Closing Functional Coverage With Deep Reinforcement Learning

A Compression Encoder Example

Eric Ohana
Verification Consultant - Infineon Technologies

Agenda

- Introduction
- RTL Verification and Reinforcement Learning (RL)
- The LZW Compression Encoder Functional Coverage Problem
- Co-simulating a SystemVerilog Test Bench and a PyTorch Agent
- The Deep Q-Learning Agent (DQN)
- Simulation Results: Standard Approach Vs. DQN
- Conclusion
- THANK YOU! (Questions)

Introduction

Closing Functional Coverage

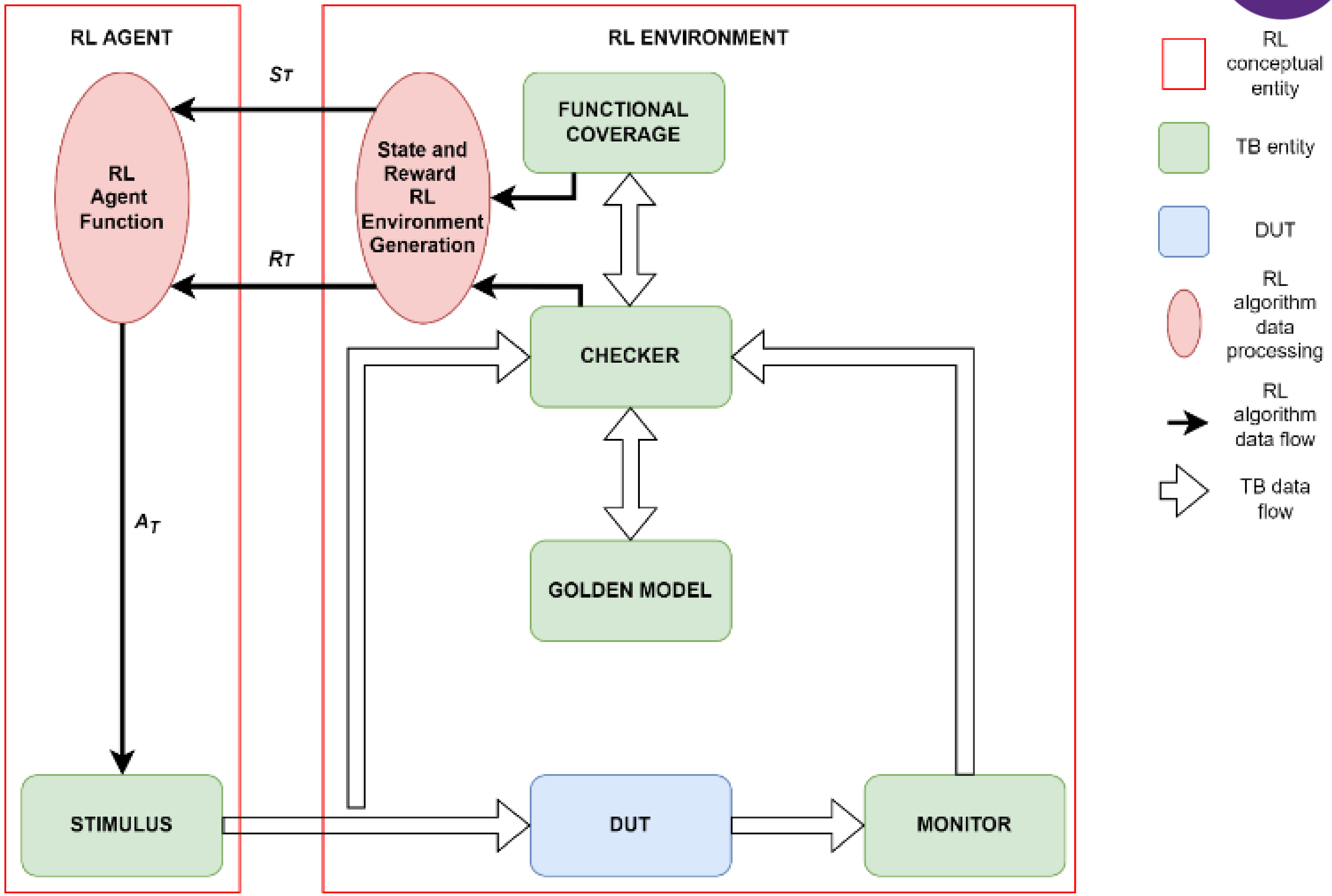
- Hitting the last functional coverage (FC) bins on an RTL design, has traditionally been an obstacle to verification closure

