Data acquisition system for a spectrometer to analyze the habitability of Europa

The development of electronics for the in-situ Spectroscopic Europa Explorer (iSEE), a Raman spectrometer proposed for the Europa Lander mission

Austin Stover
B.S. Electrical Engineering
stover.a@wustl.edu

Weronika Pach
B.S. Systems Engineering
wpach@wustl.edu

Advised by:
Pablo Sobron, Ph.D.
Impossible Sensing, LLC

An ESE 498/499 Capstone Design Project Final Report
Submitted to Profs. Jason Trobaugh and Dorothy Wang of the Department of Electrical and Systems Engineering
Abstract

The next lander sent to Jupiter’s moon Europa requires instruments to analyze organic and pre-biotic surface material. One proposed instrument for this analysis is the In-situ Spectroscopic Europa Explorer (iSEE), a Raman spectrometer to determine biomolecule composition. This project develops the data acquisition and management systems for a lab-based prototype of this instrument. We develop the requirements for this data acquisition, examine several possible configurations that satisfy these requirements, and develop partial prototypes to demonstrate the systems and verify that the design meets timing requirements. The final design comprises a microcontroller as a Command and Data Handler that acts as a hub for data and control, a time-digital converter that performs raw data acquisition, and a client graphical interface on an external computer that controls all instruments and outputs results to the end-user.

1 Introduction

1.1 Background

Missions to land on other planets and moons in the Solar System often require tools to analyze surface and subsurface features for signs of life, including biomarkers such as pre-biotic and organic compounds. This mission envelope requires rapid and robust in-situ spectroscopic analysis to determine organic chemical composition, an application particularly well-suited for Raman spectroscopy. Raman spectroscopy is a non-destructive chemical analysis technique that can identify molecules based on the wavelength shift in light scattered off these molecules, which depends on the molecules’ characteristic quantum mechanical vibrational and rotational energy states.

In recent years, Raman spectroscopy has been found to provide a possible method for in-situ planetary sample analysis, in part because it requires minimal or no sample preparation, and can sample solid, liquid, or gaseous samples, whether transparent or opaque [5]. A fully integrated Raman spectrometer is included on the Mars 2020 rover in the SHERLOC instrument [1] and is planned for the ExoMars rover mission.

However, thus far instruments such as the one on ExoMars have required limited core-and-analyze approaches. Impossible Sensing’s Raman instrument, the In-situ Spectroscopic Europa Explorer (iSEE) [2], intends to overcome past limitations by performing a faster and more sensitive type of arm-mounted analysis without the need for coring or sampling [3]. The speed and sensitivity of this novel instrument, with its ability to detect organic compounds of concentrations down to 1 part-per-billion, make it suitable for investigating the habitability of icy worlds. It is therefore particularly relevant for the Europa Lander, a Flagship interplanetary astrobiology mission under development: the Europa Lander Report identified vibrational spectroscopy as a candidate for the baseline payload of the Europa Lander and identified sub-part-per-billion sensitivity to organic compounds as a requirement to achieve its mission goals.

iSEE provides a tool for future terrain-based spacecraft to conduct onboard rapid characterization of surface chemistry. It performs high resolution spectroscopic analysis of organic compounds on an arm mounted instrument.

The iSEE instrument comprises 4 main components: a controller, a spectrometer, a laser, and an optical assembly that includes a mirror and dichroic filter (fig. 1). The controller fires the laser and records data measured by the spectrometer. The laser light propagates along an optical fiber to the optical assembly mounted on the external arm and into the
Figure 1: Diagram of the iSEE Raman spectrometer

sample. Raman-scattered light then is reflected through another optical fiber and into the spectrometer, which returns data to the controller. Post-processing on this data converts it into Raman spectra, such as those in fig. 2, with peaks characteristic of the types and abundances of molecules in the sample.

1.2 Problem Statement

The development of iSEE requires the design and implementation of a data acquisition (DAQ) system to convert raw data to Raman spectra. Data representing Raman spectra is input in the form of 2 high-speed pulses on the order of 10 nanoseconds apart, an event which repeats on the 100 microsecond timescale. The time difference between these pulses must then be analyzed, processed, and converted into Raman spectra. The system must also trigger the laser that shines on the sample and results in the high-speed pulses, and it must process and present the Raman spectra on a client computer with a graphical user interface (GUI). An optical controller and laser controller also must communicate with the client.

This system requires increasing the technology readiness level of the fully integrated iSEE instrument from 2 to 4, taking it from the proof-of-concept stage to a fully functioning laboratory prototype. The design must be adaptable to a finished product that will be qualified for space and sent to Europa, although the current design only needs to work in the lab and eventually be qualified for vibrations and minimal thermal tolerance.

