Member-only story

Fundamentals of High-Level Synthesis Part 1: Basic Concepts

Mohammad Hosseinabady · Follow

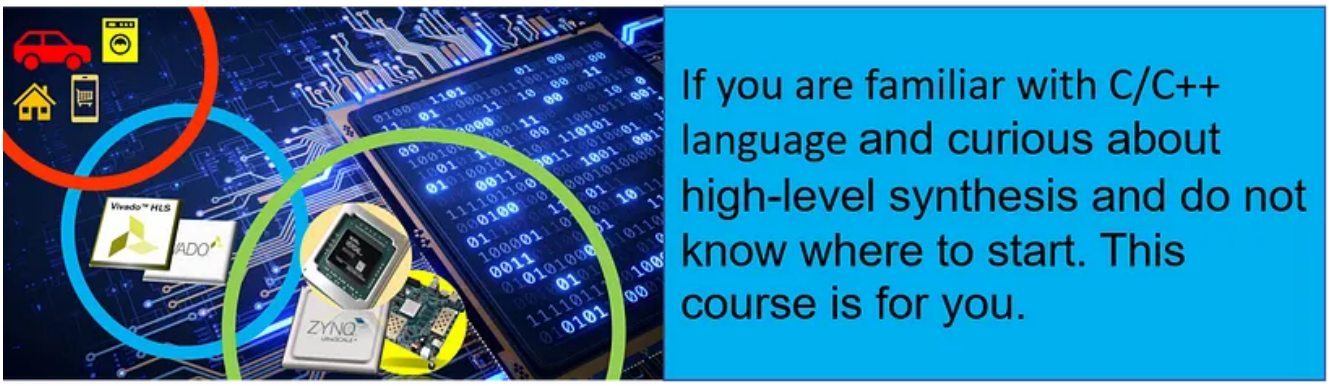
5 min read · Sep 23, 2019

With the emergence of compute-intensive applications in different areas such as Artificial Intelligence (AI), Machine Learning (ML), Convolutional Neural Networks (CNN), Internet of Things (IoT), image processing, computer vision, advanced driver-assistance systems (ADS) just to name a few, the demand for application-specific accelerators has risen. The main goal of these accelerators is improving the performance by reducing the latency and increasing the throughput. However, reducing energy consumption is also a key factor for using the accelerators in today embedded systems which commonly draw their power from batteries. To address this demand, designers have been exploring three computational platforms, including FPGA, GPU, and ASIC.

GPUs are the hardware platforms that have been adapted for accelerating compute-intensive parallel algorithms which cover most of the new applications. The availability, easy programming, and debugging are the main benefits of these platforms. However, high energy consumption is the main drawback that prevents them from using in energy-critical embedded systems. Therefore, researchers are working to make them energy-efficient in such systems. A sound track record of research and development products is Nvidia Jetson Family.

ASICs have recently been the main focus of the industry to provide AI-related hardware accelerators. Examples are Intel® Movidius™ Neural Compute Stick and Google Coral USB. Although these accelerators have been successful in reducing energy consumptions, they are not as versatile as GPUs.

FPGAs are the hardware platforms that provide high-performance, low energy consumption, and versatility. Traditionally, FPGAs were reconfigurable hardware platforms for prototyping and low-level controller design. However, by significant advancement in the new FPGA architectures (especially integrating floating-point operator modules and hardware embedded processors), their usage as computing accelerators has become more popular.



Although FPGAs can potentially deliver high-performance accelerators, their programming is not straightforward. Conventionally, hardware description languages (HDLs) such as VHDL or Verilog and their related design flow were the only way to design for FPGAs effectively. The design complexity associated with HDLs has prevented FPGAs from being ubiquitous. The main reason behind this phenomenon is the cycle accuracy required by synthesizable HDLs. To support the cycle accuracy, specific design methodologies, techniques, and templates along with an in-depth knowledge of hardware architectures should be learned by designers. This hardware-close implementation makes the design flow tedious, error-prone, and hard to debug. The essential technique to overcome these problems is removing the cycle accuracy from the input description by raising the design level from register transfer level (RTL) to functional level and using high-level languages such as C/C++ instead of HDLs. This approach leaves the task of generating the corresponding cycle-accurate HDL to the compilers or high-level synthesis (HLS) tools. Figure 1 depicts the relationship between a typical HLS compiler and the low-level compilers.

