Embedded Ada with Alire

Fabien Chouteau

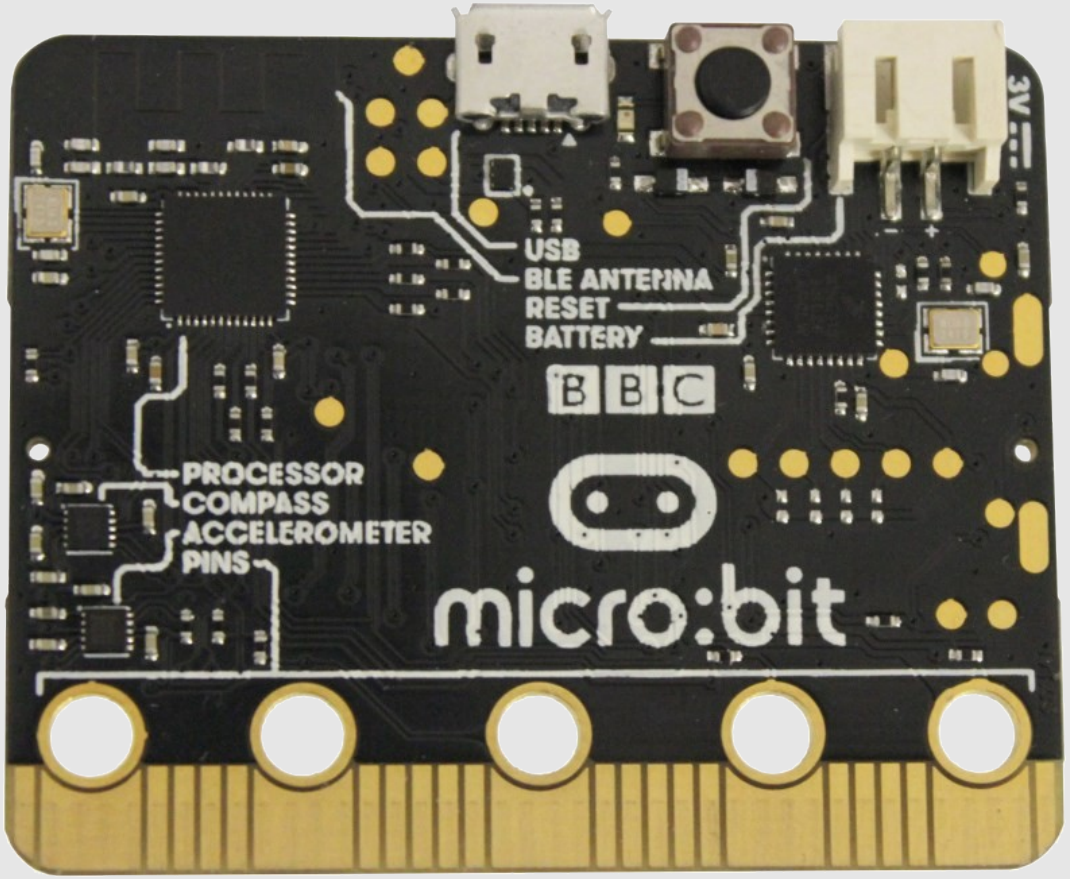
Embedded Software Engineer

Twitter : @DesChips
GitHub : Fabien-Chouteau
Hackaday.io: Fabien.C

Scope: MCUs

- Micro-controllers
  - “Simple” devices
  - A few KiB sometimes MiB of RAM and ROM
  - No virtual memory
  - A lot of inputs/outputs
- No Operating System (bare-metal)

The hardware



What do we need?

- Toolchain
- Board Support Package (BSP)
  - Run-time
  - Startup code
  - Linker scripts
  - Drivers
- Libraries

Architecture of the crates

Create the `nrf51_hal` crate

```
$ alr init --lib nrf51_hal
$ cd nrf51_hal
```

Add `cortex_m` dependency

```
$ alr with cortex_m
```

Add `gnat_arm_elf` dependency

```
$ alr with gnat_arm_elf
```

Edit GPR file

Create the `microbit_bsp` crate

```
$ cd ..
$ alr init --lib microbit_bsp
$ cd microbit_bsp
```

Add a pin dependency to `nrf51_hal`

```
$ alr with nrf51_hal --use=../nrf51_hal
```

Configure run-time in GPR file

Zero-FootPrint run-times without parts that are specific to a given MCU or board

That means without:

- Linker script
- Startup code (crt0.S)

Add device configuration in GPR file

Get and build startup-gen

```
$ alr get --build startup_gen
```

Use startup-gen generator

```
$ startup-gen -P microbit_bsb.gpr \
              -l src/link.ld \
              -s src/crt0.S
```

Add crt0 + linker script in GPR file

Create the my_application crate

```
$ cd ..
$ alr init --bin my_application
$ cd my_application
```

Add a pin dependency to `microbit_bsp`

```
$ alr with microbit_bsp --use=../microbit_bsp
```

