VHDL IMPLEMENTATION OF CDMA SYSTEM
Chapter 2

LITERATURE SURVEY

The cellular concept originated at Bell Labs in 1947. The first automatic analog cellular system started operation in Japan in 1979 and in the Nordic countries in 1981. The first commercial AMPS wireless cellular system in the United States started in October 1983 in Chicago. Analog cellular service operates on the 800 MHz frequency band, and are based on FDMA (Frequency Division Multiple Access). Within a few years after analog cellular systems were introduced in 1983, it became apparent that higher capacity, more reliable, and lower cost wireless systems were needed to meet booming demand. Predictions were made that system capacity would be saturated by the 1990’s, first in the largest cities and then in other locations. When consumer demand saturates the capacity of a cellular system, there are three ways to expand: move into new spectrum bands, split existing cells into smaller cells, or introduce new technology to make more efficient use of existing bandwidth. Since no new radio spectrum would be available, and splitting cells requires very expensive additional network infrastructure especially in congested areas, new technology seemed to be the best route. In 1988, a Cellular Technology Industry Association (CTIA) subcommittee was established in the United States to identify technology requirements. Cellular service operators and the manufacturing industry worked with CTIA to define a series of specific milestones to be achieved, with the goal of introducing new technology products and services by 1991. The Telecommunications Industry Association (TIA) was asked to create a specification based on these requirements. Many proposals and much debate ensued, with major factions backing Time Division Multiple Access technology and others backing Frequency Division Multiple Access (FDMA). Both technologies evolved from the original AMPS. TDMA Interim Standard 54 (IS-54) was released in early 1991. TDMA equipment was demonstrated and tested in 1991 in Dallas and Sweden. Newer, more comprehensive standards have been released since, including TDMA IS-136 (also called Digital AMPS or D-AMPS), and CDMA IS-95, and the European GSM standard. Each of these has inherent advantages over AMPS technology.
2.1 Multiple access schemes

Multiple access schemes are used to allow many simultaneous users to use the same fixed bandwidth radio spectrum. In any radio system, the bandwidth that is allocated to it is always limited. For mobile phone systems the total bandwidth is typically 50 MHz, which is split in half to provide the forward and reverse links of the system. Sharing of the spectrum is required in order to increase the user capacity of any wireless network. FDMA, TDMA and CDMA are the three major methods of sharing the available bandwidth to multiple users in wireless system. Among these multiple access techniques CDMA provides less interfered and more secured type communication which is of more concern.

2.1.1 Code Division Multiple Access

Code Division Multiple Access (CDMA) is a spread spectrum technique that uses neither frequency channels (FDMA) nor time slots (TDMA). With CDMA, the narrow band message (typically digitized voice data) is multiplied by a large bandwidth signal that is a pseudo random noise code (PN code). All users in a CDMA system use the same frequency band and transmit simultaneously. The transmitted signal is recovered by correlating the received signal with the PN code used by the transmitter. In a CDMA transmitting-receiving system, the principal parts which define the communication performances are code sequences, their lengths and synchronization between the reception and the emission. A good synchronization and a large spreading length reduce enormously the multiple access interference. The CDMA technology was originally developed by the military during World War II. Researchers were spurred into looking at ways of communicating that would be secure and work in the presence of jamming. Some of the properties that have made CDMA useful are:

- Anti-jam and interference rejection
- Information security
- Accurate Ranging

All the users occupy a common RF spectrum simultaneously. This concept can be visualized with time, frequency and code in three axes as shown in the Figure 2.1.

CDMA is based on the core concept of orthogonality and spread spectrum. It is actually a kind of spread spectrum multiple access technique (SSMA). SSMA uses pseudo-noise (PN) sequence to convert a narrowband signal to a wideband noise like signal before transmission. The resulting signal has much greater bandwidth than required by the original message signal.
2.2 Direct Sequence CDMA (DS-CDMA)

The direct spread multiple access technique is more commonly known as CDMA. In CDMA the narrowband message signal is multiplied by a very large bandwidth signal called the spreading signal. This spreading of a narrow band signal using a wideband signal is shown in figure 2.2. The spreading signal is a PN code sequence that has a chip rate 1 which is orders of magnitude greater than the data rate of the message. Each user in CDMA has its own pseudorandom code. The receiver performs a time correlation operation to detect only the specific desired codeword. All other code words appear as noise due to the decorrelation. Besides PN sequences, Walsh codes can also be used to uniquely identify each user and spread narrow band signal.

In practical implementation of CDMA the multiplication between the data and the spreading code is achieved by exclusive-ORing the data with the spreading code. At the receiver the data is recovered by again exclusive-ORing the multiplexed data with the spreading code of the particular user. Thus, exclusive-ORing the information bits with the spreading sequence twice (once prior to transmission and once after reception) reproduces the original information bits. The process of spreading and despreading is shown in figure 2.3

2.3 Multiple Access using CDMA

Spreading can be used as a multiplexing technique by developing a series of orthogonal spreading codes. These codes may be Walsh codes or PN codes. However PN codes are preferred over Walsh codes because of its one additional feature: If any two different
orthogonal spreading codes are exclusive-ORed bit by bit, the resulting series of bits will itself be a PN code. Thus, if a signal is spread using one code and then despread using another orthogonal code, the result will just look like a PN code and will have a power spectral density similar to wideband white Gaussian noise. Consider the system shown in Figure 2.4, with each of the three source pairs employing mutually orthogonal spreading codes to transmit information over a common channel. Source A uses one spreading code (Let us call it Spreading Code A) and transmits the spread spectrum signal shown in the figure. This signal is wideband and, from the viewpoint of the channel and any observer who does not know the code, the signal looks exactly like additive white Gaussian noise. Source B uses a second, orthogonal spreading code (Spreading Code B) and transmits another spread spectrum signal. Source C uses a third, orthogonal spreading code (Spreading Code C) to transmit a message across the channel. The sum of the signals, which has a power spectral density similar to wideband noise, carries across the channel and arrives at the receivers being employed by the three users. Some noise from the channel itself may also be added.

