Zephyr on Beetle

Vincenzo Frascino
Senior Engineer

OpenIoT, Portland, Oregon
22/02/2017

©ARM 2017
Agenda

- Overview
- Zephyr Architecture
- Zephyr on Beetle
- Contributing to Zephyr
- Beetle Examples and Demo
Overview
Zephyr Overview

- Zephyr is an OS that runs on MCUs with a small memory footprint
- Its initial codebase has been established around 2000 and has been made Open Source on February 2016
- It is licensed under Apache 2.0
- It is Modular and Configurable
- It does not provide user-space and dynamic run-time
- It allocates memory and resources statically where possible
- It is cross-platform:
  - ARM
  - IA32
  - ARC
  - ....
Zephyr Architecture
Zephyr Building Blocks

- Application
- Network Communication Protocols
- UART
- GPIO
- I2C
- SPI
- ...
- HW Platform (i.e. Beetle)
- C API
- Security
- Device Management
- Unified Kernel
- Board Support
- Device Drivers
- Power Management
- SoC
- IP
- Security
- Network Communication Protocols
- UART
- GPIO
- I2C
- SPI
- ...
Zephyr on Beetle

Setup the environment
ARM Beetle IoT Evaluation Platform

ARM BEETLE Technical Highlights:

- ARM Cortex-M3
- ARM IoT Subsystem for Cortex-M
- CORDIO Bluetooth Smart radio
- Memory
  - 256KB of embedded flash
  - 128KB SRAM
  - 2MB of external QSPI flash.
- Debug
  - JTAG, SWD & 4 bit TRACE
  - CMSIS-DAP with a virtual UART port
- Arduino interface
  - GPIO, UART, SPI, I2C
  - Analog signals

Beetle docs: https://www.zephyrproject.org/doc/boards/arm/v2m_beetle/doc/v2m_beetle.html

©ARM 2017
hello_world example on ARM Beetle (1/3)

- Setup the environment:
  - This example has been tested on Ubuntu 14.04.
  - Install the development environment:
    > sudo apt-get update
    > sudo apt-get install git make gcc g++ python3-PLY
    > wget https://nexus.zephyrproject.org/content/repositories/releases/org/zephyrproject/zephyr-sdk/0.9/zephyr-sdk-0.9-setup.run
    > chmod +x zephyr-sdk-0.9-setup.run
    > ./zephyr-sdk-0.9-setup.run (The simplest way is to use the default dir: /opt/zephyr-sdk)

- Download the zephyr source code:
  > git clone https://gerrit.zephyrproject.org/r/zephyr zephyr-project

- Create the zephyrrc file in your home dir:
  > cat <<EOF > ~/.zephyrrc
  export ZEPHYR_GCC_VARIANT=zephyr
  export ZEPHYR_SDK_INSTALL_DIR=/opt/zephyr-sdk
  EOF
hello_world example on ARM Beetle (2/3)

- Zephyr supports even alternative toolchains.
- Setup an alternative environment (i.e. gcc-arm-none-eabi):
  - This example has been tested on Ubuntu 14.04.
  - Install the development environment:
    > sudo add-apt-repository ppa:team-gcc-arm-embedded/ppa
    > sudo apt-get update
    > sudo apt-get install git make gcc g++ python3-ply gcc-arm-embedded
- Download the zephyr source code:
  > git clone https://gerrit.zephyrproject.org/r/zephyr zephyr-project
- Create the zephyrrc file in your home dir (gcc-arm is installed in /usr/bin):
  > cat <<EOF > ~/.zephyrrc
  export ZEPHYR_GCC_VARIANT=gccarmemb
  export GCCARMEMB_TOOLCHAIN_PATH=/usr
  EOF
Build the hello_world example:

```bash
> cd zephyr-project
> . zephyr-env.sh
> cd samples/hello_world
> make BOARD=v2m_beetle
```

- This will generate a binary called zephyr.bin into outdir.
- Flash the binary into the Beetle copying it into the MBED drive.
- Reset the board and you should see something like:
Zephyr on Beetle
BSP Porting
Porting a BSP to Zephyr OS

To port a BSP to Zephyr OS the following components are required:

- **SoC** (arch/<arch>/soc)
- **Board** (boards/<arch>)
  - defconfig (boards/<arch>/<board>/<board>_defconfig)
- **Drivers**
  - Pinmux (drivers/pinmux)
  - GPIO (drivers/gpio)
  - UART (drivers/serial)
  - Watchdog (drivers/watchdog)
  - ...
- **Documentation** (mainly in doc/)
Zephyr OS Boot (on ARM) (1/3)

