




# BENCHLAB HW INTERNALS



Roman Khvatov  
SITO

# Introduction

BenchLab is a reference project built around the SitoFA hardware and software infrastructure. It consists of a logic analyser, a binary pattern generator, an oscilloscope, and arbitrary user control panels.

BenchLab hardware features:

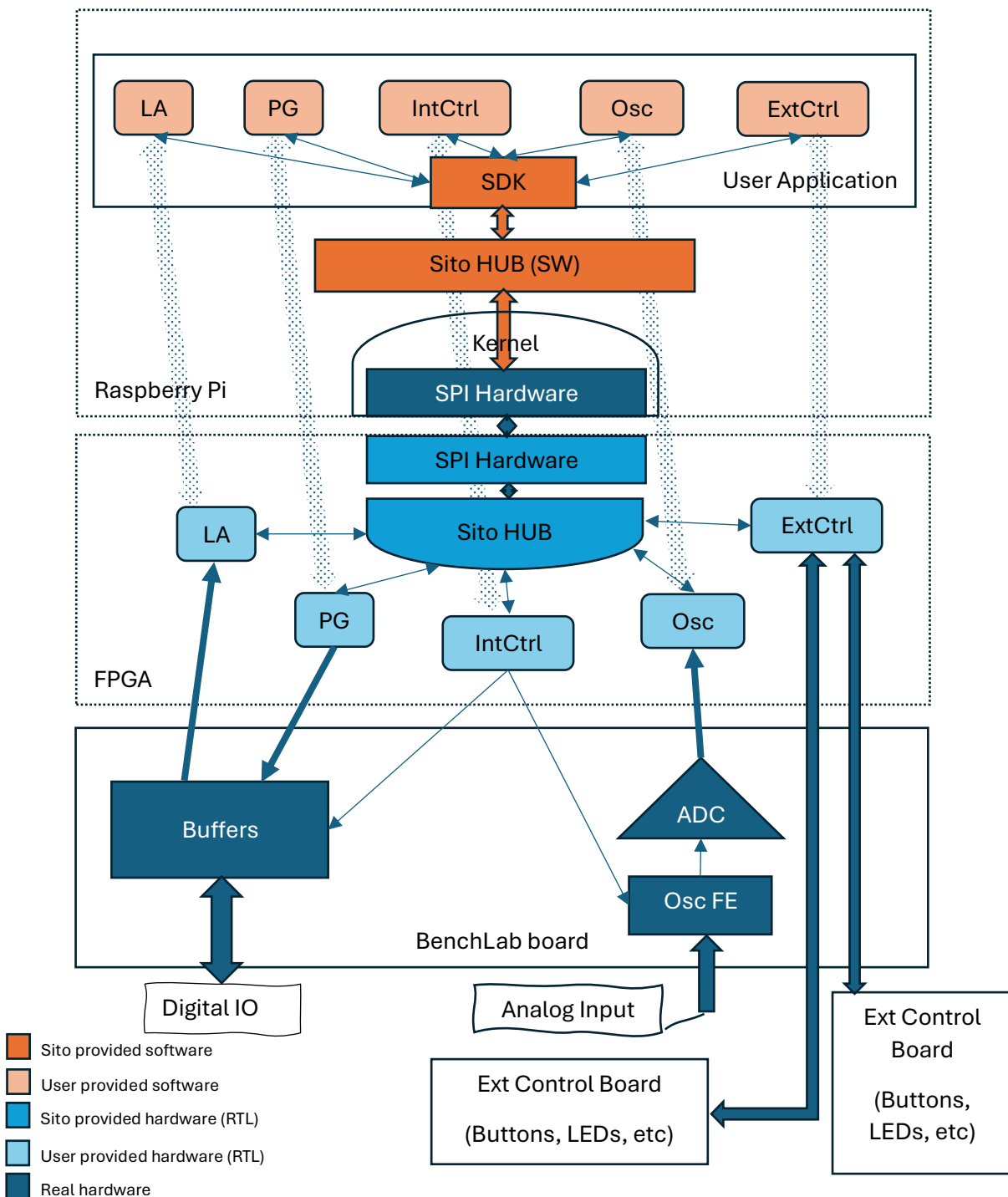
- Logic Analyser.
  - Input bits: up to 32
  - Trace length: up to all available FPGA BRAM
  - Sampling frequency: 100 MHz
- Pattern Generator.
  - Output bits: up to 32.
  - Total input+output bits (of Logic Analyser and Pattern generator): up to 32
  - Granularity for input/output bits selection: 8 bit.
  - Trace length: up to all available FPGA BRAM
  - Total number of supported hardware loops: up to 4
  - Nested loops support: yes
- Oscilloscope.
  - ADC bits: 10
  - Input voltage range (maximum): 50mV – 300V
  - Bandwidth: 20 MHz
  - Sampling frequency: 100 MHz
  - Averaging mode: from 4 to 32768 input samples per output sample
  - Averaging results: single value or minimum/maximum/average triple
  - Triggers: up to 4 (with programmable hysteresis)
- Global triggering system:
  - Complex logic expression evaluator for Logic Analyser part
    - Supports arbitrary NDF expression for all inputs and up to 4 conjunctions.
  - All triggers from Oscilloscope
  - State Machine with 8 states
- External hardware user supplied Control Panels.
  - Up to 2 panels with 9 and 10 I/O pins.
  - Support variety of hardware controls:
    - Buttons
    - LED
    - BiColour LEDs
    - Joysticks
    - Quadrature Encoders
    - Full colour LED strips
  - Fully customizable in software and hardware.

