Sketch outline of Chromation spectrometer `interface` for datasheet

Table of Contents

—e-n-d—

Spectrum Measurement System Overview

- A complete spectrum measurement system consists of:
  - spectrometer
  - SPI slave digital interface to communicate with the spectrometer
  - SPI master to bridge the digital interface and other slave devices to a system host
  - the system host application to request measurements and present the data
- TODO: diagram of complete system
- Chromation produces the spectrometer in a surface mount package compatible with standard reflow profiles
- Chromation offers the spectrometer un-mounted or mounted:
  - *un-mounted*: a narrow QFN-48 package for PCB assembly by the customer
  - TODO: photo of chip
  - *mounted*: QFN-48 package is assembled on a small PCB breakout to an 8-pin flat-flexible cable
  - TODO: photo of chip on daughterboard

- Chromation also offers a complete measurement system for evaluating the spectrometer:
  - mounted spectrometer
  - SPI slave digital interface
  - SPI master bridge
  - system host application for basic spectrum measurements
  - TODO: photo of complete system
  - TODO: screenshot of eval app

Spectrometer Overview

Components

The main components are the *photonic crystal* and the *linear pixel array*.

- TODO: diagram showing photonic crystal and linear pixel array

photonic crystal

- the *photonic crystal* is manufactured by Chromation
- light input to the spectrometer is spectrally separated by the *photonic crystal*
- the strong spectral separation of the *photonic crystal* provides high spectral resolution in a small form factor
- TODO: photo of photonic crystal

linear pixel array

- the spectrometer uses linear pixel array *Dynamax#* `LIS-770i`
- the *linear pixel array* measures the spectrally-separated light output by the photonic crystal
- each pixel collects a narrow wavelength bin from the spectrally-separated light
- the pixel's location in the linear array determines which narrow wavelength bin it collects
  - the system that interfaces the spectrometer handles conversion from pixel number to wavelength
  - the conversion maps each pixel to the center wavelength in that pixel's wavelength bin
  - Chromation currently provides this pixel-to-wavelength mapping data for each spectrometer

Measurement

Every measurement is a *pixel exposure* immediately followed by a *pixel readout*. After completing one measurement the `LIS-770i` can sit idle arbitrarily long before taking the next measurement.

pixel exposure

- the duration of *pixel exposure* is commonly referred to as the *integration time*
  - *integration time* varies from several milliseconds to several seconds depending on the measurement illumination power
- the `LIS-770i` has electronic switches to control the duration of *pixel exposure* to the light from the photonic crystal
- the pixels *accumulate charge* while exposed, resulting in a final voltage stored on each pixel at the end of the exposure period
- these pixel voltages represent the amount of light measured by each pixel, i.e., the amount of light in that pixel's wavelength bin

pixel readout

- immediately after the exposure ends, the pixel voltages are output sequentially for digital conversion by an external ADC
- the `LIS-770i` has electronic switches to control which pixel is connected to the `VOUT` pin

Digital Interface Overview

The `LIS-770i` has internal switching logic to execute the *exposure* and *readout* sequence, but like most other linear pixel arrays, the `LIS-770i` requires external signals to drive this internal switching. The `LIS-770i` also requires an external ADC to convert its analog output.

Since the spectrometer is simply optical components mounted around the `LIS-770i`, *the spectrometer interface is the `LIS-770i` interface*. External hardware is required to drive the spectrometer to achieve the higher-level functionality of *acquiring a spectrum*. This external hardware is the *digital interface*.

The purpose of the digital interface is to hide the operation details of the `LIS-770i` and ADC, and instead provide the measurement system with a standard interface. Chromation recommends presenting the spectrometer as a *SPI slave*. A *SPI master* in the upstream measurement system executes requests from the system host application by writing high-level commands recognized by the *SPI slave*.

