NTURACING

MCU Programming

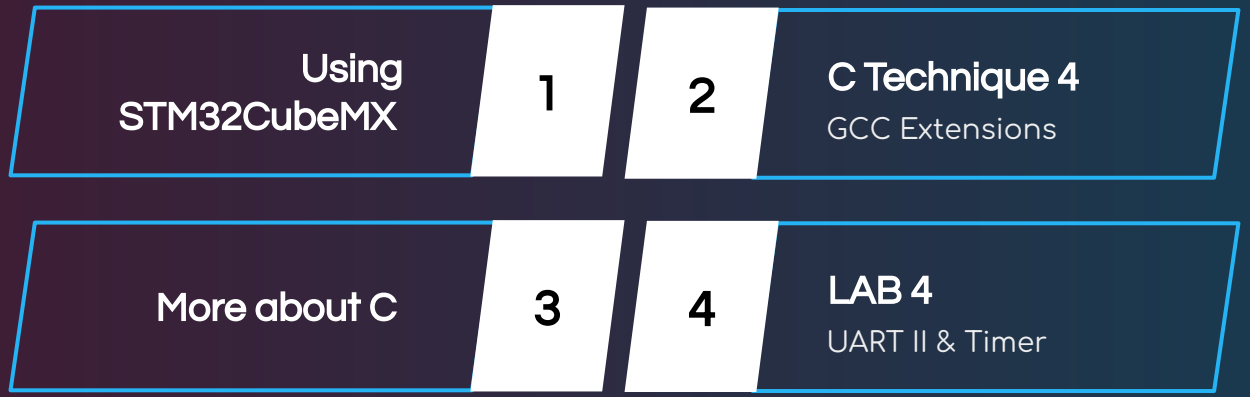
W4: Miscellaneous

by 羅紀翔

2024-25 FSAE Season

Agenda

NTURACING



NTURACING

Using STM32CubeMX

References: CH4, *The Definitive Guide*

NTURACING

STM32CubeMX Configuration Tool

- Clock
- Peripherals
- Pins
- Middleware
- Generated code layout

NTURACING

C Technique 4:

GCC Externsions

References: GCC reference:
https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

NTURACING

GCC Extensions

- Extensions to standard C language provided by GCC
- Why extension? Because standard C is lame.
  - Some GCC extension has become the standard.
  - Other compilers also adopted some GCC extensions.
- GCC extension: https://gcc.gnu.org/onlinedocs/gcc/C-Extensions.html

GCC Extensions

NTURACING

Variable-Length Array

- Standardized in C99

```c
int print_array(int len, int a[len]) {
  for (int i = 0; i < len; i++) { printf("%d ", a[i]); }
  printf("\n");
  return 0;
}
int main() {
  int len; scanf("%d", &len); // 5
  int a[len];
  for (int i = 0; i < len; i++) { a[i] = i; }
  printf("len = %d, sizeof(a) = %lu\n", len, sizeof(a)); // len = 5, sizeof(a) = 20
  print_array(len, a); // 0 1 2 3 4
  return 0;
}
```

GCC Extensions

NTURACING

Designated Initializers

```c
// the following declarations are the same
int a[6] = {0, 1, 1, 1, 0, 0};
int b[6] = {[1] = 1, [2] = 1, [3] = 1};
int c[6] = {[1] = 1, 1, 1};
int d[6] = {[1 ... 3] = 1};  // GCC extension
```

```c
struct point { int x; int y; };

// the following declarations are the same
struct point p = {1, 2};
struct point q = {.y = 1, .x = 2};
```

GCC Extensions

NTURACING

Compound Literals

- Standardized in C99

```c
// the following declarations are the same
struct point p = {.x = 1, .y = 2};
struct point q = (struct point){1, 2};

// compound literals are lvalues
struct point *r = &(struct point){1, 2};

void print_point(struct point *p) { printf("(%d, %d)\n", p->x, p->y); }
int main() {
  print_point(&(struct point){1, 2});
  return 0;
}
```

