TCP/UDP Control of the Drift Channels

William E. Earle

May 16, 2007

This document describes the TCP and UDP data transfers needed to control up to 256 low-power drift channels. Each of these channels consists of charge input circuitry, an A/D converter, and 16k words of memory for storing A/D conversions at a rate of about 700 kilosamples/sec. The TCP/UDP interface controls starting and stopping of the channel conversions, setup of channel operating conditions, and readback of the stored A/D conversions.

Data transfers to/from channel memory use TCP but all other communication with the channels occurs via reading and writing registers using UDP.

TCP data transfers do not occur directly with the channel memories but rather via two FIFOs, a write FIFO for writing data to a channel memory and a read FIFO for reading memory data. That is, to write data via TCP to a channel memory (e.g. for test purposes), a TCP transfer is used to first write the data to the write FIFO and then a command register accessed via UDP is loaded with a command that causes the data in that FIFO to be transferred to the channel.

Similarly, to read memory data via TCP, the command register is loaded with a command that causes data stored in the memory of a particular channel to be read and stored in the read FIFO, then a TCP transfer reads the contents of that FIFO.

UDP transfers make use of 12 different byte-wide registers as shown in the attached table. The command register is the most important, since only when this register is loaded with a command does any communication with a channel occur. There are ten valid commands for this register, each command coded as a four-bit value in the lower four bits. The most significant bit of the command register is used to specify that the command refers to all channels or to just one channel. Being able to address all channels is particularly useful
when setting up the channel operating conditions before starting actual data collecting, since one rather than multiple commands suffices to write data to all channels.

To specify an individual channel, the register “channel” shown in the table must be loaded with the channel number. Although information can be sent to all channels simultaneously, any information that is read back always comes from the single channel specified by the “channel” register.

1 General Operation of Each Channel

Before describing the details of communicating with the channels, it’s important to understand the operating characteristics of each channel.

The analog signal to the A/D converter (from the amplifier following the charge preamplifier) is converted to a 12 bit unsigned digital number, one conversion for every 17 cycles of the A/D input clock. Initially this clock is set to 12.5 MHz, so the time per conversion is 1360 ns. This time can be lengthened (but not shortened, because of timing constraints of the A/D converter and the RAM memory) by reducing the clock frequency and/or increasing the number of clock cycles per A/D conversion. At the current 1360 ns per conversion, continuous A/D conversions will fill the entire 16,384 word memory in slightly more than 22 mS.

If a channel has been “started”, meaning it has been commanded to start storing A/D conversions in its RAM, then the RAM address pointer is set to zero and 16 bit A/D samples are stored in the RAM, one sample per 1360 ns. Each A/D conversion produces a 12 bit number, but the output from the A/D converter is actually a 16 bit value, the upper 4 bits being zeros.

Two conditions will cause a channel to stop storing A/D conversions in its RAM, a “stop” command sent via the UDP interface, or detection of a trigger condition. These operate differently in that a “Stop” command causes an immediate stop of the sample storage, whereas the detection of a trigger condition will allow data storage to continue for an additional 1024 post samples before stopping. Each channel contains a 4 bit parameter register that must be initialized before “starting” the channel. This register specifies the bit number of the 12 bit digitized sample that, when set or exceeded, will generate a trigger condition. For instance, if this value is 5, then a sample with any of bits 5 through 12 set will generate a trigger condition. Setting this value to 13 or higher will prevent any trigger condition from
being generated for that channel and setting it to zero will mean that every sample will generate a trigger condition.

The trigger signals of all channels are connected together, so that a trigger originating in any channel appears as a trigger signal in all channels, ultimately causing all channels to cease data-collecting and “stop”, after the 1024 post-samples have been stored.

Once data-collecting has been started, the quantity of data stored in a RAM can easily exceed the available 16,384 words before a stop occurs, so it is likely that the RAM data has been overwritten multiple times. Thus, in order to be able to read back the particular sample that caused the trigger signal which resulted in the eventual “stop”, the memory address when the stop occurred needs to be saved in a register in each channel. By reading this “stop” address and then subtracting the known number of post-samples, one can determine the memory address of the sample that caused the trigger signal.

Unfortunately, since any channel could have caused the trigger signal, assuming multiple channels were running, one might have to read back the stored data from every channel just to find the particular channel that caused the trigger. To avoid having to do just that, the channel that caused the trigger signal provides a readable trigger bit so that the triggering channel can be identified without the need to read back any sample data. Only the channel that caused the trigger signal will have its trigger bit set.

In addition to a trigger bit, a stop bit for each channel can also be read back to determine if a channel has stopped. Normally, once started, a channel stop bit should be monitored to determine when that channel has generated (or received from another channel) a trigger signal that ultimately caused the channel to stop collecting sample data. Although a stop bit can only be read back from one channel at a time, all channels should stop simultaneously (unless the post-sample count differs amongst channels) so reading the stop bit from a single channel should suffice to determine when all channels have stopped saving A/D data.

The starting address for reading or writing RAM data is specified by the concatenation of two registers, “ram_add_h” and “ram_add_l” in the table of registers. Only the lower 14 bits are significant.

Data read back from a RAM is first read from the RAM to a read FIFO, the number of words to be transferred stored in two UDP-accessible registers called “word_count_h” and “word_count_l”, only the lower 14 bits of the concatenated registers being significant.
The 16k serial RAM contains an internal 8 bit status register that has to be initialized before the RAM can be used. Two of these bits set the mode, which should be Burst Mode and another bit determines whether the “HOLD” function should be allowed, which it should be. This register should be set to 40 hex for all channels.

