**ĐỀ TÀI ĐỒ ÁN LIÊN QUAN ĐẾN RISC-V**

1. **Một số đề tài :**
2. **Đề tài 1 : Thiết kế mở rộng tập lệnh (ISA Extension) :**

* Mục tiêu : dựa trên kiến trúc cơ bản, thêm vào đó một nhóm lệnh (ví dụ như lệnh vector để xử lý DSP, hoặc lệnh thao tác bit nâng cao) vào kiến trúc hiện có
* Phạm vi :
  + Nghiên cứu tập lệnh gốc (RV32IM hoặc RV64IM)
  + Xác định nhu cầu ứng dụng
    - Xử lý ảnh
    - Mã hóa/giải mã
  + Thiết kế encoding và sematics cho lệnh mới
  + Cập nhật assembler (GNU AS) hoặc viết patch cho binutils để hỗ trợ lệnh mới
* Kết quả :
  + Module RTL thực thi được lệnh mở rộng
  + Bộ testbenches đơn vị để kiểm tra đúng chức năng của lệnh
  + So sánh hiệu năng (LUT, FF, …) với một số báo cáo trước đó

1. **Đề tài 2 : Xây dựng trình mô phỏng hướng sự kiện (Event – Driven Simulator) :**

* Mục tiêu : viết một simulator RISC-V đơn giản chạy theo mô hình cycle-accurate hoặc event-driven, dùng dể phân tích chi phí chu kì và băng thông
* Phạm vi :
  + Parse file ELF/RV32 binaries
  + Mô hình pipeline 5 stages cơ bản
  + Quản lý hazard (stall/bypass) đơn giản
  + Thu thập số liệu, tỷ lệ hazard, số chu kì chờ
* Kết quả :
  + Code Cpp hoặc Python chạy được các chương trình mẫu
  + Báo cáo phân tích bottleneck pipeline

1. **Đề tài 3 : Triển khai RISC-V trên FPGA với peripherals cơ bản :**

* Mục tiêu : sử dụng một core RISC-V (Rocket hoặc PicoRV32) tích hợp vào SoC nhỏ với bộ nhớ RAM, UART, timer, GPIO
* Phạm vi :
  + Chọn hoặc tự cài PicoRV32 (open-source)
  + Thiết kế bus (AXI-lite hoặc wishbone)
  + Viết driver (bare-metal C) cho UART và GPIO
  + Chạy một demo (chẳng hạn “Hello, world!” qua UART, điều khiển LED qua GPIO)
* Kết quả :
  + Bitstream FPGA đầy đủ
  + Code firmware và tài liệu hướng dẫn build/deploy

1. **Đề tài 4 : Nghiên cứu bảo mật (Physical Memory Protection) :**

* Mục tiêu : hiện thực hoặc mở rộng module PMP trong RISC-V để cô lập vùng nhớ giữa các process nhỏ hoặc luồng an toàn
* Phạm vi :
  + Tìm hiểu chuẩn RISC-V PMP (Spec v2.2 trở lên)
  + Mô phỏng hành vi violation khi truy cập trái phép
  + Viết testbench kiểm tra
* Kết quả :
  + Module RTL hỗ trợ PMP
  + Kịch bản lỗi và báo cáo kết quả ngăn chặn truy cập

1. **Đề tài 5 : Tích hợp Co – processor / Domain-Specific Accelerator :**

* Mục tiêu : thiết kế một co-processor xử lý một bài toán chuyên biệt (ví dụ khử nền ảnh, xử lý FFT nhỏ) và tích hợp vào pipeline RISC-V
* Phạm vi :
  + Xác định giao diện co-processor (CSR hoặc custom instruction)
  + Viết RTL cho accelerator đơn giản
  + Cập nhật decode logic để dispatch vào co-processor
  + Giao tiếp qua bus nội bộ
* Kết quả :
  + Demo so sánh tốc độ với các code chạy trên core chính
  + Tài liệu chỉ ra kiến trúc và giao tiếp

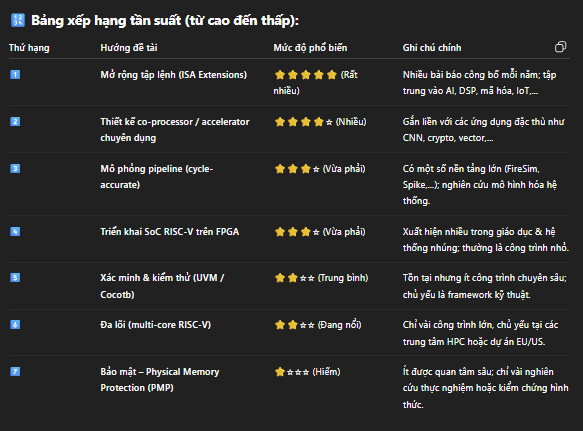
1. **Đề tài 6 : Xây dựng Framework Verification với UVM hoặc Cocotb :**