BenchLab includes custom RTL for all of its blocks and a dedicated expansion board for the SitoFA board. This board contains digital buffers, an ADC, and an oscilloscope interface.

BenchLab can be run without a dedicated hardware board. In this case, the logic analyser and pattern generator will use the FPGA's raw I/O pins (which can be dangerous for an FPGA). The oscilloscope block can use the FPGA's built-in ADC (available in the two top FPGA models), but the oscilloscope front end will be unavailable (only one input voltage level and only DC coupling can be used). There is also no hardware overvoltage protection.

User-supplied hardware control panels can be connected even without a dedicated hardware board.

All BenchLab internal blocks are customizable, and some can be eliminated completely.



BenchLab diagram.

Sito HUB (software and hardware) connects each RTL block with the corresponding software block. The connection is represented as a bidirectional packet exchange channel. Each RTL and software block has a unique identifier (8-bit number) that is used to connect them.

Each packet (in either direction) is represented as an array of 32-bit words. Each message in the channel contains this array and its length. The most significant byte in the first word of the array is reserved for the identifier, other bytes/words can carry arbitrary information.

## LogicAnalyzer Block

The LogicAnalyzer (LA) block is built around a RAM block that stores sampled data. The RAM size is configurable (and this size will be specified in the project's static JSON configuration). Each RAM word consists of 2 halves:

1. The high half (always 32 bits) is the TimeStamp value. This is a monotonically increasing counter with a frequency of 100 MHz.
2. The low part (configurable width, up to 32 bits) is the LA inputs

The LA stores the LA inputs in a RAM location when the state of the inputs has changed or when all the TimeStamp bits are 1 (FFFFFFFF).

The LA continuously stores samples in RAM immediately after power-up, and when the Trigger is fired, it simply remembers the RAM address in a special block (see below). This allows us to see what happened BEFORE the trigger, but it requires special handling in software, since the LA trace no longer starts at the beginning of the RAM.

The LA module provides read-only access to both halves of the RAM.

## The HOST -> FPGA packet format

<ID:8> <section:4> <data:20>
[ <data> ... ] - Optional

### Sections:

0. BRAM read request (low word - LA inputs). Data is the starting address. The size to read is determined by the Section #3 packet. Reading starts after the end of this packet.
1. BRAM read request (high word - timestamps). Data is the starting address. The size to read is determined by the Section #3 packet. Reading starts after the end of this packet.
2. Write trigger RAM. Data is the starting address, data to write follows (see Trigger Block Definition)
3. Set the size of the BRAM read request. Data is the size of the BRAM to read

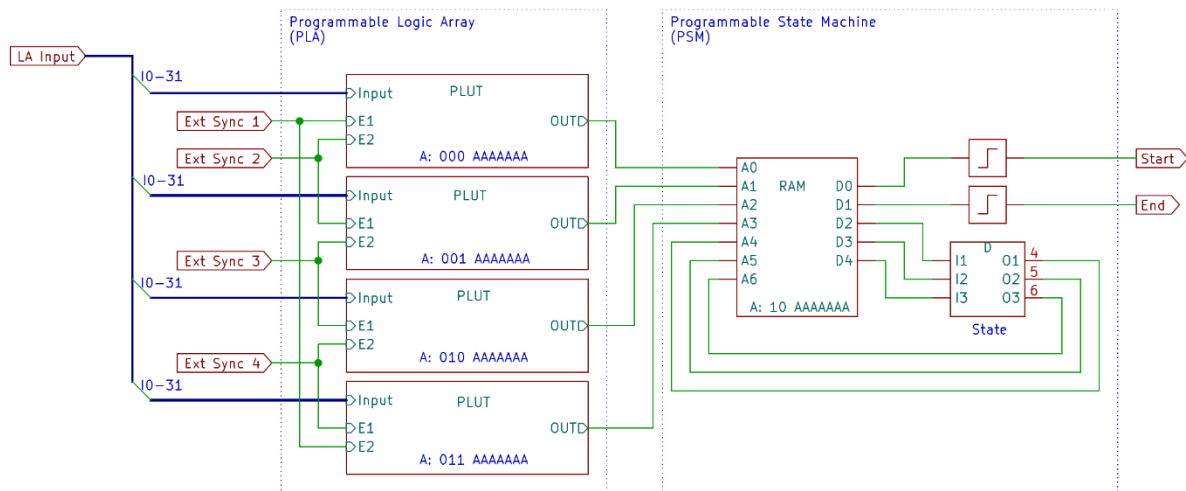
## The FPGA -> HOST packet format

<ID:8><original 24 bits from request>
<read RAM image> ...

## LA Trigger Block

This block analyses the inputs from the LA (and the oscilloscope) and generates start/stop events for the entire system.

It consists of 2 parts: a Programmable Logic Array (PLA) and a Programmable State Machine (PSM).



