Development of a FPGA based serial-to-parallel converter for PTSM

David Menichelli

Report of the work carried out during May and June 2002 at
Santa Cruz Institute for Particle Physics
University of California at Santa Cruz
1. INTRODUCTION

1.1 Overview

The aim of this work was to design an FPGA based serial-to-parallel converter block, to be used for the Particle Tracking Silicon Microscope (PTSM) project at Santa Cruz Institute for Particle Physics (SCIPP). The FPGA role inside the PTSM system is shown in Fig. 1.1. The FPGA controls the operation of 4 readout ASICS chips. Each readout chip is connected to \( N = 64 \) analog channels from a double side silicon microstrip detector, and its operation is regulated by two synchronized clock signals: a frame clock \( CLK1X \) and a data clock \( CLK5X \), five times faster. The readout chip checks the analog channels during every frame cycle, and assigns to each channel the low logic state if its signal is below a fixed threshold, or the high logic state in the opposite case. Subsequently the channels are divided in groups of 8, and the state of each channel is coded by a single bit in a Double Data Rate (DDR) serial data stream at the frequency of \( CLK5X \). Note that, inside a frame cycle, 10 DDR bits can be allocated at the \( CLK5X \) rate; however, the fifth \( CLK5X \) cycle (the "garbage cycle" hereafter) is a service one needed by the readout chip for internal use, and cannot be used to store and transmit information. Thus, each frame contains 8 significant bits, and the output of a readout chip consists of \( N/8 \) digital channels, operating at the \( CLK5X \) rate. The LVDS transmission protocol, a differential one, has been chosen for these channels, in order to improve the noise sensitivity. Thus, two wires will be needed for each channel. The encoding from \( N \) to \( N/8 \) channels operated by the readout chips is motivated by the necessity to reduce the number of connection wires between the chips and the FPGA. However, inside the FPGA, each channel must be monitored to detect particle hits, and a serial-to-parallel conversion must be operated prior to any further data treatment.

The FPGA serial-to-parallel converter is constituted by an array of \( N/8 \times 4 = N/2 = 32 \) identical "small converters". Three of them are shown in the figure as shaded blocks. Each "small converter" reads a DDR channel from the readout electronics, and converts it into 8 parallel channels operating at the frame rate. The 4x\( N \) = 256 parallel channels are then processed inside the FPGA to extract the relevant hits information (channel number, start and end of the hit detection) and ship it to the computer through another LVDS connection.

Fig. 1.1. Operation of the FPGA inside the PTSM system. The small shaded serial-to-parallel converters are the subject of the present work.
1.2 Specifications

Design and test of the "small converter" are the subjects of this work. Its operation is displayed with some detail in Fig. 1.2. The serial input channel \(i\) is sampled during each frame, in correspondence of the rising and falling edges of CLK5X to grab the state of the 8 significant bits. The content of the two bits inside the garbage cycle is ignored. During the next frame cycle, the acquired information is transferred to the parallel output \(o(0:7)\). The converter is designed to work at the frame frequency of 20 MHz, with LVDS input.

![Fig. 1.2. Operation of the 1 to 8 serial-to-parallel converter described in the following sections.](image)

1.3 Some design consideration

The design is intended to be implemented on a Virtex 2000E Xilinx FPGA, because it was the chip available at SCIPP at the time. Small modification to IO buffers and clock management may be needed if a powered new-generation FPGA, such as Virtex II, will be adopted at the end.

The ISE 4.2 development software by Xilinx and the ModelSim simulation tool by Modern Technology have been used to develop the FPGA program. All the schematics presented in the following have been generated by ISE 4.2, and all the signal simulations have been obtained using ModelSim. I improved the schematics graphic prior to import them inside this document, without substantial modifications.

The clock routing definition is an open problem, since the final structure has not been presently decided. In the present work it will be assumed that a reference clock is generated by a source outside the FPGA, and that the FPGA itself produces the frame and data clock from internal use and for the readout chips. However, different schemes may be adopted, without influencing the design of the converter. It is just a matter of clock managing. For instance, the frame clock might be directly generated by the personal computer interface.