- In this presentation, we tap into reinforcement learning tools and techniques to assist in the simulation based constrained random coverage driven functional verification process

- Specifically, we use a DeepMind Technologies inspired Deep Q-Learning (DQN) agent to target a functional coverage category reluctant to the standard constrained random verification techniques

RTL Verification and Reinforcement Learning

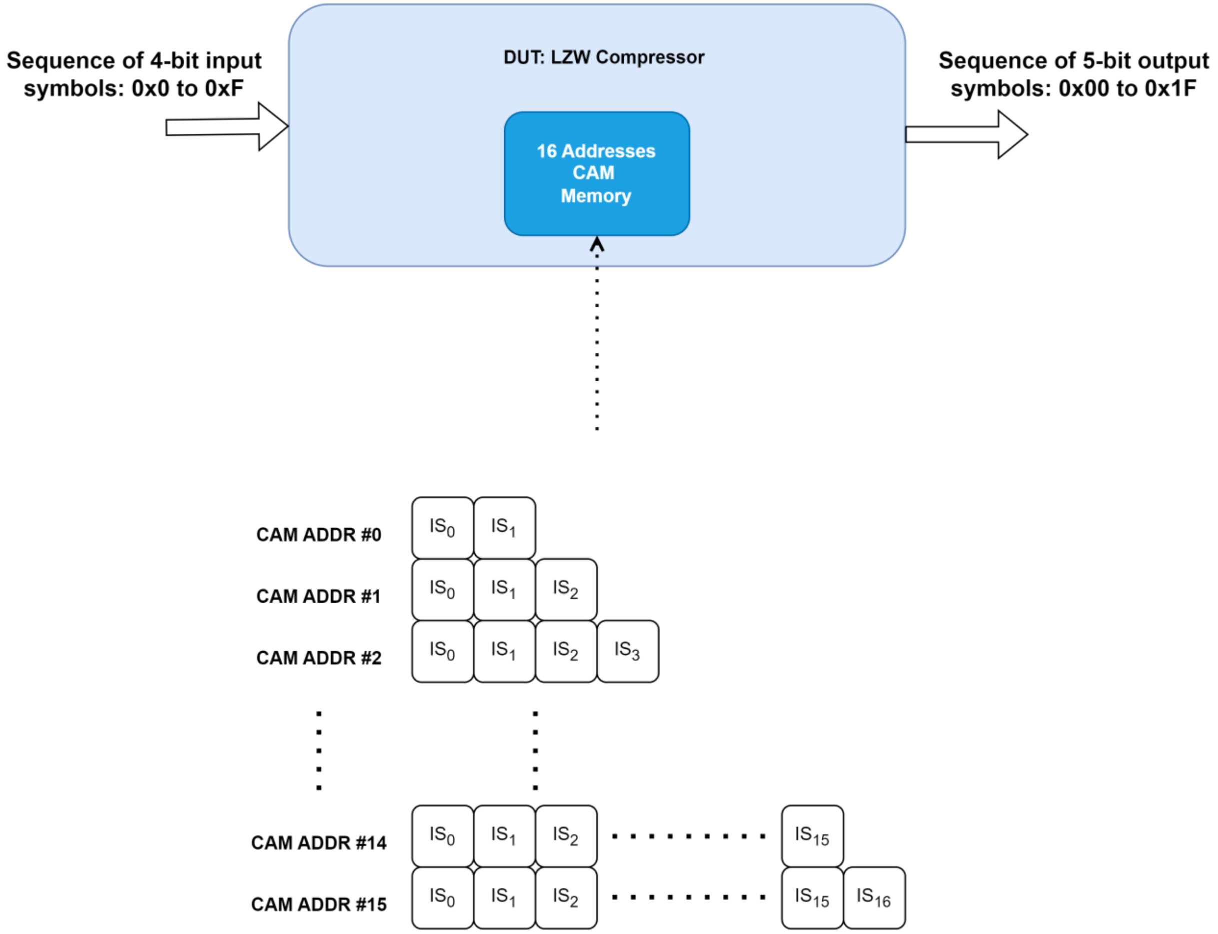
- A Reinforcement Learning (RL) system is a sequential interaction between an agent and an environment
- At every iteration, the agent processes a state and a reward value from the environment, then issues back an action to the environment
- Action <> Transaction
- Reward <> FC bins hits/misses, the harder to hit the FC, the higher the RL reward
- State <> Some representation of how we reached the current FC state (Markov Decision Process)