- At reset the `reset_handler` is executed and it is responsible to:
  - Setup an initial stack.
  - If running an XIP (eXecute In Place) kernel (CONFIG_XIP=y), copy initialized data from ROM to RAM.
  - Lock the interrupts (will be unlocked when switch to the main task).
  - If present, initialize the board specific Watchdog.
  - Switch stacks from MSP to PSP
  - Jump to `_PrepC()` (arch/arm/prep_c.c) to finish setting up the system to be able to run C code. `_PrepC()`:
    - Relocates the vector table (if the option is enabled)
    - Enables the FPU (if the option is enabled)
    - Zeroes the BSS section
    - Jumps to `_Cstart()` (kernel/init.c) which is responsible for the early kernel init.
Zephyr OS Boot (on ARM) (2/3)

- **_Cstart()** is responsible for context switching out of the fake context running at start-up into the main thread. Now we are able to execute C code. **_Cstart()**: 
  - Initializes the kernel data structures and the interrupt sub-system.
  - Performs the basic hardware initialization via init levels:
    - **_SYS_INIT_LEVEL_PRIMARY** (deprecated)
    - **_SYS_INIT_LEVEL_PRE_KERNEL_1**
    - **_SYS_INIT_LEVEL_PRE_KERNEL_2**
  - Initializes stack canaries.
  - Prints the OS banner (if enabled).
  - Switches to the main thread (**switch_to_main_thread()**).
The switch to the main thread on the ARM architectures is implemented via (_arch_switch_to_main_thread()):

- Moves the PSP to the higher address of the stack
- Unlocks the interrupts
- Branches to the entry of the thread main (_thread_entry(_main ...)).

_main() performs the remaining init levels:

- _SYS_INIT_LEVEL_POST_KERNEL
- _SYS_INIT_LEVEL_SECONDARY (deprecated)
- _SYS_INIT_LEVEL_NANOKERNEL (deprecated)
- _SYS_INIT_LEVEL_MICROKERNEL (deprecated)
- _SYS_INIT_LEVEL_APPLICATION

_main() initializes the static threads (i.e. idle)

_main() jumps to the application main().

---

Stack Setup

XIP Check

_PrepC()

_Cstart()

switch_to_main_thread()

_main()

main()
Zephyr uses Kconfig to define the build configuration parameters.

<table>
<thead>
<tr>
<th>Kconfig.soc</th>
<th>Kconfig.series</th>
</tr>
</thead>
<tbody>
<tr>
<td>choice</td>
<td>config SOC_SERIES_BEETLE</td>
</tr>
<tr>
<td>prompt &quot;ARM Beetle SoC&quot;</td>
<td>bool &quot;ARM Beetle MCU Series&quot;</td>
</tr>
<tr>
<td>depends on SOC_SERIES_BEETLE</td>
<td>depends on ARM</td>
</tr>
<tr>
<td></td>
<td>select CPU_CORTEX_M</td>
</tr>
<tr>
<td></td>
<td>select CPU_CORTEX_M3</td>
</tr>
<tr>
<td></td>
<td>select SOC_FAMILY_ARM</td>
</tr>
<tr>
<td></td>
<td>select CPU_HAS_SYSTICK help</td>
</tr>
<tr>
<td>config SOC_BEETLE_R0</td>
<td>Enable support for Beetle MCU Series</td>
</tr>
<tr>
<td>bool &quot;ARM BEETLE R0&quot;</td>
<td></td>
</tr>
<tr>
<td>endchoice</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kconfig.defconfig.series</th>
<th>Kconfig.defconfig.series.beetle_r0</th>
</tr>
</thead>
<tbody>
<tr>
<td>• SOC_SERIES</td>
<td>• SOC</td>
</tr>
<tr>
<td>• NUM_IRQ_PRIO_BITS</td>
<td>• SRAM_SIZE</td>
</tr>
<tr>
<td>• NUM_IRQS</td>
<td>• FLASH_SIZE</td>
</tr>
<tr>
<td>• SYS_CLOCK_HW_CYCLES_PER_SEC</td>
<td></td>
</tr>
<tr>
<td>• SRAM_BASE_ADDRESS</td>
<td></td>
</tr>
<tr>
<td>• FLASH_BASE_ADDRESS</td>
<td></td>
</tr>
</tbody>
</table>
The SoC code defines:
- Boot Entry code (soc.c)
- IRQs (soc_irqs.h)
- Pins (soc_pins.h)
- Registers (soc_registers.h)
- Power Management (power.c)

The SoC init it is added at compile time at the “init level queue” and executes as PRE_KERNEL_1
Zephyr OS Port – Boards (1/2)

- Each board is located in boards/<arch>/<board_name> and contains:
  - Board definitions
  - Initial pinmuxing
  - Board configuration file
  - Main platform Makefile
  - Board documentation

- One of the boards acceptance criteria is to enable them against the automated test cases (sanitycheck).

