Synchronous Latency Insensitive Design

In FPGA

Master thesis performed in Electronic Devices

By

Cheng Sheng

Report number:LITH-ISY-EX--3675--2005
Linköping University 2005
Synchronous Latency Insensitive Design

In FPGA

Master thesis

Electronic Devices at

Linköping Institute of Technology

by

Cheng Sheng

LiTH-ISY-EX-3675-2005

Supervisor: Christer Svensson
Examiner: Christer Svensson
Linköping, Feb.11, 2005
<table>
<thead>
<tr>
<th>Language</th>
<th>Report category</th>
<th>ISBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Svenska/Swedish</td>
<td>Licentiatavhandling</td>
<td>LITH-ISY-EX-3675-2005</td>
</tr>
<tr>
<td>X Engelska/English</td>
<td>Examensarbete</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C-uppsats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D-uppsats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Övrig rapport</td>
<td></td>
</tr>
</tbody>
</table>

**URL för elektronisk version**
http://www.ep.liu.se/exjobb/isy/2005/3675/

**Titel**  
Synchronous Latency Insensitive Design in FPGA

**Författare**  
Cheng Sheng

**Sammanfattning**

A design methodology to mitigate timing problems due to long wire delays is proposed. The timing problems are taking care of at architecture level instead of layout level in this design method so that no change is needed when the whole design goes to backend design. Hence design iterations are avoided by using this design methodology. The proposed design method is based on STARI architecture, and a novel initialization mechanism is proposed in this paper. Low frequency global clock is used to synchronize the communication and PLLs are used to provide high frequency working clocks. The feasibility of new design methodology is proved on FPGA test board and the implementation details are also described in this paper. Only standard library cells are used in this design method and no change is made to the traditional design flow. The new design methodology is expected to reduce the timing closure effort in high frequency and complex digital design in deep submicron technologies.

**Nyckelord**

GALS, STARI, FIFO approach, Initialization.
Abstract

A design methodology to mitigate timing problems due to long wire delays is proposed. The timing problems are taking care of at architecture level instead of layout level in this design method so that no change is needed when the whole design goes to backend design. Hence design iterations are avoided by using this design methodology. The proposed design method is based on STARI architecture, and a novel initialization mechanism is proposed in this paper. Low frequency global clock is used to synchronize the communication and PLLs are used to provide high frequency working clocks. The feasibility of new design methodology is proved on FPGA test board and the implementation details are also described in this paper. Only standard library cells are used in this design method and no change is made to the traditional design flow. The new design methodology is expected to reduce the timing closure effort in high frequency and complex digital design in deep submicron technologies.
Acknowledgements

First I would like to give my thanks to my supervisor, Professor Christer Svensson, for giving me this opportunity and his valuable guidance.

I would also like to thank Jonas Nilsson from Hardi Electronics AB. His helpful advices and useful discussions helped me a lot and saved much time for me.

Last but not least, I would like to express my appreciation to my parents and my girlfriend for their encouragement and support to my study here.
1. Introduction

Nowadays the working frequency, complexity and density of integrated systems increase rapidly and deep sub-micron process technologies are widely used. With the advent of new technologies, the wire delay plays a much more important role in logic synthesis and optimization. Recent research on interconnect optimization methods can help to reduce interconnect delays, but it can not turn the trend. Since delay of long wires become larger relative to gate delays, the estimation of wire delays become increasingly sensitive to layout. Thus the timing problems caused by these unwanted wire delays can only be solved at layout level and it will cost a lot. Many methods to mitigate these problems are proposed, but most of them more or less have their special design flow or nonstandard library cells. However, a nonstandard design flow is not easily accepted by industry.

The goal of our research is to find a methodology that can handle the unpredictable wire delays meanwhile minimize the influence on existing design flows so that it can be easily accepted. In this paper a scheme to mitigate timing problems is proposed. The wire delays are taking care of at the architecture level instead of layout level, and the functionality of the model including wire delays is guaranteed all the way to the layout level. In this scheme, the designer can complete his clock-true verification before synthesis, and only use existing tools, design flow and standard library cells all the way to layout.

In section 2 related works are described and in section 3 the methodology is presented. Implementation details are shown in
section 5 and testing results and analysis are in section 6. The paper is finished by a discussion and a conclusion in section 7 and 8.

2. Related Work

Our methodology is based on GALS (Globally Asynchronous, Locally Synchronous) architecture. The basic idea of GALS is to partition a large design into isochronous blocks with limited size so that in each block the wire delay is small enough and each block can be considered as synchronous. Between these blocks synchronization methodologies are used to eliminate timing and phase errors during communication.

The essence part of GALS architecture is the synchronization methodologies. There are lots of researches focusing on it and many approaches are proposed. One approach is proposed by Luca Carloni and Alberto Sangiovanni-Vincentelli in [3]. The basic idea of their approach is to modify a design that works under the assumption of zero-delay connections between blocks by encapsulating them with suitable wrappers and connecting them through internally pipelined blocks complying with a formally defined protocol that guarantees identity of behavior. The wrappers use flags like “void”, “stop” to decide which data is needed. This method is useful if each block is "patient", that is its function only depends on the order of event reaching it. The functionality of a system built by this method is independent of the latency of the communication channels between blocks. However, this methodology guarantees functionality but not performance. The performance of the system is unknown until after backend.
Alternative implementations of the physical layer of the Latency Insensitive Design (LID), based on FIFO's, were proposed in [1]. These FIFO's perform synchronization between multiple clock domains in a similar way as some earlier proposed synchronizers based on parallel latches. It expands the LID from single time domain into multiple time domains by using several new mixed-timing relay stations.

Another approach, based on STARI (Self-Timed at Receiver's Input), is proposed by Mark R. Greenstreet[4]. It uses a global clock to establish the rate of data-transmission and uses a self-timed FIFO at the receiver side to compensate for clock skew and variations in the delay along the data path. The FIFO is initialized to be roughly half-full, and during one clock cycle of global clock, one value is inserted into the FIFO and one is removed by the receiver. This ensures the FIFO never overflows or underflows, and the FIFO does not need any flow control signals so that it avoids the latency required for reliable synchronization of flow control signals. However, it will take some time for the data to propagate through the self-timed FIFO and how to initialize the FIFO is also a problem. In [2] the concept of STARI was extended to multiple clock frequencies. A single latch is used as one stage FIFO and clocked by a clock generated from the transmitter and receiver clocks.

In [7] a new method of synchronization is described. Phase synchronization is accomplished by using a split FIFO as phase synchronizer. Reset arbiter/synchronizer and a digital delay locked loop, DLL, which centers the data clock midway in the data bit, are used to provide proper initialization of the split FIFO synchronizer. This method may work well at high-speed communication by using split FIFO, but it also introduces extra considerations for mixed signal design by using DLL in the
3. Methodology

The main goal of this paper is to explore a design method that safely mitigates the problem of timing closure. It is based on STARI and is achieved as follows. At the beginning of high-level system design, the whole circuit is divided into several function blocks with limited size. These function blocks are small enough so that there will be no timing problems when they are treated as separate designs at actual working clock frequency and typically are able to perform basic functions. Then the maximum delay for the longest data path is estimated. Also, the inter-block clock skew is estimated. With these estimations a value of \( n \) which can decide how much pipelining is needed in the link can be calculated. In some cases a lower value of \( n \) may be chosen if it can be foreseen that the floorplan allows a physically shorter link. However, if the calculated value is used in all data paths, no floorplan decisions are required until back-end design flow starts. Next, in the clock-cycle true model, the selected value for each data link is implemented as an \( n \) clock-cycle long delay unit. This clock-cycle true model is used all the way to the back-end design and for clock-cycle true verification. Though all the data link delays are predefined, the functionality of the whole design is guaranteed and no change is needed in the back-end design. Thus, those design and verification iterations due to implementation problems are avoided.

During the following synthesis process, those \( n \)-cycle delays are replaced by two-port memory synchronizers. These memory synchronizers are generated with flip-flops from standard cell
libraries so that no special fabrication technology is needed. These cells are placed in the receiving blocks to avoid synchronization problems between these synchronizers and function blocks.

4. Reset Algorithm

As mentioned before, in STARI system, how to initialize the FIFO is the essential aspect of the system. It is also the main goal of this paper. In this thesis, an active-low global reset signal is used to initialize the whole system and a low frequency global clock is used to synchronize this global reset signal. The frequency of this global clock should be low enough so that the whole system seems synchronous under this clock even taking in account transfer delays. The global reset signal is synchronized by the low frequency clock when it enters the chip and then distributed to each block along with the global clock. Assume the clock period of this global clock is T; the skew between global reset signal and the global clock signal should be less than T/2 so that no timing problem occurs. When the synchronized global reset signal and the global low frequency clock reach the sub-block, the global clock synchronizes the reset signal again with opposite clock edge. Then the reset signal is synchronized for the third time by the actual working frequency. The advantage of this algorithm can be shown through the waveform below:
In the waveform above, “Global clock TX” and “Global Clock Rx” presents the global clock signals which routes to transmitter and receiver, respectively. “Global Reset” is the global reset signal and the “Global Reset1” is the reset signal after the first synchronization with the global clock. “Greset1 Tx” and “Greset1 Rx” describe what the signals look like when the “Global Reset 1” arrives at Transmitter and Receiver.

In the waveform, it takes $t_1$ for the global clock to reach transmitter and $t_2$ to reach the receiver. When the global reset is released and the falling edge of global clock detects the releasing, the signal “Global Reset1” is then released. Thus the global reset signal is synchronized by the falling edge of global clock. After this synchronization, the global clock and the “global reset1” signal is distributed to the transmitter and receiver. Assume the clock period of global clock is $T$ and it will take $t_3$ and $t_4$ for the reset signal to reach the transmitter and receiver. When the reset
signal arrives at transmitter and receiver as “Greset1 Tx” and “Greset1 Rx”, we can see from the figure that as long as $|t_3 - t_1|$ is smaller than half the period of global clock, the releasing of “Greset1 Tx” will be always detected by the following rising edge of “Global clock Tx”. It is also the same in the receiver side. It is obvious that no matter which side, transmitter or receiver, the reset signal arrives first; the skew between reset signals on both sides after the second synchronization is only depends on the global clock skew. The skew between reset signals is then absorbed by this algorithm. Thus, the total relative delay is $(T_{data\_delay} - T_{clk\_skew})$, and the offset value can be calculated by formula below:

$$n = \frac{(T_{data\_delay} - T_{clk\_skew})}{T_{period}};$$

Where $T_{data\_delay}$ is the data link delay, $T_{clk\_skew}$ is the clock skew between receiver and transmitter. It is positive when the clock arrives at transmitter first but the value will be negative if the clock arrives at receiver first. $T_{period}$ is the period of global clock.

5. Implementation

The work in this paper is implemented on HARDI FPGA Test Board FPGA2X3. A daughter board TEST1x1 is also used to aid implementation. The Testing board contains a Xilinx Virtex2 XCV6000 -5 FPGA chip. The package is FF 1517. The synthesis tool used in this work is XILINX ISE 5.1.

The whole design contains 3 main parts: Transmitter, Receiver and FIFO. A clock of max frequency 25MHz is used as the global clock and a signal controlled by a pushbutton on the daughter board is used as the global reset signal to initialize the whole system. The actual working frequency is 8 times the global clock.
frequency. The system also contains some small function blocks for debugging such as clock dividers, decoders and so on. The global clock is provided by a clock generator and the data will be shown on a LCD display screen. Figure 2 shows what the whole system looks like.

Figure 2 Overview of Whole System

Figure 3 below shows the placements of 3 main parts and how the reset algorithm is achieved.

When the reset signal is released, the transmitter will send out 4 8-bit words which it gets from external devices during reset period to the channel, and then the FIFO gets these 4 words from the channel and sends them to the receiver. The receiver will
store these 4 words in its memory. The values of receiver's memory can be shown on the LCD of the daughter board to examine the functionality of the system.

In the transmitter, the low frequency global clock is used to synchronize the reset signal, and a PLL is used to provide actual working local clock with the frequency 8 times the global clock. During the reset period, the transmitter will get 4 words from the external device and store them in a buffer. At the second rising edge of global clock after releasing the reset signal, the transmitter will start working. The transmitter will wait for 1 local clock cycle after starting, then send out these 4 words in the buffer sequentially to the receiver and stop. A strobe is also sent out from transmitter to the receiver along with the data. The strobe will start when the data transfer begins and stop when the last word is sent out. This strobe is used as the writing clock for FIFO at the receiver.
The receiver contains a 4-word data buffer. It can get an offset value from external devices during reset period. This offset value indicates how many clock cycles the receiver should wait before it gets data from the FIFO. In the receiver, the low frequency global clock is used to synchronize the reset signal, and another PLL is used to provide actual local clock with the frequency 8 times the frequency of global clock. The receiver will get the offset value from external device during reset period. When the reset signal is released, the receiver will start at the second rising edge of global clock. Then the receiver will wait for n local clock-cycle according to the offset value it receives, and then activate the read pointer of the FIFO and get data from the FIFO. Since the reading pointer is reset to 7 and the required data is in
0~3, the first fetched data is ignored. The required data are then fetched and put into the data buffer in the receiver.

