



Modern C++ to Impress Your Embedded Dev Friends

STEVE BUSH



MODERN C++ TO IMPRESS YOUR EMBEDDED DEV FRIENDS

Steve Bush - Research Fellow

Digital Innovation - Smart Products

The Procter & Gamble Company



ABSTRACT

C++ is often talked about in terms of what cannot or should not be done in the context of embedded systems. In contrast, this talk is about some of the things that modern C++ idioms allow us to do better or more expressively than comparable operations in C. We will cover several patterns that allow an embedded developer to express intent clearly, ease maintenance, and encourage re-use while avoiding overheads that are costly in resource-constrained systems. Examples include initialization structures that are self-describing and IDE-friendly, small code implementations that make available sections of the C++ Standard Library for embedded use, and tools for understanding and controlling our use of limited memory resources.

ABOUT ME

- Research Fellow at Procter & Gamble
- P&G is a global consumer goods manufacturer
- Embedded product developer, C++ evangelist







OBJECTIVE

Making embedded code easier (and more pleasant) to write and maintain



AGENDA

- GPIO configuration
- Compile-Time Lookup Table Generation
- Developer-Friendly Numeric Constructs
- Lean stream-based IO
- Heap memory management for allocate-once
- Use `std::chrono` like a boss
- A `std::random` topic



COMPANION REPOSITORY

(<https://github.com/sgbush/cppcon2022>



CONFIGURE GPIO DECLARATIVELY



DECLARATIVE GPIO

The Old Way

```
1 GPIOInit_t config;
2 config.PinNumber = 1;
3 config.Mode = GPIO_MODE_PUSH_PULL;
4 config.Pull = GPIO_PULLUPDOWN_NONE;
5 config.Speed = GPIO_SPEED_MEDIUM;
6 config.AlternateFunc = GPIO_AF_6;
7
8 extern GPIO_Type* GPIOA;
9
10 GPIO_Configure(GPIOA, &config);
11 GPIO_WritePin(GPIOA, 1, GPIO_LOGIC_HIGH);
12
13 config.PinNumber = 4;
14 config.AlternateFunc = GPIO_AF_2;
15
16 GPIO_Configure(GPIOA, &config);
```



DECLARATIVE GPIO

The Old Way

```
1 GPIOInit_t config;
2 config.PinNumber = 1;
3 config.Mode = GPIO_MODE_PUSH_PULL;
4 config.Pull = GPIO_PULLUPDOWN_NONE;
5 config.Speed = GPIO_SPEED_MEDIUM;
6 config.AlternateFunc = GPIO_AF_6;
7
8 extern GPIO_Type* GPIOA;
9
10 GPIO_Configure(GPIOA, &config);
11 GPIO_WritePin(GPIOA, 1, GPIO_LOGIC_HIGH);
12
13 config.PinNumber = 4;
14 config.AlternateFunc = GPIO_AF_2;
15
16 GPIO_Configure(GPIOA, &config);
```



DECLARATIVE GPIO

The Old Way

```
1 config.PinNumber = 1;
2 config.Mode = GPIO_MODE_PUSH_PULL;
3 config.Pull = GPIO_PULLUPDOWN_NONE;
4 config.Speed = GPIO_SPEED_MEDIUM;
5 config.AlternateFunc = GPIO_AF_6;
6
7
8 extern GPIO_Type* GPIOA;
9
10 GPIO_Configure(GPIOA, &config);
11 GPIO_WritePin(GPIOA, 1, GPIO_LOGIC_HIGH);
12
13 config.PinNumber = 4;
14 config.AlternateFunc = GPIO_AF_2;
15
16 GPIO_Configure(GPIOA, &config);
17 GPIO_WritePin(GPIOA, 4, GPIO_LOGIC_LOW);
```



DECLARATIVE GPIO

The Old Way

```
1 config.PinNumber = 1;
2 config.Mode = GPIO_MODE_PUSH_PULL;
3 config.Pull = GPIO_PULLUPDOWN_NONE;
4 config.Speed = GPIO_SPEED_MEDIUM;
5 config.AlternateFunc = GPIO_AF_6;
6
7
8 extern GPIO_Type* GPIOA;
9
10 GPIO_Configure(GPIOA, &config);
11 GPIO_WritePin(GPIOA, 1, GPIO_LOGIC_HIGH);
12
13 config.PinNumber = 4;
14 config.AlternateFunc = GPIO_AF_2;
15
16 GPIO_Configure(GPIOA, &config);
17 GPIO_WritePin(GPIOA, 4, GPIO_LOGIC_LOW);
```



DECLARATIVE GPIO

Better Way: Strong types in an iterable container

```
1 using GPIODEF = \
2 struct IODefStruct {
3     GPIO_TypeDef* GPIO;
4     uint32_t PinNumber;
5     enum IOFUNCTION:uint32_t { INPUT = 0, OUTPUT = 1, ALT = 2, ANALOG = 3 } Fu
6     uint32_t AltFunc;
7     enum IOTYPE:uint32_t { NORMAL = 0, OPENDRAIN = 1 } Type;
8     enum IOSPEED:uint32_t { LOW = 0, MEDIUM = 1, HIGH = 2, VERYHIGH = 3 } Spee
9     enum IOPULL:uint32_t { NONE = 0, PULLUP = 1, PULLDOWN = 2 } Bias;
10    enum IOSTATE:uint32_t { LOGIC_LOW, LOGIC_HIGH, DONT_CARE } InitialState;
11 };
12
13 static const std::array<GPIODEF, 2> gpiodefs = { {
14     // green LED
15     {GPIOB, 0,
16      GPIOFunction::TOFUNCTION::OUTPUT |
```



DECLARATIVE GPIO

Better Way: Strong types in an iterable container

```
13 static const std::array<GPIODEF, 2> gpiodefs = { {
14     // green LED
15     {GPIOB, 0,
16      GPIODEF::IOFUNCTION::OUTPUT, 1,
17      GPIODEF::IOTYPE::NORMAL,
18      GPIODEF::IOSPEED::LOW,
19
20      GPIODEF::IOPULL::NONE,
21      GPIODEF::IOSTATE::LOGIC_LOW},
22     // red LED
23     {GPIOB, 2,
24      GPIODEF::IOFUNCTION::OUTPUT, 1,
25      GPIODEF::IOTYPE::NORMAL,
26      GPIODEF::IOSPEED::LOW,
27      GPIODEF::IOPULL::NONE,
28      GPIODEF::IOSTATE::LOGIC_LOW},
29 }
```



DECLARATIVE GPIO

Better Way: Strong types in an iterable container

```
14     // green LED
15     {GPIOB, 0,
16      GPIODEF::IOFUNCTION::OUTPUT, 1,
17      GPIODEF::IOTYPE::NORMAL,
18      GPIODEF::IOSPEED::LOW,
19
20      GPIODEF::IOPULL::NONE,
21      GPIODEF::IOSTATE::LOGIC_LOW},
22
23     // red LED
24     {GPIOB, 2,
25      GPIODEF::IOFUNCTION::OUTPUT, 1,
26      GPIODEF::IOTYPE::NORMAL,
27      GPIODEF::IOSPEED::LOW,
28      GPIODEF::IOPULL::NONE,
29      GPIODEF::IOSTATE::LOGIC_LOW},
30  } };
```