The new HLS tools can efficiently transform a high-level description into the equivalent cycle-accurate HDLs. They still need a little help from designers by augmenting the algorithm code with the compiler directives to achieve a high-performance implementation.

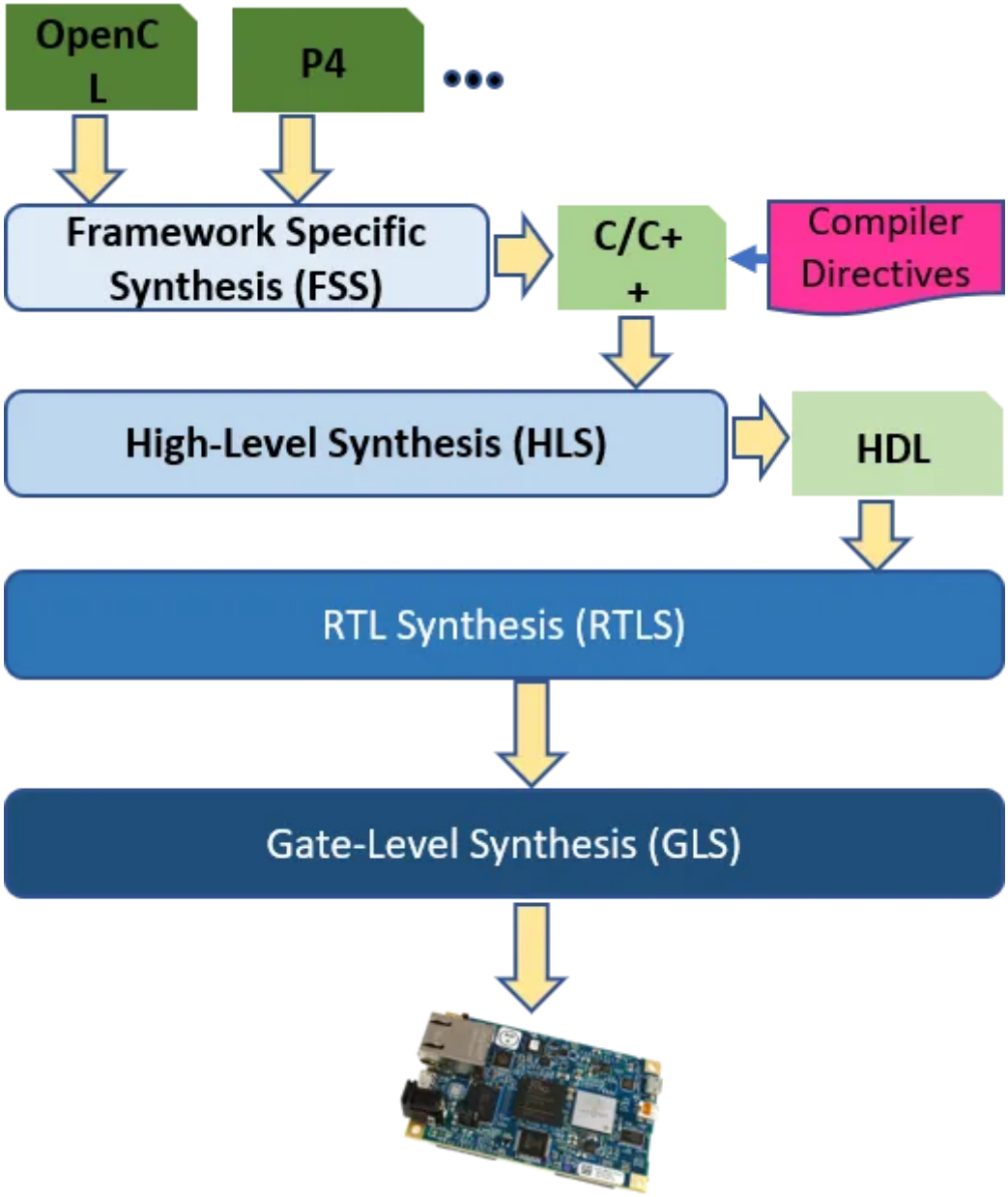


Figure 1 FPGA design flow

Let us take the following *dotProduct* function as an example to clarify the impact of compiler directives. Also, let us assume the latency of each addition or multiplication is three clock cycles.

```
void dotProduct(float a[N], float b[N], float &c) {
  float d = 0;
  for (int i = 0; i < N; i++) {
    d+=a[i]*b[i];
  }
  c=d;
}
```

An HLS tool can generate the corresponding HDL code for this naïve software implementation of the *dotProduct* function. However, all the loop iterations are run sequentially. As each iteration requires two read operations that can be performed in parallel and an addition after a