הפקולטה להנדסת חשמל ואלקטרוניקה - מערכות משובצות מחשב

ספר פרויקט בנושא:

# מימוש אלגוריתם למידת מכונה לעיבוד תמונה על גבי FPGA

# Implementation of a machine learning algorithm for image processing on FPGA


נאור דוד

מנחה

מר שמואל מעודה

# תוכן עניינים


תקציר ................ 4

Abstract ................ 5

Executive Summary ................ 6

רשימת איורים ................ 11

רשימת משוואות ................ 13

קיצורים ומונחים ................ 14

פרק 1 - מבוא ................ 15

1.1 רקע כללי ................ 15

1.2 תיאור הבעיה ................ 15

1.3 מטרת הפרויקט ................ 16

1.4 מבנה הספר ................ 16

פרק 2 - רקע תיאורטי ................ 17

2.1 CPU ................ 17

2.2 Python ................ 17

2.3 FPGA ................ 18

2.4 VHDL ................ 19

2.5 עיבוד תמונה דיגיטלי ................ 21

2.6 למידת מכונה ................ 22

2.6.1 סוגי בעיות בלמידת מכונה ................ 22

2.6.2 רשת נוירונים מלאכותית ................ 23

2.6.3 רשת Fully Connected Neural Network ................ 24

2.6.4 מושגים בלמידת מכונה ................ 26

2.7 פתרונות קיימים בשוק ................ 27

2.7.1 HLS/OpenCL ................ 27

2.7.2 RTL מודולרי - ALAMO ................ 27

2.8 יתרונות מימוש אלגוריתם ב-FPGA ................ 28

פרק 3 - מימוש הפרויקט ................ 29

3.1 כלי תכן וסביבת הפיתוח ........ 29
3.1.1 בחירת לוח פיתוח DE2-115 ........ 29
3.1.2 סביבת פיתוח חומרה ........ 31
3.1.3 סביבת אימון ותוכנה ........ 31
3.2 בחירת האלגוריתם ומימושו ........ 32
3.2.1 בחירת האלגוריתם ........ 32
3.2.2 אופן פעולת מודל רשת הנוירונים ........ 33
3.2.3 מימוש מודל רשת הנוירונים בסביבת תוכנה ........ 34
3.2.4 הכנת מודל רשת הנוירונים למימוש חומרתי ........ 38
3.2.5 מימוש מודל רשת הנוירונים בסביבת חומרה ........ 41
**פרק 4 - תוצאות ........ 69**
4.1 ניצול משאבים (Resource Utilization) ........ 69
4.2 ניתוח תזמון (Timing Analysis) ........ 70
4.3 אימות חומרתי ותוצאות ריצה ........ 72
4.3.1 הרצה ראשונה ........ 72
4.3.2 הרצה שנייה ........ 74
4.3.3 הרצה שלישית ........ 76
**פרק 5 - מסקנות ........ 78**
5.1 מסקנות ........ 78
5.2 הצעות עבודה להמשך ........ 78
**ביבליוגרפיה ........ 79**

# תקציר

הפרויקט עוסק במימוש חומרתי של אלגוריתם למידת מכונה על גבי רכיב FPGA, במטרה להדגים את יתרונות המימוש המקבילי בחומרה לעומת מימוש סדרתי על גבי CPU. האלגוריתם שנבחר הוא מודל Fully Connected Neural Network לסיווג ספרות ממסד הנתונים MNIST.

בשלב הרקע נסקרו יסודות טכנולוגיית ה-FPGA והעקרונות המרכזיים בלמידת מכונה, לצד סקירה של פתרונות קיימים למימוש חומרה של אלגוריתמים, הממחישים גישות שונות להאצת חישובים על פלטפורמות מתקדמות.

בשלב האפיון התוכנתי, המודל אומן בסביבת Python (PyTorch) על גבי Google Colab, תוך חלוקת מאגר הנתונים לקבוצות אימון, ולידציה ובדיקה. לאחר האימון והערכת הביצועים, נשמר המודל המאומן והומר לייצוג Fixed-Point, המותאם למימוש יעיל בחומרה.

בהמשך הופקו קבצי זיכרון ייעודיים (MIF) ותוכננה ארכיטקטורה חומרתית ב-VHDL. הארכיטקטורה כוללת רכיבי זיכרון (ROM/BRAM), רכיב MAC המשלב 10 יחידות חישוב מקבילות, מכונת מצבים (FSM), בלוק ArgMax, ורכיבים נוספים למדידת זמן ריצה והצגת התוצאה על גבי תצוגת 7-Segment.

הפרויקט מדגים את הפוטנציאל של FPGA בתחום למידת מכונה, תוך הדגשת יתרונות כגון ניצול יעיל של משאבים, מקביליות גבוהה וזמני ריצה קצרים בהשוואה למימוש תוכנתי.

# Abstract

This project focuses on the hardware implementation of a machine learning algorithm on an FPGA device, aimed at demonstrating the advantages of parallel execution in hardware compared to serial execution on a CPU. The chosen algorithm is a Fully Connected Neural Network for classifying handwritten digits from the MNIST dataset.

In the background stage, the fundamentals of FPGA technology and the core principles of machine learning were reviewed, alongside a survey of existing hardware implementations, illustrating different approaches to accelerating computations on advanced platforms.

In the software design stage, the model was trained in Python (PyTorch) on Google Colab, with the dataset divided into training, validation, and test subsets. After training and performance evaluation, the trained model was saved and converted into a Fixed-Point representation, making it suitable for efficient hardware deployment.

Subsequently, dedicated memory initialization files (MIF) were generated, and a hardware architecture was designed in VHDL. The architecture includes memory blocks (ROM/BRAM), a MAC unit featuring 10 parallel computation units, a finite state machine (FSM), an ArgMax block, and additional components for cycle counting and displaying the result on a 7-segment display.

The project demonstrates the potential of FPGA in machine learning, emphasizing advantages such as efficient resource utilization, high parallelism, and reduced execution time compared to software implementations.

# Executive Summary

### Introduction and Objectives

This project demonstrates a hardware implementation of a machine learning (ML) algorithm on an FPGA (Field-Programmable Gate Array) and evaluates the benefits of parallel, deterministic execution in hardware versus sequential execution on a conventional CPU.
The selected workload is a handwritten-digit classifier for the MNIST dataset based on a compact Fully Connected Neural Network (FCNN/MLP).
The objectives are:

- To establish a clear, verifiable proof-of-concept for end-to-end ML inference on FPGA.
- To quantify resource utilization, timing closure, and latency.
- To compare the achieved inference performance against a software baseline (CPU implementation).

### Background and Rationale

ML-particularly artificial neural networks-has become a core enabler for image processing tasks. While CPUs offer programming flexibility, their fixed microarchitectures and sequential execution often limit real-time performance and energy efficiency in embedded scenarios. FPGAs, by contrast, allow designers to tailor the underlying micro-architecture to the model's dataflow and exploit massive fine-grained parallelism. The project survey reviewed two dominant implementation paths:

- Automated High-Level Synthesis (HLS/OpenCL) flows that accelerate development cycles.
- Handcrafted RTL (e.g., modular pipelines in works such as ALAMO) that maximize hardware efficiency at the expense of engineering effort.
  Handcrafted RTL (e.g., modular pipelines in works such as ALAMO) that maximize hardware efficiency at the expense of engineering effort.

## Methodology Overview

The development followed a software-to-hardware workflow:

- **Model training (software):** The Fully Connected Neural Network (single-layer FCNN) was trained in Python/PyTorch on Google Colab using the standard MNIST split (train/validation/test). After training, the model was validated on the test set to ensure generalization and then exported.
- **Hardware preparation (quantization & packaging):** We folded normalization into the parameters, quantized weights to 8-bit integers and biases to 32-bit fixed-point, and generated Memory Initialization Files (MIF) for inputs, weights, and biases.
- **Hardware implementation (RTL in VHDL):** The inference datapath and control were implemented as modular RTL blocks on a Terasic DE2-115 board (Intel/Altera Cyclone IV E EP4CE115). The design integrates on-chip ROM/BRAM, a 10-lane MAC engine, an ArgMax unit, a cycle counter, and a finite state machine (FSM) controller. The top-level maps to board I/O for observability (7-segment display, LEDs) and repeatability.

## Model and Dataflow

The classifier consumes a 28×28 grayscale image that is flattened into a 784-element vector. A single fully connected layer maps the input directly into logits for 10 classes (digits 0–9), the final decision is the argmax index.
The hardware mirrors this dataflow:

- **Parallel MACs:** Ten multiply-accumulate lanes process the feature stream in lockstep - one pixel per cycle per lane - accumulating partial sums for all classes concurrently.
- **Fixed-point arithmetic:** 8-bit weights and 8-bit inputs feed into 32-bit accumulators, striking a practical balance between classification accuracy and efficient use of FPGA hardware resources.
- **Deterministic control:** An FSM sequences ROM reads, MAC accumulation across 784 cycles, bias addition, and argmax selection. A one-cycle done pulse marks inference completion. The cycle counter exposes end-to-end latency in clock cycles.

## Implementation Details

- **Toolchain:** Quartus handled synthesis, fitting, and on-board programming, ModelSim was used for unit-level and subsystem simulation (e.g., bias load, accumulator updates, argmax correctness, FSM sequencing).
- **Memory packaging:** Three MIFs provide:
  - 784-byte input image (XQ).
  - 784 words of 80-bit packed weights (one 8-bit weight per class).
  - A single 320-bit word for ten 32-bit biases.

    Packing improves read bandwidth and simplifies ROM addressing.
- **Top-level integration:** The design is wrapped in a board-level shell that directly drives user I/O: HEX0 shows the predicted digit, LEDR reflects the cycle counter for direct empirical latency readings at 50 MHz.

## Verification and Evaluation

- **Functional validation:** The RTL blocks (ROMs, MAC10, ArgMax10, Controller FSM, cycle counter, and 7-segment decoder) were unit-tested and then integrated. Waveform inspection confirmed correct initialization, bias/accumulator behavior, and maximum-index selection across representative test vectors.
- **Accuracy baseline:** In software, the trained MLP achieved test accuracy in the range 92-93%, with confusion-matrix analysis and per-class accuracy plots highlighting typical digit confusions (e.g., {4,9} and {3,5}). This baseline informed the acceptable precision targets for fixed-point deployment.
- **Resource utilization:** On EP4CE115, the design used a tiny fraction of available resources (logic elements, registers, on-chip RAM, multipliers/DSPs), leaving ample headroom for deeper models, batching, or I/O extensions.
- **Timing closure and operating margin:** With a board clock of 50 MHz, static timing closed comfortably, worst-case slack remained strongly positive, and the computed Fmax exceeded the operating point, providing margin for future extensions or modest frequency scaling.
- **Latency and throughput:** One image inference completes in a fixed number of cycles dominated by 784 MAC cycles plus control/epilogue steps. On-board cycle-counter readings confirmed sub-millisecond latency at 50 MHz.

### Results (Representative Highlights)

- **End-to-end latency:** ~800 cycles per inference at 50 MHz (≈16 μs), measured on hardware via the cycle counter and visualized on LEDs, HEX0 displayed the predicted digit for each run.
- **Software accuracy:** ~92–93% test accuracy (software baseline) with stable loss convergence, per-class analysis reflects common handwritten digit confusions.
- **FPGA vs. CPU:** The FPGA achieved a speedup of roughly 12×–16× compared to a software inference of ≈0.2 ms on the Colab CPU, the observed average speedup was ~14× across runs.
- **Utilization:** <1% of logic elements, modest on-chip memory usage (~2%), and ~2% of DSP/multiplier blocks - leaving substantial capacity for scaling.
- **Timing:** Positive worst-case slack at 50 MHz, tool-reported Fmax near 78 MHz, indicating healthy timing margin.

### Discussion

The design validates the central premise: even simple, single-layer fully connected networks map naturally to FPGA fabrics. Packing weights and using a 10-lane MAC pipeline delivered single-image inference with constant latency, while fixed-point arithmetic provided an excellent accuracy-efficiency trade-off. The resulting footprint is extremely light, so deeper MLPs, early-exit branches, or even shallow convolutional front-ends could be added without exhausting resources. Compared to HLS-based approaches, handwritten RTL required more upfront design discipline but granted precise control over datapath width, schedule, and memory layout, which directly translated to predictable timing and resource usage.

## Limitations

- **Model compactness:** The model is intentionally compact and optimized for clarity and determinism rather than SOTA accuracy. CNNs would typically outperform a simple single-layer fully connected network on MNIST.
- **Single-image processing:** The prototype ingests one image at a time, the design does not yet exploit inter-image parallelism (batching) nor DMA-assisted streaming from external sources.
- **Energy measurements:** Energy measurements were out of scope. while fixed-point FPGAs are typically energy-efficient, a full power/energy comparison versus GPU/CPU would strengthen the argument.

## Conclusions

This project delivers a complete, reproducible demonstration of FPGA-based ML inference: from training and quantization through RTL implementation, timing closure, and on-board validation. The FPGA solution met timing with margin at 50 MHz, achieved sub-millisecond (≈16 μs) deterministic latency per image, and exhibited an order-of-magnitude speedup versus a CPU software baseline - all while consuming a very small fraction of the device resources. The results substantiate the case for FPGAs as practical, efficient ML inference engines in embedded, real-time contexts.

## Future Work

- **Model expansion:** Implementing deeper neural networks - through multiple fully connected layers or by adding convolutional layers- would likely improve classification accuracy and capture richer image features.
- **Multi-input support:** Extending the design to process multiple images in parallel (batch processing) would make better use of FPGA parallelism and increase throughput.
- **External communication interfaces:** Integrating UART, Ethernet, or PCIe would allow real-time data input from external systems and direct transmission of inference results back.
- **Cross-platform comparisons:** Conducting experiments on additional hardware platforms, such as GPUs or SoCs with ARM cores, would provide a broader perspective on the trade-offs between different machine learning deployment technologies.

# רשימת איורים


איור 1 - ארכיטקטורת ה-FPGA ........ 19
איור 2 - הגדרת ישות andgate ב-VHDL ........ 20
איור 3 - מימוש ארכיטקטורת Behavioral לישות AND Gate ........ 20
איור 4 - ייצוג RTL של שער AND לאחר סינתזה ........ 20
איור 5 - תרשים סכמתי של ANN ........ 23
איור 6 - תרשים סכמתי של MLP ........ 25
איור 7 - לוח הפיתוח DE2-115 ........ 30
איור 8 - תרשים זרימה של מודל רשת הנוירונים ........ 33
איור 9 - מודל רשת הנוירונים ........ 34
איור 10 - גרף ירידת ה-Loss לאורך ה-epochs ........ 35
איור 11 - גרף עליית ה-Accuracy ........ 35
איור 12 - Final Test Accuracy ........ 36
איור 13 - Confusion Matrix ........ 36
איור 14 - דיאגרמת Per-class Accuracy ........ 37
איור 15 - דוגמאות לתמונות ........ 37
איור 16 - תרשים בלוקים של הארכיטקטורה הכללית למימוש החומרתי ........ 41
איור 17 - קוד חבילת types_pkg בשפתVHDL ........ 42
איור 18- קוד חבילת pack_utils בשפת VHDL ........ 42
איור 19- סכמת RTL חיצונית של רכיב xq_rom8 ........ 44
איור 20 - סכמת RTL פנימית של רכיב xq_rom8 ........ 44
איור 21 - סכמת RTL חיצונית של רכיב wq_rom80 ........ 46
איור 22 - סכמת RTL פנימית של רכיב wq_rom80 ........ 46
איור 23 - סכמת RTL חיצונית של רכיב bstar_rom320 ........ 48
איור 24 - סכמת RTL פנימית של רכיב bstar_rom320 ........ 48
איור 25 - סכמת RTL חיצונית של רכיב mac10 ........ 50
איור 26 - סכמת RTL פנימית של רכיב mac10 ........ 50
איור 27 - סימולציה של טעינת Bias ........ 53
איור 28 - סימולציה של טעינת ערכי האוגרים (acc) ........ 53
איור 29 - סכמת RTL חיצונית של רכיב argmax10 ........ 54
איור 30 - סכמת RTL פנימית של רכיב argmax10 ........ 54
איור 31 - סימולציה של קבלת הערך המקסימלי ........ 56
איור 32 - סכמת RTL חיצונית של רכיב controller_fsm ........ 57
איור 33 - סכמת RTL פנימית של רכיב controller_fsm ........ 57
איור 34 - דיאגרמת מכונת מצבים ........ 58
איור 35 - סימולציה של FSM ........ 60
איור 36 - סכמת RTL חיצונית של רכיב cycle_counter ........ 61
איור 37 - סכמת RTL פנימית רכיב cycle_counter ........ 61
איור 38 - סימולציה של ספירת המחזורים ........ 62

איור 39 - סכמת RTL חיצונית של רכיב decoder_4to7 ...................................................... 63
איור 40 - סכמת RTL פנימית של רכיב decoder_4to7...................................................... 63
איור 41 - מיפוי קווי האותות בין תצוגת - HEX0 לבין רכיב ה-FPGA (Cyclone IV E)......... 64
איור 42 - סכמת RTL חיצונית של רכיב top_mnist_fc ...................................................... 65
איור 43 - סכמת RTL פנימית של רכיב top_mnist_fc...................................................... 65
איור 44 - סכמת מעטפת של הרכיב top_mnist_fc למיפוי אותות אל ממשקי הקלט/פלט בלוח DE2-115 ........................................................................................................................ 68
איור 45 - דו"ח הסינתזה.......................................................................................... 69
איור 46 - דו"ח ה-Timing Analyzer Summary............................................................... 70
איור 47 - דו"ח ה-Multicorner Timing Analysis Summary ............................................ 70
איור 48 - דו"ח ה-Fmax ............................................................................................ 71
איור 49 - מצב האיפוס בלוח DE2-115 : תצוגת 7-Segment מציגה 0,ונוריות ה- LED כבויות 71
איור 50 - תמונת ספרה 2 מתוך קבוצת ה-Test של מסד הנתונים MNIST.............................. 72
איור 51 - מצב ביצוע לאחר הרצה 1 : תצוגת 7-Segment מציגה את הספרה 2......................... 72
איור 52 - קוד הרצה ובדיקת זמן חיזוי (Inference) של המודל המאומן ב-Python ................. 73
איור 53 - תמונת ספרה 7 מתוך קבוצת ה-Test של מסד הנתונים MNIST.............................. 74
איור 54 - מצב ביצוע לאחר הרצה 2 : תצוגת 7-Segment מציגה את הספרה 7......................... 74
איור 55 - קוד הרצה ובדיקת זמן חיזוי (Inference) של המודל המאומן ב-Python ................. 75
איור 56 - תמונת ספרה 1 מתוך קבוצת ה-Test של מסד הנתונים MNIST.............................. 76
איור 57 - מצב ביצוע לאחר הרצה 3 : תצוגת 7-Segment מציגה את הספרה 1......................... 76
איור 58 - קוד הרצה ובדיקת זמן חיזוי (Inference) של המודל המאומן ב-Python ................. 77

# רשימת משוואות

משוואה 1 - נוסחת Artificial Neuron Equation ........ 24
משוואה 2 - נוסחת Gradient Descent ........ 26
משוואה 3 - נוסחת Softmax ........ 26

# קיצורים ומונחים

CPU - Central Processing Unit

GPU - Graphical Processing Unit

FPGA - Field Programmable Field Array

RTL - Register Transfer Level

HDL - Hardware Description Language

VHDL - VHSIC Hardware Description Language

VHSIC - Very High-Speed Integrated Circuit

GPIO - General-Purpose Input/Output

USB - Universal Serial Bus

ML - machine learning

NN - Neural Networks

ANN - Artificial Neural Network

CNN - Convolutional Neural Networks

RNN - Recurrent Neural Networks

AI - Artificial Intelligence

MLP - Multilayer Perceptron

DSP - Digital Signal Processing

ADC - Analog to Digital Converter

JPEG - Joint Photographic Experts Group

HLS4ML - High Level Synthesis For Machine Learning

HPS - Hard Processor System

DDR-SDRAM - Double Data Rate Synchronous Dynamic Random-Access Memory

HLS - High-Level Synthesis

MIF - Memory Initialization File

# פרק 1 - מבוא

## 1.1 רקע כללי

תחום הבינה המלאכותית ולמידת המכונה (Machine Learning - ML) עבר בעשור האחרון מהפכה של ממש. אלגוריתמים חישוביים, ובעיקר רשתות עצביות מלאכותיות (Artificial Neural Networks - ANN), מאפשרים למערכות לבצע משימות שבעבר נחשבו מורכבות במיוחד, כגון זיהוי דיבור, עיבוד תמונה, זיהוי פנים, והסקה חכמה מנתוני חיישנים. אחד התחומים המרכזיים בהם מיושמות רשתות עצביות הוא עיבוד תמונה (Image Processing), ובפרט זיהוי ספרות מתוך מאגרי מידע סטנדרטיים כגון MNIST, המהווה בסיס לבחינת יכולות סיווג חזותי.

היישום של אלגוריתמים אלו מתבצע לרוב על גבי מעבדים מרכזיים (CPU). אמנם פלטפורמות אלו מציעות גמישות תכנותית רבה, אך הן מאופיינות בצריכת הספק גבוהה ובביצועים שאינם תמיד מותאמים ליישומי זמן אמת (Real-Time). כאשר נדרש ליישם אלגוריתמים בסביבות מוגבלות משאבים - לדוגמה מערכות משובצות (Embedded Systems), רכיבים ניידים או מערכות IoT - נדרש פתרון יעיל יותר הן מבחינת ביצועי חישוב והן מבחינת צריכת אנרגיה.

בנקודה זו נכנסים לתמונה רכיבי FPGA (Field Programmable Gate Array). בניגוד ל-CPU בהם הארכיטקטורה קבועה מראש, FPGA מאפשר תכנות מחדש של מבנה החומרה הפנימית בהתאם לאלגוריתם הרצוי. הדבר מאפשר ביצוע מקבילי (Parallelism), ניצול טוב יותר של משאבי החומרה, והתאמה ייעודית לצורכי היישום. יכולות אלו הפכו את ה-FPGA לפלטפורמה מועדפת במימוש אלגוריתמים עתירי חישוב, ובפרט בתחום עיבוד אותות ותמונה.

## 1.2 תיאור הבעיה

אלגוריתמי למידת מכונה, ובעיקר רשתות Fully Connected ורשתות קונבולוציה, מצריכים פעולות כפל וחיבור רבות המבוצעות על מטריצות גדולות. כאשר אלגוריתמים אלו מיושמים על גבי CPU, זמן הריצה עלול להיות ארוך באופן יחסי, ובמקרים רבים הוא מונע שימוש בזמן אמת.

במערכות משובצות (Embedded) או במערכות ניידות - לדוגמה רחפנים, רובוטים, מערכות חכמות לניטור סביבתי - נדרשת פלטפורמה חסכונית, יעילה ובעלת זמן תגובה קצר. לפיכך, הבעיה המרכזית הניצבת היא כיצד לממש אלגוריתם למידת מכונה בצורה יעילה בחומרה, כך שניתן יהיה להדגים יתרון ממשי בביצועים ובצריכת ההספק לעומת מימושו על גבי CPU.

### 1.3 מטרת הפרויקט

מטרת פרויקט הגמר היא להציג הוכחת תכן (Proof of Concept) למימוש אלגוריתם למידת מכונה על FPGA ולהדגים את יתרונות הפלטפורמה החומרתית ביחס ל-CPU.

בפרויקט נבחר אלגוריתם בסיסי לזיהוי ספרות מתוך מאגר MNIST, המבוסס על רשת Fully Connected פשוטה ומסווג ליניארי. המימוש יתבצע בשני שלבים מרכזיים:

שלב האימון - הרשת תאומן בסביבת Google Colab באמצעות שפת Python וספריית PyTorch, עד להשגת מודל מבוסס משקלים המאומן על סט הנתונים.

