Week 8

Codefest: Building and co-simulating an SPI interface in Python and Verilog

ECE 410/510
Spring 2025

Challenge #25

Overview and context:
Tomlinson, Li, and Andreou used an *Serial Peripheral Interface* (SPI) in their paper on "*Designing Silicon Brains using LLM: Leveraging ChatGPT for Automated Description of a Spiking Neuron Array*" to interface with the hardware: https://doi.org/10.1109/CAE59785.2024.10487167
SPI is a conceptually simple bus and one of the most widely used interfaces between microcontrollers and peripheral ICs. The throughput of an SPI bus can be 10Mbps and higher.

The above authors used the following prompts to design SPI modules:

- *I want you to now create a SPI interface to communicate with the network module above.*
- *Can you create a top file to connect this SPI module with the network module?*

Learning goals:

- (Vibe) code an SPI interface between your SW (e.g., Python) and HW (e.g., Verilog).
- Do a high-level co-simulation of the HW and SW and benchmark the system-level performance.
- Use cocotb for the co-simulation.

cocotb (reminders):

- https://www.cocotb.org
- cocotb is an open source coroutine-based cosimulation testbench environment for verifying VHDL and SystemVerilog RTL using Python.
- cocotb works with any hardware design that your preferred simulator (and cocotb supports all major simulators) can simulate – be it in (System)Verilog, VHDL, a mixed language, or even a mixed-signal design.
- With cocotb, you write testbenches and verification code in Python. In addition to all the goodies of the Python programming language and its ecosystem, cocotb provides a lean framework to efficiently write verification code .

Tasks:

1. Either find an SPI IP module, design your own, or create one using vibe coding.
    a. My Claude prompt: "*Can you design an SPI interface that interfaces with a software component in Python and a Verilog component?"*
    b. Claude automatically suggested and generated a cocotb SPI interface, which allows to directly control Verilog signals.
    c. You may have to explicitly ask your LLM to do that.
2. Use cocotb to simulate and test the SPI.
    a. My Claude prompt: "*How would I test this in a combined simulation?"*
    b. Claude then generated an entire test suite, include setup scripts and makefiles.
3. Determine the throughput and latency of the SPI interface.
    a. My Claude prompt: "*Can you write a script that measures the throughput and the latency of the SPI interface?*"
    b. Claude generated (1) a standalone Python tool that can measure real-world performance metrics of your SPI interface in various configurations and (2) an advanced cycle-accurate performance analysis cocotb by hooking directly into Verilog.

```
SPI BENCHMARK SUMMARY
============================================================
Average Latency: 32.45 µs
Maximum Throughput: 3850.21 kbps
SPI Clock Frequency: 4.96 MHz
Protocol Overhead: 12.67 µs

Performance by Packet Size:
Size (bytes)    Throughput (kbps)    Efficiency (%)
------------------------------------------------------------
1               320.12               77.52
8               1240.45              79.87
16              2005.32              82.43
32              2680.76              84.92
64              3150.45              86.73
128             3450.67              87.85
256             3677.33              88.54
512             3780.12              89.12
1024            3850.21              89.43
```

Figure 1: Benchmark summary example. Efficiency shows the percentage of time is spent on actual data transfer vs. overhead.