Member-only story

Fundamentals of High-Level Synthesis Part 2: Concurrency vs Parallelism

Mohammad Hosseinabady · Follow

5 min read · Oct 4, 2019

Fig. 1 source: geralt

Concurrency and parallelism are two main concepts in high-level synthesis (HLS) design flow that their understanding is crucial in implementing an algorithm efficiently on FPGAs.

Whereas the concurrency is a concept at the level of algorithm, parallelism is a hardware-dependent concept. For example, Let's consider the two L1 and L2 *for*-loops in the following snippet code. As there is not any data or control dependency between them, they are concurrent and potentially can be executed in parallel.

However, their parallel execution depends on the underlying computational platform.

```
L1:for (int i=0; i < N; i++)
     c[i]= a[i]+b[i]
L2:for (int j=0; j < M; j++)
     d[j]= e[j]*f[j]
```

If there is only one ALU to perform the addition and multiplication, then these loops should be executed sequentially one after the other or their iterations can be interleaved and run serially. If there is more ALUs available, then the two loops can be performed in parallel. Therefore, *the amount of parallelism extracted from a concurrent code depends on the underlying hardware resources*. Note that as loop iterations are independent, the concurrency can also be explored among the iterations of each loop. The interested reader can think about different cases of parallelism in implementing the loop iterations. Fig. 2 illustrates only a few cases.

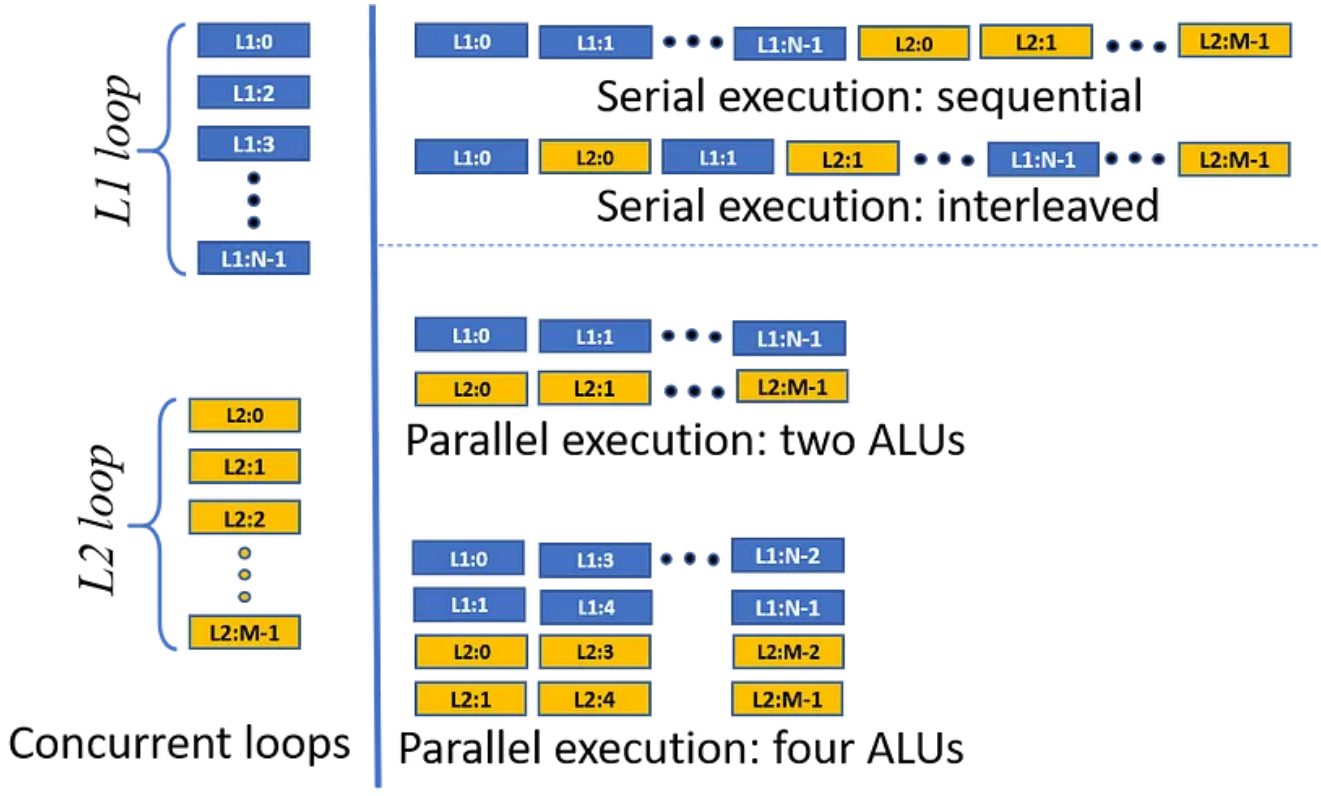
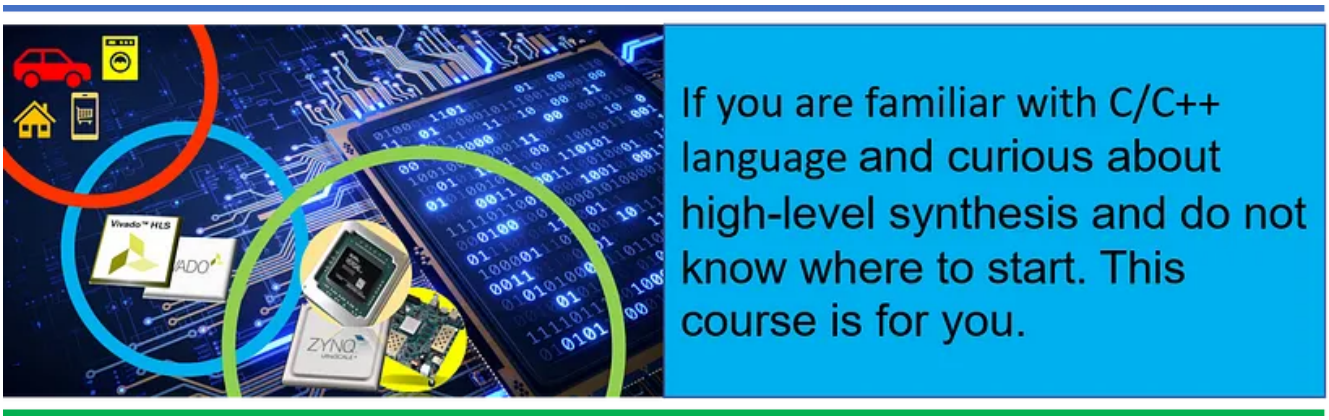


