DRAMPyML

A Formal Description of DRAM Protocols with Timed Petri Nets

Derek Christ, Thomas Zimmermann, Philippe Barbie, Dmitri Saberi, Yao Yin, Matthias Jung

Julius-Maximilians-Universität Würzburg, Fraunhofer IESE, Normal Computing

2025-10-15

JEDEC DRAM Standards

Approximately every year one new standard!

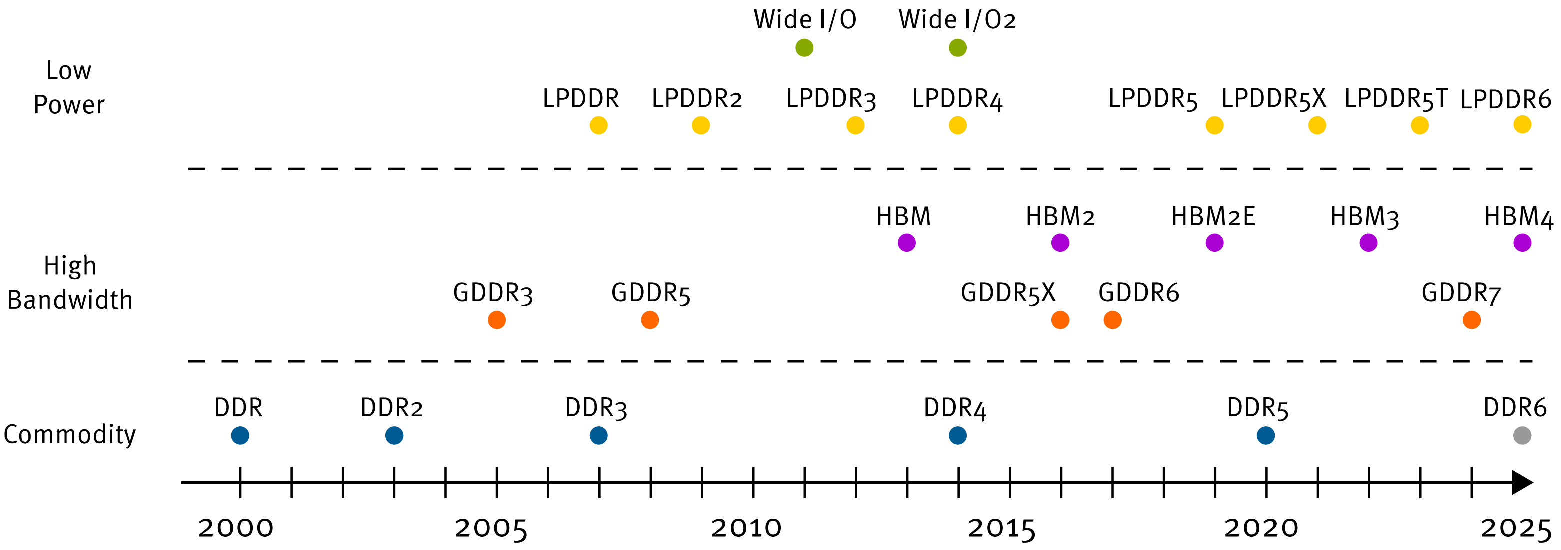


Figure 1: JEDEC standard releases over time [1].

[1] M. Jung, K. Kraft, T. Soliman, C. Sudarshan, C. Weis, and N. Wehn, "Fast Validation of DRAM Protocols with Timed Petri Nets," in *Proceedings of the International Symposium on Memory Systems*, Washington District of Columbia USA: ACM, Sep. 2019, pp. 133–147. doi: 10.1145/3357526.3357556.

JEDEC DRAM Standards

- Defines commands, states, timings and interface properties
- Protocols get more and more complex
  - DDR5: 522 pages
  - LPDDR6: 586 pages
- Descriptions are not formal and hard to understand

JEDEC defines simplified state machines for singular banks, but...

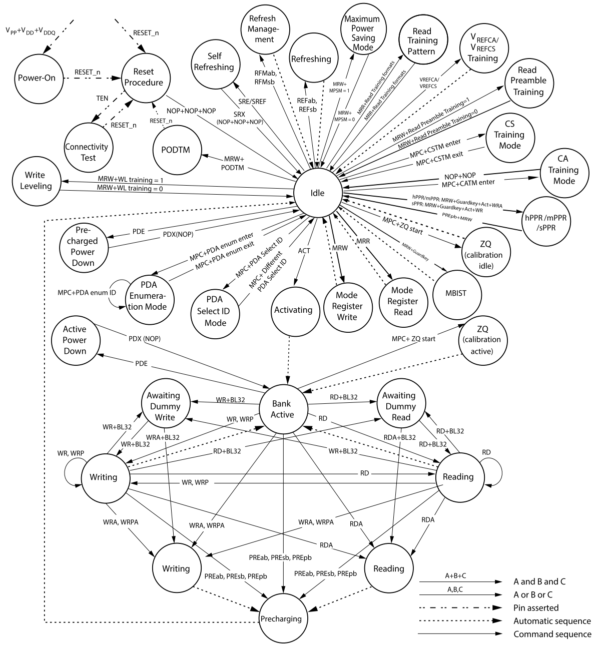


Figure 2: DDR5 state machine of a single bank defined in the JEDEC standard [1].

[1]JEDEC Solid State Technology Association, “JESD79-5C DDR5 SDRAM.” Apr. 2024.

JEDEC DRAM Standards

JEDEC defines simplified state machines for singular banks, but...