NB. A: xxx and B: xxx (in pictures below) refer to address of this block in Configuration space (see later)

The PLA takes the LA inputs and up to 4 additional external signals (currently triggers from the oscilloscope block) and generates signals to change the state of the state machine.

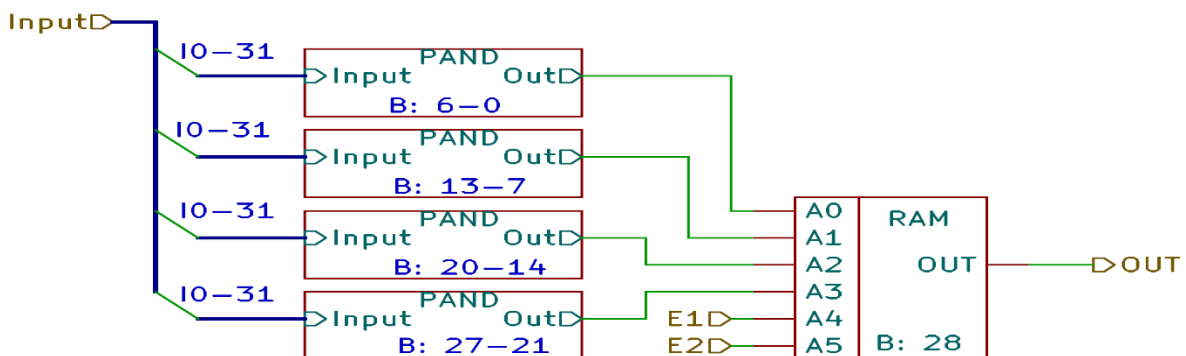
The PSM machine tracks this information and generates 2 output signals - the start and end of the data acquisition session.

Both the PLA and the PSM used RAM blocks to implement arbitrary logic. The FPGA itself uses the same elements for the same purpose.

In FPGA, this is called a LUT (lookup table), and it takes 6 inputs (which are used as an address in the LUT RAM) and 1 output (the RAM output).

Thus, LUT can implement any 6-variable logic function by simply filling RAM with the appropriate content. We use the same RAM (inside LUT) for the same purpose - implementing a programmable logic function.

The PLA itself consists of 4 identical blocks (let's call them PLUTs), which are fed with LA input signals (the same for all 4 blocks) and 2 of 4 external synchronization signals (different for all 4 blocks).

