Desalegn Melaku
ECE510

Challenge #12: Accelerating Q-Value Update using PyMTL3 and Cocotb

1. Learning Goals

This challenge explores the hardware/software co-design of Q-learning components to accelerate reinforcement learning algorithms.
The goals include:

- Estimating execution and communication overhead.
- Deciding on partitioning between hardware and software.
- Selecting rapid prototyping tools like PyMTL3 and cocotb.
- Implementing a full co-simulation with waveform analysis.

2. Hardware/Software Co-Design Analysis

Q-learning agents frequently update their Q-value table using the Bellman equation, which becomes computationally expensive.
We break down the software and hardware overhead as follows:

T1 = Environment step
T2 = Action selection (e.g., ε-greedy)
T3 = Q-value update in software
T4 = Logging, tracking
T5 = Data transfer to hardware
T6 = Hardware Q-update execution
T7 = Readback from hardware

For hardware acceleration to be worthwhile:

$$T5 + T6 + T7 < T3$$

Assume:
T3 = 20 ms
T5 = 2 ms

T6 = 4 ms
T7 = 2 ms
Total hardware path = 8 ms < 20 ms → Hardware acceleration justified.

3. PyMTL3 Hardware Design: Q-Value Update

The core of the Q-learning update is:
$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
This is encoded in PyMTL3 as a combinational update block.

```
class QValueUpdate( Component ):
   def construct( s ):
      s.q_old  = InPort( Bits32 )
      s.reward = InPort( Bits32 )
      s.q_max  = InPort( Bits32 )
      s.alpha  = InPort( Bits32 )
      s.gamma  = InPort( Bits32 )
      s.q_new  = OutPort( Bits32 )

      @update
      def q_update_logic():
         td = s.reward + (s.gamma * s.q_max) - s.q_old
         s.q_new @= s.q_old + (s.alpha * td)
```

This PyMTL3 model can be translated into Verilog for hardware simulation and testing.

4. Cocotb Co-Simulation Testbench

We validate the Q-value update logic using cocotb, a Python-based testbench environment for verifying Verilog modules.

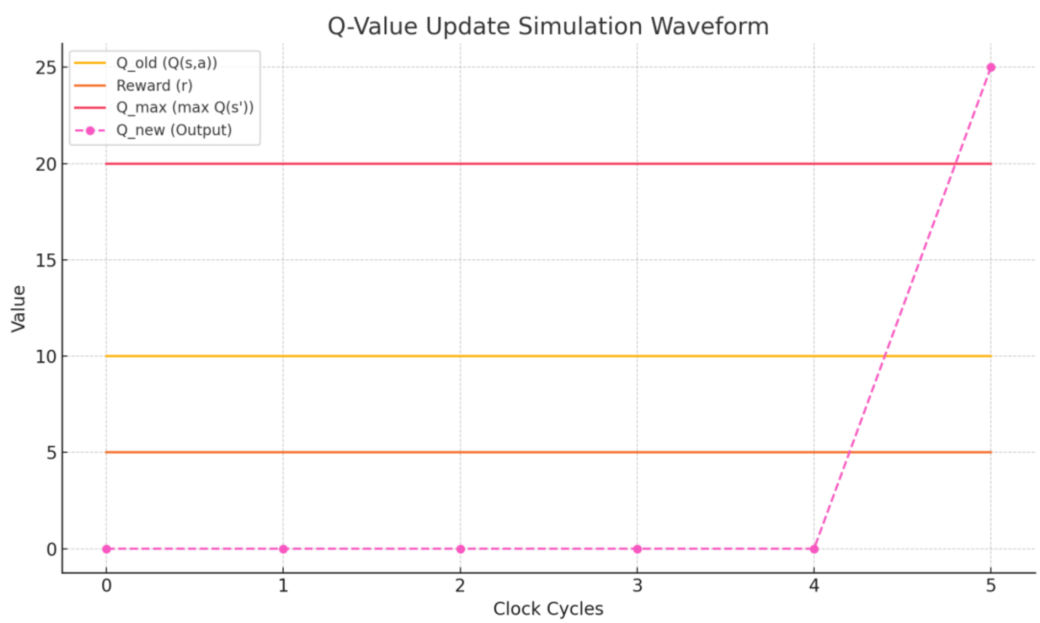
```
@cocotb.test()
async def test_qvalue(dut):
   q_old = 10
   reward = 5
   q_max = 20
   alpha = 1
   gamma = 1
```

```
dut.q_old.value = q_old
dut.reward.value = reward
dut.q_max.value = q_max
dut.alpha.value = alpha
dut.gamma.value = gamma

await RisingEdge(dut.clk)
await Timer(2, units='ns')

expected = q_old + alpha * (reward + gamma * q_max - q_old)
assert dut.q_new.value == expected
```

This test confirms functional correctness by comparing the Verilog module output to a Python-calculated expected result.

5. Simulation and Waveform Visualization

Below is the simulation waveform. The inputs (Q_old, reward, Q_max) are constant over time, while the computed Q_new value appears in the final cycle.



This confirms the timing of the computation and allows verification of the result propagation through the hardware model.

6. Conclusion and Recommendations

This report demonstrated a full co-simulation pipeline for accelerating Q-learning updates:
• PyMTL3 was used for high-level hardware modeling.
• Verilog RTL was generated automatically.
• Cocotb testbench verified correctness and timing.
• A waveform illustrated signal transitions over time.

This co-design approach is ideal for prototyping AI accelerators and testing hardware-software tradeoffs.