# Developing a Quantized CNN Accelerator on Nexys-A7 FPGA for VLSI/AI Hardware Specialization

## 1. Introduction

### 1.1. Project Overview

The objective of this three-month project is to implement a quantized Convolutional Neural Network (CNN) accelerator on a Nexys-A7 Field-Programmable Gate Array (FPGA). This endeavor serves a dual purpose: to gain hands-on experience in the intricate intersection of Very Large Scale Integration (VLSI) design and Artificial Intelligence/Machine Learning (AI/ML) hardware acceleration, and concurrently, to prepare for specialized VLSI/AI-hardware roles within the burgeoning Indian semiconductor industry. The project will involve a detailed, weekly and day-by-day plan, encompassing necessary resources and strategic considerations to achieve both technical proficiency and career readiness.

### 1.2. Report Scope

This report provides a comprehensive blueprint for the aforementioned project. It delves into the technical landscape of VLSI, AI/ML, and FPGAs, detailing essential concepts such as quantization, fixed-point arithmetic, and various hardware optimization techniques. Furthermore, it outlines a structured three-month plan, breaking down complex tasks into manageable weekly and daily activities. Finally, the report offers specific guidance on navigating the career landscape for VLSI/AI hardware engineers in India, emphasizing the skills and experiences gained through

this project that are highly valued by industry leaders.

## 2. Understanding the Technical Landscape: VLSI, AI/ML, and FPGAs

### 2.1. The Intersection of VLSI and AI/ML Hardware Acceleration

The rapid advancement of Artificial Intelligence and Machine Learning has fundamentally reshaped the demands on computing infrastructure. Traditional Central Processing Units (CPUs), even high-performance ones, often struggle with the massive parallel processing requirements of deep learning workloads.[1] This is where specialized hardware accelerators become indispensable, and VLSI technology plays a foundational role in their development.

VLSI enables the design and integration of millions, and even billions, of transistors onto compact semiconductor chips.[2] This miniaturization and optimization are critical for AI/ML applications, which demand immense computational power while often operating under strict power and memory constraints. VLSI techniques are fundamental to creating custom AI hardware accelerators such as Application-Specific Integrated Circuits (ASICs), Graphics Processing Units (GPUs), and FPGAs.[1] For instance, Google's Tensor Processing Units (TPUs) and NVIDIA's AI-focused GPUs are prime examples of hardware built using advanced VLSI methodologies.[2]

FPGAs, in particular, offer unique advantages for AI/ML acceleration. Unlike ASICs, which are fixed-function chips, FPGAs are reconfigurable logic devices that allow for the implementation of entirely custom compute architectures.[2] This reconfigurability is invaluable in the rapidly evolving AI landscape, enabling continuous optimization and redeployment of neural networks.[3] FPGAs excel in parallel processing, a core requirement for neural networks, and can often achieve this in a smaller package with less power consumption compared to GPUs, making them ideal for embedded and edge AI applications.[2] Their high I/O counts also make them a preferred platform for sensor fusion, where multiple data streams need to be captured and processed

on-chip.[3] While general-purpose processors struggle with the specific computation patterns of CNNs, FPGAs offer a compelling balance of performance, energy efficiency, and development flexibility.[4]

The convergence of VLSI design and AI/ML is not limited to just accelerating AI workloads on hardware. A significant shift is occurring where AI itself is becoming a powerful tool *in* VLSI design. Machine Learning algorithms are increasingly integrated into Electronic Design Automation (EDA) tools to automate and optimize intricate design processes.[6] This includes tasks such as layout optimization, enhancing predictive accuracy for power, performance, and area (PPA) metrics, and ultimately reducing the overall design cycle time, which is crucial for market competitiveness.[6] Techniques like Bayesian Optimization and Neural Networks are being used for design space exploration, while Reinforcement Learning and Convolutional Neural Networks are applied to placement strategies and identifying congestion hotspots.[6] This symbiotic relationship means that AI is not just the workload but also a transformative design tool, creating a new paradigm for hardware development. Future VLSI/AI hardware engineers will increasingly need skills not just in traditional hardware design but also in applying ML techniques to design processes and understanding AI algorithms at a deep hardware level. The current project, by focusing on a CNN accelerator, directly contributes to this converging skill set, preparing for a future where AI-driven innovation defines the semiconductor industry.[2]

### 2.2. Quantization and Fixed-Point Arithmetic for AI/ML on FPGAs

Convolutional Neural Networks, while powerful, demand a tremendous amount of multiplication and accumulation (MAC) operations and significant memory resources.[5] Implementing these networks using full-precision (32-bit floating-point) weights and activations, common in software training, is computationally burdensome and memory-intensive for resource-constrained hardware like FPGAs.[7] This is where quantization becomes a crucial technique.

Quantization involves converting high-precision floating-point numbers into lower-bitwidth fixed-point integers.[7] This transformation significantly reduces computational complexity, memory footprint, and power consumption while striving to maintain acceptable model accuracy.[7] For instance, converting MAC operations into bit-wise integer operations, especially with very low precision, is critical for reducing

the computational load on FPGAs.[7]

Fixed-point numbers are represented in the Qm.n format, where 'Q' denotes the fixed-point format, 'm' signifies the number of integer bits, and 'n' represents the number of fractional bits.[16] This fixed binary point simplifies hardware implementation compared to the dynamic binary point of floating-point numbers.[10] The choice of 'm' and 'n' is critical, as it directly impacts the range and precision of the number: more integer bits (m) increase the range, while more fractional bits (n) improve precision.[16] Balancing these two aspects is essential for meeting specific application requirements and preventing issues like overflow (when a result exceeds the representable range) or underflow (when a result is smaller than the smallest representable value).[16] Solutions for these challenges include saturation arithmetic or careful format selection and scaling.[16]

Fixed-point arithmetic operations, such as multiplication, also require careful handling. Multiplying two fixed-point numbers (e.g., Qm1.n1 and Qm2.n2) results in a number with a larger bit-width, specifically Q(m1+m2).(n1+n2).[16] This product often requires truncation or rounding to fit into the desired output format, which can impact precision.[16] Division is even more complex, often necessitating iterative algorithms or dedicated hardware dividers for efficiency.[16]

When implementing fixed-point arithmetic in hardware, it is important to remember that the hardware itself only recognizes fields with certain bit widths, not the abstract concept of fixed-point numbers.[17] Therefore, a layer above the hardware is needed to interpret raw binary words into the chosen format.[17] A common and highly effective verification technique is the use of a "golden model," typically implemented in a high-level language like Python.[17] This model provides correct target outputs against which the Verilog/SystemVerilog hardware implementation can be checked, allowing for identification and correction of bugs.[17] Python functions can convert numbers between standard floating-point and the desired fixed-point representations for comparison.[17] For simplifying fixed-point arithmetic in SystemVerilog, libraries like

fplib exist, which abstract away the error-prone task of managing integer and fractional bits during operations, promoting more generic and parameterizable designs.[13]