Now consider the receivers. The receiver associated with User B applies Spreading
Code B to the total received signal. This despreads the portion of the signal transmitted from Source B, but leaves all other portions of the received signal with a wideband noise-like power spectral density. Using a narrow bandpass filter, User B may now extract the portion of the signal associated with Source B, with the channel noise and the interference from Users A and C significantly reduced. Similarly, User A and User C may use their respective spreading codes to extract their intended message. The spreading operation minimizes the effects of narrowband interference and channel noise while minimally affecting the desired receiver signal. The despreading concentrates the desired signals power into a bandwidth that is $1/G_p$ times the bandwidth of the spread signal, but despreading does not affect the power spectral density of the interfering signals from the other transmitters or the channel noise. Thus, after despreading and filtering, the SNR increases by a factor of $G_p$ which is called the Processing Gain. This parameter gives an approximate measure of the interference rejection capability of the spread spectrum system. The processing gain is defined as $G_p = B_{ss}/B$ Where B is the original message bandwidth and $B_{ss}$ is the spread spectrum signal bandwidth.

### 2.4 Generation of Spreading Sequences (PN sequences)

An ideal spreading sequence would be random sequences of binary ones and zeros. However, because it is required that transmitter and receiver must have a copy of the random bits stream, a predictable way is needed to generate the same bit stream at transmitter and receiver and yet retain the desirable properties of a random bit stream. The re-
quirement is met by a PN generator. A PN generator will produce a periodic sequence that eventually repeats but that appears to be random. PN sequences are generated by an algorithm using some initial value called seed. The algorithm is deterministic and therefore produces sequence of numbers that are not statistically random. However, if the algorithm is good, the resulting sequences will pass many reasonable tests of randomness. Such numbers are often referred to as pseudorandom numbers or pseudonoise sequence (PN sequences). An important point is that unless the algorithm and the seed are known, it is impractical to predict the sequence. Two important properties of PNs are randomness and unpredictability. The following two criteria are used to validate that a sequence of numbers is random: Uniform distribution: The distribution of numbers in the sequence should be uniform; that is, the frequency of occurrence of each of the numbers should be approximately the same. For a stream of binary digits, only two numbers (1 and 0) are used, following two properties are desired Balance property: In a long sequence the fraction of binary ones should approach 50 percent. Independence: No one value in the sequence can be inferred from the others. In spread spectrum application, the correlation property should be such that if a period of the sequence is compared term by term with any cycle shift of itself, the number of terms that are the same differs from those that are different by at most one.

2.4.1 Linear Feedback Shift Register (LFSR) implementation

The PN generator for spread spectrum is usually implemented as a circuit consisting of exclusive-OR gates and a shift register, called a Linear Feedback Shift Register (LFSR). The LFSR is a string of 1-bit storage devices. Each device has an output line, which indicates the value currently stored, and an input line. At discrete time instants, known as clock times, the value in the storage device is replaced by the value indicated by its input line. The entire LFSR is clocked simultaneously, causing a 1-bit shift along the entire register. The circuit implementation of LFSR contains n registers (flip flops) 1 to (n-1) XOR gates

![Figure 2.5: linear feedback shift register](image)

The linear shift-register sequence is specified by a primitive polynomial \( h(x) \) of degree \( m \),
shown in equation 2.1

\[
h(x) = h_m x^m + h_{m-1} x^{m-1} + h_{m-2} x^{m-2} + \ldots + h_1 x + h_0 \quad (2.1)
\]

The LFSR in figure 2.5 is specified by the generator polynomial \( h(x) = x^4 + x^3 + x + 1 \). The output of an LFSR is periodic with maximum period of \( N = 2^n - 1 \). The all zero sequence occurs only if the initial contents of the LFSR are all zero. A feedback configuration can always be found that gives a period of \( N \); the resulting sequences are called maximal-length or m-sequences. The m-sequences are used in CDMA. For many communication applications, the 0, 1 sequence is changed to a +1 or -1 sequence by representing a binary 1 with +1 and binary 0 with -1. A related function, also important in the spread spectrum context, is the cross correlation function. The cross-correlation between two sources, A and B, is defined as equation 2.2

\[
R_{A,B}(t) = \frac{1}{N} \sum_{k=1}^{N} A_k B_{k-t} \quad (2.2)
\]

The cross correlation between two different m-sequences is low, and this property is useful for CDMA applications because it enables a receiver to discriminate among spread spectrum signals generated by different m-sequences.

### 2.5 CDMA in Mobile Communication Air Interface

A number of different terms are used to refer to CDMA implementations. The original standard spearheaded by Qualcomm was known as IS-95, the IS referring to an Interim Standard of the Telecommunications Industry Association (TIA). IS-95 is often referred to as 2G or second generation cellular. The Qualcomm brand name cdmaOne may also be used to refer to the 2G CDMA standard. After a couple of revisions, IS-95 was superseded by the IS-2000 standard. This standard was introduced to meet some of the criteria laid out in the IMT-2000 specification for 3G, or third generation, cellular. It is also referred to as 1xRTT which simply means "1 times Radio Transmission Technology" and indicates that IS-2000 uses the same 1.25-MHz shared channel as the original IS-95 standard. A related scheme called 3xRTT uses three 1.25-MHz carriers for a 3.75-MHz bandwidth that would allow higher data burst rates for an individual user, but the 3xRTT scheme has not been commercially deployed. More recently, Qualcomm has led the creation of a new CDMA-based technology called 1xEV-DO, or IS-856, which provides the higher packet data transmission rates required by IMT-2000 and desired by wireless network operators. The Qualcomm CDMA system includes highly accurate time signals (usually referenced to a GPS receiver in the cell base station), so cell phone CDMA-based clocks are an increasingly popular type of radio clock for use in computer networks. The main advantage of using CDMA cell phone signals for reference clock purposes is that they
work better inside buildings, thus often eliminating the need to mount a GPS antenna on the outside of a building. Also frequently confused with CDMA is W-CDMA. The CDMA technique is used as the principle of the W-CDMA air interface, and the W-CDMA air interface is used in the global 3G standard UMTS and the Japanese 3G standard FOMA, by NTT DoCoMo and Vodafone; however, the CDMA family of standards (including cdmaOne and CDMA2000) are not compatible with the W-CDMA family of standards.
Chapter 3

IMPLEMENTATION DETAILS

3.1 DESIGN

The four user CDMA system consists of four mobile users who can transmit and receive data independently. For the transmission of data from four users over a common channel, four different 15 bit long mutually orthogonal spreading codes are used. The orthogonal spreading codes are generated by using linear feedback shift register configuration. The entire LFSR is clocked simultaneously that produces a single bit shift along the entire register. Each user data bits are spreaded with the corresponding PN code so as to produce a wide band signal which is modulated and transmitted. At the receiver end the signal is being demodulated and de spreaded with the same orthogonal PN code pair as that of transmitter. As a result each individual user data can be extracted from the wide band signal.

