Technolution
An Open Source Board for NuttX
Dave Marples
• Why we need a development setup
• Desirable requirements
• Actions to address those requirements
  – Hardware
  – Software
  – Debug and Systems Support
• Examples of the tools we can use
• Next steps
• Lead times on product development are becoming **too compressed** to allow development from scratch
• Hardware and software are **too complex** for a single product development to carry all of the risk
• We want to **retain** and **reuse** what we know from one development to the next

• Think of it like part-baked bread... all the ingredients are there and mixed, there's just the final cooking to be done.
• **Hardware:** Find wide families of devices that have good abilities and common characteristics. Build prototypes, encourage open use and generally ‘learn how to bake’ using them.

• **Software:** Find wide families of software that have flexible and well documented structures. Build prototypes, encourage open use, and more baking.

• **Tooling:** Build, or buy, best of breed open tooling that minimises the effort needed to design, build and qualify any one specific product
Hardware
Hardware
VersiBoard2
# Hardware

## Why the imxrt?

<table>
<thead>
<tr>
<th>Product</th>
<th>CPU</th>
<th>DSP</th>
<th>Package</th>
<th>Memory</th>
<th>2D Acceleration</th>
<th>LCD</th>
<th>CSI</th>
<th>USB with PHY</th>
<th>Ethernet</th>
<th>CAN</th>
<th>Quad ENC/ Flex/ PWM</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.MX RT1064</td>
<td>Cortex-M7 @600 MHz</td>
<td>-</td>
<td>196 BGA</td>
<td>4MB Flash 1MB SRAM 32 kB I-cache 32 KB D-cache</td>
<td>PxP</td>
<td>8/16/24-bit Parallel</td>
<td>8/10/16-bit Parallel</td>
<td>OTG, HS/FS x2</td>
<td>2x 10/100</td>
<td>2x FlexCAN, 1x CANFD</td>
<td>4/4/4</td>
</tr>
<tr>
<td>i.MX RT1060</td>
<td>Cortex-M7 @600 MHz</td>
<td>-</td>
<td>196 BGA</td>
<td>1MB SRAM 32 kB I-cache 32 kB D-cache</td>
<td>PxP</td>
<td>8/16/24-bit Parallel</td>
<td>8/10/16-bit Parallel</td>
<td>OTG, HS/FS x2</td>
<td>2x 10/100</td>
<td>2x FlexCAN, 1x CANFD</td>
<td>4/4/4</td>
</tr>
<tr>
<td>i.MX RT106A</td>
<td>Cortex-M7 @600 MHz</td>
<td>AVS Software</td>
<td>196 BGA</td>
<td>1MB SRAM 32 kB I-cache 32 kB D-cache</td>
<td>PxP</td>
<td>8/16/24-bit Parallel</td>
<td>8/10/16-bit Parallel</td>
<td>OTG, HS/FS x2</td>
<td>2x 10/100</td>
<td>2x FlexCAN, 1x CANFD</td>
<td>4/4/4</td>
</tr>
<tr>
<td>i.MX RT1050</td>
<td>Cortex-M7 @600 MHz</td>
<td>-</td>
<td>196 BGA</td>
<td>512 kB SRAM 32 kB I-cache 32 kB D-cache</td>
<td>PxP</td>
<td>8/16/24-bit Parallel</td>
<td>8/10/16-bit Parallel</td>
<td>OTG, HS/FS x2</td>
<td>1x 10/100</td>
<td>2x FlexCAN</td>
<td>4/4/4</td>
</tr>
<tr>
<td>i.MX RT1020</td>
<td>Cortex-M7 @500 MHz</td>
<td>-</td>
<td>100 LQFP</td>
<td>144 LQFP 256 kB SRAM 16 kB I-cache 16 kB D-cache</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>OTG, HS/FS x1</td>
<td>1x 10/100</td>
<td>2x FlexCAN</td>
<td>2/2/2</td>
</tr>
<tr>
<td>i.MX RT1015</td>
<td>Cortex-M7 @500 MHz</td>
<td>-</td>
<td>100 LQFP</td>
<td>128 kB SRAM 16 kB I-cache 16 kB D-cache</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>OTG, HS/FS x1</td>
<td>-</td>
<td>-</td>
<td>1/1/1</td>
</tr>
<tr>
<td>i.MX RT1010</td>
<td>Cortex-M7 @500 MHz</td>
<td>-</td>
<td>80 LQFP</td>
<td>128 kB SRAM 16 kB I-cache 8 kB D-cache</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>OTG, HS/FS x1</td>
<td>-</td>
<td>-</td>
<td>0/0/1</td>
</tr>
</tbody>
</table>
• Starting point; GCC, GDB

...but what about gprof, gcov, top etc? Its not enough to be able to write and deploy good code, we need to be able to verify and validate its correct operation too.

