International Journal of Research
Available at https://journals.pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 07 Issue 06
June 2020

VLSI Architectures for 8 Bit Data Comparators for Rank Ordering Image Applications

C Satish Babu[1], S. Ashok Reddy[2] Mallikarjuna[3]
[1]P.G. Scholar, [2]Guide, Assistant Professor, [3]Head Of The Department
[1,2,3]Branch : Vlsi Design
[1,2,3] Geethanjali College of Engineering and Technologies
(Formerly AVR & SVR Engineering College)
Email: [1]satishbabji143@gmail.Com, [2]singasaniashokreddy@gmail.com

Abstract
The main objective of the proposed work is to develop a new Data comparator which gives an economical solution for sorting / Rank ordering networks on the basis of speed, power, and area. The proposed work comprises a design of six different comparators for 8 -bit comparison. The performances of these six different comparators were targeted for XCV1000-4bg560 using Xilinx 7.1i compiler tool using VHDL. It was found that Carry select logic based data comparator requires less area and suitable for reduced area applications. The Conventional bit wise logic based data comparator operated with less delay. Hence this architecture can be used for high speed application.

The Twos complement using binary to excess one data converter consumes less power. Hence a twos complement based implementation is suitable for low power implementations. The Different architecture for data comparators were applied on parallel and pipelined architecture of modified shear sorting. These three architectures were compared with other existing median finding architectures. It was found that the CSLA based parallel architecture, CBC based pipelined architecture and 2BEC based pipelined architecture were stand out in area, speed and power when compared with conventional median finding algorithms.

Keywords : Data Comparator, Modified shear sorting, Parallel architecture, Pipelined architecture, VHDL.

Introduction

THE comparator is a very useful combinational circuit used for testing whether the binary number at one input is greater or less than to another binary number. An XOR gate can be used as an essential comparator. The Comparators are comprised of two types

(a).Magnitude Comparator

(b). Data Comparator.

The former compares only the magnitude of the two binary numbers and the later gives the greater and a lesser data itself. The Magnitude comparator has two outputs to indicate whether first input is greater than second input or vice versa. Whereas data comparator can also be referred to two cell comparator, as it compares word X with word Y and gives out a Higher and lower value respectively A compressed, good quality cost, high performance, and low power comparator play a significant role in almost all hardware comparators. The work attempt to observe the features of certain comparator circuits which assure better performance compared to existing circuits. Karpagaabirami and Ramamoorthy developed an Adaptive Rank Order filter (AROF) with VLSI implementation had been developed to remove impulse noise and pipelining with

parallel processing in order to speed up filtering process.

The advantage of Decision Rank Order Filter (DROF) consumes less area and also architecture is simple compared to Decision Tree Based De-noising Method (DTBDM).The disadvantage of VLSI DTBDM involves too many architectures for detection of noise and reconstruction of noisy pixel. Ayeesha et al explored the design of high speed and low power comparator since it operates only with 1volt power and less propagation delay and its architecture includes two stage CMOS op-amp circuit. In this work, comparator is designed with cadence tool with 0.18micrometer technology.

Bharat et al introduced a special types of comparators and these circuits are simulated with 1 Volt DC supply voltage in LTspice-IV using PTM 45nm technology. Unlike static and dynamic characteristics of all these comparators are considered and compared and it operates with higher speed and provide more stabilized output compare to 90nm and 180nm. Mehamood et al developed various new designs to reduce the area and power consumption as small saving in area and power of a circuit yield a large overall saving. From the evaluation, found that full-custom design saves about 50%in area and 35%in power consumption when compared to auto-generated design. Vasanth et al developed a parallel architecture and pipelined architecture for modified shear sorting. The method introduced an area efficient data comparator for sorting 9 elements. The basic processing element is area optimized two cell sorter. A group of three two cell sorter form a three cell sorter which in turn uses compare and swap approach for ordering the data sequence.

Keeping chan developed a VLSI algorithms and implementation architectures for a class of nonlinear filters. The class of filters contains all of the functions earlier defined by stack filters, where rank-order and median filters are special cases. The function of a stack filter can be realized in k-step recursive use of one binary processing circuit.

Vasanth et al introduced a novel Borrow Look Ahead Logic based Comparator (BLAC) and implemented the output in both pipelined and parallel architecture of modified shear sorting. Vasanth et al introduced finite state machine based VLSI architecture for Decision based Unsymmetrical trimmed midpoint filter. Vasanth proposed a novel methodology for finding median of an array using modified selection sort. Vasanth et al introduced a novel 8 bit data comparator which used carry select logic. The proposed bit-serial architecture is very suitable for VLSI implementation. Hence a suitable Data comparator has to be formulated that uses a les area, consumes low power and operates at very high speed.

PROPOSED MODEL

A. Basic Structure of Data Comparator

The basic structure for the Proposed Data Comparator consists of two inputs and two outputs. The internal structure of the data comparators consists of a logic circuits that identifies which of the two inputs is greater by generating a signal which is in turn drives two multiplexers which yield a high or a low value depending upon the generated signal. The block diagram of the basic structure of the Data comparators is given in Fig. 1.

The basic operation of comparator is subtraction of two numbers. The paper deals with different types of implementation of subtraction done to yield borrow generation for data comparator output. The work focus on six basic styles to design data comparators which are given below:

Conventional bitwise data comparator
Carry select logic data comparator (CSLC)
Borrow Look ahead logic data Comparator

International Journal of Research
Available at https://journals.pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 07 Issue 06
June 2020

Decoder based Data comparator (DDC)
Multiplexer based data comparator (MDC)
Twos complement comparator using Binary to excess one converter (2BEC)

B. Conventional bitwise data comparator

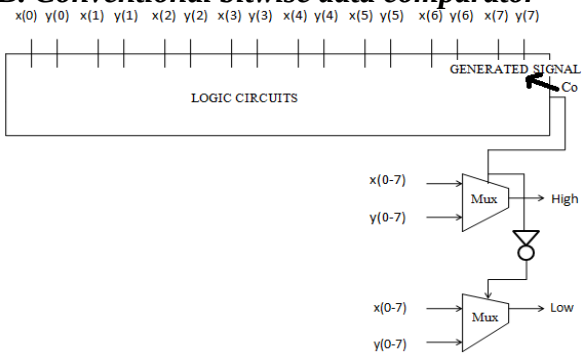

Fig. 1. Block Diagram of a Data Comparator

Conventional data comparator uses the methodology of magnitude comparator and acts like a word comparator. A 1- bit magnitude comparator is used as a processing element for 8-bit comparison where a, b are two 1 bit number and there are two outputs corresponding to a>band a<b respectively as shown in the Fig. 2. The result of the comparison is given as input to the multiplexer to determine whether the word of each 8-bit with another word b of each 8-bit is greater or not. If it is found greater then it provides the output as a maximum (max) otherwise minimum (min) respectively. The block diagram of the conventional Data Comparator is shown in Fig. 2.