Fig. 2 Concurrency vs Parallelism

The main goal of an ideal HLS tool is transforming the concurrency in an algorithm into parallelism in the underlying hardware, automatically, without any help from designers. If there were an infinite amount of hardware resources without any

overhead (or cost), then transforming the concurrency to parallelism would be a trivial task. But, because of limited computational and memory resources on the target FPGAs and their associated costs, the HLS tool should solve difficult optimisation problems, which several of them are NP-hard.



As shown in Fig. 3, **constraints** describe the hardware resources limitations, and the HLS tools use them as a model of the hardware (along with the cost model). Since HLS tools are not ideal, designers usually should help the compilers to do their tasks efficiently. This help is generally added to the code in two forms: *Firstly*, the code should follow a coding style suitable for the hardware. *Secondly*, designers should add compiler directives (e.g., pragmas) to the algorithm code to tell the compiler more information about the appropriate underlying hardware parallelism of the final hardware microarchitecture.

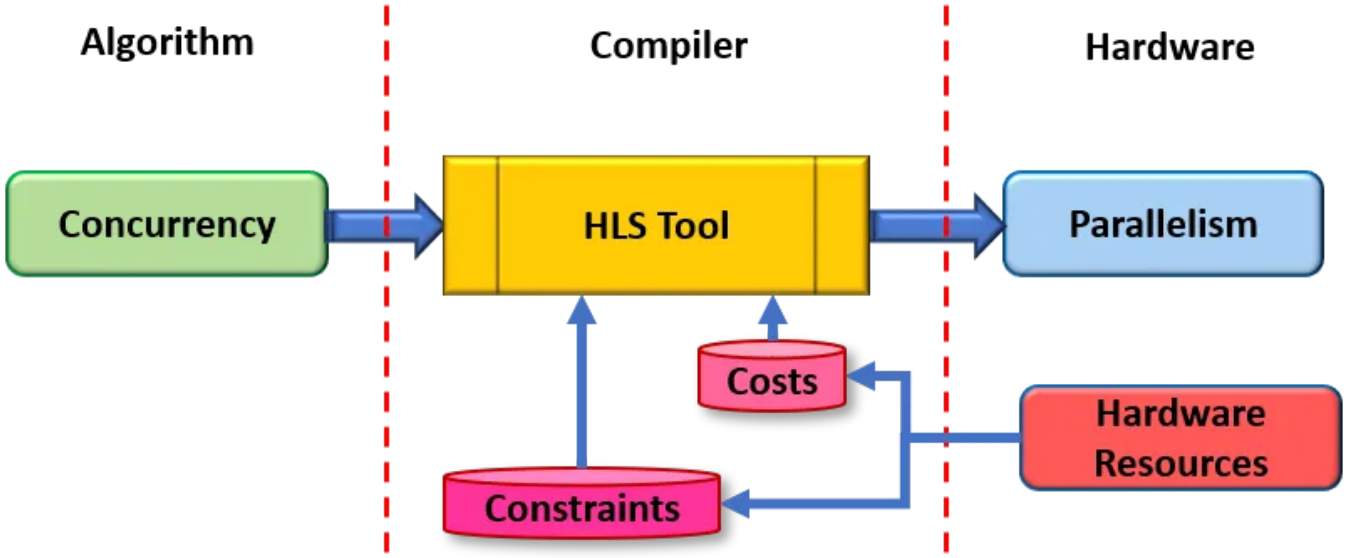


Fig. 3 Concurrency-Parallelism transformation

Note that, it is better to understand the concurrency in an algorithm and techniques to model that. Having these models in mind, designers can systematically choose an

efficient coding style and proper compiler directives.

Fig. 4 source: Lee Davy

. . .

Concurrency is all about the variable dependency among statements, instructions, or blocks in an algorithm that determines they can perform together. This dependency comes in two different forms: *data dependency* and *control dependency*.

Data dependency: If two statement accessing the data in the same variable (whether for reading or writing), then there is a data dependency between them. This dependency can have different forms. The simplest one is when one statement writes to a variable, and another read that variable. For example, consider the following instructions in an algorithm code. The first instruction writes into the *x* variable. And the second one reads the result. This dependency is called *flow* (or *true*) dependency and prevents the two statements from performing at the same time. Therefore, they are generally not concurrent.

```
…
 s1: x = b + c;
 s2: y = a + e;
…
```

The data dependency can be modelled by a graph illustrated in Fig. 5, which shows the execution flow of statements. As can be seen, statement *s2* must be run after *s1*.

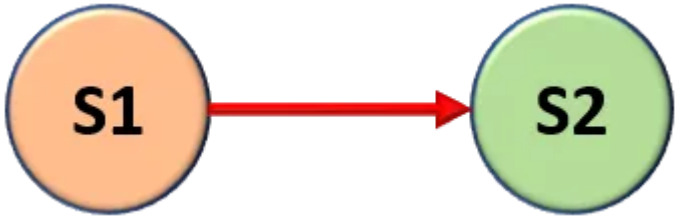


Fig 5. Data dependency graph

Control dependency: If the execution of two statements is controlled by a third statement such as a conditional *if*, then two statements may not perform concurrently. The following code is a simple example.

```
…
 s1: if (a < 2) {
 s2:   x = y + z;
 s3: } else {
 s4:   t = p + t
 s5: }
 s6: z=x+t ;
…
```

Although the s2 and s3 statements are independent, they cannot be concurrent as always only one of them should be executed. In this case, the s1 instruction controls this selection.

. . .

