

Introduction to the STM32 L432KC MCU & Refresher on C Programming

Lecture 06

Josh Brake
Harvey Mudd College

Outline

- Introduction to the STM32-L432KC
- Review of basic architecture
- C refresher
 - Common idioms to set/clear bits
 - Pointers and arrays
 - Structures

Learning Objectives

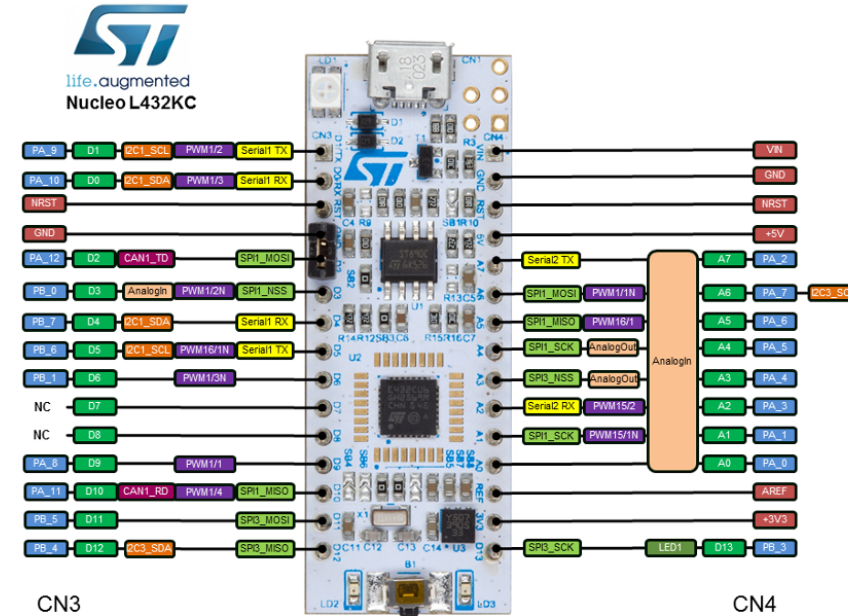
By the end of this lecture you should be able to...

- List the basic details of the architecture of the ARM Cortex-M4 CPU used in our STM32 MCU.
- Recall basic C programming idioms and concepts (e.g., pointers, arrays, structures).

STM32 Nucleo-32 board

Components

- MCU – STM32L432KC
- External flash memory
- 24 MHz crystal oscillator
- On-board ST-LINK debugger/programmer. Virtual COM port and debug port.
- 1 user LED and 1 reset push button
- Arduino Nano V3 form factor