- The default board defconfig has to be named <board_name>_defconfig

- The documentation has to provide a clear idea on what is the board's IP list and what is currently supported by the Zephyr OS BSP.

- The documentation has to provide at least the description of one example (i.e. hello_world) against which the board can be tested.
Zephyr OS Port – Boards (2/2)

<table>
<thead>
<tr>
<th>v2m_beetle_defconfig</th>
<th>Kconfig.defconfig</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG_ARM=y</td>
<td>if BOARD_V2M_BEETLE</td>
</tr>
<tr>
<td>CONFIG_SOC_FAMILY_ARM=y</td>
<td>config BOARD</td>
</tr>
<tr>
<td>CONFIG_SOC_BEETLE_R0=y</td>
<td>default v2m_beetle</td>
</tr>
<tr>
<td>CONFIG_SOC_SERIES_BEETLE=y</td>
<td>if GPIO</td>
</tr>
<tr>
<td>CONFIG_BOARD_V2M_BEETLE=y</td>
<td>config GPIO_CMSDK_AHB</td>
</tr>
<tr>
<td>CONFIG_CORTEX_M_SYSTICK=y</td>
<td>def_bool y</td>
</tr>
<tr>
<td>CONFIG_RUNTIME_NMI=y</td>
<td>config GPIO_CMSDK_AHB_PORT0</td>
</tr>
<tr>
<td>CONFIG_CLOCK_CONTROL=y</td>
<td>def_bool y</td>
</tr>
<tr>
<td># 24MHz system clock</td>
<td>config GPIO_CMSDK_AHB_PORT1</td>
</tr>
<tr>
<td>CONFIG_SYS_CLOCK_HW_CYCLES_PER_SEC=24000000</td>
<td>def_bool y</td>
</tr>
<tr>
<td># GPIOs</td>
<td>config GPIO_CMSDK_AHB_PORT2</td>
</tr>
<tr>
<td>CONFIG_GPIO=y</td>
<td>def_bool y</td>
</tr>
<tr>
<td>...</td>
<td>config GPIO_CMSDK_AHB_PORT3</td>
</tr>
<tr>
<td></td>
<td>def_bool y</td>
</tr>
<tr>
<td></td>
<td>endif # GPIO</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Zephyr OS Port – Drivers

- Zephyr OS supports different types of device drivers.
- Zephyr OS provides a consistent device model for configuring the drivers that are part of a system.
- The device model is responsible for initializing all the drivers configured into the system.
- Each type of driver is supported by a generic type API.
- The driver APIs are provided by `device.h`.
- Each device class has a device independent subsystem API associated.
 typedef int (*subsystem_do_this_t)(struct device *device);
 typedef void (*subsystem_do_that_t)(struct device *device);

 struct subsystem_api {
   subsystem_do_this_t do_this;
   subsystem_do_that_t do_that;
 };

 static inline int subsystem_do_this(struct device *device) {
   struct subsystem_api *api;
   api = (struct subsystem_api *)device->driver_api;
   return api->do_this(device);
 }

 static int my_driver_do_this(struct device *device) {
   ...
 }

 static struct subsystem_api my_driver_api_funcs = {
   .do_this = my_driver_do_this,
   .do_that = my_driver_do_that
 };
Zephyr OS Port – Drivers – Pinmux

- Zephyr OS supports two types of Pinmuxing:
  - Board (SoC) Pinmuxing
    - It configures the boot default Multiplexing of the present IO ports
    - It is contained in the pinmux.c and sits into the board directory
    - It is executed at PRE_KERNEL_1 init level

```
static void arm_v2m_beetle_pinmux_defaults(void)
{
    uint32_t gpio_0 = 0;
    uint32_t gpio_1 = 0;
    /* Set GPIO Alternate Functions */
    gpio_0 = (1<<0); /* Sheild 0 UART 0 RXD */
    gpio_0 |= (1<<1); /* Sheild 0 UART 0 TXD */
    /* ... */
    CMSDK_AHB_GPIO0_DEV->altfuncset = gpio_0;
    /* ... */
}

static int arm_v2m_beetle_pinmux_init(
    struct device *port)
{
    ARG_UNUSED(port);
    arm_v2m_beetle_pinmux_defaults();
    return 0;
}

SYS_INIT(arm_v2m_beetle_pinmux_init, PRE_KERNEL_1, CONFIG_KERNEL_INIT_PRIORITY_DEFAULT);
```
“Runtime” Pinmuxing

