Chapter 11

High-Speed Deterministic-Latency Serial IO

Raffaele Giordano, Vincenzo Izzo and
Alberto Aloisio

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/67012

Abstract

In digital systems, serial IO at speeds in the range from 1 to 20 Gbps is realized by means of dedicated transceivers, named serializer-deserializers (SerDeses). In general, due to their internal architecture, the data transfer delay, or the latency, may vary after a reset of the device. On the other hand, some applications, such as high-speed transfer protocols for analog-to-digital and digital-to-analog converters, trigger and data acquisition systems, clock distribution, synchronization and control of radio equipment need this delay to be constant at each reset. In this chapter, we focus on a serial IO architecture based on configurable transceivers embedded in field-programmable gate arrays (FPGAs). We will show how it is possible to achieve deterministic-latency operation in a line-code-independent way. As a case study, we will consider a synchronous 2.5-Gbps serial link based on an 8b10b line code.

Keywords: FPGA, serial links, line coding, latency, high-speed data transfers

1. Introduction

Deterministic-latency serial IO is highly desirable in applications such as high-speed transfer protocols for analog-to-digital and digital-to-analog converters (ADCs and DACs), trigger and data acquisition systems, clock distribution, synchronization and control of radio equipment. Unfortunately, deterministic-latency operation of serial transceivers generally requires dedicated circuitry and it is not generally supported by most of the devices on the market. Let us discuss a few examples.

Since their appearance on the market, the performance of high-speed DACs and ADCs in terms of sample rate and bit range improved continuously. This in turn generated the need for faster digital interfaces, evolving from traditional parallel single-ended buses in CMOS technology running at few hundred megabits per second to low voltage differential signalling (LVDS) operating up to 1 Gbps, also leading to higher power consumption. The CMOS and LVDS interfaces required laying out multiple traces on the printed circuit board (PCB) from the converter to the data processor and they imposed the usage of several pins on integrated circuits. In 2006, the Joint Electron Device Engineering Council (JEDEC) proposed a serial IO protocol designed for interfacing data processors to ADCs and DACs. In the latest revision of the standard (JESD204B [1]), line rates can reach up to 12.5 Gbps per serial lane and specific signals are introduced to synchronize the read out of multiple converters in the same system.

The Precision Time Protocol (PTP) is defined by the IEEE 1588 [2] standard, which defines specifications for synchronizing clocks in a networked system to a reference, precise clock (the grandmaster clock). The precision of the synchronization can be better than 1 μs, depending on the timing determinism of the network and on the asymmetries in up and down link delays. The synchronization is based on the exchange of messages between the sub-systems pertaining to the grandmaster clock and the slave clocks for estimating the clock offset and correcting it. The protocol assumes the up and down link delays to be equal, which is true only down to a certain resolution. The timing offset between the clocks grows linearly with the delay difference. Minimizing this difference, also by achieving deterministic-latency transmission at each system reset, allows the system to improve the accuracy of the synchronization [3].

One of the main goals in radio transmission systems is to keep the technological evolution of radio equipment (RE) and radio equipment control (REC) independent. Protocols, such as the Common Public Radio Interface (CPRI) [4], have been designed for this purpose. The protocol explicitly sets constraints on the latency and on the round trip delay of the data paths between RE and REC. All the delay requirements may be satisfied by means of deterministic-latency transmission between RE and REC, even for the multi-hop paths foreseen by the protocol.

Deterministic-latency links [5–8] find also application in data acquisition systems of nuclear and sub-nuclear physics experiments, specifically in the trigger sub-systems, where it is crucial to preserve the timing information associated with the transferred signals.

As we have exemplified, several application domains exist, very different from each other, which need fixed-latency data transmissions. In general, different applications adopt different protocols; therefore, in this chapter, we discuss the basic requirements for a fixed-latency link architecture and how to customize it in order to support any line coding or serial protocol. We highlight dependency of major blocks from the protocol and we discuss the aspects to be taken care of during the implementation phase. As a case study, we consider the GTP SerDes embedded in Xilinx Virtex-5 Field Programmable Gate Arrays (FPGAs), but the methodology can be easily exported to other SerDes devices.

In the following sections, we present concisely the GTP transceiver's architecture and we briefly outline the possible causes of variable latency in a serial link. We discuss the relationship between the logical alignment and the latency of the link and we present a general protocol-independent link architecture. Eventually, we show two specific implementations based on the 8b10b protocol and we highlight which features in a SerDes are keys for achieving fixed latency.

2. A configurable SerDes: the GTP transceiver

The SerDes used for discussing deterministic-latency concepts in this chapter is the configurable GTP transceiver [9] of the Xilinx Virtex 5 [10] FPGA family. GTPs are available as configurable hardware blocks. Each block hosts two transmitter (Tx) and receiver (Rx) pairs. The architecture of one pair is schematically represented in **Figure 1**. Some components, such as a phase locked loop, the dynamic reconfiguration port and the reset logic, are shared by the Tx and Rx. The GTP does not work with fixed latency in configurations based on its internal resources. The user has to develop a configuration based on an external logic controlling the alignment, which forces the SerDes to have a deterministic latency through its data path. We will now concisely present the features of the GTP essential to fixed-latency operation. The reader willing to get deeper insight might find more details in the device user's guide.

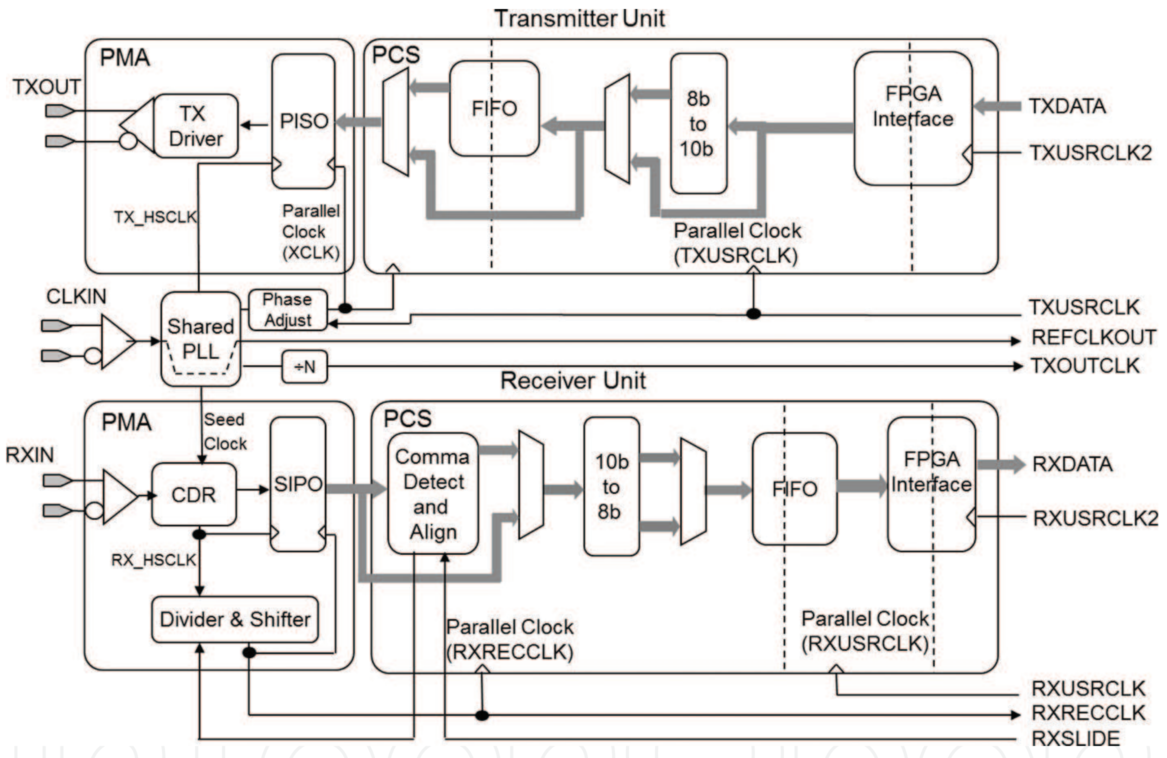

Figure 1. Simplified block diagram of the GTP transceiver. Half of the configurable hardware block is shown.

