FPGA Based Frequency Measurement for The Purpose of Synchronization

Communications Laboratory – University of Kassel

Project Work
by
Moneer Kaid Al-Bokhaiti (28246752)

October 14th, 2010

Supervisor: Dipl.–Ing. Thomas Edlich
Declaration

I declare that the present Project Report was made by myself with use of the named aids.

Signature

-------------------------------------------------------------
Kassel - October 14th, 2010
Abstract

Oscillators’ drifting is one of the impacts that affect the communication systems performance. Recalibrating these oscillators to work again at the correct frequency is an important task. In our project we will try to detect the frequency difference between two oscillators of 10 MHz one of them works as a master and the second one works as a slave. Each oscillator drifts slowly either to the positive direction or the negative direction. We will develop a measurement technique utilizes the advanced facilities of the Field Programmable Gate Array (FPGA) to measure the frequency difference with an accuracy of 40 mHz at the aforementioned oscillators frequency. Before describe the calculation algorithm a small description about the FPGA and its different resources are introduced. We will measure the frequency of each oscillator using a hybrid method. This hybrid method counts the complete cycles of the oscillator signal by counting the positive edge of each cycle. Whereas the incomplete cycles (the first and the last cycles) are measured by aid of internal clock with a frequency equal to 250 MHz. All the necessary steps which are required to program the device are introduced.
Table of Contents

1. Introduction ................................................................................................................. 1
   1.1 FPGA in Brief ........................................................................................................... 1
   1.2 FPGA Resource Overview ....................................................................................... 1
   1.3 FPGA Development Tools ..................................................................................... 3

2. FPGA Basic Internal Structure ..................................................................................... 4
   2.1 Configurable Logic Block ....................................................................................... 4
      2.1.1 Look-Up Table ................................................................................................. 6
      2.1.2 Storage Elements ............................................................................................ 6
      2.1.3 Multiplexers .................................................................................................... 7
      2.1.4 Carry Logic ....................................................................................................... 7
      2.1.5 Distributed RAM ............................................................................................. 7
      2.1.6 Shift Registers .................................................................................................. 7
   2.2 I/O Block Architecture ............................................................................................. 7
      2.2.1 Input/Output Block (IOB) ................................................................................. 8
      2.2.2 Input/Output Logic Resources (IOLOGIC) ..................................................... 8
   2.3 Clock Resources ....................................................................................................... 9
   2.4 Other Resources ....................................................................................................... 10

3. Application Description ............................................................................................... 11
   3.1 Frequency Measurement Background .................................................................... 11
   3.2 Hybrid Method ........................................................................................................ 12
   3.3 Software Description ............................................................................................... 13
   3.4 Error Sources .......................................................................................................... 15

4. FPGA Design Flow ....................................................................................................... 17
   4.1 Hardware Used ........................................................................................................ 17
   4.2 Development Tools ............................................................................................... 18
      4.2.1 Xilinx ISE Overview ....................................................................................... 18
   4.3 Design Hierarchy and Its Block Diagram ............................................................... 20
   4.4 FPGA Design Flow Processes .............................................................................. 22
      4.4.1 Design Entry Process ..................................................................................... 22
      4.4.2 Design Synthesis Process .............................................................................. 30
      4.4.3 Design Implementation Process ..................................................................... 31
      4.4.4 Device Programming ..................................................................................... 31

5. Conclusions and Future Work .................................................................................... 35

Bibliography ..................................................................................................................... 36
1. Introduction

1.1 FPGA in Brief

Field Programmable Gate Array (FPGA) is a digital integrated circuit (IC) consists of a large amount of configurable logic block (CLB) islands embedded within a sea of configurable interconnects to connect them to each other and to the input/output (I/O) cells. The designer can use the programmability property of these logic blocks and interconnects to construct a specific design. Internal configurations of the FPGA devices are differing from manufacturer to another manufacturer and also differ a little bit from one series to another series of the same manufacturer. But in general there is a generic architecture for all FPGAs as you see in figure 1.1.

![Generic FPGA Architecture](image)

Many semiconductor manufacturers introduce a wide range of FPGA modules as well as development tools which are used to help the designers in the designing process.

1.2 FPGA Resource Overview

Approximately every FPGA module contains the following basic hard resources:
- CLBs resources.
- Input/output resources.
- Clocking resources.
FPGA Based Frequency Measurement for The Purpose of Synchronization.

- Memory resources.

Many FPGA modules can come with many additional advanced hard resources to increase the efficiency and performance. From these resources:

- Embedded hard microprocessors.
- Digital Signal Processing (DSP) slices.
- Ethernet Media Access Controller (MAC) block.
- Peripheral Component Interconnect (PCI) express block.
- Multi-Gigabit (GTP/GTX) transceivers.

The number of each embedded hardware resource differs from FPGA module to another one to give the designers a wide variety of choices in order to match between the need and price according to the intended market. For example wireless system designers want many DSP slices, but on the other hand the embedded system designers want hard microprocessors, block memories and memory control support and so on. Therefore, the designer should take into account when choosing the device his requirements of resources. Figure 1.2 shows how the hard resources are arranged within the FPGA chip.

![FPGA chip with columns of embedded multipliers and RAM blocks](image)

Figure 1.2: FPGA chip with columns of embedded multipliers and RAM blocks [2].

