EECS 388: Embedded Systems

2. Software Development

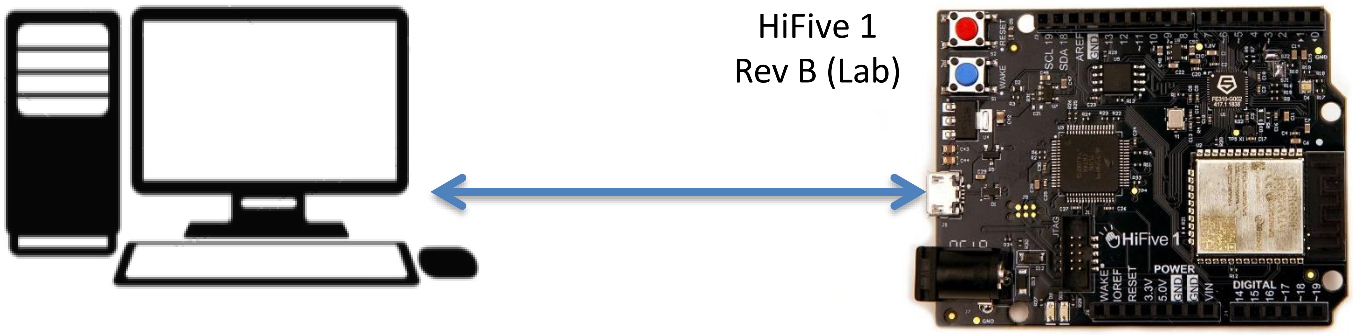
Heechul Yun

Agenda

- Embedded software development
  - Development models
  - Programming languages
  - Case study: KU AFS

Development Models

- Host/Target model
  - Edit, (cross) compile, debug, deploy on host (PC)
  - Target embedded system stores only the final compiled program image (firmware)



Host

Target

PC

HiFive 1
Rev B (Lab)

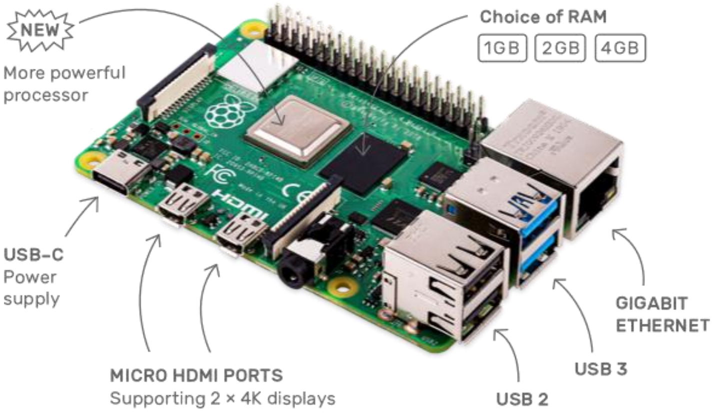
Where the software
is being developed

Where the software
is being run
(usually resource constrained)

Development Models

- Standalone model
  - Host/Target is the same system (same ISA)
  - Native compilation, debugging

Rasbperry Pi 4
(Lab)



Develop & execute on the same platform

Embedded Software Development Challenges

- Limited resources
  - Low computing performance
  - Small amount of memory and storage
- Low-level access to hardware
  - Memory-mapped I/O
  - For efficiency, low latency
- High diversity, complexity
  - Not well standardized
  - Difficult to develop (but better than used to be)

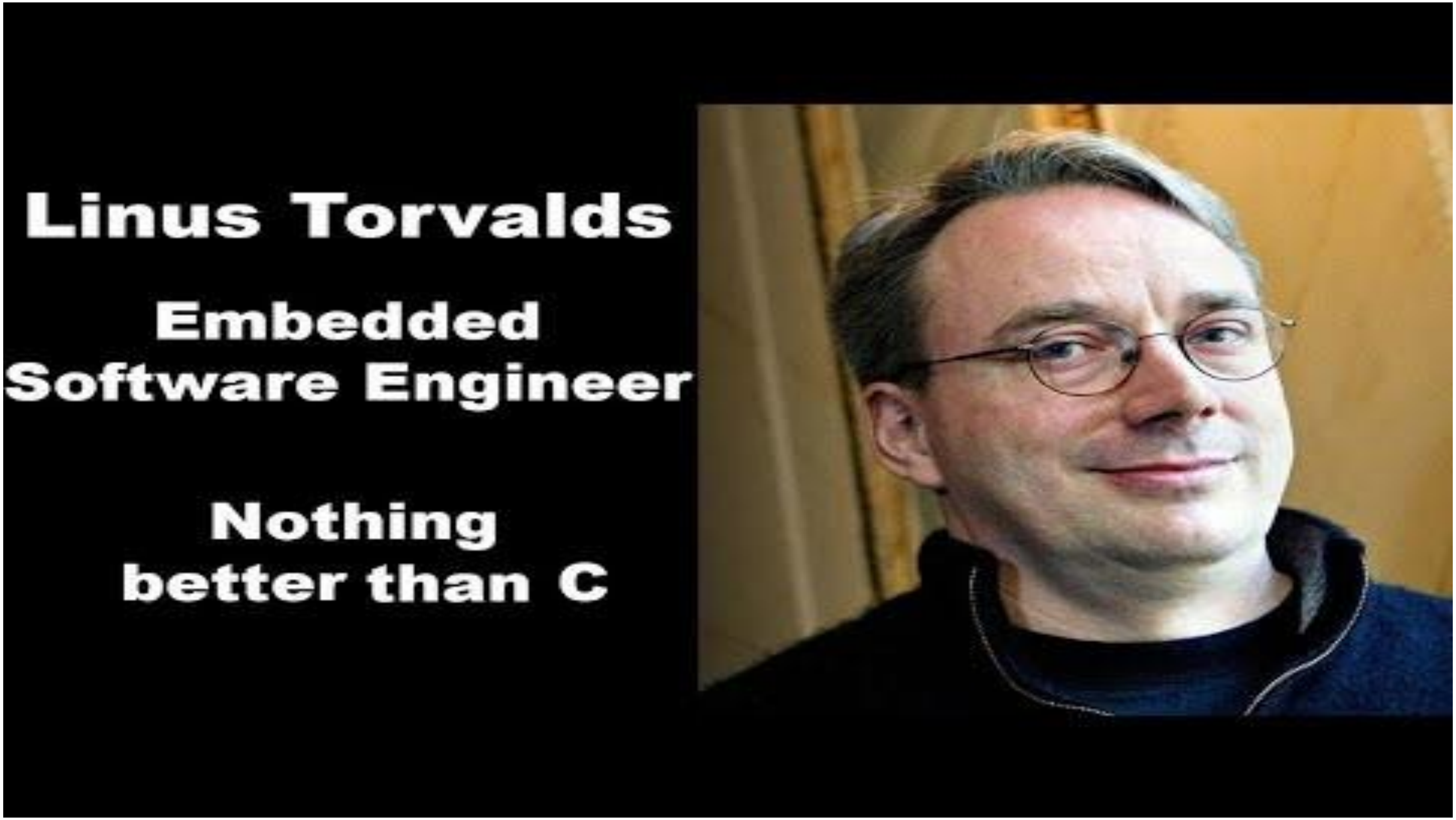
Programming Languages

- **C**
  - (**Still**) the most popular for embedded systems
- C++
- Java
- JavaScript
- Python
- Rust
- …

C

- History
  - 1972. At AT&T Bell Labs, On PDP-11. by Dennis Richie.
  - 1978. K&R
  - 1990. C89, ANSI-C
  - 1999. C99
  - 2007. C11
  - 2018. C18

Linus Torvalds: "Nothing better than C"



https://www.youtube.com/watch?v=CYvJPra7Ebk

C

- Why popular?
  - Fast, efficient, and portable
  - Close to machine (assembly-like control)
  - Pointer, minimal type checking
- Problems
  - Pointer, minimal type checking
  - Require manual control of dynamic memory
  - Unsafe (memory leak, undefined behavior, ..)
  - Difficult to write correct, safe, secure code

Number Systems

- Decimal (base 10)
  - Symbols: 0,1,…,9
  - E.g., $123_{10} = 1\text{x}10^2 + 2\text{x}10^1 + 3\text{x}10^0$
- Binary (base 2)
  - Symbols: 0,1
  - E.g., 1011_2 = **0b1011** = $1\text{x}2^3 + 0\text{x}2^2 + 1\text{x}2^1 + 1\text{x}2^0$
- Hexadecimal (base 16)
  - Symbols: 0,1,…,9,A,B,…,F
  - E.g., 123_{16} = **0x123** = $1\text{x}16^2 + 2\text{x}16^1 + 3\text{x}16^0$

Number Systems

- Examples

Decimal	Hexadecimal	Binary
0	0x0	0b0
2		
9		
	0xA	
	0xF	
	0x1F	
		0b1000 0000
		0b1000 0011
		0b1000 0000 0000 0000

Number Systems

- Examples

Decimal	Hexadecimal	Binary
0	0x0	0b0
2	0x2	0b10
9	0x9	0b1001
10	0xA	0b1010
15	0xF	0b1111
31	0x1F	0b1 1111
128	0x80	0b1000 0000
131	0x83	0b1000 0011
32768	0x8000	0b1000 0000 0000 0000

Data Types

- Char (8 bit)
  - Smallest addressable unit size (8 bit) integer (%c)
- Integer (16~64 bits)
  - Integer (%d), long integer (%li)
- Float (32 bit)
  - Single precision real number. (%f)
- Double (64 bit)
  - Double precision real-number (%lf)

Data Types

- Modifiers
  - long, short, unsigned, signed
- Examples

Data type	Storage	Range
char	8 bits	[-128,+127]
unsigned char	8 bits	[0,_____]
short int	16 bits	[-32768,+32767]
unsigned short int	16 bits	[0,_______]
int	16 or 32 bits	[-2^{15}, 2^{15}-1] or __________
long int	32 or 64 bits	[-2^{31}, 2^{31}-1] or __________
long long int	64 bits	[-2^{63}, 2^{63}-1]

Data Types

- Modifiers
  - – long, short, unsigned, signed
- Examples

Data type	Storage	Range
char	8 bits	[-128,+127]
unsigned char	8 bits	[0, 255]
short int	16 bits	[-32768,+32767]
unsigned short int	16 bits	[0, 65535]
int	16 or 32 bits	$[-2^{15}, 2^{15}-1]$ or $[-2^{31}, 2^{31}-1]$
long int	32 or 64 bits	$[-2^{31}, 2^{31}-1]$ or $[-2^{63}, 2^{63}-1]$
long long int	64 bits	$[-2^{63}, 2^{63}-1]$

Variables

- <modifier> <data type> <variable name>

```
char ch = 127; // 0x7f or 0b0111 1111
unsigned char uch = 255; // 0xff or 0b1111 1111
int ivar = 1234;
long int livar = 1234567890123;
float fvar = 1.234;
double dvar = 1.23456;
long double ddvar = 1.2345678;

printf("%c %c %d %li %f %lf %Lf\n",
        ch, uch, ivar, livar, fvar, dvar, ddvar);
```

Variables

- What will be the outputs?

```
char ch = 128;
unsigned char uch = 256;
int ivar = 2147483648;

printf("%d %d %d\n", ch, uch, ivar);
```

- Results on my PC
  - -128 0 -2147483648
- Why?

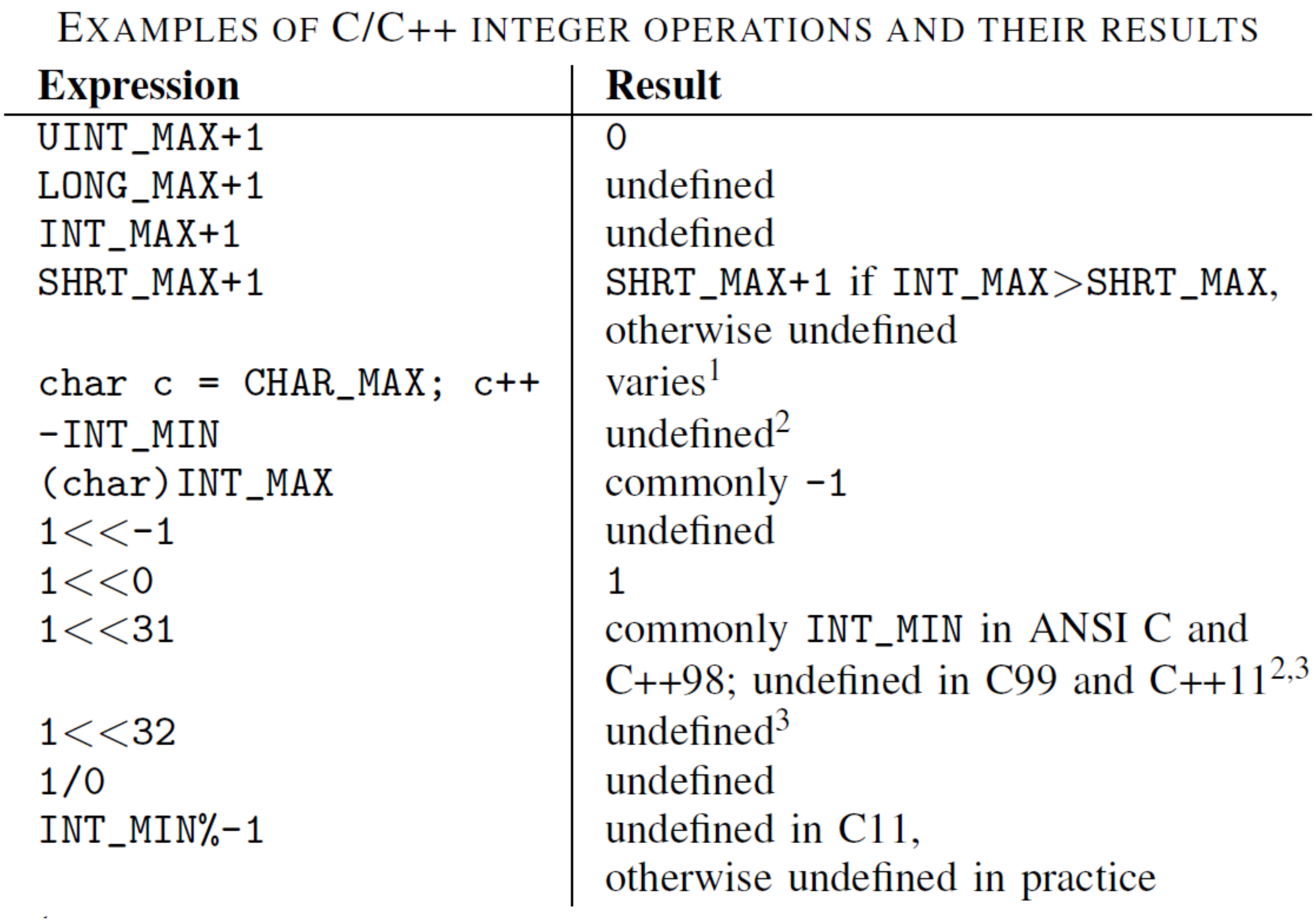
Integer Overflow

"These errors can lead to serious software failures, e.g., a truncation error on a cast of a floating point value to a 16-bit integer played a crucial role in the destruction of Ariane 5 flight 501 in 1996."

