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

A High-Speed FPGA Implementation of an RSD-Based ECC Processor

Chimme Pedda Swamulu[1], S.Mahaboob Basha[2]
[1]P.G. Scholar, [2] Guide, Head of the Department
[1,2] BRANCH : VLSI
1,2,3 Geethanjali Engineering College,
Email.Id : [1]peddaswamy745@gmail.com, [2]syedmahaboob45@gmail.com

Abstract
In this paper, an exportable application-specific instruction-set elliptic curve cryptography processor based on redundant signed digit representation is proposed. The processor employs extensive pipelining techniques for Karatsuba–Ofman method to achieve high throughput multiplication. Furthermore, an efficient modular adder without comparison and a high throughput modular divider, which results in a short datapath for maximized frequency, are implemented. The processor supports the recommended NIST curve P256 and is based on an extended NIST reduction scheme. The proposed processor performs singlepoint multiplication employing points in affine coordinates in 2.26 ms and runs at a maximum requency of 160 MHz in Xilinx Virtex 5 (XC5VLX110T) field-programmable gate array.

Keywords
Application-specific instruction-set processor (ASIP), elliptic curve cryptography (ECC), field-programmable gate array (FPGA), Karatsuba–Ofman multiplication, redundant signed digit (RSD).

INTRODUCTION

ELLIPTIC curve cryptography (ECC) is an asymmetric cryptographic system that provides an equivalent security to the well-known Rivest, Shamir and Adleman system with much smaller key sizes. The basic operation in ECC is scalar point multiplication, where a point on the curve is multiplied by a scalar. A scalar point multiplication is performed by calculating series of point additions and point doublings. Using their geometrical properties, points are added or doubled through series of additions, subtractions, multiplications, and divisions of their respective coordinates. Point coordinates are the elements of finite fields closed under a prime or an irreducible polynomial. Various ECC processors have been proposed in the literature that either target binary fields prime fields or dual field operations.

In prime field ECC processors, carry free arithmetic is necessary to avoid lengthy datapaths caused by carry propagation. Redundant schemes, such as carry save arithmetic (CSA) redundant signed digits (RSDs) or residue number systems (RNSs) have been utilized in various designs. Carry logic or embedded digital signal processing (DSP) blocks within field programmable gate arrays (FPGAs) are also utilized in some designs to address the carry propagation problem. It is necessary to build an efficient addition datapath since it is a fundamental operation employed in other modular arithmetic operations.

Modular multiplication is an essential operation in ECC. Two main approaches may be employed. The first is known as interleaved modular multiplication using Montgomery's method. Montgomery multiplication is widely used in implementations where arbitrary curves are desired.

Another approach is known as multiply-then-reduce and is used in elliptic curves

built over finite fields of Merssene primes. Merssene primes are the special type of primes which allow for efficient modular reduction through series of additions and subtractions. In order to optimize the multiplication process, some ECC processors use the divide and conquer approach of Karatsuba–Ofman multiplications, where others use embedded multipliers and DSP blocks within FPGA fabrics.

Since modular division in affine coordinates is a costly process, numerous coordinate representation systems have been proposed to compensate this cost by means of extra multiplications and additions (e.g., Jacobian coordinates). Conversion back to affine representation can be mechanized using Fermat's little theorem. Such processors may implement a dedicated squarer to speed up the inversion process.

The complexity of modular division algorithms is approximately O(2n), where n is the size of operands and the running time is variable and depends directly on the inputs. This paper proposes a new RSD-based prime field ECC processor with high-speed operating frequency. The processor is an application-specific instruction-set processor (ASIP) type to provide programmability and configurability. In this paper, we demonstrate the performance of left-to-right scalar point multiplication algorithm; however, the ASIP feature of the processor allows different algorithms to be performed by the through read-only memory (ROM) programming. The overall processor architecture is of regular cross bar type with 256 digit wide data buses. The design strategy and optimization techniques are focused toward efficient individual modular arithmetic modules rather than the overall architecture. Such architecture allows for easy replacement of individual blocks if different algorithms or modular arithmetic techniques are desired. Different efficient architectures of individual modular arithmetic blocks for various algorithms are proposed. The novelty of our processor evolves around the following.