All of the last mentioned resources are hard parts embedded within the FPGAs. There are other types of resources which are soft parts or what so called “Intellectual Property (IP)”. The IP is a specific design already implemented either by the manufacturer or a third party. This IP uses the hard resources to create a new one with a new function. You can use the IP in your design without the need to redesign it again. As examples in embedded systems “PicoBlaze 8-bit processor” is open sources free IP from Xilinx, “MicroBlaze 32-bit processor” is also from Xilinx, but not for free. As examples in the wireless systems: FIR filters, FFT, IFFT, DUC, DDC and many other IPs that save your design time and increase the time to market. You can see more details about IP properties in the Xilinx IP center website [3].
1.3 FPGA Development Tools

Regarding the design tools, every manufacturer introduces with his product its own design tools. Also there are many design tools from third parties can deal with FPGAs from different manufacturers. These design tools enable the designers to control the project development cycle. Another important thing the tools can introduce is the reduction of time-to-market, because some development tools can help the designer in generating many parts of the design using IP wizards or core wizards.

As examples of design tools introduced by the manufacturers: “Xilinx ISE” from Xilinx, “Quartus II” from Altera, “Libero IDE” from Actel, and “Lattice Diamond” from Lattice semiconductor. As an example from third party is the “Synplify Pro” or “Synplify Premier” from Synopsys. Each of the aforementioned tools come with a free version with basic capabilities, but the non-free versions come with advanced capabilities used for the development of embedded systems, DSP systems or any other advanced systems.

Of course, the efficiency of the development tools and the facilities of the free or non-free tools differ from vendor to another, but finally the decision will depends on several factors like:

- The type of project: logic system, embedded system, DSP system or a collection of them.
- The chosen hardware: from which company and what are its capabilities.
- The cost.
- The introduced support by the vendor: the introduced documentations, tutorials, learning videos and other learning materials either for the development tool or the hardware.
2. FPGA Basic Internal Structure

In this chapter we introduce some details of the basic FPGA resources which are the basement for understanding how we can develop any design with FPGA. The development tools automatically utilize these resources in optimized manner and they give the user the necessary wizards to adjust the different attributes of these resources.

The used board is the “SP605 evaluation board” which is based on “Xilinx Spartan-6 LX45T FPGA”, so we will restrict our description of the internal resources to this device. The differences between different devices are not too much, so this restriction will not affect our understanding of FPGA.

2.1 Configurable Logic Block

The CLBs are the responsible resources for implementing sequential and combinatorial circuits. All CLBs are connected to each other through programmable interconnection and each CLB connect to the programmable interconnection through a switch matrix [see figure 2.1].

![Figure 2.1: CLB array and interconnect channels [4].](image)

CLB consists of two slices without direct connection between them as shown in figure 2.2.
 Spartan-6 FPGA has three types of slices which are SLICEX, SLICEL, and SLICEM. SLICEX occupy 50% of the available slices, whereas each one of the other two slices occupies 25% of the available slices. The two slices of the CLB are of different types. One of them is SLICEX and another one is either SLICEL or SLICEM. Every slice in the Spartan-6 FPGA consists of four 6-bit-input look-up table (LUT) and eight storage elements, but some slices have an additional features don’t exist in other slices. Table 2.1 shows all the features of the three kinds of slices and their availability for each kind.

Table 2.1: Slice Features [4].

<table>
<thead>
<tr>
<th>Feature</th>
<th>SLICEX</th>
<th>SLICEL</th>
<th>SLICEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-Input LUTs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>8 Flip-Flops</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Wide Multiplexers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Carry Logic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distributed RAM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Shift Registers</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2.2 shows the summary of logic resources for each CLB in Spartan-6 devices.

Table 2.2: Logic resources in one CLB [4].

<table>
<thead>
<tr>
<th>Slices</th>
<th>LUTs</th>
<th>Flip-Flops</th>
<th>Arithmetic and Carry chains (2)</th>
<th>Distributed RAM (1)</th>
<th>Shift Registers (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>16</td>
<td>1</td>
<td>256 bits</td>
<td>128 bits</td>
</tr>
</tbody>
</table>

Notes:
1. SLICEM only, SLICEL and SLICEX do not have distributed RAM or shift registers.
2. SLICEM and SLICEL only.
2.1.1 Look-Up Table

The LUT is primarily used to implement the function generators. As you see in figure 2.3 the 6-bit-input LUT in Spartan-6 FPGA is not a real 6-bit-input LUT, but it is a combination of two 5-bit-input LUTs. This LUT can implement 6-bit-input function generator with one output (O6). It can also implement two arbitrary 5-bit-input function generators with two independent outputs (O5) and (O6). SLICEL and SLICEM contain the feature of wide multiplexers which can be used to combine two 6-bit-input LUTs to implement 7-bit-input function generator or combine four 6-bit-input LUTs to implement 8-bit-input function generator.

![Figure 2.3: Spartan-6 FPGA LUT](image)

The concept of implementing a function generator using LUT is simple. Basically the possible output values of the function generator are stored in the Static Random Access Memory (SRAM) cells of the LUT and the inputs of the function generator are used to select the desired SRAM cell and rout it to the output.

Figure 2.4 shows a graphical representation of the internal structure of 3-bits-input LUT. Of course the actual LUT is different, but this is just for the purpose of clarifying the working idea.