- the main high-level command for the *SPI slave digital interface* is *acquire a spectrum measurement*:
  - run the *expose* and *readout* sequence
  - during the *readout*, acquire and store the pixel values
- other high-level tasks are:
  - configure the `LIS-770i`:
    - set the programmable gain
    - select the active pixels
    - turn pixel summing on or off
  - auto-expose:
    - determine the optimal exposure time for the peak counts in the measured signal to reach the target value

Acquire a spectrum measurement

- from the perspective of the `LIS-770i`, a *spectrum measurement* is just a *frame of data*: sequential output of the voltages on each pixel immediately after exposure
- to collect one frame of data from the `LIS-770i`, the SPI slave:
  - clocks the LIS for the desired exposure period
  - clocks out the pixel voltages while communicating with the ADC to convert the voltages to 16-bit values
  - stores the 16-bit value for each pixel
    - Chromation recommends configuring the `LIS-770i` as 392 pixels
    - therefore the digital interface needs 784 bytes of storage for one measurement
    - the photonic crystal does not illuminate the entire pixel array, so the actual storage requirement is less
    - the `LIS-770i` can also be configured as 784 pixels (not recommended), requiring 1568 bytes to store one measurement
- the ADC is a SPI slave to the digital interface, so in the context of converting voltages during pixel readout, the digital interface is a SPI master
- the digital interface signals the SPI-to-Host Bridge when readout is finished

Convert spectrometer analog output to digital

- during readout, the digital interface converts the analog pixel voltage to counts
- the digital interface uses a 16-bit SAR ADC with *differential input* and *unipolar transfer function*
  - *differential input*: the ADC has *two* analog inputs and it converts the voltage difference between them

  - *unipolar transfer function*: the allowed analog voltage ranges from `0V` to the ADC external voltage reference
  - the ADC voltage reference is `2.048V`

Dark-correct analog output in hardware before conversion

- the `LIS-770i` offers a dark-voltage output to allow dark-correcting the pixel voltages *before* conversion to digital counts
- the digital interface takes advantage of this analog dark-correction by connecting the dark-voltage to the negative terminal at the ADC differential input:
  - `ADC.IN+` connects to LIS-770i analog output `VOUT`
    - `VOUT` is the pixel voltage output by the LIS-770i
    - the specific pixel available at `VOUT` is controlled during readout
  - `ADC.IN-` connects to LIS-770i analog output `VREF`
    - `VREF` is the *dark voltage reference* output by the LIS-770i
  - the difference:

```
ADC.IN+ - ADC.IN-
```

```
- is the *dark-corrected pixel voltage*
```

SPI-to-Host Bridge Overview

- the SPI-to-host bridge is a SPI master for controlling the spectrometer and any other slave devices like an LED driver for light-source-triggered measurements
- the SPI master relays communication between the system host app and the spectrometer digital interface
- for light-source-triggered measurements, the SPI master coordinates the light-source and spectrometer:
  - enable the light source immediately before requesting a frame from the spectrometer digital interface
  - disable the light source before reading the frame from the spectrometer digital interface

System Host Overview

- each frame of readout is a spectrum measurement
- the SPI master relays the measured spectrum to a system host for display, saving/logging, averaging, and for more complex data-processing
- the simplest data-processing is pixel-to-wavelength conversion
- examples of more complex data-processing:
  - power-normalizing the spectrum

  - converting from wavelength to color space or some application space
  - matching spectra to a database
  - displaying a go/no-go output
  - combining sample and baseline measurements as in an absorbance measurement

Dark-Correction Is Not Required

- spectrometer data typically requires digital dark-correction by the system host before processing:
  - a frame of dark-data is collected at the same integration time as the measurement
  - the system host subtracts the dark-frame from the measurement
- the Chromation spectrometer does not require the system host to dark-correct if the spectrometer digital interface subtracts the `LIS-770i` dark-voltage from the `LIS-770i` pixel voltage in analog hardware
- Chromation recommends the spectrometer digital interface perform an analog dark-correction by taking a differential measurement at the ADC input, so each measurement received by the system host is already dark-corrected

