Serial Configuration Bus Protocol Specification for the ATLAS LArg FEB's SCA Controller and Gain Selector ASICs

Stephan Böttcher Philippe Cros Douglas Gingrich

$Id: protocol.tex,v 1.5 2000/06/13 03:31:55 stephan Exp $

Abstract

This document describes the protocol, hardware levels, and address space of the serial configuration bus that is used to transfer configuration parameters and status readback to and from the ASICs on the FEB.

1 Bus topology

The configuration bus is driven by a single master, and drives multiple receivers. The *master* resides the *Configuration Controller* chip, which interfaces the parallel port of the SPAC slave to the service requirements of the FEB.

The *receivers* are the configurable ASICs on the FEB: The SCA Controller and the Gain Selector chips.

The Configuration Controller provides four independent configuration buses, two for each half of the board. The SCA Controller and Gain Selectors of each half board may share a bus or use indepemdent busses.

The master must provide the ability to send write frames to both halfs of the FEB simultaniousely, with identical address and data, in addition to sending frames to a selected bus. This allows to sychronously reset the SCA controllers, or initiate test triggers.

2 Serial transport

Each configuration bus consists of a single write signal, driven by the master, and a single readback signal, driven by the receiver that is addressed for readback. The data is transfered bit serial, synchronous to the 5 MHz readout clock (RCLK).

In the scope of this document, the *rising edge* of RCLK is defined as the RCLK clock edge that is close in phase with the active readout clock edge of the SCA.

The configuration master asserts each bit, and samples the readback bits, on the rising edge of RCLK. The receivers sample the serial bits, and drive the readback bits, on the falling edge of RCLK. This arrangement ensures close to 100 ns setup and hold time for every signal.

Rationale for the clock edge assignment: The Gain Selectors need to operate on the negative RCLK edge, and the config operation is closely coupled to the internal timing of the autogain scan operation within the chip. Disadvantage: The SCA controller needs to operate the serial sample command interface to the Gain Selectors on the rising edge of RCLK, so this chip needs to cope with diferent RCLK edges, but in different funtional blocks.

3 Electrical Specifications

Each bus signal is driven and received differentially on two lines. The voltage swing on the bus will be about 400 mV. The common mode voltage will be in mid-rail, about 2.5 V, assuming that all chips are implemented in DMILL. The drivers on the chips will drive rail-to-rail CMOS levels. The voltage swing will be adjusted by external resistive dividers. DMILL receiver implementations may need to cope with common mode input well below mid-rail (LVDS), e.g., in case the SCA Controller is implemented in $0.25\,\mu$ CMOS.

4 Serial frame format

All transfers are initiated by the master.

The master can initiate one of two operations: *write* or *read*. Each operation is initiated by sending an address frame. In case of a write operation, the address frame is immediately followed by the data bits.

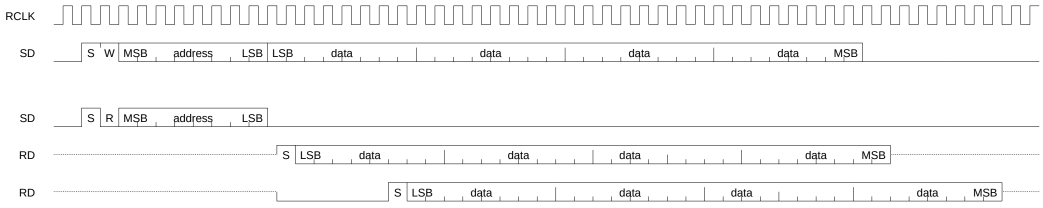

Figure 1: Serial configuration protocol. The topmost waveform is the readout clock, with the rising edge close in phase to the active SCA clock edge. The second waveform shows a write operation. The data follows the address frame immediately. The third waveform shows a readback address frame. The last two waveforms show the earliest and a slightly delayed response from the receiver.

The address frame consists of a start bit, a direction bit, and eight address bits.