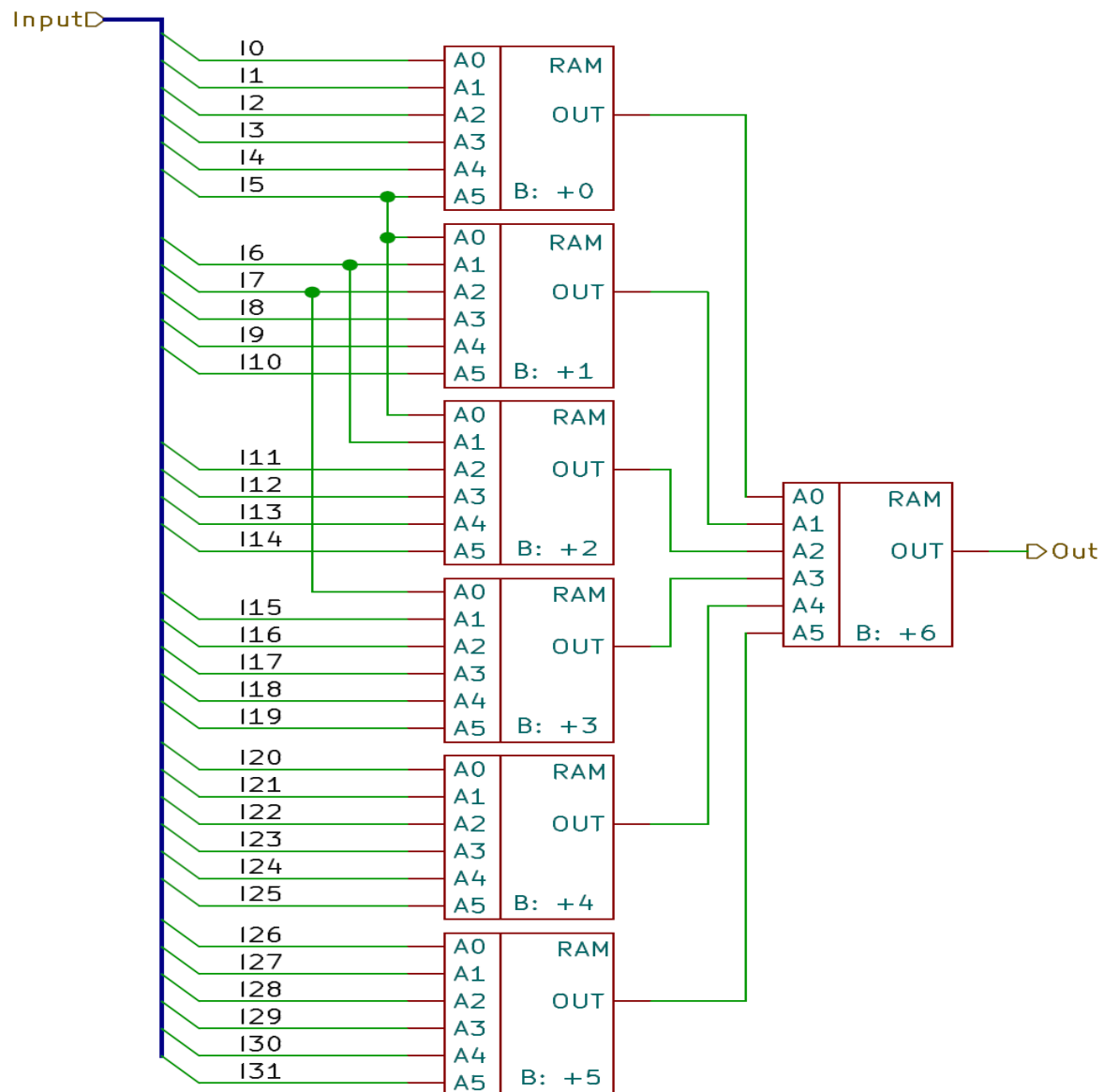


Distributing external event signals across blocks increases the flexibility of event processing.

The outputs of these 4 blocks are used directly as an event for the PSM block.

PLUT, in turn, consists of 4 PAND blocks and output combining logic.

The bottom level of the logic matrix is PAND.

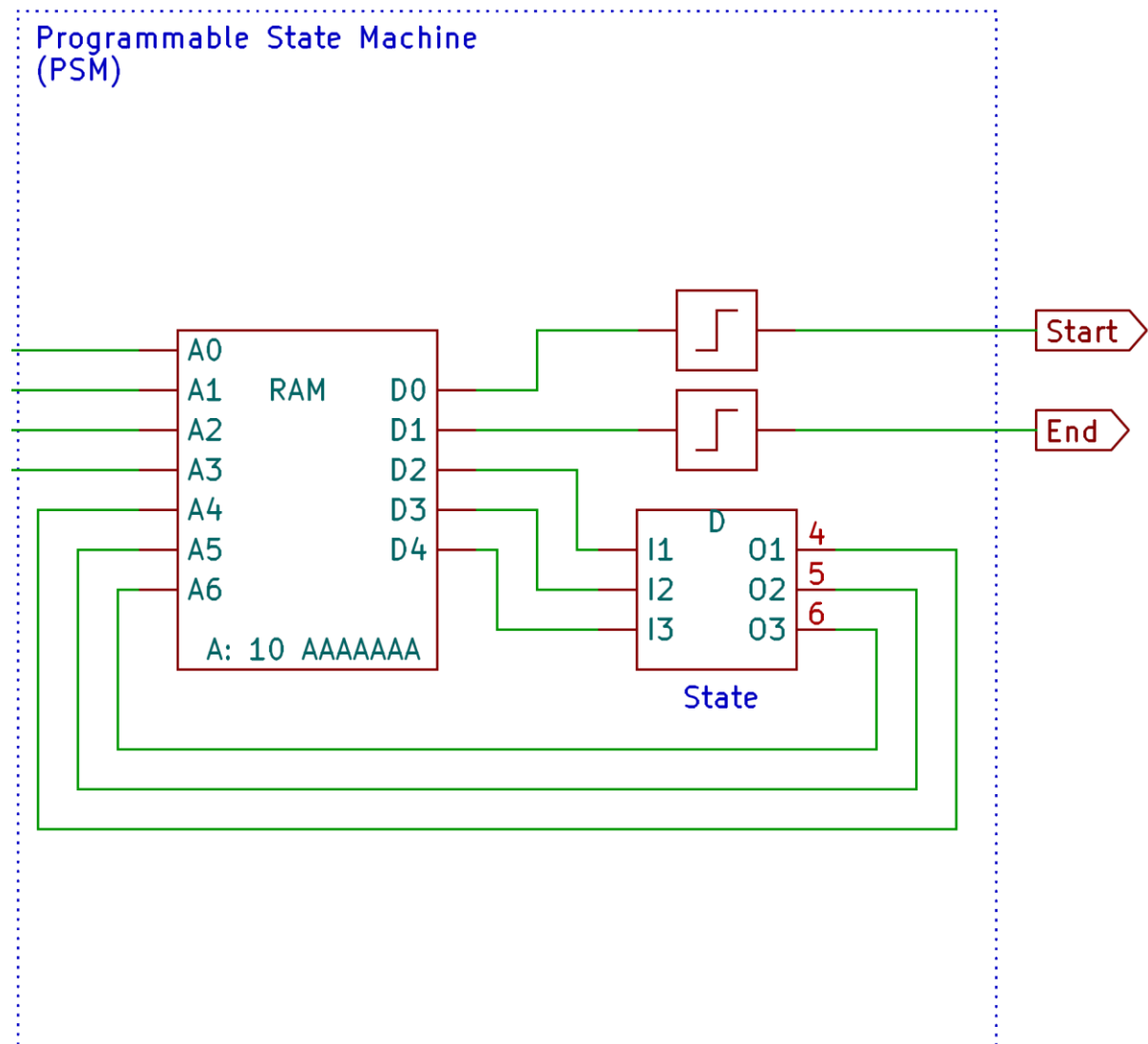


The main idea of the PLUT structure is the ability to represent any logical function in disjunctive normal form. For example,  $A \wedge B$  is the same as  $A \& \sim B \mid \sim A \& B$ . Each PAND matrix evaluates one disjunction over all input values.