Details

- TODO: put the specifics of frame readout here
  - describe in terms of the actual LIS pins and the signals needed
  - do not put specifics for the ADC
  - pull this info from section: `LIS-770i frame readout`

SPI communication for requesting a frame

SPI master requests a frame

- SPI master *selects* the *spectrometer digital interface* MCU by pulling the `SS` (*slave select*) pin *LOW*
  - the *spectrometer digital interface* MCU is now in *SPI slave* mode
- SPI master loads SPI data register with byte to `request_frame`
  - this initiates the SPI transmission
- the byte in the SPI data register is shifted out
- SPI master pulls `SS` *HIGH*
  - this releases the *spectrometer digital interface* MCU from *slave* mode
- SPI slave parses the byte and recognizes it as `request_frame`
- SPI slave coordinates with the ADC and LIS to get a frame of data

SPI slave responds with the frame

SPI slave prepares the frame of data

- SPI slave collects the entire frame of data
- SPI slave loads the first byte of data into the SPI data register ##### SPI slave signals the frame is ready
- if the *spectrometer digital interface* is on the same PCB as the SPI-to-Host bridge, dedicate a pin on the MCU to send the *frame is ready* signal
- Chromation has the *spectrometer digital interface* and the SPI-to-Host bridge on separate PCBs, requiring a cable to connect them
- to avoid wasting a connector pin on this *frame is read* signal, Chromation uses the SPI `MISO` pin to signal when the frame is ready
- SPI slave pulls `MISO` *LOW* to signal that a frame is ready
- this is a deviation from SPI protocol because the SPI slave is driving `MISO` while its SPI bus is supposed to be idle
- this deviation is safe as long as the SPI master follows a contract:
  - when the SPI master sends `request_frame`, it waits for the frame
  - the SPI master does not open SPI communication with any other slaves until it gets the requested frame
- the other potential pitfall with this deviation is how the MCU SPI hardware implements driving the `MISO` pin
  - usually the SPI slave output is implemented for use with a pull-up resistor: the `MISO` pin is driven *LOW* and pulled-up *HIGH*
  - therefore, immediately after SPI communication ends, if the last bit out on `MISO` was a *LOW*, `MISO` slowly rises high via its pull-up, it does not immediately swing *HIGH*
  - if the SPI master sent `request_frame` and is not waiting for `MISO` *LOW* as the signal that the frame is ready, it will mistake this slowly-rising `MISO` signal as `MISO` being ready
  - the solution is for the SPI slave to disable its SPI hardware immediately after recieving `request_frame` so it can actively pull `MISO` *HIGH*, and then re-enable the SPI hardware
  - the SPI master waits to see `MISO` go *HIGH* before it starts checking that `MISO` is *LOW*, signaling the frame is ready ##### SPI master reads out the frame
- pull `SS` *LOW*
  - this places the *spectrometer digital interface* MCU in *slave* mode
- SPI master initiates the transmission of each byte by writing to the SPI data register
  - it does not matter what the master writes to the SPI data register
- pull `SS` *HIGH*
  - this releases the SPI slave MCU from *slave* mode
  - SPI slave considers the frame finished

Recommended configuration of the LIS-770i

- pixel binning: on
- use all five rows
- analog gain: 1x

Communication protocols in the measurement system

USB and FT1248

- the system app runs on a *USB* Host
- the SPI-to-Host Bridge has an FTDI USB-Bridge IC and uses the FTDI *FT1248* protocol to communicate with the USB Host:
  - the *SPI-to-Host Bridge* is the *FT1248 master*, meaning that it initiates FT1248 communication
  - the *USB Host* is the *FT1248 slave*
    - when the USB Host writes to the SPI-to-Host Bridge, the data sits in a `1K Rx buffer` in the FTDI USB-Bridge IC
    - the SPI-to-Host Bridge sees the buffer contains a message and, being the FT1248 master, it initiates FT1248 communication to read the message
    - similarly, when the SPI-to-Host Bridge writes to the USB Host, the data sits in a `1K Tx buffer` in the FTDI USB-Bridge IC
    - the USB Host receives the data when it is ready
  - the system app uses the FTDI `d2xx.dll` library to read and write the SPI-to-Host Bridge
    - the library contains many useful functions for setting up a serial communication library:
      - enumerating the connected devices,
      - checking the number of bytes available,
      - opening devices by name, closing devices
      - writing/reading bytes to/from an open device