In summary,

> *— Concurrency is an algorithm-related concept, whereas parallelism is a hardware-related.*
> *— Selecting proper parallelism increases the performance*
> *— Concurrency is transformed to parallelism by a compiler*
> *— Designers have direct control to describe the concurrency embedded in an algorithm*
> *— Concurrency should be defined in such a way that compilers can easily explore and exploit the parallelism*
> *— Whereas the coding style help to describe the concurrency the compiler directives help the compiler to realise the parallelism*

Now we have two main problems to solve.

1- How to choose a suitable coding style to describe the concurrency in a given algorithm

2- How to use compiler directives (e.g., pragmas) to help the compilers to select the most efficient form of parallelism

I will try to handle these questions in future blogs.

Programming | Hardware Accelerator | Hardware Startup | Embedded Systems | Hackster

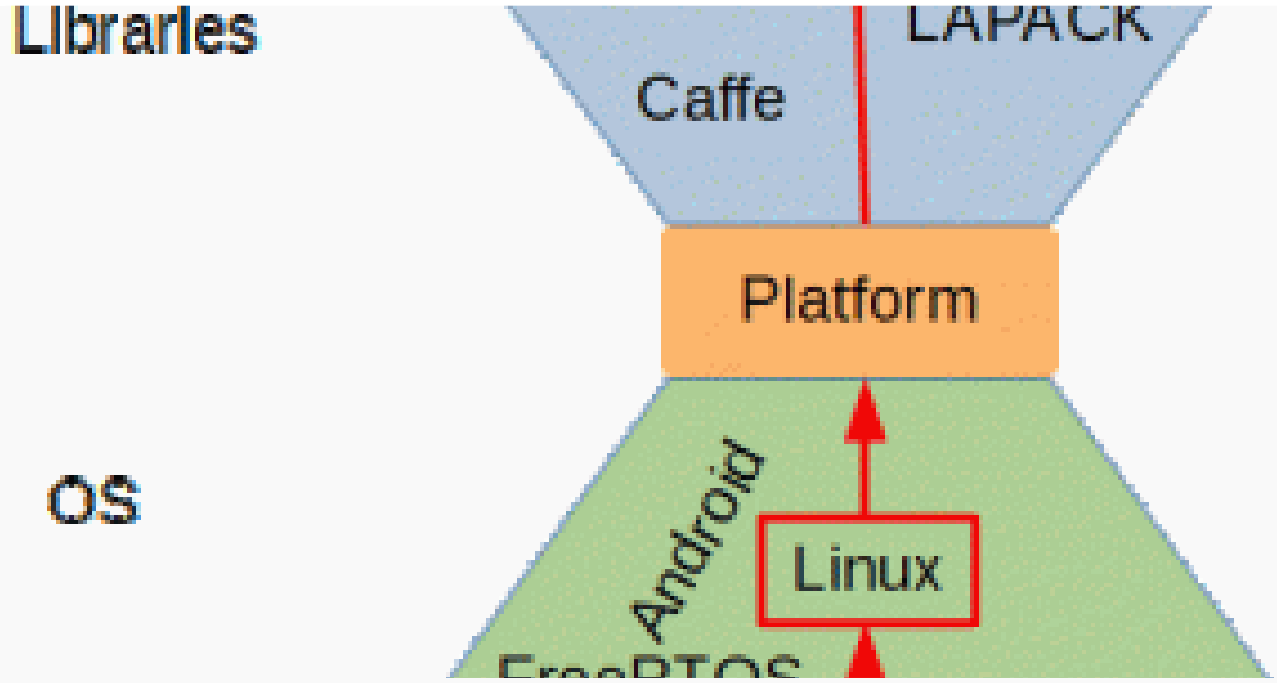
Follow

Written by Mohammad Hosseinabady

38 Followers

Designing digital systems and accelerating functions with HLS for FPGAs are fun.

More from Mohammad Hosseinabady



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...