1) We introduce the first FPGA implementation of RSD-based ECC processor.
2) Extensive pipelining and optimization strategies are used to obtain a high-throughput iterative Karatsuba multiplier which lead to a performance improvement of almost 100% over the processor proposed.
3) To the best of our knowledge, the proposed modular division/inversion is the fastest to be performed on FPGA device. This is done through a new efficient binary GCD divider architecture based on simple logical operations.
4) A modular addition and subtraction is proposed without comparison.
5) Most importantly, exportable design is proposed with specifically designed multipliers and carry free adders that provided in competitive results against DSPs and embedded multipliers-based designs.

LITERATURE SURVEY

This survey begins by reviewing some of the previous studies in ECG signal feature extraction and analysis techniques. The survey is divided into feature extraction and classification techniques

ECG Feature Extractions

The first step in the analysis of ECG signal is the denoising of ECG signal. Denoising or pre-processing of ECG signal is important because noise severely limits the utility of the recorded ECG. After pre-processing, the second stage towards classification is to detect certain features of ECG signals mostly QRS complex, P and T waves. The features, which represent the classification information contained in the signals, are used as inputs to the classifier used in the classification stage.

The goal of the feature extraction stage is to find the smallest set of features that enables acceptable classification rates to be achieved. In general, the developer cannot

estimate the performance of a set of features without training and testing the classification system

Franc Jager developed a new approach to feature extraction which is Kahunen Lo‟eve Transform (KLT) which is an attractive and powerful approach to the feature extraction and shape representation process. It has the solution if the probability densities of population of pattern vectors of a problem domain are unknown. The problem about this method is, it is too sensitive to noisy pattern of ECG signal.

According to P. Ranjith et al. which used WT to detect myocardial ischemia, the WT is obtained using the quadratic spline wavelet. These correspond to the detection of T wave and P wave. Their methods showed a comparatively higher sensitivity and nominal positive predictivity value. It can be easily extended to detect other abnormalitites of the ECG signal. But this method also has the limitation that computations required are higher than those required by other methods.

According to M. H. Kadbi et al. highlighted those three features for feature extraction stage which are time frequency, 2-time domain features and 3-statistical feature. These features have been used in their project because these features can overcome the limitations of other methods in classifying multiple kinds of arrhythmia with high accuracy at once. These methods have been combined with PCA method to reduce the redundancy caused by the frequency coefficient in the feature dimension to make sure the average of the classification accuracy can be increased.

G.G. Herrero et al. used the independent component analysis and matching pursuits for the features extraction for extracting additional spatial features from multichannel electrographic recordings. It test the classification performance of 5 largest classes of heart beats in the MIT-BIH arrhythmia database which are normal sinus beats (NSB), left bundle branch block (LBBB), right bundle branch block (RBBB), premature ventricular contraction (PVC) and paced beats (PB). The performance of the system is remarkably good, with specificities and sensitivities for the different classes. They have a problem because the complicated separation between ventricular PBs and PVCs because of the inverted T wave.

K.S. Park et al. applied two morphological feature extraction methods which are higher order statistics and Hermite basis function. Their research results showed that hierarchical classification method gives better performance than the conventional multiclass classification method. They used the support vector machines to compare the feature extraction methods and classification methods to evaluate the generalization performance.

B. ECG Training and Classification Algorithms

In ECG training and classification analysis stages, some researchers have tried to maximize the detection level of accuracy in many different ways. The performance of the developed detection systems is very promising but they need further evaluation. The automatic detection of ECG waves is important to cardiac disease diagnosis. A good performance of an automatic ECG analyzing system depends heavily upon the accurate reliable detection of the disease.