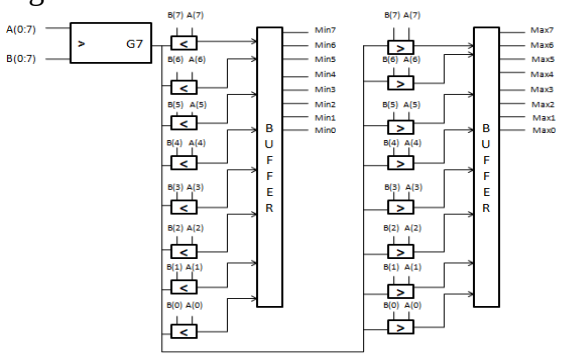

Fig. 2. Block Diagram of Conventional Data Comparator

C. Carry select logic data comparator

The carry select logic comparator design was developed for an 8-bit comparator. To implement a comparator an alternative basic operation is a subtraction. Hence we use a rippled borrow output of an 8 bit full subtractor to select the data from the multiplexer. The carry select logic will be used to so that High and low values are obtained using compare and swap function. In this work, carry select logic is set for an X-Y to subtract.

In the above design, Xi and Yi are the inputs which are feed into a carry select logic circuit to perform the subtraction and to obtain the output as a carry. The output of the carry select logic block gives the final carry directly to a multiplexer with Xi and Yi as inputs which in turn gives the lowest value and inverted output of the carry select logic block is given to another multiplexer gives the highest value.

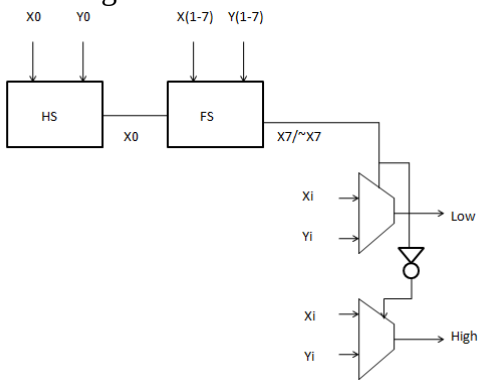

Fig. 3. Block Diagram of Carry select data Comparator

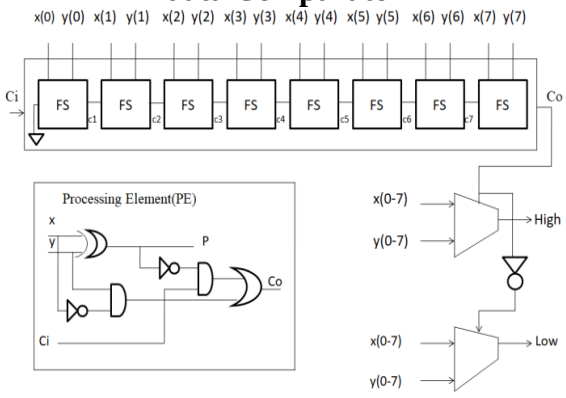

Fig. 4. Block Diagram of Borrow look ahead Data Comparator

In this work to compare and swap two 8 bit numbers, we need one- half

International Journal of Research
Available at https://journals.pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 07 Issue 06
June 2020

subtractor, seven full subtractors and two 2:1 multiplexer. The multiplexer is controlled by the carryout of the subtract function. The carry logic circuit output is obtained from onehalf subtractor and seven full subtractors. the final full subtractor is given as input to a multiplexer from which the Low value is obtained and the inverted Carryout The borrow of the value is given to another multiplexer which gives the High value. The block diagram of the Carry select logic Data Comparator is shown in Fig. 3. In this approach both the half and full subtractor uses borrow equations to select output of data multiplexers shown in the Fig. 3. The borrow ripples to the next stage. Hence the dependency of borrow arises.

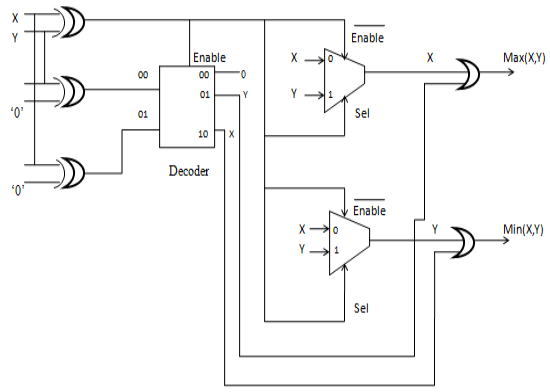

Fig. 5. Block Diagram of Decoder based Data Comparator

D. Borrow look ahead logic data comparator (blac)

The borrow dependency problem is eliminated using borrow look ahead logic data comparator. An 8-bit comparator that uses Borrow Look Ahead Select Logic (BLAC) [8] in the place of logic circuits that generates a carry that is not dependent on the previous stage as given in equations below. This carry will find the higher and lower value of two numbers. This basically works as a Data comparator. The borrow look ahead select logic arrangement is implemented mainly to eliminate the carry dependencies that arises in the previous stages. To understand the working of BLAC, consider the basic borrow equation of a full subtractor in equation. Let x and y are the two 8 bit inputs and Ci refers to initial carry (which is generally 0) and Co refers to carry out.

Co = ((not x) and y) or ((not x) and Ci) or (y and Ci) (1)

On assuming two functions to represent the basic borrow functions named as Generate (G), Propagate (P) as shown in equation (2) and (3).where i represents the number of bits.

Gi = (not xi) and yi (2)

Pi = xi xor yi (3)

Substituting the equations (3) and (2) in equation(1) we get.

