1 Trademarks

Using the SPI Library for TMS37157

Stefan Recknagel

ABSTRACT

This document describes the SPI Commands that are necessary to control the TMS37157 passive low-frequency interface (PaLFI) device and access its EEPROM. The functions of the SPI library are in C and can be easily called from any C program that includes the header files. The header files are optimized for the MSP430F2274 and can be easily adapted to other microcontrollers. The SPI library does not require detailed knowledge of SPI, because the details of communication handling are integrated in the functions. The sample code described in this document can be downloaded from www.ti.com/lit/zip/SWRA272.

Contents

2 Introduction

The MSP430F2274 communicates with the TMS37157 through a 3-wire SPI interface. For efficient communication, the busy port of the TMS37157 must be connected to one input port of the MSP430 microcontroller. This library requires the busy port of the TMS37157 connected to one input of the MSP430F2274 and the push port connected to one output port of the MSP430F2274

This application report includes library files that are intended to be compiled in larger projects. These files include all SPI functions that are described in the users guide of the TMS37157; they are identical to the header files used for the eZ430-PaLFI demo tool firmware.

3 Hardware Connections Between MSP430 and TMS37157

Figure 1 shows the necessary connections between the MSP430 and the TMS37157 front end to work together with the SPI library. The port pins of MSP430 can be changed in the PaLFI_Transponder.h file. If different ports are used (for example, P1 instead of P2) additional changes must be made in SPI_Stack.c and SPI_LowLevel.c (for example, P2IN replaced by P1IN), but these changes are trivial.

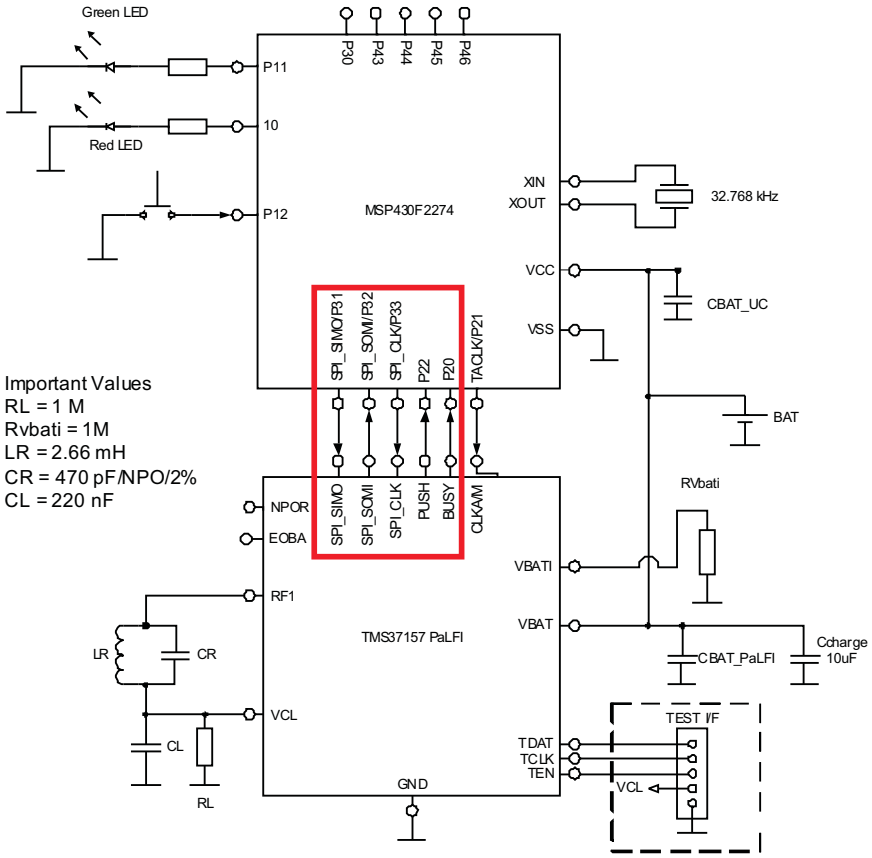

Figure 1. SPI Connection, MSP430 to TMS37157

4 SPI Library - Use from C

This code example shows how to use the SPI library of the TMS37157 for a memory read and program command.

```
#include "msp430x22x4.h"
#include "PaLFI_Transponder.h

void main (void)
{
   unsigned char ucPageData[5] = {0{;
   Wake_PaLFI();
   SPI_Read_PCU_State();
      SPI_Read_SerialNum();
   SPI_Read_UserPage(Page9,ucPageData);
   SPI_Program_UserPage(Page10,ucPageData);
   SPI_Power_Down();
}
```

5 Included Library Files

5.1 PaLFI_Transponder.h

This file contains the definitions for the function and variables used within the SPI library. This file must be adapted if connections between TMS37157 and MSP430F2274 other that those shown in Figure 1 are used.

5.2 SPI_LowLevel.c

This file is for the low level communication between the TMS37157 and MSP430F2274. This file must be adapted if other connections between TMS37157 and MSP430F2274 other that those shown in Figure 1 are used.

5.3 SPI_Stack.c

This file includes the functions for the transponder access commands and the enhanced commands.

6 Function Description

Functions that are only used internally by other functions are not described. Initialization functions, transponder access functions, and enhanced command functions are described.

MSP_430_SPI_Init (void)— This function initializes the SPI interface of the MSP430F2274. If another microcontroller is used, the function may be adapted. CLK_AM is not necessary for SPI communication, but it is necessary for trimming. If trimming is not used, CLK_AM connections can be deleted.

