ELEKTROTEHNIŠKI VESTNIK 82(4): 205-211, 2015
ORIGINAL SCIENTIFIC PAPER

A Generic Timing Receiver for Event-Driven Timing Systems

Benjamin Ocepek

Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška 25, 1000 Ljubljana, Slovenija
Cosylab d.d., Teslova ulica 30, 1000 Ljubljana, Slovenija
E-pošta: benjamin.ocepek@cosylab.com

Abstract. An important part of the modern particle accelerators is the timing system. It is a big and complex system that contains many different technologies and requires a wide spectrum of knowledge. Development of such a system or even just configuring the "of the shelf" solutions can be a frustrating and time consuming job. As a part of an internship at Cosylab d.d., I created a simple receiver for an event-driven type of the timing system to be used in many particle accelerators. The device provides a tool for an easier and faster debugging. The module is simple to use and at the same time also optionally generic to meet the needs of different facilities using different variants of the timing system. Currently it is implemented on an FPGA-based development board and can be connected to the user's workstation using a serial interface. Software with a graphical user interface was also created.

Keywords: timing, control, accelerator, FPGA, EPICS

Generični sprejemnik dogodkovno gnanega sistema za sinhronizacijo takta ure

Pomemben del modernega pospeševalnika delcev je sistem, ki skrbi, da se stvari dogajajo ob pravem času. Tak sistem je zelo obsežen in zapleten, saj vsebuje veliko različnih tehnologij in posledično zahteva širok spekter znanja. Razvoj takega sistema ali zgolj pravilna nastavitev že obstoječe komercialno dostopne rešitve je lahko neprijetno in dolgotrajno opravilo. V članku je predstavljen sprejemnik za dogodkovno gnan sistem za sinhronizacijo urnega takta, ki se uporablja v pospeševalnikih delcev. Namen naprave je, da deluje kot orodje za hitrejši in lažji razvoj ter razhroščevanje. Naprava mora biti preprosta za uporabo, a hkrati tudi čim bolj univerzalna, saj različne ustanove uporabljajo različne variacije sistema. Naprava je trenutno razvita za razvojno ploščo FPGA in je s serijsko povezavo povezana z uporabniško delovno postajo. Razvita je bila tudi programska oprema z grafičnim vmesnikom.

1 CONTROL SYSTEM

Particle accelerators are some of the most complex and expensive structures built by men. They are used for many different applications, not only for the fundamental physics research as one would imagine, so consequently they come in many different forms and sizes. Bigger projects can have thousands of devices which can come from thousands of different sources. For the accelerator to work, all these devices need to work together towards the desired goal. An additional complication is that the devices can use different communication protocols to communicate with the environment. It is obvious that some kind of a unified control system (CS), i.e. a "software framework", is needed to address all these devices in the same way.

Usually, one of the existing CS packages is used. A very popular solution also used for the purpose of this project is the open-source EPICS (Experimental Physics and Industrial Control System). It is a distributed control system that uses a network-based client/server model. The servers are called IOCs (input-output controllers). They are usually connected to variety of hardware such as diagnostic cards, power supplies, timing cards, etc. The data acquired is then available to the clients using the CA (channel access) network protocol. Typical clients are the operator screens, alarm servers, archive servers, etc. An important point is that all data is available to the clients in the same way, regardless of the underlying hardware or communication protocol used by the specific device to connect to the IOC.

In EPICS IOCs, the values are stored and manipulated using different types of records. Apart from values, the records can also store other information, like the alarm status and names of other records connected to it. A record is a structure made of different fields where all this information is stored. A record name and field name combined give the name of a PV (process variable). In order to exchange the data between the CS building blocks, CA needs a PV name.

The records communicating with the hardware need to have a specified device support. The device support is a program that provides an interface between the EPICS records and the device.

Received 13 July 2015
Accepted 12 August 2015

2 TIMING SYSTEM

Some parts of the accelerator have hard real-time requirements, so a separate timing system (TS) is required. In order to achieve the tight requirements and determinism, TS is implemented on FPGAs (field-programmable gate array), which is essentially a programmable hardware.

There are generally two TS types. They can be based on time synchronization or they can be event-driven. The time synchronization TSs use special techniques to synchronize all the timing devices to the same time and clock frequency/phase. Then a time table of the actions to be taken is generated and distributed over the Ethernet protocol to all the devices.

In the event-driven TSs there are in principle two basic types of the devices, i.e. the timing generators and timing receivers. The generators generate an event, an eight bit number, which is then broadcasted over a dedicated deterministic network to each receiver. The events can be triggered by different sources such as external signals or periodically by internal counters. The receivers are then programmed to respond to specific event codes in various ways. Usually, they provide a trigger for some other device, such as a diagnostic card or power supply.

The rest of the paper is focused on the event-driven TSs, since the developed and described device is compatible only with the event-driven TSs.

