PROOF
DOI: 10.37394/232020.2024.4.11

AI-Driven WSN for Precise Aquatic Pollution Detection Using an Intelligent Monitoring Approach

V.V. JAYA RAMA KRISHNAIAH[1], P. G. K. SIRISHA[2], S. PARVATHI VALLABHANENI[3], D. V. V. CHANDRASHEKAR[4], K. JAGAN MOHAN[5], G. RAJESH CHANDRA[6], VENKATA KISHORE KUMAR REJETI[7]

[1]Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, INDIA.
[2, 7]Department of CSE, KKR & KSR Institute of Technology and Sciences, Guntur, INDIA.
[3]Department of IT, PVP Siddhartha Institute of Technology, Vijayawada, INDIA.
[4]Department of CSE, TJPS College, Guntur, INDIA.
[5]Department of AI, SVR Engineering College, Nandyal, INDIA.
[6]Department of CSE, Vignan's Lara institute of technology and science, Vadlamudi, INDIA.

Abstract: - The proliferation of digital devices, sensors, and interconnected systems has led to an explosion of data. Simple sensors like pH, conductivity, and Turbidity sensors can be used for the classification of gasoline and diesel in water. These sensors are easy to set up and deploy, so they can be installed in vast numbers and can get real-time data from the site. FPGAs are very fast in processing complex data and have low latency time compared to other traditional microcontrollers. But FPGA accepts coding in Hardware Description Languages like Verilog or VHDL, which can be very complex to code complex models that are trained in other high-level languages like Python and C++. Simple classification models in machine learning are implemented using High-Level Synthesis tools, which accept codes written in languages like C, C++, or SystemC, and translate them into hardware-implementable RTL (Register- Transfer Level) code. The data from two major components of oil, gasoline, and petrol are used to train various classification models with widely used libraries in Python. The trained parameters are extracted from the trained model. The parameters are then assembled and then coded in C++ as currently most of the tools support C++. Some modifications need to be made to the original code to make it compatible with the synthesis tool.

Key-Words: - Digital Sensors, FPGA, High-Level Synthesis (HLS), Machine Learning, Real-Time Data Processing.

Received: April 29, 2024. Revised: September 19, 2024. Accepted: November 11, 2024. Published: December 10, 2024.

1 Introduction

In recent years, the field of machine learning (ML) has witnessed unprecedented growth and impact across various domains. One major application explored is oil spill, where early detection is essential to minimize damage in aquatic environment [1]. Machine learning, a branch of artificial intelligence, is centered on creating algorithms that allow computers to learn from data and make predictions or decisions without explicit programming. Supervised learning, a key aspect of machine learning, involves training a model using labeled data, where the algorithm learns to associate input data with the correct output labels [2]. Field-Programmable Gate Arrays (FPGAs) are integrated circuits (ICs) that can be programmed to carry out specific tasks tailored to particular applications [3].

As machine learning and deep learning models are get- ting more complex, more powerful embedded systems like FPGA are used [4]. Machine learning models are implemented in FPGAs stemming from the need for high-performance, low-power solutions for real-time inference tasks in edge and embedded systems. Traditional computing platforms may struggle to meet the stringent requirements of

these applications, especially in scenarios where power consumption, latency, and resource constraints are critical factors. FPGAs offer a promising solution by providing hardware acceleration for ML algorithms, enabling efficient parallel processing and low-latency inference. Furthermore, FPGA-based ML imple- mentations offer flexibility and scalability, allowing for customization and optimization of hardware architectures tailored to specific application requirements [5]. This flexibility is particularly advantageous in domains where standard ML models need to be adapted or optimized for specialized tasks or hardware platforms. The major factor that is holding back in using FPGA is using of Hardware description language which can be very complex and requires a deep understanding of digital design concepts and hardware architectures, making them challenging for beginners and those with primarily software engineering backgrounds. Making design changes and exploring alternative architectures in HDLs can be time- consuming and resource-intensive, particularly for complex designs [6], [7].