During the reset period, the contents in the FIFO will be cleared. The writing pointer of the FIFO is set to 0 and the reading pointer is set to 7 during the reset period. When the reset signal is released, the FIFO will start at the first rising edge of global clock. Then the FIFO uses the strobe sent along with the data as the writing clock and the local clock of the receiver as the reading clock. Since the FIFO starts 1 global clock cycle earlier then the transmitter and receiver, as long as the frequency of global clock is low enough, the pointers reset of FIFO will always be performed before the strobe arrives.

During high-level system design, a channel model is used to simulate the real data link. This channel model consists of 5 parameters: tx_clock_delay, rx_clk:_delay, data_delay, reset_tx_delay and reset_rx_delay. By setting these parameters the time for the global clock to reach transmitter and receiver, the data link delay and the delays for the reset signal to reach transmitter and receiver can be defined.

In the following synthesis process, this channel model is replaced by a real data bus. The bus consists of 8 data bits to transfer words and 1 strobe bit that transfers the strobe along with the data. Each bit line in this 9-bit bus should have exact the same delay so that no timing error occurs when data arrives the receiver. Extra work is needed in routing to satisfy this constraint. Unlike routing in ASIC, Routing in FPGA is based on existing wires in the FPGA and designers can not route wires wherever they want. To solve this problem, a series of repeaters are added into this data link and the location of each repeater is fixed to the desired place. In this way designers can force the routing scheme of the data bus and make sure the delay of each bit is
matched. A constraint file is required by the synthesis tool to fix the location of these repeaters. In the constraint file the constraint command "LOC" can be used to define the location of those repeaters. "LOC" defines where a design element can be placed within a FPGA. The basic syntax of this command is:

\[
\text{INST "instance_name" LOC=location;}
\]

Where location is a legal location for the part type.

The Virtex-II architecture uses a Cartesian-based XY designator at the slice level. The slice-based location specification uses the form: SLICE_XmYn. The XY slice grid starts as X0Y0 in the lower left CLB tile of the chip. The X values start at 0 and increase horizontally to the right in the CLB row, and the Y values start at 0 and increase vertically up in the CLB column. The block RAMs, TBUFS and multipliers in Virtex-II have their own specification different from the SLICE specifications. Therefore, the location value must start with "SLICE","RAMB","TBUF", OR "MULT",etc. A convenient way to find legal location names is use the FPGA Editor or Floorplanner. By this constraint command designers can control the routing schemes and check what the layout looks like after routing and placement. The layout of the design is shown below in Figure 4.
As shown in Figure 4, the transmitter is put in the left bottom corner of the FPGA chip and the receiver and FIFO are put in the top right corner of the chip. In this condition the data links between transmitter and receiver can fully utilize the size of the FPGA chip and thus longer delay can be achieved.

Two DCMs (Digital Clock Manager) are used to provide actual working clock for the transmitter and receiver. The Virtex-II DCM offers a wide range of powerful clock management features such as clock De-skew, Frequency Synthesis and Phase Shifting. In this design only frequency synthesis is used. One of the DCM's outputs CLKFX can be used to produce clocks at the following
frequency:
FREQCLKFX= (M/D) FREQCLKIN
Where M and D are two integers. By connecting the low frequency global clock to the input of DCM and specifying M to 8, D to 1, designers can get the actual working clock with the frequency as 8 times the frequency of global clock. More characteristics of DCM are described in [11].

There are 12 DCMs in Virtex-II XCV6000 and they are located differently. There is one DCM near the lower left corner and is used to generate working clock for transmitter. Another DCM near the top right corner is used to provide working clock for receiver. "LOC" command is also used to specify the locations of DCMs.

In order to make the debugging easier, the values in the transmitter buffer and in the receiver are shown on the LCD screen of the daughter board. The four digits on the LCD are in 7-segment format and thus 4 decoders are provided to translate the values in the buffer from bit format into 7-segment code. Since the LCD also needs a low frequency clock around 30Hz~300Hz to refresh the display, a clock divider is used to scale down the frequency of global clock into 30Hz~300Hz for the LCD display. Each data in the buffers is 8 bits and thus 2 digits on the LCD are needed to present the full value of a data. From left to right, the first two digits are used to present values in the transmitter buffer and the other two are used to present data in the receiver buffer. There are 4 values in the transmitter buffer but only one can be displayed at a time due to the limited size of the LCD, therefore a pushbutton is used to switch between these 4 values. When the pushbutton is pressed, a clock edge is sent out to the transmitter and the address which controls which value in the buffer should be sent out to the display increases at each clock edge. By this means all the values in the transmitter buffer
can be easily checked. Another pushbutton is used in the same way to switch the display of contents in the receiver buffer.

Also, the addresses of current displayed data can be shown on the LCD. Switching displays from data value to address value is controlled by a pushbutton. When the pushbutton is pressed down, the addresses of both current data is displayed on the LCD; when the pushbutton is released, the data values will be presented on the LCD.

The offset value which the receiver reads during the reset period is controlled by another pushbutton. When the pushbutton is pressed once, the offset value will increase 1. Then during next reset period the receiver will get the new offset value. When the offset value reaches the upper bound which in this design is 7, it will go to 0.

In this design there are two reset buttons. One is for resetting the whole circuit and the other one is for resetting the DCMs. DCM Resets is only needed when the global clock frequency is changed. The other one is used as global reset signal. Once the button is pressed, the transmitter, receiver, FIFO and offset value are all reseted to initial state.

When the test begins, first the global clock is set to a certain frequency, and then the two DCMs are reset to provide actual working clock for the transmitter and receiver. Next, the whole circuit is reset by the global reset signal and start working. At last the data in the transmitter and receiver's buffers are checked to see whether the transfer succeeds or not. This is achieved as follows: First check the 4 data in the transmitter buffer to see what values are sent out by the transmitter, then check the data in the receiver's buffer and compare data with same address in these two buffers. If data in the receiver's buffer are exact the
same as corresponding data in the transmitter's buffer, then it proves the transfer is safe. The offset value is changed later and the global reset signal is set again to restart the whole circuits in order to explore the lowest offset value that guarantees the safety of the transfer at certain frequency. Next the global clock frequency is changed and the testing procedures above are retaken to find the lowest offset value at this frequency. A set of data is collected by these tests.

6. Simulation & Analysis

First high-level simulation is examined. As mentioned above, during high level simulation, a channel model is used instead a real data link. During this high level simulation, it is assumed that 2 ns is needed for the global clock to reach the transmitter and 12 ns is needed to reach the receiver. It is also assumed that 12 ns and 20 ns are needed for the global reset signal to reach the transmitter and receiver respectively since the global reset signal is first synchronized when it enter the design. The delay for the data transfer from transmitter to receiver is set to 35 ns. At first the global clock frequency is set to 25MHz. Since the actual working frequency is 8 times the global clock frequency, the actual working clock is equal to 200MHz. When the offset value is set to 5, the simulation waveform is shown below:
Figure 5  Simulation result 200MHz, offset =5, High Level Simulation

The waveforms above, from top to bottom, shows the global reset signal, the global clock, actual working clock of transmitter, actual working clock of receiver, global clock at the transmitter, global clock at the receiver, read enable signal for the FIFO, data strobe sent out by the transmitter along with data, output from transmitter to channel, output data from channel to FIFO, output data from FIFO to receiver, reset signals for transmitter and receiver, data strobe at the FIFO, reset signal for FIFO, contents of data buffer in transmitter and contents of data buffer in receiver.

It can be seen that the read out enable signal for the FIFO is given almost at the same time as the FIFO starts to read data.
from channel. If the offset value is smaller than the current one, the FIFO will read out data before it get valid data from channel which will cause malfunction. It means that the current offset value, n, is the lowest valid offset value and the relative delay is larger than (n-1) x T <sub>period</sub> and smaller than n x T <sub>period</sub>. Since the offset value is set to 5 and the working clock is 200MHz, we can conclude that the relative delay is around 20 ns~ 25 ns.

Without changing the parameters of the channel model, the global clock frequency is carefully changed and each offset value is tested to find out the lowest offset value under a certain frequency. When the global clock frequency is set to 20MHz and the offset value is set to 4, the simulation waveform as shown below:

![Simulation waveform](image)

**Figure 6 Simulation result 160MHz, offset =4, High Level Simulation**

In this waveform the situation is just like the situation in waveform
above. The FIFO starts to read out almost at the same time as it starts to write in. Thus the lowest offset value at 20MHz is 4 and the delay is around 18.75 ns ~ 25 ns. With the same method, the lowest offset values at different frequencies are found out.

Then the parameters of channel model are changed. It will cost 12 ns for the global clock to reach the transmitter and 22 ns to reach the receiver, but only 2ns for the synchronized global reset signal to reach the transmitter and 12 ns to reach the receiver. The delay for data transfer from transmitter to receiver is still 35 ns. By these settings the reset signal will arrive at transmitter and receiver earlier than the clock after the reset signal is synchronized by the global clock. By setting the global clock frequency to 25MHz and offset value to 5, the simulation waveform is got as below in Figure 8:

Figure 7 Clock Frequency-Offset Curve
By the same methods, another set of lowest offset values at different working frequencies is collected. The results are shown below in Figure 9.

At last, the channel model is changed again so that the global clock and reset signal will arrive at receiver first. It will cost 2 ns for the global clock to reach the receiver and 12 ns to reach the transmitter, and 12 ns for the synchronized reset signal to reach the receiver and 20 ns to reach the transmitter. The delay time for the data from transmitter to receiver is 15 ns. The collected offset values are shown in Figure 10.
Figure 9 Frequency-Offset Curve

Figure 10 Frequency – offset Curve
It is obvious that the three curves are exact the same. According to the formula about how to calculate the offset value which is proposed in section 4, the offset value \( n = \frac{T_{\text{data delay}} - T_{\text{clk skew}}}{T_{\text{period}}}. \) In this formula \( T_{\text{data delay}} \) means the transfer delay time for the data from transmitter to receiver, \( T_{\text{clk skew}} \) presents the clock skew between receiver and transmitter, and \( T_{\text{period}} \) indicates the clock period of actual working clock. In first two cases, the \( T_{\text{data delay}} \) is set to 35 ns, and \( T_{\text{clk skew}} \) is 12 ns- 2 ns=10 ns, thus the relative delay time of the first two cases are the same. In the third case, the \( T_{\text{data delay}} \) is set to 15 ns, and the clock skew is 10 ns, but in this case the clock arrives at receiver first. This will make the receiver starts first and the receiver will wait several clock cycles for the transmitter to start. Thus this delay should be added to the \( T_{\text{data delay}} \) and the total relative delay is then 25 ns, the same as in the first two cases. The high-level simulation results perfectly match the calculation results of the formula in all these three cases.

After successfully proven the feasibility of the proposed design methodology in high-level simulation, the design is also implemented on the FPGA developing board and some testing is taken to examine the performance of the proposed design on real data links. First, the routing of the clock tree is manually defined by a constraint file so that the global clock arrives at the transmitter first. Then the design is burned into the FPGA chip and performance testing begins. A set of offset values at different clock frequencies is collected. After the testing is over, the routing of the clock tree is changed so that the global clock reaches the transmitter first. Then the same testing is performed and testing data is collected. Through the synthesis tool the clock skew and data link delay can be estimated. In the Xilinx ISE 5.1, there is a tool named Timing Analyzer. By this tool the post-placed & route static timing can be estimated. Designers can choose the starting and ending points of a wire and the delay.
will be calculated. Also a timing report of this wire will be generated. In this report designers can find how many flip-flops this wire passes through and the delay for each part. More details of Timing Analyzer can be found in 13. From the tools we know that in both situations, the clock skew is about 10 ns and the data link delay is about 12 ns. The testing results are shown below in Figure 11 and Figure 12.