The scope of this project extends to:

1. Designing the high-level system, system components, and system requirements

2. Implementing prototypes of components with demonstrable ability to acquire input from the sensing elements on the required timescales

3. Incorporating a graphical interface to manage the experiment and perform data analysis
This project excludes the laser control and optical controller, which are to be implemented later. To summarize, there are two main timing regimes in the project which drive the design:

1. **Data acquisition**
   One repeatable cycle of data acquisition must occur on the order of 10 nanoseconds. This acquisition requires a fast, flexible interface and should be designed in a way that enables eventual transition to a final product.

2. **Data management**
   One repeatable cycle of data management must occur on the order of 100 microseconds. Data management requires a real-time system (one with predictable timing) and must be flexible for rapid prototyping and development.

These timing regimes impose the most stringent constraints on the project and thus drive the project objectives.

### 1.3 Objectives

In this project we design and implement the electronic sensor, actuator, and client interfaces and computation for instrument data acquisition. The main subsystems we develop comprise the following:

1. **The CDH**
   The CDH is the main computer for the iSEE instrument. It acts as a hub for peripherals and communicates with a client computer. It manages the communication between all peripherals: an FPGA, optical computer, and laser controller and trigger. The scope of CDH development in our project will extend to the FPGA, client,
and optical computer control and interfacing. The CDH must be able to perform all communication with a timescale on the order of hundreds of microseconds, but also requires a rapid prototyping and development timeline.

2. The Client GUI
This interface controls and communicates with the CDH. The GUI must be developed in a time-efficient manner; based on our skillset and the skillset of Impossible Sensing, we use the LabView GUI-development and system-design environment to build the controller and communication interface. The client device must process the pulse data sent from the CDH and convert it into floating-point delta-times. It must then count the number of pulse delta-times sent by the CDH that fall within some predetermined interval to output Raman spectra.

3. The DAQ
We require a timescale on the order of nanoseconds to read delta-time pulse input from sensing elements.

Figure 3 demonstrates how components in the system are connected.

![Figure 3: A functional diagram of the project.]

The other devices in the system are:
1. A laser with an external trigger and controller interface.
2. An optical controller that manipulates the experiment.
3. 2 sensing elements which generate nanosecond pulses to be converted to Raman spectra.

Given the project constraints and operational restrictions, the objectives are threefold:
1. Design the overall system and define detailed system requirements, including data acquisition, communication protocols, and data processing.

2. Develop the CDH on the required timescale. Perform communication with the DAQ module.

3. Develop the client data processing algorithm and the client GUI to control the experiment and show initial results.

Original project desirables included the implementation and validation of a fully-functioning laboratory prototype, but the project was scaled back to design and partial implementation, with limited testing, due to operational restrictions from the spring 2020 United States COVID-19 disease outbreak. Access to the lab and equipment was restricted, thus limiting project development in regards to hardware configuration and testing.

2 Methods

2.1 System Design

2.1.1 Command and Data Handler

The CDH requires a combination of speed, with a timescale on the order of hundreds of microseconds, and rapid development. The most convenient way to implement this design requirement is using a fast microcontroller. For ease of the client, this device will communicate to the GUI over a serial protocol. It will also communicate with the DAQ over serial and must send and receive commands from the optical controller over I$^2$C.

Possible design alternatives also exist for the CDH. Instead of a discrete microprocessor, a Microblaze soft processor running on an FPGA could be used for all communication. This would allow the product to have a smaller footprint, since the data acquisition was also proposed to be running on the same FPGA, but even on a faster FPGA, the Microblaze runs slow enough that it could possibly violate timing requirements, and the wealth of open source libraries for Arduino-compatible devices enables faster prototyping and modification. One other possible alternative might be to implement the entire design on a system-on-a-chip (SOC), an FPGA interconnected with a full processor, which would correct the speed issues posed by the Microblaze. However, most SOCs require a full operating system, which brings with it unfixed interrupt latencies and other unpredictable timing characteristics. Real-time operating systems solve these timing issues, but typically require significant time and financial overhead to learn and implement. A fast discrete-microprocessor solution therefore better satisfies the project requirements.

To determine the type of microcontroller processor required for the CDH to meet timing requirements, we estimated the timing required for the CDH program. A typical `digitalWrite()` takes about 50 clock cycles to perform, and a `Serial.write()` takes 320 clock cycles to perform. These are some of the longest-running instructions that one may typically use. Assuming, then, that on average each line of code takes 100 clock cycles to perform, and that the CDH program executes 400 lines of code each timing period, the clock needs to run at 400 MHz to execute the necessary code in the required time. With a factor of safety equal to 1.5, this gives a desired clock speed of 600 MHz, which happens to be the clock speed of the Teensy 4.0 microcontroller that we are going to use. On the Teensy 4.0 600 MHz ARM Cortex M7 processor, this conservative estimate predicts a runtime of 67 $\mu$s per time period.
2.1.2 Data Acquisition

Data acquisition requires determining the time delay, on the order of 10ns, between 2 pulses with widths also on the order of 10ns. The first pulse comes from sensing element 1, and the second pulse from element 2. Additional pulses might also be detected from element 2. Recording the times between the first pulse and these additional pulses could expand the capabilities of the spectrometer; however, these times are not strictly necessary to determine the Raman spectra.

