AN-171

HI-3717 Evaluation Board Software Application Note

March 12, 2015

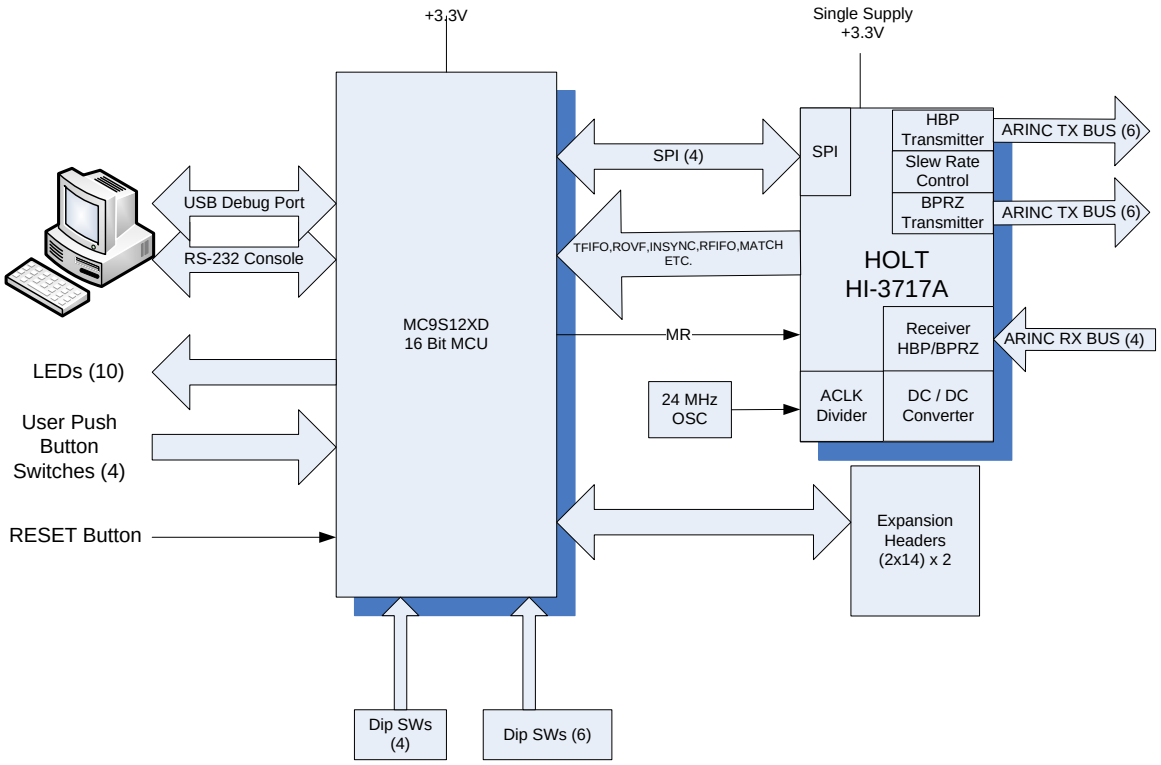
Introduction:

This application note provides an overview of the demo software provided in the Holt HI-3717 ARINC 717 Evaluation Kit. The main sections of this application note are:

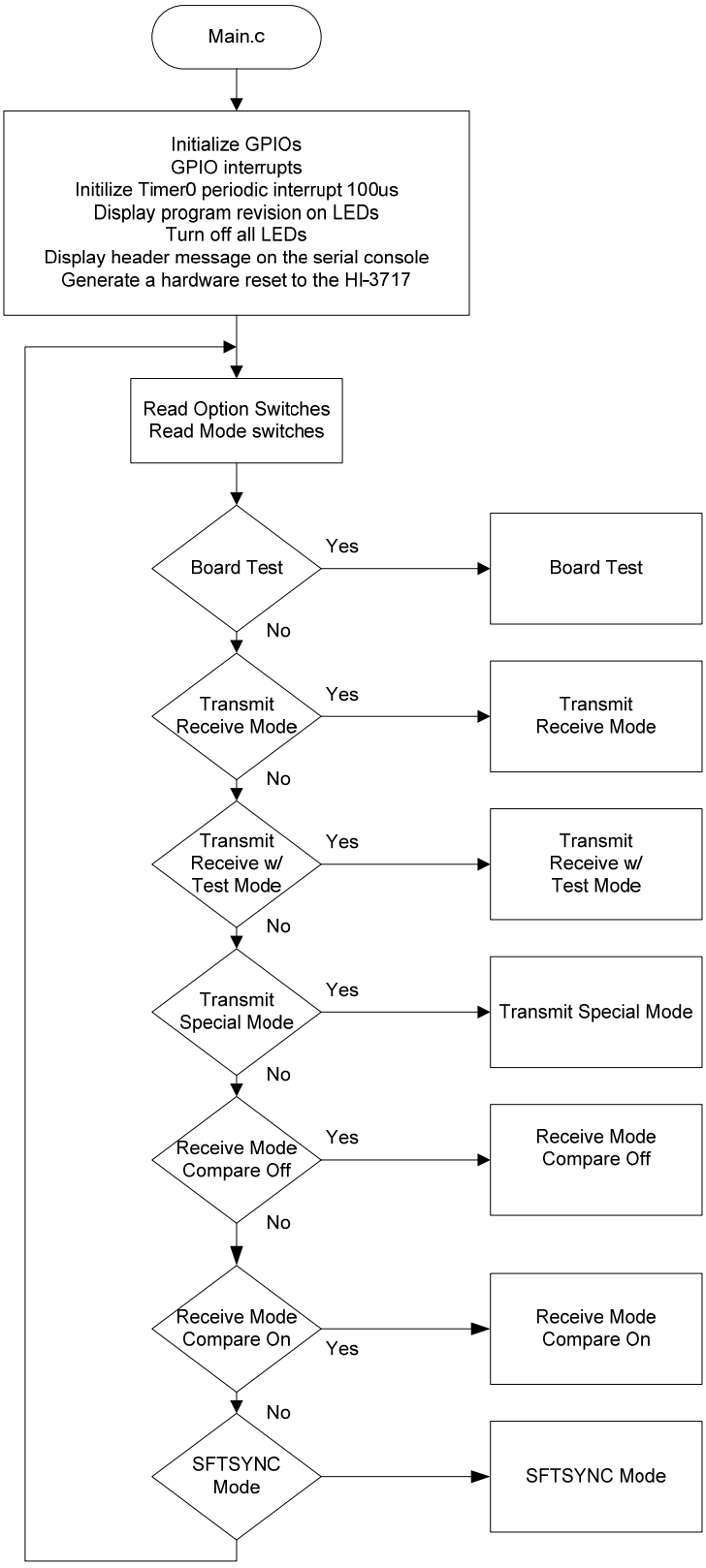
- Demo software overview
- Low level HI-3717A driver functions
- Demo Project setup with Freescale CodeWarrior™.