High-Level Synthesis (HLS) is a design methodology used in digital circuit design and FPGA programming. It allows developers to write high-level descriptions of digital circuits using programming languages such as C, C++, or SystemC, which are then automatically translated into hardware implementations. HLS abstracts the details of hardware design, allowing developers to focus on algorithmic and architectural aspects rather than low-level hardware descriptions. Developers can write algorithms and specify behavior at a higher level of abstraction, making the design process more intuitive and efficient. HLS significantly reduces the time and effort required for designing complex digital circuits compared to traditional hardware description languages (HDLs) like Verilog or VHDL.

In environmental monitoring and management, the detection and mitigation of oil spills in water bodies are of paramount importance to safeguard ecosystems, human health, and economic activities. Leveraging advancements in sensor technology and data analytics, there has been a growing interest in developing automated monitoring systems for early detection and response to oil spills. Less expensive sensors like pH, conductivity, and Turbidity sensors can be used for real-time observation on site. By integrating sensor data with machine learning algorithms, predictive models can be developed to detect unusual patterns that may signal the occurrence of oil spills, enabling timely intervention and mitigation efforts.

The research paper is structured as follows: After the introduction, Section II presents a review of related work. Sections III and IV cover the preliminaries and the proposed methodology, respectively. Section V discusses the computational experiments and their analysis. Finally, Section VI concludes the study.

2 Literature Survey

With the increasing significance of evolving artificial intelli- gence, many researchers are developing prototypes to get input from low-cost and portable sensors and perform computations using machine learning models. These days, machine learning models can easily be integrated into multiple platforms.

Yan Ferreira da Silva et al., utilized Wireless Sensor Net- works (WSN) with sensors for temperature, pH, turbidity, and conductivity on a float to continuously gather real-time data. This data is sent to an ESP32 microcontroller that converts analog signals to digital. They employed a convolutional neural network to train the model with gasoline and diesel data, transmitting the output wirelessly via Wi-Fi for real- time monitoring and alerts for oil presence. While the network detects the presence of oil, it doesn't give what kind of oil is present [8]. Ahmed et al. developed an oil spill segmentation model utilizing 50 Sentinel-1 SAR images, employing a custom data generator within a Seg-Net model, which is implemented through a

Conditional Generative Adversarial Network (CGAN). In this model, a modified Seg-Net functions as the generator, while a Patch-GAN serves as the discrim- inator, significantly improving oil spill segmentation results compared to the Seg-Net model, even with a relatively small training dataset. However, obtaining the data from the radar is a complex process that consumes lots of power and storage for processing [9].

Yi-Jie Yang et al., developed a deep learning-based oil spill detector using Sentinel-1 Synthetic Aperture Radar (SAR) imagery. The authors employed advanced deep-learning tech- niques to analyze SAR images for identifying oil spills. Their approach leverages the high-resolution and wide coverage of Sentinel-1 data to detect oil spills accurately and efficiently. The model was trained on a substantial dataset of SAR images, enabling it to distinguish between oil spills and look-alike features such as natural phenomena or man-made objects. But models, especially those handling large datasets like Sentinel- 1 SAR imagery, require significant computational resources and time for training and inference [10]. Gianluca Tabella et al., have developed Wireless Sensor Networks (WSNs) aimed at detecting and pinpointing subsea oil leaks. This research emphasizes sensors making localized binary decisions on the existence of a spill through an energy test. These individual decisions are then sent to a Fusion Center (FC), which integrates them into a more reliable global binary decision. The study evaluates the performance of the Counting Rule (CR) against a modified Chair-Varshney Rule (MCVR). Thresholds are designed using an objective function that is based on the Receiver Operating Characteristic (ROC) curve. The main challenges faced in this design are maintaining reliable wireless communication underwater is difficult due to signal attenuation and interference, which can impact data transmission, and harsh underwater conditions, such as high pressure, strong currents, and biofouling, can affect sensor performance and durability [11].