![Figure 2.4: A transmission gate-based LUT](image)

2.1.2 Storage Elements

The storage elements of the slice are the flip-flops. In Spartan-6 FPGA each slice has two types of flip-flops. The first type consists of four flip-flops work as a D-type flip-flop or level-sensitive latches, whereas the second one consists of four flip-flops work as a D-type flip-flop.
FPGA Based Frequency Measurement for The Purpose of Synchronization.

2.1.3 Multiplexers
The wide multiplexers feature is available only for the SLICEL and SLICEM. These multiplexers can be used to combine two LUTs or four LUTs in order to implement 7-input or 8-input function generator respectively.

2.1.4 Carry Logic
This feature is also available only for the SLICEL and SLICEM. It is used to help the slice in performing a fast arithmetic addition and subtraction operations.

2.1.5 Distributed RAM
This feature is restricted to the SLICEM only, because this slice has an additional data input and write enable signals to allow the LUTs of this slice to use their SRAM cells as a distributed RAM. So 25% of the available slices can be used as a distributed RAM. RAM cells within the four LUTs of the SLICEM can be configured to create a distributed RAM with different size. Distributed RAM is fast, localized, and ideal for small data buffers, FIFO, or register files.

2.1.6 Shift Registers
The SRAM cells in every LUT of SLICEM can be arranged in a chain to create a shift register as you see in figure 2.5. Each LUT can work as a 32-bit shift register. Two, three or four LUTs can be configured as 64-bit, 96-bit or 128-bit shift register respectively. More than 128-bit shift register can be created by using more than one SLICEM, because each slice contains only four LUTs.

![Figure 2.5: Configuration cells linked in a chain](image)

2.2 I/O Block Architecture

Spartan-6 FPGA can support a wide variety for standards through a programmable high-performance drivers and receivers. The I/O pins of the FPGA are grouped into different banks. Each bank has a certain mixture of standards which are common in reference and output drive voltages.
The I/O tile consists of the following parts: two input/output blocks (IOBs), two input logics (ILOGICs), two output logics (OLOGICs), and two input/output delays (IODELAYs). Figure 2.6 shows in a hierarchy mode the I/O tile parts [5].

2.2.1 Input/Output Block (IOB)
Input/output blocks (IOBs) as shown in figure 2.7 are responsible of driving or receiving signals from the device physical pins through input, output, 3-state SelectIO drivers. Single-ended I/O standards use one IOB, whereas differential I/O standards use the two IOBs of the tile.

2.2.2 Input/Output Logic Resources (IIOLOGIC)
There are two ILOGIC for each I/O tile which are arranged as a master and slave. These master and slave resources can operate independently in the single-ended I/O case or concatenated in the differential I/O case [see figure 2.8].

For more information about the functionality of these ILOGICs, please refer to the Spartan-6 FPGA SelectIO Resources user guide [5].
FPGA Based Frequency Measurement for The Purpose of Synchronization.

2.3 Clock Resources

All the synchronous elements inside FPGA need clock signals for the synchronization purpose. The clock signals come from the outside world to the FPGA through special clock input pins, and then routed through the device by using a clock routing network. The routing network routes the clock signals to the synchronous elements like tree that routs the water to its leaves. Figure 2.9 gives a simplified representation shows how the clock routing network work.

![Figure 2.9: A simple clock tree](image)

In Spartan-6 FPGA we can divide the clocking resources into two parts. The first part is the routing networks and the second is the clock management tile (CMT).

There are two types of clock networks in Spartan-6 FPGA:

- Global clock network.
- I/O regional clock network.

CMT is a combination of digital and analog technology and consists of two digital clock managers (DCMs) and one phase locked loop (PLL).

- DCM: is an advanced clock management system used to solve a variety of common clocking issues like:
  - eliminates clock skew;
  - optionally phase shifts the phase of the output clock;
  - optionally multiply or divide the input clock frequency to synthesize a new clock frequency;
  - produces a clean output clock with 50% duty cycle;
  - jitter filtering;

- PLL: serves as a frequency synthesizer for a wide range of frequencies. Also works in conjunction with the DCMs as a jitter filter for either the internal or external clocks.
2.4 Other Resources

There are also different hard and soft resources introduced by Spartan-6 FPGA. As we said in the introduction, one can choose a suitable FPGA module which satisfies his requirements by looking to the different kinds of resources available for this module. As an example of other different resources available for the Spartan-6 FPGA:

- Block RAM.
- Memory controller block.
- Soft 8-bit PicoBlaze processor.
- Soft 32-bit MicroBlaze processor.
- DSP48A1 slices.
- GTP transceiver.
- Integrated endpoint block for PCI Express design.

More details about resources and IPs of Spartan-6 FPGA can be found in the documentation support on Xilinx website [6].
3. Application Description

In our application we have a master and slave oscillators with the same frequency which is equal to 10 MHz. These oscillators drift slowly with time and after a period of time the frequency difference between them becomes a few, tens or hundreds of milli Hertz. Our application is dedicated to measure this frequency difference for the purpose of synchronization. The accuracy of this measurement is 40 mHz at the aforementioned frequency.

Our problem is divided into three general tasks. The first one is the measurement of each frequency oscillator with a high precision (i.e. measuring the frequency in hertz and milli Hertz). The second task is the calculation of the difference between them. The third task is sending the last measurements to the UART port of the board which is connected to the PC through a USB cable.