Visualization and instrumentation is as important for writing good code as the compiler and the OS on which it runs.
Time honored method (i.e. from my day) of figuring out what's going on

```c
#define SETUP_BUSY GPIOB->CRH=((GPIOB->CRH)&0xFFF0FFFF)|0x30000
#define AM_IDLE GPIOB->BRR=(1<<12)
#define AM_BUSY GPIOB->BSRR=(1<<12)

SETUP_BUSY;
while (1)
{
    flagSet = flag_get();
    if (!flagSet)
    {
        AM_IDLE;
        __WFI();
        AM_BUSY;
    }
    else
    {
        <Do things>
    }
}
```
Use the low pass filter on your scope to watch the behavior of the system in realtime...

https://www.youtube.com/watch?v=5UFpp3ao460
Serial Port I/O

- Another Traditional method but it needs program support and dedicated hardware
- Output is easy, input demands polling or interrupts
- Approx. 11500 bytes/sec.

```c
void txDbgString(char *s) {
    while (*s) {
        while (!USART1→SR&USART_SR_TXE);
        USART1→DR=*s++;
    }
}

int rxDbgChr(char *r) {
    if (!USART1→SR&USART_SR_RXNE)
        return 0;
    *r=USART1→DR;
    return 1;
}
```
Semihosting...

- Comes for free with JTAG or SWD with most probes
- Provides ‘standard’ file semantics, including stdio – great for test cases!
- Relatively slow, around 1200 bytes/sec

```
extern void initialise_monitor_handles(void); /* prototype */
int main(void)
{
    initialise_monitor_handles();
    fprintf(stdout,"Hello, good evening and welcome\n");
    ...
}
```

>telnet localhost 2333
Hello, good evening and welcome

Big Warning!!
Wrecks Realtime Semantics!
Enter the SWO...

- Comes for free with SWD with most probes – not available via JTAG (re-uses TDO pin)

- Provides lots of functionality;
  - 32 Distinct debug channels
  - Hardware tracing and sampling
  - Debug Watch and Traps
  - Time stamps

- Approx 111K bytes/sec
- Packetised format, needs to be depacketised to be useful (in the general case).
Enter the SWO...
Enter the SWO

SWO Feed

Orbuculum

Orbtop

Orbstat

Multi-channel text and variable output

Sampled performance data

Runtime Code behaviour
Cortex M SWO SWV Demux and Postprocess

<table>
<thead>
<tr>
<th>Branch</th>
<th>Latest commit message</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>Merge branch 'master' of <a href="https://github.com/mubes/orbuculum">https://github.com/mubes/orbuculum</a></td>
<td>26 Feb 2018</td>
</tr>
<tr>
<td>Docs</td>
<td>Addition of SEGGER support</td>
<td>2 years ago</td>
</tr>
<tr>
<td>inc</td>
<td>Updates to orbrace to use high speed spi host side interface and sup...</td>
<td>2 years ago</td>
</tr>
<tr>
<td>src</td>
<td>fix build without FPGA support</td>
<td>last year</td>
</tr>
<tr>
<td>support</td>
<td>ITM Protocol simplifications and BMP ITM speedup</td>
<td>2 years ago</td>
</tr>
<tr>
<td>tools/or_trace</td>
<td>Complete but lightly tested version</td>
<td>2 years ago</td>
</tr>
<tr>
<td>config</td>
<td>Modifications to support build and operation on Linux</td>
<td>2 years ago</td>
</tr>
<tr>
<td>ortrace</td>
<td>Integration of SUMP2 functionality</td>
<td>2 years ago</td>
</tr>
<tr>
<td>sump2</td>
<td>Integration of SUMP2 functionality</td>
<td>2 years ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Addition of orcast utility (and bugfixes to network streaming)</td>
<td>2 years ago</td>
</tr>
<tr>
<td>CONTRIBUTORS</td>
<td>Reversion of uart changes and parameter setting in toplevel</td>
<td>2 years ago</td>
</tr>
<tr>
<td>COPYING</td>
<td>Initial Commit</td>
<td>2 years ago</td>
</tr>
<tr>
<td>Makefile</td>
<td>fix build without FPGA support</td>
<td>last year</td>
</tr>
<tr>
<td>README.md</td>
<td>Merge branch 'master' of <a href="https://github.com/mubes/orbuculum">https://github.com/mubes/orbuculum</a></td>
<td>last year</td>
</tr>
</tbody>
</table>