שלב ההסקה (Inference) - המודל המאומן יומר לייצוג חומרתי, וימומש על גבי FPGA באמצעות שפת תיאור חומרה VHDL. שלב זה יאפשר לבחון את יעילות החומרה בהשוואה למימוש תוכנה (Software) על גבי CPU.

באמצעות יישום זה תודגם יכולת ה-FPGA להאיץ אלגוריתמי למידת מכונה של עיבוד תמונה ולשמש פלטפורמה חלופית ויעילה למימושי CPU במקרים של דרישות זמן אמת או מגבלות צריכת אנרגיה.

### 1.4 מבנה הספר

פרק 1 - מבוא: מציג רקע כללי, תיאור הבעיה ומטרות הפרויקט.

פרק 2 - רקע תיאורטי: סוקר את הספרות והבסיס התיאורטי הנדרש, כולל עקרונות למידת מכונה, עיבוד תמונה, מבנה FPGA והבדלים מול CPU, פתרונות קיימים.

פרק 3 - מימוש הפרויקט: תיאור האלגוריתם שנבחר (Fully Connected MNIST), סביבת האימון (Google Colab) והבדיקה, סביבת המימוש (Quartus, FPGA),מימוש האלגוריתם.

פרק 4 - תוצאות: מציג את תוצאות המימוש וההרצות שבוצעו.

פרק 5 - מסקנות: מנתח את התוצאות שהוצגו בפרק 4 ומציע כיווני עבודה להמשך.

בסוף הספר תופיע רשימת ביבליוגרפיה.

# פרק 2 - רקע תיאורטי

בפרק זה נפרוש את היסודות התאורטיים הדרושים להבנת הפרויקט. תחום מימוש אלגוריתמי למידת מכונה לעיבוד תמונה על גבי חומרה.

## CPU 2.1

יחידות עיבוד מרכזיות (Central Processing Units - CPU) מהוות את ליבת מערכות המחשב זה עשרות שנים, והן אחראיות על ביצוע רציף של סט הוראות המוגדרות בתוכנה [8].
המעבד מבצע תוכנית באופן סדרתי (ליניארי) ולכן הוא אופטימלי עבור יישומים בהם נדרש חישוב חד תהליכי (single process) בעל רצף פעולות ברור.
עם זאת, ל-CPU קיימת מגבלה ביכולת מימוש מקביליות רחבה (Massive Parallelism), הנדרשת לעיבוד נתונים בהיקף גדול או בזמן אמת.

מבנה החומרה הפנימי של המעבד נקבע על ידי היצרן ואינו ניתן לשינוי על ידי המשתמש, ולכן השימוש בו נשאר כללי וסטנדרטי. גישה זו מאפשרת ל-CPU להריץ מגוון רחב של פונקציות ותוכנות, אך הופכת אותו לפחות יעיל במשימות ייחודיות עתירות חישוב - לדוגמה, דחיסת וידאו או הפעלת רשתות עצביות, אשר דורשות חומרה ייעודית ומקבילית [8].

לאחר סקירת מאפייני CPU והבנת מגבלותיו בביצועי חישוב מקביליים, נציג את שפת Python, המשמשת ככלי מרכזי לפיתוח ואימון מודלים בלמידת מכונה.

## Python 2.2

Python היא שפת תכנות דינמית מהנפוצות ביותר כיום, שזכתה להכרה בזכות הקריאות הגבוהה של הקוד והמבנים הפשוטים שהיא מציעה. השפה תוכננה כך שתאפשר ביטוי של תוכניות מורכבות בצורה תמציתית וברורה, תוך שמירה על פשטות תחזוקתית. בכך, Python מצליחה לשלב בין נגישות וידידותיות למשתמש לבין יעילות חישובית המספקת מענה ליישומים הנדסיים ומדעיים מתקדמים.

בקהילת Python, Machine Learning תפסה מקום מרכזי הודות למגוון רחב של ספריות ייעודיות:

- TensorFlow 2 - ספרייה חופשית בקוד פתוח ללמידת מכונה, עם דגש על אימון והסקת רשתות עצביות עמוקות. הארכיטקטורה הגמישה שלה מאפשרת יישום אלגוריתמים על מגוון פלטפורמות - החל ממעבדים (CPU) דרך מאיצים גרפיים (GPU) ועד מעבדי TPU ייעודיים, ובקנה מידה הנע ממחשבים אישיים ועד אשכולות שרתים ומכשירים ניידים.
- PyTorch - ספרייה בקוד פתוח המפותחת על ידי Facebook AI Research, שנחשבת לאחת הפלטפורמות הנפוצות ביותר כיום למחקר וליישומים מעשיים בלמידת מכונה [11]. היא מתאפיינת בגמישות, פשטות תפעול, ותמיכה דינמית בגרפים חישוביים, ולכן משמשת רבות במחקר אקדמי וגם ביישומי תעשייה.

- NumPy - ספרייה מתמטית בסיסית המאפשרת עבודה יעילה עם מערכים רב-ממדיים, לצד מגוון רחב של פונקציות מתמטיות, והיא מהווה רכיב יסוד כמעט בכל פרויקט של למידת מכונה ב-Python.
- OpenCV - ספרייה לעיבוד תמונה וראייה ממוחשבת, המאפשרת עבודה עם תמונות ווידאו, פילוחים, זיהוי תבניות ועוד. היא מהווה כלי משלים חשוב ליישומים המשלבים עיבוד תמונה עם למידת מכונה.

מעבר לספריות אלו קיימות גם ספריות נוספות (כגון Scikit-learn, Pandas), אולם הכלים שהוזכרו לעיל מהווים את התשתית העיקרית שעליה מתבססים מרבית מימושי ה-Machine Learning וה-Deep Learning בפועל.

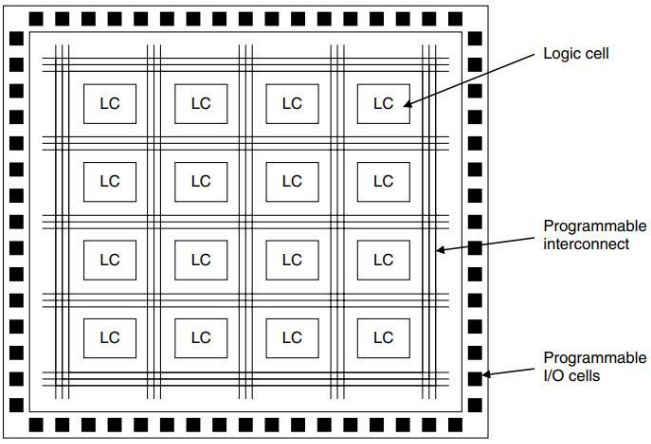
עם זאת, בעוד ש-Python מספקת סביבת פיתוח ואימון נוחה וגמישה לאלגוריתמים של למידת מכונה, שלב המימוש החומרתי דורש פלטפורמות ייעודיות וניתנות להתאמה. אחת המרכזיות שבהן היא ה-FPGA, אשר תוצג בסעיף הבא.

## FPGA 2.3

FPGA (Field Programmable Gate Array) הוא שבב דיגיטלי המורכב ממערך רחב של יחידות לוגיות ניתנות לתכנות, אשר מקושרות ביניהן באמצעות רשת חיבורים גמישה. המשתמש יכול להגדיר מחדש את תפקודן של יחידות אלו בהתאם לצרכים ספציפיים, ובכך לקבל פתרון חומרתי מותאם ליישום הנבחר [8]. יתרון מרכזי של רכיבי FPGA הוא היכולת לבצע עיבוד מקבילי: במקום להריץ סדרה של פעולות באופן טורי (כפי שמתרחש ב-CPU), ניתן לפרק את המשימה למספר תתי פעולות המבוצעות במקביל, וכך להשיג שיפור ניכר בביצועים ובזמני הריצה.

הגדרת הפונקציונליות של ה-FPGA מתבצעת באמצעות שפת תיאור חומרה (HDL), כגון VHDL או Verilog. סביבת הפיתוח (בפרויקט זה - Quartus) ממירה את הקוד הכתוב בשפת HDL דרך שלבי סינתזה ו-Place & Route למימוש פיזי של שערים לוגיים ומבני חיבור ביניהם. למעשה, תהליך זה מתרגם קוד טקסטואלי לייצוג חומרתי אמיתי המבוסס על שערים לוגיים ומבני זיכרון פנימיים [10].

בפרויקט זה נבחר להשתמש ב-FPGA כבסיס למימוש שלב ההסקה (Inference) של האלגוריתם, זאת בשל יתרונו המובהק בביצוע חישובים מקביליים. תכונה זו מאפשרת חלוקת פעולה רציפה יחידה למספר פעולות עצמאיות המתבצעות בו-זמנית, ובכך קיצור משמעותי של זמן החישוב לעומת מימוש על גבי CPU בלבד. תרשים סכמתי של מבנה פנימי טיפוסי של FPGA מוצג באיור 1.




*איור 1 - ארכיטקטורת ה-FPGA*

עם זאת, על מנת לנצל את יכולות ה-FPGA ולממש עליו אלגוריתם ייעודי, יש צורך בשפת תיאור חומרה (HDL) שתאפשר לתאר את פעולת הרכיב ברמה לוגית ולתרגם את האלגוריתם למבנה חומרתי אמיתי. אחת השפות המרכזיות והנפוצות בתחום זה היא VHDL, אשר תשמש גם בפרויקט הנוכחי.

## VHDL 2.4

בעשור האחרון התפתחה באופן משמעותי תעשיית הרכיבים המתוכנתים בצפיפות גבוהה (Programmable Logic Devices) מה שהוביל לעלייה ניכרת בשימוש בשפות תיאור חומרה שפות אלו מהוות נדבך עיקרי בתכנון מערכות דיגיטליות מודרניות, בהגדרת ארכיטקטורות מורכבות, ובביצוע סימולציות ובדיקות מקיפות [10] .

VHDL (VHSIC Hardware Description Language) היא אחת השפות הנפוצות והיעילות ביותר בתחום, וזאת ממספר סיבות עיקריות:

- יכולתה לתאר התנהגות מערכת מורכבת מאפשרת לטפל בתכנונים הכוללים מיליוני שערים בצורה יעילה.
- תכנון בשפת VHDL הינו מודולרי, דבר המקל על תחזוקה ושדרוג של מערכות קיימות.
- פשטות ומהירות הפיתוח בשפה מצמצמות עלויות וסיכונים בפיתוחים מורכבים.
- היותה תקנית מאפשרת יישום תפיסות תכנון מגוונות, תוך ביצוע סינתזה וסימולציה מתקדמת.
- כלי הפיתוח התומכים ב-VHDL כוללים ספריות יצרן המאפשרות אופטימיזציה וביצועים גבוהים.

לימוד השפה נחשב מהיר יחסית ומאפשר השגת תוצאות משמעותיות כבר בשלבים מוקדמים של הפרויקט. בנוסף, קיים מגוון רחב של ספרות מקצועית, מדריכים ואתרים ברשת, מה שמקל על מהנדסים ומפתחים להעמיק את הידע ולמצוא פתרונות לבעיות בזמן אמת.

תהליך הפיתוח באמצעות VHDL מתבצע בהתאם לעקרונות התכנון הספרתי המקובל: בניית ארכיטקטורה למערכת, הגדרת יחסי גומלין בין יחידות מבצעות ובקרות, כתיבת קוד, סימולציה, ולבסוף סינתזה לייצור המעגל הפיזי. לדוגמה, ניתן להגדיר ישות פשוטה (Entity) שמופיעה באיור 2 המייצגת שער לוגי AND ההגדרה כוללת את מבנה הכניסות והיציאות (I/O Ports) לאחר מכן מימוש ארכיטקטורת הרכיב (Architecture) שמופיעה באיור 3 המתארת את ההתנהגות - כלומר חיבור הכניסות כך שביציאה יתקבל הפלט הלוגי A AND B .

לאחר תהליך הקומפילציה, סביבת הפיתוח למשל Quartus ממשת את הקוד כתצורה חומרתית אמיתית, ומציגה אותו במבנה RTL (Register Transfer Level) שמופיע באיור 4 הממחיש כיצד ההגדרה הטקסטואלית הופכת למעגל לוגי ממשי המבוסס על שערים.

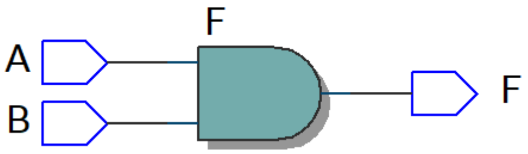
שלבים:

```
entity andgate is
    port( A: in std_logic;
          B: in std_logic;
          F: out std_logic
        );
end andgate;
```

*איור 2 - הגדרת ישות andgate ב-VHDL*

```
architecture Behavioral of andgate is
begin
    F <= A and B;
end Behavioral
```

*איור 3 - מימוש ארכיטקטורת Behavioral לישות AND Gate*




*איור 4 - ייצוג RTL של שער AND לאחר סינתזה*

## 2.5 עיבוד תמונה דיגיטלי

עיבוד תמונה דיגיטלי הוא תחום במדעי ההנדסה העוסק בשימוש במחשב דיגיטלי לצורך עיבוד, שיפור וניתוח של תמונות. מדובר בתת תחום של עיבוד אותות דיגיטליים, אשר מציע יתרונות רבים על פני עיבוד תמונה אנלוגי שהיה מקובל בעבר [9] .

תהליך העיבוד מתחיל בהמרת התמונה מהמישור האנלוגי (אות רציף) למישור הדיגיטלי באמצעות ממיר אנלוגי לדיגיטלי (ADC) לאחר ההמרה, התמונה מיוצגת כטבלה דו-ממדית של פיקסלים, כאשר כל פיקסל מייצג ערך מספרי המתאר את עוצמת הבהירות או הצבע. ייצוג זה מאפשר החלת מגוון רחב של אלגוריתמים מתמטיים וחישוביים על גבי התמונה [9] .

במהלך העיבוד ניתן לבצע שורה של פעולות כגון סינון (Filter) באמצעות קונבולוציות, הסרת רעשים, חידוד קצוות ושיפור איכות התמונה. כמו כן, ניתן לאחסן את התוצר הסופי בפורמטים דיגיטליים סטנדרטיים כגון JPEG או PNG מה שמאפשר ניתוח חוזר, השוואות לתמונות אחרות, או יישומים מתקדמים יותר כגון זיהוי תבניות, סיווג אובייקטים או מעקב אחר תנועה.

התפתחות התחום נובעת משלושה גורמים מרכזיים :

- התקדמות טכנולוגית במחשוב - מעבדים מהירים וחומרה ייעודית כגון GPU ו- FPGA מאפשרים עיבוד בזמן אמת של תמונות באיכות גבוהה.
- שיפורים מתמטיים ואלגוריתמיים - פיתוח שיטות חדשות בעיבוד אותות, סטטיסטיקה ולמידת מכונה, המייעלות את אלגוריתמי העיבוד.
- הביקוש הגובר ליישומים - תחום עיבוד התמונה נמצא בשימוש נרחב בתחומים מגוונים, החל ברפואה וחקלאות, דרך תעשיית הביטחון והמדע, ועד מערכות חכמות מבוססות בינה מלאכותית.

השילוב של שלושת הגורמים הללו הפך את עיבוד התמונה הדיגיטלי לכלי מרכזי בעולם ההנדסה והמחקר, ומבסס את מקומו כתשתית טכנולוגית למערכות מתקדמות רבות.

עיבוד תמונה דיגיטלי מספק את הכלים ההנדסיים הבסיסיים לייצוג ולשיפור תמונות, אך בשנים האחרונות הועצם תחום זה באמצעות שילוב של טכניקות למידת מכונה. בעוד שעיבוד קלאסי מתבסס על פילטרים ואלגוריתמים מתמטיים מוגדרים מראש, למידת מכונה מאפשרת למערכות ללמוד דפוסים מתוך הדאטה עצמו ולבצע משימות מתקדמות כגון סיווג ספרות, זיהוי עצמים ומעקב אחר תנועה בדיוק רב ובזמן אמת.

## 2.6 למידת מכונה

למידת מכונה (Machine Learning) היא תת-תחום בבינה מלאכותית (Artificial Intelligence), המתמקד בפיתוח אלגוריתמים ומודלים המאפשרים למחשב ללמוד מתוך נתונים ולשפר את ביצועיו ללא צורך בהוראות מפורשות לכל משימה. בשונה מתוכנה קלאסית, המבוססת על סט הוראות מוגדר מראש, בלמידת מכונה המערכת מפתחת מודל מתמטי על בסיס דוגמאות (Data), ומסוגלת לבצע חיזוי, סיווג או זיהוי של דפוסים חדשים שלא נצפו בעבר.

תהליך הלמידה כולל לרוב שני שלבים מרכזיים:

- אימון (Training) - בשלב זה המודל נחשף לנתונים רבים ולומד את הקשרים הסטטיסטיים ביניהם באמצעות שיטות אופטימיזציה שונות.
- הסקה (Inference) - בשלב זה המודל המאומן מיישם את הידע שרכש על דוגמאות חדשות, ומספק תשובות או חיזויים בזמן קצר ובדיוק גבוה.

למידת מכונה מהווה כיום בסיס למגוון רחב של יישומים מתקדמים - מזיהוי דיבור ותמונה, דרך מערכות המלצה מותאמות אישית, ועד יישומי רפואה, תחבורה חכמה ובקרה תעשייתית.

### 2.6.1 סוגי בעיות בלמידת מכונה

בלמידת מכונה ניתן לזהות מספר סוגי בעיות מרכזיות, אשר לכל אחת מהן מטרות וגישות חישוב שונות:

- רגרסיה (Regression) - בעיה שמטרתה חיזוי ערך מספרי רציף על בסיס נתוני קלט. דוגמאות שכיחות הן חיזוי מחירי נדל״ן על פי מאפייני דירה או תחזית טמפרטורות בהתבסס על נתוני עבר [7].
- אשכולות (Clustering) - בעיה בה מבוצעת חלוקה של הדאטה לקבוצות על פי דמיון פנימי, מבלי לדעת מראש את מספר המחלקות. יישומים אפשריים כוללים קיבוץ לקוחות לפי דפוסי קנייה או גילוי תתי קבוצות גנטיות בנתונים ביולוגיים [7].
- סיווג (Classification) - בעיה שבה המחשב נדרש לנבא את המחלקה (קטגוריה) שאליה שייך קלט חדש. דוגמאות לכך הן זיהוי ספרות, סיווג תמונות לפי אובייקטים, או סינון מיילים כספאם לעומת מיילים לגיטימיים. בפרויקט נתמקד בסוג בעיה זה באמצעות סיווג ספרות מתוך מאגר MNIST [11].

### 2.6.2 רשת נוירונים מלאכותית

רשת נוירונים מלאכותית (Artificial Neural Network - ANN) המופיעה באיור 5 היא מודל חישובי המנסה לחקות את אופן פעולת המוח האנושי, ובפרט את אופן העיבוד של נוירונים ביולוגיים. ברשת כזו קיימות שכבות של יחידות חישוביות (נוירונים מלאכותיים), כאשר כל יחידה מבצעת פעולה מתמטית פשוטה : חיבור משוקלל של הקלטים והעברת התוצאה דרך פונקציית הפעלה לא ליניארית.

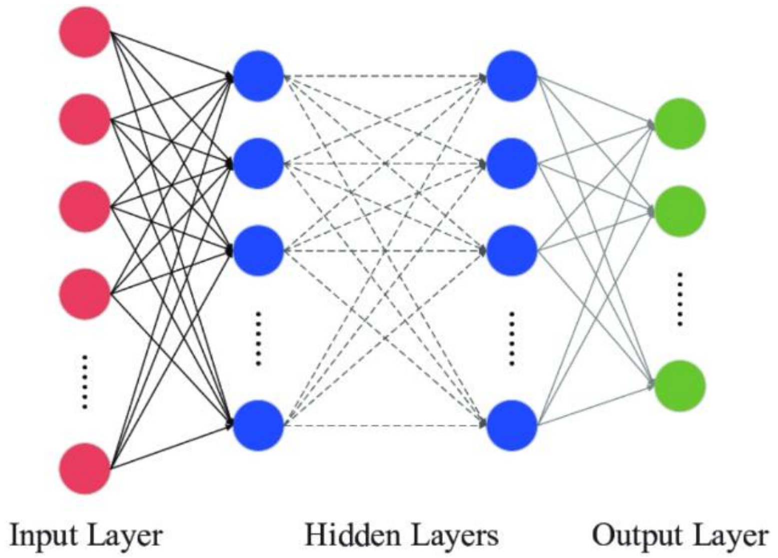
הארכיטקטורה הבסיסית של רשת נוירונים כוללת שלושה רכיבים עיקריים :

- שכבת קלט (Input Layer) - מקבלת את נתוני הגלם (למשל, ערכי הפיקסלים בתמונה).
- שכבות חבויות (Hidden Layers) - מבצעות את עיקר החישוב באמצעות חיבורים משוקללים בין הנוירונים, ובכך מאפשרות לרשת ללמוד ייצוגים מורכבים של הנתונים.
- שכבת פלט (Output Layer) - מפיקה את התוצאה הסופית, לדוגמה הקטגוריה שאליה שייך הקלט.

בתהליך האימון (Training), משקולות החיבורים בין הנוירונים מותאמות בהדרגה באמצעות אלגוריתם אופטימיזציה (בדרך כלל Gradient Descent), כך שהרשת תלמד למזער את פונקציית השגיאה (Loss Function). לאחר סיום האימון, הרשת עוברת לשלב הסקה (Inference) שבו היא מיישמת את הידע שנרכש על דוגמאות חדשות.

יתרונן של רשתות נוירונים הוא ביכולתן לבצע למידה לא ליניארית מורכבת, ולזהות דפוסים שקשה להגדיר באמצעות אלגוריתמים קלאסיים. מאפיין זה הופך אותן לכלי עוצמתי במיוחד לבעיות של סיווג תמונה, זיהוי דיבור, עיבוד אותות ובקרת מערכות.

בפרויקט זה נתמקד ברשת פשוטה מסוג Fully Connected Neural Network, המתאימה למשימות סיווג בסיסיות כגון זיהוי ספרות כתב יד מתוך מאגר MNIST.




*איור 5 - תרשים סכמתי של ANN*

### 2.6.3 רשת Fully Connected Neural Network

Multilayer Perceptron (MLP) המופיעה באיור 6 היא אחת הארכיטקטורות הבסיסיות ביותר בעולם הרשתות הנוירונים המלאכותיות. מדובר ברשת בה כל נוירון בשכבה מחובר לכל הנוירונים בשכבה העוקבת, ולכן היא נקראת Fully Connected.

הרשת מורכבת משלושה חלקים עיקריים:

- שכבת קלט - מקבלת את הנתונים בצורת וקטור תכונות.
- שכבות חבויות - מעבדות את המידע באמצעות חיבורים משוקללים ופונקציות הפעלה לא ליניאריות.
- שכבת פלט - מחזירה את התוצאה הסופית, לרוב בצורה של הסתברות או קטגוריה.

החישוב בכל נוירון ניתן לתיאור מתמטי באמצעות:

$$y = f(\sum_{i=1} w_i x_i + b)$$

*משוואה 1 - נוסחת Artificial Neuron Equation*

כאשר:

$x_i$ - ערכי הקלטים (features).

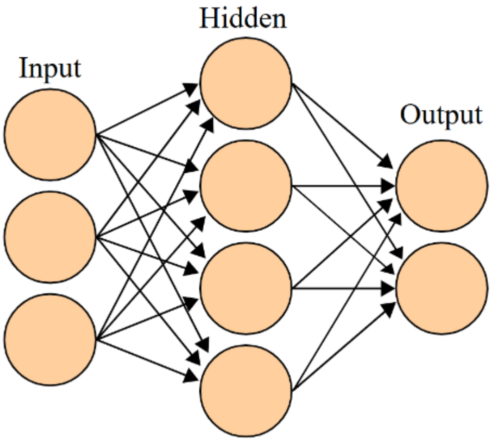
$w_i$ - משקלים הנלמדים בתהליך האימון.