What is an MCU

- MCU = MicroController Unit
- Processor core + peripherals

Figure 1. System architecture

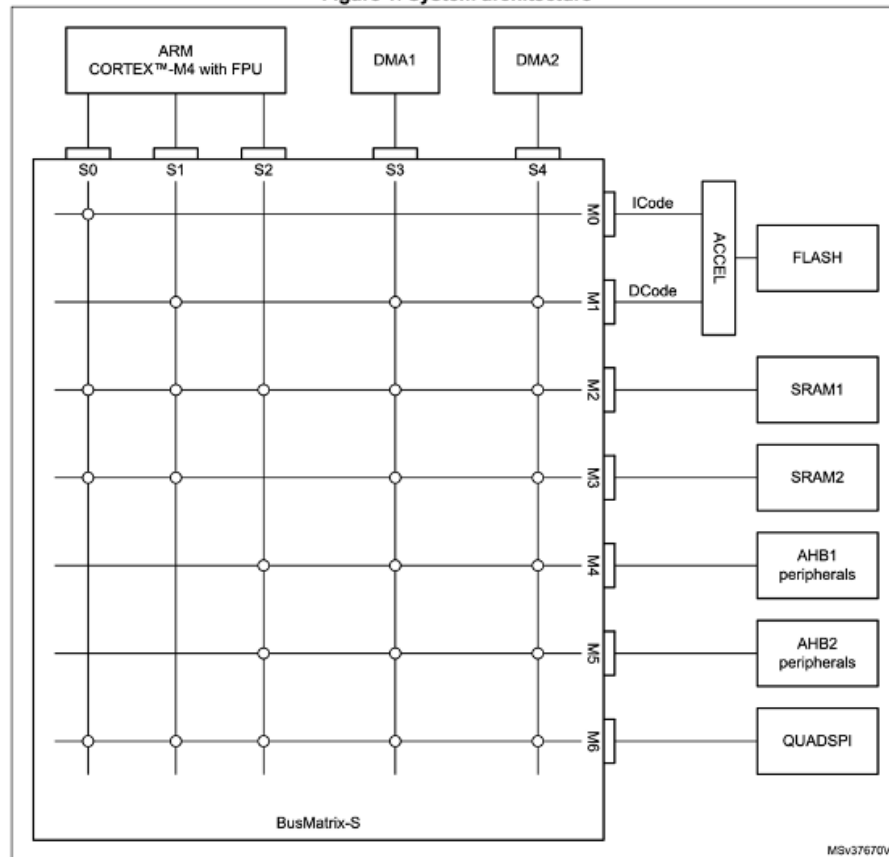
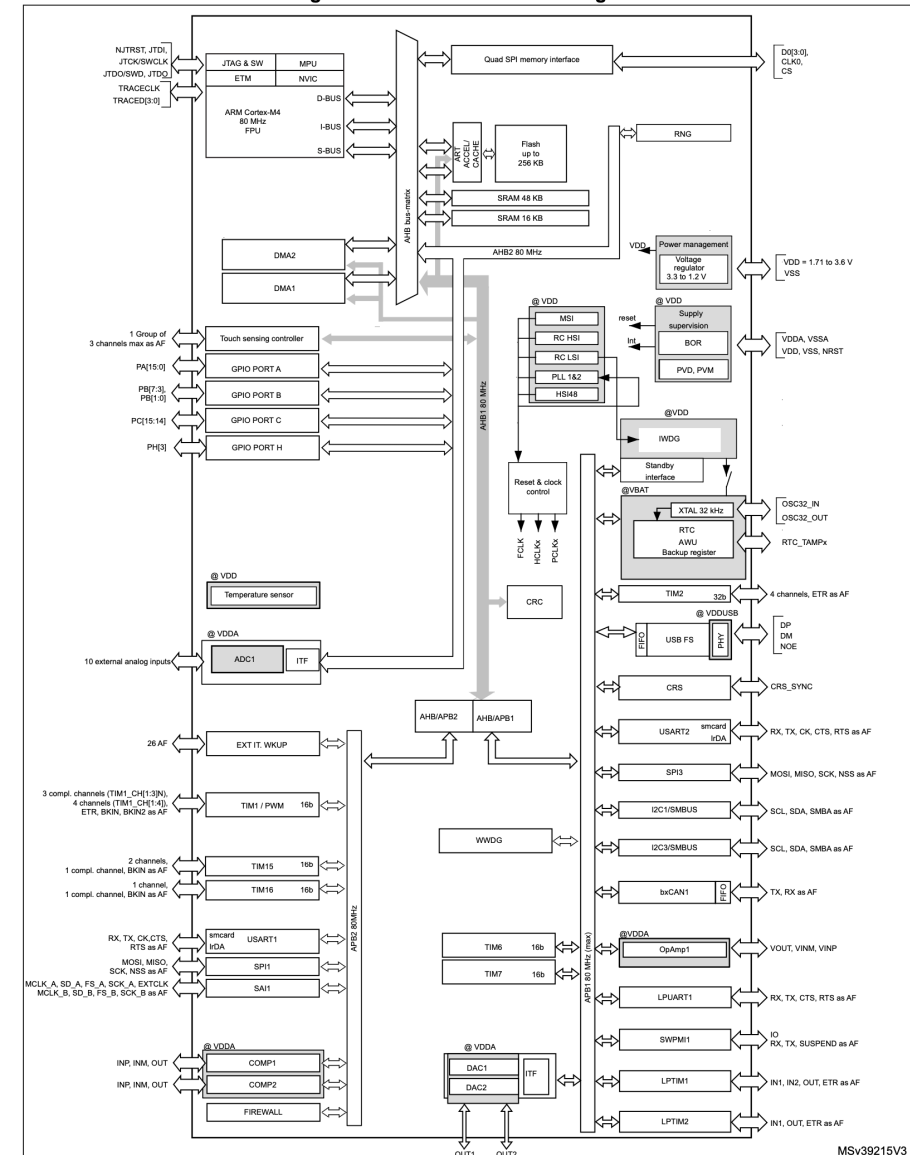


Figure 1. STM32L432xx block diagram



Note: AF: alternate function on I/O pins.