Quantization techniques vary in their approach. Post-Training Quantization (PTQ) converts a floating-point model to fixed-point after training.[7] In contrast, Quantization-Aware Training (QAT) integrates the quantization process into the training loop itself, often leading to better accuracy retention.[7] Libraries like Brevitas,

built on PyTorch, support both PTQ and QAT.[7] A critical aspect of quantization is bit-width selection; each layer of a CNN may benefit from a different fixed-point bit-width to optimally balance accuracy and hardware resource utilization.[7] Lower bit-widths (e.g., 1-bit weights and 1-bit activations, or 2-bit weights and 2-bit activations) significantly reduce the usage of Look-Up Tables (LUTs), Flip-Flops (FFs), Digital Signal Processing (DSP) units, and Block RAM (BRAM), leading to reduced energy consumption and hardware costs.[7] Furthermore, multiplier-free quantization strategies, such as K-hot quantization, can substitute traditional multiplications with simpler additions and shifts, further minimizing DSP usage on FPGAs.[8]

The pervasive need for quantization highlights that it is not merely an optimization but a fundamental co-design principle for efficient AI hardware. It mandates a deep understanding of numerical precision trade-offs. The project must involve a systematic approach to quantization, including experimenting with different bit-widths for weights and activations, understanding their impact on model accuracy, and meticulously verifying fixed-point arithmetic against a floating-point "golden model." This reinforces the need for strong numerical analysis skills alongside hardware design expertise.

## 3. The Nexys-A7 FPGA and Development Tools

### 3.1. Nexys-A7 Specifications

The Nexys A7 board serves as a robust and accessible digital circuit development platform, built around the AMD Artix™ 7 Field Programmable Gate Array.[19] It is particularly well-suited for projects ranging from introductory combinational circuits to complex embedded processors, making it an excellent choice for this CNN accelerator implementation.

The board comes in two primary variants: the Nexys A7-100T and the Nexys A7-50T. The A7-100T, being the larger variant, features the XC7A100T-1CSG324C FPGA.[20] Key programmable logic resources on the A7-100T include 15,850 Logic Slices, which translate to 63,400 6-input LUTs and 126,800 Flip-Flops.[19] For computationally

intensive tasks like CNNs, the board provides 240 dedicated DSP Slices, essential for efficient Multiply-Accumulate (MAC) operations, and 4,860 Kbits of fast Block RAM (BRAM) for on-chip data storage.[19] Clock management is handled by 6 Phase-Locked Loops (PLLs), supporting internal clock speeds exceeding 450MHz.[19]

Beyond the FPGA core, the Nexys A7 is equipped with generous external memory and a rich set of on-board peripherals. It includes 128 MiB of DDR2 SDRAM for larger data storage needs, a 16 MB Quad-SPI Flash for non-volatile configuration storage, and a microSD card slot for additional memory and programming.[19] Connectivity options are comprehensive, featuring a 10/100 Mbps Ethernet PHY, USB-JTAG programming circuitry, a USB-UART bridge for serial communication, and a USB HID Host for connecting peripherals like keyboards, mice, and memory sticks.[19] For user interaction and sensing, the board provides a 12-bit VGA output, PWM audio output, a PDM microphone, a 3-axis accelerometer, a temperature sensor, two 4-digit 7-segment displays, 16 slide switches, 16 LEDs, and two tri-color LEDs.[20] Expansion is facilitated by five Pmod connectors, including one for XADC signals and four for general-purpose FPGA I/O.[20]

### 3.2. Development Tools

The primary development environment for the Nexys A7 is the AMD Vivado™ Design Suite.[20] This comprehensive suite handles the entire FPGA design flow, from synthesis and implementation (place & route) to bitstream generation. A free Standard Edition of Vivado is available, which helps keep development costs down for students and individual developers.[20]

For accelerating the design of complex algorithms like CNNs, Vitis High-Level Synthesis (HLS) is an invaluable tool.[23] Vitis HLS allows designers to express algorithms in high-level languages such as C, C++, or OpenCL, which are then synthesized into Hardware Description Language (HDL).[23] This significantly speeds up the design process and enables rapid exploration of the design space for performance, area, and power trade-offs, a task that would be far more time-consuming with manual RTL coding.[23] Vitis HLS provides powerful pragmas (directives) like

#pragma HLS Unroll, #pragma HLS Pipeline, and #pragma HLS Dataflow that guide the synthesis tool to create highly parallel and pipelined hardware structures.[25] In

cases where HLS tools do not natively infer specific hardware blocks (like DSP slices for MAC operations), custom design flows can be employed to instantiate these blocks directly, ensuring optimal performance.[26]

While the broader Vitis AI platform offers a comprehensive integrated development environment for AI inference acceleration, primarily targeting Zynq, Versal, and Alveo cards, its direct support for Artix-7 FPGAs for DPU deployment is not explicitly confirmed in the available documentation.[23] Nevertheless, the principles and methodologies of Vitis AI, such as quantization, model compilation, and deployment strategies, remain highly relevant and provide valuable insights for custom accelerator development.[23] For programming the Nexys-A7 board, the Digilent Adept software can also be used.[20]

The choice of development flow, whether pure RTL or HLS-based, significantly impacts the project timeline and design complexity. HLS offers a strategic advantage for algorithm-to-hardware mapping, allowing for faster iteration and optimization. For a three-month project, leveraging HLS for the computationally intensive parts of the CNN (e.g., convolution, pooling, activation functions) will be a strategic decision. This approach enables a focus on algorithmic optimization and high-level architectural choices, rather than getting bogged down in low-level RTL for every detail. The project plan should explicitly incorporate HLS as a primary tool for kernel development, complementing traditional HDL skills.

### 3.3. CNN Accelerator Design Methodologies and Optimizations

Designing an efficient CNN accelerator on an FPGA necessitates a deep understanding of architectural methodologies and optimization techniques to maximize throughput, minimize latency, and efficiently utilize limited hardware resources.

**Pipelining and Parallelism:** These are fundamental strategies for enhancing performance.

- **Pipelining** involves processing consecutive layers of the neural network in a pipeline, allowing different stages of computation to operate concurrently.[7] This approach significantly reduces latency and increases overall throughput. High-Level Synthesis (HLS) tools greatly facilitate this by providing pragmas like #pragma HLS Pipeline and #pragma HLS Dataflow, which guide the synthesis

process to create efficient pipelined hardware.[25]

- **Parallelism** involves replicating hardware units to process multiple data elements simultaneously. This can manifest as channel-parallelism, where multiple input or output channels are processed concurrently, or as Single-Instruction Multiple-Data (SIMD) structures, where shared hardware units execute operations from different layers in a multiplexed manner.[7] Loop unrolling, achieved through HLS pragmas like
  #pragma HLS Unroll, creates multiple instances of operations that execute in parallel, further reducing latency at the cost of increased resource utilization.[25]

**Modular Hardware Design Principles:** AI accelerators benefit immensely from modular designs that support dynamic reconfiguration and resource reassignment.[1] This modularity allows the system to adapt to changing workloads and to dynamically optimize for energy-accuracy trade-offs in real-time.[32] Key capabilities enabled by such principles include structural plasticity (the ability to add or remove neurons and synapses) and fine-grained reconfigurability of dataflow paths.[32]

**Dataflow Optimization and the Roofline Model:** A critical challenge in designing FPGA-based CNN accelerators is ensuring that computation throughput is well-matched with memory bandwidth.[5] A mismatch can lead to under-utilization of either logic resources or memory bandwidth, hindering overall performance.

