Integrated extreme gradient boost with c4.5 classifier for high level synthesis in very large scale integration circuits

Thillai Rani M[1], *Rajkumar* R[2], *Sai Pradeep* K.P[3], *Jaishree* M [4] and *Rahul* S.G[5]

[1]Associate Professor, Department of ECE, Sri Krishna College of Technology, Coimbatore, Tamilnadu, India.

[2] Associate Professor, Department of ECE, Vel Tech Rangarajan Dr.Sagunthala R&D Institute of Technology, Chennai ,.

[3]Assistant Professor, Department of ECE, Dr.N.G.P Institute of Technology, Coimbatore, Tamilnadu, India.

[4]Assistant Professor, Department of ECE, Sri Ramakrishna Engineering College, Coimbatore, Tamilnadu, India.

[5] Assistant Professor, Department of ECE, Vel Tech Rangarajan Dr.Sagunthala R&D Institute of Technology, Chennai.

Abstract. High-level synthesis (HLS) is utilized for high-performance and energy-efficient heterogeneous systems designing. HLS is assist in field-programmable gate array circuits designing where hardware implementations are refined and replaced in target device. However, the power-process-voltage-temperature-delay (PPVTD) variation in VLSI circuits undergoes many problems and reduced the performance. In order to address these problems, C4.5 with eXtreme Gradient Boosting Classification based High Level Synthesis (C4.5-XGBCHLS) Method is designed for afford better runtime adaptability (RA) with minimal error rate. VLSI circuits are designed using the behavioral input and results are measured at running condition. When VLSI circuit's results get reduced, the language description of the circuit is considered as an input. Then, compilation process convert high level specification into Intermediate Representation (IR) in control/data flow graph (CDFG). CDFG computes data and control dependencies among operations. eXtreme Gradient Boosting (XGBoost) Classifier is exploited in C4.5-XGBCHLS method to classify the error causing functional unit (FU) with minimal error rate. XGBoost Classifier exploited C4.5 decision tree as base classifier to enhance classification of error causing FU in VLSI circuits. After that, FU gets allocated in place of error causing FU from functional library based on the design objectives and PPVTD variations. Finally, operation scheduling and binding process is executed for register transfer level (RTL) generation to form VLSI circuits with improved RA. The simulation results shows that the C4.5-XGBCHLS method enhances the performance of functional unit selection accuracy (FUSA) with minimal error rate (ER) and circuit adaptability time (CAT).

Keywords: High-level synthesis, eXtreme gradient boosting classification, C4.5 decision tree, functional unit, C4.5-XGBCHLS, register transfer level generation

1 Introduction

HLS algorithm is employed to synthesize the design depending on different technologies by different module libraries. In [1], GenFin include non-dominated logic circuits where timing, leakage power and dynamic power yields are established. However, RA for PPVDT variations was not considered. A multi-objective mixed-integer linear programming model (MOMILP) was designed in [2] to resolve complex issues. The designed MOMILP was employed to consider scheduling, allocation and binding algorithm. It integrated the decisions about multiplexers that are necessary components of integrated circuit. MOMILP model was addressed through the augmented ε-constrained method. But, the ER was higher. In order to overcome the issue, a novel method termed C4.5-XGBCHLS Method is designed.

In [3], a high-level data-flow optimization and synthesis method was presented. An optimization method was introduced through functional decomposition of multivariate polynomial to acquire good building blocks and vanishing polynomials to add/delete redundancy. But, the optimal FU was not selected for providing RA. Based on motion energy with VLSI architecture and inexpensive embedded systems, a bio-inspired visual motion assessment algorithm was designed in [4].The computational cost was reduced, the performance of VLSI circuits was not improved. An analytical optimization model was introduced in [5] depending on Integer Linear Programming for prediction of hardware cost. But, the time consumption was higher.

In [6], Power Shut-Off (PSO) was joined with model-based hardware flow to acquire automated Low Power-HLS (LP-HLS) methodology. The designed methodology was introduced for reducing the design effort and attain target system implementations. Design Space Exploration (DSE) methodology was introduced in [7]. But, the computational complexity remained unaddressed using DSE methodology. In [8] Dynamic MCML circuits integrate the benefits of MCML circuits with those that of the dynamic logic families for achieving great performance using a low-supply voltage along with low-power dissipation.

High level synthesis (HLS) methodologies were introduced in [9] with two approaches to lessen complexity and time consumption. But, the ER was higher. A set of new methods was introduced in [10] to enhance the HLS solutions and to enhance conventional design metrics. In [11], a transparent HLS approach with the scheduling and binding process was introduced. The issues reviewed from the above existing works are lesser FUSA, high computational complexity, higher ER, higher CAT, higher computational cost and so on. To resolve these issues, a novel method termed C4.5-XGBCHLS Method is designed.

The major contribution of the C4.5-XGBCHLS method is explained as,

- The main contribution of proposed C4.5-XGBCHLS method is introduced for improving the performance of VLSI circuits with better RA and lesser error rate.

- Compilation process changed language description of VLSI circuit to IR in terms of CDFG through code optimization.
- C4.5-XGBCHLS method applies the XGBoost Classifier to classify the error causing FU with minimal error rate. XGBoost Classifier used C4.5decision tree as base learner for finding the error causing FU in VLSI circuits. The entire C4.5 decision tree is combined to form the strong classification results. The novelty of Hinge loss function is used to measure the actual output and the predicted outcome with improve the classification accuracy and reduce the error rate.
- The new design of a suitable functional unit is allocated, scheduled and binded by the functional library to replace error causing functional units based on design objectives and PPVT constraints.
- The new initiative of RTL Generation is carried out for efficient VLSI design with better RA and minimal time consumption.

This article is ordered as follows: Related works are described in Section 2. Section 3 explains the brief description of the C4.5-XGBCHLS method. Simulation settings and results analysis of C4.5-XGBCHLS method are presented in section 4. Conclusion is given in Section 5.

