XDF XILINX DEVELOPER FORUM

F1 Acceleration for Montecarlo: financial algorithms on FPGA

Presented By

Liang Ma, Luciano Lavagno

Dec 10th 2018

XILINX

Contents

- Financial problems and mathematical models
- High level synthesis
- Optimization
- Accelerators on Amazon web services

Financial products

- Bank deposit
- Various debits
- Bond
- Stocks
- Derivatives
- …

Options

- **A contract**
  - A right to buy or sell an instrument at a given price on certain date in the future
  - Expiration date T
  - Stock price, $S(t),\ t \in (0, T)$
  - Strike price, K, specified in the contract
- **Call option**
  - To buy the instrument at K
- **Put option**
  - To sell the instrument at K

Options

- **Execution**
  - Call option
    - To buy the instrument at K
  - Put option
    - To sell the instrument at
  - Example
    - Take call option as an example
    - if $S(T) > K$
    - Profit: $S(T) - K$
    - else
    - Profit: 0

Options

- **Option types**
  - European vanilla option
    - Execute at date T
    - $P_{call} = \max(S_T - K, 0)$
    - $P_{put} = \max(K - S_T, 0)$
  - American vanilla option
    - Execute before date T
  - European barrier option
    - Execution condition: stock price must stay within the preset barriers
  - Asian option
    - Compute the payoff price with the average stock price
- **Option price**

Option pricing model

- **Black-Scholes model**
  - $dS = rSdt + \sigma Sdz$
- **Itö Lemma**
  - $S(t + \Delta t) = e^{\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\varepsilon\Delta t}$

Option pricing model

- **Heston model**
  - $dS = rSdt + \sqrt{V}Sdz_1$
  - $dV = \kappa(\theta - V)dt + \sigma_V\sqrt{V}Sdz_2$

Solution

- **Monte Carlo Method**
  - Stochastic process
    - Random numbers
  - Time dependent differential equations
    - Preset time partition: $t_0 = 0,\ t_1, t_2 \ldots, t_m \ldots, t_M = T$
    - Path simulation: $S_0 = S_{t_0}, S_1, S_2 \ldots, S_m = S_{t_m} \ldots, S_M = S_{t_M}$
    - Large amount of paths (N) in total to achieve convergent result

Solution

- **Random numbers**
  - Mersenne-Twister Algorithm
  - Box-Muller transformation

Algorithm

- **B-S model**
- **Heston model**

Algorithm 1 Monte Carlo algorithm for option pricing model

Input: parameters for the stock and option, id, N_S and M
Output: payoff price

```
Initialization:
prng(id)
opM(optionData)
for i = 1 to N_S do
    for k = 1 to M do
        U_0, U_1 ← MersenneTwist()
        ε_0, ε_1 ← BoxMuller(U_0, U_1)
        opM.OptionStatusUpdate(option[i], ε_0, ε_1)
    end for
end for
return P_payoff = mean(option[].price())
```

Performance on CPU

Model	Option	F1 CPU [s]
Blcak-Scholes Model	European vanilla option	3.56
	Asian option	3.88
Heston Model	European vanilla option	5.16
	European barrier option	1.25

Acceleration on GPU

Model	Option	Nvidia GTX 950 [ms]	Power [W]	Nvidia Tesla P100 [ms]	Power [W]
Blcak-Scholes Model	European vanilla option	11.15	84	2.4	170
	Asian option	11.17	84	2.11	170
Heston Model	European vanilla option	26.3	91	4.31	181
	European barrier option	26.13	87	4.33	180