multiplication, it takes $l=(1+3+3)=7$ clock cycle to complete, as shown in Figure 2. Considering this timing, the latency of the function would be about $(N*l)$ clock cycles which means $71.68\mu s$ if $N=1024$ and $f=100\ MHz$ (the design frequency).

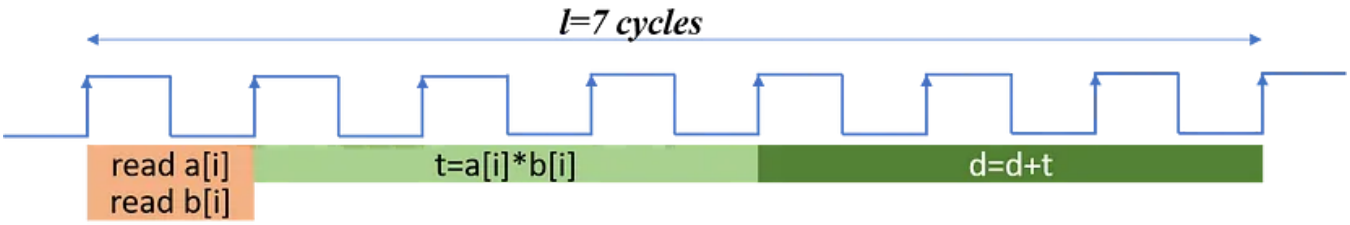


Figure 2 One Loop iteration timing

Now if we add a pipeline directive to the loop as shown in the following code, then the HLS compiler uses the pipelined microarchitecture to implement this loop which the resulting timing diagram is shown in Figure 3. The pipeline microarchitecture allows the loop iterations are executed with some overlap in parallel if enough computing and memory resources are available and the overlap does not violate the data dependencies inside and among iterations.

```
void dotProduct(float a[N], float b[N], float &c) {
  float d = 0;
  for (int i = 0; i < N; i++) {
#pragma HLS pipeline
    d+=a[i]*b[i];
  }
  c=d;
}
```