PLUT evaluates conjunctions of them. Thus, any DNF of at most 4 conjunctions is supported.

In some corner cases, much more complex logical expressions can be evaluated. For example, any expression with 6 inputs can be evaluated. Some expressions can only be evaluated if the inputs are grouped in a certain order.

The output PSM is implemented as 128x5 RAM.



4 address inputs fed by the output of PLUT blocks, 3 inputs and outputs are used to store the state of the state machine (8 states in total). And the last 2 RAM outputs are used to generate start/stop signals (using leading edges on the corresponding outputs)

## Configuration loading schema.

All RAM images for the entire Trigger block are loaded from the LA block (in section #2). The lower 5/6 bits of the RAM address in the input stream are used as the RAM address for all Trigger RAM blocks (6 bits for the state machine RAM, 5 for all others).

All RAM PLUTs are programmed in one configuration word - different bits of it are directed to different RAMs (the configuration bit indices are shown in the corresponding figures).

Different PLUTs and state machine RAMs are programmed at different addresses of the configuration stream (the addresses are shown in the corresponding figures)

Thus, the effective configuration stream is organized as a RAM image of 284 x 29 bits.

# Pattern Generator Block

This block generates binary signal patterns at its output. Up to 32 bits of output data are supported. The depth of the generated sequence depends on the memory size and can vary (depending on the available RAM on the FPGA).

The block uses the same approach as the logic analyser - it uses <Time Stamp> + <Output Binary Vector> pairs stored in RAM blocks.

But unlike the logic analyser, this block uses the time stamp not as an absolute one, but as a time (in 100 MHz ticks) from the previous one.

The block also contains a Loop generator, which allows generating up to 4 loops (including nested ones). The Loop addresses (the beginning and end of the loop body) and the number of repetitions are fixed and are provided as configuration data.

Loops can be organized in any form, but they cannot intersect with each other. Nested Loops can be placed in any of the 4 available Loop Slots, except when multiple Loops have the same end address.

In this case, they must be placed in the order of their nesting - nested Loops must have a Loop Slot index greater than its parent.

## HOST -> FPGA packet format

<ID:8><Section:4><Data:20>
----------------------------

### Sections:

- 0 - Configuration. <Data> - Configuration data (see below)
- 1-4 - Loop slot 1-4 - Loop Slot setup (see below)
- 5 - Generator RAM. Data is the starting address, followed by the RAM content.

## Configuration

Data: **0...0RA**

- R - Reset. Write 1 here to reset the Generator
- A - AutoStart. If this bit is 1, the Generator will start generating immediately upon system startup. If 0 - only after a Start event received from the Trigger block

## LoopSlot

Data: **<Address>**. The actual configuration data is contained in the 2nd and subsequent words. <Address> specifies the starting index in the Loop Configuration space.

Loop Configuration space layout:

0. Loop parameters: 0...0UE
  - U - The loop is unconditional. If set, the loop will jump to its beginning every time (the loop counter is not used)
  - E - Enabled. If 0, this Loop Slot will be ignored
1. End of loop address. When the generator reaches this RAM address, it jumps to the beginning of the loop (see next word)



2. Start of loop address. This is where the loop begins
3. Counter (32 bits). How many times the loop will be repeated. (Note: A zero repeat counter is not supported)

## RAM data

Data: **<Start RAM address>\*2**. Contents - RAM image. One slot in RAM is represented by 2 words in the input stream:

- $2*A+0$  - Data for the generator
- $2*A+1$  - Timestamp

**FPFA -> HOST packets** - are not generated. The block does not send anything to HOST.

## Oscilloscope Block

The oscilloscope block accumulates the ADC values into the internal RAM. The ADC is 10 bits wide and the RAM is 32 bits wide. Thus, each RAM word contains 3 ADC samples. The upper 2 bits of the RAM word contain the number of samples (1-3). The value 0 denotes the triplet stored from the decimation block.

The ADC is always clocked at its maximum frequency (100 MHz), and to slow down the sampling rate, this block supports decimation of the input samples.

Decimation is performed by averaging the input samples to values that are powers of two. Supported values are from 4 to  $2^{15}$  (32768). Thus, the minimum sampling frequency is about 3 kHz.

The decimation module can provide single samples, in which case they will be packed 3 at a time into a single RAM word (as without decimation).

It can also provide triplets - min-value/max-value/avg-value. In this case, the triplet will be stored in a single RAM word.

The oscilloscope block supports various representations of ADC values. For bipolar input values, it supports two's complement and offset representation (where zero is represented as 1000, maximum negative as 0, and maximum positive as 1FFF).