3.1 Frequency Measurement Background

In order to measure the frequency we have two general methods. The first method depends on measuring the clock cycles within one second by counting the number of positive edges of the measured clock [see figure 3.1]. This method of measurement is efficient for high frequencies. It introduces a measurement error equals to ± 1 Hz. In the case of small frequencies such as a few Hertz’s signal the error ± 1 Hz will be a significant number in comparison with the measured frequency. So we cannot use this method to measure such small frequencies.

![Figure 3.1: counting the number of cycles within one second.](image)

The second method depends on measuring the period $T$ of one clock cycle of the measured clock signal and then the frequency is simply the reciprocal of this period. One cycle period is measured by the aid of already known clock called internal clock which has much higher frequency than the measured clock frequency. In our design the frequency of the internal clock is equal to 250 MHz. The measured clock is used as a CLOCK ENABLE “CE”. The first positive edge enables the counting process and the second positive edge disables the counting process. The counter counts the internal clock cycles and then the period $T$ is simply the number of counted cycles multiplied by the period $T_i$ of the internal clock [see figure 3.2]. This method of measurement is efficient for very
small frequencies (i.e. in Hertz or mille Hertz). The error with this measurement technique is equal to \( \pm T_i \), so the frequency of the measured clock signal will be \( 1/(T \pm T_i) \). If we try to measure high frequencies in the range of the internal clock with this method then we will get a large error, because the error \( \pm T_i \) represents a significant number in comparison with the period \( T \).

As we see by applying the last two measurement techniques we cannot measure signals of high frequency with high precision to get the milli Hertz part of the frequency. So we will use a hybrid method to measure high frequency signal and at the same time reduce the error from \( \pm 1 \) Hz to \( \pm 40 \) mHz at internal clock frequency equal to 250 MHz and measured clock frequency equal to 10 MHz.

### 3.2 Hybrid Method

The basic idea of the hybrid method is to measure the complete cycles of the measured clock with a separate counter as in the first method and measure the missed incomplete cycles (one before the first complete cycle and the another one after the last complete cycle) by other counters with the aid of internal clock. Let’s call the counter that measure the first incomplete cycle the pre-counter and the counter that measure the last incomplete cycle the post-counter.

In order to implement this idea we need to construct three stages pre-counter stage, counter stage and post-counter stage as shown in figure 3.3. Pre-counter and post counter stages deal with the first and last incomplete cycles respectively. The pre-counter counts the number of internal clock cycles from the beginning of the one second control signal till the first positive edge of the measured clock signal. The post-counter counts the number of internal clock cycles from the last positive edge of the measured clock signal till the end of the one second control signal. In the counter stage the counter counts all the positive edges from the first positive edge till the last positive edge within the current one second.
3.3 Software Description

Now our problem is how we could find the relationship between the counted *internal clock* cycles, which are the *pre* and *post counters* values and one *measured clock* cycle in order to find the exact number of cycles and sub cycles within one second for both inputs “SMA_input_1” and “SMA_input_2”. This is what the software part of the design should do.

The software performs three tasks:

- The first one is the calculation of the “SMA_input_1” clock frequency from the values of the counters groups “sma1_A_counters” or “sma1_B_counters”.
- The second task is the calculation of the “SMA_input_2” clock frequency from the values of the counters groups “sma2_A_counters” or “sma2_B_counters”.
- The third task is calculation of the difference between the last two frequencies.

The algorithm for the first and second task is the same; the difference is only in the sources of input data which are “sma1_A_counters” or “sma1_B_counters” for the first task and “sma2_A_counters” or “sma2_B_counters” for the second task.

Before begin the description of the calculation algorithm let’s take in our mind that all the division operations are integer mode division, because our microcontroller is an 8-bit bus wide and performing real division leads to more complexity in the assembly program.

The following steps describe the calculation algorithm progress:

- **Step #1:** Set $f_i = 250$ MHz as a constant value which is the chosen frequency for the *internal clock* of our design. Also read the idle counter group values (either “A” or “B” according to the “ab_switch” control signal) from the PicoBlaze microcontroller input port and store them in variables as follow.

  1. $N_c$: the number of *measured clock* cycles which are counted by the *counter* within one second.
  2. $N_{pre}$: the number of *internal clock* cycles which are counted by the *pre_counter*.
  3. $N_{post}$: the number of *internal clock* cycles which are counted by the *post_counter*.

- **Step #2:** Calculate the following variables:

  1. $N_i$: The number of the *internal clock* cycles per one measured cycle.

      This is simply given by the following integer mode division:

      $$N_i = \frac{T}{T_i} \approx \frac{f_i}{f_c} \text{ cycles}$$

      Where: $f_c$ is the approximate frequency of the *measured clock* which is counted by the *counter* within one second. It is simply the variable $N_c$. 


The last -integer mode- division gives an error in the calculation process which depends on the drift value. This error will be discussed in the section error sources.

2. \( N_{\text{pre+post}} \): The number of the total counted internal clock cycles which is given by:

\[
N_{\text{pre+post}} = N_{\text{pre}} + N_{\text{post}}
\]

- **Step #3**: Compare \( N_{\text{pre+post}} \) with \( N_i \) to check if the counted internal clock can be map to one complete measured cycle, more than one measured cycle or less than one measured cycle. Adjust \( N_c \) according to this comparison and calculate the remaining internal clock cycles after the mapping process and store them in the variable \( N_{\text{net}} \).

\[
N_{\text{net}} = N_{\text{pre+post}} - N_i \quad \& \quad N_{c,\text{new}} = N_c
\]  
if \(( N_{\text{pre+post}} \geq N_i )\)  \hspace{1cm} 3-a

\[
N_{\text{net}} = N_{\text{pre+post}} \quad \& \quad N_{c,\text{new}} = N_c - 1
\]  
if \(( N_{\text{pre+post}} < N_i )\)  \hspace{1cm} 3-b

Where: \( N_{c,\text{new}} \) is the new value of the counter after modification.