Orbuculum - ARM Cortex Debug Output Processing Tools

Stop press: Orbuculum now has an active Gitter channel at https://gitter.im/orbcode/orbuculum ... come join the discussion.
```c
static __INLINE uint32_t ITM_SendChar (uint32_t c, uint32_t ch) {
    if (((CoreDebug->DEMCR & CoreDebug_DEMCR_TRCENA_Msk) && /* Trace enabled */
        (ITM->TCR & ITM_TCR_ITMENA_Msk) && /* ITM enabled */
        (ITM->TER & (1ul << c) )) /* ITM Port c enabled */
    {
        While (0 == ITM->PORT[c].u32); /* Port available? */
        ITM->PORT[c].u8 = (uint8_t) ch; /* Write data */
    }
    return (ch);
}

void sendString(uint32_t ch, char *s) {
    while (*s) ITM_SendChar(*s++,ch);
}
```

```
>orbuculum -b swo/ -c 0,debug,"%c" -c 1,clientEvents,"%c" -c 2,Actions,"%c" -c 3,Z,"Z=%d\n" -c 4,Temperature,"Temp=%d\n"
>ls
  swo/
    debug
    clientEvents
    Actions
    Z
    Temperature
    hwevent
>_
```
sendString(DEBUG_CHANNEL, "Debug Event Happened");

sendUint32(Z_CHANNEL, 12345);

sendString(CLIENTEVENTS_CHANNEL, "Port Opened");

gdb>dwtTraceException 1

>cat debug
Debug Event Happened

>cat Z
Z=12345

>orbcat -c 0,"%c" -c 1,"%c" -c 3,"Z=%d
Port Opened

>cat hwevent
1,2,Resume,Thread
1,989,Enter,SysTick
1,6,Exit,SysTick
1,1,Resume,Thread
1,989,Enter,SysTick
1,4,Exit,SysTick
>Support/orbplot_top &
>orbtop -e ../STM32F103-skel/ofiles/firmware.elf –dumpfile.out

97.90%  4308 ** Sleeping **
1.25%   55 USB_LP_CAN1_RX0_IRQHandler
0.20%   9 xTaskIncrementTick
0.13%   6 Suspend
0.09%   4 SysTick_Handler
0.06%   3 Resume
0.06%   3 __WFI
0.04%   2 vTaskSwitchContext
0.04%   2 TIM_Cmd
0.02%   1 prvAddCurrentTaskToDelayedList
0.02%   1 xTaskResumeAll
0.02%   1 vTaskDelay
0.02%   1 PendSV_Handler
0.02%   1 __ISB
0.02%   1 taskIn
0.02%   1 statsGetRTVal
0.02%   1 taskOut