I assumed all clock signals to have a 50% duty cycle. If the ASICS had a different requirement for the frame clock, some intermediate conditioning circuit will be needed between the FPGA and the readout chips.

In order to test the converter using a "real" DDR LVDS input signal, a signal source must be provided. I decided to use a part of the FPGA as a "pattern generator", and designed a FPGA based pattern generator too. The design is quite simple, as described in the next section. It allows to
generate a fast DDR serial signal, and to send it from a pair of FPGA pins (the pattern generator output) to another pair (the converter input) through a LVDS connection, by a twisted cable.
2. THE FPGA PROGRAM

2.1 Architecture overview

The block structure of the program implemented inside the FPGA is shown in Fig. 2.1, while the complete list of input and output signals is reported in Tab. 2.2. The system is composed of three parts, which are placed on three horizontal lines in fig. 2.1; from the top to the bottom they are: the clock generator, the DDR pattern generator and the serial-to-parallel converter.

In this design, an intermediate frequency clock is provided to the FPGA from outside (CLK2_5_I). This clock signal is used by the clock generator block (clkgen5) to produce all the timing signals needed by the system. The clkgen5 block generates the fast clock at readout data rate (CLK5X) and the frame clock (CLK1X). An even faster signal (CLK10X) is generated. It is not used by the serial-to-parallel converter, but is needed by the signal generator block to simulate a DDR data stream. In addition, it is helpful for the alignment of the received LVDS signal to the internal clock.

The signal generator block (funcgen2) simulates the DDR serial signal, which, in reality, will be transmitted to the FPGA from the readout electronics. Since the readout chip output will operate using LVDS logic, the signal generated by funcgen block is transmitted to the FPGA output using an LVDS driver.

This serial output SERO must be connected by a wire to the input SERLI of the receiver, as explained in Sec. 3.1. After passing through a LVDS buffer, the serial signal is processed by a synchronization block (delayblk2). Its purpose is to set the phase shift between serial signal and internal clocks in such a way that the data are stable during the rising and the falling edges of

Fig.2.1 General architecture of the FPGA program. Note the different IO buffers used in the design. The output of the pattern generator and the input of the delay block pass through LVDS buffers. All the other input and output buffers are standard Low Voltage CMOS. In the case of the input LVDS buffer it suffices to insert in the schematic the positive wire (SERLI_p) attached to the buffer symbol (IBUF_LVDS) taken from the library. The output LVDS buffer must be constructed adding a NOT gate to the library symbol OBUF_LVDS, and explicitly putting in the schematics the negative wire (SERO_n). Different graphical convention might be necessary when using a chip different from Virtex E.
CLK10X clock. The ideal phase shift between serial data and CLK10X is 90°. The synchronized serial signal is then processed by the serial-to-parallel converter and, at the end, the parallel output is produced.

Note that, in addition to the intermediate frequency clock, the data stream generated by funcgen2 and the serial output, several other signal are transmitted to the FPGA output to allow an easy bench testing, as described in Sec. 3.1. These signals are all the clock signals, the output of the delayblk block and the flag signal (LOCKED) generated by the clock generator.

