Report for the Advanced Logic Design course's project

Written by Leonardo Zappalà and Oussema Kasraoui

Github: Ossyk/ECG-anomaly-detection-FPGA

ECG anomaly-detector - Introduction:

Cardiovascular diseases remain one of the leading causes of mortality worldwide, and early detection of cardiac abnormalities is essential for effective clinical intervention. Electrocardiogram (ECG) monitoring is a widely used diagnostic tool which is often put together with traditional software-based processing devices that have to meet strict latency, energy, and reliability constraints required for on-device diagnosis.

In this context, hardware acceleration through Field-Programmable Gate Arrays (FPGAs) offers an interesting solution. FPGAs combine parallel computation, low latencies and an efficient use of energy so to make them suitable for biomedical signal-processing workloads. At the same time, Convolutional Neural Networks (CNNs) have demonstrated state-of-the-art performance in ECG classification tasks, capturing subtle morphological features that even experts' eyes often do not notice.

This project presents the design and implementation of an FPGA-based ECG anomaly-detection accelerator powered by a quantized CNN model. The system integrates a MicroBlaze soft processor, AXI-interconnected custom CNN IP cores, on-chip BRAM for inputs and outputs, and a UART communication system with a Python GUI for visualization.. The accelerator is deployed on a Nexys4 Artix 7 FPGA board and evaluated using a dataset of annotated ECG segments (Mit-bih-arrhythmia database v1.0.0).

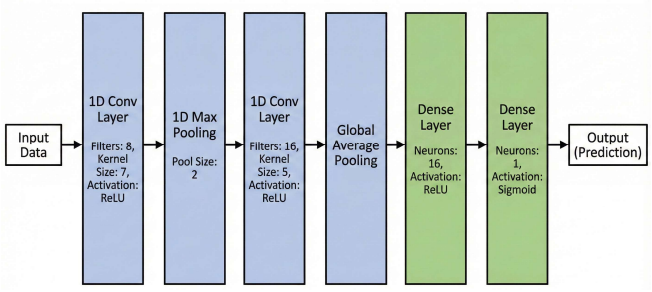
The report explains the full hardware / software flow, including the CNN model training and quantization, Vivado block design choices, Vitis HLS optimization, FPGA system integration in Vivado of the Vitis HLS component, communication protocol design and performance evaluation.

Neural network in the ECG analyser:

In this section we will describe the Convolutional Neural Network used for the anomaly detection and all the passages for the development of the CNN inside the FPGA architecture, starting from the training phase to the integration of the model as a hardware accelerator in the system.

The choice of the CNN over other classical machine learning models was made because of the nature of the task: ECG classification requires identifying subtle morphological variation in cardiac cycle which appear as shapes, local patterns and waveform deviations. CNN are suitable because they learn features directly from the raw input waveform, eliminating the need for manual feature design. CNNs excel at detecting complex waveform anomalies and at reducing false negatives. The network was designed as a lightweight 1D Convolutional Neural Network optimized for embedded inference. The architecture consists of:

- Conv1D (8 filters, kernel size 7) + ReLU
- MaxPooling1D (pool size 2)
- Conv1D (16 filters, kernel size 5) + ReLU
- Global Average Pooling
- Dense (16 neurons, ReLU)
- Dense (1 neuron, Sigmoid)



This design was selected for three reasons:

1. 1D convolutions effectively extract morphological ECG features such as QRS width, ST deviation, or T-wave abnormalities directly from the raw signal.
2. Compactness: the model uses few parameters, making it suitable for real-time inference in resource-constrained FPGA fabrics.

3. Hardware friendliness: convolution and fully connected operations can be expressed as parallel multiply-accumulate (MAC) pipelines, ideal for synthesis using Vitis HLS. It's also very cheap in terms of energy consumption.

The choice of a 1D convolutional network can be explained by the fact that ECG are essentially 1D temporal sequences, where the information is encoded along time rather than spatial dimensions. Morphological features such as QRS complexes, P-waves, and T-waves appear as temporal patterns. Each sample in the dataset (heart beat) is represented as a single 1D vector of 720 samples, not as a matrix.

The dataset used in this project was taken from the MIT-BIH records.

The CNN was trained in Python using Tensorflow and class weighting has been applied to reduce false negatives in abnormal samples. Early stopping prevented overfitting during the training phase of the network.

Since floating-point networks are not hardware-efficient on FPGA, the trained model was quantized before deployment. The export pipeline performed the following operations:

- Convolution kernels and dense weights were quantized to int8.
- Biases were quantized to int16 to preserve precision.
- A layer-specific scaling factor was computed to map floating-point weights into fixed-point ranges.
- The quantized arrays were written to hexadecimal .mem files, one for each layer's weights and biases. The files are automatically formatted for direct initialization of Block Ram inside the FPGA board.
-

Within the Vivado block design, the accelerator was instantiated as a memory-mapped IP block connected to the MicroBlaze subsystem through an AXI-Lite interface. This interface exposes control registers for starting the computation and retrieving completion status, as well as memory-mapped locations for input and output buffers. The quantized weight files generated during export were bound to BRAM blocks via the AXI BRAM Controller, allowing the CNN IP to retrieve parameters with zero-cycle latency. The MicroBlaze processor was responsible for receiving ECG windows through the UART interface, transferring them into the accelerator's dedicated input buffer, and issuing the command to initiate inference. Once computation is completed, the accelerator writes the output prediction score to the dedicated output section of the AXI BRAM, which the MicroBlaze then reads and forwards to the host machine for visualization.

