Smart Products Lab 3

Tyler Morrison

February 10, 2019

Section 1.

a) Algorithms for manipulating, reading and writing to registers.

- `int pinMode(int pin_number, PinModes p_mode) override:`
  1. use `getPtr` to get the memory register needed to write the pin mode.
  2. use `readRegBits` to get the current register state.
  3. cast the pin mode integer to the associated (binary) register state.
  4. bit shift the desired register state over to the correct registers based on the pin number.
  5. bit shift a mask of zeros to the target registers.
  6. get the new state of the register by ANDing the mask and the old pin state and ORing that with the desired register changes.
  7. use `setRegBits` to write the new desired state to the target register.
- `int digitalRead(int pin_number) override:`
  1. use `getPtr` to get the memory register needed to write the pin mode.
  2. use `readRegBits` to get the current register state.
  3. bit shift the single-bit mask over to the target bit in the register.
  4. AND the mask with the current regsiter state and shift the pin to the front of the register.
  5. convert the register binary to an `int` and return it.
- `int digitalWrite(int pin_number, DigitalOut out_value) override:`
  1. use `getPtr` to get the memory register needed to write the pin mode.
  2. use `readRegBits` to get the current register state.
  3. bit shift a 1 over to the target bit in the register.
  4. bit shift a single bit mask to the target bit in the register.
  5. AND the mask with the current register state and OR that with the desired desired output bit.
  6. use `setRegBits` to write the new desired state to the target register.

b) Efficiency considerations

In order to improve efficiency, I used static arrays to get the registers and shifts for each pin for read, write high, write low, and pin mode. This might use a little bit more memory but it precludes a bunch of logic and modular arithmetic that might otherwise be used to determine them at runtime each time you need to operate on a pin.

c) Extra functions

`uint32_t readRegBits(void*):`	Read the bits in a register from the pointer.
`int setRegBits(void*, uint32_t):`	Write the bits in a register given by the pointer.
`int readPinMode(int):`	Read the pinmode of a pin number
`void updatePinSettings(int, PinSettings &):`	Update only the pin settings of a pin that can change, i.e: the state and mode.
`void showPins():`	Print all the pins to the screen.
`void showPins(const std::vector<int> &):`	Print the selected pins to the screen.
`void main:`	I added the ability to accept an optional pin number argument to my main function. This helped me test a bunch of pins on my Pi with LEDs without recompiling.

d) Bitwise operators

Bitshifts were used to shift desired outputs and masks over to the correct bits in the register. Bit-wise AND's and OR's were used with bitmasks to change single bits in the register without affecting bits other than the targets.

e) Error checking

The main error checking I did was check for and throw errors for pin numbers that were outside of the allowed range. When writing to a pin, I also first check if it is set to write mode. For both of these common errors, the behavior is to return from the function with a flag: `-1`.

Section 2.

a) What information would have helped me better understand the lab?

I'd like to know a little bit more about memory mapping and the exact mechanics of how that works in this lab, but that's all I can think of.

b) Output

Program 1: Output of labthree.cpp

```
1 default pin set to 27
2 Toggling Pin Mode ...
3 toggle complete
4 Writing Low :
5 Value of pin 27 is 0
6 Writing High :
7 Value of pin 27 is 1
```