In order to transfer data to the FPGA fabric, the GTP offers a parallel IO interface which can be configured to be 8-, 10-, 16- or 20-bit wide. The lower two sizes are referred to as single-width, the higher ones as the double-width. The so-called physical medium attachment (PMA) sub-layer performs the actual data serialization and deserialization, whereas the physical coding sub-layer (PCS) processes parallel data. A reference clock (CLKIN) is routed to the shared PLL, which generates the high-speed clock for the serializer (TX_HSCLK), the parallel-side clock (XCLK) for the parallel input to serial out (PISO) block and a seed clock for the clock and data recovery circuit (CDR).

At the transmitter side, data flow from the fabric clocked by TXUSRCLK2 through the FPGA interface. When the FPGA interface is configured with a 16- or 20-bit size (double width modes), data are multiplexed 2:1 into 8- or 10-bit words and retimed on the TXUSRCLK clock, which in this case runs at double the rate of TXUSRCLK2. When the FPGA interface is configured for single-width operation, data are passed through without any processing and the two TXUSRCLK and TXUSRCLK2 coincide. A dedicated encoder can be activated for 8b10b-based protocols, while an elastic buffer (i.e. a first in first out memory) is included to cross the clock domain boundary from TXUSRCLK and XCLK reliably. In some applications, it may happen that XCLK and TXUSRCLK are derived from the same clock, therefore, they toggle at the same average frequency, with a constant phase difference. In this case, the elastic buffer can be bypassed and a dedicated circuitry is used to ensure a safe transfer of data from the TXUSRCLK clock domain to XCLK. The PISO block serializes data and outputs them synchronously with TX_HSCLK. It is worth mentioning that the PLL produces also another clock (TXOUTCLK), which can be routed to a clock buffer in the fabric and used as a TXUSRCLK. Unfortunately, due to architectural constraints of the GTP, this signal cannot be used when the elastic buffer is not in use.

At the receiver side, the CDR extracts the receiver high-speed clock (RX_HSCLK) from the stream and recovers the serial data. A dedicated prescaler divides RX_HSCLK down to generate the RXRECCLK, namely the recovered clock for clocking data out from the parallel output block and for the PCS operation. Since it is synchronous with the parallel data in the PCS, this clock can also be forwarded to the fabric and it can be used to synchronize the logic processing the deserialized data. An interesting and very useful block is the "Comma Detector and Aligner" which can search for special symbols in the serial stream and align the symbol boundary to them automatically, saving the designer to perform this operation in the fabric. The rest of the blocks in the receiver's PCS are symmetrical to the transmitter ones, they perform elastic buffering toward the RXUSRCLK clock domain and data demultiplexing when needed (FPGA interface). The RXUSRCLK2 signal synchronizes the data from the FPGA interface into the fabric. For single-width operation modes, RXUSRCLK and RXUSRCLK2 are the same signal, while for double-width modes they are edge-aligned but RXUSRCLK2 toggles at half the frequency with respect to RXUSRCLK.

3. Variation of the latency in a SerDes device

A SerDes may show latency variations related to its serial and/or parallel sub-components. In the serializer, the transmission clock that strobes the parallel data into the device is multiplied to provide the high-speed serial clock for the PISO. On the other hand, in the deserializer, the high-speed serial clock is recovered from the stream and divided back to obtain the clock for the parallel data. These clocks are used to clock, respectively, the serial and parallel side of the SIPO. However, the clock division leads to an uncertainty of the phase of the recovered clock. The phase of the recovered clock may vary in integer multiples of the unit intervals (UIs) and it causes a consequent variation of the delay of the data strobed by the clock.

Let us imagine to multiply a signal *clk* in frequency by a factor M and let us call clk_M, the result of this operation (**Figure 2**). We can label each clk_M edge with an integer number from 0 to M - 1. If T_{clk} is the clock period of clk, the *i*th edge of clk_M will be shifted by a delay $\frac{i}{M}T_{clk}$ with respect to the rising edge of *clk*. Let us now suppose to use a counter to divide clk_M in frequency back by a factor *M*, there are now *M* possible phases for the result, which are represented by the clk_i signals (with $i = 0$ to $M-1$). The obtained signal depends on which edge of the clk_M signal marks the 0 in the counter. Data crossing the clock domain from *clk* to one of the clk_i will do it with a latency related to their relative phase. After a reset or a power cycle of the system, the resulting clk_i signal might vary and the data latency with it. The system designer has to foresee a dedicated logic to remove this variation and to generate always the same clk_i signal and consequently the same data delay.

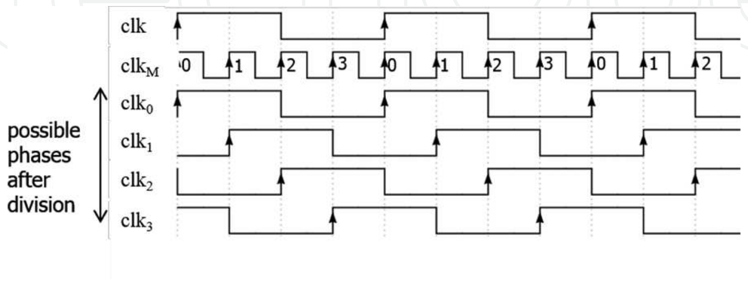

Figure 2. Clock multiplication and subsequent division (case *M*=4).

In the parallel sub-components of the SerDes, elastic buffers might induce variations of latency. Even if a buffer is written to and read from at the same frequency, after each reset or power cycle its latency depends on the difference between the internal write and read pointers. This difference in turns depends on the functionality of the logic accessing the buffer and it induces variations in terms of integer multiples of the clock period used for reading and writing. A dedicated logic has to be added in order to match the number of written words before they start being read at each reset or power up.

Therefore, the overall latency variation ΔL between two resets or power cycles of the device is as follows:

$$\Delta L = nT_{ser} + mT_{par} \tag{1}$$

where T_{ser} is the high-speed serial clock period, T_{par} is the parallel clock period, n and m are integers and their ranges depend on the SerDes device and how it is configured and operated.

4. Word alignment mechanisms

The recognition of the symbol boundary in the serial stream and the consequent alignment operation has a direct impact on the latency of a deserializer. The Xilinx GTP transceiver enables user logic implemented in the fabric to control the alignment. The RXSLIDE signal can be used to make the device shift the symbol boundary by one bit. Two modes are supported for performing this operation: a first one realized in the PMA, which shifts the recovered clock phase and combines it with the logical shifting of the data ("PMA mode") and a second one which only shifts the data logically ("PCS mode") while leaving the phase of the recovered clock unchanged. As discussed in Section 3, in order to remove latency variation, a dedicated mechanism for selecting always the same recovered clock phase must be added. Therefore, the PMA with its clock phase shifting capability allows the design to partially remove the phase variation.