Neural Network

The classification of the ECG using NNs has become a widely used method in recent years. The network architectures for modelling process modelling in NNs include the feed forward network, the radial basis function (RBF) network, recurrent network, and other advanced network architecture as explained by the Centre for Process Analytics and Control Technology (1999) and Sjoberg. The efficiency of these classifiers depends upon a number of factors including network training. It has the inputs models in the training parameters and the output indicated the point at which training should stop. Simple feed forward neuron

model was shown by Dayong Gao et al. Researcher from Harvard University, M. Sordo indicated that the training and testing of the models was based on the results from the signal database of the normal patient and heart disease patient.

Rosaria Silipo and Carlo Marchesi also developed an automatic ECG analysis based on ANN. This project presented the result by carrying out the classification tasks for the most common features of ECG analysis which are arrhythmia, mycordial ischemia and chronic alterations and achieved high classification accuracy.

Kei-ichiro Minami et al. developed a method to discriminate life threatening ventricular arrhythmias by observing the QRS complex of the electrocardiogram (ECG) in each heartbeat. Changes in QRS complexes due to rhythm origination and conduction path were observed with the Fourier transform, and three kinds of rhythms were discriminated by a neural network.

Neuro-Fuzzy Approach

So the Neuro Fuzzy is the most suitable technique because it is more tolerance to morphological variations of the ECG waveforms.

The Neuro-Fuzzy techniques which refers to the combinations of fuzzy set theory and neural networks with the advantages of both which can handle any kind of information, numeric, linguistic, logical, imperfect information, resolve conflicts by collaboration and aggregation, self-learning, self-organizing and self-tuning capabilities, no need of prior knowledge of relationships of data, mimic human decision making process and fast computation using fuzzy number operation in order to do the classification task.

Hidden Markov Models

This technique was successfully used since the mid 1970s to model speech waveforms for automatic speech recognition. The Hidden Markov modeling approach combines structural and statistical knowledge of the ECG signal parametric model. The model constructed contains multiple states per excitation field, model parameter ant training algorithms as explained by K. Seymore et al.

W.T. Cheng and K.L. Chan have discovered the method of Hidden Markov Model (HMM) in classifying arrhythmia. They have developed a fast and reliable method of QRS detection algorithm based on a one-pole filter which is simple to implement and insensitive to low noise levels. The disadvantage are that the observations are very sensitive to baseline wander, DC drift and heart rate variation.

Support Vector Machine

The Support Vector Machine based Expert System that have been described by C.J.C. Burges Stanislaw Osowski et al. and Van der C. M Walt and E. Barnard also the best method to apply in ECG analysis. The recognition system that uses the support vector machine (SVM) working in the classification mode. Support vector machine map input vectors to a higher dimensional space where a maximal separating hyperplane is constructed. Two parallel hyperplanes are constructed on each side of the hyperplane that separates the data. The separating hyperplane shows the maximize distance. The larger the distance between these parallel hyperplanes, the better the generalization error of the classifier.

Stanislaw Osowski et al. performed their studies of Heartbeat Regulation using SVM based Expert System. This recognition system has used the different preprocessing methods for generation of features which are higher order statistics (HOS) while the second is the Hermite characterization of QRS complex for the registered ECG waveform. Their paper presented the combination of multiple classifiers by the weighted voting principle. In their studies, stated that a good recognition system should depend on the features representing the ECG signals in such a way, that the differences among the ECG waveforms are suppressed for the waveforms of the same type but are emphasized for the waveforms

belonging to different types of beats. It is an important item, since the observed signal is a high variation of signals among the same type of beats.