❌ Parallel Operation

❌ Timing Dependencies

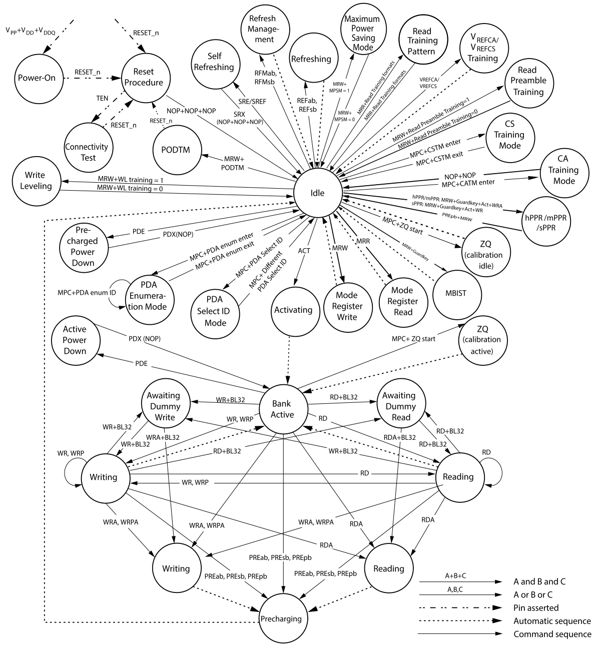
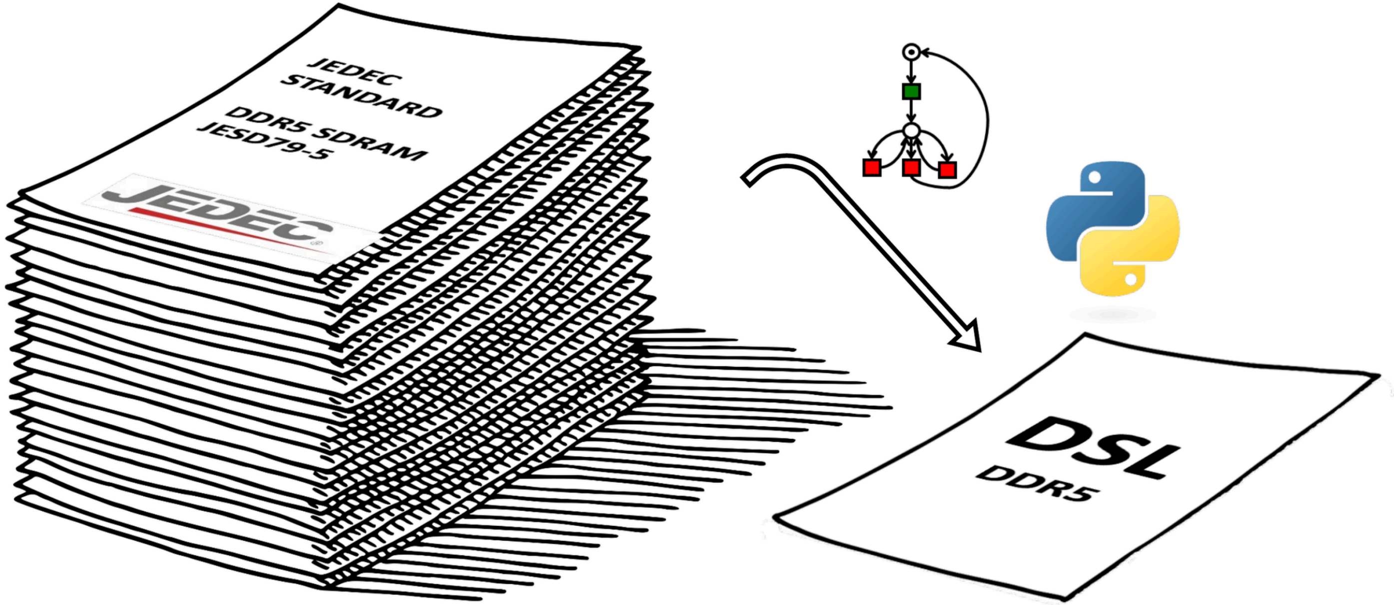


Figure 2: DDR5 state machine of a single bank defined in the JEDEC standard [1].

[1] JEDEC Solid State Technology Association, “JESD79-5C DDR5 SDRAM.” Apr. 2024.

Simplified and Formal Description

What is needed is a simple and formal representation of the specification:



Petri Nets

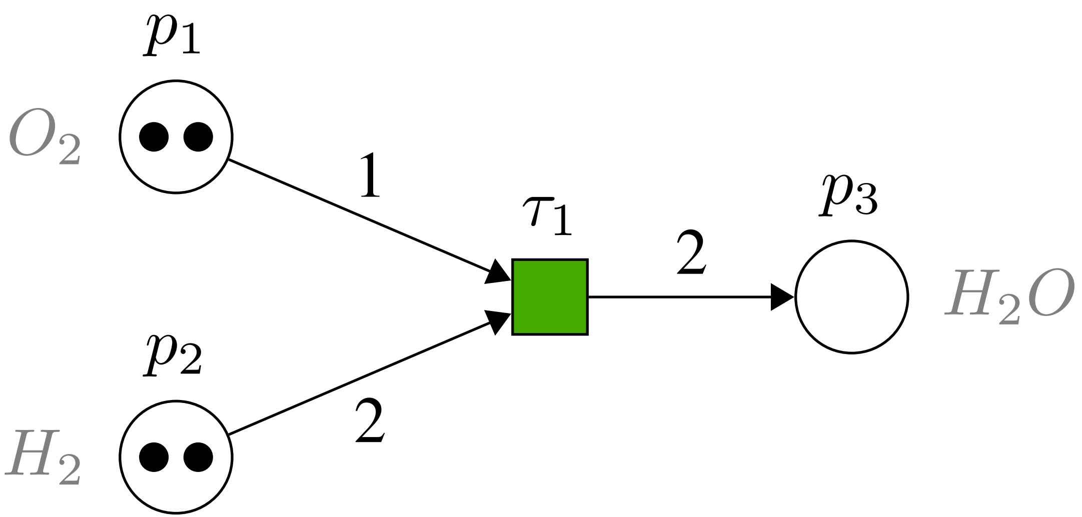


Figure 3: Petri net example of a $2H_2 + O_2 \to 2H_2O$ reaction [1].

