Introduction

The CPL ADC Board is an AD7732 based fully differential two-channel, 15.075 kHz, 24-bit, +/-10V analog to digital converter. Its digital interface is composed of an 8-bit wide data bus and a clock line. The data is clocked out asynchronously with a maximum rate of 1MHz. Once programmed, at runtime one can select which channels are active as well as the sampling frequency. One can also select at runtime how many of the 8 data bus bits are used (the minimum is 2 bits) if the host doesn't have enough free lines. At the largest sampling rates with both channels enabled, the chip buffers 0.14 seconds of data. In addition to the ADC data, additional error information is sent along with every data points. A synchronization signal is sent every few data points to ensure the host stays synchronized with the board. Finally, as defined in the protocol, each bit is sent twice - the second time being the one's complement of the bit to reduce errors.

The CPL ADC Board is composed from an AD7732 ADC and an interfacing microcontroller – an AVR ATmega1284P. The AD7732 functions as the analog to digital converter while the microcontroller serves as the controller, buffer, and interface. The analog inputs are 3.5mm phone connectors, with the sleeve as the negative input and the tip as positive input. If the pin is not connected and the port is empty, both input on the ADC get shorted to ground. On the digital side, all pins have internal pull ups enabled (20-50 kOhms) when the board is not initialized.

Brief specifications

**Clock input**: Asynchronous, 1MHz maximum frequency.

**Secondary input**: pin 7 of the data bus is also used as a clock and input during initialization (see protocol).

**Data bus**: 2-8 output pins, clocked by clock pin from host asynchronously. The pins start counting down from pin 7 (see protocol).

**Automatic reset**: 2 sec duration after initialization.

**Port pull-up**: 20-50 kOhms.

**Reset pin pull-up**: 20-60 kOhms.

**Tclk,min**: the minimum pulse width.

Configuration

Hardware configuration

A host can vary the number of pins connected to the data bus.

Software configuration

After a reset or power up, the ADC board needs to be reconfigured. In the default state the ADC is shut down to conserve power and the board doesn't do any processing. As described in the protocol, before ADC starts the user supplies the board with 16 bits of configuration parameters.

Data point format

Programming the board

Protocol

Hardware setup

The board has an **8-bit data bus** as well as a single **clock line**. A host toggles the clock line in order to force the uC to update the data bus with new data. During the initialization sequence, one of the data bus pins becomes an input and is used by the host to initialize the uC in conjunction with the clock pin. The clock pin is always configured as an input (from the uC's point of view), while, except during initialization, all data bus pins are configured as outputs.

The user can select how many of the data bus pins will be used to send data. The minimum is 2 pins while the maximum is 8. Before and during initialization, it is assumed that only two pins are used. Following initialization, the number of pins indicated at initialization will be configured accordingly. The n pins connected start from pin 7. That is, if for example 3 data bus pins are used then these pins are pins 5-7. If 5 pins are used then they are pins 3-7. Because the minimum number of pins is 2, pins 6-7 will always be used. The unused pins, except before initialization, will also be set as outputs.

Pre-initialization

During power up all pins are tri-stated. Immediately following startup, all pins are set as inputs (including the data bus) which then enable their (internal) pull-ups. At this point, the ADC is powered down and the automatic reset is disabled (see reset). The uC will remain in this state until part one of the initialization sequence ends.

Initialization sequence

The initialization sequence is composed of three parts.

Part 1: Following startup or reset we wait for the host to toggle the clock 40 times with a low followed by a high. The count starts the first time the clock goes low and ends the 40th time that the clock goes high. When completed, we wait for pin 7 on the data bus to toggle 10 times low followed by high. This ensures that precise initial synchronization isn't required, because even if one toggles the clock line 200 times it will still wait until pin 7 is toggled 10 times. This means, if one doesn't know how many times the clock line has been toggled yet, but we know that pin 7 hasn't been toggled yet, then one just toggles the clock an additional 40 times followed by the toggling of pin 7 to arrive at the proper place.