Yan Li et al., introduced a deep-learning classification model designed to automatically detect marine oil spills in images from Landsat-7 and Landsat-8 satellites. This model inte-grates fully convolutional networks (FCN) with ResNet and GoogLeNet architectures. The performance of the classifica- tion algorithms, namely FCN-GoogLeNet and FCN-ResNet, is compared against the state-of-the-art Support Vector Machine (SVM) method. But the computational intensity of FCNs may limit their applicability in real-time monitoring scenarios where rapid detection and response are crucial. Also, the model might struggle to adapt to different environmental conditions not represented in the training data, such as varying weather patterns, sea states, or types of oil spills [12]. Thomas De Kerf et al., developed an innovative framework for de- tecting oil spills in port environments using unmanned aerial vehicles (UAVs) equipped with thermal infrared (IR) cameras, capable of detecting oil even at night. The collected data is uti- lized to train a convolutional neural network (CNN). However, the accuracy of infrared imaging for oil spill detection can be influenced by environmental factors such as temperature fluctuations, weather conditions, and sea state. Additionally, implementing and maintaining machine learning models for this purpose requires expertise in both machine learning and infrared imaging, which may not be readily available [13].

In a separate study, Zahra Ghorbani and Amir H. Behzadan employed VGG16 transfer learning convolutional neural net- works to train on a visual dataset of oil spills, comprising images taken from different altitudes and geographic regions. They employed Mask R-CNN and PSPNet models for oil spill segmentation and precise pixel-level detection of spill boundaries. Additionally, they trained a YOLOv3 model to identify the presence of oil rigs or vessels near detected oil spills, providing a comprehensive view of the spill area. A significant challenge in this approach is the deployment in resource-limited environments, as training and implementing multi-class CNNs demand

substantial computa- tional resources, which may not always be feasible [14]. M. Konik and K. Bradtke used an object-oriented approach to oil spill detection using ENVISAT/ASAR images. Their approach involves pre-processing using optimized filters, hierarchical segmentation to detect spills of different sizes, forms, and homogeneity, and a decision-tree procedure for classifying dark objects visible in SAR images reflecting the probability of oil spill presence. This model relies heavily on accurate image segmentation, which can be challenging and may result in errors if the segmentation process is not precise. Also, the presence of similar-looking features such as natural slicks, seaweed, and low-wind areas can interfere with the detection process, leading to false positives [15].

3 Preliminary

A. Machine Learning Classification Models

In this work, binary classification models are used which will give the output as true or false. Classification models can be generally divided into three types: binary classification, multi-class classification, and multi-label classification. These categories are determined by the number of classes or labels present in the target variable [16].

1) Logistic Regression: Logistic regression is a key classification algorithm frequently employed in machine learning for binary classification tasks. It assesses the degree of dependence between a categorical dependent variable and one or more independent variables by utilizing the logistic function. The result of logistic regression is a probability value ranging from 0 to 1, indicating the likelihood or confidence that a given input belongs to the positive class. Similar to linear regression, it aims to determine the coefficients for the input variables. However, in logistic regression, the output is transformed using a non-linear (logistic) function. This transformation is achieved through the logistic (or sigmoid) function, which converts raw predictions into probabilities. Let the independent input features be:

2) K Nearest Neighbors: K-Nearest Neighbors (KNN) is a straightforward and intuitive algorithm used for both classifi-cation and regression in machine learning. It is an instance- based method, meaning it memorizes the training instances and predicts new instances based on their similarity to these stored instances. The algorithm finds the 'K' nearest data points to a given input and predicts the output based on the majority class or average value of those neighbors. As a non-parametric algorithm, KNN does not assume any underlying data distribution.