The LZW Compression Encoder Functional Coverage Problem

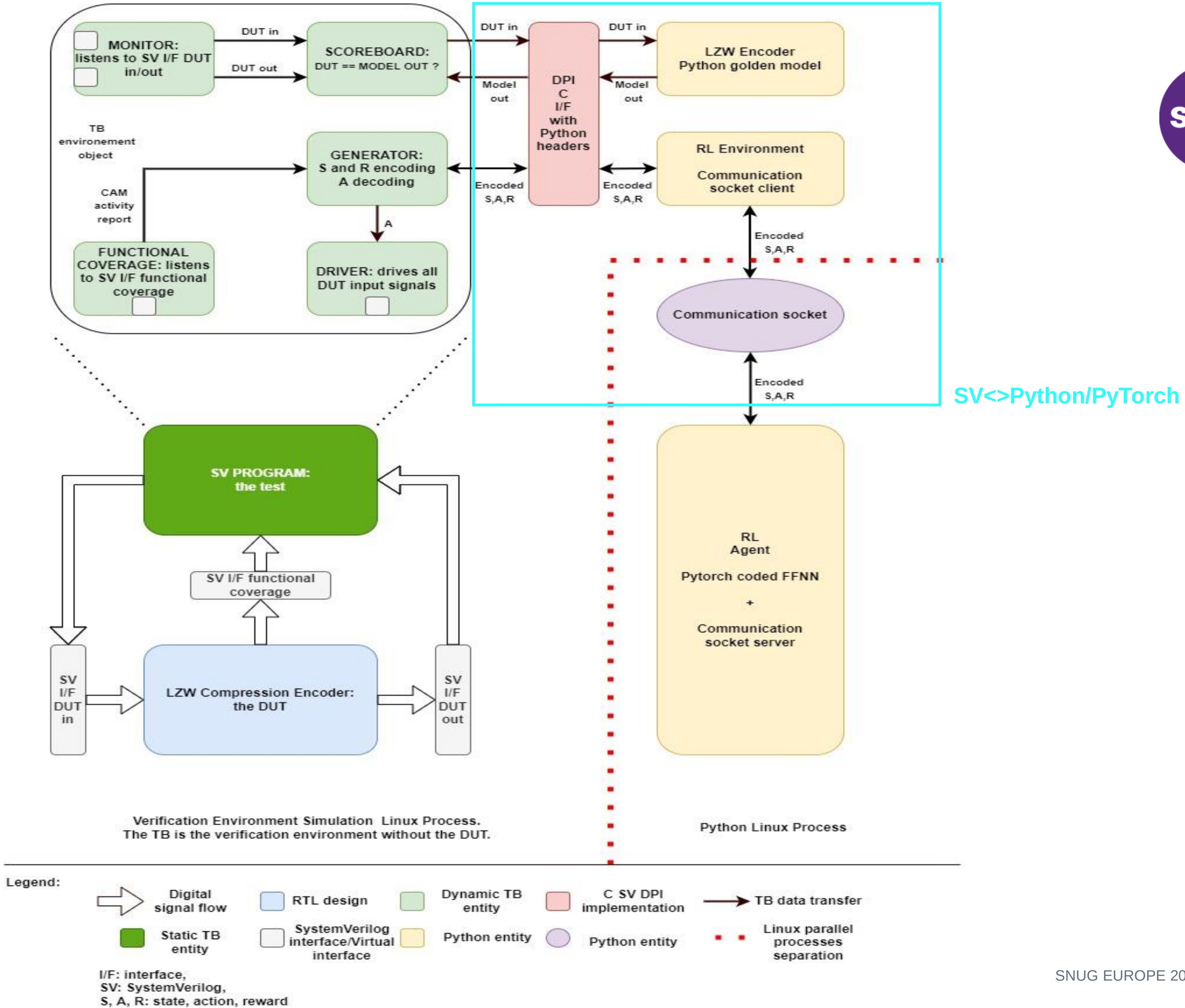
Timestep	Input Symbol 4-bit HEX	CAM[address]	Output Symbol 5-bit HEX
Start: #1	A	-	-
#2	B	CAM[0] = AB	0A
#3	A	CAM[1] = BA	0B
#4	B	Match on CAM[0]	-
#5	A	CAM[2] = ABA	10
#6	B	Match on CAM[0]	-
End: #7	A	Match on CAM[2]	12

