User Manual
FPGA-based control system
buffer board versions v1.2-v1.4
Trenkwalder Andreas (andit0815@gmail.com)
19/12/2024, Firenze, Italy

1. Introduction

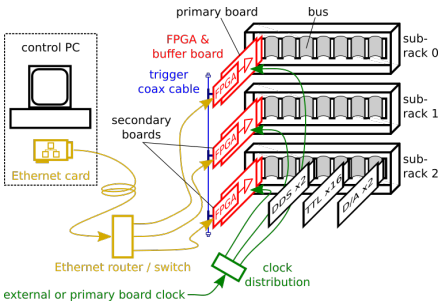
The FPGA-based control system is a replacement and upgrade of the outdated and not anymore supported DIO-64 card from Viewpoint Systems (referred as "old control system"). While the DIO-64 card must be installed into a PCI (not PCIe) slot of the control computer and is connected via a ribbon cable to the hardware rack, the Cora-Z7 is a single-board computer (also known as "embedded system") which runs independently of the control computer and is directly installed in the hardware rack. Data between the control computer and the Cora-Z7 is exchanged via Gigabit Ethernet. A rack contains one or several sub-racks which contains the electronics devices for controlling the experiment, like digital and analog outputs and direct-digital-synthesizers (DDS). Each sub-rack contains one FPGA board which communicates via the backplane bus of the sub-rack with these devices.

This configuration has several advantages over the previous system: the control computer does not need to have a device driver for the board installed, which permits to use any operating system of choice. The control software does not need to communicate with the device driver directly, which allows to choose from a broader spectrum of control software and facilitates development. The only requirement is the ability to communicate via Ethernet. During the execution of the experiment, the control computer is much less busy than before where the driver had to update rather small DMA buffers for the DIO-64 card over the rather slow PCI bus. Now the control computer receives during the experimental run only status information at a programmable rate (typically 20Hz) and waits for the experiment to finish. This significantly reduces performance demands on the computer, allows to do data analysis directly on the control computer. On the hardware side no outdated PCI slot is required, the control computer can be at a much larger distance from the experiment and electronic isolation is intrinsically ensured via Ethernet's decoupling transformers (called "magnetics").

The FPGA control system consists of the low-cost FPGA development board (Cora-Z7-07S or Cora-Z7-10 from Digilent Inc.), and of a buffer card which hosts the FPGA development board and buffers to interface with the hardware in the sub-rack. This buffer card replaces the former opto-coupler card inserted into the sub-rack. The actual versions of the control system (v1.2-v1.4) have essentially the same firmware, only the buffer card differs. Version 1.2 is a simpler version with fewer buffers, versions 1.3 and 1.4 allow to control two sub-racks with a single FPGA development board (but require two buffer cards). Version 1.4 vs. 1.3 has optional decoupling resistors and a differential clock signal after the clock buffer into the FPGA development board. Both features are for better resilience against external perturbations like electrical spikes which can cause the external clock signal to be lost for short times.

Several FPGA development boards can be synchronized within 10ns. For this purpose the buffer card provides edge connectors with external clock input and output and trigger input and output. All boards can be locked to one common external clock. Alternatively, one of the boards provides its internal clock to the other boards. This so-called primary board also sends a trigger signal to all other boards via a single

Figure 1, system overview



coaxial cable to synchronous start execution of an experiment.

Run-time delays of the trigger signal and clock phase shifts can be corrected by software. The largest system so far tested consists of two development boards at a separation of 30m and synchronization on the 1ns could be achieved with a more advanced automatic synchronization scheme (see Ref. [1]).

See the glossary for the used terms and additional information.

2. Prerequisites

Basically, each sub-rack requires one FPGA board and one buffer card. When two sub-racks are less than about 3m apart it is possible to control two sub-racks with a single FPGA development board. In this case still two buffer cards version 1.3 or 1.4 are needed and a flat-ribbon cable (2x32, 1.27" spacing) with two headers (female) have to be provided to connect the two racks (for details see section 4.2 below). Each FPGA board needs an Ethernet cable and one SD card (8GB is more than sufficient, use high-quality cards like the ones recommended for Raspberry Pi) and a short power supply cable with 4.0mm jack at 90°. See table 1. If more than one FPGA development boards are used, the boards need to be synchronized for which additional two coaxial cables (with each 2x SMA connectors) and an Ethernet switch or router is needed to communicate between the control computer and all development boards.

Table 1
bill of material per FPGA development board (controls 1x sub-rack + optional 2nd sub-rack nearby)
1x Cora-Z7-10 or Cora-Z7-07S
1x buffer card v1.2-v1.4 (v1.3 or v1.4 needed when controlling 2nd sub-rack)
1x SD card (high quality)
1x power supply cable (>10cm, 4.0mm jack, 90°, RS 656-3850)
1x Ethernet cable
optional in addition to control a second nearby sub-rack:
1x buffer card v1.3 or v1.4
1x 2x32pin flat-ribbon cable, 1.27" spacing
2x 2x32 female headers
1x Ethernet cable
1x Ethernet switch or router

3. Power:

The FPGA board can be powered via a 4mm power Jack (we use this with the buffer card, see below) or via USB. The jumper JP3 must be set to "EXT" or "USB". In both cases it requires 5V(+/-1V) DC with about 0.5A steady state current (including the buffer card with all clocks active) but at boot time the current might be higher. The manufacturer (Digilent) recommends at least 1A which is more than the USB (2.0) can probide, however, without buffer card and not too long USB cable I could easily work with the board also on USB. Be careful not to exceed +6V, otherwise the board might be permanently damaged. When the voltage is too low the board either does not boot or randomly stops responding. When the board is powered, ensure the red LED close to the Jack (LD7) switches on.

4. Booting:

The board can be directly programmed via USB (internally using JTAG) without additional hardware using the Xilinx Vivado software (Vitis or its predecessor Eclipse/SDK), but the board does not have an onboard flash memory such that it needs to be re-programmed after each power cycle. But it has a micro-SD card slot which allows to program it "automatically" after powering up. This is the configuration we are using. Ensure the jumper JP2 "mode" is shortened and an SD card, formatted with "FAT32" (sometimes also labeled a "DOS") is inserted and has at least these 3 files: BOOT.bin, boot.scr and image.ub. The BOOT.bin file contains the bootloader (u-boot) and the bitfile which the bootloader uses to program the FPGA. The boot.scr file is a configuration file for the bootloader. The image.ub file contains the linux image, i.e. the file system, which the bootloader loads after the FPGA is configured. I usually provide another file, server.config, which allows user-configuration of specific features of the board, like the IP address, see section configuration file below. All four files can be found in the "firmware" section of my github page. Choose the files for your board (Cora-Z7-07S or Cora-Z7-10) and for your buffer board version. After the board has successfully booted the second red LED "Done" close to the USB host port (opposite to the power Jack) port (LD6) should be on. Depending on the firmware version other 3-color LEDs might be on as well. When this is not the case, check that the SD card is properly inserted and the correct files for your board are on the SD card. Sometimes, especially with old SD cards, newly formatting the SD card helps.

4. Clocking:

The internal time-base of the FPGA board is fixed f_{system} = 100MHz, i.e. 10ns per cycle. This gives the maximum time resolution the board can achieve with the standard firmware. The bus output sampling rate is generated by dividing f_{system} by the content of the clock divider register f_{bus} = 100MHz/clk_div[7:0], i.e. when clk_div[7:0] = 100 the bus output sampling rate f_{bus} = 1MHz. This is the default f_{bus} for most of our experiments.

The maximum tested f_{bus} = 20MHz with clock divider = 5 works without restrictions, only the strobe timing has to be adapted (see below). For f_{bus} = 25MHz with clock divider = 4 the output is still possible but the normal strobe generation does not work anymore and a different strobe generation, like toggle trigger is required (see the strobe section below).

Above 10MHz up to about 200MHz a real clock output can be realized, but requires fixed output frequency given at compile time of the firmware. I.e. the clock divider register cannot be used anymore to change the output rate. In this case the PLL (phase-locked loop) of the board must be programmed directly which allows to change the phase, i.e. setup time, at run-time but not so easily to change the frequency, although its possible. If you have specific clocking requests please let me know and we can find feasible solutions and adapt the firmware.

For 30-40MHz the board cannot sustain continuous output at the maximum rate (limited by the DMA rate), but short bursts (up to the size of the FIFO buffer of 8k samples) of data output should be possible. Higher contiguous output rates are possible but the data rate must be reduced: we currently transmit 8bytes/sample which gives for the maximum DMA rate of 340MBytes/s (see paper) a maximum rate of 42.5MHz. Reducing to 4bytes/sample would allow 85MHz. This could be done when the timestamp (see data structure) is not transmitted, i.e. when each "tick" produces a sample. When one takes also into account that we need only to output 24bits/sample one could achieve 113.3MHz contiguous output rate. A dynamic switching between this and the normal mode might be possible. Even higher rates can be achieved by further reducing the output bits: for example with 8, 5 or 2 bits output rates of 340MHz, 544MHz and 1360MHz would be possible. The

firmware would need to be changed, but I have successfully tested generation of 1ns pulses - however at lower repetition rate. In case you have special requirements let me know.