$b$ - פרמטר הביאס (Bias) שמאפשר גמישות נוספת בפונקציה.

$f$ - פונקציית ההפעלה (Activation Function).

$y$ - הפלט של הנוירון, המועבר לשכבה הבאה ברשת.

באופן זה, כל נוירון מהווה יחידת חישוב פשוטה, אך כאשר רבים מהם מקושרים יחד במספר שכבות, נוצר מודל חישובי עוצמתי המסוגל ללמוד ייצוגים מורכבים של נתונים ולבצע משימות כגון סיווג, חיזוי וזיהוי תבניות.




*איור 6 - תרשים סכמתי של MLP*

הייחוד של MLP טמון ביכולת שלה ללמוד קשרים לא ליניאריים מתוך הדאטה, דבר שהופך אותה לכלי בסיסי למגוון רחב של בעיות למידת מכונה - ובעיקר לבעיות סיווג. בשל הפשטות המבנית שלה, MLP משמשת לעיתים קרובות כמודל כניסה ללימוד ליישום עקרונות למידת מכונה, וכבסיס תאורטי למודלים מתקדמים יותר כמו CNN או RNN.

רשתות נוירונים קונבולוציוניות (CNN) הן הארכיטקטורה הנפוצה ביותר כיום לבעיות עיבוד תמונה, שכן הן מאפשרות הפחתה משמעותית במספר הפרמטרים לצד למידה יעילה של תכונות מקומיות בתמונה.

עם זאת, בפרויקט זה לא נעשה שימוש ב-CNN, אלא נבחר מבנה Fully Connected - מקרה פרטי פשוט של MLP - אשר מספק פתרון מספק לבעיה הנבחרת, סיווג ספרות ממאגר MNIST, ומאפשר הדגמה ישירה וברורה של מימוש תהליך ההסקה על גבי FPGA.

2.6.4 מושגים בלמידת מכונה

בפרויקט זה נעשה שימוש במספר מונחים בסיסיים מעולם למידת המכונה, אשר יוצגו להלן לצורך אחידות והבנת התוכן:

- Dataset (מערך נתונים): אוסף הדוגמאות (Samples) שעליו מתבצע תהליך הלמידה. במרבית המקרים הדאטה מחולק לסט אימון (Training Set), סט ולידציה (Validation Set) וסט בדיקה (Test Set). בפרויקט נעשה שימוש במאגר MNIST.
- Feature (תכונה): מאפיין מספרי של הדוגמה, המהווה את הקלט לאלגוריתם הלמידה. לדוגמה, ערכי פיקסלים בתמונה.
- Label (תווית): הפלט המצופה (Ground Truth) המשויך לדוגמה מסוימת. במאגר MNIST מדובר בספרה 0-9.
- Epoch: מעבר אחד מלא על כל הדאטהסט במהלך האימון. בכל Epoch המודל מעדכן את המשקלים בהתאם לדוגמאות שעובדו.
- Loss Function (פונקציית שגיאה): פונקציה מתמטית שמודדת את הפער בין הפלט שהמודל הפיק לבין הפלט הרצוי (Label). מטרת תהליך האימון היא למזער פונקציה זו.
- Optimizer (אופטימייזר): אלגוריתם לעדכון המשקלים במודל כך שה-Loss יצטמצם. בפרויקט נעשה שימוש באלגוריתם Adam.
- Gradient Descent (ירידת מפל): אלגוריתם חישובי לחיפוש נקודת המינימום של פונקציית השגיאה.

  העדכון מתבצע על-פי הנוסחה: $w \leftarrow w - \alpha \nabla_i L_i(w)$

  *משוואה 2 - נוסחת Gradient Descent*

  - $w$ - מייצג את משקל המודל.
  - $\alpha$ - קצב הלמידה (Learning Rate).
  - $\nabla_i L_i(w)$ - נגזרת (Gradient) של פונקציית השגיאה ביחס למשקל.
- Activation Function (פונקציית הפעלה): פונקציה לא ליניארית המוחלת על הנוירונים, ומאפשרת לרשת ללמוד ייצוגים מורכבים (למשל Sigmoid).
- Softmax: פונקציה מתמטית הממירה וקטור ערכים רציפים (logits) לוקטור הסתברויות, כך שכל ערך נמצא בטווח [0,1] וסכום כל ההסתברויות שווה ל-1. פונקציה זו מאפשרת לפרש את הפלט של המודל כהסתברות להשתייכות לכל אחת מהמחלקות.

  הנוסחה היא: $\frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$

  *משוואה 3 - נוסחת Softmax*

  - $z_i$ - הערך עבור מחלקה $i$ .
  - $n$ - מספר המחלקות הכולל.
- Inference (הסקה): השלב שבו מודל מאומן מקבל דוגמאות חדשות ומספק תחזית (Prediction) או סיווג.

## 2.7 פתרונות קיימים בשוק

### 2.7.1 HLS/OpenCL

אחת הגישות המרכזיות למימוש אלגוריתמי למידה עמוקה על גבי FPGA מבוססת על שימוש ב-מודלים מאומנים בפורמטים פתוחים, כגון ONNX (Open Neural Network Exchange), והמרתם באופן אוטומטי ליישום חומרתי באמצעות כלי HLS (High-Level Synthesis). תהליך זה כולל תרגום המודל המאומן לקוד ברמת תיאור גבוהה (C/C++/OpenCL), ולאחר מכן מיפויו לארכיטקטורת חומרה באמצעות כלים ייעודיים. במסגרת המיפוי מתבצעת בניית צנרת (Pipeline), חלוקה וניצול של יחידות DSP ו-BRAM, וכן תזמון יעיל של זרימת הנתונים בתוך ה-FPGA.

במחקרים שונים הודגם כי ניתן לבצע ייצור אוטומטי של kernels ממודלי CNN המאוחסנים בפורמט פתוח, באופן שמקצר משמעותית את המרחק בין תיאור המודל ברמת עיבוד גבוהה לבין ליבת FPGA ברת מיפוי [2]. בנוסף, מימושים מבוססי OpenCL הציגו יכולת לבצע מעבר מהיר מתוכנה לחומרה במודלים אלגוריתמיים שונים - כגון אלגוריתם K-Means - תוך אינטגרציה נוחה עם כלי הפיתוח של יצרני FPGA [4].

לסביבה זו מספר יתרונות בולטים: קיצור משמעותי של זמן הפיתוח, עקומת למידה נוחה יחסית למפתחים המגיעים מעולם התוכנה, ויכולת מעבר טבעית מסביבות פיתוח של Python ו-ML ליישום חומרתי. עם זאת, קיימות גם מגבלות - יעילות ניצול המשאבים והשהיית החישוב תלויות באופן ישיר באיכות האופטימיזציות שמבצע כלי ה-HLS, ולעיתים נופלות ביעילותן בהשוואה למימוש RTL ייעודי המתוכנן ידנית.

### 2.7.2 RTL מודולרי - ALAMO

ALAMO (Acceleration of deep Learning Algorithms with a Modularized RTL compiler) היא מסגרת פיתוח שהוצעה בשנים האחרונות לצורך מימוש יעיל של רשתות למידה עמוקה על גבי FPGA. בשונה מזרימות פיתוח ברמה גבוהה (HLS/OpenCL) ALAMO מבצעת פירוק של המודל העילי ליחידות חישוב בסיסיות (פרימיטיביים) - כגון פעולות מכפלה חיבור (MAC), קונבולוציה או pooling - ולאחר מכן ממפה אותן באופן ישיר ל-RTL מודולרי. גישה זו מאפשרת ליצור קוד Verilog/VHDL מותאם משימה, אשר ניתן לסינתזה ישירות על גבי ה-FPGA.

במחקר שבוצע על ידי Ma ועמיתיו [1], הודגם כי ALAMO משיגה שיפור משמעותי ב-throughput בהשוואה לזרימות כלליות מבוססות HLS. החוקרים מימשו שתי רשתות CNN עתירות חישוב, שכל אחת מהן דרשה מעל מיליארד פעולות לכל תמונת קלט, והראו כי הפירוק המודולרי והשליטה המלאה בצנרת (pipelining), במיפוי ל-DSP ובניהול הזיכרון מאפשרים ניצול מיטבי של משאבי ה-FPGA.

היתרון המרכזי של ALAMO טמון בכך שהיא משלבת בין גמישות תכנונית - הודות למבנה מודולרי הניתן להתאמה למשימות שונות - לבין יעילות חומרתית גבוהה שמקורה במימוש RTL ישיר. יחד עם זאת, החיסרון הוא זמן פיתוח ארוך יותר בהשוואה ל-HLS, וכן דרישה לרמה גבוהה יותר של ידע בארכיטקטורת FPGA ותכנון RTL.

## 2.8 יתרונות מימוש אלגוריתם ב-FPGA

רכיבי FPGA מציעים יתרונות מהותיים במימוש אלגוריתמי למידת מכונה בהשוואה למעבדים כלליים (CPU) או למאיצים גרפיים (GPU). ראשית, היכולת להגדיר מחדש את מבנה החומרה מאפשרת התאמה מלאה של הארכיטקטורה לאלגוריתם הספציפי, ובכך מתקבלת יעילות גבוהה יותר לעומת ארכיטקטורות קבועות מראש של CPU או GPU [8]. גמישות זו מאפשרת יישום מותאם משימה, תוך שיפור ביצועים והפחתת זמני השהיה.

יתרון מרכזי נוסף הוא עיבוד מקבילי נרחב: בעוד שמעבד כללי מבצע את החישוב באופן סדרתי, רכיב FPGA מאפשר לפרוס את החישובים במקביליות גבוהה ולבצע פעולות מכפלה חיבור (MAC) רבות בו-זמנית. יכולת זו הודגמה במספר עבודות מחקר, בהן נמצא כי מימושי FPGA מצליחים להאיץ באופן משמעותי את תהליכי ההסקה ברשתות עצביות בהשוואה למימושים רכים [1], [3].

בנוסף, נמצא כי ל-FPGA יתרון בצריכת הספק נמוכה. שימוש בגישות כמו קוונטיזציה וייצוג Fixed-Point מאפשר להקטין את רוחב המילה של המשקלים והאקטיבציות, ובכך להפחית את הדרישה למשאבי DSP וזיכרון פנימי - תוך שמירה על דיוק חישובי מספק [5], [6]. מחקרים מצביעים כי גישה זו הופכת את ה-FPGA למתאים במיוחד ליישומי זמן אמת במערכות משובצות, שבהן מגבלת ההספק היא קריטית.

יתרון נוסף הוא גמישות תכנון ופיתוח: בעזרת כלים מודרניים כמו HLS או OpenCL ניתן לקצר משמעותית את זמני הפיתוח ולגשר בין רמת התוכנה לרמת החומרה. עם זאת, מחקרים מראים כי שימוש ב-RTL מותאם אישית עדיין מספק ניצול מיטבי של משאבי הרכיב [2], [4]. גישה זו מאפשרת איזון בין פשטות הפיתוח לבין מקסום הביצועים בפועל.

לסיכום, מימוש אלגוריתמי למידת מכונה על גבי FPGA מעניק שילוב ייחודי של ביצועים גבוהים, צריכת הספק נמוכה, וגמישות הנדסית, מה שהופך אותו לפתרון אטרקטיבי במיוחד ליישומי זמן אמת הדורשים האצה חישובית לצד מגבלות משאבים.

# פרק 3 - מימוש הפרויקט

## 3.1 כלי תכן וסביבת הפיתוח

### 3.1.1 בחירת לוח פיתוח DE2-115

לצורך מימוש הפרויקט נבחר לוח הפיתוח DE2-115 של חברת Terasic, המוצג באיור 7. הלוח מבוסס על רכיב FPGA מדגם EP4CE115F29C7N מסדרת Cyclone® IV E של חברת Altera (כיום Intel).

לוח זה נבחר מאחר שהוא מספק שילוב של משאבי חומרה נרחבים לצד אפשרויות קלט/פלט מגוונות, אשר מתאימים לצרכים של מימוש רשת נוירונים מלאכותית (MLP) לצורך סיווג ספרות ממאגר MNIST. כפי שניתן לראות באיור 7, הלוח כולל ממשקי משתמש, זיכרונות חיצוניים ורכיבי עזר התומכים בתהליך הפיתוח.

מאפייני רכיב FPGA:

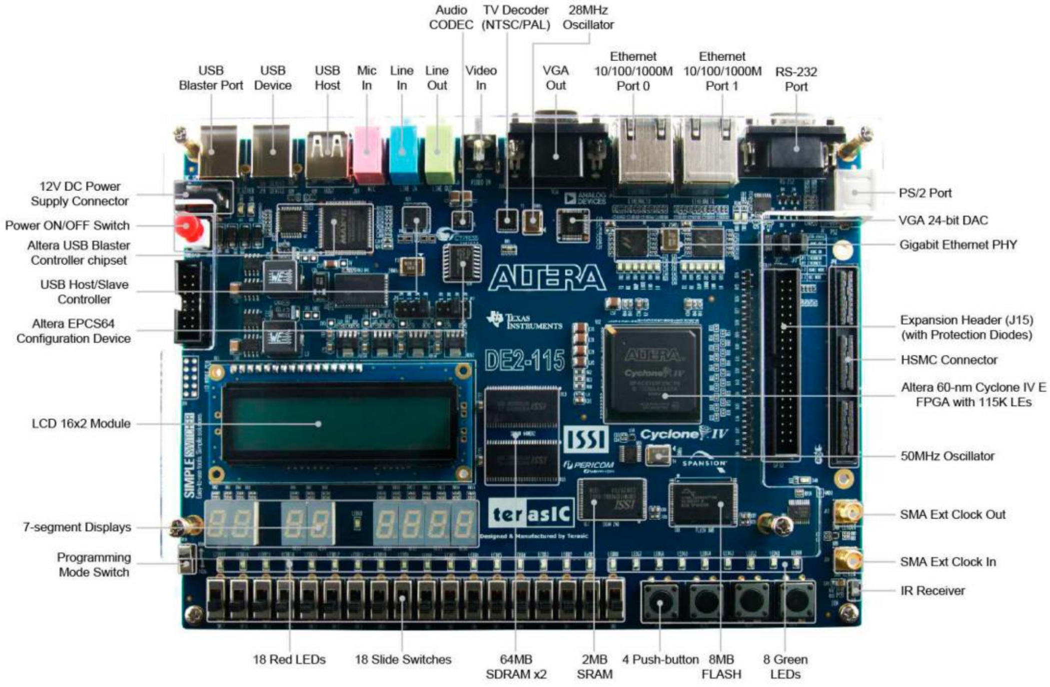
- 114,480 אלמנטים לוגיים (Logic Elements - LEs).
- זיכרון פנימי בנפח 3.888 Mbit.
- ארבעה רכיבי PLL פנימיים להכפלת תדרי שעון ולניהול סנכרון.
- 528 פורטי קלט/פלט (I/O pins) לממשק משתמש ולחיבורים חיצוניים.

רכיבי חומרה נוספים בלוח DE2-115:

- צורב מובנה לרכיב ה-(JTAG/USB Blaster) FPGA.
- זיכרונות חיצוניים מסוגים שונים: SRAM, Flash, EEPROM, ותמיכה בכרטיסי SD.
- ממשקי משתמש בסיסיים: 4 לחצנים, 18 מפסקים, 9 נורות ירוקות, 18 נורות אדומות ו-6 תצוגות שבעת-המקטעים (7 Segment Displays).
- רכיב קידוד/פענוח שמע עם יציאות קול ייעודיות.
- תצוגת LCD בגודל 16×2 תווים.
- ממשק HSMC לחיבור כרטיסי הרחבה נוספים.
- יציאת וידאו VGA עם ממיר DAC (4 ביט לכל צבע - RGB).
- יציאת RS232 לתקשורת UART עם ה-FPGA.
- חיבור PS/2 למקלדת/עכבר (תקנים ותיקים).
- יציאות USB היכולות לשמש גם כ-Host וגם כ-Device, מחוברות ישירות לרכיב ה-FPGA.

נימוק לבחירה:

השימוש ב-DE2-115 מאפשר עבודה בסביבת פיתוח מוכרת (Quartus, ModelSim), תוך ניצול משאבים לוגיים וזיכרון פנימי בהיקף רחב. בנוסף, מגוון הממשקים שעל הלוח מאפשר בחינה עתידית של הרחבות. בכך מתקבל איזון בין יכולות חומרה גבוהות לבין נגישות וכלי פיתוח נתמכים, המבטיחים מימוש יעיל וישים של שלב ההסקה על גבי FPGA.




*איור 7 - לוח הפיתוח DE2-115*

3.1.2 סביבת פיתוח חומרה

לצורך פיתוח, מימוש ובדיקת רכיבי החומרה בפרויקט נעשה שימוש בשני כלים מרכזיים:

Quartus - תוכנה זו משמשת כפלטפורמה העיקרית לתכנון ומימוש רכיבי FPGA באמצעות שפות תיאור חומרה (HDL). Quartus מאפשרת תהליך מלא של סינתזה, אופטימיזציה וצריבה של התכן אל רכיב ה-FPGA עצמו. בין יכולותיה העיקריות: הפקת שרטוטי RTL לצורך ניתוח מבנה פנימי, יצירת דיאגרמות של מכונות מצבים (FSM), וניהול תהליך הצריבה בלוחות פיתוח תואמים כדוגמת DE2-115. כלי זה מהווה את הליבה של זרימת העבודה, החל מכתיבת קוד ה-VHDL ועד לאימותו על גבי החומרה.

ModelSim - תוכנה זו נועדה לביצוע סימולציות פונקציונליות ולוגיות לקוד ה-HDL שנכתב בפרויקט. ModelSim מאפשרת הדמיה ברמת אותות, בדיקה של תגובת המערכת לקלטים שונים, וכן מדידת זמני השהיה ובחינת עמידה בדרישות תזמון. בנוסף, היא מאפשרת צפייה באותות דיגיטליים ואנלוגיים במגוון פורמטים וקידודים, לרבות Waveforms מפורטים. שילוב הסימולציה ב-ModelSim טרם מימוש החומרה בפועל מצמצם משמעותית את הסיכון לשגיאות תכן ומבטיח תהליך פיתוח אמין ויעיל.

3.1.3 סביבת אימון ותוכנה

בנוסף לכלי הפיתוח החומרתיים, נעשה בפרויקט שימוש בסביבת תוכנה ייעודית לאימון ולבחינת האלגוריתם טרם המרתו ל-FPGA. לשם כך נבחרה שפת Python, הנחשבת לשפה המרכזית בתחום ה-Machine Learning, יחד עם ספריות ייעודיות כגון PyTorch ו-NumPy.

האימון של המודל בוצע בסביבת Google Colab, המאפשרת הרצת קוד Python על גבי שרתי ענן עם תמיכה ב-CPU, GPU ואף TPU. סביבת Colab נבחרה בזכות נגישותה, היכולת להאיץ חישובים בעזרת חומרה מתקדמת, וכן בזכות הממשק הידידותי בסגנון Notebook התומך בשילוב קוד, תוצאות וגרפים בצורה אינטראקטיבית.

באמצעות שילוב זה של Python ו-Colab בוצעו שלבים מרכזיים בתהליך הפיתוח: טעינת ועיבוד מאגר הנתונים MNIST, בניית ארכיטקטורת רשת Fully Connected Neural Network, ביצוע שלבי Training ו-Validation תוך תיעוד תוצאות Accuracy ו-Loss, ולבסוף הפקת גרפים לאנליזה של תהליך האימון.

## 3.2 בחירת האלגוריתם ומימושו

### 3.2.1 בחירת האלגוריתם

בפרויקט זה נבחר מודל של רשת נוירונים מלאכותית מסוג Fully Connected Neural Network לצורך סיווג ספרות ידניות ממאגר הנתונים MNIST.

בחירה זו נבעה משיקולים הנדסיים ומתודולוגיים כאחד:

- פשטות יחסית - רשתות Fully Connected נחשבות לארכיטקטורה בסיסית בעולם למידת המכונה, והן מתאימות במיוחד לבעיות סיווג קלאסיות. הדבר מאפשר תהליך מימוש ישיר יותר בהשוואה לרשתות מורכבות כגון CNN, הדורשות חישובי קונבולוציה עתירי משאבים.
- התאמה לבעיה הנבחנת - מסד הנתונים MNIST הכולל ספרות ידניות בגודל 28×28 פיקסלים, מהווה דוגמה מובהקת לבעיה של Classification(סיווג).
רשת Fully Connected מספקת מענה מספק לבעיה זו ברמת דיוק גבוהה, תוך שמירה על מורכבות חישובית נמוכה יחסית.
- ישימות חומרתית - מבנה הרשת מתבסס על חישובי מכפלה חיבור (MultiplyAccumulate), אותם ניתן ליישם בצורה טבעית על גבי FPGA באמצעות יחידות MAC ייעודיות. בכך מתקבלת הזדמנות להציג את יתרונות החומרה המקבילית של FPGA לעומת מימוש סדרתי במעבד CPU.

הרשת שנבחרה בפרויקט כוללת:

- שכבת קלט בגודל 784 נוירונים (התאמה ל-28×28 פיקסלים).
- שכבת Fully Connected יחידה המקשרת ישירות בין הקלט לפלט.
- שכבת פלט הכוללת 10 נוירונים, המייצגים את ההסתברות לשיוך התמונה לכל אחת מהספרות 0-9. במימוש בפועל מתקבלים ערכי Logits, אשר מומרים להסתברויות על ידי פונקציית ה-Loss.

בשלב האימון, שבוצע בסביבת Python (PyTorch) על גבי Google Colab, נלמדו המשקלים וההטיות (Weights & Biases) של הרשת על בסיס עשרות אלפי דוגמאות ממאגר MNIST . תהליך זה אפשר בניית מודל המסוגל לבצע Inference(הסקה) מדויק על דוגמאות חדשות שלא נראו קודם.

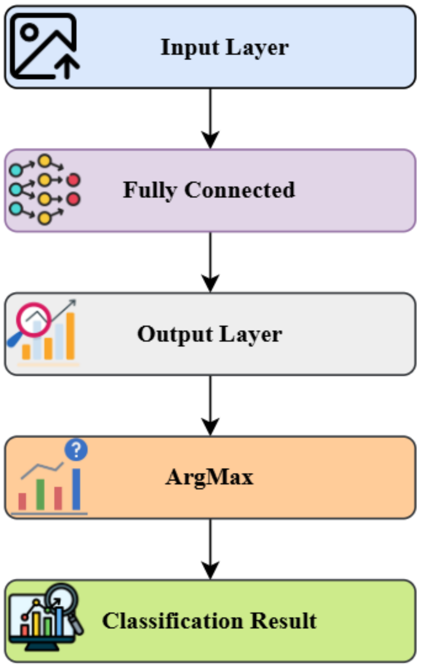
הבחירה בארכיטקטורה זו מייצרת איזון מיטבי בין פשטות מימוש חומרתי ,יעילות חישובית , ויכולת הדגמה ברורה של יתרונות מימוש FPGA לעומת CPU.