To make predictions, KNN calculates the similarity between the input instance and all training instances, often using Euclidean distance. For classification, it uses majority voting among the k nearest neighbors to assign the class label. The choice of 'K' and the distance metric significantly impacts the algorithm's performance. The optimal value of 'K' is critical and often chosen based on input data characteristics, with higher values of 'K' recommended for data with more noise

3) Support Vector Machines: Support Vector Machines (SVMs) are robust supervised learning models used for both classification and regression. They excel in high-dimensional spaces and when the number of features exceeds the number of samples. SVMs work by identifying the optimal hyper plane that best separates data points of different classes in the feature space, maximizing the margin between the nearest points of the classes. This optimal hyper plane is known as the maximum-margin hyper plane or hard margin. In cases with outliers, SVM introduces a penalty for misclassifications, resulting in a soft margin. SVMs can also handle non-linear data using the kernel trick, which maps input features into a higher-dimensional space where linear separation is possible. Common kernels include linear, polynomial, radial basis function (RBF), and sigmoid. The regularization parameter (C) balances margin maximization and classification error minimization, with smaller

values allowing more misclassifications and larger values penalizing them more heavily.

4) Decision Tree: Decision Trees are flexible

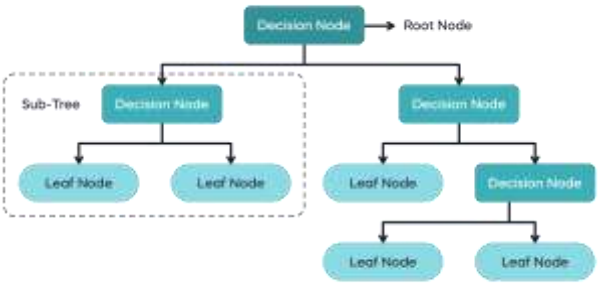

Fig. 1. Basic Decision Tree Node Representation

and interpretable models used for classification and regression tasks. They work by recursively dividing the feature space into distinct regions, each corresponding to a specific class or predicted value. The structure of a Decision Tree includes a root node, internal nodes, and leaf nodes shown in Fig. 1. The root node represents the feature that best separates the data, while internal nodes make decisions based on feature values and thresholds. Leaf nodes provide the final prediction for the data subsets. Decision Trees use criteria like Gini impurity, entropy (information gain), and classification error to determine the optimal splits at each node. To avoid overfitting, pruning techniques are applied: pre-pruning stops tree growth early based on certain conditions, while post-pruning removes less significant branches after the tree is fully grown.

B. Conversion into C++ code

Currently, most of the High-Level Synthesis tools support C++ as its input. Therefore, the parameters from the trained Python models are extracted and refactored into C++. This C++ code can be fed into the Vitis HLS tool, which is converted into RTL. The parameters are obtained from the trained models and then C++ code is developed.

C. High-Level Synthesis Tools

called Vitis HLS. The steps followed for the generation of RTL from the HLS code and the deployment of RTL code in FPGA:

Step 1: Running C simulation to verify if the provided code is syntactically correct and performs as per the user's requirement.

Step 2: Running C synthesis for the conversion of the provided C++ code to RTL using the Vitis conversion tool.

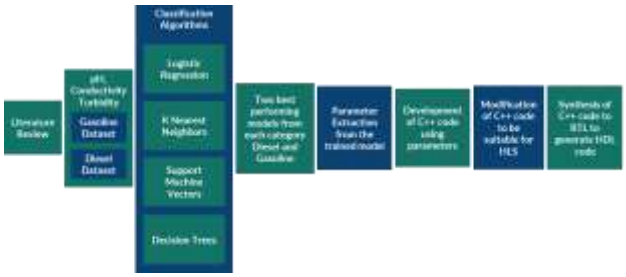

Fig. 2. Block diagram of workflow

Step 3: The C/RTL Co-Simulation step verifies that the RTL generated by synthesis behaves the same as the original C/C++ source code.

Step 4: Exporting RTL for integration with hardware tools like Vivado for Xilinx FPGAs. The system-level design is then synthesized into a bitstream that can be loaded onto the FPGA.