Normally the board is clocked from the internal 50MHz oscillator, but an external clock input can be used for a more stable reference. This must be enabled in the FPGA control register (see register map). The buffer board provides the necessary clock buffers for input of a 10MHz sine-wave or LVPECL signal (see buffer board section below). It also allows to output the internal 10MHz of the board to another (secondary board) as LVPECL signal. The external input and output clock frequencies are at the moment hard-coded for 10MHz. Other frequencies can be chosen but require re-compiling of the firmware. Ask me if you need this.

Before using an external clock ensure it is stable and is within the limits of the input clock buffer (see buffer card external clock section). If the external clock stops during the experiment cycle the board will immediately abort execution which might be dangerous for the experiment! The board will go into error state (red LED is on and the bit STATUS_ERR_LOCK is set in the status register) and the board needs to be reset by software to restore it into the normal state. In labscript - BLACS the "external clock" can be enabled in the FPGA board "general" section and it can be enabled in the connection table for FPGA_board as worker_args option "ext_clock": True.

We have experienced that the clock input is quite sensitive to electrical discharges which can cause that the clock is lost only for few clock cycles with the PLL immediately re-locking afterwards. To prevent the board from immediately going into the error state, the FPGA control register has the CTRL_ERR_LOCK_EN bit which is normally enabled. When this bit is disabled, short clock loss of a few clock cycles will not cause an error. However, when the clock loss takes longer or is permanent the clock loss error should be still generated. Be cautious when disabling this bit! It can be that the board executes for prolonged times with the internal clock only or stops working for prolonged time until the PLL manages to re-lock. The synchronization between boards will certainly be lost. Synchronization error of order of several μs can occur. In labscript the "ignore clock loss" can be selected in the FPGA-board "general" section and in the connection table it can be enabled for FPGA_board as worker_args option "ignore_clock_loss": True. A warning message will be generated when a short clock loss is detected. But consider carefully before using this setting.

5. Strobe pulse:

First I give a short motivation why we need the strobe and how it was generated without the FPGA. The strobe (also called "pseudo-clock") has all features of a "real" clock, but it is not contiguously active but only when devices on the bus should be addressed. Same as for a real clock, its phase must be delayed with respect to that of the data and address lines to allow to damp oscillations after switching and to compensate for unequal delays between the bits. Additionally, the it must change state at twice the rate as the data on bus in order to "trigger" devices always at the rising (or falling) edge. In our "traditional" design we use the simplest way to generate such a strobe: one address bit (unfortunately, also called "strobe bit", abbreviated as STRB bit) is not directly connected to the bus but goes through hardware which generates a short pulse on the bus whenever this bit is changing. For each sample our software automatically changes the state of this address bit to generate the strobe pulse. All address decoders of our devices on the bus have this address bit set to "high" such that they accept only the data together with the strobe pulse. This way a fixed delay and duration of the pulse can be achieved without having the strobe address bit required to change at twice the data rate. Additionally, when the strobe bit might be used by the software to skip certain samples when they are not needed. As far as I know only the Yb experiment with several sub-racks used in parallel is using this feature.

With the FPGA the strobe pulse can be directly generated by the FPGA for each sample with a programmable delay and duration without the need of an additional address bit toggling. Therefore the STRB bit has become obsolete which simplifies the software generating the data. For experiments using it to skip samples it can be still enabled (see I/O configuration), however an easier way to skip samples is by enabling the NOP (no-operation) bit which disables the actual sample when set and does not require to change all following samples as the STRB bit needs if only a single sample should be changed.

The FPGA board provides two independent strobe signals to the buffer board (labeled strobe_0 and strobe_1; see buffer board section). This allows to drive two sub-racks with different timing and to address one or the other sub-rack.

Irrespective if the STRB bit is disabled, or not, the timing for the generated strobe pulse can be adjusted independently (using the strobe control register, see register map). The timing is defined by the setup time, which is the time between the change of the data lines and the rising edge of the pseudo-clock, and the hold time, which is the time how long the strobe pulse remains high. For the FPGA board the sample and hold times can be adjusted by software with 10ns resolution. Typical values for the bus output sample rate of 1us is 300ns sample time plus 400ns hold time. For faster bus output rates the timing needs to be adjusted accordingly.

For higher output sample rates of more than (at the moment) 20MHz the strobe cannot anymore be generated in this way and it must be configured to toggle its state after the setup-time instead of generating a pulse for each data transmission. This reduces the clock rate to the output sample rate but requires that the hardware on the bus triggers on the rising and falling edge of the strobe (double-data-rate/DDR mode) but which has to be supported by the devices on the bus. True, contiguous clock outputs at much higher rates are possible, see clocking section above.

6. External Triggers:

Similar to the old control system the board has start and stop triggers, an additional restart trigger can be defined in order to distinguish between the initial start of the board and the restart after the stop trigger. Several or all triggers can use the same external input and the levels/edges can be different or the same. In addition, the stop trigger can be activated by a programmable data bit (labeled STOP bit) of the sample. All triggers can be configured by software with the input control registers, see I/O configuration and register map below.

All triggers are executed by the board at integer multiple of the bus output rate, e.g. for 1MHz output rate the board triggers at integer multiple of 1µs after the experiment start time. The stop trigger outputs the actual sample and generates the strobe pulse before the board stops. When the restart and stop triggers are active at the same time and the board is running, then the board will stop, while when the board is already stopped, the board will restart. If both triggers continue to be active, the board will step through the active and stop states every integer multiple of the bus output rate.

a) start trigger: when enabled after the board is started the first time, the board waits for the start trigger condition (external input) before executing the first command. During waiting time the ready, and wait bits in the status register are set, but the run bit is reset. When the board is running the run bit is set. In cycling mode the board waits at the beginning of each new cycle for the starting condition.

b) stop trigger: when enabled the experimental sequence can be stopped by the stop trigger condition (external input or a data bit). When the board is in the stopped state the run and the wait bits are both set. To restart the board either a restart trigger must be enabled or the run bit in the control register must be reset and set again (using STOP and START commands). Note that during the stopped state the board can be re-configured by software. To abort and not restart the board reset the board using the RESET server command.

c) restart trigger: same as the start trigger (external input), but applies only after stop trigger is active. This allows to have different input and/or levels for start and restart.

7. cycling mode (in development):

This mode allows to repeat an experiment cycle a programmed number of times (cycles) without repeated uploading of the data and without delay between the cycles. The start of the next cycle is exactly $1/f_{bus}$ after the end of the previous cycle. This mode can be enabled in the FPGA control register by setting the DIO_CTRL_RESTART_EN bit and by programming ≠ 1 cycles with the SERVER_CMD_OUT_START command. The board will remain in the run state until all cycles are executed, for each new cycle the restart bit is toggled (removed from status register but accessible as output source) and after all cycles are executed the end bit is set. When the number of cycles is set to 0 in the SERVER_CMD_OUT_START then the board continues cycling infinite and the SERVER_CMD_OUT_STOP must be send in order to stop the board. Stop and restart triggers can be employed as normal to stop and restart the board also in the cycling mode.

At the moment the cycling mode is fully implemented (and simulated) on the FPGA but it is not tested with the DMA driver. The DMA driver uses already appropriate ring-buffers for this mode but the length of the buffers needs to be adjusted and for too few data samples it might not work since the driver needs at least 2 buffers. For too few samples the samples must be copied by the driver to fill 2 buffers. This should be quite "easy" to be done but needs testing.

Therefore, cycling mode is at the moment not available but when I have sufficient time I'll continue working on it since this might be very useful for experiments with short cycle times which require a lot of runs.

8. Sample data structure:

With the new FPGA board, time and data information needs to be sent to the board via Ethernet. In the standard configuration 32bits of time (also called "timestamp") and 32bits of data, i.e. 8 bytes, is sent to the FPGA board for each sample. The timestamp is an unsigned integer where one increment means a time step $\Delta t = 1/f_{bus}$ (also known as "tick") with f_{bus} the bus output rate (typically 1MHz, i.e. $\Delta t = 1\mu s$). When two sub-racks are used, as in the Yb experiment, 2x32bits of data need to be sent, i.e. 12bytes per sample. This option can be set in the firmware but is not fully implemented in the latest version. Please ask me when you need this or other options. I plan to allow configuration of the data structure by software in a future version.

9. Synchronization of several boards:

To synchronize several boards a common time base (clock) and start signal (start trigger) needs to be used. An external clock can be provided to all boards or one board, called the primary board, provides its internal clock to the other boards. The primary board always provides the start trigger

to the other boards. The buffer board already is supplied with clock input and outputs and the start and stop trigger signals with SMA connectors on the edge of the PCB (see buffer board section below). Connect the clock and trigger outputs from the primary board to the secondary board clock and trigger inputs. Several secondary boards can be clocked and triggered using power splitter along the clock and trigger coaxial cables.

The propagation delay of the trigger signal and the phase shifts of the external clock can be compensated with a software configurable waiting time and phase shift, see sync_delay and sync_phase registers. After the primary board has generated the start trigger, or after the secondary board has received the start trigger, it will wait for the a specified time (sync_delay in units of 10ns) before it starts executing the experiment. Additionally, the clock phase to which the PLL of each secondary board is locked can be fine-adjusted and allows a resolution of better than 1ns limited only by the jitter of the start trigger.

The waiting time and phase shift can be saved into the configuration file on the SD card or they can be set during the configuration of each experimental run. See sections configuration file and server commands below.