- **Dataflow optimization** focuses on efficient data movement. Strategies like "output stationary" dataflow, where partial sums remain on-chip for accumulation, are crucial for minimizing slow and power-intensive off-chip memory accesses.[5] Techniques such as loop tiling and transformation are mandatory to fit data on-chip and maximize data reuse, thereby reducing the total communication volume.[5]
- The **Roofline Model** serves as an analytical design scheme to quantitatively analyze computing throughput and required memory bandwidth.[5] By plotting computational performance against the Computation to Communication (CTC) ratio (FLOPs per DRAM byte access), designers can identify optimal solutions that are not bottlenecked by either compute power or memory bandwidth.[5] A higher CTC ratio indicates lower bandwidth requirements, which translates to fewer I/O ports, LUTs, and hardwired connections for data transfer, ultimately leading to more efficient designs.[5] This emphasis on data movement highlights that the dominant bottleneck in FPGA CNN design is often data movement, not solely computation. Even if compute units are highly parallel, if data cannot be supplied quickly enough, performance will suffer. Therefore, a significant portion of the design effort must be dedicated to optimizing data movement and memory

access patterns, including efficient on-chip buffering and a well-thought-out dataflow.

**High-Level Synthesis (HLS) for Optimization:** HLS tools, such as Xilinx Vitis HLS, are powerful enablers for optimization.[23] They allow designers to express algorithms in C/C++ and synthesize them into HDL, facilitating faster design space exploration for performance, area, and power trade-offs.[28] HLS pragmas are instrumental in driving parallelism and pipelining.[25] Furthermore, custom design flows can be employed to instantiate specific hardware blocks like DSPs for MAC operations when HLS tools do not infer them natively, ensuring optimal hardware utilization.[26] This reinforces the understanding that HLS is a strategic enabler for complex parallelism and pipelining, offering a faster path for algorithm-to-hardware mapping compared to manual RTL.

The following table summarizes key optimization techniques applicable to FPGA-based CNN accelerators:

| Technique | Description | Primary Benefit | Key Considerations/ Trade-offs | Relevant Tools/Pragmas |
|---|---|---|---|---|
| Layer Pipelining | Processing consecutive network layers concurrently in a pipeline. | Latency Reduction, Throughput Increase | Increased resource usage (registers between stages). | HLS pipeline, dataflow |
| Channel Parallelism | Processing multiple input/output channels of a layer simultaneously. | Throughput Increase | High resource usage (MAC units, memory bandwidth). | HLS unroll |
| Loop Unrolling | Replicating operations within loops to execute them in parallel. | Latency Reduction, Throughput Increase | Significant increase in logic resources (LUTs, FFs, DSPs). | HLS unroll |
| Dataflow Optimization | Designing efficient data movement | Throughput Increase, Power Saving | Requires careful memory partitioning | Roofline Model analysis |

| | | | | |
|---|---|---|---|---|
| | patterns (e.g., output stationary) to maximize on-chip data reuse and minimize off-chip memory access. | | (BRAMs) and complex control logic. | |
| Fixed-Point Quantization | Reducing bit-width of weights and activations from floating-point to fixed-point integers. | Resource Reduction, Power Saving | Potential loss of model accuracy; careful bit-width selection is crucial. | Brevitas, QAT/PTQ |
| Clock Gating | Disabling clock signals to idle circuit blocks to prevent unnecessary switching. | Power Saving | Can introduce clock skew; requires careful design to avoid functional errors. | HLS clock gating insertion |
| Operand Isolation | Placing gates to prevent inputs of combinational blocks from toggling when not in use. | Power Saving | Adds slight latency and area overhead. | Manual RTL, HLS optimizations |
| Glitch Minimization | Reducing spurious switching activity in combinational logic. | Power Saving | Requires balanced paths, careful logic restructuring. | HLS reordering, RTL techniques |
| Resource Sharing | Reusing hardware components (e.g., MAC units, BRAMs) across different | Resource Reduction, Power Saving | Can increase latency due to multiplexing; requires complex | HLS resource directive |

| | | | | |
|---|---|---|---|---|
| | operations or layers. | | scheduling. | |

### 3.4. Low-Power Design Considerations for Inference

Power efficiency is a paramount design constraint in modern digital systems, particularly for AI/ML inference on embedded and edge devices.[3] Its direct impact on battery life, thermal reliability, and overall system cost necessitates that it is considered a first-class design constraint, not an afterthought.[28] FPGAs hold a distinct advantage here due to their ability to implement highly custom and optimized compute architectures. This customization allows for the elimination of unused interfaces and repetitive operations, leading to significantly lower power usage compared to more general-purpose CPUs or GPUs for specific AI tasks.[3]

Achieving aggressive power reductions demands a holistic, cross-layer effort, spanning from algorithmic modeling and architectural analysis to RTL design and physical implementation.[28] No single technique is sufficient on its own to meet stringent power targets.[28] This emphasizes the importance of algorithmic-hardware co-optimization for maximum power efficiency.

Key low-power techniques include:

- **Quantization:** As previously discussed, reducing the bit-width of weights and activations directly translates to lower power consumption by simplifying arithmetic operations and reducing memory access energy.[7]
- **Architectural & RTL-level Optimizations:**
  - **Clock Gating:** This widely adopted technique reduces dynamic power by stopping the clock signal to idle circuit blocks, thereby preventing unnecessary switching activity.[28] HLS tools can often automatically infer and insert clock gates, simplifying this process.[28]
  - **Operand Isolation:** Placing isolation gates to prevent the inputs of combinational blocks (such as multipliers) from toggling when they are not actively performing computations reduces dynamic power.[28]
  - **Glitch Minimization:** Spurious switching activity, or "glitches," in combinational logic can account for a substantial portion (up to 40%) of dynamic power consumption.[28] Techniques like reordering operations in HLS for more stable intermediate signals, or RTL techniques such as balanced

path decomposition and gate merging, help mitigate this.[28]
    - **High-Level Synthesis (HLS) Optimizations:** HLS tools are instrumental in exploring power-performance-area trade-offs early in the design flow.[28] Power-aware heuristics within HLS schedulers can be employed to minimize peak switching activity.[28]
    - **Resource Sharing:** Efficiently sharing hardware resources like DSP slices and BRAMs across different operations or layers reduces redundancy, improving overall performance and power efficiency.[16]
- **Backend (Physical) Optimizations:** While more directly applicable to ASIC design, some principles extend to FPGAs. Power-aware placement and routing tools can minimize the capacitance of high-activity nets by ensuring shorter wire lengths, thus reducing switching power.[28] Dynamic Power Management techniques like Dynamic Voltage and Frequency Scaling (DVFS) or Multi-VDD (dividing the chip into voltage islands) can significantly reduce dynamic power.[28] However, FPGAs typically have limited native DVFS support for their logic fabrics.[28]

The pervasive nature of power as a design constraint means that decisions made from the earliest stages of the project—from algorithm selection to architectural choices and clocking strategies—must be evaluated for their power implications. Vivado's power analysis tools should be utilized early and frequently to guide design decisions and identify potential power hotspots. The project aims to quantify power savings as a key achievement, demonstrating a deep understanding of this critical design aspect. Furthermore, the necessity for a holistic, cross-layer effort means the project should actively apply power-aware design principles at every stage. This includes selecting a CNN model amenable to aggressive quantization, designing efficient data paths to minimize redundant switching, and applying RTL-level techniques like clock gating and operand isolation where appropriate. The project documentation will highlight how these multi-level optimizations contribute to the overall power efficiency of the accelerator.