DECLARATIVE GPIO

Better Way: Strong types in an iterable container

```
14     // green LED
15     {GPIOB, 0,
16      GPIODEF::IOFUNCTION::OUTPUT, 1,
17      GPIODEF::IOTYPE::NORMAL,
18      GPIODEF::IOSPEED::LOW,
19
20      GPIODEF::IOPULL::NONE,
21      GPIODEF::IOSTATE::LOGIC_LOW},
22
23     // red LED
24     {GPIOB, 2,
25      GPIODEF::IOFUNCTION::OUTPUT, 1,
26      GPIODEF::IOTYPE::NORMAL,
27      GPIODEF::IOSPEED::LOW,
28      GPIODEF::IOPULL::NONE,
29      GPIODEF::IOSTATE::LOGIC_LOW},
30  } };
```



DECLARATIVE GPIO

- Weak enums are used because they convert implicitly to their underlying type
 - But they are constrained to tight scope to minimize errors
- Syntax is expressive - and easy to maintain
- Easy to consolidate an entire project's IO configuration in one place



DECLARATIVE GPIO

Configuring and Using IO Definitions

- Preferred: Configure all in one go
- Not Preferred: Passing around IO Defs to delegates



DECLARATIVE GPIO

Configuring - write one hardware-specific function

```
bool Configure(const GPIODEF& def)
{
    // ... chip-specific register settings ...
}
```



DECLARATIVE GPIO

Configuring: the iterators way

```
template<typename Iter>
bool Configure(Iter begin, Iter end)
{
    while ( begin != end ) Configure(begin++);
}
```



DECLARATIVE GPIO

Configuring: the ranges way

```
template<typename T> requires std::ranges::range<T>
bool Configure(const T& definitions)
{
    for (auto& def : definitions) Configure(def);
}
```



DECLARATIVE GPIO

Passing GPIO references around

```
struct SPIBus
{
    SPIBus(SPI_TypeDef* peripheral) { }
};

struct SPIProtocol
{
    enum class SPIMode { Mode1, Mode2, Mode3, Mode4 };
    SPIProtocol(SPIMode mode, uint32_t speed) { }
};

struct SPIConnection
{
    SPIConnection(const SPIBus& bus,
                  const SPIProtocol& conn,
                  const GPIO_TypeDef* pin,
                  uint32_t speed)
        : bus_(bus), conn_(conn), pin_(pin), speed_(speed) { }
};
```



DECLARATIVE GPIO

```
1 class AssertType {};
2 class AssertTypeLogicHigh
3     : public AssertType { static constexpr bool ValueWhenAsserted = true; };
4 class AssertTypeLogicLow
5     : public AssertType { static constexpr bool ValueWhenAsserted = false; };
6
7 template<typename Assert> requires std::derived_from<Assert, AssertType>
8 class GPIOAssertFunctor
9 {
10     const GPIODEF& mGPIO;
11
12     public:
13         constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) {}
14         void operator()(bool enable)
15         {
16             if (enable == Assert::ValueWhenAsserted)
17             {
18                 mGPIO.setPinValue(enable);
19             }
20         }
21     };
22 }
```



DECLARATIVE GPIO

```
1 class AssertType {};
2 class AssertTypeLogicHigh
3     : public AssertType { static constexpr bool ValueWhenAsserted = true; };
4 class AssertTypeLogicLow
5     : public AssertType { static constexpr bool ValueWhenAsserted = false; };
6
7 template<typename Assert> requires std::derived_from<Assert, AssertType>
8 class GPIOAssertFunctor
9 {
10     const GPIODEF& mGPIO;
11
12     public:
13         constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) {}
14         void operator()(bool enable)
15         {
16             if (enable == Assert::ValueWhenAsserted)
17             {
18                 mGPIO.setPinValue(enable);
19             }
20         }
21 }
```



DECLARATIVE GPIO

```
9  {
10     const GPIODEF& mGPIO;
11
12     public:
13         constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) { }
14
15     void operator()(bool enable)
16     {
17         if ( enable == Assert::ValueWhenAsserted )
18         {
19             mGPIO.GPIO->ODR |= ( 0b1 << mGPIO.PinNumber );
20         }
21         else
22         {
23             mGPIO.GPIO->ODR &= ~ ( 0b1 << mGPIO.PinNumber );
24         }
25     };
26 }
```



DECLARATIVE GPIO

```
1 template<typename Assert>
2 struct SPIConnection
3 {
4     const GPIOAssertFunctor<Assert>& mEnableFunction;
5
6     SPIConnection(const SPIBus& bus,
7                  const SPIProtocol& conn,
8                  const GPIOAssertFunctor<Assert>& enable)
9         : mEnableFunction(enable) { }
10
11    bool ReadWrite(std::span<char> outdata,
12                   std::span<char> indata,
13                   size_t length)
14    {
15        mEnableFunction(true);
16        // write to the bus
17    }
18}
```



DECLARATIVE GPIO

```
1 template<typename Assert>
2 struct SPIConnection
3 {
4     const GPIOAssertFunctor<Assert>& mEnableFunction;
5
6     SPIConnection(const SPIBus& bus,
7                  const SPIProtocol& conn,
8                  const GPIOAssertFunctor<Assert>& enable)
9         : mEnableFunction(enable) { }
10
11    bool ReadWrite(std::span<char> outdata,
12                   std::span<char> indata,
13                   size_t length)
14    {
15        mEnableFunction(true);
16        // write to the bus
17    }
18}
```



DECLARATIVE GPIO

```
1 template<typename Assert>
2 struct SPIConnection
3 {
4     const GPIOAssertFunctor<Assert>& mEnableFunction;
5
6     SPIConnection(const SPIBus& bus,
7                  const SPIProtocol& conn,
8                  const GPIOAssertFunctor<Assert>& enable)
9         : mEnableFunction(enable) { }
10
11    bool ReadWrite(std::span<char> outdata,
12                   std::span<char> indata,
13                   size_t length)
14    {
15        mEnableFunction(true);
16        // write to the bus
17    }
18}
```



DECLARATIVE GPIO

```
4   const GPIOAssertFunctor<Assert>& mEnableFunction;
```

```
5
```

```
6   SPIConnection(const SPIBus& bus,
```

```
7           const SPIProtocol& conn,
```

```
8           const GPIOAssertFunctor<Assert>& enable)
```

```
9   : mEnableFunction(enable) { }
```

```
10
```

```
11  bool ReadWrite(std::span<char> outdata,
```

```
12          std::span<char> indata,
```

```
13          size_t length)
```

```
14  {
```

```
15      mEnableFunction(true);
```

```
16      // write to the bus....
```

```
17      mEnableFunction(false);
```

```
18  }
```

```
19 };
```