10. Special data bits:

Special data bits can be configured with the input control registers (see register map and I/O configuration). The NOP and the STRB bits allow to ignore data (see strobe generation), the STOP bit can be used to stop execution (see external triggers) at a specific instruction and the IRQ bit can generate an interrupt at the specific instruction. Further bits can be implemented on request.

a) NOP bit: when this bit is enabled and set the sample for this sub-rack is ignored. This allows to temporarily disable a sample without deleting it from the data. This is more efficient than to move data in memory. This bit was used by the former driver to mark alignment bytes, but is not anymore needed with the present driver version. The NOP bit only prevents that the data is put on the bus, but it does not affect the timing or other special data bits like the STRB bit still has to toggle state and the STOP bit stops the board.

b) STRB bit: when this bit is enabled and not toggling state with respect to the previous sample, then the sample of the corresponding sub-rack is ignored. The first sample is always used regardless of the state of the toggle bit, unless the NOP bit is set. This prevents that even or odd number of samples impact if the first sample is executed, which was a problem with the old DIO-64 system. This historical bit (see section strobe generation) can be used to skip samples but the NOP bit is more convenient and maybe in a future version support for this might be dropped.

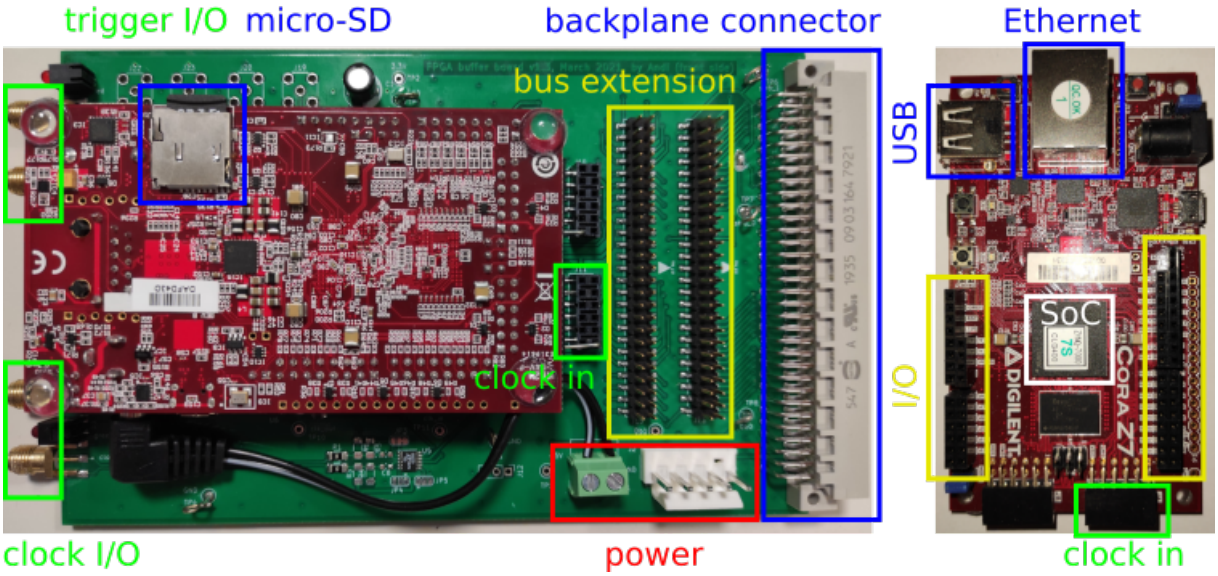
c) STOP bit: when this bit is enabled and set the execution is stopped after the specified time given in the timestamp of the sample. For more details, see external trigger section.

d) IRQ bit: when this bit is enabled and set then an interrupt is generated on the FPGA board and is transmitted to the control software which can detect it with the FPGA_irq bit set in the status register. This is useful if the control software has to execute some code at a specified time during the experimental sequence. For example, GPIB or USB or Etherent controlled devices can be programmed with such a (software) interrupt at the specified time. This is much more efficient and jitters less in time than polling by software for the actual time of the experimental cycle.

11. Buffer board

A custom PCB buffer board is needed to interface and buffer the commercial FPGA board with our standard hardware. This is needed since the FPGA board works with 3.3V (LVCMOS) level while your bus works with 5V (TTL) level. Additionally, the current driving capability of the pins of the FPGA board is not sufficient to directly drive our backplane bus which most of the time just a flat ribbon cable.

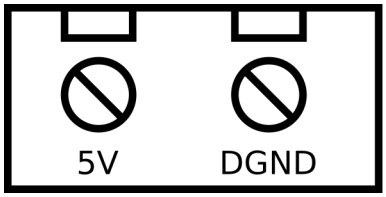
Figure 2 left shows the buffer board (green, version 1.3) with the FPGA board (red) mounted on top, right shows the FPGA board from the front side:

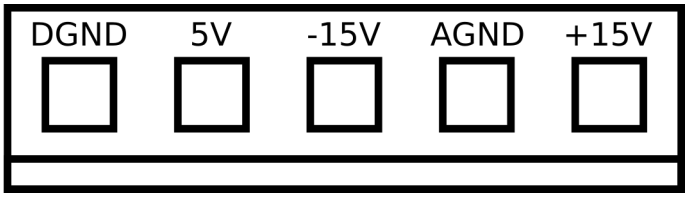


The FPGA board is equipped with Arduino type headers (yellow I/O on the right panel) which are used to stack it on top of the buffer board and to interface with it. The buffer board is inserted into one free slot of the sub-rack and the backplane connector (blue) is used to supply power and the digital data to the sub-rack.

a) Power:

Figure 3 shows the power connections of the buffer board.



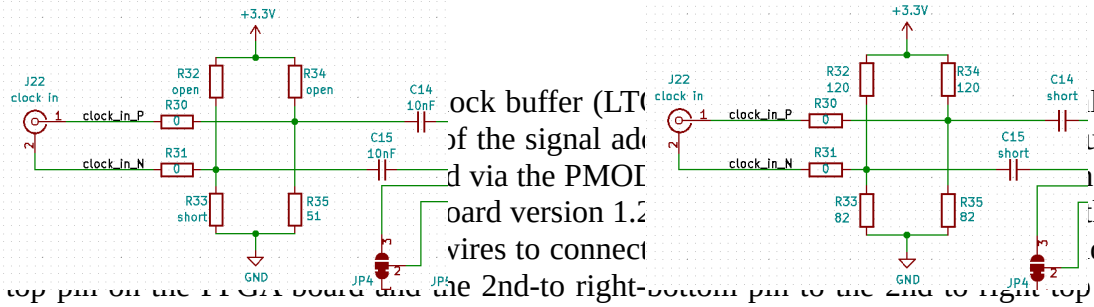


The FPGA board and the buffer board are powered via the two power connectors (red in Fig. 2) from the power supply of the sub-rack. This gives (on the 5-pole connector from left to right) digital ground, 5V, -15V, analog ground, +15V. The (2-pole connector from left to right) screw terminal provides the 5V and digital ground to the FPGA board via the 90° jack (4.0mm, outside ground, inside 5V) cable. Note that the backplane connector of the buffer board also supplies the power to the backplane bus. Ensure to set the jumper JP3 on the FPGA board to "EXT" and not to "USB". ATTENTION: Avoid hot-plugging of the buffer board and any of the devices of the sub-rack! I.e. always power off the sub-rack when inserting or removing any device in or out of the sub-rack. I have experienced already several times that the bus buffers of the digital or analog output cards were broken after a device (DDS or floating analog out) was inserted or removed.

b) External clock:

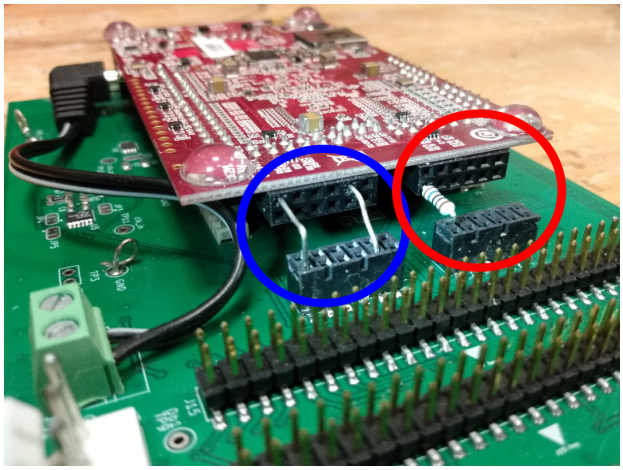
The buffer board has two SMA connections for the external clock (left-bottom green in Fig. 2) input (bottom) and output (top). The external clock input is needed to synchronize several boards in time (see synchronization section) or if a well-defined time scale is needed. If no external clock is provided, the 50MHz oscillator of the FPGA board is used (for more details see clocking section). The clock output is used if no external clock is available and one or several boards are used. In this case the primary board provides the trigger signal and also the clock signal to the other boards. The clock input accepts 10-300MHz with default frequency of 10MHz. At the moment the frequency cannot be configured by software, so in case this needs to be changed please ask for the firmware update. The input clock can be either a sine wave (0.5-2Vpp, 50Ω) centered at 0V offset voltage or 3.3V LVPECL standard (0.8Vpp with 2.0V offset, 50Ω). The two different signals need different input resistors/capacitors, see Figure 4 for the sine input (left) and LVPECL input (right) for the buffer board version 1.4. Other buffer board versions are equivalent, only different part labels are used. The clock output is always a 3.3V LVPECL signal.

Figure 4, external clock input configuration for buffer board version 1.4. Left for sine wave input, right for LVPECL input.