3 PURPOSE OF THE DEVICE

When developing a new timing device or configuring an “of the shelf” solution the time spent can be significantly reduced by using good debugging tools. Currently the available devices for an event-driven TSs provide rather poor debugging utilities and require quite a lot of configuring. Therefore the developed timing device should be easy to use and at the same time generic.

The generic timing receiver developed at Cosylab can be connected to TS and can quickly determine if TS is working. The timing events broadcasted by the event generator can be latched by the generic timing receiver and forwarded to the user’s workstation, where they can be displayed in the form of a graph using the developed software. Besides this key feature, the device can be configured to produce pulses on expansion pins and LEDs upon receiving the selected events. Some of the settings that can also be configured are described below.

4 HARDWARE

ML507 development board equipped with all the needed peripheral ports is used. It contains a Xilinx Virtex 5 FPGA which is powerful enough and has all the needed IP primitives. The code is written in VHDL using the Xilinx ISE development environment.

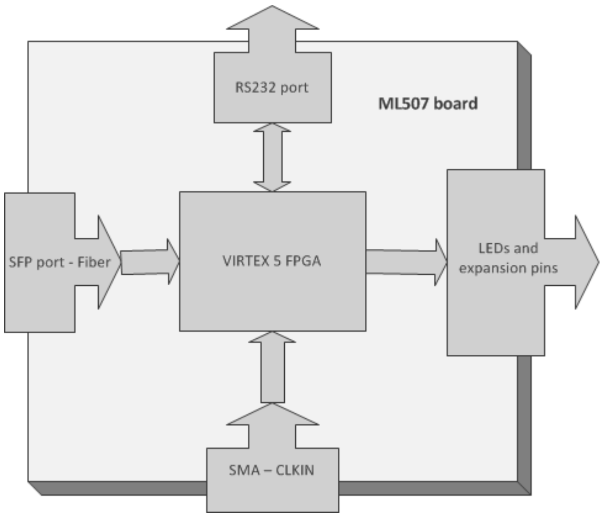


Figure 1. Block diagram of the hardware

It is connected to an optical fiber channel with an SFP (small form-factor pluggable) high-speed transceiver. For communication with the user’s workstation, an RS-232 port is used, optionally in combination with an external RS-232 to USB serial converter. Some status signals are routed to the on-board LEDs. The output pulses are also routed to LEDs in addition to expansion pins. An SMA connector is used as an input for an optional external clock source. Figure 1 shows the used hardware modules.

5 FIRMWARE

The basic design idea is shown in Figure 2. A high speed serial transceiver IP core (GTX) is used to align to the fiber channel data stream and to perform decoding and deserialization. The data is available as two-byte wide frames, where one byte is considered a timing event channel. The event characters are then forwarded to a mapping RAM, where it is decided which actions will be taken. Depending on the content of the mapping RAM, an event can trigger generation of pulses on the output, reset the internal timestamp counter and/or forward itself and the current timestamp to the user’s workstation through a serial interface. Additional actions can be implemented with a relatively little effort. The relation between the events and actions is many to many, so more than one event can cause one action and one event can cause multiple actions concurrently.

A soft-core UART (Universal asynchronous receiver/transmitter) from the OpenCores with a custom made UART parser is used to provide a serial interface to the user’s workstation. The settings are stored in an internal memory and are directly available to the user who can see the generic timing receiver as a memory-mapped device.

The design is divided into different clock regions. GTX requires a reference clock. After successful alignment, it provides a recovered clock from the fiber