DECLARATIVE GPIO

```
static const std::array<GPIODEF, 12> IOPins = { /* ... */ };
bool success = Configure(IOPins);
auto bus = SPIBus(SPI1);
auto protocol = SPIProtocol(SPIProtocol::SPIMode::Mode1, 1'000'000);
auto chipselect = GPIOAssertFunctor<AssertTypeLogicLow>(IOPins[4]);
auto connection = SPIConnection(bus, protocol, chipselect);
```



DECLARATIVE GPIO

- Centralize your IO definitions
- Abstract IO functions - with little or no cost



CREATE LOOKUP TABLES USING COMPILE-TIME EXPRESSIONS



COMPILER-DRIVEN LOOKUP TABLE GENERATION

The Old Way

```
const float TemperatureFromThermistor[] = { 25.0, 26.2, 27.5, ... };  
// ADC= 0 , 1 , 2 , ... 255 //
```

- Generate via spreadsheet and copy and paste
- Generate via script and copy and paste
- Generate via script and incorporate into build system



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Some considerations

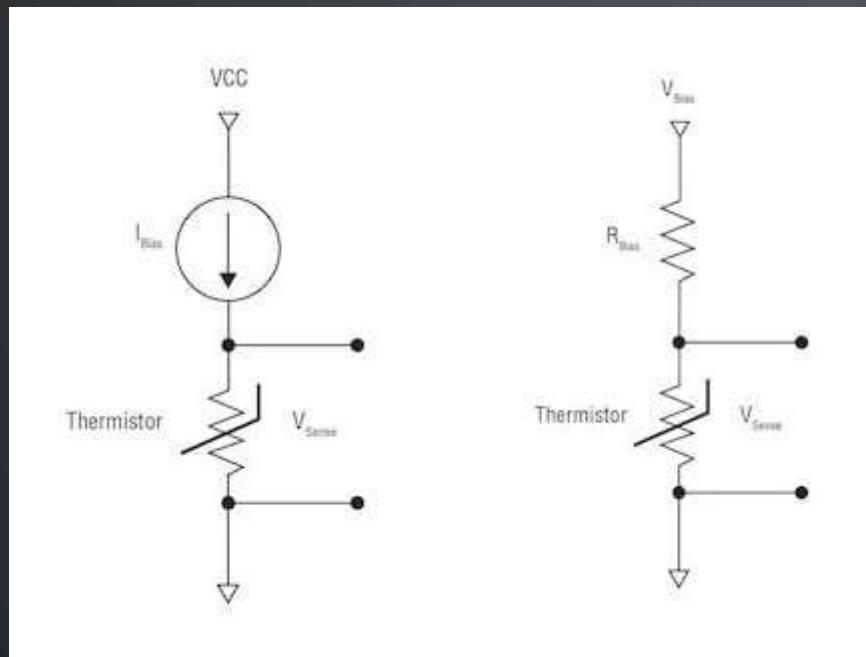
- Constants in embedded system must be carefully crafted to end up (usually) in the `.rodata` section
 - this results in constants being placed in flash - and not copied to RAM



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Some considerations

- It is common for hardware and firmware to have to evolve together
 - example: changing resistor values to tune the performance of the system



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Incorporate governing equations into the code, and let the compiler generate constant tables

- Analog Conversion

$$V_x = \text{code} \cdot \frac{V_0}{2^n}$$

- Resistor Divider

$$R = \frac{V_x R_1}{V_0 - V_x}$$

- Steinhart-Hart Model (general form)

$$\frac{1}{T} = \sum_{n=0}^{\infty} a_n \left(\ln\left(\frac{R}{R_0}\right) \right)^n$$



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
1 constexpr float VoltageFromCode(const float Vref, const size_t n, const uint16
2 {
3     float V = Vref*code/std::pow(2, n);
4
5     return V;
6 }
7
8 constexpr float ResistanceFromDivider(const float V0, const float V, const flo
9 {
10    float R = R0*V0*(V0 - V);
11
12    return R;
13 }
14
15 template<size_t N>
16 constexpr float ThermistorValue(const std::array<float, N> coefficients, const
```



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
8 constexpr float ResistanceFromDivider(const float V0, const float V, const float
9 {
10     float R = R0*V0*(V0 - V);
11
12     return R;
13 }
14
15 template<size_t N>
16 constexpr float ThermistorValue(const std::array<float,N> coefficients, const
17 {
18     float denom = 0.0;
19
20     size_t index = 0;
21     for( auto& coeff : coefficients )
22     {
23         denom += coeff*std::pow(std::log(R), index);
24         .
25         .
26     }
27 }
```



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
32 ThermistorTable(sta::array<float,N> coeff, const float R0, const float V0, con
33 {
34     std::array<float,256> table = {0};
35
36     size_t code = 0;
37     for( auto& element : table )
38     {
39         auto V = VoltageFromCode(Vref, 8, code);
40         auto R = ResistanceFromDivider(V0, V, R0);
41         element = ThermistorValue(coeff, R);
42         code += 1;
43     }
44
45     return table;
46 }
```



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
32 ThermistorTable::operator[](size_t index) const {
33     if (index >= table.size())
34         return 0;
35
36     size_t code = 0;
37     for (auto& element : table)
38     {
39         auto V = VoltageFromCode(Vref, 8, code);
40         auto R = ResistanceFromDivider(V0, V, R0);
41         element = ThermistorValue(coeff, R);
42         code += 1;
43     }
44
45     return table;
46 }
```



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
32 ThermistorTable(std::array<float,N> coeff, const float R0, const float V0, const float Vref) {
33     std::array<float,256> table = {0};
34
35     size_t code = 0;
36     for( auto& element : table ) {
37         {
38             auto V = VoltageFromCode(Vref, 8, code);
39             auto R = ResistanceFromDivider(V0, V, R0);
40             element = ThermistorValue(coeff, R);
41             code += 1;
42         }
43     }
44
45     return table;
46 }
```



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Instantiation

```
static constexpr std::array<float,4> ThermistorCoefficients { 1.0e-3, 1.0e-4, 1.0e-5, 1.0e-6 };

constexpr auto ThermistorLookup = ThermistorTable(ThermistorCoefficients, 10.0e3, 100.0e3);

int main(int , char** )
{
    for( auto element : ThermistorLookup ) std::cout << element << "\r\n";
}
```



COMPILER-DRIVEN LOOKUP TABLE GENERATION

```
1 $ readelf --sections --wide main.elf
2 Section Headers:
3 [Nr] Name           Type      Address     Off      Size   ES Flg
4 [ 0] .interp        PROGBITS 0000000000000000 000000 000000 00 A
5 [ 1] .note.gnu.property NOTE    000000000000318 000318 00001c 00
6 [ 2] .note.gnu.build-id NOTE   000000000000338 000338 000030 00
7 [ 3] .note.ABI-tag      NOTE   000000000000368 000368 000024 00
8 [ 4] .gnu.hash        GNU_HASH 00000000000038c 00038c 000020 00 A
9 [ 5] .dynsym         DYNSYM   0000000000003b0 0003b0 000028 00 A
10 [ 6] .dynstr          STRTAB  0000000000003d8 0003d8 000120 18 A
11 [ 7] .gnu.version    VERSYM   0000000000004f8 0004f8 00015f 00 A
12 [ 8] .gnu.version_r  VERNEED  000000000000658 000658 000018 02 A
13 [ 9] .rela.dyn       RELA    0000000000006d0 0006d0 000108 18 A
14 [10] .rela.plt       RELA    0000000000007d8 0007d8 000060 18 AI
15 [11] .init          PROGBITS 0000000000001000 001000 00001b 00 AY
```



COMPILER-DRIVEN LOOKUP TABLE GENERATION

