University of Technology, Sydney

Faculty of Engineering and Information Technology

Subject:	**48434 Embedded Software**
Assessment Number:	**4**
Assessment Title:	**Lab 4 – SPI and ADC**
Tutorial Group:	

Students Name(s) and Number(s)

Student Number	Family Name	First Name

Declaration of Originality:

The work contained in this assignment, other than that specifically attributed to another source, is that of the author(s). It is recognised that, should this declaration be found to be false, disciplinary action could be taken and the assignments of all students involved will be given zero marks. In the statement below, I have indicated the extent to which I have collaborated with other students, whom I have named.

Statement of Collaboration:

Signature(s)

key

Marks

Opening comments / function descriptions / naming conventions / code structure	/0.5
SPI HAL	/2
Analog sampling	/3
Analog filtering	/1.5
Protocol implementation	/1
TOTAL	/8

Office use only ☺

Assessment Submission Receipt

Assessment Title:	**Lab 4 – SPI and ADC**	**Mark**
Student’s Name:		
Date Submitted:		
Tutor Signature:		Office use only ☺

Lab 4 – SPI and ADC

Serial peripheral interface. Analog-to-digital conversion.

Introduction

The Serial Peripheral Interface (SPI) is a popular 3-wire interface that is used for synchronous high-speed serial communication with local peripherals. Many peripherals, such as analog-to-digital converters, digital-to-analog converters, Flash memory, real-time clocks, temperature monitors, etc. come with built-in SPI interfaces, making it easy for them to connect to a variety of microcontrollers.

An analog-to-digital converter (or ADC) is used to quantize an external analog signal so as to represent it digitally. If the samples of an analog signal are taken at a sufficiently high rate, then the samples furnish enough information for the analog signal to be reconstructed exactly. Once the analog signal has been converted to a digital form, it can be filtered, manipulated, and processed. The processed signal can then be converted back to an analog signal through the use of a digital-to-analog converter (DAC).

Objectives

1. To implement a hardware abstraction layer for the Serial Peripheral Interface.
2. To use an analog-to-digital converter to acquire an analog signal.
3. To perform simple signal processing on a signal.
4. To expand the implementation of the ModCon serial protocol.

L4.2

Equipment

- 1 ModCon microcontroller board – UTS
- 1 ModCon analog interface board – UTS
- 1 HCS12 USB programmer / debugger – PEMicro
- 2 USB cables
- 1 5V DC power supply, 1 12V DC power supply, PC connector
- Freescale CodeWarrior IDE for the MC9S12
- 1 function generator

Safety

Cat. A lab

This is a Category A laboratory experiment. Please adhere to the Category A safety guidelines (issued separately).

Critical Sections

Critical sections are handled differently in this lab compared to Lab 3.

A better way of implementing critical sections that preserves the stack and allows subroutines to be called from within a critical section is by declaring the following macros:

```
#define EnterCritical() { asm tfr ccr,a; asm staa savedCCR; asm sei; }
#define ExitCritical()  { asm ldaa savedCCR; asm tfr a,ccr; }
```

You need to declare a local variable called `savedCCR` in the function that contains the critical section, before using these macros, as shown in the example below:

```
void function(void)
{
  unsigned char savedCCR;
   ...
  EnterCritical(); // make atomic, entering critical section
  // we have exclusive access to global variables
  ...
  ExitCritical(); // end critical section
}
```

Since the compiler has now reserved space for the `savedCCR` on the stack, it is safe to call functions (either implicitly or explicitly) within the critical section.

Software Requirements

1. The software is to incorporate all the features of Lab 3.
2. The real-time clock must send a packet to the PC every second, regardless of the debug mode.
3. The main program should use the modulus down counter to periodically take analog samples every 10 ms.
4. A hardware abstraction layer (HAL) is to be written for the Serial Peripheral Interface (SPI) module. The HAL should support the setting up of the SPI module for various modes of operation, including: master / slave; active low / high clock; even / odd edge clocking; LSB / MSB first; and baud rate. It should also provide an interface for reading a desired analog conversion result from a specific channel.

   The SPI is to be set up with the following parameters:

SPI Parameter	**Value**
Master / Slave	Master
Active low / high clock	High
Even / odd edge clocking	Even
LSB / MSB first	MSB
Baud rate	1 Mbit/s

   For reasons of speed (the SPI operates at 1 Mbps), it is not appropriate to implement FIFO buffers to communicate with the SPI hardware. It is much faster for a background thread to wait for a SPI operation to finish. Therefore, you do **NOT** need to implement a SPI interrupt service routine.

5. The analog-to-digital conversion is to be handled by the external analog interface board, **NOT** the analog-to-digital (ATD) converter internal to the MC9S12. The external ADC communicates with the MC9S12 via a SPI interface.

   You will need to refer to the ModCon and analog interface board schematics to determine the SPI channel and the appropriate "SPI Chip Select" (`SPICSn`) signal to use to interface to the ADC, and to determine appropriate ports and pins to manipulate on the MC9S12.

   An analog-to-digital conversion (ADC) software module should be written that uses the SPI to initiate and read an ADC conversion result. In addition, the module should filter the returned conversion result, using a median filter of the last three conversion results, i.e. a "sliding window" of the three most recent samples.

   The datasheet for the ADC is readily available on the Internet.