Self-Organizing Map

Meanwhile in the ECG analysis of the Ischemia detection with a self organizing map supplemented by supervised learning has been developed in 2001 by Papadimitriou et al.. It is to solve problem of maximizing the performance of the detection of ischemia episodes. The basic selforganizing map (SOM) algorithm modified with a dynamic expansion process controlled with entropy based criterion that allows the adaptive formation of the proper SOM structure. This extension proceeds until the total number if training patterns that are mapped to neurons with high entropy reduces to a size manageable numerically with a capable supervised model. Then, a special supervised network is trained for the computationally reduced task of performing the classification at the ambiguous regions only. The utilization of SOM with supervised learning based on the radial basis functions and SVMs has resulted in an improved accuracy of the ischemia detection.

Fuzzy Logic

W. Zong and D. Jiang described the method of fuzzy logic approach single channel ECG beat and rhythm detection. The method summarized and makes use of the medical knowledge and diagnostic rules of cardiologists. Linguistic variables have being used to represent best features and fuzzy conditional statements perform reasoning. The algorithm can identify rhythms as well as individual beats. This method also handling the beat features and reasoning process is heuristic and seems more reasonable as stated in their paper. It also presented that this method may be of great utility in clinical applications such as multi-parameter patient monitoring systems, where many physiological variables and diagnostic rules exist.

Bayesian Method

Dayong Gao et al. pointed out that the Bayesian network are improved methods in determining the arrhythmia diagnosis system. This method is able to deal with nonlinear discrimination between classes, incomplete or ambiguous input patterns, and suppression of false alarms. It develops new detection schemes with a high level of accuracy. This approach is potentially useful for generating a pattern recognition model to classify future input sets for arrhythmia diagnosis.

M Wiggins et al. evolved a Bayesian classifier for ECG classification. The patients classification was according to statistical features extracted from their ECG signals using a genetically evolved Bayesian network classifier and the identification depend on the variables of interest. The Bayesian network has an ability to

Genetic Algorithm

Chris D Nugent et al. reported in depth on the prediction models in ECG classifiers using genetic programming approach. In their studies they developed the prediction models to indicate the point at which training should stop for NN based ECG classifiers in order to ensure maximum generalization. According to them, this good wave prediction could exhibit good generalization. They found that it could give benefit to developers of NNs, not only in the presented case of NN based ECG classifiers, but indeed any classification problems.

Autoregressive Model

Dingfei Geo et al. have extended the study of cardiac arrhythmia classification using autoregressive modeling. This computer-assisted arrhythmia recognition have been proposed to classify normal sinus rhythm (NSR) and various cardiac arrhythmias including atrial premature contraction (APC), premature ventricular contraction (PVC), super-ventricular tachycardia (SVT), ventricular tachycardia (VT) and ventricular fibrillation (VF). Their studies have shown the AR coefficients were classified using a generalized linear model (GLM) based algorithm in various

stages. From their study, they found that the AR modeling is useful for the classification of cardiac arrhythmias, with reasonably high accuracies. From the study, they found that AR modelling based classification algorithm has demonstrated good performance in classification. The algorithms are also easy to implement and the AR coefficient can be easily computed. AR modelling can lead to a low cost, high performance, simple to use portable telemedicine system for ECG offering a combination of diagnostic capability with compression. Therefore, it revealed that enhancement is suitable for real time implementation and can be used for compression as well as diagnosis.

INTRODUCTION TO VLSI

Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistor-based circuits into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. The term is no longer as common as it once was, as chips have increased in complexity into the hundreds of millions of transistors.

5.1 Overview:

The first semiconductor chips held one transistor each. Subsequent advances added more and more transistors, and, as a consequence, more individual functions or systems were integrated over time. The first integrated circuits held only a few devices, perhaps as many as ten diodes, transistors, resistors and capacitors, making it possible to fabricate one or more logic gates on a single device. Now known retrospectively as "small-scale integration" (SSI), improvements in technique led to devices with hundreds of logic gates, known as large-scale integration (LSI), i.e. systems with at least a thousand logic gates. Current technology has moved far past this mark and today's microprocessors have many millions of gates and hundreds of millions of individual transistors.

