A Reinforced Learning Solution for Clock Skew Engineering to Reduce Peak Current and IR Drop

Sayed Aresh Beheshti*, Ashkan Vakil*, Sai Manoj*, Ioannis Savidis†, Houman Homayoun¶, Avesta Sasan*
†Department of ECE, Drexel University, e-mail: is338@drexel.edu
¶Department of ECE, University of California, Davis, e-mail: hhomayoun@ucdavis.edu
*Department of ECE, George Mason University, e-mail: {sbehesht, avakil, spudukot, asasan}@gmu.edu

ABSTRACT

This paper purposes a Reinforcement Learning solution for peak current reduction by clock skew engineering. The reinforcement learning agent learns how to adjust each register's clock arrival time to maximize the clock arrival's distribution. The use of reinforcement learning allows us to explore optimization opportunities in clock tree synthesis beyond the heuristic algorithms used in modern EDA tools. Our experimental results support this claim as we report over 35% drop in peak current and major reduction in IR drop (from package to transistor) in the selected benchmarks. The agent explores despite creating timing violations and receives a large negative reward for its action. The agent, however, can receive a bonus reward in the future if the timing violation was fixed later by adjusting the clock arrival time of other registers, resulting in a broader spread in clock arrival distribution.

CCS CONCEPTS

• **Hardware** → **Clock-network synthesis**; • **Computing methodologies** → **Reinforcement learning**.

KEYWORDS

Clock Tree Synthesis, Peak Current Reduction, Reinforcement Learning, Physical Design

ACM Reference Format:
Sayed Aresh Beheshti-Shirazi, Ashkan Vakil, Sai Manoj, Ioannis Savidis, Houman Homayoun, and Avesta Sasan. 2021. A Reinforced Learning Solution For Clock Skew Engineering to Reduce Peak Current and IR drop. In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21), June 22-25, 2021, Virtual Event, USA. ACM, USA, 6 pages.*
https://doi.org/10.1145/3453688.3461754

1 INTRODUCTION

Clock Tree Synthesis (CTS) is a physical design step in which the clock network is implemented. Historically, the EDA tools were designed to build balanced clock trees by minimizing the clock skew, giving each register-to-register timing path equal time. The problem with the zero skew clock tree is that all registers launched simultaneously, resulting in a surge of demanded current from the battery on the active edge of the clock. This concept is illustrated in Figure 1. The Power Delivery Network (PDN) is a resisting and inductive network. The surge of demanded current results in a larger resister IR drop and results in a greater inductive voltage drop [4]. This result in complication in design of the package, and also degradation in performance [6]. The package design cost increases because the package designer needs to include additional decoupling capacitors to offset the inductive drop to reduce low-frequency oscillation by preventing the demanded current from being imposed on the inductive board. Simultaneously, the higher IR drop results in reduced voltage at the transistor level and degradation in performance. It also results in larger cycle-to-cycle voltage variation, a.k.a voltage noise. The larger voltage noise results in a higher clock jitter that the physical designer should account for. The higher clock jitter modeled using the uncertainty margin for each endpoint. The large uncertainty, in turn, reduces available slack for each reg-to-reg timing path, which in turn results in an inferior design and removed power, performance, and area (PPA) optimization opportunities.

Modern EDA solutions are equipped with heuristic algorithms to take advantage of the useful clock skew. Skewing the clock tree allows for time borrowing from preceding or proceeding reg-to-reg timing paths, increasing the available slack at the target timing path. Time borrowing allows for fixing timing violations without setting the IC's clock frequency to the most extended reg-to-reg path propagation delay. Useful skew has also been used in designing heuristic solutions for spreading the clock arrival times for peak current reduction in [4, 6]. The problem with heuristic solutions is that they are designed based on expert physical and EDA designers' experience and knowledge. Considering the massive optimization space, these heuristic solutions, although improving the optimization objective, fail to reach Pareto-optimal frontiers.

In this paper, we approach the subject of peak current reduction from a machine learning perspective. Rather than using a heuristic algorithm, we train a reinforcement learning agent that exploits opportunities to spread the clock arrival time and explore opportunities to discover new Pareto-frontier optimal spaces beyond the optimization space covered by heuristic algorithms. We use a two-phase solution for training our reinforcement learning agent. First, we allow the agent to perform a random walk of the environment to thoroughly explore the design and fully warm up its Q-table. Then, we will enable the agent to enter into an episodic training phase and employ a decaying epsilon greedy policy for its exploration. As the number of episodes increases the epsilon decreases.

2 BACKGROUND

In a physical design with a nearly balanced clock tree, the clock transition at the clock pin of all registers occurs within a short timing window. Let's consider a rising-edge triggered design where launch registers launch the data path in the clock's rising edge. The

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8393-6/21/06…$15.00
https://doi.org/10.1145/3453688.3461754

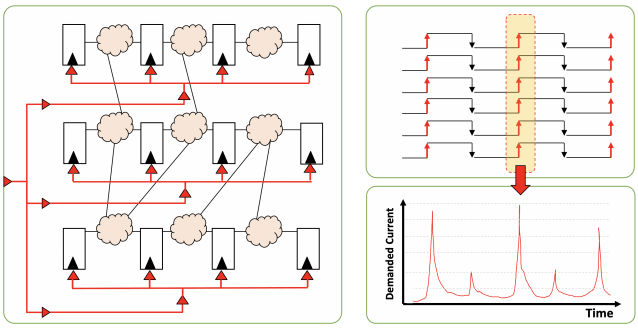

Figure 1: (Left): A sequential netlist with balanced clock tree, (right-top): Simultaneous Arrival time of the clock to all register, (right-bottom): surge of demanded current on active edge of the clock