### 3.2.2 אופן פעולת מודל רשת הנוירונים

תהליך הפעולה של מודל רשת הנוירונים המופיע באיור 8 מתבצע באופן הבא:

1. קלט (Input Layer) - תמונת ספרה בגודל 28×28 פיקסלים מתוך מאגר MNIST מומרת לוקטור חד-ממדי באורך 784 רכיבים. כל רכיב מייצג את ערך הפיקסל המתאים (בגווני אפור).
2. שכבת Fully Connected - הוקטור מוזן לשכבת נוירונים Fully Connected. בכל נוירון מתבצע חישוב של סכום משוקלל של הקלטים עם משקלים וההטיות, פלט השכבה הוא וקטור בגודל 10, שבו כל רכיב מייצג את הנטייה של המודל לשייך את התמונה למחלקה מסוימת.
3. שכבת פלט (Output Layer) - הפלט המתקבל הוא וקטור logits באורך 10 (ערכים לא מנורמלים), שבו כל ערך מתאים לאחת מהספרות האפשריות 0–9.
4. שלב ArgMax - על וקטור ה-logits מופעלת פונקציית ArgMax, הבוחרת את האינדקס בעל הערך הגבוה ביותר. אינדקס זה מייצג את הספרה שהמערכת מסווגת כתשובה הסופית. שלב זה אינו מהווה שכבה נוירונית אלא שלב חישובי פשוט לאחר הפלט.

באופן זה מתקבל תהליך מלא של מיפוי תמונה מספרית לקלאס דיגיטלי, באופן המתאים לבעיית סיווג הספרות ממאגר MNIST.




*איור 8 - תרשים זרימה של מודל רשת הנוירונים*

### 3.2.3 מימוש מודל רשת הנוירונים בסביבת תוכנה

בשלב הראשון בוצע מימוש של מודל הרשת בסביבת Python (PyTorch), על גבי פלטפורמת Google Colab. מטרת שלב זה הייתה לבחון את הארכיטקטורה הנבחרת, לאמן את המודל על בסיס הנתונים MNIST, ולהשיג תוצאות שישמשו בסיס להמשך המימוש בחומרה.

שלבי העבודה:

- עיבוד מקדים לנתונים - כל תמונה בגודל 28×28 פיקסלים הומרה לוקטור חד-ממדי באורך 784, ולאחר מכן בוצע נרמול לערכים בטווח סטנדרטי (באמצעות ממוצע וסטיית תקן של מאגר MNIST), כדי לשפר את יציבות האימון.

- חלוקת הדאטהסט - מאגר הנתונים MNIST כולל 70,000 תמונות ספרות ידניות בגודל 28×28פיקסלים (בגווני אפור) ,
  המחולקות לשתי קבוצות עיקריות:
  - 60,000 דוגמאות - מיועדות לאימון המודל (Training).
  - 10,000 דוגמאות - מיועדות מראש לבדיקת המודל (Test).

  לצורך האימון בפרויקט זה בוצעה חלוקה פנימית של 60,000 דוגמאות האימון:
  - 55,000 דוגמאות - קבוצת אימון (Training set).
  - 5,000 דוגמאות - קבוצת ולידציה (Validation set), ששימשה לניטור ביצועי המודל ולזיהוי Overfitting.

  קבוצת ה-Test (10,000 תמונות נפרדות) נשמרה בצד, ולא נעשה בה שימוש במהלך האימון או כוונון ההיפר פרמטרים. היא שימשה אך ורק להערכת ההכללה הסופית של המודל.
  חלוקה זו מבטיחה תהליך פיתוח הוגן, שבו ולידציה מנחה את בחירת ההגדרות, בעוד Test מייצג מדד אובייקטיבי לביצועים אמיתיים.

- בניית המודל המופיע באיור 9 - הוגדרה רשת Fully Connected פשוטה: שכבת קלט בגודל 784, שכבת Fully Connected יחידה המקשרת את הקלט ישירות לפלט, ושכבת פלט עם 10 נוירונים (Logits) המייצגים את עשרת המחלקות (הספרות 0-9). ההחלטה הסופית התקבלה באמצעות שלב ArgMax.

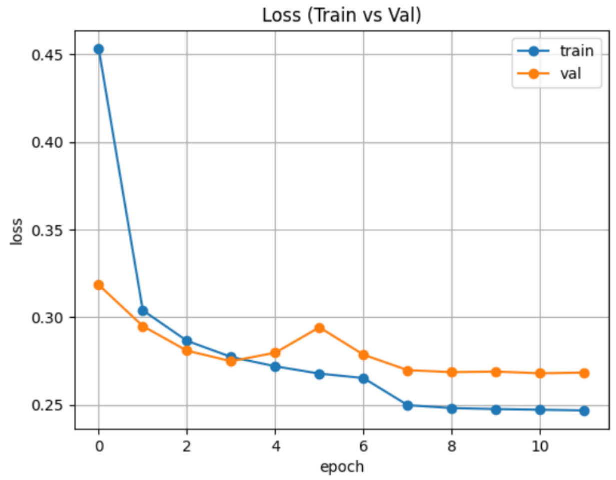
```python
# Step 3: model (linear 784->10)
class SimpleFC(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 10)  # single layer with bias

    def forward(self, x):
        x = x.view(x.size(0), -1)        # flatten
        return self.fc1(x)               # logits

model = SimpleFC().to(device)
```

*איור 9 - מודל רשת הנוירונים*

- אימון המודל - בוצע אימון באמצעות אלגוריתם Adam (Learning Rate ראשוני 0.001, משקל דעיכה 1e-4) ובשימוש בפונקציית עלות Cross-Entropy Loss, הכוללת חישוב פנימי של SoftMax ומתאימה לבעיית סיווג. האימון התבצע על 55k תמונות, תוך שימוש ב-Validation Set של 5k דוגמאות.
בוצעו 12 Epochs ובכל אחד מהם נמדדו ערכי Loss ו- Accuracy עבור קבוצות האימון והולידציה.

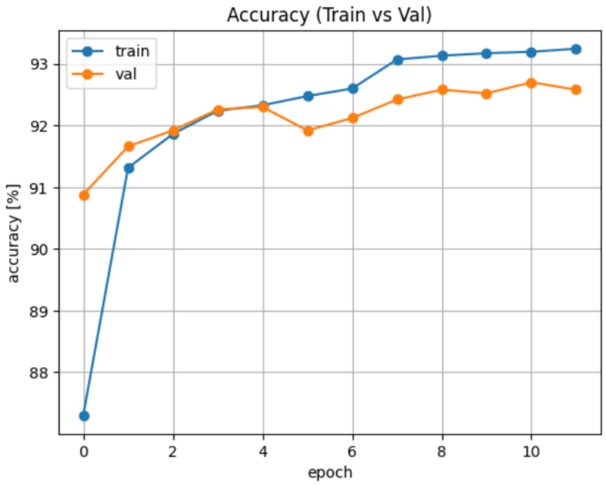
- בדיקות ולידציה - במקביל לאימון חושב ביצוע המודל על קבוצת הוולידציה, כדי לנתר Overfitting ולהתאים את ה-Learning Rate באמצעות ReduceLROnPlateau.

- תוצאות וגרפים - במהלך האימון נאספו נתונים והוצגו בגרפים:
  - גרף ירידת ה-Loss לאורך ה-epochs (Train מול Validation) המופיע באיור 10, ירידה יציבה ב-train_loss והתייצבות של val_loss, פער קטן בין העקומות מעיד על הכללה טובה והיעדר Overfitting חריג.




*איור 10 - גרף ירידת ה-Loss לאורך ה-epochs*

  - גרף עליית ה-Accuracy עד להתייצבות סביב 92-93% המופיע באיור 11,




*איור 11 - גרף עליית ה-Accuracy*

- הערכה סופית על קבוצת הבדיקה (Test) - לאחר סיום תהליך האימון והכיוונון, המודל הוערך על קבוצת ה-Test שכללה 10,000 דוגמאות שלא נעשה בהן שימוש קודם. התוצאות הראו דיוק סופי (Final Test Accuracy) של כ-92%-93% עם ערך Loss נמוך כפי שמופיע באיור 12, דבר המצביע על יכולת הכללה טובה של המודל מעבר לדוגמאות שראה באימון.

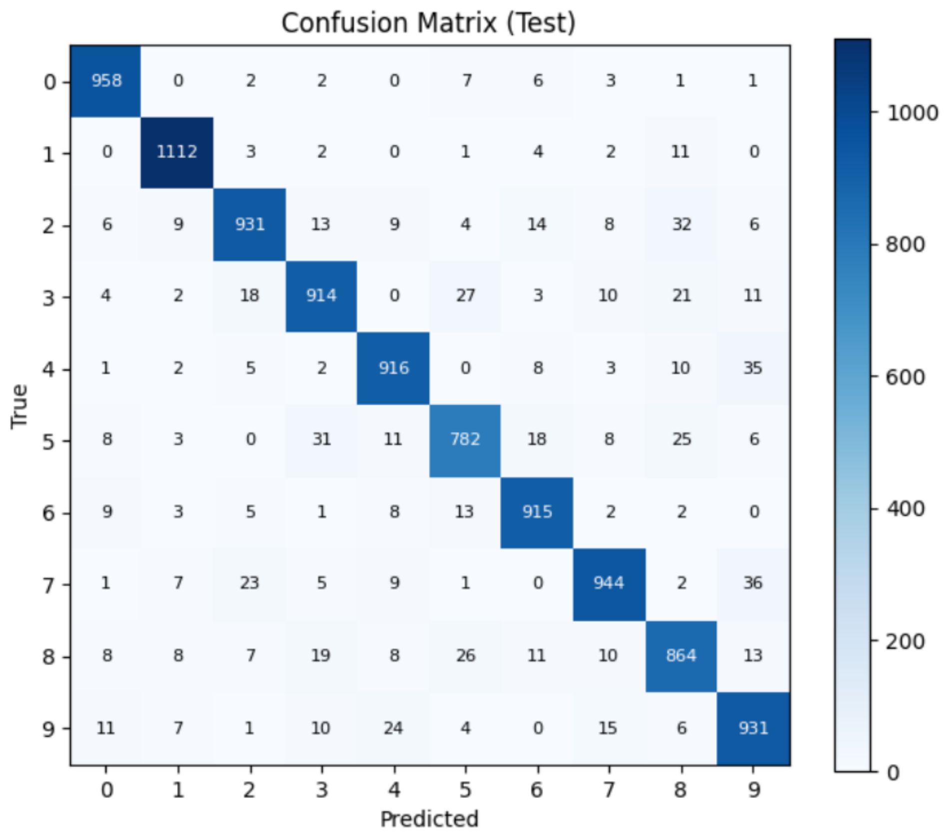
```
Final test accuracy: 92.67% (loss=0.2636)
```

*איור 12 - Final Test Accuracy*

בנוסף להצגת מדד הדיוק הכולל (Accuracy), נבנתה מטריצת בלבול (Confusion Matrix) המופיעה באיור 13 אשר מאפשרת ניתוח מפורט של טעויות הסיווג. מהמטריצה ניתן לראות כי ברוב המקרים הספרות זוהו בהצלחה גבוהה, אולם קיימת נטייה לבלבול בין ספרות בעלות דמיון צורני.

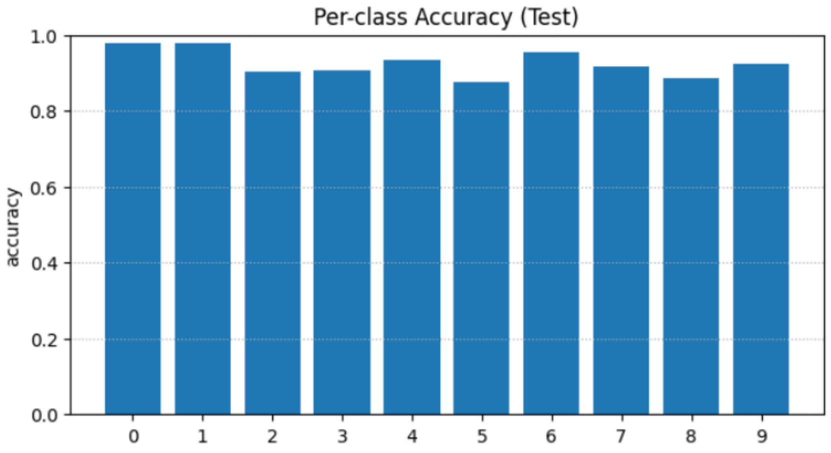
לדוגמה:

- הספרות 4 ו-9, שנוטות להופיע בצורה דומה בכתבי יד מסוימים.
- הספרות 3 ו-5, אשר במקרים מסוימים מציגות קווים עגולים דומים.

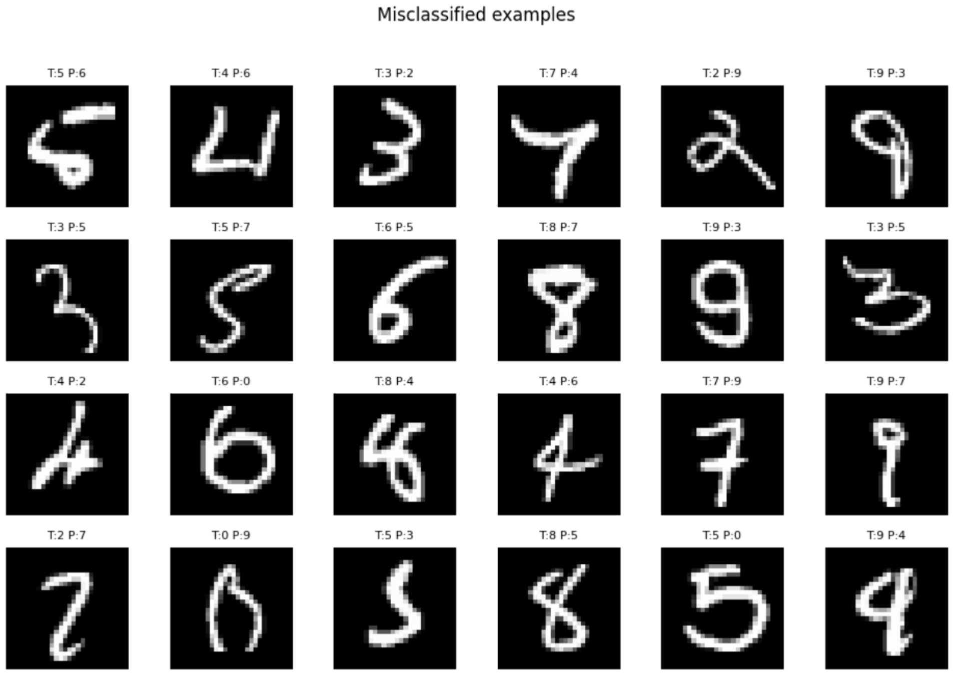



*איור 13 - Confusion Matrix*

חישוב דיוק לפי מחלקה (Per-class Accuracy) המופיע באיור 14 חיזק את הממצא הזה, כאשר רוב הספרות הוצגו בדיוק של מעל 90%, בעוד מחלקות מסוימות הציגו דיוק נמוך יותר באופן יחסי.




*איור 14 - דיאגרמת Per-class Accuracy*




*איור 15 - דוגמאות לתמונות*

ניתוח זה מספק תובנה חשובה: גם באמצעות מודל פשוט יחסית - רשת Fully Connected חד שכבתית - ניתן להגיע לרמת ביצועים גבוהה בסיווג ספרות ידניות. המודל שהוגדר בפרויקט השיג דיוק של כ-92%-93% על קבוצת ה-Test. עבור מודל חד שכבתי, תוצאה זו נחשבת לסטנדרט מקובל במחקר ובהוראה. תוצאה זו ממחישה כי גם ארכיטקטורה בסיסית, המבוססת על שכבת קלט, שכבה חבויה אחת ושכבת פלט, מסוגלת להתמודד בהצלחה עם בעיית סיווג סטנדרטית, ולספק בסיס איתן להמשך הרחבות ושיפורים עתידיים בארכיטקטורה או במימוש החומרתי.

### 3.2.4 הכנת מודל רשת הנוירונים למימוש חומרתי

לאחר תהליך האימון והערכת הביצועים, נשמר המודל המאומן לקובץ בפורמט pth. לצורך שימוש עתידי. שמירת המודל מאפשרת הטמעה חוזרת של הפרמטרים שנלמדו (Weights & Biases) מבלי צורך לבצע אימון מחדש. בשלב הבא בוצע תהליך Inference - הפעלת המודל המאומן על דוגמאות חדשות מקבוצת ה-Test. תהליך זה שימש לאימות היכולת של המודל להתמודד עם נתונים שלא נצפו במהלך האימון, ומהווה שלב מעבר חיוני ממודל מחקרי למודל ישים.

בהמשך, בוצע שלב נוסף שמטרתו להכין את הפרמטרים למימוש חומרתי על גבי FPGA.
שלב זה כלל:

- שליפת פרמטרים מהמודל המאומן - משקלי שכבת הפלט (fc1.weight) וההטיות (fc1.bias) הומרו למערכים של NumPy.
- ביטול פעולת הנרמול (Un-Normalization) - מאחר שהאימון בוצע על נתונים מנורמלים, בוצע "קיפול" (Folding) של פעולת הנרמול לתוך המשקלים וההטיות, כך שהם מותאמים לקלט המקורי של התמונות.
- כימות (Quantization) - המשקלים כומתו לפורמט int8 בטווח [-128,127] תוך שימוש בפקטור סקייל (Scale Factor), וההטיות כומתו לפורמט int32. כימות זה מאפשר מימוש יעיל ב-FPGA: רכיבי החומרה מותאמים באופן טבעי יותר לאריתמטיקה בפורמט Fixed-Point מאשר Floating-Point, ובכך מתקבלת חיסכון במשאבים ועלייה במקביליות החישובים.
- כתיבה לקבצי זיכרון (MIF - Memory Initialization File) - נוצרו שלושה קבצים:
  - xq_image.mif - מייצג תמונת קלט (784 פיקסלים, כל פיקסל ב-8 סיביות).
  - wq80.mif - מכיל את המשקלים (784×10), נארזים כ-80 ביט לכל פיקסל (10×8 ביט).
  - bstar320.mif - מכיל את ההטיות (10 הטיות, כל אחת 32 ביט, נארזות ל-320 ביט).

פירוט מבנה הקבצים:

xq_image.mif - קלט (Input Image):

- מייצג תמונת ספרה אחת בגודל 28×28 סה״כ 784 פיקסלים.
- כל פיקסל עובר כימות ל-8 סיביות (int8/u8) אחרי נרמול וסקייל.
- הקובץ מכיל 784 שורות, כשבכל שורה כתובת (addr) וערך הקוד ההקסדצימלי של הפיקסל.
- הפורמט:

```
WIDTH=8;
DEPTH=784;
ADDRESS_RADIX=UNS;
DATA_RADIX=HEX;
CONTENT BEGIN
0 : 3F;
1 : 12;
...
783 : 7A;
END;
```

wq80.mif - משקלים (Weights):

- המטריצה המקורית של המשקלים היא בגודל 10×784 (10 מחלקות × 784 פיקסלים).
- לאחר כימות, כל משקל מיוצג ב-8 ביט (int8).
- לצורך ייעול הקריאה בחומרה, נארזים כל 10 המשקלים המתאימים לפיקסל נתון יחדיו למילה אחת ברוחב 80 ביט, באופן שבו מיקום כל 8 סיביות במילה מוגדר מראש למחלקה מסוימת:
  - [7:0] - המשקל של מחלקה 0
  - [15:8] - המשקל של מחלקה 1
  - ...
  - [79:72] - המשקל של מחלקה 9
- הקובץ מכיל 784 שורות (עמודה אחת לכל פיקסל), בכל שורה מילה אחת של 80 ביט (מיוצגת כ-20 תווים הקסדצימליים).

- הפורמט:

```
WIDTH=80;
DEPTH=784;
ADDRESS_RADIX=UNS;
DATA_RADIX=HEX;
CONTENT BEGIN
0 : 00FF80...;
...
783 : 1122AA...;
END;
```

bstar320.mif - הטיות (Biases):

- לכל אחת מ-10 המחלקות יש הטיה (Bias) ב-32 סיביות (int32).
- כל ההטיות נארזות יחד למילה אחת של 320 ביט, כך שכל 32 סיביות במילה מייצגות את ההטיה של מחלקה אחת:
  - [31:0] - Bias למחלקה 0
  - [63:32] - Bias למחלקה 1
  - ...
  - [319:288] - Bias למחלקה 9
- עומק הקובץ הוא 1 בלבד, שכן מדובר בסט יחיד של 10 הטיות.
- הפורמט:

```
WIDTH=320;
DEPTH=1;
ADDRESS_RADIX=UNS;
DATA_RADIX=HEX;
CONTENT BEGIN
00 : 0AABBCC...FFEEDD;
END;
```

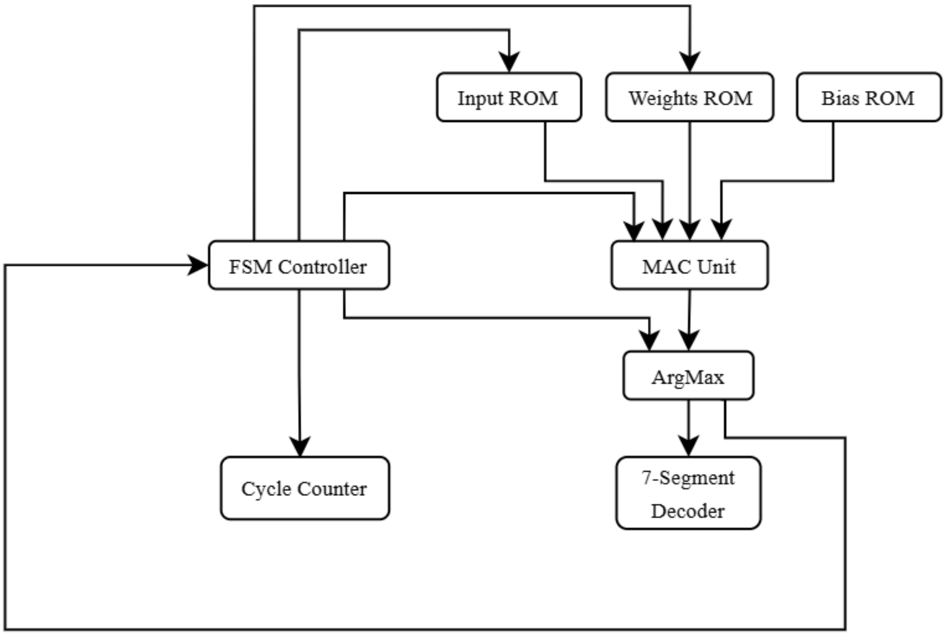
### 3.2.5 מימוש מודל רשת הנוירונים בסביבת חומרה

בשלב זה בוצע מימוש חומרתי של שלב ההסקה (Inference) על גבי רכיב ה-FPGA. המימוש התבסס על עקרונות הפעולה של המודל המאומן, תוך התאמתו לייצוגים בדידים (Fixed-Point) ולמבנה מקבילי האופייני ל-FPGA.

התכנון בוצע במבנה מודולרי, כאשר כל רכיב פותח כמנוע עצמאי, נבדק בסימולציה (ModelSim) ולאחר מכן שולב במערכת המלאה. גישה זו אפשרה לוודא את נכונות הפעולה של כל בלוק עוד לפני אינטגרציה מלאה, ולפשט את תהליך הדיבוג.

באיור 16 מוצג תרשים בלוקים סכמתי של הארכיטקטורה הכללית של המערכת, המתאר את רכיבי הליבה ואת הקשרים ביניהם. בהמשך מפורטים הרכיבים השונים:

- רכיבי זיכרון (ROM/BRAM) - שלושת קבצי ה-MIF (קלט, משקלים, הטיות) נטענו לזיכרון פנימי ומספקים נתוני קלט לחישובים.
- רכיב MAC - יחידות Multiply-Accumulate למימוש החישוב המשוקלל עבור כל נוירון מחלקה.
- רכיב FSM - מכונת מצבים לניהול רצף הפעולות: קריאת קלט, חישוב MAC, הוספת Bias,בחירת פלט.
- רכיב ArgMax - בלוק לבחירת המחלקה בעלת הערך הגבוה ביותר מתוך עשרת הנוירונים בשכבת הפלט .
- רכיב Counter - רכיב לספירת מחזורי שעון וזמן הביצוע הכולל של תהליך ההסקה.
- רכיב Decoder ל-7-Segment - ממיר את ערך הפלט הסופי (0-9) לקוד תואם תצוגת 7 - Segment ומאפשר הצגת התוצאה בזמן אמת על לוח ה-FPGA.