ECG Block Design and implementation in Vivado

This section describes the hardware architecture developed in Vivado for deploying the ECG anomaly-detection accelerator on the Nexys FPGA. The system follows a mixed hardware – software co-design paradigm in which a MicroBlaze soft processor manages control and communication tasks, while a custom CNN accelerator performs the computationally intensive inference operations.

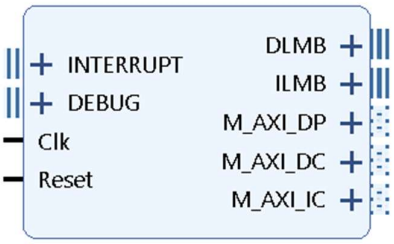
The final implementation integrates several AXI-interconnected peripherals, on-chip memory modules, and custom IP components into a coherent and deterministic processing pipeline.

Overview of the system:

The system is clocked at 100 MHz and includes the following major components:

- MicroBlaze soft processor
- CNN accelerator IP (ecg_cnn_0)
- AXI BRAM Controller + Block Memory Generator
- AXI SmartConnect interconnect fabric
- UART Lite interface for PC communication
- GPIO interfaces for board-level I/O
- Clocking Wizard and Processor System Reset unit

MicroBlaze configuration



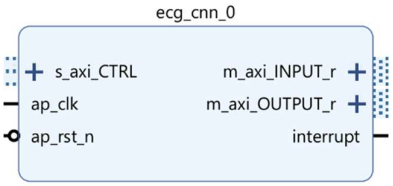
This processor executes all the control-plane operations including: data movement, peripheral configuration and communication with the host PC. It runs the firmware (C application written in Vitis and compiled into .ELF file) loaded in the on-chip BRAM during the FPGA configuration "specifically the first half of the BRAM"

The microblaze also directs the CNN accelerator by handling the start, the status polling and the result retrieval. It handles the UART communication with the user by continuously monitoring the receiving registers for new incoming bytes (sent by the host pc thorough a serial protocol: 115200 baud on port 5). The microblaze at the end of the whole process will send the inference results back to the python GUI on the host via the UART Lite module.

The microblaze writes the input ECG samples to the BRAM Controller through "M_AXI_DP" and configures the accelerator via the AXI-Lite control interface by writing to the memory-mapped control registers generated by the HLS tool. These registers include the start bit, done flag, idle status, and optional interrupt enables. Setting the start bit triggers the accelerator to begin processing the input ECG window stored in BRAM, while polling the status register allows the firmware to determine when inference has completed.

The MDM (MicroBlaze debug module) was included in the design to ensure maintainability of the overall MicroBlaze logic since this powerful debug system was essential for validating: AXI connectivity, BRAM accessibility, CNN accelerator behaviour, interrupt and polling logic.

CNN accelerator IP (ecg_cnn_0)



It is the custom HLS-generated IP core implementing the "inference step" in the ECG analyser. It is the element performing convolution, pooling, dense operations and activation functions entirely in hardware.

The accelerator receives start/stop commands through the AXI-Lite "s_axi_CTRL" signal. It read the ECG input window from the BRAM via "m_axi_INPUT_r" and writes the inference result via the "m_axi_OUTPUT_r". The AXI master is fundamental: instead of having the samples passed directly by the Microblaze (computationally slow), the result are read by the accelerator directly from the BRAM at AXI speed, simulating a DMA behaviour. At the same way, when the result is computed the data is loaded into BRAM's output buffer, that will get accessed by the Microblaze.

The fig. below shows the registers used for input reading and output storing. It is to note that they are the same just because the AXI master interfaces are allowed to access the same physical memory block, however inputs and outputs occupy different offsets within the BRAM block.

/ecg_cnn_0					
/ecg_cnn_0/Data_m_axi_INPUT_r (64 address bits : 16E)					
/axi_bram_ctrl_0/S_AXI	S_AXI	Mem0	0xC000_0000	64K	0xC000_FFFF
Excluded (4)					
/axi_gpio_0/S_AXI	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
/axi_gpio_1/S_AXI	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4061_0000	64K	0x4061_FFFF
/ecg_cnn_0/s_axi_CTRL	s_axi_CTRL	Reg	0x8000_0000	64K	0x8000_FFFF
/ecg_cnn_0/Data_m_axi_OUTPUT_r (64 address bits : 16E)					
/axi_bram_ctrl_0/S_AXI	S_AXI	Mem0	0xC000_0000	64K	0xC000_FFFF

```
0xc0000000 - 0xC0007FFF -- 32KB --> "Code/Data"
0xc0008000 - 0xc0008FFF -- 4 KB --> "Input Buffer"
0xc0009000 - 0xc000FFFF -- 28KB --> "Output/Reserved"
```

The code, data, vectors, stack, and heap occupy the beginning of the BRAM (the first half). The input and output buffers lay on a later offset position, starting in the second half of the BRAM. Specifically, the input buffer is located immediately at the base of this second half with the output buffer following it at a short, defined offset. Using a unified memory space, such as this BRAM, simplifies the architecture, reduces hardware resources, and allows seamless coordination between the MicroBlaze and the accelerator without requiring multiple physical memories.

