Embedded Ada/SPARK Programming in 2022

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)
Ada/SPARK for embedded: Pros

- Benefits of high-level language
  - Contracts
  - Tagged types
  - Discriminated types
  - Arrays
  - Etc.
- Representation Clauses
- Interfacing with C
- Less debugging (debugging is hard in embedded)
- SPARK: formal proof
- Alire!
- Ravenscar Tasking
Ravenscar Tasking

A.K.A There’s a mini-RTOS in my language\textsuperscript{1}

- Tasks (threads)
- Time handling
  - Clock
  - Delays
- Protected Objects:
  - Mutual exclusion
  - Synchronization between tasks
  - Interrupt handling

\textsuperscript{1}\texttt{blog.adacore.com/theres-a-mini-RTOS-in-my-language}
Ada/SPARK for embedded: Cons

- Ravenscar Tasking
- Secondary Stack
- Toolchain availability
- Drivers/BSP availability
- Library ecosystem
What do you need?
What do you need?

- Board Support Package (BSP)
  - Run-time
  - Startup code
  - Linker scripts
  - Drivers
- Libraries
- Toolchain
<table>
<thead>
<tr>
<th>Startup</th>
<th>Drivers</th>
<th>Functional</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>Ada</td>
<td>Ada</td>
<td>Ada</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>Ada</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>Ada/C</td>
<td>Ada</td>
<td>C/Ada</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>Ada</td>
<td>...</td>
</tr>
</tbody>
</table>
Board Support Package
ARM Microcontroller Market

- Dozens of vendors (ST, Microchip, NXP, Nordic, Cypress, Infineon, nuvoTon, TI, Raspberry Pi, etc.)
- 8 variants (Cortex-M0/M0+/M1/M3/M4/M7/M23/M33)
- Thousands of individual parts (4000+):
  - STM32F446RET6
  - nRF51822-QFAA-R
  - APM32F103C6
  - ATSAME54N20A
  - HT32F22366
  - XMC1302-Q040x0032
  - EFM32GG280F1024
  - MB9AF155M
  - S6E2CC8H0A
  - MK60DN256xxx10
  - LM4F122H5QD
  - LPC1114FHN33/202
  - TMPM3H2FWDUG
How can we support so many devices?
Startup and run-time
The “generic” ZFP run-times

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

That means without:

- Linker script
- Startup code (crt0.S)
package Device_Configuration is
  for Cpu_Name use "ARM Cortex-M4F";
  for Memories use ("HSRAM", "FLASH");
  for Boot_Memory use "FLASH";

  for Mem_Kind ("FLASH") use "rom";
  for Address ("FLASH") use "0x08000000";
  for Size ("FLASH") use "0x80000";

  for Mem_Kind ("HSRAM") use "ram";
  for Address ("HSRAM") use "0x20000000";
  for Size ("HSRAM") use "0x30000";
end Device_Configuration;
$ alr get/with startup_gen
$ startup-gen -P samd51.gpr -l src/link.ld -s src/crt0.S
Peripheral Drivers
Memory Mapped Registers

Flash

RAM

Peripherals

0x00000000

0xFFFFFFFF
### Memory Mapped Registers

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Sense</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sense:** Pin sensing mechanism

- 0: Disabled
- 2: Sense for high level
- 3: Sense for low level
#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;
-- 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);
-- High level view of the register

```haskell
type IO_Register is record
    Reserved_A : UInt4;
    SENSE : Pin_Sense;
    Reserved_B : UInt2;
end record;
```

-- Hardware representation of the register

```haskell
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;
Let’s focus on one microcontroller

- STM32F446RET6
- 46 peripherals
- 881 memory mapped registers
- 6820 fields in the registers

Who wants to write all the representation clauses for that beast?
<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>
  </enumeratedValues>
[...]

SVD2Ada

Generates Ada representation clauses from SVD file.

$ alr get/with svd2ada
Libraries
Embedded projects with Alire

- shoot_n_loot
- minisamd51_example
- pico_examples
- pico_hub
- pico_keys
- pygamer_bsp
- minisamd51_bsp
- samd51_hal
- cortex_m
- hal
- rp2040_hal
- usb_embedded
- bbqueue
- MIDI

---

Application  | Board Support  | MCU  | Architecture  | Library
Available Toolchains

ARM
RISC-V
AVR
What about SPARK?
What about SPARK?

- All of the above apply
- Easier to do SPARK in embedded
- Easier to start with libraries
Questions ?
Thank you!