The buf ock buffer (LT l out of the sine f the signal ad utput of the buff d via the PMOI Fig. 2; J11 on l oard version 1.2 the blue circle ir vires to connect de with the left- top pin on the FPGA board and the 2nd-to right-bottom pin to the 2nd to right top pin on the FPGA board. For buffer board version 1.4 uses on the PCB the SMB connector J24 to connect a short coaxial cable to the left-top pin on the FPGA board and the next pin right of this for the inner and outer conductors of J24 respectively. When the FPGA board locks to the external clock the LED between the clock SMA connectors and a blue LED on the FPGA board are switched on.

Figure 5, clock input (blue circle) and strobe_1 output (red circle) at FPGA board.



c) Input/outputs:

The buffer board has two SMA connections for the external triggers (left-top green in Fig. 2) input (top) and output (bottom). The trigger signal is used to synchronize the starting time for several boards (see synchronization section). When a new experiment should be started, the primary board sends the trigger to the secondary board(s), waits a software-configurable time until the secondary board(s) have received the trigger and all boards start execution of the experiment synchronously. The required waiting time can be measured with an oscilloscope and can be set as the "wait" option of the server.config file. When the trigger option is not used these I/O pins can be used as additional general-purpose I/O ports.

Depending on the buffer board version it has additional general-purpose digital inputs and outputs. Buffer board version 1.3 and above have in total 3x inputs and 3x outputs. The first I/O pair is on the edge of the buffer board with SMA or SMB edge connectors and two more I/O pairs are located on the PCB on the side the FPGA board (SMA or SMB through-hole connectors, see figure 2). All digital inputs/outputs use LVC standard, i.e. they use 3.3V I/O level but can be used also with 5V input without damage. The outputs are pulled up to 5V with 2.2kΩ resistors. These I/O's are NOT designed for fast switching with 50Ω load.

All digital I/O's can be configured by software, see I/O configuration for details. The input can be used for different triggers, see trigger section. The outputs including the LEDs (see below) can be configured to represent certain internal states of the board (like running, waiting, etc.), which is very useful for debugging. Note that most versions of the buffer board use inverting buffers and that due to the pull-up the falling edge of the outputs is much faster than the rising edge. Therefore, all signals can be inverted to use the ideal polarity for most situations.

d) LEDs:

The buffer board has 3 LEDs on the edge which allows to fast check the status. During booting of the board all LEDs are switched on and they should switch off after maximum 20 seconds (on the latest version its faster) when the booting is finished and the board is ready for communication with the software. During the experimental run the green LED is switched on and switches off at the end of the cycle. If there is an error the red LED is switched on (this might be only briefly visible when the software resets the board immediately). The third LED, located on the bottom between the clock SMA connectors is either green or blue and indicates the presence of an external clock. The LEDs on the buffer board can be programmed by software (see I/O configuration).

All 3 LEDs have corresponding red, green and blue LEDs located on the FPGA board itself. These cannot be configured by software.

All 6 LEDs are dimmed using pulse-width modulation. The amount of dimming depends on the buffer board version. Let me know if they are too bright or too weak.

e) Controlling two sub-racks from a single FPGA board:

One FPGA board can control two sub-racks when they are not too far from each other (about 3m). For this option either a second buffer board version >=1.3 is needed or an old opto-coupler card used for the DIO-64 card can be used. Starting from version 1.3 the buffer board is equipped with additional buffers which allow to connect a ribbon cable (2x32 pin, 1.27" spacing) from the "daughter connector" J21 (in version 1.4, J3 in v1.3) to a second buffer board J21 (J3) which is

inserted into the second sub-rack and does not need to be equipped with a FPGA board. On this second buffer board the buffers which are driving the daughter connector J21 (J3) need to be disabled by setting the solder jumper JP3 (v1.4, JP2 on v1.3) from ground to 5V. This is done by cutting with a sharp knife the connection to ground and soldering a bridge from the center pad of JP3 (JP2) to 5V. This way the signal from the first FPGA board is fed into the second buffer board where another set of buffers ensure the signal is refreshed before it is put on the bus. Additionally, the FPGA board provides two independent strobe signals (see section f below and section strobe above) which can be used to control the two sub-racks independently with individual adjustable timings.

If the old opto-coupler board should be used, then the toggle strobe (see strobe section) can be used in order to simulate the old STRB bit, otherwise the strobe bit in the data must be used. Note that this board has a fixed pseudo-clock timing with a significant larger jitter than the new board and that it cannot work with more than 1MHz bus output rate. However, it is convenient for testing since its existing flat ribbon cable can be directly connected to the new buffer card without changing anything in the sub-rack. Be careful when removing the ribbon cable: do not exert force on the cable itself since it easily gets ripped out of the connector, but pull the connectors directly, maybe with the help of a small flat-blade inserted from the side.

When the two sub-racks are too far apart or when the electrical isolation is an issue or more than two sub-racks need to be controlled then it is recommended to use independent FPGA boards in each sub-rack, see synchronization section.

f) Strobe selection:

See strobe section about more details for the strobe.

The FPGA board provides two independent strobe signals to the buffer board (labeled strobe_0 and strobe_1). With solder jumper JP1 one can select the pseudo-clock signal put on the backplane. This allows to have independent timings of two sub-racks and to separate the address space. The default is strobe_0. When strobe_1 should be used then JP1 needs to be soldered accordingly and an additional wire from the FPGA board needs to be connected as indicated by the red circle in Figure 5. A decoupling resistor of 100-200Ω can be used to connect the left-top pin on the PCB side to the left-bottom pin on the FPGA side.

12. I/O configuration:

a) Input configuration:

Three external inputs (buffer board version >= 1.3) can be used to generate start, stop and restart trigger events with arbitrary level or edge triggering (see external trigger section). The stop trigger can be programmed as well from a data bit. Another three special data bits can be defined for special purposes (see special data bits section).

To enable and configure these 6 signals, the FPGA uses two registers: ctrl_in0 and ctrl_in1, see register map. Each possible signal "destination" is summarized in Table 3, and can be "connected" with one of the sources in Table 4 by setting the corresponding register bits (see right column of Table 3) to the source value (middle column in Table 4). Not enabled destinations are set to 0 (i.e. source "None"). Not all sources can be given for all destinations, see center column of Table 3. For example, to enable start and restart trigger on input 0 rising edge and stop trigger on input 1 low

level you have to set ctrl_in0[5:0] = 3, ctrl_in0[11:6] = 6 and ctrl_in0[17:12] = 3 which gives ctrl_in = 3 + 6«6 + 3«12 = 12675 = 0x3183.

If you use labscript you can use in BLACS the FPGA board - input configuration section to get the required register values by simply selecting the configuration you want. The register values are output as hexadecimal integer in the "value 0x" field. Note that the register value can also be input there directly and the selection is changing accordingly. Note that in BLACS the selected configuration is immediately written into the registers but it is not permanent. To make the configuration permanent you can give it in the connection table as a dictionary (see example connection table) or in the configuration file.

Table 3, input destinations

destination	allowed sources	register
start trigger	0-12	ctrl_in0[5:0]
stop trigger	0-12, 32-63	ctrl_in0[11:6]
restart trigger	0-12	ctrl_in0[17:12]
NOP bit	32-63	ctrl_in1[5:0]
IRQ bit	32-63	ctrl_in1[11:6]
STRB bit	32-63	ctrl_in1[17:12]

Table 4, input sources

source	value	allowed destinations
None	0	all
input 0	1	start/stop/restart trigger
input 0 inverted	2	start/stop/restart trigger
input 0 rising edge	3	start/stop/restart trigger
input 0 falling edge	4	start/stop/restart trigger
input 1	5	start/stop/restart trigger
input 1 inverted	6	start/stop/restart trigger
input 1 rising edge	7	start/stop/restart trigger
input 1 falling edge	8	start/stop/restart trigger
input 2	9	start/stop/restart trigger
input 2 inverted	10	start/stop/restart trigger
input 2 rising edge	11	start/stop/restart trigger
input 2 falling edge	12	start/stop/restart trigger
data bit 0-31	32-63	stop trigger, NOP/IRQ/STRB

b) Output configuration:

Three external outputs (buffer board version >= 1.3), two bus-enable signals and the three LEDs on the buffer board can be configured to output different signals of the FPGA board. See Tables 5 for outputs = "destinations" and Table 6 for the sources below. The "sync out" signal on the falling edge is used to trigger the secondary board(s) from the primary board. The other signals are mainly to indicate status information but are also very valuable for fast debugging with an oscilloscope.

Two registers (ctrl_out0 and ctrl_out1, see register map) need to be programmed for the different output signals. The configuration is similar to the input registers with 6 destination bits reserved for each output which need to be set to the desired source. Source "low" and "high" (value 0 and 1) means the output is at a fixed low or high value. Note that the different buffer board versions might have inverted outputs and they are usually just pulled high such that the falling slope is usually much faster than the rising slope. Therefore, the signals are provided with the inverted version which makes it easier to trigger always on the falling slope.

For example to configure the red LED with error, the green LED with run and the blue with external clock locked you have to set ctrl_out1[5:0] = 14, ctrl_out1[11:6] = 18, ctrl_out1[17:12] = 8, i.e. ctrl_out1 = 14 + 18«6 + 8«12 = 33934 = 0x848e.

As for the inputs you can directly get the register values with labscript - BLACS in the output configuration section.