- The shortest sequence possible is of 2 input symbols. There are 136 CAM write FC bins to cover, out of 152 CAM locations
- The CAM write functional coverage category necessitates very specific sequences!
- It is virtually impossible to reach them randomly!
- Can our DQN agent help?

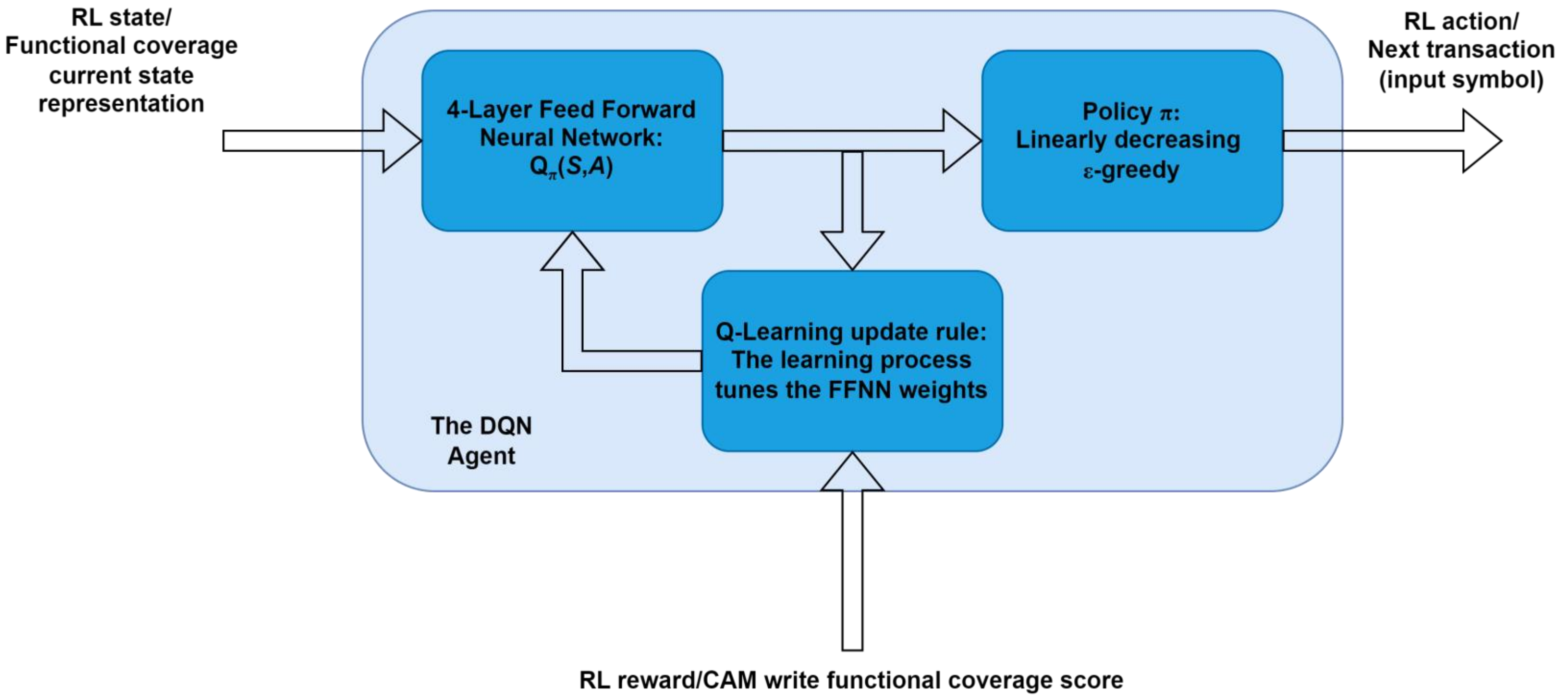


Co-simulating a SystemVerilog Test Bench and a PyTorch Agent

- Our RL agent runs in Python and PyTorch
- Our SV Design & Verification environment run on a digital simulator like VCS