4 Proposed Work

The block diagram shown in Fig. 2, the approach and methodology developed for the proposed work:

A. Generation Dataset

For data preparation, for the detection of gasoline, the pH range is between 6.5 to 7 and turbidity is between 2.36 to 2.42 NTU. The conductivity of seawater is typically 50000 microS/cm. As gasoline is volatile and lightweight the conductivity will reduce only by 1 to 3 %. With the range of seawater pH between 5 to 8 and turbidity between 2.3 to 3 NTU the dataset is generated in Python using CSV format. For the detection of diesel, the pH range is between 5.8 to 6.5 and turbidity is between 2.28 to 2.35 NTU.

B. Training Classification Models

The model is trained using Scikit-Learn, an open-source Python library known for its user-friendly and efficient tools for data mining and analysis. Scikit-Learn offers a variety of supervised and unsupervised learning algorithms, as well as features for model selection, evaluation, and data preprocess- ing. In this project, four classification models are implemented: Logistic Regression, K-Nearest Neighbors, Support Vector Machines, and Decision Trees. The two datasets, diesel and gasoline are fed into the four classification models and the accuracy is calculated. The following table gives the accuracy of the classification models for diesel and gasoline.

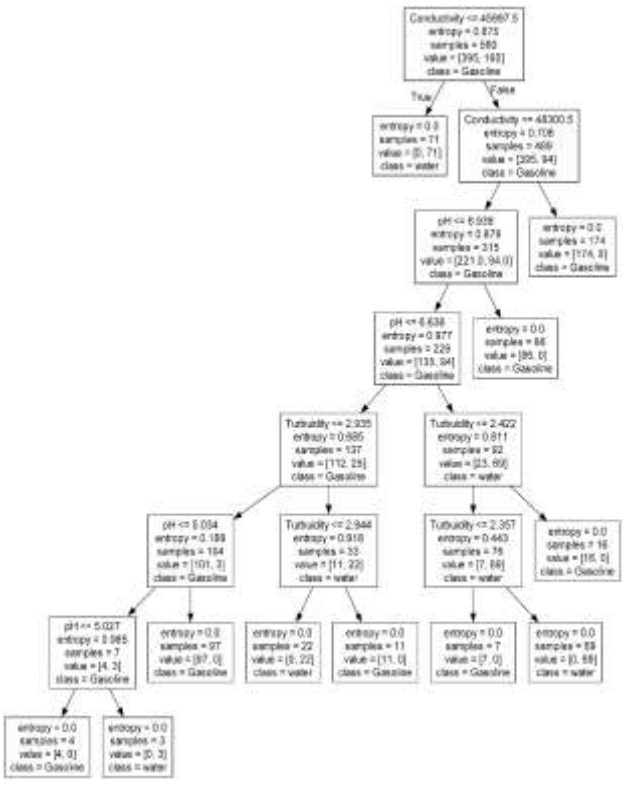

Fig. 3. Decision Tree Node Structure Obtained after Parameter Extraction

C. Parameter Extraction and C++ code generation

For the gasoline dataset, the decision tree performs very well for classification. The decision tree model breaks the model into various nodes based on various nested conditions based on the model fit and Gini impurity. These conditions are then inferred and C++ code is developed. This decision function is implemented in the C++ code. The parameters are then extracted are: Support Vectors (xi) Lagrange multipliers (i) Class Labels (1 or -1) Bias term (b) Gamma.

D. Methodology adopted to code the trained model in C ++

• For the Decision Tree model, break the different conditions using if-else statements according to the value threshold provided. The model either returns true or false shown in Fig. 3.

• Define and initialize the constant values like support vectors, Lagrange multipliers, and bias term gamma. Also, the mean and variance of each feature are assigned.

• For the Support Vector Machine model, separate function modules are created for the RBF function and decision func- tion. The RBF function is computed and the value is passed to the decision function.