"These errors are also a source of serious vulnerabilities, such as integer overflow errors in OpenSSH and Firefox, both of which allow attackers to execute arbitrary code."

W. Dietz et al., "Understanding Integer Overflow in C/C++", ICSE, 2012

Undefined Behavior



EXAMPLES OF C/C++ INTEGER OPERATIONS AND THEIR RESULTS

Expression	Result
UINT_MAX+1	0
LONG_MAX+1	undefined
INT_MAX+1	undefined
SHRT_MAX+1	SHRT_MAX+1 if INT_MAX>SHRT_MAX, otherwise undefined
char c = CHAR_MAX; c++	varies[1]
-INT_MIN	undefined[2]
(char)INT_MAX	commonly -1
1<<-1	undefined
1<<0	1
1<<31	commonly INT_MIN in ANSI C and C++98; undefined in C99 and C++11[2,3]
1<<32	undefined[3]
1/0	undefined
INT_MIN%-1	undefined in C11, otherwise undefined in practice

W. Dietz et al., “Understanding Integer Overflow in C/C++”, ICSE, 2012

Recap: C

- Why is C popular for embedded?
  - Fast, efficient, and portable
  - Close to machine (assembly-like control)
  - Pointer, minimal type checking
- What are the problems of C for embedded?
  - Pointer, minimal type checking
  - Require manual control of dynamic memory
  - Unsafe (memory leak, undefined behavior, ..)
  - **Difficult to write correct, safe, secure code**

Recap: Number Systems

- Binary
  - Symbols: 0,1
  - E.g., 1011_2 = **0b1011** = $1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$
- Hexadecimal
  - Symbols: 0,1,…,9,A,B,…,F
  - E.g., 123_{16} = **0x123** = $1x16^2 + 2x16^1 + 3x16^0$
- Exercise
  - 0b1100 in hexadecimal? in decimal?
  - 0xFF in binary? in decimal?

Recap: Integer Overflow

- Integer data types (char, int) in C use finite bits
- Must be careful about possible overflow
- Example

```
int ivar = 2147483648;

printf("%d\n", ivar);

---
-2147483648
```

Basic Operators

- Arithmetic
  +, -, *, /, %
- Conditional
  ==, >, <, >=, <=
- Logical operators
  && (AND), || (OR), ! (NOT)
- Bitwise operators
  & (AND), | (OR), ^ (XOR), ~ (complement)
- Shift operators
  <<, >>
- Assignment operators
  +=, -=, *=, /=, %=, |=, &=, …

```
int va, vb, vc;
…
vc = va % vb;

if (va == 1000) {…}

if (va && vb) {…}

vc = va ^ vb;
vc = ~vb;

1 << 31;
va >> 16;

va += 10;
va *= 2;
…
```

Basic Operators

- Examples
  - Assume: a = 0b1000, b = 0b0001

Expression	Result
`a && b`	______
`a & b`	______
`a || b`	______
`a | b`	______
`(a>>3) & b`	______
`a & (b<<3);`	______
a == b	______
a >= b	______

Basic Operators

- Examples
  - Assume: a = 0b1000, b = 0b0001

Expression	Result
a && b	1
a & b	0
a || b	1
a | b	9
(a>>3) & b	1
a & (b<<3);	8
a == b	0
a >= b	1

Control

```
if (condition) {
  // code
}

if (condition) {
  // code
} else {
  // code
}

if (condition) {
  // code
} else if (condition) {
  // code
} else {
  // code
}
```

```
switch (expression) {
  case const-exp1:
    // code
    break;
  case const-exp2:
    // code
    break;
  …
  default:
    // code
    break;
}
```

Loop

```
while ( condition ) {
  // code
}

do {
  // code
} while (condition);

for (init; condition; expression) {
  // code
}
```

Function

main.c

```
#include <stdio.h>

int add(int a, int b);

void main()
{
   int c = add(1, 1);
   printf(“%d\n”, c);
}
```

mylib.c

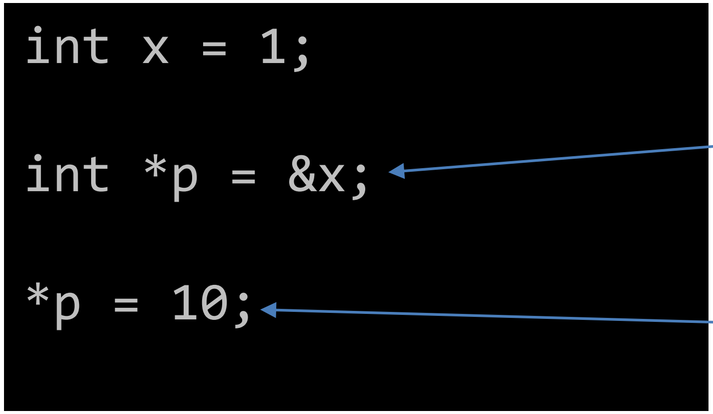
```
int add(int a, int b)
{
  return a + b;
}
```

Function implementation

Function declaration

func_type func_name (param_type1 param_name1, …)

Pointer



```
int x = 1;

int *p = &x;

*p = 10;
```

Declare an integer type pointer *p*, which points to x’s memory address

Update an integer value of what p is pointing to.

Pointer

```
int x = 1;

int *p = &x;

*p = 10;
```

Declare an integer type pointer *p*, which points to x's memory address

Update an integer value of what p is pointing to.

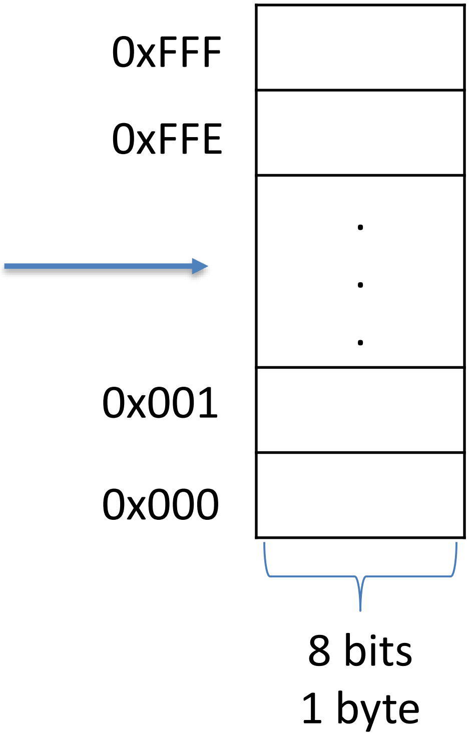
```
printf("addr(x) = %p\n", &x);
printf("addr(p) = %p\n", &p);
printf("p = %p\n", p);
printf("*p = %d\n", *p);
printf("x = %d\n", x);
```

```
addr(x) = 0x100
addr(p) = 0x104
p = 0x100
*p = 10
x = 10
```

Memory Address

- Byte addressed
  - Minimum unit = 1 byte (8 bits)

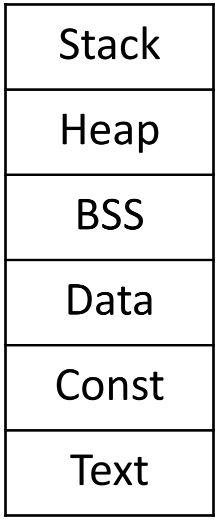
What's the size of this memory?



- Maximum addressable memory
  - Depends on the CPU architecture
    - 32 bit CPU: 2^{32} bytes
    - 16 bit CPU: ___ bytes
  - Depends on the platform
    - Some regions are mapped to ram, flash, or I/O devices
    - Some are unmapped.

Memory Regions

- Code
  - text program binary
- Data
  - const read-only constants
  - data initialized variables
  - bss zero initialized or uninitialized variables
  - heap dynamically allocated memory (malloc)
  - stack temporary storage for functions



Stack

Heap

BSS

Data

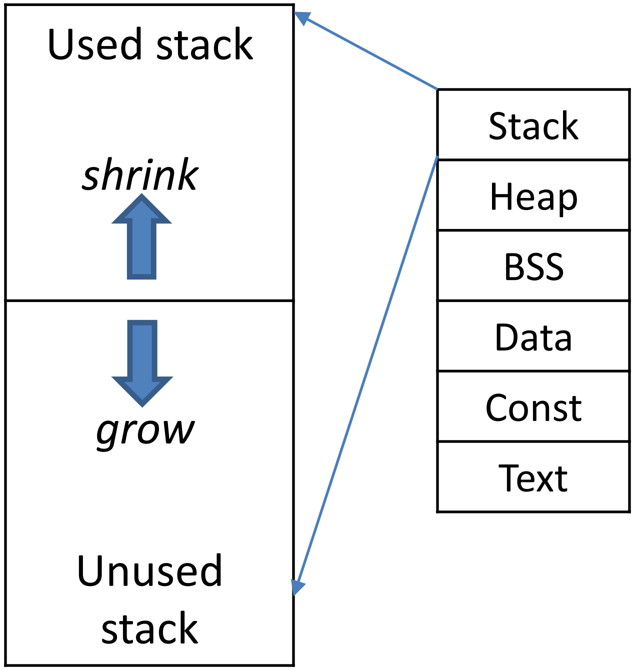
Const

Text

Stack

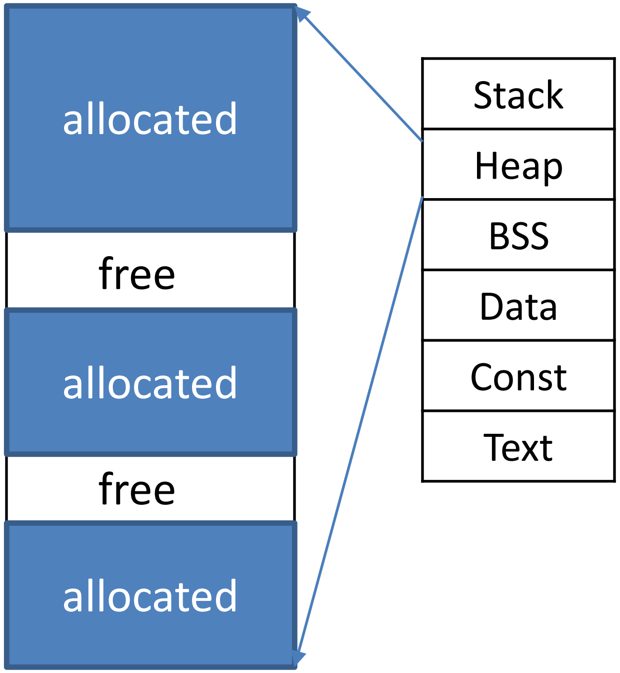
- Temporary storage
  - For functions
- Grow/shrink dynamically
  - Call a function → grow
  - Exit a function → shrink
- A stack frame
  - Local variables
  - Input parameters
  - Return address/value
  - Previous stack frame pointer

  …



Heap

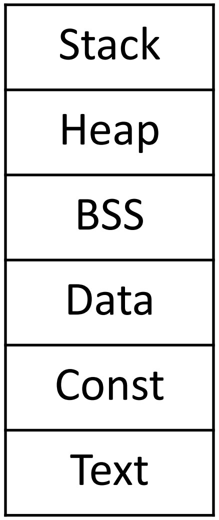
- Software managed dynamic memory
- Reserved at compile time
- Allocated/freed at runtime
  - malloc()
  - free()
- Potential issues
  - Memory leak
  - Fragmentation



Not recommended to use for critical embedded applications (e.g., automotive)

Example

```
int sum;

int sum2 = 100;

const int sum3 = 1000;

int add(int a, int b)
{
   int c = a + b;
   return c;
}
void main()
{
   char *buf = (char *)malloc(10);
   sum = add(1, 1);
}
```