Documentation

_____ : Information about all peripherals and their control registers

_____ : System block diagram, Pin functions, electrical characteristics, timing specs

_____ : Information about the architecture (e.g., Cortex-M4), supported assembly instructions, registers, memory map, etc.

Questions for a new MCU

- What is the _____ set?
- What does the _____ look like?
- What _____ are used?
- What types of _____ exist?
- What _____ are available?

STM32 L432KC Architecture

- STM32 MCU has an ARM Cortex-M4.
- It runs the ARMv7E-M architecture. This is a 32-bit architecture.
- Also supports Thumb-2 execution. Thumb-2 is a set of compressed, 16-bit instructions.
- One special thing to watch with Thumb is conditional execution. With Thumb execution you can only use conditional execution within an “if-then” block which can hold up to four successive instructions.

Architecture Overview

- Instructions are 32-bit
- Sixteen 32-bit registers R0-R15
- Supports condition codes
- Most instructions operate on two registers and put result in a third.

Microarchitecture Flashback

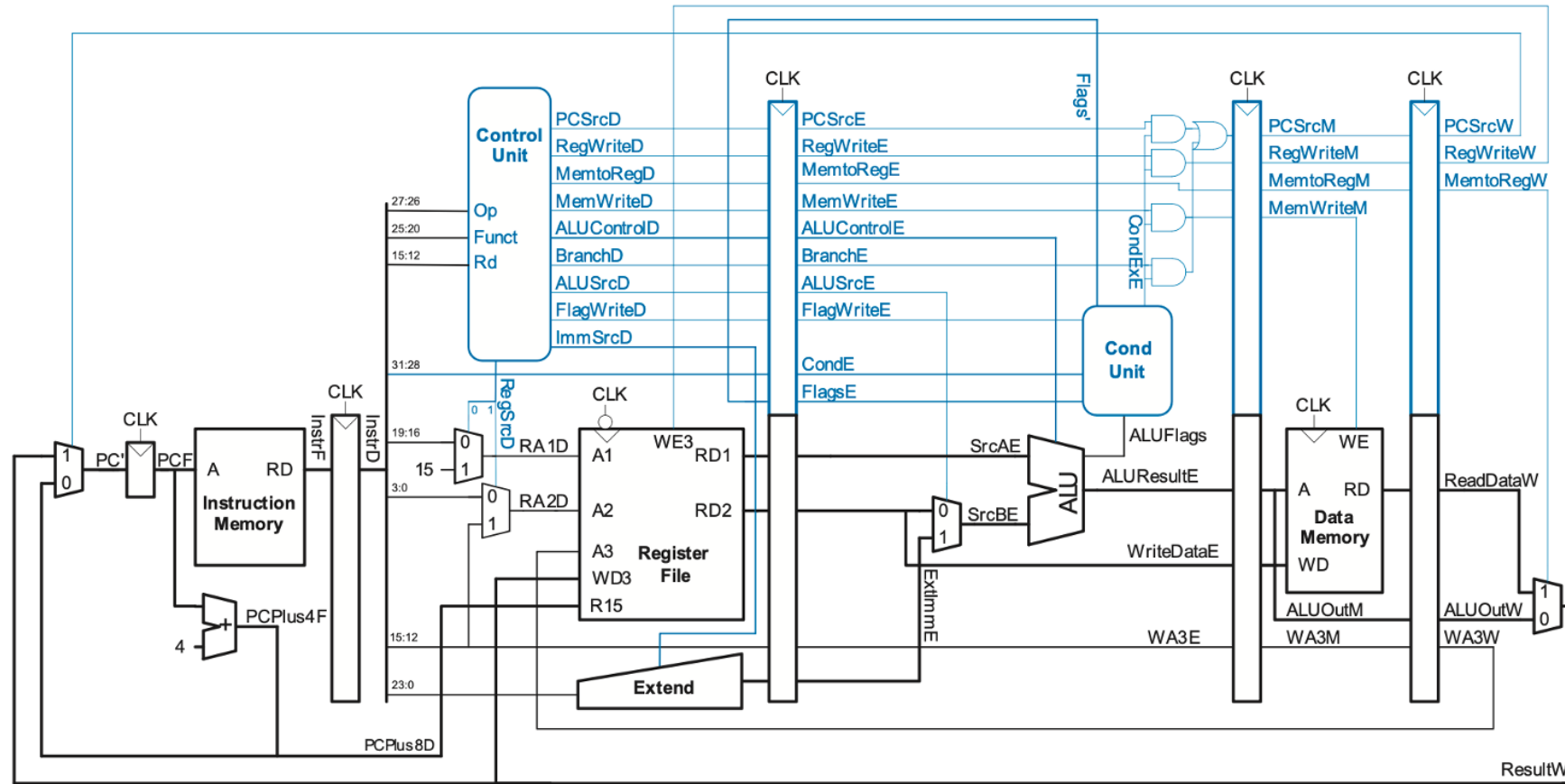


Figure 7.47 Pipelined processor with control

Harris and Harris, *Digital Design and Computer Architecture ARM Ed.*, p. 430

Register Set

- R15 is _____
- R14 is _____. Holds return addresses
- R13 by convention used as the _____.
- Four condition codes in current program status register (CPSR)
 - N – _____
 - Z – _____
 - C – _____
 - V – _____

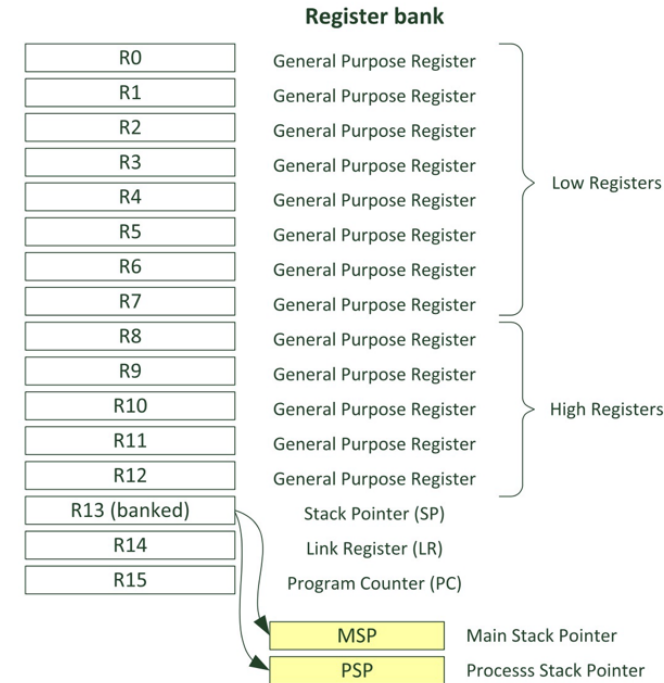


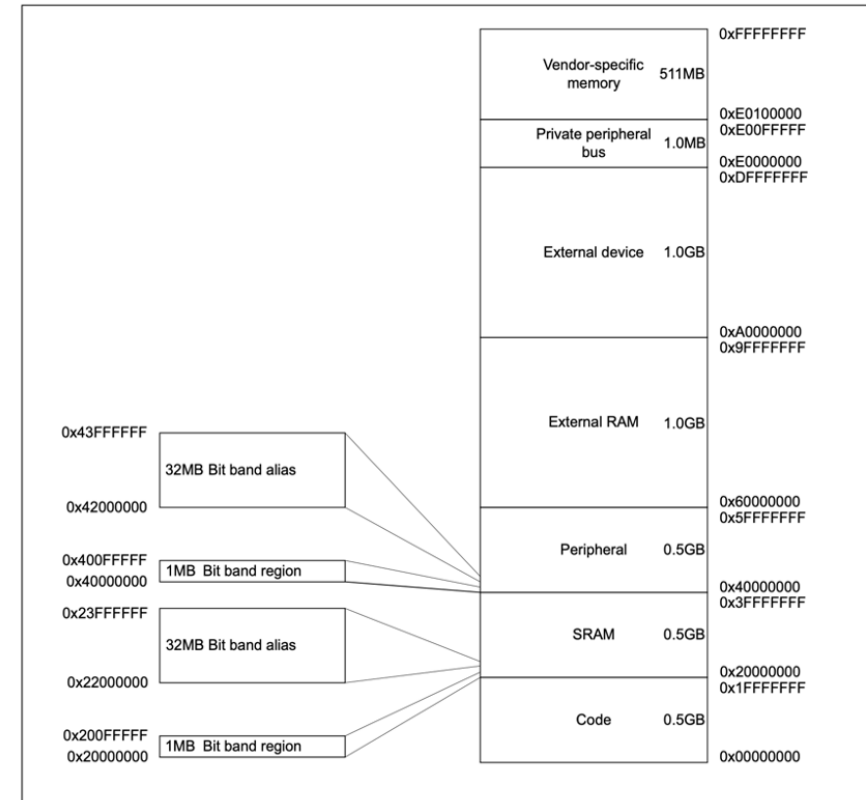
FIGURE 4.3

Registers in the register bank

Memory Map

- Flat 32-bit instruction set
- Addressed in bytes.
- _____ bytes of memory accessible (4 Gigabytes)
- Instructions are always aligned on word _____ in standard ARM and halfword _____ boundaries in Thumb mode.