Since there is not an official documentation about the internal architecture of the PMA, after some experimental tests, we have built a conceptual model of how the GTP performs the phase shift in PMA mode (**Figure 3**). The RX_HSCLK is recovered by the CDR running at half the line rate (i.e. T_{RX_HSCLK} = 2 UIs) and both its edges are used for sampling the serial stream. The serial input shift register of the SIPO is therefore double data rate (DDR) and synchronous with RX_HSCLK. A register in the SIPO synchronizes the output from the shift register on the recovered clock (RXRECCLK). The "Clock Divider and Shifter" divides RX_HSCLK down by 5 and routes it to a 5-bit shift-register, which provides five copies of the input signal, where *n*th copy is delayed by $n \cdot T_{RX\_HSCLK}$ with $n = 0$ to 4, therefore, spanning a full RXRECCLK period. The RXSLIDE signal from the user logic drives a modulo-10 counter, which in turn drives the selector of a multiplexer. According to the number of RXSLIDE pulses received, a specific phase for RXRECCLK is chosen. At the interface between the serial and parallel sections of the SIPO, this phase determines which bit of the parallel register latches the least significant bit of the shift register. In other words, changing the selection of the multiplexer modifies the alignment of the parallel data with respect to the serial data. Due to the double data rate operation, this mechanism shifts the recovered clock phase relatively to the stream in steps of 2 UIs. In order to produce a correct logical shift for any number of required bit slides, the devices use the least significant bit (lsb) of the counter to drive a barrel shifter. If the number of required slides is odd, the lsb is set and the barrel shifter shifts the data by one bit, otherwise it leaves the data unchanged (**Figure 4**). Data are therefore always correctly shifted, but for each of the five possible recovered clock phases, there are two possible alignments of the data, one for odd bit slides and one for even slides. Operation in PMA mode requires external logic to drive the RXSLIDE signal, while in PCS mode the internal comma detector and aligner can be used. We remark that, by themselves, none of the native alignment modes (PCS or PMA) operate with fixed latency. We will see in the coming sections how a smart configuration of the GTP plus a dedicated logic implemented in the fabric allows the firmware designer to achieve fixed-latency operation.

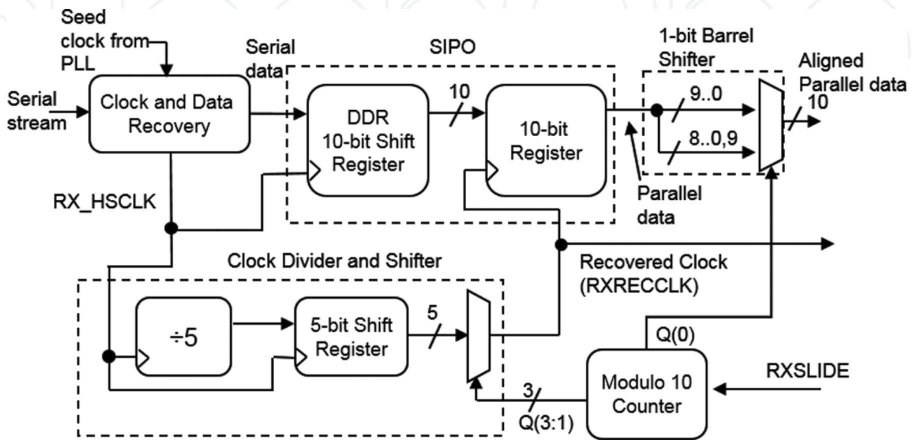

Figure 3. Conceptual block diagram of the shift architecture used in PMA mode.

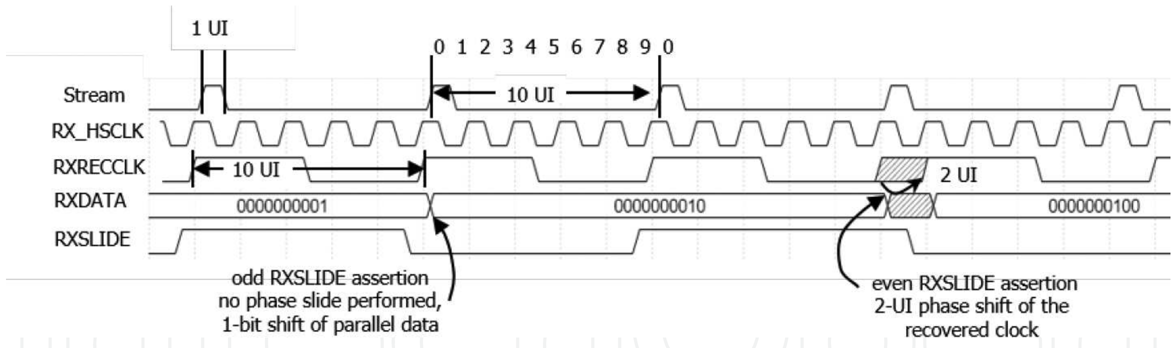

Figure 4. Recovered clock phase adjustment by means of bit slips.

5. Encoding-independent, fixed-latency operation

This section discusses key points that the designer should keep in mind in the implementation of fixed latency operation, independently of line coding and communication protocol. We are going to discuss a block diagram of the architecture shown in **Figure 5**. In order to make the discussion more practical, let us suppose the GTP runs at 2.5 Gbps with a 10-bit interface to the fabric.

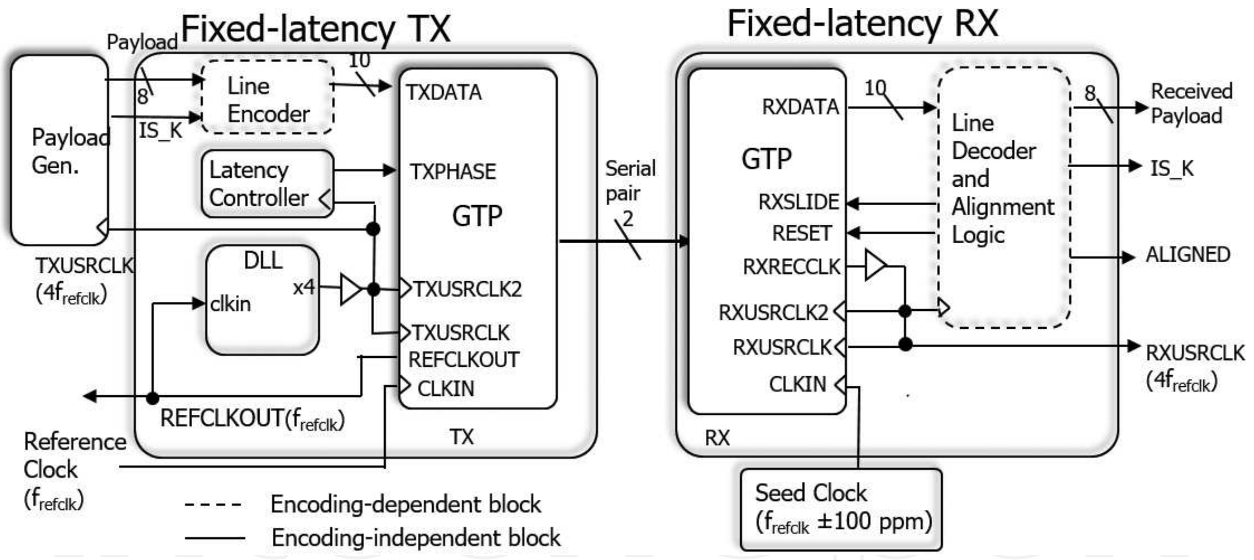

Figure 5. Customizable link architecture based on the GTP transceiver. Protocol-dependent blocks are outlined with dashed lines, protocol-independent blocks with solid lines.

Since this is single data width configuration, the GTP User's Guide prescribes the transmit clocks (TXUSRCLK and TXUSRCLK2) must be tied together and as well as the receive clocks (RXUSRCLK and RXUSRCLK2). We use a delay locked loop (DLL) for deriving the transmit clocks (toggling at 250 MHz) from the reference clock (toggling at 62.5 MHz).