High-level synthesis (HLS) is utilized for high-performance and energy-efficient heterogeneous systems designing. HLS is assist in field-programmable gate array circuits designing where hardware implementations are refined and replaced in target device. However, the power-process-voltage-temperature-delay (PPVTD) variation in VLSI circuits undergoes many problems and reduced the performance. In order to address these problems, C4.5 with eXtreme Gradient Boosting Classification based High Level Synthesis (C4.5-XGBCHLS) Method is designed for afford better runtime adaptability (RA) with minimal error rate. VLSI circuits are designed using the behavioral input and results are measured at running condition. When VLSI circuit's results get reduced, the language description of the circuit is considered as an input. Then, compilation process convert high level specification into Intermediate Representation (IR) in control/data flow graph (CDFG). CDFG computes data and control dependencies among operations. eXtreme Gradient Boosting (XGBoost) Classifier is exploited in C4.5-XGBCHLS method to classify the error causing functional unit (FU) with minimal error rate. XGBoost Classifier exploitedC4.5 decision tree as base classifier to enhance classification of error causing FU in VLSI circuits. After that, FU gets allocated in place of error causing FU from functional library based on the design objectives and PPVTD variations. Finally, Operation scheduling and binding process is executed for Register Transfer Level (RTL) generation to form VLSI circuits with improved RA. The simulation results shows that the C4.5-XGBCHLS method enhances the performance of functional unit selection accuracy (FUSA) with minimal error rate (ER) and circuit adaptability time (CAT).

2. Related Works

Hardware implementation of Jacobi algorithm was introduced in [12]. However, the CAT was not reduced using Jacobi algorithm. In [13], A parametric yield-driven resource binding algorithm was designed to categorize the power and delay distributions to improve power yield. But, delay and power variations failed to provide better RA in VLSI circuits.

Auto Pilot was introduced in [14] for unbiased performance, usability and productivity of HLS. Auto Pilot was employed by embedded benchmark kernels. For providing better HLS suitability of real-world applications, stereo matching techniques were exploited. But, the ER was not reduced using Auto Pilot. An integrated approach was introduced in [15] for high-level verification with formal HLS tool. But, the CAT was not reduced using integrated approach.

In [16], Bacterial foraging optimization algorithm (BFOA) was designed for design space exploration (DSE) of datapath in HLS. However, the computational cost was not reduced using BFOA.

The profit enhancement was carried out in [17]. But, the ER was not lessened. A new approach was introduced in [18] to create multi-cycle transient and multiple transient fault resilient design in HLS via dual modular redundancy. However, the computational complexity remained unaddressed.

In [19], a new designed methodology was designed for nested uneven, All loops synthesis via outer vectorization. The port assignment algorithm was introduced in [20] to identify the valid solution. However, the FUSA was not improved using port assignment algorithm.

For RDR architectures, a thermal-aware HLS algorithm was presented in [21]. The designed algorithm equalized the energy consumption between islands. The designed algorithm balance energy consumption between the islands and lessened the peak temperature. However computational complexity was more.

3. Methodology

HLS is an automatic design procedure which understands algorithmic description of particular behavior and creates digital hardware. Code is examined, limited and scheduled to produce RTL hardware description language (HDL). HLS main objective is to allow hardware designers to build and authenticate hardware via providing enhanced control over optimization of design architecture and explaining design at higher level of abstraction. The conventional HLS methods failed to afford RA in PPVTD variations.

For improving VLSI circuit's performance with better RA, C4.5-XGBCHLS Method is introduced. The main aim of C4.5-XGBCHLS method is to identify error causing FU to offer better RA with lesser ER. Architectural diagram of C4.5-XGBCHLS method is portrayed in Fig 1.

Initially, VLSI circuits are taken as input. Then, language description is given for input VLSI circuits. High-level transformation is to transform one behavioral description into another behavioral description. CDFG analysis indicates series of operations in DFG to implement specified behavior. To categorize error causing FU from other FU, Classification is performed. After classification, scheduling, allocation and binding are three essential processes for HLS. Scheduling is used to describe the states in finite-state machine. Each control step comprises one small section carried out in single clock cycle in hardware.

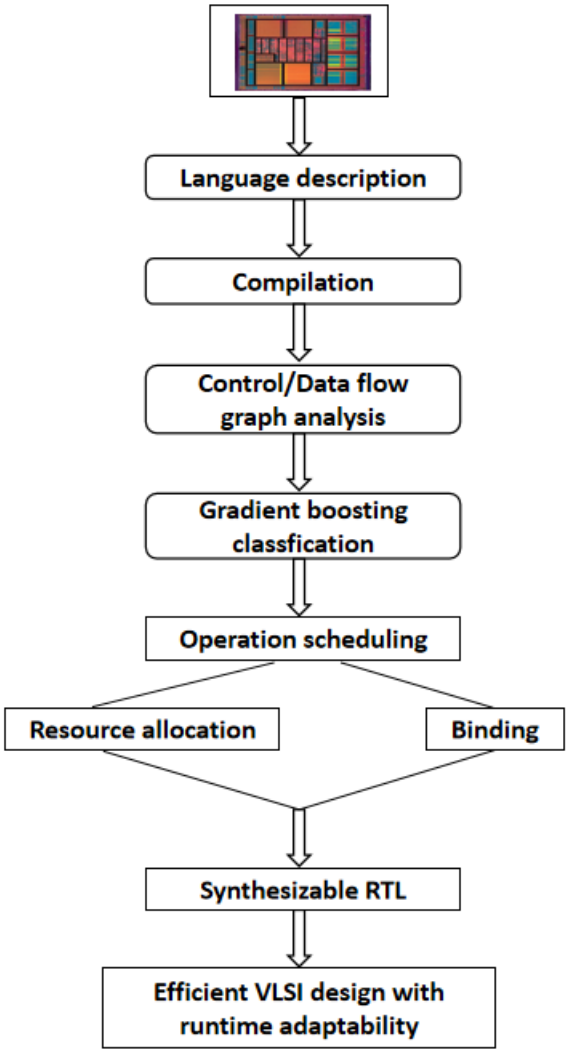

Fig. 1. Architectural diagram of C4.5-XGBCHLS method

Allocation and binding process in HLS maps the instructions and variables to hardware components, multiplexers and registers of the data path.