AXI BRAM Controller and Block Memory Generator:

The AXI BRAM Controller exposes the BRAM on-chip an addressable memory region. The Block Memory Generator "blk_mem_gen_0" make disposable the memory used as CNN input buffer, CNN output buffer and some temporary workspace if needed. The BRAM controller handles the accesses for input retrieval and output writing through the "Smart Connect" block.

The AXI Smart Connect automatically routes AXI communication between masters and slaves. In our case allows the 2 master (Microblaze and ECG_cnn_0) to access the same memory.

UART Lite interface:

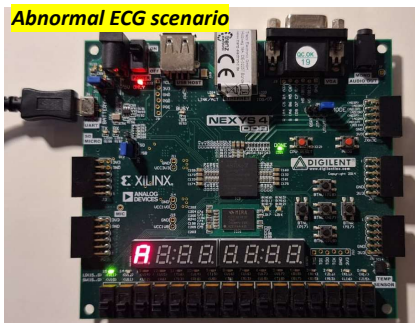
UART is the serial communication protocol used between the FPGA board and the host PC to send and receive data. The UART Lite proved to be simple, reliable and perfectly integrated with the MicroBlaze component.

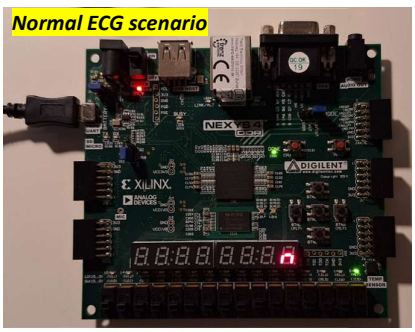
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4061_0000	64K	0x4061_FFFF
/ecg_cnn_0/s_axi_CTRL	s_axi_CTRL	Reg	0x8000_0000	64K	0x8000_FFFF
/ecg_cnn_0/s_axi_CTRL	s_axi_CTRL	Reg	0x8000_0000	64K	0x8000_FFFF
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0	64K	0xFFFF
/microblaze_0/Instruction (32 address bits : 4G)					
/axi_bram_ctrl_0/S_AXI	S_AXI	Mem0	0xC000_0000	64K	0xC000_FFFF
/axi_gpio_0/S_AXI	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
/axi_gpio_1/S_AXI	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4061_0000	64K	0x4061_FFFF

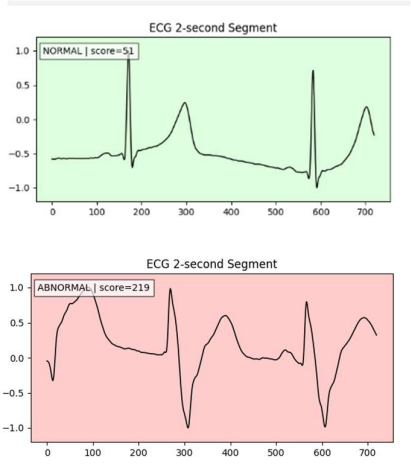
UART Lite & GPIO registers' memory addresses