[1] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989, doi: 10.1109/5.24143.

Petri Nets

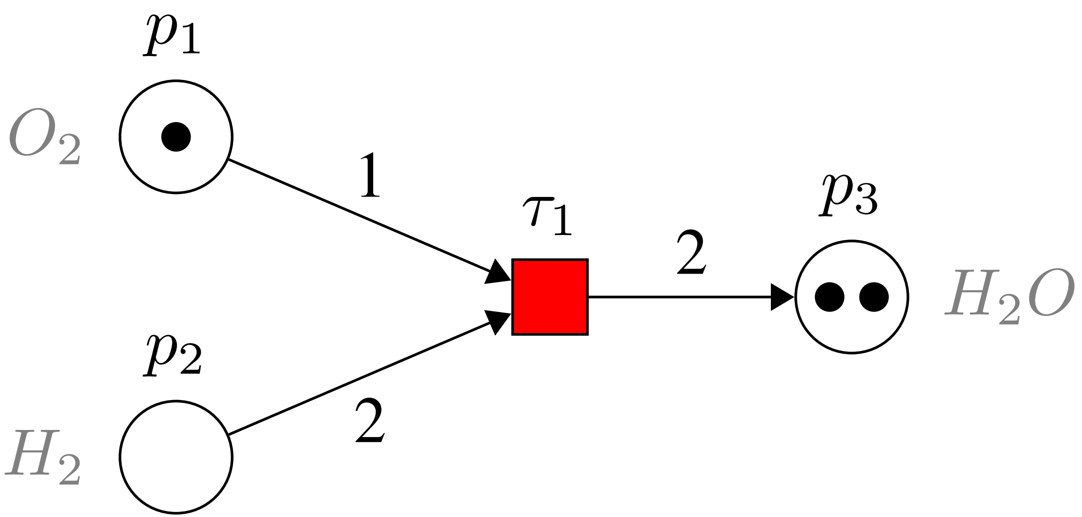


Figure 3: Petri net example of a $2H_2 + O_2 \rightarrow 2H_2O$ reaction [1].

[1] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989, doi: 10.1109/5.24143.

DRAM States and Commands

Table 1: DRAM States

Type	State	Explanation
Normal	*ACTIVE*	At minimum one bank is active, no powerdown (`cke=1`), no internal refresh.
	IDLE	All banks are closed and precharged, no power-down (`cke=1`), no internal refresh.
Power-Saving	*PDN*	*Power-Down*: Banks are active or precharged and no internal refresh (`cke=0`).
	SREF	*Self-Refresh*: All banks are precharged, the DRAM internal self-timed refresh is triggered (`cke=0`).

Table 2: DRAM Commands

Type	Command	Explanation
Row	`ACT`	*Activate*: A specific row in one bank is activated.
	`PRE`	*Precharge*: The currently activated row is closed and the bank is precharged.
Column	`RD/RDA`	*Read/Read with Auto-Precharge*: Read from an activated row (and precharge afterwards).
	`WR/WRA`	*Write/Write with Auto-Precharge*: Write to an activated row (and precharge afterwards).
Entire DRAM	`PREA`	*Precharge All*: All active banks are precharged.
	`REFA`	*Auto-Refresh*: Refresh one or more rows in all banks.
	`PDE/PDX`	*Power-Down Entry/Exit*: Enters or exits the PDN mode.
	`SRE/SRX`	*Self-Refresh Entry/Exit*: Enters or exits the SREF mode.

Formal Description of DRAMs Using Petri Nets

Additions to standard Petri nets:

- Inhibitor Arcs (—●)
- Reset Arcs (→→)

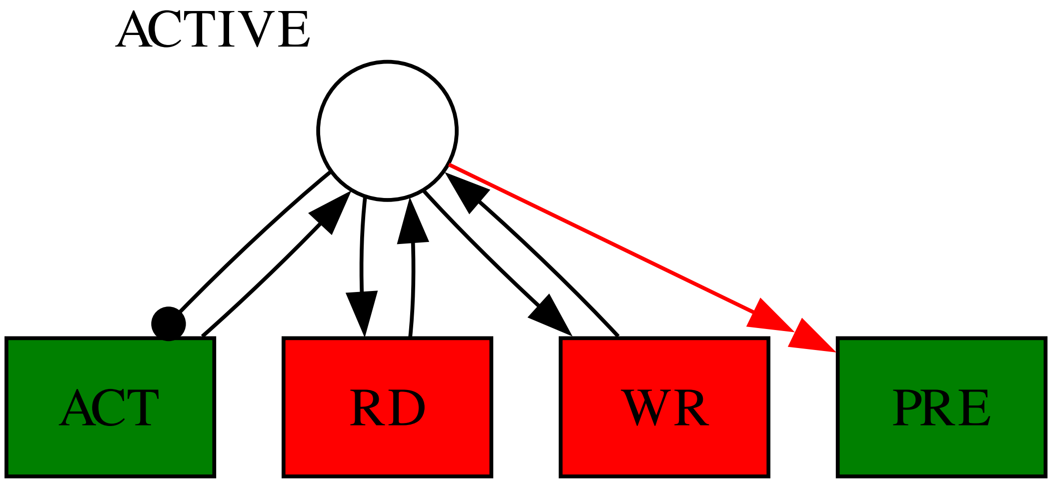


Figure 4: Simplified Petri net of a single DRAM bank.

