Intro to embedded systems and drivers as selected subject

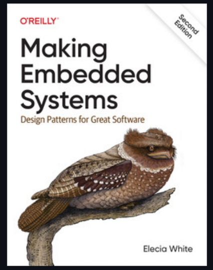
DHT22 Temp/humidity and LCD1602 display on Pi Pico W microcontroller

by: **Simon Lalonde**

For: **IFT-769** (Theoritical concepts CS)

📖 **Project overview (1/2) -** Read 'Making Embedded Systems' by Elecia White

Making Embedded Systems 2nd edition by Elecia White



Book overview:

- **Introduction** to embedded systems architecture and design
- How to work with various **I/O** devices (sensor, display, etc.)
- Learn how to **optimize** and **debug** within resource constraints
- **Advanced** topics like **RTOS**, **networking**, **security**, etc.

White, Elecia. Making Embedded Systems. 2nd ed., O'Reilly Media.

🛠️ **Project overview (2/2) -** Apply the concepts from 1^{st} half of reference book

Make a **Temperature** 🌡️ and **humidity** 💧 station with DHT22 sensor and LCD1602 display on Raspberry Pi Pico W microconstroller.

➡️ **Design** a simple embedded system with a microcontroller.
➡️ **Learn** to work with I/O on a microcontroller.
➡️ **Write custom C drivers** for each peripheral.

<u>(Optional goal)</u>
Take advantage of the Pico W microcropressor's 📡 chip and run a **web server** to **display** the data on a web page via local network.

💬 **Project overview (3/3) -** Present and apply relevant concepts from the main reference

<u>**Relevant concepts**</u> (from the 1st half of the book):

- Create **system diagram** and **flowchart** for the project (ch. 2)
- Choosing and understanding **hardware** (ch.3)
- **I/O** and **interrupts** (ch. 4-5)
- Drivers and **communication protocols** (ch. 7)
- **Flow** of activity and **hollistic system** view (ch.6 and 8)

🎯 Project goals

1. **Understand** the basics of embedded systems and drivers.
2. **Learn** to work with I/O devices on a microcontroller.
3. **Write** custom C drivers for each peripheral.
4. **Apply** the concepts from the reference book to the project.
5. **Present** and **apply** relevant concepts from the main reference.

📅 Project timeline - (1/2)

Theoritical concepts

- Read a chapter of the book every week

Applied Project

- Write **System diagram** and **flowchart** for the project
- **Choosing** and **understanding** hardware
- Setup **development environment** and **toolchain**
- Start **writing** the DHT22 driver

📆 **Project timeline** - End-of-term objectives

Theoritical concepts

Continue reading the book past the applied objectives.

Applied project

- **Finish** the DHT22 driver
- **Write** the LCD1602 driver
- **Integrate** the drivers and **test** the system
- (OPTIONAL) Run a **web server** to display the data

🙋 What are embedded systems?

- **Dedicated** computing devices that are part of a larger system. They are designed to perform a specific task or set of tasks.
- Often **resource-constrained** (sometimes < 1Kb of RAM and CPU < 1MHz).
- Need to be **reliable** and operate in **real-time**.
- Some might have **no OS** or a **real-time OS**.

Examples

IoT devices (smart 📺 ⌚), game controllers 🎮, medical devices 🚑 etc.

⚙️ Typical hardware components

Microcontroller (CPU, RAM, ROM, I/O)

The **brain** of the system. It executes the program and interacts with the peripherals.

Peripherals (I/O devices)

Input and output devices that interact with the environment. Sensors, displays, motors, etc.

Power supply

Provides power to the system. Can be a battery, USB, etc.

Communication interfaces

Ways to communicate with the system. Serial, I2C, SPI, etc.

👩‍💻 Hardware and software design and integration

Ideal Workflow:

1. *Hardware:* SysDesign/Schematics -> Printed Circuit Board (PCB) -> Assembly -> Board bring-up
2. *Software:* Read datasheets -> Write drivers -> Write application code
3. *Integration:* Test and debug -> Optimize -> Repeat
4. *Deployment:* Production -> Maintenance

Both software and hardware/electrical engineers need to work together to design and integrate the system.

White, Elecia. Making Embedded Systems. 2nd ed., O'Reilly Media.