- Using SV DPI/C/C Embedded Python and a client/server networking protocol, both can communicate efficiently!



The Deep Q-Learning Agent (DQN)



See Appendix for the DQN Agent Main Equations!

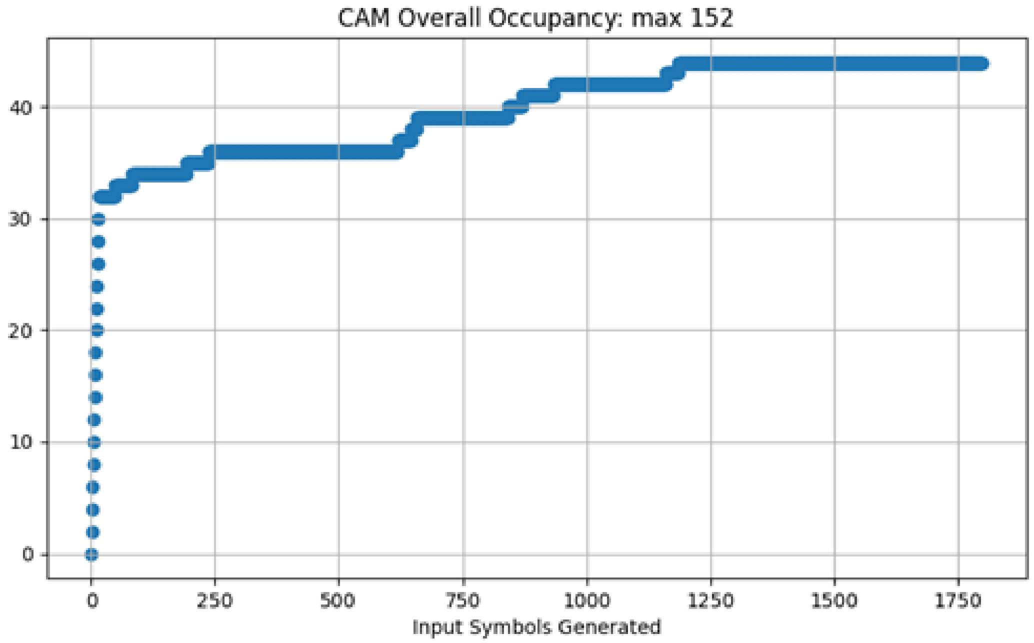
Simulation Results: Standard Approach Vs. DQN

Standard Simulation

- By running a uniform input symbols distribution, over many episodes, where an episode starts with an empty CAM and ends with a full CAM.