Figure 8. Memory map

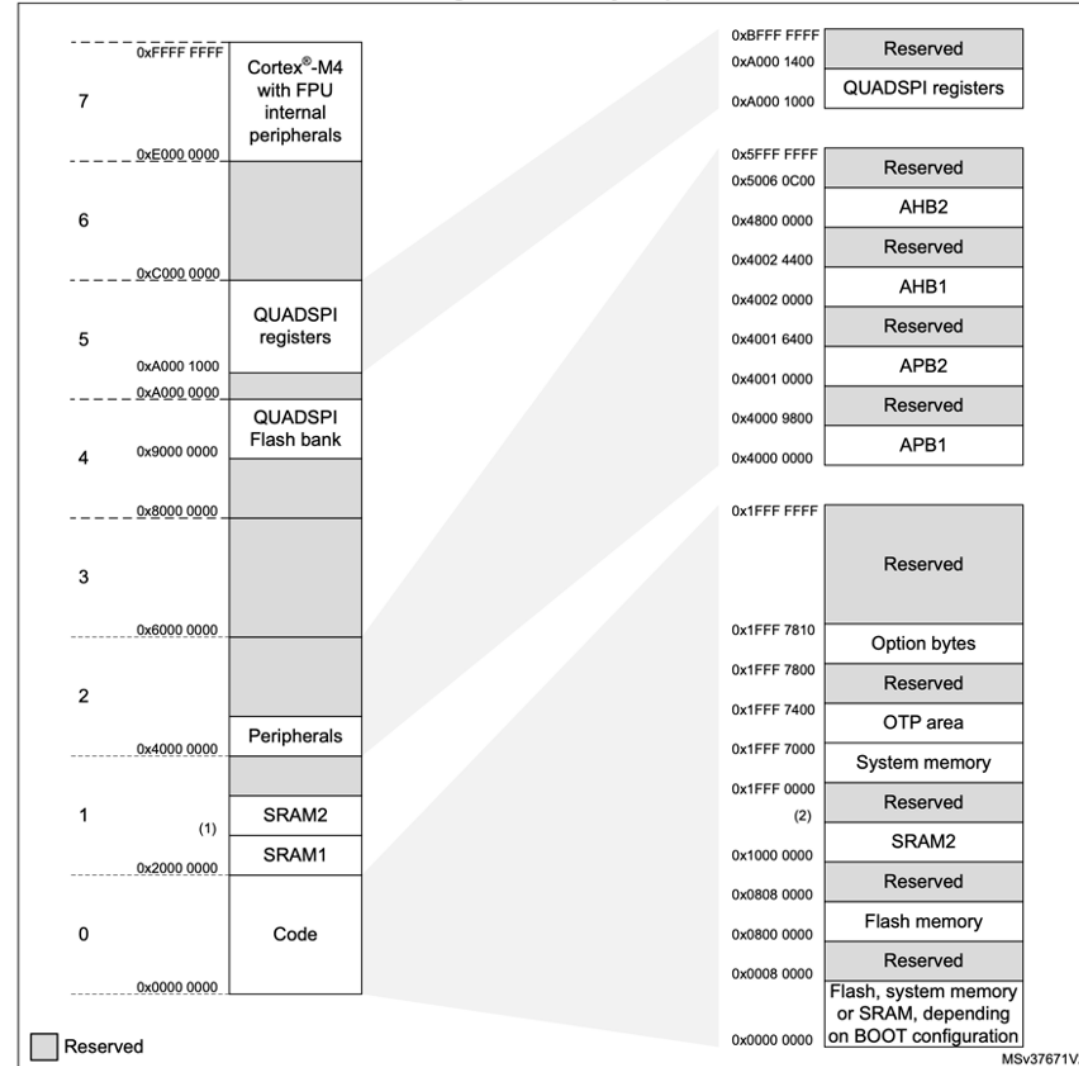


PM0214 Cortex-M4 Programmers Manual

Memory Map - STM32L432KC

RM0394 p. 67

Figure 2. Memory map



C Programming Review

Important Concepts in C

- C is libertarian by nature. You can stomp on any memory address you want!
- There is no memory management built in. You must manually allocate (and deallocate!) any memory you need.

