Lab 4 – SPI and ADC

Serial peripheral interface. Analog-to-digital converter.

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 (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).

The LTC1859 is an 8-channel, low power, 16-bit ADC that can sample at up to 100 kilosamples per second (ksps). It can be software-programmed for different input voltage ranges. The internal 8-channel multiplexer can be programmed for single-ended inputs or pairs of differential inputs or combinations of both. The LTC1859 communicates with microprocessors via a SPI interface.

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 Tower serial protocol.

L4.2

Equipment

- 1 TWR-K70F120M-KIT – UTS
- 1 TWR-ADCDAC-LTC board – UTS
- 2 USB cables – UTS
- 1 function generator – UTS
- 1 oscilloscope - UTS
- NXP Kinetis Design Studio

Safety

Cat. A lab

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

Software Requirements

1. The software is to incorporate all the features of Lab 3. You should also implement or update the public functions and variables, as "advertised" in the supplied header files.

2. 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; frame size; and baud rate.

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

SPI Parameter	Value
Master / Slave	Master
Continuous clock	false
Clock polarity	Inactive low
Clock phase	Data is captured on leading edge
LSB / MSB first	MSB
Frame size	16 bits
Baud rate	1 Mbit/s

   For reasons of speed (the SPI operates at 1 Mbps), in this application there is no need to utilise FIFO buffers to communicate with the SPI hardware. Also, it is acceptable for a thread to wait for a SPI operation to finish. Therefore, you do **NOT** need to implement a SPI interrupt service routine.

3. The analog-to-digital conversion is to be handled by the external Tower TWR-ADCDAC-LTC board, **NOT** the analog-to-digital converter (ADC) internal to the K70. The external ADC communicates with the K70 via a SPI interface.

Refer to the Tower schematics (TWRADCDACLTCSCH.pdf) to determine how the LTC1859 has been set up, and to determine appropriate ports and pins to manipulate on the K70.

Jumpers on the TWR-ADCDAC-LTC board have been set up so that:

1. The TWR-K70 Board uses SPI2 from PTD11-15 which goes to the elevator SPI0.
2. The elevator SPI0 becomes the TWR-ADCDAC-LTC SPI0.
3. Jumper J14 has been set to 1-2 to set SPI CS encoding Bit 2 to 3V3.

An "analog" software module should be written that uses the SPI to initiate and read an ADC conversion result from the LTC1859. The ADC is to be used in single-ended mode, with a voltage range of ±10 V.

The analog module should capture a "sliding window" of sample values and provide support to filter the sampled values using a median filter.

The number of samples in the "sliding window" is a constant set by the application – we will choose a **size of 5**. The implementation of the "sliding window" should be optimised for speed (i.e. minimize the CPU time taken to implement).

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

Only Channels 0 & 1 of the LTC1859 will be used.

4. A median filter module should be written that can find the median of an array of signed 16-bit numbers, of a length up to 1024. The median is the "middle score", which involves sorting the array of numbers into ascending (or descending) order, and then choosing the middle number. If the length of the array is even, then the average of the two middle sorted values is taken.

   The sorting of the array is to be efficient. Note that a "bubble sort" is not efficient for large arrays.

5. In the main program, two different modes of communicating analog information via packets are to be implemented: synchronous and asynchronous (the default mode).

    - **Asynchronous mode:** the Tower should 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 **intervals of 10 ms**, the Tower will send an analog value packet **only if** the analog value has changed.
    - In synchronous mode, analog values from Channels 0 & 1 should be sent to the PC at **intervals of 10 ms,** regardless of whether the analog values **have changed or not**.

    The Tower should still respond to packets sent from the PC whilst in either mode.

6. The main application will take analog samples at **intervals of 10 ms**.

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

Tower to PC	PC to Tower
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 Tower should send, in addition to all other "startup" packets:

    - a "0x0A – Protocol – Mode" packet

9. The Tower PC Interface has a tab specifically for Lab 4. It graphs the values sent back by the Tower. You should use this to verify that your software is working correctly.

10. Git must be used for version control. Note that version control will be assessed in Lab 5 based on the development of the software from Lab 1 through to Lab 5.

Hints

1. To initialise the SPI module, you will need to:

   (i) Enable the clock gates to the SPI2, Port D and Port E modules.
   (ii) Set up the Module Configuration Register (MCR).
   (iii) Set up a Control and Transfer Attributes Register (CTAR).
   (iv) Initialise values on any chip select pins.

2. In setting `MCR` in this application, the following fields are fixed:

DCONF	00
FRZ	1
MTFE	0
PCSSE	0
ROOE	0
PCSIS	000001
DOZE	0
MDIS	0
DIS_TXF	1
DIS_RXF	1

   Other bits may be set by the user / module as necessary.

3. The SPI module is very flexible and can "talk" in many different ways – it uses a CTAR to define different transfer attributes. In our application we are using `CTAR0` only. It is acceptable to set up this register once in `SPI_Init()`. In setting `CTAR0` in this application, the following fields are fixed:

DBR	0
FMSZ	15
PCSSCK	00
PASC	00
PDT	00
CSSCK	0000
ASC	0000
DT	0000

   Other bits may be set by the user / module as necessary.

