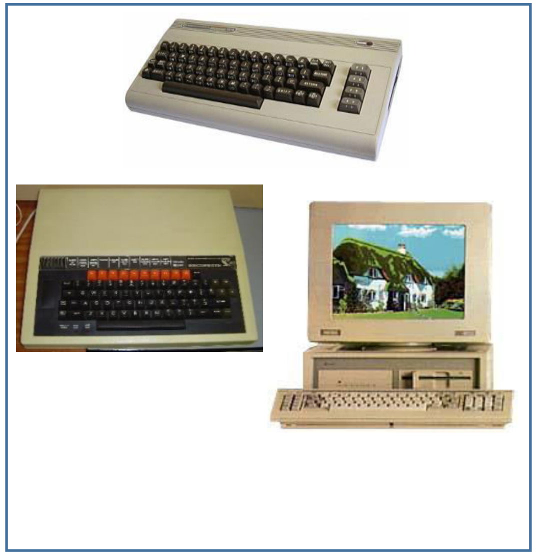
Code Smells

Improving Sense of Smell for Low-level Debugging

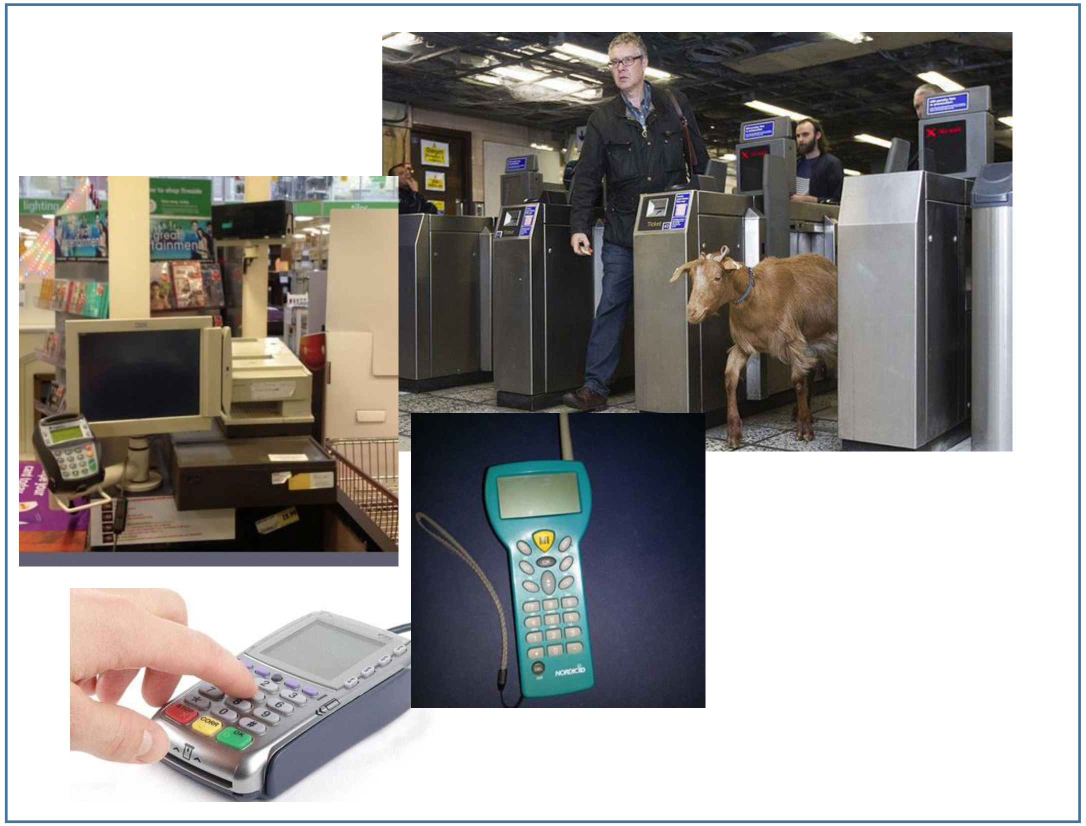
Session Goals

- Raise awareness of the pros and cons of low level debugging techniques
- Discuss metaphor of smell
- Talk about a few smells
- Try some mob programming out!