Stack

Heap

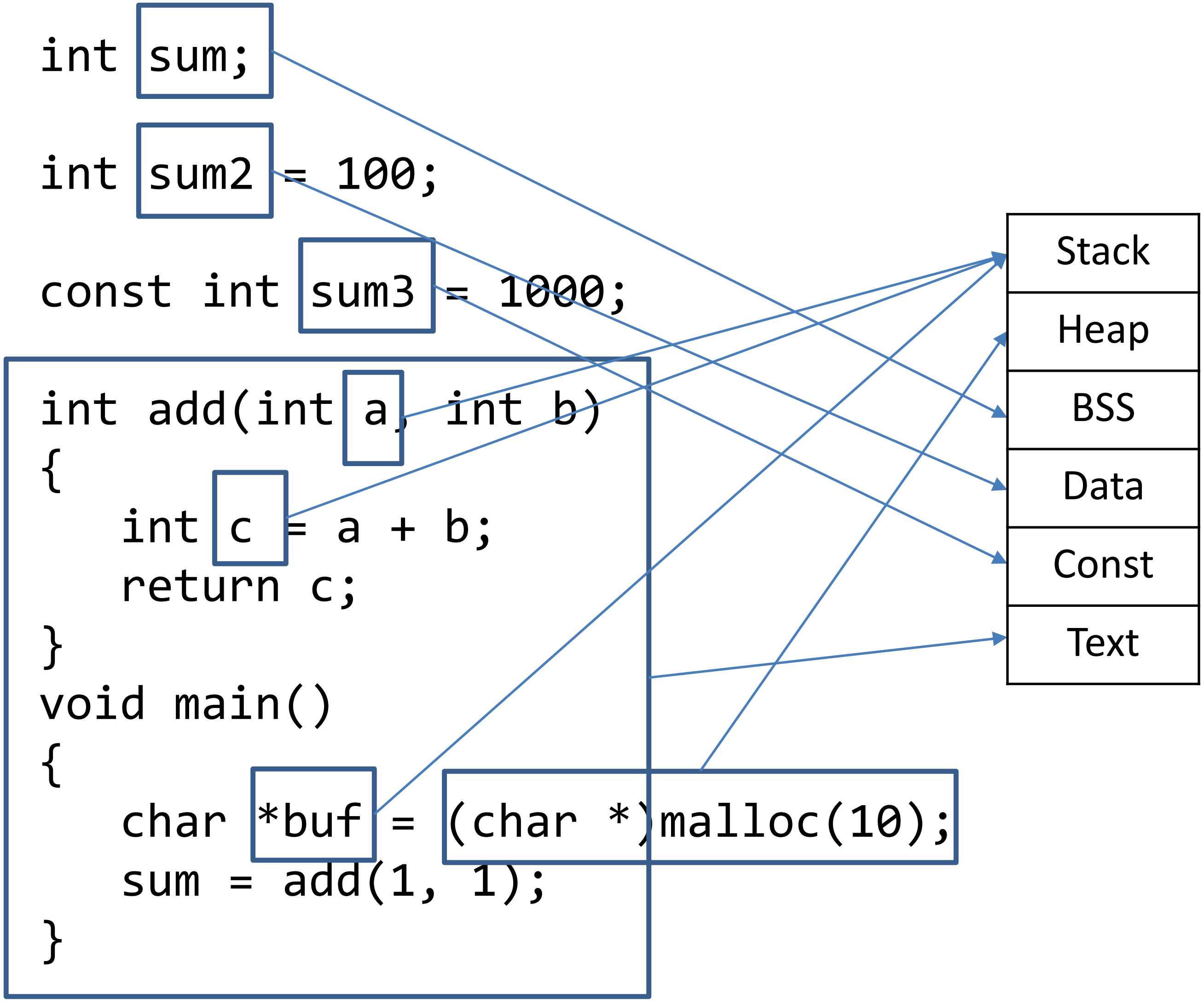
BSS

Data

Const

Text

Example



```
int sum;

int sum2 = 100;

const int sum3 = 1000;

int add(int a, int b)
{
    int c = a + b;
    return c;
}
void main()
{
    char *buf = (char *)malloc(10);
    sum = add(1, 1);
}
```

Stack

Heap

BSS

Data

Const

Text

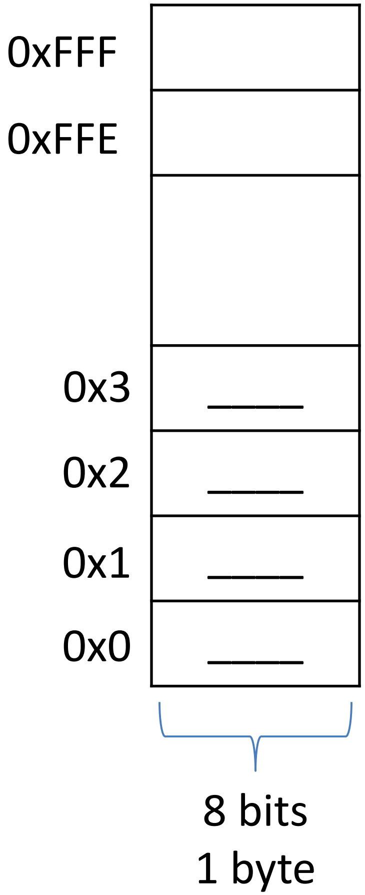
Variable Lifetime

```
int sum;

int sum2 = 100;

const int sum3 = 1000;

int add(int a, int b)
{
   int c = a + b;
   return c;
}
void main()
{
   char *buf = (char *)malloc(10);
   sum = add(1, 1);
}
```

- Program
  - Global variables
- Function
  - Local variables
  - Parameters
- Custom
  - Dynamically allocated memory

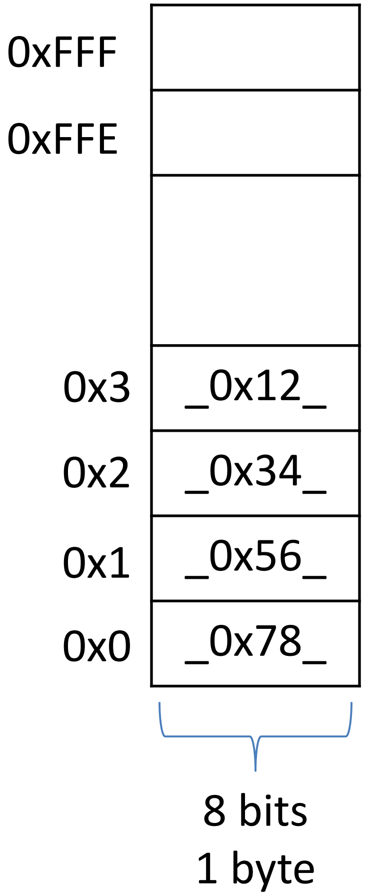
Endian

- Byte ordering
  - On storing multi-byte variables (short, int, long, etc.) on memory
- Example: MSB LSB
  - int x = 0x12345678;
  - assume &x = 0x0;
- Little endian: LSB first
- Big endian: MSB first



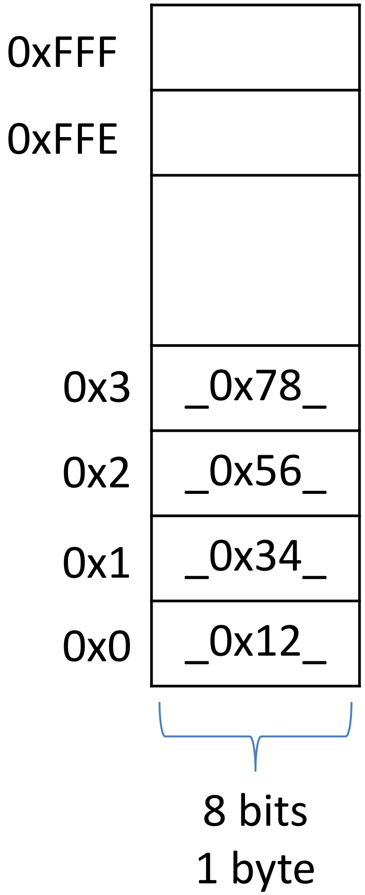
Endian

- Byte ordering
  - On storing multi-byte variables (short, int, long, etc.) on memory
- Example: MSB LSB
  - int x = 0x12345678;
  - assume &x = 0x0;
- **Little endian: LSB first**
- Big endian: MSB first



Endian

- Byte ordering
  - On storing multi-byte variables (short, int, long, etc.) on memory
- Example: MSB LSB
  - int x = 0x12345678;
  - assume &x = 0x0;
- Little endian: LSB first
- **Big endian: MSB first**



0xFFF

0xFFE

0x3 _0x78_

0x2 _0x56_

0x1 _0x34_

0x0 _0x12_

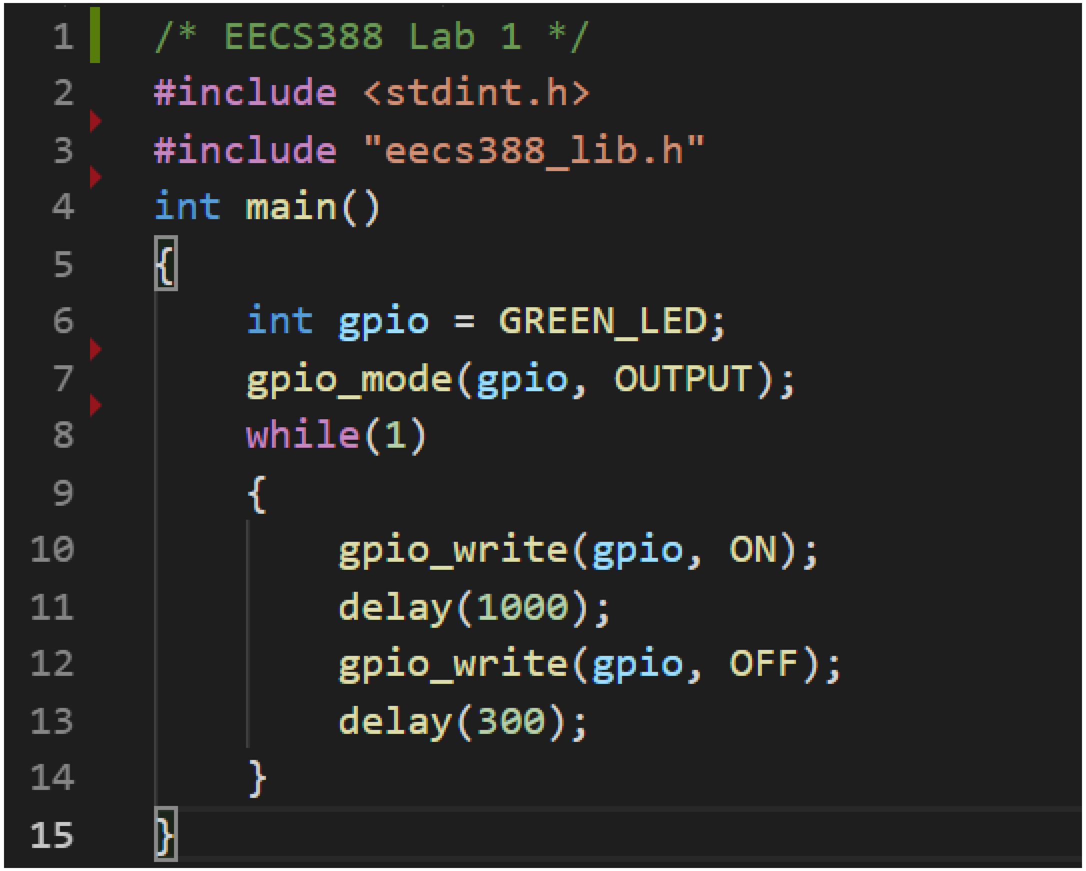
8 bits

1 byte

Recap

- C language review
- Memory regions
- Endian

C Program Example



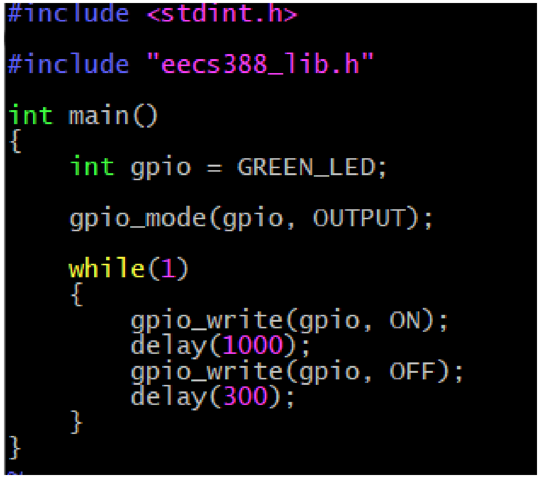
```c
/* EECS388 Lab 1 */
#include <stdint.h>
#include "eecs388_lib.h"
int main()
{
    int gpio = GREEN_LED;
    gpio_mode(gpio, OUTPUT);
    while(1)
    {
        gpio_write(gpio, ON);
        delay(1000);
        gpio_write(gpio, OFF);
        delay(300);
    }
}
```