capture registers capture the propagated logic value on the next rising edge of the clock. The simultaneous launch of registers (and activation of data paths) results in a surge of switching activity[4]. Additionally, in each clock transition (rise/fall), the clock level transition results in a further rise of current discharging and charging the capacitive clock network. As shown in Fig. 1, the accumulated impact is a sudden change in demanded battery current at each rising edge of the clock. In a rising-edge triggered design, the surge of current is less at the clock's falling edge, as the clock's falling edge only charges/discharges the clock network and does not result in additional switching activity in the datapath (unless there exist half-cycle timing paths). With the sudden reduction in switching activity between clock edges, the current surge is rapidly suppressed [15]. The PDN is an RLC network. Increase in demanded current results in the transistors experience both resistive and inductive voltage drop. The resistive drop is proportional to the current that flows through the resistive PDN (IR), and the inductive IR drop is proportional to the rate of change in the current (L.di/dt). The sudden change in the current in a short timing window significantly increases the inductive IR drop. That, in turn, results in significant voltage drop and voltage noise. The more substantial voltage drop reduces transistors' speed, and the voltage noise increases the clock tree's jitter. The reduction in the average transistor-level voltage and increase in the clock jitter should be accounted for during the physical design. Hence, a designer needs to set a lower rail voltage when performing static timing analysis to account for increased IR drop and increase each register endpoint's uncertainty to account for voltage noise. The reduction in peak current reduces the extent of inductive IR drop and also reduces the voltage noise. That means a higher voltage on transistor level and reduced clock jitter, resulting in a more extensive availability of timing slacks, which in turn could be used to improve the PPA of design under implementation. Hence, reducing the peak current can significantly impact the PPA optimality outcome of the physical design.

Many scholars have previously investigated the utilization of useful clock skew for sparing the arrival time of clock (with and without Machine Learning (ML)) solutions. In this section, we first review these prior art solutions to differentiate our approach. Then, we provide a brief background on Reinforcement Learning, which is necessary to understand our proposed solution.

2.1 Prior Art Solutions

2.1.1 ***Heuristic Solutions for Peak Current Reduction:*** Reducing the simultaneous switching of the clock network by operating half of the clock network buffers in the rising edge and the other half in the falling edge as cited by [17] is orthogonal to our solutions and could be applied simultaneously. However, having clock networks that operate in both rising and falling edges complicate clock network design. It requires the design and use of edge-triggered registers (triggered in both the rising and falling edge). The added capacitive load of such endpoint registers is about twice the capacitive load of regular rising-edge or falling-edge triggered registers. Hence, the added capacitive load of endpoint registers takes away a significant portion of the benefit resulted from operating the clock network in half frequency. The state replication and re-encoding technique proposed in [5] to minimize the switching activity of a Finite State Machine (FSM), which reduces the peak current, is also orthogonal to our solution. However, it is only applicable to the FSM portion of a netlist. Most related to our solution are a group of prior art solutions [4, 6, 8, 10, 12, 14, 15], where the objective of peak current reduction achieved through expanding the spread of the Clock Arrival Times (CAT). We pursue the same objective in this paper; however, instead of using a heuristic solution, we rely on a reinforcement learning agent that learns how to distribute the clock arrival times without causing timing violation.

2.1.2 ***Learning Solutions for Peak Current Reduction:*** The work in [16] uses a heuristic solution based on the genetic algorithm to perform clock-skew optimization. The paper introduces clustering techniques to account for limitations on the maximum number of clock drivers. The work in [9] employs a hybrid approach to optimize the CTS outcome, using Generative Adversarial Network (GAN) augmented by Reinforcement Learning. Traditional GAN comprises two components, one being a generator and the other being discriminator. In this solution, the reinforcement learning uses a pre-trained regression model as a supervisor of the generator.

Another recent work presented by [7] utilizes Artificial Neural Network (ANN) to estimate the clock tree components such as the numbers of clock buffers and wire-loads. The proposed technique uses ANN during CTS to identify the number of buffer insertions needed to meet the target clock skew and maximize input transition times for clock buffers and clock sinks. Furthermore, [11] discusses the employment of Machine Learning techniques such as the Support Vector Machine (SVM) algorithm to estimate clock buffer and wire sizing. The experiment focused on replacing expensive circuit-level simulation and reduced clock skew without any significant increase in power dissipation.

2.2 Background on Reinforcement Learning

Reinforcement learning (RL) is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. It is concerned with how an intelligent AGENT should take a sequence of actions to maximize the cumulative REWARD from the ENVIRONMENT. In an RL solution, the agent observes its STATE in the environment, and it takes an ACTION. The action results in a reward and a new state. The accumulated reward and change in the state provide the agent with delayed feedback on the consequence of the agents' course of actions. Agent's end goal is to maximize the total reward G_t in equation 1:

$$G_t = R_{t+1} + \sum_{k=1}^{\infty} \gamma^k (R_{t+k+1}) = R_{t+1} + \gamma^k (R_{t+2}) + \gamma^{(k+1)} (R_{t+3}) + \ldots \quad (1)$$

In this equation, t is the time step, R is the reward, γ is a discount factor, and it is an indication of agents behavior toward future rewards, and k is an integer index used for adjustment of time and discount factor for future time steps. The equation expresses the total reward as the summation of immediate reward R_{t+1} obtained when transitioned into the next-state S_{t+1} plus discounted future rewards that follow from the next step. The discount factor differs

between 0 and 1. The value closer to zero means the agent does not value future rewards and focuses on immediate rewards. The value closer to one means the agent is more focused on long-term rewards versus immediate rewards. Using the total reward formula, the agent would determine the expected return. In the training process, the agent learns how to maximize the reward and develop a policy π for its actions. The expected reward is the total reward expected when starting from the current state s_t and following the policy π. The value of being in a state is evaluated based on expected future rewards from that state using equation 2:

$$V_\pi(s_t) \doteq E_\pi[G_t|s_t] = E_\pi[R_{t+1} + \sum_{k=1}^{\infty} \gamma^k (R_{t+k+1})|s_t] \quad (2)$$