- Allows to change the pins configuration from the main application
- It is mainly used for testing purposes and early prototyping
- To not damage permanently the board, it is always better to refer to the TRM before doing any operation with this driver

```c
static int pinmux_set(struct device *dev,
                        uint32_t pin, uint32_t func)
{
    /* ... */
    /*
    * The irq_lock() here is required to prevent
    * concurrent callers to corrupt the pin
    * functions.
    */
    key = irq_lock();
    tmp = port->altfuncset;
    tmp |= (1 << (pin % PINS_PER_PORT));
    port->altfuncset = tmp;
    irq_unlock(key);
    /* ... */
}
/* ... */

static struct pinmux_driver_api api_funcs = {
    .set = pinmux_set,
    .get = pinmux_get,
    .pullup = pinmux_pullup,
    .input = pinmux_input
};
/* ... */
DEVICE_AND_API_INIT(pmux_dev, CONFIG_PINMUX_DEV_NAME,
                        &pmux_dev_init, NULL, NULL, PRE_KERNEL_1,
                        CONFIG_KERNEL_INIT_PRIORITY_DEFAULT,
                        &api_funcs);
```
## Zephyr OS Port – Drivers – GPIO

### gpio_cmsdk_AHB.c

```c
/* Port 0 */
#define CONFIG_GPIO_CMSDK_AHB_PORT0

static void gpio_cmsdk_AHB_config_0(struct device *dev);

static const struct gpio_cmsdk_AHB_conf gpio_cmsdk_AHB_0_conf = {
    .port = (((volatile struct gpio_cmsdk_AHB *)CMSDK_AHB_GPIO0)),
    .gpio_config_func = gpio_cmsdk_AHB_config_0,
    .gpio_cc_as = {.bus = CMSDK_AHB, .state = SOC_ACTIVE,
                  .device = CMSDK_AHB_GPIO0},
    .gpio_cc_ss = {.bus = CMSDK_AHB, .state = SOC_SLEEP,
                   .device = CMSDK_AHB_GPIO0},
    .gpio_cc_dss = {.bus = CMSDK_AHB, .state = SOC_DEEPSLEEP,
                    .device = CMSDK_AHB_GPIO0},
};

static struct gpio_cmsdk_AHB_dev_data gpio_cmsdk_AHB_0_data;

DEVICE_AND_API_INIT(gpio_cmsdk_AHB_0,..., POST_KERNEL,
                     CONFIG_KERNEL_INIT_PRIORITY_DEVICE,
                     &gpio_cmsdk_AHB_drv_api_funcs);

static void gpio_cmsdk_AHB_config_0(struct device *dev)
{
    IRQ_CONNECT(IRQ_PORT0_ALL, CONFIG_GPIO_CMSDK_AHB_PORT0_IRQ_PRI,
                gpio_cmsdk_AHB_isr,
                DEVICE_GET(gpio_cmsdk_AHB_0), 0);
    irq_enable(IRQ_PORT0_ALL);

    #ifdef CONFIG_GPIO_CMSDK_AHB_PORT0
    static void gpio_cmsdk_AHB_config_0(struct device *dev);
    static const struct gpio_cmsdk_AHB_conf gpio_cmsdk_AHB_0_conf = {
        .port = (((volatile struct gpio_cmsdk_AHB *)CMSDK_AHB_GPIO0)),
        .gpio_config_func = gpio_cmsdk_AHB_config_0,
        .gpio_cc_as = {.bus = CMSDK_AHB, .state = SOC_ACTIVE,
                       .device = CMSDK_AHB_GPIO0},
        .gpio_cc_ss = {.bus = CMSDK_AHB, .state = SOC_SLEEP,
                       .device = CMSDK_AHB_GPIO0},
        .gpio_cc_dss = {.bus = CMSDK_AHB, .state = SOC_DEEPSLEEP,
                        .device = CMSDK_AHB_GPIO0},
    };
    static struct gpio_cmsdk_AHB_dev_data gpio_cmsdk_AHB_0_data;

    DEVICE_AND_API_INIT(gpio_cmsdk_AHB_0,..., POST_KERNEL,
                        CONFIG_KERNEL_INIT_PRIORITY_DEVICE,
                        &gpio_cmsdk_AHB_drv_api_funcs);
    static void gpio_cmsdk_AHB_config_0(struct device *dev)
    {
        IRQ_CONNECT(IRQ_PORT0_ALL, CONFIG_GPIO_CMSDK_AHB_PORT0_IRQ_PRI,
                    gpio_cmsdk_AHB_isr,
                    DEVICE_GET(gpio_cmsdk_AHB_0), 0);
        irq_enable(IRQ_PORT0_ALL);
    }
    #endif /* CONFIG_GPIO_CMSDK_AHB_PORT0 */
```