Formal Description of DRAMs Using Petri Nets

Additions to standard Petri nets:

- Inhibitor Arcs ( )
- Reset Arcs ( )

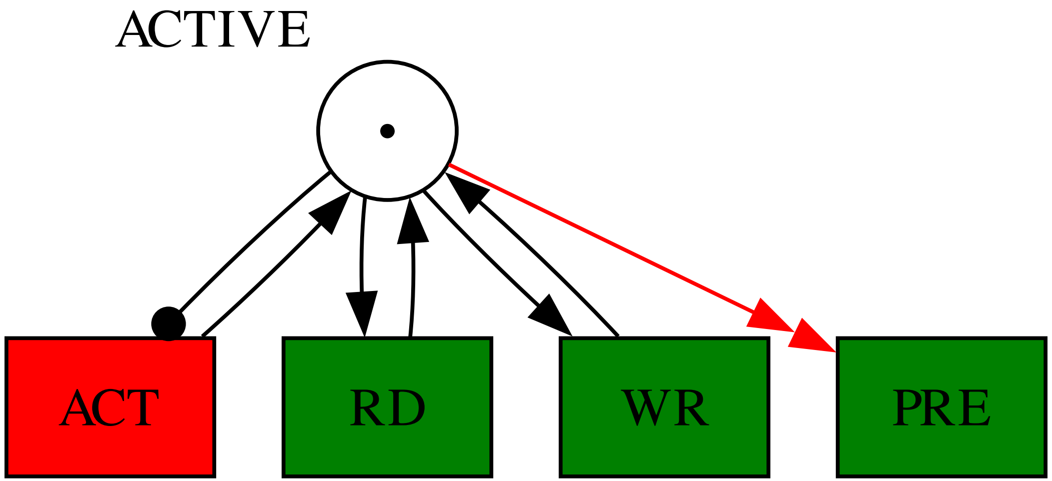


Figure 4: Simplified Petri net of a single DRAM bank.

DDR3 Petri Net

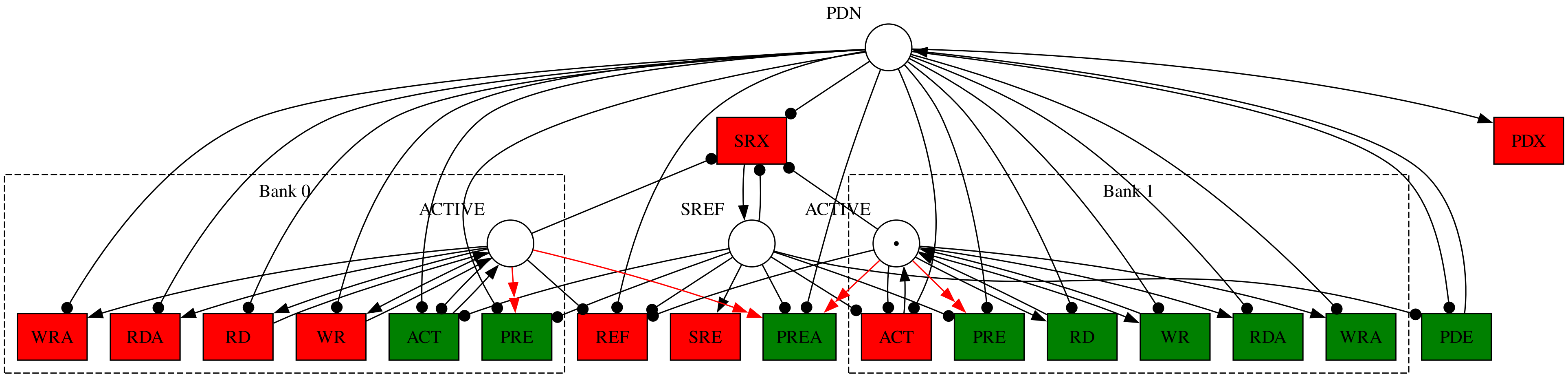


Figure 5: Petri net of a two-bank DDR3-like DRAM.

DRAMPyML

Listing 1: Creation of the Petri net for DDR3 in Python code.

```python
def create_petri_net(memspec: dict[Expr, int]) -> PetriNet:
    graph = rx.PyDiGraph()
    p = Parameters()

    for rank in range(memspec["nbrOfRanks"]):
        rank_coord = Coordinate(rank=rank, bank=None)
        t_refab = graph.add_node(Transition(REF, coordinate=rank_coord))
        t_preab = graph.add_node(Transition(PREA, coordinate=rank_coord))
        # ...
        for bank in range(memspec["nbrOfBanks"]):
          bank_coord = Coordinate(rank=rank, bank=bank)
          p_active = graph.add_node(
            Place(PlaceType.ACTIVE, coordinate=bank_coord))
          t_act = graph.add_node(Transition(ACT, coordinate=bank_coord))
          # ...
```