C(i+1)=Gi + (not Pi) Ci (4)

Now the equation(a) gets modified as equation(d) . Vary the value of i from 0 to 7 resulting in a carry generation in C7 referred as carryout.

i = 0 C0 = G0 (5)

i = 1 C1 = G1 + P1G0 (6)

i = 2 C2 = G2 + P2G1 + P2P1G0 (7)

i = 3 C3 = G3 + P3G2 + P3P2G1 + P3P2P1G0 (8)

i = 4 C4 = G4 + P4G3 + P4P3G2 + P4P3P2G1 +P4P3P2P1G0 (i)i = 5 C5 = G5 + P5G4 + P5P4G3 +P5P4P3G2 + P5P4P3P2G1 + P5P4P3P2P1G0 (9)

i = 6 C6 = G6 + P6G5 + P6P5G4 + P6P5P4G3 +P6P5P4P3G2 + P6P5P4P3P2G1 + P5P4P3P2P1G0 (10)

i = 7 C7 = G7 + P7G6 + P7P6G5 + P7P6P5G4 + P7P6P5P4G3 + P7P6P5P4P3G2 + P7P6P5P4P3P2G1+ P7P6P5P4P3P2P1G0 (l1)

i= 15 C15 = G15 +P15G14 + P15P14G13 + P15P14P13G12 + P15P14P13P12G11 + P15P14P13P12P11G10 + P15P14P13P12P11P10G9 + P15P14P13P12P11P10P9G8 + P15P14P13P12P11P10P9P8G7+P15P14P13P12P11P10P9P8P7G6+P15P14P13P12P11P10P9P8P7P6G5+P15P14P13P12P11P10P9P8P7P6P5G4+P15P14P13P12P11P10P9P8P7P6P5P4G3+P15P14P13P12P11P10P9P8P7P6P5P4P3G+P15P14P13P12P11P10P9P8P7P6P5P4P3P2G1 + P15P14P13P12P11P10P9P8P7P6P5P4P3P2P1G0

From equation 11 it is vivid that the generated borrowout value will depend only on initial carry only. The carry out will act as an input for the first Multiplexer which

gives the output High value and the inverted value is fed to another multiplexer which gives the output as Low value. The Block Diagram of Borrow Look Ahead logic Data Comparator is shown in Fig.4.

E. Decoder based data comparator

The Decoder-based comparator is used to control the flow of logic i.e data transfer [7]. The proposed logic circuit is simple and effective that compares MSB of first 8-bit with MSB of another 8-bit binary number and provides the output by performing basic XOR gate operation of X and Y respectively. The output of these is fed to decoder, that operates when the bits are identical the decoder is disabled and when the bits be different from each other, the decoder is enabled and put them in the proper order. However, the decoder based data comparator consider only the two cases of X>Y and X<Y, therefore, it is important whether the output is the Max(X, Y) or the Min(X, Y), thus we include 2 multiplexers. The output of the multiplexers is based on the selection line which acts as one of the inputs to both OR gates and second input depends on the decoder line as shown in the figure. If the decoder line is '01'i.e Y is selected to perform OR operation of X and Y, then it gives maximum value and the inverted operation i.e '10' line of Y and X gives the minimum value. The block diagram of Decoder based Data Comparator shown in Fig. 5.

F. Multiplexer based data comparator

A Processing Element (PE) consists of OR gate, AND gate and Multiplexer. The inspiration of the work is derived from Keshab parhi adder using multiplexer. The truth table of full subtractor is shown in Table I. The table does not require a difference as we are interested only in borrow out. From inspection of truth table, the following inference was made. The borrow generation circuit required for data comparator is derived from the truth table.

The processing element is built using the truth table values. In this multiplexer based Data comparator any one of the input will act as a selection line and other two input will form the data. A zero in the selection line will choose the "OR" operation output of two inputs and a one will choose the "AND" operation of two inputs. Based on the selection line (third input) the borrow is chosen.

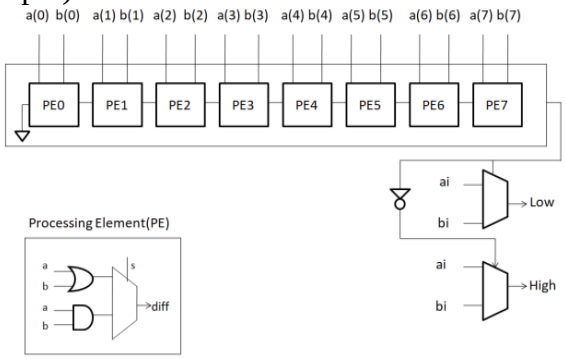

Fig. 6. Block diagram of multiplexer based Data Comparator

The comparator operation can be implemented as a subtractor and the borrow generation is facilitated using this multiplexer based approach. In this work, for 1-bit processing element, the output of these gates is taken as xx1 and xx2 which perform the operation of individual gates based on selection line(s) as shown in the figure. The output of these gates is fed to 2:1 multiplexer gives the difference (diff) as output. The logic generation block is replaced by series of ripple carry multiplexers and Finally, the generated output of PE7 is fed to multiplexer that compares whether the binary number is greater or not. Here if it is greater which gives Low value and otherwise High value. The block diagram of multiplexer based Data Comparator is shown in Fig. 6. Table I gives the truth table for Multiplexer based comparator. The red rectangle in input side indicate the selection line for the multiplexer and the red rectangle in output resembles the truth table of an OR gate. The black oval in input indicate the selection line as 1 and the green oval in output resemble the output of an AND gate.

International Journal of Research
Available at https://journals.pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 07 Issue 06
June 2020

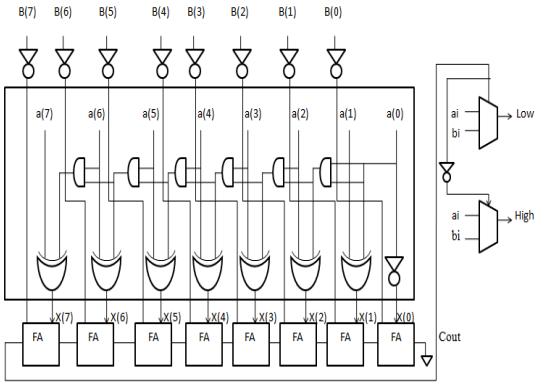

Fig. 7. Block diagram of the Twos complement based Data Converter using binary to excess one logic

G. Two's complement based data converter:

This data comparators uses twos complement implementation to implement a Data comparator. The basic twos complement implementation is given by A+B'+1. In this design, B'+1 is implemented using binary to excess one converter instead of a full adder. Adding the result of B'+1 to A is by using a set of full adders. This summation result in a carry. This carry will drive the multiplexers to yield High and low values. The main logic of this 8 Bit Binary to Excess one Converter (BEC) is given in equation 12- 19.The Boolean expressions of the 8-bit BEC are

X0 = not B0 (12)
X1 = B0xor B1 (13)
X2 = B2xor (B0 and B1) (14)
X3 = B3xor (B0 and B1 and B2) (15)
X4 = B4xor (B0 and B1 and B2 and B3) (16)
X5= B5xor (B0and B1 and B2 and B3and B4) (17)
X6= B6xor (B0 andB1 and B2and B3 and B5andB5) (18)
X7= B7xor (B0 and B1 andB2and B3and B4andB5andB6) (19)

MODIFIED SHEAR SORTING (MSS)

Modified shear sorting is simple and fast algorithm which is used to determine median of list of elements using compare and swap operation. The following steps are followed to perform modified shear sorting are shown in Fig. 8.