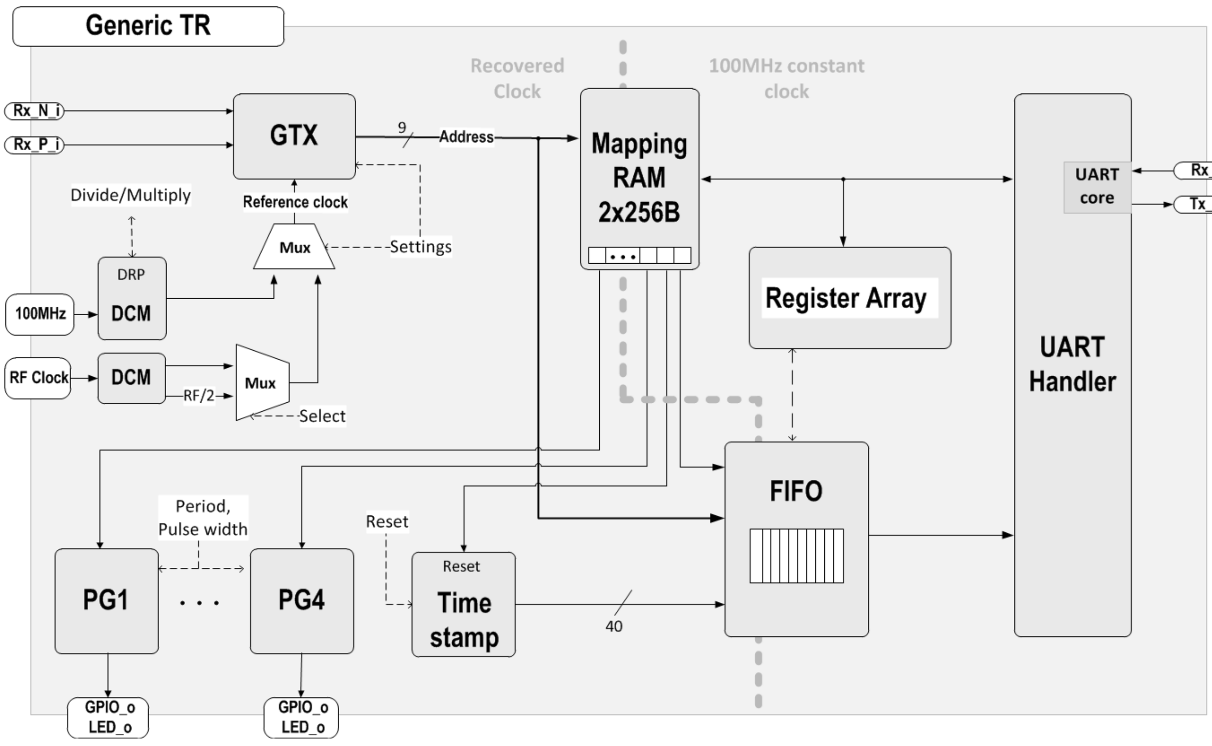


Figure 2. Block diagram of FPGA firmware

on which the whole TS operates. It is also known as an event clock. The timestamp and pulse generators always operate on the event clock. There is another clock source driving the UART. It enables a fixed baud rate regardless of the fiber status. Different methods of synchronization between the clock regions have been implemented, such as the use of dual-port RAMs and signal synchronization modules to prevent metastability [1].

5.1 *GTX transceiver*

The fiber channel operates on the line rate in the GHz order. This means that it is not possible to use the ordinary primitives to manipulate the data stream so dedicated hardware is used. The version of Virtex 5 used provides a dedicated high-speed serial transceiver called GTX that supports high data rates and a variety of protocols. Configuring it proves to be a non-trivial task.

The protocol employed in the event-driven TS uses an 8B10B encoding scheme which maps the eight bit data to the ten bit symbols. A feature of the 8B10B code is DC balance (the number of zeros and ones emitted is on average the same) and there are no more than five emitted ones or zeros in a row [2].

In addition to the 256 possible bytes, there are also some special characters, called control characters, that are also a valid sequence of ten bits, but they don’t have a corresponding eight bit data byte. One of these control characters is called the comma character which is sent from time to time (typically every fourth character). This enables data alignment and clock recovery on the receiver.

The data is transmitted in two-byte wide frames. One byte is an event bus on which the actual event codes are transmitted and the other byte can be used for other purposes for various TS designs. My design supports switching between the bytes on request.

Another important TS aspect is determinism. It can be achieved only if all the receivers run on the same clock. In this case this is the recovered clock from the fiber. Consequently the elastic buffer in GTX can be bypassed and the delays are constant. There is another potential problem related to the barrel shifter which is used as part of the serial to the parallel conversion process [3]. This can cause non determinism in the phase of the recovered clock. It is not yet determined if this behavior occurs in my design too.

5.2 *Clocking*

The GTX transceiver requires a reference clock with the frequency within 100 ppm (parts per million) of the event clock. The event clock is 20 times lower than the line rate on the fiber, so for example, in case of a 2 GHz line rate, the event clock is 100 MHz. The event clock frequency is chosen depending on various physical properties of a given particle accelerator, so this value can be practically anything inside the supported bounds.

There are two options for the reference clock provided in the generic timing receiver. The first option is to use an external clock source, such as a functional generator or an accelerator master oscillator that can be brought to the SMA connector. PLL (phase locked loop) is used to divide the external clock by two if needed. The second option is to use an on-board 100 MHz oscillator and generate the required frequency using a DCM (digital clock manager) IP core which is configurable "on the fly" using the DRP (dynamic reconfiguration port) protocol. The main drawback of this option is that DCM supports only the limited values of the multiply and divide values for PLL used inside. As a result, not all frequencies can be achieved that way. A possible solution would be to use a programmable fractional-N frequency synthesizer for clock generation. It can generate basically any frequency in the supported band (with less than a 100 ppm offset) [4]. It would potentially allow implementation of another useful feature, i.e. frequency scanning. The idea is to slowly increase the frequency of reference clock until GTX is aligned to the fiber data stream. This way, the knowledge of the exact frequency of the event clock would not be required. It is not yet determined whether this idea can be implemented. However, adding a fractional synthesizer is a high priority task for the future development of this device.