Table 5, output destinations

destination	allowed sources	register
output 0	0-53	ctrl_out0[5:0]
output 1	0-53	ctrl_out0[11:6]
output 2	0-53	ctrl_out0[17:12]
bus enable 0	0,1 (low,high)	ctrl_out0[23:18]
bus enable 1	0,1 (low,high)	ctrl_out0[29:24]
LED red	0-53	ctrl_out1[5:0]
LED green	0-53	ctrl_out1[11:6]
LED blue	0-53	ctrl_out1[17:12]

Table 6, output sources

source	value	allowed destinations
fixed low	0	all
fixed high	1	all
sync out	2	all except bus enable
sync out inverted	3	all except bus enable
sync enable	4	all except bus enable
sync enable inverted	5	all except bus enable
sync monitor	6	all except bus enable
sync monitor inverted	7	all except bus enable
external clock locked	8	all except bus enable

external clock locked inverted	9	all except bus enable
external clock used	10	all except bus enable
external clock used inverted	11	all except bus enable
external clock lost	12	all except bus enable
external clock lost inverted	13	all except bus enable
error	14	all except bus enable
error inverted	15	all except bus enable
ready (first data in FIFO)	16	all except bus enable
ready inverted	17	all except bus enable
run (active also during wait)	18	all except bus enable
run inverted	19	all except bus enable
wait (for start/restart trigger)	20	all except bus enable
wait inverted	21	all except bus enable
end (of experiment)	22	all except bus enable
end inverted	23	all except bus enable
restart (toggles every cycle when num_cycles ≠ 1)	24	all except bus enable
restart inverted	25	all except bus enable
start trigger (toggle)	26	all except bus enable
start trigger inverted (toggle)	27	all except bus enable
stop trigger (toggle)	28	all except bus enable
stop trigger inverted	29	all except bus enable
restart trigger (toggle)	30	all except bus enable
restart trigger inverted	31	all except bus enable
strobe 0	32	all except bus enable
strobe 0 inverted	33	all except bus enable
strobe 0 contiguous	34	all except bus enable
strobe 0 contiguous inverted	35	all except bus enable
strobe 1	36	all except bus enable
strobe 1 inverted	37	all except bus enable
strobe 1 contiguous	38	all except bus enable
strobe 1 contiguous inverted	39	all except bus enable
IRQ TX	40	all except bus enable
IRQ TX inverted	41	all except bus enable
IRQ RX	42	all except bus enable
IRQ RX inverted	43	all except bus enable
IRQ FPGA	44	all except bus enable

IRQ FPGA inverted	45	all except bus enable
TX FIFO full	46	all except bus enable
TX FIFO full inverted	47	all except bus enable
TX FIFO empty	48	all except bus enable
TX FIFO empty inverted	49	all except bus enable
RX FIFO full	50	all except bus enable
RX FIFO full inverted	51	all except bus enable
RX FIFO empty	52	all except bus enable
RX FIFO empty inverted	53	all except bus enable

13. Experimental cycle:

The experimental cycle starts with resetting the board, i.e. the SERVER_RESET command is sent to the server. The reset does not reset most of the FPGA configuration registers, except of the FPGA control register which needs to be reprogrammed. The reset is required for the DMA section which otherwise gives an error when started after stopped. I will try to fix this problem in the next version such that no reset and reprogramming of the FPGA control register is needed for each experimental cycle.

If the board was never configured after a reboot then it needs to be configured by sending the SERVER_CMD_OUT_CONFIG server command. This programs the clock divider, strobe delay, input and output control registers, and the FPGA control register. When several boards are used, then also sync_delay and sync_phase must configured as well. Repetitions (cycles) and number of samples are ignored. These are part of the older configuration structure used for the DIO-64 board.

If the board was already configured before, then only the FPGA control register needs to be set with the SERVER_SET_REG server command. As already mentioned, in the next version this might not needed anymore, unless the configuration should be changed.

After configuration, the data samples can be uploaded to the board with the server command SERVER_CMD_OUT_WRITE where data = number of bytes to be uploaded. The server responds with SERVER_ACK and the data can be sent to the server. The maximum number of samples on the FPGA board is 10^7 which seems a lot but for high bus output data rates might be a limitation. With the current firmware the data needs to be uploaded for each experimental cycle but in the next version this will not be needed anymore. I am still not sure how to implement this, but probably attempting to upload new data will reset all data buffers from a previous cycle. If no data is uploaded then the old buffers are re-used for the current cycle. Reset will reset the buffers.

To start the output of the samples, the server command SERVER_CMD_OUT_START is sent where data = number of cycles = repetitions of the same experiment. Leave cycles = 1 for the moment since this option is implemented in the FPGA but the DMA driver most likely needs to be updated and tested with this option. When the board is configured to wait for the start trigger it will wait for this input before the output generation will start.

During execution of the experiment the server command SERVER_GET_STATUS_IRQ should be sent to the server in order to get status information at about 20Hz rate. In order that this works and does not return with SERVER_NACK (timeout) the DIO_CTRL_IRQ_FREQ_EN bit in the FPGA control register must be enabled which generates the interrupt required for this command. The

server will not immediately return the requested data but only after the IRQ occurred and when the status has changed (or after timeout of typically 100ms). This avoids that the software has to permanently poll the server for status update. Alternatively, SERVER_GET_STATUS can be used also during running state and is immediately returning the actual board status. However "polling" with this function is not recommended since it might return the same state for several consecutive calls. With the present firmware the SERVER_GET_STATUS_IRQ will timeout when the board is in the wait state, i.e. when it waits for the start or restart trigger since in this case the board timer is not running which is also used for the IRQ generation. This should be fixed in a future version.

Both server status function return the content of the board status register where the software can check the run and end bits (see register map). The board is running while the run bit is set. This remains set also while the board is waiting for the restart trigger (at the moment it is not set while waiting for the start trigger, but this might be changing in the next version). When the run bit is reset after the run state then the board is either in the end state (end bit is set) or is in the error state (end bit not set and any of the error bits is set). When the board is in cycling mode (see below) then the run bit remains set until all cycles are executed, while the restart bit is toggling state for each new cycle (and restart trigger). The wait bit is set while the board is waiting for the start or restart trigger.

By sending SERVER_CMD_OUT_STOP to the server during the board running will stop the execution of the experiment at the next integer multiple of $1/f_{bus}$ and the run bit will be reset and the wait bit will set in the status register. The board resumes execution where it stopped after sending the SERVER_CMD_OUT_START command. This allows to stop and restart the board by software. When instead SERVER_RESET is sent after SERVER_CMD_OUT_STOP then the actual execution is aborted and cannot be resumed with SERVER_CMD_OUT_START.

When the board is in end state the software should send the SERVER_CMD_OUT_STOP command before the next experimental cycle can be restarted with the SERVER_RESET.

14. Server commands:

A server application running on the FPGA board allows to the control software to interact with the board via Ethernet TCP/IP. The server is programmed in C++ (see fpga-server source code on my github page) and the possible server commands are defined in dio24_server.h file. Table 7 gives a list of available server commands.

The commands SERVER_CMD_... are representing functions used in the Windows dynamic-link-libary (see the DLL section) for the old DIO-64 board which use the same data structures as these functions. This allowed a simpler transition from the old control system to the new one. Not all functions in the DLL are represented by server commands and not all data entries might be used.

Each server command starts with a 16-bit SERVER_CMD code which can be generated with the MAKE_CMD macro (cmd«10 + size) and contains an arbitrary command integer (cmd) and the size in bytes of the command including the SERVER_CMD. Different versions of the firmware might have slightly different sizes of the data structure. Additionally, the Here we give the sizes of the last version. If no data should returned by the server it answers with SERVER_ACK if the command was executed successfully, otherwise with SERVER_NACK. Commands returning some data usually have two command codes, one for the request SERVER_GET_... and one for the responds SERVER_RSP_... with the corresponding data structures.

Not all commands can be sent at arbitrary times, especially, when the board is running only SERVER_GET_STATUS_IRQ, SERVER_GET_STATUS and SERVER_CMD_OUT_STOP can be sent. If a command is sent to the server at the wrong time, it answers with SERVER_NACK and if an unknown command is sent it just closes the connection.

Table 7, server commands for actual version. The structures are summarized in tables 8-13.

command	code	structure	description
SERVER_ACK	0x0402	SERVER_CMD	responds: ok
SERVER_NACK	0x0802	SERVER_CMD	responds: error
SERVER_RESET	0x0c02	SERVER_CMD	reset board
SERVER_GET_STATUS_FULL	0x1c02	SERVER_CMD	get full status info
SERVER_RSP_STATUS_FULL	0x1cb6 0x1cba	client_status_full	responds to get full status info
SERVER_GET_STATUS	0x2002	SERVER_CMD	get client status immediately
SERVER_RSP_STATUS	0x2012	client_status	responds get client status immediately
SERVER_GET_STATUS_IRQ	0x2402	SERVER_CMD	wait for client status
SERVER_RSP_STATUS_IRQ	0x2412	client_status	responds to wait for client status
SERVER_GET_REG	0x280a	client_sr32	get register value reg = register address data = returned value
SERVER_SET_REG	0x2c0a	client_sr32	set register value reg = register address data = new value
SERVER_SET_EXT_CLOCK	0x3006	client_data32	set external clock data = sync_phase register
SERVER_CMD_OPEN	0x8002	SERVER_CMD	open board XP driver version
SERVER_CMD_OPEN_RESOURCE	0x8402	SERVER_CMD	open board VISA driver version
SERVER_CMD_CLOSE	0x9002	SERVER_CMD	close board
SERVER_CMD_OUT_CONFIG	0x9432	client_config	configure board
SERVER_CMD_OUT_WRITE	0x9c06	client_data32	upload data data = bytes to upload
SERVER_CMD_OUT_START	0xa006	client_data32	start board
SERVER_CMD_OUT_STOP	0xa402	SERVER_CMD	stop board

The data structures are defined in dio24_server.h where some use FPGA internal data structures which are defined in dio24_driver.h (uint32_t = 32bit unsigned integer).