   The ADC is used in "single-ended" mode. Also note that the hardware "inverts" the incoming signal, so to convert the ADC conversion result to a bipolar value you will have to do something like:

   ```
   analogInputValue = ADC_OFFSET - (INT16)conversionResult;
   ```

   Only Channels 1 & 2 of the ModCon Analog Interface will be used.

   Note that the Modcon Analog Interface (the front panel with the 4mm sockets) labels the analog channels 1-8 whilst the ADC chip labels its inputs 0-7. You can check the schematic to see that a sensible approach has been taken in connecting and labelling the channels.

6. In the main program, two different modes of communication are to be implemented: synchronous and asynchronous.

   - **Asynchronous communication is the default mode** where the ModCon responds to commands sent from the PC. In addition, the ModCon should now initiate the sending of data packets to the PC when any of its analog values changes. It is important to only send data packets to the PC when a change in analog value occurs, rather than continuously, so that the PC is not bombarded with extraneous packets. Therefore, at **<u>intervals of 10 ms</u>**, the ModCon will send an analog value packet **<u>only if</u>** the analog value has changed.
   - In synchronous mode, analog values from Channels 1 & 2 should be sent to the PC at **<u>intervals of 10 ms,</u>** regardless of whether the analog values **<u>have changed or not</u>**. **The ModCon should still respond to packets sent from the PC whilst in this mode.**

7. Extra commands of the ModCon serial protocol to be implemented are:

ModCon to PC	PC to ModCon
0x0A Protocol – Mode	0x0A Protocol – Mode
0x50 Analog Input – Value	

8. On startup, or in response to reception of a “0x04 Special –Startup” packet from the PC, the ModCon should send, in addition to all other “startup” packets:
    - a “0x0A – Protocol – Mode” packet
9. The ModCon PC Interface has a tab specifically for Lab 4. It graphs the values sent back by the ModCon. You should use this to verify that your software is working correctly.
10. TortoiseSVN must be used for version control.

Marking

The software should be ready for marking on the date specified in the Timetable in the Learning Guide.

Software marking will be carried out in the laboratory, in the format of an oral exam.

Marking criteria are on the front page. Also refer to the document “Software Style Guide”.

Software Specification

The following header files are suggested, but not mandatory.

SPI.h

```
// ----------------------------------------
// Filename: SPI.h
// Description: I/O routines for MC9S12 serial
//   peripheral interface
// Author: (your name)
// Date: (the date)

#ifndef SPI_H
#define SPI_H

// new types
#include "types.h"

typedef struct
{
  BOOL isMaster;
  BOOL activeLowClock;
  BOOL evenEdgeClock;
  BOOL LSBFirst;
  UINT32 baudRate;
} TSPISetup;

// ----------------------------------------
// SPI_Setup
//
// Sets up the Serial Peripheral Interface
// Input:
//   aSPISetup is a structure containing the parameters to
//     be used in setting up the SPI:
//       isMaster is a Boolean value indicating master or slave
//       activeLowClocks is a Boolean value indicating whether the
//         clock is active low or active high
//       evenEdgeClockPhase is a Boolean value indicating whether the
//         data is clocked on even or odd edges
//       LSBFirst is a Boolean value indicating whether the data is
//         transferred LSB first or MSB first
//       baudRate is the baud rate in bits/sec of the SPI clock
//   busClk is the bus clock rate in Hz
// Output:
//   none
// Conditions:
//   none
// ----------------------------------------

void SPI_Setup(const TSPISetup* const aSPISetup, const UINT32 busClk);

// ----------------------------------------
// SPI_ExchangeChar
//
// Transmits a byte and retrieves a received byte from the SPI
// Input:
//   dataTx is a byte to transmit
//   dataRx is a pointer to a byte to receive
// Output:
//   none
// Conditions:
//   Assumes SPI has been set up

void SPI_ExchangeChar(const UINT8 dataTx, UINT8 * const dataRx);

#endif
```

L4.8

analog.h

```c
// ----------------------------------------
// Filename: analog.h
// Description: Routines for setting up and
//   reading from the ADC/DAC
// Author: (your name)
// Date: (the date)

#ifndef ANALOG_H
#define ANALOG_H

// new types
#include "types.h"

// Maximum number of channels
#define NB_INPUT_CHANNELS 8
#define NB_OUTPUT_CHANNELS 4

// Number of channels used
extern TUINT16 NbAnalogInputs;
extern TUINT16 NbAnalogOutputs;

typedef enum
{
  Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7, Ch8
} TChannelNb;

typedef struct
{
  TINT16 Value, OldValue;
  INT16 Value1, Value2, Value3;  // variables for median filtering
} TAnalogInput;

extern TAnalogInput Analog_Input[NB_INPUT_CHANNELS];

// ----------------------------------------
// Analog_Setup
//
// Sets up the ADC and DAC
// Input:
//   none
// Output:
//   none
// Conditions:
//   none

void Analog_Setup(const UINT32 busClk);

// ----------------------------------------
// Analog_Get
//
// Gets an analog input channel's value based on the mode
// Input:
//   channelNb is the number of the anlog input channel to read
// Output:
//   a Boolean value indicating whether the output was set successfully
// Conditions:
//   Assumes that the ADC has been set up

BOOL Analog_Get(const UINT8 channelNb);

#endif
```