Compilation

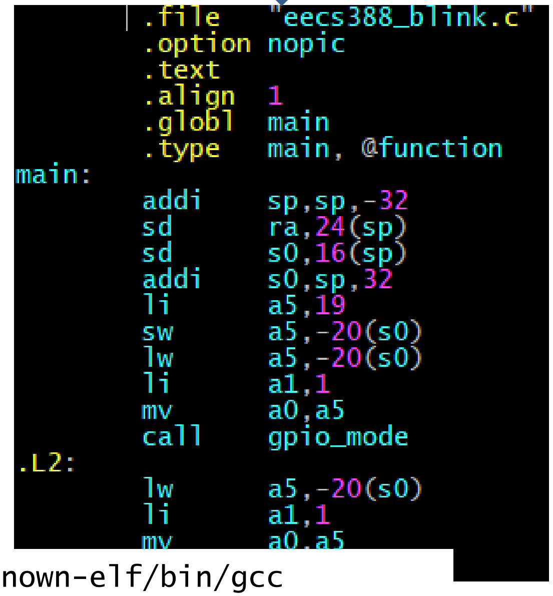
- **Compiler**
  - **C source -> assembly**

    $ gcc -c eecs388_blink.c -S

- Assembler
- Linker



```
#include <stdint.h>

#include "eecs388_lib.h"

int main()
{
    int gpio = GREEN_LED;

    gpio_mode(gpio, OUTPUT);

    while(1)
    {
        gpio_write(gpio, ON);
        delay(1000);
        gpio_write(gpio, OFF);
        delay(300);
    }
}
```



```
        .file   "eecs388_blink.c"
        .option nopic
        .text
        .align  1
        .globl  main
        .type   main, @function
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        li      a5,19
        sw      a5,-20(s0)
        lw      a5,-20(s0)
        li      a1,1
        mv      a0,a5
        call    gpio_mode
.L2:
        lw      a5,-20(s0)
        li      a1,1
        mv      a0,a5
```

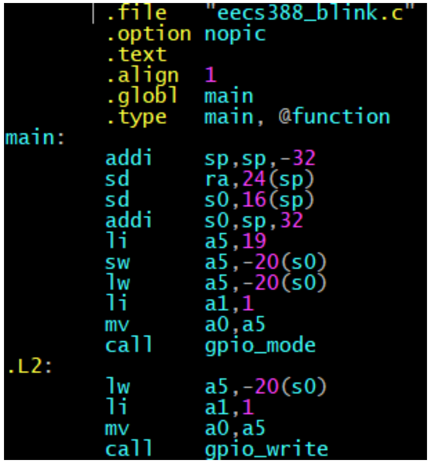
`<home>/.platformio/packages/toolchain-riscv/riscv64-unknown-elf/bin/gcc`

Compilation

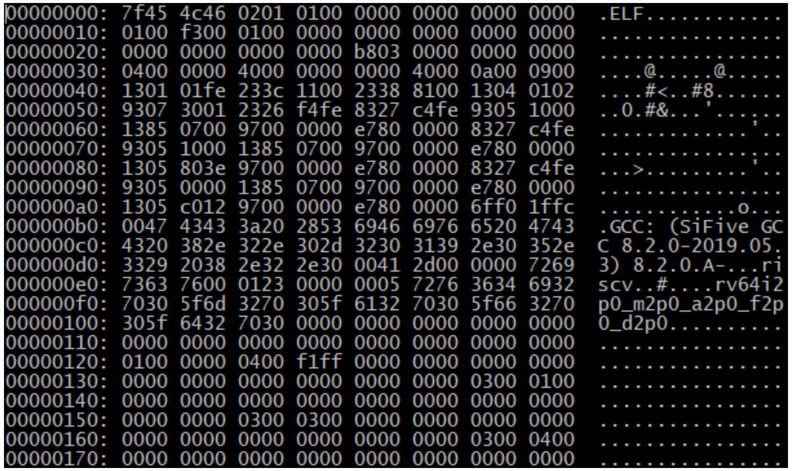
- Compiler
  - C source -> assembly
- **Assembler**
  - **Assembly -> binary object**

```
$ as eecs388_blink.s -o eecs388_blink.o
$ as eecs388_lib.s -o eecs388_lib.o
```

- Linker



```
        .file   "eecs388_blink.c"
        .option nopic
        .text
        .align  1
        .globl  main
        .type   main, @function
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        li      a5,19
        sw      a5,-20(s0)
        lw      a5,-20(s0)
        li      a1,1
        mv      a0,a5
        call    gpio_mode
.L2:
        lw      a5,-20(s0)
        li      a1,1
        mv      a0,a5
        call    gpio_write
```

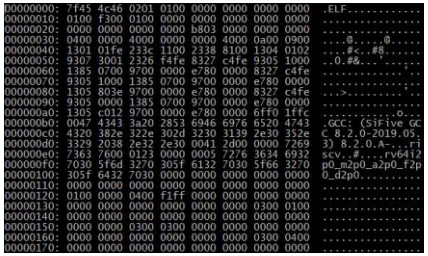


```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 f300 0100 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 b803 0000 0000 0000  ................
00000030: 0400 0000 4000 0000 0000 4000 0a00 0900  ....@.....@.....
00000040: 1301 01fe 233c 1100 2338 8100 1304 0102  ....#<..#8......
00000050: 9307 3001 2326 f4fe 8327 c4fe 9305 1000  ..0.#&...'......
00000060: 1385 0700 9700 0000 e780 0000 8327 c4fe  .............'..
00000070: 9305 1000 1385 0700 9700 0000 e780 0000  ................
00000080: 1305 803e 9700 0000 e780 0000 8327 c4fe  ...>.........'..
00000090: 9305 0000 1385 0700 9700 0000 e780 0000  ................
000000a0: 1305 c012 9700 0000 e780 0000 6ff0 1ffc  ............o...
000000b0: 0047 4343 3a20 2853 6946 6976 6520 4743  .GCC: (SiFive GC
000000c0: 4320 382e 322e 302d 3230 3139 2e30 352e  C 8.2.0-2019.05.
000000d0: 3329 2038 2e32 2e30 0041 2d00 0000 7269  3) 8.2.0.A-...ri
000000e0: 7363 7600 0123 0000 0005 7276 3634 6932  scv..#....rv64i2
000000f0: 7030 5f6d 3270 305f 6132 7030 5f66 3270  p0_m2p0_a2p0_f2p
00000100: 305f 6432 7030 0000 0000 0000 0000 0000  0_d2p0..........
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0100 0000 0400 f1ff 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0300 0100  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0300 0300 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0300 0400  ................
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

`<home>/.platformio/packages/toolchain-riscv/riscv64-unknown-elf/bin/as`

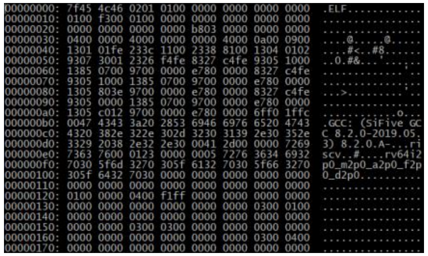
Compilation

- Compiler
  - C source -> assembly
- Assembler
  - Assembly -> binary object
- **Linker**
  - **Binary objects -> executable**
  - **Resolve memory addresses**



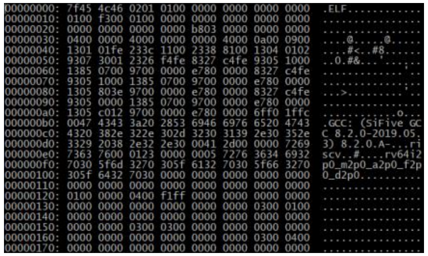
```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 f300 0100 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 b803 0000 0000 0000  ................
00000030: 0400 0000 4000 0000 0000 4000 0a00 0900  ....@.....@.....
00000040: 1301 01fe 233c 1100 2338 8100 1304 0102  ....#<..#8......
00000050: 9307 3001 2326 f4fe 8327 c4fe 9305 1000  ..0.#&...'......
00000060: 1385 0700 9700 0000 e780 0000 8327 c4fe  ..............'.
00000070: 9305 1000 1385 0700 9700 0000 e780 0000  ................
00000080: 1305 803e 9700 0000 e780 0000 8327 c4fe  ...>.........'..
00000090: 9305 0000 1385 0700 9700 0000 e780 0000  ................
000000a0: 1305 c012 9700 0000 e780 0000 6ff0 1ffc  .............o..
000000b0: 0047 4343 3a20 2853 6946 6976 6520 4743  .GCC: (SiFive GC
000000c0: 4320 382e 322e 302d 3230 3139 2e30 352e  C 8.2.0-2019.05.
000000d0: 3329 2038 2e32 2e30 0041 2d00 0000 7269  3) 8.2.0.A-...ri
000000e0: 7363 7600 0123 0000 0005 7276 3634 6932  scv..#....rv64i2
000000f0: 7030 5f6d 3270 305f 6132 7030 5f66 3270  p0_m2p0_a2p0_f2p
00000100: 305f 6432 7030 0000 0000 0000 0000 0000  0_d2p0..........
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0100 0000 0400 f1ff 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0300 0100  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0300 0300 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0300 0400  ................
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

eecs388_blink.o



```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 f300 0100 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 b803 0000 0000 0000  ................
00000030: 0400 0000 4000 0000 0000 4000 0a00 0900  ....@.....@.....
00000040: 1301 01fe 233c 1100 2338 8100 1304 0102  ....#<..#8......
00000050: 9307 3001 2326 f4fe 8327 c4fe 9305 1000  ..0.#&...'......
00000060: 1385 0700 9700 0000 e780 0000 8327 c4fe  ..............'.
00000070: 9305 1000 1385 0700 9700 0000 e780 0000  ................
00000080: 1305 803e 9700 0000 e780 0000 8327 c4fe  ...>.........'..
00000090: 9305 0000 1385 0700 9700 0000 e780 0000  ................
000000a0: 1305 c012 9700 0000 e780 0000 6ff0 1ffc  .............o..
000000b0: 0047 4343 3a20 2853 6946 6976 6520 4743  .GCC: (SiFive GC
000000c0: 4320 382e 322e 302d 3230 3139 2e30 352e  C 8.2.0-2019.05.
000000d0: 3329 2038 2e32 2e30 0041 2d00 0000 7269  3) 8.2.0.A-...ri
000000e0: 7363 7600 0123 0000 0005 7276 3634 6932  scv..#....rv64i2
000000f0: 7030 5f6d 3270 305f 6132 7030 5f66 3270  p0_m2p0_a2p0_f2p
00000100: 305f 6432 7030 0000 0000 0000 0000 0000  0_d2p0..........
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0100 0000 0400 f1ff 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0300 0100  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0300 0300 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0300 0400  ................
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

eecs388_lib.o



```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 f300 0100 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 b803 0000 0000 0000  ................
00000030: 0400 0000 4000 0000 0000 4000 0a00 0900  ....@.....@.....
00000040: 1301 01fe 233c 1100 2338 8100 1304 0102  ....#<..#8......
00000050: 9307 3001 2326 f4fe 8327 c4fe 9305 1000  ..0.#&...'......
00000060: 1385 0700 9700 0000 e780 0000 8327 c4fe  ..............'.
00000070: 9305 1000 1385 0700 9700 0000 e780 0000  ................
00000080: 1305 803e 9700 0000 e780 0000 8327 c4fe  ...>.........'..
00000090: 9305 0000 1385 0700 9700 0000 e780 0000  ................
000000a0: 1305 c012 9700 0000 e780 0000 6ff0 1ffc  .............o..
000000b0: 0047 4343 3a20 2853 6946 6976 6520 4743  .GCC: (SiFive GC
000000c0: 4320 382e 322e 302d 3230 3139 2e30 352e  C 8.2.0-2019.05.
000000d0: 3329 2038 2e32 2e30 0041 2d00 0000 7269  3) 8.2.0.A-...ri
000000e0: 7363 7600 0123 0000 0005 7276 3634 6932  scv..#....rv64i2
000000f0: 7030 5f6d 3270 305f 6132 7030 5f66 3270  p0_m2p0_a2p0_f2p
00000100: 305f 6432 7030 0000 0000 0000 0000 0000  0_d2p0..........
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0100 0000 0400 f1ff 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0300 0100  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0300 0300 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0300 0400  ................
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

firmware.elf

```
$ ld eecs388_blink.o eecs388_blink.o -o firmware.elf -T <linker_script>
```

```
<home>/.platformio/packages/toolchain-riscv/riscv64-unknown-elf/bin/ld
```

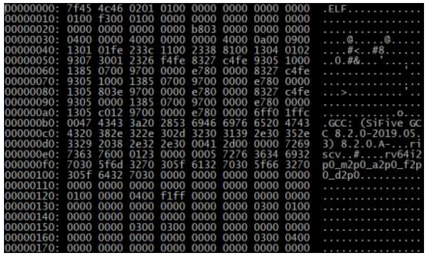
Linker Script Example

```
OUTPUT_ARCH("riscv")
ENTRY(_enter)
MEMORY
{
        flash (rxai!w) : ORIGIN = 0x20010000, LENGTH = 0x6a120
        ram (wxa!ri) : ORIGIN = 0x80000000, LENGTH = 0x4000
}
SECTIONS
{
        .init : > flash
        .text: > flash
        .rodata: > flash
        .data: > ram
        .bss: > ram
        .stack: > ram
        .heap: > ram
}
```

`<home>/.platformio/packages/framework-freedom-e-sdk/bsp/sifive-hifive1-revb/metal.default.lds`

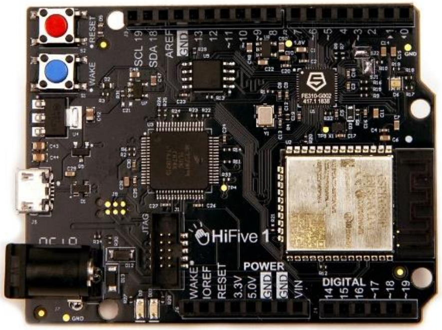
Execution

- Compiler
  - C source -> assembly
- Assembler
  - Assembly -> binary object
- Linker
  - Binary objects -> executable
- **Flashing/Loading**
  - **Executable -> (target) memory**