```
36     W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
37     L (link order), O (extra OS processing required), G (group), T (TLS),
38     C (compressed), x (unknown), o (OS specific), E (exclude),
39     D (mbind), l (large), p (processor specific)
40
41 $ readelf --symbols --wide main.elf
42 Symbol table '.dynsym' contains 12 entries:
43     Num: Value          Size Type Bind Vis Ndx Name
44       0: 0000000000000000      0 NOTYPE LOCAL DEFAULT UND
45       1: 0000000000000000      0 FUNC   GLOBAL DEFAULT UND __libc_start_main@G
46       2: 0000000000000000      0 FUNC   GLOBAL DEFAULT UND __cxa_atexit@GLIBC_
47       3: 0000000000000000      0 FUNC   GLOBAL DEFAULT UND std::basic_ostream<
48
49       4: 0000000000000000      0 FUNC   GLOBAL DEFAULT UND std::ios_base::Init
50       5: 0000000000000000      0 FUNC   GLOBAL DEFAULT UND std::ostream& std::
51       6: 0000000000000000      0 NOTYPE WEAK  DEFAULT UND deregisterTMCl
```



COMPILER-DRIVEN LOOKUP TABLE GENERATION

82	23: 00000000000010c0	11	FUNC	GLOBAL	DEFAULT	16	main
83	24: 0000000000004000	0	OBJECT	GLOBAL	HIDDEN	25	__dso_handle
84	25: 000000000000122c	0	FUNC	GLOBAL	HIDDEN	17	_fini
85	26: 0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	__libc_start_main@G
86	27: 0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	__cxa_atexit@GLIBC_
87	28: 0000000000001140	38	FUNC	GLOBAL	DEFAULT	16	_start
88	29: 0000000000001000	0	FUNC	GLOBAL	HIDDEN	12	_init
89	30: 0000000000004008	0	OBJECT	GLOBAL	HIDDEN	25	__TMC_END__
90	31: 0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	_ZSt16__ostream_ins
91	32: 0000000000004040	272	OBJECT	GLOBAL	DEFAULT	26	_ZSt4cout@GLIBCXX_3
92	33: 0000000000004158	0	NOTYPE	GLOBAL	DEFAULT	26	_end
93	34: 0000000000004008	0	NOTYPE	GLOBAL	DEFAULT	26	__bss_start
94	35: 0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	_ZNSt8ios_base4Init
95	36: 0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	_ZNSo9_M_insertIdEE
96	37: 0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	_ITM_deregisterTMCl
97	38: 0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	__gmon_start__



COMPILER-DRIVEN LOOKUP TABLE GENERATION

84	25: 000000000000122C	0 FUNC	GLOBAL HIDDEN	UND _TINIT
85	26: 0000000000000000	0 FUNC	GLOBAL DEFAULT	UND __libc_start_main@GLIBC_
86	27: 0000000000000000	0 FUNC	GLOBAL DEFAULT	UND __cxa_atexit@GLIBC_
87	28: 0000000000001140	38 FUNC	GLOBAL DEFAULT	16 _start
88	29: 0000000000001000	0 FUNC	GLOBAL HIDDEN	12 _init
89	30: 0000000000004008	0 OBJECT	GLOBAL HIDDEN	25 __TMC_END__
90	31: 0000000000000000	0 FUNC	GLOBAL DEFAULT	UND __ZSt16__ostream_ins
91	32: 0000000000004040	272 OBJECT	GLOBAL DEFAULT	26 __ZSt4cout@GLIBCXX_3
92	33: 0000000000004158	0 NOTYPE	GLOBAL DEFAULT	26 _end
93	34: 0000000000004008	0 NOTYPE	GLOBAL DEFAULT	26 __bss_start
94	35: 0000000000000000	0 FUNC	GLOBAL DEFAULT	UND __ZNSt8ios_base4Init
95	36: 0000000000000000	0 FUNC	GLOBAL DEFAULT	UND __ZNSo9_M_insertIdEE
96	37: 0000000000000000	0 NOTYPE	WEAK DEFAULT	UND __ITM_deregisterTMCl
97	38: 0000000000000000	0 NOTYPE	WEAK DEFAULT	UND __gmon_start__
98	39: 0000000000000000	0 NOTYPE	WEAK DEFAULT	UND __ITM_registerTMClon
99	40: 0000000000000000	0 FUNC	GLOBAL DEFAULT	UND __ZNSt8ios_base4Init



COMPILER-DRIVEN LOOKUP TABLE GENERATION

- Lookup tables can be very fast or space-saving
 - Sometimes critical for embedded, real-time applications
- We can place the design support in the source code
 - No external processes to cause mistakes
 - No complexities in the build process
- Tables reside in nonvolatile memory rather than precious RAM



**CODE WITH NUMBER STRUCTURES HUMANS CAN
ACTUALLY READ AND EDIT**



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
 - 128-bit UUIDs



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
 - 128-bit UUIDs
 - 48-bit MAC address



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
 - 128-bit UUIDs
 - 48-bit MAC address
- Internet Protocol



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
 - 128-bit UUIDs
 - 48-bit MAC address
- Internet Protocol
 - 32-bit IPv4 addresses



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
 - 128-bit UUIDs
 - 48-bit MAC address
- Internet Protocol
 - 32-bit IPv4 addresses
 - 128-bit IPv6 addresses



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
 - 128-bit UUIDs
 - 48-bit MAC address
- Internet Protocol
 - 32-bit IPv4 addresses
 - 128-bit IPv6 addresses
- USB



ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
 - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
 - 128-bit UUIDs
 - 48-bit MAC address
- Internet Protocol
 - 32-bit IPv4 addresses
 - 128-bit IPv6 addresses
- USB
 - Unicode encoded string (descriptors)



ADDRESS-LIKE STRUCTURES

The Old Way

```
// canonical UUID
// { 01234567-89AB-CDEF-0123-456789ABCDEF }
static constexpr uint8_t uuid1[] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0x12, 0xF0, 0x4E, 0xA6, 0xB8, 0xD2, 0xE5, 0xF1, 0x00 }
```



ADDRESS-LIKE STRUCTURES

The Old Way

```
// canonical MAC
// 12:34:56:78:90:AB
static constexpr uint8_t mac1[] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB };
```



ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // canonical IPv4 address
2 // 192.168.0.1/24
3 static constexpr struct IP4Addr { uint8_t Address[4]; uint8_t MaskBits; } addre
4
5 // canonical IPv6 address
6 // 2001:db8::1:0
7 uint16_t IPv6Addr[] = { 0x2001, 0x0db8, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001,
```



ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // canonical IPv4 address
2 // 192.168.0.1/24
3 static constexpr struct IP4Addr { uint8_t Address[4]; uint8_t MaskBits; } addre
4
5 // canonical IPv6 address
6 // 2001:db8::1:0
7 uint16_t IPv6Addr[] = { 0x2001, 0x0db8, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001,
```



ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // Unicode string constant
2 static constexpr char devicedesc[] = { 'W', 0, 'i', 0, 'd', 0, 'g', 0, 'e', 0,
3
4 // USB string descriptor
5 #define USB_STRING 0x03
6 static constexpr char DeviceStringDescriptor[] = { 14, USB_STRING, 'W', 0, 'i',
7
8 #define CHAR_TO_UNICODE(x) x, 0x00
9 static constexpr uint8_t DeviceDescription[] = { CHAR_TO_UNICODE('M'), CHAR_TO_
```



ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // Unicode string constant
2 static constexpr char devicedesc[] = { 'W', 0, 'i', 0, 'd', 0, 'g', 0, 'e', 0,
3
4 // USB string descriptor
5 #define USB_STRING 0x03
6 static constexpr char DeviceStringDescriptor[] = { 14, USB_STRING, 'W', 0, 'i',
7
8 #define CHAR_TO_UNICODE(x) x, 0x00
9 static constexpr uint8_t DeviceDescription[] = { CHAR_TO_UNICODE('M'), CHAR_TO_
```



ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // Unicode string constant
2 static constexpr char devicedesc[] = { 'W', 0, 'i', 0, 'd', 0, 'g', 0, 'e', 0,
3
4 // USB string descriptor
5 #define USB_STRING 0x03
6 static constexpr char DeviceStringDescriptor[] = { 14, USB_STRING, 'W', 0, 'i',
7
8 #define CHAR_TO_UNICODE(x) x, 0x00
9 static constexpr uint8_t DeviceDescription[] = { CHAR_TO_UNICODE('M'), CHAR_TO_
```



ADDRESS-LIKE STRUCTURES

```
class UUID
{
    private:
        std::array<uint8_t, 16> mData {0};

    public:
        UUID() = default;

    friend constexpr UUID operator""_uuid(const char* text, size_t length);
};
```



ADDRESS-LIKE STRUCTURES

```
1 constexpr UUID operator""_uuid(const char* text, size_t length)
2 {
3     UUID result;
4     size_t index = 0;
5     size_t count = 0;
6     while ( index < length )
7     {
8         if ( ( text[index] >= '0' ) && (text[index] <= '9') )
9         {
10             result.mData[index] = text[index] - '0';
11             index += 1;
12             count += 1;
13         }
14         else if ( ( text[index] >= 'a' ) && (text[index] <= 'f') )
15         {
16             result.mData[index] = text[index] - 'a'.
```



ADDRESS-LIKE STRUCTURES

```
7      {
8          if ( ( text[index] >= '0' ) && (text[index] <= '9') )
9          {
10             result.mData[index] = text[index] - '0';
11             index == 1;
12             count += 1;
13         }
14         else if ( ( text[index] >= 'a' ) && (text[index] <= 'f') )
15         {
16             result.mData[index] = text[index] - 'a';
17             index == 1;
18             count += 1;
19         }
20         else if ( ( text[index] >= 'A' ) && (text[index] <= 'F') )
21         {
22             result.mData[count] = text[index] - 'A';
```



ADDRESS-LIKE STRUCTURES

```
20     else if ( (text[index] >= A) && (text[index] <= R) )
21     {
22         result.mData[count] = text[index] - 'A';
23         index == 1;
24         count += 1;
25     }
26 else
27 {
28     // bogus character checking here
29     index += 1;
30 }
31 }
32 if (count != 32) throw std::invalid_argument("Invalid UUID string");
33
34 return result;
35 }
```



ADDRESS-LIKE STRUCTURES

```
20     else if ( (text[index] >= 'A') && (text[index] <= 'F') )
21     {
22         result.mData[count] = text[index] - 'A';
23         index += 1;
24         count += 1;
25     }
26 else
27     {
28         // bogus character checking here
29         index += 1;
30     }
31 }
32 if (count != 32) throw std::invalid_argument("Invalid UUID string");
33
34 return result;
35 }
```



ADDRESS-LIKE STRUCTURES

```
20     else if ( (text[index] >= A) && (text[index] <= R) )
21     {
22         result.mData[count] = text[index] - 'A';
23         index == 1;
24         count += 1;
25     }
26 else
27 {
28     // bogus character checking here
29     index += 1;
30 }
31 }
32 if (count != 32) throw std::invalid_argument("Invalid UUID string");
33
34 return result;
35 }
```



ADDRESS-LIKE STRUCTURES

```
20     else if ( (text[index] >= A) && (text[index] <= R) )
21     {
22         result.mData[count] = text[index] - 'A';
23         index == 1;
24         count += 1;
25     }
26 else
27     {
28         // bogus character checking here
29         index += 1;
30     }
31 }
32 if (count != 32) throw std::invalid_argument("Invalid UUID string");
33
34 return result;
35 }
```



ADDRESS-LIKE STRUCTURES

```
static const UUID uuid = "{ 01234567-89AB-CDEF-0123-456789ABCDEF }"_uuid;
static const MACAddr mac = "12:34:56:78:90:AB"_macaddr;
static const IP4Addr addr = "192.168.0.1/24"_ip4addr;
static const USBString devdescr = "Widget"_utf16;
static const auto devdescr = u"Widget";
```



USE STREAM-BASED IO, BUT SKIP THE LIBRARY



LEAN STREAM-BASED IO

```
#include <iostream>

int main(int , char** )
{
    std::cout << "Hello, world!" << "\r\n";
}
```



LEAN STREAM-BASED IO

```
#include <iostream>

int main(int , char** )
{
    std::cout << "Hello, world!" << "\r\n";
}
```

```
$ arm-none-eabi-g++ -g0 -Os -std=c++20 \
-march=armv7e-m+fp -mfpu=fpv4-sp-d16 -mfloat-abi=hard -mtune=cortex-m4 \
-Wall -Wextra -Wpedantic -Wno-psabi \
-fno-rtti -fno-exceptions -ffunction-sections \
-specs=nosys.specs -Wl,--gc-sections \
main.cpp -o main.elf && arm-none-eabi-objcopy -O binary main.elf main.bin
```



LEAN STREAM-BASED IO

```
#include <iostream>

int main(int , char** )
{
    std::cout << "Hello, world!" << "\r\n";
}
```

```
$ arm-none-eabi-g++ -g0 -Os -std=c++20 \
-march=armv7e-m+fp -mfpu=fpv4-sp-d16 -mfloat-abi=hard -mtune=cortex-m4 \
-Wall -Wextra -Wpedantic -Wno-psabi \
-fno-rtti -fno-exceptions -ffunction-sections \
-specs=nosys.specs -Wl,--gc-sections \
main.cpp -o main.elf && arm-none-eabi-objcopy -O binary main.elf main.bin
```

```
$ du -bh main-bare.bin main-conventional.bin
1.7K    main-bare.bin
157K    main-conventional.bin
```



LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
1 namespace mcu
2 {
3     class FileStream
4     {
5         public:
6             using Radix_t = enum class RadixEnum { Binary=2, Octal=8, Decimal=10, Hexa
7
8         protected:
9             size_t mFileDescriptor;
10            Radix_t mRadixSetting;
11
12        public:
13            FileStream(size_t fd) : mFileDescriptor(fd), mRadixSetting(RadixEnum::Hexa
14
15        public:
16            FileStream& operator<<(const char* string)
```



LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
1 namespace mcu
2 {
3     class FileStream
4     {
5         public:
6             using Radix_t = enum class RadixEnum { Binary=2, Octal=8, Decimal=10, Hexa
7
8         protected:
9             size_t mFileDescriptor;
10            Radix_t mRadixSetting;
11
12         public:
13             FileStream(size_t fd) : mFileDescriptor(fd), mRadixSetting(RadixEnum::Hexa
14
15         public:
16             FileStream& operator<<(const char* string)
```



LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
7
8     protected:
9         size_t mFileDescriptor;
10        Radix_t mRadixSetting;
11
12    public:
13        FileStream(size_t fd) : mFileDescriptor(fd), mRadixSetting(RadixEnum::Hexa
14
15    public:
16        FileStream& operator<<(const char* string)
17        {
18            const size_t length = std::strlen(string);
19
20            _write(mFileDescriptor, string, length);
21
22            return *this;
23        }
```



LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
15     public:  
16         FileStream& operator<<(const char* string)  
17     {  
18         const size_t length = std::strlen(string);  
19         _write(mFileDescriptor, string, length);  
20     }  
21     return *this;  
22 }  
23  
24 template<typename U> requires std::integral<U>  
25 FileStream& operator<<(const U value) { ... }  
26  
27 template<typename U> requires std::floating_point<U>  
28 FileStream& operator<<(const U value) { ... }  
29 }
```



LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
15     public:  
16         FileStream& operator<<(const char* string)  
17     {  
18         const size_t length = std::strlen(string);  
19         _write(mFileDescriptor, string, length);  
20     }  
21     return *this;  
22 }  
23  
24 template<typename U> requires std::integral<U>  
25 FileStream& operator<<(const U value) { ... }  
26  
27 template<typename U> requires std::floating_point<U>  
28 FileStream& operator<<(const U value) { ... }  
29 }
```



LEAN STREAM-BASED IO

Instantiate Filestreams for Each Output

```
namespace mcu
{
    FileStream debug(1);
    FileStream swo(2);
}
```



LEAN STREAM-BASED IO

Some Plumbing for Output Modes

```
extern "C" int _write(int fd, const void* data, size_t length)
{
    switch (fd)
    {
        case 1:
            SWOWrite((const char*)data, length);
            break;
        case 2:
            UARTWrite((const char*)data, length);
            break;
        case 3:
            UDPWrite(connection, (const char*)data, length)
            ...
    }
}
```



LEAN STREAM-BASED IO

The Traditional Use Case

```
#include "fstream.hpp"

int main(int , char** )
{
    mcu::swo << "Hello, Serial Wire Debug!" << "\r\n";
    mcu::debug << "Hello, UART Debug!" << "\r\n";
}
```



LEAN STREAM-BASED IO

It's Pretty Small

```
#include "fstream.hpp"

int main(int , char** )
{
    mcu::swo << "Hello, Serial Wire Debug!" << "\r\n";
    mcu::debug << "Hello, UART Debug!" << "\r\n";
}
```



LEAN STREAM-BASED IO

It's Pretty Small

```
#include "fstream.hpp"

int main(int , char** )
{
    mcu::swo << "Hello, Serial Wire Debug!" << "\r\n";
    mcu::debug << "Hello, UART Debug!" << "\r\n";
}
```

```
$ du -bh *.bin
1.7K    main-bare.bin
2.3K    main-better.bin
157K   main-conventional.bin
```



LEAN STREAM-BASED IO

Add Some Bells and Whistles

```
void TroublesomeFunction()
{
    mcu::debug << mcu::FileStream::RadixEnum::Hexadecimal << 32 << "\r\n";
}
```



LEAN STREAM-BASED IO

Add Some Bells and Whistles

```
void TroublesomeFunction()
{
    mcu::debug << mcu::FileStream::RadixEnum::Hexadecimal << 32 << "\r\n";
}
```

0x20



LEAN STREAM-BASED IO

Extending to Other Types

```
1 template<typename T>
2 class Tensor { ... };
3
4 FileStream& operator<<(FileStream& stream, const Tensor<float>& tensor)
5 {
6     for(size_t i=0; i < tensor.Dimension(1); i++)
7     {
8         for(size_t j=0; j < tensor.Dimension(0); j++)
9             // ...
10            stream << tensor.Element(i,j);
11        // ...
12    }
13 }
```



LEAN STREAM-BASED IO

Extending to Other Types

```
1 template<typename T>
2 class Tensor { ... };
3
4 FileStream& operator<<(FileStream& stream, const Tensor<float>& tensor)
5 {
6     for(size_t i=0; i < tensor.Dimension(1); i++)
7     {
8         for(size_t j=0; j < tensor.Dimension(0); j++)
9             // ...
10            stream << tensor.Element(i,j);
11        // ...
12    }
13 }
```



LEAN STREAM-BASED IO

Extending to Other Types

```
Tensor<float> result( {10,10} );
mcu::debug << result;
```



LEAN STREAM-BASED IO

Extending to Other Types

```
Tensor<float> result( {10,10} );
mcu::debug << result;
```

```
{ {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, } }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
```



DARE TO USE THE HEAP



DARE TO USE THE HEAP

(WITH SOME CAVEATS)



**DARE TO USE THE HEAP
(WITH SOME CAVEATS)
(IN NON-SAFETY-RELATED APPLICATIONS)**



USING HEAP IN EMBEDDED APPLICATIONS



USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion



USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion
- Long runtimes can cause (unsolvable) fragmentation



USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion
- Long runtimes can cause (unsolvable) fragmentation
- Heap errors have no graceful resolution



USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion
- Long runtimes can cause (unsolvable) fragmentation
- Heap errors have no graceful resolution
- Many applications stick to static allocation



USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion
- Long runtimes can cause (unsolvable) fragmentation
- Heap errors have no graceful resolution
- Many applications stick to static allocation
- Off Limits: std::vector std::map std::list std::deque



USING HEAP IN EMBEDDED APPLICATIONS

Use in allocate-once scenarios

```
class CalculationNode
{
public:
    using NodeHandle = std::unique_ptr<CalculationNode>;

private:
    size_t mNodeNumber;
    std::vector<NodeHandle> mNodeCollection;

public:
    virtual bool Calculate()
    {
        for ( node : mNodeCollection ) node->Calculate();
    }
};
```



USE ARENA ALLOCATORS



USE ARENA ALLOCATORS

- For monotonic, allocate-once applications
 - Fast
 - Low overhead



USE ARENA ALLOCATORS

- For monotonic, allocate-once applications
 - Fast
 - Low overhead
- Preferably deterministic memory usage



USE ARENA ALLOCATORS

- For monotonic, allocate-once applications
 - Fast
 - Low overhead
- Preferably deterministic memory usage
- See also Lakos (2017) and Steagall (2017)



USE ARENA ALLOCATORS

Overriding the global new operator