## HOST -> FPGA packet format

<ID:8><Section:4><Data:20>
----------------------------

### Sections:

- 0 - Control register. Data - control register image (bitset):
  - 0 - use\_tripple\_dec - Store 3 values from the decimator: min/max/avg (the value tag will be 0). If 0, store only avg
  - 1 - signed\_repr - Set to 1 if ADC value is signed (binary's complement)
  - 2 - shifted\_repr - ADC uses zero-shifted representation (ADC value =  $V + 10'b10000000000$ )
  - 3-6 - decimation - Decimator setup. The value of this field is a frequency divider in power of 2 ( $1 \ll \text{decimation}$ ). Minimal allowed value in this field is 2 (or 0 to turn decimation off). The ADC will run at 100 MHz, dividing by averaging samples. Minimum effective sample rate is ~3 kHz

- 1 - RAM Read Size. Data is the packet size to read.
- 2 - RAM Read Request. Data part is the starting address of RAM. Total number of words to read took from write to previous section
- 3-7 - Trigger 1-4 setup. Data is the trigger setup image (bitset):
  - 0 - tr\_down - Trigger on negative signal slope
  - 1 - tr\_up - Trigger on positive signal slope
  - 2-9 - delta - Trigger hysteresis
  - 10-19 - value - Trigger value

If delta is not equal to 0, then the trigger will be activated with hysteresis {value + delta, value - delta}.

## FPGA -> HOST packet

<ID:8><original from request>
-------------------------------

<data> ... - RAM image
------------------------

## Sequencer Block

This block controls the start/stop of the entire system. It also monitors RAM write activity and stores RAM start/stop addresses for all blocks that have RAM for storing external information (currently these are the logic analyser and oscilloscope blocks).

The sequencer blocks themselves do not contain RAM, but they do contain a number of control and data registers. The registers of this block are divided into read-only and write-only registers (there are no registers that are read/write).

The HOST -> FPGA packets are used to access the registers and accept commands.

The sequencer block processes the start/stop signals from the trigger block. It also monitors timing and RAM write activity and forcibly terminates the accumulation session if there is a timeout or RAM overflow.

To perform this task, the user must program this block for the maximum accumulation session time (in 100 MHz ticks) and the maximum number of words that can be written to all available RAM.

Care must be taken to limit RAM writes to a maximum of the RAM size to avoid overwriting the RAM.

## HOST -> FPGA packet format

<ID:8 bits> <section:4 bits> <data:20 bits>
---

### Sections:

0. Commands. Data is a bit set of:
  - 0 - Enable the sequencer block. After writing a 1 to this bit, the entire system will start (and wait for the Trigger)
1. Write register(s). Data is the address of the register to write, followed by the data to write. Multiple data words allowed - multiple registers will be written.
2. Read register. Data is the address of the register. The reading process begins after the end of this packet. Only one register can be read at a time.

## FPFPGA -> HOST packet format

<ID:8 bit><2 (section):4 bit><reg-index:20 bit>	full copy of HOST->FPGA packet header
<register image: 32 bit>	Register image response to register read request.

### Read-only registers (can be read via read request in section #2)

0. Sequencer status (bit set)
  0. Sequencer running
  1. Sequencer recording started by Trigger
  2. Sequencer stop expected
  3. Sequencer stopped by maximum sequencer execution range
1. Sequencer start timestamp (in 100 MHz ticks). Contains the timestamp value when the trigger was fired.
2. Sequencer end timestamp (in 100 MHz ticks). Contains the TimeStamp value when the accumulation session was terminated (by trigger or TS/RAM limits)
3. Sequencer Start RAM Address. Contains the RAM address when the Trigger was fired. This is important because all RAMs in the system are filled continuously, without waiting for a Trigger.
4. Sequencer End RAM Address. Contains the RAM address at the end of the accumulation session.
5. ....

Registers 3 and 4 will be replicated (to addresses 5 and 6, 7 and 8, etc.) to all RAMs in the system.

### Write-Only Registers (System Setup. Can be written via Section #1)

0. Sets the maximum execution range in 100 MHz ticks
1. Sets the session end deferral in 100 MHz ticks. If this register is non-zero, the acquisition session will be terminated with a delay. Care must be taken with the maximum RAM fill limit when using this register - the deferral will be unconditional, so RAM overflow is possible.
2. Setting for maximum filled RAM addresses in words.
3. Setting for deferral of termination in RAM addresses. If this register is non-zero, then the completion of the accumulation session will be delayed until this RAM has collected additional words.
4. ...

Registers 2 and 3 will be replicated (to addresses 4 and 5, 6 and 7, etc.) for all RAMs in the system.

## Internal control unit

This unit controls the BenchLab hardware. This unit directly controls 2 chips:

1. **TCA9535** - I2C expander, which in turn controls several one wire control points inside the hardware (see below)
2. **LMH6881** - PGA in the ADC path (controls the gain of the oscilloscope part)