Primitive Data Types in C

Table eC.2 Primitive data types and sizes

Type	Size (bits)	Minimum	Maximum
char	8	$-2^{-7} = -128$	$2^7 - 1 = 127$
unsigned char	8	0	$2^8 - 1 = 255$
short	16	$-2^{15} = -32,768$	$2^{15} - 1 = 32,767$
unsigned short	16	0	$2^{16} - 1 = 65,535$
long	32	$-2^{31} = -2,147,483,648$	$2^{31} - 1 = 2,147,483,647$
unsigned long	32	0	$2^{32} - 1 = 4,294,967,295$
long long	64	-2^{63}	$2^{63} - 1$
unsigned long long	64	0	$2^{64} - 1$
int	machine-dependent		
unsigned int	machine-dependent		
float	32	$\pm 2^{-126}$	$\pm 2^{127}$
double	64	$\pm 2^{-1023}$	$\pm 2^{1022}$

Primitive Data Types in `stdint.h`

Table eC.2 Primitive data types and sizes

Type	Size (bits)	Minimum	Maximum
char	8	$-2^{-7} = -128$	$2^7 - 1 = 127$
unsigned char	8	0	$2^8 - 1 = 255$
int	machine-dependent		
unsigned int	machine-dependent		
int16_t	16	$-2^{15} = -32,768$	$2^{15} - 1 = 32,767$
uint16_t	16	0	$2^{16} - 1 = 65,535$
int32_t	32	$-2^{31} = -2,147,483,648$	$2^{31} - 1 = 2,147,483,647$
uint32_t	32	0	$2^{32} - 1 = 4,294,967,295$
int64_t	64	-2^{63}	$2^{63} - 1$
uint64_t	64	0	$2^{64} - 1$
float	32	$\pm 2^{-126}$	$\pm 2^{127}$
double	64	$\pm 2^{-1023}$	$\pm 2^{1022}$

Operators and Operator Precedence

Table eC.3 Operators listed by decreasing precedence

Category	Operator	Description	Example
Unary	++	post-increment	a++; // a = a+1
	--	post-decrement	x--; // x = x-1
	&	memory address of a variable	x = &y; // x = the memory // address of y
	~	bitwise NOT	z = ~a;
	!	Boolean NOT	!x
	-	negation	y = -a;
	++	pre-increment	++a; // a = a+1
	--	pre-decrement	--x; // x = x-1
	(type)	casts a variable to (type)	x = (int)c; // cast c to an // int and assign it to x
	sizeof()	size of a variable or type in bytes	long int y; x = sizeof(y); // x = 4
Multiplicative	*	multiplication	y = x * 12;
	/	division	z = 9 / 3; // z = 3
	%	modulo	z = 5 % 2; // z = 1
Additive	+	addition	y = a + 2;
	-	subtraction	y = a - 2;
Bitwise Shift	<<	bitshift left	z = 5 << 2; // z = 0b00010100
	>>	bitshift right	x = 9 >> 3; // x = 0b00000001
Relational	==	equals	y == 2
	!=	not equals	x != 7
	<	less than	y < 12
	>	greater than	val > max
	<=	less than or equal	z <= 2
	>=	greater than or equal	y >= 10

Table eC.3 Operators listed by decreasing precedence—Cont'd

Category	Operator	Description	Example
Bitwise	&	bitwise AND	y = a & 15;
	^	bitwise XOR	y = 2 ^ 3;
		bitwise OR	y = a b;
Logical	&&	Boolean AND	x && y
		Boolean OR	x y
Ternary	? :	ternary operator	y = x ? a : b; // if x is TRUE, // y=a, else y=b
Assignment	=	assignment	x = 22;
	+=	addition and assignment	y += 3; // y = y + 3
	-=	subtraction and assignment	z -= 10; // z = z - 10
	*=	multiplication and assignment	x *= 4; // x = x * 4
	/=	division and assignment	y /= 10; // y = y / 10
	%=	modulo and assignment	x %= 4; // x = x % 4
	>>=	bitwise right-shift and assignment	x >>= 5; // x = x >> 5
	<<=	bitwise left-shift and assignment	x <<= 2; // x = x << 2
	&=	bitwise AND and assignment	y &= 15; // y = y & 15
	=	bitwise OR and assignment	x = y; // x = x y
	^=	bitwise XOR and assignment	x ^= y; // x = x ^ y

Operator Precedence Tip!