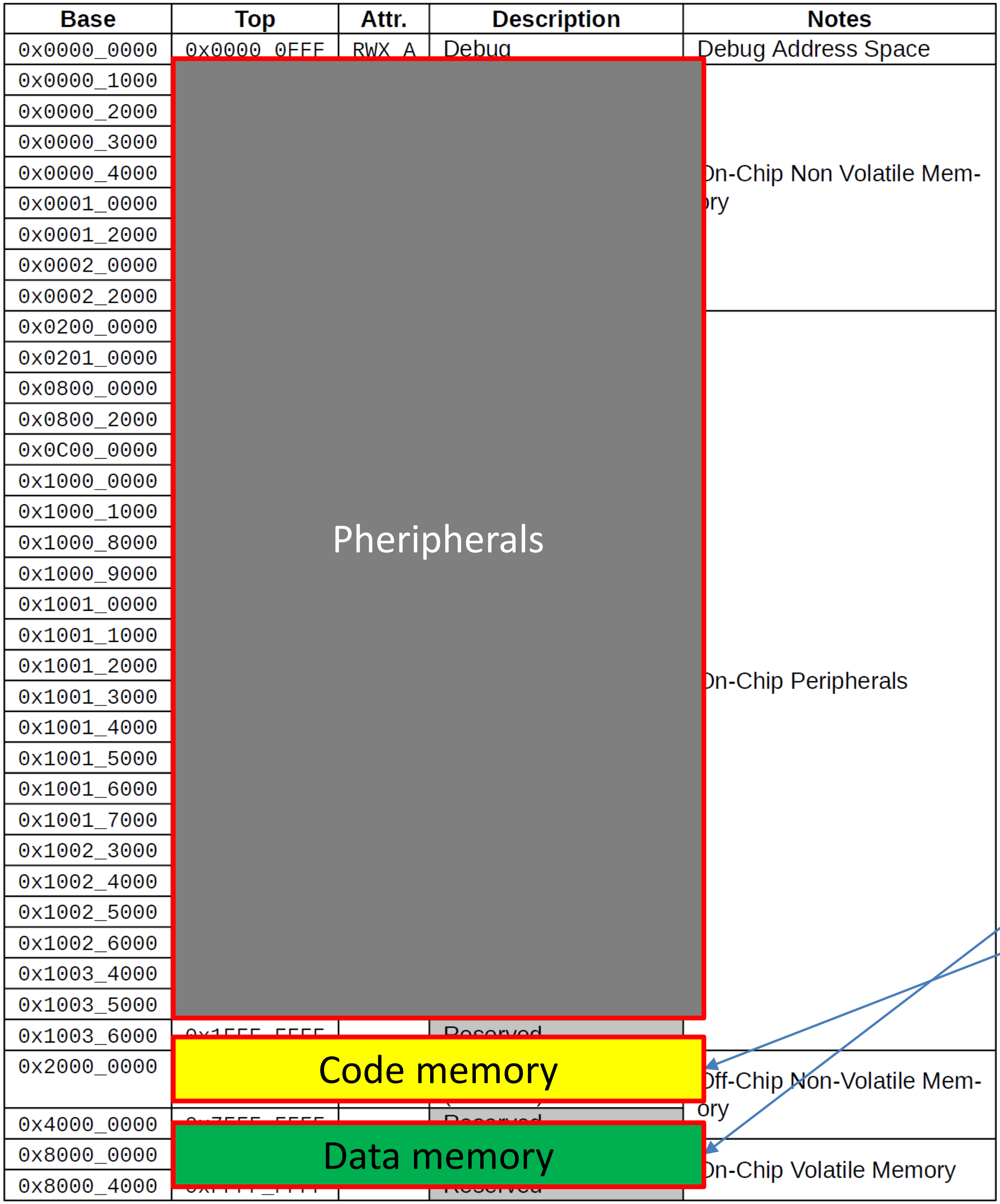


```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 f300 0100 0000 0000 0000 0000 0000  ................
00000020: 0000 0000 0000 0000 b803 0000 0000 0000  ................
00000030: 0400 0000 4000 0000 0000 4000 0a00 0900  ....@.....@.....
00000040: 1301 01fe 233c 1100 2338 8100 1304 0102  ....#<..#8......
00000050: 9307 3001 2326 f4fe 8327 c4fe 9305 1000  ..0.#&...'......
00000060: 1385 0700 9700 0000 e780 0000 8327 c4fe  .............'..
00000070: 9305 1000 1385 0700 9700 0000 e780 0000  ................
00000080: 1305 803e 9700 0000 e780 0000 8327 c4fe  ...>.........'..
00000090: 9305 0000 1385 0700 9700 0000 e780 0000  ................
000000a0: 1305 c012 9700 0000 e780 0000 6ff0 1ffc  ............o...
000000b0: 0047 4343 3a20 2853 6946 6976 6520 4743  .GCC: (SiFive GC
000000c0: 4320 382e 322e 302d 3230 3139 2e30 352e  C 8.2.0-2019.05.
000000d0: 3329 2038 2e32 2e30 0041 2d00 0000 7269  3) 8.2.0.A-...ri
000000e0: 7363 7600 0123 0000 0005 7276 3634 6932  scv..#....rv64i2
000000f0: 7030 5f6d 3270 305f 6132 7030 5f66 3270  p0_m2p0_a2p0_f2p
00000100: 305f 6432 7030 0000 0000 0000 0000 0000  0_d2p0..........
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0100 0000 0400 f1ff 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0300 0100  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0300 0300 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0300 0400  ................
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

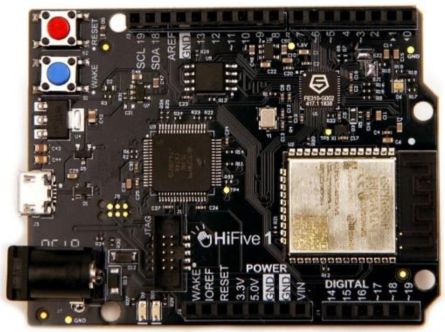
firmware.elf



Memory Map of SiFive FE310



Base	Top	Attr.	Description	Notes
0x0000_0000	0x0000_0FFF	RWX A	Debug	Debug Address Space
0x0000_1000				On-Chip Non Volatile Memory
0x0000_2000				
0x0000_3000				
0x0000_4000				
0x0001_0000				
0x0001_2000				
0x0002_0000				
0x0002_2000				
0x0200_0000				On-Chip Peripherals
0x0201_0000				
0x0800_0000				
0x0800_2000				
0x0C00_0000				
0x1000_0000				
0x1000_1000				
0x1000_8000				
0x1000_9000				
0x1001_0000				
0x1001_1000				
0x1001_2000				
0x1001_3000				
0x1001_4000				
0x1001_5000				
0x1001_6000				
0x1001_7000				
0x1002_3000				
0x1002_4000				
0x1002_5000				
0x1002_6000				
0x1003_4000				
0x1003_5000				
0x1003_6000	0x1FFF_FFFF		Reserved	
0x2000_0000				Off-Chip Non-Volatile Memory
0x4000_0000	0x7FFF_FFFF		Reserved	
0x8000_0000				On-Chip Volatile Memory
0x8000_4000	0xFFFF_FFFF		Reserved	



CPU: 32 bit RISC-V

Clock: 320 MHz

SRAM: 16 KB (D)

Flash: 4MB

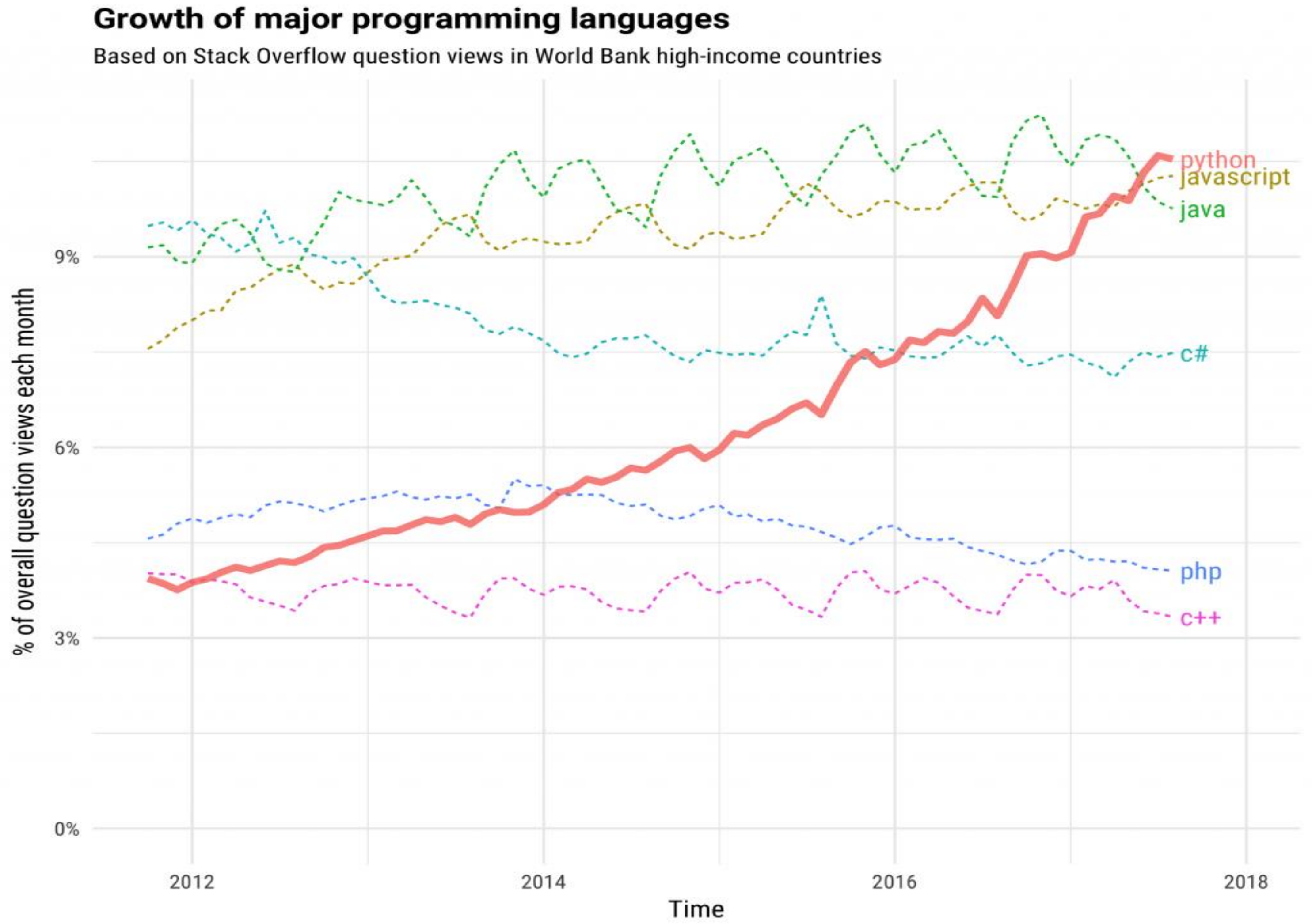
MISRA-C

- Coding guidelines for C to improve safety, security, portability, reliability in embedded C applications
- Defined by Motor Industry Software Reliability Association (MISRA)
- Widely adopted in automotive, aerospace, medical devices, defense, railways, …
- Example guidelines
  - **Use fixed width types (e.g., int32_t over int)**
  - **Avoid dynamic memory allocation**

C++

- History
  - 1979. Created as an extension of C ("C with Classes") by Stroustrup at AT&T Bell Labs.
  - 1983. Renamed to C++
  - 1985. First commercial implementation of C++
  - 1989, C++ 2.0
  - 2011, C++11
  - 2014, C++14
  - 2017, C++17
- Comparison to C
  - Much more powerful than C, yet still fast, efficient, portable, and widely available (albeit a bit less so than C).
  - Can be quite complex (e.g., template)

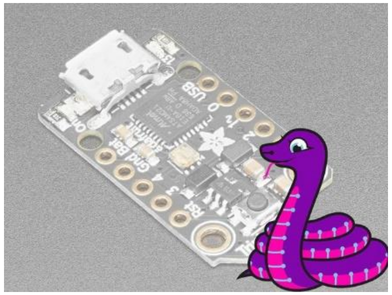
Python



Source: Stack Overflow

Python

- Why popular?
  - Easy to program
  - Powerful standard library packages
- Problems
  - Need more oomph, memory, storage
  - Not good for strict real-time applications
- Trends
  - Embedded systems are becoming more powerful
  - Many (e.g., Raspberry Pi) afford to run python
- Python for microcontrollers
  - MicroPython, CircuitPython
  - Works for 32bit ARM Cortex-M class microcontrollers



Python

- Integer types in python
  - Effectively no limit to how long an integer value can be (only constrained by system memory size)