-----------------
4400 Samples
# Using SWO For Instrumentation

```c
__attribute__((no_instrument_function)) void __cyg_profile_func_enter (void *this_fn, void *call_site)
{
    if (!((ITM->TER&(1<<TRACE_CHANNEL))))
        return;
    uint32_t oldIntStat=__get_PRIMASK();
    __disable_irq();

    while (ITM->PORT[TRACE_CHANNEL].u32 == 0);
    ITM->PORT[TRACE_CHANNEL].u32 = (*((uint32_t *)0xE0001004))&0x03FFFFFF)|0x40000000;

    while (ITM->PORT[TRACE_CHANNEL].u32 == 0);
    ITM->PORT[TRACE_CHANNEL].u32 = (uint32_t)(call_site)&0xFFFFFFFE;
    while (ITM->PORT[TRACE_CHANNEL].u32 == 0);
    ITM->PORT[TRACE_CHANNEL].u32 = (uint32_t)this_fn&0xFFFFFFFE;
    __set_PRIMASK(oldIntStat);
}

__attribute__((no_instrument_function)) void __cyg_profile_func_exit (void *this_fn, void *call_site)
{
    if (!((ITM->TER&(1<<TRACE_CHANNEL))))
        return;
    uint32_t oldIntStat=__get_PRIMASK();
    __disable_irq();

    while (ITM->PORT[TRACE_CHANNEL].u32 == 0);
    ITM->PORT[TRACE_CHANNEL].u32 = (*((uint32_t *)0xE0001004))&0x03FFFFFF)|0x50000000;

    while (ITM->PORT[TRACE_CHANNEL].u32 == 0);
    ITM->PORT[TRACE_CHANNEL].u32 = (uint32_t)(call_site)&0xFFFFFFFF;
    while (ITM->PORT[TRACE_CHANNEL].u32 == 0);
    ITM->PORT[TRACE_CHANNEL].u32 = (uint32_t)this_fn&0xFFFFFFFF;
    __set_PRIMASK(oldIntStat);
}
```
>orbstat -e firmware.elf -y out.dot
dot out.dot -o -Tpng > out.png
evince out.png

```
None
  +-----------------------+
  | 2579                 |
  | PendSV_Handler       |
  | 1833                 |
  | vTaskSwitchContext   |
  | 265                  |
  | prvIdleTask          |
  | 122                  |
  | _ledFlashTask        |
  +-----------------------+
thirdparty/USB/src/usb_lstr.c
  | USB_LP_CAN1_RX0_IRQHandler |
  | 2001                   |
  | 491                    |
  +-----------------------+
thirdparty/USB/src/usb_pwr.c
  | Resume                |
  | 29                    |
  | Suspend               |
  +-----------------------+
src/stats.c
  | TIM4_IRQHandler       |
  | 29                    |
  | taskIn                |
  | taskOut               |
  | taskReady             |
  +-----------------------+
src/gpio.c
  | GPIO_WriteBit         |
  | 1                     |
  +-----------------------+
thirdparty/STM32F10x__Periph_Driver/src/stm32f10x_gpio.c
  | TIM_ClearITPendingBit |
  | 31                    |
  | TIM_GetITStatus       |
  +-----------------------+
thirdparty/STM32F10x__Periph_Driver/src/stm32f10x_tim.c
  | vApplicationIdleHook  |
  | 1833                  |
  | xTaskIncrementTick    |
  | 1                     |
```
>orbstat -e firmware.elf -z demo.out
gprof2dot -f callgrind demo.out > demo.dot
dot demo.dot -o -Tpng > out.png
evince out.png
Using SWO For Instrumentation
Interactive Output

```bash
> orbstat -e firmware.elf -z demo.out
> kcachegrind demo.out
```
Enter the SWO...
TRACE Feed

Orbuculum

Orbtrace
Instrumentation-free
Realtime code
Trace and
coverage

Orbtop
Multi-channel
text and
variable output

Orbstat
Sampled
performance
data

Runtime
Code
behaviour

SWO to TRACE...
What TRACE can do
(Ozone/J-TRACE examples)