A Quick Start Guide and a complete User's Guide for HI-3717 evaluation kit can be found on the CD-ROM. Use these guides to become familiar with the board setup and operation of the demo software.

Evaluation Board Diagram



Demo Software Overview Block Diagram



The program enters the mode selected by the mode configuration DIP switches. To restart into a different mode, reconfigure the mode switches and reset the board.

MCU Clock and SPI Frequencies

The Freescale MC9S12XDT512 (MCU) on the main board uses a 4MHz crystal for operation. The built-in PLL multiplies this by 20 to achieve an 80MHz system clock. This system clock is divided by two for a 40MHz Bus Clock used internally for the MCU peripherals.

The PLL is programmed to multiply by 20 by this line of code in the Peripherals.c module:

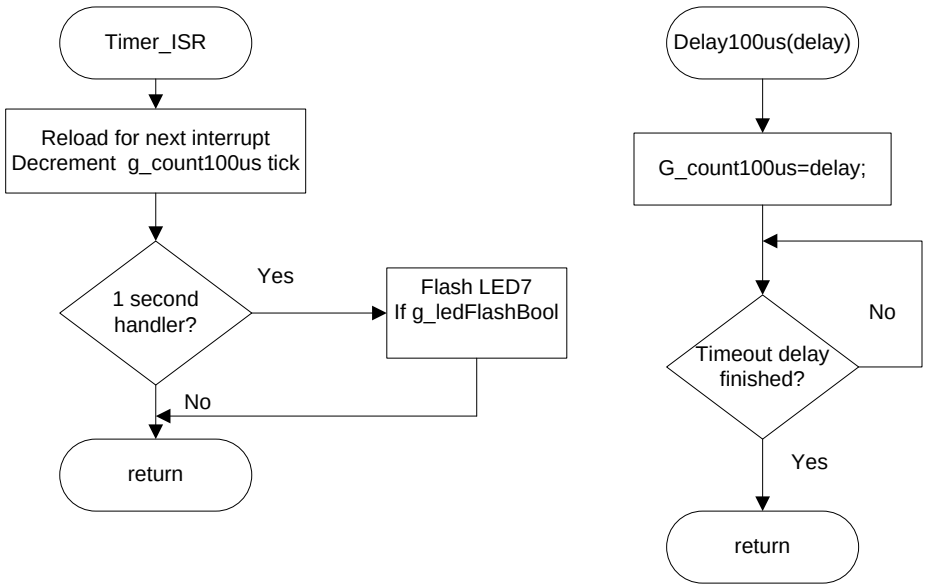
```
SYNR = 9;                   // 80Mhz PLL system clock
```

The SPI frequency is set by this line of code in the Peripherals.c module:

```
SPI0BR = SPI_5MHZ;          // SPI CLK = 5MHz (see "Peripherals.c" for other
                            // rates)
```

The maximum SPI frequency for the HI-3717A is 10MHz.

Timing and Delay Functions



These functions provide the basic timing for the program. The Delay100us() can be used anywhere an accurate delay is needed in the program .