Part 2: At the 10th high of pin 7 part 2 begins. In this phase, 16 bits of configuration data is sent to the board by the host. Also, automatic reset is enabled (see reset). When this part begins, pin 6 on the data bus is reconfigured as an output, while the other pins remain inputs. Therefore, the host should configure its pin connected to pin 6 as in input before starting this stage. During this stage, 16 bits are sent bit by bit, with most significant bit first. See the register configuration section for details on the bits. Three pins are involved in this transfer; the clock and pins 6-7. The clock clocks the data in, pin 7 functions as the data pin where the host writes the bits to the board, while the board mirrors on pin 6 what the host wrote on pin 7 for error checking.

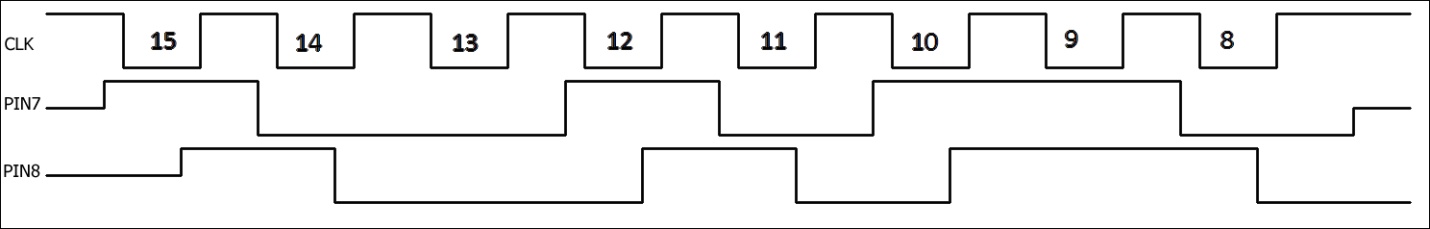
In particular, configuration is as follows (assuming we're running at the highest frequency – 1MHz). The host outputs bit 15 at pin 7 and pulls the clock low for 1uS. During this time, the board reads pin 7, saves the value and outputs the bit on pin 6. After the 1uS, the host reads pin 6 and pulls the clock high and waits another 1uS. The host can then compare the value it read on pin 6 to what it output on pin 7 for error analysis. It repeats this sequence for the rest of the 15 bits (see graph).

Figure 1 shows sequence for sending 0x96 (0b10010110), half of the 16 bits. Each clock width is minimum 1uS (16 uS shown).

After the 16 bits has been sent, the clock should be back high. At this point, pin 7 is still configured as input, pin 6 is output and the rest have pull-up enabled.

Part 3: In part three we configure the pins to their final states where all data bus pins are outputs. The board will wait for a single clock low and high toggle (1uS minimum each) before it will update the bus to outputs and start the ADC. Therefore, before proceeding with this toggle, the host should change all the pins connected to the board's data bus to inputs and be ready to starts retrieving data as described in the next section. At the end of this section the clock should be high.

Once initialization is complete, the microcontroller will power-up and configure the ADC and start collecting data. The host should immediately go into the data retrieval protocol in order to receive the data as they come in.

Retrieving data

During data retrieval, the data bus pins are all configured as outputs, while the clock pin is still an input. The host controls when data is shifted out by toggling the clock.

The general retrieval protocol involves the host toggling the clock line forcing the board to output new data, if available, on its data port. The host doesn't know if new data is ready, so it continually toggles the clock line and reads the data bus. Looking at the data read the host can extract the data, if present. Overall, the protocol is for the clock to go low, new data is output by the board (if no new data is ready, the data bus stays the same), the new data is read by the host after tclk,min and the host pulls the clock high. After the clock goes high, the board again outputs the same data as one's complement. After another tclk,min from when the clock went high, the host reads the complemented data and pulls the clock low again and new data is written by the board.

Table 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # pins connected | pins connected | ADC point bit width | # bytes/ data point | # trans/ data point | # trans/ cycle | # bytes/ cycle |
| 2 | 6-7 | 2 | 3 | 12 | 12 | 3 |
| 3 | 5-7 | 8 | 8 | 3 |
| 4 | 4-7 | 6 | 6 | 3 |
| 5 | 3-7 | 4.8 | 8 | 5 |
| 6 | 2-7 | 4 | 8 | 6 |
| 7 | 1-7 | 3.4 | 8 | 7 |
| 8 | 0-7 | 3 | 3 | 3 |
| 2 | 6-7 | 3 | 4 | 16 | 16 | 4 |
| 3 | 5-7 | 10.7 | 8 | 3 |
| 4 | 4-7 | 8 | 8 | 4 |
| 5 | 3-7 | 6.4 | 8 | 5 |
| 6 | 2-7 | 5.3 | 8 | 6 |
| 7 | 1-7 | 4.6 | 8 | 7 |
| 8 | 0-7 | 4 | 4 | 4 |