🌡️ Weather station project hardware design

Weather station that displays **temperature and humidity** on an **LCD screen** (custom drivers) on a **Raspberry Pi Pico W** microcontroller.

Components:

- **Raspberry Pi Pico W microcontroller**: Microcontroller with RP2040 chip and WiFi capabilities
- **DHT22 sensor**: Tempeature and humidity sensor with proprietary protocol (DHT22)
- **LCD1602 display**: Small 2.5" LCD display with I2C communication interface
- Breadboard, jumper wires, resistors, etc.

Adafruit. "DHT22 Temperature-Humidity Sensor." Adafruit Learning System, 2021.

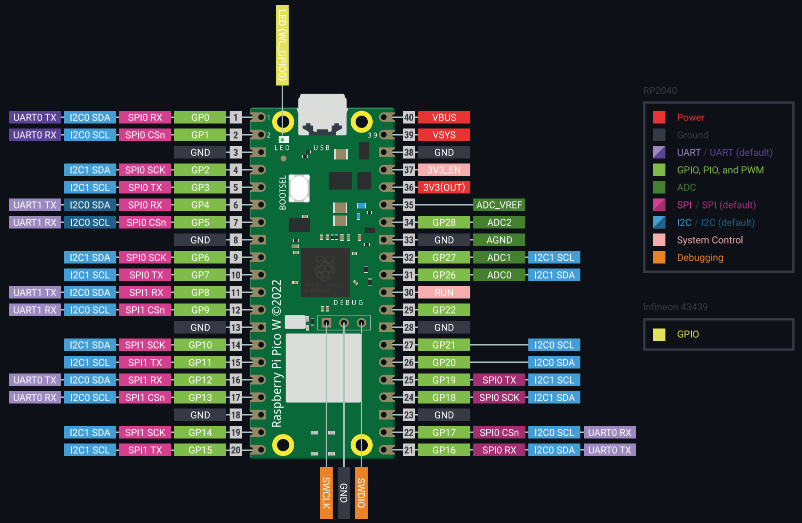
LCD1602 Display. "LCD1602 Display." RoHS, 2021.

Raspberry Pi Foundation. "Raspberry Pi Pico." Raspberry Pi, 2021.

🤖 **Pico W Microcontroller** - Datasheet (DS) overview

Datasheet overview and Pinout

- RP2040 microcontroller (2MB flash MEM)
- Dual-core ARM Cortex-M0+ processor (133MHz)
- 26 GPIO pins (23 digital + 3 ADC)
- Micro USB-B for power and data
- SRAM: 264KB
- 2.4GHz WiFi and Bluetooth 5.0
- Comms: SPI, I2C, UART, etc.



Raspberry Pi Foundation. "Raspberry Pi Pico-W Datasheet." Raspberry Pi, 2024.

🤖 **Pico W Microcontroller** - DS Applications information (1/2)

Programming the flash

Reprogram the flash memory with a new program using the USB bootloader.

GPIO pins and ADC

- Each pin can be configured as a digital input/output, analog input, or a special function (UART, SPI, I2C, etc.)
- Pins are 3.3V tolerant with 3 of them ADC capable (convert analog to digital)

USB and power supply

- Micro USB-B port for power and data (range of 1.8V to 5.5V; can be powered by battery)
- USB bootloader for programming the flash memory

Raspberry Pi Foundation. "Raspberry Pi Pico-W Datasheet." Raspberry Pi, 2024.

🤖 **Pico W Microcontroller** - DS Applications information (2/2)

Wireless interface

- 2.4GHz WiFi and Bluetooth 5.0 for wireless communication

Debugging

- Using the SWD (Serial Wire Debug) interface
- printf to UART0 (default) or USB CDC ACM

LIMITATIONS:

- Cannot use CLK and VSYS monitor at the same time
- Cannot check for IRQs when SPI transaction is in progress

Raspberry Pi Foundation. "Raspberry Pi Pico-W Datasheet." Raspberry Pi, 2024.

📝 **Setting up the development environment** - Overview

Steps

1. **Download** and **install** the Pico C/C++ SDK from including submodules
2. Install the **toolchain**: `CMake` and GCC cross compiler `gcc-arm-embedded` (using `nix-shell`)
3. Make a c_cpp_properties.json file for the **VSCode** IDE to recognize SDKs and includes
4. Create a **hello_usb** project by use the SDKs provided libs (configure c/cpp compiler and include in IDE)
5. Compile with the `CMakeLists.txt` and `pico_sdk_import.cmake` scripts. Automated with custom `picow-build.sh` script.
6. **Flash** the program to the Pico W microcontroller using the USB bootloader.