Whenever the command register is loaded with a command, the “busy bit” of the “busy” register goes high and remains high until the command has finished executing. A new command should not be sent to the command register until this bit has returned low.

2 The Commands

The functions and bit codes, in hex, of the 10 possible commands sent to the Spartan 3 via UDP writes to the command register are:

- **stop**: 0: Stops sample storage for one or all channels.
- **start**: 1: Starts sample storage for one or all channels.
- **status_read**: 2: Reads the RAM status byte from the channel specified by the channel register. The data appears in the UDP-accessible register “data_in_l”.
- **status_write**: 3: Writes the value in the UDP-accessible register “data_out” to the RAM status register of one or all channels. This value must be 40 hex.
- **params_write**: 4: Writes the lower four bits of the “data_out” register to the parameter register of one or all channels. These bits specify the bit number of the sample that will cause a trigger signal to be generated.
- **params_read**: 5: Reads the four bit parameter register of the channel specified by the channel register to the lower four bits of register “data_in_l”. The upper four bits are cleared.
- **stop_add_read**: 6: Reads the 14 bit value of the stop address of the channel specified by the channel register to the concatenated registers “data_in_h” and “data_in_l”, setting the upper two bits to 0.
- **flags_read**: 7: Reads the trigger and stop bits from the channel specified by the channel register to the lower two bits of the “data_in_l” register, the trigger bit as bit 0 and the stop bit as bit 1.

- **ram_write**: 8: Causes the data in the write FIFO to be written to the RAMs of one or all channels. A TCP transfer should have first loaded the FIFO with data to be written. Data will be read from the FIFO and written to the RAM(s) until the empty flag of the FIFO is true.

- **ram_read**: 9: Reads RAM data from the channel specified by the channel register, storing it in the read FIFO. The number of words to be read is specified by the lower 14 bits of the concatenated registers “word_count_h” and “word_count_l”.

### 3 Typical Operating Procedure

A partial typical operating procedure for using the channels to collect sample data might be as follows:

1. Write 40 hex to register “data_out”. This is the byte that will be written to the RAM status register of all channels.

2. Continually read the busy bit, the LSB of the busy register, until it is 0, indicating that the command has finished.

3. Write the command “status_write” to the command register, with the MSB set to 0 so as to write to all channels.

4. Wait until “busy bit” is 0.

5. Write the desired parameter value to register “data_out”.

6. Wait until “busy bit” is 0.

7. Write the command “params_write” to the command register, with the MSB set to 0 so as to write to all channels.

8. Wait until “busy bit” is 0.

9. Write the command “start_a2d”: to the command register, with the MSB set to 0 so as to start all channels.
10. Wait until “busy bit” is 0.

11. Write the command “flags_read” to the command register. The MSB is irrelevant.

12. Wait until “busy bit” is 0.

13. Read register “data_in_l” and check the LSB. If 0, no trigger has occurred, so repeat steps 11 through 13. If 1, a trigger has occurred and all channels have stopped.

Once the A/D has been started by command 1, only the stop command (command 0) or the read-flags command (command 7) should be used until the A/D has returned to its stopped state, as indicated by a ‘1’ for the stop bit returned by the read-flags command. The red/green LED’s also show the stop/run state.

At this point, one can simply read the sample data for all channels, making sure to start the readout somewhat ahead of the trigger event. This requires reading back the RAM address where data-collecting stopped and subtracting the post-sample count. Or, one can simply read all 16k words of sample data from all channels and use one of the RAM stop addresses to determine where the trigger occurred.

Another possibility is to determine which channel caused the trigger, by sequentially interrogating the trigger bits (this requires changing the channel numbers between reading the trigger bits), and then just read the data for that channel.

4 Testing the RAM

At startup, it may be desirable to test the ability to write and read RAM data for all channels (a memory test). To do this, 40 hex should first be written to the status register of all channels and then be read back from each channel to verify that it was written correctly. Then, test data should be written to the RAMs of all channels; all 16,384 locations should be written. This test data should then be read back from each channel to verify that all locations of all RAMs can be written and read correctly.
<table>
<thead>
<tr>
<th>UDP Address</th>
<th>Register</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>command</td>
<td>Read/Write</td>
<td>4 bit command to be executed</td>
</tr>
<tr>
<td>1</td>
<td>data_out</td>
<td>Read/Write</td>
<td>RAM status byte or parameters for output</td>
</tr>
<tr>
<td>2</td>
<td>channel</td>
<td>Read/Write</td>
<td>Channel number, 0 to 256</td>
</tr>
<tr>
<td>3</td>
<td>word_count_l</td>
<td>Read/Write</td>
<td>Lower byte of word count</td>
</tr>
<tr>
<td>4</td>
<td>word_count_h</td>
<td>Read/Write</td>
<td>Upper 6 bits of word count</td>
</tr>
<tr>
<td>5</td>
<td>ram_add_l</td>
<td>Read/Write</td>
<td>Lower byte of RAM address</td>
</tr>
<tr>
<td>6</td>
<td>ram_add_h</td>
<td>Read/Write</td>
<td>Upper 6 bits of RAM address</td>
</tr>
<tr>
<td>7</td>
<td>data_in_l</td>
<td>Read-only</td>
<td>Lower byte of readback data</td>
</tr>
<tr>
<td>8</td>
<td>data_in_h</td>
<td>Read-only</td>
<td>Upper byte of readback data</td>
</tr>
<tr>
<td>9</td>
<td>busy</td>
<td>Read-only</td>
<td>LSB is busy bit</td>
</tr>
</tbody>
</table>

UDP-addressable registers