1. Arrange the elements of row in ascending order.
2. Arrange the elements of column in descending order.
3. Arrange the elements of right diagonal in ascending order,

and then we get the semi sorted processing window. Thus, we get the first element of the array as minimum (min), second element as median (med) and third element as maximum (max) , which are used for ranking elements in proper order.

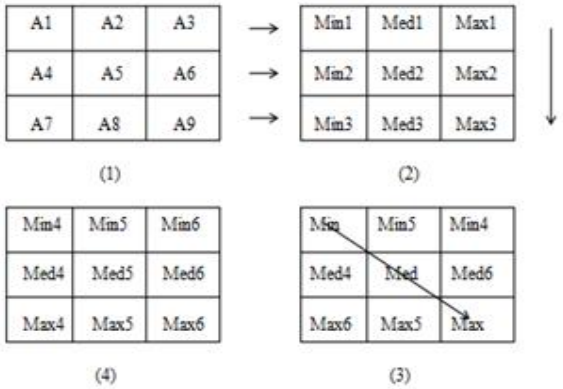

Fig. 8. Illustrates the methodology of Modified shear sorting

A. Parallel architecture for modified shear sorting

The methodology of modified shear sorting is implemented using compare and swap operation that works as a parallel architecture. The different architecture for data comparator also referred as two cell sorter is used to build a small processing element called three cell sorter. A three cell sorter is a basic unit that compares three elements and gives out Maximum, median and minimum value of three elements. These three cell sorter are the basic processing element for the Modified Shear sorting. The fundamental operation of parallel architecture is two cell sorter and the processing element of these architecture is a three cell sorter because three cell sorter is equal to three two cell sorters. In this architecture it basically requires a network in order to determine median value from the

International Journal of Research
Available at https://journals.pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 07 Issue 06
June 2020

given nine elements by using the methodology of sorting.

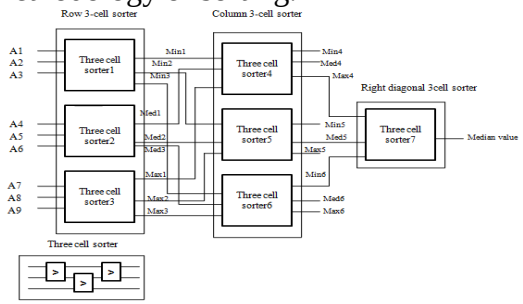

Fig. 9. Illustrates the parallel architecture for Modified shear sorting

The architecture shown in Fig. 9 is used with seven three cell sorters to perform the operation either in row, column and right diagonal which contain nine elements. As shown in figure, it is vivid that we use three sets of 3-cell sorter for arranging three elements of the row and column in increasing order. Fig. 8. Illustrates the methodology of Modified shear sorting One three cell sorter is used for sorting elements in the diagonal. The three cell sorter arranges the outputs as minimum, median and maximum respectively. The lowest value of first, second and third row 3-cell sorter are given as input to first column 3-cell sorter. The mid values of first, second and third row3-cell sorter are given as input to second column 3-cell sorter. The highest values of first, second and third row3-cell sorter are given as input to third column 3-cell sorter. Then the higher value of first column 3-cell sorter, middle value of second column 3-cell sorter and finally the lower value of third column 3-cell sorter is compared with right diagonal 3-cell sorter as shown in the following Fig. 10. Thus, the output of right diagonal 3-cell sorter is considered as median value.

B. Pipelined architecture for modified shear sorting

Pipelining design is a suitable VLSI architecture for decreasing the critical path, meanwhile it can reduces the speed. The basic operation of these architecture is performed by using clock (clk) sequence. These are the steps while processing is as follows:

Step1: During first clock pulse, the first set of three elements is fed into a first 3-cell sorter and the output from this is sent into three set of registers.

Step2: At second clock pulse, the second set of three elements is fed into a same 3-cell sorter and the output from this is stored into the subsequent three set of registers.

Step3: At third clock pulse, final set of three elements is fed into a same 3-cell sorter and at the same time elements are stored in the two set of registers are compare with another 3- cell sorter. The result obtained at the end of third clock pulse Step4: At the fifth clock pulse, the outputs of two sorter are compared and the result from this is again passed into the 3- cell sorter.

Step5: During seventh clock pulse, the result obtained is considered as median value

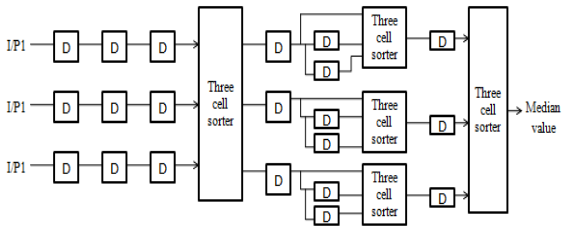

Fig. 10. Pipelined architecture for modified shear sorting

INTRODUCTION TO XILINX

Migrating Projects from Previous ISE Software Releases:

When you open a project file from a previous release, the ISE® software prompts you to migrate your project. If you click Backup and Migrate or Migrate Only, the software automatically converts your project file to the current release. If you click Cancel, the software does *not* convert your project and, instead, opens Project Navigator with no project loaded.

Note: After you convert your project, you *cannot* open it in previous versions of the ISE software, such as the ISE 11 software. However, you can optionally create a

backup of the original project as part of project migration, as described below.

To Migrate a Project

1. In the ISE 12 Project Navigator, select **File > Open Project**.
2. In the Open Project dialog box, select the .xise file to migrate.

Note You may need to change the extension in the Files of type field to display .npl (ISE 5 and ISE 6 software) or .ise (ISE 7 through ISE 10 software) project files.

3. In the dialog box that appears, select **Backup and Migrate** or **Migrate Only**.
4. The ISE software automatically converts your project to an ISE 12 project.

Note If you chose to Backup and Migrate, a backup of the original project is created at *project_name*_ise12migration.zip.

5. Implement the design using the new version of the software.

IP Modules:

If your design includes IP modules that were created using CORE Generator™ software or Xilinx® Platform Studio (XPS) and you need to modify these modules, you may be required to update the core. However, if the core netlist is present and you do not need to modify the core, updates are not required and the existing netlist is used during implementation.