The following section deals with the explanations of transmitter and receiver independently.
3.1.1 Transmitter

The eight bit input data corresponding to a particular user is converted into serial form by an eight bit PISO. The PISO is clocked by Fmaster divided by 15 clock where Fmaster is 0.5GHz. Then it is spreaded by the 15 bit PN code. The PN code generator is clocked by Fmaster. Spreaded data of all the four users are summed up and generated the signal to be transmitted.

Figure 3.1: Transmitter block diagram
3.1.2 Receiver

After de-spreading the received signal with the corresponding code, it is compared with the same PN code, which is converted into parallel, using an 8 bit comparator. The comparator uses 0.33GHz clock frequency. If the actual transmitted data was a high then the de-spread output will be same as that of the PN sequence. So the comparison function is performed in such a way that, it compares the de-spread output with PN sequence. If it is same, then it can be concluded that the data sent is a high and if it is not, then the data will be a low. So the comparator output corresponds to the actual transmitted data of a particular user. Thus it is able to reconstruct the original data from the spreaded output.

Figure 3.2: Receiver block diagram

3.2 INTRODUCTION TO VHDL

VHDL is a language for describing digital electronics systems. It arose out of the US Government’s Very High Speed Integrated Circuits (VHSIC) Program, initiated in
1980. In the course of this program, it became clear that there was a need for a standard language for describing the structure and function of integrated circuits (ICs). Hence the VHSIC hardware description language (VHDL) was developed, and subsequently adopted as a standard by the Institute of Electrical and Electronic Engineers (IEEE) in the US. VHDL is designed to fill a number of needs in the design process. Firstly, it allows description of the structure of a design that is how it is decomposed into sub designs, and how those sub designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before being manufactured so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototype.

3.2.1 VHDL Terms

Before we go any further, let's define some of the terms that are being used. These are the basic VHDL building blocks that are used in almost every description, along with some terms that are redefined in VHDL to mean something different to the average designer.

3.2.1.1 Entity

All designs are expressed in terms of entities. An entity is the most basic building block in a design. The uppermost level of the design is the top-level entity. If the design is hierarchical, then the top-level description will have lower-level descriptions contained in it. These lower-level descriptions will be lower-level entities contained in the top-level entity description.

3.2.1.2 Architecture

All entities that can be simulated have an architecture description. The architecture describes the behaviour of the entity. A single entity can have multiple architectures. One architecture might be behavioural while another might be a structural description of the design.

3.2.1.3 Configuration

A configuration statement is used to bind a component instance to an entity-architecture pair. A configuration can be considered like a parts list for a design. It describes which behaviour to use for each entity, much like a parts list describes which part to use for each part in the design.
3.2.1.4 Package

A package is a collection of commonly used data types and subprograms used in a design. Think of a package as a toolbox that contains tools used to build designs.

3.2.1.5 Driver

This is a source on a signal. If a signal is driven by two sources, then when both sources are active, the signal will have two drivers.

3.3 ALGORITHMS

The following section deals with the algorithm used for the program and explanations of different components used in the structural modeling.

3.3.1 Transmitter

1. Generate master clock having a frequency of 0.5GHz (time period=2ns) and a second clock with frequency 0.03GHz (time period=30ns)

2. Generate orthogonal PN sequences for spreading having a length of 15 bits using linear feedback shift registers. Time shifted versions of a PN sequence will be nearly orthogonal.

3. Provide all the four 8 bit long parallel inputs to four parallel in serial out shift register. The shift registers are provided with a clock of 0.5GHz and a mode control input that controls the shift or load operation of the register.

4. Multiply the serial output of each shift registers with the generated PN sequences which are time shifted versions of one another and hence orthogonal.

5. Add the multiplier outputs of each user and transmit it through the channel.
Figure 3.3 shows the RTL schematic of main entity of transmitter. It has 37 inputs and 13 outputs. Four data inputs are 8 bit wide. "Init1" signal which is 4 bit wide is used for initialising the PN sequence. sl is the mode control for shift or load of the parallel in serial out shift register. opn1 to opn4 will show the output PN sequences and spr1 to spr4 shows the spreaded outputs of each user and "sum" shows the final output that is to be transmitted.

Transmitter RTL schematic is shown in the figure 3.4. It consists of six components. They are clock generator, PN sequence generator, and four parallel in serial out shift registers.
3.3.2 Receiver

1. Generate master clock for the receiver which is having the same frequency that is being used in the transmitter.

2. Generate PN sequences similar to that in the transmitter. The same logic used in the transmitter should be used in the receiver.

3. Using serial to parallel converter, convert the generated PN sequences to parallel.

4. Provide the output from the transmitter to the receiver. Give the received data to the four separate multipliers where the received data is multiplied with the four separate PN sequences.

5. The multiplied output is converted into parallel by serial to parallel converter similar.
6. The output of multiplier is to be given to a comparator in parallel form which compares this parallel data with parallel converted PN sequence.

7. The final output is obtained at the output of comparator.

Figure 3.5: Main Entity of Receiver

Figure 3.5 shows the main entity of the complete receiver which consists of a sum input as well as 4 initialising bits. At the output section there are 4 pn sequences generated, despreaded data as well as the reconstructed data bits od1, od2, od3 and od4. The completel structure contains 6 input pins and 68 output pins.
Figure 3.6: RTL Of Complete Receiver
3.3.3 Clock generator

For the implementation of CDMA system two clocks are needed. A master clock and another with a frequency which is obtained by dividing the master clock by 15. The clocks are generated by using testbench program. Generation of a pulse waveform using VHDL module is a difficult task. So for the generation of clock signals, testbench program is used. So in the program for clock there are two ports for two separate clocks. In the program to generate clock first two signals are set to a value. Then after a particular time it is complemented and repeated. For master clock the wait time is 1 ns, and for next clock it is 15 ns. Hence the time period of the master clock is 2 ns.