Background



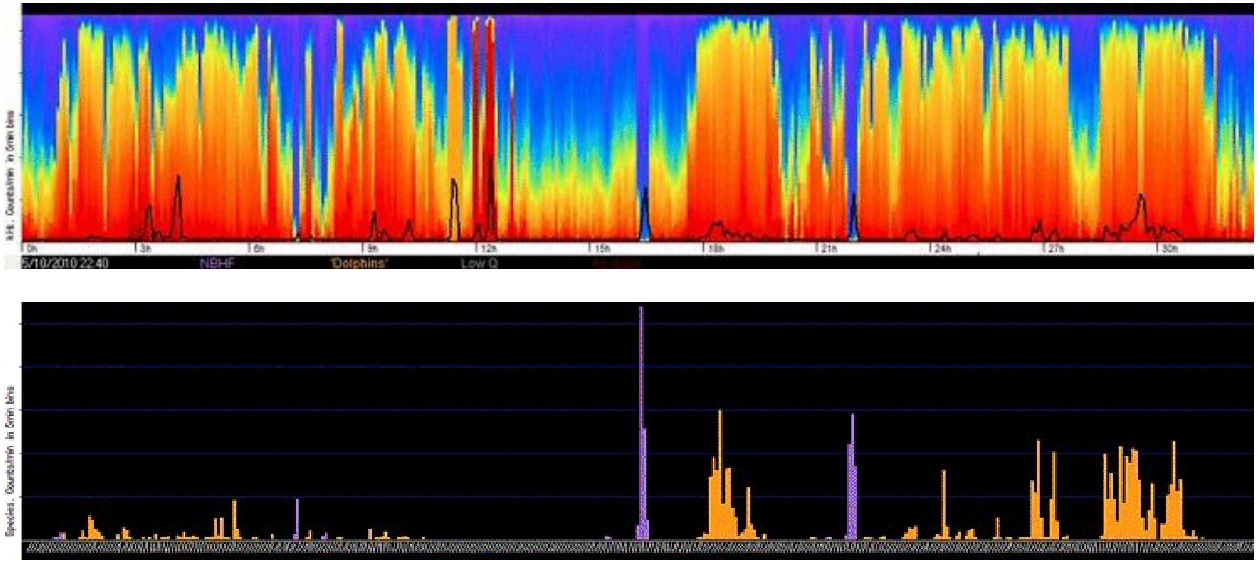
Sun
microsystems



18.2
ZTC-505 Digital Master Controller
09:51
Glucose
12.4
mmol/L
22 Oct 2015 - 11:11

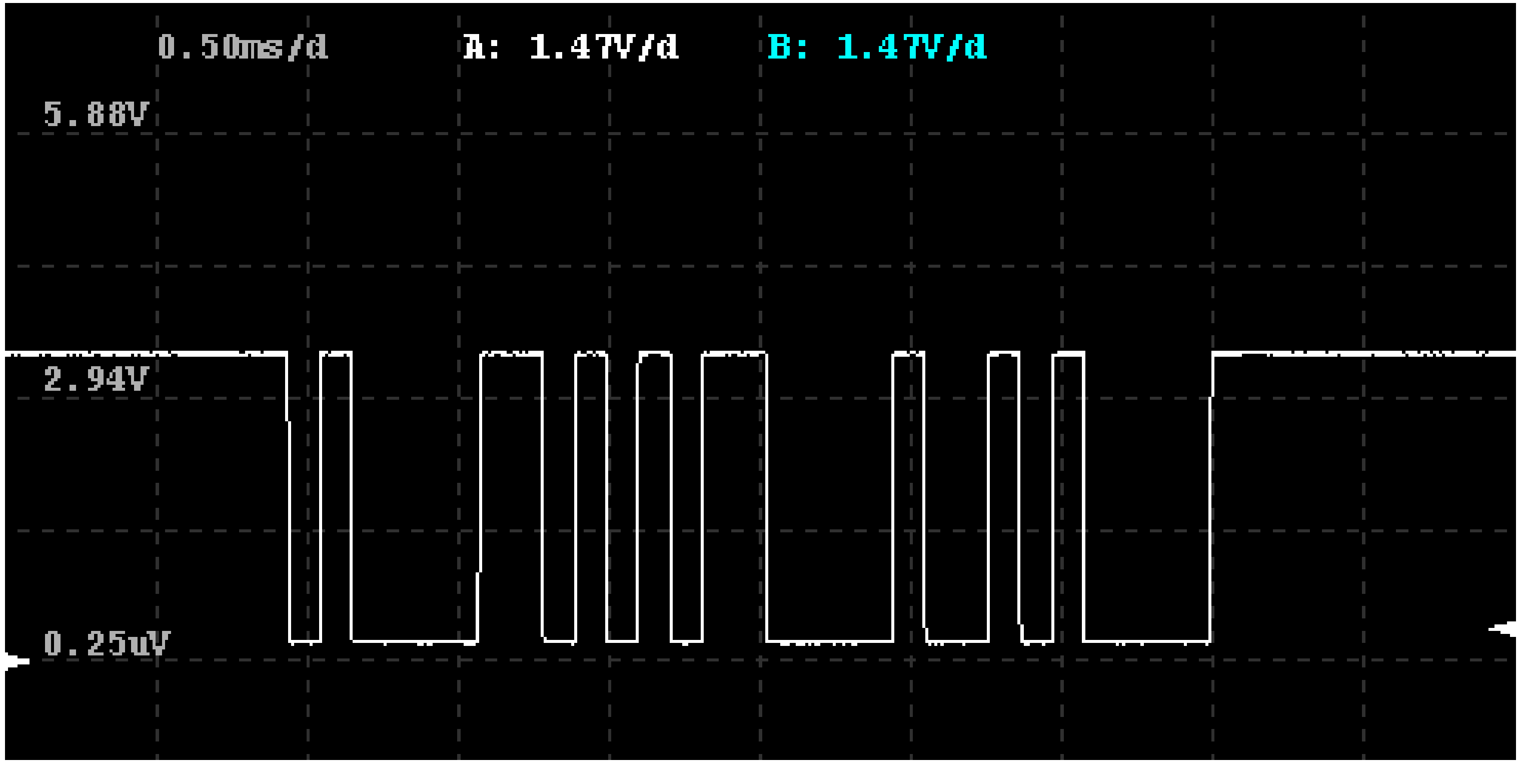