After the last calculation the value of \( N_{\text{net}} \) is always less than \( N_i \).

- **Step #4**: calculate the following variables:

1. \( Y_i \): This is the measured clock cycle part that can be represented by one internal clock cycle.

\[
Y_i = \frac{1}{N_i}
\]  
But our software performs an integer mode division, so we can overcome this problem by change the dividend to “1000” instead of “1” and change the unit to milli measured cycles instead of measured cycle.

\[
Y_i = \frac{1000}{N_i} \text{ milli}
\]  
(5)

2. \( Y_{i,\text{net}} \): This is the total value that can be represented by the net internal clock cycles \( N_{\text{net}} \).

\[
Y_{i,\text{net}} = N_{\text{net}} \times Y_i \text{ milli}
\]  
(6)

The last value of \( Y_{i,\text{net}} \) is measured within one second, so we can say:

\[
f_{\text{mhz}} = Y_{i,\text{net}}/\text{sec.}
\]

Where: \( f_{\text{mhz}} \) is the remaining measured clock frequency value.

*For example:*

If \( f_c = 10 \text{ MHz} \) and \( f_i = 250 \text{ MHz} \).
then \( N_i = 250 \text{ MHz}/10 \text{ MHz} = 25 \) internal cycles.

\[
Y_i = \frac{1000}{N_i} = 1000/25 = 40 \text{ milli}.
\]

\[
f_{\text{mhz}} = 40 \text{ mHz}.
\]
The last calculation algorithm should be carried out for both inputs “SMA_input_1” and “SMA_input_2”. Let’s assume that the master oscillator is connected to the input “SMA_input_1” and the slave oscillator is connected to the input “SMA_input_2”. The difference between these two frequencies is given by:

\[ f_{\text{diff}} = f_{\text{master}} - f_{\text{slave}} \quad (7) \]

Finally the master and slave oscillators’ frequencies and the difference between them are sent to the UART port. The frequency is sent as two variables with a dot character between them as follow:

\[ \text{SMA input} = f_c \cdot f_{\text{mHz}} \text{ Hz} \]

But before send the result to the UART port we should first convert each variable from binary to binary coded decimal (BCD) and then to American standard code for information interchange (ASCII). After that we can send the ASCII characters to the output port.

### 3.4 Error Sources

The last measurement technique has three sources of errors as follow:

1. Error #1: this error comes from the pre-counter if the first positive edge of the measured clock signal comes after the positive edge of the accompanying internal clock cycle as shown in figure 3.4. So error #1 means from zero to one internal clock cycle be added by the pre-counter.

2. Error #2: this error comes from the post-counter if the last positive edge of the measured clock signal comes after the positive edge of the accompanying internal clock cycle as shown in figure 3.5. So error #2 means from zero to one internal clock cycle be lost by the post-counter.
FPGA Based Frequency Measurement for The Purpose of Synchronization.

3. Error #3: this error comes from the calculation of the number of internal clock cycles per one cycle of the measured clock $N_i$ [see eq. 1] due to the integer mode division which is used in step #2 to calculate $N_i$. If the relation between the division operands $f_i$ and $f_a$ is an integer relation then there is no error, but if the relation between them is not an integer relation then an error of ±1 will occur. This error leads to an error in the calculation of $N_{net}$ [see eq. 3] and $N_{rem}$ [see eq. 5].

The following equation is used to evaluate the value of error #3 if $N_{pre+post} \geq N_i$:

$$N_{rem-err#3} = \pm \left\{ \left[ \frac{N_{net}}{N_i} \right] - \left[ \frac{N_{net} \pm 1}{(N_i \pm 1)} \right] \right\}$$  \hspace{1cm} (8)

The following equation used to evaluate the value of error #3 if $N_{pre+post} < N_i$:

$$N_{rem-err#3} = \pm \left\{ \left[ \frac{N_{net}}{N_i} \right] - \left[ \frac{N_{net}}{(N_i \pm 1)} \right] \right\}$$  \hspace{1cm} (9)

To calculate the total error due to the last three error sources just replace every $N_{net}$ with $N_{net} \pm 1$ (+1 means error #1 and -1 means error #2). The final maximum error relation will be:

If $N_{pre+post} \geq N_i$:

$$N_{rem-err} = \pm \left\{ \left[ \frac{N_{net} \pm 1}{N_i} \right] - \left[ \frac{N_{net} \pm 2}{(N_i \pm 1)} \right] \right\}$$  \hspace{1cm} (10)

if $N_{pre+post} < N_i$:

$$N_{rem-err} = \pm \left\{ \left[ \frac{N_{net} \pm 1}{N_i} \right] - \left[ \frac{N_{net} \pm 1}{(N_i \pm 1)} \right] \right\}$$  \hspace{1cm} (11)

Figure 3.6 shows the maximum error for the range of frequencies that can be measured with our design. The error increases when we increase the measured frequency, because the number of internal clock cycles per one measured cycle will reduce.