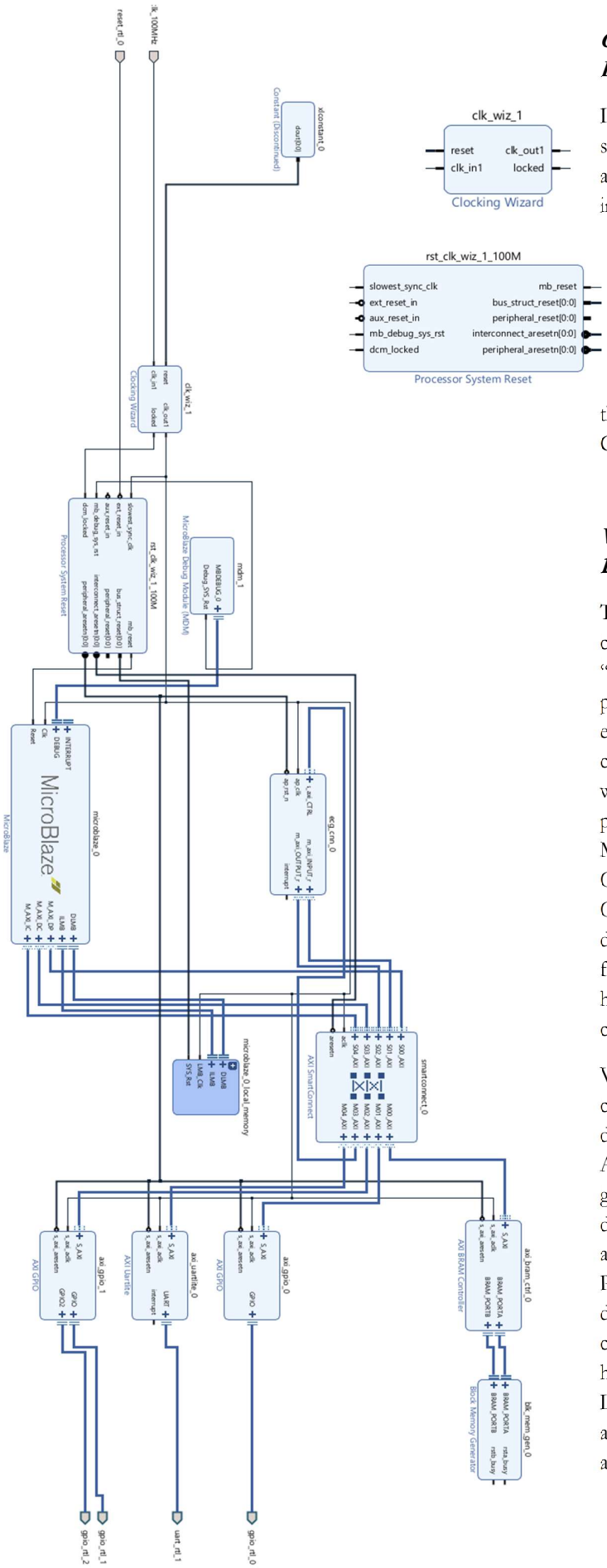
GPIO (General Purpose Input/Output) Subsystem:

The AXI GPIO subsystem answers the question of how we can represent the output visually on the board. It provides general purpose digital I/O capabilities that we used for the demonstration output. It enables the Microblaze to toggle LEDs and the 7-segment display during the execution of the ECG classification task, showing "A" if the output score give as output abnormal, and "n" if no anomalies were detected during the inference.









Clocking wizard and Processor System Reset:

It is the component generating the internal system clock used by all synchronous blocks and converts the external 100 MHz oscillator into clean, phase aligned clocks. This component goes hand-in-hand with the processor system reset which has the aim of creating synchronized reset signals for all logic in the design. Thanks to this block the microblaze resets safely and the AXI behaves correctly at startup, it also ensures that the accelerator, the BRAM, UART and GPIO remain idle until the system is stable.

VITIS ACCELERATOR DEVELOPMENT:

The software components of the ECG classification system were developed using the "*Xilinx Vitis Unified Software Platform*", which provides an integrated environment for embedded programming, hardware/software co-design and accelerator development. Vitis was used because the architecture of the project is based on an interaction between a MicroBlaze soft processor and a custom CNN hardware accelerator synthesized from C++ code. This hybrid approach requires a development framework capable of managing firmware, drivers, memory maps, and hardware accelerators within a single, coherent workflow.

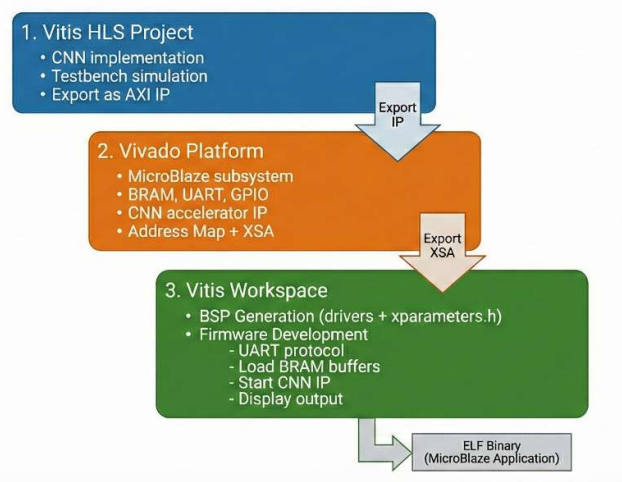
Vitis satisfies these requirements by combining traditional embedded software development tools with direct support for AXI-based peripherals and Vitis HLS–generated IP blocks. After the Vivado block design is exported as an XSA platform, Vitis automatically generates a Board Support Package (BSP) containing the low-level drivers, address definitions, and system configuration files required to control the hardware. This includes drivers for UART Lite, AXI GPIO, the AXI BRAM controller, and the custom CNN accelerator's AXI-Lite and AXI master interfaces.