All settings regarding clocking are stored in a register array and are visible and configurable through the UART.

5.3 Memory

The event mapping RAM is a table defining the actions to be taken when a particular event number comes from GTX. The events address RAM where the bitwise content of RAM on this address determines which actions need to be carried out. The content is configurable through a serial interface. Since UART is in a different clock region than the received events, a dual port RAM is used.

The address space is divided into three parts:

- Event mapping RAM defining what will happen when a specific event character is received.
- Control character mapping RAM defining what will happen when a specific control character is received.
- Configuration register array (a 32 bit-wide distributed memory) is storing settings for various components of the device.

As the block memory (dual-port RAM) has no reset signal, RAM is reset by manually writing default values on all RAM addresses. This is implemented in the software.

5.4 UART

The MiniUART IP Core from the OpenCores (developed by Philippe Carton) is used to make an interface between the generic timing receiver and the user's workstation. The core receives and sends one byte at a time and does not include FIFO. A special UART handler is developed to store and parse the input strings, extract the data and decode it from the ASCII notation. All control signals are correctly driven and correct responses are generated with the data from the memory coded in the ASCII notation and placed in the right place in the string.

When enabled, the device automatically up-streams the received events marked for this action. Along the event number, the local timestamp is also sent. The timestamp is simply a current value of a 40-bit free running counter operating on the event clock. FIFO makes sure that no events are missed even when a high burst of events occurs. If the FIFO overflows, the whole upstream is disabled.

The UART handler is implemented as a complex FSM (finite-state machine) which proved to be a bad decision for making it hard to add new features and modifications. However, some tests on a small group of data showed that all the received upstream data is correct and undamaged. Large-scale tests with emphasis on the simultaneous downstream and upstream operations (read and write to the registers) are still needed to verify that no data is lost or corrupted by UART handler.

6 SOFTWARE

The software is developed as an EPICS IOC with the StreamDevice 2 module acting as device support (interface between the EPICS records and the device). CSS (Control System Studio) is used as a platform for the GUI (graphical user interface). Each of these components is described below.

6.1 EPICS

Every function of the generic timing receiver has a group of associated records. There are separate records for the input and output signals, the so-called "getters" and "setters". Whenever the value of any register is to be changed, an appropriate "setter" record is processed and the forward link to the "getter" record is asserted and consequentially the "getter" record is also processed. This makes it possible to check if the data is actually saved in FPGA.

The other important issue solved with this approach is what should be done at IOC initialization? The records in IOC need to be synchronized with the FPGA registers. This can be done in two ways. One way is to select the default values and then write them to FPGA at every IOC initialization. The other way is to simply read the values from the FPGA registers and store them in the EPICS records at every IOC initialization. Second way is used in my case.

The other types of records are mostly associated with data manipulation and reset sequence. When a user asserts the record dedicated to global reset, a global event is posted. Each record group includes a reset record triggered by the global event. The default values (usually 0) stored in reset records are written to all FPGA registers when a reset occurs.

Display of the received timing events in a graph form is done by using two synchronized records or two fields of the same record; one storing event number and the other storing the timestamp. In my case the timestamp is a 40-bit wide integer but the used EPICS records can only store 32-bit integers. To solve both problems, an aSub type of the record is used. It is a MIMO (multiple in multiple out) system that can call the custom-written C subroutines (which can internally support 64-bit integers). A C function is written to construct a timestamp and convert it to a floating point value presented on the output. The function also takes in the event number and simply forwards it to the second output. Since the record is processed only once for every received event, the outputs are synchronized.

For every functionality there are minimally three records, making their total number to be some 4000 records.

For the future development it is planned to implement a way to save the events with the corresponding timestamps into a text file, to be used by other data-analyzing programs.

6.2 *Stream Device*

The StreamDevice 2 device support module parses and formats the incoming and outgoing strings. It is a supplement to the AsynDriver which handles the actual serial transmission. The module is relatively easy to use. It generally requires only regular serial protocol parameters, such as the baud rate and the number of stop bits. Additionally, it requires the so-called protocol files. These are the ASCII files in which the “shape” of the transmitted strings is defined. The format converters, similar to the ones used in C, define which part of the string is the actual data to be saved in the record and in which notation it is written (binary, decimal, hexadecimal, float, etc.).

