Using Zephyr for hard real-time applications: motor control

Gerard Marull-Paretas
gerard@teslabs.com
9th June 2021

Outline

Introduction

Why Zephyr?

SPINNER

Conclusions

Introduction

What is an electric motor?

- An **electric motor** is a machine that **converts electrical energy** into **mechanical energy**
- Electric motors **generate torque** through the **interaction** between their **magnetic field** and their **winding currents**
- Motors can be **categorized** by their **power source type** and **internal construction**:

- Electric Motors
  - AC
    - Asynchronous
    - Synchronous
  - DC
    - Brushed
    - Brushless
      - BLDC
      - Stepper

BLDC motors

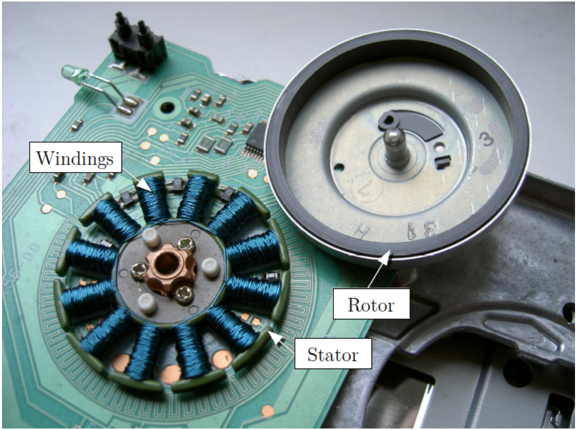

Figure: A BLDC (outer structure) from a floppy disk drive (Sebastian Koppehel, CC BY 3.0).

BLDC motors

- BLDC: a **synchronous motor** using **direct current** (DC) power supply
- If back-EMF is sinusoidal: **PMSM** (Permanent Magnet Synchronous Motor)
- **High efficiency**, **high power-to-weight ratio**, **high speed**. . .
- The **rotor contains permanent magnets** that create a **constant magnetic field**
- Driven by an **inverter** controlled by a **microcontroller**

BLDC motors

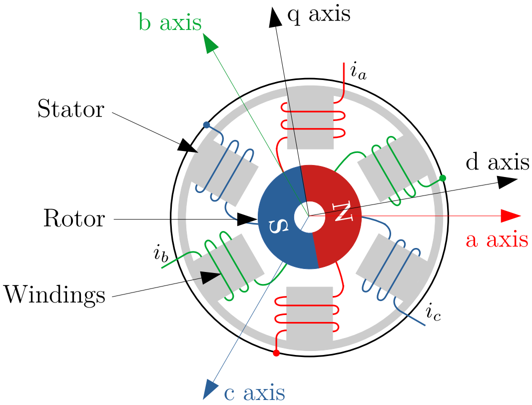

Figure: Schematic of a BLDC and its frames of reference.

How does a BLDC motor spin?

- Motor is driven by $120°$**-phased sinusoidal voltages**
- This results in a **rotating vector** in the abc frame constant in magnitude known as the **space-vector**
- **Currents** flowing through the windings will **induce a rotating magnetic field**
- The rotor **permanent magnet** will **rotate to keep aligned** with the generated field

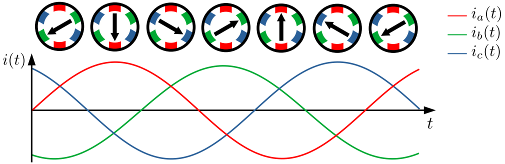

Figure: Rotating magnetic field generated by sinusoidal phase currents (original: Svjo, CC-0).