What is low level?

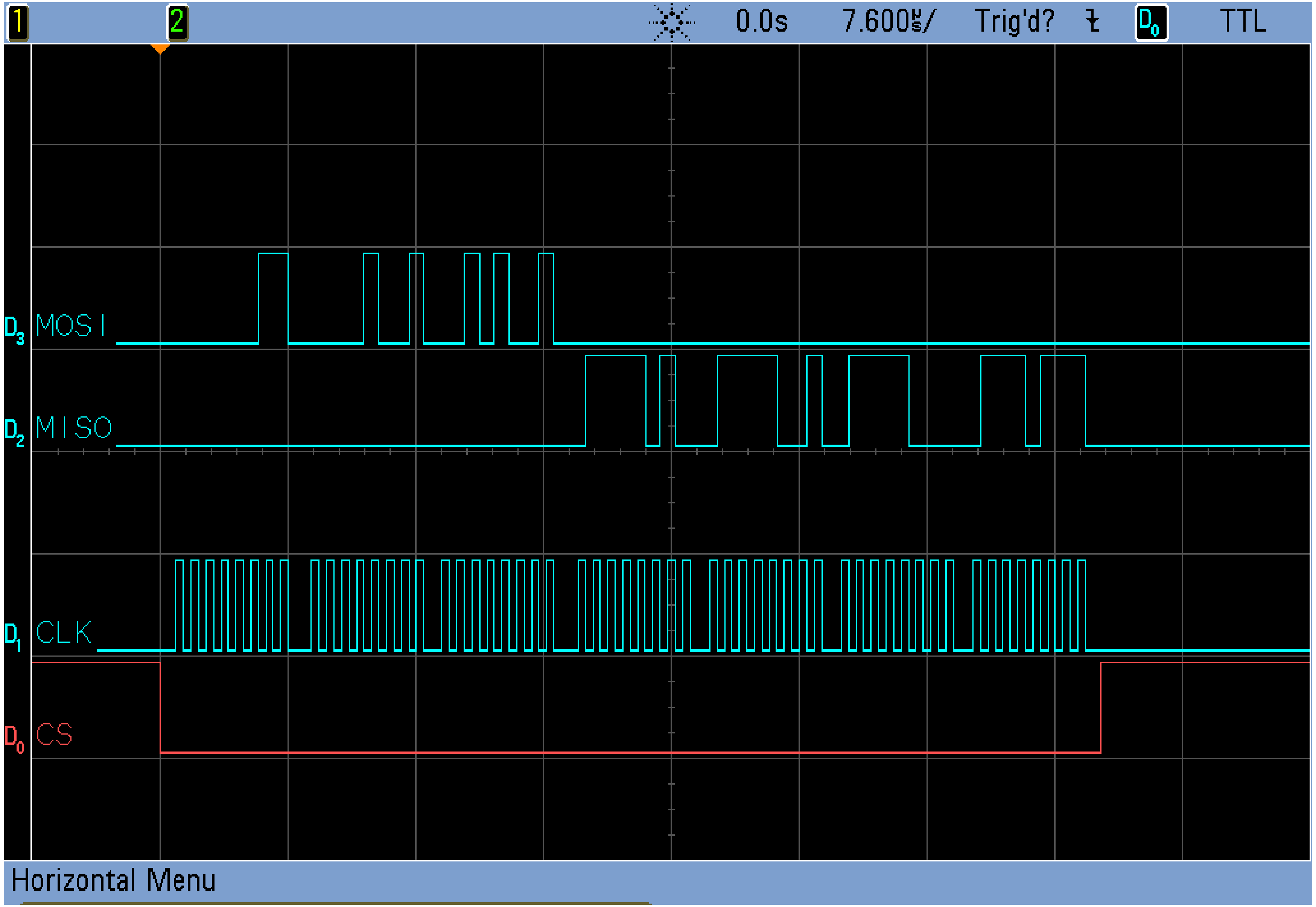
- Proximity to “the metal” (memory, processor, any devices)
- Protocols and stacks (USB, TCP etc.)
- When every moment matters