Let us focus on the transmitter node. The serial coding is provided by the line encoder, which operates on data words before they enter the GTP. The latency controller adjusts the latency through the GTP transmitter to be constant at each power up or reset. The encoder is implemented by means of fabric resources in order to show how to achieve fixed latency data transfers with any coding, not only the internally supported 8b10b. A vast majority of commercial protocols allow the user to send data and control symbols. In this example, the IS_K input of the line encoder determines whether the data word will be encoded as a data or control symbol. The parallel clock for the PISO in the transmitter (XCLK) is generated by multiplying of the reference clock and then by dividing it back to obtain the desired frequency. As we discussed in Section 3, at each power up or reset, its phase can be different with respect to previous power ups or resets. The latency controller exploits a dedicated phase alignment circuit internal to the GTP, which trims the phase of XCLK to the one of the reference clock. The procedure is based on GTP features and it can be implemented by following guidelines provided in the documentation. The payload generator, not really part of the link, is explicitly included in the block diagram in order to show that data are synchronous with the transmit clock, rather than with the reference clock. It is also possible to generalize this architecture in order to transmit data synchronously with the reference clock, as we will show in Section 7. It is important to remark that, on the transmitter, there is not dependence between latency control and data encoding. There is no exchange of information between the line encoder and the latency controller, which ensures fixed latency operation by itself.

Let us now discuss the receiver node. There is a single block performing line decoding and logic alignment and it is implemented in the fabric. On the contrary of what happens for the transmitter, now the decoding and alignment are interdependent. In fact, alignment requires processing deserialized data, which in general might need decoding. The GTP embeds an 8b10b line decoder and an alignment logic which operate with variable latency. Therefore, we reimplement in the fabric the decode and alignment logic. A clock source with a frequency within 100 ppm with respect to the transmitter reference clock is needed as a seed for the correct lock-up of the CDR. As we discussed in Section 3, at each CDR lock-up, the recovered clock edge might have 10 possible phases. We configure the GTP in PMA slide mode, therefore, the alignment is controlled by asserting the RXSLIDE signal. For each possible recovered clock phase with respect to the stream, there are two possible logical alignments, one requiring an odd number of bit slides and one requiring an even number. Since we require a bi-unique relationship between the number of slides and the recovered clock phase, we reject CDR locks pertaining to one of the two possibilities, for instance, we reject those requiring odd slides. Which possibility we decide to reject is immaterial, but the alignment logic has to perform this rejection. It is important to remark that although the recovered clock shifting feature of the GTP is useful for achieving fixed-latency operation, it is not necessary. Other strategies can be implemented for SerDes devices which do not support that, as we will discuss at the end of Section 6.

Some serial protocols (e.g. SONET [11]) need to decoding for assessing the correctness of the alignment. The alignment logic checks received data according to protocol-specific criteria and if the check is failed, it changes the symbol alignment. When the check is passed, the correct alignment is found. On the other hand, a serial line code might not need data decoding for finding the correct alignment. For instance, the 8B10B code uses special bit sequences, called commas, which cannot be obtained by concatenating two symbols of the code. Finding a comma in the encoded stream allows the receiver to determine the word boundaries and performs the alignment, without the need for data decode.

For a given line code, once the correct number of bit slides needed to achieve the correct logic alignment is determined, the pertaining logic must check whether the SerDes can implement that sliding by changing the recovered clock phase.

If that is not possible, the alignment logic can force the CDR to lose the lock (for instance, by resetting the CDR or by resetting the whole SerDes) and wait for a relock.

If it is possible, the alignment logic can use appropriate features of the SerDes, such as the RXSLIDE signal for the GTP, to shift the recovered clock phase. When this procedure is complete, data are correctly aligned to the symbol boundary and the recovered clock edge has a known phase relationship with respect to the stream. This technique is referred as the *roulette* approach and it will be further discussed in the following sections.

6. An 8B10B serial link implementation

This section shows how to include, in our architecture, one of the most used coding schemes for serial data: the 8B10B encoding. At the transmitter end, the encoder has to be configured to use the 8B10B coding, according to the architecture described in Ref. [12] or in the original 8b10b patent [13]. At the receiver side, we need to include three different elements for designing the correct decoder and alignment logic: a 10b to 8b decoder, a comma detector and an aligner (**Figure 6**). At the output of the decoder, besides the 8-bit decoded data, also a flag is provided, that, when active, indicates that the received word is a control character (IS_K). The Comma Detector module looks at the deserialized data and searches for specific 10-bit symbols. When the Comma Detector finds an expected symbol, its bit offset with respect to the word boundary is sent out (on the 4-bit bus "Bit-offset") and a "Found" flag is activated. The bit shifter of the GTP is driven by the Aligner block. When the "Found" flag is asserted a number of RXSLIDE pulses corresponding to the bit-offset are generated by the Aligner block, otherwise a reset to the GTP is produced. It is worth noting that according to the 8B10B coding, some symbols can be represented with two different 10-bit words, where one word is the complement of the other. The 8B10B encoder chooses one word or its complement by minimizing the so-called "running disparity," i.e. the difference between the number of 1s and 0s sent on the serial channel. The Comma Detector (**Figure 7**) can search in the incoming data for two independent 10-bit symbols. The searched symbols are described in the hardware description language (HDL) source code as two parameters, which can be modified before the synthesis and implementation of the design. The two symbols were programmed in order to be the two different possible versions of the 8B10B coded word to be found (indicating them as Comma+ and its complement Comma-). In our design, the serial stream is sliced into 10-bit words, for this reason, part of a comma word can be in a 10-bit word and the remaining part can be in an adjacent word. Thus, the search procedure has to look into the stream and to examine each symbol and the first 9 bits from the next symbol. Following this consideration, we designed a 2-level pipeline in the Comma Detector that we used to combine each incoming 10-bit word (DATAIN bus) with the 9 adjacent bits from the next 10-bit word, so to build an overall 19-bit word (WORD bus). All the 10 possible portions of 10 adjacent bits of the 19-bit WORD bus (WORD(9+i:i) with i = 0, 1⋯9) are compared with the Comma- and Comma+ symbols, by means of an array of comparators. When the comparator finds a match with the WORD(9+i:i) segment, the comparator also asserts the corresponding iFound(i) signal. The "Binary Encoder" block collects all the iFound signals and then produces a 4-bit binary code (Bit-offset), obtained by encoding the index i for the asserted iFound(i) signal. The "Binary Encoder" block also asserts the "Found" output, when at least one iFound(i) signal is asserted. A closer look into the Aligner block reveals that it consists of a finite state machine (FSM) (**Figure 8**), which continuously checks the outputs of the Comma Detector; the Aligner logic is also made of a register and a counter (not shown for simplicity). When the FSM is in the "Idle" state, it is continuously waiting for a comma: when a comma is found, the FSM captures the data on the Bit-offset input into a special register, which keeps the data on the internal bus "iBit-offset." Then, the FSM performs the "roulette approach" algorithm, in particular,

1. When the data on the iBit-offset bus is zero, the FSM asserts the "Aligned" flag and returns back into the "Idle" state.
2. When the data on the iBit-offset bus is non-zero and odd, the machine performs a full reset of the GTP and then waits for the CDR to lock again.
3. When the data on the iBit-offset bus is non-zero and even, the FSM generates a sequence of pulses on the RXSLIDE output, where each pulse requests a bit slide to the GTP. According to the GTP specifications, each RXSLIDE pulse must stay active for one clock cycle and, between two consecutive pulses, a minimum interval of two clock cycles is required. A specific counter (Pulses bus) is used to store the amount of sent pulses. When the "Pulses bus" value reaches the same value latched on the iBit-offset bus, the production of the RXSLIDE pulses is stopped, the "Aligned" flag is activated and the FSM returns back to the "Idle" state.