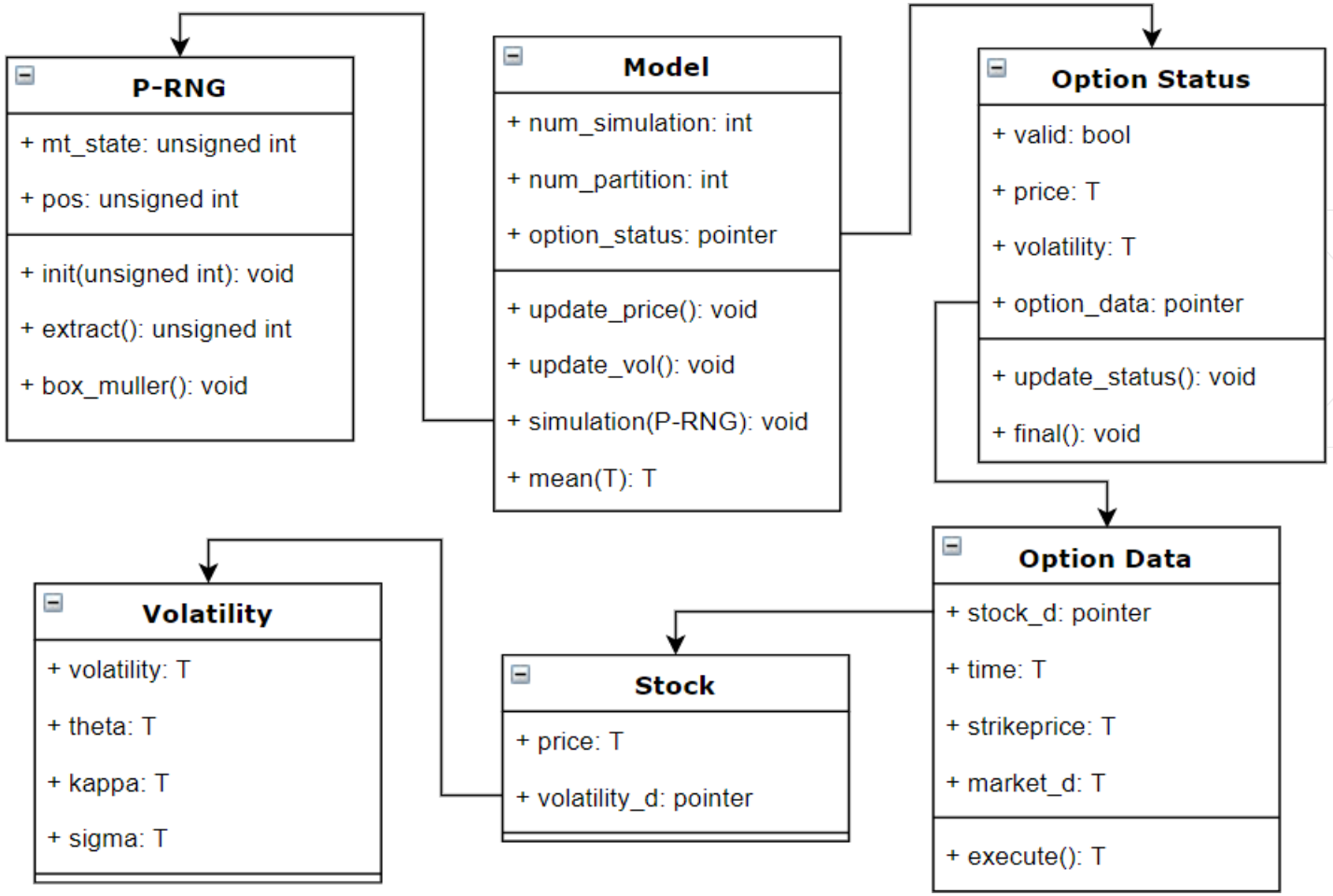
Device	Process [nm]	CUDA cores	Frequency [GHz]	Power [W]
Nvidia GTX 950	28	640	0.9	75
Nvidia Tesla P100	16	3584	1.2	250

Acceleration on FPGA

- **High level synthesis (HLS)**
  - is a design methodology at the system level or algorithm level, to design the hardware system. As the design abstraction from the gate level to RTL, the migration from RTL to HLS makes the design more productive and easy to be maintained and verified High level synthesis (HLS) is a design methodology at the system level or algorithm level, to design the hardware system. As the design abstraction from the gate level to RTL, the migration from RTL to HLS makes the design more productive and easy to be maintained and verified.