I inserted a general reset line (RST_I) in the design. The main reason is that ModelSim simulation requires a complete initialization of all the state variables, to prevent the occurring of unpredictable states. However, many physical devices inside the FPGA are self-cleared at switch-on, and a general reset is not really needed by the converter. In fact, during experimental tests I kept the reset line grounded all the times.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Logic</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK2_5_I</td>
<td>intermediate freq. clock</td>
<td>(low voltage)</td>
<td>CMOS</td>
</tr>
<tr>
<td>CLK1X_O</td>
<td>frame clock</td>
<td>(low voltage)</td>
<td>CMOS</td>
</tr>
<tr>
<td>CLK5X_O</td>
<td>readout chip clock</td>
<td>(low voltage)</td>
<td>CMOS</td>
</tr>
<tr>
<td>CLK10X_O</td>
<td>DDR clock</td>
<td>(low voltage)</td>
<td>CMOS</td>
</tr>
<tr>
<td>SERO</td>
<td>DDR simulated signal</td>
<td></td>
<td>LVDS</td>
</tr>
<tr>
<td>SER_test</td>
<td>synchronized serial received signal</td>
<td>(low voltage)</td>
<td>CMOS</td>
</tr>
<tr>
<td>PAR_O</td>
<td>parallel output</td>
<td>(low voltage)</td>
<td>CMOS</td>
</tr>
<tr>
<td>LOCKED</td>
<td>flag of clock generator</td>
<td>(low voltage)</td>
<td>CMOS</td>
</tr>
<tr>
<td>RST_I</td>
<td>overall reset</td>
<td>(low voltage)</td>
<td>CMOS</td>
</tr>
<tr>
<td>SERLI</td>
<td>serial DDR input</td>
<td></td>
<td>LVDS</td>
</tr>
</tbody>
</table>

Table 1. Complete listing of the input and output signals implemented on the FPGA.

Fig. 2.2. The clock generator block clkgen5. The "clock divide" frequency property has been set to different values for the two DLL: 2 (the default value) for the lower DLL, 2.5 for the upper one. Note that many DLL outputs are not used.
2.2 Clock Generator

The clock generator is implemented by simply using two Delay Locked Loops (CLKDLL), as shown in Fig. 2.2. Each DLL is capable of doubling the frequency of the input signal, or divide it by a fixed constant (chosen in a set of possible ones). The phase alignment between input and output is provided by the DLL through the feedback path. The first DLL generates the frame clock CLK1X (by dividing the frequency of CLK2_5X by 2.5) and the readout clock CLK5X (by multiplying the frequency of CLK2_5X by 2). The second DLL generate the very fast clock CLK10X used for DDR simulation and serial input synchronization. Each DLL will need several tens of input clock cycles to achieve phase synchronization between input and output. Once the synchronization is achieved, the LOCKED flag goes up. In our scheme the second DLL is enabled to work only when the first has achieved its synchronization, thanks to the LOCKED-RST connection.

Fig. 2.3 General architecture of the signal generator funcgen2. The INIT value below each LookUp Table (LUT) defines the relationship between the inputs I0-I3 and the output. The inputs D13-D15 of the multiplexer are grounded because only 13 frame patterns out of 16 available are defined in this implementation.
2.3 Pattern generator

The overall structure of the signal generator block (funcgen2) is shown in Fig. 2.3. It is composed of three main parts: a double counter (syncount), a set of $M$ lookup tables and a multiplexer. The purpose of the signal generator is to transfer continuously a sequence of $M$ fixed frames to the SER_OUT output.

The double counter uses the frame clock $CLK1X$ and the very fast clock $CLK10X$ to count the frame ($Q1X(0:3)$) and the DDR bit position inside the frame ($Q10X(1:3)$). The frame count ranges from 0 to $(M-1)$, while the bit count ranges, obviously, from 0 to 9. In this case $M=13$, but 16 different frame can be defined using a 4-bit $Q1X$ counting. The frame number is used by the multiplexer to select the output of a particular lookup table. The bit number determines the output of the selected lookup table.

![Figure 2.4](image)

**Fig. 2.4.** (a) The double counter syncount used by the signal generator to determine the frame number and the bit position inside the frame. (b) The state machine countres included in the double counter to synchronize the fast and the slow counts.

The inner structure of the double counter is represented in Fig. 2.4 (a). It is realized simply using two four-bit counters driven by the frame clock $CLK5X$ and by the very fast clock $CLK10X$, respectively. A particular care is needed to reset the counters. Each counter must be reset at the end of its count, and this is accomplished through the two AND gates shown in the figure. In addition, the fast counter has to count "zero" each time the slow count changes. The fast counter is forced to
do so by the synchronization block \textit{COUNTRES}, whose inner structure is shown in Fig. 2.4 (b). It is a simple state machine, which generates a reset pulse for the fast counter each time the frame clock \textit{CLK1X} has been found "up" in correspondence of 5 falling edges of the very fast clock \textit{CLK10X} one following the other.

