Digichess C code style guide

Big Chungus

February 8, 2023

1 Indentation

- Use tabs or else.
- Indent cases in switch statements

2 Naming

- Constants in UPPER_SNAKE_CASE
- Local variables in lower_snake_case
- Function names in lower_snake_case
- Typedef struct and enum names in PascalCase
- Enum values in UPPER_SNAKE_CASE
- Struct members in lower_snake_case
- Don't use globals
- For variables that depend on a physical unit, use a unit suffix, e.g. time_ms

3 Headers

- Header file names are of the format lower_snake_case.h
- Include guard macros of the format __UPPER_SNAKE_CASE_H
- For domain-specific code, e.g. a driver, use a prefix for functions and types, e.g.

```
// Bad
Point a;
draw_pixel(a);

// Good
LCD_Point a;
lcd_draw_pixel(a);
```

4 Comments

- Space at the start of a comment, e.g. `// Comment here`
- Every line in a multiline comment starts with a `//`
- Doxygen comments use a triple slash, e.g. `/// @doxygenhere`

5 Macros

- Always use parentheses around values, e.g. `#define CONST_NAME (VALUE)`
- If it's a macro used in a limited scope, e.g. just for specific driver header files, use a double underscore, e.g. `#define __PRIVATE_MACRO_NAME`

6 Variables

- Multiple same-type variable declarations in a single line are fine, but don't do single-line multi-variable assignments, e.g.

```
int a, b, c; // Fine
int a = 1, b = 2, c = 3; // No
```

- Pointer asterisk next to type, e.g. `int* ptr`
- Don't do multi-variable single-line pointer assignments.

```
int *a, *b; // Bad

// Good
int* a;
int* b;
```

7 Flow control

- Spaces around keywords and same-line braces, e.g. `if (condition) {`
- Empty lines around blocks, e.g.

```
// Block 1
if (cond1) {
    foo();
}

// Block 2
while (cond2) {
    bar();
```

```
}
```

- Else and else if statements start on the same line as the brace, e.g.

```
if (cond1) {
    a();
} else if (cond2) {
    b();
} else {
    c();
}
```

- Single-line **if** statements without braces are fine if they're kept short, e.g.

```
// Bad
int foo(int a) {
    if (a == 5) return long_and_complex_function_or_operation(a);

    return 0;
}

// Better
int foo(int a) {
    if (a == 5) {
        return long_and_complex_function_or_operation(a);
    }

    return 0;
}

// Good
int bar(int a) {
    if (a == 5) return 1;

    return 0;
}
```

- In all other cases avoid single-line **if** statement and split them into a new line with braces.

8 Functions

- Use return guards when possible, e.g.

```
void some_func(int a, int b) {
    if (a == 0) return;
```

```
    if (b == 5) return;

    return a + b;
}
```

- For the love of god don't use **goto**
- Wrap short inline assembly in their own functions. Longer assembly should go into .S files.

9 Operators

- Spaces around operators, e.g. 5 + 3
- Always use parentheses around bitwise operations, e.g. 2 + (5 << 3)
- If splitting a long chain of operators into multiple lines, put the operator the line is split around in the new line, e.g.

```
if (
    arg1
    || arg2
    || arg3
) {
    int value =
        123
        + 456
        + 789;

    stuff();
}
```

10 Misc

- Try to keep the line width under 80 characters
- For stuff that's not listed, use your best judgement