Obsolete Source File Types:

The ISE 12 software supports all of the source types that were supported in the ISE 11 software. If you are working with projects from previous releases, state diagram source files (.dia), ABEL source files (.abl), and test bench waveform source files (.tbw) are no longer supported. For state diagram and ABEL source files, the software finds an associated HDL file and adds it to the project, if possible. For test bench waveform files, the software automatically converts the TBW file to an HDL test bench and adds it to the project. To convert a TBW file *after* project migration, see Converting a TBW File to an HDL Test Bench.

Using ISE Example Projects:

To help familiarize you with the ISE® software and with FPGA and CPLD designs, a set of example designs is provided with Project Navigator. The examples show different design techniques and source types, such as VHDL, Verilog, schematic, or EDIF, and include different constraints and IP.

To Open an Example

1. Select **File > Open Example**.
2. In the Open Example dialog box, select the Sample Project Name.

Note To help you choose an example project, the Project Description field describes each project. In addition, you can scroll to the right to see additional fields, which provide details about the project.

3. In the Destination Directory field, enter a directory name or browse to the directory.
4. Click **OK**.

The example project is extracted to the directory you specified in the Destination Directory field and is automatically opened in Project Navigator. You can then run processes on the example project and save any changes.

Note If you modified an example project and want to overwrite it with the original example project, select **File > Open Example**, select the Sample Project Name, and specify the same Destination Directory you originally used. In the dialog box that appears, select **Overwrite the existing project** and click **OK**.

Creating a Project:

Project Navigator allows you to manage your FPGA and CPLD designs

using an ISE® project, which contains all the source files and settings specific to your design. First, you must create a project and then, add source files, and set process properties. After you create a project, you can run processes to implement, constrain, and analyze your design. Project Navigator provides a wizard to help you create a project as follows.

To Create a Project

1. Select **File > New Project** to launch the New Project Wizard.
2. In the **Create New Project page,** set the name, location, and project type, and click **Next**.
3. *For EDIF or NGC/NGO projects only*: In the **Import EDIF/NGC Project page**, select the input and constraint file for the project, and click **Next**.
4. In the **Project Settings page**, set the device and project properties, and click **Next**.
5. In the **Project Summary page**, review the information, and click **Finish** to create the project

Project Navigator creates the project file (*project_name*.xise) in the directory you specified. After you add source files to the project, the files appear in the Hierarchy pane of the

Design panel:

Project Navigator manages your project based on the design properties (top-level module type, device type, synthesis tool, and language) you selected when you created the project. It organizes all the parts of your design and keeps track of the processes necessary to move the design from design entry through implementation to programming the targeted Xilinx® device.

Note For information on changing design properties, see **Changing Design Properties.**

You can now perform any of the following:

Create new source files for your project.
Add existing source files to your project.
Run processes on your source files.
Modify process properties.

Creating a Copy of a Project:

You can create a copy of a project to experiment with different source options and implementations. Depending on your needs, the design source files for the copied project and their location can vary as follows:

- Design source files are left in their existing location, and the copied project points to these files.
- Design source files, including generated files, are copied and placed in a specified directory.
- Design source files, excluding generated files, are copied and placed in a specified directory.

Copied projects are the same as other projects in both form and function. For example, you can do the following with copied projects:

- Open the copied project using the File > Open Project menu command.
- View, modify, and implement the copied project.
- Use the Project Browser to view key summary data for the copied project and then, open the copied project for further analysis and implementation, as described in

Using the Project Browser:

Alternatively, you can create an archive of your project, which puts all of the project contents into a ZIP file. Archived projects must be unzipped before being opened in Project Navigator. For information on archiving, see

Creating a Project Archive.

To Create a Copy of a Project

Select **File > Copy Project**.

In the Copy Project dialog box, enter the **Name** for the copy.

Note The name for the copy can be the same as the name for the project, as long as you specify a different location.

1. Enter a directory **Location** to store the copied project.
2. Optionally, enter a **Working directory**.

By default, this is blank, and the working directory is the same as the project directory. However, you can specify a working directory if you want to keep your ISE® project file (.xise extension) separate from your working area.

3. Optionally, enter a **Description** for the copy.

The description can be useful in identifying key traits of the project for reference later.

4. In the Source options area, do the following:

Select one of the following options:

- **Keep sources in their current locations -** to leave the design source files in their existing location.

If you select this option, the copied project points to the files in their existing location. If you edit the files in the copied project, the changes also appear in the original project, because the source files are shared between the two projects.

Copy sources to the new location - to make a copy of all the design source files and place them in the specified Location directory.

If you select this option, the copied project points to the files in the specified directory. If you edit the files in the copied project, the changes do *not* appear in the original project, because the source files are not shared between the two projects.

Optionally, select **Copy files from Macro Search Path directories** to copy files from the directories you specify in the Macro Search Path property in the **Translate Properties** dialog box. All files from the specified directories are copied, not just the files used by the design.

Note: If you added a net list source file directly to the project as described in **Working with Net list-Based IP**, the file is automatically copied as part of Copy Project because it is a project source file. Adding net list source files to the project is the preferred method for incorporating net list modules into your design, because the files are managed automatically by Project Navigator.

Optionally, click **Copy Additional Files** to copy files that were not included in the original project. In the Copy Additional Files dialog box, use the **Add Files** and **Remove Files** buttons to update the list of additional files to copy. Additional files are copied to the copied project location after all other files are copied.To exclude generated files from the copy, such as implementation results and reports, select **Exclude generated files from the copy**:

When you select this option, the copied project opens in a state in which processes have not yet been run.

5. To automatically open the copy after creating it, select **Open the copied project**.

Note By default, this option is disabled. If you leave this option disabled, the original project remains open after the copy is made.

Click **OK**.

Creating a Project Archive:

A project archive is a single, compressed ZIP file with a .zip extension. By default, it contains all project files, source files, and generated files, including the following:

User-added sources and associated files
Remote sources
Verilog `include files
Files in the macro search path
Generated files
Non-project files

To Archive a Project:

1. Select **Project > Archive**.
2. In the Project Archive dialog box, specify a file name and directory for the ZIP file.
3. Optionally, select **Exclude generated files from the archive** to exclude

generated files and non-project files from the archive.

4. Click **OK**.

A ZIP file is created in the specified directory. To open the archived project, you must first unzip the ZIP file, and then, you can open the project.

Note Sources that reside outside of the project directory are copied into a remote_sources subdirectory in the project archive. When the archive is unzipped and opened, you must either specify the location of these files in the remote_sources subdirectory for the unzipped project, or manually copy the sources into their original location.

INTRODUCTION TO VERILOG

In the semiconductor and electronic design industry, **Verilog** is a hardware description language(HDL) used to model electronic systems. *Verilog HDL*, not to be confused with VHDL (a competing language), is most commonly used in the design, verification, and implementation of digital logic chips at the register-transfer level of abstraction. It is also used in the verification of analog and mixed-signal circuits.