## 4. The 3-Month Project Plan: Weekly Breakdown

This section provides a detailed, week-by-week breakdown of the project, structuring the complex task into manageable phases and integrating the technical concepts and

architectural considerations discussed previously.

### 4.1. Month 1: Foundation and Core Module Development

Week 1: Environment Setup & Basic FPGA Design Flow.
The initial week is dedicated to establishing a functional development environment and gaining foundational proficiency with the FPGA toolchain and board. This lays the groundwork for future debugging and verification, ensuring a solid understanding of the chosen toolchain's capabilities and limitations.

- **Day 1-2:** Install the AMD Vivado™ Design Suite (Standard Edition), which is the primary tool for FPGA development on the Nexys-A7.[20] Ensure all necessary licenses are correctly set up.
- **Day 3-4:** Familiarize with Vivado's basic design flow, including project creation, adding Hardware Description Language (HDL) files, running simulations, performing synthesis, executing implementation (place & route), and generating the final bitstream.[23]
- **Day 5:** Program a simple "Hello World" design, such as an LED blinker or a switch-controlled LED, onto the Nexys-A7. This can be done using Vivado Hardware Manager or Digilent Adept software.[20] Verify that the board functions as expected.
- **Day 6-7:** Set up the Python environment. Install essential ML libraries such as PyTorch or TensorFlow, along with Brevitas for quantization, and NumPy for numerical operations.[7] Run a basic CNN inference example in Python to verify the software stack is correctly installed and operational.
- **Expected Outcome:** A fully functional Vivado installation, successful basic FPGA programming, and a verified Python ML environment.

Week 2: Fixed-Point Arithmetic Unit Design (Verilog/SystemVerilog).
This week focuses on mastering fixed-point arithmetic concepts and implementing the core arithmetic units essential for the accelerator. This is foundational for the entire quantized CNN, as the accuracy and efficiency of these basic arithmetic units directly impact the overall accelerator's performance and resource utilization.

- **Day 1-2:** Conduct a deep dive into fixed-point number representation, specifically the Qm.n format, understanding its range, precision, and the inherent challenges of overflow and underflow.[16] Grasp how these numerical precision considerations apply directly to CNN operations.
- **Day 3-4:** Implement parameterized fixed-point addition and multiplication

modules using Verilog or SystemVerilog.[13] Pay close attention to handling sign bits, proper scaling, and potential truncation or rounding operations.

- **Day 5-7:** Develop a Python "golden model" for fixed-point arithmetic operations. This model will serve as a reference to verify the correctness of the Verilog implementations.[17] Write comprehensive testbenches for the Verilog modules, comparing their outputs against the golden model to ensure functional and numerical accuracy.
- **Expected Outcome:** Verified fixed-point add and multiply modules, coupled with a robust understanding of fixed-point numerical nuances.

Week 3: CNN Layer Implementation - Convolution Engine (HLS/Verilog).
The goal for this week is to design and verify the core convolution engine, integrating fixed-point arithmetic and initial parallelism. The convolution layer is the most computationally intensive part of a CNN, making its efficient implementation paramount for accelerator performance.

- **Day 1-2:** Thoroughly study the architecture of Convolutional Neural Networks, focusing on the mathematical operations, data dependencies, and computational patterns of the convolution layer.[5]
- **Day 3-5:** Begin the design of the convolution engine. Strategically consider using Vitis HLS for rapid prototyping of the core computation, leveraging pragmas like #pragma HLS Unroll and #pragma HLS Pipeline to exploit parallelism and pipelining.[23] If HLS proves challenging or resource-intensive for the target Artix-7, a more direct Verilog implementation based on existing open-source examples can be pursued.[11]
- **Day 6-7:** Integrate the previously developed fixed-point arithmetic into the convolution engine. Create a Python golden model specifically for a single convolutional layer to verify the hardware module's functional correctness and numerical precision.
- **Expected Outcome:** An initial, functional fixed-point convolution engine, along with a clear understanding of the trade-offs between HLS and pure RTL development for this component.

Week 4: CNN Layer Implementation - Activation & Pooling (HLS/Verilog) & Initial Integration.
This week focuses on implementing the remaining fundamental layers (activation and pooling) and integrating them with the convolution engine. This completes the fundamental building blocks of the CNN accelerator and provides a first look at resource consumption on the Nexys-A7.

- **Day 1-2:** Design the Rectified Linear Unit (ReLU) activation function and the Max Pooling unit.[11] These layers are generally simpler than convolution and can often be implemented directly in Verilog or with minimal HLS. Ensure proper handling of

fixed-point data.

- **Day 3-4:** Integrate the convolution, activation, and pooling modules to form a basic single-layer CNN pipeline.[11] Focus on establishing efficient data flow between these interconnected modules.
- **Day 5-7:** Conduct thorough module-level simulations and initial integrated simulations of the single-layer pipeline. Use the Python golden model for end-to-end verification of this pipeline. Begin preliminary resource estimation (LUTs, FFs, DSPs, BRAMs) using Vivado's synthesis reports.
- **Expected Outcome:** A verified single-layer CNN pipeline and an initial resource utilization report for the Nexys-A7.

### 4.2. Month 2: Advanced Optimization and Full Network Integration

Week 5: Quantization Strategy and Model Selection.
This week's focus is on refining the quantization strategy and selecting a suitable quantized CNN model that is feasible for the Nexys-A7's resources. This step directly links the software-side ML model to the hardware implementation, ensuring the chosen model fits the FPGA constraints and maintains accuracy.

- **Day 1-3:** Research and gain a deep understanding of different quantization levels (e.g., 1-bit weights/1-bit activations (W1A1), 2-bit weights/2-bit activations (W2A2)) and their respective impacts on model accuracy and hardware resource usage.[7] Explore the trade-offs between Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) using libraries like Brevitas within PyTorch.[7]
- **Day 4-5:** Select a small, quantized CNN model (e.g., LeNet-5 for MNIST digit classification, or a very compact custom CNN) that is well-suited for the Nexys-A7's available resources.[5] Consider model optimization techniques such as reducing input size, which can significantly save parameters and computation time.[12]
- **Day 6-7:** Train and/or quantize the chosen model in a Python framework (e.g., PyTorch with Brevitas) to generate the fixed-point weights and biases required for hardware implementation. Create a comprehensive golden model for the entire selected network to serve as the ground truth for hardware verification.
- **Expected Outcome:** Quantized CNN model parameters (weights, biases, bit-widths) and a complete golden model for the full network.