We call a single clock low and high toggle - a transaction. Because we can only send 8 bits maximum in a single transaction and because a data point is larger than 8 bits, it takes multiple transactions to complete a data point. In addition, as explained in configuration, there are 7 different configurations for the data bus because a host can select to only read from 2-8 of the data bus pins. Also, a data point can be 3 or 4 bytes. So in order to send a full data point, we must break it down into multiple transactions, each transaction sending 2-8 bits. However, we cannot always send a full data point exactly within n transactions. For example, if 7 pins are connected and a data point is 4 bytes, then after 4 transactions we are left with 4 bit to be sent which don't fit exactly into 7 bits. Therefore, we define a cycle as the number of transactions it takes to send a certain number of bytes and instead of using a data point when talking about being done sending, we use the cycle. In other words, after sending a full cycle, we consider that we are done and we go on to sending the next cycle. The exact number of bytes and transactions per cycle is defined in the table for the different configurations.

The reason for defining a cycle is to help with synchronization. As mentioned, each cycle is composed of a number of transactions with each transaction sending some bits and the one's complement of the bits. However, at the first transaction of each cycle, the bit on pin 7 (highest bit) is not complemented. So for example, if 3 pins are connected, then on the first transaction of the cycle when the clock goes high pin 5-6 will output the one's complement of the data that was there when the clock was low, while pin 7 will keep the same value from when the clock was low. This allows the host to know whether the board has lost synchronization with the host.

Each data point is composed of either 3 or 4 bytes as described in the data point format section. If only one ADC channel is enabled, then the data point is simply sent data point after data point. If both channels are enabled, then first we send the data point for channel 0, followed by the data point of channel 1, followed again by the next data point of channel 0 and so on. In all cases, the bits of a single data point are sent most significant bits first. Sometimes we have to patch together multiple data points to be able to complete a transaction.

For example, if 5 pins are connected and there are 3 bytes (24 bits) per data point, then the first transaction sends bits 19-23 on pins 3-7 respectively, the second sends bits 18-22, the third send bits 13-17, the forth sends bits 8-12, and the fifth sends bits 3-7. There are 3 bits remaining now. Therefore, we move on to the next data point (or we wait until ready if there's no other data point yet). On the sixth transaction, we send bits 0-2 on pins 5-7 and bits 22-23 of the next data point on pins 3-4. The seventh transaction sends bits 17-21 and the eighth sends bits 12-16. This completes a full cycle since we sent 8 bytes (see table). On the next transaction, since we are at the start of the cycle again, bit 7 will not be complemented.

The host clocks out the data by toggling the clock. However, if no new data is clocked for 2 seconds the board will reset to the uninitialized state. The minimum duration for each clock low or high pulse is 1uS, however, it could be increased if there are data corruption problems. Normally a host clocks continuously with some small breaks in order to get the data as fast as possible. However, if the sampling rate is low so that the amount of data to be transferred is small enough clocking could be reduced. These parameters are discussed in the clock rates section.

A host needs to be able to figure out when new data was output by the board is response to a clock toggle. For example, say all the data pins are connected and their initial state is 0x00. The data to be output on the next clock is 0xFF. When the clock is pulled low, the board quickly updates the pins to 0xFF. The host then reads the pins and pulls the clock high. In response, the board outputs the complement - 0x00 (or 0x80 if this is the first transaction of a cycle) which is read by the host before it pulls the clock low again. The host saw that before it pulled the clock high there was data x on the port, while the complement of x was present when it pulled the clock high. This signifies that new data was read. Now say the host continues clocking but no new data is ready to be sent, then the board will keep its output at 0x00. Now say the board now has new data and the next byte is 0x00. On the next low clock it will output 0x00, but this will still be indistinguishable from the previous data. However, when the clock goes high again the complement – 0xFF will be output so the host can recognize that new data was present. The point is that the host should not simply read the data the board puts out after the clock is pulled low because it might not be new data, it needs to compare the data output after the clock went low AND after the clock went high in order to be sure that new data was output. See the image for an example where the data point is 3 bytes, with all 8 pins connected and the data point to be sent is 0xF00FF0.Since a full cycle is shown, we also show that for the first transaction bit 7 is not complemented.