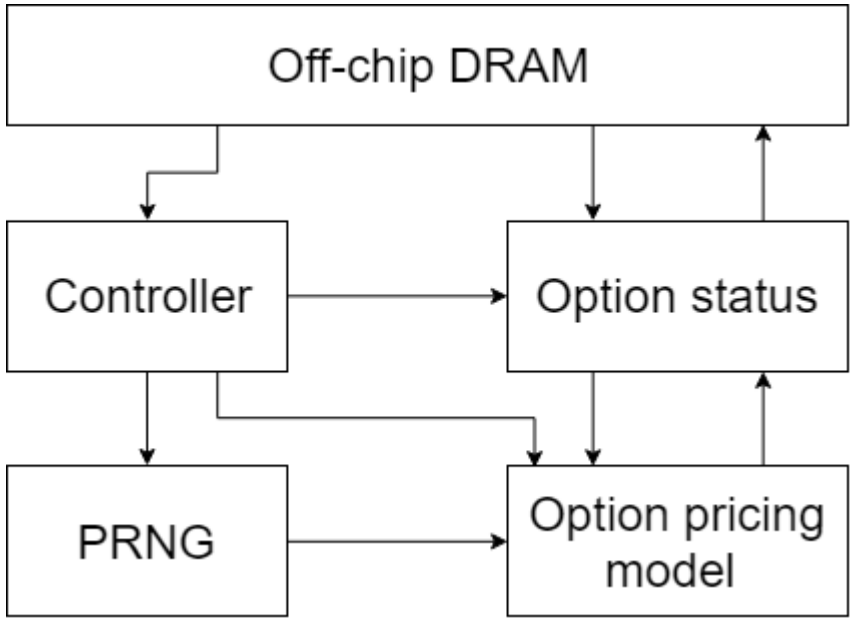
Software design

> Modular desing



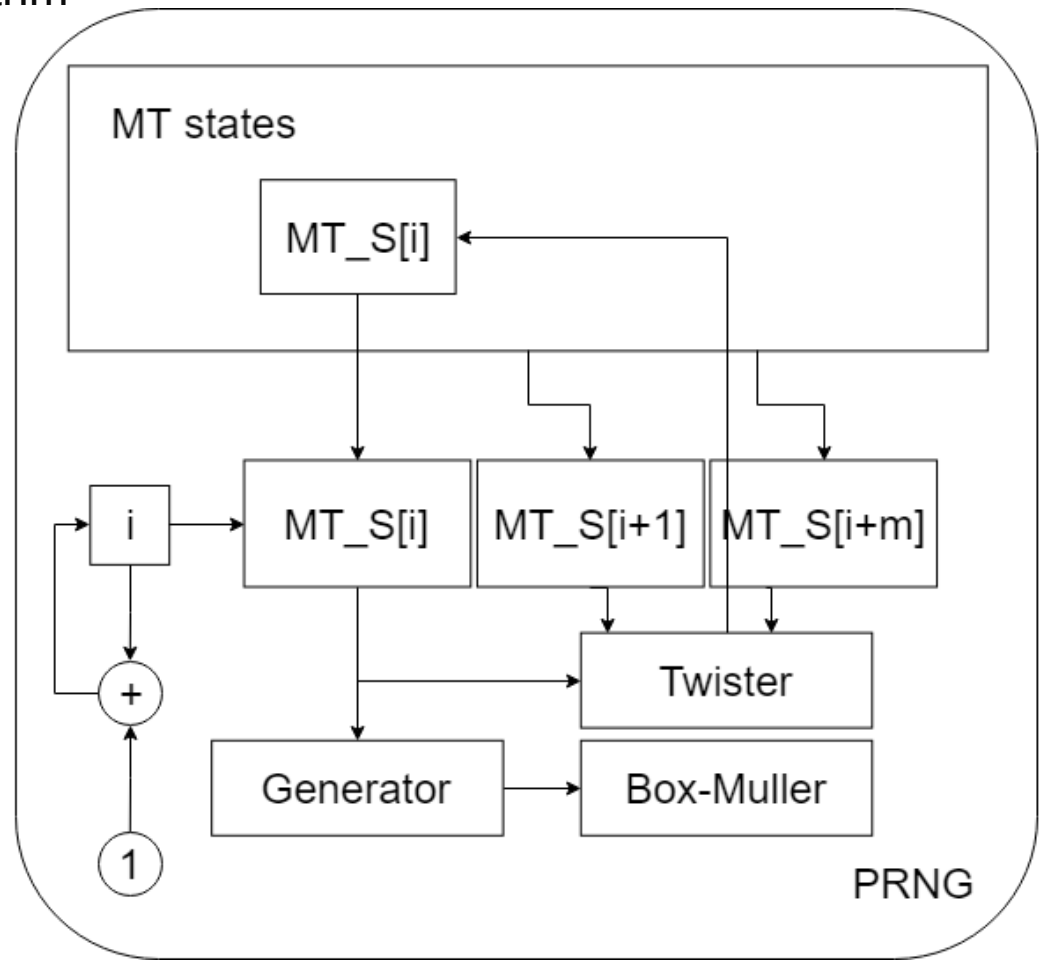
Architecture

- **General modules**