• For the SVM model, first the input data should be normalized before processing. Therefore, a separate function for normalizing (Z-score normalization) is used based on the formula: z=(x-)/

• The combined model first uses the decision function for the detection of gasoline. If the gasoline detected is true, then gasoline is output. If false then, it checks for the SVM model for the detection of diesel. If diesel detected is true, then diesel is output. If false then, no oil.

Converting normal C++ code to High-Level Synthesis (HLS) compatible code involves several steps to ensure that the code can be efficiently implemented on an FPGA. HLS allows you to describe hardware functionality using C/C++ code, which is then synthesized into hardware description language (HDL) code like Verilog or VHDL. The following are some considerations that are taken into account when converting the given code to an HLS-compatible format:

•HLS tools support a subset of C++ data types and libraries. Using fixed-size data types (e.g., int, float) is preferable to dynamically

allocated memory (e.g., new, malloc).

• HLS tools rely on loop-level parallelism for efficient hardware implementation. Optimizing the code to maximize loop parallelism and minimize loop dependencies will largely lead to performance improvements.

•Compiler directives and pragmas to guide optimization and resource allocation, such as loop pipelining, loop unrolling, and array partitioning.

• HLS tools support fixed-point arithmetic more efficiently than floating-point arithmetic. Conversion of floating-point operations to fixed-point operations where applicable. Using pragmas and directives to specify fixed-point formats, precision, and rounding modes to achieve desired numerical accuracy is shown in Fig. 4.

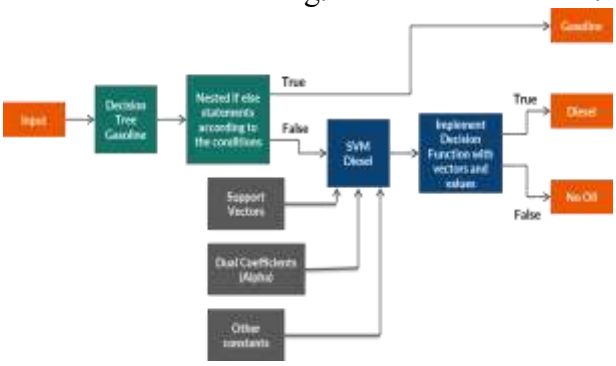

Fig:4 Flow diagram of methodology to develop hardware from HLS code

E. Methodology developed for High-Level Synthesis

• Store the constant values like support vectors, and La- grange values in separate memory like BRAM so that these values can be initialized during the start of the program and remain fixed throughout the execution. These values should be initialized while running the main function therefore separate modules are created and called.

•The datatype is defined in bits and pragma is used to define the high-level synthesis functions and inputs. Various ports for pH, Turbidity, and Conductivity are defined. A stream is created to get the feature inputs. These inputs will act as an array.

•For the SVM model, the constant single values like mean, variance, bias, and gamma are defined in the registers. Differ- ent modules are defined for the implementation of the decision function and value normalization. The property of pipelining and loop unrolling is used to improve the performance and reduce latency. The HLS math library is used for square root and exponential calculation.

A separate function is created to call the decision tree model for prediction. If gasoline is predicted, then the output is gasoline. If gasoline is not predicted then, the function calls the SVM model. If the SVM model predicts diesel then output is diesel. Or else output is no oil.