The operation of each lookup table is set by specifying a 4-digit hexadecimal number (\textit{INIT}). These numbers are showed in figure close to the LUT blocks. The hexadecimal number allows assigning an output value to any input configuration of the lookup table. See the Xilinx Software Documentation for further details.

The time evolution of the various signals is shown in Fig. 2.5 (a). In this example the serial output pattern is (0000100000) during the frame (0010), (0000010000) during the frame (1010) and so on. Note the differential behavior of the two \textit{SERO} wires, which carry a LVDS signal. The internal signal \textit{CRST} goes up during the transition of the slow count \textit{Q1X} and force the fast counter to start again its work. A detail of this operation is represented in Fig. 2.5 (b).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{diagram}
\caption{(a) The operation of the signal generator. (b) Detail of the synchronization between fast count \textit{Q10X} and slow count \textit{Q1X}. These data come from a ModelSim "behavioral simulation", and do not account for implementation and routing delays.}
\end{figure}

\section*{2.4 Serial-to-parallel conversion}

According to the general scheme of Fig. 2.1, the serial input \textit{SERLI} is processed by two blocks, \textit{delayblk2} and \textit{serpar}, before the parallel output \textit{PAR_O(0:7)} is generated. The first block just adds a time shift to the input serial signal, while the second block is the "real" serial-to-parallel converter.

The \textit{delayblk2} is needed because, in general, the serial input may be completely out of phase with respect to the FPGA inner clocks. The \textit{delayblk2} must achieve two goals:

1) Bit synchronization with respect to the fast clock \textit{CLK5X}. \textit{SERIN}, the serial input of \textit{serpar} is sampled during the falling and the rising edges of \textit{CLK5X}. Thus the serial input must be stable during these transitions. In other words, a phase lag of about 90° must exist between \textit{CLK5X} and the \textit{SERIN}.

2) Synchronization between the serial data stream and the frame clock. The serial input is consists of a continuous stream of DDR frames. Each frame contains 8 significant bits. The information of bit no. 0 inside each word must be stable during the first rising edge of \textit{CLK5X} after the start of the
frame, i.e. after the rising edge of $CLK1X$. As it can be seen in fig. 2.7, the output of the input LVDS buffer ($SERSTR$) cannot be used as the input of the $serpar$ block, because bit no. 0 would be recognized as the bit no. 1, the bit no. 9 as the bit no. 0 and so on. Thus $SERSTR$ must be properly delayed.

2.4.1 Signals alignment

The scheme of the alignment block $delayblk2$ is depicted in fig. 2.6. The serial input is simply delayed using two shift registers. This arrangement is completely equivalent to 7 FFD driven by the falling edge of $CLK10X$. I used two shift registers instead of many FFD just for the sake of simplicity. The data are shifted from one cell to the other during the falling edge of the very fast clock $CLK10X$. In this case the input signal delay $\Delta t$ is

$$\Delta t = (6 + stg)T_{10X}$$

where $T_{10X}$ is the period of $CLK10X$, and $stg$ ranges from 0 to 1, depending on the time lag between the edges of the incoming data and of $CLK10X$. This particular structure is adequate for the test bench setup described in Sec. 3.1. A simulation of signal timing including the effect of the delay block is shown in Fig. 2.7. As it can be seen, the output of $delayblk2$, $SER\_OUT$, is a replica of its input, $SERSTR$, properly delayed. Note that the bit no. 0 of $SERIN$ is active during the first rising edge of $CLK5X$ after the beginning of the frame.