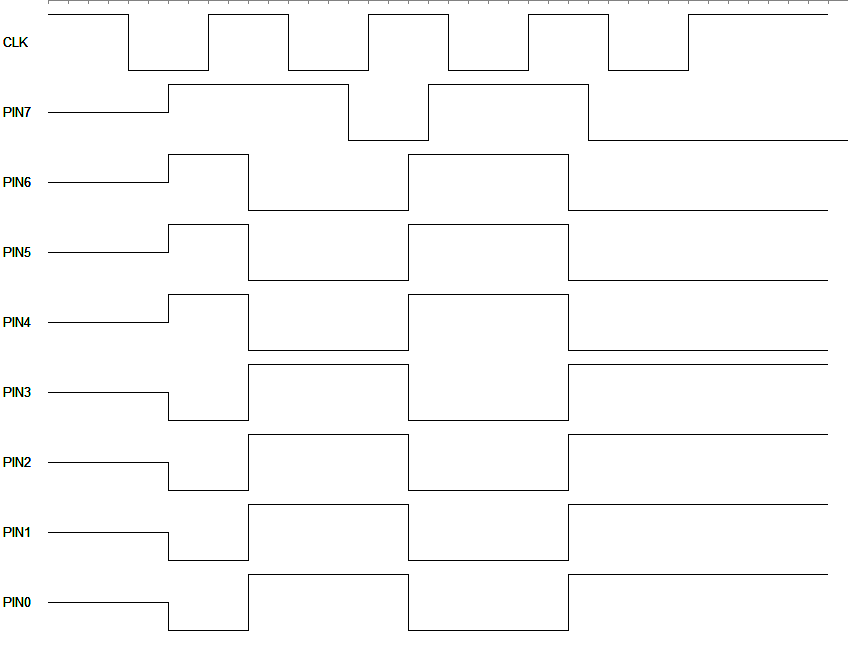


Figure 2

An additional check that the correct data was read comes from the format of the data point. As described elsewhere, the highest 8 bits of each data point is a register with error values. In particular, some bits must be zero, and if they are not we know data is corrupt. In addition, one of the bits describe whether the host collected the data too slowly and the buffer got overrun. If the buffer wasn't overrun, the assumption is that every data point follows the previous data point by 1/sampling rate.

Clock rates

Definitions

nChans: Is the number of analog channels currently enabled. Its values are 1-2, because either channel 0, channel 1 or both can be enabled.

nBytesPerPoint: Is the number of bytes per data point per channel, it could be 3 or four bytes (16 bit or 24 bit data, status register is always sent as described in data point section).

SPS: The ADC sampling rate in samples per second, e.g. 1000Hz.

uCRate: Is the clock rate of the microcontroller, i.e. the number of cycles the microcontroller cycles in a second, e.g. 20MHz.

CWasted: Is the number of cycles wasted per data point, see below.

TWasted: Is the total time the microcontroller wastes in a second, see below.

USBEffectiveRate: The effective rate at which data is clocked by the host.

USBRate: The actual rate at which the host clocks out data.

nBitsPerSent: Is the number of bits sent at a time from the ADC to host. This is equal to the number of data bus pins connected.

nBytesPerTrans: Is the number of bytes the host clocks out in a single transaction (transaction has a different meaning in this section as described below).

nUSBTrans: Is the number of high level transactions in a second carried out by the host.

USBFreeCyclesTrans: Is the estimated number of cycles (in host rate) per transaction that the ADC is actually sending data in response to the host toggling the clock.

USBFreeCycles: Is the total number of cycles (in host rate) that the ADC is actually sending data.

TBuffered: Is the maximum amount of time that the microcontroller can buffer ADC data before it overflows.

NCyclesData: Is the estimated total number of host cycles, for a given ADC sampling rate and data point bit width and nBitsPerSent that it takes to send a second of data.