Two possibilities were considered to implement data acquisition. The first is the use of fast ADCs and an FPGA to centroid pulses and measure times between the first pulse and a chosen number of additional pulses. This method allows for a high degree of flexibility and remains robust to the degradation of pulse shape due to parasitic capacitance and inductance in the circuit. However, this method also requires a high degree of development time and is limited in time resolution to the ADC sample rate.

On this FPGA, signals pass directly to ADCs, and a custom IP block quickly processes ADC input. The FPGA also needs to interface with the CDH to communicate its findings, so it would use a Microblaze soft processor that would send data over a serial line to the CDH.

The DAQ must perform several tasks while meeting timing requirements. It must be able to detect a pulse from sensing element 1, wait a set amount of time, and then accept input from sensing element 2. To accomplish this set wait period, a counter must be implemented that counts clock cycles between the two pulses. Any other element of the DAQ system that handles these pulses, including the ADCs and ADC interface, may delay these pulses as long as each pulse handler delays the pulses the same amount. Thus, the only truly time-critical component is the counter between the two pulses.

To achieve a time resolution on the order of 1 nanosecond, the counter must be able to work at clock speeds around 1 nanosecond. Xilinx documentation for its Binary Counter IP v12.0 reveals that UltraScale+ FPGA architectures are capable of implementing a binary counter with a maximum clock speed on the order of 1GHz, thus exceeding DAQ timing requirements of a 10 ns timescale [7].

![Figure 4: Oscilloscope capture of a characteristic pulse from a sensing element.](image)

Furthermore, fast ADCs are required to meet timing requirements. Due to the speed of the sensing element input pulses and the capacitance and slew rate of intermediate electron-
ics, pulses are stretched and flattened; a pulse taken from a sensing element is shown in fig. 4. The Analog Devices AD-FMCDAQ2-EBZ ADC demo FMC board provides a 1 GHz sample rate for 2 ADCs. This makes it capable of resolving and centroiding pulses on the 10 ns timescale, once again meeting DAQ timing requirements. Figure 5 illustrates a prototype system design using ADCs and an FPGA.

Figure 5: A functional diagram of an early prototype of the project, including an FPGA and discrete ADCs.

The second possibility considered is the use of a Time-Digital Converter (TDC), which directly measures the time between the rising edges of two pulses. This method requires much less development time but significantly decreases flexibility (with the ability to measure time only between the first pulse and the next 4 pulses). A TDC provides increased time resolution but also decreases robustness to noisy signals since it only measures pulse rising-edges instead of centroiding them, which makes it susceptible to parasitic capacitances and inductances.

### 2.2 Time Digital Converter

Two TDCs on the market were identified as prospective candidates for data acquisition, and fit general project requirements for a TDC: the Texas Instruments TDC7201 and the Sciosense TDC-GPX2. Both instruments provide sufficient time resolution, on the order of 10ps, accuracy, and minimum and maximum time between pulses for the application. However, the TDC7201 allows for flexibility in the pulse voltage threshold and standard. The Sciosense GPX2 requires a Low-Voltage Differential Signal (LVDS) type input, a differential-style signal, but since sensing element 1 and 2 adhere to a different range of voltages with a single-ended signal, the added complexity of an amplifier and differential-to-single-ended converter that could lead to parasitics and timing uncertainty makes the TDC7201 a more desirable candidate, although the TDC7201 is somewhat less accurate than the GPX2.

The TDC7201 requires an initial trigger pulse that starts clock cycle counting for its 2 channels. This is connected to the laser trigger output on the CDH. The signals from sensing
elements 1 and 2 then propagate into Stop channels 1 and 2 on the TDC, respectively, and
the TDC measures the times T1 and T2 from the trigger or REF pulse to each of the stop
pulses, STOP1 and STOP2 (see fig. 6). The difference between these times T3 is then
recorded by the CDH and sent over UART to the client, along with uncertainties and other
desired metadata.

Use of the TDC7201 requires communication over the SPI protocol. The interface works
through the following sequence of commands:

1. Start TDC
2. Request measurements through SPI
3. Loop or interrupt
4. Get data
5. Send data to client for processing

Steps 2, 3, and 4 occur in a loop until a series of measurement cycles is complete. This
way, the requirement for a pulse measurement cycle period on the order of 100 microseconds
may be more easily satisfied; serial communication is time consuming (on this timescale,
at least) and can happen after cycles are complete, with pulse measurements stored in a
buffer in the CDH until data may be sent to the client with less stringent time restrictions.
This method also relieves time constraints on the client–since the client device may vary,
the speed of serial communication, serial buffer clearing, and data processing on the client is
undefined. Thus performing time-intensive data transfer is best left for after measurement
cycle completion.