<table>
<thead>
<tr>
<th>Source Coverage</th>
<th>Function</th>
<th>Inst. Coverage</th>
<th>Run Count</th>
<th>Fetch Count</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>87.5% (14/16)</td>
<td>_start</td>
<td>80.4% (42/47)</td>
<td>1</td>
<td>104393</td>
<td>87.26%</td>
</tr>
<tr>
<td>54.5% (12/22)</td>
<td>up_enable_dcache</td>
<td>65.8% (48/73)</td>
<td>1</td>
<td>9549</td>
<td>7.98%</td>
</tr>
<tr>
<td>100.0% (11/11)</td>
<td>modfrequ2</td>
<td>100.0% (48/48)</td>
<td>24</td>
<td>1152</td>
<td>0.96%</td>
</tr>
<tr>
<td>76.9% (29/36)</td>
<td>imrt_lomux_config</td>
<td>88.9% (80/90)</td>
<td>12</td>
<td>954</td>
<td>0.80%</td>
</tr>
<tr>
<td>86.7% (13/15)</td>
<td>imrt_gpio_config</td>
<td>94.1% (48/51)</td>
<td>11</td>
<td>539</td>
<td>0.45%</td>
</tr>
<tr>
<td>100.0% (3/3)</td>
<td>imrt_up2pck_con</td>
<td>100.0% (22/22)</td>
<td>24</td>
<td>520</td>
<td>0.44%</td>
</tr>
<tr>
<td>56.5% (13/23)</td>
<td>imrt_config_gpio</td>
<td>65.8% (52/79)</td>
<td>12</td>
<td>520</td>
<td>0.43%</td>
</tr>
<tr>
<td>38.0% (6/26)</td>
<td>imrt_pilsetup</td>
<td>41.2% (28/68)</td>
<td>1</td>
<td>418</td>
<td>0.35%</td>
</tr>
<tr>
<td>87.5% (7/8)</td>
<td>imrt_0ciy_select</td>
<td>97.5% (39/40)</td>
<td>11</td>
<td>359</td>
<td>0.30%</td>
</tr>
<tr>
<td>95.3% (82/86)</td>
<td>imrt_clockconfig</td>
<td>98.5% (266/278)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>40.0% (2/5)</td>
<td>imrt_padm_admi</td>
<td>65.2% (15/23)</td>
<td>12</td>
<td>192</td>
<td>0.16%</td>
</tr>
<tr>
<td>40.0% (2/5)</td>
<td>imrt_padm_addres</td>
<td>65.2% (15/23)</td>
<td>12</td>
<td>192</td>
<td>0.16%</td>
</tr>
<tr>
<td>75.5% (3/4)</td>
<td>imrt_padm_map</td>
<td>77.8% (14/18)</td>
<td>12</td>
<td>198</td>
<td>0.15%</td>
</tr>
<tr>
<td>61.9% (13/21)</td>
<td>imrt_gpio_config</td>
<td>80.6% (58/72)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>90.0% (19/21)</td>
<td>imrt_gpioconfi</td>
<td>97.4% (37/38)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>77.8% (7/9)</td>
<td>imrt_gpio_output</td>
<td>84.6% (33/39)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>60.0% (6/10)</td>
<td>mpu_control</td>
<td>83.8% (31/37)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>62.5% (5/8)</td>
<td>imrt_gpio_output</td>
<td>70.2% (32/42)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>80.0% (4/5)</td>
<td>imrt_gpio_inte</td>
<td>96.9% (31/32)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>91.7% (11/12)</td>
<td>imrt_tom menable</td>
<td>96.8% (30/31)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>100.0% (5/5)</td>
<td>imrt_gpio_direc</td>
<td>100.0% (30/30)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>92.3% (12/13)</td>
<td>imrt_lowesetp</td>
<td>96.3% (26/27)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>90.0% (9/10)</td>
<td>up_enable_dcache</td>
<td>96.8% (24/25)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>100.0% (4/4)</td>
<td>arm_clz</td>
<td>100.0% (13/13)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>100.0% (3/3)</td>
<td>imrt_gpio_sect</td>
<td>100.0% (11/11)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>100.0% (5/5)</td>
<td>imrt_mpu_initia</td>
<td>100.0% (10/10)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>100.0% (2/2)</td>
<td>mpu_showtype</td>
<td>100.0% (6/6)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>66.7% (2/3)</td>
<td>imrt_niisod_set</td>
<td>85.7% (6/7)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
<tr>
<td>100.0% (3/3)</td>
<td>imrt_boardinitialze</td>
<td>100.0% (5/5)</td>
<td>1</td>
<td>270</td>
<td>0.23%</td>
</tr>
</tbody>
</table>
What TRACE can do
(Ozone/J-TRACE examples)
The future...

- Full Profiling of code run
- Pre-trigger recording (I’ve crashed, how did that happen?)
- Non-intrusive, full speed, code path observation with no requirement for instrumentation
- Full speed software channel output
- Triggers

..more at www.shadetail.com

www.github.com/mubes/orbuculum
Thanks for listening
Technolution