  - Datapath



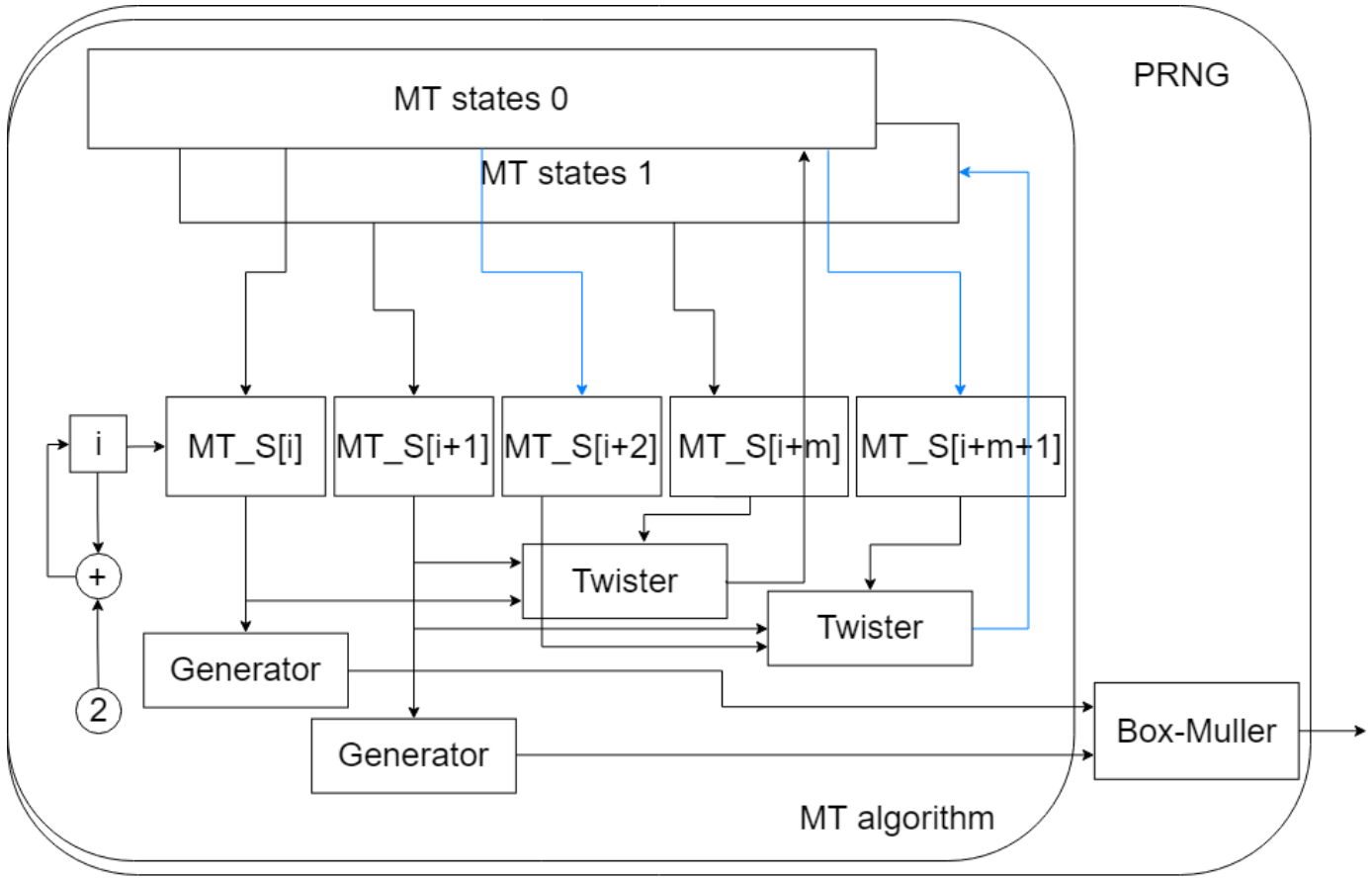
Architecture

- **PRNG**
  - Mersenne Twister algorithm



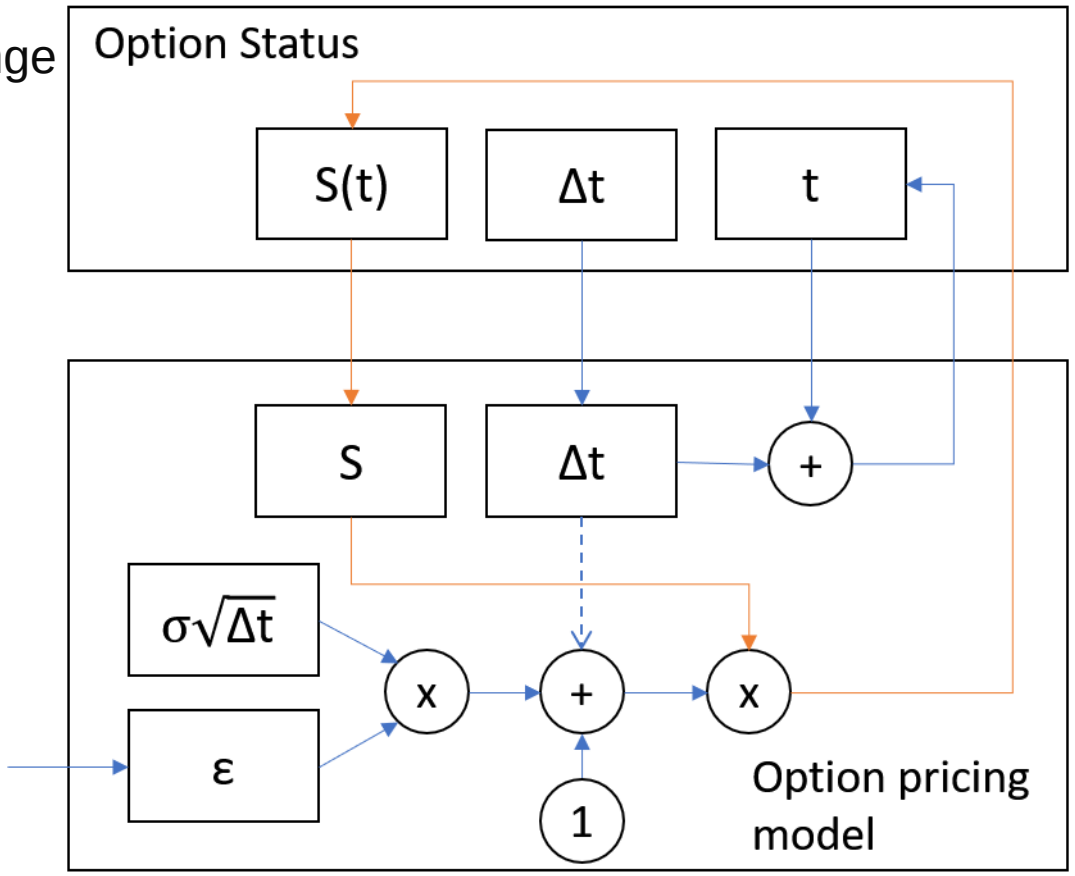