### Zephyr OS exposes the GPIO API via gpio.h

- The GPIO API offers the common set of functions to access and drive one or more GPIOs:
  - Config
  - Read
  - Write
  - Callback and IRQ management
- Pinmux and GPIO drivers on Beetle act on the same IP (set of registers)
Zephyr OS Port – Drivers – UART

- Zephyr OS exposes the GPIO API via uart.h
- The UART drivers can work in two ways:
  - Interrupt Driven
  - Polling
- The UART driver supports Baudrate configuration.
- The UART driver is initialized at init level PRE_KERNEL_1 to allow early print.

```
static const struct uart_driver_api uart_cmsdk_apb_driver_api = {
    .poll_in = uart_cmsdk_apb_poll_in,
    .poll_out = uart_cmsdk_apb_poll_out,
    #ifdef CONFIG_UART_INTERRUPT_DRIVEN
        .fifo_fill = uart_cmsdk_apb_fifo_fill,
        .fifo_read = uart_cmsdk_apb_fifo_read,
        .irq_tx_enable = uart_cmsdk_apb_irq_tx_enable,
        .irq_tx_disable = uart_cmsdk_apb_irq_tx_disable,
        .irq_tx_ready = uart_cmsdk_apb_irq_tx_ready,
        .irq_rx_enable = uart_cmsdk_apb_irq_rx_enable,
        .irq_rx_disable = uart_cmsdk_apb_irq_rx_disable,
        .irq_tx_empty = uart_cmsdk_apb_irq_tx_empty,
        .irq_rx_ready = uart_cmsdk_apb_irq_rx_ready,
        .irq_err_enable = uart_cmsdk_apb_irq_err_enable,
        .irq_err_disable = uart_cmsdk_apb_irq_err_disable,
        .irq_is_pending = uart_cmsdk_apb_irq_is_pending,
        .irq_update = uart_cmsdk_apb_irq_update,
        .irq_callback_set = uart_cmsdk_apb_irq_callback_set,
    #endif
}; /* CONFIG_UART_INTERRUPT_DRIVEN */
```
Zephyr OS Port – Drivers – WDOG

- Zephyr OS exposes the watchdog API via watchdog.h.
- On Beetle the watchdog triggers an NMI interrupt.
- In order to detect it correctly CONFIG_RUNTIME_NMI needs to be enabled and a proper interrupt handler needs to be provided.
- CONFIG_RUNTIME_NMI allows to override the default NMI handler.

```c
static int wdog_cmsdk_apb_init(struct device *dev)
{
    volatile struct wdog_cmsdk_apb *wdog = WDOG_STRUCT;
    wdog_r = dev;
    /* unlock access to configuration registers */
    wdog_cmsdk_apb_unlock(dev);
    /* set default reload value */
    wdog->load = reload_s;
    #ifdef CONFIG_RUNTIME_NMI
    /* Configure the interrupts */
    _NmiHandlerSet(wdog_cmsdk_apb_isr);
    #endif
    #ifdef CONFIG_WDOG_CMSDK_APB_START_AT_BOOT
    wdog_cmsdk_apb_enable(dev);
    #endif
    return 0;
}
```

Zephyr OS exposes the watchdog API via `watchdog.h`. On Beetle the watchdog triggers an NMI interrupt. In order to detect it correctly `CONFIG_RUNTIME_NMI` needs to be enabled and a proper interrupt handler needs to be provided. `CONFIG_RUNTIME_NMI` allows to override the default NMI handler.
# Zephyr OS Port – Build a Driver