Week 6: Pipelining and Dataflow Optimization.
This week is crucial for implementing full network pipelining and optimizing dataflow to

maximize throughput and minimize memory bottlenecks. This directly addresses the critical data movement bottleneck, aiming to improve overall performance and reduce latency.

- **Day 1-3:** Design the overall network architecture, with a strong emphasis on layer-pipelining (enabling consecutive layers to process data concurrently) and channel-parallelism where feasible.[7]
- **Day 4-5:** Apply advanced dataflow optimization techniques, such as loop tiling and local memory promotion, to maximize on-chip data reuse and minimize slow off-chip memory accesses.[5] Explore and implement an output-stationary dataflow if it aligns with the chosen CNN architecture and resource constraints.[26]
- **Day 6-7:** Implement the pipelined data path in Verilog or using HLS. Begin integrating the memory interface for accessing the Nexys-A7's DDR2 SDRAM for storing weights and intermediate feature maps, and efficiently utilizing on-chip BRAMs.
- **Expected Outcome:** A fully pipelined network data path and an initial, functional memory interface.

Week 7: Resource Optimization & Floorplanning.
The objective for this week is to optimize the utilization of FPGA resources (LUTs, FFs, DSPs, BRAMs) and perform initial floorplanning for efficient placement on the Nexys-A7. Maximizing the use of available FPGA resources, especially DSP slices for MAC operations, is crucial for performance. Efficient floorplanning ensures timing closure and avoids routing congestion.

- **Day 1-3:** Analyze detailed resource utilization reports generated by Vivado after synthesis and implementation. Apply HLS pragmas (e.g., ARRAY_PARTITION for memory, RESOURCE for specific IP blocks) or perform manual RTL adjustments to optimize resource usage and meet target constraints.[25]
- **Day 4-5:** Conduct design space exploration to identify optimal parallelism and pipelining factors that balance the desired performance with the specific resource constraints of the Artix-7 FPGA on the Nexys-A7.[5]
- **Day 6-7:** Perform initial floorplanning and pin assignments within Vivado. This involves mapping logical I/O ports to physical pins on the Nexys-A7 board, carefully considering I/O standards and physical constraints.[11] Address any critical warnings related to I/O or timing that arise during this stage.[11]
- **Expected Outcome:** A resource-optimized design that fits within the Nexys-A7's capabilities, along with a preliminary Vivado implementation.

Week 8: Low-Power Design Integration & Timing Closure.
This week focuses on integrating low-power techniques and achieving timing closure for the full design. Power efficiency is a key differentiator for edge AI hardware, and achieving timing closure ensures the design operates reliably at the desired frequency.

- **Day 1-3:** Implement clock gating at appropriate levels within the design, for

instance, for idle blocks or unused Multiply-Accumulate (MAC) units, to reduce dynamic power consumption.[28] Consider incorporating operand isolation techniques for DSP blocks to minimize unnecessary toggling.[28]

- **Day 4-5:** Conduct comprehensive timing analysis in Vivado. Identify and address critical paths by strategically adjusting pipelining stages, restructuring logic, or refining clock constraints to meet performance targets.[11]
- **Day 6-7:** Generate the initial bitstream for the full quantized CNN accelerator. Perform basic on-board testing, such as using UART for simple data transfers or LEDs for status indicators, to confirm fundamental functionality on the Nexys-A7.
- **Expected Outcome:** A timing-closed bitstream, an initial power report, and successful basic on-board functional verification.

### 4.3. Month 3: Testing, Refinement, and Documentation

Week 9: Comprehensive Functional Verification.
This week is dedicated to thoroughly verifying the accelerator's functionality and numerical accuracy. Robust verification is non-negotiable for hardware design, and the golden model approach is critical for ensuring the quantized hardware behaves as expected.

- **Day 1-4:** Develop extensive testbenches for the entire CNN accelerator. Feed various input patterns, such as images from the MNIST dataset, and systematically compare the hardware simulation outputs against the precise results from the Python golden model.[11]
- **Day 5-7:** Meticulously debug any discrepancies identified between the hardware simulation and the golden model. This iterative process is crucial for identifying and rectifying numerical precision issues, potential overflow/underflow conditions, or underlying logic errors within the design.
- **Expected Outcome:** High confidence in the functional correctness and numerical accuracy of the implemented CNN accelerator.

Week 10: Performance & Power Characterization.
The goal for this week is to quantitatively measure and optimize the accelerator's performance (throughput and latency) and power consumption. These are the key metrics demonstrating the success of the hardware acceleration.

- **Day 1-3:** Implement on-chip mechanisms to accurately measure the accelerator's throughput (e.g., frames per second) and latency (e.g., clock cycles per inference) on the FPGA.[12]
- **Day 4-5:** Utilize Vivado's integrated power analysis tools to generate detailed

power reports for the implemented design.[11] Analyze both dynamic and static power consumption components to identify areas for further optimization.
- **Day 6-7:** Iterate on design optimizations based on the measured performance and power results. This might involve fine-tuning parallelism factors, adjusting pipelining depths, or making further refinements to quantization bit-widths. Document all achieved metrics thoroughly.
- **Expected Outcome:** Quantified performance (throughput, latency) and power consumption metrics for the CNN accelerator.

Week 11: Final Integration & Demonstration.
This week focuses on preparing the final demonstration and integrating any remaining components to create a complete and presentable system. A functional demonstration is the tangible output of the project, showcasing its practical application.

- **Day 1-3:** Implement a user-friendly interface for input and output. This could involve using the USB-UART bridge for data transfer, LEDs for status indications, or, if time and ambition allow, the VGA port for displaying processed images.[20]
- **Day 4-5:** Prepare a compelling demonstration of the quantized CNN accelerator running on the Nexys-A7. This could involve classifying real-time MNIST digits or processing a small image stream to showcase its inference capabilities.
- **Day 6-7:** Begin drafting the comprehensive project report. Focus on detailing the technical architecture, implementation specifics, verification methodology, performance and power analysis results, and outlining any challenges encountered and future work.
- **Expected Outcome:** A working demonstration of the CNN accelerator and an initial draft of the project report.

Week 12: Documentation & Career Preparation.
The final week is dedicated to completing comprehensive project documentation and focusing on career readiness. This ensures the project translates into tangible career benefits, showcasing acquired skills and knowledge to potential employers.

- **Day 1-3:** Finalize the project report, ensuring it covers all aspects: detailed architecture, implementation specifics, verification methodology, comprehensive performance and power analysis, and a section on future work. Create a well-organized GitHub repository for the project, including all Verilog/HLS code, testbenches, Python scripts, and the final report.[11]
- **Day 4-5:** Update the resume and professional portfolio with detailed project descriptions. Emphasize quantifiable achievements (e.g., throughput in FPS, latency in clock cycles, resource utilization percentages, power reduction in mW) and clearly articulate the demonstrated skills.[41] Tailor these descriptions specifically for VLSI/AI-hardware roles in India.