Timing Dependencies

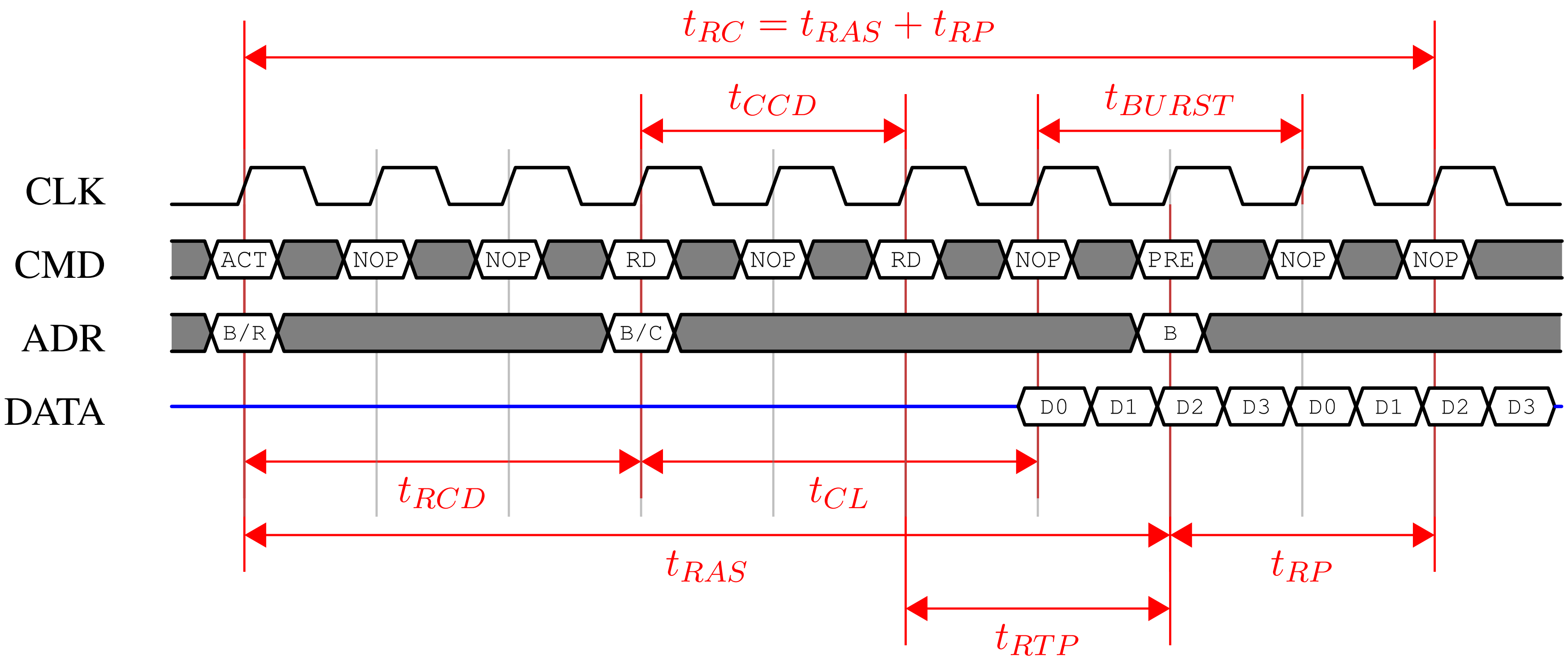


Figure 6: Timing Dependencies between Commands.

Timing Dependencies

Table 3: DRAM Timing Parameters

Name	Explanation
t_{CK}	DRAM Clock: The clock of the DRAM interface.
t_{RCD}	Row-to-Column Delay: The time interval between row access and data ready.
t_{CCD}	Column-to-Column Delay: The minimum column command timing.
t_{RC}	Row Cycle: The time between two ACT commands to the same bank.
t_{RRD}	Row-to-Row Delay: The time between two ACT commands to different banks.
t_{CL}, t_{RL}, t_{WL}	CAS Latency: Delay from a RD/WR command until data on the bus.

Timing Dependencies

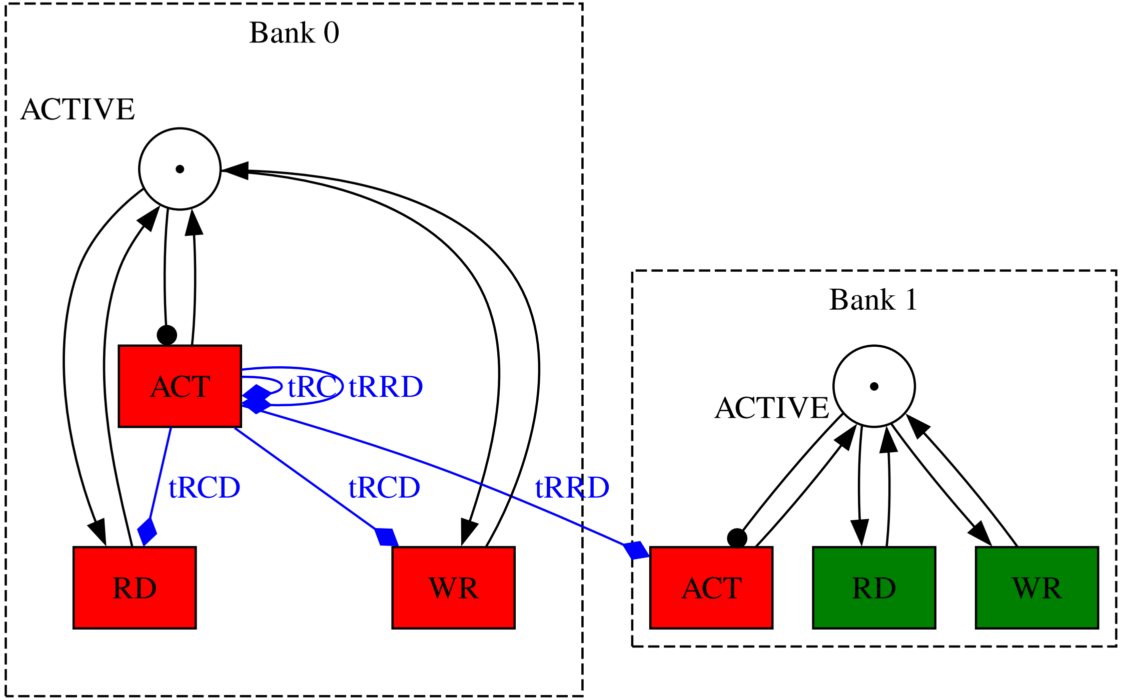
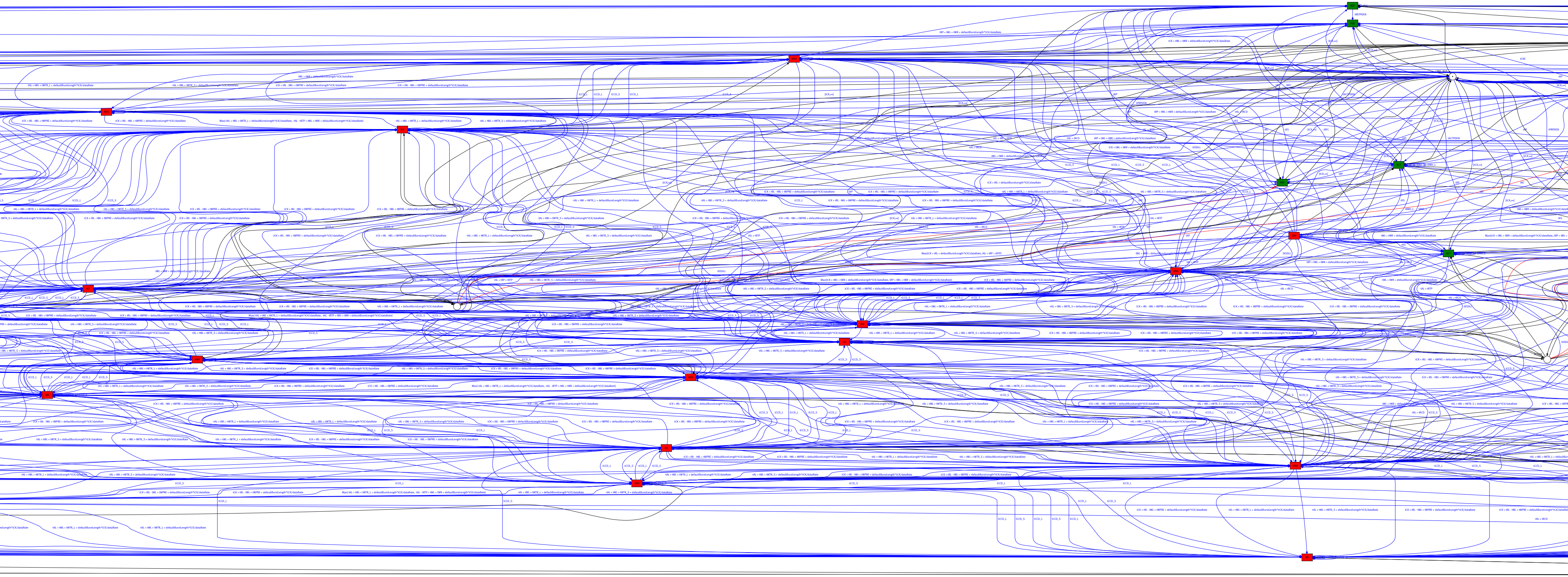