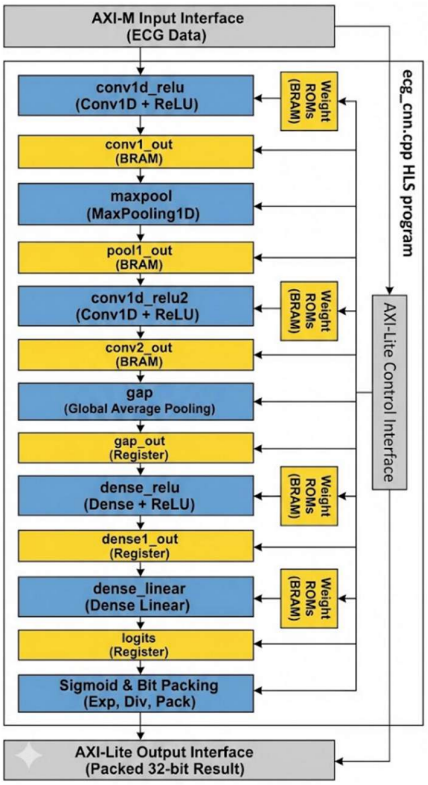
As a result, the firmware can interact with the hardware through simple memory-mapped I/O operations.

Another advantage of Vitis is its integration with "*Vitis High-Level Synthesis (HLS)*". The CNN inference function was designed and tested in HLS, synthesized into RTL, and exported as a Vivado IP core. Vitis ensures that the resulting accelerator is visible in the BSP and that its control registers are accessible through automatically generated header files. This simplifies hardware–software integration guarantees consistency between the synthesized hardware and the corresponding software drivers.

Vitis constitutes the central software development platform of the project, enabling the implementation of the real-time ECG anomaly detection pipeline and its execution on the MicroBlaze processor in close coordination with the hardware CNN accelerator.

DEVELOPMENT PIPELINE:





1. HLS accelerator:

The first stage of the Vitis workflow involved designing the CNN accelerator using Vitis HLS writing the code responsible for the neural network. Once validated, the design was synthesized in HLS, generating RTL (Verilog/VHDL) and an AXI-compliant hardware interface. The IP was exported and imported into the Vivado block design as the custom accelerator "ecg_cnn_0" present in the block design diagram.
This step ensured that the accelerator was ready to be deployed and controlled by the MicroBlaze subsystem in the completed system-on-chip architecture. Once imported into Vivado, the accelerator became a self-contained hardware block capable of performing the full CNN inference pipeline.
The accelerator implements all computations using fixed-point arithmetic (ap_fixed<12,3>), which significantly reduces DSP and memory utilization while maintaining precision for ECG analysis. Within the HLS source, each neural network layer is expressed as a synthesizable C++ function, augmented with targeted optimization directives.
Intermediate feature maps and layer outputs are stored in BRAM-based buffers with two-port access, enabling simultaneous reading and writing across consecutive pipeline stages.

All CNN weights and biases are stored in ROM structures inside the accelerator IP that are implemented using the BRAM primitives of the FPGA. And they are marked as STABLE variables, ensuring constant latency and eliminating the need for dynamic weight loading during operation

```
static const weight_t ROM_conv1_w[8][1][7] = {
    { { 0.808355, 0.274597, -0.268904, 0.097335, -0.464580, -0.572587, -0.570781 } },
    { { 0.006863, -0.072012, -0.354196, -0.356624, -0.069203, 0.164639, 0.511230 } },
    { { -0.316640, -0.330161, -0.004772, -0.055982, -0.265593, -0.369781, -0.392396 } },
    { { 0.442869, -0.119984, -0.455584, -0.181636, -0.461232, -0.297729, -0.229759 } },
    { { 0.639480, 0.854410, 0.079155, 0.328793, 0.473816, 0.788829, 0.606516 } },
    { { 0.395674, 0.145351, 0.252308, -0.273229, -0.420919, -0.434611, -0.789018 } },
    { { 1.000467, 0.031223, -0.154578, -0.714550, -0.255084, 0.014901, 0.577641 } },
    { { 0.475729, 0.289055, -0.532650, -0.117671, 0.044238, 0.439691, 1.638405 } },
};
```

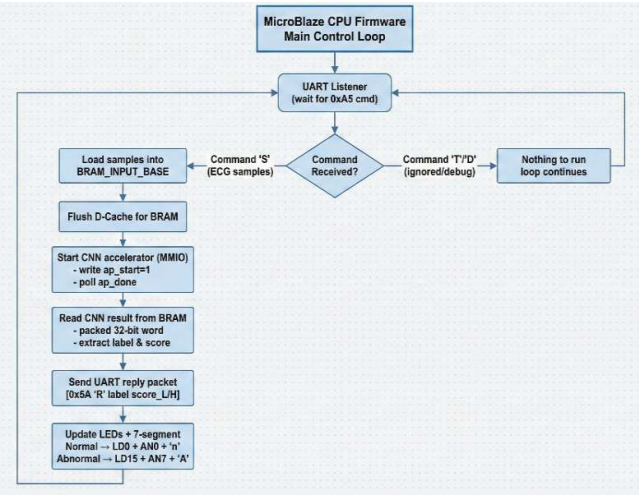
The top-level HLS function declares three primary interfaces: an AXI-Lite control interface for programming the accelerator, and two AXI4 master interfaces for reading the input ECG window and writing back the classification result.