- **Day 6-7:** Research leading semiconductor companies in India that are active in VLSI and AI hardware, such as Bharat Electronics Limited (BEL), HCL Technologies, Hitachi Energy, Qualcomm, NVIDIA, and Marvell.[44] Network with professionals in the field, explore relevant job descriptions, and begin preparing for interviews, focusing on required skills like Verilog/SystemVerilog, HLS, digital design principles, timing analysis, scripting (Python, Perl, Tcl), EDA tool proficiency, verification methodologies, and low-power design techniques.[12]
- **Expected Outcome:** Comprehensive project documentation, updated career materials, and a targeted job search strategy for the Indian VLSI/AI hardware market.

## 5. Career Prospects in VLSI/AI Hardware in India

### 5.1. Job Market Overview

India's technology and innovation landscape has been significantly shaped by the advent and integration of Artificial Intelligence. Government initiatives like "Digital India" and substantial private sector investments have fueled research and applications of AI across various sectors, including healthcare, education, finance, and agriculture, establishing India as a crucial player in the global AI landscape.[53] This transformative force is profoundly impacting the job market, particularly the demand for AI Engineers.

The Indian tech sector is projected to generate 2.73 million new tech jobs by 2028, with AI identified as a key catalyst for this job creation, especially in roles requiring advanced technical skills.[54] The demand for AI Engineers in India is high, and this is reflected in the compensation packages. Entry-level AI Engineers (0-3 years of experience) can expect starting salaries ranging from INR 6 lakhs to INR 8 lakhs per annum, while senior AI professionals with over five years of experience or in leadership roles can command salaries upwards of INR 20 lakhs to INR 30 lakhs per annum, with some high-impact roles at leading tech companies offering even more.[49]

Metropolitan cities, known for their vibrant tech industries, offer higher average

salaries. Bangalore, often dubbed India's Silicon Valley, leads with average salaries for AI Engineers ranging from INR 10 lakhs to over INR 30 lakhs per annum. Hyderabad, with its burgeoning tech scene, sees salaries between INR 9 lakhs to INR 28 lakhs annually. Pune, Mumbai, and Delhi-NCR also present significant opportunities, with competitive salary ranges.[53]

Key employers in this domain include major Indian tech companies such as TCS, Infosys, and Wipro, which offer salaries typically between INR 7 lakhs to INR 20 lakhs per annum. Startups in the AI space offer highly variable compensation, sometimes reaching INR 10 lakhs to INR 30 lakhs, often supplemented with equity.[53] Global tech firms with a strong presence in India, such as Qualcomm, NVIDIA, and Marvell, are also significant employers for AI hardware engineers.[45] The overall market growth for AI is robust, with the global AI market value expected to reach $267 billion by 2027, and a projected Compound Annual Growth Rate (CAGR) of 37.3% from 2023-2030.[53]

### 5.2. Essential Skills for VLSI/AI Hardware Roles

To thrive in the specialized field of VLSI and AI hardware engineering in India, a diverse and robust skill set is required.

- **Core Technical Skills:** Proficiency in Hardware Description Languages (HDLs) like Verilog and SystemVerilog is paramount for Register Transfer Level (RTL) design, testbench development, FPGA prototyping, and ASIC development.[45] A strong understanding of digital electronics fundamentals, including Boolean algebra, logic gates, flip-flops, and finite state machines, is essential.[52] Expertise in timing analysis and constraints, FPGA architectures, high-speed interfaces (e.g., PCIe, DDR, Ethernet), and embedded systems is also critical.[45]
- **EDA Tools:** Hands-on experience with industry-standard Electronic Design Automation (EDA) tools is a must. This includes Xilinx Vivado (for FPGA synthesis and implementation), Intel Quartus, and simulation tools like Cadence Xcelium, Synopsys VCS, and Mentor Questa.[45] Familiarity with tools for logic synthesis (e.g., Synopsys Design Compiler) and analog/mixed-signal design (e.g., Cadence Virtuoso) is also beneficial.[49]
- **Verification:** Strong knowledge of verification methodologies is crucial to ensure functional correctness before manufacturing. This includes proficiency in SystemVerilog, Universal Verification Methodology (UVM), functional verification, assertion-based verification, and formal verification.[49] Debugging techniques

using tools like ILA/Chipscope, Logic Analyzers, and Oscilloscopes are also highly valued.[45]

- **Programming/Scripting:** Scripting languages like Python, Perl, and Tcl are indispensable for automation, parsing log files, generating reports, creating custom verification flows, and interfacing with EDA tools.[17] Proficiency in C/C++ is important for hardware modeling and particularly for High-Level Synthesis (HLS) flows.[23]
- **AI/ML Specific Knowledge:** A deep understanding of Convolutional Neural Networks (CNNs), various quantization techniques (e.g., fixed-point, low-bitwidth), and fixed-point arithmetic is fundamental for AI hardware acceleration.[7] Knowledge of low-power design principles, modular hardware design, and hardware-software co-design is also highly sought after.[2]
- **Soft Skills:** Beyond technical prowess, problem-solving abilities, meticulous attention to detail, strong teamwork, and effective communication skills are vital for collaborating with cross-functional teams and translating AI algorithm requirements into hardware specifications.[41]

### 5.3. Leveraging the Project for Career Advancement

The completion of a 3-month project implementing a quantized CNN accelerator on a Nexys-A7 FPGA provides invaluable hands-on experience in a highly sought-after intersection of VLSI and AI/ML.[2] This project serves as a strong foundation for a career in AI hardware engineering.

To effectively leverage this project for career advancement, it is crucial to:

- **Quantify Achievements:** When presenting the project on a resume or in a portfolio, emphasize measurable results. This includes metrics such as achieved throughput (frames per second), latency (clock cycles per inference), resource utilization (e.g., percentage of LUTs, DSPs, BRAMs used), and any demonstrated power reduction (e.g., in milliwatts).[41] For example, stating "Designed and implemented a quantized CNN accelerator on Nexys-A7, achieving X FPS throughput and Y% power reduction compared to baseline" is far more impactful than a generic description.
- **Align Projects with Job Descriptions:** Tailor the resume and portfolio to align with specific job descriptions. Highlight experience with AI accelerators, the implementation of particular AI algorithms (like CNNs), and the demonstrated

ability in hardware-software co-design.[41] If a company focuses on computer vision, emphasize the CNN aspect; if they focus on edge AI, highlight the low-power and resource-constrained aspects of the project.

- **Demonstrate Cross-Functional Collaboration:** Emphasize the ability to translate abstract AI algorithm requirements into concrete hardware specifications. Showcase any instances of successful cross-disciplinary teamwork, for example, working with Python-based ML models to derive hardware parameters or verify results.[41]
- **Create a Robust Portfolio:** Develop a well-documented GitHub repository for the project. This repository should include all source code (Verilog/SystemVerilog, HLS C/C++), testbenches, Python scripts for golden models and data generation, a detailed project report, and potentially demonstration videos of the accelerator in action.[11] A strong online presence with a well-curated portfolio significantly enhances visibility to potential employers.