4. To set the baud rate (i.e. the bits per second), see section 54.3.3 of the K70 Reference Manual. For a list of values corresponding to each `PBR` setting, see the field descriptions on p.1817. For a list of values corresponding to each `BR` setting, see Table 54-38. Do an exhaustive search to find an achievable baud rate that is closest to the requested baud rate.

5. To transmit SPI data, you need to perform a 32-bit write into the PUSHR register. The top half-word contains command information, such as what the peripheral chip select signal should be doing, which CTAR to use, etc. For this application, these fields should be set as follows:

CONT	0
CTAS	000
EOQ	0
CTCNT	0
Reserved	0000
PCS	000001

   SPI2 in the K70 has two Peripheral Chip Select signals, called `SPI_PCS[1:0]` (see Table 3-78). The hardware jumpers on the TWR-ADCDAC-LTC are set up to only utilise `PCS[0]`. As the table shows, `CONT` and `PCS` need to be set in the top half-word of the `SPI2_PUSHR` register when you are transmitting data.

6. An example SPI timing diagram is given in Figure 6b of the LTC1859 datasheet. This is what the SPI hardware will do for you when you transmit/receive data (once it is set up correctly). You will need to set up a "Delay After Transfer" of 5 μs to allow the ADC to complete the conversion before interrogating the ADC for the result. See the K70 reference manual for details, specifically the CTAR fields `PDT` and `DT`. You should do an "exhaustive search" to achieve a delay that achieves the smallest **positive** error between actual and desired delay.

Note that the K70 reference manual has an error in it. The reference manual refers to the protocol clock period, with symbol $1/f_p$, when in fact it should refer to the module clock.

7. The `SPI_SR_TFFF` bit is used to determine whether the SPI bus is idle. You should check this before you initiate a communication, and clear it straight after (by writing a 1 to it). The `SPI_SR_RFDF` bit is used to indicate that the hardware RX FIFO has entries that need to be removed. You should use this to wait for a SPI transaction to complete, and then clear it (by writing a 1 to it).

8. Note that if the Receive FIFO Overflow Overwrite Enable (ROOE) bit is clear in the MCR (as it is for our setup), then you will need to **always** read the `SPI_POPR` register after each SPI transaction – otherwise no new data will be written into the Receive FIFO Registers. This will need to be done even if the `dataRx` parameter is `NULL`.

L4.10

Marking

1. **The software to be assessed must reside in the remote git repository before the start of your timetabled activity on the date specified in the Timetable in the Learning Guide.**
2. **Please create a "tag" called "Lab4Submission" (no spaces allowed) to the particular commit that you want marked. Markers will then create a branch from this tag called "Lab4Marking" in order to assess it.**
3. **Software marking will be carried out in the laboratory, in the format of a code review.**
4. **Refer to the document "Software Style Guide" for more details of some of the assessment criteria.**

Assessment Criteria

Your lab will be assessed according to the following criteria:

Item	Detail	Evaluation	Mark
Opening comments / function descriptions	File headers and function descriptions are correct.	**E G A P N**	**/0.5**
Naming conventions / code structure	Names and code structure conform to the Software Style Guide.	**E G A P N**	**/0.5**
Doxygen comments	Comments for all functions, variables and modules.	**E G A P N**	**/0.5**
SPI HAL	Public and private functions.	**E G A P N**	**/3**
Analog HAL	Public and private functions.	**E G A P N**	**/1.5**
Median filtering	Efficient implementation.	**E G A P N**	**/1**
Application	Architecture and implementation. Protocol expanded.	**E G A P N**	**/1**
		TOTAL	**/8**

L4.12

When we evaluate an assessment item, we will use the following criteria:

Evaluation	Mark (%)	Description
Excellent	100	**All relevant material is presented in a logical manner showing clear understanding, and sound reasoning.** For software – correct coding style, correct software architecture including: modularity; functions; parameters; and types, very efficient implementation (code and time) and/or novel (and correct) code.
Good	75	**Nearly all relevant material is presented with good organisation and understanding.** For software – mostly correct coding style, mostly correct software architecture including: modularity; functions; parameters; and types, reasonably efficient implementation (code and time).
Acceptable	50	**Most relevant material is presented with acceptable organisation and understanding.** For software – inconsistent coding style, reasonable software architecture (but could show improvement in modularity, use of functions, parameters, or types), some code may be prone to errors under certain operating conditions (e.g. input parameters) or usage, occasional inefficient or incorrect code.
Poor	25	**Little relevant material is presented and/or poor organisation or understanding.** For software – Conceptual difficulty of the underlying concepts, numerous coding style errors, functionality missing, poor software architecture, inappropriate or incorrect use of functions, parameters or types. Very inefficient and / or incorrect code.
No attempt	0	**No attempt.** For software – missing modules and/or functionality.

Oral Defence

During the assessment of your work you will be asked questions based on material which you have learnt in the subject and then used to implement the assessment task. You are expected to know exactly how your implementation works and be able to justify the design choices which you have made. If you fail to answer the questions with appropriate substance then you will be awarded **zero** for that component.