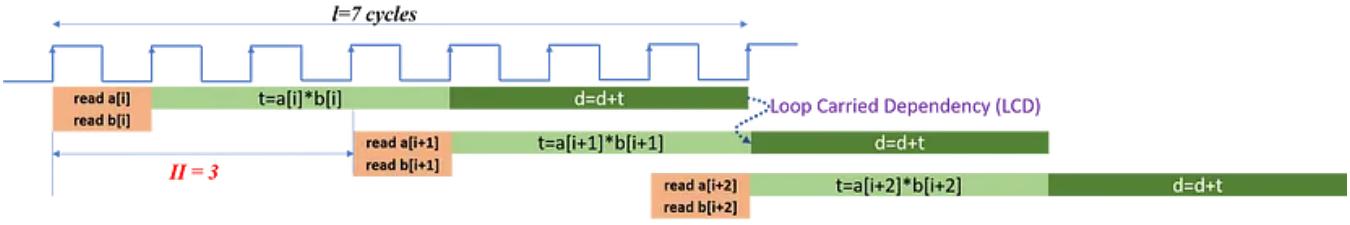
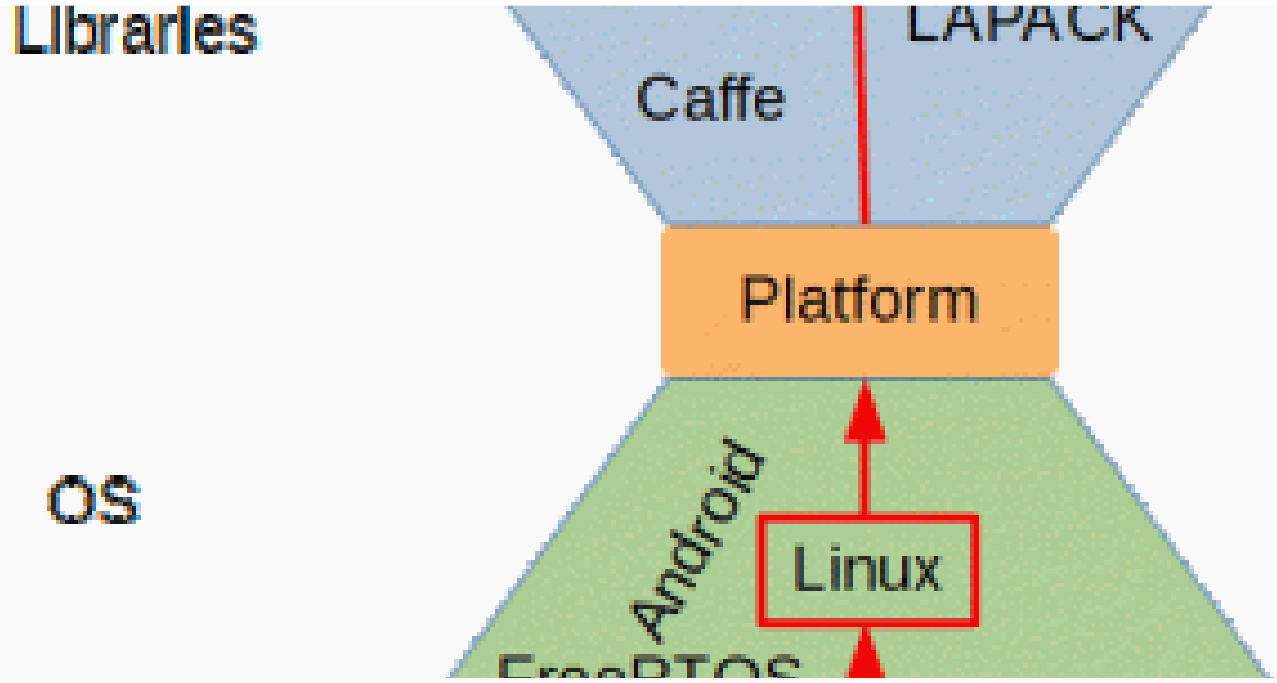


Figure 3 Pipelined loop timing diagram

In this example, the loop carried dependency (LCD) on variable *d* (LCD is the data dependency among iterations) results in a three clock cycles gap between the execution of two consecutive iterations. This timing interval is called Initiation Interval (*II*) which is the main factor determining the performance of a pipelined loop. In the ideal case, the *II* should be 1 for the maximum performance. In this case, if *N=1024* and *f=100MHz*, the loop requires *((N-1)*II+l)* clock cycles to finish, that means *30.76*µs, which is *2.33* times faster than the naïve implementation.

The necessity of adding compiler directives to the C/C++ description reduces the adoption of HLS design flow by software engineers. To alleviate this impact, framework-specific synthesis (FSS) approaches are introduced by academia and industry considering frameworks such as OpenCL, CUDA, P4, among others. These new high-level descriptions reduce the number of compiler-directives required for high-performance implementation.