Field Oriented Control

- Field Oriented Control (FOC) is a commonly used control technique
- Operates in the **dq space**, a **stationary frame** with respect to the rotor position (DC quantities!)
- Generated **torque is proportional** to the controlled variable $\mathbf{i_q}$
- Based on a set of **transformations** (Clarke and Park) and the knowledge of the **rotor position**, usually provided by a sensor

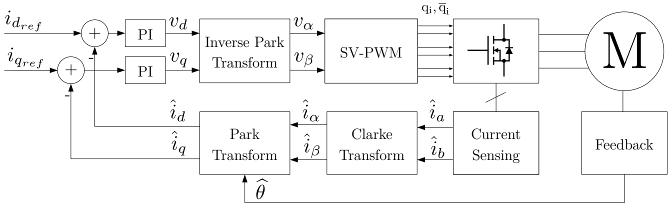

Figure: Current (torque) control with Field Oriented Control (FOC).

Why Zephyr?

Before Zephyr. . . why an RTOS?

- Motor controllers can quickly become a **complex system**:
  - Communications interface (MODBUS, CANopen. . .)
  - Additional peripherals (e.g. non-volatile memory, sensors. . .)
  - Monitoring tasks
  - . . . and more!
- **Hard to manage** everything in a bare-metal ***super-loop*** **architecture**
- An **RTOS** provides a **scalable solution**:
  - Allows **focusing on application development**
  - Program **functions** are split into **self-contained tasks**
  - **Tasks are scheduled when needed**, improving program flow and response time
  - **Shared resources** are **easier to manage**

Why Zephyr?

- Zephyr is **not only an RTOS**, it is an **ecosystem**
- **Modern build system** based on CMake
- Support for **Devicetree** and **Kconfig**
- **Vendor independent**, permissive license (Apache 2.0)
- **Best-in-class development practices**
- **Generic APIs** (e.g. serial, GPIO, I2C, SPI, sensors. . .)
- **Built-in features** relevant for motor control:
  - Direct access to vendor HALs
  - Access to CMSIS packages, e.g. DSP
  - Industrial field buses: CANopen, MODBUS
  - Advanced debugging and tracing facilities
  - . . . and more!

Devicetree

- Devicetree: a hierarchical data structure that describes hardware
- `devicetree.h` API gives access to the information from the drivers or application
- A **versatile solution** compared to *endless configuration headers*

```
&adc1 {
currsmp: currsmp {
    compatible = "st,stm32-currsmp-shunt";
    pinctrl-0 = <&adc1_in1_pa0 &adc1_in7_pc1 &adc1_in6_pc0>;

    adc-channels = <1 7 6>;
    adc-trigger = <STM32_ADC_INJ_TRIG_TIM1_TRGO>;
  };
};
```

https://docs.zephyrproject.org/latest/guides/dts/intro.html

Kconfig

- Kconfig: a language to describe **software configuration options**
  - Supports **multiple types** (bool, int. . .)
  - **Dependencies** can be specified
  - Options can be **browsed and edited interactively**
- Options **can be accessed** from both **C** code and the **build system**

```
config SPINNER_SVPWM_STM32
  bool "STM32 SV-PWM driver"
  default y if SOC_FAMILY_STM32
  select USE_STM32_LL_TIM
  select SPINNER_SVM
  select SPINNER_UTILS_STM32
  help
    Enable SV-PWM driver for STM32 SoCs
```

https://docs.zephyrproject.org/latest/guides/kconfig/index.html

Kconfig

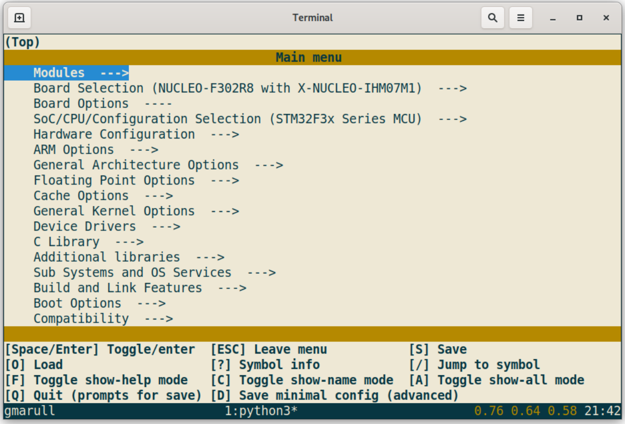

Figure: Interactive Kconfig browser.