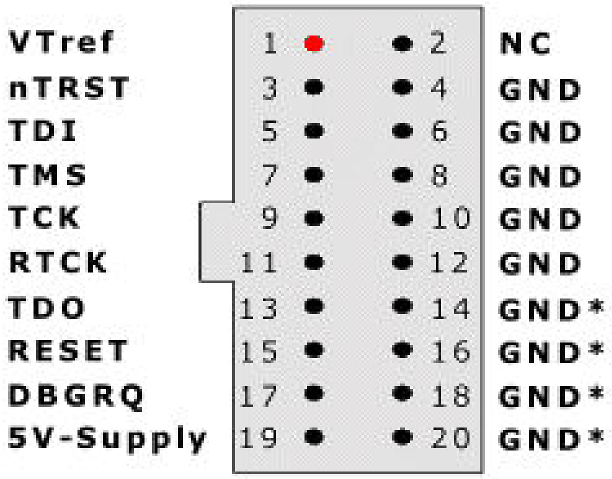
C++…??

~~stdout~~


0.50ms/d
A: 1.47V/d
B: 1.47V/d
5.88V
2.94V
0.25uV

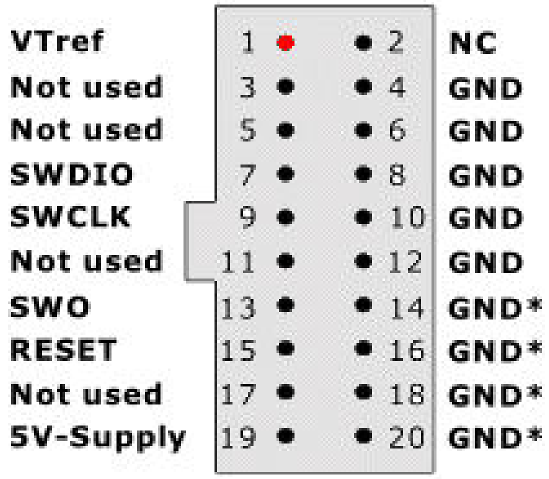

1
2
0.0s
7.600µs/
Trig'd?
D0
TTL
D3 MOSI
D2 MISO
D1 CLK
D0 CS
Horizontal Menu

Pinout for JTAG



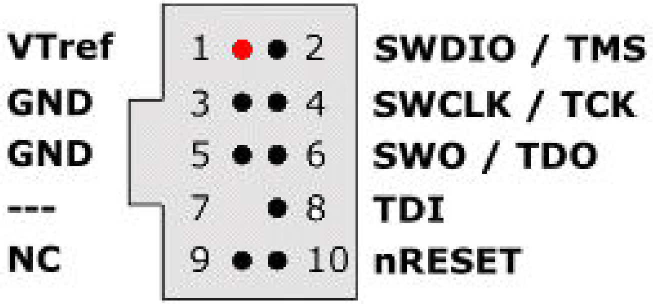
VTref	1	2	NC
nTRST	3	4	GND
TDI	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
RTCK	11	12	GND
TDO	13	14	GND*
RESET	15	16	GND*
DBGRQ	17	18	GND*
5V-Supply	19	20	GND*