Verilog-95

With the increasing success of VHDL at the time, Cadence decided to make the language available for open standardization. Cadence transferred Verilog into the public domain under the Open Verilog International (OVI) (now known as Accellera) organization. Verilog was later submitted to IEEE and became IEEE Standard 1364-1995, commonly referred to as Verilog-95. In the same time frame Cadence initiated the creation of Verilog-A to put standards support behind its analog simulator Spectre. Verilog-A was never intended to be a standalone language and is a subset of Verilog-AMS which encompassed Verilog-95.

Verilog 2001

Extensions to Verilog-95 were submitted back to IEEE to cover the deficiencies that users had found in the original Verilog standard. These extensions became IEEE Standard 1364-2001 known as Verilog-2001.

Verilog-2001 is a significant upgrade from Verilog-95. First, it adds explicit support for (2's complement) signed nets and variables. Previously, code authors had to perform signed operations using awkward bit-level manipulations (for example, the carry-out bit of a simple 8-bit addition required an explicit description of the Boolean algebra to determine its correct value). The same function under Verilog-2001 can be more succinctly described by one of the built-in operators: +, -, /, *, >>>. A generate/endgenerate construct (similar to VHDL's generate/endgenerate) allows Verilog-2001 to control instance and statement instantiation through normal decision operators (case/if/else). Using generate/endgenerate, Verilog-2001 can instantiate an array of instances, with control over the connectivity of the individual instances. File I/O has been improved by several new system tasks. And finally, a few syntax additions were introduced to improve code readability (e.g. always @*, named parameter override, C-style function/task/module header declaration).

Verilog 2005

Not to be confused with SystemVerilog, *Verilog 2005* (IEEE Standard 1364-2005) consists of minor corrections, spec clarifications, and a few new language features (such as the uwire keyword).

A separate part of the Verilog standard, Verilog-AMS, attempts to integrate analog and mixed signal modeling with traditional Verilog.

SystemVerilog

SystemVerilog is a superset of Verilog-2005, with many new features and capabilities to aid design verification and design modeling. As of 2009, the SystemVerilog and Verilog language standards were merged into SystemVerilog 2009 (IEEE Standard 1800-2009).

The advent of hardware verification languages such as OpenVera, and Verisity's e language encouraged the development of Superlog by Co-Design Automation Inc. Co-Design Automation Inc was later purchased by Synopsys. The foundations of Superlog and Vera were donated to Accellera, which later became the IEEE standard P1800-2005: SystemVerilog.

In the late 1990s, the Verilog Hardware Description Language (HDL) became the most widely used language for describing hardware for simulation and synthesis. However, the first two versions standardized by the IEEE (1364-1995 and 1364-2001) had only simple constructs for creating tests. As design sizes outgrew the verification capabilities of the language, commercial Hardware Verification Languages (HVL) such as Open Vera and *e* were created. Companies that did not want to pay for these tools instead spent hundreds of man-years creating their own custom tools. This productivity crisis (along with a similar one on the design side) led to the creation of Accellera, a consortium of EDA companies and users who wanted to create the next generation of Verilog. The donation of the Open-Vera language formed the basis for the HVL features of SystemVerilog.Accellera's goal was met in November 2005 with the adoption of the IEEE standard P1800-2005 for SystemVerilog, IEEE (2005).

The most valuable benefit of SystemVerilog is that it allows the user to construct reliable, repeatable verification environments, in a consistent syntax, that can be used across multiple projects

Some of the typical features of an HVL that distinguish it from a Hardware Description Language such as Verilog or VHDL are

- Constrained-random stimulus generation
- Functional coverage
- Higher-level structures, especially Object Oriented Programming
- Multi-threading and interprocess communication
- Support for HDL types such as Verilog's 4-state values
- Tight integration with event-simulator for control of the design

There are many other useful features, but these allow you to create test benches at a higher level of abstraction than you are able to achieve with an HDL or a programming language such as C.

System Verilog provides the best framework to achieve coverage-driven verification (CDV). CDV combines automatic test generation, self-checking testbenches, and coverage metrics to significantly reduce the time spent verifying a design. The purpose of CDV is to:

- Eliminate the effort and time spent creating hundreds of tests.
- Ensure thorough verification using up-front goal setting.
- Receive early error notifications and deploy run-time checking and error analysis to simplify debugging.

Examples

Ex1: A hello world program looks like this:

```
module main;
initial
begin
$display("Hello world!");
$finish;
end
endmodule
```

Ex2: A simple example of two flip-flops follows:

```
module toplevel(clock,reset);
input clock;
input reset;
```

```
reg flop1;
reg flop2;
always @ (posedge reset or posedge clock)
if (reset)
begin
flop1 <= 0;
flop2 <= 1;
end
else
begin
flop1 <= flop2;
flop2 <= flop1;
end
endmodule
```

The "<=" operator in Verilog is another aspect of its being a hardware description language as opposed to a normal procedural language. This is known as a "non-blocking" assignment. Its action doesn't register until the next clock cycle. This means that the order of the assignments are irrelevant and will produce the same result: flop1 and flop2 will swap values every clock.

The other assignment operator, "=", is referred to as a blocking assignment. When "=" assignment is used, for the purposes of logic, the target variable is updated immediately. In the above example, had the statements used the "=" blocking operator instead of "<=", flop1 and flop2 would not have been swapped. Instead, as in traditional programming, the compiler would understand to simply set flop1 equal to flop2 (and subsequently ignore the redundant logic to set flop2 equal to flop1.)

Ex3: An example counter circuit follows:

```
module Div20x (rst, clk, cet, cep, count, tc);
// TITLE 'Divide-by-20 Counter with enables'
// enable CEP is a clock enable only
// enable CET is a clock enable and
// enables the TC output
// a counter using the Verilog language
parameter size = 5;
parameter length = 20;
input rst; // These inputs/outputs represent
input clk; // connections to the module.
input cet;
input cep;
output [size-1:0] count;
output tc;
reg [size-1:0] count; // Signals assigned
// within an always
// (or initial)block
// must be of type reg
wire tc; // Other signals are of type wire
// The always statement below is a parallel
// execution statement that
// executes any time the signals
// rst or clk transition from low to high
always @ (posedge clk or posedge rst)
if (rst) // This causes reset of the cntr
count <= {size{1'b0}};
else
if (cet && cep) // Enables both true
begin
if (count == length-1)
count <= {size{1'b0}};
else
count <= count + 1'b1;
end
// the value of tc is continuously assigned
// the value of the expression
assign tc = (cet && (count == length-1));
endmodule
```

Ex4: An example of delays:...

```
reg a, b, c, d;
wire e;...
always @(b or e)
 begin
 a = b & e;
 b = a | b;
 #5 c = b;
 d = #6 c ^ e;
 end
```

The always clause above illustrates the other type of method of use, i.e. the always clause executes any time any of the entities in the list change, i.e. the b or e change. When one of these changes, immediately a is assigned a new value, and due to the blocking assignment b is assigned a new value afterward (taking into account the new value of a.) After a delay of 5 time

units, c is assigned the value of b and the value of c ^ e is tucked away in an invisible store. Then after 6 more time units, d is assigned the value that was tucked away.