Architecture

- **Optimization on the PRNG**
  - BRAM partition



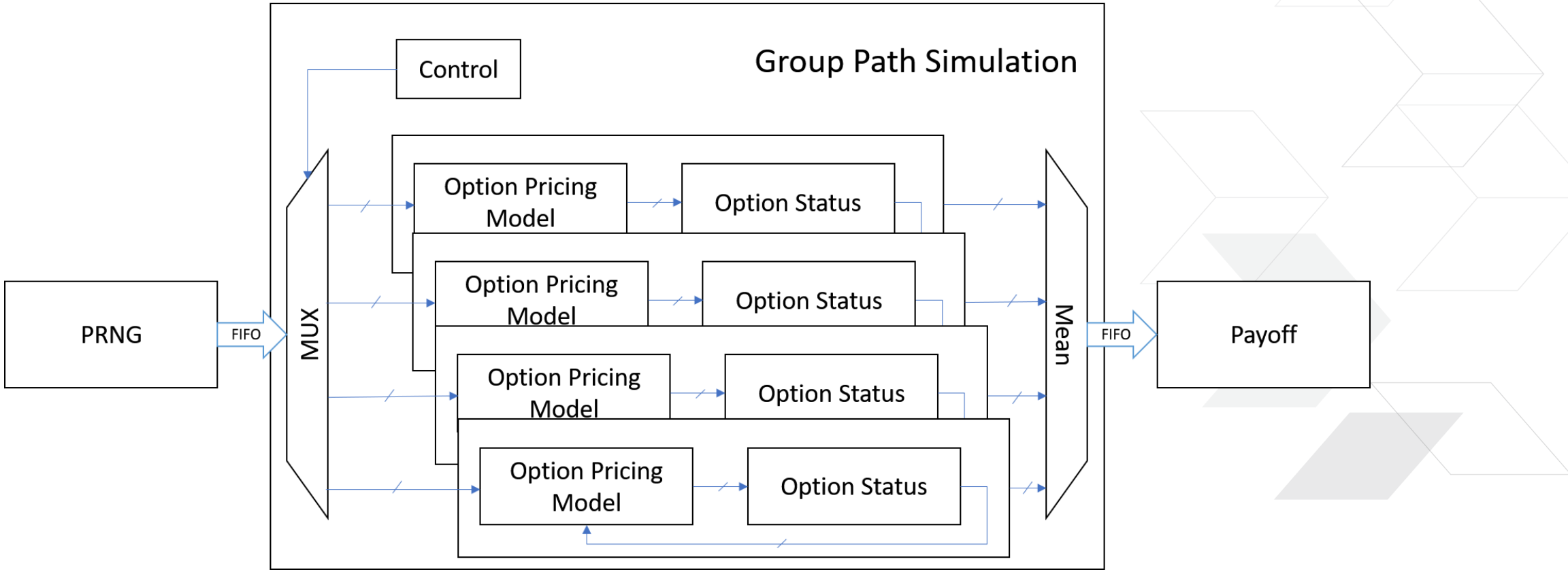
Architecture

- **Step simulation**