In this equation $V_\pi(s_t)$ is the value of being in state S at time t and following policy π, and $E_\pi[G_t|s_t]$ is the expected reward from state S_t following policy π. There are many variants of RL algorithms. We are utilizing the State-Action-Reward-State-Action (SARSA), which is a variation of the Q-Learning algorithm. Q-Learning algorithm is an off-policy and model-free RL that agent uses Q-values stored in a Q-table. Q-values are the values associated with different possible actions when being at any particular state. In Q-learning, the RL agent takes an iterative approach to improve the quality of Q-table entries. In contrast, SARSA is an on-policy version of Q-Learning that uses the value obtained by the current action taken to learn and improve the Q-table values. The Q-value for each action is calculated based on the following formula:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (3)$$

In this equation, $Q(s_t, a_t)$ is the action value function at state s_t and time a_t, and α is the learning rate, and γ is the discount factor.

3 PROPOSED SOLUTION

Figure 2 captures the high-level setup of our RL solution for peak current reduction. We use SARSA Q-Learning with a decaying epsilon greedy policy to optimize the distribution of clock skews. The RL agent moves from register to register, and in each register, it decides on its actions. One solution for the agent's movement in the netlist is for the agent to move from one register to all registers connected to it (any launch register in its incoming or any capture register in its outgoing timing paths). However, building such an environment carries many complications as the adjacency matrix for netlist connectivity is very sparse. This complicates the design of the q-table, as the q-table should be constructed to support the large in-degree and out-degree reg-to-reg connectivity. We decided to decouple the agent movement from netlist connectivity and arrange the registers in a grid-word (Torus, to be exact!) to simplify the problem. In this arrangement, from any register, the agent can go to only four other registers on its left, right, top, and bottom. The arrangement of registers in the grid world is random. Our experimental results indicated that as far as the number of steps in each episodic task is large enough, it does not matter how the grid word is arranged and how such superficial connectivity is established. To handle the agent's movement in the boundary of the grid-world, we converted the grid-world into a torus. For example, an agent on the left side of the grid can decide to move left, and it will be moved to the same row but on the right side of the grid. In our RL solution, the state, action, and reward are defined as follows:

Current-State (S_t): The location of the agent in the grid, the number of clock buffers leading to each endpoint register (indicating CAT for that register), the resulting distribution of the clock arrival times (CAT Dictionary), and a dictionary of Timing Violations (TV).

Next-State (S_{t+1}): The new location of the agent, an updated number of clock buffers leading to each endpoint (The CAT of the register is updated), the updated CAT dictionary, the updated TV dictionary.

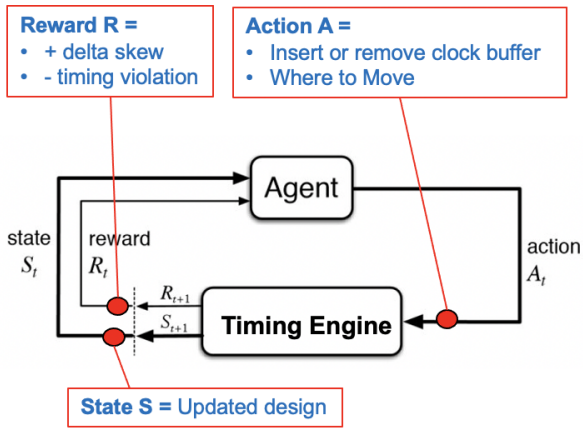

Figure 2: Reinforcement Learning Solution for Peak Current Reduction, and description of rewards in this work.

Action: A set of available activities to the agent at any specific state (register.) In our design, at any particular state, the agent has to take two actions: 1) insert or remove up to 5 buffer (11 actions: five adds, five removes, and no-action), 2) move to a neighbor register (4 actions: Move Up (U), Down (D), Left (L), Right (R)). The adjacent register is determined based on the grid-world arrangement and not the actual register connectivity. The evaluation of action or move is performed in their corresponding q-tables using the same reward function. The agent learns one policy for movement and another for insertion or removal of the buffers. The policy learned for the buffer insertion or removal is independent of the grid-world. However, the policy learned for movement depends on the grid-world connectivity.

Reward: The reward is the feedback realized by the agent upon entering the new state. Our agent receives two rewards: 1) A positive/negative reward (r_σ) if the spread of clock arrival times is increased/reduced proportionally to the delta of the new distribution versus old. 2) A large negative/positive reward (r_t) if a timing violation created/resolved proportional to the extent of the timing violation. The reward (r_σ) is the difference between the standard deviation of clock arrival distribution before and after the agent's buffer insertion/removal action and it is computed using:

$$r_\sigma = \sigma_{new} - \sigma_{old} \quad (4)$$

The r_t, on the other hand, is the accumulated sum of timing violation (in picoseconds) added or removed after insertion/removal of the buffer for all timing paths to/from the Agent's resident register. The r_t is obtained by performing setup timing checks twice: once for all timing paths leading to the Agent's register (with the Agent's register used as capture register). And once for all timing paths starting from that register (with the Agent's register used as launch register). Algorithm 1 captures how the environment behaves, how the rewards r_σ and r_t are computed, and how the updated stated is computed and fed back to the Agent.