![Fig. 2.6. The structure of the block delayblk2, needed to synchronize the input serial stream to the inner clock. This special implementation works in the case of our test setup. A different number of shift registers of FFD may be needed in other situations, as explained in the text.](image)

![Fig. 2.7. Alignment between the input serial stream and the inner clocks, as operated by the delayblk2 block. The data are obtained from a “post place & route” ModelSim simulation, accounting for all the delays due to the logic implementation and routing. As it can be seen, $SER\_OUT$ is just a delayed replica of $SERSTR$. The vertical arrows indicate the times at which the $serpar$ block expects to find the bit no. 0. The splitting of the serial signal $SERSTR$ and $SER\_OUT$ into single bits is indicated by the small vertical segmentation. The bit numbers inside the frames are also indicated.](image)
A different delay may be needed if the FPGA driver were changed, or if the serial input were taken directly from the readout electronics. In its most general form, the alignment block can be constructed using a chain of D flip flops, which change their state during the rising or the falling edge of \( \text{CLK10X} \). The finest delay adjustment is \( T_{10X}/2 \), a value adequate to always permit the correct synchronization of the serial input. Such a small delay is obtained by putting a negative-edge driven FFD after a positive-edge driven FFD, or vice versa. Differently, two FFD of the same kind, one following the other, determine a delay of \( T_{10X} \).

### 2.4.2 Serial-to-parallel conversion

The overall architecture of the serial-to-parallel converter (the \textit{serpar} block) is shown in Fig. 2.8. The converter is composed of two Serial Input - Parallel Output (SIPO) memories and of a 7-bits D flip flop. One of the memories (\textit{sipo2p}) is driven by the rising edge of \( \text{CLK5X} \), and the other one (\textit{sipo2n}) by the falling edge. \textit{Sipo2p} grabs the data bits nos. 0, 2, 4, 6, while \textit{sipo2n} grabs the data bits nos. 1, 3, 5, 7 of each frame. All these data are transferred to the parallel output \textit{PAROUT(0:7)} at the beginning of the next frame by the 7-bits D flip flop.

![Fig. 2.8. The overall architecture of the serial-to-parallel converter.](image)

The inner structure of the block \textit{sipo2p} is shown in Fig.2.9. The four horizontal flip flops constitute a shift register, which stores the bits no. 0, 2, 4 and 6 of each frame. The four data bits stored in the shift register are transferred to the output of the block during the rising edge of \( \text{CLKENp} \). \( \text{CLKENp} \) is generated by the \textit{clocken2p} block to stop the shift register operation during the garbage cycle. The \textit{clocken2} block is the simple state machine depicted in Fig. 2.9 (b). The architecture of \textit{sipo2n} is identical, but the four D flip-flops are driven during the falling edge of \( \text{CLK5X} \). The clock enable generator (\textit{clocken}), is slightly different in the case of \textit{sipo2n}, too.
Fig. 2.9. (a) The inner structure of the *SIP* O shift register. (b) The *clocken*2 block, which inhibits the operation of the shift register during the garbage cycle.

Fig. 2.10. Operation of the *serpar* block. Vertical arrows indicate the relevant data transitions. This data are taken from a behavioral ModelSim simulation, thus delays generated by implementation and routing are not accounted for. Note that, in this case, the phase lag between the serial input *SERIN* and the clock signals retains its ideal value of 90°. In practice, even a smaller phase lag may be effective.
The signal timing is shown in Fig. 2.10. It is illustrated here how the serial data of the $n^{\text{th}}$ frame are converted to parallel. The data bits of the $(n-1)^{\text{th}}$ frame are schematically indicated as (xxxxxxxx). In this example the data bit of the $n^{\text{th}}$ frame are (00001000). Inside the $n^{\text{th}}$ frame, the even bits 0, 2, 4, 6 are grabbed during the rising edge of the $CLK5X$ clock, while the odd bits 1, 3, 5, 7 are grabbed during the falling edge. This process is indicated in Fig. 2.10 by the vertical arrows. Note that the $CLKEN$ signals go up to prevent the grabbing of the bits 8 and 9 inside the garbage cycle, which carry no information. This inhibition is represented in the graph by two stars. During the rising edge of $CLKENp$ the even bits are transferred to the parallel output $D$ of the $SIPOp$ memory. At this point the odd bits of $D$ are still the one of the old $(n-1)^{\text{th}}$ frame: $D$ reads (0x0x0x0x). During the rising edge of $CLKENn$ the odd bits are transferred to the output of $SIPOn$ memory. At this point the 8 channels of $D(0:7)$ contain the information of $n^{\text{th}}$ frame in parallel form. Finally, during the next rising edge of the frame clock $CLK1X$ the whole result of the parallel conversion is transferred to the output of the $serpar$ block. Thus, the serial information of the $n^{\text{th}}$ frame is available to the parallel output during the $(n+1)^{\text{th}}$ frame.
3 EXPERIMENTAL TESTS

