Embedded Ada with Alire

Fabien Chouteau

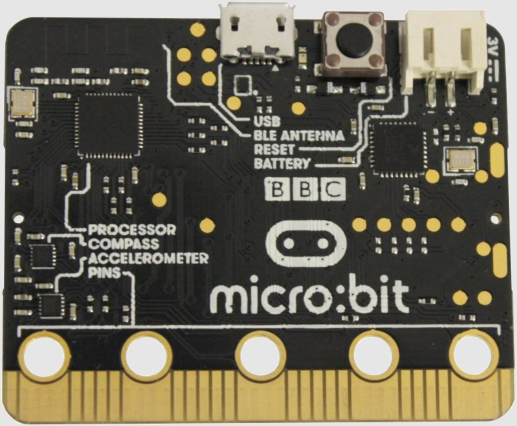
Embedded Software Engineer

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

AdaCore

Scope: MCUs

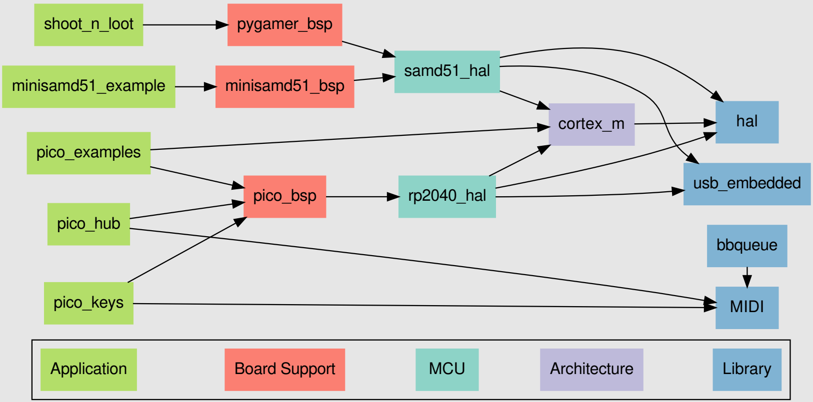
- Micro-controllers
  - "Simple" devices
  - A few KiB sometimes MiB of RAM and ROM
  - No virtual memory
  - A lot of inputs/outputs
- No Operating System (bare-metal)


USB
BLE ANTENNA
RESET
BATTERY
BBC
PROCESSOR
COMPASS
ACCELEROMETER
PINS
micro:bit
3V

What do we need?

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

Architecture of crates



AdaCore

Crates in this tutorial

- nrf51_hal
- microbit_bsp
- my_application

Create the `nrf51_hal` crate

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

Add useful dependencies

```
$ alr with cortex_m
$ alr with hal
```

AdaCore

Add `gnat_arm_elf` dependency (toolchain)

```
$ alr with gnat_arm_elf
```

AdaCore

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
```

AdaCore

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)

```
for Target use "arm-elf";
for Runtime ("Ada") use "zfp-cortex-m0";
```

Get and build startup-gen

`startup-gen` generates startup files (crt0 and linker script) based on properties of the target device such as: architecture, memory layout, number of interrupts.

```
$ cd ..
$ alr get --build startup_gen
```

Add device configuration in GPR file

```
package Device_Configuration is
   for CPU_Name use "ARM Cortex-M0";
   for Number_Of_Interrupts use "32";

   for Memories use ("flash", "ram");

   for Mem_Kind ("flash") use "rom";
   for Address  ("flash") use "0x00000000";
   for Size     ("flash") use "256K";

   for Mem_Kind ("ram") use "ram";
   for Address  ("ram") use "0x20000000";
   for Size     ("ram") use "16K";

   for Boot_Memory use "flash";
end Device_Configuration;
```

Use startup-gen generator

```
$ cd microbit_bsp
$ alr exec -- ../startup_gen_22.0.0_85f5a122/startup-gen \
              -P microbit_bsp.gpr \
              -l src/link.ld \
              -s src/crt0.S
```

Add crt0 + linker script in GPR file

Add `Asm_CPP` language to build `crt0.S`:

```
for Languages use ("Ada", "Asm_CPP");
```

Define a linker switch variable for the linker script:

```
Linker_Switches := ("-T", Project'Project_dir & "/src/link.ld");
```

This variable will be used by the application.

Create the my_application crate

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

AdaCore

Add a pin dependency to `microbit_bsp`

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

AdaCore

Configure GPR file

```
   -- [...]
with "microbit_bsp.gpr";

project My_Application is
   -- [...]

   for Runtime ("Ada") use MicroBit_BSP'Runtime ("Ada");
   for Target use MicroBit_BSP'Target;

   package Linker is
      for Default_Switches ("Ada") use
        MicroBit_BSP.Linker_Switches &
        ("-Wl,--print-memory-usage",
         "-Wl,--gc-sections");
   end Linker;
```

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

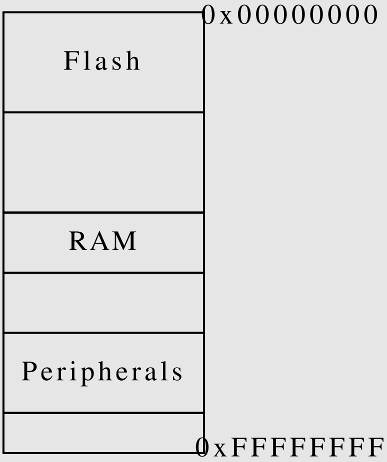
```
$ alr build
```

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

```c
#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);
```

AdaCore

Hardware Mapping

```ada
--  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)

```xml
<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

```
$ cd ..
$ alr get --build svd2ada
```

Run SVD2Ada

```
$ cd nrf51_hal
$ ../svd2ada_0.1.0_6eb0b591/bin/svd2ada \
    ../svd2ada_0.1.0_6eb0b591/CMSIS-SVD/Nordic/nrf51.svd \
    --boolean \
    -o src/ \
    -p nRF51_SVD \
    --base-types-package HAL \
    --gen-uint-always
```

Basic RNG driver spec: `src/nrf51_hal-rng.ads`

```
with HAL;
package Nrf51_Hal.RNG is
   function Read return HAL.UInt8;
end Nrf51_Hal.RNG;
```

Basic RNG driver body: src/nrf51_hal-rng.adb

```
with nRF51_SVD.RNG;
package body Nrf51_Hal.RNG is
   function Read return HAL.UInt8 is
      use HAL;
      use nRF51_SVD.RNG;
   begin
      RNG_Periph.EVENTS_VALRDY := 0; -- Clear event
      RNG_Periph.TASKS_START := 1; -- Start generator

      while RNG_Periph.EVENTS_VALRDY = 0 loop
         null; -- Wait for value ready
      end loop;

      return RNG_Periph.VALUE.VALUE;
   end Read;
end Nrf51_Hal.RNG;
```

Update Application to use RNG driver

```
with Ada.Text_IO;
with Nrf51_Hal.RNG;

procedure My_Application is
begin

   for X in 1 .. 10 loop
      Ada.Text_IO.Put_Line ("Hello World!" &
                                    Nrf51_Hal.RNG.Read'Img);
   end loop;
end My_Application;
```

Build again

```
$ cd ..
$ cd my_application
$ alr build
```

AdaCore

Run on QEMU

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

AdaCore