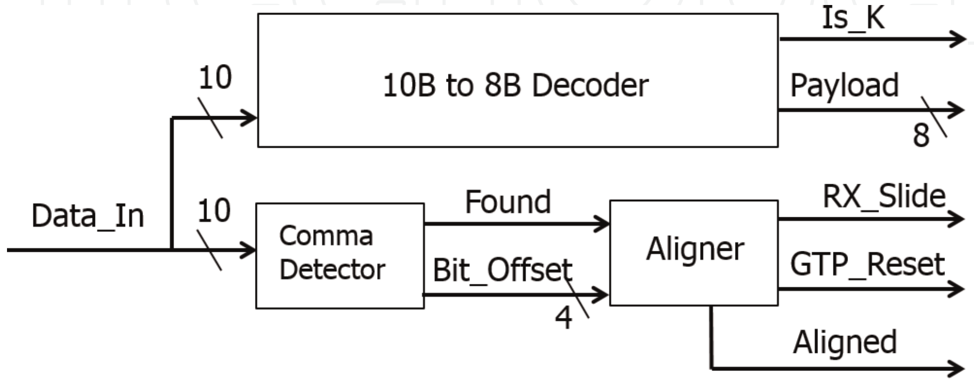

Figure 6. Internals of the line decoder and alignment logic.

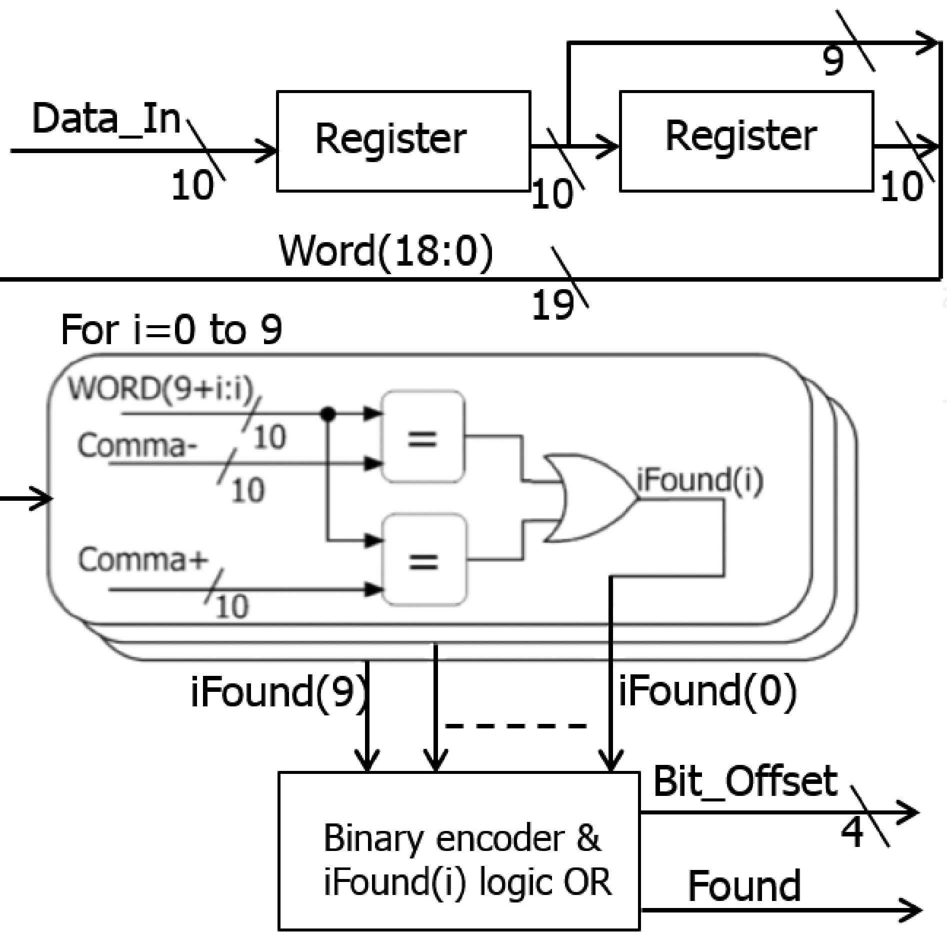

Figure 7. Simplified block diagram of the Comma Detector.

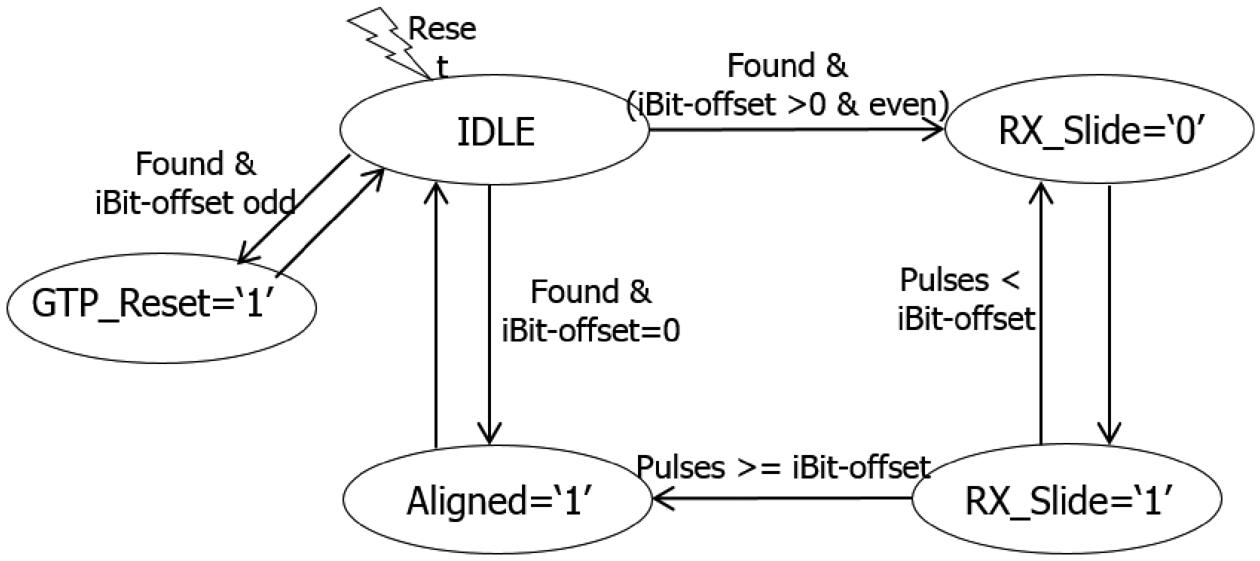

Figure 8. Simplified bubble diagram of the Aligner.

By looking at the algorithm, when there is a non-convenient value of the bit offsets (e.g. the odd ones), the CDR can be just reset, in order to wait for a relock on a most advantageous bit offset (e.g. an even one); thus, the alignment technique based on the "roulette approach" just described can be used to bypass the limitation on the GTP, that is capable only to shift by 2-UI steps. This approach brings to a lock time of the link that doubles, on average, as the lock of the CDR is rejected the 50% of the times.

When the bit offset is odd, solutions can be adopted, instead of resetting the CDR. For instance, the recovered clock phase can be shifted by 1 UI using a programmable delay (e.g. by means of a DLL, a PLL or an open-loop fine-grained programmable delay [14]). Also this method has a drawback, as it requires a higher complexity in the circuitry surrounding the GTP and it might introduce a higher clock jitter and a possible phase-skew between the alignment obtained with odd bit offset and the alignment obtained with even bit offsets, as the different delay elements are used in the two different cases.