Configure GPR file

Write hello-world

```
with Ada.Text_IO;

procedure My_Application is
begin
   for X in 1 .. 10 loop
      Ada.Text_IO.Put_Line ("Hello World!");
   end loop;
end My_Application;
```

First build

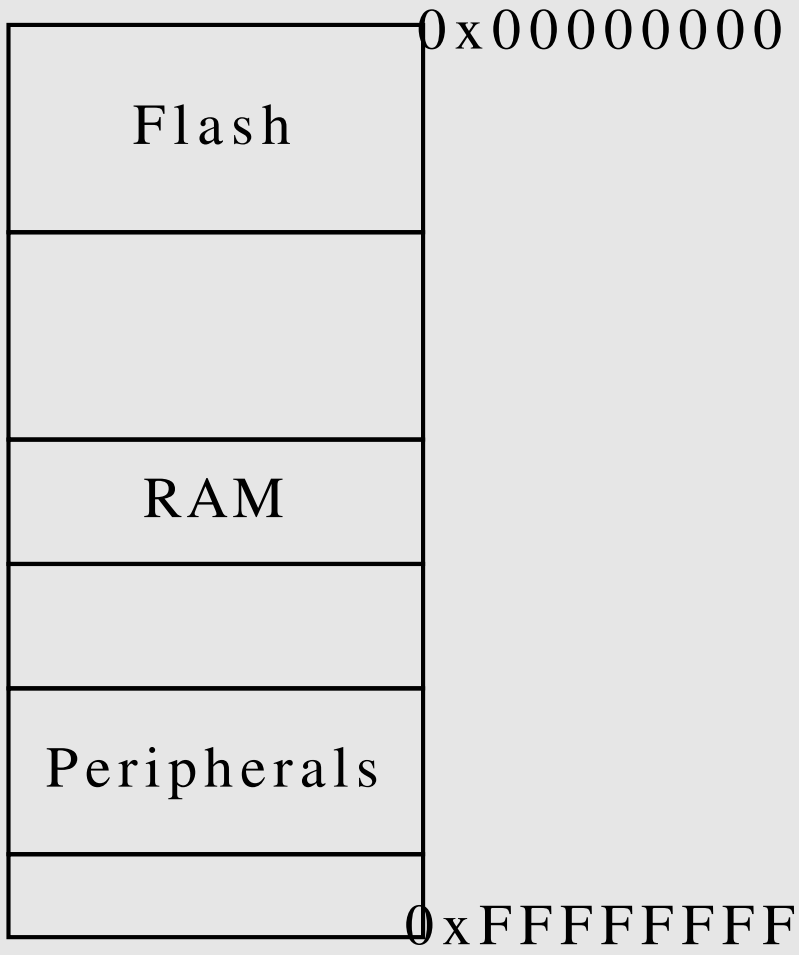
```
$ alr build
```

Run hello-world on QEMU

```
$  qemu-system-arm -nographic -no-reboot \
                   -semihosting -M microbit \
                   -kernel bin/my_application
```

Peripheral Drivers

Memory Mapped Registers



Memory Mapped Registers

7	6	5	4	3	2	1	0
Reserved		Sense		Reserved			

Sense: Pin sensing mechanism

0: Disabled

2: Sense for high level

3: Sense for low level

Hardware Mapping

```
#define SENSE_MASK     (0x30)
#define SENSE_POS      (4)

#define SENSE_DISABLED (0)
#define SENSE_HIGH     (2)
#define SENSE_LOW      (3)

uint8_t *register = 0x80000100;

// Clear Sense field
*register &= ~SENSE_MASK;
// Set sense value
*register |= SENSE_DISABLED << SENSE_POS;
```

Hardware Mapping

```
--  High level view of the Sense field
type Pin_Sense is
  (Disabled,
   High,
   Low)
  with Size => 2;

--  Hardware representation of the Sense field
for Pin_Sense use
  (Disabled => 0,
   High     => 2,
   Low      => 3);
```

Hardware Mapping

```
--  High level view of the register
type IO_Register is record
   Reserved_A : UInt4;
   SENSE      : Pin_Sense;
   Reserved_B : UInt2;
end record;

--  Hardware representation of the register
for IO_Register use record
   Reserved_A at 0 range 0 .. 3;
   SENSE      at 0 range 4 .. 5;
   Reserved_B at 0 range 6 .. 7;
end record;
```

Hardware Mapping

```
Register : IO_Register
  with Address => 16#8000_0100#;
```

```
Register.SENSE := Disabled;
```

Mapping for the nRF51

- nRF51
- 28 peripherals
- 414 memory mapped registers
- 903 fields in the registers

Who wants to write all the representation clauses?

System View Description (SVD)

```
<field>
  <name>SENSE</name>
  <description>Pin sensing mechanism.</description>
  <lsb>4</lsb> <msb>5</msb>
  <enumeratedValues>
    <enumeratedValue>
      <name>Disabled</name>
      <description>Disabled.</description>
      <value>0x00</value>
    </enumeratedValue>
 [...]
```

Get and build svd2ada

```
$ alr get --build svd2ada
```