SPINNER

What is SPINNER?

- A proof-of-concept motor control firmware based on the **Field Oriented Control** principles
- Built on top of **Zephyr**
- Implements the **current (torque) control loop** using FPU
- Provides **driver interfaces** for:
  - Feedback sensors, e.g. Halls
  - SV-PWM
  - Current sampling
- Driver implementations for **STM32** (F3xx)

`https://github.com/teslabs/spinner`

Supported boards

Figure: P-NUCLEO-IHM002 (NUCLEO-F302R8 + X-NUCLEO-IHM07M1).

SPINNER components

- **SV-PWM**: Driver responsible for synthesizing space-vector using modulated (PWM) signals
- **Current sensing**: Driver responsible for sampling motor currents and calling current control loop
- **Feedback**: Driver responsible for sensing rotor position
- **Current control loop**: Component responsible for the motor currents regulation using FOC

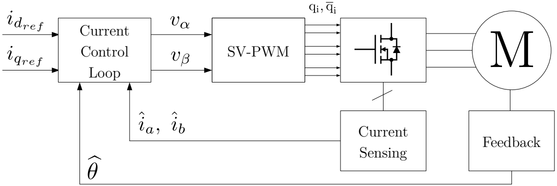

Figure: Block diagram of SPINNER core components.

Design principles

- Drivers provide a **generic interface**, allowing better **portability** and **testability**
- **Vendor HALs** (STM32 LL) are used: allows building on top of **battle-tested code**
- **Devicetree** is used to obtain all hardware description, including pinmux
- **Kconfig** is used to specify all software dependencies
- Firmware is **structured as a module**, following the Zephyr `example-application`

https://github.com/zephyrproject-rtos/example-application

Current sampling

- Driver responsible for **sampling motor phase currents**, usually by means of **shunt resistors**
- **Measurements** are **synchronized with SV-PWM**
- **Sampling rate** ranges **from 10 to 50 kHz**
- ADC **completion IRQ** calls the **current control loop**
- **Current control loop** needs to be run at a **predictable** rate

Current sampling: Zero Latency Interrupts

- Zero Latency Interrupts (ZLI) execute at the **highest priority**
- **Not affected by interrupt locking**, i.e. `irq_lock ()`
- Combined with Direct Interrupts results in ***bare-metal* like performance**
- **Cannot interoperate** with the **Kernel**
- Only supported on Cortex-M if `CONFIG_ZERO_LATENCY_IRQS` is enabled

```
ISR_DIRECT_DECLARE(irq_routine)
{
    ...
}

IRQ_DIRECT_CONNECT(irq, priority, irq_routine, IRQ_ZERO_LATENCY);
```

Current sampling: Zero Latency Interrupts

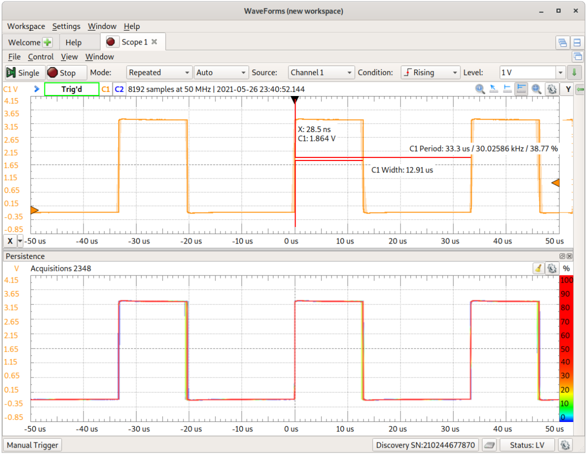

Figure: Current sampling ADC JEOS interrupt configured as ZLI calling current control loop @ 30 kHz (P-NUCLEO-IHM002).