The global g_count100us variable is decremented at the 100us timer rate. This variable is used by a general delay function which can be called with a specified number of delay intervals. The g_count100us variable is a 16-bit integer so the delay ranges from 100us to 6.5536 seconds.

```
// -------------------------------------
// General timer tick 100us for delays
// -------------------------------------
void Delay100us(unsigned int delay){
    g_count100us=delay;
    while(g_count100us);

}
```

A number of predefined constants are defined which can be used by calling the function with these constants.

```
#define K_1MS   10        // 1ms
#define K_10MS  100       // 10ms
#define K_100MS 1000      // 100ms
#define K_1SEC  10000     // 1 second
```

```
Usage: Delay100us(K_1SEC); // delay for one second
```

A one second interrupt handler in the TIMER_ISR is provided. Any code placed here automatically executes every second.

```
if(!count100us)
  {                                  // 1 second scheduler
       count100us = K_1SEC;
       if(ON==g_ledFlashBool)         // Flash the LED7 if enabled
         LED7 ^= TOGGLE;              // Alive 1 second blink
  }
```

GPIO interrupt Handlers

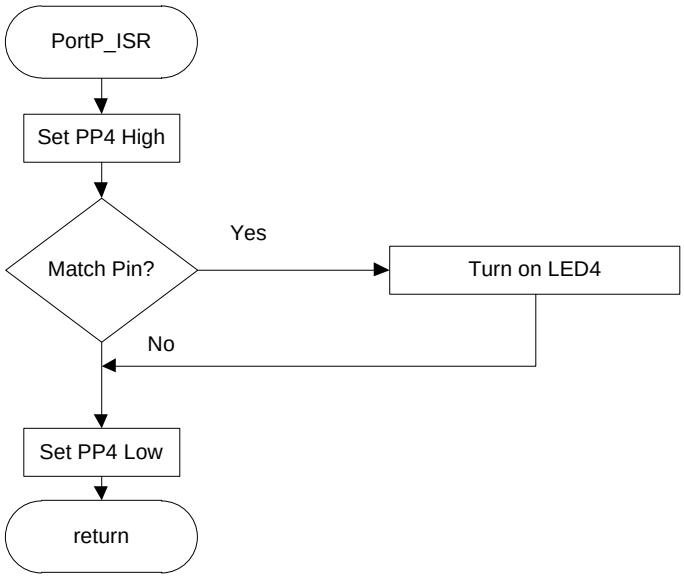
An interrupt handler manages most of the interrupts from the HI-3717A.

```
interrupt 56 void PORTP_ISR(void)
```

Pins that can generate interrupts from the HI-3717A are shown below. The MATCH pin is the only interrupt demonstrated in the code.

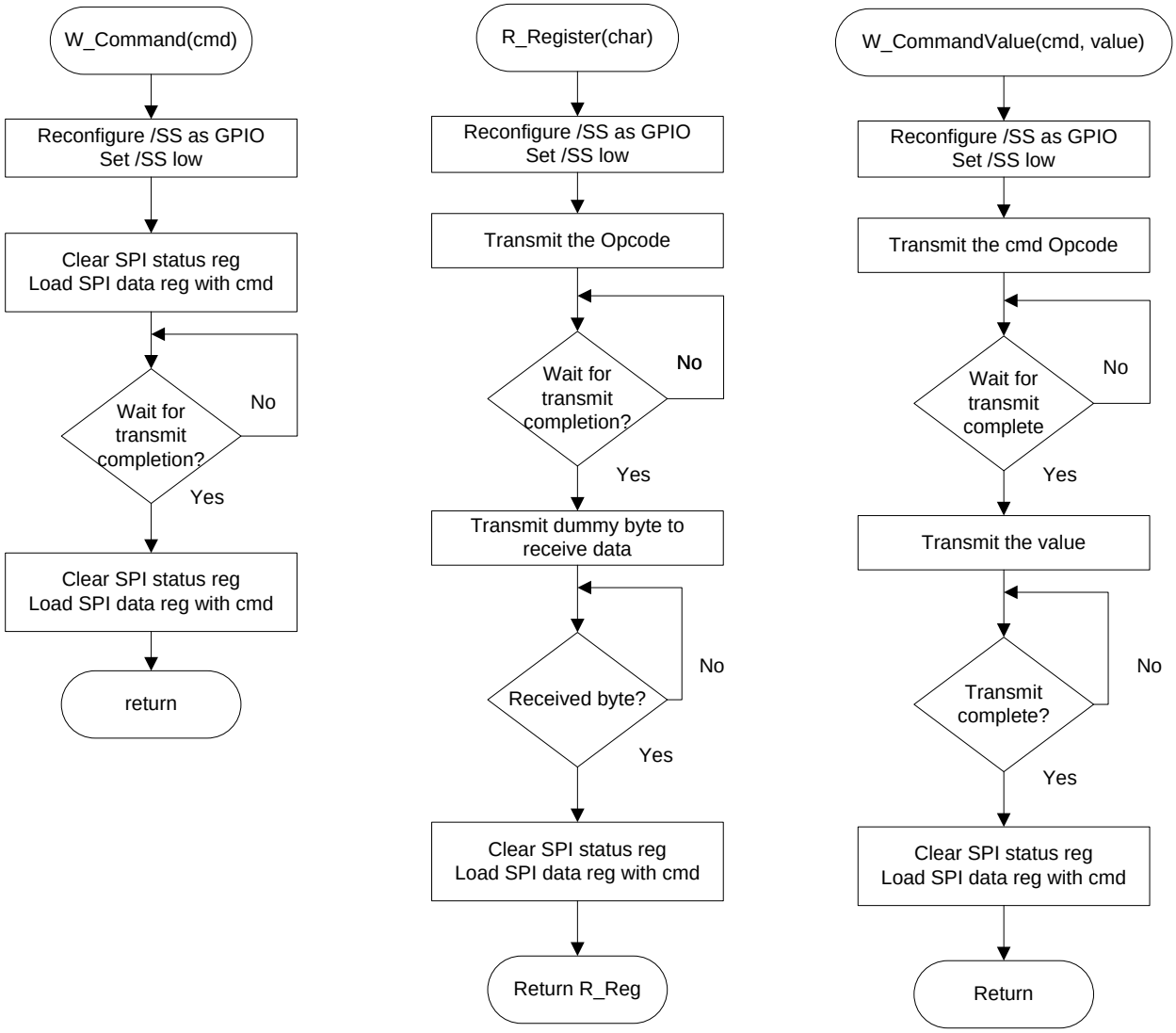
MCU Port Pin	HI-3717A	CONNECTOR
PP0	RFIFO	J10-2
PP1	MATCH	J10-14
PP2	ROVF	J9 10
PP3	TFIFO	J9-8
PP7	TEMPY	J10-20