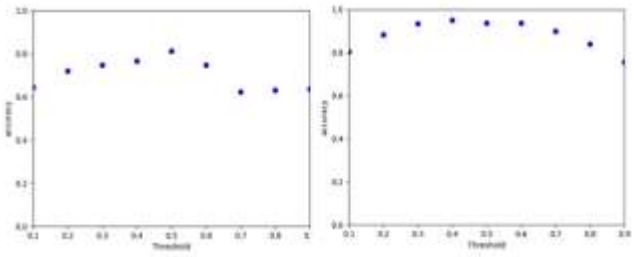

Fig. 5. Accuracy values for different threshold values for gasoline (left) and diesel (right)

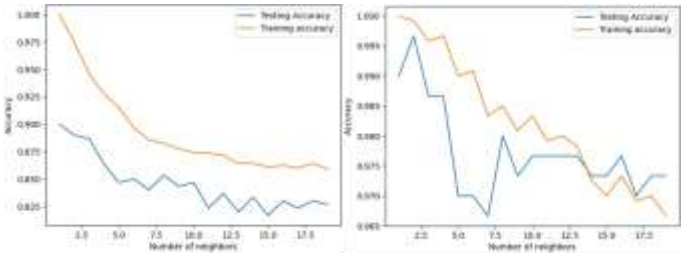

Fig. 6. Accuracy for different K-NN values for gasoline (left) and diesel (right

5 Results

A. Checking accuracies for models

For the logistic regression model, the below plot shows the accuracy of the values for different thresholds shown in Fig.5. The selection of the hyper parameter k in KNN is critical, as illustrated in Fig. 6. Choosing a small k value can cause over fitting, whereas a large k value might lead to under fitting. The optimal k value can be identified through methods like cross-validation or grid search. The following graph shows the different accuracy values for various k parameters.

B. Converting to RTL

The below image shows that the code has successfully synthesized and RTL is exported. The HLS tool will create the necessary Look Up Tables (LUT), Flipflops (FF), and BRAM in the implemented RTL is shown in Fig. 7. It shows that the latency of one iteration is 4391 ns, which is significantly lesser than a traditional microcontroller which has latency in milliseconds The latency of the model can be further reduced using optimization techniques in RTL simulations, like pipelining, loop unrolling, function inlining, and removing reductant gates.

6 Conclusion

In this work, various classification models are trained with the dataset obtained from the research paper using the Python Ski-kit Learn library, and the trained parameters are obtained. As most of the current high-level synthesis tools support C++ or C language, a C++ code is developed by interpreting the logic of the classification models and including the parameters. Then the code should be converted to a format that is com- patible with the HLS tool by including the necessary header files, converting the data types, loop optimization techniques, etc.

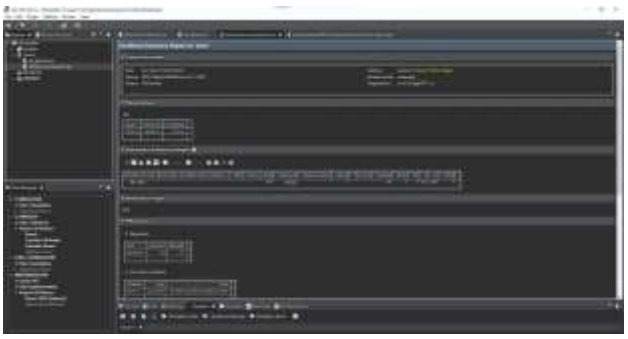

Fig. 7. Result obtained after C synthesis

The code is then loaded into the HLS tool, in this case Vitis HLS. After debugging the code, it is synthesized to RTL. Exporting the RTL code will generate the corresponding Verilog and VHDL code. Therefore, the hardware can be described using high-level programming languages like C++, which makes it much easier for other developers outside the hardware field to leverage the potential of FPGA and other ASIC devices without knowing any HDL language. However basic understanding of the hardware architecture and timing constraints are required. Even complex models like artificial neural networks or models with enormous data sets can be deployed in FPGA.