The data word size is 32 bits. The receivers may ignore any number of data bits, but should mask start bit detection until the full 32 bit data frame is passed.

The start bit is always high. The direction bit tells if it is a write or read operation. High means write, low means read. The write line is driven low when the bus is idle.

In case of a read operation, the addressed receivers drives the readback signal low until it is ready to return the data. When ready, the receiver sends a start bit immediately followed by data bits. The earliest bit position that is accepted as a readback start bit is two clock cycles after the last address bit is asserted, i.e., one cycle after the last address bit is completed. That means that the addressed receiver does not need to drive the readback signal until one clock cycle after it sampled the last address bit.

The master must timeout a read operation after 32 clock cycles.

Fig. 1 illustrates the frame format.

The rational for allowing variable response time to readback requests is that the Gain Selectors may not be able to provide the readback data immediately, in case that an autogain scan operation is in progress. The maximum delay that may occur for this reason is 8 clock cycles.

4.1 Bit endianess

The addresses are sent MSB first, and the data LSB first. The rationale for MSB first address is that a receiver may take advantage of seeing the MSB bits early and decide if it is addressed or not.

The rationale for LSB first data is that the receiver may ignore any number of data bits, but those bits that are used should a) be sent first, and b) be the least significant bits of the full word.

5 Address space

The second address bit of the address frame, $A[6]$, distinguishes the SCA Controller from the Gain Selectors. If $A[6]$ is one, the SCA Controller is addressed, if it is zero, a Gain Selector is addressed.

Address bit $A[7]$ is intendend to signal broadcast commands, which cause some synchronous event to happen (test trigger, BCID reset, EVN reset, SCA initialization).