From the figure below, the rough relative delay can be calculated through the formula derived in section 4, \( n = \frac{T_{\text{data\_delay}} - T_{\text{clk\_skew}}}{T_{\text{period}}} \). From this formula another equation can be derived: 

\[ N \times T_{\text{period}} = T_{\text{data\_delay}} - T_{\text{clk\_skew}} \]

Since \( T_{\text{period}} \) is the reciprocal of clock frequency, the slope of the curve is the reciprocal of the total relative delay. When calculating the delay from the figures it should be noticed that the offset value can only be integers. Thus only those points on the turning corner of the curve should be chosen for measuring delays. For example, in Figure 12, the offset value jumps from 3 to 4 at 188MHz and jumps from 4 to 5 at 216MHz, but stays still from 189MHz to 215MHz. Thus in Figure 12 only points at 188MHz and 216MHz should be chosen to calculate the delays. When the clock arrives at transmitter first, the clock skew compensates the data link delay and the total delay is almost 0. Thus the offset value is always 0 in Figure 11. When the global clock arrives at receiver first, the calculated relative delay is about 22 ns. Taking estimation errors into account, the testing results match the estimated results from Synthesis tool very well. Also through the observation in the testing, the transfer of data is safe under these offset values. Thus we can say this design methodology works in this design.
Figure 11 Frequency-Offset, Clock Reaches Transmitter First

Figure 12 Frequency-Offset, Clock Reaches Receiver First
7. Discussion

In this proposed methodology, the reset signal is synchronized twice by the low frequency global clock. Actually, during exploration we tried to reduce the number of synchronizations. First we tried just to synchronize it when the global reset signal reaches the transmitter and receiver, but the simulation results showed that there is a probability that the different times of releasing global reset signals will cause different performance. As shown below:

![Figure 13 One time Synchronization waveform](image)

In the figure, the “Global Clock Tx” and “Global Clock Rx” describes the signal waveform when the global clock reaches transmitter and receiver respectively. “Reset Tx” and “Reset Rx” presents what the signal looks like when the reset signal arrives at transmitter and receiver.
Assume the global clock arrives at transmitter before receiver, and the skew is $t_1$; the global reset signal also reaches the transmitter $t_2$ earlier than the receiver, and the rising edge of global clock is used to detect the releasing of reset signal. As shown in the figure, in situation 1, the transmitter will get the reset signal first and start working. However, even when the reset signal arrives first at transmitter side, in condition 2, the receiver will start earlier than the receiver. It is not acceptable because it is then not possible to anticipate which condition will appear when the reset button is pressed and thus it is not possible to anticipate the performance or whether the transfer is safe or not.

Another implementation method is also considered during the implementation. By removing the low frequency global clock and making the global reset signal only to be synchronized once by the actual working clock when it enters the transmitter and receiver, the performance of the whole design might be improved. But another problem also arises. It is not easy to make sure the FIFO will be ready before the data comes. However, this method is worth future exploration.

In the design methodology proposed in this paper, estimation of clock skew and data link delays must be done by the designers in order to calculate the suitable offset value. The safety and performance is more or less threatened due to the error in estimation. An interesting method to find a robust point of offset value is mentioned in [6]. In [6] data is transmitted from transmitter, through receiver and then back to transmitter. Number of clock cycles of this transfer is measured and the path delay could be calculated by this number. Thus it is possible to find a robust point. Prior estimation is not needed anymore in this method but the performance is unknown until the layout process finishes, and different routing algorithms will bring different performances.
In this new methodology, the safety of data transfer is the main concern. Using a low frequency clock to synchronize the reset signal twice will add extra waiting time to the transfer process, but it guarantees the safety of data transfer and thus guarantees that no iteration on RTL design or verification is needed and a lot of time and resource cost of the design process is saved. Next the influence of this methodology on circuit performance should be estimated to see what improvements can be done and more efficient methodology might be found.

Chip-to-Chip communication testing was included in this proposed work before. In the proposal the data is sent out from an external connector on the test board and is received through another connector, and a ribbon cable is used to simulate the long data path between two chips. But as the work goes on, we found out due to the package technology of FPGA and the delay differences of wires on the test board to connect external connector and FPGA, it is very hard to balance the delays between each bit in the data bus. The test was cancelled but the problem remains. This situation may also occur in real designs. Data paths between chips may be very long and environments might be very complex. Due to crosstalk, routing or unexpected situations, it might be quite hard to balance the delays between each bit in the bus. Some research can be done on solving these problems.
8. Conclusion

In this paper a new method addressing the timing closure problem in Integrated Circuits Design is proposed. This method is based on STARI [4]. A global clock is used to establish the transmission rate and a two port FIFO is inserted at the receiver side to compensate for clock skew and variations in the delay along the data link. The Initialization methodology is the key point of STARI and also is the main concern of this paper. In the methodology proposed in this paper, the reset signal is synchronized twice by the global clock so that the total relative delay only depends on the clock skew and data link delay, and different skew of the reset signal between transmitter and receiver due to different routing mechanisms will not influence the functionality and performance of the whole design. This method prevents redesign due to timing issues in back-end design and is robust to late changes in RTL-code or floorplan. Little routing consideration is needed because maximum possible delay is already inserted into the high level system model. Since the two-port FIFO is made of cells in standard libraries, any standard digital cell library will support this method.

9. Acknowledgements

The author would like to thank Jonas Nilsson from Hardi Electronics AB for his support and useful discussions.
10. References


7. Jerry Jex, Prantik Nag, Ted Burton and Randy Mooney, “Split FIFO Phase Synchronization for High Speed Interconnect”,

- 29 -


Appendix A  Constraint file for Synthesis

#INST "u_dcm_1"  CLKIN_PERIOD = 40;
#INST "u_dcm_2"  CLKIN_PERIOD = 40;

#clock pin location definition
NET "clk" LOC = "C21";

#pins to LCD location definition
NET "display_rx<0>" LOC = "M30";
NET "display_rx<1>" LOC = "N30";
NET "display_rx<2>" LOC = "K31";
NET "display_rx<3>" LOC = "L31";
NET "display_rx<4>" LOC = "M31";
NET "display_rx<5>" LOC = "N31";
NET "display_rx<6>" LOC = "P30";
NET "display_rx<7>" LOC = "D32";
NET "display_rx<8>" LOC = "C34";
NET "display_rx<9>" LOC = "C31";
NET "display_rx<10>" LOC = "C32";
NET "display_rx<11>" LOC = "C33";
NET "display_rx<12>" LOC = "L30";
NET "display_rx<13>" LOC = "C36";
NET "display_tx<0>" LOC = "D27";
NET "display_tx<1>" LOC = "D28";
NET "display_tx<2>" LOC = "C27";
NET "display_tx<3>" LOC = "C28";
NET "display_tx<4>" LOC = "C29";
NET "display_tx<5>" LOC = "D30";
NET "display_tx<6>" LOC = "D29";
NET "display_tx<7>" LOC = "C20";
NET "display_tx<8>" LOC = "C23";
NET "display_tx<9>" LOC = "D22";
NET "display_tx<10>" LOC = "C24";
NET "display_tx<11>" LOC = "C25";
NET "display_tx<12>" LOC = "D25";
NET "display_tx<13>" LOC = "D24";

#reset signal, push bottons location definition

NET "greset" LOC = "B17";
NET "push_botton_rx" LOC = "A17";
NET "push_botton_tx" LOC = "B16";
NET "clk" TNM_NET = "clk";

# Transmitter, receiver, FIFO location definition
INST "transmitter_component_**" LOC =SLICE_X0Y0:SLICE_X10Y10;
INST "receiver_component_**" LOC
=SLICE_X160Y180:SLICE_X168Y190;
INST "fifo**" LOC =SLICE_X160Y180:SLICE_X168Y190;

#Locations of repeaters on the data path. To define the routing of data bus.
INST "inverter_u1**" LOC =SLICE_X170Y0:SLICE_X172Y3;
INST "inverter_u2**" LOC =SLICE_X170Y95:SLICE_X172Y97;

#reset arrives at transmitter first, repeaters on the reset signal, clock signal.
#INST "inverter_reset_u1**" LOC =SLICE_X80Y0:SLICE_X81Y1;
#INST "inverter_reset_u2**" LOC =SLICE_X10Y0:SLICE_X10Y1;
#INST "inverter_reset_u3**" LOC =SLICE_X10Y0:SLICE_X10Y1;
#INST "inverter_reset_u4**" LOC =SLICE_X160Y0:SLICE_X161Y1;
#INST "inverter_clk_u1**" LOC =SLICE_X80Y0:SLICE_X81Y1;
#INST "inverter_clk_u2**" LOC =SLICE_X10Y0:SLICE_X10Y1;
#reset arrives at receiver first, repeaters on the reset signal, clock signal.
INST "inverter_reset_u1**" LOC =SLICE_X80Y180:SLICE_X80Y181;
INST "inverter_reset_u2**" LOC =SLICE_X160Y180:SLICE_X160Y181;
INST "inverter_reset_u3**" LOC =SLICE_X160Y150:SLICE_X160Y150;
INST "inverter_reset_u4**" LOC =SLICE_X10Y10:SLICE_X11Y11;

INST "inverter_clk_u1**" LOC =SLICE_X80Y180:SLICE_X81Y181;
INST "inverter_clk_u2**" LOC =SLICE_X160Y180:SLICE_X161Y181;
INST "inverter_clk_u3**" LOC =SLICE_X160Y150:SLICE_X161Y151;
INST "inverter_clk_u4**" LOC =SLICE_X11Y11:SLICE_X12Y12;

#location of PLLS
INST "u_dcm_1" LOC =DCM_X0Y0;
INST "U0_BUFG**" LOC =BUFGMUX5S;
INST "U1_BUFG**" LOC =BUFGMUX2P;

INST "u_dcm_2" LOC =DCM_X5Y1;
INST "U2_BUFG**" LOC =BUFGMUX3P;
INST "U4_BUFG**" LOC =BUFGMUX4S;

# location definition of decoder and clock dividers.
INST "decoder**" LOC =SLICE_X0Y160:SLICE_X10Y170;
INST "clock_divider**" LOC =SLICE_X0Y160:SLICE_X10Y170;

# location definition of pins for LCD display, push buttons.
NET "colon" LOC = "D31";
NET "dp<0>" LOC = "C26";
NET "dp<1>" LOC = "C30";
NET "dp<2>" LOC = "C35";
NET "lcd_com<0>" LOC = "D15";
NET "lcd_com<1>" LOC = "C16";
NET "push_botton_off" LOC = "A18";
NET "dcm_reset" LOC = "B18";
NET "push_botton_reg" LOC = "A19";
NET "tx_strobe" TNM_NET = "tx_strobe";
NET "clk_inside_tx" LOC = "E37";
NET "clk_inside_rx" LOC = "E36";

#location definition of external chip communication data bus pins. Disabled
#NET "data_tx_to_channel<0>" LOC = "E3";
#NET "data_tx_to_channel<1>" LOC = "E4";
#NET "data_tx_to_channel<2>" LOC = "F3";
#NET "data_tx_to_channel<3>" LOC = "F4";
#NET "data_tx_to_channel<4>" LOC = "G3";
#NET "data_tx_to_channel<5>" LOC = "G4";
#NET "data_tx_to_channel<6>" LOC = "H3";
#NET "data_tx_to_channel<7>" LOC = "H4";
#NET "tx_strobe" LOC = "J3";
#NET "strobe_rx" LOC = "P34";
#NET "data_channel_to_rx<0>" LOC = "V32";
#NET "data_channel_to_rx<1>" LOC = "W32";
#NET "data_channel_to_rx<2>" LOC = "Y34";
#NET "data_channel_to_rx<3>" LOC = "W34";
#NET "data_channel_to_rx<4>" LOC = "V34";
#NET "data_channel_to_rx<5>" LOC = "U34";
#NET "data_channel_to_rx<6>" LOC = "T34";
#NET "data_channel_to_rx<7>" LOC = "R34";
Appendix B  Timing Reports of Xilinx ISE

1. Clock reaches transmitter first, reset signals timing report.

Release 5.1i - Timing Analyzer F.23
Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.

Design file: testbench.ncd
Physical constraint file: testbench.pcf
Device,speed: xc2v6000,-5 (PRODUCTION 1.110 2002-07-03, STEPPING level 0)
Report level: verbose report

Environment Variable Effect
---------------------  -----
NONE                   No environment variables were set

======================================================================
Timing constraint: PATH "PATHFILTERS" = FROM TIMEGRP "SOURCES" TO TIMEGRP "DESTINATIONS" ;

2 items analyzed, 0 timing errors detected.
Maximum delay is 21.898ns.