![RTL Of Clock generator](image)

3.3.4 PN sequence Generator

Linear feedback shift registers are used for generating PN sequences. Components of D flip flops are used for this since structural modeling is used. To generate the sequence, first it is necessary to initialize the flip flops to a particular value. Since 15 bit long PN sequence is being used, four flip flops are required and these four flip flops are required to be initialized. For that purpose, init signals are used. After the initialization, the xor feedback logic will provide a method to generate a PN sequence. Orthogonal sequences are required in this system. Time shifted versions of a PN sequence will be nearly orthogonal. So to shift the sequences, shift registers are used in which the sequence is given as input to the registers. The outputs from intermediate flip flops are taken which will be time shifted. So at the output of PN generator four PN sequences are obtained.
3.3.5 PISO

Parallel in serial out shift register is made using D-flip flops. The D flip flops are connected serially one by one. A mode control is used for shifting or loading the data into the register. If the mode control is high, each D input of the eight flip flops will be given the input data else if it is low, with each master clock pulse, each Q output is shifted to next flip flop. A serial output will be obtained at the Q output of the eighth flip flop, after eight clock pulses.
3.3.6 Multiplier

The spreading operation is done by multiplication. The PN sequence and the serial converted data are multiplied for every master clock pulse. If the serial data is high, then the PN sequence will be the output of the multiplier, else the output will be zero. Each multiplier output of the four users are to be added. So they are given as input of an OR gate. The output of the OR gate is transmitted. Dispersing is also done in multiplier. The received data is multiplied with PN sequence so as to get the de-spread data.
3.3.7 SIPO

It is also done by using D flip flops. D flip flops are connected in series. The serial data is connected to the D input of the first flip flop. Since each flip flops are connected in series, at each clock pulse the data will be shifted to the next flip flop. So at the Q output of each flip flop the parallel data will be available. These are considered for each pulse of the second clock.

![Diagram of SIPO](image)

Figure 3.10: RTL Of serial in Parallel out shift register

3.3.8 Comparator

The inputs of the comparator will be the parallel converted PN sequence and de spreaded output. For each user the de spreaded data will be similar to the PN sequence if the data sent was logic high. Otherwise a random output will be obtained. So the comparator is designed such that it provides a high output if the de spread output is a PN sequence, otherwise the output will be zero.
3.4 HARDWARE IMPLEMENTATION

3.4.1 Choosing the Device - CPLD

PLAs and PLDs are useful for implementing a wide variety of small digital circuits. Each device can be used to implement circuits that do not require more than the number of inputs, product terms, and outputs that are provided in the particular chip. These chips are limited to fairly modest sizes, typically supporting a combined number of inputs plus outputs of not more than 32. For implementation of circuits that require more inputs and outputs, either multiple PLAs or PALs can be employed or else a more sophisticated type of chip, called a complex programmable logic device (CPLD), can be used. A CPLD comprises multiple circuit blocks on a single chip, with internal wiring resources to connect the circuit blocks.

3.4.2 CoolRunner II

Xilinx CoolRunner-II CPLDs deliver the high speed and ease of use associated with the XC9500/XL/XV CPLD family with the extremely low power versatility of the XPLA3 family in a single CPLD. This means that the exact same parts can be used for high-speed data communications or computing systems and leading edge portable products, with the added benefit of In System Programming. Low power consumption and high-speed operation are combined into a single family that is easy to use and cost effective.

3.4.2.1 Architecture Description

CoolRunner II CPLD is a highly uniform family of fast, low power CPLDs. The underlying architecture is a traditional CPLD architecture combining macrocells into Function Blocks (FBs) interconnected with a global routing matrix, the Xilinx Advanced Interconnect Matrix (AIM). The FBs use a Programmable Logic Array (PLA) configuration which allows all product terms to be routed and shared among any of the macrocells of the FB. Design software can efficiently synthesize and optimize logic that is subsequently fit to the FBs and connected with the ability to utilize a very high percentage of device resources. Design changes are easily and automatically managed the software, which exploits the 100 percentage routability of the programmable Logic Array within each FB. This extremely robust building block delivers the industry’s highest pinout retention, under very broad design conditions. The design software automatically manages these device resources so that users can express their designs using completely generic constructs without knowledge of these architectural details. More advanced users can take advantage of these details to more thoroughly understand the software’s choices and direct its results.

Figure shows the high level architecture whereby FBs attach to pins and interconnect to each other within the internal interconnect matrix. Each FB contains 16 macrocells. The
BSC path is the JTAG Boundary Scan Control path. The BSC and ISP block has the JTAG controller and In-System Programming Circuits.

3.4.2.2 Functional Block

The CoolRunner-II CPLD FBs contain 16 macrocells, with 40 entry sites for signals to arrive for logic creation and connection. The internal logic engine is a 56 product term PLA. All FBs, regardless of the number contained in the device, are identical. At the high level, the product terms (p-terms) reside in a programmable logic array (PLA). This structure is extremely flexible, and very robust when compared to fixed or cascaded product term FBs. Classic CPLDs typically have a few product terms available for a high-speed path to a given macrocell. They rely on capturing unused p-terms from neighboring macrocells to
expand their product term tally, when needed. The result of this architecture is a variable
timing model and the possibility of stranding unusable logic within the FB.
The PLA is different and better. First, any product term can be attached to any OR
gate inside the FB macrocell(s). Second, any logic function can have as many p-terms as
needed attached to it within the FB, to an upper limit of 56. Third, product terms can
be re-used at multiple macrocell OR functions so that within a FB, a particular logical
product need only be created once, but can be re-used up to 16 times within the FB.
Naturally, this plays well with the fitting software, which identifies product terms that
can be shared.
The software places as many of those functions as it can into FBs, so it happens for free.
There is no need to force macrocell functions to be adjacent or any other restriction save
residing in the same FB, which is handled by the software. Functions need not share a
common clock, common set/reset, or common output enable to take full advantage of
the PLA. Also, every product term arrives with the same time delay incurred. There
are no cascade time adders for putting more product terms in the FB. When the FB
product term budget is reached, there is a small interconnect timing penalty to route
signals to another FB to continue creating logic. Xilinx design software handles all this
automatically.