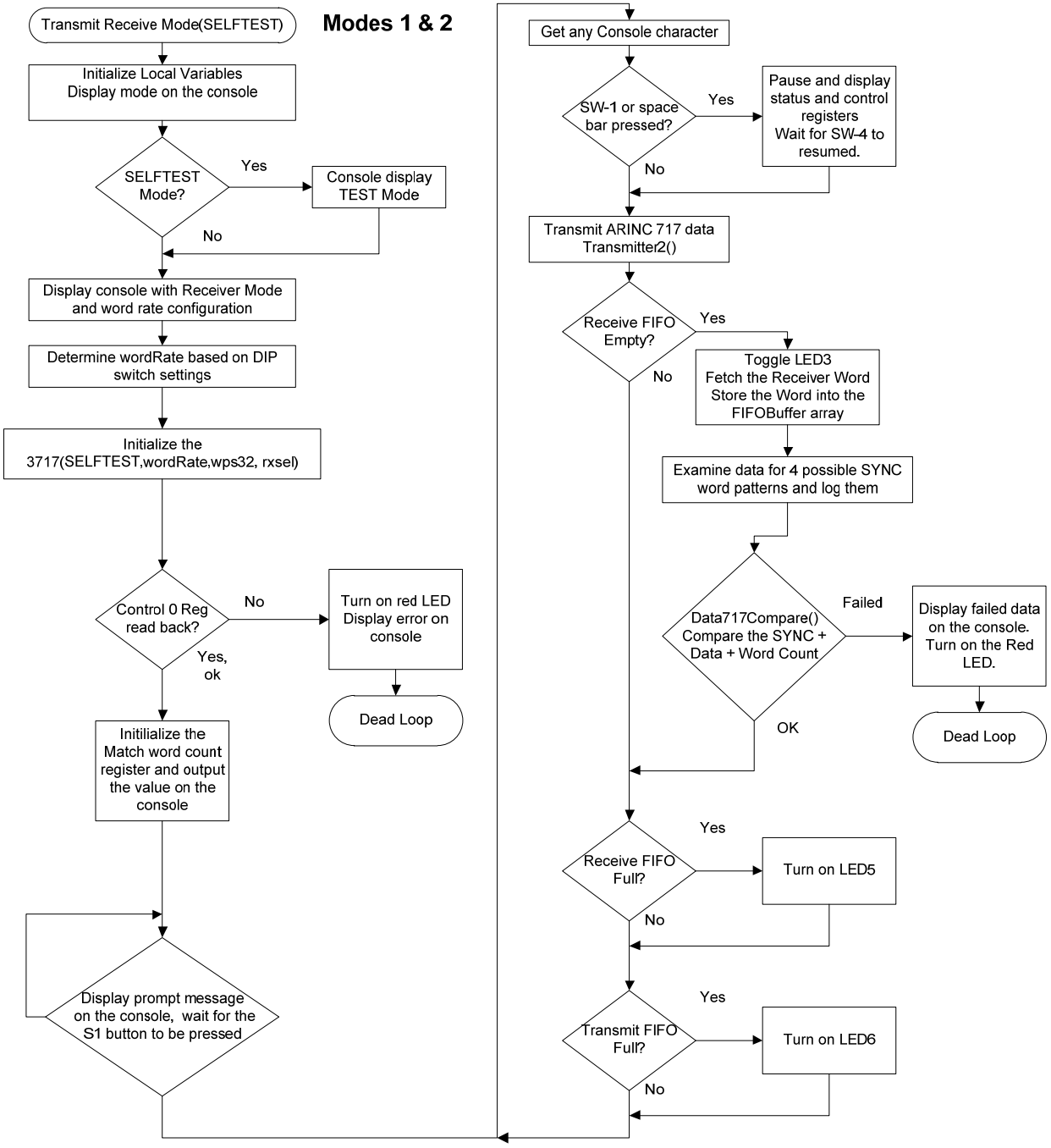
CAUTION: if code added to the interrupt handlers accesses the SPI interface, then any SPI access in the foreground code must be made atomic by first disabling interrupts then re-enabling interrupts after the SPI is accessed. Failure to protect foreground SPI access will most likely cause corrupted data on the SPI interface. There are some test instructions in the handlers which pulse the PP4 MCU pin (MISO2) J7-1 on the main board a number of times as a debugging tool. By viewing these pulses on an oscilloscope, the interrupt handler can be identified. This test code can be removed.