An essential variable in this algorithm is the notion of adjusted CAT, denoted by Δ. Consider a register whose skew (negative or positive) becomes large enough such that its launch is closer to the mean arrival of the clock (across all registers) in the previous or next cycle. In this case, the additional increase in the skew of the register pushes it further towards another clock edge. Hence,

Algorithm 1 Reinforcement Learning Environment Implementation

Input:
d_s ▷ Added delay in current step to adjust the CAT
FF ▷ Target register for take the action
DIC ▷ Dictionary of connections, CAT, T, and t_{pd}
TV ▷ A list of current Timing Violation
Δ ▷ Adjusted CAT based on proximity to clock edge
Output:
r_σ ▷ reward for widening the CAT distribution
r_t ▷ Reward for timing violation
DIC ▷ Updated dictionary of connections, CAT, T, and t_{pd}
TV ▷ Updated list of Timing Violation
Δ ▷ Updated adjusted CAT based on proximity to clock edge
procedure ENV(d_s, FF, DIC, TV, Δ)
$T \leftarrow$ Clock Period for design from table 1
$CAT \leftarrow DIC[0]$ ▷ Extract dictionary of clock arrival times
$CONNECT \leftarrow DIC[1]$ ▷ Extract dictionary of register connectivity
$Slack_{old} \leftarrow DIC[2]$ ▷ Extract slacks of previous step
$CAT_{FF} \leftarrow CAT(FF) + d_s$
$r_{to} \leftarrow 0$
$r_{from} \leftarrow 0$
$TV_{tmp} \leftarrow TV$ ▷ create a local copy
for all FF_c capture registers in $CONNECT$ where FF is a launch register **do**
$t_{cq} \leftarrow$ clock to Q for register FF ▷ Stored in DIC
$t_{pd} \leftarrow$ propagation delay from FF to FF_c ▷ Stored in DIC
$t_{su} \leftarrow$ setup time for register FF_c ▷ Stored in DIC
$Slack = T + CAT(FF_c) - CAT_{FF} - t_{cq} - t_{pd} - t_{su}$
if ($Slack > T$) **then** ▷ action invalid –> Return 0 reward
Return (0,0,DIC,TV,Δ)
else if (0 < Slack < T) **then**
if ($FF\_to\_FF_c$ in TV_{tmp}) **then** ▷ reward for removing violation
remove $FF\_to\_FF_c$ from TV_{tmp}
r_{to} += $-Slack_{old}$
end if
else ▷ $slack < 0$
if ($FF\_to\_FF_c$ in TV_{tmp}) **then** ▷ r_{to} = change in negative slack
r_{to} += $Slack - Slack_{old}$
else
Append $FF\_to\_FF_c$ to TV_{tmp}
r_{to} += $Slack$ ▷ r_{to} proportional to negative slack
end if
end if
end for
for all FF_l launch registers in $CONNECT$ where FF is a capture register **do**
$t_{cq} \leftarrow$ clock to Q for register FF_l ▷ Stored in DIC
$t_{pd} \leftarrow$ propagation delay from FF_l to FF ▷ Stored in DIC
$t_{su} \leftarrow$ setup time for register FF ▷ Stored in DIC
$Slack = T + CAT_{FF} - CAT(FF_l) - t_{cq} - t_{pd} - t_{su}$
if ($Slack > T$) **then** ▷ action invalid –> Return 0 reward
Return (0,0,DIC,TV,Δ)
else if (0 < Slack < T) **then**
if $FF_l\_to\_FF$ in TV_{tmp} **then** ▷ reward for removing violation
remove $FF_l\_to\_FF$ from TV_{tmp}
r_{from} += $-Slack_{old}$
end if
else ▷ $slack < 0$
if ($FF_l\_to\_FF$ in TV_{tmp}) **then** ▷ r_{from} = change in negative slack
r_{from} += $Slack - Slack_{old}$
else
Append $FF_l\_to\_FF$ to TV_{tmp}
r_{from} += $Slack$ ▷ r_{to} proportional to negative slack
end if
end if
end for
$CAT(FF) \leftarrow CAT_{FF}$ ▷ update the DIC
$\sigma_{old} \leftarrow$ Standard Deviation of Δ
$\delta \leftarrow CAT(FF)$ - $Avg(\Delta)$
$\Delta(FF) \leftarrow min(\delta, T - \delta)$. ▷ skew is computed from closest clock edge
$\sigma_{new} \leftarrow$ Standard Deviation of Δ
$r_\sigma \leftarrow \sigma_{new} - \sigma_{old}$. ▷ change in standard deviation as reward
$r_t \leftarrow r_{to} + r_{from}$ ▷ reward/penalty for reducing/increasing timing violation
Return ($r_\sigma, r_t, DIC, TV_{tmp}, \Delta$)
end procedure

when widening CAT distribution, we need to be aware of the clock arrival distribution in the previous and next cycle. More precisely, we need to make sure that the tail of the distribution is not widened to the point that coincides and contributes to the peak current in the previous or next cycle. To handle this scenario, we introduced the dictionary of adjusted CAT denoted by Δ. Initially, the CAT dictionary copied to Δ. However, with each update to the clock arrival time of register FF, the Δ dictionary is updated as follows:

$$\delta \leftarrow CAT(FF) - Avg(\Delta) \tag{5}$$

$$\Delta(FF) \leftarrow min(\delta, T - \delta) \tag{6}$$

The reward r_σ is also computed based on the change in the standard deviation of Δ. The use of the adjusted CAT dictionary, Δ, assures that the agent does not only consider the reduction of peak current in one clock transitions but also across multiple clock cycles, resulting in a more significant overall reduction.