3.4.2.3 Macrocell

The CoolRunner-II CPLD macrocell is extremely efficient and streamlined for logic cre-
ation. Users can develop sum of product (SOP) logic expressions that comprise up to
40 inputs and span 56 product terms within a single function block. The macrocell can
further combine the SOP expression into an XOR gate with another single p-term expres-
The resulting logic expression’s polarity is also selectable. As well, the logic function can be pure combinational or registered, with the storage element operating selectably as a D or T flip-flop, or transparent latch. Available at each macrocell are independent selections of global, function block level or local p-term derived clocks, sets, resets, and output enables. Each macrocell flip-flop is configurable for either single edge or Dual EDGE clocking, providing either double data rate capability or the ability to distribute a slower clock (thereby saving power). For single edge clocking or latching, either clock polarity can be selected per macrocell. When configured as a D-type flip-flop, each macrocell has an optional clock enable signal permitting state hold while a clock runs freely. Note that Control Terms (CT) are available to be shared for key functions within the FB, and are generally used whenever the exact same logic function would be repeatedly created at multiple macrocells. The CT product terms are available for FB clocking (CTC), FB asynchronous set (CTS), FB asynchronous reset (CTR), and FB output enable (CTE).
Figure 3.14: CoolRunner-II CPLD Macrocell
3.5 CPLD IMPLEMENTATION

In the proposed method each user has 8 bit data. For transmitting a data by individual user requires separate input devices that produce an 8 bit binary equivalent for corresponding data, else 8 switches each for the corresponding 8 bit. Also in the receiver section 4 user requires 4 output devices separately that displays the data from the transmitter 8 bit binary data else 8 LEDs for each user to show the status of received binary data.

The device chosen is XC 2C256 belonging to CoolRunner 2 family. And the CPLD burner kit is "VVSM-06 universal VLSI trainer". Due to the unavailability of I/O devices the input and output bits should be reduced in size. Under this circumstance the implementation of 4 user CDMA system is not possible. For implementing the proposed method the input and output bits should be reduced from 8 bit to single bit for each user. The VHDL code for single bit transmitter and receiver was implemented and verified. For burning the single bit transmitter and receiver VHDL code into the CPLD, the program should be made completely error free. After running the program a new source-implementation constraints file- is to be added. It is done as, Project => New source => implementation constraint file. It’s a file with extension ".ucf". Then different input and output signals are to be assigned to different pins. For that an option assign package pin is used. Using this option different inputs are assigned to switches in the kit and outputs are assigned to LED’s in the kit, as shown in figure3.7.
Figure 3.15: Assigning pins - screen shot
Then the program is rerunned. After that the kit is connected to the computer via JTAG cable. The code is burned to the CPLD using iMPACT (configure device) option. In the burning wizard, JTAG is to be selected. The screen will show an image of the chip. It is to be selected and burning is performed. The screen will show "Program Succeeded". The screen shots at different instants of burning the program into the CPLD is shown in the figure 3.8.

![Program Succeeded - screen shot](image-url)

Figure 3.16: Program succeeded - screen shot
Figure 3.17: vvs-universal VLSI Trainer
Chapter 4

RESULTS AND DISCUSSIONS

Successfully implemented and verified the 4 user CDMA system with an 8 bit user data. The transmitted data was efficiently reconstructed at the receiver end. The simulation results are shown in the figure 4.1.

Figure 4.1: Transmitter output

In the figure 4.1; d1,d2,d3, and d4 are the four 8 bit long data and they are converted into serial by the parallel in serial out shift register. The serial data are o1,o2,o3, and o4 and the spreaded the data from four users by using the time shifted versions of a PN sequence with each having length of 15 bits. In the figure d1,d2,d3, and d4 are the four 8 bit long data and they are converted into serial by the parallel in serial out shift register. The serial data are o1,o2,o3, and o4 and the shift or load mode control input is sl. opn1,opn2, opn3 and opn4 are the four PN sequences those are shifted form of one another. Then multiplied each data with particular PN sequences and the corresponding outputs are represented using spr1, spr2, spr3, and spr4. spr1 is obtained by multiplying the data output o1 with the PN sequence opn1. Similarly other three outputs are obtained.
all these four multiplied outputs are added and the final spreaded output data to be transmitted is obtained. sum represents this final output for transmission.

![Figure 4.2: Receiver output](image)

By multiplying the sum with four separate PN sequences, it produces another data sequence. Then passed the newly received data and generated PN sequences through two serial in parallel out shift registers and compared it. The PN sequence used was the same PN sequence that was used in the transmitter section. Theoretically, received data is such that when a high in the transmitter, the PN sequence used to multiply the data will be there in it. So when comparing, if same PN sequence is obtained, then the transmitted data will be a high and if not then it will be a low.
The device chosen is XC 2C256 belonging to coolrunner 2 family. And the CPLD burner kit is "VVSM-06 universal VLSI trainer". Due to the unavailability of I/O devices the input and output bits should be reduced in size. Under this circumstance the implementation of 4 user CDMA system is not possible. For implementing the proposed method the input and output bits should be reduced from 8 bit to single bit for each user. The VHDL code for single bit transmitter and receiver was implemented and verified. The four user CDMA system implemented using single bit input data was also simulated and the output waveform is as shown in the figure 4.3.

Figure 4.3: single bit transmitter and receiver output

The hardware model of the 4 user CDMA system was implemented in coolrunner II CPLD XC 2c256. The data bits were spreaded and summed together and also efficiently decoded the single bit data.
Figure 4.4: Different input combinations
Figure 4.5: Different input combinations
The figure 4.4 and 4.5 show the states of output LEDs and input switches at different input combinations. 8 input switches and 15 output LEDs are used in the demonstration.

(a) Initial setup in which LEDs L2, L3, L4, L5 shows the PN sequences generated and L14, L15 shows clock signal C1 and C2.

(b) The input switches SW1, SW2, SW3, SW4 represent the data input corresponding to each user. Input is equal to '1000' produces the same decoded data at L10, L11, L12, L13. The LED L1 shows the sum and L6, L7, L8, L9 shows the spreaded data corresponding to each user.