Table 8, SERVER_CMD structure

name	type	description
cmd	6bits	arbitrary integer value
size	10bits	size of entire structure

Table 9, client_data32 structure

name	type	description
cmd	SERVER_CMD	server command code
data	uint32_t	arbitrary data

Table 10, client_sr32 structure

name	type	description
cmd	SERVER_CMD	server command code
reg	uint32_t	register address
data	uint32_t	register value

Table 11, client_config structure

name	type	description
cmd	SERVER_CMD	SERVER_CMD_OUT_CONFIG
clock_Hz	uint32_t	f_{sys} = 100,000,000 Hz
scan_Hz	uint32_t	f_{bus} in Hz
config	uint32_t	FPGA control register
ctrl_in[2]	uint32_t x 2	input control registers
ctrl_out[2]	uint32_t x 2	output control registers
cycles	uint32_t	0, not used
samples	uint32_t	0, not used
strb_delay	uint32_t	strobe delay register. 0 = use value in server.config file.
sync_wait	uint32_t	synchronization delay register. 0 = use value in server.config file.
sync_phase	uint32_t	synchronization phase register. 0 = use value in server.config file.

Table 12, client_status structure

name	type	description
cmd	SERVER_CMD	server command code
status	uint32_t	FPGA status register
board_time	uint32_t	board time register
board_samples	uint32_t	board time register
board_samples	uint32_t	board samples register

Table 13, client_status_full

name	type	description
cmd	SERVER_CMD	SERVER_RSP_STATUS_FULL
ctrl_FPGA	uint32_t	FPGA control register
ctrl_in0	uint32_t	input control register 0
ctrl_in1	uint32_t	input control register 1
ctrl_out0	uint32_t	output control register 0
ctrl_out1	uint32_t	output control register 1
set_samples	uint32_t	number of samples register
set_cycles	uint32_t	number of cycles register
clk_div	uint32_t	clock divider register
strb_delay	uint32_t	strobe delay register
sync_delay	uint32_t	synchronization delay register
sync_phase	uint32_t	synchronization phase register
force_out	uint32_t	force output register
status_FPGA	uint32_t	FPGA status register
board_time	uint32_t	board time register
board_samples	uint32_t	board samples register
board_time_ext	uint32_t	extra board time register
board_samples_ext	uint32_t	extra board samples register
board_cycles	uint32_t	board cycles register
sync_time	uint32_t	synchronization time register
version	uint32_t	firmware version
info	uint32_t	firmware information
FPGA_temp	uint32_t	FPGA temperature
phase_ext	uint32_t	accumulated external clock phase
phase_det	uint32_t	accumulated detection clock phase
ctrl_dma	uint32_t	DMA control register
status_TX	uint32_t	DMA TX status register
status_RX	uint32_t	DMA RX status register

dsc_TX_p	uint32_t	DMA prepared TX descriptors
dsc_TX_a	uint32_t	DMA active TX descriptors
dsc_TX_c	uint32_t	DMA completed TX descriptors
dsc_RX_p	uint32_t	DMA prepared RX descriptors
dsc_RX_a	uint32_t	DMA active RX descriptors
dsc_RX_c	uint32_t	DMA completed RX descriptors
err_TX	uint32_t	DMA TX last error code
err_RX	uint32_t	DMA RX last error code
err_FPGA	uint32_t	FPGA last error code
irq_TX	uint32_t	number of TX interrupts
irq_RX	uin32_t	number of RX interrupts
irq_FPGA	uint32_t	number of FPGA interrupts
TX_bt_tot	uint32_t	DMA TX transmitted bytes
RX_bt_tot	uint32_t	DMA RX transmitted bytes
bt_tot	uint32_t	total transmitted bytes
RD_bt_max	uint32_t	read bytes maximum
RD_bt_act	uint32_t	read bytes
RD_bt_drop	uint32_t	read bytes dropped
reps_act	uint32_t	DMA repetitions done
timeout	uint32_t	DMA timeout in ms
last_sample[0]	uint32_t	last sample timestamp
last_sample[1]	uint32_t	last sample data 0
last_sample[2]	uint32_t	last sample data 1 only transmitted for 12bytes/sample

15. register map:

Here all registers used by the FPGA are listed. They are defined in dio24_driver.h. All registers are 32bit wide and the addresses must be multiple of 32bit. The register address might be different in different version of the firmware. We give here the address used in the last version.

All registers can be directly read and written via the server commands SERVER_GET_REG and SERVER_SET_REG respectively. Note that no with exception of the FPGA control register the validity of the written value is NOT checked and the FPGA might behave unexpected for invalid values. Therefore, it is safer to use the specialized server commands for a specific purpose, like SERVER_CMD_OUT_CONFIG, which perform more thorough checks before writing to the register.

Table 14, FPGA register map

name	address	description
DIO_REG_CTRL	0x00	***FPGA control register*** bit[0] = DIO_CTRL_RESET: software reset (not user settable, use server command SERVER_RESET) bit[1] = DIO_CTRL_READY: driver ready (not user settable) bit[2] = DIO_CTRL_RUN: start/stop board (not user settable, use server command SERVER_CMD_OUT_START) bit[4] = DIO_CTRL_RESTART_EN: restart board in cycling mode (effective only when DIO_REG_NUM_CYCLES ≠ 1) bit[5] = DIO_CTRL_AUTO_SYNC_EN: enable sync_out pulse bit[6] = DIO_CTRL_AUTO_SYNC_PRIM: if set primary board, otherwise secondary bits [9:8] = DIO_CTRL_BPS96: 0 = 8bytes/sample, 1 = 12bytes/sample, data = bytes 3-7 in sample 2 = not defined 3 = 12bytes/sample, data = bytes 8-11 in sample bit[10] = DIO_CTRL_EXT_CLK: use external clock bit[15] = DIO_CTRL_ERR_LOCK_EN: if set external clock loss causes error (default), if not set external clock loss for short time does not cause error. bit[20] = DIO_CTRL_IRQ_EN: enable FPGA IRQs. disable to reset active IRQ's. bit[21] = DIO_CTRL_IRQ_END_EN: enable IRQ at end. bit[22] = DIO_CTRL_IRQ_RESTART_EN: enable IRQ when restarting board. bit[23] = DIO_CTRL_IRQ_FREQ_EN: enable repeated IRQ's at ~20Hz bit[24] = DIO_CTRL_IRQ_DATA_EN: enable

		IRQ with IRQ bit in data
DIO_REG_CTRL_IN0	0x10	***input control register 0*** see I/O configuration + input configuration
DIO_REG_CTRL_IN1	0x14	***input control register 1*** see I/O configuration + input configuration
DIO_REG_CTRL_OUT0	0x20	***output control register 0*** see I/O configuration + output configuration
DIO_REG_CTRL_OUT1	0x24	***output control register 1*** see I/O configuration + output configuration
DIO_REG_CLK_DIV	0x30	***clock divider*** bits[7:0] = 5-255 fully supported. with 4 output works but strobe not (maybe toggle trigger but not tested)
DIO_REG_STRB_DELAY	0x34	***strobe delay*** bits[7:0] = strobe_0 setup time in 10ns units bits[15:8] = strobe_0 setup+hold time in 10ns units, if 0 toggle trigger bits[23:16] = strobe_1 setup time in 10ns units bits[31:24] = strobe_1 setup+hold time in 10ns units, if 0 toggle trigger
DIO_REG_NUM_SAMPLES	0x40	***number of samples*** not supposed to be set by user. set by SERVER_CMD_OUT_START
DIO_REG_NUM_CYCLES	0x44	***number of cycles*** not supposed to be set by user. set by SERVER_CMD_OUT_START
DIO_REG_SYNC_DELAY	0x60	***synchronization delay/wait time*** bits[9:0] = 0..1023 = wait time in 10ns units
DIO_REG_SYNC_PHASE	0x64	***synchronization phase*** bits[11:0] = detection phase in steps bits[23:12] = external clock phase in steps. 560 = 360° = 10ns. the phase shift is relative to the actual phase phase_ext/det returned by SERVER_GET_STATUS_FULL. negative values are not permitted in current version. use SERVER_SET_EXT_CLOCK to set to absolute phase.
DIO_REG_FORCE_OUT	0x78	***force output register*** bits[15:0] = 16 data bits on bus bits[22:16] ([23:16]*) = address bits on bus bit[23] ([24]*) = strobe_0 bits[30:24] ([31:25]*) = JB on FPGA board bit[31] = strobe_1 (strobe_1 not existent*) *if firmware uses 8 address bits instead of 7. in this case strobe_1 does not exist. set this register only when board is not running! when this register is written the bits are