Algorithm 2 describes the agent behavior and how it interacts with the environment, taking action, observing the next state and reward from the environment, and updating its Q-tables. The agent goes through two phases of learning: 1) delay start, 2) decaying epsilon greedy SARSA learning. In delay start (phase 1), the agent is set on full exploration mode. It selects two random actions in each step, one for the move and one for buffer insertion from the set of possible actions. The above allows the agent to explore the environment stochastically and warm up its Q-tables with consequences or values associated with each state's potential actions. In the SARSA learning phase (phase 2), the agent follows an epsilon greedy policy. The value epsilon (ϵ) is originally set to 0.9. At each step, a random number between 0 and 1 generated. The agent takes a random action if the generated value is less than ϵ and takes the most greedy actions for insertion/removal and move (based on the value stored in Q_M and Q_A Q-tables) when the generated value is greater than ϵ. By using an epsilon greedy policy, the chances of the agent taking random action decays over time. This allows the agent to first explore aggressively and change behavior towards exploiting the largest accumulated reward over time. In the last episode, the agent only exploits, generating the largest possible reward. The agent uses a weighted measure of both rewards to compute a single reward value (r) for its selected action calculated using:

$$r = r_\sigma + s \times r_t \tag{7}$$

Where s is a scalar factor used to scale the significance of a timing violation concerning the benefit/loss resulting from an increase/decrease in the spread of clock arrival times. The scale factor is used to assure that negative reward for timing violations is many times larger than positive reward obtained for widening the distribution. This forces the agent to minimize or avoid creating timing violations unless during its exploratory stage in the hope of fixing it elsewhere. When the timing violations are resolved, the agent receives a similarly large reward back. One option that could be explored in the future is to escalate the value of s in each episode, allowing the agent to create timing violations in the early episode and fix it elsewhere (increasing exploration). We will investigate this solution in our future work. The reward that is calculated using the above formula would be used in formula 3 to update the Q-values associated with state-actions (buffer insertion/removal and movement) in each of the two Q-tables (Q_M and Q_A). The calculated reward would directly affect agent behavior next time (in future episodes) when the agent visits this state (register).

4 EXPERIMENTAL RESULTS

We implemented the reinforcement learning solution using the environment described in Algorithm 1, and the agent as described in algorithm 2. We tested our solutions in selected (larger) ISCAS89 benchmarks [1], AES and Ethernet. Each benchmark is implemented

Algorithm 2 SARSA Learning Agent

$\alpha \leftarrow 0.9$ ▷ Learning rate
$\gamma \leftarrow 0.9$ ▷ Discount factor
$\epsilon \leftarrow 0.9$ ▷ Control knob for exploration vs exploitation
$d_{BUF} \leftarrow 30ps$ ▷ delay of buffer used in clock tree
s = 100 ▷ Scale factor for reward
steps, episodes, delay-Start ← from table 1 ▷ Benchmark specific variables
start-point ← choose a start point register randomly
$DIC_{CAT} \leftarrow$ build a dictionary of current design clock arrival times
$DIC_{Connectivity} \leftarrow$ build a dictionary for connectivity of each register
$DIC \leftarrow [DIC_{CAT}, DIC_{Connectivity}]$
$TV \leftarrow []$ ▷ empty list for storage of Timing Violations
$\Delta \leftarrow CAT$ ▷ Adjusted CAT based on proximity to Clk edge (initially equal to CAT)

▷ ——————————**Phase 1** ——————————- Delay start to warm up the Q-tables
Initialize Q_M and Q_A (arbitrarily) ▷ Move and Action Q-tables
$FF \leftarrow$ start-point
for (st=1, st <= delay-start, st++) **do**
  Randomly select Actions a ▷ $a \in$ [add, remove, none]
  if $(a = add(x) \parallel a = remove(x))$ **then** ▷ x ∈ [1,5] = number of buffers
    $d_s \leftarrow x \times d_{BUF}$
    $(r_\sigma, r_t, DIC, TV, \Delta) \leftarrow \text{Env}(d_s, FF, DIC, TV, \Delta)$ ▷ Algorithm 1
    $r \leftarrow r_\sigma + s \times r_t$ ▷ weighted reward
    $Q_A(s,a) \leftarrow Q_A(s,a) + \alpha \cdot (r + \gamma \cdot (Q_A(s',a') - Q_A(s,a))$
  end if
  Randomly select Actions m ▷ $m \in [U, D, L, R]$
  $FF \leftarrow$ Move in direction m of current FF
  $Q_M(s,m) \leftarrow Q_M(s,m) + \alpha \cdot (r + \gamma \cdot (Q_M(s',m') - Q_M(s,m))$
end for