3.1 Experimental setup

The setup adopted to test the serial-to-parallel converter is depicted in Fig. 3.1. The FPGA is a Virtex XCV2000E FPGA produced by Xilinx, a 560 pins chip. The FPGA is allocated on its prototype board (AFX-BG560-100), which allows an easy connection with external integrated circuits (such as the oscillator indicated in the figure), bias and download circuitry. In addition, an easy access to all the 560 chip pins is provided through the four arrays of connectors surrounding the chip socket. Note that a FPGA with different footprint would need a different prototype board.

Fig. 3.1. Experimental setup used for the self-test of the serial-to-parallel converter. Two oscilloscope standard probe and a differential one are shown in the figure as an example; however, this oscilloscope allows monitoring 4 channels at a time. The LVDS connection adopts a twisted cable with characteristic impedance in the range 50-100 Ω. Four arrays of connections surround the FPGA socket, providing access to the 560 pins of the chip. The position of instrumentation connections on the four arrays are not necessarily the real one, which depends on the pin assignment defined in the constraint file.

The FPGA program is downloaded from the PC through the parallel port. The connection between the parallel port and the prototype board is established by the Xilinx Parallel Cable IV. The whole download process is assisted by development software ISE 4.2.

An external 50 MHz oscillator (SG531PCVC) provides the intermediate frequency $CLK2_{-5X}$. The frame and data frequency are, consequently, 20 MHz and 100 MHz, respectively. Note that both square-wave and sinusoidal oscillators can be used; in the latter case, the FPGA will detect the input threshold, then reconstructing a 50% duty cycle square wave.

The serial signal, which feeds the converter, is generated by the FPGA itself. The serial output and input are connected according to the LVDS standard, through a twisted cable. Proper terminations must be used at transmitter (serial output) and receiver (serial input) pins. A 100 Ω termination between receiver pins is strictly required by LVDS protocol. A further termination, specified by Xilinx documentation, would slightly improve the system performance. Note that a digital pattern generator is included in the logic analyzer, but it was not used for advanced testing. The reason is that its highest operation frequency is 50 MHz; this means that if a DDR signal is...
generated, its frequency can be no more that 25 MHz. In addition the pattern generator allows only TTL and ECL operation, but not LVDS.

A 500 MHz digital oscilloscope (TSD650A) and a HP logic analyzer are used to monitor the FPGA activity. Note that the shorter sampling time of the analyzer is 4 ns; while the $CLK5X$ period is $T_{5X}=10$ ns. It follows that the analyzer can be used only to monitor the general evolution of the 25 MHz parallel output (but not its transitions). The oscilloscope can be used to check all the relevant signals, even the LVDS serial flow by using the 400 MHz differential probe (Tektronix P6246). The drawback of the oscilloscope is obviously that it allows monitoring only 4 channels at a time.