2.3 Timing Challenges

The strict timing requirements of data acquisition from sensing elements 1 and 2 present
several challenges that require addressing during the design of hardware such as the printed
circuit boards (PCBs) that will connect the TDC, and sensing elements. Since the TDC
measures time from pulse rising edges instead of centroiding pulses (see section 2.1), main-
taining the integrity of the pulse edge shape is critical to timing accuracy.

A pulse edge, modeled as a step input signal to an approximately linear time-invariant
circuit, is susceptible to several deteriorating attributes:

1. Reflections
   Reflections occur for extremely high frequency signals when the characteristic impedan-
ce of the signal line does not match that of the circuit at the end of the line. Impedance
   matching, then, is necessary before each component on a circuit board to minimize
Figure 7: Oscilloscope measurement of sensing element 1 and 2 inputs. Pulse width on the order of 10 ns. The input from sensing element 2 contains the initial, desired pulse as well as a pulse trail of unwanted reflections due to impedance mismatch.

reflections. The signal from sensing element 2 in fig. 7 contains reflections from the initial pulse due to poor impedance matching at the end of the line.

Impedance matching on a PCB requires placing a resistor with resistance equivalent to the line impedance in parallel with the ground plane just before a component with a high-impedance input. For components with a lower impedance input, the equivalent resistance of the component and the resistor in parallel should be equal to the line impedance. One can also place a resistor in series with the low-impedance output of the last component in the circuit, but this is unnecessary since it only reduces back-reflections and could dissipate a significant amount of power.

The characteristic impedance of the PCB required for the TDC is 50 Ω.

2. **Parasitic inductance**
Parasitic inductance, such as strip inductance, typically arises from excessive trace length or vias placed on the PCB for testing. The trace to certain time-critical components such as the TDC Start and Stop connectors may need to be lengthened to delay the signal, which can then cause this inductance and lead to oscillations in the PCB circuit step response. This inductance can be controlled by calculating the proportion of the trace width and distance to ground plane to create a series-inductance parallel-capacitance circuit with a constant impedance equal to the characteristic impedance.

3. **Parasitic capacitance**
Parasitic capacitance is the most difficult attribute to reduce. On a PCB, it can be controlled along the trace by balancing it with the strip inductance, but each component, connector, and wire introduces additional capacitance that acts as a low pass filter and smooths out the step response, decreasing the accuracy of a rising edge pulse measurement. The cable used for the project, the SMA cable, supports microwave frequency signals around the required range. Cables and connectors available at reasonable prices are rated at up to an 18 GHz corner frequency, which means they
have a low enough capacitance to allow a rise time of [6]

\[ \tau_r \approx \frac{0.35}{f_{3\text{db}}} \]

or 0.02 ns, assuming the SMA cable acts close to a perfect low-pass filter, which is more than enough for an accurate TDC reading.

Careful control of parasitic capacitance and inductance is required in circuit design, since pulse shape integrity is especially difficult to measure experimentally due to oscilloscope frequency limitations.

2.4 Client Interface

The client interface (fig. 8) is constructed using the NI-VISA API in LabVIEW. A graphical user interface is automatically generated in LabVIEW as its corresponding block diagram is created. The user first configures the connected hardware within LabVIEW and then establishes serial protocol settings. The program then writes to the CDH to initiate laser firing. Serial data collection is facilitated by the service request method due to the strict timing requirements and amount of samples collected per run. Once received, each packet is read by the program, its information organized within the program for Raman analysis, and backed up to the host computer. Error analysis is included. The program must receive complete time and calibration data to perform necessary calculations and generate spectra figures.

2.5 Client Data Analysis

A serial protocol design was developed and partially implemented (see Appendix A) to communicate TDC pulse times from the CDH to the client for processing. This serial protocol consists of separate data packets detailed in full in fig. 9.

Each serial data packet consists of 42 bytes. Bytes 2-37 relay the time between pulses from sensing elements 1 and 2, which are represented as the time data from TDC1 and 2, respectively, and are used to calculate the difference T3. If T3 is within an acceptable
range, the sample is accepted. Bytes 20-22 indicate the optical controller setting at which
the sample was taken. The client program accumulates accepted counts in an array; once
this is complete, the Raman spectra is generated from this array as a plot of intensity vs.
wave number. The combination of absolute intensities and peaks at specific wave numbers
represent the functional groups in a sample, which help determine the chemical structure of
the compound analyzed.

![Data packet for serial communication of data from the CDH to the client.](image)

### Figure 9: Data packet for serial communication of data from the CDH to the client.

#### 2.6 Data Acquisition Equipment

1. **Data acquisition** The 2 sensing elements and an oscilloscope with greater than 1 GHz
time resolution, as well as the spectrometer laser.