- We have managed to hit 28 CAM write bins out of 136 with a CAM overall occupancy of 44 out of 152

29% CAM write FC



- We first run 500 episodes with an ε-greedy linearly decreasing

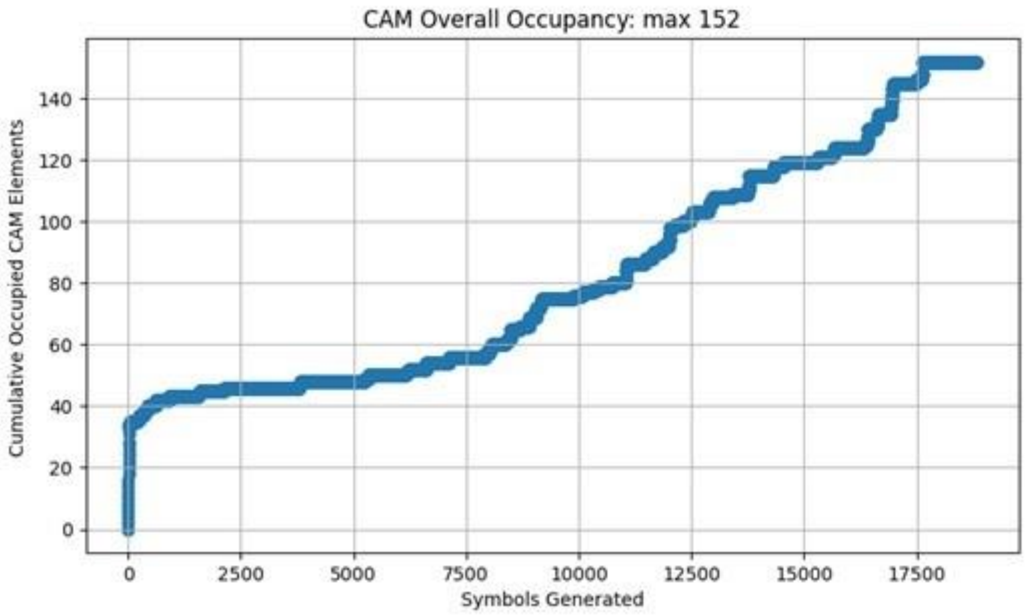
- We observe a constant incremental increase in the CAM overall occupancy

- We have managed to hit 133 CAM write bins out of 136 with a CAM overall occupancy of 152 out of 152
- To target the 3 remaining CAM write FC bins, we run 750 episodes, to allow a smoother transition from exploration to exploitation

- By merging both DQN simulations, we reach 100% CAM write functional coverage

97.8% CAM write FC

DQN Simulation



Conclusion

- We identified a functional coverage category which is hard to fully cover using standard means: the CAM write functional coverage for the LZW compression encoder

- We defined an action-value function for a DQN agent, linking between input symbols and the expected future rewards expressed as CAM write functional coverage bins hits

- We used a simple ε–greedy policy allowing a transition from exploration (full randomness) to exploitation (using reinforcement learning lessons) to reach 100% functional coverage

THANK YOU

Our
Technology,
Your
Innovation™

Appendix

- A Deep Q-Networks (DQN) agent uses a neural network to model an action-value function

- Our action-value function called $\mathbf{Q_\pi(\mathit{S},\mathit{A})}$ processes the environment state $\boldsymbol{S}$ and issues an output vector value representing the expected future reward $\boldsymbol{R}$ for every action $\boldsymbol{A}$ called $\mathbf{E}(\boldsymbol{R}|\boldsymbol{A},\boldsymbol{S}, \boldsymbol{\pi})$

- In the verification realm, it just means that, given the current functional coverage state, every input symbol we can chose for the next transaction, has a particular impact on the CAM write functional coverage overall score

- $\boldsymbol{\pi}$ is called a policy and is just a way of selecting the next transaction from the output vector: $\mathbf{E}(\boldsymbol{R}|\boldsymbol{A},\boldsymbol{S}, \boldsymbol{\pi})$

- See details in generic paper: https://www.researchgate.net/publication/369187045_Closing_Functional_Coverage_With_Deep_Reinforcement_Learning_A_Compression_Encoder_Example