A design based on the roulette approach can be deployed in many applications, where the deserializer architecture does not offer phase-shifting capabilities of the recovered clock. The aligner should simply monitor that the received comma has a certain bit offset (e.g. zero) and, in this case, it should perform a reset of the CDR until the required bit offset is detected. The roulette approach greatly simplifies the logic of the aligner block (**Figure 9**) and easily helps to obtain a recovered clock with a fixed-phase, without the need to perform a phase shift. As already noted, a disadvantage of the approach is the increase in the average lock time. As an example, the average lock-time is increased by a factor 10 (as the bit offset of a comma has the required value 1 time out of 10). When used in bidirectional links, an increase in the number of commas to be sent before the lock of the link is reached may help to soften this effect. For instance, the JESD204B protocol foresees an initial transmission of commas to be interrupted only after the receiver has locked and the increase of lock time would be minimal in this case. Anyway, using the roulette approach always requires a trade-off between the complexity of the aligner logic and the average lock time.

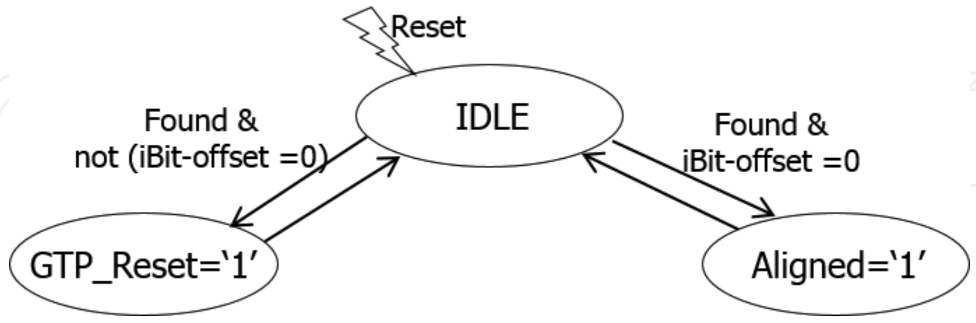

Figure 9. Simplified bubble diagram of the aligner implementing a pure roulette approach.

6.1. Frequency performance and resource occupancy

Tables 1 and **2** show the resource occupancy for the presented fixed-latency architecture, with details of resources for both the transmitter and the receiver design. For each block constituting the design, the usage of FPGA primitives is shown, separated by type; moreover, the used percentage of the design resources is shown, for a medium-size Virtex-5 FPGA.

Module\primitives	Slices	Slice Regs	LUTs	BUFG	DCM_ADV	GTP_DUAL
Latency controller	15	27	8	0	0	0
8B10B encoder	4	6	2	0	0	0
DLL	1	0	1	2	1	0
GTP	3	9	0	1	0	1
Total	23	42	11	3	1	1
Available in a V5LX50T	7200	28,800	28,800	32	12	6
% Used in a V5LX50T	0.3	0.1	0.04	9	8	17

Table 1. Resource occupation of the fixed-latency transmitter.

Module\primitives	Slices	Slice Regs	LUTs	BUFG	DCM_ADV	GTP_DUAL
Aligner	7	13	14	0	0	0
Comma detector	15	34	39	0	0	0
10B8B decoder	4	2	8	0	0	0
GTP	3	9	0	2	0	1
Total	29	58	61	2	0	1
Available in a V5LX50T	7200	28,800	28,800	32	12	6
% Used in a V5LX50T	0.4	0.2	0.2	6	0	17

Table 2. Resource occupation of the fixed-latency receiver.

A small logic foot-print (in terms of slice occupancy) is used for both the transmitter and receiver, respectively, requiring 23 and 29 slices, which correspond to 0.3% and 0.4% of a V5LX50T. On the transmitter side, the DLL block requires one digital clock manager primitive (DCM_ADV) and three clock buffers (BUFGs): one is used for driving the reference clock, one is used for the transmit clock and one for the DLL input clock. On the receiver side, only two buffers are needed (one for the reference clock and one for the receive clock) as the clock requirements are simpler.

Given such a small use of logic in the fabric, in most cases, there is no effort needed to reduce or optimize it. However, the designer should make an additional effort in order to have the system working with transmit clock and reference clock having the same frequency, so to avoid needing a DLL and its additional buffer. Such a simplified architecture lowers the occupation of fabric resources and also reduces the power consumption.

The transmitter has a maximum clock frequency (for the fabric resources) of about 370 MHz, which is essentially defined by the encoder logic, as reported by static timing analysis. The receiver has a maximum frequency of 330 MHz, in this case limited by the comma detector.

Thus, the presented architecture is able to operate up to the maximum transfer rate supported by the GTP (3.125 Gbps) and there is no need for increasing the frequency performance in the fabric.

7. Fixed-latency, packet-based transmission

In the previously described design, the data clock (which is used to transmit and receive the data) operates at 250 MHz, which is a frequency that should be easily reached when the clock is limited on a single board; however, such frequency could be too high for the propagation of the clock into a larger or more complex system, e.g. a crate or a board network. Moreover, in some applications (e.g. when using a 64b/66b encoding), an 8-bit parallel word size could be too short and might require a complex additional circuitry. In order to overcome these limitations, we show how to enlarge our architecture, so to build a packet made of several data words, but still having a fixed latency on the link. In this case, we need special care in the clock division and in the word de-multiplexing blocks, in order to obtain our objective. In this section, we describe a link with the same data-rate of 2.5 Gbps (as the previous one) but transmitting 32-bit words at 62.5 MHz, i.e. larger transmitted words. We remark that in Section 5, we presented an iso-synchronous architecture, i.e. the clock for the data source is produced by the link itself. In many applications, the designer could have a system clock which drives a data source and would prefer to use that system clock to transmit the data over the link. In these architectures, the reference clock for the PLL of the transmitter and the transmission clock for the parallel data are the same clock driving the payload. In **Figure 10**, the "fixed-latency tx" or FLT block (and the analogous block "fixed- latency rx" or FLR) represent a GTP transmitter (and the analogous GTP receiver) when opportunely configured and equipped with the logic need for implementing fixed latency operations.

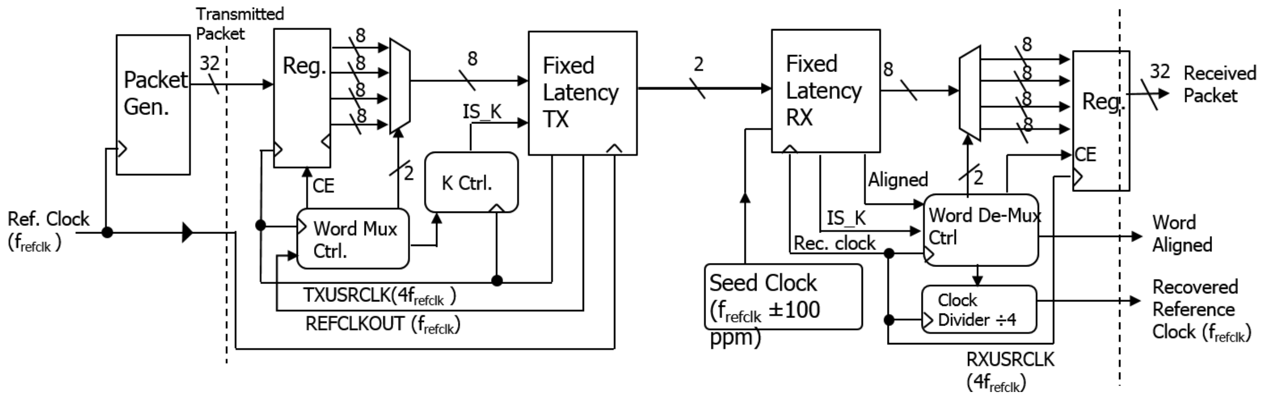

Figure 10. Synchronous implementation of our link architecture.