## 6. Conclusions and Recommendations

### 6.1. Project Feasibility and Key Learnings

The implementation of a quantized CNN accelerator on a Nexys-A7 FPGA within a three-month timeframe is a challenging yet highly feasible and rewarding project. This endeavor provides a comprehensive learning experience at the confluence of VLSI design and AI/ML hardware acceleration. The project's structured approach, from environment setup and core arithmetic unit design to full network integration, verification, and optimization, ensures a deep understanding of the entire hardware development lifecycle for AI accelerators.

Key technical challenges addressed include the meticulous implementation of fixed-point arithmetic, which is fundamental for efficient hardware deployment of CNNs. This involves careful consideration of bit-width selection, scaling, and handling numerical precision trade-offs to balance accuracy with resource constraints. The project also highlights the critical importance of quantization as a co-design principle, not just an afterthought, impacting both performance and power. Furthermore, the

development process underscores that data movement, rather than just raw computation, often represents the dominant bottleneck in FPGA-based CNN accelerators. This necessitates a strong focus on pipelining, parallelism, and efficient dataflow optimization, guided by principles like the Roofline Model. The strategic use of High-Level Synthesis (HLS) is recognized as a powerful enabler for rapidly exploring complex architectural optimizations and translating algorithms into hardware. Finally, the project emphasizes that power efficiency is a first-class design constraint, requiring a holistic, cross-layer approach to achieve aggressive power reductions.

### 6.2. Strategic Recommendations for Career Path

For individuals aspiring to specialize in VLSI/AI-hardware roles in India, this project provides an exceptional foundation. To further solidify career prospects, the following actionable recommendations are provided:

1. **Continuous Learning and Specialization:** The field of AI hardware is rapidly evolving. Continue to deepen knowledge in advanced topics such as more complex CNN architectures (e.g., MobileNetV2, ResNet), emerging quantization techniques, and specialized memory architectures (e.g., High Bandwidth Memory - HBM). Explore advanced HLS techniques and potentially delve into custom ASIC design principles for deeper understanding.
2. **Expand Tool Proficiency:** While Vivado and Vitis HLS are central, familiarity with other EDA tools (e.g., Intel Quartus, Cadence/Synopsys tools for ASIC flow) and simulation environments will broaden employability.
3. **Strengthen Verification Skills:** Verification is a significant part of any hardware design cycle. Invest in advanced verification methodologies like Universal Verification Methodology (UVM) and formal verification, which are highly valued in the industry.
4. **Network Actively:** Engage with the VLSI and AI hardware community in India through conferences, workshops, and online forums. Networking can open doors to opportunities and provide valuable insights into industry trends.
5. **Refine Communication and Collaboration:** Practice articulating complex technical concepts clearly and concisely. Highlight instances of successful cross-functional collaboration in project work, as the ability to bridge hardware and software teams is highly sought after.
6. **Maintain a Dynamic Portfolio:** Continuously update the project portfolio with

new enhancements, performance improvements, or additional features. A well-maintained GitHub repository with clear documentation and quantifiable results serves as a powerful testament to practical skills.

7. **Targeted Job Search:** Research specific companies and their focus areas (e.g., edge AI, data center accelerators, specific application domains like automotive or IoT). Tailor resumes and cover letters to align with their specific needs and technological stacks, emphasizing the direct relevance of the quantized CNN accelerator project.

By meticulously executing this project plan and strategically leveraging the acquired knowledge and skills, individuals can position themselves strongly for impactful and rewarding careers at the forefront of VLSI and AI hardware innovation in India.

**Works cited**


1. AI Accelerator Modules: Hardware Design Challenges & Solutions - Geniatech, accessed July 26, 2025, https://www.geniatech.com/ai-hardware-design-challenges/
2. What Role Does VLSI Play in AI and Machine Learning? - ChipEdge, accessed July 26, 2025, https://chipedge.com/resources/what-role-does-vlsi-play-in-ai-and-machine-learning/
3. AI and FPGAs | Efinix, Inc., accessed July 26, 2025, https://www.efinixinc.com/blog/ai-and-fpgas.html
4. A CNN Accelerator on FPGA Using Depthwise Separable ... - arXiv, accessed July 26, 2025, https://arxiv.org/pdf/1809.01536
5. Optimizing FPGA-based Accelerator Design for Deep Convolutional ..., accessed July 26, 2025, https://ceca.pku.edu.cn/media/lw/b73586e2ac7b5f8d63e8e584f398f17f.pdf
6. Machine Learning Algorithms in VLSI Design: Revolutionizing Circuit Development, accessed July 26, 2025, https://medium.com/@devhuria60/machine-learning-algorithms-in-vlsi-design-revolutionizing-circuit-development-954f2352a353
7. FPGA-QNN: Quantized Neural Network Hardware Acceleration on ..., accessed July 26, 2025, https://www.mdpi.com/2076-3417/15/2/688
8. Trainable Fixed-Point Quantization for Deep Learning Acceleration on FPGAs - arXiv, accessed July 26, 2025, https://arxiv.org/html/2401.17544v1
9. Trainable Fixed-Point Quantization for Deep Learning Acceleration on FPGAs - arXiv, accessed July 26, 2025, https://arxiv.org/pdf/2401.17544
10. Do neural networks using fixed point (as opposed to floating point) arithmetic exist?, accessed July 26, 2025, https://ai.stackexchange.com/questions/46063/do-neural-networks-using-fixed-point-as-opposed-to-floating-point-arithmetic-e
11. Mattjesc/Efficient-FPGA-CNN-Accelerator: Efficient FPGA-Based Accelerator for