*איור 16 - תרשים בלוקים של הארכיטקטורה הכללית למימוש החומרתי*

### 3.2.5.1 חבילות (Packages) VHDL

במערכת הוגדרו שתי חבילות VHDL ייעודיות, שמטרתן לארגן את ההגדרות והפונקציות המשותפות לכלל רכיבי המימוש :

- חבילת types_pkg המופיעה באיור 17- מרכזת את ההגדרות הבסיסיות :
  - קבועים (כגון מספר המחלקות וגודל הקלט).
  - תתי טיפוסים (Unsigned 8-bit עבור פיקסל, Signed 8-bit עבור משקל, Signed 32-bit עבור Accumulator).
  - טיפוסי וקטורים (וקטור משקלים באורך 10, וקטור Accumulator באורך 10).

```
package types_pkg is
constant CLASSES : integer := 10;
constant PIXELS : integer := 784; -- 28*28

subtype x_t is unsigned(7 downto 0); -- input pixel: 0..255
subtype w_t is signed(7 downto 0); -- weight: -128..127 (int8)
subtype acc_t is signed(31 downto 0); -- accumulator (int32)

type w_vec_t is array (0 to CLASSES-1) of w_t;
type acc_vec_t is array (0 to CLASSES-1) of acc_t;

end package;

package body types_pkg is end package body;
```

*איור 17 - קוד חבילת types_pkg בשפתVHDL*

- חבילת pack_utils המופיעה באיור 18- מרכזת פונקציות עזר (Utilities) לניתוח ופרשנות של נתוני הזיכרון :
  - פונקציה slv80_to_wvec - ממירה מילה אחת של 80 ביט (8×10) לוקטור משקלים שלם.
  - פונקציה slv320_to_accvec - ממירה מילה אחת של 320 ביט (32×10) לוקטור ההטיות.

פונקציות אלו מפשטות את הקוד ברכיבי המימוש, בכך שכל רכיב ניגש ישירות לערכים מבלי לעסוק בפירוק מחרוזות ביטים.

```
package pack_utils is
  function slv80_to_wvec(s : std_logic_vector(79 downto 0)) return w_vec_t;
  function slv320_to_accvec(s : std_logic_vector(319 downto 0)) return acc_vec_t;
end;

package body pack_utils is
  -- [8*i+7 : 8*i] = weight of class i, i=0..9
  function slv80_to_wvec(s : std_logic_vector(79 downto 0)) return w_vec_t is
    variable r : w_vec_t;
  begin
    for i in 0 to 9 loop
      r(i) := signed(s(8*i+7 downto 8*i));
    end loop;
    return r;
  end;

  -- [32*i+31 : 32*i] = bias of class i, i=0..9
  function slv320_to_accvec(s : std_logic_vector(319 downto 0)) return acc_vec_t is
    variable r : acc_vec_t;
  begin
    for i in 0 to 9 loop
      r(i) := signed(s(32*i+31 downto 32*i));
    end loop;
    return r;
  end;
end;
```

*איור 18- קוד חבילת pack_utils בשפת VHDL*

יתרונות השימוש בחבילות:

- מודולריות ואחידות: כל הרכיבים במערכת משתמשים באותם טיפוסים והגדרות.
- פשטות תחזוקה: שינוי ברוחב ייצוג או במספר המחלקות דורש עדכון במקום אחד בלבד.
- קוד קריא ומסודר: הפונקציות מוסתרות בחבילה, והרכיבים עצמם נראים "נקיים" ופשוטים יותר.

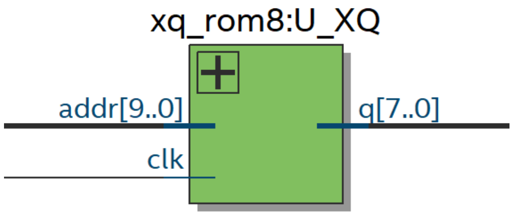
שילוב החבילות בארכיטקטורה:

החבילות אינן רכיבי חומרה פיזיים בפני עצמן, אלא משמשות כשכבת תשתית לוגית ("Library Layer") התומכת בכלל המערכת:

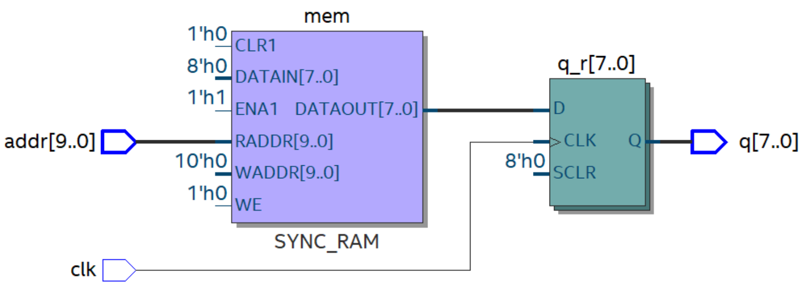
- types_pkg מבטיחה אחידות פורמטים לכל הרכיבים.
- pack_utils מאפשרת לרכיבים, בעיקר זיכרון ו-MAC, לפרש נתונים גולמיים מקבצי ה-MIF.

3.2.5.2 רכיב xq_rom8

שרטוט RTL :




איור 19- סכמת RTL חיצונית של רכיב xq_rom8




איור 20 - סכמת RTL פנימית של רכיב xq_rom8

מטרת הרכיב בארכיטקטורה :

הרכיב משמש לאחסון תמונת הקלט (784 פיקסלים בגודל 28×28). הפיקסלים מאוחסנים מראש בקובץ זיכרון מסוג xq_imag.mif ומסופקים לרכיב ה-MAC לפי הכתובת הנדרשת בכל מחזור שעון.

כניסות ויציאות :

לרכיב 2 כניסות ויציאה 1.

הכניסות הן :

- clk - אות שעון, המשמש לקריאת נתונים סינכרונית.
- addr - כתובת (10 ביט, 0-783) המייצגת את אינדקס הפיקסל הרצוי.

היציאה היא :

q - יציאת נתון (8 ביט) המכילה את ערך הפיקסל שנקרא מהזיכרון.

לוגיקת הרכיב:

- הזיכרון מוגדר כסוג mem_t, מערך של 784 תאים, כל אחד ברוחב 8 ביט.
- ייחוס לאטריבוט ram_init_file מאפשר את טעינת התוכן מקובץ ה-xq_imag.mif בזמן סינתזה/צריבה.
- הקריאה מתבצעת באופן סינכרוני: בכל חזית עולה של השעון (rising_edge(clk)) הערך המתאים לכתובת הנוכחית (addr) נטען לרגיסטר פנימי (q_r).
- היציאה q מקבלת את הערך הרשום ברגיסטר, ולכן קיימת השהיה של מחזור שעון אחד (latency = 1).

במימושי FPGA, רכיבי זיכרון פנימיים (BRAM/ROM) יכולים להיות מאותחלים מראש באמצעות קבצי MIF. גישה זו מאפשרת להגדיר את תוכן הזיכרון כבר בשלב הסינתזה, כך שבזמן טעינת ה-bitstream אל ה-FPGA, בלוקי הזיכרון הפנימיים מאותחלים באופן אוטומטי.

במערכת זו, תמונת הקלט נטענת מראש מקובץ xq_imag.mif אל תוך זיכרון פנימי מסונכרן (Sync ROM). כתוצאה מכך, הרכיב מסוגל לספק לכל מחזור שעון את ערך הפיקסל המתאים לפי הכתובת שנשלחת אליו, מבלי להזדקק לתקשורת חיצונית או לבקרי זיכרון נוספים.

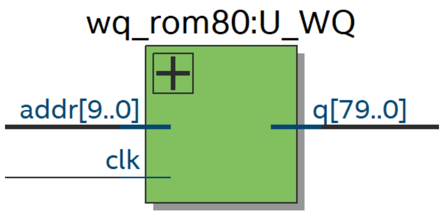
יתרונות שיטה זו:

- יעילות חומרתית - שימוש במשאבי BRAM ייעודיים של ה-FPGA במקום בלוגיקה כללית.
- נגישות מיידית - הנתונים זמינים כבר עם תחילת הפעולה של המערכת, ללא צורך בתהליך טעינה בזמן ריצה.
- קביעות (Determinism) - התוכן קבוע מראש, מה שמאפשר בדיקות חוזרות ונשנות בתנאים זהים.
- ביצועים - קריאת נתונים מתבצעת בקצב השעון של ה-FPGA, עם השהיה קבועה של מחזור אחד בלבד.

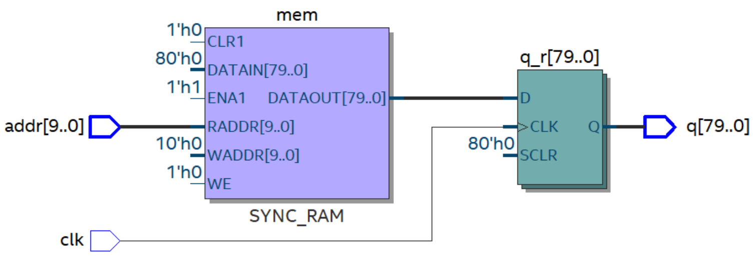
באופן זה, רכיב הזיכרון מהווה שכבת בסיס במערכת: הוא מספק את נתוני הקלט לרכיב ה-MAC בדיוק ובמהירות, תוך הסתמכות על התשתית הפנימית של ה- FPGA לטעינת נתונים מראש.

3.2.5.3 רכיב wq_rom80

שרטוט RTL:




*איור 21 - סכמת RTL חיצונית של רכיב wq_rom80*




*איור 22 - סכמת RTL פנימית של רכיב wq_rom80*

מטרת הרכיב בארכיטקטורה:

הרכיב משמש לאחסון מטריצת המשקלים של המודל המאומן. לכל אחד מ-784 הפיקסלים יש וקטור משקלים בגודל 10 (אחד לכל מחלקה), כאשר כל משקל מיוצג ב-8 ביט. כדי לייעל את הקריאה בחומרה, הוקטור כולו נארז למילה אחת ברוחב 80 ביט (8×10). הרכיב מספק בכל מחזור שעון את הוקטור המתאים לכתובת הנדרשת.

כניסות ויציאות:

לרכיב 2 כניסות ויציאה 1.

הכניסות הן :

- clk - אות שעון, המשמש לקריאת נתונים סינכרונית.
- addr - כתובת (10 ביט, 0-783) המייצגת את אינדקס הפיקסל.

היציאה היא :

- q - יציאה (80 ביט) המכילה את כל עשרת המשקלים של הפיקסל הנבחר, באריזה צפופה ([w0|...|w9]).

לוגיקת הרכיב:

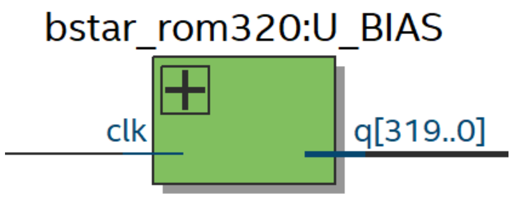
- הזיכרון מוגדר כסוג mem_t, מערך של 784 תאים, כל אחד ברוחב 80 ביט.
- יחוס לאטריבוט ram_init_file מאפשר את טעינת התוכן מקובץ ה-wq80.mif בזמן סינתזה/צריבה.
- קריאה מתבצעת באופן סינכרוני: בכל חזית עולה של השעון (rising_edge(clk)) הערך המתאים לכתובת הנוכחית (addr) נטען לרגיסטר פנימי (q_r).
- היציאה q מקבלת את הערך הרשום ברגיסטר, ולכן קיימת השהיה של מחזור שעון אחד (latency = 1).

טעינת זיכרון משקלים:

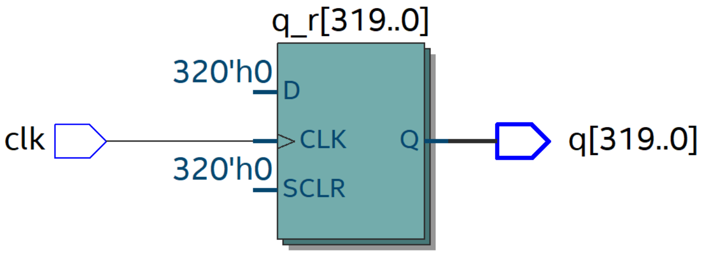
השימוש בזיכרון פנימי (BRAM/ROM) מאפשר לאחסן מראש את כל המשקלים הכמותיים של המודל המאומן. מבנה האריזה (10×8 ביט → 80 ביט) נבחר כדי להתאים למימוש מקבילי: ברגע קריאת פיקסל, כל המשקלים הדרושים לעשר המחלקות זמינים בו-זמנית לרכיב ה-MAC. בכך נחסך הצורך בגישה סדרתית או במבנה בקרה מורכב, והמערכת משיגה יעילות גבוהה בזמן ריצה.

3.2.5.4 רכיב bstar_rom320

שרטוט RTL:




*איור 23 - סכמת RTL חיצונית של רכיב bstar_rom320*




*איור 24 - סכמת RTL פנימית של רכיב bstar_rom320*

מטרת הרכיב בארכיטקטורה:

הרכיב משמש לאחסון ההטיות (Biases) של המודל המאומן. קיימת הטיה אחת לכל מחלקה (סה"כ 10 מחלקות), כאשר כל הטיה מיוצגת ב-32 ביט. כל ההטיות נארזות יחד למילה אחת ברוחב 320 ביט ונשמרות בקובץ זיכרון bstar320.mif. הקריאה מתבצעת פעם אחת בלבד, והוקטור המלא של ההטיות מסופק במקביל לרכיב ה- MAC לצורך השלמת חישוב הפלט.

כניסות ויציאות:

לרכיב  כניסה 1 ויציאה 1.

הכניסה היא:

- clk - אות שעון, המשמש לקריאה סינכרונית.

היציאה היא :

- q - יציאה (320 ביט) המכילה את כלל ההטיות (10×32 ביט), מסודרות באריזה אחת.

לוגיקת הרכיב:

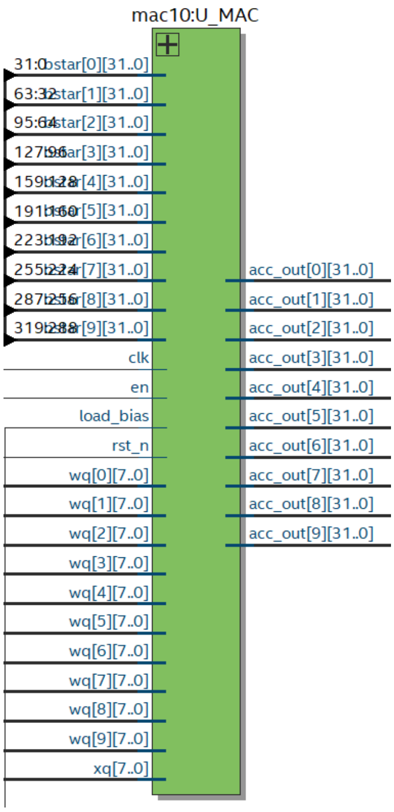
- הזיכרון מוגדר כמערך בגודל 1 (mem_t), מכיוון שכל ההטיות מאוחסנות במילה בודדת ברוחב 320 ביט.
- שימוש באטריבוט ram_init_file מאפשר טעינה אוטומטית של התוכן מקובץ ה-bstar320.mif בזמן סינתזה/צריבה.
- קריאה מתבצעת באופן סינכרוני: בכל חזית עולה של השעון (rising_edge(clk)), הערך מאוחסן ברגיסטר פנימי (q_r) ומוזרם ליציאה q.
- היציאה q מקבלת את הערך הרשום ברגיסטר, ולכן קיימת השהיה של מחזור שעון אחד (latency = 1).

טעינת זיכרון Biases:

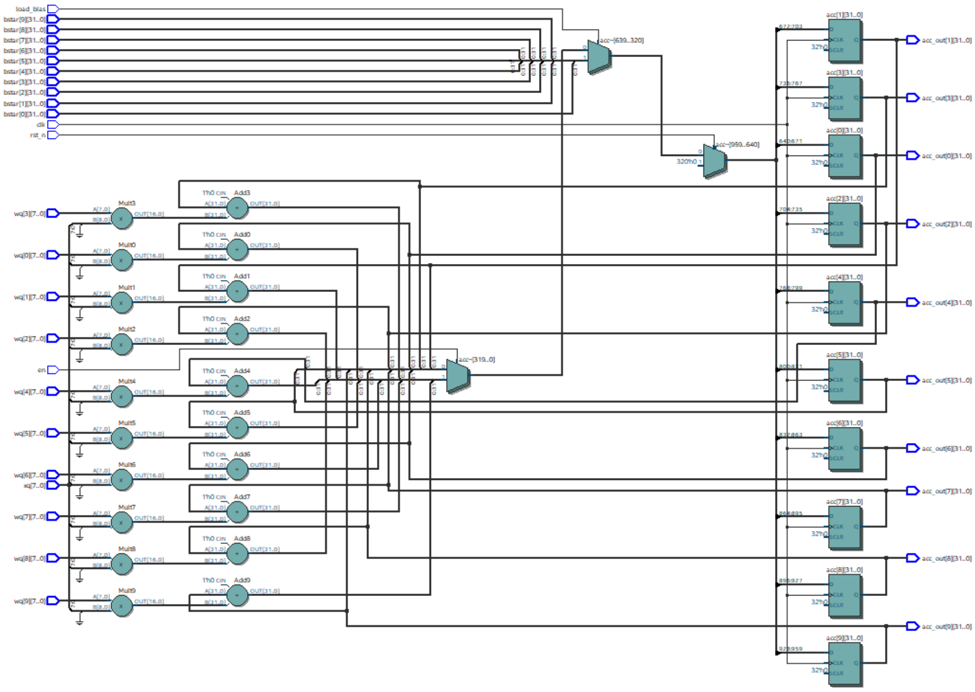
באופן זה, כל ההטיות זמינות בו-זמנית ל-10 המחלקות השונות. זה מאפשר הוספה מקבילית של ה-Bias לכל וקטור תוצאה שהצטבר ברכיב ה-MAC. הבחירה באריזה של כל ההטיות למילה אחת ברוחב 320 ביט משקפת את הגישה המקבילית של ה-FPGA ומפשטת את ממשק הקריאה מהזיכרון.

### 3.2.5.5 רכיב mac10

שרטוט RTL:




*איור 25 - סכמת RTL חיצונית של רכיב mac10*



*איור 26 - סכמת RTL פנימית של רכיב mac10*

מטרת הרכיב בארכיטקטורה:

הרכיב מהווה את הליבה החישובית של המערכת, ובאמצעותו מתבצע חישוב סכום משוקלל של הפיקסלים עם המשקלים, כולל הטיות.

הרכיב אחראי לבצע את חישובי המכפלה חיבור (MAC) עבור כל עשרת הנוירונים בשכבת הפלט של הרשת.
בכל מחזור שעון מוזן פיקסל יחיד (xq), מוכפל בעשרת המשקלים המתאימים לו (wq), ותוצאת ההכפלה מצטברת לאוגרי הביניים (Accumulators).
בנוסף, בתחילת התהליך ניתן לטעון את ערכי ההטיות (bstar) ישירות לאוגרים.

כניסות ויציאות:

לרכיב 7 כניסות ויציאה 1.

הכניסות הן:

- clk - אות שעון, המשמש לקריאת נתונים סינכרונית.
- rst_n - איפוס סינכרוני פעיל נמוך.
- load_bias - אות בקרה המורה לטעון את ערכי ההטיות לאוגרים.
- en - אות בקרה המפעיל חישוב MAC עבור פיקסל בודד.
- xq - ערך הפיקסל הנוכחי (8 ביט).
- wq - וקטור של 10 משקלים תואמים לפיקסל (10×8 ביט).
- bstar - וקטור ההטיות (10×32 ביט).

היציאות הן:

- acc_out - וקטור של 10 אוגרים (Accumulators), כל אחד ברוחב 32 ביט, המכילים את הסכום המצטבר.

לוגיקת הרכיב:

- בעת איפוס (rst_n= 0), כל האוגרים מאותחלים ל-0.
- כאשר load_bias=1, נטענים ערכי ההטיות (bstar) ישירות אל האוגרים.
- כאשר en=1:
  - ערך הפיקסל (xq) מומר ל-16 ביט signed.
  - כל משקל wq(i) מומר ל-16 ביט signed.
  - מבוצעת מכפלה (xq * wq(i)) המורחבת ל-32 ביט.
  - התוצאה מצטברת לערך הקודם באוגר acc(i).

- כל פעולת MAC מתבצעת במחזור שעון יחיד. בסיום 784 מחזורים (מספר הפיקסלים בתמונה), מתקבל בוקטור האוגרים (acc_out) הסכום המשוקלל הסופי עבור כל אחת מעשר המחלקות.

משמעות חישובית:

לאחר 784 מחזורי שעון, מתקבל וקטור Logits ברוחב 10, כאשר כל תא מייצג את "ציון הסיווג" עבור מחלקה מסוימת. שלב זה משלים למעשה את חישוב שכבת הפלט של רשת הנוירונים במימוש חומרתי.

יתרונות המימוש:

- מקביליות: כל 10 המכפלות מתבצעות באותו מחזור שעון, מה שמנצל היטב את היכולות המקביליות של ה-FPGA.
- מודולריות: רכיב עצמאי, ניתן לשלב אותו בארכיטקטורות אחרות של MLP.
- יעילות: ייצוג Fixed-Point (int8 עבור משקלים, int32 עבור סכימה) חוסך משאבים לעומת Floating-Point.

אימות וסימולציה:

מטרה:

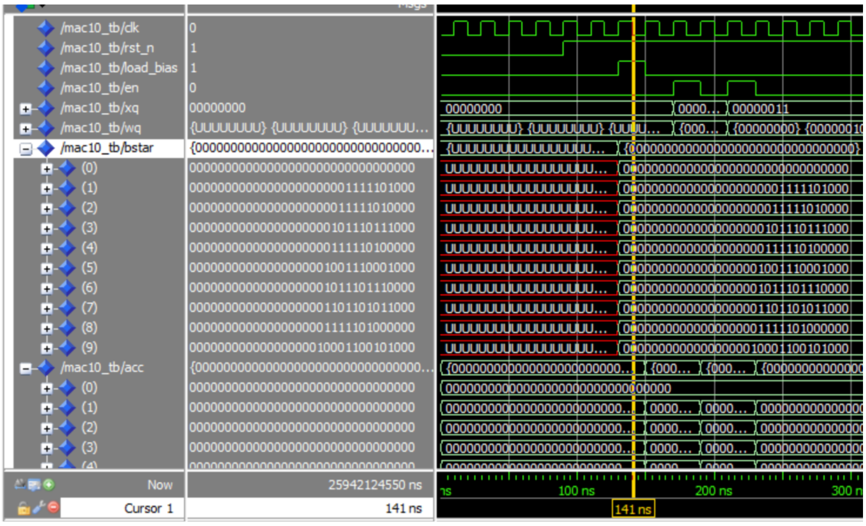
לאמת את פעולת רכיב ה-MAC10 הכולל 10 נתיבי חישוב מקבילים, ולוודא נכונות הצטברות הערכים תחת תרחישים שונים.