You should only have to remember multiplication/division before addition/subtraction.
For everything else, use parentheses!

Important Keywords in C

- volatile – prevents the compiler from using a cached value (forces load)
- const – “read-only”. Prevents you from assigning a value to the variable.
- static
 - Inside a function: retains its values between calls.
 - Applied to a function: visible only in this file
- extern
 - Applied to a function definition: has global scope (redundant)
 - Applied to a variable: defined elsewhere
- void
 - As return type of function: doesn't return a value
 - In a pointer declaration, the type of a generic pointer
 - In a parameter list: takes no parameters

Important Libraries

- _____ – standard fixed-width types (e.g., uint32_t)
- _____ - standard input and output. Contains functions like printf or fprintf.
- _____ – standard library: random number generation (rand and srand), allocating or freeing memory (malloc and free).
- _____ – math library: standard math functions like sin, cos, sqrt, log, exp, floor, ceil.
- _____ – string library: functions to compare, copy, concatenate, and determine the length of a string.

Setting and Clearing Bits

C Idioms for Setting and Clearing Bits

```
1  #define GPIOA_BASE 0x48000000
2  #define GPIOA_MODER (*((volatile unsigned long *) (GPIOA_BASE + 0x00)))
3
4  // Set bit 3 of the GPIOA_MODER to 1.
5
6
7  // Clear bit 7 of the GPIOA_MODER (i.e., set to 0)
8
9
10
```

①

②

①

②

Pointers and Arrays

Pointers and Arrays in C: Arrays

```
1 int * p = (int*) 0x20000000;  
2  
3 int a = *p;  
4  
5 int b = *(p+3);  
6  
7 *(p+5) = b;
```

①

②

③

- ① Equivalent to **a**= _____
- ② Equivalent to **b**= _____, address _____
- ③ Equivalent to **p[5]**= _____, address _____

Pointers and Arrays in C: Strings

```
1 char * str = (char *) 0x20001000;  
2  
3 str[13] = 'A';
```

①

① Address _____

Dereferencing

```
1
2 int * p = (int*) 0x20000000;
3
4 int a = *p;
5 int * aptr = &a;
6
7 *aptr = 3;
8
9 int * ptr = &p[0];
10
11 ptr = &p[5]
12
13 *ptr = 42;
```

①

②

③

④

⑤

⑥

- ① Equivalent to a = _____
- ② **aptr** stores address of _____
- ③ same as **a=3**
- ④ **ptr=** _____
- ⑤ **ptr=** _____
- ⑥ **p[5]=** _____

Structures

```
1 struct optional_tag {  
2     type_1 identifier_1;  
3     type_2 identifier_2;  
4     ...  
5     type_N identifier_N;  
6 } optional_variable_definitions ;
```

Structures

```
1 struct contact {  
2     char name[30];  
3     unsigned long long phone;  
4     float height;  
5 };  
6  
7 struct contact jbrake; // example variable definition
```

How many bytes does this structure occupy in memory?

Using structures as part of a new type

Can also wrap in a typedef to avoid needing to use the `struct` keyword.

```
1 typedef struct my_tag {int i;} my_type; // Declaration of new type
2
3 // Creating variable with struct keyword
4
5
6
7 // Creating variable using new type
8
```

①

①

②

Arrow Operator

Use a structure to access a chunk of memory in a specific location.

```
1 typedef struct {
2     char first_name[30];
3     unsigned long long phone;
4     float height;
5 } contact_type;
6
7 contact_type jbrake;
8 strcpy(jbrake.first_name, "Josh Brake");
9 jbrake.phone = (unsigned long long) 9096218553;
10 contact_type * contact_type_ptr = &jbrake;
11 unsigned long long phone_num = contact_type_ptr->phone;
```

Wrap Up

- C is libertarian – will allow you to do many things, not all of which are good for you.
- Understanding certain C data structures like pointers and structures will enable you to more easily and naturally write code to control your MCU.
- The MCU reference manual contains the information needed to write code to configure and manipulate the peripherals using memory-mapped I/O.