NDataRepeated: The number of times each data point is repeated when sent to the host. This is 2 because every bit is sent twice (the bit and its complement).

Here we calculate the time the ADC is not available for communication with the host because it's collecting new data. The number of cycles the microcontroller wastes for each data point set (whether one or two channels are enabled, a set means both data points if two channels are enabled), CWasted, is nChans\*(nBytesPerPoint\*(6\*8+3)+5)+31+20. For example, if both channels are enabled and each channel has 4 bytes per data point the number of cycles wasted per data point is 2\*(4\*(6\*8+3)+5)+51 = 469 on average. The total time wasted in a second, TWasted, is CWasted\*SPS/uCRate. Running the ADC at the highest sampling rate results in about 15075 samples per second or 15075\*469= 7.1E6 wasted cycles in a second. Since the microcontroller is running at 20MHz, with each cycle taking 50ns the total time wasted is 50nS\*7.1E6= 354ms. In any interval, 35.4% of the time the ADC is not available for communication with the host because it's busy collecting new data.

Here we calculate the total number of cycles that the host is toggling the clock during which the board is sending data to the host. In particular, if the USB is a host, then even though the USB can toggle the clock at 1MHz, in a second at the most efficient settings only 8E5 cycles will be observed (at 1MHz). The reason is that the USB must start a new transaction when sending a new group of data, this setup time is where the USB wastes 2E6 cycles in a second. For a given USBRate and USB settings, the USBEffectiveRate, or actual number of cycles clocked is determined experimentally. For example, in FT2232H, if the baud rate is 0.2MHz, then USBRate is 1MHz and USBEffectiveRate is about 0.8MHz. This basically means that 20% of the time will be wasted on setup.

nBytesPerTrans is determined by the settings when setting up the ADC in Barst. The maximum for the FT2232H is 65280 bytes per transactions. At 0.8MHz effective rate, the number of transactions to clock out 0.8M bytes per second, nUSBTrans, is 0.8MHz/65280 is 12.26 or nUSBTrans= USBEffectiveRate /nBytesPerTrans. The total time each USB transaction takes is nBytesPerTrans/USBRate (65ms above, the total time being nBytesPerTrans/USBRate\*nUSBTrans, or 0.8s in the above case). Of this time, the amount of time the ADC is actually sending data is nBytesPerTrans/USBRate\*(1-SPS\*CWasted/uCRate), which using the above numbers is 42ms of the 65 ms. Now the number of USB cycles in the 42ms, USBFreeCyclesTrans, which is the cycles during which data is being communicated is nBytesPerTrans/USBRate\*(1-SPS\*CWasted/uCRate)\*USBRate, or 42.2kHz from the above numbers. USBFreeCycles is the total number of USB cycles during which data is communicated, which is USBFreeCyclesTrans\*nUSBTrans, or 517193Hz with the above numbers. Consequently, at the highest sampling rate, with both channels enabled and sending 24 bit numbers, the free USB cycles is about 517193Hz.

Using the same numbers from above, if each data point is 4 bytes wide, both channels are enabled, and 7 bits of data is sent at a time, the total number of USB cycles it takes to send a second of data is nBytesPerPoint\*8/nBitsPerSent\*nChans\*SPS\*NDataRepeated, or 4\*8/7\*2\*15075\*2= 275657 cycles. This means about half the cycles the USB clocks out are unused. If only 4 bits are sent at a time, then the cycles increase to 482400, which is almost at capacity. Decreasing the data point width from 24 bits to 16 bits should remedy this.

Reset timeout

Automatic reset, or watchdog reset, when enabled will reset the board into the uninitialized state like after power-up if a data point is not sent from the board to the host for 2 seconds. That is, if a data point hasn't been sent by the board for 2 seconds, then the board assumes that it's not connected to a host anymore and will reset itself and place itself into conditions following power-up, i.e. it will go through pre-initialization and wait at part 1 of initialization. To prevent this timeout the host must continually read data. However, automatic reset is only enabled once part 1 of the initialization sequence is complete to ensure we don't continually reset if no host is connected. One also uses the reset feature if in the middle of data collection one loses synchronization or something goes wrong. Stopping to write to the board will force it to reset which will bring it back to a clearly defined state.

Errata

The 3.5 mm input channels are marked flipped.