תרחישי בדיקה:

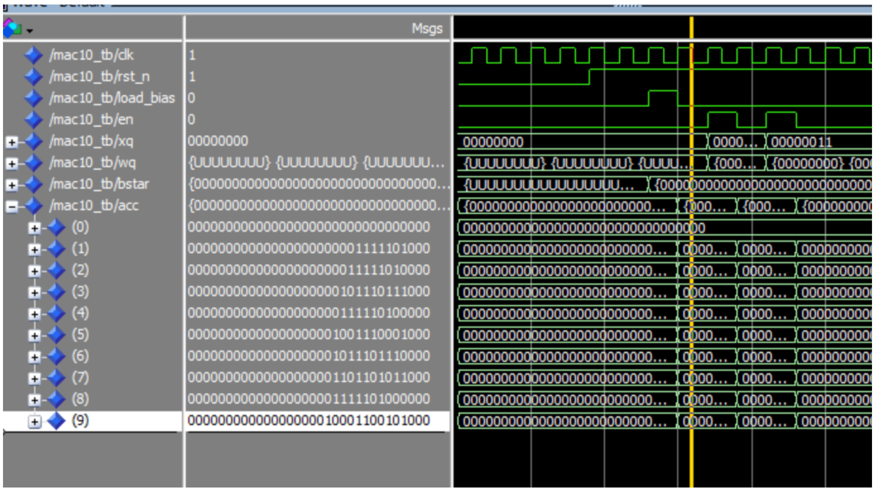
- איפוס - בדיקה שכל האוגרים מאותחלים ל-0.
- טעינת Bias - בדיקה שכל ערכי ההטיה נטענים לאוגרים במחזור בודד .
- חישוב מצטבר - בדיקה שפיקסל מוכפל בכל אחד מהמשקלים, והתוצאה מצטברת נכון לאוגרים.
- תנאי קצה - ערכים קיצוניים (xq=0, xq=255, משקל ±127) כדי לוודא פעולה תקינה ללא חריגות.

בכל התרחישים הפלטים תאמו במדויק לערכים הצפויים. נצפה עיכוב של מחזור אחד (latency=1), והצטברות הערכים התבצעה כמצופה לכל 10 הנתיבים במקביל.

בסימולציה ניתן לראות כי בתחילה האותות מופיעים כבלתי מאותחלים (U, מסומן באדום), מצב זה תקין ונובע מכך שהאוגרים עדיין לא עברו אתחול. לאחר הפעלת האיפוס וטעינת ערכי ה-Bias, מתקבלת יציבות באותות. באיור 27 ניתן לראות את שלב טעינת ההטיות (Bias) אל תוך האוגרים, ואילו באיור 28 מחזור שעון לאחר מכן - ניתן לראות כי ערכי האוגרים (acc) אכן הוטענו בהצלחה מתוך זיכרון ההטיות.



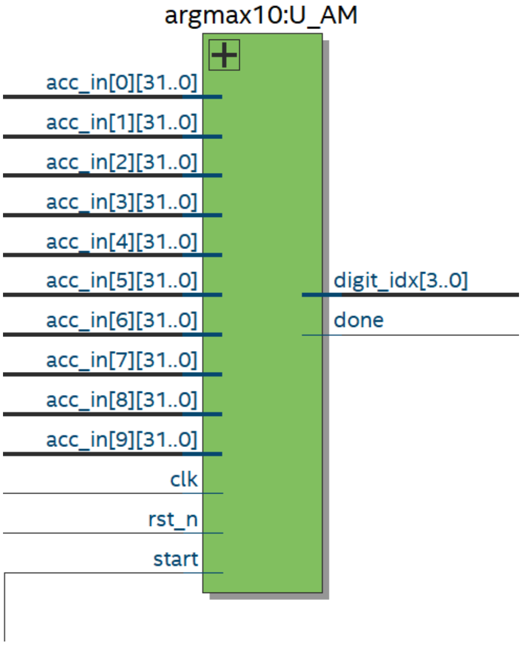

איור 27 - סימולציה של טעינת Bias



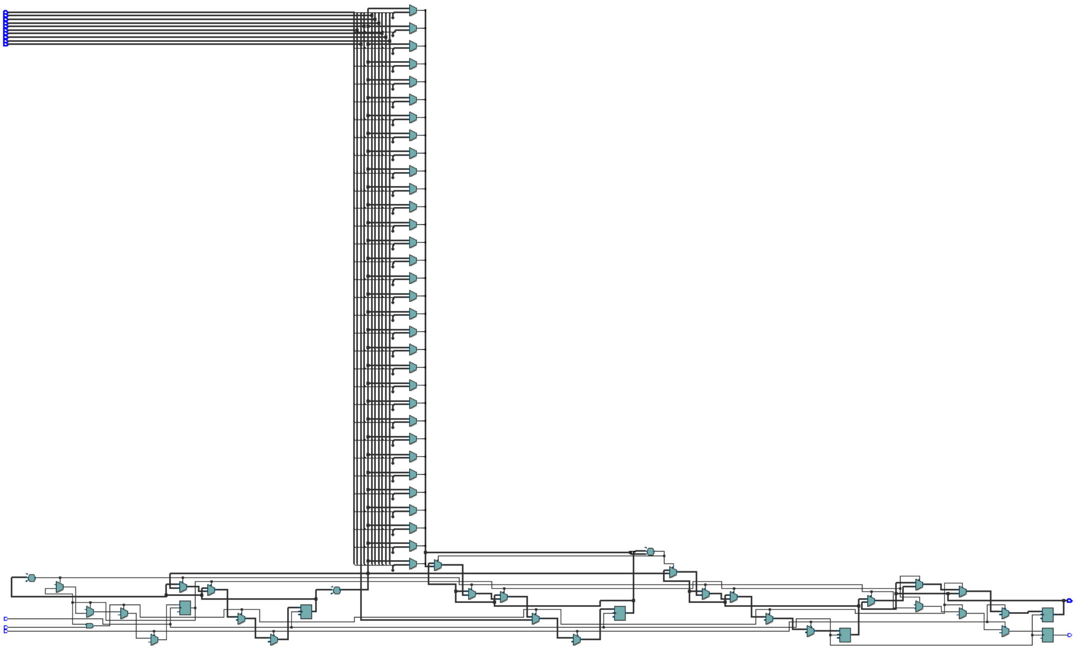

איור 28 - סימולציה של טעינת ערכי האוגרים (acc)

### 3.2.5.6 רכיב argmax10

שרטוט RTL:




*איור 29 - סכמת RTL חיצונית של רכיב argmax10*



*איור 30 - סכמת RTL פנימית של רכיב argmax10*

מטרת הרכיב בארכיטקטורה:

הרכיב אחראי לבחירת התוצאה הסופית של הסיווג - כלומר, לאתר את האינדקס של הנוירון בעל הערך הגבוה ביותר מתוך וקטור הפלט (acc_in). אינדקס זה מייצג את הספרה (0-9) שסווגה על ידי המערכת.

כניסות ויציאות:

לרכיב 4 כניסות ויציאות 2.

הכניסות הן :

- clk - אות שעון, המשמש לקריאת נתונים סינכרונית.
- rst_n - איפוס סינכרוני פעיל נמוך.
- start - פולס בקרה בן מחזור אחד, מתחיל את תהליך החיפוש.
- acc_in - וקטור של 10 ערכים חתומים (int32), תוצאות המצטברות מהרכיב MAC.

היציאות הן :

- done - פולס בקרה בן מחזור אחד, מאותת שהתוצאה מוכנה.
- digit_idx - אינדקס (4 ביט) של הערך המקסימלי, כלומר הספרה שסווגה.

לוגיקת הרכיב:

- בעת איפוס (rst_n=0) מאופסים כל האוגרים הפנימיים.
- כאשר start=1 והרכיב פנוי (busy=0), מתחיל סריקה: הערך הראשון ((acc_in(0)) נשמר כערך מקסימום ראשוני.
- בכל מחזור שעון נבדק ערך נוסף מ-acc_in(i+1) מול הערך המקסימלי הנוכחי:
  - אם הערך החדש גדול יותר הוא נבחר כ-maxv ומעודכן האינדקס.
  - אם הערך שווה נשמר הערך הראשון (כלומר "First Max Wins").
- לאחר סריקת כל 10 הערכים, הפלט digit_idx נרשם, ויוצא פולס done למחזור אחד.

משמעות חישובית:

- לאחר 10 מחזורים (מספר הכניסות = 10), digit_idx מכיל את הספרה הסופית שסווגה.
- רכיב זה למעשה מממש את שלב ArgMax שמבצעים בסוף כל רשת נוירונים לצורך המרת logits לתווית (label).

יתרונות המימוש:

- פשטות ויעילות: הרכיב סורק את הערכים באופן סדרתי לאורך 10 מחזורים בלבד.
- התאמה לארכיטקטורה: מאפשר חיבור ישיר לוקטור הפלט של רכיב ה-MAC, ומוציא תוצאה נקייה לשלב התצוגה (Decoder 7-Segment).

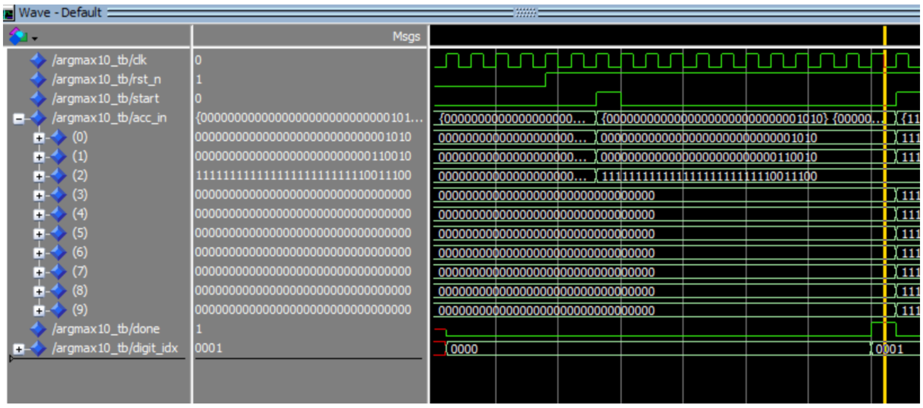
אימות וסימולציה:

מטרה:
לאמת את פעולת רכיב ה-argmax10 שבוחר את הערך המקסימלי מתוך עשרת הערוצים (acc_in) ולוודא שהאינדקס המוחזר (digit_idx) ואות הסיום (done) תקינים.

תרחישי בדיקה:

- איפוס - בדיקה שכל הערכים מאותחלים, והפלט digit_idx חוזר ל-0.
- מקסימום ייחודי - כאשר ערוץ אחד גדול מכולם (למשל index=1), נבדק שהפלט מחזיר את אותו אינדקס.
- ערכים שליליים - נבדק שהרכיב מזהה נכון את הערך ה"פחות שלילי" כמקסימום (למשל index=7 עם ‎-3).
- מדרג עולה - סדרה 0..9, נבדק שהפלט הוא index=9, הערך הגבוה ביותר.
- שוויון ערכים (tie) - בדיקה עם ערכים זהים ביותר מערוץ אחד, ונבדק שהמדיניות FIRST MAX WINS אכן מיושמת והאינדקס הנמוך יותר נבחר.

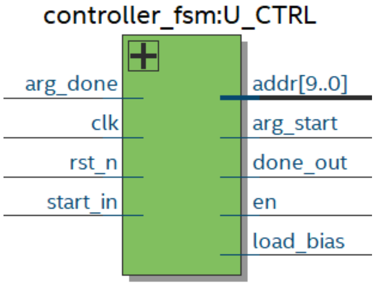
בסימולציה באיור 31 ניתן לראות כי בתחילה האותות מופיעים במצב בלתי מאותחל (U, באדום) תופעה תקינה הנובעת מכך שהאוגרים טרם עברו איפוס. לאחר הפעלת אות האיפוס וטעינת ערכי ה-Bias, מתקבלת יציבות באותות. בשלב זה הוזנו לכניסה acc_in עשרה ערכים ידועים מראש, כאשר הערך המקסימלי הוגדר באינדקס 1. לאחר סריקה מלאה של כל עשרת הערכים, בתום 10 מחזורי שעון, ניתן לראות כי הפלט digit_idx מתייצב על הערך 1, ואות done מונף ל-1 למשך מחזור שעון יחיד - דבר המסמן שהתוצאה זמינה ומאשר כי הרכיב פעל כמצופה.



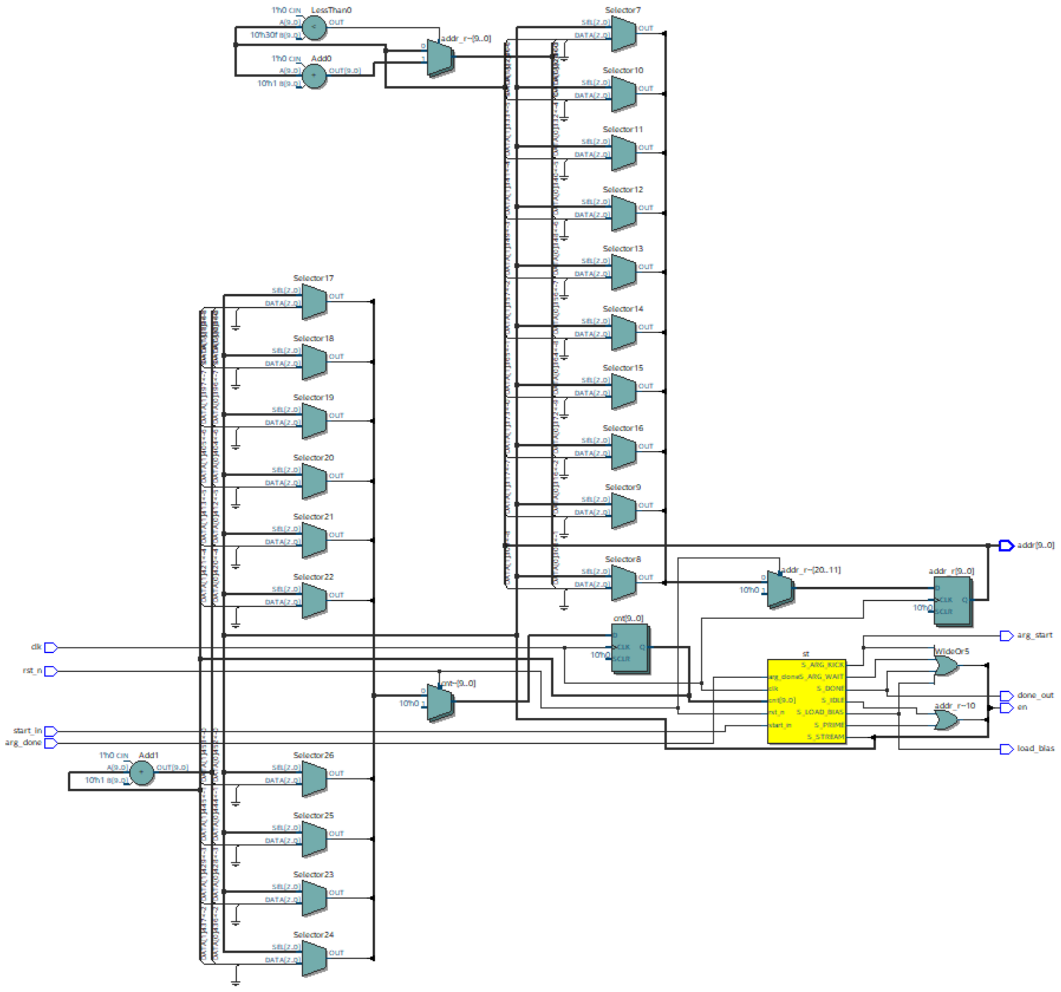

*איור 31 - סימולציה של קבלת הערך המקסימלי*

### 3.2.5.7 רכיב controller_fsm

שרטוט RTL:

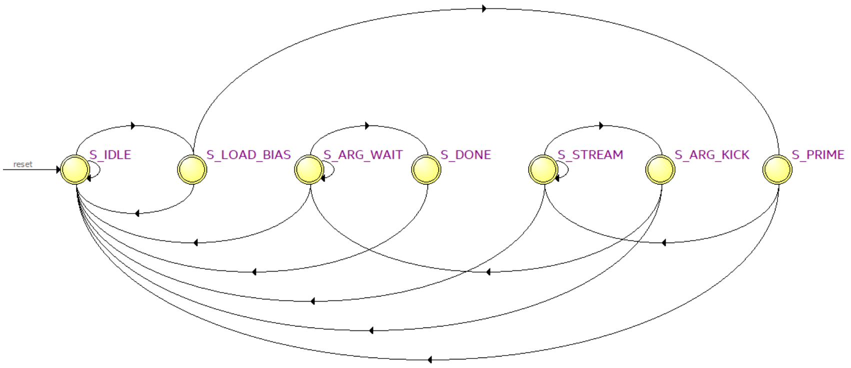



*איור 32 - סכמת RTL חיצונית של רכיב controller_fsm*




*איור 33 - סכמת RTL פנימית של רכיב controller_fsm*

מכונת מצבים:




*איור 34 - דיאגרמת מכונת מצבים*

מטרת הרכיב בארכיטקטורה:

רכיב ה-Controller FSM משמש כיחידת הבקרה המרכזית של המערכת. הוא מתזמן את הפעולות של כל שאר הרכיבים (ROM, MAC, ArgMax) ומנהל את רצף שלב ההסקה (Inference) מתחילתו ועד סופו.

באמצעות מכונת מצבים סופית (Finite State Machine - FSM), הרכיב מוודא שכל שלב מתבצע בסדר הנכון: טעינת ההטיות, עיבוד הפיקסלים, ובסוף בחירת המחלקה בעלת הערך הגבוה ביותר.

כניסות ויציאות:

לרכיב 4 כניסות ויציאות 5.

הכניסות הן :

- clk - אות שעון סינכרוני לכל פעולת הבקרה.
- rst_n - איפוס סינכרוני פעיל נמוך, מחזיר את ה-FSM למצב התחלתי.
- start_in - פולס יחיד לפתיחת ריצת אינפרנס חדשה.
- arg_done - אות מה-ArgMax שמודיע שהחישוב הושלם.

היציאות הן :

- addr - כתובת לקריאת פיקסלים מזיכרון הקלט.
- load_bias - פולס חד-פעמי בתחילת התהליך לטעינת ההטיות.
- en - אות בקרה להפעלת יחידת ה-MAC עבור כל פיקסל.
- arg_start - פולס חד-פעמי להתחלת פעולת ה-ArgMax לאחר סיום סריקת הפיקסלים.
- done_out - אות סיום גלובלי, מסמן שהמערכת סיימה מחזור מלא של אינפרנס.

לוגיקת הרכיב:

הרכיב מבוסס על FSM עם המצבים הבאים:

1. S_IDLE - מצב מנוחה, ממתין ל-start_in.
2. S_LOAD_BIAS - טעינת ערכי ההטיות לאוגרים של יחידת ה-MAC.
3. S_PRIME - מחזור המתנה להעמסת הפיקסל הראשון מה-ROM.
4. S_STREAM - קריאה רציפה של כל 784 הפיקסלים מה-ROM והפעלת חישובי MAC.
5. S_ARG_KICK - שליחת פולס arg_start להפעלת ה-ArgMax.
6. S_ARG_WAIT - המתנה לסיום פעולת ה-ArgMax (arg_done).
7. S_DONE - שליחת אות סיום (done_out) וחזרה ל-S_IDLE.

פירוט מצב S_STREAM:

במצב זה מתבצעת קריאה סדרתית של כל פיקסלי התמונה המאוחסנים בזיכרון הקלט (ROM). בכל מחזור שעון אות הכתובת (addr) מוגדל ביחידה אחת, כך שנקרא הפיקסל הבא מתוך התחום 0-783 (סה״כ 784 פיקסלים). באופן זה נוצר רצף נתונים סינכרוני: בכל מחזור מוזן ערך פיקסל חדש לרכיב ה-MAC, במקביל לעשרת המשקלים המתאימים לו. קריאה סדרתית זו מבטיחה כי כלל הפיקסלים יעברו תהליך חישוב מלא, כך שבסיום הזרימה יכילו אוגרי הצבירה (Accumulators) את הסכום המשוקלל הסופי עבור כל אחת מעשר המחלקות.

משמעות חישובית:

הרכיב מבטיח שכל שלבי תהליך ה-Inference יתבצעו בצורה מתוזמנת:

- כל פיקסל מוזן בדיוק במחזור המתאים.
- ההטיות נטענות רק פעם אחת בתחילת החישוב.
- ה-ArgMax מופעל רק אחרי שכל הנתונים עובדו.
- מתקבלת שליטה מלאה על תהליך הסיווג, עם סנכרון בין כל הרכיבים.

יתרונות המימוש:

- מודולריות: כל יחידת חישוב (ROM, MAC, ArgMax) עובדת עצמאית, וה-FSM מחבר ביניהן.
- בקרה פשוטה: כל המצבים מוגדרים בבירור ומקלים על דיבוג בסימולציה.
- התאמה לחומרה: FSM הוא מנגנון סטנדרטי לניהול זרימה בארכיטקטורות FPGA.

אימות וסימולציה:

מטרה:

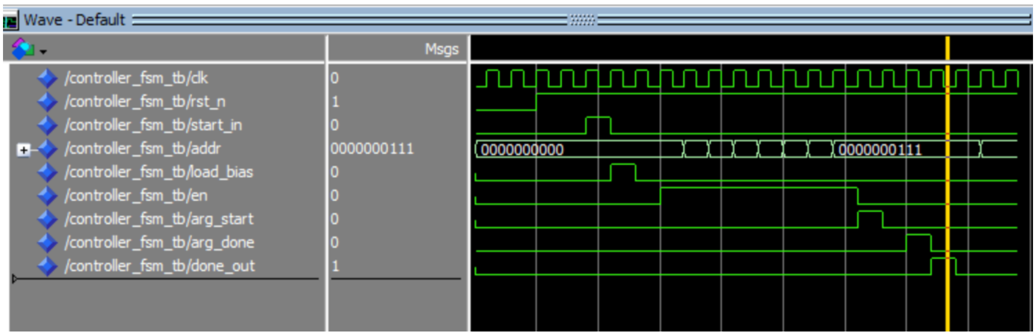
לאמת את פעולת רכיב ה-controller_fsm האחראי על ניהול רצף החישוב: טעינת ה-Bias, קריאת הפיקסלים מה-ROM, הפעלת ה-MAC, והעברת הבקרה לבלוק ה-ArgMax.

תרחישי בדיקה:

- איפוס - בדיקה שהמצב ההתחלתי הוא S_IDLE, הכתובת מאותחלת ל-0, ואין אותות בקרה פעילים.
- טעינת Bias - בדיקה שאות load_bias מונף למחזור בודד מיד לאחר קבלת אות start_in.
- שלב STREAM - בדיקה שאות en נשאר במצב פעיל במשך כל מחזורי הזרימה, ושהכתובת (addr) עולה ברצף מ-0 ועד DEPTH−1.
- מעבר ל-ArgMax - בתום קריאת כל הפיקסלים, נבדק שה-FSM מפיק פולס בקרה arg_start למשך מחזור אחד בלבד.
- סיום - לאחר קבלת arg_done מהרכיב ArgMax, נבדק שה-FSM מעלה את אות done_out, ומחזיר את המערכת למצב S_IDLE.

בסימולציה באיור 35 ניתן לראות כי לאחר מתן פולס התחלה (start_in), ה-FSM פועל לפי כל שלבי הבקרה שהוגדרו: תחילה מופק פולס load_bias לטעינת ההטיות, לאחר מכן מתחיל שלב ה-STREAM שבו האות en נשאר פעיל לאורך כל הקריאה. במקביל, הכתובת (addr) עולה ברצף מ-0 ועד לערך הסופי (במקרה זה DEPTH=8). עם סיום הסריקה מופק פולס יחיד של arg_start, ואחר קבלת arg_done מונף האות done_out, המסמן את סיום התהליך.

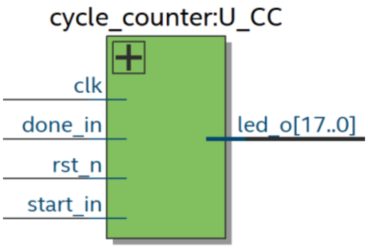
התוצאה מראה שהאותות הופיעו בדיוק במחזורי השעון הצפויים, כל שלבי הרצף בוצעו בסדר הנכון, והרכיב חזר למצב S_IDLE בסיום הפעולה.