```
>>> print(2147483648 + 1)
2147483649
>>> print(21474836482147483648 + 1)
21474836482147483649
>>> print(214748364821474836482147483648 + 1)
214748364821474836482147483649
>>> print(2147483648214748364821474836482147483648 + 1)
2147483648214748364821474836482147483649
```

Compiler vs. Interpreter

- Compiler
  - Translate the source code into machine code
  - The machine code runs directly on the machine
  - Compiled programs are usually (much) faster than interpreted ones
- Interpreter
  - Read and directly execute the source code
  - The interpreter and source code are needed to execute
  - Interpreted programs are usually (much) slower than compiled programs
- A language can be either compiled or interpreted.
  - C/C++ programs are almost always compiled
  - Python programs are mostly interpreted (can be partly compiled).

MicroPython vs. C/C++

https://github.com/micropython/micropython/wiki/Performance

On Teensy 3.1: (96Mhz ARM)

```
def performanceTest():
    millis = pyb.millis
    endTime = millis() + 10000
    count = 0
    while millis() < endTime:
        count += 1
    print("Count: ", count)
```

```
void setup() {
    Serial1.begin(115200);
    uint32_t endTime = millis() + 10000;
    uint32_t count = 0;
    while (millis() < endTime)
        count++;
    Serial1.print("Count: ");
    Serial1.println(count);
}
```

Count: 1,098,681

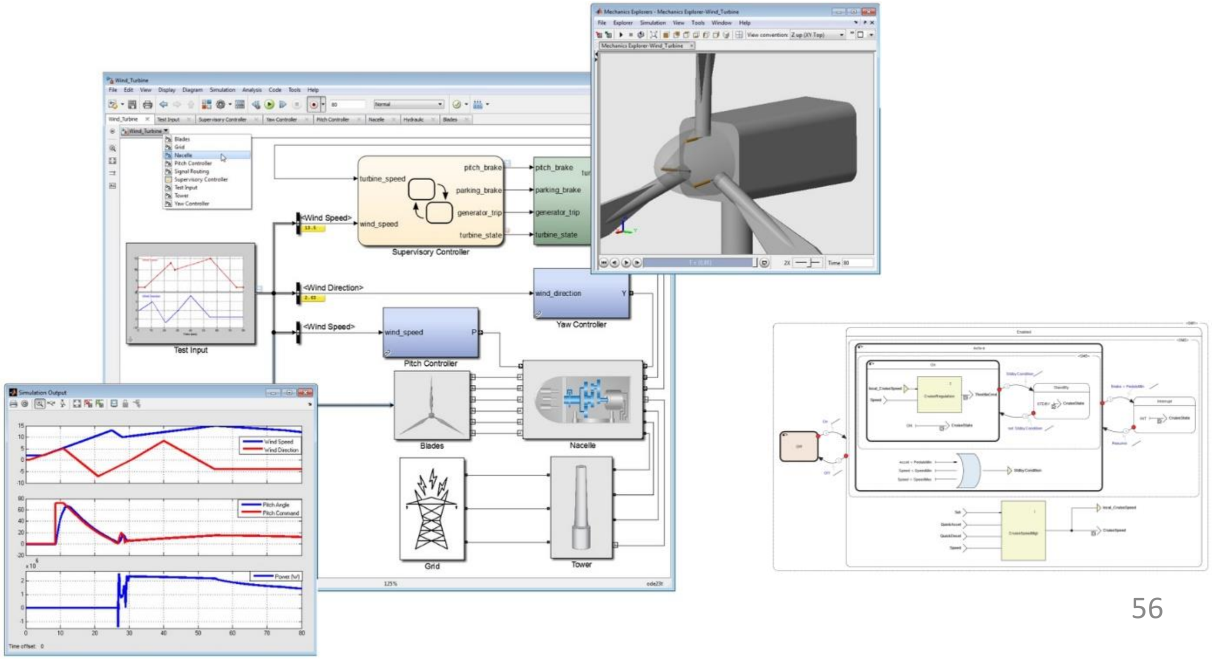
Count: 95,835,923

~100X slower than C!

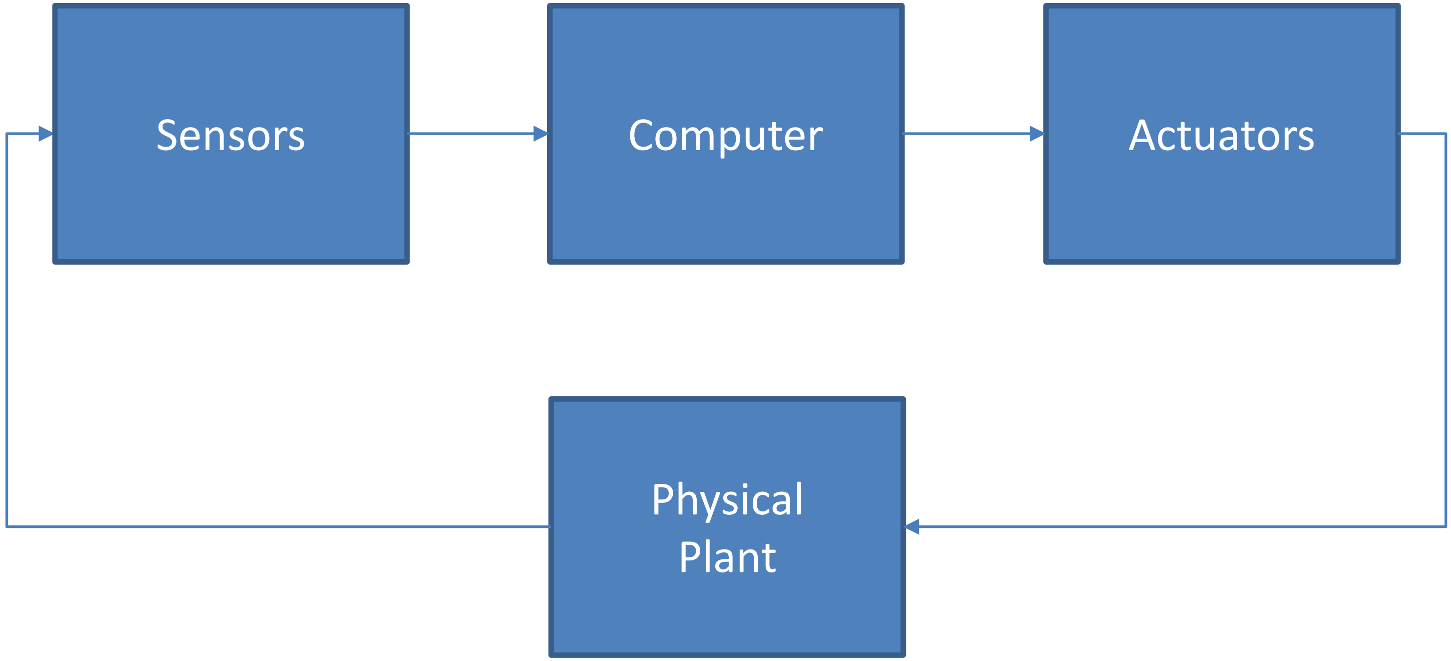
WARNING: Not a very rigorous performance comparison