* Dựng môi trường kiểm thử tự động cho một core RISC-V, phát hiện bug và coverage đạt mức tối thiểu
* Phạm vi :
  + Chọn UVM (system verilog) hoặc Cocotb (Python)
  + Viết các sequence generator sinh ngẫu nhiên lệnh
  + Đo coverage nhóm lệnh, pipeline, hazard, exception
* Kết quả :
  + Test suite tự động report coverage
  + Báo cáo lỗ hỏng

1. **Đề tài 7 : Đa lõi (Multi-core) cơ bản với Shared Memory :**

* Mục tiêu : thêm tối thiểu 2 core RISC-V, chia sẻ bộ nhớ DRAM qua bus và arbiter, thực hiện ví dụ producer – consumer
* Phạm vi :
  + Sử dụng 2 instance PicoRV32
  + Thiết kế arbiter đơn giản round-robin
  + Chạy demo nhỏ (core-A ghi, core-B đọc)
* Kết quả :
  + Verilog SoC đa lõi, testbench cho producer-consumer
  + Báo cáo phân tích contention và latency

1. **Tần suất xuất hiện của các đề tài :**

****

* Dựa vào tần suất xuất hiện này thì đánh giá khả năng thực hiện :
  + ✅ Thiết kế RISC‑V đa lõi với cơ chế đồng bộ nhẹ
  + ✅ Xác minh bảo mật phần cứng (PMP) với test formal hoặc cocotb
  + ✅ ISA extension cho ứng dụng rất đặc thù (ví dụ: mạng cảm biến, AI rìa – edge AI)

1. **Timeline cụ thể cho từng project :**

* Thiết kế RISC-V đa lõi với cơ chế đồng bộ nhẹ

|  |  |
| --- | --- |
| Tuần | Milestone / công việt chính |
| 1 – 2 | Nghiên cứu và chọn core |
| Chọn kiến trúc đa lõi : số lỗi ( 2 – 4 lõi), kiểu arbiter (round – robin) |
| Lên sơ đồ block và giao thức bus |
| 3 – 4 | Viết RTL arbiter & bus arbiter |
| Tích hợp 2 instance core, bus và shared memory |
| 5 – 6 | Thiết kế cơ chế đồng bộ : barrier hoặc lock – free queue |
| Viết testbench producer-consumer đơn giản |
| 7 – 8 | Mở rộng testbench : nhiều lõi hơn, đo contention, latency |
| Phân tích hiệu năng (CPI, throughput) |
| 9 – 10 | Tối ưu hóa bus/arbiter (nếu cần) : pipelining arbiter, FIFO |
| Viết documentation và tutorial simulation |
| 11 – 12 | Synthetic benchmark (ví dụ matrix-product song song) |
| Báo cáo kết quả, so sánh với single-core, hoàn thiện báo cáo |

* Xác minh bảo mật phần cứng (PMP) với test formal hoặc Cocotb

|  |  |
| --- | --- |
| Tuần | Milestone (công việc chính) |
| 1 – 2 | Đọc spec RISC-V PMP. Chọn tool formal (SymbiYosys/Coq/Viper) hoặc Cocotb |
| Lập kế hoạch test : các trường hợp vi phạm quyền, valid access, boundary check |
| 3 – 4 | Viết RTL PMP (nếu cần mở rộng) hoặc lấy module PMP reference |
| Viết testbench Cocotb cơ bản : access pass/fail |
| 5 – 6 | Viết assertions cho formal : kiểm chứng thuộc tính (no out – of bounds, no bypass) |
| Chạy formal verification, fix violations |
| 7 – 8 | Mở rộng testbench Cocotb : random testing, coverage report |
| Phân tích coverage, thêm test case cho corner – cases |
| 9 – 10 | Tích hợp PCA (Power Control Access) hoặc CSR kiểm soát level truy cập |
| Viết report kết quả verification/formal proof |
| 11 – 12 | Thực nghiệm trên mẫu SoC nhỏ (nếu có) : mô phỏng truy cập bị chặn |
| Hoàn thiện tài liệu : test plan, kết quả, recommendation |

* ISA Extension cho các ứng dụng rất đặc thù (Edge AI / Mạng cảm biến)

|  |  |
| --- | --- |
| Tuần | Milestone / Công việc chính |
| 1 – 2 | Khảo sát workload : chọn ví dụ (layer CNN nhỏ, lọc dữ liệu cảm biến) |
| Xác định các phép tính lặp (MAC, shift, bit – field) cần gia tốc |
| 3 – 4 | Thiết kế encoding extension (custom opcode) |
| Cập nhật assembler (thên vào GNU ASM hoặc viết tool dịch đơn giản) |
| 5 – 6 | Viết RTL thực thi lệnh mới |
| Viết testbench RTL cho unit extension |
| 7 – 8 | Viết benchmark C nhỏ sử dụng lệnh mở rộng, cross – compile, chạy simulator |
| Đo speed up vs mã thuần RV 32IM |
| 9 – 10 | Tối ưu hóa encoding (có thể gom nhóm lệnh) và RTL (pipeline integration) |
| Triển khai demo trên FPGA |
| 11 – 12 | Viết báo cáo kết quả : thiết kế ISA, performance, đánh giá trade – off |
| Soạn tài liệu hướng dẫn build toolchain và sử dụng extension |