Signals that are driven from within a process (an initial or always block) must be of type reg. Signals that are driven from outside a process must be of type wire. The keyword reg does not necessarily imply a hardware register.

Constants

The definition of constants in Verilog supports the addition of a width parameter. The basic syntax is:

<Width in bits>'<base letter><number>
Examples:

- 12'h123 - Hexadecimal 123 (using 12 bits)
- 20'd44 - Decimal 44 (using 20 bits - 0 extension is automatic)
- 4'b1010 - Binary 1010 (using 4 bits)
- 6'o77 - Octal 77 (using 6 bits)

7.4 Synthesizable Constructs

There are several statements in Verilog that have no analog in real hardware, e.g. $display. Consequently, much of the language can not be used to describe hardware. The examples presented here are the classic subset of the language that has a direct mapping to real gates.

```
// Mux examples - Three ways to do the same thing.
 // The first example uses continuous assignment
wire out;
assign out = sel ? a : b;
 // the second example uses a procedure
// to accomplish the same thing.
 reg out;
always @(a or b or sel)
  begin
    case(sel)
      1'b0: out = b;
      1'b1: out = a;
    endcase
  end
 // Finally - you can use if/else in a
// procedural structure.
reg out;
always @(a or b or sel)
  if (sel)
    out = a;
  else
    out = b;
```

The next interesting structure is a transparent latch; it will pass the input to the output when the gate signal is set for "pass-through", and captures the input and stores it upon transition of the gate signal to "hold". The output will remain stable regardless of the input signal while the gate is set to "hold". In the example below the "pass-through" level of the gate would be when the value of the if clause is true, i.e. gate = 1. This is read "if gate is true, the din is fed to latch_out continuously." Once the if clause is false, the last value at latch_out will remain and is independent of the value of din.

```
EX6: // Transparent latch example
 reg out;
always @(gate or din)
 if(gate)
   out = din; // Pass through state
   // Note that the else isn't required here. The variable
   // out will follow the value of din while gate is high.
   // When gate goes low, out will remain constant.
```

The flip-flop is the next significant template; in Verilog, the D-flop is the simplest, and it can be modeled as:

```
reg q;
always @(posedge clk)
  q <= d;
```

The significant thing to notice in the example is the use of the non-blocking assignment. A basic rule of thumb is to use <= when there is a **posedge** or **negedge** statement within the always clause.

A variant of the D-flop is one with an asynchronous reset; there is a convention

that the reset state will be the first if clause within the statement.

```
reg q;
always @(posedge clk or posedge reset)
  if(reset)
    q <= 0;
  else
    q <= d;
```

The next variant is including both an asynchronous reset and asynchronous set condition; again the convention comes into play, i.e. the reset term is followed by the set term.

```
reg q;
always @(posedge clk or posedge reset or posedge set)
 if(reset)
   q <= 0;
 else
 if(set)
   q <= 1;
 else
   q <= d;
```

Note: If this model is used to model a Set/Reset flip flop then simulation errors can result. Consider the following test sequence of events. 1) reset goes high 2) clk goes high 3) set goes high 4) clk goes high again 5) reset goes low followed by 6) set going low. Assume no setup and hold violations.

In this example the always @ statement would first execute when the rising edge of reset occurs which would place q to a value of 0. The next time the always block executes would be the rising edge of clk which again would keep q at a value of 0. The always block then executes when set goes high which because reset is high forces q to remain at 0. This condition may or may not be correct depending on the actual flip flop. However, this is not the main problem with this model. Notice that when reset goes low, that set is still high. In a real flip flop this will cause the output to go to a 1. However, in this model it will not occur because the always block is triggered by rising edges of set and reset - not levels. A different approach may be necessary for set/reset flip flops.

Note that there are no "initial" blocks mentioned in this description. There is a split between FPGA and ASIC synthesis tools on this structure. FPGA tools allow initial blocks where reg values are established instead of using a "reset" signal. ASIC synthesis tools don't support such a statement. The reason is that an FPGA's initial state is something that is downloaded into the memory tables of the FPGA. An ASIC is an actual hardware implementation.

Initial Vs Always:

There are two separate ways of declaring a Verilog process. These are the **always** and the **initial** keywords. The **always** keyword indicates a free-running process. The **initial** keyword indicates a process executes exactly once. Both constructs begin execution at simulator time 0, and both execute until the end of the block. Once an **always** block has reached its end, it is rescheduled (again). It is a common misconception to believe that an initial block will execute before an always block. In fact, it is better to think of the **initial**-block as a special-case of the **always**-block, one which terminates after it completes for the first time.

```
//Examples:
initial
  begin
    a = 1; // Assign a value to reg a at time 0
    #1; // Wait 1 time unit
    b = a; // Assign the value of reg a to reg b
  end

always @(a or b) // Any time a or b CHANGE, run the process
begin
  if (a)
    c = b;
  else
    d = ~b;
```

end *// Done with this block, now return to the top (i.e. the @ event-control)*
always @(**posedge** a)*// Run whenever reg a has a low to high change*
a <= b;

These are the classic uses for these two keywords, but there are two significant additional uses. The most common of these is an **always**keyword without the **@(...)** sensitivity list. It is possible to use always as shown below:

always
begin *// Always begins executing at time 0 and NEVER stops*
clk = 0; *// Set clk to 0*
#1; *// Wait for 1 time unit*
clk = 1; *// Set clk to 1*
#1; *// Wait 1 time unit*
end *// Keeps executing - so continue back at the top of the begin*

The **always** keyword acts similar to the "C" construct **while(1) {..}** in the sense that it will execute forever.

The other interesting exception is the use of the **initial** keyword with the addition of the **forever** keyword.