Convolutional Neural Networks - GitHub, accessed July 26, 2025, https://github.com/Mattjesc/Efficient-FPGA-CNN-Accelerator
12. FPQNet: Fully Pipelined and Quantized CNN for Ultra-Low Latency Image Classification on FPGAs Using OpenCAPI - MDPI, accessed July 26, 2025, https://www.mdpi.com/2079-9292/12/19/4085
13. Fixflow: A Framework to Evaluate Fixed-point Arithmetic in Light-Weight CNN Inference - arXiv, accessed July 26, 2025, https://arxiv.org/pdf/2302.09564
14. A Hardware-Friendly Low-Bit Power-of-Two Quantization Method for CNNs and Its FPGA Implementation - MDPI, accessed July 26, 2025, https://www.mdpi.com/1424-8220/22/17/6618
15. A High-speed and Low-power FPGA Implementation of Spiking Convolutional Neural Network Using Logarithmic Quantization - ResearchGate, accessed July 26, 2025, https://www.researchgate.net/publication/374831926_A_High-speed_and_Low-power_FPGA_Implementation_of_Spiking_Convolutional_Neural_Network_Using_Logarithmic_Quantization
16. How to Perform Fixed-Point Arithmetic on an FPGA, accessed July 26, 2025, https://runtimerec.com/how-to-perform-fixed-point-arithmetic-on-an-fpga/
17. Adding Fixed point arithmetic to your design - theDataBus.in, accessed July 26, 2025, https://thedatabus.in/fixed-point/
18. SkyworksSolutionsInc/fplib: Fixed point math library for ... - GitHub, accessed July 26, 2025, https://github.com/SkyworksSolutionsInc/fplib
19. Nexys A7 - Farnell, accessed July 26, 2025, http://www.farnell.com/datasheets/4557879.pdf
20. Nexys A7 AMD Artix™ 7 FPGA Trainer Board - Digilent, accessed July 26, 2025, https://digilent.com/shop/nexys-a7-amd-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/
21. Digilent Nexys A7 Development Board For Xilinx Artix-7 FPGA | ipXchange, accessed July 26, 2025, https://ipxchange.tech/evaluation-boards/digilent-nexys-a7-development-board-for-xilinx-artix-7-fpga/
22. Nexys A7 Reference Manual - Digilent Reference, accessed July 26, 2025, https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual
23. Vitis™ Tutorials — Vitis™ Tutorials 2021.1 documentation, accessed July 26, 2025, https://xilinx.github.io/Vitis-Tutorials/2021-1/build/html/index.html
24. Implementation of a Convolutional Neural Network Algorithm on FPGA Using High-Level Synthesis - Webthesis, accessed July 26, 2025, https://webthesis.biblio.polito.it/19272/1/tesi.pdf
25. FPGA-Based Convolutional Neural Network Accelerator with Resource-Optimized Approximate Multiply-Accumulate Unit - MDPI, accessed July 26, 2025, https://www.mdpi.com/2079-9292/10/22/2859
26. CNN Accelerator Throughput Improvement using High-Level Synthesis for FPGA - University of Twente Student Theses, accessed July 26, 2025, https://essay.utwente.nl/90495/1/Minnen_MA_EEMCS.pdf
27. Fused-Layer CNN Accelerators - Stony Brook University, accessed July 26, 2025,

https://compas.cs.stonybrook.edu/~mferdman/downloads.php/MICRO16_Fused_Layer_CNN_Accelerators.pdf
28. Low-Power Techniques for FPGA and ASIC Design ... - Preprints.org, accessed July 26, 2025, https://www.preprints.org/frontend/manuscript/c47e5b957cc039235328203e4bd49e35/download_pub
29. Vitis AI Developer Hub - AMD, accessed July 26, 2025, https://www.amd.com/en/developer/resources/vitis-ai.html
30. Downloads - AMD, accessed July 26, 2025, https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html
31. Vitis AI is Xilinx's development stack for AI inference on Xilinx hardware platforms, including both edge devices and Alveo cards. - GitHub, accessed July 26, 2025, https://github.com/Xilinx/Vitis-AI
32. Design Principles for Lifelong Learning AI Accelerators - OSTI, accessed July 26, 2025, https://www.osti.gov/pages/servlets/purl/2572151
33. Nexys A7 Example Projects - Digilent Reference, accessed July 26, 2025, https://digilent.com/reference/programmable-logic/nexys-a7/demos/start
34. Verification of Convolution Neural Network using Universal Verification Methodology - IRJET, accessed July 26, 2025, https://www.irjet.net/archives/V7/i7/IRJET-V7I7599.pdf
35. thedatabusdotio/fpga-ml-accelerator: This repository hosts ... - GitHub, accessed July 26, 2025, https://github.com/thedatabusdotio/fpga-ml-accelerator
36. sumanth-kalluri/cnn_hardware_acclerator_for_fpga: This is ... - GitHub, accessed July 26, 2025, https://github.com/sumanth-kalluri/cnn_hardware_acclerator_for_fpga
37. doonny/PipeCNN: An OpenCL-based FPGA Accelerator for ... - GitHub, accessed July 26, 2025, https://github.com/doonny/PipeCNN
38. 8krisv/CNN-ACCELERATOR: Hardware accelerator for convolutional neural networks, accessed July 26, 2025, https://github.com/8krisv/CNN-ACCELERATOR
39. Examples of AI Accelerators - GPU, AWS Inferential and Elastic Inference - GitHub, accessed July 26, 2025, https://github.com/shashankprasanna/ai-accelerators-examples
40. MelihGulum/Comprehensive-Data-Science-AI-Project-Portfolio - GitHub, accessed July 26, 2025, https://github.com/MelihGulum/Comprehensive-Data-Science-AI-Project-Portfolio
41. 2025 AI Hardware Engineer Resume Example (+Free Template) - Teal, accessed July 26, 2025, https://www.tealhq.com/resume-example/ai-hardware-engineer
42. 6 Fpga Engineer Resume Examples & Templates for 2025 [Edit ..., accessed July 26, 2025, https://himalayas.app/resumes/fpga-engineer
43. How Meta keeps its AI hardware reliable, accessed July 26, 2025, https://engineering.fb.com/2025/07/22/data-infrastructure/how-meta-keeps-its-ai-hardware-reliable/
44. Best Semiconductor Stocks in India 2025 - Groww, accessed July 26, 2025,

https://groww.in/blog/best-semiconductor-stocks-in-india
45. Emulation(FPGA Design) Engineer /Bluetooth - Staff - Qualcomm Careers, accessed July 26, 2025, https://careers.qualcomm.com/careers/job/446705740007-emulation-fpga-design-engineer-bluetooth-staff-bangalore-karnataka-india?domain=qualcomm.com
46. Jobs at NVIDIA | NVIDIA Careers, accessed July 26, 2025, https://www.nvidia.com/en-in/about-nvidia/careers/
47. Careers | At Marvell, Your Future is Now, accessed July 26, 2025, https://www.marvell.com/company/careers.html
48. www.guvi.in, accessed July 26, 2025, https://www.guvi.in/blog/top-vlsi-design-job-roles/#:~:text=A%20VLSI%20Design%20Engineer%20is,optimize%20circuits%20for%20specific%20applications.
49. Top 8 VLSI Design Job Roles [2025] | GUVI-Blogs, accessed July 26, 2025, https://www.guvi.in/blog/top-vlsi-design-job-roles/
50. AI-Assisted Accelerator Design for FPGA technologies - Queen's University Belfast, accessed July 26, 2025, https://www.qub.ac.uk/courses/postgraduate-research/phd-opportunities/aiassisted-accelerator-design-for-fpga-technologies.html
51. chipxpert.in, accessed July 26, 2025, https://chipxpert.in/career-roadmap-in-vlsi-skills-roles-and-salary/
52. Top VLSI Engineer Skills in 2025: What Companies Really Want - GUVI, accessed July 26, 2025, https://www.guvi.in/blog/top-vlsi-engineer-skills/
53. AI Engineer Salary in India: The Lucrative World of AI Engineering - Simplilearn.com, accessed July 26, 2025, https://www.simplilearn.com/ai-engineer-salary-article
54. India workforce grows in the AI Skills & Jobs Report – ServiceNow Press, accessed July 26, 2025, https://www.servicenow.com/nl/company/media/press-room/india-jobs-growth-ai-report.html