2. **Data analysis** The Texas Instruments TDC7201 TDC,

3. **Collection of processed data** Using the Teensy 4.0 microcontroller

4. **Client use and management** A Windows computer running LabVIEW with USB
ports configured to the CDH through the Measurement and Automation Explorer
(MAX)

5. **Signal communication** A USB cable, breadboard, jumper wires, SMA-to-SMA 18
GHz cables, and a SparkFun UART-to-USB converter breakout board part FT232RL

#### 3 Results

The TDC control code on the CDH was tested to ensure the runtime for a single laser trigger
and pulse time difference measurement fit did not exceed the $\sim 100 \mu s$ period requirement.
The code to produce this timing measurement setup can be found in Appendix A; the output
it produced was:

$$\text{Period length micros: 20}$$

This $20 \mu s$ runtime excludes time spent waiting for the laser to fire and a measurement to
complete after a trigger is sent to the laser, since this wait period could only be ascertained by
testing that was not possible to perform without access to the testing environment. However,
the $20 \mu s$ runtime leaves plenty of time for the trigger and measurement (which should
happen within a few microseconds) and in addition is over 3 times as fast as the conservative runtime of 67 µs predicted in section 2.1.1. Communicating pulse time difference data only after a sequence of measurement periods allowed the TDC control code to be fairly compact, time-wise, and adequately satisfy the requirements.

This TDC control code could not be tested to actually ensure that the correct pulse time was recorded by the CDH, but since timing can broadly be tested without analog pulse input, excluding the time between trigger and measurement, the timing results comprise a successful partial test until a full test including analog pulse inputs and oscilloscope confirmation of results can be conducted.

Client-side results include the graphical interface prototype seen in fig. 8 and the ability to generate plots from sample data, such as those in fig. 2. Due to testing restrictions, the plots shown are created using sample data provided by Impossible Sensing, but the spectra are generated in a sub-vi within the client GUI.

Figure 10: Left: Raman spectra of Mg Copiapite. Right: Raman spectra of potassium sulfate.

4 Deliverables

Finished deliverables include an overall system design, detailed TDC and CDH communication protocols and operations, timing considerations for TDC PCB design, and a minimally tested prototype CDH program. The CDH program compiles and runs, and the commands, which consist of register reads and writes, result in the desired responses from the TDC given the lack of ability to test analog input pulses. The CDH program executes one full timing period, running well within the desired ~ 100 µs period, and gathers the required data from the TDCs necessary to calculate the time between two rising edge pulses only nanoseconds apart.

The current version of the client GUI establishes a connection with a device connected through a USB, a CDH if it was accessible. The program includes a component for detecting a service request from the device, and reads the raw output. Potential methods for decoding data packets are being explored, but would need to be tested with sample raw data. The
GUI also includes a sub-vi that plots intensity vs. wavenumber when that calculated data is provided.

5 Discussion

Several key concepts were developed in the design and initial implementation of the DAQ systems for the iSEE Raman spectroscopy instrument:

1. TDCs provide significant benefits over an FPGA-digitizer design. The capacitance requirement benefits of pulse centroiding can be mitigated by using cables with a high-enough corner frequency and performing impedance matching by controlling PCB capacitance and inductance. Table 1 details the tradeoffs between the FPGA and TDC designs. Further hardware development includes the design of a printed circuit board with both the TDCs and Cortex M7 processor on the same chip.

2. High speed Arduino-compatible microcontrollers such as the Teensy 4.0 can provide convenience and lower development times while delivering real-time, 100 µs-scale interfaces between TDCs and a slower client computer with more processing power. Additional development requires testing the TDCs with analog pulse inputs and finishing and testing the client-CDH serial communication protocol.

3. A client-side program implemented in LabVIEW provides a flexible graphical interface for the end-user of the laboratory instrument and allows one to process TDC and optical controller data packets from a microcontroller to generate Raman spectra; however, more work is required to fully implement the packet-based serial protocol developed in this paper and to convert raw TDC output to floating-point pulse time difference measurements.

<table>
<thead>
<tr>
<th>Trade-off Parameter</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Time</td>
<td>TDC</td>
</tr>
<tr>
<td>Precision &amp; Accuracy</td>
<td>+</td>
</tr>
<tr>
<td>Complexity</td>
<td>+</td>
</tr>
<tr>
<td>Capacitance Requirements</td>
<td>-</td>
</tr>
<tr>
<td>Customizability</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Tradeoffs between use of a TDC and an FPGA with standard ADCs for analog signal input. A “+” implies advantages and a “−” disadvantages.

6 Conclusion

The overall goal of this project was to advance a portion of the equipment that makes up the iSEE instrument from TRL 2 to 4. With this work, the system qualifies for TRL 3, showing analytical and experimental critical function. The use of the Teensy 4.0 as a central hub for data transfer and management was shown to satisfy the ∼100 µs measurement cycle timing
requirements without sacrificing cost, time, or compatibility. Both TDCs and FPGAs were considered for \(~ 10\) ns raw data acquisition, but TDCs were found to be more favorable for minimizing development time and complexity. TDC limiting factors such as a low tolerance for parasitic capacitance were found to be circumvented by proper impedance matching and cable choice. In addition, a fast, high-level system composed of suitable hardware and a relevant GUI was designed to be robust and flexible enough for further development.