Current designs, as opposed to the earliest devices, use extensive design automation and automated logic synthesis to lay out the transistors, enabling higher levels of complexity in the resulting logic functionality. Certain high-performance logic blocks like the SRAM cell, however, are still designed by hand to ensure the highest efficiency (sometimes by bending or breaking established design rules to obtain the last bit of performance by trading stability).

VLSI and systems:

These advantages of integrated circuits translate into advantages at the system level:

- Smaller physical size. Smallness is often an advantage in itself-consider portable televisions or handheld cellular telephones.
- Lower power consumption. Replacing a handful of standard parts with a single chip reduces total power consumption. Reducing power consumption has a ripple effect on the rest of the system: a smaller, cheaper power supply can be used; since less power consumption means less heat, a fan may no longer be necessary; a simpler cabinet with less shielding for electromagnetic shielding may be feasible, too.
- Reduced cost. Reducing the number of components, the power supply requirements, cabinet costs, and so on, will inevitably reduce system cost. The ripple effect of integration is such that the cost of a system built from custom ICs can be less, even though the individual ICs cost more than the standard parts they replace.

Understanding why integrated circuit technology has such profound influence on the design of digital systems requires understanding both the technology of IC

manufacturing and the economics of ICs and digital systems.

Applications of VLSI:

Electronic systems now perform a wide variety of tasks in daily life. Electronic systems in some cases have replaced mechanisms that operated mechanically, hydraulically, or by other means; electronics are usually smaller, more flexible, and easier to service. In other cases electronic systems have created totally new applications. Electronic systems perform a variety of tasks, some of them visible, some more hidden:

- Personal entertainment systems such as portable MP3 players and DVD players perform sophisticated algorithms with remarkably little energy.
- Electronic systems in cars operate stereo systems and displays; they also control fuel injection systems, adjust suspensions to varying terrain, and perform the control functions required for anti-lock braking (ABS) systems.
- Digital electronics compress and decompress video, even at high-definition data rates, on-the-fly in consumer electronics.
- Low-cost terminals for Web browsing still require sophisticated electronics, despite their dedicated function.
- Personal computers and workstations provide word-processing, financial analysis, and games. Computers include both central processing units (CPUs) and special-purpose hardware for disk access, faster screen display, etc.
- Medical electronic systems measure bodily functions and perform complex processing algorithms to warn about unusual conditions. The availability of these complex systems, far from overwhelming consumers, only creates demand for even more complex systems.

ASIC:

An Application-Specific Integrated Circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed solely to run a cell phone is an ASIC. Intermediate between ASICs and industry standard integrated circuits, like the 7400 or the 4000 series, are application specific standard products (ASSPs).

As feature sizes have shrunk and design tools improved over the years, the maximum complexity (and hence functionality) possible in an ASIC has grown from 5,000 gates to over 100 million. Modern ASICs often include entire 32-bit processors, memory blocks including ROM, RAM, EEPROM, Flash and other large building blocks. Such an ASIC is often termed a SoC (system-on-a-chip). Designers of digital ASICs use a hardware description language (HDL), such as Verilog or VHDL, to describe the functionality of ASICs.

Field-programmable gate arrays (FPGA) are the modern-day technology for building a breadboard or prototype from standard parts; programmable logic blocks and programmable interconnects allow the same FPGA to be used in many different applications. For smaller designs and/or lower production volumes, FPGAs may be more cost effective than an ASIC design even in production.

- An application-specific integrated circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use.
- A Structured ASIC falls between an FPGA and a Standard Cell-based ASIC
- Structured ASIC's are used mainly for mid-volume level design. The design task for structured ASIC's is to map the circuit into a fixed arrangement of known cells.

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.

To Migrate a Project

1. In the ISE 12 Project Navigator, select **File > Open Project**.

1. 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.