Ignore anything beyond here - -these are details to include later or not

Digital Interface is SPI master to its ADC

- communicate with ADC over SPI to read the pixel voltages as 16-bit values:
  - ADC is the SPI slave
  - ATmega328 in *spectrometer digital interface* is the SPI master
    - uses the USART in SPI master mode

    * Slave select (or chip select) is just a general purpose i/o
    * the clock is USART XCK
    * MISO is USART RxD
    * there is no MOSI connection: the ADC is a read-only device ## FT1248 reference ### Hardware Specific FT1248 Settings in `FT_Prog`
- Run `FT_Prog`
- Scan for devices
- locate the `ChromationSpect-NNNN-NN` (`NNNN-NN` is the serial number)
- Select `Hardware Specific -> Ft1248 Settings`
- *8-bit wide config used on the LIS-770i interface:*
  - Clock Polarity High: unchecked
  - Bit Order LSB: *checked*
  - Flow Ctrl not selected: *checked* #### Flow Ctrl
- if `Flow Ctrl not selected` is unchecked, then the Ft1248 slave does not use MISO and MIOSIO[0] to send Rx and Tx buffer status signals
- read this as `Use flow control when the slave is inactive`

Bit Order

- the bit order is LSB
- example: send the byte `0x01`
  - if `Bit Order LSB` is checked
  - then the USB host reads this as `0x01`
  - if `Bit Order LSB` is unchecked
  - then the USB host read this as `0x80`

Clock Polarity

- unchecked means the CPOL is 0 which means SCK idles low
- since the clock phase CPHA is always 1, this means SCK goes high to signal pushing data onto the bus and SCK goes low to signal pulling data from the bus
- See FTDI Application Note `AN_167` > When `CPOL` is 1, the idle state of the clock is high. > When `CPOL` is 0, the idle state of the clock is low. > There are 4 possible modes which are determined by Clock Polarity (`CPOL`) and > Clock Phase (`CPHA`) signals. For the FT1248 slave, only 2 of these 4 modes are > supported. `CPHA` will always be set to 1 in the FT1248 slave because data is > available or driven on to MIOSIO wires on the first clock edge after `SS_n` is > active and is therefore sampled on the trailing edge of the first clock pulse.

FT1248 format of combined command and bus-width byte

The *bus-width* (BW) and command are sent in a byte on a subset of the `MIOSIO[7:0]` pins over one or more clock cycles.

For example, if BW is 1-bit:

- bit[7..0] is clocked out on MIOSIO[0]
- takes eight clock cycles

And the opposite example, if BW is 8-bit:

- `bit[7..0]` is clocked out on `MIOSIO[7:0]`
- takes one clock cycle

On the **first falling edge** of `SCK`, the FTDI chip gets the **bus-width ID** and therefore knows whether it's clocking out more bits, or reading the entire `CMD[3..0]` on the `MIOSIO` pins.

The Chromation *spectrometer digital interface* uses `CPOL = 0` (`SCK` idles `LOW`).

- command/bus-width byte format:
  - bit[7] = X
  - bit[6] = CMD[0]
  - bit[5] = CMD[1]
  - bit[4] = 0 if BW=8, X otherwise
  - bit[3] = CMD[2]
  - bit[2] = 0 if BW=4, X otherwise
  - bit[1] = 0 if BW=2, X otherwise
  - bit[0] = CMD[3]
  - if bits[4,2,1] are all X, the BW=1
- if BW=8, all eight pins are used:
  - MIOSIO[7] = X
  - MIOSIO[6] = CMD[0]
  - MIOSIO[5] = CMD[1]
  - MIOSIO[4] = 0 (BW=8)
  - MIOSIO[3] = CMD[2]
  - MIOSIO[2] = X
  - MIOSIO[1] = X
  - MIOSIO[0] = CMD[3]
- if BW=4, only MIOSIO[3:0] are used:
  - MIOSIO[7] = X
  - MIOSIO[6] = X
  - MIOSIO[5] = X
  - MIOSIO[4] = X
  - on 1st clock:
    - MIOSIO[3] = CMD[2]
    - MIOSIO[2] = 0 if BW=4
    - MIOSIO[1] = X