To ensure correctness before hardware synthesis, the HLS model was validated using a testbench. This simulation step confirms that the quantized fixed-point network reproduces the expected behaviour of the original floating-point TensorFlow model on representative ECG segments. Only after this verification is the accelerator synthesized into RTL and packaged as a reusable Vivado IP block. Through this workflow, Vitis HLS provides a seamless path from high-level neural network description to optimized hardware implementation, enabling the CNN to run as a dedicated accelerator tightly coupled with the MicroBlaze subsystem.

2. ***Vivado Platform***:

Once the CNN accelerator was synthesized and exported from Vitis HLS as an IP block, the next stage of the development workflow consisted of integrating this accelerator into a complete FPGA platform using Vivado. This platform acts as the backbone on which the firmware runs and the accelerator executes real-time ECG analysis. Once the block design, address map and bitstream are finalized, the platform is exported to Vitis as an XSA file that contains full hardware description. This file is later used by Vitis to generate BSP (Board Support Package) which includes the UART/GPIO drivers and memory base address macros. This creates a seamless transition from hardware to software and ensures that firmware development is aligned with the actual synthesized platform.

3. ***Firmware development:***



After the system is initialized, the firmware enters an infinite control loop that continuously waits for incoming commands from the host PC. Communication is started with a synchronization byte (0xA5), after it is possible to send a command character. Once the synchronization byte is detected, the firmware reads the command and dispatches execution accordingly:

- ***Command 'S':*** Indicates that the host PC is sending an ECG window. The firmware reads each sample as a signed 16-bit integer and writes it into the accelerator's input BRAM region (BRAM_INPUT_BASE) as a 32-bit sign-extended value. If the host provides fewer than 720 samples, the remaining positions in BRAM are explicitly zero-padded. In the firmware, the data cache is flushed after all ECG samples have been

written into the BRAM input buffer and zero-padded. This ensures that the AXI master interface of the CNN accelerator reads the updated values from BRAM rather than stale cached data.

- ***Command 'T' or 'D'***:
  These commands do not require accelerator execution. 'T' discards a two-byte threshold value, since the threshold is fixed inside the model (0.4) and cannot be updated at runtime. But for protocol compatibility, the firmware still accepts the T command but discards it. While 'D' prints the first 20 BRAM entries for debugging.

```
while (1) {
    int len = load_ecg_from_uart_to_bram();
    if (len < 0) {
        // unknown command, just keep listening
        continue;
    }
    if (len == 0) {
        // 'T' threshold command consumed, nothing to run
        continue;
    }
    // 'S' command received and samples loaded
    run_ecg_cnn_mmio();
    send_cnn_reply_packet();
    Xil_DCacheFlushRange((UINTPTR)BRAM_OUTPUT_BASE, N_SAMPLES * sizeof(uint32_t));

}
```

After the ECG samples have been fully written to the BRAM input buffer and the data cache has been flushed, the firmware must instruct the hardware accelerator to begin processing. This is accomplished through the AXI-Lite control interface exposed by the HLS-generated IP core. The accelerator uses a standard control register (AP_CTRL), where writing a 1 to bit 0 asserts the ap_start signal. As soon as this write occurs, the accelerator begins reading input data through its AXI master port and executes the CNN pipeline autonomously in hardware.

Since the MicroBlaze must not proceed until inference is complete, the firmware enters a tight polling loop that continuously reads the same control register and checks for the assertion of the ap_done bit (bit 1). The loop continues until the accelerator signals completion.

```
static void run_ecg_cnn_mmio(void)
{
    // ap_start = 1
    Xil_Out32(ECG_CNN_BASEADDR + XECG_CNN_CTRL_ADDR_AP_CTRL, 0x01);
    // wait for ap_done (bit 1)
    while (!(Xil_In32(ECG_CNN_BASEADDR + XECG_CNN_CTRL_ADDR_AP_CTRL) & 0x2)) {
        // spin
    }
}
// -------------------- Input loading from UART (handles 'T' and 'S') --------------------
//
// 'T' command: 0xA5 'T' q15_lo q15_hi   (we just consume and ignore threshold)
// 'S' command: 0xA5 'S' len_lo len_hi   then len*2 bytes of int16 samples
```

When inference completes, the accelerator writes a single 32-bit packed result into the output BRAM region (BRAM_OUTPUT_BASE). The results are decoded as follows:

- bit 0 → predicted label (0 = Normal or 1 = Abnormal)
- bits [16:1] → Q8.8 fixed-point probability score

To provide real-time visual feedback, the firmware drives two AXI GPIO peripherals connected to the board's LEDs and seven-segment display:

- Normal (label=0) → Led LD0 is lighted and the display shows "n"
- Abnormal (label=1) →Led LD15 is enabled and the display shows "A"

PERFORMANCE ANALYSIS

1. Timing:

The design was implemented using a 100 MHz system clock (period = 10 ns). All timing constraints are successfully met as the "Design Timing Summary" confirms.