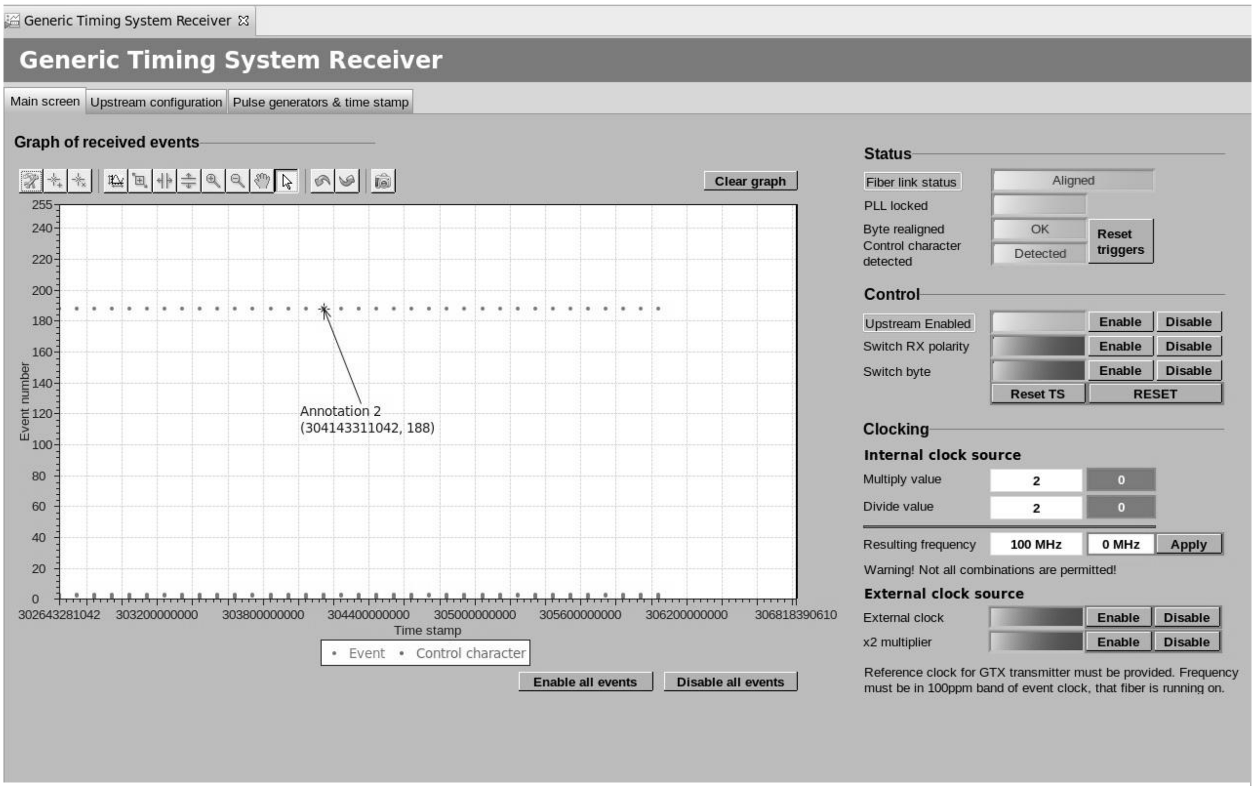
In case of the generic timing receiver, the communication takes place in three different ways:

- When writing in the registers, the address and data are sent to the device which saves the data but does not respond.
- When reading from the registers, the address of the register is sent to the device which responds by sending back the content of that register.
- When the event upstream is enabled, the device automatically sends the event numbers and timestamps without any request from the software.

These three ways of communication do not play well together in terms of the StreamDevice. As a result, when an automatic event upstream is enabled no other communication with the device works properly. This problem will be addressed in future.

6.3 *Graphical User Interface*

CSS (Control System Studio) is an application based on the Eclipse framework [5]. It is mainly used for development of graphical interfaces. It acts as an EPICS client. As it communicates with IOC using the CA network protocol, it does not have to run on the same physical machine.

