**ELEC 5803 – Behavioural Synthesis of ICs: Project Instruction**

**Term:** Winter 2026

Most graduate students enter hardware courses with a fragmented view of computing systems: they've written software for processors and perhaps HDL for logic, but the software-hardware link feels vague. In modern design, however, that link is rigorously engineered. Algorithms are synthesized into optimized hardware via high-level descriptions and automation tools. This course teaches you to work in that space.

The ELEC 5803 project mirrors real-world hardware development for AI, signal processing, neuromorphic systems, security, and embedded applications. Chips today aren't hand-drawn gates or lengthy RTL; engineers use high-level descriptions, explore architectures, and employ behavioral synthesis to generate and evaluate implementations. The key skill is controlling this process, deciding what to hardware-accelerate, organizing computation spatially/temporally, and evaluating speed, area, and energy.

In this course project, all students will begin from the same canonical baseline, based on this paper (*Toker, O. GitHub Repo for RISC-V RV32I HLS CPU Core H1. 2021*). This processor is available online, which is a minimal RISC-V processor implemented in C++ for High-Level Synthesis. Real systems center on a programmable processor orchestrating specialized accelerators; here, you'll extend the RISC-V core with custom hardware for your chosen domain (e.g., vision, neural networks, cryptography). You'll identify a key computation, move it from software to hardware via behavioral C++ and high-level synthesis (HLS), then explore multiple implementations (pipelining, duplication, memory changes, precision tweaks) to quantify trade-offs and select the best. In this way, all projects start uniform but diverge as you add custom accelerators/instructions, yielding unique RISC-V-based systems by term's end—research-quality prototypes you deeply understand. The iterative process is captured in sequenced reports documenting your design evolution, tool use, AI assistance, and data-driven decisions. Ultimately, you'll gain the mindset of a hardware architect: transforming algorithms into justified, optimized hardware while mastering modern tools and AI collaboration. That's the core of ELEC 5803.

This project focuses on behavioural synthesis, architecture exploration, and design optimization, not on physical board bring-up. In modern chip design, the most important architectural decisions are made long before a design is placed on a physical FPGA or manufactured in silicon. High-Level Synthesis and RTL simulation already provide accurate information about performance, resource usage, and design trade-offs, which is exactly what this course aims to teach. By working at the C++ → HLS → RTL → simulation level, you can explore many architectural variants quickly, reproducibly, and without being limited by board availability, wiring, or I/O constraints. This is the same level of abstraction used in early-stage industrial and academic chip research.

## Required Software and Design Tools for ELEC 5803 Project

| Category | Tool | Purpose in the Project |
|---|---|---|
| **High-Level Synthesis** | **Xilinx / AMD Vitis HLS (Vivado HLS)** | Core behavioural synthesis tool used to convert C/C++ descriptions of the RISC-V core and hardware accelerators into RTL, and to report timing, area, and scheduling information. |
| **RISC-V Compiler** | **riscv64-unknown-elf-gcc** or **LLVM/Clang for RISC-V** | Compiles C and assembly programs into RISC-V binaries that run on the synthesized RISC-V processor. |
| **Binary Utilities** | **RISC-V binutils (objdump, objcopy)** | Converts compiled RISC-V programs into memory images (HEX or binary) that can be loaded into the simulated SoC memory. |
| **RTL Simulation** | **Verilator** or **ModelSim / Questa** | Runs the HLS-generated Verilog to verify that the RISC-V processor and accelerators execute programs correctly. |
| **Python Environment** | **Python 3 with numpy, pandas, matplotlib** | Used for processing synthesis reports, performing design-space exploration, and generating plots for reports and the final paper. |
| **Version Control** | **Git** | Tracks multiple architecture variants and design iterations throughout the project. |
| **AI Design Assistants** | **ChatGPT, Gemini, Grok, Claude, or similar** | Used for generating HLS code, proposing architectures, debugging synthesis issues, and exploring optimization strategies. |

**Reports and Timing**

You will submit 7 milestone reports + final paper.

Each submission is one PDF with two parts:

- **Part A** — AI & Engineering Log
- **Part B** — Technical Report

| Report | Due | What it must contain |
|---|---|---|
| **R1** | End of Week 3 | Baseline RISC-V build, runs, timing & area |
| **R2** | End of Week 4 | CPU-only software version of your kernel |
| **R3** | End of Week 5 | Accelerator or CPU-extension design |
| **R4** | End of Week 7 | First HLS accelerator + RTL metrics |
| **R5** | End of Week 8 | Optimized HLS variant (pipeline/unroll) |
| **R6** | End of Week 10 | Full RISC-V system integration |
| **R7** | End of Week 11 | Design-space exploration (≥3 variants) |
| **Final** | Exam period | IEEE-style paper + presentation |

**Grading**

| Component | Weight |
|---|---|
| **R1–R7 (progress & rigor)** | 40% |
| **Final paper + presentation** | 60% |

Each report is graded lightly but seriously on:

- technical correctness
- architectural depth
- progress
- quality of AI usage

**What makes a good project?**

A strong project shows:

- a real performance bottlenecks
- multiple HLS design variants
- quantitative trade-offs
- and a clear final architecture

This is a **hardware research studio**, not a coding assignment.