3.1 C4.5 with eXtreme Gradient Boosting Classification based High Level Synthesis (C4.5-XGBCHLS) Method

High level synthesis in C4.5-XGBCHLS method translates the behavioral specification of process into structural description. Structural description is provided in netlist at RTL. Synthesis indicates process of transforming digital system from behavioral specification into execution structure. Input to HLS process is provided in algorithmic-level specification. The specification offers necessary mapping from inputs sequences to sequences outputs. From input specification, synthesis system creates the data path description, registers, FU, multiplexers and buses. In HLS, essential steps of C4.5-

XGBCHLS method are behavioral analysis, classification, design-style selection, operation scheduling, data-path allocation and module binding.

3.1.1 Compilation

In C4.5-XGBCHLS method, compilation is a one-to-one transformation of initial specification into new internal representation of behavior for synthesis. Graphs are employed for internal representation. Code optimization is an initial process carried out by the compiler. The compiler enhances the quality of program in runtime called code optimization. Program enhancement includes the change of instruction sequence, removal of instructions and variation in instruction itself while maintaining the original code. The variations are termed as the code transformations. Code optimization comprises the four components, namely discovering transformation opportunities, safe transformation, profitable guaranteeing application and code rewriting. Code optimization lessens the runtime through evading the unnecessary computations. Code optimization minimizes the number of circuit modules with lesser area, lesser power and higher frequency. Compiler optimization comprises the constant propagation, dead-code elimination, sub-expression elimination, global flow analysis, inline development of subprograms and loop unrolling.

3.1.2 Control/Data Flow Graph Analysis

Control/data-flow analysis is employed for information gathering of estimated values at diverse points in computer program. CDFG identifies the parts of program where the particular value gets allocated to the variable. The information gathering is employed when optimizing program. The process of HLS initiates through examines data dependencies among different steps in algorithm. The analysis resulted in the Data Flow Graph (DFG) description.

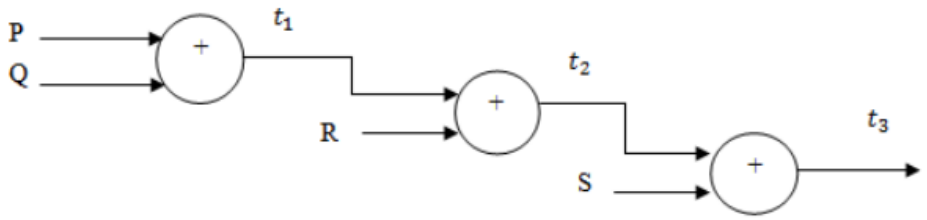

Fig. 2 Data Flow description

From Fig 2, 'P', 'Q', 'R' and 'S' are the inputs for the data flow. 't_3' is the output of the data flow description. The data-flow analysis set up data-flow equations for each control flow graph node and resolves them via calculating outcome from input at every node till system stabilizes. Every node of DFG symbolizes operation described in C++ code with add operator. The connection amid nodes indicates data dependencies and operations order. CDFG denotes design specification at various level than final hardware execution. Nodes symbolize hardware operators. It does not comprise any multiplexer's specification and handle logic needed for execution. Edges in CDFG signify the hardware value, the register value based on schedule.

3.1.3 eXtreme Gradient Boosting (XGBoost) Classification

By using decision trees, gradient boosting addresses the issues of regression and classification with prediction model. Gradient boosting creates model in stage-wise manner and generalizes via optimization of arbitrary differentiable loss function. XGBoost Classifier used Gradient Boosting concept to control the over-fitting.

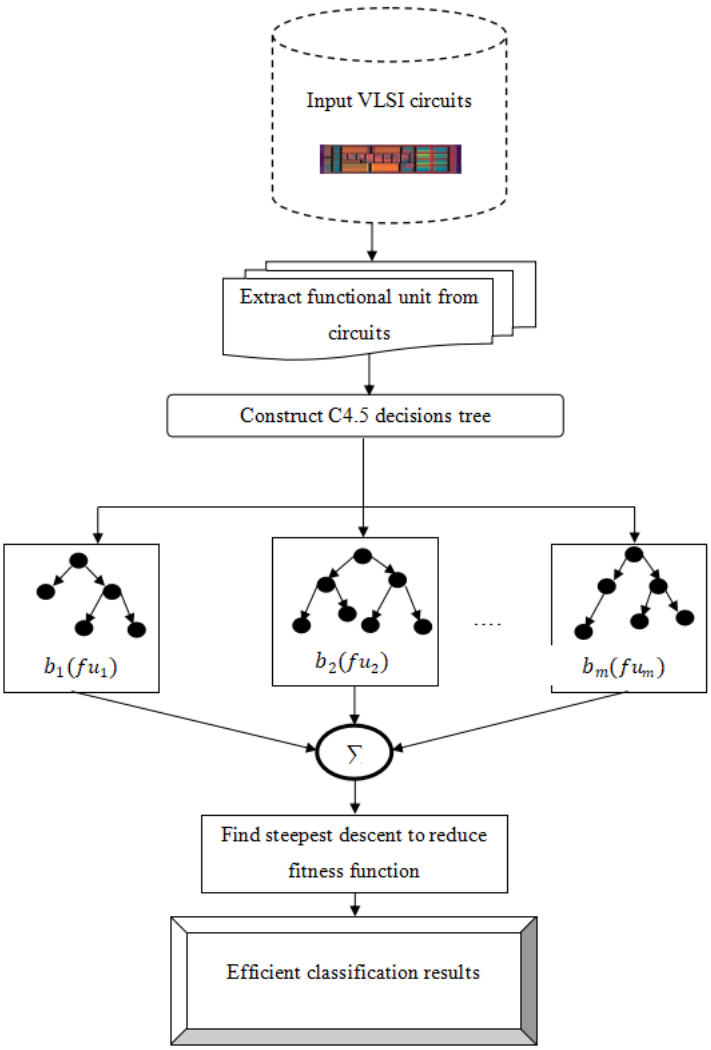

Fig. 3. Flow process of EXtreme Gradient Boosting(XGBoost) Classifier

In VLSI circuits, C4.5 with XGBoost Classifier categorizes the error causing FU from other FU with minimal ER for providing RA. The flow process of the C4.5 with XG Boost Classification is described in Fig 3. Fig 3 describes the XG Boost classifier with C4.5