======================================================================
Delay: 21.898ns (data path - clock skew)
Source: reset (FF)
Destination: receiver_component_reset (FF)
Data Path Delay: 21.898ns (Levels of Logic = 5)
Clock Skew: 0.000ns
Source Clock: clk_ibufg falling
Destination Clock: clk_inv_4 rising

Constraint Improvement Wizard

Data Path: reset to receiver_component_reset

<table>
<thead>
<tr>
<th>Delay type</th>
<th>Delay(ns)</th>
<th>Logical Resource(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcko</td>
<td>0.493</td>
<td>reset</td>
</tr>
<tr>
<td>net (fanout=14)</td>
<td>5.124</td>
<td>reset</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u1_clk_out1</td>
</tr>
<tr>
<td>net (fanout=3)</td>
<td>2.471</td>
<td>greset_inv_1</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u2_clk_out1</td>
</tr>
<tr>
<td>net (fanout=2)</td>
<td>0.032</td>
<td>greset_inv_2</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u3_clk_out1</td>
</tr>
<tr>
<td>net (fanout=2)</td>
<td>5.127</td>
<td>greset_inv_3</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u4_clk_out1</td>
</tr>
<tr>
<td>net (fanout=2)</td>
<td>5.719</td>
<td>greset_inv_4</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>receiver_component__n00051</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>0.778</td>
<td>receiver_component__n0005</td>
</tr>
<tr>
<td>Tsrck</td>
<td>0.244</td>
<td>receiver_component_reset</td>
</tr>
</tbody>
</table>

Total 21.898ns (2.647ns logic, 19.251ns route)
(12.1% logic, 87.9% route)

---

Delay: 10.810ns (data path - clock skew)
Source: reset (FF)
Destination: transmitter_component_reset (FF)
Data Path Delay: 10.810ns (Levels of Logic = 3)
Clock Skew: 0.000ns
Source Clock: clk_ibufg falling
Destination Clock:  clk_inv_2 rising

Constraint Improvement Wizard

Data Path: reset to transmitter_component_reset

<table>
<thead>
<tr>
<th>Delay type</th>
<th>Delay(ns)</th>
<th>Logical Resource(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcko</td>
<td>0.493</td>
<td>reset</td>
</tr>
<tr>
<td>net (fanout=14)</td>
<td>5.124</td>
<td>reset</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u1_clk_out1</td>
</tr>
<tr>
<td>net (fanout=3)</td>
<td>2.471</td>
<td>greset_inv_1</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u2_clk_out1</td>
</tr>
<tr>
<td>net (fanout=2)</td>
<td>0.798</td>
<td>greset_inv_2</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>transmitter_component__n00051</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>0.534</td>
<td>transmitter_component__n0005</td>
</tr>
<tr>
<td>Tsrck</td>
<td>0.244</td>
<td>transmitter_component_reset</td>
</tr>
</tbody>
</table>

Total          10.810ns (1.883ns logic, 8.927ns route)
                (17.4% logic, 82.6% route)

All constraints were met.

Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock clk
----------------------------------------
<table>
<thead>
<tr>
<th>Src:Rise</th>
<th>Src:Fall</th>
<th>Src:Rise</th>
<th>Src:Fall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest:Rise</td>
<td>Dest:Rise</td>
<td>Dest:Fall</td>
<td>Dest:Fall</td>
</tr>
</tbody>
</table>
----------------------------------------
clk       |         | 21.898|         |
WARNING: Timing: 2718 - Clock nets using general routing resources were found in this design. Clock skew on these resources will not be automatically addressed during path analysis. Please run trce with the -skew option in order to evaluate clock skew and hold violations.

The following clock nets use non-dedicated resources:
  clk_ibufg  clk_inv_2  clk_inv_4

Timing summary:
-----------------
Timing errors: 0  Score: 0
Constraints cover 2 paths, 0 nets, and 15 connections (1.0% coverage)

Analysis completed Thu Feb 03 00:44:43 2005
-----------------------------------------------

Timing Analyzer Settings:
-------------------------
OpenPCF testbench.pcf
Speed -5
DoHoldRaceCheck False
IncludeNets
ExcludeNets
SelectFailingTimingConstraint False
IncludeNoTimingConstraint False
Report Normal
MaxPathsPerTimingConstraint 3
DefineEndpoints ToFFs receiver_component_reset
transmitter_component_reset
DefineEndpoints FromFFs reset
OmitUserConstraints False
DropTimingConstraint
SetForce Off

2. Clock reaches transmitter first, clock signals timing report.

Release 5.1i - Timing Analyzer F.23
Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.

Design file: testbench.ncd
Physical constraint file: testbench.pcf
Device,speed: xc2v6000,-5 (PRODUCTION 1.110 2002-07-03, STEPPING level 0)
Report level: verbose report

Environment Variable Effect
--------------------- ----- 
NONE No environment variables were set

Timing constraint: PATH "PATHFILTERS" = FROM TIMEGRP "SOURCES" TO TIMEGRP "DESTINATIONS";

2 items analyzed, 0 timing errors detected.
Maximum delay is 31.944ns.

Delay: 31.944ns (data path)
Source: clk (PAD)
Destination: receiver_component_reset (FF)
Data Path Delay: 31.944ns (Levels of Logic = 5)

Constraint Improvement Wizard

Data Path: clk to receiver_component_reset

<table>
<thead>
<tr>
<th>Delay type</th>
<th>Delay(ns)</th>
<th>Logical Resource(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiopi</td>
<td>0.718</td>
<td>clk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_ibufg</td>
</tr>
<tr>
<td>net (fanout=17)</td>
<td>16.737</td>
<td>clk_ibufg</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u1_clk_out1</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>2.396</td>
<td>clk_inv_1</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u2_clk_out1</td>
</tr>
<tr>
<td>net (fanout=19)</td>
<td>0.239</td>
<td>clk_inv_2</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u3_clk_out1</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>5.497</td>
<td>clk_inv_3</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u4_clk_out1</td>
</tr>
<tr>
<td>net (fanout=3)</td>
<td>4.829</td>
<td>clk_inv_4</td>
</tr>
</tbody>
</table>

Total 31.944ns (2.246ns logic, 29.698ns route)
(7.0% logic, 93.0% route)

-----------------------------------------------
Delay: 21.381ns (data path)
Source: clk (PAD)
Destination: transmitter_component_reset (FF)
Data Path Delay: 21.381ns (Levels of Logic = 3)

Constraint Improvement Wizard

Data Path: clk to transmitter_component_reset

<table>
<thead>
<tr>
<th>Delay type</th>
<th>Delay(ns)</th>
<th>Logical Resource(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiopi</td>
<td>0.718</td>
<td>clk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_ibufg</td>
</tr>
<tr>
<td>net (fanout=17)</td>
<td>16.737</td>
<td>clk_ibufg</td>
</tr>
</tbody>
</table>
Tilo 0.382 inverter_clk_u1_clk_out1
net (fanout=1) 2.396 clk_inv_1
Tilo 0.382 inverter_clk_u2_clk_out1
net (fanout=19) 0.766 clk_inv_2

Total 21.381ns (1.482ns logic, 19.899ns route)
(6.9% logic, 93.1% route)

All constraints were met.

Data Sheet report:
--------------
No constraints were found to generate data for the Data Sheet Report section.
Use the Advanced Analysis (-a) option or generate global constraints for each
clock, its pad to setup and clock to pad paths, and a pad to pad constraint.

Timing summary:
--------------
Timing errors: 0  Score: 0
Constraints cover 2 paths, 0 nets, and 6 connections (0.4% coverage)

Analysis completed Thu Feb 03 00:50:25 2005
Timing Analyzer Settings:

OpenPCF testbench.pcf
Speed -5
DoHoldRaceCheck False
IncludeNets
ExcludeNets
SelectFailingTimingConstraint False
IncludeNoTimingConstraint False
Report Normal
MaxPathsPerTimingConstraint 3
DefineEndpoints ToFFs receiver_component_reset transmitter_component_reset
DefineEndpoints FromPads clk
OmitUserConstraints False
DropTimingConstraint
SetForce Off

3. Clock reaches receiver first, reset signals timing report.

Release 5.1i - Timing Analyzer F.23
Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.

Design file: testbench.ncd
Physical constraint file: testbench.pcf
Device,speed: xc2v6000,-5 (PRODUCTION 1.110 2002-07-03, STEPPING level 0)
Report level: verbose report

Environment Variable Effect
-------------------- -----
NONE No environment variables were set

- 42 -
Timing constraint: PATH "PATHFILTERS" = FROM TIMEGRP "SOURCES" TO TIMEGRP "DESTINATIONS";

2 items analyzed, 0 timing errors detected.
Maximum delay is 18.832ns.

Delay: 18.832ns (data path - clock skew)
Source: reset (FF)
Destination: transmitter_component_reset (FF)
Data Path Delay: 18.832ns (Levels of Logic = 5)
Clock Skew: 0.000ns
Source Clock: clk_ibufg falling
Destination Clock: clk_inv_4 rising

Constraint Improvement Wizard
Data Path: reset to transmitter_component_reset

<table>
<thead>
<tr>
<th>Delay type</th>
<th>Delay(ns)</th>
<th>Logical Resource(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcko</td>
<td>0.493</td>
<td>reset</td>
</tr>
<tr>
<td>net (fanout=14)</td>
<td>2.060</td>
<td>reset</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u1_clk_out1</td>
</tr>
<tr>
<td>net (fanout=3)</td>
<td>2.881</td>
<td>greset_inv_1</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u2_clk_out1</td>
</tr>
<tr>
<td>net (fanout=2)</td>
<td>1.234</td>
<td>greset_inv_2</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u3_clk_out1</td>
</tr>
<tr>
<td>net (fanout=3)</td>
<td>9.223</td>
<td>greset_inv_3</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u4_clk_out1</td>
</tr>
<tr>
<td>net (fanout=2)</td>
<td>0.019</td>
<td>greset_inv_4</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>transmitter_component__n00051</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>0.768</td>
<td>transmitter_component__n0005</td>
</tr>
<tr>
<td>Tsrck</td>
<td>0.244</td>
<td>transmitter_component_reset</td>
</tr>
</tbody>
</table>

- 43 -
Total 18.832ns (2.647ns logic, 16.185ns route)  
(14.1% logic, 85.9% route)

-------------------------------------------------------------------------------------------------
Delay: 7.905ns (data path - clock skew)
Source: reset (FF)
Destination: receiver_component_reset (FF)
Data Path Delay: 7.905ns (Levels of Logic = 3)
Clock Skew: 0.000ns
Source Clock: clk_ibufg falling
Destination Clock: clk_inv_2 rising
Constraint Improvement Wizard
Data Path: reset to receiver_component_reset

<table>
<thead>
<tr>
<th>Delay type</th>
<th>Delay(ns)</th>
<th>Logical Resource(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcko</td>
<td>0.493</td>
<td>reset</td>
</tr>
<tr>
<td>net (fanout=14)</td>
<td>2.060</td>
<td>reset</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u1_clk_out1</td>
</tr>
<tr>
<td>net (fanout=3)</td>
<td>2.881</td>
<td>greset_inv_1</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_reset_u2_clk_out1</td>
</tr>
<tr>
<td>net (fanout=2)</td>
<td>0.023</td>
<td>greset_inv_2</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>receiver_component__n00051</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>1.058</td>
<td>receiver_component__n0005</td>
</tr>
<tr>
<td>Tsrck</td>
<td>0.244</td>
<td>receiver_component_reset</td>
</tr>
</tbody>
</table>

-------------------------------------------------------------------------------------------------
Total 7.905ns (1.883ns logic, 6.022ns route)  
(23.8% logic, 76.2% route)

-------------------------------------------------------------------------------------------------

All constraints were met.
Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock clk

<table>
<thead>
<tr>
<th>Src:Rise</th>
<th>Src:Fall</th>
<th>Src:Rise</th>
<th>Src:Fall</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.832</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WARNING: Timing: 2718 - Clock nets using general routing resources were found in this design. Clock skew on these resources will not be automatically addressed during path analysis. Please run trce with the -skew option in order to evaluate clock skew and hold violations.

The following clock nets use non-dedicated resources:
- clk_ibufg
- clk_inv_4
- clk_inv_2

Timing summary:
-----------------
Timing errors: 0  Score: 0

Constraints cover 2 paths, 0 nets, and 15 connections (1.0% coverage)

Analysis completed Thu Feb 03 02:36:56 2005
---------------------------------------------------------------

Timing Analyzer Settings:
OpenPCF testbench.pcf
Speed -5
DoHoldRaceCheck False
IncludeNets
ExcludeNets
SelectFailingTimingConstraint False
IncludeNoTimingConstraint False
Report Normal
MaxPathsPerTimingConstraint 3
DefineEndpoints ToFFs receiver_component_reset transmitter_component_reset
DefineEndpoints FromFFs reset
OmitUserConstraints False
DropTimingConstraint
SetForce Off

4. Clock reaches receiver first, clock signals timing report.

Release 5.1i - Timing Analyzer F.23
Copyright (c) 1995-2002 Xilinx, Inc. All rights reserved.