2. In the dialog box that appears, select **Backup and Migrate** or **Migrate Only**.
3. The ISE software automatically converts your project to an ISE 12 project.
4. Implement the design using the new version of the software.

Note Implementation status is not maintained after migration.

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.

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**.

6.6 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.

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 iles, 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

1. Select **File > Copy Project**.
2. 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.

3. Enter a directory **Location** to store the copied project.
4. 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.

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

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

6. 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

6.10 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.

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

Click **OK**.

To Archive a Project:

Select **Project > Archive**.

1. In the Project Archive dialog box, specify a file name and directory for the ZIP file.
2. Optionally, select **Exclude generated files from the archive** to exclude generated files and non-project files from the archive.
3. 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 ofdigital logic chips at the register-transfer level of abstraction. It is also used in the

verification ofanalog and mixed-signal circuits.

Verilog's concept of 'wire' consists of both signal values (4-state: "1, 0, floating, undefined") and strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive a common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths.

History

Beginning

Verilog was the first modern hardware description language to be invented. It was created by Phil Moorby and Prabhu Goel during the winter of 1983/1984. The wording for this process was "Automated Integrated Design Systems" (later renamed to Gateway Design Automation in 1985) as a hardware modeling language.

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-2001 is the dominant flavor of Verilog supported by the majority of commercial EDA software packages.

Verilog 2005

Not to be confused with System Verilog, 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.

7.3 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.

7.5 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.

7.6 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.

7.8 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

In this paper, a NIST 256 prime field ECC processor implementation in FPGA has been presented. An RSD as a carry free representation is utilized which resulted in short datapaths and increased maximum frequency. We introduced enhanced pipelining techniques within Karatsuba multiplier to achieve high throughput performance by a fully LUT-based FPGA implementation. An efficient binary GCD modular divider with three adders and

shifting operations is introduced as well. Furthermore, an efficient modular addition/subtraction is introduced based on checking the LSD of the operands only. A control unit with add-on like architecture is proposed as a reconfigurability feature to support different point multiplication algorithms and coordinate systems. The implementation results of the proposed processor showed the shortest datapath with a maximum frequency of 160 MHz, which is the fastest reported in the literature for ECC processors with fully LUT-based design. A single point multiplication is achieved by the processor within 2.26 ms,

which is comparable with ECC processors that are based on embedded multipliers and DSP blocks within the FPGA. The main advantages of our processor include the exportability to other FPGA and ASIC technologies and expandability to support different coordinate systems and point multiplication algorithms.

REFERENCES

[1] N. Koblitz, "Elliptic curve cryptosystems," Math. Comput., vol. 48, no. 177, pp. 203–209, Jan. 1987.

[2] W. Stallings, Cryptography and Network Security: Principles and Practice, 5th ed. Englewood Cliffs, NJ, USA: Prentice-Hall, Jan. 2010.

[3] C. Rebeiro, S. S. Roy, and D. Mukhopadhyay, "Pushing the limits of high-speed GF(2m) elliptic curve scalar multiplication on FPGAs," in Proc. Cryptograph. Hardw. Embedded Syst. (CHES), vol. 7428. Jan. 2012, pp. 494–511.

[4] Y. Wang and R. Li, "A unified architecture for supporting operations of AES and ECC," in Proc. 4th Int. Symp. Parallel Archit., Algorithms Programm. (PAAP), Dec. 2011, pp. 185–189.

[5] S. Mane, L. Judge, and P. Schaumont, "An integrated prime-field ECDLP hardware accelerator with high-performance modular arithmetic units," in Proc. Int. Conf. Reconfigurable Comput. FPGAs, Nov./Dec. 2011, pp. 198–203.

[6] M. Esmaeildoust, D. Schinianakis, H. Javashi, T. Stouraitis, and K. Navi, "Efficient RNS implementation of elliptic curve point multiplication over GF(p)," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 8, pp. 1545–1549, Aug. 2012.