Wake_PaLFI (void)— This function wakes the TMS37157 from standby mode (MSP430F2274 toggles push) and returns when the TMS37157 is ready to receive data (indicated by busy going from high to low).

NOTE: This function operates correctly only if the MSP430F2274 is connected to the push port of the TMS37157 and is not powered by VBATI of the TMS37157.

SPI_Read_SerialNum (void)— This function reads the serial number, Page2 and Page1 from the TMS37157. It saves the data in TRP_Data.

SPI_Read_PCU_State (void)— This function reads the PCUsState out of the TMS37157. The PCU state defines if the TMS37157 woke up because of a push interrupt or an MSP access command. If state is 0x01 a push interrupt occurred, if state is 0x02 a MSP access interrupt occurred. The state is saved in TRP_Data.ucPCU_Mode.

SPI_Read_UserPage (unsigned char ucPage, unsigned char *ucData)— This function reads one user page (ucPage) out of the EEPROM of the TMS37157. The intrinsic values for ucPage are saved in “PALFI_Transponder.h”. The content of the user page is saved in *ucData.

SPI_Program_UserPage (unsigned char ucPage, unsigned char *ucData)— This functions programs *ucData in one page (ucPage) in the EEPROM of the TMS37157. The intrinsic values for ucPage are saved in “PALFI_Transponder.h”. The content of the user page is saved in *ucData.

SPI_Read_CU_Data (unsigned char *ucData)— This function should be performed if an MSP access is received by the TMS37157. It receives 6 bytes of data from the front end and saves it to *ucData. The TMS37157 waits for an answer from the MSP430 (SPI_Write_CU_Data).

SPI_Write_CU_Data (unsigned char *ucData)— This function can only be performed after a SPI_Read_CU_Data command. It sends 6 bytes to the TMS37157 (*ucData), which forwards this data through its LF interface. The TMS37157 shuts down after sending out the data.

NOTE: The data for a program page command is always 5 bytes (except Page 1 and Page 2). The data for an MSP430F2274 access command is always 6 bytes. This must be considered when handing data to the functions.

SPI_CLKA_ON (void)— This function turns on the LC tank circuit of the TMS37157 and switches it out to the CLK_AM pin. It introduces a delay for the frequency to stabilize. This is useful for trimming the device.

SPI_CLKA_OFF (void)— This function turns off the LC tank circuit and the output to CLK_AM.

SPI_TRIM_WO_PROG (char data)— This function programs the trim capacitors of the TMS37157, but it does not program the trim EEPROM. A reset of the TMS37157 restores the values from the EEPROM to the trim capacitors.

SPI_TRIM_W_PROG (char data)— This function programs the trim capacitors and the trim EEPROM of the TMS37157.

SPI_CRC_Calc (….)— This function requests a CRC16 calculation performed by the TMS37157 with the start value 0x3791. ucStart defines which byte of *ucData is the first byte for the CRC calculation. ucLength defines on how many bytes (including ucStart) the CRC is calculated. The CRC is saved in *ucCRC.

SPI_Power_Down (void)— This function turns off the TMS37157. To access the memory of TMS37157, it must be turned on.

7 MSP Access Commands

The MSP access commands are special cases—they work only if the TMS37157 receives an MSP access command through its RF interface. The MSP access commands are used to transfer data through the RF interface directly to the microcontroller and back. In the default application, the MSP430F2274 is in LPM4 waiting for an interrupt and the TMS37157 is in standby mode, resulting in overall ultra-low power consumption. If the TMS37157 receives an MSP access command, it sets busy high. This can be used as an interrupt for the MSP430F2274. The TMS37157 shows its readiness by resetting busy. Now the MSP430F2274 can request the data from the TMS37157. The TMS37157 waits until the MSP430F2274 sends 6 bytes of data back to the TMS37157. During this time, the field of the RFID reader must stay on, supplying the TMS37157 with energy. The TMS37157 sends the data back to the RFID reader when the RFID reader switches off the field. The following code example shows how to use the MSP access commands in connection with a busy interrupt. It is assumed that busy is connected to P2.1 of the MSP430F2274.

```
#include "msp430x22x4.h"
#include "PaLFI_Transponder.h"

void main (void)
{

   unsigned char MSP_Access_Data[6] = {0};

   P2OUT = 0; //
   P2DIR &= ~CU_BUSY; // Busy Input P2.1 CU_BUSY = 0x002
   P2IFG &= ~CU_BUSY; // reset busy Interrupt
   P2IE |= CU_BUSY; // busy Interrupt enabled

   While(1)
   {
      if((P2IFG & CU_BUSY) == CU_BUSY) // Test for Interrupt
      {
         While ((P2IN & CU_BUSY) == CU_BUSY); // wait until TMS37157 ready
         SPI_Read_CU_Data(MSP_Access_Data); // read Data from TMS37157
         MSP_Access_Data[1] = MSP_Access_Data[2] + MSP_Access_Data[3]; // change data
         SPI_Write_CU_Data(MSP_Access_Data); // Write Data to TMS37157
         P2IFG &= ~CU_BUSY; // reset Interrupt Flag
         P2IE |= CU_BUSY; // set Interrupt enabled
      }
      __bis_SR_register(LPM4_bits + GIE); // Enter LPM4, global Interrupts Enabled
   }
}

#pragma vector=PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
   P2IE &= ~CU_BUSY;
   __bic_SR_register_on_exit(LPM4_bits+GIE);
}
```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265