decision tree to identify the error causing FU for providing RA. Let us consider the number of functional units'$fu_1, fu_2, fu_3 \ldots\ldots, fu_n$' is collected from the input VLSI circuits. It is given as,

$$fu_1, fu_2, fu_3 \ldots\ldots, fu_n \in C \quad (1)$$

From (1), 'C' represent the input VLSI circuits. XG Boost classifier model exploits number of weak learners and combines them to produce the strong classification results. XG Boost classifier model utilizes the training set $\{(fu_1, y_1), (fu_2, y_2), \ldots, (fu_n, y_n)\}$ where $'fu'$represents the input (i.e. functional unit) and 'y' represent the classifier output. In the C4.5 decision tree, the training set and test set are divided for predicting the class labels. The training set is considered as the root. In the tree, each node act as a test set for several attribute. Every edge descending from the node respective to the possible answers to the test set. This process is repeated for each subtree rooted in the fresh node. Predictive accuracy is based heavily on a choice of the test and training data. Then, the C4.5 decision tree is used to provide the better predictive accuracy. From Fig, C4.5 decision tree is utilized as base learner for ensemble boosting classifier. C4.5 creates decision trees from training set samples with information entropy. At each decision node tree, C4.5 pick attribute of data which partitions training set samples into subsets to produce classification outcomes. The splitting criterion is depends on normalized information gain. Information gain is computed as,

$$IG = h(fu_i) - \sum_{s \in fu} p(s)h(s) \quad (2)$$

From (2), 'IG'represents an information gain, $h(fu_i)$denotes entropy of FU, 's'is the subset produced from splitting set 'fu'of node. The FU entropy is given as,

$$H(fu_i) = \sum_{i=1}^{n} \sum_{fu_i \in y} -p(fu_i)p(fu_i) \quad (3)$$

From (3), '$p(fu_i)$'signifies probability of 'fu' belonging to class 'i'.When all functional units belong to the same class, it creates the leaf node for the decision tree to select that class. The process gets repeated through splitting the nodes and adding those nodes as children of that node. By this way, the classification results are produced for every C4.5 decision tree. The each decision tree results are considered as base or weak classifiers. However, the accurate classification was not carried out with minimum loss function. For enhancing classification accuracy, XG boost classifier computes loss function for C4.5 decision trees to create strong classifier. Strong classifier output is the summation of each individual C4.5 decision tree is given by,

$$y_i = \sum_{i=1}^{n} \sum_{m=1}^{k} b_m(fu_i) \quad (4)$$

From (4), 'y_i' represents the actual output of the strong classifier, b_m denotes an output of the base classifier for every functional unit. XG Boost classifier is given as,

$$f(x) = T_l + \sigma \quad (5)$$

From (5),'$f(x)$'represents fitness function, 'T_l'symbolizes the training loss, 'σ'denoted as the regularization term. The regularization term is exploited to reduce complexity. XGBoost classifiers lessen the training loss and evade over-fitting. XGBoost classifier calculates pseudo residuals for each iteration. Residual is described as gradient of mean squared error (i.e. hinge loss function).Hinge loss function is computed by,

$$H_l(fu_i) = \sum_{i=1}^{n} 1 - y_i b_m(fu_i) \quad (6)$$

From (6), '$H_l(fu)$' represents the hinge loss function, 'y_i' denotes an actual output, '$b_m(fu_i)$'is the predicted outcome. Based on loss function, pseudo residuals is calculated as,

$$\beta = -\left[\frac{\partial(y_i, H_l(fu_i))}{\partial H_l(fu_i)}\right] \quad \text{for } i = 1,2,3, \ldots . n \quad (7)$$

From (7), 'β'represent pseudo residuals for every functional unit. After that, fit base learner to the pseudo residuals. The steepest descent is used in pseudo residual values for identifying the minimum loss function of the base classifier. It is given by,

$$S_d = arg \sum_{i=1}^{n} [y_i, H_l(fu_{i-1})) + \tau b_m(fu_i)] \quad (8)$$

From (8), 'S_d'symbolizes the steepest descent step-size, argument minimum (arg min) function discover the minimum error of weak learner,'τ'represents the coefficient to identify the local minimum of the function. The predictive model is updated to categorize FU as error causing FU or normal FU to offer the RA. The updation model is defined as follows,

$$y_i = \sum_{i=1}^{n} H_l(fu_{i-1}) + S_d b_m(fu_i) \quad (9)$$

From (9) 'y_i'denotes the classification results of functional units. The attained strong classifier output categorizes FU for better RA. Ensemble of weak classifier is combined to create strong classifiers for discover the error causing FU.

Input: Number of functional units $fu_1, fu_2, fu_3 \ldots . , fu_n$
Output: Error causing the functional unit
Step 1: Begin
Step 2: **For** each training data 'fu_i'
Step 3: Construct a C4.5 decision tree using information gain and entropy
Step 4: Classify the fu_i based on y_i
Step 5: **For** each iteration
Step 6: Compute pseudo residuals β
Step 7: Fit base learner to the β
Step 8: Find steepest descent step-size
Step 9: Update the model 'y_i'
Step 10: Obtain strong classification results
Step 11: **End for**
Step 12: **End for**
Step 13: End

Algorithm 1 Extreme Gradient Boosting Classifier Algorithm

XGBoost classifier is illustrated in Algorithm 1 to classify the error causing FU from other FU. Let us take number of functional units from the input VLSI circuits. Initially, the FU in VLSI circuits is taken as an input. Decision tree classifies FU as error causing FU or normal unit by using information gain and entropy. After that, the XGBoost classifier combines all C4.5 decision trees to form strong classifier. The pseudo-residual is calculated and then base learner is fit to pseudo-residuals. The best steepest descent step

size is calculated to attain strong classifier results. Then, the ER is reduced. Consequently, a C4.5-XGBCHLS method effectively identifies the error causing FU.

