Main
// ---------------------------------------------------------------------------
void main(void) {

  unsigned short j, rxword;
  unsigned long i, k, txadd;
```

```
unsigned long i1,i2,i3,i4,i5,i6,i7,i8;

//unsigned char rxbyte;
DisableInterrupts;
PeriphInit();                        // initialize microprocessor

// apply HI-3585 and HI-3598/HI-3599 Master Reset.
// by software, return only after completion...
//   txOpCode_5 (0x01,1);
//   txOpCode_8 (0x07,1);

// or, by hardware, set MR , delay, reset MR...
MR3585 = 1;
MR3598 = 1;
for (i = 100; i > 0; i--) ;
MR3585 = 0;
MR3598 = 0;

//read both HI-3585 and HI-3598/HI-3599 status registers
j = readStatusReg_5();
j = readStatusReg_8();



// initialize the HI-3585 Control Register
writeControlReg_5();
// read back HI-3585 Control Register
rxword = readControlReg_5();
// update LED1 to show match or mismatch
if (rxword == ControlReg_5) LED1 = ON;
 else LED1 = OFF;

// initialize the HI-3598 or HI-3599 Control Register
writeControlReg_8(0x01);
writeControlReg_8(0x02);
writeControlReg_8(0x03);
writeControlReg_8(0x04);
writeControlReg_8(0x05);
writeControlReg_8(0x06);
writeControlReg_8(0x07);
writeControlReg_8(0x08);


// read back HI-3598 or HI-3599 Control Register
rxword = readControlReg_8(0x01);
rxword = readControlReg_8(0x02);
rxword = readControlReg_8(0x03);
rxword = readControlReg_8(0x04);
rxword = readControlReg_8(0x05);
rxword = readControlReg_8(0x06);
rxword = readControlReg_8(0x07);
rxword = readControlReg_8(0x08);

// update LED2 to show match or mismatch for channel 8
if (rxword == ControlReg_8) LED2 = ON;
 else LED2 = OFF;


// if labels are to be used, use the following code
/*
copyAllLabels_5();
rxword = checkAllLabels_5();

//an example to set only 1 label
modifyLabel_5(0x12,1);

//an example to reset only 1 label
modifyLabel_5(0x12,0);

copyAllLabels_8(0x01);
rxwordlab = checkAllLabels_8(0x01);
copyAllLabels_8(0x02);
rxwordlab = checkAllLabels_8(0x02);
copyAllLabels_8(0x03);
```

```
rxwordlab = checkAllLabels_8(0x03);
copyAllLabels_8(0x04);
rxwordlab = checkAllLabels_8(0x04);
copyAllLabels_8(0x05);
rxwordlab = checkAllLabels_8(0x05);
copyAllLabels_8(0x06);
rxwordlab = checkAllLabels_8(0x06);
copyAllLabels_8(0x07);
rxwordlab = checkAllLabels_8(0x07);
copyAllLabels_8(0x08);
rxwordlab = checkAllLabels_8(0x08);

 */
//initialize transmit word data
txadd = 0x01010101;
transmitwords:

//load transmit word data to registers
TxBusWord_5[0] =   0x00002008 + txadd;
txadd = txadd + 0x01010101; //add sum to change data

//write transmit word to HI-3585 TX FIFO for immediate transmission
writeTxFIFO_5(1);

//delay for ARINC word to transmit
for (i = 0x1FF; i > 0; i--) ;

RECEIVERFIFO:
LED3=OFF;
LED4=OFF;
//check if HI-3585 receiver has data
while (!(readStatusReg_5()&0x01)){

//HI-3585 receiver contains data, read all
  RxBusWord_5[0] = read1RxFIFO();
  if (RxBusWord_5[0]==TxBusWord_5[0]) LED3=ON;
}