![Figure 3.6](image-url)

Figure 3.6: measurement error for different frequencies can be measured with our design.
4. FPGA Design Flow

Design flow is the required processes for loading any digital electronic system into FPGA device without any mistakes that can change the functionality of the system [see figure 4.1]. The following processes are necessary for any digital system we want to load it into the FPGA device:
- Design entry.
- Design synthesis.
- Design implementation.
- Device programming.

Figure 4.1: FPGA design flow [7].

In this chapter we will learn the function of each process in conjunction with the required practical steps to implement it, but before doing that we should take a look on the hardware and development tool which are used for this task as well as the hierarchy of our design.

4.1 Hardware Used

Our design is based on the evaluation kit SP605 which is designed by Xilinx Corporation. The SP605 evaluation kit comes with different parts to provide the designer all the ability to begin the design process without the need for additional parts. Xilinx introduces a lot of documentations, reference designs, demos and other files to help the designer to getting started with the evaluation kit contents [6].

The kit SP605 includes:
- SP605 Base Board with the XC6LX45T-FGG484-3 CES FPGA.
FPGA Based Frequency Measurement for The Purpose of Synchronization.

- ISE design suit logic edition.
- Universal 12 V power supply.
- Cables: 2 USB, 1 Ethernet, 1 DVI-VGA adapter.
- Documentations, reference designs, demos, schematics and PCB files.

Figure 4.2: SP605 Base Board with the XC6LX45T-FGG484-3 CES FPGA [8].

4.2 Development Tools

With the SP605 evaluation kit Xilinx introduces the logic edition of their development tool which is called Integrated Software Environment (ISE). You can also download the free web package version of the ISE design suite from the Xilinx website.

4.2.1 Xilinx ISE Overview

Xilinx ISE is a suite of different programs which enable the designer to carry out the various processes of the design flow as well as give the designer all the flexibility to use FPGA resources. The project navigator interface is the main window of this ISE suite which allows the designer to reach to the different tools, wizards, files and documents needed by the designer to complete his design [see figure 4.3].

The project navigator window consists of three main subwindows, each subwindow can contains more than one panel.

1. Left subwindow consists of Design, Files, and libraries panels. This subwindow allows the designer to access all the source files.
2. Right subwindow is the Workplace window and it is a multi-document interface (MDI) window, so you can open different files or documents at the same time.
3. Bottom subwindow consists of Consol, Error, and Warning panels. This subwindow shows the status, error, and warning messages.

Most of our work will be on the Design panel. This panel consists of two lists:
1) **Source** list: this is the upper list of the **Design** panel which shows the target device and all the source files hierarchically either in the **Implementation** view or **Simulation** view. The **Implementation** option is used to show the source files for the purpose of carrying out the various design flow processes, whereas the **Simulation** option is used to allow the designer to simulate his design using any kind of the simulation tools.

2) **Processes** list: this is the lower list of the **Design** panel which shows all the required processes to load the design into the target FPGA device.

Figure 4.3: ISE project navigator.
4.3 Design Hierarchy and Its Block Diagram

In the next section we will see how we can load the FPGA device with the “Master/Slave oscillators’ frequency difference” application which was described in chapter 3, but at first we should understand the hierarchy of our source files as well as the function of each file as shown in figure 4.4.

Figure 4.4: Source files hierarchy of the Master/Slave oscillators’ frequency difference application.
Figure 4.5: the electronic circuit of the design.
FPGA Based Frequency Measurement for The Purpose of Synchronization.

4.4 FPGA Design Flow Processes

In this section we will learn the practical steps which are required to load the design into the FPGA device, so copy all the VHDL source files from the attached CD to your own directory on the hard disk.

4.4.1 Design Entry Process

In this process the designer should describe his design in a form understandable by the synthesis tool which performs the next process (synthesis process).

The famous methods which are used for this task are:

- Hardware Description Language (HDL) based design entry.
- Schematics based design entry.

Schematic based design entry is better for the designer who wants to deal more with hardware, whereas HDL based design entry is better for complex designs or an algorithmic way based designs. Also there is a possibility to use a combination of the last two methods to describe the design.

Now let’s begin with the design entry practical steps:

a) Create a new project:

1) First open the Xilinx project navigator by Double-click the ISE Project Navigator icon on your desktop.

2) Select the command new project from the file menu to open the New Project Wizard dialog box as shown in figure 4.6.

![Figure 4.6: new project wizard (project name and location).](image-url)
3) Write the name of the project in the field Name and the project location in the field Location.

4) In the top-level source type list box choose the HDL design entry method and click next.

5) The second dialog box in the New Project Wizard allows you to choose the target device and adjust the project properties as you see in figure 4.7.

Choose the same values as in the above figure.

6) Go ahead by clicking the Next button till you reach the final dialog box which is the summary of the created project and then click finish.

b) The second step in the design entry is to add the already implemented source files to our project.

1) Select Project>Add Copy of Source... to open Add Copy of Source dialog box. Brows to your own directory which stores the VHDL source files and select all the VHDL files as in figure 4.8. Click open to add them to the project.
2) The next dialog box shows you the status of the files being added [see figure 4.9]. Click **OK** to finish the process.

![Figure 4.9: add copy of source (status screen).](image)

The added files will be shown in the **Sources** list of the **Design** panel as you see in figure 4.10.

![Figure 4.10: Source list before creating dcm_manager.](image)

c) As you see in figure 4.10 the “dcm_manager” component is still not included in the design, so we will create it using the new source wizard as in the next step.