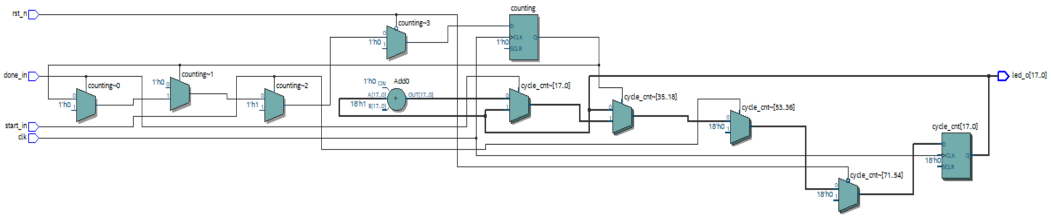

*איור 35 - סימולציה של FSM*

3.2.5.8 רכיב cycle_counter

שרטוט RTL:




*איור 36 - סכמת RTL חיצונית של רכיב cycle_counter*



*איור 37 - סכמת RTL פנימית רכיב cycle_counter*

מטרת הרכיב בארכיטקטורה:

הרכיב משמש למדידת זמן הביצוע של שלב ההסקה (Inference) על גבי ה-FPGA. הוא סופר את מספר מחזורי השעון העוברים בין תחילת התהליך (Start) לבין סיומו (Done). נתון זה מאפשר להעריך את יעילות המימוש החומרתי מבחינת זמן ריצה והשוואה מול מימוש תוכנתי.

כניסות ויציאות:

לרכיב 4 כניסות ויציאה 1.

הכניסות הן:

- clk - אות שעון, המשמש כסנכרון לספירה.
- rst_n - איפוס סינכרוני פעיל-נמוך, מאפס את המונה ואת מצב הספירה.
- start_in - פולס של מחזור אחד, מסמן את תחילת המדידה.
- done_in - פולס של מחזור אחד, מסמן את סיום המדידה.

היציאה היא:

- led_o - וקטור פלט (18 ביט) המציג את מספר מחזורי השעון שנספרו, בהמשך באמצעות לדים פיזיים על גבי הלוח.

לוגיקת הרכיב:

- כאשר rst_n=0, המונה מתאפס והספירה נעצרת.
- כאשר מתקבל פולס start_in, מופעל מצב ספירה (counting=1), והמונֶה מאותחל ל-0.
- כל עוד counting=1:
  - אם מתקבל פולס done_in, הספירה נעצרת (counting=0).
  - אחרת, המונֶה גדל ב-1 בכל חזית עולה של השעון.
- היציאה led_o משקפת תמיד את ערך המונה הנוכחי.

משמעות חישובית:

הערך הסופי של המונה מייצג את זמן הריצה (Latency) של כל תהליך ה-Inference ביחידות של מחזורי שעון. באמצעות ערך זה ניתן להעריך את ביצועי הארכיטקטורה בפועל, להשוות בין תצורות שונות, ולכמת את היתרון של מימוש החומרה לעומת המימוש התוכנתי.

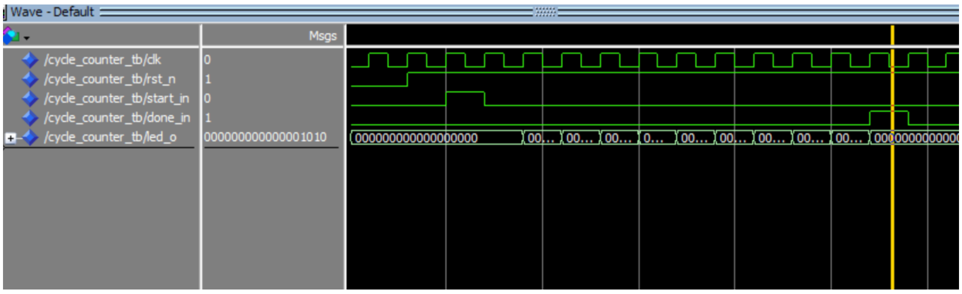
אימות וסימולציה:

מטרה:
לאמת את פעולת רכיב ה-cycle_counter , שמודד את מספר מחזורי השעון בין האותות start_in ו-done_in, ולוודא שהספירה נכונה והפלט led_o תואם למספר המחזורים בפועל.

תרחישי בדיקה:

- איפוס - בדיקה שהמונה מאותחל ל-0 כאשר rst_n=0.
- ספירה בסיסית - הפעלת start_in, המתנה למספר ידוע של מחזורי שעון (למשל 10), ולאחר מכן הפעלת done_in. נבדק שהפלט led_o שווה למספר המחזורים שחלפו.
- דיוק עצירה - אימות שהמחזור שבו done_in מתקבל אינו נספר (כפי שמוגדר במפרט).
- בדיקת יציבות - בדיקה שהמונה נשאר קבוע לאחר done_in ואינו ממשיך לספור.

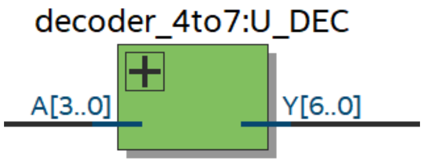
בסימולציה באיור 38 ניתן לראות כי לאחר הפעלת אות האיפוס (rst_n), המונה מאותחל ל-0. עם מתן פולס התחלה (start_in), המונה החל לספור מחזורי שעון. לאחר 10 מחזורים הוזן פולס סיום (done_in), ובעקבותיו נעצרה הספירה והפלט led_o הציג את הערך 10, בדיוק כמצופה. תוצאה זו מאשרת שהרכיב סופר מחזורי שעון בצורה נכונה ומפסיק את הספירה ברגע קבלת אות הסיום.



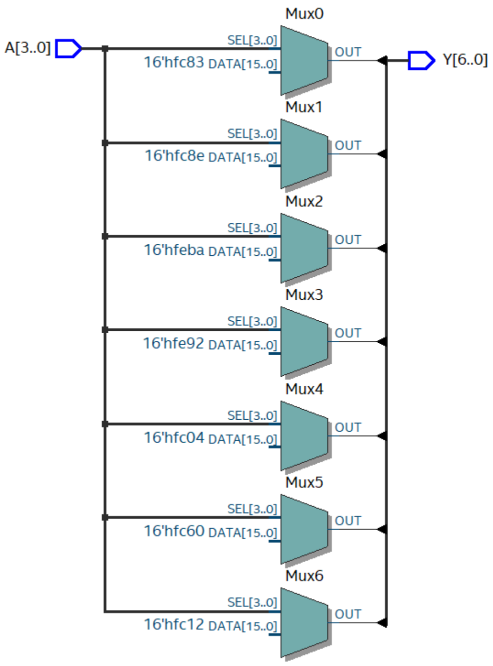

*איור 38 - סימולציה של ספירת המחזורים*

3.2.5.9 רכיב decoder_4to7

שרטוט RTL:




*איור 39 - סכמת RTL חיצונית של רכיב decoder_4to7*




*איור 40 - סכמת RTL פנימית של רכיב decoder_4to7*

מטרת הרכיב בארכיטקטורה:

הרכיב משמש כמודול תצוגה סופי, אשר ממיר את הפלט המספרי (0-9) שהתקבל מרכיב ה-ArgMax לקוד מתאים עבור תצוגת 7-Segment. בצורה זו, ניתן להציג באופן חזותי על גבי הלוח את הספרה שסווגה על ידי רשת הנוירונים.

כניסות ויציאות:

לרכיב כניסה 1 ויציאה 1.

הכניסה היא:

- A - כניסת קוד בינארי ברוחב 4 ביט (unsigned), מייצגת את האינדקס שנבחר על ידי רכיב ה-ArgMax (0-9).

היציאה היא:

- Y - יציאת קוד ברוחב 7 ביט, מייצגת את הדלקת/כיבוי הסגמנטים של תצוגת ה-7- Segment לפי הספרה הרצויה.

לוגיקת הרכיב:

- המימוש מבוסס על case statement:
  - כל ערך אפשרי של A ("0000" עד "1001") מוגדר פלט Y מתאים אשר מדליק את הסגמנטים הדרושים להצגת הספרה המתאימה (0-9).
  - כאשר מתקבל ערך שאינו בתחום התקין, הפלט Y מוגדר כ-"1111111" (מצב BLANK, כלומר כיבוי מלא של התצוגה).
- לדוגמה:
  - עבור "A="0000", Y="1000000 - מוצגת הספרה 0.
  - עבור "A="0001", Y="1111001 - מוצגת הספרה 1.
  - עבור "A="0010", Y="0100100 - מוצגת הספרה 2.

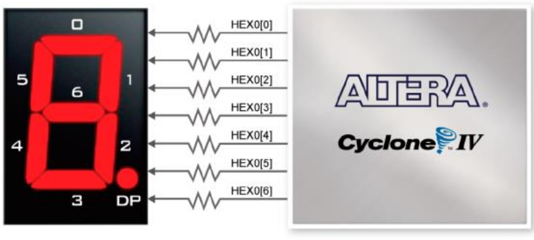



Figure 4-10 Connections between the 7-segment display HEX0 and Cyclone IV E FPGA

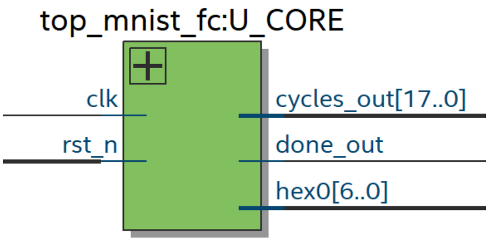
*איור 41 - מיפוי קווי האותות בין תצוגת HEX0 לבין רכיב ה-FPGA (Cyclone IV E)*

משמעות חישובית:

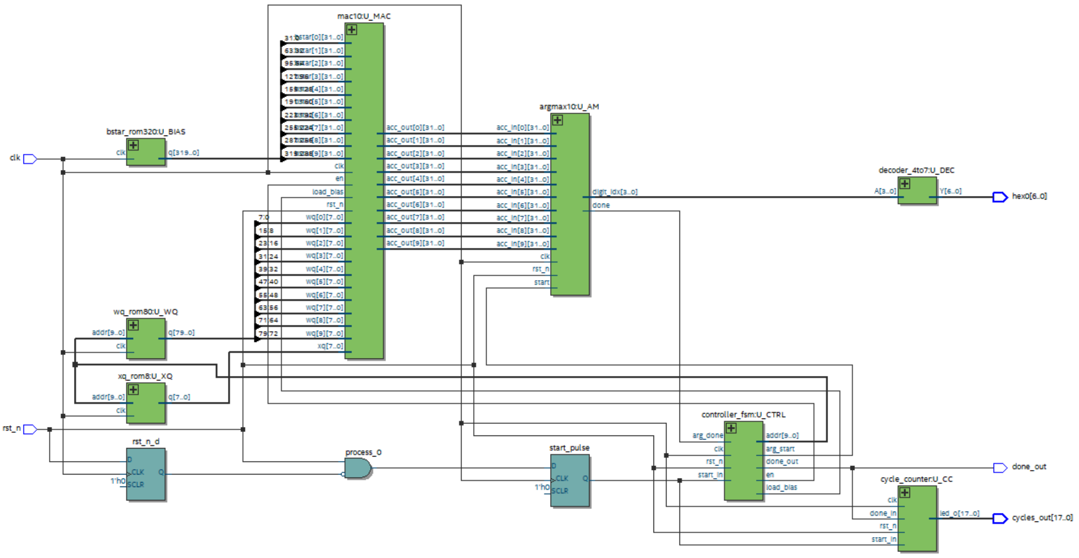
רכיב זה אינו משפיע על חישוב הסיווג עצמו, אלא מהווה שכבת ממשק למשתמש (Output Interface). באמצעותו ניתן לוודא בזמן אמת שהפלט של המערכת החומרתית אכן תואם את התוצאה הצפויה, בצורה פשוטה ונגישה לעין.

### 3.2.5.10 רכיב top_mnist_fc

שרטוט RTL:




*איור 42 - סכמת RTL חיצונית של רכיב top_mnist_fc*




*איור 43 - סכמת RTL פנימית של רכיב top_mnist_fc*

מטרת הרכיב בארכיטקטורה:

הרכיב מהווה את הליבה המרכזית של המערכת כולה. הוא משלב בתוכו את כל רכיבי המשנה (ROMs, MAC, ArgMax, FSM, Counter, Decoder) ויוצר את שרשרת ההסקה (Inference) המלאה על גבי ה-FPGA.

כניסות ויציאות:

לרכיב 2 כניסות ויציאות 3.

הכניסות הן :

- clk - אות השעון הראשי של המערכת, המשמש לסנכרון פעולת כל הרכיבים.
- rst_n - אות איפוס סינכרוני, פעיל נמוך. כאשר האות במצב '0', כל הרכיבים מאותחלים לערכי ברירת מחדל, והמערכת מוכנה להתחלה מחודשת.

היציאות הן :

- done_out - פולס בקרה בודד (One-Cycle Pulse) המופק עם סיום תהליך ההסקה (Inference), ומסמן שהמערכת סיימה לעבד את התמונה הנוכחית.
- cycles_out - וקטור בגודל 18 ביט המשמש כמד מחזורי שעון (Cycle Counter). ערך זה מייצג את משך הזמן (במחזורי שעון) שנדרש למערכת לביצוע מלא של תהליך ההסקה.
- hex0 - וקטור בקרה (7 ביט) עבור תצוגת 7-Segment המובנית בלוח הפיתוח. הפלט מייצג את הספרה (0-9) שסווגה על ידי המערכת, ומאפשר הצגה ויזואלית בזמן אמת.

לוגיקת הרכיב:

- אתחול והתחלה (Start Pulse Generator):
  קיים מנגנון פנימי ליצירת פולס התחלה יחיד (Start Pulse) מיד לאחר שחרור האיפוס. המנגנון מזהה את המעבר של אות האיפוס ממצב פעיל (נמוך) למצב לא פעיל (גבוה), ומפיק פולס בקרה של מחזור שעון אחד בלבד. פולס זה מועבר למכונת המצבים (FSM) ומסמן את תחילת תהליך ההסקה.
  שימוש בפולס יחיד מבטיח התחלה מסונכרנת ומונע התנעות כפולות של המערכת.

- טעינת נתונים מה-ROM:
  רכיב ה-FSM מספק בכל מחזור את הכתובת (addr) לשלושת רכיבי ה-ROM (קלט, משקלים, הטיות). הזיכרונות מחזירים את ערך הפיקסל הנוכחי, את עשרת המשקלים המתאימים לו, ואת וקטור ההטיות השלם.

- חישוב משוקלל (MAC):
  בכל מחזור שעון מוזן פיקסל יחיד ליחידות ה-MAC.
  - כל 10 המכפלות (פיקסל × משקל) מבוצעות במקביל.
  - התוצאות מצטברות באוגרי הביניים (Accumulators).
  - בתחילת התהליך נטענות ההטיות ישירות אל האוגרים, כך שהתוצאה הסופית משקפת סכום משוקלל מלא (פיקסלים × משקלים + Bias).

- בחירת מחלקה (ArgMax):
עם סיום עיבוד כל 784 הפיקסלים, ה-FSM מפעיל את רכיב ArgMax. רכיב זה סורק את עשרת האוגרים, מזהה את הערך המקסימלי, ומפיק את האינדקס המתאים לו. אינדקס זה מייצג את הספרה שסווגה בפועל.

- מדידת ביצועים (Cycle Counter):
במקביל לביצוע ההסקה, רכיב ה-Cycle Counter מודד את מספר מחזורי השעון שחלפו בין תחילת התהליך לבין סיומו. הערך המתקבל מוצג על גבי נוריות הלוח, ומהווה מדד ישיר ליעילות המימוש החומרתי.

- הצגת התוצאה (7-Segment Decoder):
האינדקס שנבחר מועבר לרכיב ה-Decoder, אשר ממיר אותו לתבנית תואמת להצגה על גבי תצוגת 7-Segment. כך מוצגת בזמן אמת הספרה שסווגה על ידי המערכת.

משמעות חישובית:

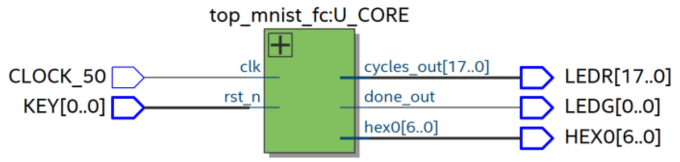
הרכיב מהווה את השכבה האינטגרטיבית של המימוש החומרתי כולו. הוא מאפשר לבצע תהליך הסקה מלא עבור תמונת MNIST בודדת: החל מקריאת נתוני הקלט ועד להצגת הספרה שסווגה בזמן אמת על גבי תצוגת ה-7-Segment.

יתרונות המימוש:

- מודולריות: כל רכיב במערכת פותח ונבדק בנפרד, ושולב כחלק מהרכיב העליון.
- מדידות ביצועים: מונה המחזורים מאפשר להעריך את היעילות החומרתית בפועל.
- נגישות למשתמש: התוצאה מוצגת מיידית בלוח החומרה, ללא צורך בממשקים חיצוניים.

3.2.5.11 רכיב top_level

שרטוט RTL :




*איור 44 - סכמת מעטפת של הרכיב top_mnist_fc למיפוי אותות אל ממשקי הקלט/פלט בלוח DE2-115*

מטרת הרכיב בארכיטקטורה :

רכיב זה מהווה את שכבת החיבור (Wrapper) בין המימוש החומרתי של רשת הנוירונים (top_mnist_fc) לבין רכיבי הקלט/פלט הפיזיים של לוח הפיתוח (DE2-115). הוא אינו מבצע חישוב לוגי נוסף, אלא אחראי על מיפוי האותות אל ממשקי המשתמש בלוח (כפתורים, נוריות, ותצוגת 7-Segment).

כניסות ויציאות :

לרכיב 2 כניסות ויציאות 3.

הכניסות הן :

- CLOCK_50 - שעון מערכת בתדר 50 MHz של הלוח.
- KEY(0) - לחצן Reset, המשמש לאיפוס סינכרוני פעיל נמוך.

היציאות הן :

- LEDG(0)- נורית ירוקה, מהבהבת כאשר מתקבל אות Done (סיום תהליך ההסקה).
- HEX0 - תצוגת 7-Segment המציגה את הספרה שסווגה.
- LEDR(17 downto 0) - מערך נוריות אדומות, מציג את מונה המחזורים (Cycle Counter) בזמן אמת.

לוגיקת הרכיב :

- הרכיב top_mnist_fc משולב כאן כאינסטנציה פנימית (U_CORE).
- האותות CLOCK_50 ו-KEY(0) משמשים כקלטים ל-FPGA וממופים ישירות ל-clk ו-rst_n של הליבה.
- היציאות done_out, hex0, ו-cycles_out ממופות פיזית ל-LEDG, HEX0, ו-LEDR בלוח.

# פרק 4 - תוצאות

לאחר שבפרק 3 הוצגו הארכיטקטורה, הרכיבים וסימולציות ברמת רכיב (Unit Tests), בוצעה הרצה של המערכת השלמה על גבי לוח ה-FPGA. בשלב זה הושלם תהליך הסינתזה ובוצעה בדיקת ניצול משאבים ותזמון (Timing) לכלל המערכת.

## 4.1 ניצול משאבים (Resource Utilization)

כפי שניתן לראות ב-איור 45, המציג את דו"ח הסינתזה לאחר הצריבה, התכן הורץ על גבי רכיב FPGA מסוג Cyclone IV E, דגם EP4CE115F29C7, ניצול המשאבים בפועל היה נמוך מאוד ביחס לקיבולת הכוללת של הרכיב:

- 694 Logic Elements בלבד, המהווים פחות מאחוז אחד מסך המשאבים.
- 414 Registers בשימוש.
- 68,992 ביטים מתוך בלוקי הזיכרון הפנימיים, שהם כ-2% מהנפח הזמין.
- 10 יחידות Multiplier/DSP, שהן כ-2% מהקיבולת הכוללת.
- משאבי ה-PLLs לא נוצלו כלל.

הממצאים מראים כי המימוש החומרתי צורך רק חלק קטן מאוד מהמשאבים של הרכיב, ומותיר גמישות משמעותית להרחבות עתידיות או לשילוב פונקציונליות נוספת במסגרת אותה פלטפורמה.

| Flow Summary | |
|---|---|
| Flow Status | Successful - Sat Sep 27 22:40:17 2025 |
| Quartus Prime Version | 20.1.1 Build 720 11/11/2020 SJ Lite Edition |
| Revision Name | top_level |
| Top-level Entity Name | top_level |
| Family | Cyclone IV E |
| Device | EP4CE115F29C7 |
| Timing Models | Final |
| Total logic elements | 694 / 114,480 ( < 1 % ) |
| Total registers | 414 |
| Total pins | 28 / 529 ( 5 % ) |
| Total virtual pins | 0 |
| Total memory bits | 68,992 / 3,981,312 ( 2 % ) |
| Embedded Multiplier 9-bit elements | 10 / 532 ( 2 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

*איור 45 - דו"ח הסינתזה*

## 4.2 ניתוח תזמון (Timing Analysis)

לאחר יצירת קובץ ההגדרות (.SDC) והגדרת אות השעון הראשי בתדר 50 MHz (מחזור 20 ns), בוצעה בדיקת תזמון מלאה באמצעות כלי Timing Analyzer המובנה בסביבת הפיתוח Quartus. כלי זה מהווה מנגנון פנימי של המערכת, המאפשר לאמת את עמידת התכן בדרישות התזמון לאחר סינתזה ו-Place & Route, תחת מודלי תזמון שונים ובתרחישי עומס מקסימליים.

כפי שניתן לראות ב-איור 46, המציג את דו״ח ה-Timing Analyzer Summary לאחר הצריבה, התכן הורץ על גבי רכיב FPGA מסוג Cyclone IV E, דגם EP4CE115F29C7, תחת מודל תזמון סופי (Final Timing Models) עם השהיות משולבות (Combined Delay Model) ותמיכה בעליות ונפילות אות. נתונים אלו מאשרים שהתכן נבדק תחת תנאי סימולציה מלאים בסביבת הפיתוח של Quartus.

| Timing Analyzer Summary | |
|---|---|
| Quartus Prime Version | Version 20.1.1 Build 720 11/11/2020 SJ Lite Edition |
| Timing Analyzer | Legacy Timing Analyzer |
| Revision Name | top_level |
| Device Family | Cyclone IV E |
| Device Name | EP4CE115F29C7 |
| Timing Models | Final |
| Delay Model | Combined |
| Rise/Fall Delays | Enabled |

*איור 46 - דו״ח ה-Timing Analyzer Summary*

בהמשך, איור 47 מציג את דו״ח ה-Multicorner Timing Analysis Summary. מהדו״ח ניתן לראות כי ערך ה-Worst-case Slack שהתקבל הוא 7.163 ns, נתון המצביע על עמידה מלאה בדרישות התזמון עבור תדר עבודה של 50 MHz (מחזור שעון 20 ns). בנוסף, בדיקות ה-Setup וה-Hold נמצאו תקינות, ולא נצפו חריגות ברוחב פולסים מינימלי (Minimum Pulse Width). תוצאה זו מאשרת כי המימוש החומרתי יציב ועומד בדרישות התזמון של לוח ה-FPGA בתדר העבודה בפועל (50 MHz).