		immediately put on the bus. SERVER_RESET or SERVER_CMD_OUT_START resets the output on the bus. the register is not reset but has not effect.
DIO_REG_STATUS	0x80	FPGA status register bit[0] = DIO_STATUS_RESET: software reset active bit[1] = DIO_STATUS_READY: first data arrived out of TX FIFO bit[2] = DIO_STATUS_RUN: running state bit[3] = DIO_STATUS_END: end state bit[4] = DIO_STATUS_WAIT: wait state bit[5] = DIO_STATUS_AUTO_SYNC: auto-sync active (might not be applicable) bit[6] = DIO_STATUS_AS_TIMEOUT: auto-sync timeout (always set) bit[7] = DIO_STATUS_PS_ACTIVE: phase shift active (after DIO_REG_SYNC_PHASE was set) bit[8] = DIO_STATUS_TX_FULL: TX FIFO full bit[9] = DIO_STATUS_RX_FULL: RX FIFO full (should never happen) bit[10] = DIO_STATUS_EXT_USED: external clock is used bit[11] = DIO_STATUS_EXT_LOCKED: external clock is present and PLL is locked bit[12] = DIO_STATUS_ERR_TX: error TX data underflow (bus output rate too high!) bit[13] = DIO_STATUS_ERR_RX: error RX not ready (should never happen) bit[14] = DIO_STATUS_ERR_TIME: error time is not increasing (invalid data from user or buffer not properly reset after error) bit[15] = DIO_STATUS_ERR_LOCK: error external clock lost bit[16] = DIO_STATUS_ERR_TKEEP: internal error in TX data transmission. should never happen. bit[20] = DIO_STATUS_IRQ_FPGA_ERR: IRQ because of an error occurred. check error bits. bit[21] = DIO_STATUS_IRQ_FPGA_END: IRQ end occurred bit[22] = DIO_STATUS_IRQ_FPGA_RESTART: IRQ restart occurred bit[23] = DIO_STATUS_IRQ_FPGA_FREQ: repeated IRQ at ~20Hz occurred bit[24] = DIO_STATUS_IRQ_FPGA_DATA: IRQ from IRQ data bit occurred

		bit[30] = DIO_STATUS_BTN_0: button 0 on FPGA board pressed bit[31] = DIO_STATUS_BTN_1: button 1 on FPGA board pressed
DIO_REG_BOARD_TIME	0x90	***actual board timer value*** while running updated at ~20Hz when DIO_CTRL_IRQ_FREQ_EN enabled, otherwise updated only at end. at end gives last timestamp +1.
DIO_REG_BOARD_TIME_EXT	0x94	for future use
DIO_REG_SYNC_TIME	0x98	***auto-sync measured round-trip time*** this is not used in the current version.
DIO_REG_BOARD_SAMPLES	0xa0	***actual board number of samples*** while running updated at ~20Hz when DIO_CTRL_IRQ_FREQ_EN enabled, otherwise updated only at end.
DIO_REG_BOARD_SAMPLES_EXT	0xa4	for future use
DIO_REG_BOARD_CYCLES	0xb0	***actual board cycles*** -1 less than expected
DIO_REG_BUS_INFO	0xc0	***bus information*** bits[7:0] = bus data bits: 16 bits[15:8] = bus address bits: 7 or 8 bits[23:16] = number strobe: 1 or 2 bits[31:24] = number bus_en bits: 0-2
DIO_REG_VERSION	0xf0	***firmware version*** bits[4:0] = day 1-31 bits[8:5] = month 1-12 bits[15:9] = year - 2000 bits[23:16] = buffer board minor version bits[31:24] = buffer board mayor version
DIO_REG_INFO	0xf4	***firmware information*** bits[15:0] = 0: Florence, 1: Innsbruck bits[7:0] = 0xc0: Cora-Z7-07S, 0xc1 = Cora-Z7-10, 0xa1: Arty-Z7-10, 0xa2 = Arty-Z7-20

16. configuration file:

In addition to the three firmware files (image.ub, BOOT.BIN, boot.scr) on the micro-SD card there is the server.config text file which allows to configure the IP address and several other parameters for each board individually. All parameters are optional and all, except IP and port, can be configured also by software (see sections server commands and register map). The software programmed values take precedence. The file is a simple text file which can be edited with any text editor. The sharp symbol '#' is used to comment lines. After each parameter name follows the equal sign '=' and the value. Do not insert quotation marks or spaces for the values.

Table 2, configuration

parameter	default value	description
info	-	arbitrary information string printed on the console during startup
IP	DHCP	static IPv4 address given as a.b.c.d with a,b,c,d decimal integers 0-127 or "DHCP" to obtain a dynamic IP address from a DHCP server.
port	49701	port number where server listens. integer 0-65535.
clk_div	100	clock divider to generate the bus output frequency f_{bus} = f_{sys}/clk_div from the system clock frequency f_{sys} = 100MHz.
strobe_0 strobe_1	3:4:3:1	strobe 0 or 1 timing parameters given as s:h:r:level. s = setup time, h = hold time and s+h+r = 1/f_{bus} with f_{bus} = bus output frequency. Resolution is 10ns. level = 1 or 2. if level = 1 pseudo-clock is high during hold time and low during setup time and remaining time (h+r). if level = 2 pseudo-clock is toggling state at time s. Ensure to set also clk_div if $f_{bus} \neq$ 1MHz.
wait	0	0-1023. waiting time in units of 10ns before the board generates data on the bus after the start trigger is generated or received.
phase	0	external clock and detection phase. bits 0-11: detection clock phase 0-560 steps = 0-360°. bits 12-23: external clock phase 0-560 steps = 0-360°. The external clock phase allows to fine-tune the synchronization between boards within ~1ns. The detection phase determines the window phase when the external trigger input is sampled, and only needs adjustment when the secondary board phase jumps randomly by +/-180°.
primary	1	0 or 1. if 1 the board is the primary board, otherwise it is the secondary board. The primary board generates the start trigger, the secondary board waits for it.
CPUs	1	1 or 2. Number of CPUs. Give 1 for the Cora-Z7-07S and 2 for the Cora-Z7-10 board. This defines how the server optimizes reading data from the Ethernet and writing the data to DMA memory. The difference is only appreciable for large amount of data.

ctrl_in	[0x0,0x0]	input configuration given as list of 2 integers. if integer starts with "0x" hexadecimal is assumed, otherwise decimal. For details see ctrl_in registers in register map. In labscript - BLACS you can copy the two hexadecimal "value 0x" integers (prepended with "0x") from "FPGA-board" - "input configuration".
ctrl_out	[0x14482,0x848e] (out 0 = sync_out, out 1 = run, out 2 = wait, bus_en 0/1 = low, LED r/g/b = error / run / ext. clock locked)	output configuration given as list of 2 integers. if integer starts with "0x" hexadecimal is assumed, otherwise decimal. For details see ctrl_out registers in register map. In labscript - BLACS you can copy the two hexadecimal "value 0x" integers (prepended with "0x") from "FPGA-board" - "output configuration".

17. Windows DLL:

The old DIO-64 card from Viewpoint Systems which the FPGA board originally replaced used a Windows (XP and a Win-7 VISA beta version) driver which was provided with a Windows dynamic-link-library (DLL) which all of our experiments used to communicate from National Instruments Labview or Labwindows/CVI with the driver. Although the individual software implementations were different, the DLL remains the same, which gave use the opportunity to simply create a new DLL with the same functions as the old one, such that we only had to replace the DLL to switch from the old to the new control system. This proofed to be very efficient and we could switch the control system within short time. Small adaptations needed still to be done but in the worst case one could always switch back to the old system (although never needed).

All server commands starting with SERVER_CMD_... are representing such functions from the old DLL and take the same data structures. This made it easy to forward the data from the new DLL to the server and wait for the responds. Although all functions from the old DLL are implemented in the new DLL (and some more), not all functions need to be forwarded to the server. Therefore, only a few of them are implemented on the server.

18. Glossary:

boot:
after powering up the board it has to go through a sequence of startup = "boot" steps before it is ready. We use the SD-card for configuration and "booting": the first-stage bootloader (part of BOOT.bin) is loading the .bit file (also contained within BOOT.bin) with the FPGA configuration and is then loading the second-stage bootloader (u-boot, also part of BOOT.bin). The second-stage bootloader is loading the file-system (from image.ub) and starting the linux kernel. During starting of the kernel the drivers are loaded among which is our custom DMA-driver (dma24) which also takes care of the FPGA part (dio24) and the analog voltage input module (XADC) which is used to measure the board temperature. The driver sets the DIO_CTRL_READY bit in the control register which resets the LEDs to indicate its ready (this might be different depending on the firmware version). When all drivers are loaded our custom initialization script (fpga-init) is launched which is reading the server.config file from the SD card and starts the server (fpga-server) with the options specified in the server.config file. The server waits until the ethernet is ready (which might take some time) and then waits for the user application to connect on the configured TCP/IP address and port.

buffer board:
custom printed circuit board which interfaces the FPGA board with the sub-rack. it contains buffers for the I/O signals and to generate and input external clock and trigger signals. At present there are 3 versions available: 1.2, 1.3, 1.4.