Design file: testbench.ncd
Physical constraint file: testbench.pcf
Device,speed: xc2v6000,-5 (PRODUCTION 1.110 2002-07-03, STEPPING level 0)
Report level: verbose report

Environment Variable Effect
---------------------- ----- NO environment variables were set
Timing constraint: PATH "PATHFILTERS" = FROM TIMEGRP "SOURCES" TO TIMEGRP "DESTINATIONS";

2 items analyzed, 0 timing errors detected. Maximum delay is  20.126ns.

Delay: 20.126ns (data path)
Source: clk (PAD)
Destination: transmitter_component_reset (FF)
Data Path Delay: 20.126ns (Levels of Logic = 5)

Constraint Improvement Wizard
Data Path: clk to transmitter_component_reset

<table>
<thead>
<tr>
<th>Delay type</th>
<th>Delay(ns)</th>
<th>Logical Resource(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiopi</td>
<td>0.718</td>
<td>clk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_ibufg</td>
</tr>
<tr>
<td>net (fanout=17)</td>
<td>1.335</td>
<td>clk_ibufg</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u1_clk_out1</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>3.125</td>
<td>clk_inv_1</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u2_clk_out1</td>
</tr>
<tr>
<td>net (fanout=4)</td>
<td>2.623</td>
<td>clk_inv_2</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u3_clk_out1</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>8.743</td>
<td>clk_inv_3</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u4_clk_out1</td>
</tr>
<tr>
<td>net (fanout=18)</td>
<td>2.054</td>
<td>clk_inv_4</td>
</tr>
</tbody>
</table>

Total 20.126ns (2.246ns logic, 17.880ns route)  
(11.2% logic, 88.8% route)

Delay: 8.040ns (data path)
Source: clk (PAD)
Destination: receiver_component_reset (FF)
Data Path Delay: 8.040ns (Levels of Logic = 3)

Constraint Improvement Wizard

Data Path: clk to receiver_component_reset

<table>
<thead>
<tr>
<th>Delay type</th>
<th>Delay (ns)</th>
<th>Logical Resource(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiopi</td>
<td>0.718</td>
<td>clk</td>
</tr>
<tr>
<td>net (fanout=17)</td>
<td>1.335</td>
<td>clk_ibufg</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u1_clk_out1</td>
</tr>
<tr>
<td>net (fanout=1)</td>
<td>3.125</td>
<td>clk_inv_1</td>
</tr>
<tr>
<td>Tilo</td>
<td>0.382</td>
<td>inverter_clk_u2_clk_out1</td>
</tr>
<tr>
<td>net (fanout=4)</td>
<td>2.098</td>
<td>clk_inv_2</td>
</tr>
</tbody>
</table>

Total 8.040ns (1.482ns logic, 6.558ns route)
(18.4% logic, 81.6% route)

All constraints were met.

Data Sheet report:
---------------
No constraints were found to generate data for the Data Sheet Report section.
Use the Advanced Analysis (-a) option or generate global constraints for each clock, its pad to setup and clock to pad paths, and a pad to pad constraint.

Timing summary:
---------------

- 48 -
Timing errors: 0  Score: 0

Constraints cover 2 paths, 0 nets, and 6 connections (0.4% coverage)

Analysis completed Thu Feb 03 02:37:52 2005
--------------------------------------------------------------------------------

Timing Analyzer Settings:
-------------------------------
OpenPCF testbench.pcf
Speed -5
DoHoldRaceCheck False
IncludeNets
ExcludeNets
SelectFailingTimingConstraint False
IncludeNoTimingConstraint False
Report Normal
MaxPathsPerTimingConstraint 3
DefineEndpoints ToFFs receiver_component_reset transmitter_component_reset
DefineEndpoints FromPads clk
OmitUserConstraints False
DropTimingConstraint
SetForce Off

5. Timing report of delay of data path
--------------------------------------------------------------------------------

Release 5.1i - Timing Analyzer F.23
Copyright (c) 1995-2002 Xilinx, Inc.  All rights reserved.

Design file: testbench.ncd
Physical constraint file: testbench.pcf
Device,speed: xc2v6000,-5  (PRODUCTION 1.110
2002-07-03, STEPPING level 0)
Report level: verbose report

Environment Variable         Effect
-------------------------------
NONE                           No environment variables were set

================================================================================================
Timing constraint: PATH "PATHFILTERS" = FROM TIMEGRP "SOURCES" TO TIMEGRP "DESTINATIONS" ;

1 item analyzed, 0 timing errors detected.
Maximum delay is 12.339ns.

==========================================================================================
Delay: 12.339ns (data path - clock skew)
Source: transmitter_component_data_tx_to_channel_0 (FF)
Destination: fifo_u1_content_0_0 (FF)
Data Path Delay: 12.339ns (Levels of Logic = 2)
Clock Skew: 0.000ns
Source Clock: tx_clk rising
Destination Clock: tx_strobe_inv_2 rising
Constraint Improvement Wizard
Data Path: transmitter_component_data_tx_to_channel_0 to fifo_u1_content_0_0
Delay type Delay(ns) Logical Resource(s)
------------------ ------------------
Tcko               0.493              transmitter_component_data_tx_to_channel_0
net (fanout=1)     3.726              data_tx_to_channel<0>
Tilo               0.382              inverter_u1_data_out<0>1

- 50 -
net (fanout=1) 3.312 data_tx_to_channel_inv_1<0>  
Tilo 0.382 inverter_u2_data_out<0>1  
net (fanout=8) 3.722 data_channel_to_fifo<0>  
Tdick 0.322 fifo_u1_content_0_0  

---------------------------------------------------------------------
Total 12.339ns (1.579ns logic, 10.760ns route)  
(12.8% logic, 87.2% route)  

---------------------------------------------------------------------

All constraints were met.

Data Sheet report:
------------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock clk
---------------------------------------
| Src:Rise| Src:Fall| Src:Rise| Src:Fall|  
Source Clock |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|  
-------------------------------+-----------------+-----------------+-----------------+-----------------+  
clk | 12.339| | | |  
-------------------------------+-----------------+-----------------+-----------------+-----------------+  

WARNING:Timing:2718 - Clock nets using general routing resources were found in 
this design. Clock skew on these resources will not be automatically 
addressed during path analysis. Please run trce with the -skew option in 
order to evaluate clock skew and hold violations.

The following clock nets use non-dedicated resources:  
  tx_strobe_inv_2  

- 51 -
Timing summary:

---------------

Timing errors: 0  Score: 0

Constraints cover 1 paths, 0 nets, and 28 connections (1.8% coverage)

Analysis completed Fri Feb 11 14:41:25 2005

--------------------------------------------------------------------------------

Timing Analyzer Settings:

-------------------------

OpenPCF testbench.pcf
Speed -5
DoHoldRaceCheck False
IncludeNets
ExcludeNets
SelectFailingTimingConstraint False
IncludeNoTimingConstraint False
Report Normal
MaxPathsPerTimingConstraint 3
DefineEndpoints ToFFs fifo_u1_content_0_0
DefineEndpoints FromFFs transmitter_component_data_tx_to_channel_0
OmitUserConstraints False
DropTimingConstraint
SetForce Off
Appendix C   ModelSim

Simulation Transcript

--In order to drive DCMs in ModelSim, it is necessary to set the resolution to ps. --Also it is important to reset the DCMs for at least 3 clock cycles.

vsim -t ps work.testbench
view signals
add wave sim:/testbench/greset
add wave sim:/testbench/clk
add wave sim:/testbench/tx_clk
add wave sim:/testbench/rx_clk
add wave sim:/testbench/tx_gclk
add wave sim:/testbench/rx_gclk
add wave sim:/testbench/read_en
add wave sim:/testbench/tx_strobe
add wave sim:/testbench/data_tx_to_channel
add wave sim:/testbench/data_fifo_to_rx
add wave sim:/testbench/tx_greset
add wave sim:/testbench/tx_greset
add wave sim:/testbench/strobe_rx
add wave sim:/testbench/fifo_reset
add wave sim:/transmitter_component/tx_reg
add wave sim:/receiver_component/rx_reg

force greset 0 0ps, 1 4000000ps, 0 5000000ps, 1 6010000ps
force dcm_reset 0 0ps, 1 4000000ps
force clk -repeat 42000ps, 0 0ps, 1 21000ps
run 8000000ps
Appendix D Photos of Test Board and Equipments
Appendix E  VHDL Codes

1. Transmitter

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity transmitter is

port(
    clk:     in std_logic;  ----global clock;
    greset, tx_clk:   in std_logic;  ----global reset, local clock;
    data_preload:   in std_logic_vector(7 downto 0);  ----data bus to load data for transfer from external devices;
    reg_address:  in std_logic_vector(1 downto 0);  ----address bus for load transfer data;
    data_tx_to_chanel: out std_logic_vector(7 downto 0);  ----data bus for transfer data to channel;
    tx_strobe:       out std_logic;  ----data strobe along with data
    reg_out:          out std_logic_vector(7 downto 0);  ----data bus for LCD display;
    out_address:     in std_logic_vector(1 downto 0)  ----data address for LCD display;
);

end transmitter;
architecture behav of transmitter is

type shift_reg is array (integer range <>) of std_logic_vector(7 downto 0);

signal reset: std_logic;
----Synchronized reset signal;
signal Tx_reg : shift_reg(3 downto 0);     ----data buffer;
signal tx_counter: integer range 0 to 15;
-----counter for count local clock edges;
signal time_counter: integer range 0 to 3;
-----counter for count global clock edges;

attribute keep: string;
attribute keep of reset: signal is "true";

begin

synchronize: process(greset, clk)
-----release synchronized reset signal at the second
-----clock rising edge of global clock after the global
-----clock is released;
begin
if clk'event and clk='1' then
if greset='0' then
reset<='0';
time_counter<=0;
elsif time_counter>=1 then
reset<='1';
else reset<='0';
time_counter<=time_counter+1;
end if;
end if;
end if;
end process;

process(reset, tx_clk, reg_address)
----send data out from data buffer to data bus ;
begin

  if tx_clk'event and tx_clk='1'
    then if reset='0'
      then
          tx_counter<=0;
          data_tx_to_channel<="11111111";

        elsif tx_counter<=5
          then if tx_counter<=4 and tx_counter>=1
            then

          data_tx_to_channel<=Tx_reg(tx_counter-1);
            end if;
          tx_counter<=tx_counter+1;

          end if;
        end if;
      end if;
    end if;
end process;

INITILIZE: process(reset, clk)
---- read transfer data from external devices during reset period;
begin
if clk'event and clk='0'

then if reset='0'

then

case reg_address is

when "00"=>  Tx_reg(0)<=data_preload;
when "01"=>  Tx_reg(1)<=data_preload;
when "10"=>  Tx_reg(2)<=data_preload;
when "11"=>  Tx_reg(3)<=data_preload;
when others=> null;

end case;

end if;

end if;

end process;

debug_process: process(reset,tx_clk)

---- read data out to LCD display according to data address;

begin

if tx_clk'event and tx_clk='1'

then

if reset='0'

then reg_out<=(others=>'0');

else

case out_address is

when "00"=>  reg_out<=Tx_reg(0);
when "01"=>  reg_out<=Tx_reg(1);
when "10"=>  reg_out<=Tx_reg(2);
when "11"=>  reg_out<=Tx_reg(3);

end case;

end if;

end if;

end process;
when others=> null;
  end case;
end if;
end if;

end process;

tx_strobe<='0' when reset='0' else
  ----send strobe along with data
  '0' when tx_counter<=1 else
  tx_clk when tx_counter<6 else
  '1' when tx_counter=6 else
  '0';

end behav;

2. Receiver

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity receiver is

port (clk,greset, tx_strobe, rx_clk: in std_logic;
  ----global clock, global reset, data strobe, local clock;
  offset: in std_logic_vector(2 downto 0);
  ----offset value;
  data_channel_to_rx: in std_logic_vector(7 downto 0);
  ----data input bus;
reg_address: in std_logic_vector(1 downto 0);
    ----address which used for LCD display;
read_en: out std_logic;
    ----read enable signal for FIFO;
fifo_reset: out std_logic;
    ----reset signal for FIFO;
data_to_display: out std_logic_vector(7 downto 0)
    ----data bus for LCD display;

); end receiver;

architecture behav of receiver is

type shift_reg is array (integer range <>) of std_logic_vector(7 downto 0);

signal rx_reg : shift_reg(3 downto 0);
    ----buffer for storing fetrhed data
signal counter: integer range 0 to 15;
    ----counter for counting offset value;
signal off_reg: std_logic_vector(2 downto 0);
    ----buffer for offset value;
signal reset: std_logic;
    ----synchronized reset signal;
signal time_counter: integer range 0 to 3;
    ----counter to count rising edge of global clock;

attribute keep: string;
attribute keep of reset: signal is "true";

begin

- 60 -
Process (greset, clk)
---- use global clock to synchronize global clock
---- at the first rising edge, the FIFO is reset;
---- at the second rising edge, the receiver is reset;

begin
if clk'event and clk='1' then
if greset='0' then
reset<='0';
time_counter<=0;
fifo_reset<='0';
elsif time_counter>=1 then
reset<='1';
fifo_reset<='1';
else time_counter<=time_counter+1;
    fifo_reset<='1';
    reset<='0';
end if;
end if;
end if;

end process;

Process (reset, offset, rx_clk)
---- read offset value from external device during reset period
---- Count clock cycles when reset signal is released.
---- When counter ends, assert read enable signal for the FIFO to
---- read data from FIFO.