//initialize counters for HI-3598/HI-3599 read
k=0;
i1=0;
i2=0;
i3=0;
i4=0;
i5=0;
i6=0;
i7=0;
i8=0;
i=0;

reading:
//read status register of HI-3598/HI-3599
//if ANY channel has words, read and store
while (!((readStatusReg_8()&0x00FF)==0x00FF)){
  k = readStatusReg_8();
  StatusWord_8 = k;

  if ((k&0x0001) == 0x0){  // check channel 1 for data
     RxBusWord_8[0][i1]=readRxFIFO(0x01);
     if (RxBusWord_8[0][i1]==TxBusWord_5[0])LED4=ON;
     i1=i1+1;
     if (i1 ==4) i1=0;
  }

  if ((k&0x0002) == 0x0){  // check channel 2 for data
     RxBusWord_8[1][i2]=readRxFIFO(0x02);
     if (RxBusWord_8[1][i1]==TxBusWord_5[0])LED4=ON;
     i2=i2+1;
     if (i2 ==4) i2=0;
  }

  if ((k&0x0004) == 0x0){  // check channel 3 for data
     RxBusWord_8[2][i3]=readRxFIFO(0x03);
     if (RxBusWord_8[2][i1]==TxBusWord_5[0])LED4=ON;
     i3=i3+1;
```

```
      if (i3 ==4) i3=0;
    }

     if ((k&0x0008) == 0x0){  // check channel 4 for data
       RxBusWord_8[3][i4]=readRxFIFO(0x04);
       if (RxBusWord_8[3][i1]==TxBusWord_5[0])LED4=ON;
       i4=i4+1;
       if (i4 ==4) i4=0;
     }

    if ((k&0x0010) == 0x0){  // check channel 5 for data
       RxBusWord_8[4][i5]=readRxFIFO(0x05);
       if (RxBusWord_8[4][i1]==TxBusWord_5[0])LED4=ON;
       i5=i5+1;
       if (i5 ==4)i5=0;
    }

    if ((k&0x0020) == 0x0){  // check channel 6 for data
       RxBusWord_8[5][i6]=readRxFIFO(0x06);
       if (RxBusWord_8[5][i1]==TxBusWord_5[0])LED4=ON;
       i6=i6+1;
       if (i6 ==4)i6=0;
    }

    if ((k&0x0040) == 0x0){  // check channel 7 for data
       RxBusWord_8[6][i7]=readRxFIFO(0x07);
       if (RxBusWord_8[6][i1]==TxBusWord_5[0])LED4=ON;
       i7=i7+1;
       if (i7 ==4) i7=0;
    }

    if ((k&0x0080) == 0x0){  // check channel 8 for data
       RxBusWord_8[7][i8]=readRxFIFO(0x08);
       if (RxBusWord_8[7][i1]==TxBusWord_5[0])LED4=ON;
       i8=i8+1;
       if (i8 ==4) i8=0;
    }
  }

  goto transmitwords;


  for(;;) {
    i++;
          if (i & 0x20000) LED3 = 1;
                else LED3 = 0;
  } /* loop forever */
  /* please make sure that you never leave main */


