Extracted from:

Build a Binary Clock with Elixir and Nerves

Use Layering to Produce Better Embedded Systems

The Pragmatic Bookshelf

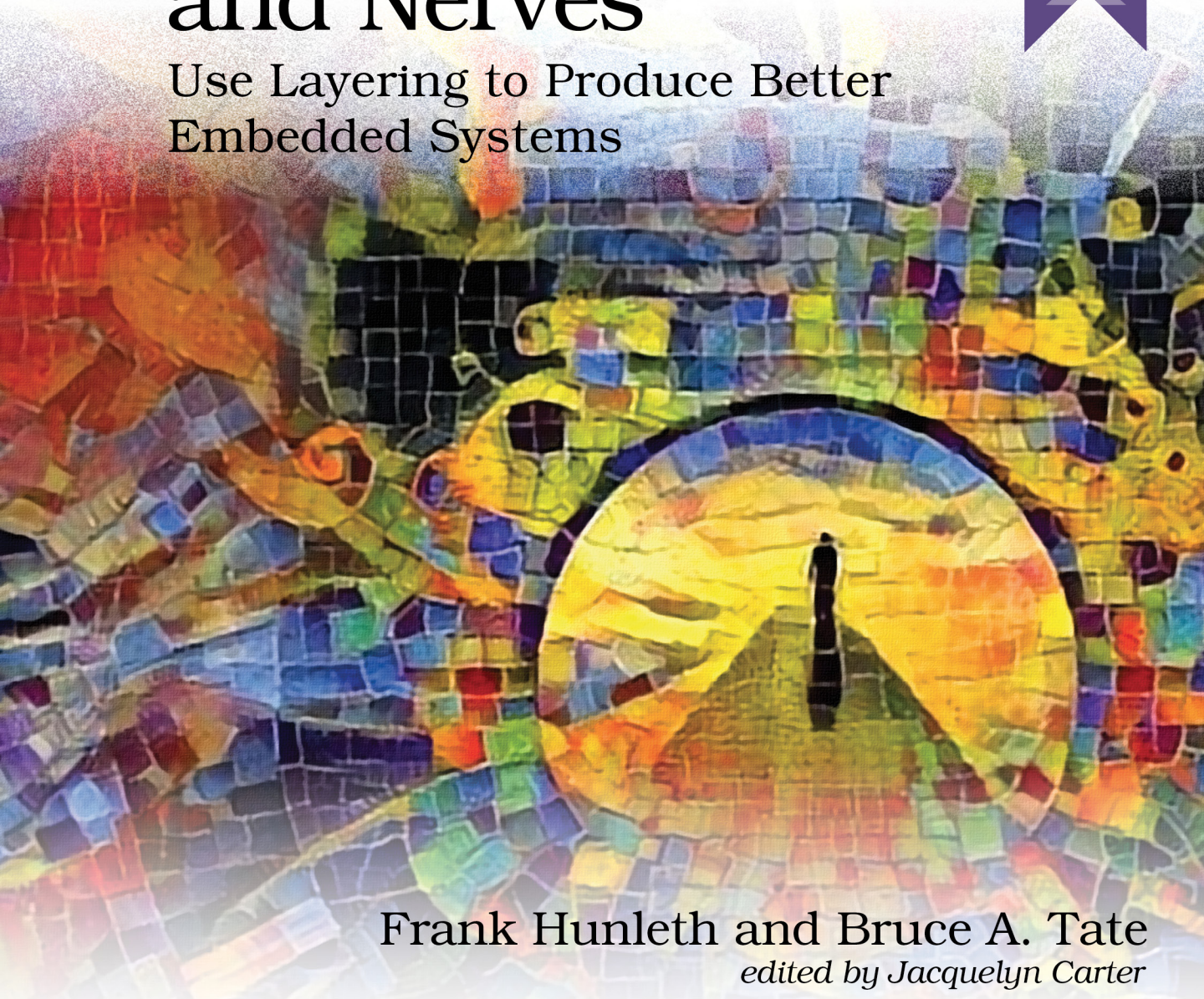
Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Build a Binary Clock with Elixir and Nerves

Use Layering to Produce Better Embedded Systems



Frank Hunleth and Bruce A. Tate

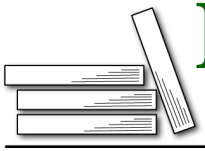
edited by Jacquelyn Carter

Build a Binary Clock with Elixir and Nerves

Use Layering to Produce Better Embedded Systems

Frank Hunleth
Bruce A. Tate

The Pragmatic Bookshelf
Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-923-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—November 3, 2021

Introduction

This book is one of a series of books about Elixir and Nerves. Each book in this series will teach one fundamental software concept, and build one complete project using the Elixir language on Nerves. Elixir[1] is a highly concurrent and reliable functional programming language, and Nerves[2] is a tool for embedding programs on it to build the internet of things, IoT. These devices are small special purpose computers used to control hardware. They show up in cars, appliances, industrial controllers, and more.

In this book, you'll build an IoT device, a *binary clock* that cryptically tells time by lighting a series of LEDs, and gets the current time from the network. Pure Elixir code will control the clock's display. While a clock is a relatively simple machine, it has many of the same parts as real-life hardware projects. Throughout this book, you'll use the very same principles to organize the software in your own clock as you'd use in any other program.