▷ ——————————**Phase 2** ——————————— SARSA Reinforcement Learning
$\xi \leftarrow 0.995$ ▷ ϵ decay factor
$\alpha \leftarrow 0.9$ ▷ Learning rate
$\gamma \leftarrow 0.35$ ▷ Discount factor
for (epi=0, epi <= episodes, epi++) **do** ▷ episodes count in table 1
  $FF \leftarrow$ start-point
  $DIC \leftarrow [DIC_{CAT}, DIC_{Connectivity}]$
  $TV \leftarrow []$ ▷ empty list for storage of Timing Violations
  for (st=0, st <= steps, st++) **do**
    $rand \leftarrow random \in [0, 1]$
    if $rand > \epsilon$ **then** ▷ Agent does exploitation
      $a \leftarrow argmax(Q_A)$ ▷ Insertion exploitation
      if $(a = add(x) \parallel remove(x))$ **then**
        $d_s \leftarrow x \times d_{BUF}$
        $(r_\sigma, r_t, DIC, TV, \Delta) \leftarrow \text{Env}(d_s, FF, DIC, TV, \Delta)$
        $r \leftarrow r_\sigma + s \times r_t$ ▷ weighted reward
        $Q_A(s,a) \leftarrow Q_A(s,a) + \alpha \cdot (r + \gamma \cdot (Q_A(s',a') - Q_A(s,a))$
        $m \leftarrow argmax(Q_M)$ ▷ Movement exploitation
        $FF \leftarrow$ Move in direction m of current FF
        $Q_M(s,m) \leftarrow Q_M(s,m) + \alpha \cdot (r + \gamma \cdot (Q_M(s',m') - Q_M(s,m))$
      end if
    else ▷ $(rand < \epsilon)$ Agent does exploration
      Randomly select Actions a ▷ $a \in$ [add, remove, none]
      if $(a = add(x) \parallel a = remove(x))$ **then** ▷ x ∈ [1,5]
        $d_s \leftarrow x \times d_{BUF}$
        $(r_\sigma, r_t, DIC, TV, \Delta) \leftarrow \text{Env}(d_s, FF, DIC, TV, \Delta)$
        $r \leftarrow r_\sigma + s \times r_t$ ▷ weighted reward
        $Q_A(s,a) \leftarrow Q_A(s,a) + \alpha \cdot (r + \gamma \cdot (Q_A(s',a') - Q_A(s,a))$
      end if
      Randomly select Actions m ▷ $m \in [U, D, L, R]$
      $FF \leftarrow$ Move in direction m of current FF
      $Q_M(s,m) \leftarrow Q_M(s,m) + \alpha \cdot (r + \gamma \cdot (Q_M(s',m') - Q_M(s,m))$
    end if
    if (epi = episodes-1) **then**
      $\epsilon \leftarrow 0.$ ▷ only exploit in the last episode
    else if $(\epsilon > 0.10)$ **then**
      $\epsilon = \epsilon \times \xi$ ▷ decay the ϵ to reduce exploration
    else ▷ if $(\epsilon > 0.10)$
      $\epsilon = 0.10$ ▷ keep small exploration
    end if
  end for
end for

twice in Synopsys ICC2 [2]: Once using the standard Place and Route (PnR) flow of the Synopsys ICC2 and once using computed clock arrival times generated from RL. In the second implementation, the computed clock arrival times from RL are provided to ICC2 as suggested arrival times using the "set_clock_balance_points" command. Note that the outcome of ICC2 may be different from our CAT list, as this input is only considered a suggested list and implemented to the extent possible. Each design is then pushed through static timing analysis and timing closure and then is subjected to the power and IR analysis using ANSYS Redhawk [3]. We describe our experimental results in detail in the rest of this section.

4.1 Agent Learning Behavior

Figure 3 illustrates how the agent learns to increase the clock arrival time distribution over time for each benchmark over time. Consider the learning curve for the AES benchmark: As illustrated after around 700 episodes of learning, the learning curve saturates, and the agent can no longer increase the CAT distribution. The number of steps in each episode is roughly set to 20X the number of registers in the benchmark (e.g., 200K steps for AES in each episode).

4.2 Change in Clock Arrival Times

Figure 4 captures the clock arrival time distribution with and without the use of the RL agent. The CAT distribution is captured after clock tree synthesis. The CAT in the baseline is generated based on ICC2 standard CTS flow. The CAT for RL solution is generated by feeding the suggested clock arrival times (output of RL) to ICC2, performing the CTS, and capturing the resulting CAT distribution. As illustrated, the RL agent has significantly widened the distribution of clock arrivals. As the result of a wider distribution, we expect a corresponding reduction in peak current and IR drop of the design.

4.3 Impact on Peak Current

Figure 5 captures the impact of RL solution in peak current reduction for selected benchmarks over ten cycles of execution. The peak current figures are generated using Ansys Redhawk [3] where the switching activity of clock and data in each clock cycle is set to 200% (rise and fall) and 10% respectively. As illustrated, the peak current for all benchmarks has reduced between 35% to 40%. This is the direct result of spreading the clock arrival time of the registers and the resulting reduction in simultaneous switching activity.

4.4 Impact on IR drop

The reduction in peak current greatly improves the overall voltage drop from package to the transistor level. Most of the improvement comes from the reduction in inductive voltage drop Ldi/dt. More specifically, the decline in the rate of change in the current, due to a wider distribution of clock arrival times, reduces the inductive voltage drop. Figure 6 shows the IR map of selected benchmarks in the base and RL-assisted CTS flow for all benchmarks.

To measure how much such reduction in IR drop improves the rail voltages seen at the transistor level, we employed the methodology described in [13] to compute the delay equivalent voltage V_{DEV} of each design (minimum voltage seen by transistors at launch) in both base and RL assisted design. We also computed the maximum cycle to cycle voltage noise (from two consecutive cycles) using a vector-less IR simulation and reported the voltage noise. The result of this analysis is reflected in table 2. As illustrated, by using the RL solution for peak current reduction, the transistors, on average, see a higher rail voltage. At the same time, the extent of cycle-to-cycle voltage variation is reduced. The decline in voltage noise relaxes the requirement uncertainty margin (accounting for smaller clock jitter). Combining a higher overall rail voltage and smaller uncertainty (voltage noise) increases the extent of available timing slack in each timing path. This, in turn, could be used to improve the PPA optimality of the end design.

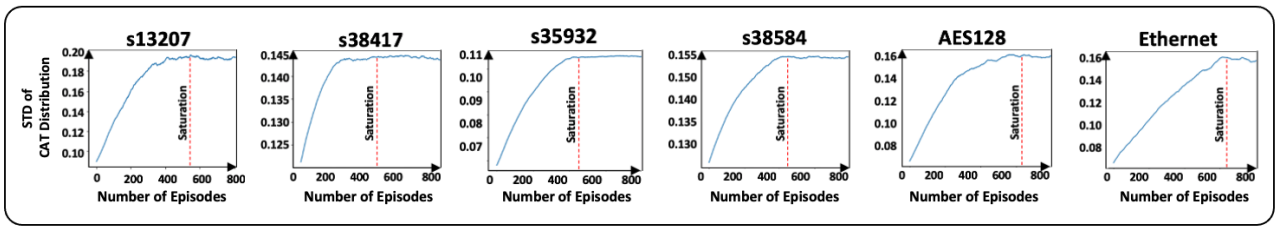

Figure 3: Increase and saturation of the standard deviation (σ) of register's clock arrival time (CAT) distribution over the number of reinforcement learning episodes. The saturation of σ indicates the RL model can not longer increase the CAT distribution, and the RL solution can be terminated.