Here, I leave a question for interested readers. What is the minimum number of clock cycles required to implement the above *dotProduct* function if *a* and *b* vectors are stored in the FPGA BRAMs? (You can consider Ultra96 FPGA board as the target hardware)



Mohammad Hosseinabady

Embedded Hardware Accelerator with Xilinx Vitis: Part 2: Create a Linux-based Platform

In the previous blog, I briefly explained the concept of platform-based design in the context of embedded FPGA. And the problems that it...

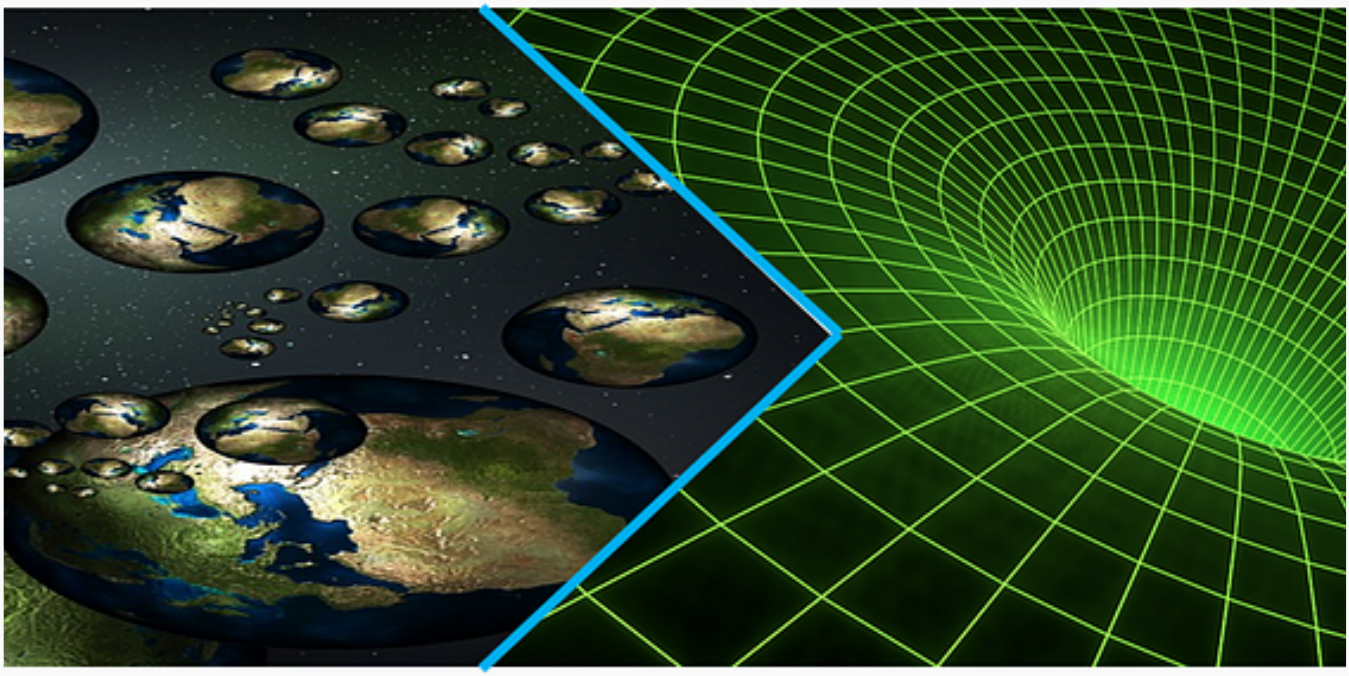
2 min read · Nov 15, 2019

Mohammad Hosseinabady