SPI Driver Functions

These three primitive SPI functions make up the basic read and write functions to access the SPI interface of the HI-3717A. There are slightly more complicated functions to perform multi-byte reads or writes which are basically derivatives of these three simpler functions. All HI-3717A SPI driver functions are included in the 3717Driver.c module and its 3717Driver.h header file. The MCU /SS pin is connected to the HI-3717A /CS pin.

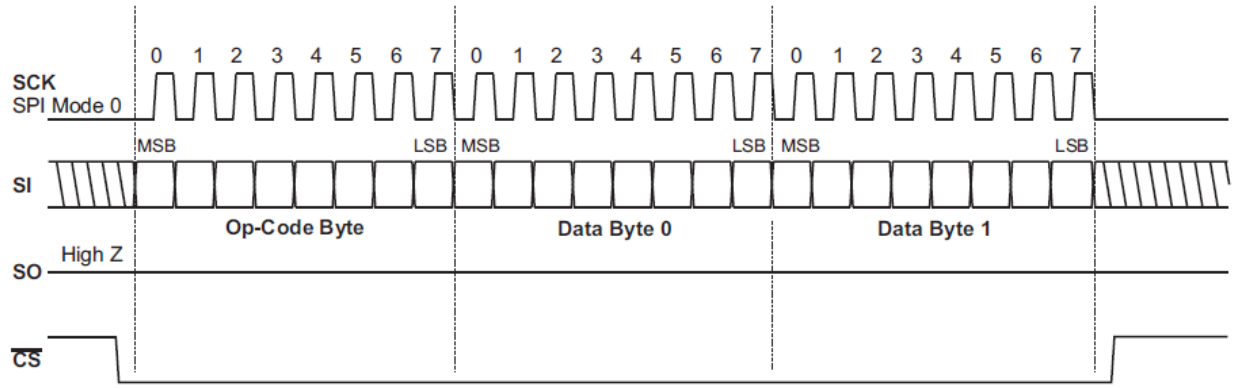



Transmit Receive Mode(SELFTEST)
Modes 1 & 2
Initialize Local Variables
Display mode on the console
SELFTEST Mode?
Yes
Console display TEST Mode
No
Display console with Receiver Mode and word rate configuration
Determine wordRate based on DIP switch settings
Initialize the 3717(SELFTEST,wordRate,wps32, rxsel)
Control 0 Reg read back?
No
Turn on red LED Display error on console
Dead Loop
Yes, ok
Initilialize the Match word count register and output the value on the console
Display prompt message on the console, wait for the S1 button to be pressed
Get any Console character
SW-1 or space bar pressed?
Yes
Pause and display status and control registers Wait for SW-4 to resumed.
No
Transmit ARINC 717 data Transmitter2()
Receive FIFO Empty?
Yes
Toggle LED3
Fetch the Receiver Word
Store the Word into the FIFOBuffer array
No
Examine data for 4 possible SYNC word patterns and log them
Data717Compare() Compare the SYNC + Data + Word Count
Failed
Display failed data on the console. Turn on the Red LED.
Dead Loop
OK
Receive FIFO Full?
Yes
Turn on LED5
No
Transmit FIFO Full?
Yes
Turn on LED6
No

Special handling of the /SS SPI signal:

All HI-3717A SPI Op-Codes require the /CS to remain low for the complete duration of the data transfer including multi-byte reads and writes. Refer to figures 3, 4 and 5 of the data sheet for timing diagram examples.

To achieve this, the default SPI slave select line /SS in the Freescale MCU must be reconfigured as a GPIO and controlled by code in the function. This technique is common for devices requiring the /CS line to remain low during multi-byte transfers. The first positive SCK edge must occur after /CS is asserted low; the last falling SCK edge must occur before the /CS is negated high as shown in the following diagram:



There are functions that read a single byte from the HI-3717A SPI port, write a command to the HI-3717A SPI port and a few others which read or write a command plus a multiple number of bytes. For example the function below is the basic function that writes a command plus one byte of data to the HI-3717A SPI port.

```
// Write SPI Command with a Value to HI-3717
void W_CommandValue (uint8 cmd, uint8 value){
  uint8 dummy;

  SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;   // disable auto /SS output, reset /SS Output
  SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK; // disable auto /SS output, reset SPI0 Mode
  SPIO_nSS = 0;                            // assert /SS low
  dummy = SPI0SR;                           // clear SPI status register
  SPI0DR = cmd;                             // SPI command
  while (!SPI0SR_SPIF);
  dummy = SPI0DR;                           // read Rx data in Data Reg to reset SPIF
  dummy = SPI0SR;                           // clear SPI status register
  SPI0DR = value;                           // Reset values
  while (!SPI0SR_SPIF);
  dummy = SPI0DR;                           // read Rx data in Data Reg to reset SPIF
  SPIO_nSS = 1;                            // assert /SS high

  SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;    // enable auto /SS output, set /SS Output
  SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;  // enable auto /SS output, set SPI0 Mode
Default
}
```