  - Critical cycle in orange



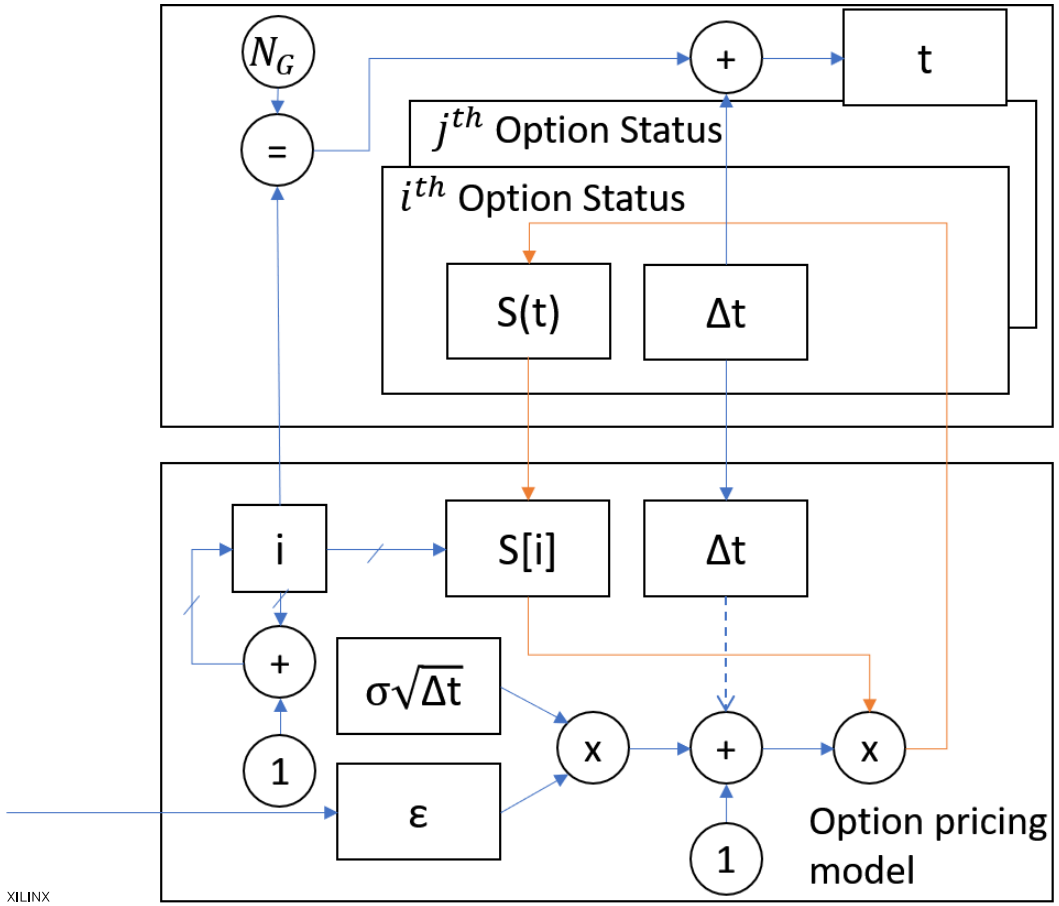
Architecture

- **Step simulation optimization**
  - Pipelining of multi-cycle step simulations