Begin
if rx_clk'event and rx_clk='1'
    then if reset='0'
        then off_reg<=offset;
        end if;
    end if;
end if;

read_en<='0';
elif reset='1' then

    if off_reg=0 then
        read_en<='1';
        off_reg<=off_reg;
    else off_reg<=off_reg-1;
        end if;
    end if;

end if;

end process;

Process (reset, rx_clk)
    ----read data from FIFO at certain time after offset value counter is ended.

Begin

if rx_clk'event and rx_clk='1'
    then if reset='0' then
        rx_reg<=(others=>'00010000');
        counter<=0;
    elseif counter<6 and off_reg=0 then
        if counter>=2 then
            rx_reg(counter-2)<=data_channel_to_rx;
            end if;
            counter<=counter+1;
        end if;
    end if;

end if;

end process;
process(reset, rx_clk, reg_address) ---- process for LCD display

begin

if rx_clk'event and rx_clk='1'
    then if reset='0'
        then data_to_display<=(others=>'0');
        elsif reset='1' then
            case reg_address is
                when "00" => data_to_display<=rx_reg(0);
                when "01" => data_to_display<=rx_reg(1);
                when "10" => data_to_display<=rx_reg(2);
                when "11" => data_to_display<=rx_reg(3);
                when others=> null;
            end case;

    end if;
end if;
end process;

end;

3. FIFO

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity FIFO is
port(
architecture behav of FIFO is

begin

process(reset, clk_in)----write pointer process
begin
if reset = '0' then
    in_counter<=0;
    content<=(others=>"00000100");
elsif clk_in'event and clk_in='1'
then
    content(in_counter)<=data_in;
    if in_counter=7 then
        in_counter<=0;
    else
        in_counter<=in_counter+1;
    end if;
end if;
end if;
end process;
end if;
end process;
end architecture behav;
end process;

process(reset, clk_out) ----read pointer process
begin
if   reset ='0' then
    out_counter<=7;

    elsif clk_out'event and clk_out='1' then
        if
            read_en='1' then if
                out_counter=7 then
                    out_counter<=0;
                else  out_counter<=out_counter+1;
            end if;
        end if;
    end if;
end if;
end process;

data_out<=content(out_counter);

end behav;

4. Clock Divider
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity clock_div is
port(clk_in: in std_logic;
    reset: in std_logic;
    clk_out: out std_logic);
end clock_div;

-----divide global clock to 30~300Hz;
architecture behav of clock_div is

    signal count: integer range 0 to 111070;

begin
    process(reset, clk_in)
    begin

        if reset='0' then count<=0;
        elsif clk_in'event and clk_in='1' then
            if count>=111069 then
                count<=0;
            else count<=count+1;
            end if;

            if count>=0 and count<55535 then
                clk_out<='0';
            else clk_out<='1';
            end if;
        end if;
    end process;

end behav;

5. Decoder
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity decoder is
  port (data_in: in std_logic_vector(7 downto 0);
        CLK:  IN STD_LOGIC;
        data_out: out std_logic_vector(13 downto 0)
  );
end decoder;

architecture behav of decoder is
  signal data_out_tmp: std_logic_vector(13 downto 0);
begin
  process(data_in)
  begin
    case data_in(3 downto 0) is
      when "0000" => data_out_tmp(6 downto 0) <= "1111110";  -- 0
      when "0001" => data_out_tmp(6 downto 0) <= "0110000";  -- 1
      when "0010" => data_out_tmp(6 downto 0) <= "1101101";  -- 2
      when "0011" => data_out_tmp(6 downto 0) <= "1111001";  -- 3
      when "0100" => data_out_tmp(6 downto 0) <= "0110011";  -- 4
      when "0101" => data_out_tmp(6 downto 0) <= "1011011";  -- 5
      when "0110" => data_out_tmp(6 downto 0) <= "1111001";  -- 6
      when "0111" => data_out_tmp(6 downto 0) <= "0011111";  -- 7
      when "1000" => data_out_tmp(6 downto 0) <= "1110000";  -- 8
      when "1001" => data_out_tmp(6 downto 0) <= "1111111";  -- 9
      when "1010" => data_out_tmp(6 downto 0) <= "1110111";  -- a
      when "1011" => data_out_tmp(6 downto 0) <= "0011111";  -- b
      when "1100" => data_out_tmp(6 downto 0) <= "1001110";  -- c
    end case;
  end process;
end behav;
when "1101" => data_out_tmp(6 downto 0) <= "0111101";  -- d
when "1110" => data_out_tmp(6 downto 0) <= "1001111";  -- e
when "1111" => data_out_tmp(6 downto 0) <= "1000111";  -- f
when others => data_out_tmp(6 downto 0) <= "0000000";  -- null
end case;

case data_in(7 downto 4) is
when "0000" => data_out_tmp(13 downto 7) <= "1111110";  -- 0
when "0001" => data_out_tmp(13 downto 7) <= "0110000";  -- 1
when "0010" => data_out_tmp(13 downto 7) <= "1101101";  -- 2
when "0011" => data_out_tmp(13 downto 7) <= "1111001";  -- 3
when "0100" => data_out_tmp(13 downto 7) <= "0110011";  -- 4
when "0101" => data_out_tmp(13 downto 7) <= "1011011";  -- 5
when "0110" => data_out_tmp(13 downto 7) <= "1011111";  -- 6
when "0111" => data_out_tmp(13 downto 7) <= "1110011";  -- 7
when "1000" => data_out_tmp(13 downto 7) <= "1111111";  -- 8
when "1001" => data_out_tmp(13 downto 7) <= "1110011";  -- 9
when "1010" => data_out_tmp(13 downto 7) <= "1110111";  -- a
when "1011" => data_out_tmp(13 downto 7) <= "0011111";  -- b
when "1100" => data_out_tmp(13 downto 7) <= "1001110";  -- c
when "1101" => data_out_tmp(13 downto 7) <= "0111101";  -- d
when "1110" => data_out_tmp(13 downto 7) <= "1001111";  -- e
when "1111" => data_out_tmp(13 downto 7) <= "1000111";  -- f
when others => data_out_tmp(13 downto 7) <= "0000000";  -- null
end case;

end process;
data_out <= data_out_tmp when clk='0' else not data_out_tmp;

end behav;
6. Inverters
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

-- inverter which is used as repeaters in the data path.

entity inverter is
port(data_in: in std_logic_vector(7 downto 0);
     data_out: out std_logic_vector(7 downto 0);
     clk_in: in std_logic;
     clk_out:out std_logic);
end inverter;

architecture behav of inverter is
begin
    data_out<=not data_in;
    clk_out<=not clk_in;
end behav;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
-- inverter which is used as repeaters for reset signal;

entity inverter_reset is
port(
    clk_in: in std_logic;
end inverter_reset is

clk_out: out std_logic);
end inverter_reset;

architecture behav of inverter_reset is

begin
clk_out<=not clk_in;
end behav;

7. Channel Model
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Channel is
generic (tx_gclk_delay, rx_gclk_delay, tx_greset_delay, rx_greset_delay, data_delay: time);
--parameters above are delay for global clock to reach transmitter and receiver, delays for global reset signal to reach transmitter and receiver, --and delay time for the data path;
port( clk, greset, tx_strobe : in std_logic;
    strobe_rx, tx_greset, rx_greset : out std_logic;
    tx_gclk, rx_gclk : out std_logic;
    data_tx_to_channel : in std_logic_vector(7 downto 0);
    data_channel_to_rx : out std_logic_vector(7 downto 0)
);

end Channel;

architecture behav of Channel is
begin

strobe_rx<= transport tx_strobe after data_delay;
data_channel_to_rx<= transport data_tx_to_channel after data_delay;
tx_greset<= transport greset after tx_greset_delay;
rx_greset<= transport greset after rx_greset_delay;
tx_gclk<= transport clk after tx_gclk_delay;
rx_gclk<= transport clk after rx_gclk_delay;

end;

8. High Level system model using channel model
library IEEE;
--library SIMPRIM;
library UNISIM;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
--use SIMPRIM.all;
use UNISIM.all;

entity testbench is
port(greset, clk: in std_logic;
----------------global reset signal, global clock;
dcm_reset: in std_logic;
----------------reset signal for dcm;
push_button_tx, push_button_rx: in std_logic;
----------------pushbuttons to decide which data is displayed on the LCD;
push_button_reg: in std_logic;
----------------display which reg is on the LCD
push_button_off: in std_logic;
-----------------adjust the value of offset
dp: out std_logic_vector(2 downto 0);
architecture behav of testbench is
--local clock at transmitter and receiver;
signal tx_clk, rx_clk: std_logic;
--global clock at transmitter and receiver;
signal tx_gclk, rx_gclk: std_logic;
--read enable signal for FIFO;
signal read_en: std_logic;
--data strobe;
signal tx_strobe: std_logic;

--data bus from transmitter to channel, from channel to FIFO,
--and from FIFO to receiver;
signal data_tx_to_channel: std_logic_vector(7 downto 0);
signal data_channel_to_fifo: std_logic_vector(7 downto 0);
signal data_channel_to_rx: std_logic_vector(7 downto 0);

--address of the buffer in transmitter and receiver;
signal tx_reg_address, rx_reg_address: std_logic_vector(1 downto 0);
--buffer to store offset value;
signal offset: std_logic_vector(2 downto 0);
--data bus for transmitter to load transfer data during reset period;
--and data bus from receiver to LCD display;
signal data_preload, data_to_display: std_logic_vector(7 downto 0);
signal out_address: std_logic_vector(1 downto 0);
signal tx_reg_out: std_logic_vector(7 downto 0);

--signals for configuration of DCMs;
signal open_pin: std_logic;
signal reset_DCM: std_logic;
signal CLK0_W_1: std_logic;
signal CLK1X_W_1: std_logic;
signal CLKFX_W_1: std_logic;
signal CLK0_W_2: std_logic;
signal CLK1X_W_2: std_logic;
signal CLKFX_W_2: std_logic;

--signals for LCD display;
signal backplane: std_logic;--clock for LCD
signal decoder_out_tx: std_logic_vector(7 downto 0);
signal decoder_out_rx: std_logic_vector(7 downto 0);

--global reset signal at transmitter and receiver,and data strobe at transmitter,
signal tx_greset, rx_greset: std_logic;
signal strobe_rx: std_logic;