3.1.4 Operation Scheduling

HLS includes design time during the process called scheduling. Operation scheduling is allocation of each operation to time slot depending on time interval. The task input comprises CDFG with hardware resources and limitations. Schedule created data control dependency and performance constraints are limited. Scheduling is operations explained in CDFG and choose when to be performed. Scheduling operations are allocated to similar time slot and affects concurrency degree. Maximum number of concurrent operations in schedule is minimal bound on crucial hardware resources. The schedules differ implementation cost and scheduling plays a crucial part in HLS.

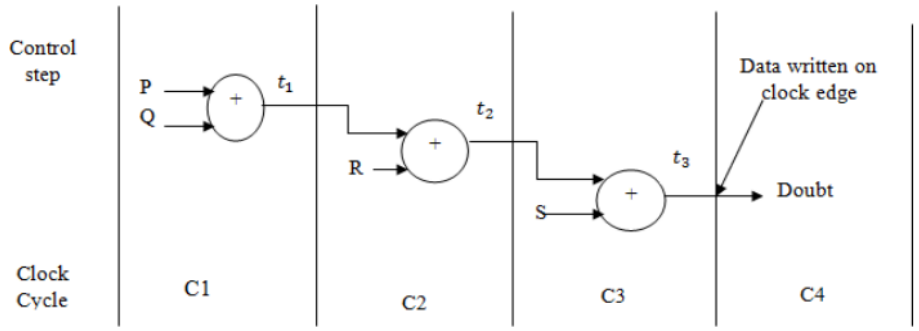

Fig.4. Operation Scheduling Process

From Fig 4, scheduling include effect of registers among operations with target clock frequency. Each add operation is scheduled with clock cycle C1, C2, C3 and C4. Registers are included among each adder.

3.1.5 Resource Allocation

Resource allocation manages issues where resources are utilized in physical implementation. Resources include registers, memory units and dissimilar FU and communication channels. Main objective is to distribute resources and additional design criteria are satisfied. Allocation and binding present's selection and hardware resources assignment for VLSI design to offer better RA. Allocation identifies type and hardware resources for design. From Fig 5, every operation is mapped onto the hardware resource during the scheduling process called as resource allocation. The resource connects to physical implementation of hardware operator. The implementation is annotated in scheduling process with timing and area information. Operator includes various hardware resource executions and different area/delay/latency trade-offs. Resources are selected from pre-characterized library which includes sufficient data points to indicate broad range of bit widths and clock frequencies. Designers handle resource allocation to include pipeline registers or limit accessible resources.

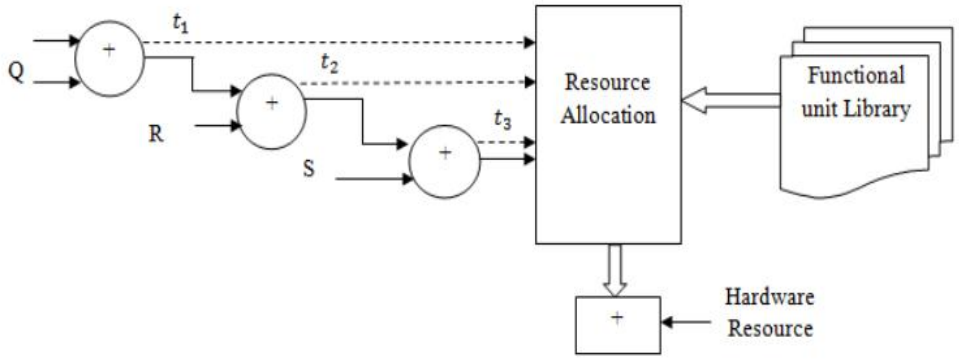

Fig. 5. Resource Allocation Process

3.1.6 Binding

Binding assigns the hardware instance resource to particular data path node. Data path operations distribute similar hardware resource if they are not performed simultaneously. An adder is shared with two additions as they are not performed during similar clock cycle. A register are employed to store values of two variables when lifetime of two variables do not overlap.

3.1.7 Synthesizable RTL

Finally, the RTL architecture gets synthesized by using all design decisions. Control generation synthesized the controller to create the suitable control signals with given schedule and binding of resource. As described in algorithm 2, VLSI circuit is taken as an input. PPVTD variations are monitored in VLSI circuits for HLS. Then, language description is compiled to form the CDFG. After that, the classification process classifies the error causing functional units due to PPVTD variation from other functional units. Allocation process select FU to meet design objectives. Then, scheduling and binding process provides improved RA. RTL gets synthesized to form efficient VLSI design with RA.

4. Simulation settings and result analysis

C4.5-XGBCHLS Method is implemented in MATLAB Simulink with 3.4 GHz Intel Core i3 processor, 4GB RAM, and windows 7 platform to offer RA via HLS in VLSI circuits. C4.5-XGBCHLS Method performed the experiment with ISCAS-89 benchmark circuits. ISCAS-89 benchmark circuits comprises four inputs, one output, three D-type flipflops, two inverters and eight gates(one AND + one NAND + two OR + four NOR). C4.5-XGBCHLS Method is designed for finding the error causing FU and compared with [1] and [2]. The C4.5-XGBCHLS Method is evaluated with metrics like ER, FUSA and CAT.

In the proposed HLS algorithm, the PPVTD measurement/estimation scenarios are include C4.5 with XGBoost Classifier. To conduct the experiments, the PPVTD variation settings comprise gate length, oxide thickness, fin thickness, and fin height. Let us consider VLSI circuits as input. The language description is taken from the VLSI circuits. The compilation process alters high level specification into IR in control/data flow graph (CDFG). CDFG

measures data and control dependencies between operations. The error causing FU is determined by using classification process. Therefore, error rate is minimized. XGBoost Classifier by C4.5 decision tree as base classifier for improving classification of error causing FU. Followed by, the scheduling, allocation, binding process are performed to choose the FU depended on the design objectives and PPVTD variations. Lastly, the efficient VLSI design is created by using RTL

Input: VLSI circuit

Output: High Functional Unit Selection Accuracy and Minimal Circuit Adaptability Time

Step 1: **Begin**

Step 2: **For** input VLSI circuit

Step 3: Monitor the 'PPVTD' variation of circuit

Step 4: Execute compilation process

Step 5: Build control and data flow graph

Step 6: Find error causing functional unit using eXtreme Gradient Boosting Classifier

Step 7: Perform functional unit allocation

Step 8: Execute functional unit scheduling

Step 9: Bind functional unit to form circuit

Step 10: Generate VLSI circuits

Step 11: **End for**

Algorithm 2 C4.5 with eXtreme Gradient Boosting Classification based High Level Synthesis Algorithm

.