Pinout for SWD



VTref	1	2	NC
Not used	3	4	GND
Not used	5	6	GND
SWDIO	7	8	GND
SWCLK	9	10	GND
Not used	11	12	GND
SWO	13	14	GND*
RESET	15	16	GND*
Not used	17	18	GND*
5V-Supply	19	20	GND*

9-pin JTAG/SWD



VTref	1	2	SWDIO / TMS
GND	3	4	SWCLK / TCK
GND	5	6	SWO / TDO
---	7	8	TDI
NC	9	10	nRESET

Used by x86, ARM, PIC, AVR, PowerPC, MIPS

The joy of TDD

- Dependency injection
- C++ has arrived on micro-controllers!
- It’s almost as easy with C anyway…
- Static mocking
- Multi-threaded testing

The joy of TDD

- Low level device driver development requires a lot of experimentation… be patient!
- Clients often find it hard to understand why you won’t use chip manufacturer libraries (despite all their code being untested trash)
- Tight code memory might preclude running tests on target completely

The joy of TDD

- Build server integration with BDD is tricky when circuit boards are involved ☺
- Protocol stacks that (almost) no-one will let you rewrite (TCP/IP, USB, radio comms)… you have to wrap them really
- Multi-threaded testing will never represent reality on the device

let's talk about trace baby…

CPU/MCU trace

Pros:

- Fast, with minimal interference
- Easy to disable/enable without code
- Can get trace output without a UART

Cons:

- Usually tricky to set up
- Can be flakey (or require expensive MCU and/or debugger)
- Might not be properly supported by your H/W design… or desktop O/S

let's talk about trace baby…

Serial / external

Pros:

- Human readable output can be read on a terminal so you can copy/paste and watch it easily
- Pretty simple to get up and running

Cons:

- Requires a UART peripheral (which means USB on a PC these days…)
- High level of interference with executing code and/or hardware
- Slow

let's talk about trace baby...

RAM

Pros:

- Doesn't require anything except a debugger, but can optionally make use of serial dumps
- Very flexible
- Low level of interference with executing code

Cons:

- Tough to actually use if you don't have serial output of some kind
- Needs a reliable debugging session
- If a lot of data is involved, it could be tricky to store it all!

let's talk about trace baby...

Oscilloscope

Pros:

- Doesn't even require a debugging session
- As real-time as it gets!!
- Least interference with executing code (still can cause problems though…)

Cons:

- Limited data storage
- You need a scope
- May require debugging pins (and instrumentation to go with them…)

#habitability

Example 1: Delay (C)

```c
static void _Delay(uint16_t microseconds)
{
    uint16_t i;

    for (i = 0u; i < microseconds; i++)
    {
        /* Delay 10usec */
        CLOCK_DELAY_US(10u);
        WATCHDOG_RESET();
    }
}
```

Example 2: Atmel SPI lock (C)

```
uint32_t QSPID_IsBusy(volatile uint8_t *QspiSemaphore)
{
        if (Is_LockFree(QspiSemaphore))
                return 1;
        else
                return 0;
}
```

Example 3: Waltzing (C)

```
case GET_SECTOR_COUNT :       /* Get number of sectors on the disk (DWORD) */
  if ((send_cmd(CMD9, 0) == 0) && rcvr_datablock(csd, 16)) {
    if ((csd[0] >> 6) == 1) { /* SDC version 2.00 */
      csize = csd[9] + ((WORD)csd[8] << 8) + 1;
      *(DWORD*)buff = (DWORD)csize << 10;
    } else {                                                /* SDC version 1.XX or MMC*/
      n = (csd[5] & 15) + ((csd[10] & 128) >> 7) + ((csd[9] & 3) << 1) + 2;
      csize = (csd[8] >> 6) + ((WORD)csd[7] << 2) + ((WORD)(csd[6] & 3) << 10) + 1;
      *(DWORD*)buff = (DWORD)csize << (n - 9);
    }
    res = RES_OK;
  }
  break;
```

A helpful analogy

- No cat flap
- Put them out after dinner or whenever they ask.
- They have litter trays...
- Go out for a few hours, come back, smells like crap in the house

Why is that?