Architecture

- **Step simulation optimization**
  - Step simulation source code



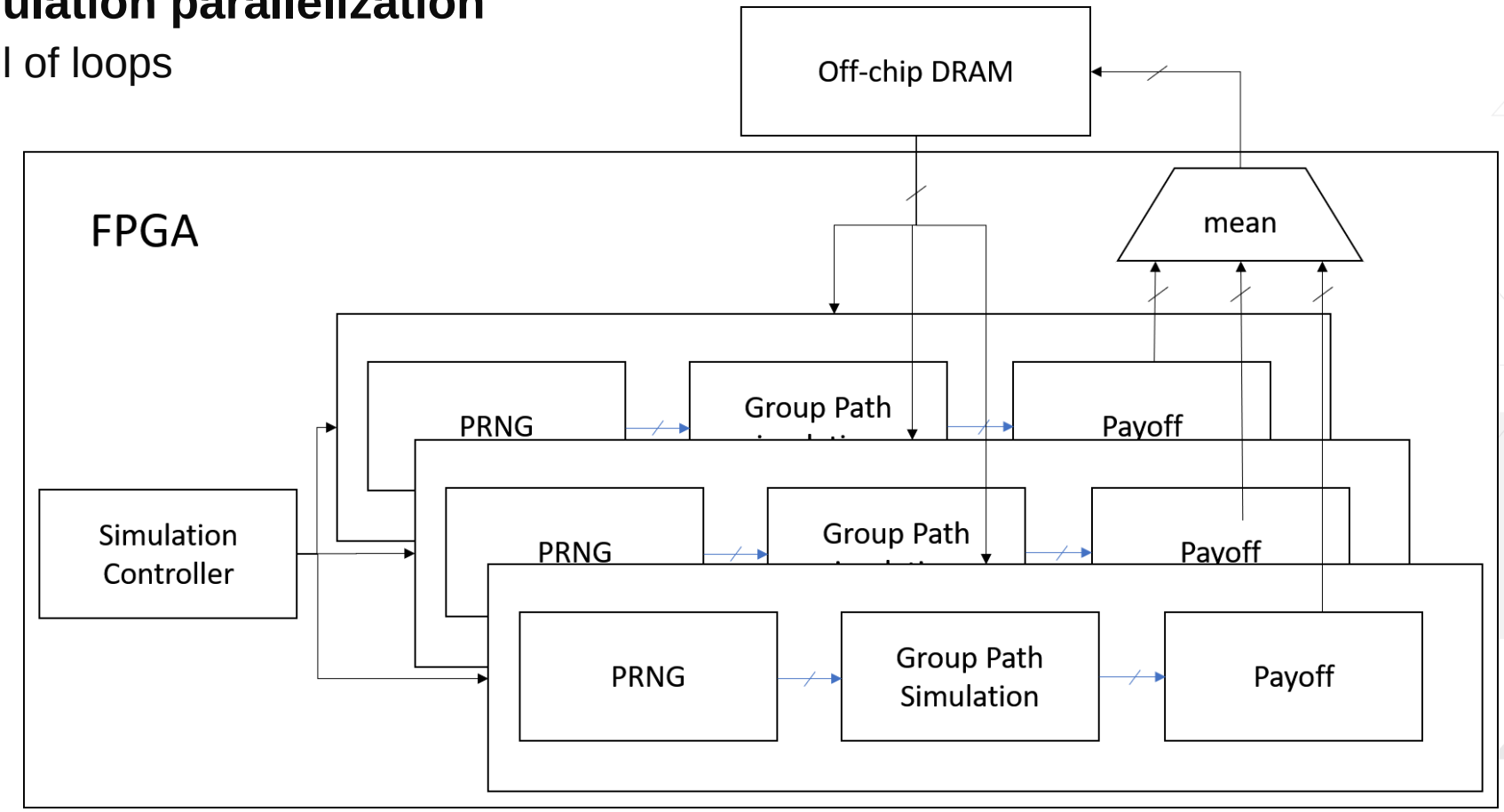
Algorithm 2 Group path simulation

Input: parameters for the stock and option, id, N_S and M
Output: payoff price

```
Initialization:
prng(id)
opM(optionData)
for i = 1 to N_S/N_G do
  for k = 1 to M do
    for g = 1 to N_G do
      this loop is pipelined
      ε_0, ε_1 ← prng.generate()
      opM.OptionStatusUpdate(option[i, g], ε_0, ε_1)
    end for
  end for
end for
return P_payoff = mean(option[].price())
```