--reset signal after the first synchronization and reset signal for FIFO;
signal reset: std_logic;
signal fifo_reset: std_logic;

component Transmitter
component Receiver
port ( clk, greset, tx_strobe, rx_clk: in std_logic;
      offset : in std_logic_vector(2 downto 0);
      data_channel_to_rx : in std_logic_vector(7 downto 0);
      reg_address : in std_logic_vector(1 downto 0);
      read_en : out std_logic;
      fifo_reset : out std_logic;
      data_to_display : out std_logic_vector(7 downto 0)
    );
end component;

component FIFO
port(
data_in: in std_logic_vector(7 downto 0);

data_out: out std_logic_vector(7 downto 0);
--out_address: out std_logic_vector(2 downto 0);
clk_in: in std_logic;
clk_out: in std_logic;
read_en:in std_logic;
reset: in std_logic);
end component;
component decoder

port ( data_in: in std_logic_vector(7 downto 0);
      clk: in std_logic;
      data_out: out std_logic_vector(13 downto 0)
);
end component;

component Channel
generic (tx_gclk_delay, rx_gclk_delay, tx_greset_delay, rx_greset_delay,
         data_delay: time);

port( clk,greset, tx_strobe : in std_logic;
      strobe_rx,tx_greset, rx_greset : out std_logic;
      tx_gclk, rx_gclk : out std_logic;
      data_tx_to_channel : in std_logic_vector(7 downto 0);
      data_channel_to_rx : out std_logic_vector(7 downto 0)
);
end component;

component BUFG

port ( I : in std_logic;
       O : out std_logic
);
end component;
component LCD_COLON
port(clk: in std_logic;
  DP: OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
  COLON: OUT STD_LOGIC
);
end component;

component clock_div
port(clk_in: in std_logic;
  reset: in std_logic;
  clk_out: out std_logic);
end component;

component DCM
  -- pragma translate_off
  -- For the list of all attributes, see Virtex-II Handbook - Chapter 2
  generic (
    DLL_FREQUENCY_MODE : string := "LOW";
    CLKFX_MULTIPLY : integer := 8 ;
    CLKFX_DIVIDE : integer := 1; 
    DUTY_CYCLE_CORRECTION : boolean := TRUE;
    STARTUP_WAIT : boolean := FALSE 
  );
  -- pragma translate_on
  port ( CLKIN : in std_logic;
     CLKFB : in std_logic;
     DSSEN : in std_logic;
     \- 76 -\n\- 76 -
PSINCDEC  : in  std_logic;
PSEN      : in  std_logic;
PSCLK     :  in  std_logic;
RST       :  in  std_logic;
CLK0      : out std_logic;
CLK90     : out std_logic;
CLK180    : out std_logic;
CLK270    : out std_logic;
CLK2X     : out std_logic;
CLK2X180  : out std_logic;
CLKDV     : out std_logic;
CLKFX     : out std_logic;
CLKFX180  : out std_logic;
LOCKED    : out std_logic;
PSDONE    : out std_logic;
STATUS    : out std_logic_vector(7 downto 0)
);
end component;

--initialization of the DCMs
attribute DLL_FREQUENCY_MODE : string;
attribute DUTY_CYCLE_CORRECTION : string;
attribute CLKOUT_PHASE_SHIFT : string;
attribute PHASE_SHIFT  : integer;
attribute CLKFX_MULTIPLY : integer;
attribute CLKFX_DIVIDE : integer;
attribute STARTUP_WAIT : string;
attribute CLKin_PERIOD : string;

attribute DLL_FREQUENCY_MODE of U_DCM_1: label is "LOW";
attribute DUTY_CYCLE_CORRECTION of U_DCM_1: label is "TRUE";
attribute CLKin_PERIOD OF U_DCM_1: label is "40";
attribute PHASE_SHIFT  of U_DCM_1: label is 0;

- 77 -
```verbatim
begin

transmitter_component: Transmitter

port map(tx_gclk, tx_greset, tx_clk, data_preload, tx_reg_address,
data_tx_to_channel, tx_strobe, tx_reg_out, out_address);

receiver_component: Receiver

port map(rx_gclk, rx_greset, strobe_rx, rx_clk, offset, data_FIFO_to_rx,
rx_reg_address, read_en, fifo_reset, data_to_display);

channel_component: channel

--generic (tx_gclk_delay, rx_gclk_delay, tx_greset_delay, rx_greset_delay,
data_delay: time);

--different settings of the parameters of the channel.
```
--generic map(2 ns,12 ns,12 ns,20 ns,35 ns)
--generic map(12 ns,22 ns,2 ns,12 ns,35 ns)
--generic map(12 ns,2 ns,20 ns,12 ns,15 ns)

generic map(5 ns,5 ns,5 ns,5 ns,25 ns)

port map(clk, reset, tx_strobe, strobe_rx, tx_greset, rx_greset, tx_gclk, rx_gclk, data_tx_to_channel, data_channel_to_rx);

FIFO_U1: FIFO
port map(data_channel_to_rx, data_FIFO_to_rx, strobe_rx, rx_clk, read_en, fifo_reset);

decoder_rx: decoder
   port map(decoder_out_rx, backplane, display_rx);

decoder_tx: decoder
   port map(decoder_out_tx, backplane, display_tx);

clock_divider: clock_div
port map(clk, greset, backplane);

LCD_DISPLAY: LCD_COLON
port map(backplane, dp, colon);

-- Architecture section:
--
-- DCM Instantiation
U_DCM_1: DCM
port map (  
   CLKIN => tx_gclk , -- insert clock input  
   CLKFB => CLK1X_W_1, -- insert clock feedback  
   DSSEN => open_pin , -- Spread spectrum enable input  
)
PSINCDEC => open_pin, -- Phase shifting - increment/decrement input
PSEN => open_pin, -- Phase shifting - enable input
PSCLK => open_pin, -- Phase shifting - clock input
RST => reset_DCM, -- DCM reset input
CLK0 => CLK0_W_1, -- clock output
--CLK90 =>, -- clock output
--CLK180 =>, -- clock output
--CLK270 =>, -- clock output
--CLK2X =>, -- clock output
--CLK2X180 =>, -- clock output
--CLKD =>, -- clock output
CLKFX => CLKFX_W_1, -- clock output
--CLKFX180 =>, -- clock output
--LOCKED =>, -- Locked signal
--PSDONE =>, -- Phase shifting done output
--STATUS => -- Status bus output
);

U_DCM_2: DCM
port map (  
  CLKIN => rx_gclk, -- insert clock input  
  CLKFB => CLK1X_W_2, -- insert clock feedback  
  DSSEN => open_pin, -- Spread spectrum enable input  
  PSINCDEC => open_pin, -- Phase shifting - increment/decrement input  
  PSEN => open_pin, -- Phase shifting - enable input  
  PSCLK => open_pin, -- Phase shifting - clock input  
  RST => reset_DCM, -- DCM reset input  
  CLK0 => CLK0_W_2, -- clock output  
  --CLK90 =>, -- clock output  
  --CLK180 =>, -- clock output  
  --CLK270 =>, -- clock output  
  --CLK2X =>, -- clock output

- 80 -
--CLK2X180 => , -- clock output
--CLKDV => , -- clock output
CLKFX =>CLKFX_W_2 -- clock output
--CLKFX180 => , -- clock output
--LOCKED => , -- Locked signal
--PSDONE => , -- Phase shifting done output
--STATUS => -- Status bus output
);

-- BUFG Instantiation for CLK0
U0_BUFG: BUFG
  port map (  
    I => CLK0_W_1,
    O => CLK1X_W_1
  );
--

-- BUFG Instantiation for CLKFX
U1_BUFG: BUFG
  port map (  
    I =>CLKFX_W_1 ,
    O => tx_clk
  );
--

-- BUFG Instantiation for CLK0
U2_BUFG: BUFG
  port map (  
    I => CLK0_W_2,
    O => CLK1X_W_2
  );
-- 81 --
-- BUFG Instantiation for CLKFX
U4_BUFG: BUFG
  port map (  
    I  => CLKFX_W_2,  
    O => rx_clk  
  );

--used for configuration of DCM;

open_pin<='0';
--Since the reset signal of DCM is active high,  
--and the pushbutton is active low, an inverter is used;  
reset_DCM<=not dcm_reset;

--reset synchronization process;
reset_sync: process (greset, clk)
begin
  if clk'event and clk='0' then
    if greset='0' then
      reset<=0;
    else reset<=1;
    end if;
  end if;
end process;

--process to provide transfer data for the transmitter;
tx_reg_address_process: process(reset,clk)
    begin
        if clk'event and clk='1'
            then if reset='1'
                then tx_reg_address<="00";
                    data_preload<="00000000";
                else
                    if tx_reg_address<="11"
                        then tx_reg_address<=tx_reg_address+1;
                    end if;
                    case tx_reg_address is
                        when "00"=> data_preload<="00000011";
                        when "01"=> data_preload<="00001111";
                        when "10"=> data_preload<="00111111";
                        when "11"=> data_preload<="00111111";
                        when others=>data_preload<="00000000";
                            tx_reg_address<="00";
                    end case;
            end if;
        end if;
    end process;

    --process to change offset value according to the action on pushbuttons
    offset_process: process(dcm_reset, push_botton_off)
    begin
        if dcm_reset='0' then
            offset<="100";
        elsif push_botton_off'event and push_botton_off='0' then
            if offset="111"
                then offset<="000";
            else offset<=offset+1;
        end if;
    end process;
9. Implementation system model use real data link

library IEEE;
--library SIMPRIM;
library UNISIM;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
--use SIMPRIM.all;
use UNISIM.all;

entity testbench is
port( greset, clk: in std_logic;
----------global reset signal, global clock;
dcm_reset: in std_logic;
----------reset singal for DCMs;
push_botton tx, push_botton_rx: in std_logic;
----------push bottons to change data ;
push_botton_reg: in std_logic;
----------push botton to display which reg is on the LCD;

end if;
end if;
end process;

clk_inside_tx<=tx_clk;
clk_inside_rx<=rx_clk;
LCD_COM(1)<=BACKPLANE;
LCD_COM(0)<=BACKPLANE;
end;

end;
push_button_off: in std_logic;
-------------push button to adjust the value of offset;
dp: out std_logic_vector(2 downto 0);
-------------signal for LCD display;
colon: out std_logic;
-------------Signal for LCD display;
LCD_COM: out STD_LOGIC VECTOR(1 DOWNTO 0);
-------------signal for LCD display;
clk_inside_tx: out std_logic;
-------------signal for detecting local signal by oscilloscope;
clk_inside_rx: out std_logic;
-------------signal for detecting local signal by oscilloscope;
display_tx, display_rx: out std_logic_vector(13 downto 0)
-------------data bus for LCD display;
);
end testbench;

architecture behav of testbench is
----local clock for transmitter and receiver;
signal tx_clk, rx_clk: std_logic;

----read enable signal for FIFO;
signal read_en: std_logic;

----data strobe;
signal tx_strobe: std_logic;

----data bus from transmitter to FIFO;
signal data_tx_to_channel: std_logic_vector(7 downto 0);

----data bus from channel to receiver;
signal data_channel_to_rx: std_logic_vector(7 downto 0);

----address for the buffer in transmitter and receiver;
signal tx_reg_address, rx_reg_address: std_logic_vector(1 downto 0);

----buffer to store offset value;
signal offset: std_logic_vector(2 downto 0);

----data bus for transmitter to read data from external devices, and bus for LCD display;
signal data_preload, data_to_display: std_logic_vector(7 downto 0);

signal out_address: std_logic_vector(1 downto 0);

----address for LCD display;
signal tx_reg_out: std_logic_vector(7 downto 0);

----data bus from transmitter to LCD;
signal open_pin: std_logic; ----used for DCM;
signal reset_DCM: std_logic; ----used for DCM reset;

--signals used for DCM connection.
signal CLK0_W_1: std_logic;
signal CLK1X_W_1: std_logic;
signal CLKFX_W_1: std_logic;
signal CLK0_W_2: std_logic;
signal CLK1X_W_2: std_logic;
signal CLKFX_W_2: std_logic;

----clock for LCD;
signal backplane: std_logic;

--signals for decoder.
signal decoder_out_tx: std_logic_vector(7 downto 0);
signal decoder_out_rx: std_logic_vector(7 downto 0);
--reset signal after the first synchronization
signal reset : std_logic;

signal fifo_reset: std_logic;
signal data_channel_to_fifo: std_logic_vector(7 downto 0);

--output signals of the repeaters on the data path;
signal data_tx_to_channel_inv_1: std_logic_vector(7 downto 0);
signal data_tx_to_channel_inv_2: std_logic_vector(7 downto 0);

--output signals of the repeaters on the data strobe;
signal tx_strobe_inv_1: std_logic;
signal tx_strobe_inv_2: std_logic;
signal tx_strobe_inv_3: std_logic;

--output signals of the repeaters on the clock routings;
signal clk_inv_1, clk_inv_2: std_logic;
signal clk_inv_3, clk_inv_4: std_logic;