Embedded System HLS with Vitis: Communication with Software

· 6 min read · Dec 21, 2019

21

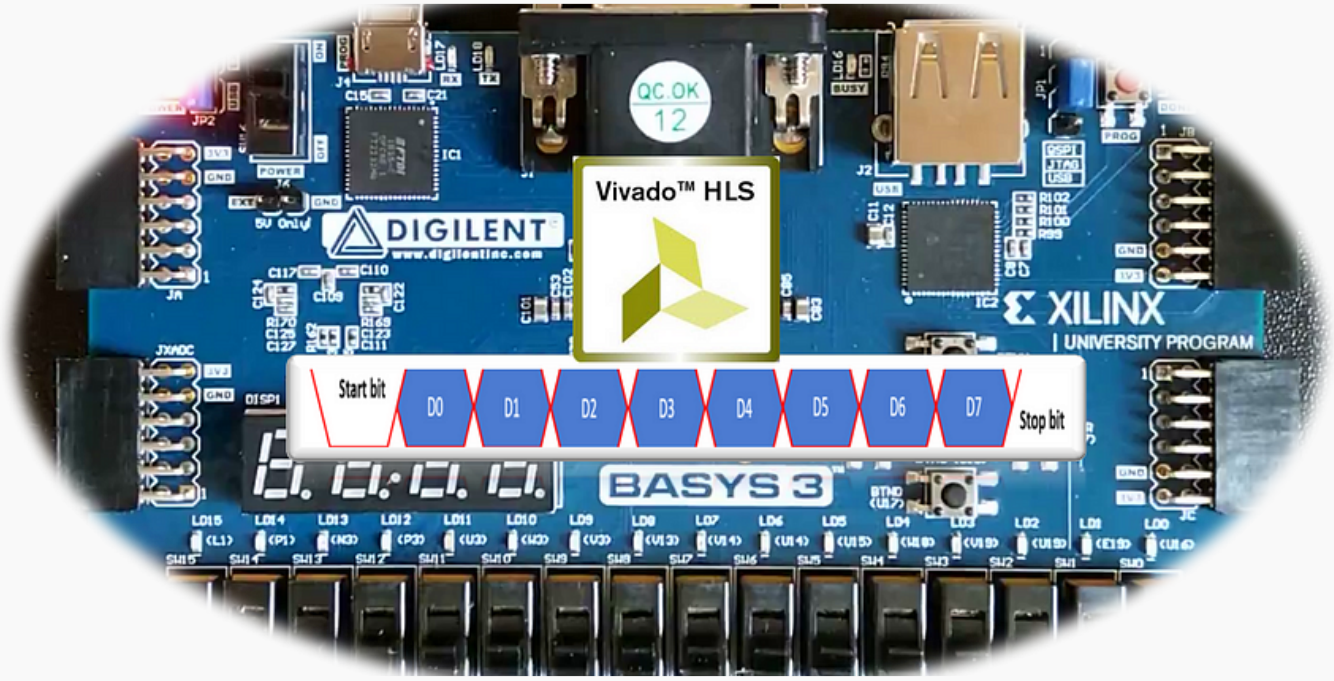


Mohammad Hosseinabady

Fundamentals of High-Level Synthesis Part 3: From Concurrency to Parallelism (Map Pattern)

In the previous blog, I explained the difference between concurrency and parallelism concepts in high-level synthesis. Whereas the...

5 min read · Nov 11, 2019



Mohammad Hosseinabady

UART Transmit with HLS for FPGA

Like my previous projects, this one also demonstrates that “Designing digital systems with HLS for FPGA is fun”. If you are interested in...