[7] D. M. Schinianakis, A. P. Fournaris, H. E. Michail, A. P. Kakarountas, and T. Stouraitis, "An RNS implementation of an Fp elliptic curve point multiplier," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 56, no. 6, pp. 1202–1213, Jun. 2009.

[8] J.-W. Lee, S.-C. Chung, H.-C. Chang, and C.-Y. Lee, "Efficient poweranalysis-resistant dual-field elliptic curve cryptographic processor using heterogeneous dual-processing-element architecture," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 22, no. 1, pp. 49–61, Feb. 2013.

[9] J.-Y. Lai and C.-T. Huang, "Energy-adaptive dual-field processor for high-performance elliptic curve cryptographic applications," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 19, no. 8, pp. 1512–1517, Aug. 2011.

[10] S.-C. Chung, J.-W. Lee, H.-C. Chang, and C.-Y. Lee, "A highperformance elliptic curve cryptographic processor over GF(p) with SPA resistance," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2012, pp. 1456–1459.

[11] J.-Y. Lai and C.-T. Huang, "Elixir: High-throughput cost-effective dualfield processors and the design framework for elliptic curve cryptography," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 16, no. 11, pp. 1567–1580, Nov. 2008.

[12] D. Karakoyunlu, F. K. Gurkaynak, B. Sunar, and Y. Leblebici, "Efficient and side-channel-aware implementations of elliptic curve cryptosystems over prime

fields," IET Inf. Secur., vol. 4, no. 1, pp. 30–43, Mar. 2010.

[13] D. Schinianakis and T. Stouraitis, Multifunction residue architectures for cryptography," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 61, no. 4, pp. 1156–1169, Apr. 2014.

[14] J. Vliegen et al., "A compact FPGA-based architecture for elliptic curve cryptography over prime fields," in Proc. 21st IEEE Int. Conf. Appl.-Specific Syst. Archit. Process. (ASAP), Jul. 2010, pp. 313–316.

[15] T. Güneysu and C. Paar, "Ultra high performance ECC over NIST primes on commercial FPGAs," in Proc. 10th Int. Workshop Cryptograph. Hardw. Embedded Syst. (CHES), 2008, pp. 62–78.

[16] P. L. Montgomery, "Modular multiplication without trial division," Math. Comput., vol. 44, no. 170, pp. 519–521, Apr. 1985.

[17] K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems," in Proc. 2nd Int. Workshop Reconfigurable Comput., Archit. Appl., vol. 3985. 2006, pp. 347–357.

[18] A. Byrne, E. Popovici, and W. P. Marnane, "Versatile processor for GF(pm) arithmetic for use in cryptographic applications," IET Comput. Digit. Tech., vol. 2, no. 4, pp. 253–264, Jul. 2008.

[19] J. Solinas, "Generalized Mersanne number," Univ. Waterloo, Waterloo, ON, Canada, Tech. Rep. CORR 99-39, 1999.

[20] B. Ansari and M. A. Hasan, "High-performance architecture of elliptic curve scalar multiplication," IEEE Trans. Comput., vol. 57, no. 11, pp. 1443–1453, Nov. 2008.

[21] N. Smyth, M. McLoone, and J. V. McCanny, "An adaptable and scalable asymmetric cryptographic processor," in Proc. Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP), Sep. 2006, pp. 341–346.

[22] C. J. McIvor, M. McLoone, and J. V. McCanny, "Hardware elliptic curve cryptographic processor over GF(p)," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 53, no. 9, pp. 1946–1957, Sep. 2006.

[23] K. Ananyi, H. Alrimeih, and D. Rakhmatov, "Flexible hardware processor for elliptic curve cryptography over NIST prime fields," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no. 8, pp. 1099–1112, Aug. 2009.

[24] M. Hamilton and W. P. Marnane, "FPGA implementation of an elliptic curve processor using the GLV method," in Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig), Dec. 2009, pp. 249–254.