Modeling Languages & Tools

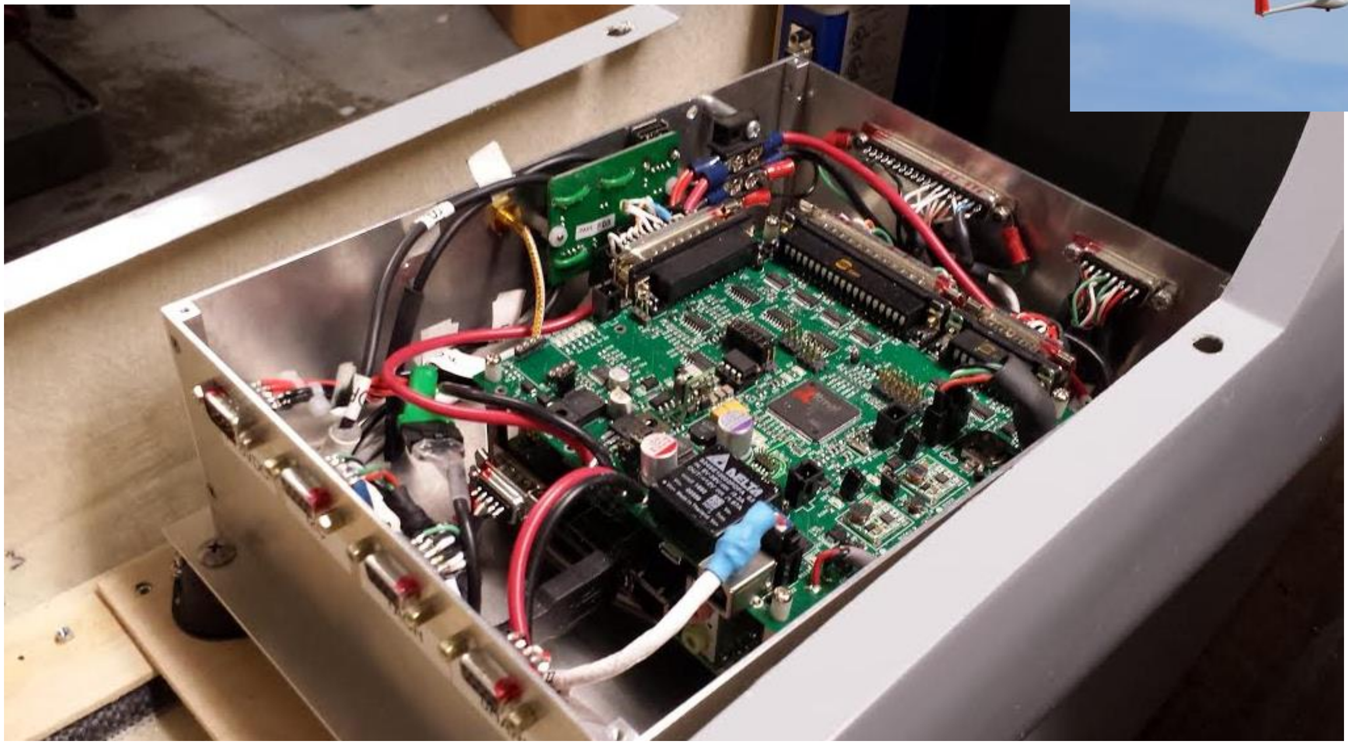
- **Matlab/Simulink**
  - Very popular for control engineers in automotive, aerospace, and other engineering domains
  - Mostly for modeling and analysis
  - Can generate C code for target deployment
- LabVIEW
- SCADE
- Modelica
- …



Embedded/Cyber-Physical System



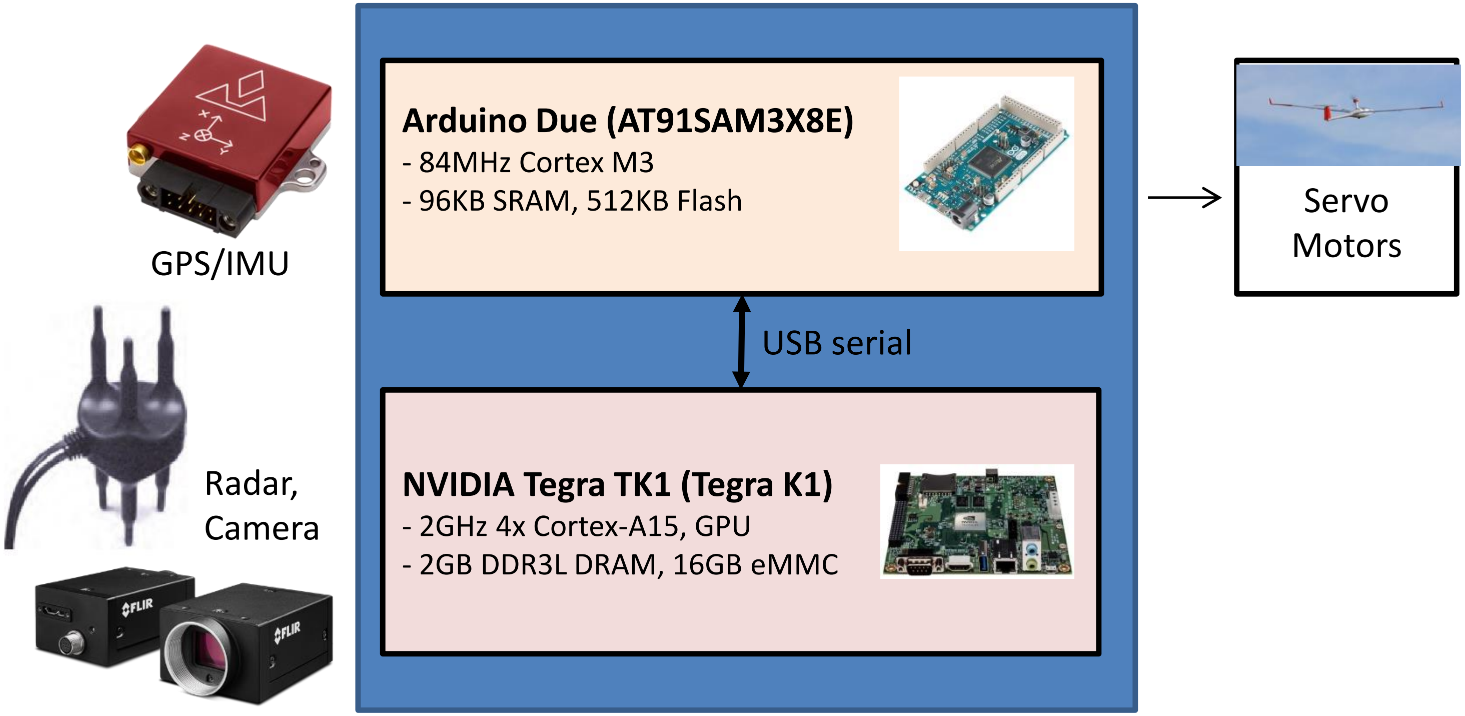
Example: KU AFS



[C] Prasanth Vivekanandan, Gonzalo Garcia, Heechul Yun, Shawn Keshmiri. A Simplex Architecture for Intelligent and Safe Unmanned Aerial Vehicles. *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA),* IEEE, 2016. [paper] [slides]

KU AFS

- Hardware



GPS/IMU

Radar,
Camera

Arduino Due (AT91SAM3X8E)
- 84MHz Cortex M3
- 96KB SRAM, 512KB Flash

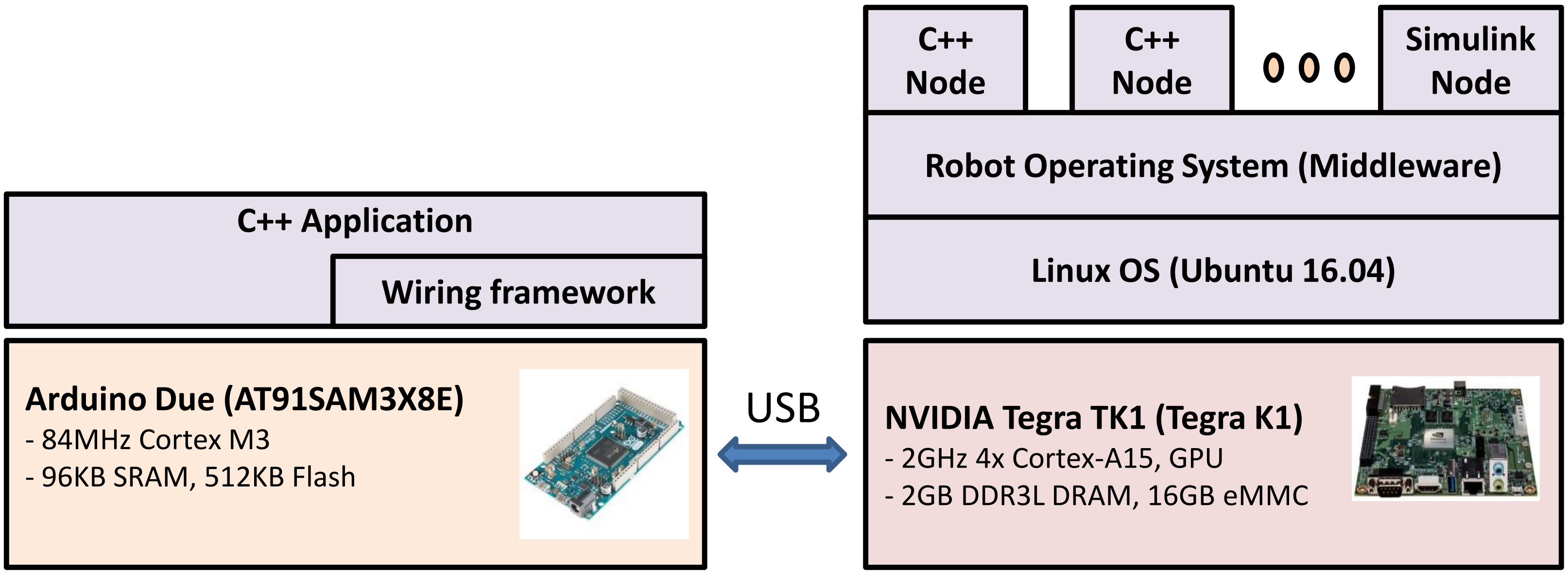
USB serial

NVIDIA Tegra TK1 (Tegra K1)
- 2GHz 4x Cortex-A15, GPU
- 2GB DDR3L DRAM, 16GB eMMC

Servo
Motors

KU AFS

- Software



C++ Node

C++ Node

Simulink Node

Robot Operating System (Middleware)

Linux OS (Ubuntu 16.04)

C++ Application

Wiring framework

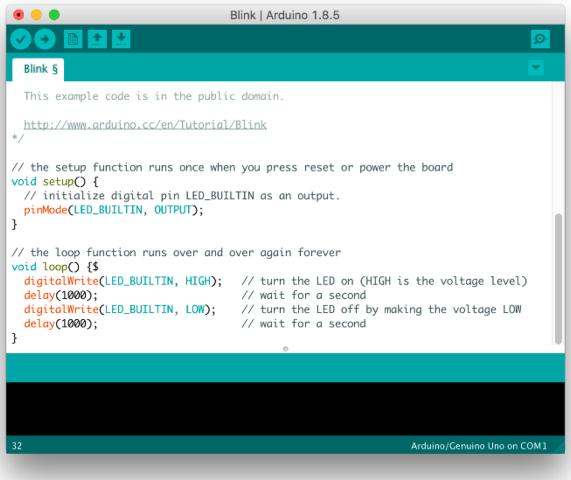
Arduino Due (AT91SAM3X8E)
- 84MHz Cortex M3
- 96KB SRAM, 512KB Flash

USB

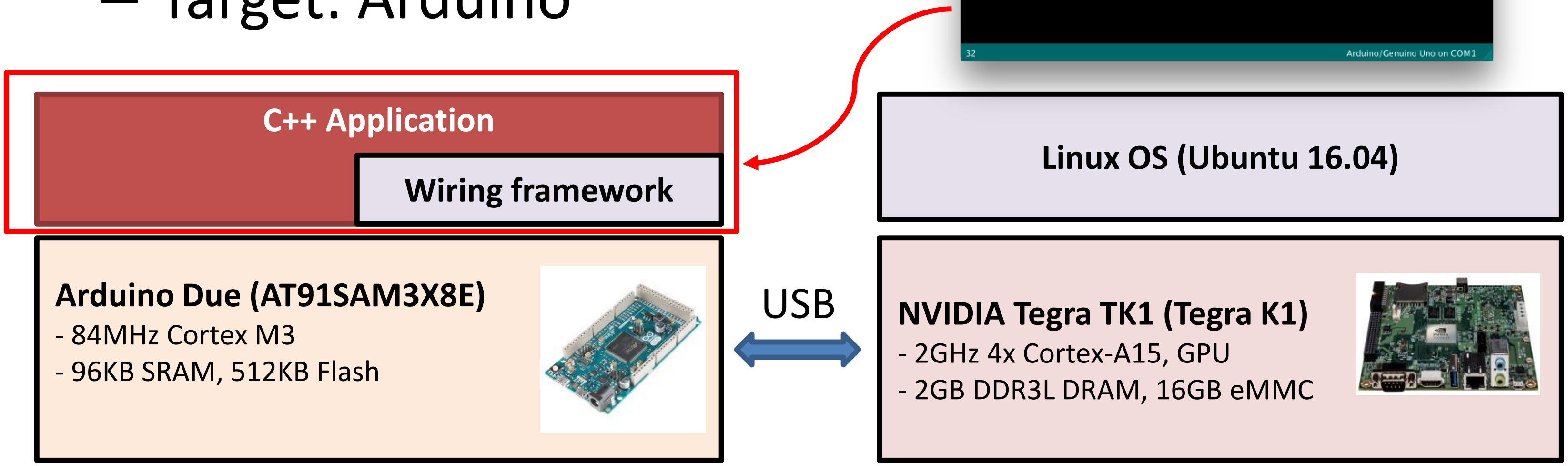
NVIDIA Tegra TK1 (Tegra K1)
- 2GHz 4x Cortex-A15, GPU
- 2GB DDR3L DRAM, 16GB eMMC

KU AFS

- Arduino Programming
  - Host/target model
  - Host: TK1
  - Target: Arduino



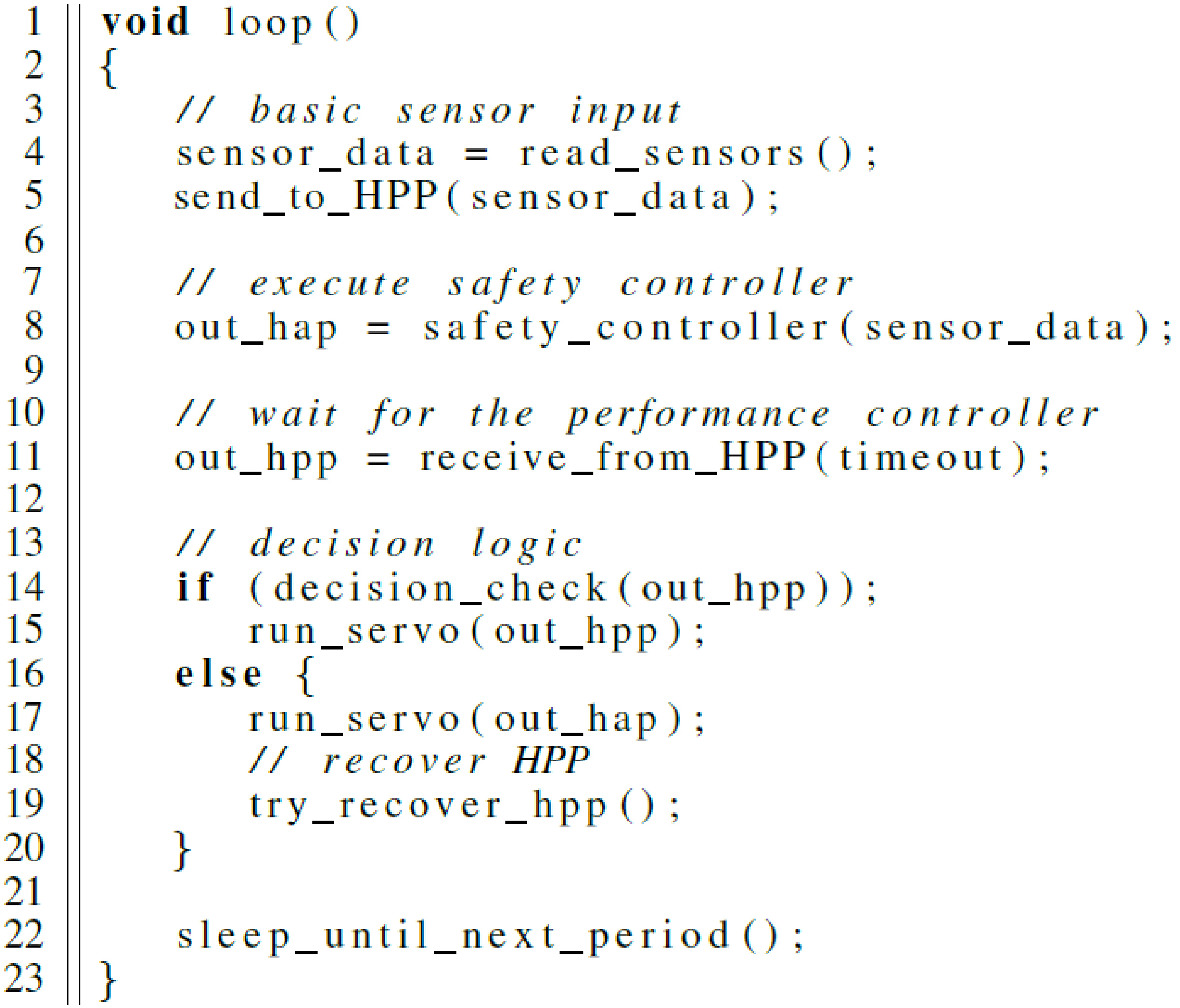
```
  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {$
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```