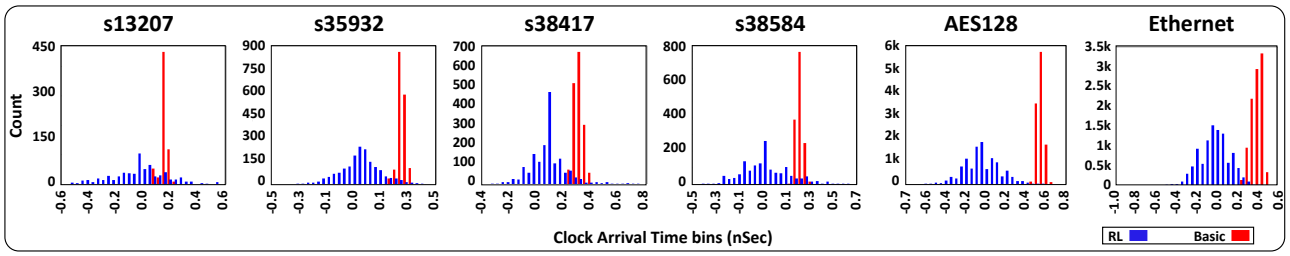

Figure 4: Histogram of Clock Arrival Time to endpoint registers before and after running the RL solution for peak current reduction. The broader distribution of clock arrival times (without causing any timing violation) reduces the extent of simultaneous switching and, in turn, would reduce the peak current and inductive voltage drop.

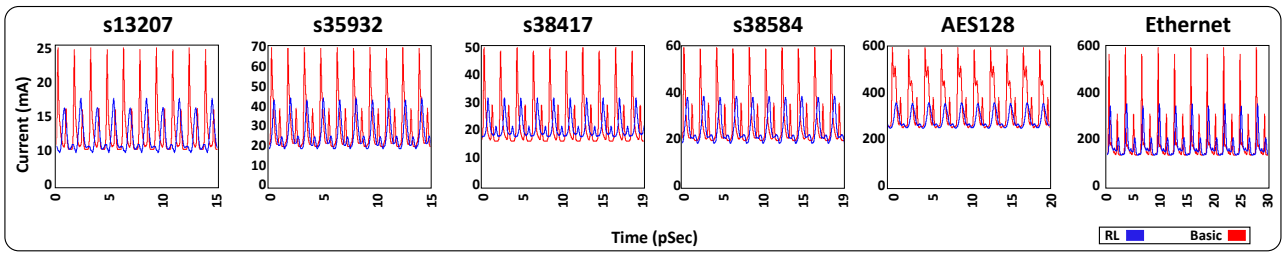

Figure 5: Current Waveform of dynamic vector-less simulation of benchmarks in 10 consecutive cycles before and after application of RL solution for peak current reduction. As illustrated, the wider CAT distribution reduces the demanded battery current across all benchmarks.

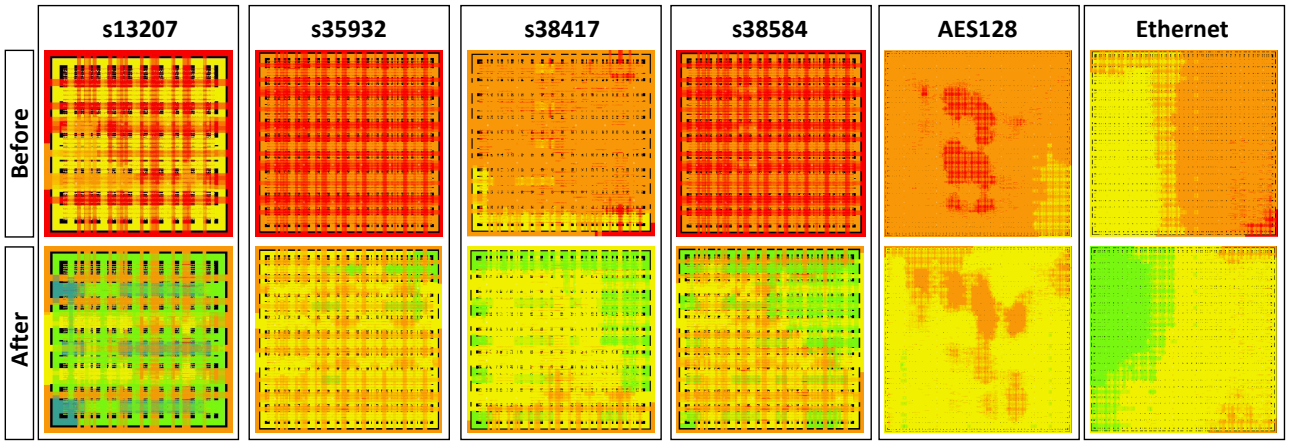

Figure 6: The IR map of benchmarks before and after applicaiton of RL solution for peak current reduction. The reduction in simultaneous switching reduces the di/dt, which in turns result in reduction in inductive IR drop (Ldi/dt).

Table 1: Experiments parameters. The clock periods for each benchmark are selected based on highest achievable performance when using standard Vt cells for physical design. Number of steps in each episode is roughly 15X to 20X the number of registers.