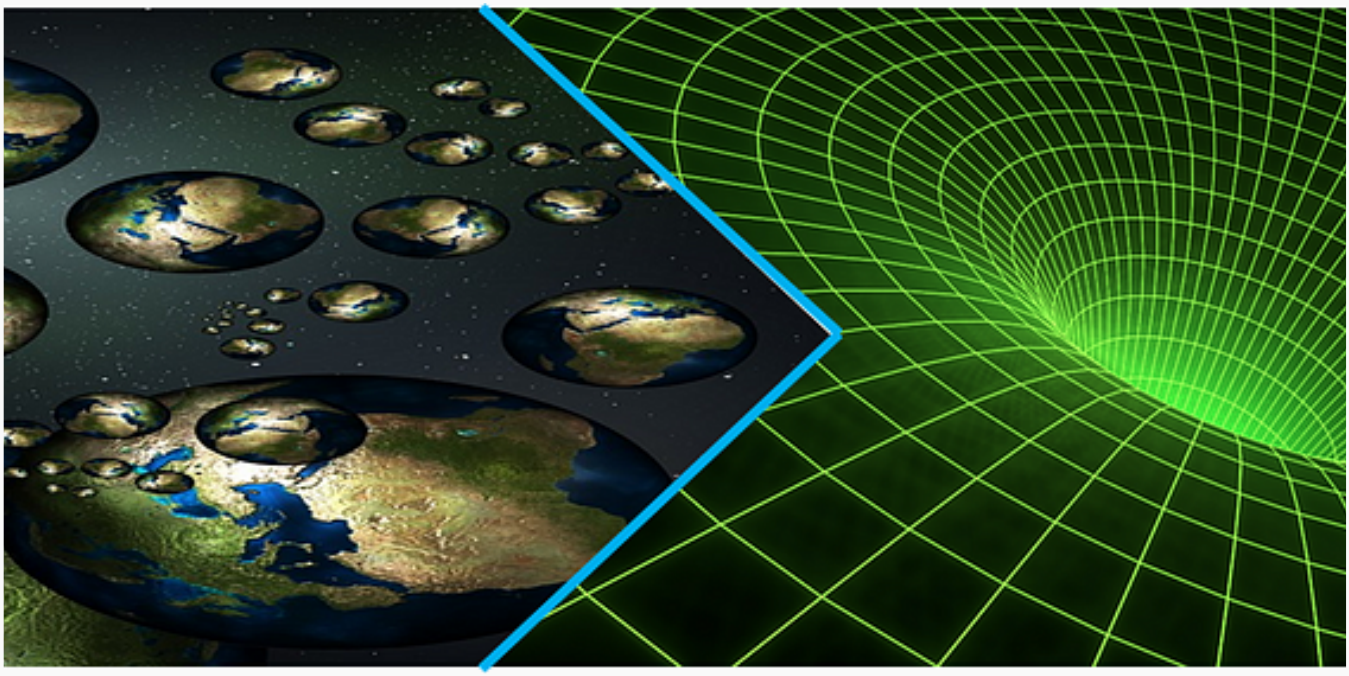
2 min read · Nov 15, 2019

Mohammad Hosseinabady

Embedded System HLS with Vitis: Communication with Software

✦ · 6 min read · Dec 21, 2019

21

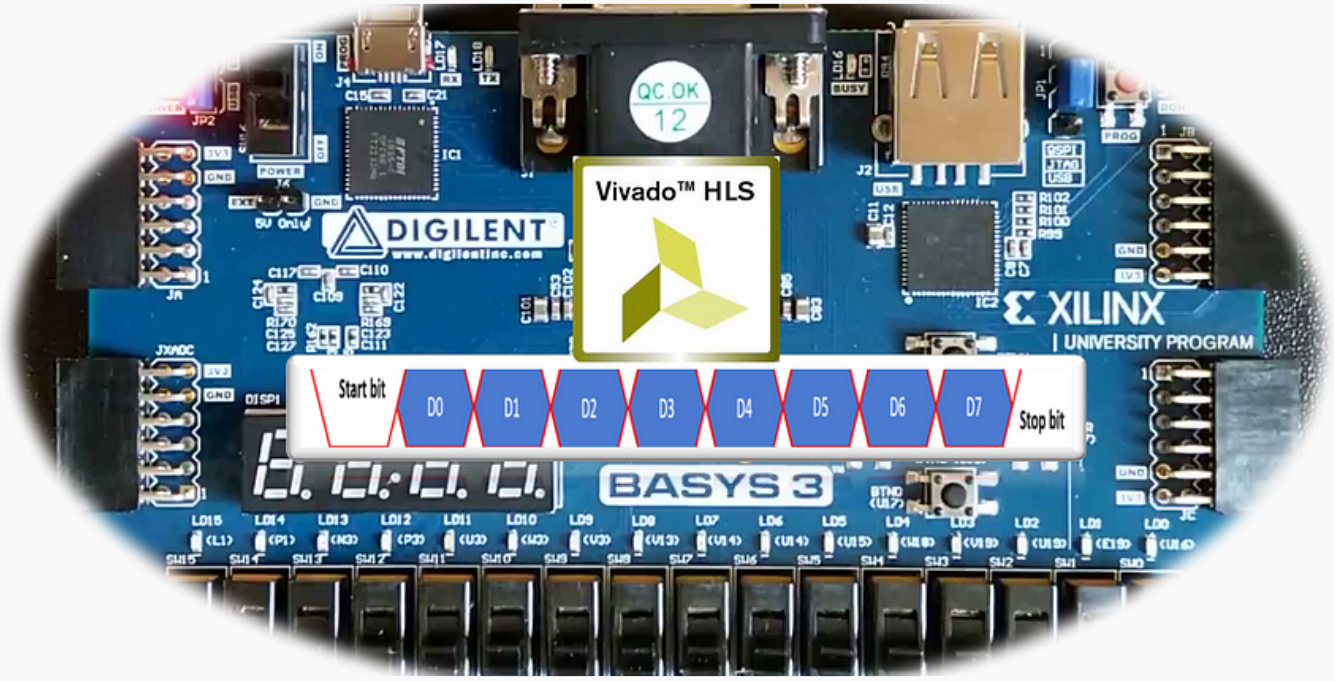


Mohammad Hosseinabady

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

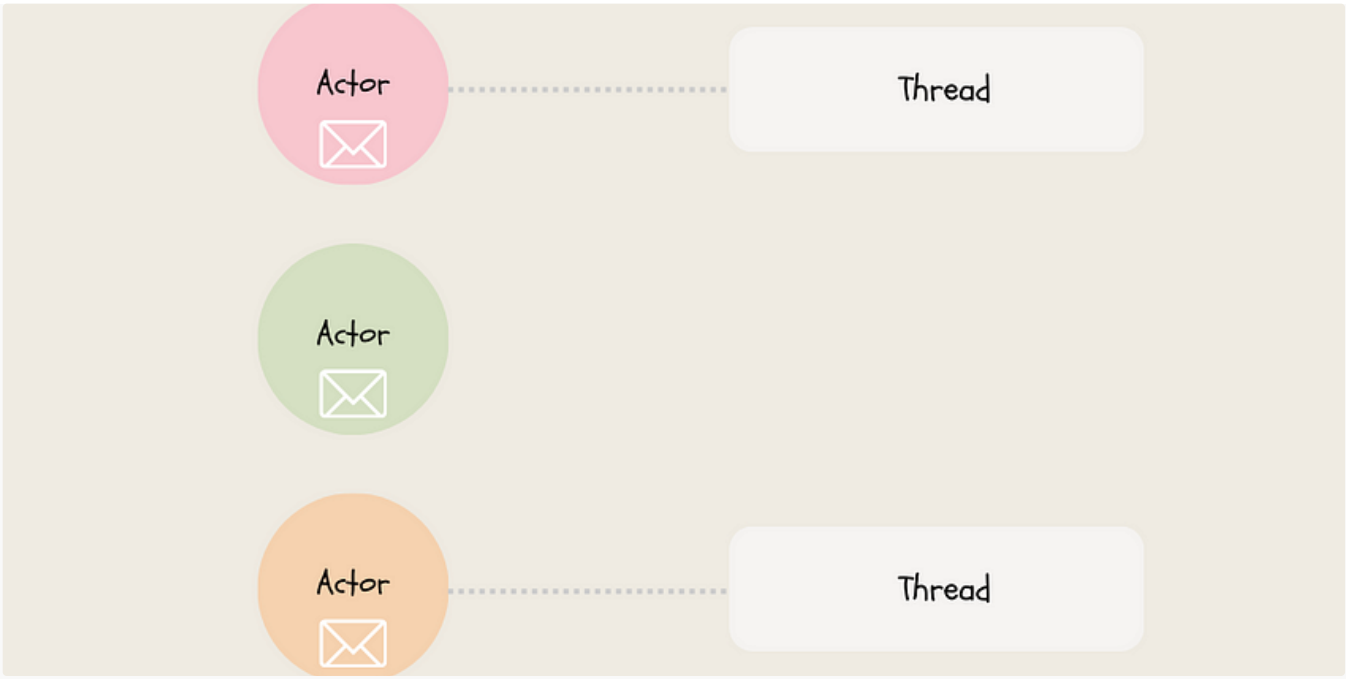
UART Transmit with HLS for FPGA

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