```
void* operator new(size_t size)
{
    static constexpr size_t ArenaSize = 1'000;
    static char Arena[ArenaSize];

    auto ptr = Arena + AllocatedBytes;
    AllocatedBytes += size;
    return ptr;
}
```



USE ARENA ALLOCATORS

Overriding new on a per-class basis

```
1 template<typename T>
2 class ArenaAllocator
3 {
4     static constexpr size_t ArenaSize = 1'000;
5     static char Arena[ArenaSize];
6
7     public:
8         T* allocate(size_t size) { /* ... */ }
9     };
10
11 class CalculationNode
12 {
13     public:
14         using NodeHandle = std::unique_ptr<CalculationNode>;
15
16     private:
```



USE ARENA ALLOCATORS

Overriding new on a per-class basis

```
10
11 class CalculationNode
12 {
13     public:
14     using NodeHandle = std::unique_ptr<CalculationNode>;
15
16     private:
17     std::vector<NodeHandle, ArenaAllocator<NodeHandle>> mCollectionOfNodes;
18
19     void* operator new(size_t size)
20     {
21         ArenaAllocator<char> alloc;
22         return alloc.allocate(size);
23     }
24 };
```



USE ARENA ALLOCATORS

Demo Time



USE ARENA ALLOCATORS

It is possible to do runtime allocation while avoiding pitfalls of fragmentation, and to avoid heap overhead.

Be safe.



IMPLEMENT ONE FUNCTION TO UNLOCK ALL OF TIME



IMPLEMENT ONE FUNCTION TO UNLOCK ALL OF TIME (FUNCTIONS)



UNLOCK STD::CHRONO

- Embedded time sources
 - Relative (Hardware timer)
 - Absolute (RTC)
- Applications
 - Performance instrumentation
 - Time-of-day and calendaring



UNLOCK STD::CHRONO

```
1 static TIM_TypeDef* ChronoHWTimer = nullptr;
2
3 bool MicrosecondClockConfigure(TIM_TypeDef* timer, volatile uint32_t& rccreg,
4 {
5     bool success = true;
6     const uint32_t timer_freq = 1'000'000;
7
8     // program timer for std::chrono functions
9     ChronoHWTimer = timer;
10    rccreg |= rccvalue;
11    timer->PSC = (ClockTree.pclk2/timer_freq) - 1;
12    timer->ARR = 0xFFFFFFFF;
13    timer->CNT = 0ul;
14
15    return success;
16 }
```



UNLOCK STD::CHRONO

```
1 static TIM_TypeDef* ChronoHWTimer = nullptr;
2
3 bool MicrosecondClockConfigure(TIM_TypeDef* timer, volatile uint32_t& rccreg,
4 {
5     bool success = true;
6     const uint32_t timer_freq = 1'000'000;
7
8     // program timer for std::chrono functions
9     ChronoHWTimer = timer;
10    rccreg |= rccvalue;
11    timer->PSC = (ClockTree.pclk2/timer_freq) - 1;
12    timer->ARR = 0xFFFFFFFF;
13    timer->CNT = 0ul;
14
15    return success;
16 }
```



UNLOCK STD::CHRONO

```
1 static TIM_TypeDef* ChronoHWTimer = nullptr;
2
3 bool MicrosecondClockConfigure(TIM_TypeDef* timer, volatile uint32_t& rccreg,
4 {
5     bool success = true;
6     const uint32_t timer_freq = 1'000'000;
7
8     // program timer for std::chrono functions
9     ChronoHWTimer = timer;
10    rccreg |= rccvalue;
11    timer->PSC = (ClockTree.pclk2/timer_freq) - 1;
12    timer->ARR = 0xFFFFFFFF;
13    timer->CNT = 0UL;
14
15    return success;
16 }
```



UNLOCK STD::CHRONO