Race Condition

The order of execution isn't always guaranteed within Verilog. This can best be illustrated by a classic example. Consider the code snippet below:

initial
a = 0;
initial
b = a;
initial
begin
#1;
$display("Value a=%b Value of b=%b",a,b);
end

What will be printed out for the values of a and b? Depending on the order of execution of the initial blocks, it could be zero and zero, or alternately zero and some other arbitrary uninitialized value. The $display statement will always execute after both assignment blocks have completed, due to the #1 delay.

7.7 **Operators**

Note: These operators are *not* shown in order of precedence.

Operator type	Operator symbols	Operation performed
Bitwise	~	Bitwise NOT (1's complement)
	&	Bitwise AND
	|	Bitwise OR
	^	Bitwise XOR
	~^ or ^~	Bitwise XNOR
Logical	!	NOT
	&&	AND
	||	OR
Reduction	&	Reduction AND
	~&	Reduction NAND
	|	Reduction OR
	~|	Reduction NOR
	^	Reduction XOR
	~^ or ^~	Reduction XNOR
Arithmetic	+	Addition
	-	Subtraction
	-	2's complement
	*	Multiplication
	/	Division
	**	Exponentiation (*Verilog-2001)
Relational	>	Greater than
	<	Less than
	>=	Greater than or equal to
	<=	Less than or equal to
	==	Logical equality (bit-value 1'bX is removed from comparison)
	!=	Logical inequality (bit-value 1'bX is removed from comparison)
	===	4-state logical equality (bit-value 1'bX is taken as literal)
	!==	4-state logical inequality (bit-value 1'bX is taken as literal)
Shift	>>	Logical right shift
	<<	Logical left shift
	>>>	Arithmetic right shift (*Verilog-2001)
	<<<	Arithmetic left shift (*Verilog-2001)
Concatenation	{ , }	Concatenation
Replication	{n{m}}	Replicate value m for n times
Conditional	? :	Conditional

System Tasks:

System tasks are available to handle simple I/O, and various design measurement functions. All system tasks are prefixed

with **$** to distinguish them from user tasks and functions. This section presents a short list of the most often used tasks. It is by no means a comprehensive list.

- $display - Print to screen a line followed by an automatic newline.
- $write - Write to screen a line without the newline.
- $swrite - Print to variable a line without the newline.
- $sscanf - Read from variable a format-specified string. (*Verilog-2001)
- $fopen - Open a handle to a file (read or write)
- $fdisplay - Write to file a line followed by an automatic newline.
- $fwrite - Write to file a line without the newline.
- $fscanf - Read from file a format-specified string. (*Verilog-2001)
- $fclose - Close and release an open file handle.
- $readmemh - Read hex file content into a memory array.
- $readmemb - Read binary file content into a memory array.
- $monitor - Print out all the listed variables when any change value.
- $time - Value of current simulation time.
- $dumpfile - Declare the VCD (Value Change Dump) format output file name.
- $dumpvars - Turn on and dump the variables.
- $dumpports - Turn on and dump the variables in Extended-VCD format.
- $random - Return a random value.

CONCLUSION

A Different architecture for data comparators were developed using VHDL for the targeted device XCV1000- 4bg560 using VHDL. It was found that three data comparators CSLA, CBC and 2BEC were chosen based on the area, speed and power consumed on the FPGA. Further to test the performance of the data comparators, the developed two cell sorters were applied on parallel and pipelined architecture for modified shear sorting and performance of the architecture was measured in terms of area speed and power. The Carry select logic based data comparator (CSLA) on parallel architecture of modified shear sorting requires less number of slices making it reduced area architecture. The conventional bitwise data comparator (CBC) on pipelined architecture was a high speed architecture and the twos complement using binary to excess one based data comparator (2BEC) on pipelined architecture was a low power architecture. Hence a variable architecture for image application in the form of rank order applications is proposed

FUTURE SCOPE

In future we can implement filters by using rank ordering comparators and also we can use this rank ordering comparators in testing technologies in testing approach.

REFERENCES

[1] Karpagaabirami. S,P. Ramamoorthy, " An Efficient VLSI Architecture for Removal of Impulse Noise in Images", International Journal of Computer Science and Mobile Computing, Vol. 3, Issue. 5, pp 567 – 574, May 2014.

[2] Aayisa Banu S, Ms. Divya R, Mr. Ramesh .K, "Design and Simulation of Low Power and High Speed Comparator using VLSI Technique", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 6, Issue. 1, pp 119 – 122, January 2017.

[3] Bharat H. Nagpara, Godhakiya Santosh M, Nagar Jay V, "Design and Implementation of Different types of Comparator", International Journal of Science, Engineering and Technology Research , Vol. 4, Issue. 5, pp 1321-1324 , May 2015.

[4] Mehmood ul Hassan, Rajesh Mehra ," Design Analysis of 1-bit CMOS

comparator", Proceedings of International Journal of Scientific Research Engineering & Technology, Vol. , Issue. , pp 68 – 72, 14-15 March 2015

[5] K.Vasanth, S.Karthik, S.Nirmal raj, Preetha mol. P, "FPGA implementation of optimized sorting networks for median filter", International Conference on robotics and automation, INTERACT 2010, Sathyabama university, Tamilnadu, India, pages 253-258, 2010.

[6] Keping CHAN, " Bit-Serial Realizations of a Class of Nonlinear Filters Based on Positive Boolean Functions" , IEEE Transactions on Circuits and Systems, Vol 36,no 6,pp 785-795, JUNE 1989.

[7] Vasanth.k, Kavirajan A.A.F, Ravi.T, NirmalRaj.S, " A Novel 8 bit digital comparator for 3x3 fixed kernel based modified shear sorting", Indian journal of science and technology, vol 7 ,issue 4, pp 452- 462, April 2014.

[8] K.Vasanth, V Elanangai, S Saravanan, G Nagarajan, "FSM-Based VLSI Architecture for the 3× 3 Window-Based DBUTMPF Algorithm", Proceedings of the International Conference on Soft Computing Systems: ICSCS 2015,Springer, Vol no 2,pp 234- 245, 2015.

[9] K.Vasanth, "VLSI Architecture of Decision based Modified Selection sort filter for Salt and pepper noise removal", International Journal on Intelligent Electronic System, Vol 13,no.4, Pages 41-56, August 2014.

[10] K.Vasanth and S.Karthik, "FPGA implementation of modified decomposition filter", International Conference on Signal and image processing, ICSIP 2010, Chennai, Tamilnadu, India, pages 526-530.

[11] Keshab K. Parhi, "Low-energy CSMT carry generators and binary adders", IEEE Transactions on Very Large Scale Integration (VLSI) Systems,volume 21, issue 4, pp791,april 2013.

[12]https://thunderwiring.wordpress.com/sorting-numbers/ [Accessed: 25- FEB-2019].