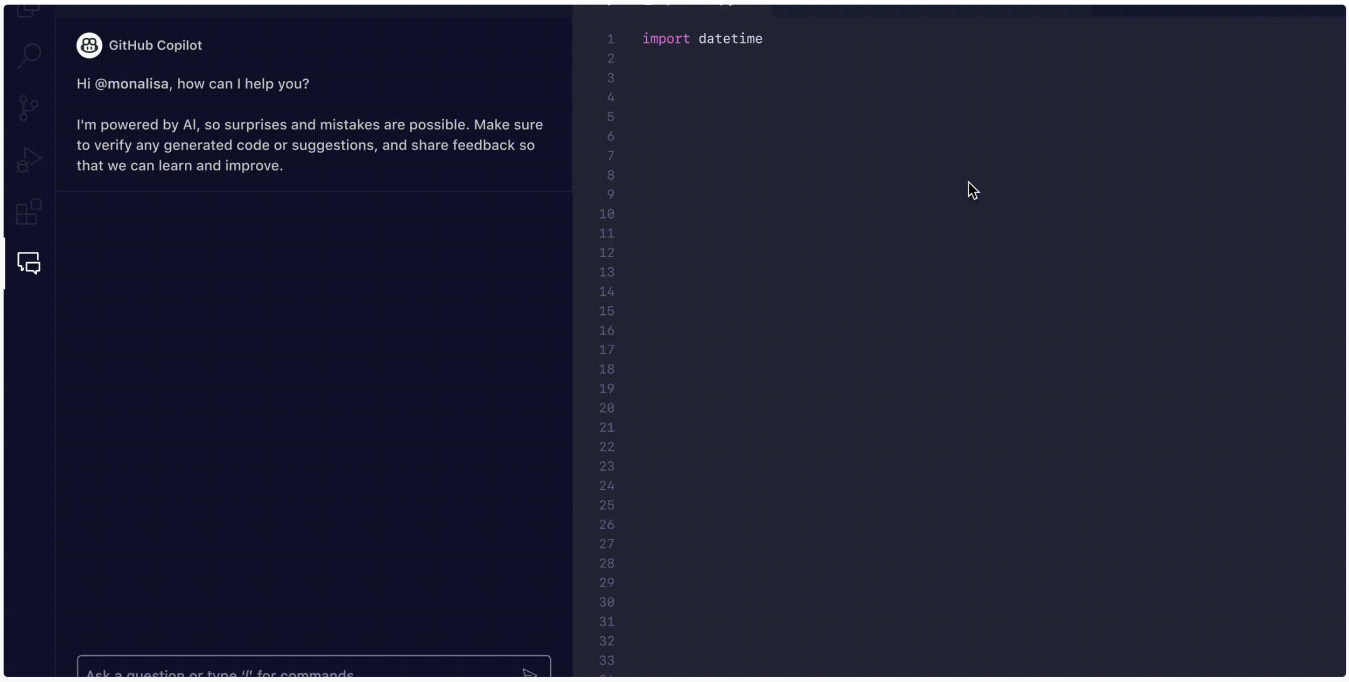


Nidhey Indurkar

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



Jacob Bennett in Level Up Coding

The 5 paid subscriptions I actually use in 2024 as a software engineer

Tools I use that are cheaper than Netflix

· 5 min read · Jan 4

2.6K 39

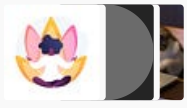
Lists

General Coding Knowledge

20 stories · 801 saves

Coding & Development