Different input combinations

(c) 1100
(d) 1110
(e) 1111
(f) 1001
(g) 0111
(h) 0001
Chapter 5

CONCLUSION AND FUTURE WORK

This project presents a comparatively easier idea to design and implement a CDMA system. The design is done and simulated using VHDL in the Xilinx ISE simulator. The hardware model is implemented in CPLD. In the simulation part the system of 4 users to transmit an 8 bit parallel data is designed. In the hardware section four user systems for transmitting single bit data is implemented. Input data can be given using switches and output is taken out using LEDs. The designed system can be used to implement a secure centralized data transfer in a local network. The transmitter receiver module to use can be RF, Bluetooth, zigbee or any other techniques which can be selected on the basis of range requirement. Also the system can be used to transfer sensed data inside an industry, where the environment is comparatively noise free and insecure. Since the design and hardware can be improved easily with low cost the idea of this project is very much suitable for such situation. The security and capacity of the system can be improved by using the techniques such as dynamic CDMA and code hopping. Thus this system can be easily implemented, improved and used for secure networks.
Chapter 6

REFERENCES


3. B. Sreedevi, V. Vijaya, CH. Kranthi Rekh, Rama Valupadasu, B. RamaRao Chunduri, FPGA implementation of DSSS-CDMA transmitter and receiver for Adhoc Networks. IEEE Symposium on computers and informatics 2011

4. K. Skahill, VHDL for Programmable Logic, Addison -Wesley, 1996

5. XESS Corporation, XSA Board V1.1, V1.2 User Manual-2005

6. D. Vanden Bout, PS/2 Keyboard Interface for the XSA Boards, XESS Corporation, 2004


Chapter 7

APPENDIX

7.1 VHDL CODES

7.1.1 D Flip Flop

library IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity dffp is
Port (clk, clr, pst : in STD_LOGIC;
d : in STD_LOGIC;
q : out STD_LOGIC);
end dffp;
architecture Behavioral of dffp is
begin
process(clk, pst, clr)
begin
if(pst='1') then
q<='1';
elsif(clr='1') then
q<='0';
elsif (clk'event and clk='1') then
q<=d;
end if;
end process;
end Behavioral;

7.1.2 Exor

- Company:
- Engineer:

- Create Date: 20:31:49 11/27/2012
- Design Name:
- Module Name: Exor - Behavioral
- Project Name:
- Target Devices:
- Tool versions:
- Description:

- Dependencies:

- Revision:
- Revision 0.01 - File Created
- Additional Comments: library IEEE
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
— Uncomment the following library declaration if instantiating
any Xilinx primitives in this code.

```vhdl
library UNISIM;
use UNISIM.V Components.all;

entity exor is
  Port ( a : in STD_LOGIC;
         b : in STD_LOGIC;
         z : out STD_LOGIC);
end exor;

architecture Behavioral of exor is
begin
  z<=a xor b;
end Behavioral;
```