Multicorner Timing Analysis Summary

| | Clock | Setup | Hold | Recovery | Removal | Minimum Pulse Width |
|---|---|---|---|---|---|---|
| 1 | Worst-case Slack | 7.163 | 0.182 | N/A | N/A | 9.201 |
| 1 | CLOCK_50 | 7.163 | 0.182 | N/A | N/A | 9.201 |
| 2 | Design-wide TNS | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 1 | CLOCK_50 | 0.000 | 0.000 | N/A | N/A | 0.000 |

*איור 47 - דו״ח ה-Multicorner Timing Analysis Summary*

בהמשך כפי שניתן לראות בדו״ח ה-Fmax באיור 48, התדר המקסימלי המחושב של התכן הוא 77.9 MHz. מאחר ותדר העבודה שנבחר בפרויקט הוא 50 MHz, הרי שהתכן עומד בדרישות התזמון ואף מציג מרווח ביטחון של כ-28 MHz מעל תדר ההפעלה. נתון זה מחזק את תקינות המימוש החומרתי ומאשר כי הלוגיקה מתוזמנת באופן יציב בסביבת ה-FPGA.

**Slow 1200mV 85C Model Fmax Summary**

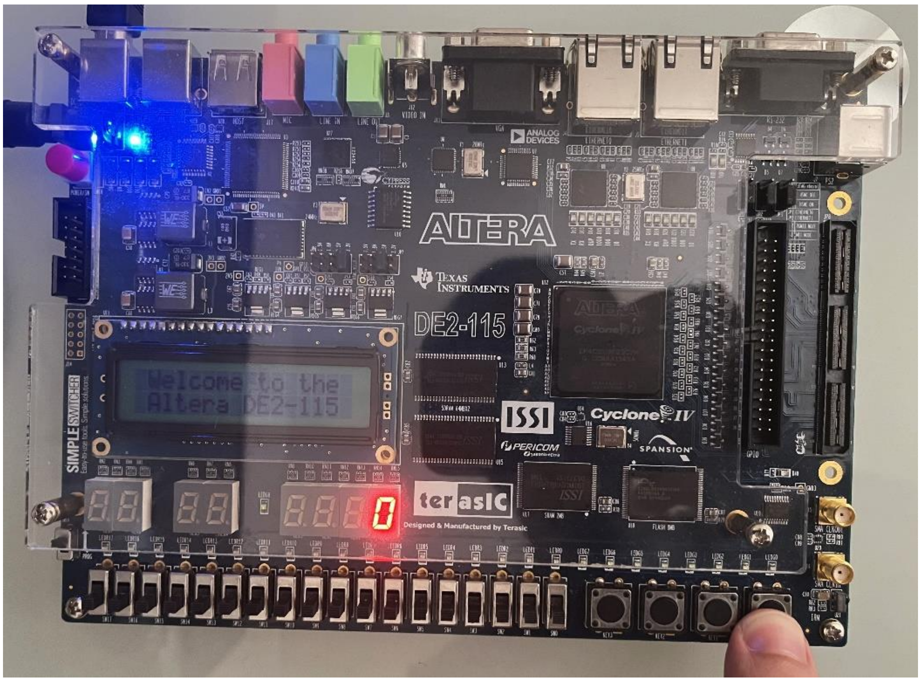
| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 77.9 MHz | 77.9 MHz | CLOCK_50 | |

*איור 48 - דו״ח ה-Fmax*

בשלב זה ניתן היה לעבור להרצות חומרה מלאות, בהן נבדקה התנהגות המימוש בפועל על גבי לוח ה-FPGA, תוך שימוש בקבצי הזיכרון (MIF) שהופקו בתהליך ההכנה. נציג את תוצאות ההרצה הפיזית, הכוללות סיווג של שלוש תמונות שונות ממסד הנתונים MNIST. התוצאות הוצגו בזמן אמת על גבי תצוגת ה-7-Segment של הלוח, ונמדדו במקביל גם על גבי ה-CPU.

לכל אחת משלוש ההרצות נמדד גם זמן ריצה בחומרה באמצעות רכיב ה-Cycle Counter, תוך המרת ספירת מחזורי השעון לזמן מוחלט לפי תדר העבודה שנקבע. נתונים אלה שימשו להשוואה כמותית בין ביצועי החומרה ב-FPGA לבין ביצועי ה-CPU.

בתחילה נבחן מצב האיפוס לאחר הצריבה, בו ניתן היה לוודא כי לוח ה-FPGA אכן מאותחל כראוי: תצוגת ה-Segment- 7 מציגה 0 וכל נורות ה-LED כבויות כפי שמופיע באיור 49 .




*איור 49 - מצב האיפוס בלוח DE2-115: תצוגת 7-Segment מציגה 0, ונוריות ה- LED כבויות*

## 4.3 אימות חומרתי ותוצאות ריצה

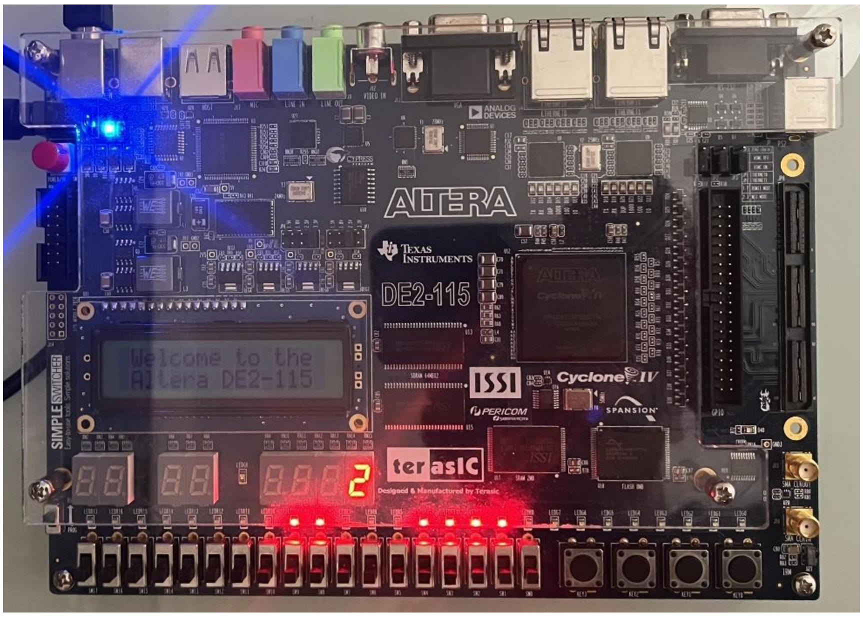
4.3.1 הרצה ראשונה

באיור 50 מוצגת התמונה הראשונה שנבחנה מתוך קבוצת ה-Test של מסד הנתונים MNIST, בעלת תווית אמת (Ground Truth Label) 2 .

*איור 50 - תמונת ספרה 2 מתוך קבוצת ה-Test של מסד הנתונים MNIST*

תוצאות המימוש החומרתי ב-FPGA :
המערכת קיבלה את התמונה כקלט והחלה בתהליך הסיווג החומרתי : קריאת כל 784 הפיקסלים, ביצוע חישוב משוקלל ברכיב ה-MAC, הוספת ערכי ה-Bias, ולבסוף בחירת המחלקה באמצעות רכיב ה-ArgMax. באיור 51 מוצגת התוצאה שהתקבלה על גבי תצוגת ה-7-Segment בלוח ה-FPGA, כאשר הספרה שהופיעה היא 2 , תוצאה התואמת את התווית האמיתית של התמונה.

בנוסף, נמדד זמן הריצה הכולל באמצעות רכיב ה-Cycle Counter, שתוצאתו הוצגה על מערך הלדים האדומים. הערך שנמדד היה 00000001100011110, המייצג 798 מחזורי שעון. מכיוון שתכנון רכיב ה-Cycle Counter במימוש הנוכחי אינו סופר את מחזור ה-done, נוספה התאמה של +1 מחזור לקבלת זמן הריצה הכולל. לפיכך, זמן הביצוע בפועל עמד על 799 מחזורי שעון - המקבילים ל-0.01598 ms בתדר עבודה של 50 MHz.



*איור 51 - מצב ביצוע לאחר הרצה 1 : תצוגת 7-Segment מציגה את הספרה 2*

תוצאות ההרצה התוכנתית ב-CPU:

ההרצה התוכנתית בוצעה על פלטפורמת Google Colab, תוך שימוש במעבד Intel Xeon @ 2.20GHz עם 2 ליבות לוגיות.

באיור 52 מוצגת תוצאת ההרצה המקבילה שבוצעה במודל המאומן על גבי CPU באמצעות Python. גם כאן התוצאה שסווגה היא הספרה 2.

זמן הריצה שנמדד עבור אותה דגימה היה 0.204 ms.

```
model.eval()
with torch.no_grad():
    t0 = time.perf_counter()
    logits = model(x_tensor.to(device))
    t1 = time.perf_counter()

pred = int(torch.argmax(logits, dim=1).item())
print(f"Prediction: {pred}")
print(f"Inference time: {(t1 - t0)*1e3:.3f} ms")
```

```
Prediction: 2
Inference time: 0.204 ms
```

*איור 52 - קוד הרצה ובדיקת זמן חיזוי (Inference) של המודל המאומן ב-Python*

### 4.3.2 הרצה שנייה

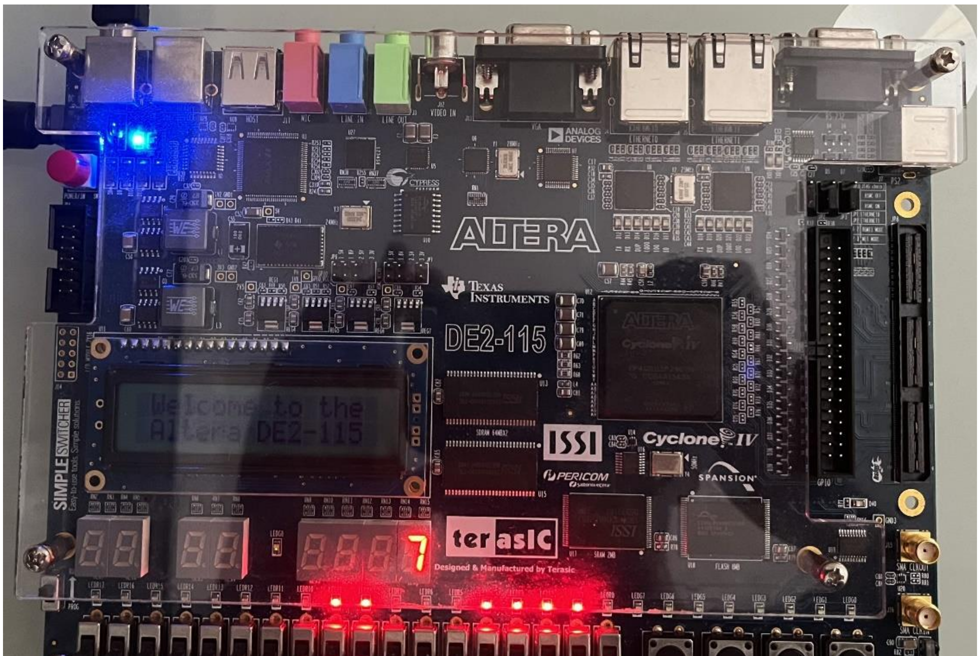
באיור 53 מוצגת התמונה הראשונה שנבחנה מתוך קבוצת ה-Test של מסד הנתונים MNIST, בעלת תווית אמת (Ground Truth Label) 7.

*איור 53 - תמונת ספרה 7 מתוך קבוצת ה-Test של מסד הנתונים MNIST*

תוצאות המימוש החומרתי ב-FPGA:

המערכת קיבלה את התמונה כקלט והחלה בתהליך הסיווג החומרתי: קריאת כל 784 הפיקסלים, ביצוע חישוב משוקלל ברכיב ה-MAC, הוספת ערכי ה-Bias, ולבסוף בחירת המחלקה באמצעות רכיב ה-ArgMax. באיור 54 מוצגת התוצאה שהתקבלה על גבי תצוגת ה-7-Segment בלוח ה-FPGA, כאשר הספרה שהופיעה היא 7 , תוצאה התואמת את התווית האמיתית של התמונה.

בנוסף, נמדד זמן הריצה הכולל באמצעות רכיב ה-Cycle Counter, שתוצאתו הוצגה על מערך הלדים האדומים. הערך שנמדד היה 00000001100011110, המייצג 798 מחזורי שעון. מכיוון שתכנון רכיב ה-Cycle Counter במימוש הנוכחי אינו סופר את מחזור ה-done, נוספה התאמה של 1+ מחזור לקבלת זמן הריצה הכולל. לפיכך, זמן הביצוע בפועל עמד על 799 מחזורי שעון - המקבילים ל-0.01598 ms בתדר עבודה של 50 MHz.




*איור 54 - מצב ביצוע לאחר הרצה 2: תצוגת 7-Segment מציגה את הספרה 7*

תוצאות ההרצה התוכנתית ב-CPU:

ההרצה התוכנתית בוצעה על פלטפורמת Google Colab, תוך שימוש במעבד Intel Xeon @ 2.20GHz עם 2 ליבות לוגיות.

באיור 55 מוצגת תוצאת ההרצה המקבילה שבוצעה במודל המאומן על גבי CPU באמצעות Python. גם כאן התוצאה שסווגה היא הספרה 7.

זמן הריצה שנמדד עבור אותה דגימה היה 0.262 ms.

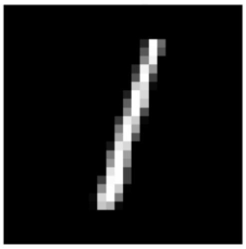
```
model.eval()
with torch.no_grad():
    t0 = time.perf_counter()
    logits = model(x_tensor.to(device))
    t1 = time.perf_counter()

pred = int(torch.argmax(logits, dim=1).item())
print(f"Prediction: {pred}")
print(f"Inference time: {(t1 - t0)*1e3:.3f} ms")
```

```
Prediction: 7
Inference time: 0.262 ms
```

*איור 55 - קוד הרצה ובדיקת זמן חיזוי (Inference) של המודל המאומן ב-Python*

### 4.3.3 הרצה שלישית

באיור 56 מוצגת התמונה הראשונה שנבחנה מתוך קבוצת ה-Test של מסד הנתונים MNIST, בעלת תווית אמת (Ground Truth Label) 1 .

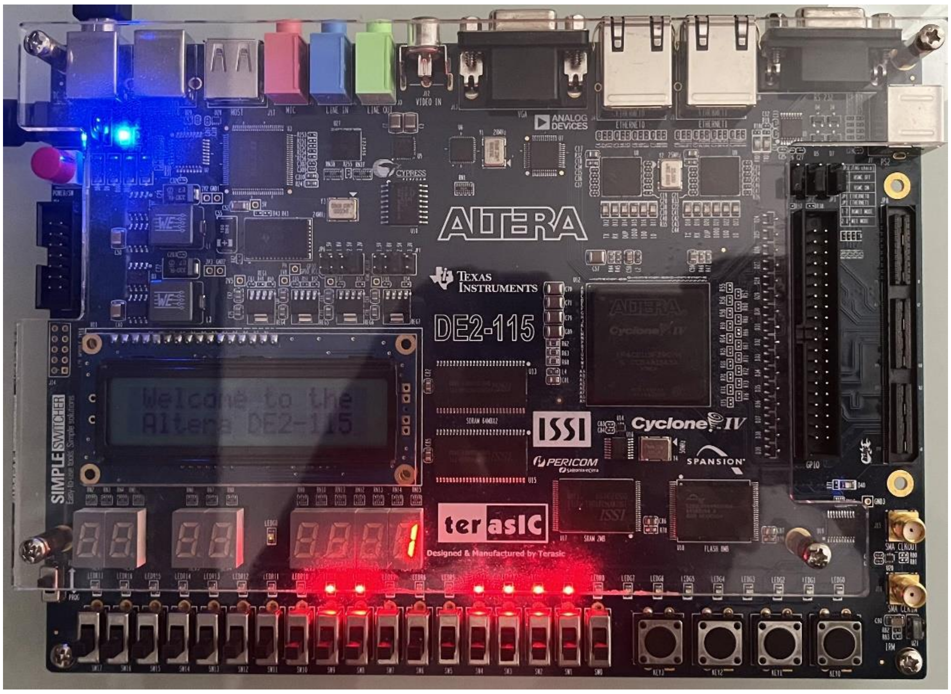


*איור 56 - תמונת ספרה 1 מתוך קבוצת ה-Test של מסד הנתונים MNIST*

תוצאות המימוש החומרתי ב-FPGA :

המערכת קיבלה את התמונה כקלט והחלה בתהליך הסיווג החומרתי : קריאת כל 784 הפיקסלים, ביצוע חישוב משוקלל ברכיב ה-MAC, הוספת ערכי ה-Bias, ולבסוף בחירת המחלקה באמצעות רכיב ה-ArgMax. באיור 57 מוצגת התוצאה שהתקבלה על גבי תצוגת ה- 7-Segment בלוח ה-FPGA, כאשר הספרה שהופיעה היא 1 , תוצאה התואמת את התווית האמיתית של התמונה.

בנוסף, נמדד זמן הריצה הכולל באמצעות רכיב ה-Cycle Counter, שתוצאתו הוצגה על מערך הלדים האדומים. הערך שנמדד היה 00000001100011110, המייצג 798 מחזורי שעון. מכיוון שתכנון רכיב ה-Cycle Counter במימוש הנוכחי אינו סופר את מחזור ה-done, נוספה התאמה של 1+ מחזור לקבלת זמן הריצה הכולל. לפיכך, זמן הביצוע בפועל עמד על 799 מחזורי שעון - המקבילים ל-0.01598 ms בתדר עבודה של 50 MHz.




*איור 57 - מצב ביצוע לאחר הרצה 3 : תצוגת 7-Segment מציגה את הספרה 1*

תוצאות ההרצה התוכנתית ב-CPU :

ההרצה התוכנתית בוצעה על פלטפורמת Google Colab, תוך שימוש במעבד Intel Xeon @ 2.20GHz עם 2 ליבות לוגיות.

באיור 58 מוצגת תוצאת ההרצה המקבילה שבוצעה במודל המאומן על גבי CPU באמצעות Python. גם כאן התוצאה שסווגה היא הספרה 1.

זמן הריצה שנמדד עבור אותה דגימה היה 0.199 ms.

```
model.eval()
with torch.no_grad():
    t0 = time.perf_counter()
    logits = model(x_tensor.to(device))
    t1 = time.perf_counter()

pred = int(torch.argmax(logits, dim=1).item())
print(f"Prediction: {pred}")
print(f"Inference time: {(t1 - t0)*1e3:.3f} ms")
```

```
Prediction: 1
Inference time: 0.199 ms
```

*איור 58 - קוד הרצה ובדיקת זמן חיזוי (Inference) של המודל המאומן ב-Python*

# פרק 5 - מסקנות

## 5.1 מסקנות

הפרויקט הדגים בהצלחה מימוש חומרתי של שלב ה-Inference במודל רשת נוירונים על גבי רכיב FPGA.

מהבדיקות וההרצות שבוצעו עולות המסקנות הבאות:

- המימוש החומרתי סיפק פלט נכון ועקבי לכל הדוגמאות שנבדקו, תוך עמידה במפרט הפונקציונלי.
- זמני הריצה בחומרה היו קצרים משמעותית בהשוואה למימוש התוכנתי, בזכות מקביליות מלאה של פעולות ה-MAC. מדידות ההרצות הראו כי ה-FPGA ביצע את החישובים מהר יותר בכ-12-16 פעמים לעומת ה-CPU, עם יחס ממוצע של פי 14. כלומר, זמן הריצה של ה-FPGA היה קצר פי 14 מזה של ה-CPU.
- השימוש במשאבים ב-FPGA היה נמוך, מה שמאפשר גמישות והרחבה עתידית לרשתות גדולות יותר או לשכבות נוספות.
- ניתוח התזמון (Timing Analysis) הצביע על עמידה בדרישות התזמון בתדר העבודה (50 MHz) ואף על מרווח ביטחון נוסף.
- ההשוואה מול CPU הדגישה את היתרון המרכזי של FPGA בלמידת מכונה - שילוב של ביצועים גבוהים עם ניצול משאבים יעיל.

## 5.2 הצעות עבודה להמשך

במהלך הפרויקט הושגו תוצאות חיוביות שהדגימו את נכונות המימוש ויתרונות ה-FPGA. יחד עם זאת, קיימים מספר כיווני פיתוח אפשריים להמשך:

- הרחבת המודל - מימוש רשת עצבית עמוקה יותר (ריבוי שכבות Fully Connected או שילוב שכבות Convolutional) לצורך שיפור הדיוק בסיווג.
- תמיכה בכניסות מרובות - הרחבת המימוש לטיפול במספר תמונות במקביל (Batch Processing) לניצול טוב יותר של מקביליות ה-FPGA.
- ממשק תקשורת חיצוני - הוספת חיבור UART/Ethernet/PCIe לצורך קבלת נתוני קלט בזמן אמת ממערכת חיצונית והעברת תוצאות חזרה.
- השוואות נוספות - ביצוע ניסויים והשוואת ביצועים מול פלטפורמות חומרה נוספות (למשל GPU או SoC הכוללים ליבות ARM), במטרה לקבל תמונה מקיפה של יתרונות וחסרונות כל טכנולוגיה במימושי למידת מכונה.

לסיכום, הצעות אלו עשויות להעמיק את המימוש ולהרחיב את תחומי היישום, ובכך לנצל את פוטנציאל ה-FPGA לא רק להדגמת יכולות בסיסיות, אלא גם לפיתוח מערכות מתקדמות ובעלות ערך יישומי גבוה.

# ביבליוגרפיה

[1] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J.-S. Seo,
"ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler," *Integration, the VLSI Journal*, vol. 62, pp. 14-23, Jan. 2018.

[2] D. Danopoulos, C. Kachris, and D. Soudris,
"Automatic generation of FPGA kernels from open format CNN models," in *Proc. 28th IEEE Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, 2020.

[3] E. Delaye, A. Sirasao, C. Dudha, and S. Das,
"Deep learning challenges and solutions with Xilinx FPGAs," in *Proc. IEEE Conf. on Field-Programmable Technology (FPT)*, 2017, pp. 908-913.

[4] A. Thomas, A. Biswas, R. Mohan, and D. Bera,
"Design and implementation of FPGA-based hardware acceleration for machine learning using OpenCL: A case study on the K-Means algorithm," in *Proc. Int. Conf. VLSI Design and Embedded Systems (VLSID)*, 2021, pp. 29-34.

[5] S. S. Lingala, S. Bedekar, P. Tyagi, P. Saha, and P. Shahane,
"FPGA based implementation of neural network," in *Proc. Int. Conf. Advances in Computing, Communication and Applied Informatics (ACCAI)*, 2022.

[6] D. H. Noronha, P. H. W. Leong, and S. J. E. Wilton,
"Kibo: An open-source fixed-point tool-kit for training and inference in FPGA-based deep learning networks," in *Proc. IEEE Int. Parallel and Distributed Processing Symp. Workshops (IPDPSW)*, 2018, pp. 178-185.

[7] M. Trigka and E. Dritsas,
"A comprehensive survey of deep learning approaches in image processing," *Sensors*, vol. 25, no. 531, pp. 1-46, Jan. 2025.

[8] "FPGA vs. CPU - Understanding the key differences," *FPGA Insights*, 2023. [Online]. Available: https://fpgainsights.com/fpga/fpga-vs-cpu-understanding-the-key-differences. [Accessed: Sep. 2025].

[9] "Digital image processing tutorial," *GeeksforGeeks*, 2024. [Online]. Available: https://www.geeksforgeeks.org/electronics-engineering/digital-image-processing-tutorial. [Accessed: Sep. 2025].

[10] “VHDL terminology and concepts,” *VHDL Whiz*, 2025. [Online]. Available: https://vhdlwhiz.com/terminology. [Accessed: Sep. 2025].

[11] “Neural networks tutorial (PyTorch),” *PyTorch Official Documentation*, 2025. [Online]. Available: https://docs.pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html. [Accessed: Sep. 2025].

[12] Terasic Technologies Inc., *DE2-115 User Manual*, 2013.

[13] אייל חברבר, שפת תיאור חומרה VHDL. שורש הוצאה לאור, 2007.