}//end main(void)
```

Figure 1: Board Schematic



Bill of Materials

Bill of Materials		Arinc SPI Evaluation Board. Rev B			Revised: 11/26/2013
Item	*Qty*	*Description*	*Reference*	*DigiKey*	*Mfr P/N*
1	1	PCB, Bare, Eval Board	N/A	--------	JetTech 35822
2	7	Capacitor, Cer 0.1uF 20% 50V Z5U 0805	C2,C10,C12,C18,C19,C22,C23	399-1176-1-ND	Kemet C0805C104M5UACTU
3	1	Capacitor, Cer 0.012uF 50V X7R 080505	C15	399-6742-1-ND	Kemet C0805C123KCRACTU
4	6	Capacitor, Cer 0.22uF 16V 10% X7R 0805	C4,C5,C7,C8,C14,C16	399-8051-1-ND	Kemet C0805C224K4RACTU
5	2	Capacitor, Cer 20pF 16V 5% NP0 0805	C3,C6	399-9987-1-ND	Kemet C0805C200J4GAC7800
6	1	Capacitor, Cer 470pF 50V 10% X7R 0805	C13	399-1142-1-ND	Kemet C0805C471K5RACTU
7	1	Capacitor Tantalum 10uF 10V 10% 1206	C20	478-3317-1-ND	Avx TPSA106K010R1800
8	4	Capacitor Tantalum 47uF 16V 20% 6032 SMD	C1,C9,C11,C17	478-3910-1-ND	Avx TAJC476K010RNJ
9	1	Header, Male 1x12, .1" Pitch	J4	S1012E-12-ND	Sullins S1012E-12-ND
10	1	Header, Male 1x10, .1" Pitch	J5	S1012E-10-ND	Sullins S1012E-10-ND
11	3	Header, Male 1x14, .1" Pitch	J1,J2,J3	S1012E-14-ND	Sullins S1012E-14-ND
12	1	Header, Male 2x3, R/A, .1" Pitch	DEBUG HEADER	S2112E-03-ND	Sullins S2112E-03-ND
13	2	Header, Male 2x10, .1" Pitch	J6,J7	S2012E-10-ND	Sullins S2012E-10-ND
14	4	Header, Male 1x2, .1" Pitch	JP1,JP42,JP43,JP44	S1012E-02-ND	Sullins S1012E-02-ND
15	4	Connector shunt, thru-hole 0.1"	JP1,JP42,JP43,JP44	A26228-ND	TE Connectivity 382811-8
16	4	LED Green 0805	LED1-LED4	160-1179-1-ND	LiteOn LTST-C170GKT
17	20	Resistor, 0 5% 1/8W 0805	R21,R27, All JPs ODD Numbers	P0.0ACT-ND	Panasonic ERJ-6GEY0R00V
18	4	Resistor, 680 5% 1/8W 0805	R2,R4,R7,R9	P680ACT-ND	Panasonic ERJ-8GEYJ681V
19	1	Resistor, 3.3K 5% 1/8W 0805	R29	P3.3KACT-ND	Panasonic ERJ-8GEYJ332V
20	1	Resistor, 4.7K 1% 1/8W 0805	R22	P4.7KCCT-ND	Panasonic ERJ-6ENF4701V
21	1	Resistor, 10 1% 1/8W 0805	R25,R26	P10.0CCT-ND	Panasonic ERJ-6ENF10R0V
22	18	Resistor, 40.2K 1% 1/8W 0805	R1,R3,R5,R6,R8,R10--R20, R23,R24	P40.2KCCT-ND	Panasonic ERJ-6ENF4022V
23	1	Resistor, 100K 5% 1/8W 0805	R28	P100KACT-ND	Panasonic ERJ-8GEYJ104V
24	1	Osc 1.0MHz 5.0V, 5 x 7 mm SMD	Osc1	CSX-750PCBCT-ND	CSX-750PCBCT
25	1	Crystal 4.0MHz 20pF SMD	Y1	XC1238CT-ND	ECS-40-20-5PX-TR
26	1	Switch Tactile SPST-NO 0.02A 15V	S1	P12943SCT-ND	Panasonic EVQ-Q2K03W
27	1	Diode Schottky 40V 2A SMB	D1	B240-FDICT-ND	Diode B240-13-F
28	1	Test Point, Red Insulator, 0.062" hole	(+)5V	5010K-ND	Keystone 5010
29	3	Test Point, Black Insulator, 0.062" hole	GND	5011K-ND	Keystone 5011
30	1	Test Point, White Insulator, 0.062" hole	(-)5V	5012K-ND	Keystone 5012
31	1	Test Point, Orange Insulator, 0.062" hole	VDD	5013K-ND	Keystone 5013
32	1	IC MCU 16BIT 64KB FLASH 80QFP	U2	MC9S12XD64CAA-ND	FREESCALEMC9S12XD64CAA
33	1	HI-3598PQ 52PQFP	U1	HOLT IC	Holt IC
34	1	HI-3585PQ 44PQFP	U3	HOLT IC	Holt IC
35	5	3M Bumpon	Install at four corners and center.	SJ5746-0-ND	3M: SJ61A1

REVISION HISTORY

Document	Rev.	Date	Description of Change
AN-145	NEW	06/09/09	Initial Release
	A	07/17/09	Corrected mistyped resistor values in schematic
	B	02/02/10	Removed use of op code 09hex and replaced with op code 08hex
	C	07/14/10	Updated source code with label examples and indicators of valid data words
	D	12/17/13	Updated schematic and BOM
	E	9/18/14	Updated schematic and BOM (R25, R26 corrected from 10 kOhm to 10 Ohm).
	F	12/13/17	Update example code.