### 7.1.3 Clock Generator

- **TestBench Template**

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY testbench IS
  port(pulse1: out std_logic;
       pulse2: out std_logic);
END testbench;
ARCHITECTURE behavior OF testbench IS
  signal clk1: std_logic := '0';
  signal clk2: std_logic := '0';
  BEGIN
    clk1<=not clk1 after 1 ns;
    pulse1<=clk1;
    clk2<=not clk2 after 15 ns;
    pulse2<=clk2;
  END;
```

### 7.1.4 Parallel In Serial Out shift regester

- **Company:**
- **Engineer:**
- **Create Date:** 20:31:49 11/27/2012
- **Design Name:**
- **Module Name:** PISO - Behavioral
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncommen the following library declaration if instan tiati ng
-- any Xilinx primitives in this code.
library UNISIM;
-use UNISIM.VComponents.all;
entity spre is
Port ( clk : in STD_LOGIC;
load : in STD_LOGIC;
data_in : in STD_LOGIC_VECTOR (7 downto 0);
piso_out : out STD_LOGIC);
end spre;
architecture Behavioral of spre is
signal Q: STD_LOGIC_VECTOR (6 downto 0);
begin
process(clk)
begin
if(clk’event and clk=’1’) then
if(load=’1’) then
Q(0)<=data_in(0);
Q(1)<=data_in(1);
Q(2)<=data_in(2);
Q(3)<=data_in(3);
Q(4)<=data_in(4);
Q(5)<=data_in(5);
Q(6)<=data_in(6);
piso_out<=data_in(7);
eelsif(load=’0’) then

end process(clk);
end if(clk’event and clk=’1’);
Q(0)<=data_in(0);
Q(1)<=Q(0);
Q(2)<=Q(1);
Q(3)<=Q(2);
Q(4)<=Q(3);
Q(5)<=Q(4);
Q(6)<=Q(5);
piso_out<=Q(6);
end if;
end if;
end process;
end Behavioral;

7.1.5 Serial In Parallel Out shift register

- Company:
- Engineer:
- Create Date: 12:04:45 12062012
- Design Name:
- Module Name: sipo Behavioral
- Project Name:
- Target Devices:
- Tool versions:
- Description:

- Dependencies:
- 
- Revision:
- Revision 0.01 File Created
- Additional Comments:
- 
---------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
library UNISIM;
entity sipo is
Port (din : in STD_LOGIC;
clk,rst : in STD_LOGIC;
qout : inout STD_LOGIC_VECTOR (14 downto 0));
end sipo;

architecture Behavioral of sipo is

begin
process(clk)
begin
if(rst='1') then
qout<="0000000000000000"
elsif(clk' event and clk='1') then
qout(0)<=din;
for i in 0 to 13 loop
qout(i+1)<=qout(i);
end loop;
end if;
end process;
end Behavioral;

7.1.6 PN Sequence generator

- Company:
- Engineer:

- Create Date: 20:51:59 11272012
- Design Name:
- Module Name: pnsequence - Behavioral
- Project Name:
- Target Devices:
- Tool versions:
- Description:


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;

entity pnsmall_1 is
Port (clock: in STD_LOGIC;
init : in STD_LOGIC_vector (3 downto 0);
pn1,pn2,pn3,pn4 : out STD_LOGIC);
end pnsmall_1;

architecture Behavioral of pnsmall_1 is
component dffp
port(clk,clr,pst,d:in std_logic;
qout std_logic);
end component;
component exor
port(a,b:in std_logic;
z:out std_logic);
end component;
signal q0:std_logic;
signal q1:std_logic;
signal x:std_logic;
signal q2:std_logic;
signal q3:std_logic;
signal n1,n2,n3,n4: std_logic;
signal qout: std_logic_vector(7 downto 0);
beg
s0:dffp port map(clock,'0',init(0),x,q0);
s1:dffp port map(clock,'0',init(1),q0,q1);
s2:dff port map(clock,'0',init(2),q1,q2);
s3:dff port map(clock,'0',init(3),q2,q3);
xx:exor port map(q0,q3,x);

process(clock)
begin
qout(0)<=q3;
if(clock'event and clock='1') then
for i in 0 to 6 loop
qout(i+1)<=qout(i);
end loop;
pn1<=qout(1);
pn2<=qout(3);
pn3<=qout(5);
pn4<=qout(7);
end if;
end process;
end Behavioral;

7.1.7 Spreader for data of one user

Company:
Engineer:
Create Date: 20:51:59 11272012
Design Name:
Module Name: spread1 - Behavioral
Project Name:
Target Devices:
Tool versions:
Description:
Dependencies:
Revision:
Revision 0.01 - File Created
Additional Comments:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
— Uncomment the following library declaration if instantiating
— any Xilinx primitives in this code.
library UNISIM;
—use UNISIM.VComponents.all;

entity newtran is
Port ( d : in STD_LOGIC_VECTOR (7 downto 0);
sl : in STD_LOGIC;
o: inout STD_LOGIC;
initi : in STD_LOGIC_VECTOR (6 downto 0);
spr,opn : out STD_LOGIC);
end newtran;

architecture Behavioral of newtran is
component testbench
p ort(pulse1: out std_logic;
pulse2: out std_logic);
end component;
component spre is
Port ( clk : in STD_LOGIC;
load: in STD_LOGIC;
data_in : in STD_LOGIC_VECTOR (7 downto 0);
piso_out : out STD_LOGIC);
end component;
component pn2 is
Port ( clock : in STD_LOGIC;
init : in STD_LOGIC_VECTOR (6 downto 0);
pn : out STD_LOGIC);
end component;
signal c1:std_logic;
signal c2:std_logic;
signal pno:std_logic;
signal ocl,och:std_logic;
begi n
s1: testbench port map(c1,c2);
s2:spre port map(c2,sl,d,o);
s3:pn2 port map(c1,initi,pno);
och<=c1;
ocl<=c2;
opn<=pno;
process(c1)
begin
if(o='0')then
spr<=pno; else
spr<=not pno;
end if;
end process;
end Behavioral;
7.1.8 Final Transmitter

- Company:
- Engineer:
- Create Date: 23:07:52 03062013
- Design Name:
- Module Name: transmitterfinal Behavioral
- Project Name:
- Target Devices:
- Tool versions:
- Description:
- Dependencies:
- Revision:
- Revision 0.01 File Created
- Additional Comments:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
-library UNISIM;
-use UNISIM.VComponents.all;

entity transmitterfinal is

Port ( d1 : in STD_LOGIC_VECTOR (7 downto 0);
d2 : in STD_LOGIC_VECTOR (7 downto 0);
d3 : in STD_LOGIC_VECTOR (7 downto 0);
d4 : in STD_LOGIC_VECTOR (7 downto 0);
s1 : in STD_LOGIC;
sum : inout STD_LOGIC;
o1: inout STD_LOGIC;
architecture Behavioral of transmitterfinal is
component testbench
port(pulse1: out std_logic;
pulse2: out std_logic);
end component;
component spre
Port ( clk : in STD_LOGIC;
load: in STD_LOGIC;
data_in : in STD_LOGIC_VECTOR (7 downto 0);
piso_out : out STD_LOGIC);
end component;
component pnsmall_1
Port ( clock : in STD_LOGIC;
init : in STD_LOGIC_VECTOR (3 downto 0);
 pn1,pn2,pn3,pn4 : out STD_LOGIC);
end component;
signal c1:std_logic;
signal c2:std_logic;
signal pno1:std_logic;
signal pno2:std_logic;
signal pno3:std_logic;
signal pno4:std_logic;
begin
s1: testbench port map(c1,c2);
s2:spre port map(c2,sl,d1,o1);
s3:pnsmall_1 port map(c1,init1,pno1,pno2,pno3,pno4);
s4:spreg port map(c2,s1,d2,o2);
s5:spreg port map(c2,s1,d3,o3);
s6:spreg port map(c2,s1,d4,o4);
process(c1)
begin
opn1<=pno1;
opn2<=pno2;
opn3<=pno3;
opn4<=pno4;
spr1<= pno1 and o1;
spr2<= pno2 and o2;