        * MIOSIO[0] = CMD[3]
    - on 2nd clock:
        * MIOSIO[3] = X
        * MIOSIO[2] = CMD[0]
        * MIOSIO[1] = CMD[1]
        * MIOSIO[0] = X

list of commands:

```
write                 0x0
read                  0x1
write buffer flush    0x4
```

FT1248 combined command and bus-width byte for an 8-bit bus

Here are the bytes to output on the MIOSIO port to specify 8-bit-wide read and write:

```
                                  // BW:8 (0)---------
                                  //                 |
                                  // CMD-nibble:  -01| 2--3
#define FT1248_CMD_WRITE8   0x86  //              1000 0110
#define FT1248_CMD_READ8    0xC6  //              1100 0110
                                  // Bit numbers: 7654 3210
```

There are four bits for the *command* nibble and four for the *bus-width* nibble. A **low** on a particular bus-width bit indicates the bus width. The other bits in the bus-width nibble should not matter, but to play it safe, the other bits in the bus-width nibble are pulled high.

one frame of LIS-770i data

- the ADC converts every pixel to a 16-bit value
- there are 784 pixels
    - the first 13 are optically dark
    - the 14th is a dummy
    - 15-784 are optically active
- 784 * 2 bytes = 1568 bytes
- hopefully the ATmega 2K SRAM can handle this
- if not, we can ignore the first 14 pixels
    - that makes it 1540 bytes
- there are probably more pixels we can ignore too
    - revisit the optical design to see which pixels are actually used ## LIS-770i power down
- see `Power Standby Mode`
- `Lis_Rst` is high

- `Lis_Clk` stops and idles low
- remains in power down mode until `Lis_Rst` is pulled low and `Lis_Clk` is restarted ## LIS-770i - power up
- see `Power Up sequence`
- `Lis_Rst` is pulled low
- `Lis_Clk` starts
- this power up sequence resets the imager
- default programming is enabled:
  - select entire active pixel array
  - set output amplifier gain to 2.5x
  - turn off the summing mode ## LIS-770i programmable setup
- see `Programmable Setup Register`
- power up
- pull `Lis_PixSelect` high
  - [x] `LisInit()` idles `Lis_PixSelect` low
- first rising clock edge reads `Lis_Rst`
  - this sets the value of the *summing mode bit*
  - *Summing Mode = 0* - normal operation
    - pixels are 7.8um wide
  - *Summing Mode = 1* - every other pixel is added together during integration
    - the size of the pixel array is halved to 392 pixels
    - pixel width becomes 15.6um
    - only 392 clocks are needed to output the data
- 2nd and 3rd rising clock edges read `Lis_Rst`
  - sets the value of gain bits `G2` and `G1`
  - hold `Lis_Rst` high for 25 clock cycles after the gain bits are set
  - this completely contradicts what the datasheet says next
- 4th through 28th rising clock edges read `Lis_Rst`
  - sets the active pixel sub-arrays in the order P25 to P1
  - see *Table 5b: Selectable Pixel Array* to see which pixel is in which of the 25 pixel sub-arrays
    - a pixel has five segments in the *pixel height* direction
    - each segment is 62.5um tall
    - each sub-array is one segment tall
    - each sub-array spans 154 pixels
  - just keep reset high, this selects every segment of every pixel ## LIS-770i frame readout
- `Lis_Clk` is running
  - [x] `LisInit()` pulls `Lis_Rst` low
  - [x] `LisInit()` starts the `Lis_Clk`
    - in the future, `LisInit()` will not start the clock
    - this will be a separate command to leave power-down mode
    - but for now intialize in power up mode
- [ ] TODO: implement a power-down mode
  - [ ] provide a SPI command to enter/exit power-down mode