Figure 7: Petri net showing timing dependencies after firing ACT of Bank 0.

Timing Dependencies



Timing Dependencies Syntax

Listing 2: Sample DDR4 command constraints for different hierarchy levels.

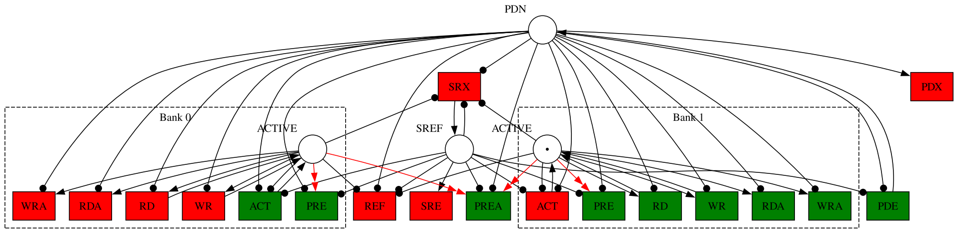
```
# Bank
CommandTimingConstraint(intra_bank, [ACT], [PRE], p.tRAS),
CommandTimingConstraint(intra_bank, [ACT], [RD, WR, RDA, WRA], p.tRCD - p.tAL),
# BankGroup
CommandTimingConstraint(intra_bank_group, [ACT], [ACT], p.tRRD_L),
CommandTimingConstraint(intra_bank_group, [RD], [RD, RDA], p.tCCD_L),
# Rank
CommandTimingConstraint(intra_rank, [ACT], [PDE], p.tACTPDEN),
CommandTimingConstraint(intra_rank, [ACT], [REF, SRE], p.tRC),
```

Listing 3: Callback function evaluating the coordinate query.

```
def intra_bank(from_coord: Coordinate, to_coord: Coordinate) -> bool:
  # ...
```

Unrolling of the Petri Net

- Executable Petri net
  - Traversal of all reachable states
    - $N(R, B) = (2^{B+1} + 1)^R$
  - Generation of minimal command traces, needed to reach these states
    - $k_{min}(R, B) = (B + 1) \cdot R$
  - Generation of all possible k-deep traces
- Recap (Figure 5):



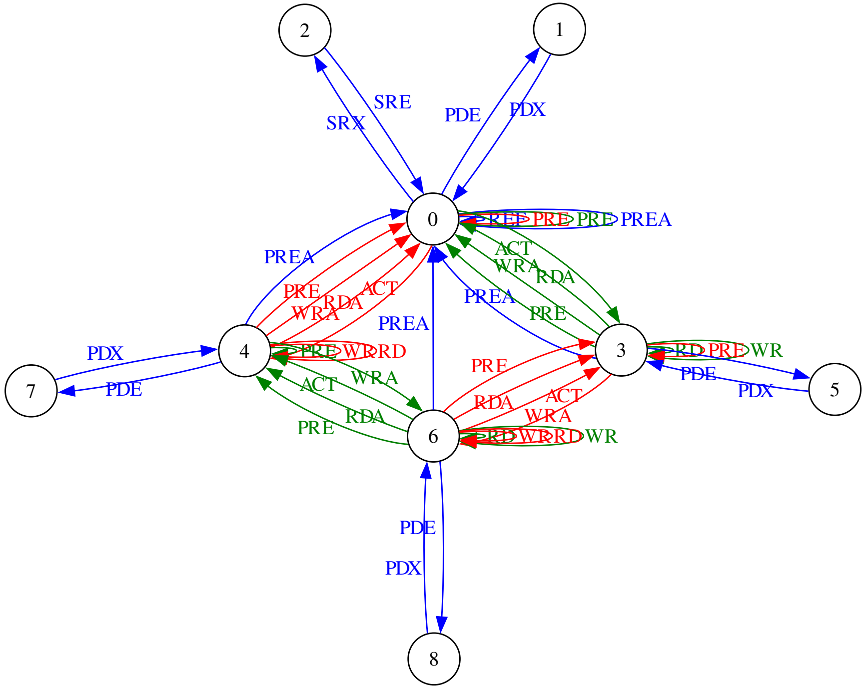


Figure 8: State machine of an unrolled Petri net. Transitions are colored whether they affect Bank 0, Bank 1 or the whole Rank.