## Project Purpose and Phases

### 1. Purpose of the Project

This project is designed as a research-grade behavioural-to-silicon design. You will not simply build a hardware block; you will start from a real RISC-V processor, explore its architectural bottlenecks, and gradually evolve it into a specialized, optimized System-on-Chip (SoC) using High-Level Synthesis (HLS) and AI-assisted design. This mirrors how modern AI accelerators, neuromorphic chips, vision processors, and secure embedded SoCs are built in industry and research. The project integrates three pillars:

- → RISC-V: a real, open ISA and software ecosystem
- → High-Level Synthesis (HLS): behavioural-to-RTL transformation
- → AI (LLMs): automated architecture and code generation assistants

### 2. The Common Starting Point

All students will begin from the same canonical baseline, based on this paper (Toker, O. GitHub Repo for RISC-V RV32I HLS CPU Core H1. 2021. Available online: https://github.com/onurtoker/hls_riscv, accessed on 14 November 2023), which is a minimal RISC-V processor implemented in C++ for High-Level Synthesis. This baseline includes:

- → A single-issue, in-order RISC-V CPU
- → Register file, ALU, PC, decoder
- → Unified instruction/data memory
- → A complete software toolchain (RISC-V GCC/LLVM)
- → Test programs and benchmarks

**Note**: This baseline is the control design. All optimizations, accelerators, and extensions will be evaluated against this reference. You may not replace this processor with another RISC-V core.

### 3. What the RISC-V Processor Represents

In this course, the RISC-V processor is the control plane, not the workload engine. You will primarily build RISC-V–based SoCs with specialized hardware blocks. The CPU:

- → Runs control code
- → Configures accelerators
- → Streams data
- → Collects results

The heavy computation happens in hardware blocks you design through this project.

**4. Three Allowed Architectural Paths**

You may evolve the system in one of three ways:

**Path A — Custom Instructions (CPU Extension)**

You modify the RISC-V core to add:
- A new instruction
- A specialized functional unit
- A secure or approximate operation

Example: "Add instructions to the CPU to run cryptography algorithms."

**Path B — Hardware Accelerators (Primary Path)**

You leave the CPU mostly unchanged and add:
- Memory-mapped accelerators
- Scratchpads
- DMA engines
- Neuromorphic or DSP blocks

This is the **recommended path** for most projects.

Example: "Add an event-driven SNN accelerator to process DVS camera events."

**Path C — Multi-Core or SoC Extensions (Advanced)**
You replicate or connect multiple RISC-V cores or subsystems.
Example: "Dual-core RISC-V with a shared accelerator for image processing."

**5. Mandatory Use of AI Tools**

You are required to use AI tools (ChatGPT, Gemini, Grok, Claude, etc.). They must be used for:
- → Architecture brainstorming
- → HLS C++ generation
- → Microarchitecture variants
- → Debugging and interpretation of synthesis results

You must document:
- → Prompts used
- → AI-generated designs
- → What you accepted, rejected, or modified
- → Why your final architecture differs from the AI's first suggestion

AI is part of the design flow. Using it secretly or without documentation is not allowed.

## 6. Project Phases

### Phase 1 — Baseline Bring-Up (Weeks 1–2)

You will:

- → Build and run the baseline RISC-V SoC
- → Run software on it
- → Measure:
  - Clock frequency
  - Area (LUT/FF/BRAM)
  - Cycles for a test program

This establishes the reference point.

### Phase 2 — AI-Guided Architecture Variants (Weeks 3–5)

You must generate at least three microarchitecture variants of the baseline using HLS directives and structural changes, such as:

- → Pipelining
- → Loop unrolling
- → Resource duplication
- → Memory partitioning

Each variant must be synthesized and measured. You must submit:

- → A table comparing all variants
- → AI prompts and code
- → Your selected “best” design

### Phase 3 — Specialization (Weeks 6–8)

You select a specialization track such as:

- → ML/AI Accelerator
- → Custom instruction
- → Security/cryptography
- → Approximation
- → AI-assisted synthesis

You integrate it into the same RISC-V SoC. You must demonstrate:

- → Correct software operation
- → Hardware integration
- → Performance vs baseline

**Phase 4 — Research-Level Optimization (Weeks 9–11)**

You explore multiple design points of your specialized system. You must produce:

→ Performance vs area curves
→ Speedup vs overhead
→ AI-generated vs human-designed architecture

This phase turns your project into a research study.

**Phase 5 — Final Research Artifact (Weeks 12–13)**

You submit:

→ IEEE-style paper
→ Conference-style presentation
→ Reproducible design package
→ AI usage appendix

**7. Weekly / Bi-Weekly Reports**

You must submit regular reports including:

→ What you built
→ What failed
→ What AI suggested
→ What you changed
→ What you will try next

These reports are a major part of your grade.

**8. How You Will Be Evaluated**

You are evaluated on:

→ Architectural reasoning
→ Optimization depth
→ Quality of experimental results
→ Use of AI as a design tool
→ Technical communication

You are not evaluated on how much code you type.

**9. What This project Is Teaching You**

This project trains you to think like a modern silicon architect. You start with behaviour, explore architecture using AI and HLS, and converge on an optimized hardware system. That is exactly how today's AI chips, vision processors, and neuromorphic devices are built.