4.1 Error Rate (ER)

ER is measured as difference of predicted value and obtained value to offer better RA based on PPVTD variation in VLSI circuits. It is calculated in percentage (%) and formulated as,

$$Error\ rate = \frac{Predicted\ value - obtained\ value}{Desired\ value} \quad (10)$$

Table 1 describes the results of ER for ten benchmark circuits. The ER performance of C4.5-XGBCHLS Method is compared to existing two methods namely GenFin Technique [1] and MOMILP Model [2] for providing the better RA. From the table, C4.5-XGBCHLS Method produces 14% ER for s9234 Benchmark Circuit while GenFin Technique [1] and MOMILP Model [2] attain 27% and 17%.

Table 1. Comparison of Error Rate

Benchmark Circuit Name	Error rate (%)		
	Existing GenFin Technique	Existing MOMILP Model	Proposed C4.5-XGBCHLS Method
s38584	25	20	12
s38417	22	17	7
s35932	21	15	5
s15850	24	19	9
s9234	27	22	14
s5378	23	20	8
s1494	28	25	15
s1488	25	23	12
s1423	27	25	14
s1238	21	15	6

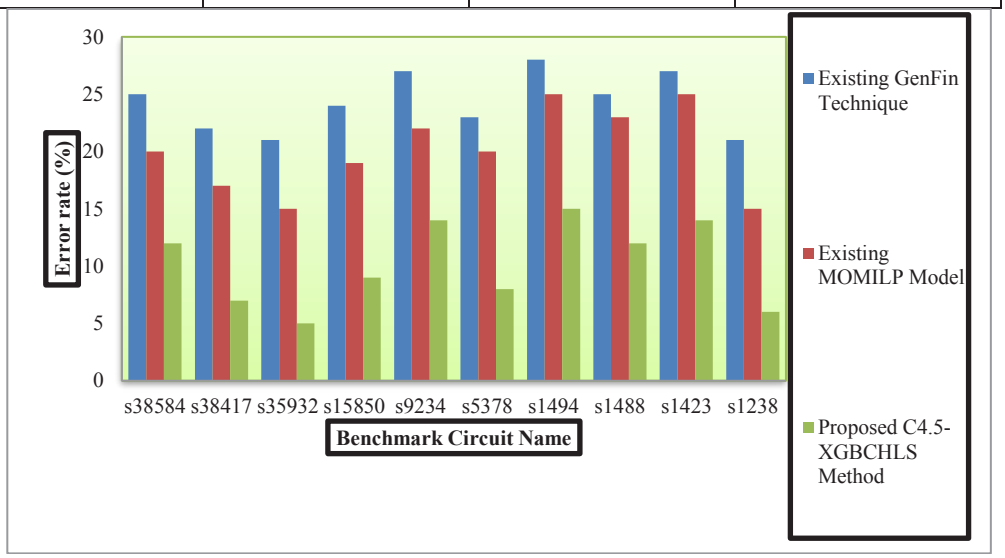

Fig 6. Comparison of Error Rate

Fig 6 shows the graphical representation of ER performance for ten different benchmark VLSI circuits. From Fig 6, ER using C4.5-XGBCHLS Method is lesser than existing [1] and [2]. This is because, the C4.5 with XGBoost classifier accurately discover the error causing FU in the VLSI circuits to offer improved RA during PPVTD variation with lesser ER.

Let us take ten different benchmark VLSI circuits for experimentation. C4.5 classifier classifies the error causing FU from other functional units in VLSI circuits. After that, XGBoost classifier boosts the C4.5 classifier performance. C4.5-XGBCHLS Method lessen ER by 59% and 33% than the existing [1] and [2].

4.2 Functional Unit Selection Accuracy (FUSA)

FUSA is defined as ratio of number of FU which are accurately selected based on PPVTD variation for providing better RA to total number of FU. It is computed in percentage (%) and given by,

$$FUSA = \frac{Number\ of functional\ unit\ correctly\ chosen}{Total\ number\ of\ functional\ units} * 100 \qquad (11)$$

From (11), the FUSA is determined. Higher FUSA, more efficient the method is said to be.

Table 2 describes the performance results of FUSA. FUSA of C4.5-XGBCHLS Method is compared to existing two methods namely GenFin Technique [1] and MOMILP Model [2] for High level synthesis in VLSI circuits. From the above table, C4.5-XGBCHLS Method produces 99% FUSA for s1488 Benchmark Circuit while GenFin Technique [1] and MOMILP Model [2]obtain 87% and 93% correspondingly. The FUSA gets changed for different benchmark circuits. Fig7 Shows the performance of FUSA. As shown in graph, it is clear that FUSA using C4.5-XGBCHLS Method is higher. Let us consider s1488 Benchmark Circuit for conducting the experiments in the first iteration. By applying C4.5-XGBCHLS Method, 95 functional units are accurately classified and the FUSA is 95%. By using existing GenFin Technique[1] and MOMILP Model [2], 84 functional unit, and 87 functional unit, are accurately classified and the FUSA is 95% respectively. For each method, ten different results are observed. The performance of the proposed C4.5-XGBCHLS Method is better than the other existing methods. XGBoost classifier is used in C4.5-XGBCHLS Method to classify the error causing FU for improving the performance of RA in VLSI circuits during PPVTD variation. After classification process, appropriate FU is assigned to offer improved RA.

Consider various benchmark circuits for HLS to provide better RA in PPVTD variations. For each input VLSI circuit, the FUSA is differed simultaneously.C4.5-XGBCHLSalgorithm classifies the error FU and selects suitable FU with higher accuracy. The FUSA of C4.5-XGBCHLS Method is improved by 14% and 6% than the existing [1] and [2].