a. **Setup time**: the Worst Negative Slack (WNS) is +1.517 ns > 0 *****, the Total Negative Slack (TNS) is 0 ns and failing endpoints = 0 ******
b. **Hold time**: the Worst Negative Slack (WNS) is +0.045 ns > 0, the Total Negative Slack (TNS) is 0 ns and failing endpoints =0. The design also meets all hold-time requirements, although the WHS is small**. ***** However, since the slack remains positive and no endpoints fail, hold timing is still satisfied

*** the longest combinational path in the design completed 1.517ns earlier than the clock period allows**
**** every synchronous path satisfies setup timing**
***** indication of the presence of very short routing paths where signals arrive close to the minimum hold requirement**

c. **Critical path**: the C.P. (referred by path 1) is fully inside the CNN accelerator "ecg_cnn_0" which is totally expected since it's by far the most complex computational block. The summary shows that for path 1 the total delay is due to logical delay (7.537ns) which tells that the delay comes from arithmetic depth rather than long wiring (net delay = 0.803ns)

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	1.517	11	2	2	system_bd_i/ec...roduct__0/CLK	system_bd_i/ec...476_reg[24]/D	8.340	7.537	0.803

Summary	
Name	Path 1
Slack	1.517ns
Source	system_bd_i/ecg_cnn_0/U0/grp_exp_12_3_s_fu_598/mul_25ns_25ns_50_1_0_U285/tmp_product__0/CLK (rising edge-triggered cell DSP48E1 clocked by clk_out1_system_bd_clk_wiz_1_0 {rise@0.000ns fall@5.000ns period=1(
Destination	system_bd_i/ecg_cnn_0/U0/grp_exp_12_3_s_fu_598/y_lo_s_reg_476_reg[24]/D (rising edge-triggered cell FDRE clocked by clk_out1_system_bd_clk_wiz_1_0 {rise@0.000ns fall@5.000ns period=10.000ns})
Path Group	clk_out1_system_bd_clk_wiz_1_0
Path Type	Setup (Max at Slow Process Corner)
Requirement	10.000ns (clk_out1_system_bd_clk_wiz_1_0 rise@10.000ns - clk_out1_system_bd_clk_wiz_1_0 rise@0.000ns)
Data Path Delay	8.340ns (logic 7.537ns (90.373%) route 0.803ns (9.627%))
Logic Levels	11 (CARRY4=9 DSP48E1=1 LUT2=1)
Clock Path Skew	-0.145ns
Clock Un...rtainty	0.074ns

Table: path 1 refers to the critical path

d. **Maximum frequency**: Total delay of the critical path is 8.34 ns. Therefore:

$$Freq\ \max = \frac{1}{Critical\ path\ delay} = \frac{1}{8.34\ ns} = 120\ MHz$$

e. **Latency**: The HLS/Vitis report shows a Latency = 607.000 ns = 607 µs .
f. **Throughput**: $\text{throughput} = \frac{1}{latency} = \frac{1}{0.000607} = 1647\ inferences\ per\ second$

The accelerator needs 0.6ms to classify one ECG window of 2s. Although our design is not as deeply pipelined as the highly optimized modals, it still runs more than 3000 faster than real-time ECG acquisition, providing a wide performance margin for continuous monitoring, without interruptions or delays.

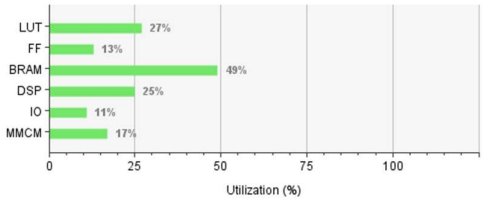
2. Utilization report:

a. **LUTs**: they implement the control logic around the system. (MB instructions and control flow, AXI interconnect logic, CNN accelerator control and arithmetic, address decoding, UART, GPIO). The usage of 27% is very reasonable and shows a logic-efficient design despite containing a sull SoC + accelerator

Resource	*Used*
LUTs	*16973*
FFs	16840
BRAM	66
DSP	60
IO	23

b. **FFs**: flip flops show a low usage since the design relies more on DSP and BRAM for heavy computations

c. **BRAM**: is the bottleneck resource in the design since it's where most of the accelerator storage lives (it stores the code, data, vectors, heap, stack, weights and biases that live inside the accelerator and the input and output buffers) The usage of 49% is the highest, yet it's normal.



d. **DSP**: usage is about 25%, meaning the CNN accelerator performs a lot of multiply-accumulate operations in Conv1D layers but still fits easily on the FPGA with plenty of room left.

e. **IO**: Only a small fraction of IOs is needed: UART TX/RX, GPIO for the LEDs and seven-segment display and the clock/reset lines.