At the transmitter end, the input data are synchronous with the 62.5 MHz and the reference clock is latched into an input register synchronous with a 250-MHz data clock (TXUSRCLK) generated by the "Fixed Latency Tx." The clock-enable pin of the input register is driven at a 62.5 MHz rate, by the "Word Multiplexer Controller" (WMC) block. The design suggests to use a multiplexer in order to split the incoming 32-bit words into 8-bit words, which are then serialized by the FLT. The "Word Multiplexer Controller" block is also used to drive the select signals of the multiplexer, so that the same byte of the incoming 32-bit word is serialized first, whenever, the circuit is powered-on. In order to perform this task, the WMC samples the edge of the reference clock (REFCLKOUT) with the TXUSRCLK clock and sends the first byte after the edge is detected. Another circuitry is needed to tag the first byte in the word, in order to allow the receiver to correctly recognize and align the 8-bit words. In order to do this, at each every power-up, the "K Controller" module is in charge to assert the IS_K input for the FLT, which then sends a control character on the first byte of the word. Even if everything seems coherent from the logic point of view, there are still some timing issues in the transmitter part of the architecture. Indeed, the two clock signals TXUSRCLK and REFCLKOUT could be edge-misaligned, due to their different paths in the FPGA's layout, even if they are phase-locked. As an example, the "Word Multiplexer Controller" samples the REFCLKOUT signal as a data on the TXUSRCLK edge: thus, the designer needs to carefully verify that the REFCLKOUT signal meets the timing constraints needed by TXUSRCLK. In order to overcome such an alignment issue, the designer can adequately program a DLL inside the FLT, thus finding an appropriate phase of TXUSRCLK with respect to REFCLKOUT. Moreover, the WMC also has to pulse the clock enable signal of the input register with a specific phase, with respect to REFCLKOUT, in order to prevent timing violations during the capture of the input payload. It should also be noted that some of the modules around the FLT (specifically the multiplexer controller, the multiplexer itself and the input register) could be replaced by a dual-port FIFO, with a 32-bit input port and a 8-bit output port. In this case, the incoming 32-bit input words could enter the FIFO synchronously with the edge of the reference clock, while the 8-bit output words could exit the FIFO synchronously with the TXUSRCLK edge. The reader could easily argue that such dual-port FIFO could represent the solution by-design for all the described timing issues; one could also note that Xilinx FPGAs provide embedded dual-port FIFOs as hardware components, thus making the implementation of a dual-port FIFO very easy. However, the use of a FIFO has the following two main drawbacks: first, including a FIFO into the design would also increase the latency, which could not be affordable in some applications, while the architecture previously described keeps the latency as low as possible; second, the latency could not be constant at each power-up and this would require to carefully handle the read operations from the FIFO and the write operations to the FIFO (see the end of Section 3).

At the receiver end, the "Fixed Latency Rx" must be fed with a seed clock for the Clock & Data Recovery circuit (CDR) with a frequency offset below 100 ppm of the reference clock of the transmitter. At its output, the FLR drives a fixed phase 250 MHz recovered clock, to be used by the surrounding logic. The FLR module provides the 8-bit deserialized words to a de-multiplexer and drives some control signals to a specific control logic: these control signals indicate that the received character is a control character (IS_K) and that the link is byte-aligned (Aligned). The comma character is used by the "Word De-Mux Controller" (WDC) in order to handle the de-multiplexer 2-bit selection signal and to drive the clock enable for the output register (driven at a 62.5 MHz rate) with the correct latency. Furthermore, when receiving a comma, the WDC resets a clock divider (by 4) that is used to produce a 62.5 MHz recovered reference clock. The 62.5 MHz clock is then used to transfer the 32-bit payload from the de-multiplexer to the following logic. The WDC has to generate the clock divider reset in such a way to set the recovered reference clock edge in the centre of the data eye of the 32-bit payload provided by the output register. The full synchronous Tx+Rx architecture gives, as a side effect, the possibility to use at the receiver a phase-locked copy of the reference clock of the transmitter, which is a very effective and profitable feature to be used in distributed systems such as TDAQ systems of high energy physics (HEP) experiments. In TDAQ applications of HEP experiments, there is very often the need to distribute a common clock signal to all the elements of the TDAQ system, with a predictable phase and a minimum jitter. These TDAQ systems often rely on serial links, which are already deployed for data transmission. The same serial links, therefore, are a very appealing medium also for delivering the clock to every destination, without the necessity for a separate clock distribution network, thus making the TDAQ system architecture simpler to be implemented and easier to be maintained. Regarding TDAQ system, applications of fixed latency serial links, some measurements can be found in the literature [15], in particular, the measurements performed on 2.5 Gbps links show that it is possible to distribute a clock signal with a rms jitter of about 20 ps. We would like to stress that the reference clock recovered at the receiver of the described architecture cannot be easily handled to achieve a synchronous retransmission with the same GTP, but it requires to pay attention in order to make it work correctly. In fact, by looking at the hardware resources inside a GTP, the reader can easily see that the internal PLL is shared between the transmitters and receivers in the same SerDes; moreover, the PLL is already locked to the seed clock. Thus, the usage of another GTP is mandatory. Alternatively, the designer might change the phase and frequency of the reference clock in order to match phase and frequency of the recovered clock smoothly enough, so that the lock of the link is neither lost at the transmitter nor at the receiver. Furthermore, it could be necessary to filter the recovered clock in order to satisfy the jitter specifications for the GTP reference clock. Such disadvantage is not present in the newer FPGA families, such as the Virtex-6 or the seven series, as these devices are equipped with transceivers that provide separate PLLs for transmitter and for receiver.

8. Key features for fixed latency

In this section, the key features of the GTP for achieving fixed latency are described, together with helpful suggestions to be used in the porting of our results to other SerDeses.

At the transmitter end of the link, there must be a predictable phase relationship between the parallel clock (which drives the PISO) and the external parallel data clock. In many transmitters, the usage of the features for serial channel bonding can be used for satisfying this condition. In fact, channel bonding is widely employed for multi-lane serial buses, such as PCI Express [16], RapidIO [17] and Infiniband [18, 19], requiring the same latency over every bonded data path. For instance, the phase alignment circuit of the GTP, which was described and exploited to lock the latency of the transmitter, has been designed for applications related to channel bonding.

At the receiver end, there must be a predictable phase relationship of the recovered clock with respect to the byte boundary in the incoming serial stream. However, the proposed architecture could be implemented only if the receiver provides a direct method to establish the phase offset or if the receiver offers some other feature to be used to calculate the phase offset indirectly. As an example, when the receiver device can output encoded and un-aligned parallel words, the phase offset could be determined outside the receiver. Anyway, as the phase offset might change (and usually changes) at each power-up of the receiver, the designer should also add an external logic, which is able to determine the offset (by looking at the data alignment) and to reset the receiver, if the calculated offset is not the desired one: this is described as roulette approach. The external logic should not reset the device, in the case that the desired phase offset is present and thus the link achieves the lock with the same latency it had (and will have) in the other successful locks. The basic idea of the roulette approach is that no alignment of the data is explicitly performed. The receiver is forced by-design to accept only the locks achieved with data already correctly aligned and it simply rejects the locks achieved with data not correctly aligned. This approach makes the additional receive logic simpler than other designs, as there is no need for a comma detector or a word aligner. The only required module is a decoder that verifies that the received data is valid. Having a simple logic has not only benefits in saving resources, but it is desirable in applications to be used in environments with radiation [20–23], in order to lower the chance of single event upsets (SEUs) and single event latch-ups (SELs). Due to the fact that many resets could be performed at the receiver, before the desired alignment is achieved, the roulette approach has an obvious drawback in the increase of the average lock time that can be shown to be proportional to the number of bits in the parallel symbol. For this reason, when planning to use the *roulette* approach, the designer should carefully evaluate a trade-off between the average lock time and the simple receive logic. Another possibility is the use of a device that is not able to output the raw data but can automatically align and decode the data to the byte boundary and possibly can store the phase-offset between the recovered clock and the stream into an internal register. This behaviour has the same effect of externally finding the phase-offset. In the architecture described in this chapter, based on a GTP, we combined the approaches and the strategies described above. Indeed, we used an external logic in the FPGA fabric, designed to inspect the encoded data and to find the phase offset between the recovered clock and the byte boundary. But we also used the roulette approach, as we can shift the phase of the recovered clock only by even numbers of UIs. In case an odd bit shift is required, due to a specific unfortunate phase offset, the logic in the fabric provides a reset of the device and keeps waiting for a phase offset requiring an even bit shift, after achieving a new lock.