Table 2. Comparison Functional Unit Selection Accuracy

Benchmark Circuit Name	Functional Unit Selection Accuracy (%)		
	Existing GenFin Technique	Existing MOMILP Model	Proposed C4.5-XGBCHLS Method
s38584	84	87	95
s38417	81	84	92
s35932	83	89	96
s15850	85	92	97
s9234	82	90	94
s5378	81	88	92
s1494	84	90	95
s1488	87	93	99
s1423	85	91	97
s1238	82	89	94

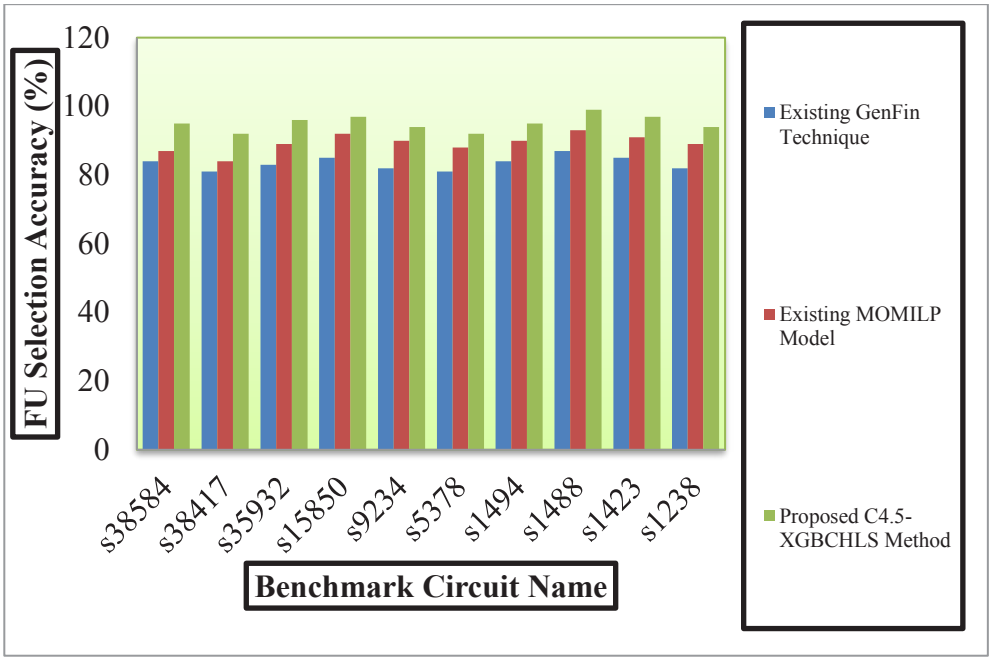

Fig 7. Comparison of Functional Unit Selection Accuracy

4.3 Circuit Adaptability Time (CAT)

CAT is described as difference of starting and ending time of circuit to getting adapted to PPVTD variation in VLSI circuits. In addition, it is given by total amount of time taken for providing better circuit adaptability. It is computed in milliseconds (ms).

$$CAT = Ending\ time - Starting\ time\ to\ get\ adaptable \quad (12)$$

From (12), CAT is computed. Minimal CAT, more effective the method is said to be.

Table 3. Comparison for Circuit Adaptability Time

Benchmark Circuit Name	Circuit Adaptability Time (ms)		
	Existing GenFin Technique	Existing MOMILP Model	Proposed C4.5-XGBCHLS Method
s38584	41	30	15
s38417	44	33	18
s35932	42	30	15
s15850	47	36	20
s9234	51	40	22
s5378	44	34	18
s1494	46	37	20
s1488	43	31	17
s1423	40	28	16
s1238	47	39	20

Table 2 explains results of CAT with various benchmark circuits. The CAT of C4.5-XGBCHLS Method is compared with existing two methods namely GenFin Technique [1] and MOMILP Model [2] for HLS in VLSI circuits. From above mentioned table, C4.5-XGBCHLS Method consumes 18ms for Circuit Adaptability in s38417Benchmark Circuit while GenFin Technique [1] and MOMILP Model [2] consume 44ms and 33ms correspondingly. The CAT gets varied for different benchmark circuits.

The result of CAT is portrayed in Fig 8 with different benchmark circuits.

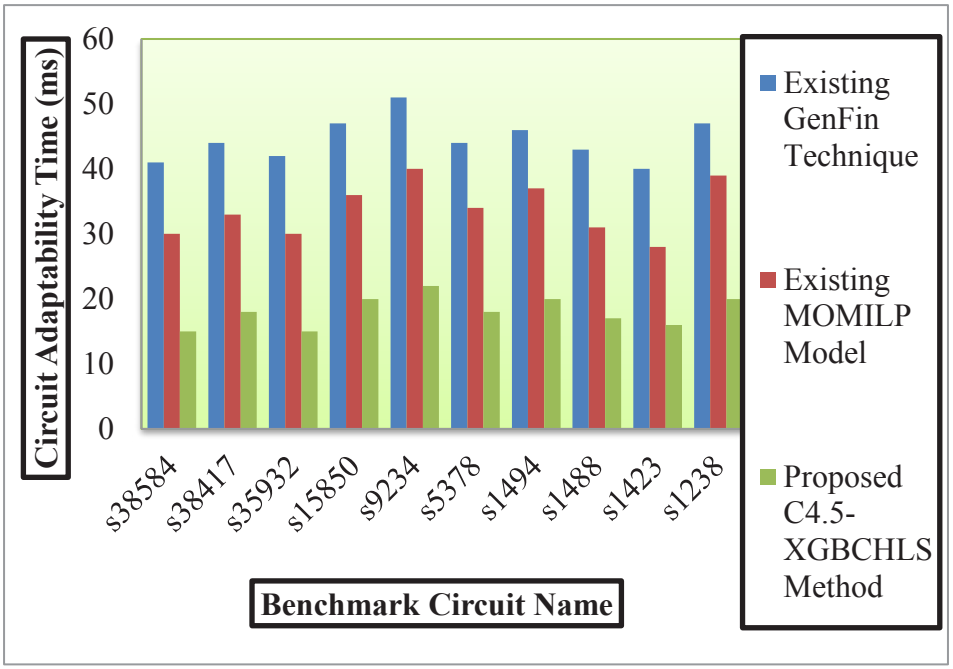

Fig. 8. Comparison of Circuit Adaptability Time

From Fig 8, the CAT using C4.5-XGBCHLS Method is minimal than existing [1] and [2] due to the use of XGBoost classifier. The designed classifier classifies the error causing FU accurately with minimal ER. After categorizing FU, suitable FU gets allocated and scheduled in place of error FUs. Finally, the FU gets bind in the VLSI circuits for providing better RA with lesser time.