Benchmark	Period	Registers	warm up steps	Episodes	Steps
s13207	1.5	649	25000	1000	10000
s35932	1.4	1728	50000	1000	20000
s38417	1.9	1564	50000	1000	20000
s38584	1.8	1451	50000	1000	20000
AES128	2.9	10015	250000	1000	200000
Ethernet	2.0	10768	250000	1000	200000

Table 2: Summary of improvement in Voltage drop and reduction in peak current for selected benchmarks. The Min Rail voltage is computed using the methodology described in [13]. The Max C2C voltage noise is the maximum amount of change in the voltage (from launch to capture) within one Cycle (voltage noise). The lower the Voltage noise, the smaller the clock jitter and the resulting requirement for endpoint uncertainty. As illustrated, the RL solution resulted in a reduction in peak current, improvement in rail voltage, and a decline in the extent of voltage noise.

Design	Model	V_{DEV} and voltage noise based on formulation in [13]		% Peak Current Reduction
		Min Rail Voltage (V)	Max C2C Voltage Noise (mV)	
s13207	RL	0.925	19	36.2%
	base	0.912	33	
s35932	RL	0.913	21	35.8%
	base	0.906	29	
s38417	RL	0.924	20	34.8%
	base	0.915	31	
s38584	RL	0.920	19	35.9%
	base	0.914	28	
AES128	RL	0.922	22	39.4%
	base	0.915	32	
Ethernet	RL	0.926	27	40.1%
	base	0.909	43	

5 CONCLUSION

This paper presented a Reinforcement Learning solution based on decaying epsilon-greedy SARSA for peak current reduction of a design at the Clock Tree Synthesis stage. Our experimental results indicated an over 35% to 40% decrease in peak current when RL-assisted solution is used compared to a baseline design generated using heuristic Clock-Tree Synthesis solution of the utilized physical design EDA tool. As the result of such peak current reduction, we reported a significant reduction in the inductive voltage drop and a favorable reduction in voltage noise across selected benchmarks.

6 ACKNOWLEDGEMENT

This work is supported by KBR and Air Force Research Lab (AFRL).

REFERENCES

[1] 2020. The ISCAS'89 benchmark circuits. http://www.pld.ttu.ee/~maksim/benchmarks/iscas89/verilog/. Accessed: 2020-09-30.
[2] 2020. *Synopsy EDA tools.* http://synopsys.com/ Accessed July 10, 2020.
[3] ANSYS Apache. 2020. *Redhawk.* https://www.apache-da.com/products/redhawk Accessed November 17, 2020.
[4] L. Bhamidipati, B. Gunna, H. Homayoun, and A. Sasan. 2017. A Power Delivery Network and Cell Placement Aware IR-Drop Mitigation Technique: Harvesting Unused Timing Slacks to Schedule Useful Skews. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 272–277.
[5] J. Gu, G. Qu, L. Yuan, and Q. Zhou. 2010. Peak current reduction by simultaneous state replication and re-encoding. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 592–595. https://doi.org/10.1109/ICCAD.2010.5654204
[6] B. Gunna, L. Bhamidipati, H. Homayoun, and A. Sasan. 2017. Spatial and temporal scheduling of clock arrival times for IR hot-spot mitigation, reformulation of peak current reduction. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6. https://doi.org/10.1109/ISLPED.2017.8009179
[7] Y. Kwon, J. Jung, I. Han, and Y. Shin. 2018. Transient Clock Power Estimation of Pre-CTS Netlist. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4. https://doi.org/10.1109/ISCAS.2018.8351430
[8] W. . C. D. Lam, Cheng-Kok Koh, and C. . W. A. Tsao. 2002. Power supply noise suppression via clock skew scheduling. In *Proceedings International Symposium on Quality Electronic Design*. 355–360. https://doi.org/10.1109/ISQED.2002.996772
[9] Y. C. Lu, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim. 2019. GAN-CTS: A Generative Adversarial Framework for Clock Tree Prediction and Optimization. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942063
[10] A. Mukherjee and R. Sankaranarayan. 2004. Retiming and clock scheduling to minimize simultaneous switching. In *IEEE International SOC Conference, 2004. Proceedings.* 259–262. https://doi.org/10.1109/SOCC.2004.1362427
[11] Rupak Samanta, Jiang Hu, and Peng Li. 2009. Discrete buffer and wire sizing for link-based non-tree clock networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18, 7 (2009), 1025–1035.
[12] Shih-Hsu Huang, Chia-Ming Chang, and Yow-Tyng Nieh. 2006. Fast multi-domain clock skew scheduling for peak current reduction. In *Asia and South Pacific Conference on Design Automation, 2006.* 6 pp.–. https://doi.org/10.1109/ASPDAC.2006.1594691
[13] Ashkan Vakil, Houman Homayoun, and Avesta Sasan. 2019. IR-ATA: IR annotated timing analysis, a flow for closing the loop between PDN design, IR analysis & timing closure. In *ASP-DAC*. 152–159.
[14] A. Vijayakumar, V. C. Patil, and S. Kundu. 2016. An Efficient Method for Clock Skew Scheduling to Reduce Peak Current. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. 505–510. https://doi.org/10.1109/VLSID.2016.24
[15] Patrick Vuillod, Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. 1996. Clock-skew optimization for peak current reduction. In *Proceedings of 1996 International Symposium on Low Power Electronics and Design*. IEEE, 265–270.
[16] P. Vuillod, L. Benini, A. Bogliolo, and G. De Micheli. 1996. Clock-skew optimization for peak current reduction. In *Proceedings of 1996 International Symposium on Low Power Electronics and Design*. 265–270.
[17] Yow-Tyng Nieh, Shih-Hsu Huang, and Sheng-Yu Hsu. 2005. Minimizing peak current via opposite-phase clock tree. In *Proceedings. 42nd Design Automation Conference, 2005.* 182–185. https://doi.org/10.1109/DAC.2005.193797