spr3<= pno3 and o3;
spr4<= pno4 and o4;
sun<= spr1 or spr2 or spr3 or spr4;
end process;

end Behavioral;

7.1.9 Final Receiver

- Company:
- Engineer:
  
  - Create Date: 23:39:22 03062013
- Design Name:
- Module Name: receiverfinal - Behavioral
- Project Name:
- Target Devices:
- Tool versions:
- Description:
  
  - Dependencies:

  - Revision:
  - Revision 0.01 File Created
  - Additional Comments:


52
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
-- library UNISIM;
-- use UNISIM.VComponents.all;

type recipient is
  Port (sum,clk : in STD_LOGIC;
       sl : in STD_LOGIC;
       y1: inout STD_LOGIC;
       y2: inout STD_LOGIC;
       y3: inout STD_LOGIC;
       y4: inout STD_LOGIC;
       init1 : in STD_LOGIC_VECTOR (3 downto 0);
       opn1 : out STD_LOGIC;
       opn2 : out STD_LOGIC;
       opn3 : out STD_LOGIC;
       opn4 : out STD_LOGIC;
       od1 : out STD_LOGIC;
       od2 : out STD_LOGIC;
       od3 : out STD_LOGIC;
       od4 : out STD_LOGIC;
       P1 : inout STD_LOGIC_VECTOR(14 DOWNTO 0);
       P2 : inout STD_LOGIC_VECTOR(14 DOWNTO 0);
       r1 : inout STD_LOGIC_VECTOR(14 DOWNTO 0);
       r2 : inout STD_LOGIC_VECTOR(14 DOWNTO 0));
end recipient;

architecture Behavioral of recipient is
  component testbench
  port(pulse1: out std_logic;
       pulse2: out std_logic);
  end component;
  component pnsmall_1
  Port ( clock : in STD_LOGIC;
         init : in STD_LOGIC_VECTOR (3 downto 0);
         pn1,pn2,pn3,pn4 : out STD_LOGIC);
  end component;

53
end component;
component sipo is
Port (din in STD_LOGIC;
clk, rst : in STD_LOGIC;
qout : inout STD_LOGIC_VECTOR (14 downto 0));
end component;
signal c1 : std_logic;
signal c2 : std_logic;
signal pno1 : std_logic;
signal pno2 : std_logic;
signal pno3 : std_logic;
signal pno4 : std_logic;
signal pno01 : std_logic;
signal pno02 : std_logic;
signal pno03 : std_logic;
signal pno04 : std_logic;
signal pno001 : std_logic;
signal pno002 : std_logic;
signal pno003 : std_logic;
signal pno004 : std_logic;
signal r1 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal r2 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal r3 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal r4 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal P1 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal P2 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal P3 : STD_LOGIC_VECTOR (14 DOWNTO 0);
signal P4 : STD_LOGIC_VECTOR (14 DOWNTO 0);
begin
s1: testbench port map (c1, c2);
s3: pnsmall_1 port map (clk, init1, pno1, pno2, pno3, pno4);
process (clk)
begin
opn1 <= pno1;
opn2 <= pno2;
opn3 <= pno3;
opn4 <= pno4;
end process;

process(clk,sum)
begin
y1<= sum and pnoo1;
y2<= sum and pnoo2;
y3<= sum and pnoo3;
y4<= sum and pnoo4;
end process;
x1:sipo port map (y1,clk,'0',r1);
x2:sipo port map (y2,clk,'0',r2);
x3: sipo port map (y3,clk,'0',r3);
x4:sipo port map (y4,clk,'0',r4);
x5:sipo port map (pno1,clk,'0',p1);
x6:sipo port map (pno2,clk,'0',p2);
x7:sipo port map (pno3,clk,'0',p3);
x8:sipo port map (pno4,clk,'0',p4);
process(c2)
begin
if(c2' event and c2='1') then
if(r1=p1) then
  od1<='1';
else
  od1<='0';
end if;
if(r2=p2) then
  od2<='1';
else
  od2<='0';
end if;
if(r3=p3) then
  od3<='1';
else
  od3<='0';
end if;
if(r4=p4) then
  od4<='1';
else
  od4<='0';
end if;
end if;
end process;
end Behavioral;
7.1.10 Single bit data Transceiver

-- Company:
-- Engineer:

-- Create Date: 15:30:52 03082013
-- Design Name:
-- Module Name: singlebitdata - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:

-- Dependencies:

-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity singlebitdata is
Port ( d1 : in STD_LOGIC;
d2 : in STD_LOGIC;
d3 : in STD_LOGIC;
d4 : in STD_LOGIC;
sum : inout STD_LOGIC;
init1 : in STD_LOGIC_VECTOR (3 downto 0);
opn1 : out STD_LOGIC;
opn2 : out STD_LOGIC;
architecture Behavioral of singlebitdata is
component testbench
port(pulse1: out std_logic;
pulse2: out std_logic);
end component;
component pnsmall_1
Port ( clock : in STD_LOGIC;
init : in STD_LOGIC_VECTOR (3 downto 0);
pn1,pn2,pn3,pn4 : out STD_LOGIC);
end component;
component sipo is
Port ( din : in STD_LOGIC;
clk,rst : in STD_LOGIC;
qout : inout STD_LOGIC_VECTOR (14 downto 0));
end component;
signal c1:std_logic;
signal c2:std_logic;
signal pno1:std_logic;
signal pno2:std_logic;
signal pno3:std_logic;
signal pno4:std_logic;
signal pnoo1:std_logic;
signal pnoo2:std_logic;
signal pnoo3:std_logic;
signal pnoo4:std_logic;
signal pnooo1:std_logic;
signal pnooo2:std_logic;
signal pnooo3:std_logic;
signal pno004: STD_LOGIC;
signal y1: STD_LOGIC;
signal y2: STD_LOGIC;
signal y3: STD_LOGIC;
signal y4: STD_LOGIC;
signal r1 :STD_LOGIC_VECTOR(14 DOWNTO 0);
signal r2 :STD_LOGIC_VECTOR(14 DOWNTO 0);
signal r3 :STD_LOGIC_VECTOR(14 DOWNTO 0);
signal r4 :STD_LOGIC_VECTOR(14 DOWNTO 0);
signal P1 :STD_LOGIC_VECTOR(14 DOWNTO 0);
signal P2 :STD_LOGIC_VECTOR(14 DOWNTO 0);
signal P3 :STD_LOGIC_VECTOR(14 DOWNTO 0);
signal P4 :STD_LOGIC_VECTOR(14 DOWNTO 0);
signal spr1 : STD_LOGIC;
signal spr2 : STD_LOGIC;
signal spr3 : STD_LOGIC;
signal spr4 : STD_LOGIC;
begin
s1: testbench port map(c1,c2);
s2: pnext1 port map(c1,init1,pno1,pno2,pno3,pno4);

process(c1)
begin
  opn1<=pno1;
  opn2<=pno2;
  opn3<=pno3;
  opn4<=pno4;
  spr1<= pno1 and d1;
  spr2<= pno2 and d2;
  spr3<= pno3 and d3;
  spr4<= pno4 and d4;
  sum<= spr1 or spr2 or spr3 or spr4;
  pno01<=pno1 after 1ns;
  pno02<=pno2 after 1ns;
  pno03<=pno3 after 1ns;
  pno04<=pno4 after 1ns;
end process;

process(c1,sum)
begin
y1 <= sum and pno o1;
y2 <= sum and pno o2;
y3 <= sum and pno o3;
y4 <= sum and pno o4;
end process;
pno o01<=pno o1 after 2ns;
pno o02<=pno o2 after 2ns;
pno o03<=pno o3 after 2ns;
pno o04<=pno o4 after 2ns;
x1:sipo port map (y1,c1,'0',r1);
x2:sipo port map (y2,c1,'0',r2);
x3:sipo port map (y3,c1,'0',r3);
x4:sipo port map (y4,c1,'0',r4);
x5:sipo port map (pno o o1,c1,'0',p1);
x6:sipo port map (pno o o2,c1,'0',p2);
x7:sipo port map (pno o o3,c1,'0',p3);
x8:sipo port map (pno o o4,c1,'0',p4);
process(c2)
begi
if(c2’ event and c2 = ’1’) then
if(r1=p1) then
od1<='1'; else
od1<='0';
end if;
if(r2=p2) then
od2<='1'; else
od2<='0';
end if;
if(r3=p3) then
od3<='1'; else
od3<='0';
end if;
if(r4=p4) then
od4<='1'; else
od4<='0';
end if;
end process;
end Behavioral;