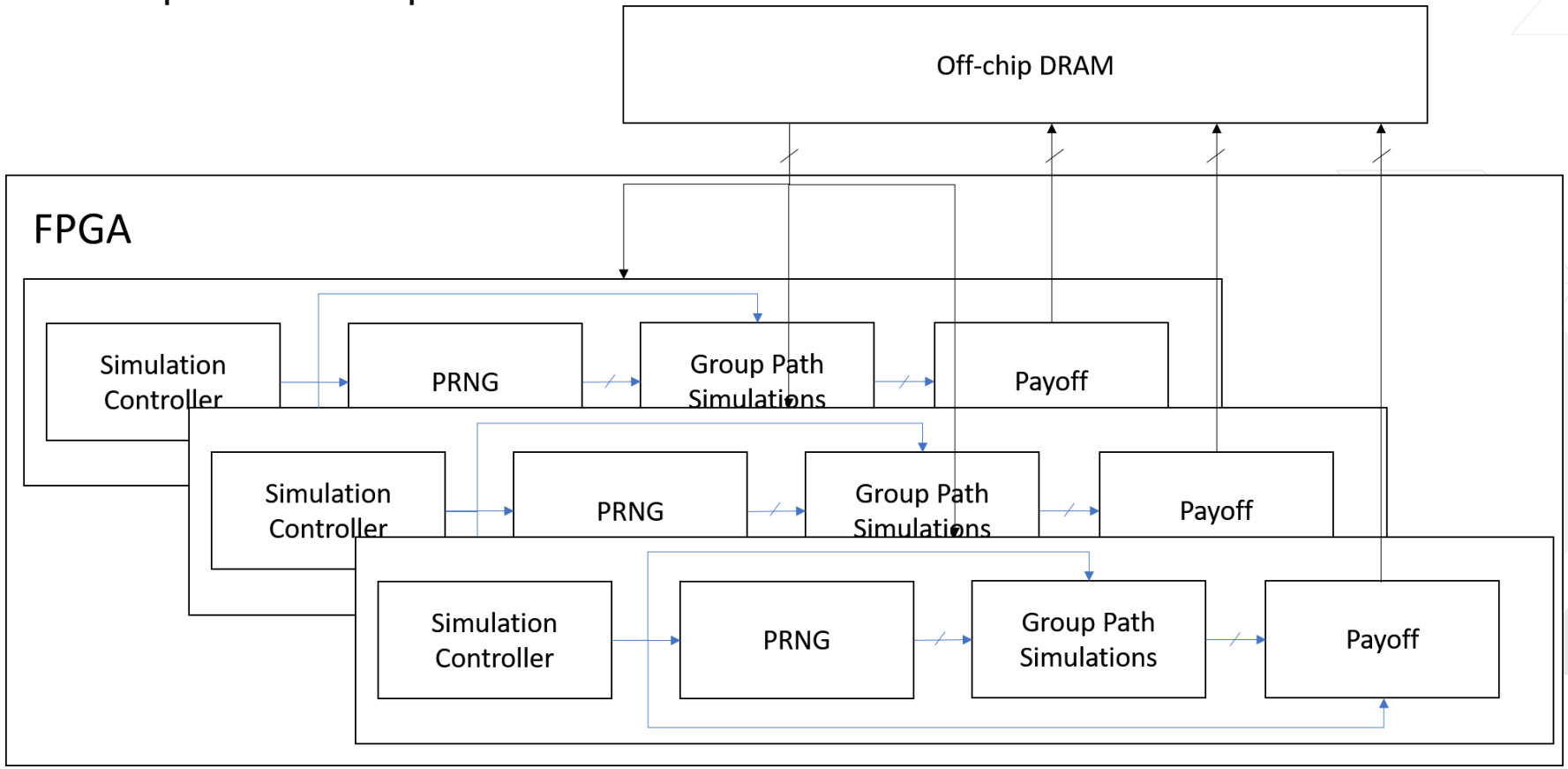
Architecture

- **Path simulation parallelization**
  - Unroll of loops



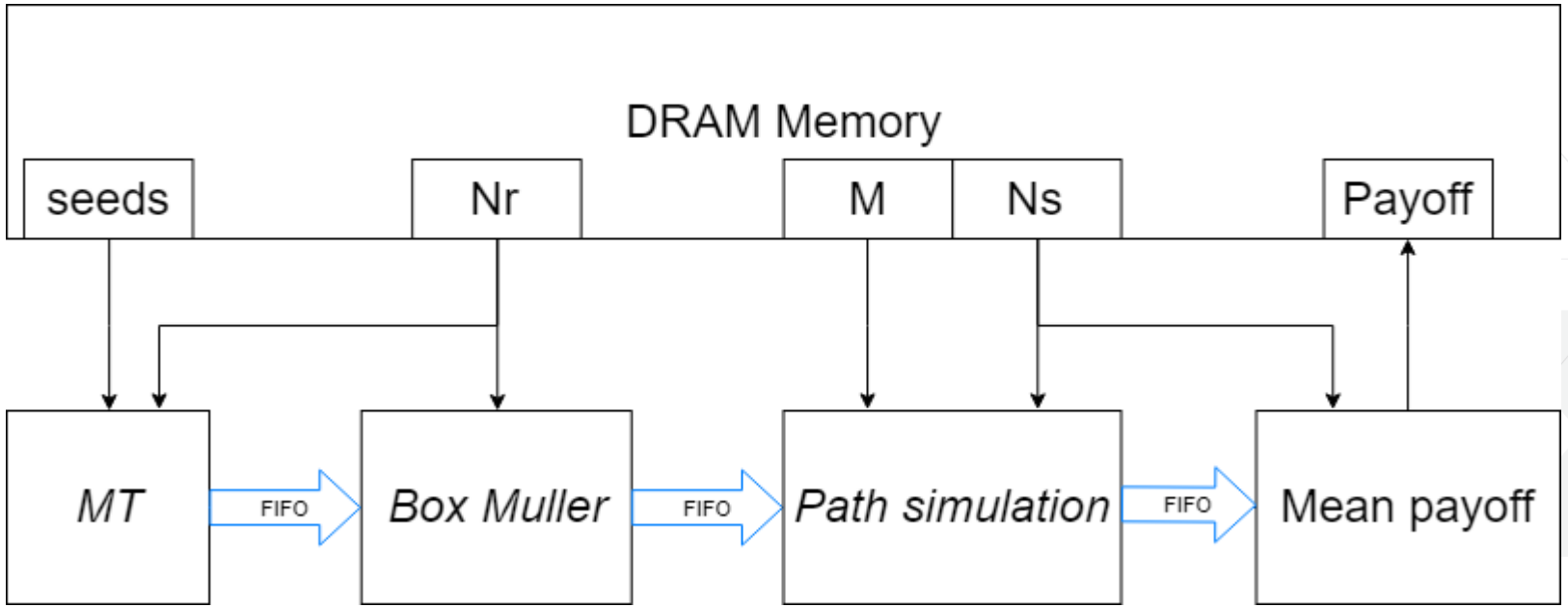
Architecture

- **Path simulation parallelization**
  - Parallel independent compute units



Architecture

- **Path simulation parallelization**
  - Dataflow optimization



Performance

- **Implementation on AWS F1**
  - ***TEMPORARY RESULT, UPDATE AS SOON AS THE NEW DATA PREPARED***

Model	Option	F1 FPGA [ms]	Power[W]
Blcak-Scholes Model	European vanilla option	3.2	80
	Asian option	3.21	80
Heston Model	European vanilla option	6.35	85
	European barrier option	6.33	85

Conclusion

- Modular design of Monte Carlo methods applied to the stock option pricing problems
- Implementation on state-of-the-art FPGAs
- Various hardware architecture optimizations using high level synthesis. Performance and resource utilization
- Comparable performance with respect to state of the art GPU implementations (and of course with respect to CPUs), with a very significant energy saving.

XDF XILINX DEVELOPER FORUM