6 Appendix: Sample implementations

The following pages present reference implementations in Verilog HDL, for a serial configuration master and a receiver.

```verilog
// $Id: conf_master.v,v 1.5 2001/01/16 00:37:34 stephan Exp $

// FEB configuration bus master
// The spac chip sends (dir/addr/[w_data]) with a write strobe to
// the conf_master.
//
// For read frames, the conf_master will listen to the return data line
// for a start bit, which must be immediately followed by data bits,
// LSB first.  The start bit must arrive withing 32 clock cycles, else the
// frame will aborted with a timeout error.
//
// Any READ or WRITE strobes from the spac during an active frame will
// flaged as busy ERRORs, and ignored.
//
// The SPAC can read status and readback data from the R_DATA lines at will.
// The READ strobe is just for busy ERROR checking.
//
// The 'clk' must be opposite polarity as for the receiver.  Since the
// gain selector runs on clk=~RCLK, this must be clk=RCLK.

module conf_master(clk,     // 5 MHz clock
                   write,   // write strobe for parallel dir, addr, and w_data
                   read,    // read strobe, for r_data, error check only
                   dir,     // data direction, 1=write
                   addr,    // parallel address, 7 bits
                   w_data,  // parallel data bus 32 bit
                   r_data,  // parallel readback bus 32 bit
                   error,   // busy error
                   timeout, // timeout error, no client
                   sout,    // serial output, to FEB
                   sin      // serial input, from FEB
                   );

   input          clk;
   input          write, read;
   input          dir;
   input  [7:0]   addr;
   input  [31:0]  w_data;
   output [31:0]  r_data;
   output         error;
   output         timeout;

   output         sout;
   input          sin;

   reg            error, timeout;


   // The counter CNT counts
   //   a) the addr sub-frame bits
   //   b) the data bits, read and write
   //   c) the read timeout.
   // The counter is started at the beginning of the frame.
   // The counter is restarted at the end of the ADDR sub-frame.
   // The counter is restarted when a start-bit is recieved in a read frame.
   // The frame terminates when CNT==31.

   reg [4:0]      cnt;  // bit counter

   reg            a_frame;                           // addr sub-frame active
   wire           acnt_tc = cnt[3]&cnt[1] & a_frame; // end of addr sub-frame
   wire           frame   = ~& cnt;                  // frame active
   wire           s_frame = ~& cnt[4:1];             // same, one clock shorter

   // frame start pulse.  Cope with arbitrary WRITE pulses.

   wire           frame_start;    // frame start phase
   reg            frame1, frame2; // edge detection

   always @(posedge clk)
     frame1 <= write & ~frame;
   always @(posedge clk)
     frame2 <= frame1;
   assign frame_start = ~frame1 & frame2;

   // The shift register.
   // serial in, serial out
   // parallel in, (parallel out, d_sr only)
   //
   // For read frames, the shift register is split, to prevent
```

```
   // bogus data bits at the output.  For write frames, the serial
   // input is disabled, to force the output to 0 at the end of the frame.

   reg            rw;   // stores the direction bit
   reg [10:0]     a_sr; // addr frame shift register
   reg [31:0]     d_sr; // data frame shift register

   always @(posedge clk)
     if (write & ~frame)
       begin
          d_sr <= w_data;
          a_sr <= {2'b01,dir,addr}; // MSB first
          rw <= dir;
       end
     else if (frame_start | frame)
       begin
          a_sr <= {a_sr[9:0], rw & d_sr[0]}; // MSB first
          d_sr <= {!rw & sin, d_sr[31:1]};   // LSB first
       end

   assign r_data = d_sr;
   assign sout = a_sr[10];

`ifdef SIMULATION
   initial a_sr <= 0;
   initial d_sr <= 0;
`endif

   // The start_bit is recognised during the timeout period, up to one
   // cycle before the end of the timeout 'frame'.

   wire           start_bit = timeout & d_sr[31] & s_frame;

   // The counter alway counts to 31 and then stops. This ensure proper
   // init behaviour.

   always @(posedge clk)
     if (frame_start | acnt_tc | start_bit)
       cnt <= 0;
     else if (frame)
       cnt <= cnt + 1;

`ifdef SIMULATION
   initial cnt = 17; // some arbitrary value
`endif

   // The a_frame JKFF

   always @(posedge clk)
     if (frame_start)
       a_frame <= 1;
     else if (acnt_tc)
       a_frame <= 0;

   // the timeout period starts at the end of the addr frame, and
   // stops by a start_bit.  frame_start clears a timeout form a
   // previous cycle

   always @(posedge clk)
     if (acnt_tc & ~rw)
       timeout <= 1;
     else if (frame_start | start_bit)
       timeout <= 0;

   // Any spac access during a frame is a busy error.

   always @(posedge clk)
     if (write | read)
       error <= frame;

endmodule // conf_master
```

```
// $Id: conf_client.v,v 1.7 2001/01/16 00:36:26 stephan Exp $

// FEB configuration bus client.

module conf_client
  ( clk,
    din, dout, oe,
    dir, addr, me,
    sw_data, sr_data, s_shift,
    write_request, write_strobe, read_request, read_strobe
    );

   input        clk;              // 5 MHz clk

   input        din;              // serial input from  master
   output       dout;             // serial output to master
   output       oe;               // output enable for dout

   output       dir;              // 8 bit address to user
   output [7:0] addr;             // write bit
   input        me;               // from user, if he feels like talking

   output       sw_data;          // serial output to user
   input        sr_data;          // serial input from user
   output       s_shift;          // serial shift enable to user

   output       write_request;    // write request to user
   output       write_strobe;     // write strobe to user
   output       read_request;     // read request to user
   input        read_strobe;      // read strobe from user

   // The addr frame shift register, in three parts:

   reg          start;  // start bit
   reg          dir;    // direction bit
   reg [7:0]    addr;   // 8 bit address, MSB first

 `ifdef SIMULATION
   initial start <= 0;
   initial dir <= 0;
   initial addr <= 8'b01010101;
 `endif

   // Collect addr frame.
   // Detect start bit.
   // Wait for read to complete.
   // Wait for the data frame to comlete.
   // Reset addr frame.

   wire         stop_cnt;  // end of data frame phase
   wire         start_cnt; // start of data frame phase

   always @(posedge clk)
     if (stop_cnt)
       {addr,dir,start} <= 0;
     else if (!start)
       {start,dir,addr} <= {dir,addr,din};

   // The data bit counter, counts cycles of the d_frame.
   // The d_frame is 32 cycles long

   reg [4:0]    cnt;
   wire         cnt_tc = & cnt;
   wire         d_frame = ~cnt_tc | start_cnt;
   always @(posedge clk)
     if (d_frame)
       cnt <= cnt + 1;

 `ifdef SIMULATION
   initial cnt <= 5;
 `endif

   // Waiting for read

   wire         frame_wait = start & cnt_tc & ~stop_cnt;
   wire         read_me   = me & ~dir;
   wire         read_wait = read_me & frame_wait;

   // The d_frame is sent to the user, if she feels addressed.