· 3 min read · Oct 26, 2020

See all from Mohammad Hosseinabady

Recommended from Medium

smoul in Practice in Public

How to be in the top 1% in 2024

8 habits to help you be a better version

7 min read · Dec 15, 2023

20K 380

Benoit Ruiz in Better Programming

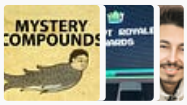
Advice From a Software Engineer With 8 Years of Experience

Practical tips for those who want to advance in their careers

22 min read · Mar 21, 2023

13.5K 251

Lists



Staff Picks

558 stories · 643 saves

Stories to Help You Level-Up at Work

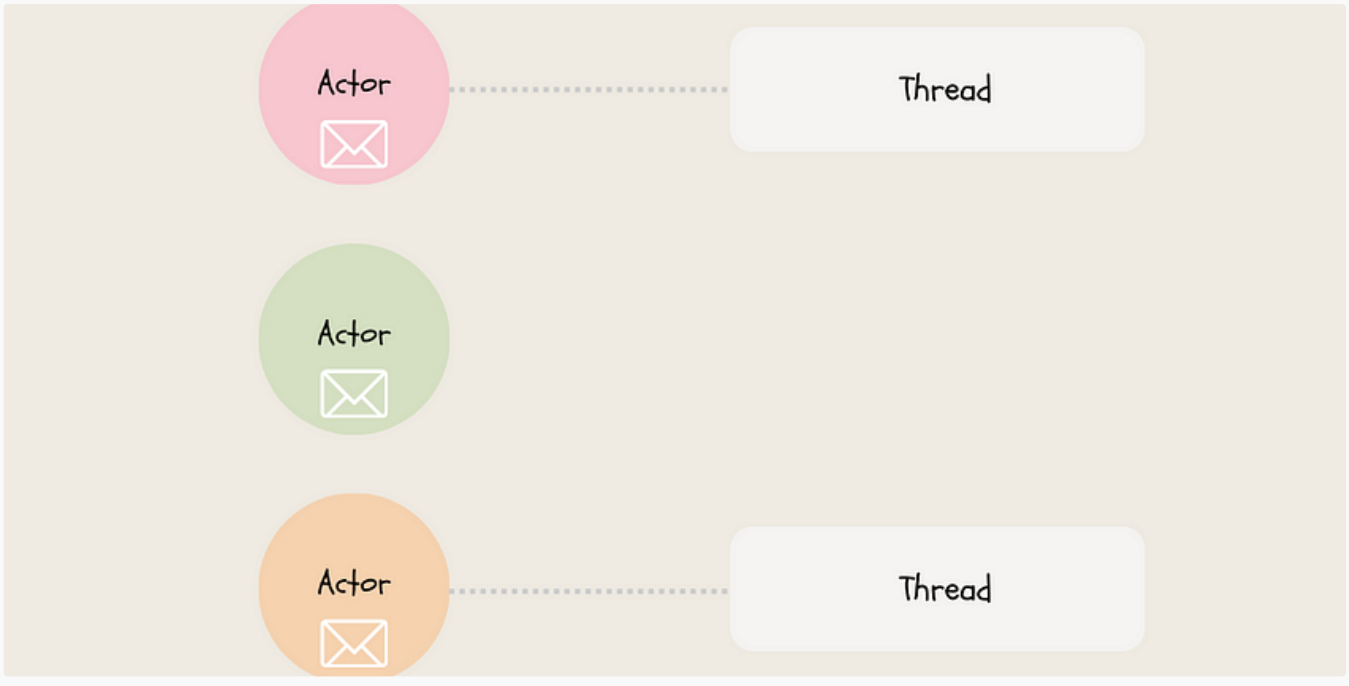
19 stories · 420 saves

Self-Improvement 101

20 stories · 1221 saves

Productivity 101

20 stories · 1118 saves



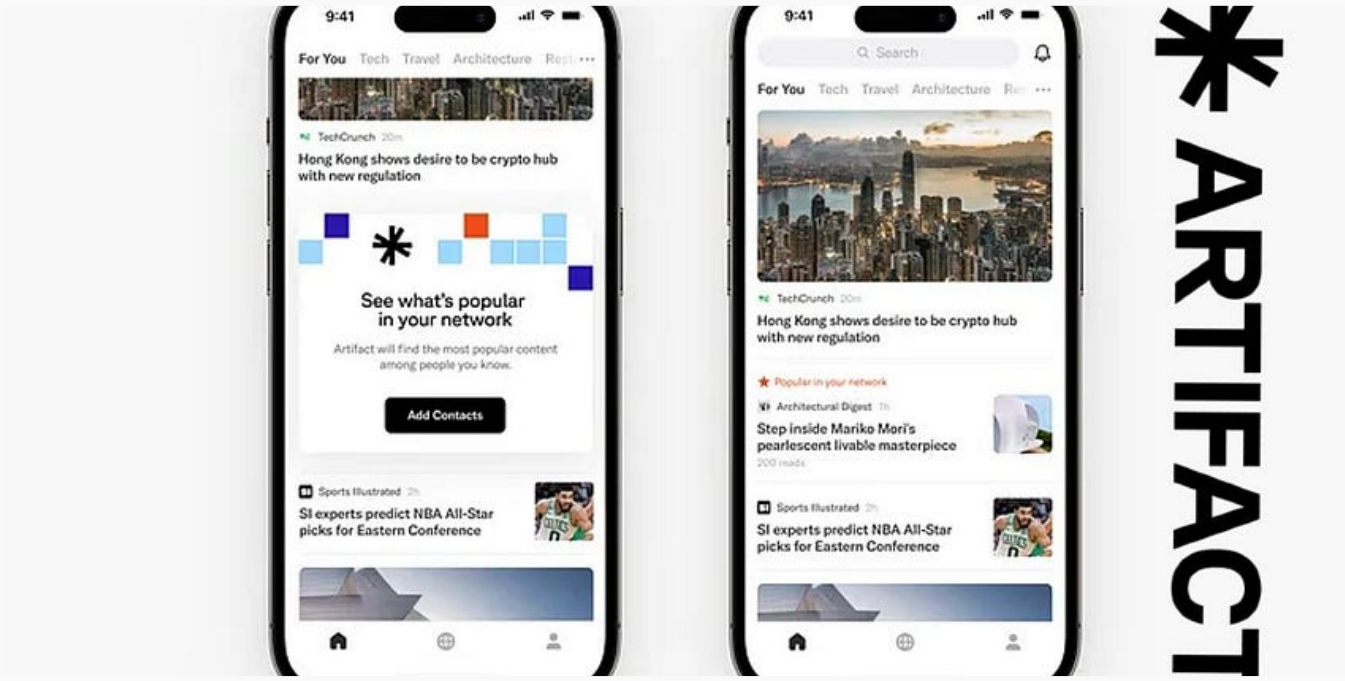
Nidhey Indurkar

How did PayPal handle a billion daily transactions with eight virtual machines?

I recently came across a reddit post that caught my attention: ‘How PayPal Scaled to Billions of Transactions Daily Using Just 8VMs’...

7 min read · Jan 1

2K 21



Gowtham Oleti

Apps I Use And Why You Should Too.

Let's skip past the usual suspects like YouTube, WhatsApp and Instagram. I want to share with you some less familiar apps that have become...

10 min read · Nov 15, 2023

14.5K 247

Unbecoming

10 Seconds That Ended My 20 Year Marriage

It's August in Northern Virginia, hot and humid. I still haven't showered from my morning trail run. I'm wearing my stay-at-home mom...

· 4 min read · Feb 17, 2022

73K 1044

Companies have found that "Agile", as it is sold, delivered, and explained to them, does not work. You can blame them if you like, or you can blame the Agile community for not packaging the right kinds of learning and support. But regardless, "Agile" as we know it is dead. And Scrum will go with it.

But companies still need _agility_. Real agility. That has been our focus.

Real agility is mostly behavioral, and in particular, it is driven by the behaviors of leaders. Leadership is the big glaring hole in the Agile Manifesto. It is like trying to make concrete without water. No wonder "Agile" did not work.

That's why Agile 2, which reimagined what "Agile" should have been,

Tamás Polgár in Developer rants

Agile has failed. Officially.

Either I'm a gifted oracle, and all of my friends are, or Agile really was just a stupid idea to begin with. After many years of agony...

2 min read · Dec 3, 2023

13.1K 407