References:

[1] z. *Asif, Z. Chen, C. An, and J. Dong, "Environmental impacts and challenges associated with oil spills on shorelines," Journal of Marine Science and Engineering, vol. 10, no. 6, p. 762, May 2022.*

[2] *J. M. Cela, D. Roca, C. E. Pereira, and J. Palacin, "A Novel Real- Time Oil Spill Detection System Using Machine Learning Techniques," Sensors, 2017.*

[3] *B. Naresh Kumar Reddy, B Seetharamulu, GS Krishna, BV Vani, "An FPGA and ASIC Implementation of Cubing Architecture," Wireless Personal Communications, Vol. 125,2022.*

[4] *Naresh Kumar Reddy and Subrat Kar "Machine Learning Techniques for the Prediction of NoC Core Mapping Performance," 26th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2021), Dec 1–4, in Perth, Australia, 2021.*

[5] *Reddy and Subrat Kar "An Efficient Application Core Mapping Algorithm for Wireless Network-on-Chip," 26th IEEE Pacific Rim International Sym posium on Dependable Computing (PRDC 2021), Dec 1– 4, in Perth, Australia, 2021.*

[6] *B N K Reddy, et al., "Optimizing Task Scheduling in Multi-thread Real-Time Systems using Augmented Particle Swarm Optimization," 37th International Conference on VLSI Design & 23rd International Conference on Embedded Systems, 2024.*

[7] *Kumar and Subrat Kar "Energy Efficient and High-performance Modified Mesh Based 2-D NoC Architecture," 22nd IEEE International Conference on High Performance Switching and Routing (HPSR), June 7– 9, in Paris, France, 2021.*

[8] *Y. F. Da Silva, R. C. S. Freire, and J. V. D. F. Neto, "Conception and Design of WSN Sensor Nodes Based on Machine Learning, Embed- ded Systems and IoT Approaches for Pollutant*

Detection in Aquatic Environments," IEEE Access, 2023.

[9] *S. Ahmed, T. ElGharbawi, M. Salah, and M. El-Mewafi, "Deep neural network for oil spill detection using Sentinel-1 data: application to Egyptian coastal regions," Geomatics, Natural Hazards and Risk, vol. 14, no. 1, pp. 76–94, 2023.*

[10] *Y.-J. Yang, S. Singha, and R. Mayerle, "A deep learning based oil spill detector using Sentinel-1 SAR imagery," International Journal of Remote Sensing, vol. 43, no. 11, 2022.*

[11] *G. Tabella, N. Paltrinieri, V. Cozzani, and P. S. Ross, "Wireless Sensor Networks for Detection and Localization of Subsea Oil Leakages," IEEE Sensors Journal, vol. 21, no. 9, 2021.*

[12] *Y. Li, X. Yang, Y. Ye, L. Cui, B. Jia, Z. Jiang, and S. Wang, "Detection of Oil Spill Through Fully Convolutional Network," Communications in Computer and Information Science, pp. 353–362, 2018.*

[13] *T. De Kerf, J. Gladines, S. Sels, and S. Vanlanduit, "Oil Spill Detection Using Machine Learning and Infrared Images," Remote Sensing, vol. 12, no. 24, p. 4090, 2020.*

[14] *Z. Ghorbani and A. H. Behzadan, "Monitoring offshore oil pollution us- ing multi-class convolutional neural networks," Environmental Pollution, vol. 289, p. 117884, 2021.*

[15] *M. Konik and K. Bradtke, "Object-oriented approach to oil spill detec- tion using ENVISAT ASAR images," ISPRS Journal of Photogrammetry and Remote Sensing, vol. 118, pp. 37–52, 2016.*

[16] *A. A. Huby, R. Sagban, and R. Alubady, "Oil Spill Detection based on Machine Learning and Deep Learning: A Review," in 5th International Conference on Engineering Technology and its Applications, 2022.*

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

V.V. Jaya Rama Krishnaiah: Conceptualization, Methodology, Supervision.
P. G. K. Sirisha: Software, Validation.
S. Parvathi Vallabhaneni: Data curation, Writing – original draft.
D. V. Chandrashekar: Formal analysis, Investigation.
K. Jagan Mohan: Resources, Writing – reviewing & editing.
G. Rajesh Chandra: Visualization, Project administration.
Venkata Kishore Kumar Rejeti: Hardware implementation, Validation.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.