Chapter 13, part 2: Clarifications and Additional Info

12/23/21

Chapter 13: Creating Loose Coupling with Queues (page 343)

1 Section: Creating a command queue (pg 346)

- **Additional info, page 346**
  - Chapter 13's example programs include a "PWM implementation". It's in `pwmImplementation.c`. It provides functions for setting the LEDs' brightness levels, i.e., 0 to 100%.
  - How the PWM implementation works is not described in the book. This study-guide provides a tutorial description, in Chapter 13, part 1.

2 Section: Deciding on queue contents (pg 346)

- **Clarification, page 346**
  - This entry provides additional info about the "interface definition" `iPWM`. This is info that was helpful to me in understanding the abstraction being used, and the code. Some of the perceived problems could be due to inexperience on my part.
  - The `iPWM` "interface definition" is used for calling functions that set a particular LED's brightness-level.
    - For example, the function `SetBlueDuty(100.0)` sets the blue LED's brightness to 100%.
    - The `iPWM` "interface definition" is an abstraction, and it is intended for calling different types of functions that set LED brightness-levels.
  - I found that the interface-definition was difficult to make sense of, as an abstraction.
    - Abstractions are generalizations of specific concrete instances. Typically, understanding an abstraction requires knowing concrete instances, and knowing the abstraction's purpose or use.
    - When reading the chapter and code, it was not clear to me what different types of LED functions would be supported by this interface-definition. The book has some discussion of that, but not until the end of the chapter. Just one additional LED type is described there, but it wasn't clear to me how the `iPWM` interface definition would help with it.
    - For instance, the `iPWM` "interface definition" is implemented as a struct typedef, named `iPWM`. The struct has one element, which is a function pointer. The book says other elements could be added to the struct, to extend the interface, but I could not envision what those elements would be. (*"Wrapping the function pointer in a struct such as iPWM provides the flexibility of adding additional functions while keeping refactoring to a minimum."*)
    - In retrospect, I can see how using function pointers could be a useful interface. For example, one function-pointer could be used for functions that turn on a green LED. The pointer could be set to a particular function that turns on a particular green LED. This would give more flexibility than directly calling a particular function by name.
  - In reading the code, I didn't know of other ways the interface definition might be used, in addition to calling the three functions for the existing LEDs, e.g., `SetBlueDuty()`. This made it harder to make sense of the interface's code.
    - If the board's 3 LEDs are all that needs to be supported, the extra code for the interface-definition is superfluous and nonsensical. For instance, `SetBlueDuty()` could just be called directly.

- **Additional info, page 346**
  - This entry describes how the `iPWM` struct is declared and initialized.
  - Regarding the `iPWM` struct, the book states, "*This struct only (currently) consists of a single function pointer: iPwmDutyCycleFunc. iPwmDutyCycleFunc is defined as a constant pointer—after the initialization of the iPWM struct, the pointer can never be changed. This helps guarantee the pointer won't be overwritten...*"
  - The typedefs for the `iPWM` struct are defined in `iPWM.h`:

```
typedef void (*iPwmDutyCycleFunc)( float DutyCycle );
typedef struct
 {
     const iPwmDutyCycleFunc SetDutyCycle;
 }iPWM;
```

  - The iPWM structs are declared as global variables, and initialized, in `pwmImplementation.c`
    - Storage for the struct is in the executable's header (ELF?). The struct is initialized in the header by the compiler and linker.

```
iPWM BluePWM = {.SetDutyCycle = SetBlueDuty};
```