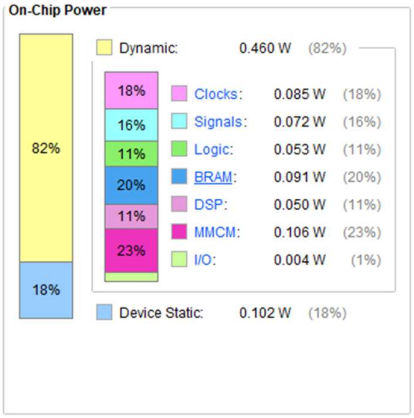
3. Power:

To analyse the power, we will set an environment with the following parameters:

- Temp grade = commercial
- Process = typical
- Junction temperature ≈ 25°C
- Ambient temperature = 25°C
- Airflow = 250 LFM (typical forced convection)
- Medium heatsinks

a. **Dynamic power (active): 0.46W (82%)** due to the changing values of signals (e.g. Charge/discharge of capacitors).

b. **Static power (leakage): 0.102 W (18%)** due to current leakage (transistors are imperfect)



Dynamic power is very high (82%) because the design is actively switching a lot of logic every clock cycle. Especially inside the BRAM, clocks, and DSP blocks. But its value of 0.46W is normal for a FPGA running a CNN accelerator at 100MHz.

The static power of 0.102W is also typical for Artix-7 devices and reflects leakage due to the silicon

Overall, the total on-chip power of 0.562W is low and expected for this type of lightweight accelerators

CONCLUSION : RESULTS AND FUTURE WORK

The trained model achieved a 68% of accuracy in the Python-ML environment before hardware deployment. Such value was considered acceptable for a demo-prototype, especially given the lightweight network architecture applied for FPGA compatibility.

After running tests on the raw ECG samples used for training, specifically by taking the 2nd 2-second window of each sample, the hardware pipeline correctly classified 29 of 48 windows, achieving a correctness of 61%. In annex 1, you will find the comparison between the expected and the predicted labels.

Overall, the results indicate that the implemented accelerator behaves consistently with the software model and executes the inference pipeline correctly on hardware.

However, the achieved accuracy should be upgraded for real clinical use. Therefore, several improvements can be made for a future design.

1- Improve the NN architecture by introducing deeper CNNs, (e.g. explore 2D representation as scalograms which enables capturing richer morphological characteristics.
2- Improve maintainability by moving the weights from static ROM inside of the hardware to external memory loaded at startup by the firmware. This accelerates the model-update cycle by eliminating the need to re-import the IP block after every retraining.
3- Support continuous real-time ECG streaming from a sensor by integrating a hardware interface (e.g. AFE: Analog Front-End), a real-time buffering to continuously collect samples into a circular buffer in BRAM and preprocessing the signal on the FPGA.

***

ANNEX 1 – Expected vs Predicted classification

sample id	Target Label	Predicted outcome	Prediction score
100	0	0	51
101	0	0	51
102	1	1	225
103	0	0	51
104	1	1	209
105	0	0	51
106	0	1	153
107	1	1	219
108	0	1	150
109	1	1	250
111	1	1	122
112	0	0	51
113	0	0	51
114	0	1	241
115	0	1	104
116	0	1	244
117	0	1	245
118	1	1	244
119	0	0	68
121	0	0	67
122	0	0	66
123	0	1	249
124	0	1	240
200	1	1	219
201	0	0	83
202	0	1	194
203	0	0	51
205	0	1	193
207	1	0	51
208	1	0	51
209	0	1	164
210	0	1	231
212	1	1	103
213	0	1	214
214	1	1	240
215	0	0	51
217	1	0	51
219	0	1	238
220	0	0	92
221	1	1	185
222	0	0	69
223	0	0	51
228	0	0	51
230	0	1	112
231	1	0	51
232	1	1	131
233	1	1	246
234	0	1	109

Contribution to the project

ML: Both

We trained four different ML models {regression / SVM / perceptron / 1D CNN} so we divided the workload between us so that each member develops two models and test them.

The highest accuracy was achieved by 1D CNN, so we admitted it.

Vitis HLS: Leonardo

I, Leonardo, handled the hardware accelerator implementation:

- Designed the functions for convolution, pooling and dense layers
- Optimized with pragmas
- Integrated fixed-point data types (ap_fixed <12,3>)
- Generated and validated the IP core
- Controlled the logic by running a testbench

Vivado Block Design: Both

We worked together to:

- Integrate the HLS IP into block design
- Configure the MicroBlaze system and connect the BRAM, AXI interfaces, clock, resets...
- Resolved the address mapping

The tutorials provided by AMD XILINX were of a lot of help in this section. For instance:

- Designing IP Subsystems • Vivado Design Suite Tutorial: Designing IP Subsystems Using IP Integrator (UG995) • Reader • AMD Technical Information Portal

Vitis application (platform & firmware): Oussema

I, Oussema, developed the firmware running on the Microblaze, including:

- UART communication, reading ECG segments, transmitting input and retrieving output
- Controlling LEDs and 7-segment displays
- Overall system logic

Python GUI: Both

We collaborated to develop the Graphical Interface used to select the ECG segment and visualize the results.

Oussema focused on the communication side (serial protocol and data formatting)
Leonardo focused on the interface (Tkinter and user interaction)