This function transmits a command byte followed by one 16-bit data word, writing one ARINC message to the FIFO.

```
        // ---------------------------------------------------------------
        // Transmits the Message Command and data contained in the passed array pointer
        // Transmit cmd + 16 bits for 2 byte transmit commands
        // Valid data range is 00-0XFFFF
        // ---------------------------------------------------------------
         void TransmitCommandAndData16(uint8 cmd, unsigned int TXBuffer)
         {
         uint8 static dummy;

           SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;         // disable auto /SS output, reset /SS
           SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;       // disable auto /SS output, reset SPI0
           SPIO_nSS = 0;                                // assert /SS low

           dummy = txrx8bits(cmd, 1);      // Transmit the whole message, ignore return values

            // Transmit upper 8 bits
            dummy = txrx8bits((unsigned char)((TXBuffer>>8) & 0xFF),1);
            // Transmit lower 8 bits
            dummy = txrx8bits((unsigned char)(TXBuffer & 0xFF),1);
          SPIO_nSS = 1;                              // assert /SS high

           SPI0CR1 |= SPI0CR1_SPE_MASK|SPI0CR1_SSOE_MASK;
           SPI0CR2 |= SPI0CR2_MODFEN_MASK;

        }
```

This example shows a very simple SPI driver function which issues one Opcode then reads back one byte. This is used for fetching single byte status' from the SPI.

```
        /* ---------------------------------------------------------------
        /  Read HI-3717 Register Read Function
        /  ---------------------------------------------------------------

        Argument(s):  Register to read

             Return:  8-bit Register Value
        */

          unsigned char R_Register (char Reg) {
          unsigned char R_Reg;

          SPI0CR1 = SPI0CR1 & ~SPI0CR1_SSOE_MASK;     // disable auto /SS output, reset /SS
          SPI0CR2 = SPI0CR2 & ~SPI0CR2_MODFEN_MASK;   // disable auto /SS output, reset SPI0 Mode
          R_Reg  = txrx8bits(Reg,1);                  // send op code (ignore returned data byte)
          R_Reg  = txrx8bits(0x00,1);                 // send dummy data / receive Status Reg
          SPI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;      // enable auto /SS output, set /SS Output
          SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;    // enable auto /SS output, set SPI0 Mode
        Fault
          return R_Reg;
        }
```

This read function fetches a 16 bit word from the FIFO into the array. The Op code is passed by the caller.

```
// -----------------------------------------------------
// This function reads the number of 16 bit words into the array by pointer
// -----------------------------------------------------
void FetchFIFOWordsWithCmd(uint8 cmd,uint8 count,unsigned int *passedArray)
{
  static uint8 WordCount;

  unsigned int dummy16;


   CS_HL();
   dummy16 = SPI0SR;         // clear SPI status register
   SPI0DR = cmd; //0xF6;     // Send the 3717 SPI command

   while (!SPI0SR_SPIF);   // wait for SPIF flag assertion
   dummy16 = SPI0DR;         // read/ignore Rx data in Data Reg, resets SPIF
   for (WordCount=0; WordCount<count; WordCount++) {
      dummy16 = (unsigned char)txrx8bits(0x00,1);  // fetch d16-d8 uppper word
      dummy16 = dummy16 << 8;                      // position to upper bits
      dummy16 |= (unsigned char)txrx8bits(0x00,1); // fetch d7-0 lower word and combine
      passedArray[WordCount] = dummy16;            // load word into array
     }
   SPIO_nSS = 1;                           // assert /SS high

   PI0CR1 = SPI0CR1 | SPI0CR1_SSOE_MASK;          // enable auto /SS output, set /SS
   SPI0CR2 = SPI0CR2 | SPI0CR2_MODFEN_MASK;        // enable auto /SS output, set SPI0
Mode Fault

}
```

The Init3717() function initializes the HI-3717A with three passed parameters (test, wordRate and rxsel) and uses these values to set the corresponding bits in the Control 1 and Control 0 Registers. Finally, the Control Register 0 is read back and checked for the expected value that was written. If the calling function does not receive this expected value, the program turns on the red LED, transmits an error message on the console and enters a dead loop.

```
/*
 Initialize the HI-3717

 RETURNS: 0 if successful, 0x0F if failed

 INPUT PARMS
  test: 1=test mode. 0=normal mode
  wordRate: 0-7 = words/sec per 3717 data sheet
  wps32: 1=32 WPS (overrides wordRate selections)
  rxsel: 1=BPRZ, 0=HBP

*/
// -------------------------------------------------------------------
uint8 Init3717(uint8 test,  uint8 wordRate, uint8 wps32, uint8 rxsel )
   {
   unsigned char cmd=0;

// Write control-1 register
   W_CommandValue(W_CTRL1, test);          // Initialize the test mode if parm set

// Write control-0 register last
   cmd =  rxsel;
```

```
   if(wps32)
      {
         cmd |= WPS32;
      } else
      {
         cmd |= slewRate[wordRate] << 1;    // Select slew rate based on bitrate
         cmd |= wordRate << 4;              // Select the actual word rate

      }

  // Initialize the Control 0 register
  // if parms are passed: WPS32, Slew Rate and word rate
   W_CommandValue(W_CTRL0, cmd);

  // Perform a check to see if writing the word to the 3717 is correct?
   if(R_Register (R_CTRL0) != cmd)      // Did 3717 receive write ok?
      return INIT_FAIL;                 // fail
   else
      return 0;                         // ok

   }
```

See the 3717Driver.h header file for the options available in the define statements.