![Fig. 3.2. Operation of the FPGA as measured using the oscilloscope. The various frames have been numbered to make easy the discussion in the text. The traces 1 to 4 correspond to the signals named as $CLK1X$, PAR_O(7), SER_test and SERLI in Fig. 2.1. The latter has been acquired using the differential probe. The horizontal scale is 100 ns/div.](image)

### 3.2 Results

An overview of the converter operation is reported in Fig. 3.2. The traces shown in the plot correspond to the signals named $CLK1X_O$ (trace 1), $PAR_O(7)$ (trace 2), $SER_test$ (trace 3) and $SERLI$ (trace 4) in Fig. 2.1. Thus trace 4 represents the LVDS input of the FPGA. Trance 3 is the input of the block $serpar$, trace 1 is the frame clock and trace 2 the 8th bit of the parallel output. $SER_test$ is just a delayed replica of $SERLIN$, as discussed in Sec. 2. Traces 1, 2 and 3 represent internal signals as delivered to the output by after same delay, determined by a single Low Voltage CMOS buffer. Thus it makes sense to compare the timing between these three traces.

The serial test signal consists of continuously repeating several frames, numbered from 1 to 11 in the figure. During the first 8 frames the data bits go up one after the other. In the 9th frame the two bits of garbage cycle go up together. During the 10th frame all the data bits are down, while during the 11th all the bits go up. Finally, several frames containing only zeros follow. This particular serial data stream proved to be particularly useful for testing purposes. Note that $PAR_O(7)$ go up during the 9th and 12th frames, because the serial bit no.7 is up during the frames 8 and 11. Usually, all the parallel output are monitored together by using the logic analyzer which, unfortunately, does not allow to export the on-screen image. When using the serial stream of trace
3, a cascade of pulses is observed on the analyzer screen if the various parallel channels are displayed one below the other.

The detailed timing of the various signals is shown in Fig. 3.3. Traces 1 to 4 correspond to the signals named $CLK1X_O$, $CLK5X_O$, $SER\_test$ and $SERLI$ in Fig. 2.1. Note that, during the $11^{th}$ frame, bit no. 0 of the serial stream goes up some ns before the rising edge of $CLK5X$. Traces 1 to 3, acquired with the standard oscilloscope probes (signal+ground reference) are not well "squared," while trace 4, acquired by the differential probe, exhibits steep fronts. This is probably due to the frequency limit of the standard probes and to the poor grounding of the prototype board. In addition the signal-to-noise ratio of the differential signal is clearly higher improved.

![Fig. 3.3. Detailed timing of serial input and internal clock, as measured with the oscilloscope. Traces 1 to 4 correspond to the signals named $CLK1X_O$, $CLK5X_O$, $SER\_test$ and $SERLI$ in Fig. 2.1. The frames have been numbered in the same order as in Fig. 3.2. The horizontal scale is 25 ns/div. Trace 4 has been acquired using a differential probe.](image)
4. CONCLUSIONS

I designed the implementation of a serial-to-parallel converter inside a Xilinx Virtex E 2000 FPGA. The converter can receive a serial signal according to the requirement of the PTSM system (Double Data Rate, LVDS protocol, fifth garbage cycle). The serial stream is then converted into 8 parallel channels for FPGA internal use. Just few tens of FFs are required to implement the converter. This is not a secondary issue, because a 1-to-8 converter will be allocated for every serial channel, up to a total amount of 32 converters.

A high frequency DDR signal is needed to test the converter, and it should be transmitted to the FPGA through a LVDS connection, if the operative conditions of the FPGA inside the PTSM system must be reproduced. To generate such a signal, I decided to use another portion of the same FPGA, designing a programmable pattern generator.

The overall system proved to work up to a frame frequency of 20 MHz. It should be possible to increase further the frame rate without great difficulties. The converter frequency limit is dictated by the alignment block, which requires a $CLK_{10X}$ signal. Thus, the frame frequency must be 10 times smaller than the frequency limit of the FPGA. Probably it is possible to design a more sophisticated alignment block, but the other logic circuitry implemented in the FPGA beside the converter might impose more severe limitation to the frequency enhancement.

REFERENCES

Detailed information, continuously updated, about PTSM project can be found in the SCIPP database. The whole documentation about Xilinx FPGAs, development software and libraries can be freely downloaded from the company website at www.Xilinx.com. In addition, many programming examples are provided in the website as Application Notes. However, I found that the quality of many application notes is not as high as that of the documentation.