Raspberry Pi Foundation. pico-sdk. GitHub Repository, 2024.

👨‍💻 **Nix shell and bash automation** (Demo!)

Nix shell

- A package manager that allows for reproducible builds.
- Use a `shell.nix` file to specify the dependencies for the project.
- Run a temporary `nix-shell` to enter the environment with the dependencies.

No need for global installations, everything is contained in the `nix-shell`.

Bash automation

Automate the docs builds and project builds with bash scripts. Can be used inside the `nix-shell` environment.

Marp

A markdown presentation tool that allows for easy slide creation.

Breadboard + Pico W + USB

stdio over USB

A USB Communication Device Class (CDC) ACM virtual serial port, using TinyUSB's CDC support. We can easily:

- Pass compiled binary (as uf2) to the Pico W which will convert it to a binary file and flash it to the microcontroller.
- Use the USB serial port to send and receive data from the Pico W (standard calls avail. like `printf`)

Raspberry Pi Foundation. "Getting Started with Raspberry Pi Pico." Raspberry Pi, 2024.

Blink onboard LED via Wireless chip's GPIO

DEMO

⌨️ **Input and output via USB and UART** (Hello World!)

UART:

- **Universal Asynchronous Receiver-Transmitter** (UART) is a serial communication protocol.
- **TX** (transmit) and **RX** (receive) pins are used to send and receive data.
- **Baud rate** is the speed of communication (bits per second).

USB:

- **Universal Serial Bus** (USB) is a common interface for connecting devices.
- **CDC ACM** (Communication Device Class Abstract Control Model) is a USB class for serial communication.

First use USB to send (binary file) and receive data (`printf`) from the Pico W microcontroller and use `minicom` to listen to data on the serial port.

⌨️ IO with Pico W's default GPIO over UART

Serial input (*stdin*) and output (*stdout*) can be directed to either serial UART or to USB CDC (USB serial). By default, printf target UART0 on Pico.

UART0	Physical Pin	GPIO Pin
GND	3	N/A
UART0_TX (sending from Pico)	1	GPIO0
UART0_RX (receiving at Pico)	2	GPIO1

See the macros in `/pico-sdk/src/boards/include/boards/pico.h` for the default UART pins.

⌨️ **Hello World!** USB output

👨‍💻

DEMO

```
# Compile the hello_usb project
cd hello_usb
./picow-build.sh

# Check for the picow device with dmesg (kernel ring buffer)
sudo dmesg -w

# In a new terminal, flash the compiled binary to the Pico W
cp build/hello_usb.uf2 /run/media/deck/RPI-RP2/

# Open a serial terminal to see the output (using the dmesg output to find the device)
RATE=115200    # The baud rate of the serial port (bits per second)
sudo minicom -D /dev/ttyACM0 -b $RATE

# USE CTL+A then X to exit minicom
```

🚧 **Project** - To Be Continued

🌡️ **DHT** - Overview

📢 Drivers and communication protocols

Drivers

- Software that allows the microcontroller to interact with peripherals.
- They abstract the hardware and provide a simple interface for the application code.

Communication protocols

- A set of rules that define how devices communicate with each other.
- Examples: I2C, SPI, UART, etc.

⏲️ I/O and interrupts

Input/Output (I/O)

- **Input**: Reading data from the environment (sensors, switches, etc.)
- **Output**: Sending data to the environment (displays, motors, etc.)

Interrupts

- A way for the microcontroller to respond to events in real-time.
- The microcontroller can stop what it's doing and handle the interrupt.

Model view controller in embedded systems

Often, embedded systems are designed using the **Model-View-Controller** (MVC) pattern. This pattern separates the system into three main components:

- **Model**: The data and logic of the system.
- **View**: The user interface.
- **Controller**: The logic that connects the model and the view. It processes user input and updates the model and view accordingly.

White, Elecia. Making Embedded Systems. 2nd ed., O'Reilly Media.

📋 System diagram

TODO ADD STATIC FILE