GCC Extensions

NTURACING

Statements and Declarations in Expressions

```c
#define MAX1(A, B) ((A) > (B) ? (A) : (B))
#define MAX2(A, B)      \
  __extension__({       \
    int _A = (A);       \
    int _B = (B);       \
    _A > _B ? _A : _B; \
  })
int one() { printf("1 "); return 1; }
int two() { printf("2 "); return 2; }
int main() {
  printf("MAX1 = %d\n", MAX1(one(), two())); // 1 2 2 MAX1 = 2
  printf("MAX2 = %d\n", MAX2(one(), two())); // 1 2 MAX2 = 2
  return 0;
}
```

GCC Extensions

NTURACING

Statements and Declarations in Expressions (cont'd)

- The last line is the value of the construct
- `__extension__` is used to suppress the warning: `warning: ISO C forbids ...` when compiles with `-Wpedantic` flag

GCC Extensions

NTURACING

Reffering to a Type with `Typeof`

```
#define MAX2(A, B)      \
  __extension__({       \
    typeof(A) _A = (A); \
    typeof(B) _B = (B); \
    _A > _B ? _A : _B;  \
  })
```

NTURACING

More about C

References: 你所不知道的C語言
(https://hackmd.io/@sysprog/c-prog/%2F%40sysprog%2Fc-programming)

More about C

NTURACING

`__attribute__`

- Provided by gcc for some high level settings, syntax: `__attribute__((attribute-list))`, separated by semicolons
- `packed`: Prevent structure padding
- `aligned`: Align variables
- `section`: Assign variable or function to specific memory section
- `used`: Keep the symbol in memory even if not used
- `weak`: If two symbols with the same name exist, the weak one will be ignored
- Ref: 來了解GNU C __attribute__ (https://medium.com/@fearless1997s/%E4%BE%86%E4%BA%86%E8%A7%A3gnu-c-attribute-f06d49af2454)

More about C

NTURACING

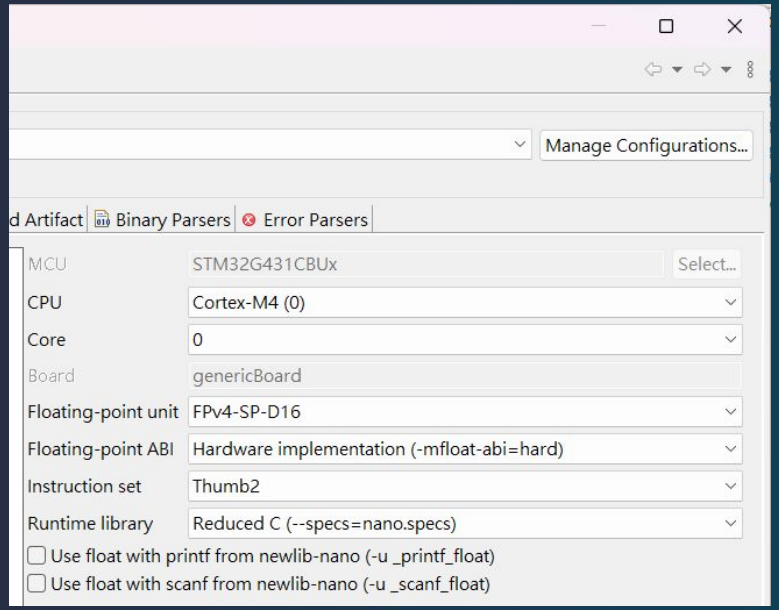
C Standard Library

- Aka **libc**, defined by the C standard
- The C standard library provides macros, type definitions and functions for tasks such as string manipulation, mathematical computation, input/output processing, memory management, and input/output.
- e.g. implements `printf`, `scanf`, `time`, etc.
- Often comes built-in in Operating Systems
  - Linux: GNU C library (**glibc**)
  - Windows: Microsoft Visual C++
- Designed for desktop computers, not suitable for microcontrollers
  - Too ROM and RAM heavy

More about C

C Standard Library (con'd)

- Alternative C libraries for microcontrollers
  - **Newlib** : Created by Cygnus and now maintained by Red Hat, designed for embedded devices
  - **Newlib-nano** : Created by ARM to further reduce the memory requirement with some tradeoff: no floating point support for `printf`, `scanf` (can be enabled by `-u _printf_float` flag) STM32CubeIDE uses newlib-nano by default
- Ref: Which Embedded GCC Standard Library? newlib, newlib-nano, … (https://mcuoneclipse.com/2023/01/28/which-embedded-gcc-standard-library-newlib-newlib-nano)