Going forward, this system will need to undergo additional testing, including through the generation of high-speed pulses. The client program must expand to implement full serial communication with the CDH and the GUI must include control elements for the optical and laser controllers.

With the development of this Raman spectrometer DAQ system, we bring iSEE one step closer to probing for pre-biotic compounds in, and possibly even discovering life on, the icy crust of Europa.

7 Schedule

![Timeline of the project](image)

**Figure 11:** Timeline of the project.

8 Acknowledgements

Thanks to Prof. James Buckley and Paul Dowkontt of Washington University in St. Louis for discussions of high-speed electronics design. Thanks also to Impossible Sensing, LLC for sponsoring this work. The iSEE project is funded under the NASA MatISSE program for technology maturation toward flight.
References


Appendix A  CDH TDC Control Code

/**
 * Author: Austin Stover
 * Date: March - April 2020
 * This program communicates with the TI TDC7201 Time- digital converter over SPI. It performs
 * one period of data collection from the TDC, storing the data required to recreate time
 * between pulses in an array of data packets to be sent to the client.
 */

#include <SPI.h>

//Define union to make working with the data packet easier
typedef union
{
    //TODO: Expand to include other vars in packet
    uint8_t bytes[36];
    struct
    {
        //Use bitfield to simulate 3-byte integers
        uint32_t t1ft1:24; //Fine time 1 on TDC1; this LSB first
        uint32_t t1ft2:24; //Fine time 2 on TDC1
        uint32_t t1ct1:24; //Coarse time 1 on TDC1
        uint32_t t1ct2:24; //Coarse time 2 on TDC1
        uint32_t t1cal1:24; //Calibration 1 on TDC1
        uint32_t t1cal2:24; //Calibration 2 on TDC1
        uint32_t t2ft1:24; //Fine time 1 on TDC2
        uint32_t t2ft2:24; //Fine time 2 on TDC2
        uint32_t t2ct1:24; //Coarse time 1 on TDC2
        uint32_t t2ct2:24; //Coarse time 2 on TDC2
        uint32_t t2cal1:24; //Calibration 1 on TDC2
        uint32_t t2cal2:24; //Calibration 2 on TDC2
    } attribute((packed)) reg;
    //Packed attr deletes padding for each struct member
} Packet;

//buffer_array contains recorded TDC data to send over serial as a byte array
const uint16_t BUFFER_ARRAY_SIZE = 1;
Packet buffer_array[BUFFER_ARRAY_SIZE] = {{{0}}};

const uint8_t PIN_ENABLE = 2;
const uint8_t PIN_CS_B1 = 3; //Active low
const uint8_t PIN_CS_B2 = 4; //Active low
const uint8_t PIN_REF_START = 5; //REFERENCE START signal (TDC and laser trigger)

const SPISettings spiSet{25000000, MSBFIRST, SPI_MODE0}; //MODE0 -> CPOL = 0, CPHA = 0, output on falling edge and capture
//Clk freq can go up to 25MHz for TDC but Teensy does 4MHz
const uint8_t BAUD_CDH = 9600;

//Register Addresses
const uint16_t CONFIG1 = 0x00;
const uint16_t CONFIG2 = 0x01;
const uint16_t INT_STATUS = 0x02;
const uint16_t INT_MASK = 0x03;
const uint16_t COARSE_CNT_OVF_H = 0x04;
const uint16_t COARSE_CNT_OVF_L = 0x05;
const uint16_t CLOCK_CNT_OVF_H = 0x06;
const uint16_t CLOCK_CNT_OVF_L = 0x07;
const uint16_t CLOCK_CNT2TOPMASK_H = 0x08;
const uint16_t CLOCK_CNT2TOPMASK_L = 0x09;
const uint16_t TIME1 = 0x10;
const uint16_t CLOCK_COUNT1 = 0x11;
const uint16_t TIME2 = 0x12;
const uint16_t CLOCK_COUNT2 = 0x13;
const uint16_t TIME3 = 0x14;
const uint16_t CLOCK_COUNT3 = 0x15;
const uint16_t TIME4 = 0x16;
const uint16_t CLOCK_COUNT4 = 0x17;
const uint16_t TIME5 = 0x18;
const uint16_t CLOCK_COUNT5 = 0x19;
const uint16_t TIME6 = 0x20;
const uint16_t CALIBRATION1 = 0x21;
const uint16_t CALIBRATION2 = 0x22;