To create the “dcm_manager” component using the **New Source Wizard**:

1) Select **Project\New Source ...** to open **New Source Wizard** dialog box [see figure 4.11].

2) Choose the **IP (CORE Generator & Architecture Wizard)** source type. Type in the field **name** the value “dcm_manager” and leave the location value as it without change. Click **Next** to continue.
FPGA Based Frequency Measurement for The Purpose of Synchronization.

3) In the tab View by Function choose FPGA Features and Design\Clocking\Clocking Wizard [see figure 4.12].

   Figure 4.12: new source wizard (IP selection).

   Go ahead by clicking Next till you reach the last dialog box and then click finish to open the clocking wizard.

4) The Clocking Wizard consists of six pages used to adjust the attributes of the clock management tile (CMT) of the Spartan-6 FPGA device in order to generate the necessary clocks for our design. The ISE development tool will use automatically the appropriate clocking resources of the Spartan-6 according to the chosen attributes in this wizard.

   In the first page adjust the value of the input frequency to 200 MHz as shown in figure 4.13.

   Figure 4.13: clocking wizards - input frequency frame.
In the second page add two output clocks $CLK_{OUT1} = 50$ MHz and $CLK_{OUT2} = 250$ MHz as shown in figure 4.14.

![Table showing output clock settings](image)

Figure 4.14: clocking wizards - output frequency frame.

In the third page uncheck the $RESET$ and $LOCKED$ options, because we don’t need them in our application [see figure 4.15].

![Reset and locked options](image)

Figure 4.15: clocking wizards (DCM control signals).

No more changes will be done, so go ahead by clicking $Next$ till the last page as it is shown in figure 4.16.

![Clocking wizard interface](image)

Figure 4.16: clocking wizards (IP summary).

The last page of the clocking wizard will give you a summary of the generated core.
Click *Generate* to finish the wizard and create the “dcm_manager” component [see figure 4.17].

![Figure 4.17: Source list after creating dcm_manager.](image)

Now all the source files of the project are created except the user constraint file (UCF). The user constraints are actually part of the design entry, because they tell the development tool about the physical pins planning and the timing constraints that should be taken into account at the implementation process.

User constrains consist of two types of constraints.

1. **I/O constraints**: assign the I/O ports of the design to the physical pins of the FPGA device.
2. **Time constraints**: put constraints on the period, duty cycle, and other timing characteristics for all inputs, outputs, and clocks signals.

First: adjust the I/O constraints as follow:

1) In the *Source* list of the *Design* panel select the main source file “frequency_counter_application.vhd”. In the *Processes* list of the same *Design* panel expand the *User Constraints* and double click on *I/O Pin Planning (PlanAhead) – Post-Synthesis* as it is shown in figure 4.18. Wait till the *PlanAhead* program opens its main window. Close any start up welcome box.

2) Look at the *I/O ports* subwindow [see figure 4.19]. If you don’t see it, you can open it from the menu *Window*. This subwindow shows you all the I/O ports of our design. Currently all the I/O ports are not assigned to the physical pins of the device as you see on the column *Location*. 

![Figure 4.18: Design panel.](image)
FPGA Based Frequency Measurement for The Purpose of Synchronization.

3) To assign a location for every I/O port just click on the required port in the I/O Ports subwindow. The I/O Port Properties subwindow automatically shows the properties of this port [see figure 4.20]. Enter the location value in the field Site. As an example the Site of the port “led(0)” is “D17”. Enter this value and click Apply to accept it.

4) Do the last step for the remaining I/O ports as follow:
   
   led(0) : D17    led(1) : AB4
   led(2) : D21    led(3) : W15
   clk_crystal : K21    sma_input_1 : M20
   sma_input_2 : M19    output_data : B21

   Note: the physical locations of the LEDs, board oscillator, SMA inputs, UART, and all the other components, which are attached to the development board, are listed in the Hardware User Guide of the SP605 evaluation kit [8].

5) Save the changes and close the PlanAhead program.

Second: adjust the time constraints:

1) In the Source list of the Design panel select the main source file “frequency_counter_application.vhd”. In the Processes list of the same Design panel expand the User Constraints and double click on Create Timing Constraints. A new
panel called **Timing Constraints** will be created and the **Timing Constraints** subwindow will show up on the workplace area as you see in figure 4.21.

2) In the **Constraint Type** list click on **Clock Domains** and then double click on **Unconstrained Clocks\clk\_crystal** to open the **Clock Period** dialog box as shown in figure 4.22.

3) Change the **Time** to 5 ns and the **Rising duty cycle** to 50% and then click **OK**.

4) Save the changes and close the **Timing Constraints** subwindow to return to the **Design** panel again.
5) In this step we will add manually two lines of timing constraints. These constraints are required in Spartan-6 programming if you want to drive a non clock loads by a clock buffer. The CLOCK_DEDICATED_ROUTE constraint is enabled by default and this constraint is responsible of forcing the clock buffers to drive the clock loads only, but in our design we want to make the clock buffer drives a non clock loads for the purpose of edge detection. So we have to disable this constraint in order to continue to the implementation process.

Double click on the “frequency_counter_application.ucf” file. Copy the following two lines and past them after the last line of the ucf file:

```
PIN "buffer_clkin_1.O" CLOCK_DEDICATED_ROUTE = FALSE;
PIN "buffer_clkin_2.O" CLOCK_DEDICATED_ROUTE = FALSE;
```

6) Save the new change and close the file.