--output signals of the repeaters on the reset routings.
signal greset_inv_1, greset_inv_2, greset_inv_3, greset_inv_4: std_logic;

component Transmitter
port(clk : in std_logic;
greset, tx_clk: in std_logic;
data_preload: in std_logic_vector(7 downto 0);
reg_address: in std_logic_vector(1 downto 0);
data_tx_to_channel: out std_logic_vector(7 downto 0);
tx_strobe: out std_logic;
reg_out: out std_logic_vector(7 downto 0);
out_address: in std_logic_vector(1 downto 0)
);
end component;
component Receiver
port (clk, greset, tx_strobe, rx_clk: in std_logic;
    offset       : in std_logic_vector(2 downto 0);
    data_channel_to_rx : in std_logic_vector(7 downto 0);
    reg_address  : in std_logic_vector(1 downto 0);
    read_en      : out std_logic;
    fifo_reset   : out std_logic;
    data_to_display : out std_logic_vector(7 downto 0)
);
end component;

component FIFO
port(
data_in: in std_logic_vector(7 downto 0);
data_out: out std_logic_vector(7 downto 0);
--out_address: out std_logic_vector(2 downto 0);
clk_in: in std_logic;
clk_out: in std_logic;
read_en: in std_logic;
reset: in std_logic
);
end component;
component decoder
port (data_in: in std_logic_vector(7 downto 0);
    clk: in std_logic;
    data_out: out std_logic_vector(13 downto 0)
);
end component;
end component;

component BUFG
    port ( I  : in std_logic; 
           O  : out std_logic
    );
end component;

component LCD_COLON
    port(clk: in std_logic; 
         DP: OUT STD_LOGIC_VECTOR(2 DOWNTO 0); 
         COLON: OUT STD_LOGIC
    );
end component;

component clock_div
    port(clk_in: in std_logic; 
         reset: in std_logic; 
         clk_out:out std_logic);
end component;

component inverter
    port(data_in: in std_logic_vector(7 downto 0); 
         data_out: out std_logic_vector(7 downto 0); 
         clk_in: in std_logic; 
         clk_out:out std_logic);
end component;
component inverter_reset
port(
    clk_in: in std_logic;
    clk_out:out std_logic);
end component;

component DCM
-- pragma translate_off
-- For the list of all attributes, see Virtex-II Handbook - Chapter 2
  generic (
    DLL_FREQUENCY_MODE : string := "LOW";
    CLKFX_MULTIPLY : integer := 8;
    CLKFX_DIVIDE : integer := 1;
    DUTY_CYCLE_CORRECTION : boolean := TRUE;
    STARTUP_WAIT : boolean := FALSE
  );
-- pragma translate_on
  port (CLKIN : in std_logic;
         CLKFB : in std_logic;
         DSSEN : in std_logic;
         PSINCDEC : in std_logic;
         PSEN : in std_logic;
         PSCLK : in std_logic;
         RST : in std_logic;
         CLK0 : out std_logic;
         CLK90 : out std_logic;
         CLK180 : out std_logic;
         CLK270 : out std_logic;
         CLK2X : out std_logic;
         CLK2X180 : out std_logic;
         CLKDV : out std_logic;
         CLKFX : out std_logic;
CLKFX180 : out std_logic;
LOCKED : out std_logic;
PSDONE : out std_logic;
STATUS : out std_logic_vector(7 downto 0);
);
end component;
--
--initialization of the DCMs
attribute maxdelay: string;
attribute keep: string;
attribute DLL_FREQUENCY_MODE : string;
attribute DUTY_CYCLE_CORRECTION : string;
attribute CLKOUT_PHASE_SHIFT : string;
attribute PHASE_SHIFT : integer;
attribute CLKFX_MULTIPLY : integer;
attribute CLKFX_DIVIDE : integer;
attribute STARTUP_WAIT : string;
attribute CLKin_PERIOD : string;

attribute DLL_FREQUENCY_MODE of U_DCM_1: label is "LOW";
attribute DUTY_CYCLE_CORRECTION of U_DCM_1: label is "TRUE";
attribute CLKin_PERIOD OF U_DCM_1: label is "TRUE";
attribute PHASE_SHIFT of U_DCM_1: label is "40";
attribute CLKFX_MULTIPLY of U_DCM_1: label is 0;
attribute CLKFX_DIVIDE of U_DCM_1: label is 8;
attribute STARTUP_WAIT of U_DCM_1: label is "FALSE";

attribute DLL_FREQUENCY_MODE of U_DCM_2: label is "LOW";
attribute DUTY_CYCLE_CORRECTION of U_DCM_2: label is "TRUE";
attribute CLKin_PERIOD OF U_DCM_2: label is "40";
attribute PHASE_SHIFT of U_DCM_2: label is 0;
attribute CLKFX_MULTIPLY of U_DCM_2: label is 8;
attribute CLKFX_DIVIDE of U_DCM_2: label is 1;
attribute STARTUP_WAIT of U_DCM_2: label is "FALSE";

-- to prevent unnecessary optimization of the synthesis tool.
-- but using attribute "keep" the signals will not be optimized off;
attribute keep of tx_strobe_inv_1: signal is "true";
attribute keep of data_tx_to_channel: signal is "true";
attribute keep of tx_strobe: signal is "true";
attribute keep of data_tx_to_channel_inv_1: signal is "true";
attribute keep of greset: signal is "true";
attribute keep of greset_inv_1: signal is "true";
attribute keep of greset_inv_2: signal is "true";
attribute keep of greset_inv_3: signal is "true";
attribute keep of greset_inv_4: signal is "true";
attribute keep of reset: signal is "true";
attribute keep of clk_inv_1: signal is "true";
attribute keep of clk_inv_2: signal is "true";
attribute keep of clk_inv_3: signal is "true";
attribute keep of clk_inv_4: signal is "true";

begin
transmitter_component: Transmitter

port map(clk_inv_4, greset_inv_4, tx_clk, data_preload, tx_reg_address,
data_tx_to_channel, tx_strobe, tx_reg_out, out_address);
receiver_component : Receiver

port map(clk_inv_2, greset_inv_2, tx_strobe, rx_clk, offset,
data_channel_to_rx, rx_reg_address, read_en, fifo_reset,
data_to_display);
FIFO_U1: FIFO  
port map(data_channel_to_fifo, data_channel_to_rx, tx_strobe_inv_2,  
rx_clk, read_en, fifo_reset);

decoder_rx: decoder  
port map(decoder_out_rx, backplane, display_rx);

decoder_tx: decoder  
port map(decoder_out_tx, backplane, display_tx);

clock_divider: clock_div  
port map(clk, greset, backplane);

-- inverters on the data path.
inverter_u1: inverter  
port map(data_tx_to_channel, data_tx_to_channel_inv_1, tx_strobe,  
tx_strobe_inv_1);

inverter_u2: inverter  
port map(data_tx_to_channel_inv_1, data_channel_to_fifo,  
tx_strobe_inv_1, tx_strobe_inv_2);

-- repeaters on the reset signal path;
inverter_reset_u1: inverter_reset  
port map(reset, greset_inv_1);

inverter_reset_u2: inverter_reset  
port map(greset_inv_1, greset_inv_2);

inverter_reset_u3: inverter_reset  
port map(greset_inv_2, greset_inv_3);

inverter_reset_u4: inverter_reset
port map(greset_inv_3, greset_inv_4);

-- repeaters on the clock signal path;
inverter_clk_u1: inverter_reset
port map(clk, clk_inv_1);

inverter_clk_u2: inverter_reset
port map(clk_inv_1, clk_inv_2);

inverter_clk_u3: inverter_reset
port map(clk_inv_2, clk_inv_3);

inverter_clk_u4: inverter_reset
port map(clk_inv_3, clk_inv_4);

LCD_DISPLAY: LCD_COLON
port map(backplane,dp,colon);

-- Architecture section:
--
-- DCM Instantiation
U_DCM_1: DCM

port map ( 
  CLKin => clk , -- insert clock input  
  CLKFB => CLK1X_W_1, -- insert clock feedback  
  DSSEN => open_pin , -- Spread spectrum enable input  
  PSINCDEC => open_pin , -- Phase shifting - increment/decrement input  
  PSEN  => open_pin , -- Phase shifting - enable input  
  PSCLK => open_pin , -- Phase shifting - clock input  
  RST   => reset_DCM, -- DCM reset input  
  CLK0 => CLK0_W_1 , -- clock output  
  --CLK90 => , -- clock output  
  --CLK180 => , -- clock output  
)
U_DCM_2: DCM

port map (
  CLKIN => clk, -- insert clock input
  CLKFB => CLK1X_W_2, -- insert clock feedback
  DSSEN => open_pin, -- Spread spectrum enable input
  PSINCDEC => open_pin, -- Phase shifting - increment/decrement input
  PSEN => open_pin, -- Phase shifting - enable input
  PSCLK => open_pin, -- Phase shifting - clock input
  RST => reset_DCM, -- DCM reset input
  CLK0 => CLK0_W_2, -- clock output
  --CLK90 => , -- clock output
  --CLK180 => , -- clock output
  --CLK270 => , -- clock output
  --CLK2X => , -- clock output
  --CLK2X180 => , -- clock output
  --CLKDV => , -- clock output
  CLKFX => CLKFX_W_2, -- clock output
  --CLKFX180 => , -- clock output
  --LOCKED => , -- Locked signal
  --PSDONE => , -- Phase shifting done output
  --STATUS => -- Status bus output
);
-- BUFG Instantiation for CLK0 of DCM_1
U0_BUFG: BUFG
    port map (
        I => CLK0_W_1,
        O => CLK1X_W_1
    );
--

-- BUFG Instantiation for CLKFX of DCM_1
U1_BUFG: BUFG
    port map (    
        I => CLKFX_W_1 ,
        O => tx_clk
    );
--

-- BUFG Instantiation for CLK0 of DCM_2
U2_BUFG: BUFG
    port map (    
        I => CLK0_W_2,
        O => CLK1X_W_2
    );
--

-- BUFG Instantiation for CLKFX of DCM_2
U4_BUFG: BUFG
    port map (    
        I => CLKFX_W_2,
        O => rx_clk
    );
--

--reset synchronization process;
reset_sync: process (greset, clk)
begin
if clk'event and clk='0' then
    if greset='1' then
        reset<='1';
    else reset<='0';
    end if;
end if;
end process;

--process to provide transfer data for the transmitter;

--process to provide transfer data for the transmitter;
tx_reg_address_process: process(reset,clk)
begin
    if clk'event and clk='1'
        then if reset='1'
            then tx_reg_address<="00";
                data_preload<="00000000";
            else
                if tx_reg_address<="11"
                    then tx_reg_address<=tx_reg_address+1;
                    end if;
                case tx_reg_address is
                 when "00"=> data_preload<="00000011";
                 when "01"=> data_preload<="00001111";
                 when "10"=> data_preload<="00111111";
                 when others => data_preload<="00111111";
                end case;
        end if;
    end if;
end process;
when "11"=> data_preload="00111111";
when others=>data_preload="00111111";
    tx_reg_address="00";
end case;

    end if;
    end if;
end process;

--process to change offset value according to the aciton on pushbuttons

offset_process: process(dcm_reset, push_botton_off)
begin
    if dcm_reset='0' then
        offset="100";
    elsif push_botton_off'event and push_botton_off='0' then
        if offset="111" then
            offset="000";
        else
            offset<=offset+1;
        end if;
    end if;
end if;
end process;

--process to change display data on the LCD;

rx_debug_process: process(reset, push_botton_rx)
begin
    if reset='0'
        then rx_reg_address="00";
    elsif push_botton_rx'event and push_botton_rx='0'
        then if rx_reg_address="11" then
            rx_reg_address="00";
        else
            --
            - 98 -
rx_reg_address<=rx_reg_address+1;
end if;
end if;
end process;

--process to change display data on the LCD;
tx_debug_process: process(reset, push_botton_tx)
begin
if reset='0'
then out_address<="00";
elsif push_botton_tx'event and push_botton_tx='0'
then if out_address="11" then
out_address<="00";
else out_address<=out_address+1;
end if;
end if;
end process;

--used to change the display on the LCD according to the actions of pushbuttons.
--when the push_button_reg is pushed down, the LCD will display the data address of current data which is displayed on the LCD.
decoder_out_tx(7 downto 3)<=tx_reg_out(7 downto 3) when reset ='1' and
push_botton_reg='1' else
"00000";
decoder_out_tx(2 downto 0)<=offset when reset ='0' else
'0'&out_address(1 downto 0) when
push_botton_reg='0'
else tx_reg_out(2 downto 0);

decoder_out_rx(7 downto 2)="000000" when push_botton_reg='0' else data_to_display(7 downto 2);
decoder_out_rx(1 downto 0)<=
    rx_reg_address when push_botton_reg='0'
    else data_to_display(1 downto 0);

--used for DCM configuration
open_pin<='0';
reset_DCM<=not dcm_reset;

clk_inside_tx<=tx_clk;
clk_inside_rx<=rx_clk;

--used for LCD display
LCD_COM(1)<=BACKPLANE;
LCD_COM(0)<=BACKPLANE;

end;
Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekomersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsmann i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/