9. Conclusions

Commercially available high-speed SerDes devices are usually designed for data transfers at variable latency. This is because fixed-latency operations require dedicated circuitry and they are often not needed in most of datacom and telecom applications. However, fixed-latency serial IO is useful, or even mandatory, in various application, such as high-speed transfer protocols for analog-to-digital and digital-to-analog converters, trigger and data acquisition systems, clock distribution, synchronization and control of radio equipment.

In this chapter, we have shown how to implement fixed-latency serial IO, essentially by opportunely configuring SerDeses (in particular, the SerDes devices embedded in commercially available Xilinx FPGAs) and by adequately adding a specific control logic to such devices. The proposed architecture is able to operate with fixed latency and it is capable to recover the clock from the serial stream with a predictable phase, which does not change after a power-cycle or a reset of the link. We presented a 2.5-Gbps 8B10B link which is able to serialize 8-bit words, as a detailed example of implementation. We also described the procedure for extending the example architecture in order to transfer packets made of several data words and to synchronously transfer data with an external clock. The presented architecture is also code-independent, i.e. it can be used with any data encoding, provided a special care to the various issues described.

Acknowledgements

This work is part of the ROAL project (CINECA Grant no. RBSI14JOUV) funded by the Scientific Independence of Young Researchers (SIR) 2014 program of the Italian Ministry of Education, University and Research (MIUR).

Author details

Raffaele Giordano[1]*, Vincenzo Izzo[2] and Alberto Aloisio[1]

*Address all correspondence to: rgiordano@na.infn.it

1 Università Degli Studi Di Napoli “Federico II” and INFN Sezione Di Napoli, Napoli, Italy

2 INFN Sezione di Napoli, Napoli, Italy

References

[1] Jedec Solid State Technology Association, JEDEC Standard, “Serial Interface for Data Converters,” JESD204B.01

[2] IEEE Standard 1588, *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2008.

[3] M. Lipiński, T. Włostowski, J. Serrano and P. Alvarez, “White rabbit: a PTP application for robust sub-nanosecond synchronization,” In: *2011 International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS),* Munich, 2011, pp. 25–30. doi:10.1109/ISPCS.2011.6070148

[4] *CPRI Specification V4.1 (2009-02-18)*, 2009, pp. 21–22 [On-line]. Available: http://www.cpri.info/downloads/CPRI_v_4_1_2009-02-18.pdf

[5] R. Giordano and A. Aloisio, "Fixed-latency, multi-gigabit serial links with Xilinx FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 1, Feb. 2010, pp. 194–201. doi:10.1109/TNS.2010.2101083

[6] R. Giordano and A. Aloisio, "Protocol-independent, fixed-latency links with FPGA-embedded SerDeses," JINST,7,P05004, 2012. doi:10.1088/1748-0221/7/05/P05004

[7] J. Wang et al., "FPGA implementation of a fixed latency scheme in a signal packet router for the upgrade of ATLAS forward muon trigger electronics," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 5, Oct. 2015, pp. 2194–2201. doi:10.1109/TNS.2015.2477089

[8] R. Giordano, V. Izzo, S. Perrella and A. Aloisio, "A JESD204B-compliant architecture for remote and deterministic-latency operation," In: *2016 IEEE-NPSS Real Time Conference (RT)*, Padua, 2016, pp. 1–2. doi:10.1109/RTC.2016.7543080

[9] "Virtex-5 FPGA RocketIO GTP Transceiver User Guide," Xilinx, UG196, v1.7, 2008 [On-line]. Available: http://www.xilinx.com/support/documentation/user_guides/ug196.pdf

[10] "Virtex-5 FPGA User Guide," Xilinx, UG190, v4.3, 2008 [On-line]. Available: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[11] M. Yan, "SONET/SDH Essentials WHITE PAPER," 2008 Exar Corporation [On-line]. Available: https://www.exar.com/uploadedfiles/home/sonet-sdh-essentials_022508.pdf

[12] A.X. Widmer and P.A. Franaszek, "A DC-balanced, partitioned-block, 8B/10B transmission code," *IBM J. Res. Dev.*, vol. 27, no. 5, 1983, p. 440

[13] A.X. Widmer and P.A. Franaszek, "Byte oriented DC balanced (0,4) 8B/10B partitioned block transmission code," U.S. Patent 4 486 739, Dec. 4, 1984

[14] R. Giordano, A. Aloisio, V. Izzo et al., "High-resolution synthesizable digitally-controlled delay lines," *IEEE Trans. Nucl. Sci.*, vol. 62, no. 6, Dec. 2015, pp. 3163–3171. doi:10.1109/TNS.2015.2497539

[15] A. Aloisio, F. Cevenini, R. Giordano and V. Izzo, "Characterizing Jitter performance of multi gigabit FPGA-embedded serial transceivers," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 2, Apr. 2010, pp. 451–455

[16] PCI Express, "PCI Express Base Specification Revision," 3.0 Nov. 10, 2010 [On-line]. Available: http://composter.com.ua/documents/PCI_Express_Base_Specification_Revision_3.0.pdf

[17] RapidIO, "RapidIO Interconnect Specification," 9/2014. [On-line]. Available:http://www.rapidio.org/wp-content/uploads/2014/10/RapidIO-3.1-Specification.pdf

[18] InfiniBand Trade Association, "InfiniBand Architecture Specification Volume 1," Mar. 3, 2015 [On-line]. Available: https://cw.infinibandta.org/document/dl/7859

[19] InfiniBand Trade Association, "InfiniBand Architecture Specification Volume 2," Nov. 6, 2012 [On-line]. Available: https://cw.infinibandta.org/document/dl/7141

[20] A. Aloisio and R. Giordano, "Testing radiation tolerance of SerDeses for serial links of the SuperB experiment," In: *Proceedings of the 2011 IEEE Nuclear Science Symposium, Medical Imaging Conference*, Valencia, Oct. 23–29, 2011

[21] A. Aloisio and R. Giordano, "Testing radiation tolerance of electronics for the SuperB experiment," In: *Proceedings of the 13th ICATPP Conference on Astroparticle, Particle, Space Physics and Detectors for Physics Applications*, Como, Oct. 3–7, 2011

[22] P. Branchini et al., "Intensive irradiation study on monitored drift tubes chambers," *IEEE Trans. Nucl. Sci.*, vol. 54, no 3, Part 2, 2007, pp. 648–653

[23] P. Branchini et al., "ATLAS MDT chamber behaviour after neutron irradiation and in a high rate background," *Nucl. Instrum. Meth. A*, 581, 2007, pp. 171–174