Now the created UCF file contains the I/O constraints as well as the time constraints which are required for our design.

4.4.2 Design Synthesis Process

The synthesis tool will convert every HDL file to a complete circuit with logical elements. The generated files from this process are called the netlist files.

To synthesis our design:

a) In the Source list of the Design panel select the main source file “frequency_counter_application.vhd”. In the Processes list of the same Design panel double click on Synthesis – XST. If there is no error in your code, then the synthesis program will continue in the synthesis process and at the end generates the synthesis report.

b) You can see the synthesis report [see figure 4.23] by double click on Design Summary/Reports in the list Processes of the Design panel and then click on Design Overview\Detailed Reports\Synthesis Report.
FPGA Based Frequency Measurement for The Purpose of Synchronization.

4.4.3 Design Implementation Process

Design implementation process consists of three sub processes: Translate, Map, and Place & Rout. The last three sub processes are responsible of generating a file contains the whole design circuit divided into logic blocks can be placed in the target FPGA device resources. The inputs of the implementation program are the netlist files and the UCF file.

To implement the design:

a) In the Source list of the Design panel select the main source file “frequency_counter_application.vhd”. In the Processes list of the same Design panel double click on Implement Design.

b) Now we can see the detailed reports of the three sub-levels of the implementation process as we did with the synthesis report.

4.4.4 Device Programming

The final step in design flow is the device programming. In this step the design is converted into a file format accepted by the target FPGA device and then loaded into the FPGA device using special software used for this purpose.

The process of loading the programming file into the target FPGA device is called “FPGA Configuration” [1]. There are different configuration methods that can be used. For our board SP605 which is based on Spartan-6 LX45T there are about four different methods. You can see more details on the “Hardware User Guide” [8].

We will use the JTAG configuration method, because it is the simplest method for the debugging process.

a) The first step of the device programming is generating the programming file which is also called the bit file.
1) In the *Processes* list of the *Design* panel right click on *Generate Programming File* and then choose *Process Properties*. Choose the *Startup Options* category. Set the value of the property *FPGA Start-Up Clock* to “JTAG Clock” as shown in figure 4.24 and click OK.

![Figure 4.24: generate programming file process properties.](image1)

2) Double click on *Generate Programming File* process to generate the bit file.

b) Now we reach the final step which is the loading process of the bit file to the target FPGA device.

1) In the *Processes* list of the *Design* panel expand the *Configure Target Device* category and double click on *Manage Configuration Project (iMPACT)* as shown in figure 4.25.

![Figure 4.25: processes list.](image2)

2) From the *File* menu of the opened iMPACT program choose *New Project*. Click *Yes* if it asks you if you want the system to automatically create and save a project file for you.

3) In the *Welcome to iMPACT* dialog box choose the same values as in figure 4.26. Click OK to continue.
FPGA Based Frequency Measurement for The Purpose of Synchronization.

4) If the board is connected to the PC through the JTAG cable, then the iMPACT program will automatically detect the available connected programming devices. If it asks you to attach the programming files to these devices, then click Yes.

5) Bypass the first device _xc7c64x-6pack_ by clicking Bypass on the brow dialog box. The focus transfer automatically to the second device which is _xc6slx45t_. Move to the project directory and select the bit file “frequencu_counter_application.bit”. Click open to finish.

6) Click No if it asks you to connect it to the SPI flash programming device.
7) Right click on the device *xc6slx45t* and choose Program to load the bit file to the target device. The message *Program Succeeded* indicates that everything is done correctly.
5. Conclusions and Future Work

In my opinion the SPARTAN-6 based SP605 evaluation kit is the most suitable kit for the purpose of learning or developing process. It gives the designer all the ability to work with it without needing for additional parts. Also it has low cost in comparison with the other kits and this is another advantage make it better than the others.

Working with FPGA is not that difficult issue. The very important thing at the learning level is the knowledge of gradient steps for FPGA learning. The first thing for the beginner is the choosing of design entry method which he will continue with it in his professional life. The development tools tutorials introduce most of the development tools features as well as the design flow processes requirements, so it is much helpful to begin with these tutorials. Also Xilinx introduces a lot of learning materials like online videos and lessons as well as online forum which help their customers to work with their product efficiently.

The main task of our project is the frequency measurement with a high resolution. The resolution and also the error of the measurement technique depend on the measurement algorithm and the FPGA device itself. In our case the measurement algorithm gives an error due to the integer mode division, whereas the FPGA device cannot deal with an internal frequency greater than 250 MHz. This limitation will make some constraints when we want to measure higher frequencies. For this reasons our design is much suitable for the range of frequencies from 1 MHz till around 12 MHz.

The measurement resolution at 10 MHz is 40 mHz and the measurement error at this frequency equals to ±40 mHz. The measurement resolution increased and the measurement error decreased when we reduce the measured frequency below 10 MHz and vice versa.

We can develop our design to reduce the measurement error introduces by the pre and post counters by measuring the positive and negative edges at the same time and change our measurement algorithm to deal with both of them instead of one edge. Also we can develop an algorithm for real mode division instead of integer mode division to remove the error caused by the integer mode division.
Bibliography

   http://www.eetimes.com/design/programmable-logic/4014815/All-about-FPGAs.  
   Last access 13/10/2010.


   Last access 13/10/2010.

   Last access 13/10/2010.


   Last access 13/10/2010.

   Last access 13/10/2010.