Unrolling of the Petri Net

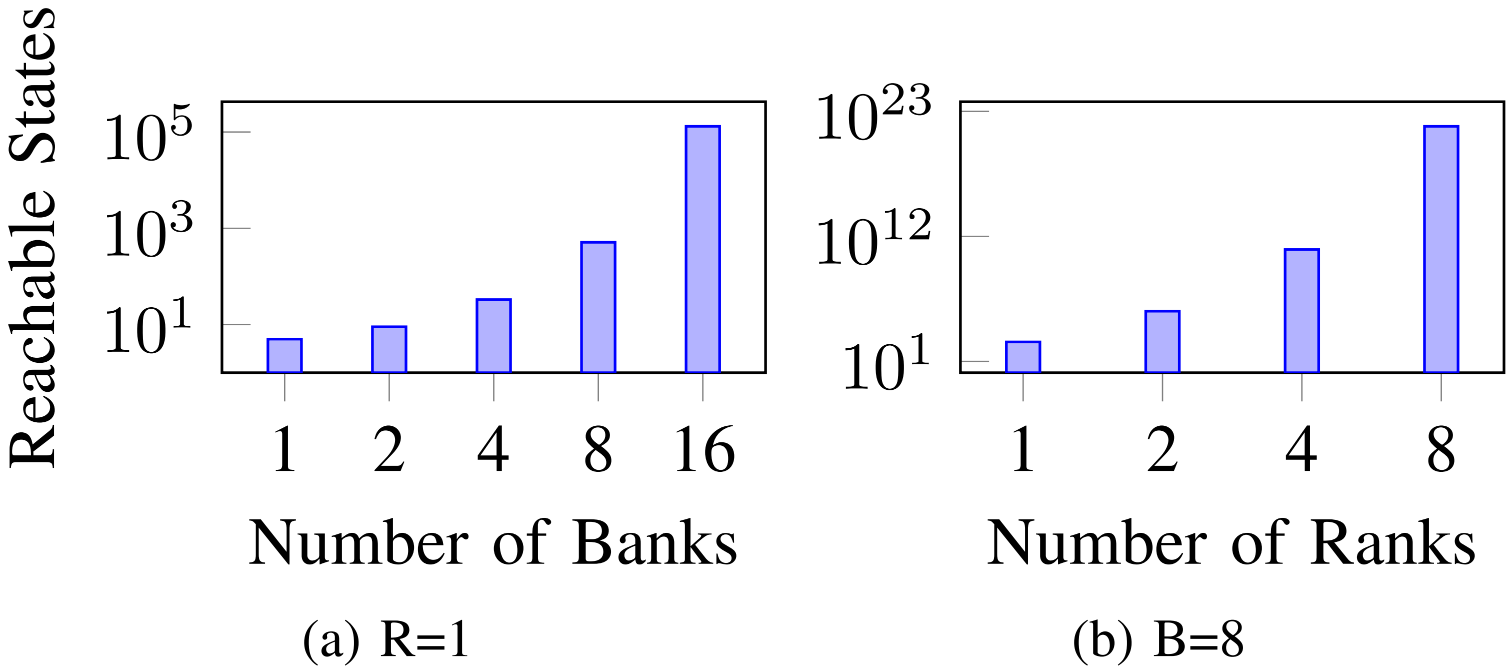


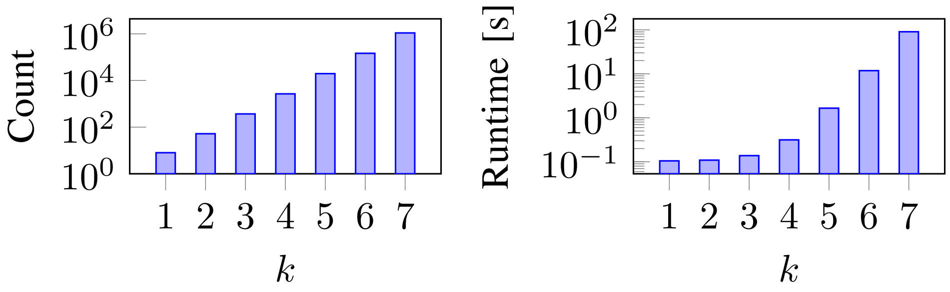
Figure 9: Total number of reachable states for different quantities of banks and ranks.

k-Deep Traces

With a known k_{min}, all possible state sequences can be generated:

Listing 4: Sample command traces with k = 3.

```
[PREA   (RA0)     ; PREA   (RA0)     ; PREA   (RA0)    ],
[PREA   (RA0)     ; PRE    (RA0BA1)  ; SRE    (RA0)    ],
[ACT    (RA0BA0)  ; ACT    (RA0BA1)  ; WR     (RA0BA0) ],
[SRE    (RA0)     ; SRX    (RA0)     ; PDE    (RA0)    ],
[ACT    (RA0BA0)  ; PREA   (RA0)     ; ACT    (RA0BA0) ],
[ACT    (RA0BA0)  ; ACT    (RA0BA1)  ; PDE    (RA0)    ],
                              ...
```