Project -> Properties -> C/C++ Build -> MCU/MPU Settings

More about C

NTURACING

C Standard Library (con'd)

- Newlib requires low level functions to implement system calls, e.g. `_getpid()`, `_exit()`, `_read()`, `_write`, etc.
- STM32Cube defines them in `syscalls.c` and `sysmem.c`
  - Functions in `syscalls.c` are just dummies and does nothing
  - `sysmem.c` implements `_sbark()` for memory allocation

More about C

NTURACING

C POSIX Library

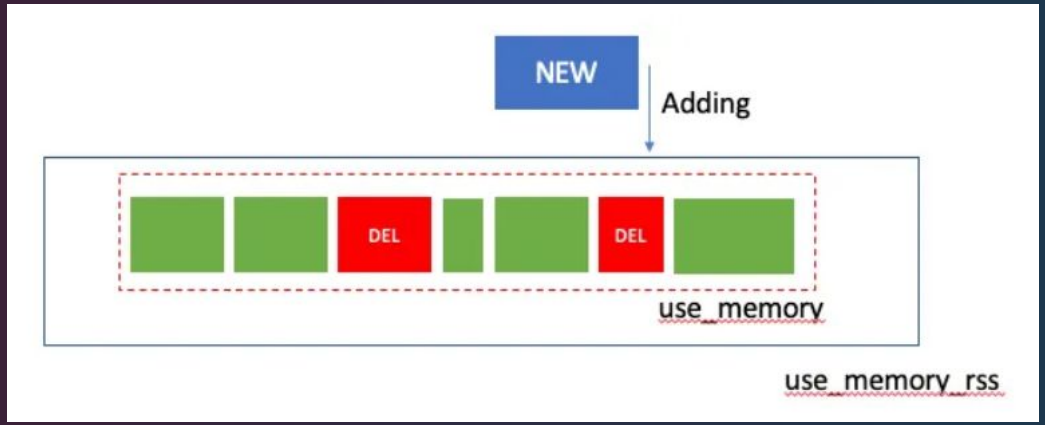
- Portable Operating System Interface (POSIX), standarized by IEEE
- A super set of C standart library
- Provides more system calls
  - Process, signal: `exec()`, `fork()`, `sleep()`, `signal()`, `raise()`, `alarm()`
  - File: `read()`, `write()`, `fopen()`, `fclose()`, `fcntl()`
  - Berkeley socket: `socket()`, `bind()`, `listen()`, `accept()`, `connect()`
- OSes that implement POSIX APIs are POSIX compatible
  - Windows is **NOT** POSIX compatible
- Application code can depend on POSIX APIs to be portable

More about C

NTURACING

Memory Allocation

- `malloc()`: Reserve a space starting at a pointer from a pool of memory
- `free()`: Mark the space not to be used anymore
- Newlib uses `_sbark()` to get more memory
- Memory fragmentation and defragmentation



More about C

NTURACING

Problems for Memory Allocation

- Searching for a contiguous area big enough and handling defrmentation my have unbounded execution time
  => Memory allocation is very frowned upon in embedded systems where deterministic behavior is required
- Real-time (bounded execution time) alternatives: TLSF, etc.
- Embedded OSes provides

NTURACING

LAB 4: UART II & Timer

References: Controllers Tech

UART II

NTURACING

UART in Interrupt-Driven and DMA Mode

- Enable UART interrupt
- Transmit using `HAL_UART_Transmit_IT()`
- Implement `HAL_UART_TxCpltCallback()`
- Setup DMA channel for UART
- Enable DMA interrupt (enabled automatically)
- Transmit using `HAL_UART_Transmit_DMA()`
- Ref: Controllers tech
    - Youtube: https://www.youtube.com/watch?v=JaMwNT0m3Sw)
    - Artical: https://controllerstech.com/stm32-uart-2-use-interrupt-dma-to-transmit-data/

UART II

NTURACING

Use Timer to Periodically Print Message

- Determine the clock frequency
- Configure the prescaler
- Enable timer interrupt
- Implement `HAL_TIM_PeriodElapsedCallback()`
- Ref: https://medium.com/%E9%96%B1%E7%9B%8A%E5%A6%82%E7%BE%8E/stm32-07-timer-interrupt-937c104cc441