clock / pseudo-clock:
In digital electronic systems a clock is a "golden" (= very good) digital signal with a well-defined frequency and on-to-off ratio, with steep slopes and with low phase noise. At the rising or falling edge of the clock (sometimes also at both edges, see DDR vs. SDR) the levels of all other digital signals are sampled and discriminated between "high" and "low" and updated according to the "program". As long as these other signals have settled to a steady state at the clock edge (see setup and hold time) their signal quality does not matter. Since the clock has to change two times the state in order to sample one time the other signals, the clock must run at twice the frequency of the other signals. To output data on a bus with output rate f_{bus}, the clock on this bus has to run at 2x f_{bus}. Put all toghether, the quality of the clock determines the performance of the entire system. Although clocks might be disabled, a clock is generally considered as always running. Contrary to clock a pseudo-clock is not running all the time but only when something should change. Nevertheless, the pseudo-clock still resembles all other characteristics of a clock. The pseudo-clock signal on our bus is also called "strobe". In our particular implementation it is not even a pseudo-clock in the strict definition of above, since it has no dedicated clock-routing which would ensure the high signal quality, but it is a "normal" digital signal which is just delayed by a programmable time with respect to the other signals on the bus. This is because the strobe frequency should be programmable in a wide range of ideally 2-200MHz, which might be attainable with some effort (resource utilization is an issue) by programming the PLL directly, but in addition to reach frequencies of 2-10MHz a clock buffer with fixed clock divider of a factor of up to 5 is needed, which then would require 1GHz to reach the upper frequency limit, which is 2-4x higher than what is possible with our low-cost FPGA. However, our requirement on the strobe is not very high and using the FPGA already improves the quality significantly with respect to before (where we have seen jitter of order of 50ns). A fixed-frequency pseudo-clock even at 2MHz should be possible using dedicated clock routing and using the enable/disable feature of the clock buffers.

console:
The FPGA board has a micro-USB plug on which debug information of the board can be monitored in realtime from an external computer. You need some serial communication program like minicom

(Linux) or PuTTY (Windows). Additionally you can execute many commands of the Linux operating system and I have created several testing applications for the FPGA and the control system. This feature can be very useful for debugging the system. Access is not password protected. ATTENTION: do not execute any of the testing applications when the control system is attached to actual hardware! Some tests create large and random data at high data rates on the bus! Before you want to do this please contact me.

CPU:
central processing unit. The FPGA-SoC has a CPU part and a logic part (see FPGA below). The CPU executes software in a linear (command by command) way and might be interrupted by interrupts (IRQ's). A classical CPU is not useful for time-critical operations (there are special microcontrollers for such applications). The CPU makes it easy to control the system via software while the FPGA part takes care of fast and precise timing.

cycling mode:
the experimental sequence can be repeated for a given number of repetitions (cycles) or infinitely (until user sends the stop command).

DDR vs. SDR:
double data rate vs. single data rate. For single data rate the data is read on the bus for each rising edge of the pseudo-clock. This requires that the pseudo-clock frequency is twice the sample rate and the pseudo-clock has a phase shift (the setup time) between when the data is changed and the rising edge. For double data rate the data is read on the bus for each rising and falling edge of the pseudo-clock. This allows to have the same frequency between the pseudo-clock and the data change but the phase shift still has to be maintained. The hardware must be capable of DDR.

DSP:
digital signal processing/processor, the FPGA part has several of such processors which allow complex manipulation of data and floating point arithmetic. In the current firmware this is not used but can be implemented when needed.

DMA:
direct memory access. This mode allows the hardware to read and write data in memory without involving the CPU. This allows much faster data transmission than using registers, but needs dedicated hardware channels, mapping and locking of dedicated memory sections and according support from the operating system and special care in the device driver.

experimental sequence, experimental cycle, experiment:
a sequence of samples describing what should happen for each time

firmware:
The collection of information defining the hardware and software configuration of the FPGA board. This is saved on the SD (secure data) card of the FPGA board and is loaded during booting of the board. It contains 3-4 files: BOOT.BIN which contains the bootloader and the bit stream which defines the hardware logic in the FPGA part of the FPGA-SoC chip. image.ub is the Linux image which contains the file system for the Linux operating system. For the newer version of the firmware a third file, boot.scr is needed which contains the boot script executed byt the bootloader. The SD card contains another file, server.config, which is a text file containing the IP address and other configurations of the FPGA board.

FPGA, FPGA board, FPGA-SoC board, FPGA-SoC chip:

commercial, low-cost development board with a field-programmable-gate-array (FPGA) and a CPU and dedicated hardware interfaces merged on a single chip into an embedded system-on-a-chip (SoC). The FPGA part is an ensemble of a huge number of hardware logic gates, flip-flops, memory, PLL's, DSP's etc. which allows to perform parallel complex logic operations with high frequency and well-defined timing. On the CPU part a simple embedded Linux operating system is executed which allows to execute arbitrary user software in a user-friendly environment and to easily interface with the hardware. The FPGA part appears as external hardware with which the user software like the TCP/IP server can interact via a custom driver using registers or via file access to send and receive data.

IRQ:
interrupt. Typically a hardware signal which tells the CPU to immediately stop what it did before and to execute a dedicated interrupt service routine for this specific interrupt. This allows to break the sequential execution of a CPU to mimic parallel execution, like performing a long-lasting calculation while still responding to user interrupts like mouse clicks. They are essential for modern computer systems but can be problematic for real-time applications where precise timing is needed.

NOP bit:
Special data bit in the sample. If this bit is enabled and set in the sample then the sample is not output on the bus, however the time of the sample and the other special data bits are still taken into account. Often this is the 31bit (highest data bit) and the user can select whatever bit from the 32-bit data word. In the previous firmware version this was needed, since the driver in the FPGA board relies on this bit to mark padded samples which should be disregarded.

PCB:
printed circuit board. The buffer board is a custom PCB with standard 160mm x 100mm x 1.6mm Eurocard size which can be conveniently inserted into the standard 19" sub-rack. See Ref. 2 for the production files.

sample:
One line in the code describing the experiment. The FPGA board supports 8 bytes/sample and 12 bytes/sample. The first 4 bytes is the timestamp, the second 4 bytes contains the data of the first sub-rack and the optional second 4 bytes contains the data of the second sub-rack. The 4 data bytes (32 data bits) of each sub-rack contains 16 data bits and 7 address bits which are directly put on the backplane bus of the sub-rack. The address bits define which device in the sub-rack should use the 16 data bits to change its state accordingly. An additional strobe bit is used to generate the pseudo-clock and the remaining 8 data bits can be used for other purposes like a NOP bit (no-operation) and a second strobe bit.

sample rate:
update rate of the backplane bus in the sub-rack in Hz. Typical value is 1MHz, which gives 1us temporal resolution. The maximum sample rate of the FPGA board is limited by the maximum DMA (direct memory access) data transmission rate of about 340Mbytes/s (see Ref. [1]) which gives a maximum sample rate of 40MHz for 8 bytes/sample. For short time the sample rate can be increased until the FIFO buffer (first-in-first-out; typically 16k samples) on the board becomes empty. Rates above 50MHz require a change of the firmware and maybe also of the design of the sub-rack backplane bus. Rates of 200MHz on 8 digital channels are possible using reduced number of data bits and without transmission of timestamp in contiguous sample mode with one sample per tick. Generation of 1ns short pulses at a fixed repetition rate have been demonstrated.

setup and hold time:
The setup time is the time between a digital signal going high or low and settle at the new value, and the clock (rising or falling) edge. The hold time is the time how long the digital signal has to remain at its settled level after the clock edge before it can change state again. Both times must be >0 and within the specified limits of a given device in order that the digital signal level "high" or "low" can be reliably determined by this device. This is important especially for high speed buses like the DDR bus used to read/write the external memory of the FPGA-SoC chip.

PLL:
phase-locked loop, allows to generate from one clock another clock with a different frequency but fixed phase relation to the original clock. The FPGA board can use an external 10-300MHz input clock to generate its internal 100MHz clock used to output the data on the bus. The external clock gives a well-defined time base and allows to synchronize in time several board.

register:
In the FPGA several 4 byte wide memories (flip-flops) are reserved to receive and send data from and to the the CPU using a dedicated bus (AXI-Lite bus). In the present case 8-bits are used for addressing, which allows up to 64 registers (2^8/4 = 64). These are divided into 32 control registers: data is sent from CPU to FPGA, and into 32 status registers: data is sent from FPGA to CPU. The Linux device driver maps these registers to fixed addresses in the memory which allows a driver or application to write or read at this address which is then automatically transmitted to and from the registers in the FPGA. This allows the driver and application to configure the FPGA and to monitor its status, however this is not intended for transmission of large amounts of data. For this purpose DMA is used.

STRB bit, strobe bit:
One data bit of the sample which can be enabled and selected for each sub-rack. If this is enabled the pseudo-clock signal is generated when this bit is toggling, otherwise the sample is ignored. If this bit is disabled for each sample the pseudo-clock signal is generated (unless the NOP bit is set). Historically, the pseudo-clock signal on the bus is sometimes called "strobe".

tick/timestamp:
the first 4 bytes of each sample contain the timestamp which is an unsigned integer counting time in units of tick = 1/sample rate where the sample rate is typically 1MHz, i.e. tick = 1us.

trigger:
external TTL signal which tells the FPGA board to start, stop or pause

TTL signal:
transistor-transistor-logic, rather old 5V digital signal level standard used in most of our experiment.

19. References:

[1] A. Trenkwalder et.al., "A flexible system-on-a-chip control hardware for atomic, molecular, and optical physics experiments", Rev. Sci. Instrum. 92, 105103 (2021), https://doi.org/10.1063/5.0058986

[2] https://github.com/INO-quantum/FPGA-SoC-experiment-control. Please contact me to get latest updates.