Consider various VLSI circuits for HLS with improved RA in PPVTD variations. In each input benchmark circuit, CAT is varied for three techniques. XGBoost classifier classifies error causing FU and schedules suitable FU for circuit adaptability with lesser time consumption. The C4.5-XGBCHLS Method minimizes the CAT by 59% and 46% than the existing [1] and [2].

5. Conclusion

C4.5-XGBCHLS Method is designed for better RA in VLSI circuits. Initially, VLSI circuits are constructed with behavioral input and results are measured at running condition. In C4.5-XGBCHLS method, the language description of circuit is considered as an input and compilation process converts the high level specification into CDFG to reveal

dependencies between the operations. XGBoost Classifier classifies the error causing FU with lesser ER. Finally, suitable FU gets scheduled, allocated and binded from the functional library based on the PPVTD constraints for RTL Generation with better RA. From simulation results, the C4.5-XGBCHLS method improves the RA. The results ofC4.5-XGBCHLS method minimizes the ER by 46% and enhances the FUSA by 10% as compared to conventional works.

References

1. Tang, A., & Jha, N. K. (2016). GenFin: Genetic Algorithm-Based Multiobjective Statistical Logic Circuit Optimization using Incremental Statistical Analysis. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 24(3), 1126-1139.
2. Aras, N., & Yurdakul, A. (2016). A new multi-objective mathematical model for the high-level synthesis of integrated circuits. Applied Mathematical Modelling, 40(3), 2274-2290.
3. Ghandali, S., Alizadeh, B., Fujita, M., & Navabi, Z. (2015). Automatic High-Level Data-Flow Synthesis and Optimization of Polynomial Datapaths Using Functional Decomposition. IEEE Transactions on Computers, 64(6), 1-14.
4. Shi, C., & Luo, G. (2018). A Compact VLSI System for Bio-Inspired Visual Motion Estimation. 28(4), 1-16.
5. Cilardo, A., Gallo, L., & Mazzocca, N. (2013). Design space exploration for high-level synthesis of multi-threaded applications. Journal of Systems Architecture, 59(10), 1171-1183.
6. Qamar, A., Muslim, F. B., Iqbal, J., & Lavagno, L. (2017). LP-HLS: Automatic power-intent generation for high-level synthesis based hardware implementation flow. Microprocessors and Microsystems, 50, 26-38.
7. Prost-Boucle, A., Muller, O., & Rousseau, F. (2014). Fast and standalone Design Space Exploration for High-Level Synthesis under resource constraints. Journal of Systems Architecture, 60(1), 79-93.
8. Pradeep, K. P. S., & Kumar, S. S. (2019). Design and development of high performance MOS current mode logic (MCML) processor for fast and power efficient computing. Cluster Computing, 22, 13387-13395.
9. Imène, M., Mhadhbi, B. A., & Slim, B. C. (2012). High Level Synthesis Design Methodologies for Rapid Prototyping Digital Control Systems. IFAC Proceedings Volumes, 45(7), 238-243.
10. Campbell, K., Zuo, W., & Chen, D. (2017). New advances of high-level synthesis for efficient and reliable hardware design. Integration, 58, 189-214.
11. Wilson, D., Shastri, A., & Stitt, G. (2017). A High-Level Synthesis Scheduling and Binding Heuristic for FPGA Fault Tolerance. International Journal of Reconfigurable Computing, 2017, 1-17.
12. Bravo, I., Vázquez, C., Gardel, A., Lazaro, J. L., & Palomar, E. (2015). High Level Synthesis FPGA Implementation of the Jacobi Algorithm to Solve the Eigen Problem. Mathematical Problems in Engineering, 2015, 1-11.
13. Chen, Y., Wang, Y., Xie, Y., & Takach, A. (2012). Parametric Yield-Driven Resource Binding in High-Level Synthesis with Multi-Vth/Vdd Library and Device Sizing. Journal of Electrical and Computer Engineering, 2012, 1-14.
14. Liang, Y., Rupnow, K., Li, Y., Min, D., Do, M. N., & Chen, D. (2011). High-Level Synthesis: Productivity, Performance, and Software Constraints. Journal of Electrical and Computer Engineering, 2012, 1-14.
15. Dossis, M. F. (2015). High-level Synthesis Integrated Verification. Engineering, Technology & Applied Science Research, 5(5), 864-870.

16. Sengupta, A., & Bhadauria, S. (2014). Exploration of Multi-objective Tradeoff during High Level Synthesis using Bacterial Chemotaxis and Dispersal. Procedia Computer Science, 35, 63-72.
17. Zhao, M., Orailoglu, A., & Xue, C. J. (2015). Joint Profit and Process Variation Aware High Level Synthesis with Speed Binning. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 23(9), 1640-1650.
18. Kachave, D., & Sengupta, A. (2016). Integrating physical level design and high level synthesis for simultaneous multi-cycle transient and multiple transient fault resiliency of application specific datapath processors. Microelectronics Reliability, 60, 141-152.
19. Lattuad, M., & Ferrandi, F. (2017). Exploiting vectorization in high level synthesis of nested irregular loops. Journal of Systems Architecture, 75, 1-14.
20. Hao, C., Ni, J., Wang, N., & Yoshimura, T. (2017). Interconnection Allocation between Functional Units and Registers in High-Level Synthesis. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(3), 1140-1153.
21. Kawamura, K., Yanagisawa, M., & Togawa, N. (2013). A Thermal-Aware High-Level Synthesis Algorithm for RDR Architectures through Binding and Allocation. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E96(1), 312-321.