## Connection of the I2C expander output (port numbers)

0-3	Direction control for the logic analyser and pattern generator bytes. 0. in the bit enables the logic analyser path (the signal will be connected from the external connector to the FPGA pin). 1. will enable the pattern generator mode (signal flow from the FPGA to the output connector)															
4	~OE for the logic analyser/pattern generator. 0. enable all inputs/outputs. 1. disable all 32 signal paths. This pin has a pull-up, so all paths will be disabled on startup.															
5	Oscilloscope AC/DC input select coupling. 0. DC coupling 1. AC coupling															
6	~CS for PGA (LMH6881). The PGA serial control lines are connected directly to the FPGA, but the CS is not. So, to access the PGA, you need to set this pin to 0, then send the appropriate packet, and finally set this pin to 1 again.															
7	ADC ~EN signal. 0. enable the ADC (and the oscilloscope) 1. disable the ADC (to save power)															
10-15	External LEDs. Set 0 to enable the LED. 3 of these LEDs should be used to indicate the activity of the oscilloscope, logic analyser, and pattern generator (which one will be announced later). The others can be used for any purpose.															
16-17	Scope Attenuator Select <table><tr><td>Bit 17</td><td>Bit 16</td><td>Attenuator setup</td></tr><tr><td>0</td><td>0</td><td>GND (selected at startup)</td></tr><tr><td>0</td><td>1</td><td>1:1</td></tr><tr><td>1</td><td>0</td><td>1:10</td></tr><tr><td>1</td><td>1</td><td>1:100</td></tr></table>	Bit 17	Bit 16	Attenuator setup	0	0	GND (selected at startup)	0	1	1:1	1	0	1:10	1	1	1:100
Bit 17	Bit 16	Attenuator setup														
0	0	GND (selected at startup)														
0	1	1:1														
1	0	1:10														
1	1	1:100														

## HOST -> FPGA Packet

<ID:8><Section:4><Data:20>

### Sections:

0. PGA SPI Write. The lower 16 bits of data are transferred serially to the PGA
1. I2C Expander. The data is the I2C data to write (see below)
2. I2C Bus Reset (not implemented yet)

## I2C Write Data Format

<R:3><L:1><B2:8><B1:8>

- L - I2C Data Packet Length (0-1 byte, 1-2 bytes)
- R - Register Index to write
- B1 - First byte to write
- B2 - 2nd byte to write (if L is 1)

I2C only supports 1/2 byte write transaction (excluding register index).

### I2C Write Format:

- 1 byte: <Start><01000000><ACK><00000RRR><ACK><B1....><ACK><Stop>
- 2 bytes: <Start><01000000><ACK><00000RRR><ACK><B1....><ACK><B2...><ACK><Stop>

<ACK> is generated by TCA9535 and verified by FPGA. If <ACK> is not present, I2C transmission stops and <Stop> is generated.

The host is notified of the error via the Communication Error Reporting Channel (not yet implemented)

## I2C Bus Reset Algorithm

- 1) The master attempts to set the SDA line to logic 1
- 2) The master still sees a logic 0, then generates a clock pulse on SCL (1-0-1 transition)
- 3) The master checks SDA. If SDA = 0, go to step 2; if SDA = 1, go to step 4 (maximum number of repetitions is 9)
- 4) Generates a STOP condition

Note that this process may need to be repeated, since the cleared SDA line may have been cleared for the next bit, which was 1.

## ExtControl Block

This block performs access to external control panel(s), attached to 2 dedicated sockets. These panels can comprise variety of physical controls.

Supports up to 19 external controls (grouped into 2 groups of 10 and 9 pins), such as:

- Disabled
- Pushbutton
- QEncoder
- LED
- Bi-colour LED
- Full-colour LED strip

Any of the 19 I/Os can support a pushbutton and an LED

A pair of configured pins are used to support a QEncoder and a bi-colour LED

1 full-colour LED strip controller can be connected to the last pin of any group (2 possible pins in total).

### Characteristics of each control:

- Disabled
  - Represented as an input (internally disabled) with pull-up
- Pushbutton
  - Has adjustable debounce (in the range of 0-127 ms)
  - Has adjustable delay (in the range of 0-15 ms)
  - Has an interrupt mask (1 bit)
  - Debounce and delay are performed with an accuracy of 1/16 ms
- QEncoder
  - Has adjustable debounce (like a pushbutton)
  - The encoder distinguishes between 8 possible edge configurations: positive/negative edge (2) x on channel A/B (2) x state of the opposite channel (2).

- The decoder generates pulses for +/- 1 for any set of edges (configurable). Edge Table:
  0. Positive Edge A + 0 on Channel B
  1. Negative Edge A + 0 on Channel B
  2. Positive Edge A + 1 on Channel B
  3. Negative Edge A + 1 on Channel B
  4. Positive Edge B + 0 on Channel A
  5. Negative Edge B + 0 on Channel A
  6. Positive Edge B + 1 on Channel A
  7. Negative Edge B + 1 on Channel A
- LED
  - LED output can be set to 0/1
  - Also used for FullColor LED strip.
- Bi-colour LED
  - LED colour is determined by the polarity applied between its contacts. LED contacts are connected to 2 serial contacts.
  - The system supports mixed colour by applying PWM on the outputs. PWM frequency is 1 kHz, duty cycle is 0, 1/4, 1/2, 3/4, 1. OFF state is also supported. Values (3 bits):
    0. OFF
    1. PWM, dcycle is 0
    2. PWM, dcycle is 1
    3. PWM, dcycle is 1/4
    4. PWM, dcycle is 1/2
    5. PWM, dcycle is 3/4
    6. Reserved
    7. Reserved
- Full colour LED strip
  - Support almost any length of LED strip.
  - Support 2 dedicated pins (last in each group). The pin used must be programmed as LED in the pin configuration

## Packet format (HOST -> FPGA)

<ID:8><Section:3><HdrData>
...