11 stories · 385 saves



Stories to Help You Grow as a Software Developer

19 stories · 718 saves

ChatGPT

23 stories · 400 saves

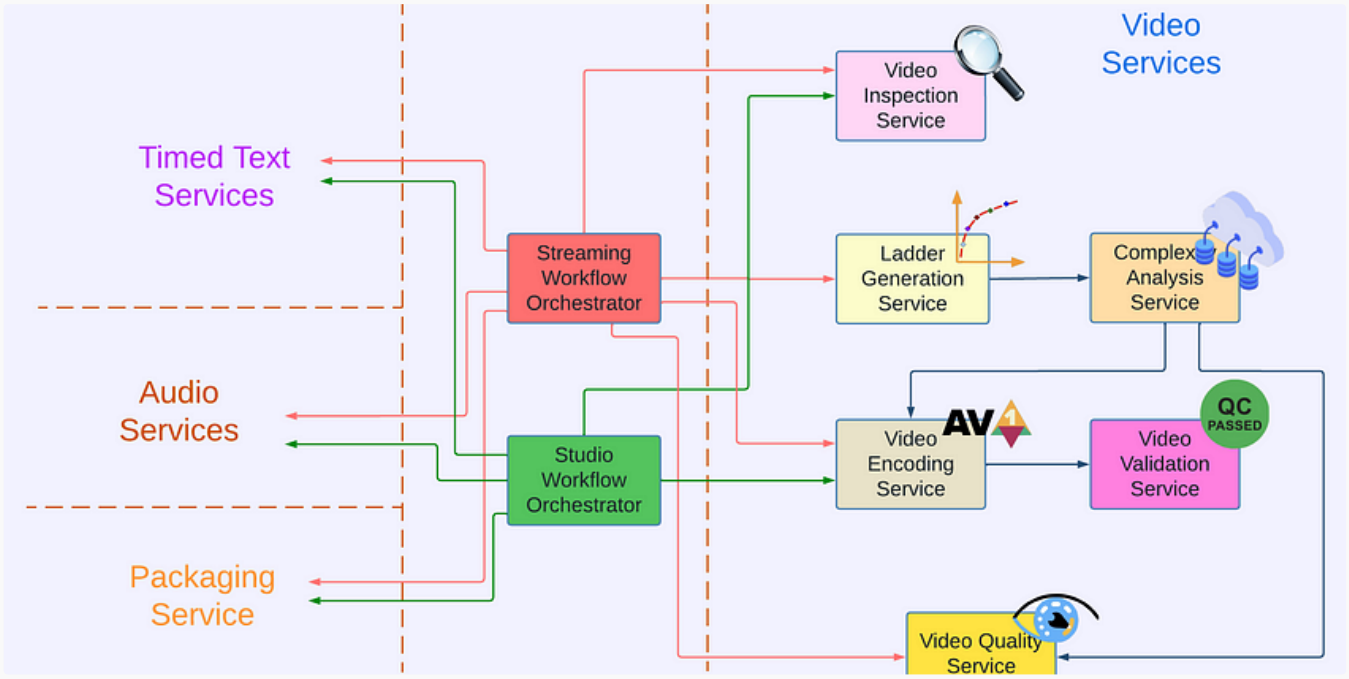
Shuai Guo in Towards Data Science

Modeling Dynamical Systems With Neural ODE: A Hands-on Guide

Concepts, case studies, step-by-step implementations

· 22 min read · 6 days ago

410



Netflix Technology Blog in Netflix TechBlog

Rebuilding Netflix Video Processing Pipeline with Microservices

This is the first blog in a multi-part series on how Netflix rebuilt its video processing pipeline with microservices, so we can maintain...

9 min read · Jan 11

1K 23

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

Yusef Mohamadi

Unveiling the Exciting Enhancements in Go 1.22: A Developer’s Delight

New Features that You Should Expect in Go 1.22.

5 min read · Jan 10