How can I notice the bad smells?

- Regularly breathe fresh air
- Check the litter
- Have visitors

Another helpful analogy: nose training

Example 4: Ring around the roses (C)

```c
bool RingBuffer_Write(RingBuffer *ringBuffer, uint8_t item)
{
  uint32_t size = ringBuffer->size;

  if (ringBuffer->count == size)
    return false;

  uint32_t end = ringBuffer->end;
  ringBuffer->items[end] = item;

  ringBuffer->end = (ringBuffer->end + 1) % size;

  ringBuffer->count++;
  return true;
}
```

Example 5: Creating a thread (C++0x)

```
void Core::CreateThread(void *stackBuffer,
                        uint32_t stackSize,
                        void (*entryPoint) (),
                        uint8_t priority)
{
   Critical_DisallowInterrupts();
   KeyaSmartAssert(mCreatedThreadCount < Config::MaxNumberOfThreads);

   uint8_t nextSlot = 0U;
   for (uint32_t i = 0; i < Config::MaxNumberOfThreads; i++) {
      if (threadPool[i].Status == ThreadStatus_Free) {
         nextSlot = i;
         break;
      }
   }

   KeyaSmartAssert(nextSlot < Config::MaxNumberOfThreads);

   // Good to go
   threadPool[nextSlot].Status = ThreadStatus_Active;
   threadPool[nextSlot].StackBottom = stackBuffer;
   threadPool[nextSlot].SavedStackPointer =
      Context_InitialiseStackFrames(stackBuffer, stackSize, entryPoint, threadShutdown);

   threadPool[nextSlot].EntryPoint = entryPoint;
   threadPool[nextSlot].Priority = priority;
   mCreatedThreadCount++;
   Critical_AllowInterrupts();
}
```

Is this unit testable?

Example 5b: Creating a thread test

```
TEST_F(TestScheduler, First_thread_is_not_free_after_it_is_created)
{
   uint8_t mockStack[TestScheduler::MockStackSize];

   // Given
   // .. Scheduler is initialised

   // When
   Scheduler::Core::CreateThread(mockStack, sizeof(mockStack), ArbitraryThreadMain);


   // Then... all are free except index returned
   for (uint8_t i = IndexOfFirstUserThread; i < Config::MaxNumberOfThreads; i++) {
      ThreadStatus threadStatus = Scheduler::Core::GetThreadStatus(i);


      if (i == IndexOfFirstUserThread) {
         ASSERT_EQ(ThreadStatus_Active, threadStatus);
      } else {
         ASSERT_EQ(ThreadStatus_Free, threadStatus);
      }
   }

   // Then... that thread was created inside a critical section,
   // plus another for the timer and kernel threads
   ASSERT_TRUE(AssertCriticalSectionWasEnteredAndLeftNTimes(3));
}
```

Example 5b: Kernel thread switch

```
TEST_F(TestScheduler,
       Start_also_causes_an_initial_context_switch_into_the_kernel_thread)
{
   // Given
   // At least one user thread is needed to stop the Start() method from asserting out.
   uint8_t mockStack[TestScheduler::MockStackSize];
   Scheduler::Core::CreateThread(mockStack, sizeof(mockStack), ArbitraryThreadMain);

   // When
   Scheduler::Core::Start();

   // Then
   ASSERT_EQ(mockContext.SwitchToKernelCalls, 1U);
   ASSERT_TRUE(MockContext_IsKernelThreadCurrentThread());
}
```

~~#habitability~~
#goodsmells
#badsmells

A typical embedded setup

arm-none-eabi toolchain (limited to C++0x)

GDB

OpenOCD

.

ST-Link

ARM STM32 (Cortex M4 core)

The Story

- Client asks us to port some code and extend it
- Some unit tests
- Try it out on the new target, with our scheduler, and we’re having problems

Mob Programming Session Rulez

1. Driver focuses on driving
2. Navigator pools ideas of group and tells driver what to do
3. Rotate every 5 minutes
4. Focus on the task in hand