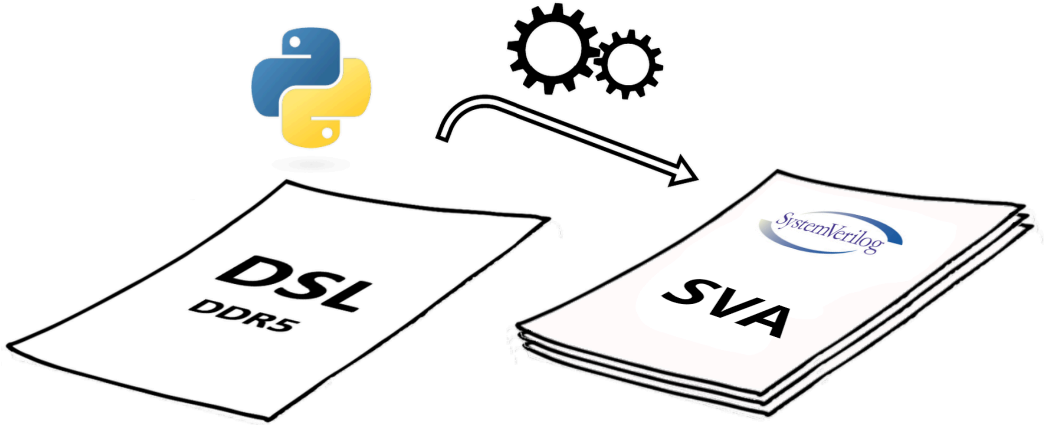


(a) Number of Valid Traces (b) Algorithm Runtime

Figure 10: Number of valid command traces for the depth k and runtime of the exploration algorithm.

Source Code Generation

SystemVerilog Assertions



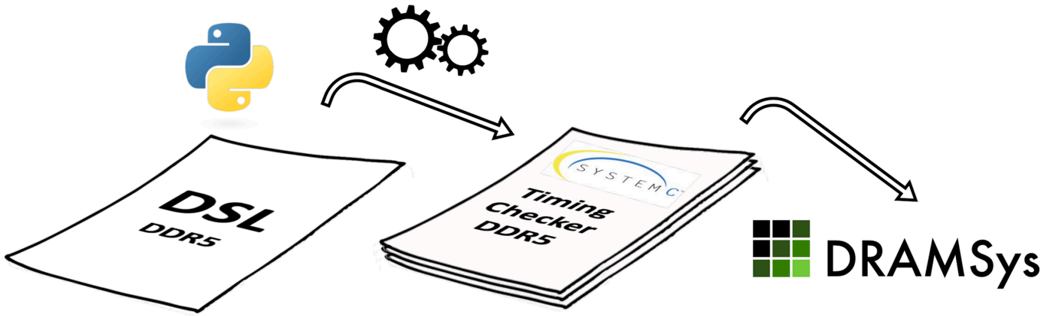
Listing 5: SVA property of an ACT [1].

```
property arc_active_act;
    @(posedge clk) disable iff (reset)
        (cmd == act) |-> (active >= 1'b1);
end property;
assert property(arc_active_act);
```

[1] L. Steiner, C. Sudarshan, M. Jung, D. Stoffel, and N. Wehn, "A Framework for Formal Verification of DRAM Controllers," in *Proceedings of the 2022 International Symposium on Memory Systems*, Washington DC USA: ACM, Oct. 2022, pp. 1–7. doi: 10.1145/3565053.3565059.

Source Code Generation

DRAMSys [1] Timing Checker



Listing 6: DRAMSys timing constraint for tRCD.

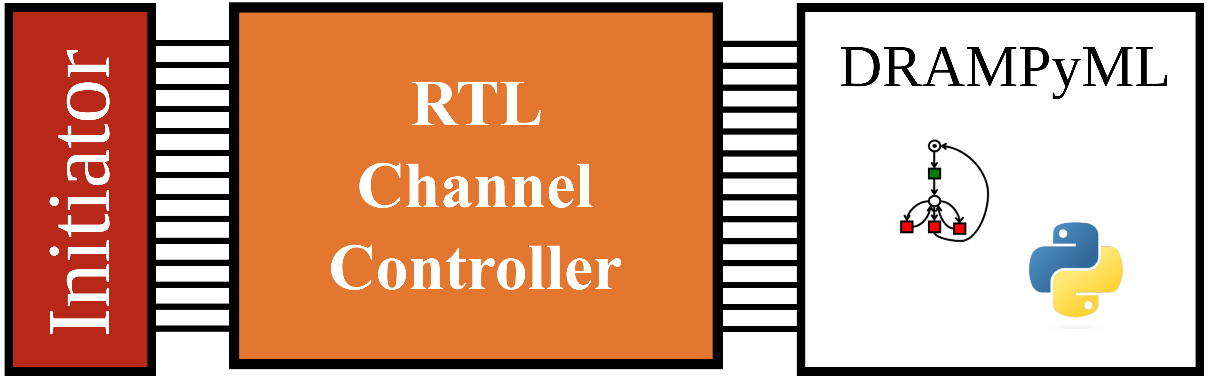
```
sc_time c = sc_time_stamp() + tRCD;
sc_time &nextPossibleTime = nextCommandByBank[RD][bank];
nextPossibleTime = std::max(nextPossibleTime, c);
```

[1] L. Steiner, M. Jung, F. S. Prado, K. Bykov, and N. Wehn, “DRAMSys4.0: An Open-Source Simulation Framework for In-depth DRAM Analyses,” *International Journal of Parallel Programming*, vol. 50, no. 2, pp. 217–242, Apr. 2022, doi: 10.1007/s10766-022-00727-4.

Source Code Generation

Verification of RTL models for protocol compliance

UVM



Conclusion and Future Work

- New Python-based formal modeling of DRAM protocols using Petri nets.
  - Modeling of functional and timing behavior.
- Executable model for analyzing traits of the protocol and verification.
- Source code generation by traversing the graph.

- Wrapping of DRAMPyML in a domain-specific language (DSL).
  - New standard for describing DRAM protocols.
- Modeling a large set of JEDEC DRAM standards.
  - DDR5, LPDDR5/6, HBM3/4

Thank you for your attention!
Questions?