- Do I need to use the *ISR* to do things on clock rising/falling edges?
- No, this can be done without interrupts by polling flags and manually clearing flags. In fact it is slightly faster without interrupts because it cuts out the function call/return overhead of the ISR.
  - The output compare unit sets the output compare flag `OCF0A` or `OCF0B` when `TCNT0` equals `OCR0A` or `OCR0B`.
  - Clear the flag in software by writing a logical one to its I/O bit location.
  - I tested with and without interrupts and the fastest performance is to do this without interrupts. That is worth the extra step of manually clearing the flag. The result code is easier to read too.
  - Unfortunately, I also discovered the high-level code makes the response time to clock edges unacceptably slow. It is so slow that attempting to follow the clock edges on Rst makes Rst output a square wave that jitters between sometimes being half the frequency, sometimes the same frequency, because it only rarely catches adjacent clock edges.
- `Lis_Rst` is low
  - `LisInit()` has `Lis_Rst` idle high for power-saving
  - but once the device is in power up mode, `Lis_Rst` is low
- `Lis_Rst` controls the integration period
  - in the following, `Lis_Rst` is pulled high and low
  - the new value is clocked in on the very next rising edge of `Lis_Clk` after `Lis_Rst` has changed
  - [x] change the value of `Lis_Rst` just after a `Lis_Clk` falling edge
    - this avoids confusion over when the LIS shifts in the new value of `Lis_Rst`
    - at 50kHz, it is 10us from the clock falling edge to the clock rising edge
    - there is a 0.6us to 0.8us delay between the actual clock falling edge and the time it takes software to catch the fall and pull `Lis_Rst` high or low
    - so in effect, I am guaranteed to have `Lis_Rst` always transition about 9us just before the next `Lis_Clk` rising edge, and to never transition during a clock edge
- `ExposureStart()` - pull `Lis_Rst` high
  - resets all pixels
  - integration begins
- `Lis_Rst` goes low
  - integration ends
  - integration time is the number of clock cycles that `Lis_Rst` was high
  - [x] decide how to track the integration period:
    - check for N clock cycles in the PWM interrupt?
      - requires the ISR know about the state of the system
      - system can take new requests while the camera integrates
      - no – ISR adds call/return overhead, slowing response time
    - or sit and poll the interrupt flag, incrementing a counter

        - localizes where the state needs to be known to precisely that place where it is used
        - entire system blocks until integration period ends
        - yes – there is nothing else the system has to do
- Exposure time
  - LisRst is high for 20us * nticks + 2us
  - But the LIS is counting clock pulses, so exposure time is exactly 20us * nticks.
- `Lis_Sync` input is pulsed high > Once RST is brought low a synchronization signal (SYNC) pulse is fired > starting on the next falling clock edge. Pixel readout then begins on the next > rising CLK edge after the SYNC signal goes low and continues for 784 clock > cyles. The SYNC signal is one clock period wide. SYNC is again fired during > the last pixel readout on the falling edge of CLK.
  - `ExposureStop()` - pull `Lis_Rst` low
    - call this just after a `Lis_Clk` falling edge
    - exposure officially stops on the next `Lis_Clk` rising edge
  - `WaitForSync()` - watch for `Lis_Sync` to go high, then low
    - call this just after calling `ExposureStop()`
    - `Lis_Sync` goes high on the next falling clock edge
    - `Lis_Sync` then goes low on the next falling clock edge
    - pixel readout starts on the very next clock rising edge
  - `PixelReadout()` - sample analog `VOUT` while the clock is high
    - video output may have disturbance during the falling clock edge
    - pixels 1 to 13 are dark
    - pixels 15 to 784 are optical