```

```
   assign       s_shift = d_frame & me;

   // read1 delays the read_strobe to reset the counter
   reg          read1;
   always @(posedge clk)
     read1 <= read_strobe & read_wait;

   // d_frame1 detects the cycle after the d_frame.
   reg          d_frame1;
   always @(posedge clk)
     d_frame1 <= ~cnt_tc;

   // The frame stops when the counter is done.

   assign       stop_cnt = d_frame1 & cnt_tc;

   // The frame starts when
   //  read1   the USER is ready to deliver bits,
   //  ~me     if the USER is not addressed,
   //  dir     immediately, if it's a write cycle.

   assign       start_cnt = ~read_me & frame_wait  |  read1;

   // The write_strobe goes to the user immediately after the d_frame.

   assign       write_strobe = stop_cnt & me & dir;

   // The read_strobe from the user goes directly onto the output
   // as a start bit.
   //
   // Both the 'dout' and 'oe' signals are supposed to be retimed at a
   // 'clk' edge, at the chip boundary.

   assign       dout = (read_strobe & ~read1) | (sr_data & d_frame);
   assign       oe = read_me & start;

   // The serial in goes straight to the user.
   // 2000-05-03: netlist-1a: SiB:
   //   Added an and gate, to avoid a feedthrough.

   assign       sw_data = din & s_shift;

   // Tell the user to get ready for a read or write

   assign       read_request  = start & read_wait;
   assign       write_request = start & me & dir;

`ifdef SIMULATION
   // !! This is actually unused, please replicate this in the testjig!
   initial
     // force the addr sr and the word cntr
     // into a defined state, any state except X
     begin
        cnt = 19;  // some arbitrary value
        start = 1;
     end
`endif

   // Power up and lockup analysis:
   //
   // The relevant registers that hold corruptable state are
   //
   //  cnt                data bit counter
   //  {start,dir,addr}   address frame register
   //
   // The idle state that must be reached from any
   // state in finite time, while the input is idle, is:
   //
   //   {start,dir,addr} == 0
   //   cnt_tc = &cnt
   //
   // Possible transient states:
   //
   //  {start,dir,addr} == 0
   //  cnt arbitrary
   //      --> cnt will count until cnt_tc, and stop.
   //      --> IDLE state reached
   //
   //  {dir,addr} != 0
   //  start == 0
```

```
    //  cnt arbitrary
    //     --> address frame will shift
    //     --> start==1
    //
    //  start == 1
    //  dir == 1 (write)
    //  cnt_tc
    //     --> start_cnt
    //     --> !cnt_tc
    //
    //  start == 1
    //  dir == 0 (read)
    //  !me
    //  cnt_tc
    //     --> start_cnt
    //     --> !cnt_tc
    //
    //  start == 1
    //  dir == 0 (read)
    //  me
    //  cnt_tc
    //     --> read_request will be asserted
    //     --> the chip MUST respond with a read_strobe
    //     --> start_cnt
    //     --> !cnt_tc
    //
    //  start == 1
    //  !cnt_tc
    //     --> cnt will reach cnt_tc, stop_cnt will reset the addr frame
    //     --> IDLE state reached

endmodule // conf_client
```