Arduino Due (AT91SAM3X8E)
- 84MHz Cortex M3
- 96KB SRAM, 512KB Flash

NVIDIA Tegra TK1 (Tegra K1)
- 2GHz 4x Cortex-A15, GPU
- 2GB DDR3L DRAM, 16GB eMMC

Arduino C++ Code



```
void loop()
{
    // basic sensor input
    sensor_data = read_sensors();
    send_to_HPP(sensor_data);

    // execute safety controller
    out_hap = safety_controller(sensor_data);

    // wait for the performance controller
    out_hpp = receive_from_HPP(timeout);

    // decision logic
    if (decision_check(out_hpp));
        run_servo(out_hpp);
    else {
        run_servo(out_hap);
        // recover HPP
        try_recover_hpp();
    }

    sleep_until_next_period();
}
```

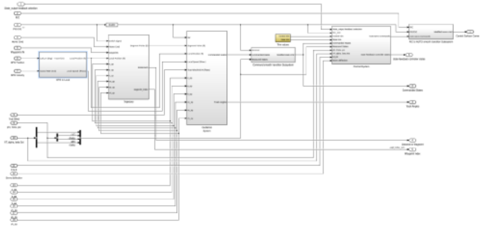
KU AFS

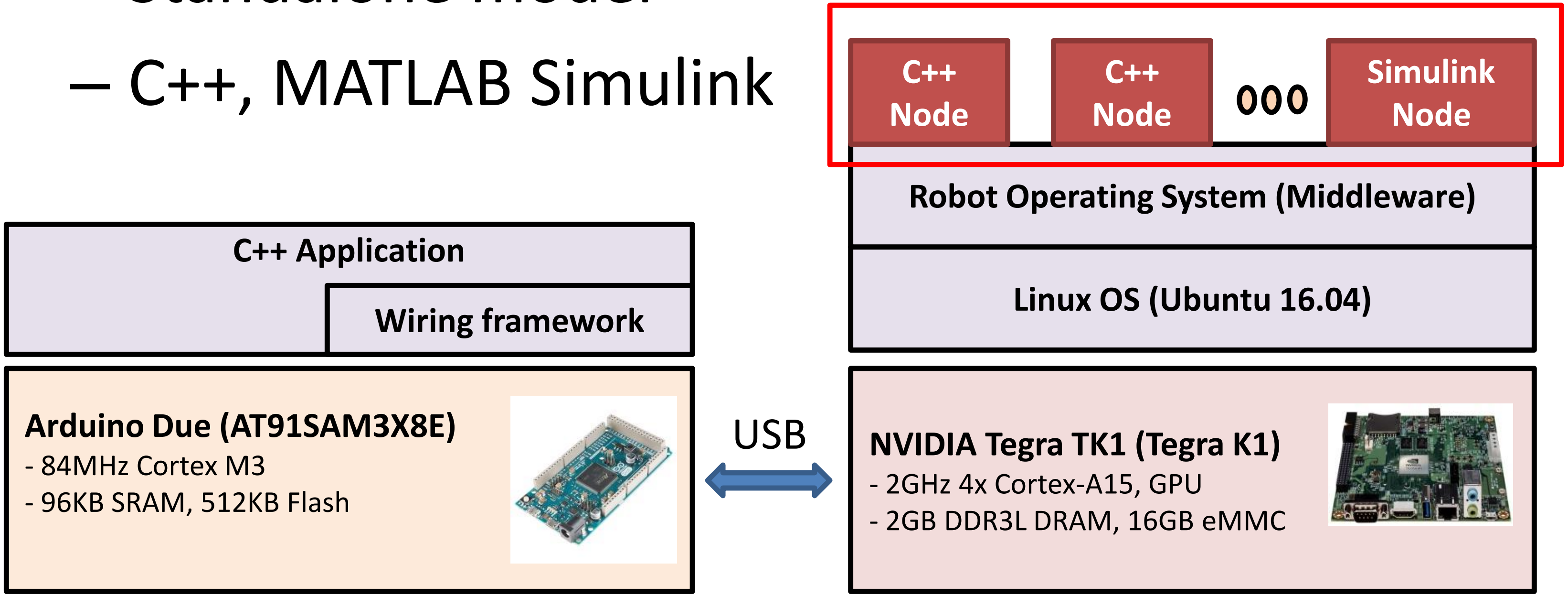
- Tegra TK1 Programming
  - Standalone model
  - C++, MATLAB Simulink

Welcome to GNU Emacs, one component of the GNU/Linux operating system.

```
Emacs Tutorial          Learn basic keystroke commands
Emacs Guided Tour       Overview of Emacs features at gnu.org
View Emacs Manual       View the Emacs manual using Info
Absence of Warranty     GNU Emacs comes with ABSOLUTELY NO WARRANTY
Copying Conditions      Conditions for redistributing and changing Emacs
Ordering Manuals        Purchasing printed copies of manuals
```

```
U:%%-  *GNU Emacs*   Top L5    (Fundamental)
For information about GNU Emacs and the GNU system, type C-h C-a.
```



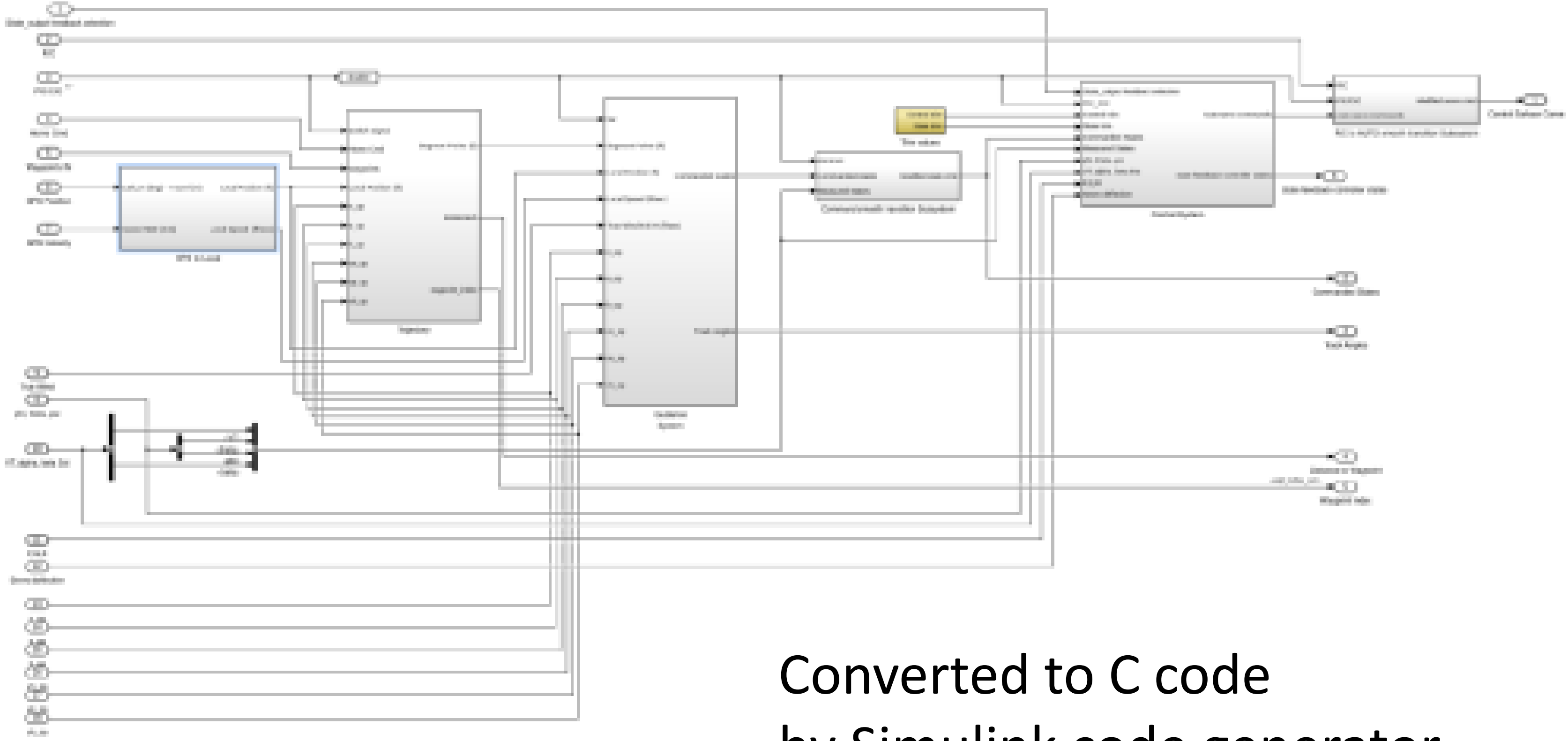


Arduino Due (AT91SAM3X8E)
- 84MHz Cortex M3
- 96KB SRAM, 512KB Flash

NVIDIA Tegra TK1 (Tegra K1)
- 2GHz 4x Cortex-A15, GPU
- 2GB DDR3L DRAM, 16GB eMMC

Simulink Model

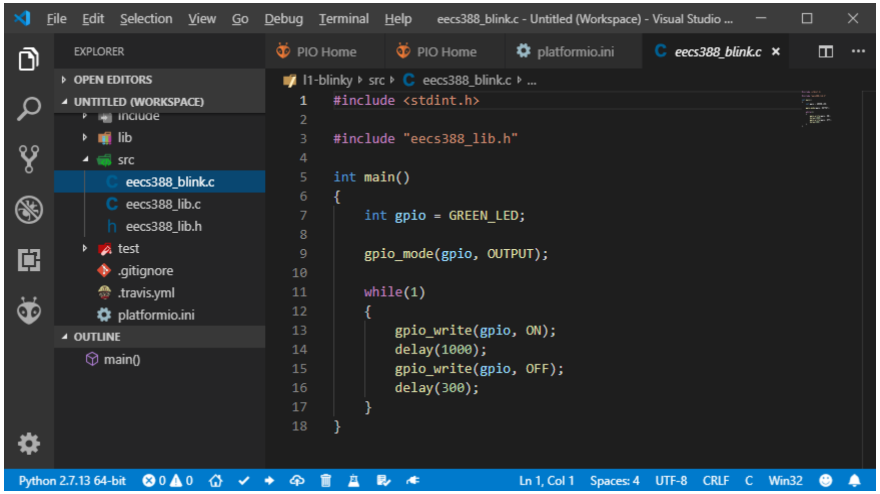
- Top-level flight control system block



Converted to C code
by Simulink code generator

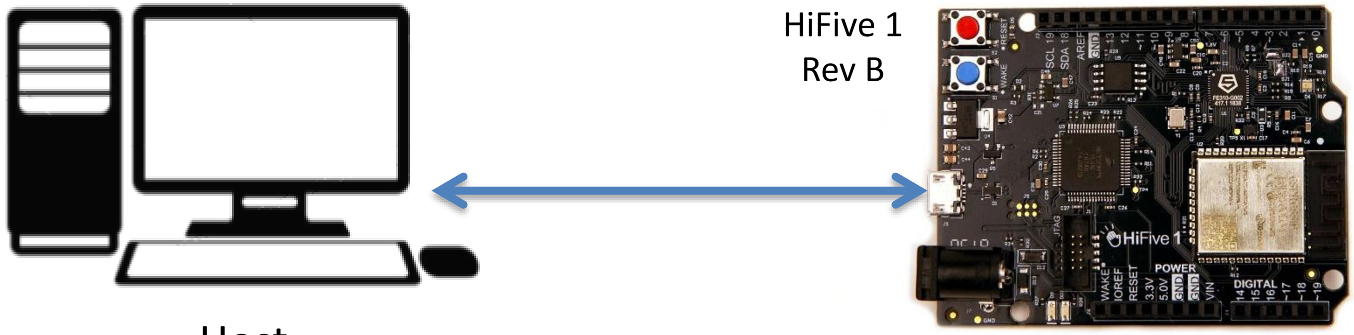
EECS388 Lab (1/2)

Visual Studio Code + PlatformIO IDE



```c
#include <stdint.h>

#include "eecs388_lib.h"

int main()
{
    int gpio = GREEN_LED;

    gpio_mode(gpio, OUTPUT);

    while(1)
    {
        gpio_write(gpio, ON);
        delay(1000);
        gpio_write(gpio, OFF);
        delay(300);
    }
}
```

- Host/target model
- C language
- Bare metal (No OS)


PC

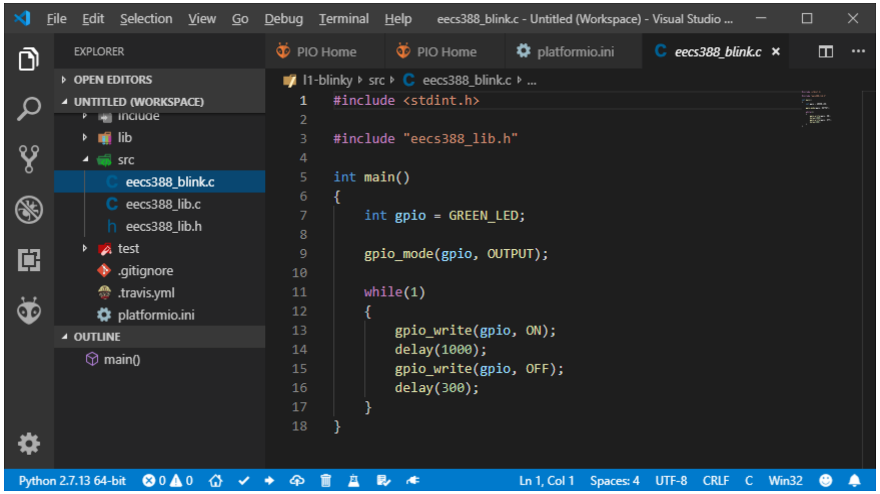
HiFive 1
Rev B

Host

Target

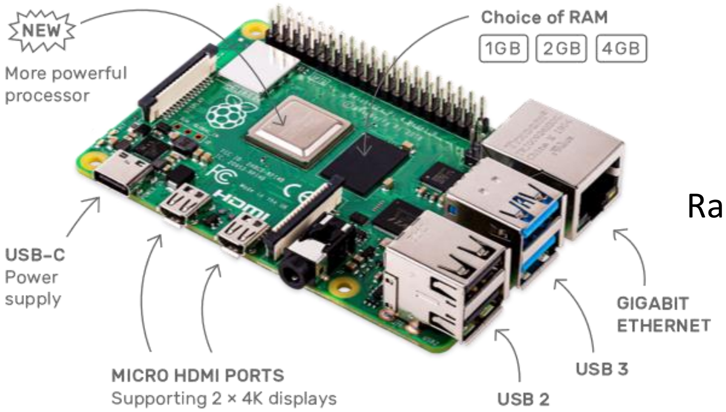
EECS388 Lab (2/2)

Visual Studio Code (or any text editor)



```c
#include <stdint.h>

#include "eecs388_lib.h"

int main()
{
    int gpio = GREEN_LED;

    gpio_mode(gpio, OUTPUT);

    while(1)
    {
        gpio_write(gpio, ON);
        delay(1000);
        gpio_write(gpio, OFF);
        delay(300);
    }
}
```

- Standalone model
- Python/C++
- Linux (Ubuntu)
- Native development too chain



Rasbperry Pi 4

Host/Target

Summary

- Embedded software development
  - Development models
    - Host/target, standalone development
  - Challenges
    - Resource constrained environment
    - low-level access to hardware, diversity, lack of standard
  - Programming languages
    - Pros and cons of C/C++, python, … in embedded systems development