## Makefile (drivers/gpio)

```makefile
... 
obj-$(CONFIG_GPIO_CMSDK_AHB) += gpio_cmosdk_ahb.o 
... 
```

## Kconfig.cmsdk_apb (drivers/gpio)

```plaintext
menuconfig GPIO_CMSDK_AHB
    bool "ARM CMSDK (Cortex-M System Design Kit) AHB GPIO Controllers"
    depends on GPIO && SOC_SERIES_BEETLE
    default n
    help
    Enable config options to support the ARM CMSDK GPIO controllers.
    Says n if not sure.
```

## Kconfig.defconfig (v2m_beetle)

```plaintext
if GPIO
    config GPIO_CMSDK_AHB
        def_bool y
    config GPIO_CMSDK_AHB_PORT0
        def_bool y
    config GPIO_CMSDK_AHB_PORT1
        def_bool y
    config GPIO_CMSDK_AHB_PORT2
        def_bool y
    config GPIO_CMSDK_AHB_PORT3
        def_bool y
endif # GPIO
```

## v2m_beetle_defconfig (v2m_beetle)

```plaintext
... 
    # GPIOs
    CONFIG_GPIO=y
    ... 
```
Zephyr on Beetle
BSP Porting – What’s Next
BSP Porting – What’s Next

TO-DO List:

- Continue to improve the codebase.
- Enable the missing IPs.
- Complete the enablement of the Power Management.
- Enable Connectivity.
- Enhance the documentation.
How to contribute to Zephyr
Contribute to the Zephyr Project

To contribute to the Zephyr project it is required to:

- Request a Linux Foundation Account
- Clone the Zephyr Source code and start hacking
- Create a Patch from your source tree
- Verify that the Coding Style and Conventions are respected
- Submit the Change for Review via Gerrit
- Wait and hope 😊
Zephyr Source Control

- Zephyr project uses git as source control: https://gerrit.zephyrproject.org/r/#/admin/projects/zephyr

- Git can be cloned via:
git clone https://gerrit.zephyrproject.org/r/zephyr
### Zephyr Mailing List and IRC

**Mailing Lists:**
- Devel: [devel@lists.zephyrproject.org](mailto:devel@lists.zephyrproject.org)
- Users: [users@lists.zephyrproject.org](mailto:users@lists.zephyrproject.org)

**IRC Channels on irc.freenode.org:**
- [#zephyrproject](irc.freenode.org): General Zephyr Development topics
- [#zephyr-bt](irc.freenode.org): Zephyr BLE related topics
Micropython on ARM Beetle (1/3)

- Source Code:
  - https://git.linaro.org/lite/linaro-aeolus.git => Linaro Releases

- Clone the repository and you will have a structure like:
  ```
  linaro-aeolus
  ├── zephyr
  │   └── micropython
  │
  └── ...
  ```
Micropython on ARM Beetle (2/3)

- If the Zephyr SDK (0.9) is not installed in your machine:
  - $ make sdk
- To initialize the project:
  - $ make update
- To build micropython for Beetle:
  - source zephyr-environ.sh
  - ./zmake micropython BOARD=v2m_beetle
- The generate binary will look like:
  - micropython-v2m_beetle-v1.8.7-111-g300ecac-zv1.5.0-4095-ga45dd12.bin
- Download the binary into the board and reset.
Micropython on ARM Beetle (3/3)

You should see something like this on the serial port:

```
>>> print("Hello from Beetle")
Hello from Beetle
```
A script to switch on and off a led

In the micropython interpreter (>>>):

• Press CTRL+E to enter in paste mode
• Copy and paste the script
• Press CTRL+D to start running the script
• In case you want to terminate it press CTRL+C

```python
import utime
import machine

led = machine.Pin("GPIO_0", 9, machine.Pin.OUT)

while True:
    led.value(1)
    utime.sleep(0.5)
    led.value(0)
    utime.sleep(0.5)
```
Summary

- Zephyr Architecture
- Environment Setup
- Zephyr on Beetle
- How to contribute to Zephyr
- Micropython
The End

Questions?
Contacts

- Project Mailing Lists:
  - Devel: devel@lists.zephyrproject.org
  - Users: users@lists.zephyrproject.org

- IRC Channels on irc.freenode.org:
  - #zephyrproject => General Zephyr Development topics
  - #zephyr-bt => Zephyr BLE related topics

- My contacts:
  - E-mail: Vincenzo Frascino vincenzo.frascino@linaro.org vincenzo.frascino@arm.com
  - IRC: fvincenzo