Uart.c Serial Port (RS-232)

The drivers to support the serial port (Console) are contained in this module. There are some function drivers to allow messages to be sent and received on the UART. This is useful to log status or data messages on Windows HyperTerminal or any other terminal program. It currently uses polling to determine when the data receive or transmit registers can be read or written.

GPI and GP2

These two pins on the main board are connected to the HI-3717A to TFIFO and ROVF respectively.

LEDs LED1-LED8

These LEDs are controlled by a function in the program. LED1-LED4s and LED8 are active-low logic and LED5-LED7s are active-high logic. Using this support function allows a universal way to turn the LEDs on and off from the program. The Freescale MC9S12DT part uses pins PE5, PE6, PE7 for configuration sensing during reset, so the logic on these three pins needs to be reversed so the MCU sees a low at reset time.

```
// -------------------------------------------------------
// Control LED1 - LED8
// ledNumber: LED_1,LED_2,LED_3,LED_4....LED_8 [1-8]
// OnOff: 1=ON, 0=OFF
// -------------------------------------------------------
void LED_CTL(uint8 ledNumber, uint8 OnOff){
#if NEWBOARD
   if(ledNumber>4 && ledNumber<8)// LEDs 5-7 have reversed HW logic so invert
                                  these 3
#else
   if(ledNumber>4)          // Old board.
#endif
```

```
        OnOff = ~OnOff;
      switch (ledNumber){
        case 1:  LED1=OnOff;  break;
        case 2:  LED2=OnOff;  break;
        case 3:  LED3=OnOff;  break;
        case 4:  LED4=OnOff;  break;
        case 5:  LED5=OnOff;  break;
        case 6:  LED6=OnOff;  break;
        case 7:  LED7=OnOff;  break;
        case 8:  LED8=OnOff;  break;
        default: break;
      }
}
```

Usage examples:

```
LED_CTL(LED_1,OFF);         // turns off LED1
LED_CTL(LED_1,ON);          // turns on LED1
```

HI-3717 Demo CodeWarrior™ Software Project

The software project is built with Freescale CodeWarrior™ version 5.9.0 using the free, limited 32K version. The demo program size is approximately 7-8K. The main functions are in main.c and the low level HI-3717A drivers are in the 3717Driver.c file. The software project "HI-3717 Demo xx" will normally be distributed in a zip file on a CD-ROM with the same name. **To develop, debug and download this software into the board, a debug cable is necessary. It is not provided in this kit.** To purchase this cable, go to the PE Micro website or purchase it from Digi-Key.

Project Files

Source Files

main.c	Main code
3717Driver.C	Low level SPI drivers for the HI-3717A
Peripherals.c	GPIO, PLL frequency setup and SPI configuration
BoardTest.c	Board Test functions
Uart.c	Low-level UART drivers
datapage.c	Freescale™ IDE support file

Include Files

Main.h	
3717Driver.h	Low level HI-3717 SPI driver header file
Peripherals.h	
BoardTest.H	
Uart.h	Uart header file
Common.h	Common defines for the project
Derivative.h	Freescale™ IDE support file
Mc9s12xdt512.h	Freescale™ IDE target part support file

CodeWarrior™ and Software Project Setup:

1. Download and install the CodeWarrior™ IDE from the Freescale website. The download links are provided below.

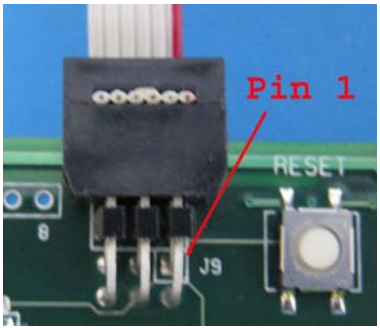
2. Unzip the HI-3717 zip file into the directory you plan to use for your project.

3. Navigate to the HI-3717 project folder and double click the HI-3717 Demo.mcp project file to launch this project with CodeWarrior. The IDE should open with the project files on the left side of the window.

4. Click Make from the Project menu to rebuild the project. The project should build without errors. Install the PE Micro USB Multilink Interface cable per the PE Micro instructions.