void setup()
{
    Serial.begin(115200); //Serial monitor output
    //Serial.begin(BAUD_CDH, SERIAL_8N1); //To client
    //Initialize pins
    pinMode(PIN_ENABLE, OUTPUT);
    pinMode(PIN_CS_B1, OUTPUT);
    pinMode(PIN_CS_B2, OUTPUT);
    digitalWrite(PIN_ENABLE, LOW);
    digitalWrite(PIN_CS_B1, LOW);
    digitalWrite(PIN_CS_B2, LOW);
    digitalWrite(PIN_REF_START, LOW);
    SPI.begin();
    digitalWrite(PIN_ENABLE, HIGH); //Low-to-high transition
delayMicroseconds(2000); //Wake up for measurement! (TODO: Base wake-up time off of clock frequency)
}
void loop()
{
uint32_t start_of_period = micros(); //The ith buffer in o_buffer_array
uint16_t i = 0;
Packet& o_buffer = o_buffer_array[i];
uint16_t spi_buffer16 = 0; //Initialize a buffer for sending register values to TDC
uint32_t spi_buffer32 = 0; //Initialize a buffer for receiving data from the TDC

//DO THE FOLLOWING TO BOTH TDCs
digitalWrite(PIN_CS1, LOW);
digitalWrite(PIN_CS2, LOW);

//SET CONFIG2 REGISTERS
//Initialize control bits (2)
uint16_t controlBits = 0b01; //No auto increment, Write
spi_buffer16 = controlBits << 14 + CONFIG2 << 8 + config2Bits;
SPI.beginTransaction(spiSet);
//delayMicroseconds(1);
SPI.transfer(&spi_buffer16, sizeof(uint16_t));

//SET CONFIG1 REGISTERS AND START NEW MEASUREMENT
//Initialize control bits (2)
controlBits = 0b01; //No auto increment, Write
uint16_t config1Bits = 0b00000011; //Last bit = 1 starts new measurement (& sends TRIGG)
spi_buffer16 = controlBits << 14 + CONFIG1 << 8 + config1Bits;
SPI.transfer(&spi_buffer16, sizeof(uint16_t));

//Send REFERENCE START signal to both TDCs
digitalWrite(PIN_REF_START, HIGH);
//TODO: Change wait time to non-blocking before pulling REF_START low again
delayMicroseconds(1);
digitalWrite(PIN_REF_START, LOW);

//ONLY READ FROM TDC1 NOW
digitalWrite(PIN_CS1, LOW);
digitalWrite(PIN_CS2, HIGH);

//LOOP (If we want to interrupt here use the INTBx pin on the TDC)
bool interrupted = false;
while(!interrupted)
{
    //Read INT STATUS
    controlBits = 0b00; //No auto increment, read
    spi_buffer16 = controlBits << 14 + INT_STATUS << 8;
    SPI.beginTransaction(kspi_buffer16, sizeof(uint16_t));
    interrupted = spi_buffer16 & (0x01 | (1 << 1) | (1 << 2)); //Measurement complete or overflow occurred
    interrupted = true; //For time testing
}

//Get data -- read clk count/time registers etc.

//If overflow
if(spi_buffer16 & ((1 << 1) | (1 << 2)))
{
    // TODO: Send OVF to client
    Serial.println("OVF detected");
    // Also if(spi_buffer16 & 0x01) //If measurement if(true) //For time testing
    // Get measurement: TIME1∗ring_osc_period + CLOCK_COUNT1∗clk_period
    // ON TDC1
    //READ FROM TIME1 (this is t1B)
    controlBits = 0b00; //No auto increment, read
    spi_buffer32 = controlBits << 14 + TIME1 << 8;
    SPI.transfer(&spi_buffer32, sizeof(byte)*3); //Get fine time 1
    o_buffer.reg.t1ft1 = spi_buffer32; //Transfer SPI message to serial buffer
    Serial.print("T1 TIME1: "); Serial.print(o_buffer.reg.t1ft1 & 0x7FFF); Serial.println(" "); //Delete parity bit
    //READ FROM TIME2 (this is t1S)
    controlBits = 0b00; //No auto increment, read
    spi_buffer32 = controlBits << 14 + TIME2 << 8;
    SPI.transfer(&spi_buffer32, sizeof(byte)*3); //Get fine time 1
    o_buffer.reg.t1ft2 = spi_buffer32; //Transfer SPI message to serial buffer
    Serial.print("T1 TIME2: "); Serial.print(o_buffer.reg.t1ft2 & 0x7FFF); Serial.println(" ");
    //READ FROM CLOCK_COUNT1
    controlBits = 0b00;
    spi_buffer32 = controlBits << 14 + CLOCK_COUNT1 << 8;
    SPI.transfer(&spi_buffer32, sizeof(byte)*3); //Get coarse time 1
    o_buffer.reg.t1ct1 = spi_buffer32; //Transfer SPI message to serial buffer
    Serial.print("T1 Clock count: "); Serial.print(o_buffer.reg.t1ct1 & 0x7FFF); Serial.println(" ");
    //READ FROM CLOCK_COUNT2
    controlBits = 0b00;
spi_buffer32 = controlBits << 14 + CLOCK_COUNT2 << 8;
SPI.transfer(&spi_buffer32, sizeof(byte)*3); //Get coarse time 14
o_buffer.reg.t1ct2 = spi_buffer32;