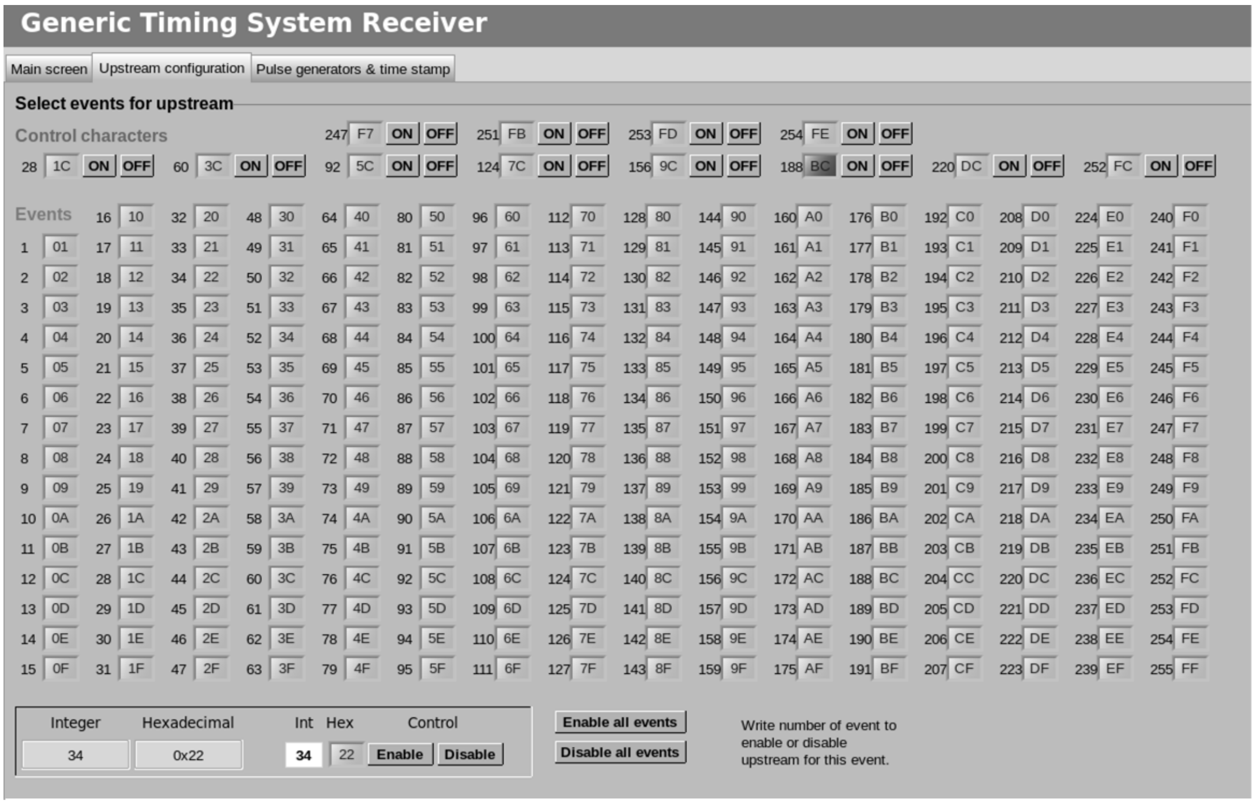


Status
Fiber link status: Aligned
PLL locked
Byte realigned: OK
Control character detected: Detected

Control
Upstream Enabled
Switch RX polarity
Switch byte

Clocking
Internal clock source
Multiply value: 2 | 0
Divide value: 2 | 0
Resulting frequency: 100 MHz | 0 MHz
Warning! Not all combinations are permitted!
External clock source
External clock
x2 multiplier
Reference clock for GTX transmitter must be provided. Frequency must be in 100ppm band of event clock, that fiber is running on.

Figure 3. Main screen


Generic Timing System Receiver

Main screen | Upstream configuration | Pulse generators & time stamp

Select events for upstream

Control characters

247	F7	ON OFF	251	FB	ON OFF	253	FD	ON OFF	254	FE	ON OFF

28	1C	ON OFF	60	3C	ON OFF	92	5C	ON OFF	124	7C	ON OFF	156	9C	ON OFF	188	BC	ON OFF	220	DC	ON OFF	252	FC	ON OFF

Events

		16	10	32	20	48	30	64	40	80	50	96	60	112	70	128	80	144	90	160	A0	176	B0	192	C0	208	D0	224	E0	240	F0
1	01	17	11	33	21	49	31	65	41	81	51	97	61	113	71	129	81	145	91	161	A1	177	B1	193	C1	209	D1	225	E1	241	F1
2	02	18	12	34	22	50	32	66	42	82	52	98	62	114	72	130	82	146	92	162	A2	178	B2	194	C2	210	D2	226	E2	242	F2
3	03	19	13	35	23	51	33	67	43	83	53	99	63	115	73	131	83	147	93	163	A3	179	B3	195	C3	211	D3	227	E3	243	F3
4	04	20	14	36	24	52	34	68	44	84	54	100	64	116	74	132	84	148	94	164	A4	180	B4	196	C4	212	D4	228	E4	244	F4
5	05	21	15	37	25	53	35	69	45	85	55	101	65	117	75	133	85	149	95	165	A5	181	B5	197	C5	213	D5	229	E5	245	F5
6	06	22	16	38	26	54	36	70	46	86	56	102	66	118	76	134	86	150	96	166	A6	182	B6	198	C6	214	D6	230	E6	246	F6
7	07	23	17	39	27	55	37	71	47	87	57	103	67	119	77	135	87	151	97	167	A7	183	B7	199	C7	215	D7	231	E7	247	F7
8	08	24	18	40	28	56	38	72	48	88	58	104	68	120	78	136	88	152	98	168	A8	184	B8	200	C8	216	D8	232	E8	248	F8
9	09	25	19	41	29	57	39	73	49	89	59	105	69	121	79	137	89	153	99	169	A9	185	B9	201	C9	217	D9	233	E9	249	F9
10	0A	26	1A	42	2A	58	3A	74	4A	90	5A	106	6A	122	7A	138	8A	154	9A	170	AA	186	BA	202	CA	218	DA	234	EA	250	FA
11	0B	27	1B	43	2B	59	3B	75	4B	91	5B	107	6B	123	7B	139	8B	155	9B	171	AB	187	BB	203	CB	219	DB	235	EB	251	FB
12	0C	28	1C	44	2C	60	3C	76	4C	92	5C	108	6C	124	7C	140	8C	156	9C	172	AC	188	BC	204	CC	220	DC	236	EC	252	FC
13	0D	29	1D	45	2D	61	3D	77	4D	93	5D	109	6D	125	7D	141	8D	157	9D	173	AD	189	BD	205	CD	221	DD	237	ED	253	FD
14	0E	30	1E	46	2E	62	3E	78	4E	94	5E	110	6E	126	7E	142	8E	158	9E	174	AE	190	BE	206	CE	222	DE	238	EE	254	FE
15	0F	31	1F	47	2F	63	3F	79	4F	95	5F	111	6F	127	7F	143	8F	159	9F	175	AF	191	BF	207	CF	223	DF	239	EF	255	FF