- ID - module ID
- Section - target section for this packet (see below).
- HdrData - header part of data. Size and contents depend on section.

## Sections

- Command: <ID:8>000<Command:21>  
Execute simple command. List of commands:
  0. NOP (do nothing)
  1. Request to send current button state
- Configuration page: <ID:8>001<SubPage:3><CfgData:18>  
Miscellaneous configuration data. The type of data is defined by the SubPage field.

- Pin configuration:

<ID:8>0010000..0AAAAA
<CfgData:32> ... [up to 10 pcs]

Section contains 32-bit configuration words (1 per pin pair + FullColor LED configuration)

Internally treated as a memory area. CfgData contains the starting address in this area for writing (A)

Format of <CfgData> see below

Each CfgData defines the configuration for 2 consecutive pins. The last pin has no next pin, so its configuration should only define the first pin, the second should be set to "Disabled".

However, you can specify a BiColor configuration for the last pin slot. In this case, you will get some kind of PWM-controlled LED

- FullColor LED configuration: **<ID:8>001 001 0..0 <FreqDiv:8>**  
Setting the frequency divider for the path shape generator. The output frequency will be  $100 \text{ MHz} / (\text{FreqDiv} * 3)$
- QEnc transition mask configuration:

<ID:8>001 010 0..0 AAA
0..0 10 x <-+> ... [up to 8 pcs.]

Writes an array of event indices for all QEncs (one word - one index for all QEncs)

CfgHdr specifies the starting address in the event array to write

QEncoder is connected as a proxy between 2 instances of the button and the output scale. It generates '+' and '-' pulses as short button presses (the first for '+', the second for '-')

Which of them ('+' or '-') will be emitted is determined by this configuration array. The QEncoder event type is used as an index into the array, and the QEncoder's own index is used as an index into the bit scale contained in this array cell.

- LED set group: **<ID:8>010 ...**
  - Bulk set: **<ID:8>010 00<D:19>**

Set all LEDs (and bicolour LEDs). Affects LEDs only.

Bicolour LEDs occupy 2 consecutive bits and set only the pure colour. The combination '11' for a bicolour LED is ignored (LED state does not change)

- Bulk set with mask:

<ID:8>010 01<D:19>
0..0 <M:19>

Update LEDs for which the M bit is 1 (as in the 'bulk set' command)

- Set individual LED: **<ID:8>010 10...0 AAAAA VVV**

Set LED number <A> to value <V>. For a bicolor LED, all 3 <V> bits are used, for a simple LED, only bit 0

- Set up a full-colour LED strip:

<ID:8>011 0..0<Total:12>PP
----------------------------

<Data> ... [up to 1023 words]
-------------------------------

Send <Data> to the LED strip.

- <Total> - Data size (in bytes). Should be equal to <number of LEDs in the strip>\*3. A maximum of 1365 LEDs in a strip are supported
- PP - Bit set {<pin 19><pin 10>} of the target pins to send to the LED strip. The target pins must be configured as LED pins.

## Packet format (FPGA -> HOST)

<ID:8>00000<BtnState:19>
--------------------------

This packet will be sent by the FPGA if any of the following events occur:

1. The state of any button (with the 'interrupt enable' bit in the configuration equal to 1) has changed
2. Any QEncoder generates a positive or negative pulse
3. The 'Request to send current button state' command has been received

The packet is generated asynchronously (except for the last case)

## Configuration word format (CfgData)

Each configuration word describes the configuration for 2 consecutive pins. For a bicolor LED and a QEncoder, it describes the configuration for that entire object.

### 2 Pin Configuration:

<T1:2><T2:2><C1:14><C2:14>
----------------------------

- T1 - Type for Pin 1
- T2 - Type for Pin 2
- C1 - Configuration Data for Pin 1
- C2 - Configuration Data for Pin 2

Types:

- 00 - Disabled
- 01 - LED
- 10 - Pushbutton
- 11 - Invalid (used for QEncoder and Bicolour LED)

## BiColor and QEnc Configuration

11<T><C:28>
-------------

- T - Type (0 - Bicolour, 1 - QEncoder)
- C - Configuration Data



## Configuration Data Format

### *Disabled*

No Configuration

### *Pushbutton*

**00ddddDDDDDDDI**

- I - Interrupt Enable. If this bit is 1, any change in the state of this input will cause an FPGA->HOST packet to be sent with the current state of all buttons
- D - Debounce time (in ms)
- d - Delay (in ms)

When a button state changes, the system will block further button changes from being sent for <D> ms. The new button state will be registered as current (and sent to the host if needed) after a delay of <d> ms.

This is used to handle buttons that can be pressed in a group.

Example use case: the "hit" button on a joystick - it will be pressed at the same time as all the "direction" buttons.

In this configuration, all the "direction" buttons should have a delay, so when any of them is reported pressed, the status of the "hit" button will be known and they can all be safely masked.

### *LED*

**0..0 V**

- V - Setting the current state of the output pin

When used to configure a FullColour LED strip, the V bit determines the inversion of the output.

### *BiColour*

**0....0 VVV**

- V - Setting the current colour of the Bicolour LED

### *QEncoder*

**2 x <button config>**

The debounce configuration is filled in the same way as 2 Button config. The <I> fields are not used (reserved, should be 0)