//Get CALIBRATION
controlBits = 0b00;
spi_buffer32 = controlBits << 14 + CALIBRATION1;
SPI.transfer(&spi_buffer32, sizeof(byte)*3);
o_buffer.reg.t1cal1 = spi_buffer32;

//Get CALIBRATION from CALIBRATION registers and CONFIG2 register
controlBits = 0b00;
spi_buffer32 = controlBits << 14 + CALIBRATION2;
SPI.transfer(&spi_buffer32, sizeof(byte)*3);
o_buffer.reg.t1cal2 = spi_buffer32;

//ONLY READ FROM TDC2 NOW
digitalWrite(PIN_CSB1, HIGH);
digitalWrite(PIN_CSB2, LOW);

//Read INT_STATUS
controlBits = 0b00; //No auto increment, read
spi_buffer16 = controlBits << 14 + INT_STATUS << 8;
SPI.transfer(&spi_buffer16, sizeof(uint16_t));
interrupted = spi_buffer16 & (0x01 | (1 << 1) | (1 << 2)); //Measurement complete or overflow occurred
interrupted = true; //For time testing

//If overflow
if(spi_buffer16 & ((1 << 1) | (1 << 2))) //TODO: Fix branch sequence so that TDC1 and 2 are on the same level
{
    //TODO: Send OVF to client
    //Serial.println("OVF detected");
    //
    else if(spi_buffer16 & 0x01) //If measurement
    {
        //READ FROM TIME1 (this is t2ft1)
        controlBits = 0b00; //No auto increment, read
        spi_buffer32 = controlBits << 14 + TIME1 << 8;
        SPI.transfer(&spi_buffer32, sizeof(byte)*3);
o_buffer.reg.t2ft1 = spi_buffer32;
        //Transfer SPI message to serial buffer
        //Serial.print("T1 TIME1: "); Serial.print(o_buffer.reg.t2ft1 & 0x7FFF); Serial.println(""); //Delete parity bit
        //READ FROM TIME2 (this is t2ft2)
        controlBits = 0b00; //No auto increment, read
        spi_buffer32 = controlBits << 14 + TIME2 << 8;
        SPI.transfer(&spi_buffer32, sizeof(byte)*3);
o_buffer.reg.t2ft2 = spi_buffer32;
        //Transfer SPI message to serial buffer
        //Serial.print("T1 TIME2: "); Serial.print(o_buffer.reg.t2ft2 & 0x7FFF); Serial.println(""); //Delete parity bit
        //READ FROM CLOCK_COUNT1
        controlBits = 0b00;
        spi_buffer32 = controlBits << 14 + CLOCK_COUNT1 << 8;
        SPI.transfer(&spi_buffer32, sizeof(byte)*3); //Get coarse time 14
        o_buffer.reg.t2ct1 = spi_buffer32;
        //Serial.println("T1 Clock count 1: "); Serial.println(o_buffer.reg.t2ct1 & 0x7FFF); Serial.println("");
        //READ FROM CLOCK_COUNT2
        controlBits = 0b00;
        spi_buffer32 = controlBits << 14 + CLOCK_COUNT2 << 8;
        SPI.transfer(&spi_buffer32, sizeof(byte)*3); //Get coarse time 14
        o_buffer.reg.t2ct2 = spi_buffer32;
        //Serial.println("T1 Clock count 2: "); Serial.println(o_buffer.reg.t2ct2 & 0x7FFF); Serial.println("");
        //Get CALIBRATION
        controlBits = 0b00;
        spi_buffer32 = controlBits << 14 + CALIBRATION1;
        SPI.transfer(&spi_buffer32, sizeof(byte)*3);
o_buffer.reg.t2cal1 = spi_buffer32;
        //Serial.println("T1 Calibration 1: "); Serial.println(o_buffer.reg.t2cal1 & 0x7FFF); Serial.println("");
        //Get CALIBRATION from CALIBRATION registers and CONFIG2 register
        controlBits = 0b00;
        spi_buffer32 = controlBits << 14 + CALIBRATION2;
        SPI.transfer(&spi_buffer32, sizeof(byte)*3);
o_buffer.reg.t2cal2 = spi_buffer32;
        //Serial.println("T1 Calibration 2: "); Serial.println(o_buffer.reg.t2cal2 & 0x7FFF); Serial.println("");
    }
}

Timing
uint32_t end_of_period = micros();
Serial.print("Period length in microseconds:"); Serial.print((end_of_period - start_of_period)); Serial.println(""), delay(10000);
//Process data and send to client
SPI.endTransaction();
}