Integer	Hexadecimal	Int	Hex	Control
34	0x22	34	22	Enable Disable

Enable all events
Disable all events

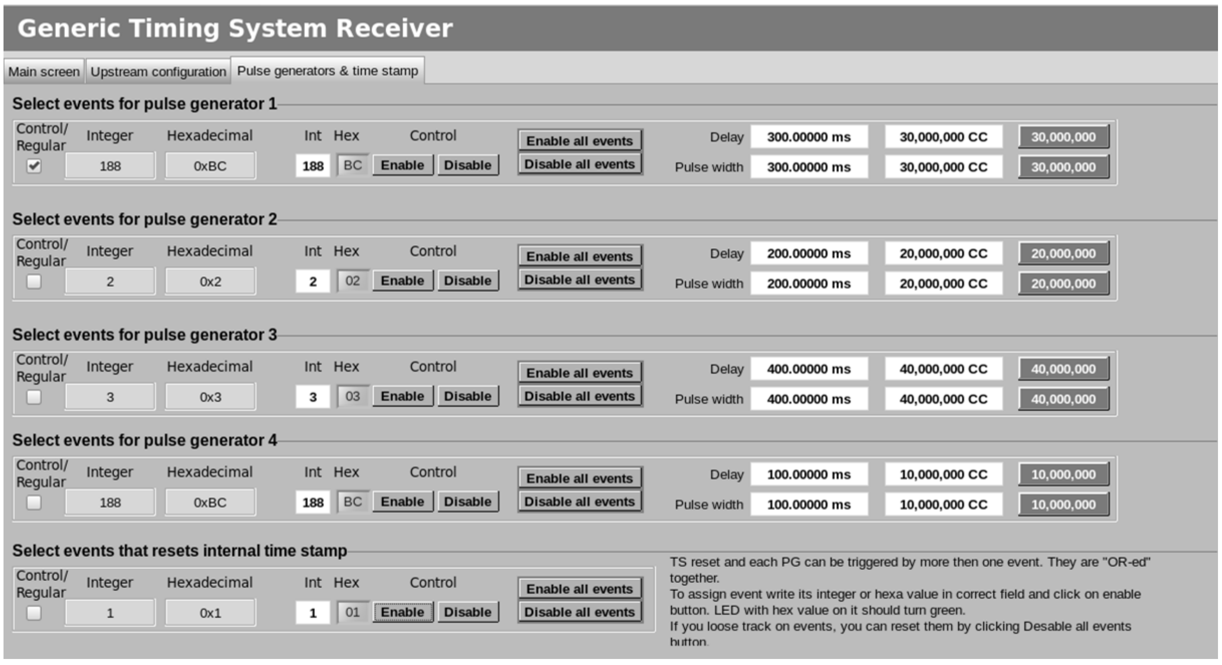
Write number of event to enable or disable upstream for this event.

Figure 4. Screen for choosing which events and control characters are forwarded from FPGA to serial line.

The developed GUI follows the modern style guidelines [6]. It is not seen from the black and white screenshots provided in this paper, but the screens mostly use different shades of gray. The colors are used only for important information like the status LEDs.

GUI reflects the shape of the underlying EPICS database. The widgets are grouped by their functionalities. Each functionality has a status widget and control widget (getter and setter design). In principle everything can be done by one widget, but in case of a transmission error, the state of the widget may not correctly reflect the state of the register in the device.