```
namespace std::chrono
{
    time_point<high_resolution_clock> high_resolution_clock::now()
    {
        return time_point(
            nanoseconds(
                1000ull*static_cast<nanoseconds::rep>(chrono_timer->CNT) ) );
    }
} // namespace std::chrono
```



UNLOCK STD::CHRONO

Demo time

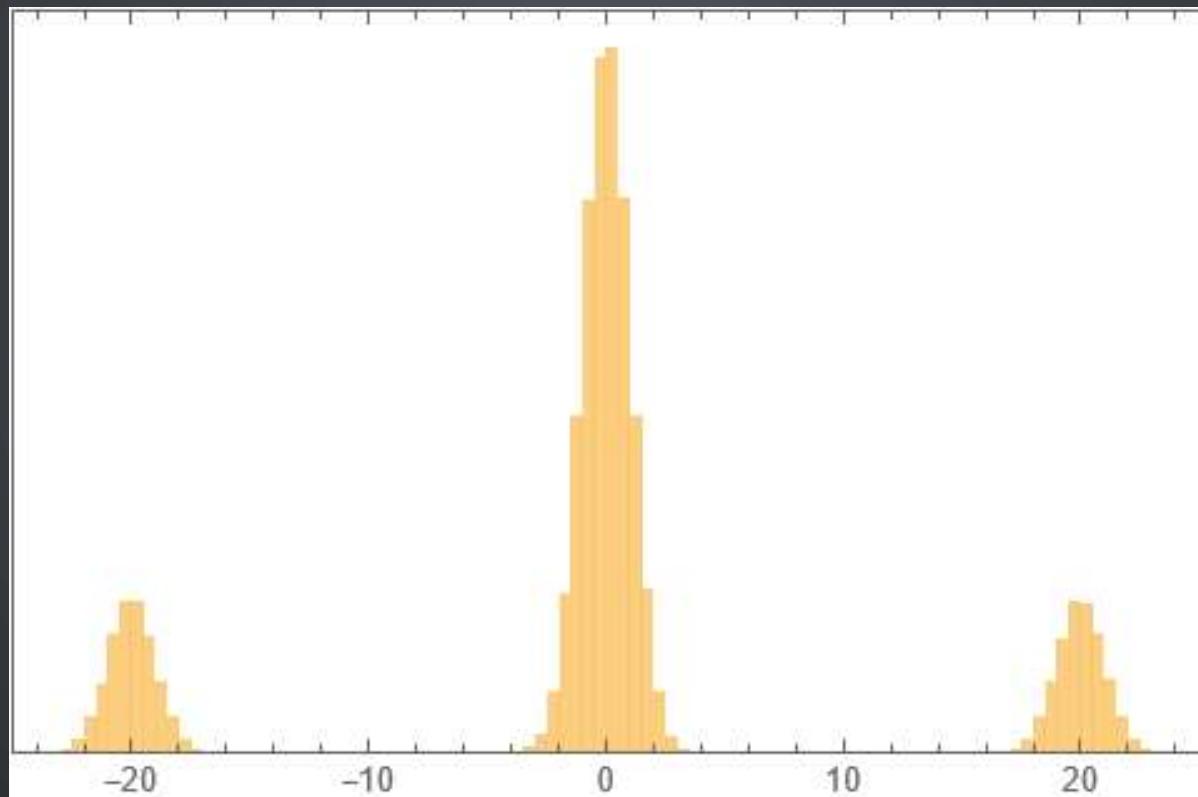


Implement `std::random_device`
and Get Free Beer Library Code



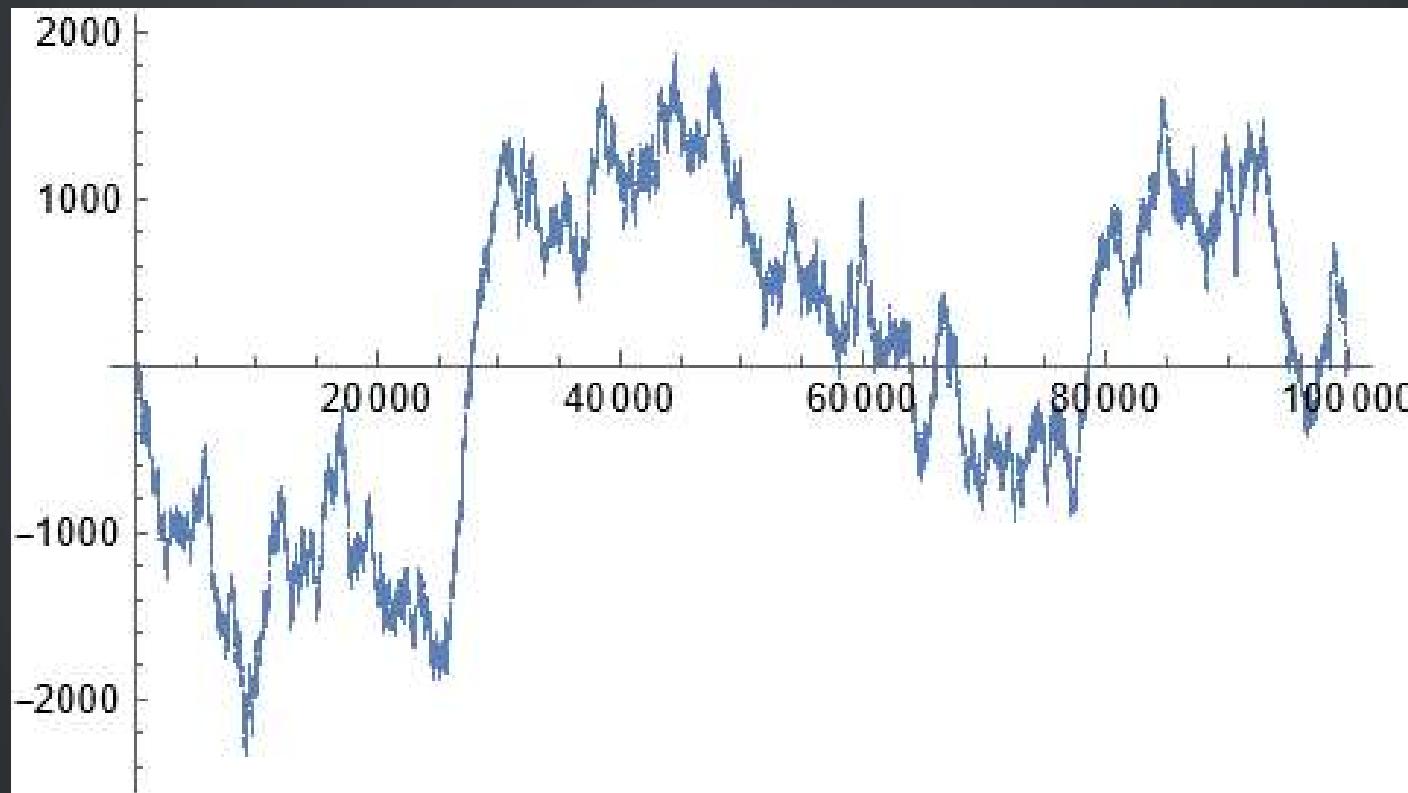
SCENARIO

- You need a random number generator with a multimodal probability distribution



SCENARIO

- This gives a random walk with occasional large changes





FUN WITH RANDOM NUMBERS

```
class BimodalDistribution
{
public:
    using ModeParamsType = std::normal_distribution<float>::param_type;

private:
    std::normal_distribution<float> mFirstModeDistribution;
    std::normal_distribution<float> mSecondModeDistribution;
    std::uniform_real_distribution<float> mWeightingDistribution;
    float mModelWeight;
```



FUN WITH RANDOM NUMBERS

```
class BimodalDistrubution
{
    ...
    template<typename Generator>
    float operator()(Generator& device)
    {
        auto x = mWeightingDistribution(device);
        float result;

        if ( std::abs(x) < mMode1Weight )
        {
            result = mFirstModeDistribution(device);
        }
        else
        {
```

... (remaining code is cut off)



FUN WITH RANDOM NUMBERS

```
auto dev = std::random_device();
auto dist = BimodalDistribution();
float x = dist(dev);
```



FUN WITH RANDOM NUMBERS

```
1 namespace mcu
2 {
3     class random_device
4 {
5     public:
6         using result_type = unsigned int;
7         static constexpr result_type min() { return std::numeric_limits<result_type>::min(); }
8         static constexpr result_type max() { return std::numeric_limits<result_type>::max(); }
9         constexpr double entropy() const noexcept { return 32.0; }
10
11     random_device()
12     {
13         RCC->AHB2ENR |= RCC_AHB2ENR_RNGEN;
14         RNG->CR |= RNG_CR_RNGEN;
15     }
16     random_device::operator=(const random_device& other) = delete;
```



FUN WITH RANDOM NUMBERS

```
○     static constexpr result_type max() { return std::numeric_limits<result_type>::max(); }
9     constexpr double entropy() const noexcept { return 32.0; }
10
11    random_device()
12    {
13        RCC->AHB2ENR |= RCC_AHB2ENR_RNGEN;
14        RNG->CR |= RNG_CR_RNGEN;
15    }
16    random_device& operator=(const random_device& other) = delete;
17    result_type operator()()
18    {
19        while ( !(RNG->SR & RNG_SR_DRDY) ) continue;
20        return RNG->DR;
21    }
22};
23}
```



FUN WITH RANDOM NUMBERS

```
○ static constexpr result_type max() { return std::numeric_limits<result_type>::max(); }
9 constexpr double entropy() const noexcept { return 32.0; }
10
11 random_device()
12 {
13     RCC->AHB2ENR |= RCC_AHB2ENR_RNGEN;
14     RNG->CR |= RNG_CR_RNGEN;
15 }
16 random_device& operator=(const random_device& other) = delete;
17 result_type operator()()
18 {
19     while ( !(RNG->SR & RNG_SR_DRDY) ) continue;
20     return RNG->DR;
21 }
22 };
23 }
```



FUN WITH RANDOM NUMBERS

```
auto dev = mcu::random_device();
auto dist = BimodalDistribution();
float x = dist(dev);
```



FUN WITH RANDOM NUMBERS

Demo Time



THANK YOU!