How to Read This Book

This books takes you step by step through the process of building an end-to-end binary clock, from the layers of software to the LEDs. If you choose to omit steps, you could wind up with a non-functional end product.

Who This Book Is For

This book is for any Elixir programmer that is comfortable with the basics of the programming language and is interested in dabbling in the world of embedded systems. No soldering or deep hardware experience is necessary, given that you will be working with off the shelf plugin and play hardware.

1. https://elixir-lang.org
2. http://nerves-project.org/

Who This Book Isn't For

If you have just a little experience with Elixir, don't worry. We'll help you with some of the more advanced concepts. If you are just getting started, you might want to put this book aside for a bit and pick up *Programming Elixir 1.6 [Tho18]*.

While Elixir 1.6 came out a few years ago now, the core language has not changed much in that time and as such the book will help you develop a solid Elixir foundation. After you read that book, feel free to pick this one up again and get your hands dirty with an IoT based project.

Building the Project

Being able to build and run your application code will be key to understanding the concepts outlined in this book. As such it is important that you have the items outlined in the next couple sections so that you have everything you need to complete the binary clock.

Software Requirements

Embedded hardware aside, you will need the following things:

- Elixir version 1.12 or greater
- A Linux, MacOS, or Windows machine to do your development on
- A wireless access point for your local area network

If you have all of those items then you are good to go from a development machine perspective, and all that is needed is the Nerves related hardware.

Hardware Requirements

While there is some flexibility with what hardware (like what version Raspberry Pi) you can buy and from where, the following items were used by the authors:

- Raspberry Pi Zero W with headers
- MicroUSB connection cables
- 4GB+ MicroSD card
- MicroSD card reader
- 20 LEDs of various colors
- Some resistors
- TLC5947 constant current driver with a SPI interface
- Jumper wires, breadboard, headers, and ribbon cables

If you don't know what these things are or where to buy them, fear not as we explain all of this in the first few chapters. You can drift away from these

parts, but you might need to change the instructions in the book slightly to get things to run.

Online Resources

All of the code for this project can be found online in the GitHub repository.[3] If you need any assistance for all things Elixir and Nerves, be sure to check out the Elixir Forums[4] where you will find a vibrant community ready to help.

With those bits of housekeeping aside, we can make a plan.

Our Plan

It's often hard to get started when working with hardware because there are so many small things that can go wrong. For that reason, it's important to establish several small quick wins instead of making one full project work end to end. So it is with Nerves.

We're going to direct you to the excellent Nerves documentation to get started. Then, we'll shift toward building a networked project that will eventually control our clock. Here's what the plan looks like in detail.

Burn Firmware

Nerves works by combining the Elixir programs that you write with everything else that a specialized embedded computer needs to run. An increasing number of these tiny devices actually run the Unix operating system, and Nerves is built to run on them.

You'll start by installing a firmware program written by the Nerves team on a an embedded computer, called a *target*. This first step will verify that you can use your Nerves tool chain to install a program on the target's firmware chip. Then, you'll snap the firmware chip into your target, and connect to it using a USB cable so you can remotely access an Elixir shell.

- You have a working Nerves tool chain for burning firmware
- You can use your host to debug your target

With working firmware, we can shift to the hardware.

3. https://github.com/groxio-learning/thnerves
4. https://elixirforum.com

Make a Circuit

The first step in building a complex hardware project is to build a simple one that works. It makes sense, then, to build the simplest of circuits, a single LED that you'll control with your target. Once you've done that much, you'll connect to your target from your development computer, called a *host* to control the LED. This step will demonstrate that you can build circuits, install them on a target, and control them with a host. When you're done, you'll know:

- You have a working circuit
- You can exercise your circuit using IEx to talk to programs on firmware

With a working firmware process and a circuit, the next step is to write a simple program.

Build a Program In Layers

After burning an existing project onto firmware, you'll write your own mix project. You'll add *compilation for a target* to your tool chain. Nerves will build an image that has your program and everything else your embedded device needs.

After you've built a program to blink an LED, you'll build in networking so you can push software changes and share data with the outside world. We'll track a common time. When you're done, you'll know how to:

- Write your own programs, and then burn them onto firmware
- Build software in layers, with functional cores that handle logic and boundaries to handle external interfaces
- Connect to your embedded device from networked computers to burn firmware, collect data, or use circuits you build, like your LED circuit

When this step is done, you'll have a working Nerves skeleton. Your *host* will have proven tools to upload firmware. Your *target* will have a working *circuit*. Finally, your *program* will control the *target*. Those three tiny steps will reap huge rewards in your confidence in a working system, and demonstrate any problems before you have to move on.

Bruce says:

Write Your Software in Layers

The trick to handling complexity is not eliminating it, but figuring out ways to deal with a little bit at a time. That's why you should write your programs in layers. Your project will be complex, but the module in your editor window doesn't have to be.

After we've established working firmware uploads, hardware, and software, we can move on to the next part of the book, building the clock. We'll save that plan for later.

Every future Nerves step will have these steps. You'll build circuits, write or update layered programs, and then push them to your firmware.

That's enough planning. We're ready to build a clock.