The most important feature, also the one causing most of the problems during the development, is the x-y graph that shows the timing event number in relation to


Generic Timing System Receiver

Main screen | Upstream configuration | Pulse generators & time stamp

Select events for pulse generator 1

Control/ Regular	Integer	Hexadecimal	Int	Hex	Control					
✓	188	0xBC	188	BC	Enable Disable	Enable all events / Disable all events	Delay	300.00000 ms	30,000,000 CC	30,000,000
							Pulse width	300.00000 ms	30,000,000 CC	30,000,000

Select events for pulse generator 2

Control/ Regular	Integer	Hexadecimal	Int	Hex	Control					
☐	2	0x2	2	02	Enable Disable	Enable all events / Disable all events	Delay	200.00000 ms	20,000,000 CC	20,000,000
							Pulse width	200.00000 ms	20,000,000 CC	20,000,000

Select events for pulse generator 3

Control/ Regular	Integer	Hexadecimal	Int	Hex	Control					
☐	3	0x3	3	03	Enable Disable	Enable all events / Disable all events	Delay	400.00000 ms	40,000,000 CC	40,000,000
							Pulse width	400.00000 ms	40,000,000 CC	40,000,000

Select events for pulse generator 4

Control/ Regular	Integer	Hexadecimal	Int	Hex	Control					
☐	188	0xBC	188	BC	Enable Disable	Enable all events / Disable all events	Delay	100.00000 ms	10,000,000 CC	10,000,000
							Pulse width	100.00000 ms	10,000,000 CC	10,000,000

Select events that resets internal time stamp

Control/ Regular	Integer	Hexadecimal	Int	Hex	Control	
☐	1	0x1	1	01	Enable Disable	Enable all events / Disable all events

TS reset and each PG can be triggered by more then one event. They are "OR-ed" together.
To assign event write its integer or hexa value in correct field and click on enable button. LED with hex value on it should turn green.
If you loose track on events, you can reset them by clicking Desable all events button.

Figure 5. Screen for configuration of pulse generators and timestamp counter.

the hardware timestamp. The graph automatically draws points when the event upstream is enabled. The main problem is that the graph does not show the events correctly when they come in short intervals. This is partially solved by reducing the buffer size in CSS, increasing the minimum time between redrawing and increasing the auto-scale offset. As some points are still missing from time to time, the system is not yet to be fully trusted, although the graph is intended to quickly evaluate if the right events are received and not for error detection. The graph widget includes some useful tools, such as different zoom options and annotations to find the exact values of the timestamp for a specific event. Figure 3 shows the main screen that also includes the graph.

In some cases, data manipulation is required to provide a more human readable user interface. Such operations are performed in the EPICS database whenever possible. As there are some cases when this cannot be done, a few scripts are written using JavaScript that can be easily integrated in CSS.

The “upstream configuration” tab of GUI shown in Figure 4 enables the user to choose which events are to be forwarded from FPGA to a serial interface. The LED matrix reflects the bits of the mapping RAM dedicated to the event upstream. A standard “getter” and “setter” design cannot be developed in this case, since the use of resources in CSS is too high when pushing the design in this direction. Instead, a special interface is implemented where the user writes the event number in a decimal or hexadecimal format and chooses to enable or disable the upstream of this event. The written number is stored as a local PV in CSS. A script is provided to convert the written number to string and use it as part of the name of a PV associated with the selected event.

The “Pulse generators and timestamp” tab shown in Figure 5 addresses the remaining bits of the mapping RAM which handles generation of pulses on the output pins and selects which event resets the local timestamp. A reduced design with no LED matrix is chosen to maintain the use of resources on a reasonable level.

7 Conclusion

The generic timing receiver is a debugging device for the event-driven timing systems that are widely used in particle accelerators. It has already proven to have a potential to be a valuable debugging tool, when it was used in testing the project regarding time distribution in timing system.

The presented device is still a prototype operating on a development board. There are still a few bugs to be solved as well as many possibilities to expand the project. Hope is that in the future the device will be useful for many projects in Cosylab and beyond.

References

[1] J. Serrano, “Digital signal processing using field programmable gate arrays,” pp. 29–38, 2008.

[2] A. X. Widmer and P. A. Franaszek, “A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code,” *IBM J. Res. Dev.*, vol. 27, no. 5, pp. 440–451, Sep. 1983.

[3] T. Hayes, K. S. Smith, and F. Severino, “A deterministic, gigabit serial timing, synchronization and data link for the RHIC LLRF,” 2011. [Online]. Available: http://www.bnl.gov/isd/documents/75367.pdf. [Accessed: 10-Jun-2015].

[4] C. Barrett, “Fractional/Integer-N PLL Basics,” *Texas Instruments - Technical Brief SWRA029*, 1999. [Online]. Available: http://www.ti.com/lit/an/swra029/swra029.pdf. [Accessed: 12-Jun-2015].

[5] J. Hatje, M. Clausen, C. Gerke, M. Moeller, and H. Rickens, “CONTROL SYSTEM STUDIO (CSS),” *ICALEPCS07*, no. MOPB03, Jan. 2007.

[6] A. Lüdeke, “Cognitive ergonomics of operational tools,” *J. Instrum.*, vol. 7, no. 10, pp. T10001–T10001, Oct. 2012.

Benjamin Ocepek is an undergraduate student at Faculty of Electrical Engineering in Ljubljana. He has a part time job at Cosylab d.d. as a hardware and software engineer.