5. Plug the USB Multilink 6-pin debug cable into the J9 debug connector and power up the board with 3.3V.

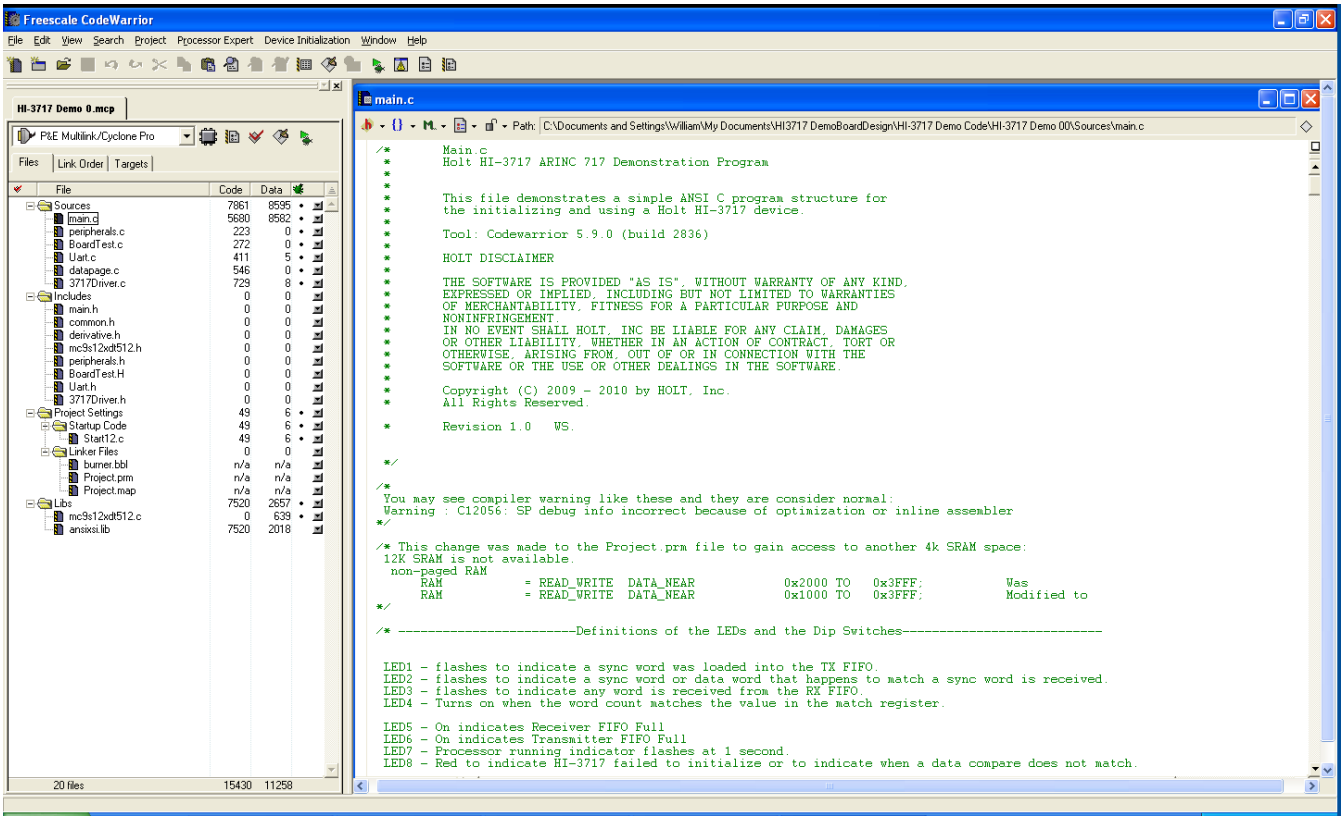
   Plug the P&E debugger ribbon cable with the 6-pin connector to the evaluation board observing the location of pin-1 indicated by a Red wire. Align the Red side of the connector so it connects to the pin-1 side of the connector on the board. Pin-1 is near the J9 silk-screen on the board and is also noted with a small white square.



6. Download the program by clicking Debug from the Project menu. The first time the program is downloaded, the debugger needs to be configured for the USB Multilink cable. After downloading is complete, the debugger window should be displayed with the first line in main.c highlighted. Press the green arrow button to run the program. Since the program has been loaded, you can power down the board and re-power the board; the program should run automatically without the debugger.

7. Note: All program console output references to HI-3717 apply to HI-3717A in this version.

Holt HI-3717 project loaded into CodeWarrior™ 5.9.0



Freescale™ MC9S12XDT512xxx Development Tools

The Freescale™ microcontroller data sheet and other documentation can be found at this link:

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=S12XD&tid=16bhp

If these links become out of date go to: http://www.freescale.com/ and search for information on "S12XD: 16-Bit Automotive Microcontroller".

A Free 32K limited version of the CodeWarrior™ IDE from Freescale™ is available:

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=CW-HCS12X&fsrch=1

The US Multilink debugger cable used for this project is:

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=USBMULTILINKBDM&parentCode=S12XD&fpsp=1

http://search.digikey.com/scripts/DkSearch/dksus.dll?Detail&name=USBMULTILINKBDME-ND

Summary

This Software Application Note provids an overview of the demo software provided in the HI-3717 evaluation kit. The complete Codewarrior™ software project is contained on the CD-ROM. To use the demo program, refer to the AN-170 Users Guide or the Quick Start Guide contained on the CD-ROM.

REVISION HISTORY

P/N	Rev	Date	Description of Change
AN-171	NEW	08/18/11	Initial Release
	A	1/8/14	Added Mode 6. New debugger connector photo.
	B	3/12/15	Change some HI-3717 references to HI-3717A.