Current sampling: Zero Latency Interrupts

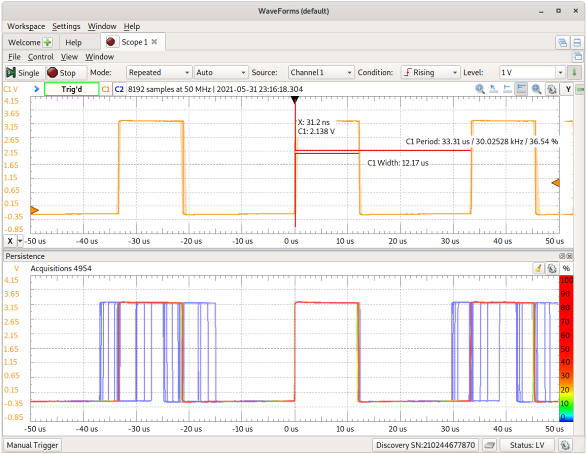

Figure: Current sampling ADC JEOS interrupt **not** configured as ZLI calling current control loop @ 30 kHz (P-NUCLEO-IHM002).

Current Loop

- Current loop **controls motor currents** (and so **torque**)
- Called after the completion of every current sampling
- Implements Field Oriented Control (FOC)
- The **most critical and resource demanding** control loop, **runs from 10 to 50 kHz**

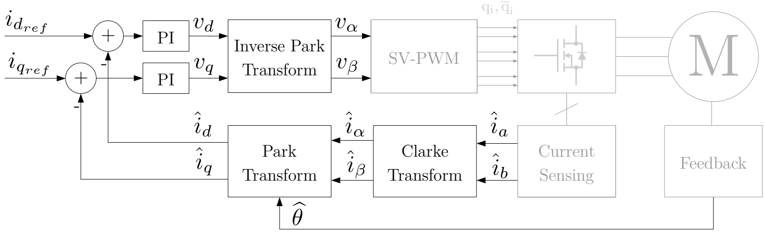

Figure: Current loop (highlighted blocks).

Current Loop: CMSIS-DSP

- CMSIS-DSP is a **suite of digital signal processing (DSP) functions** for use on Cortex-M and Cortex-A processors.
- Provides all **necessary functions** to implement the **current control loop**
- **Available on Zephyr** as a module!
- Used functionality can be enabled using `CONFIG_CMSIS_DSP_*`

Current Loop: CMSIS-DSP

```c
/* compute sin and cos of electrical angle */
arm_sin_cos_f32(eang, &sin_eang, &cos_eang);

/* i_a, i_b -> i_alpha, i_beta */
arm_clarke_f32(i_a, i_b, &i_alpha, &i_beta);
/* i_alpha, i_beta -> i_q, i_d */
arm_park_f32(i_alpha, i_beta, &i_d, &i_q, sin_eang, cos_eang);

/* PI (i_d, i_q -> v_d, v_q) */
v_d = arm_pid_f32(&pid_id, id_ref - i_d);
v_q = arm_pid_f32(&pid_iq, iq_ref - i_q);

/* v_d, v_q -> v_alpha, v_beta */
arm_inv_park_f32(v_d, v_q, &v_alpha, &v_beta, sin_eang, cos_eang);
```

https://arm-software.github.io/CMSIS_5/DSP/html/index.html

STM32 MCSDK 5.Y.1 comparison

- MCSDK: **official** STM32 motor control firmware
- Uses **fixed point arithmetic**
- Offers **more features** (sensorless, speed control, etc.)

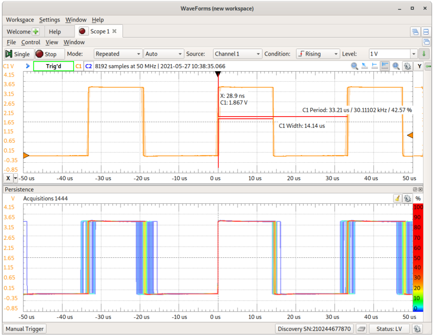

Figure: STM32 MCSDK 5.Y.1 current control loop @ 30 kHz (P-NUCLEO-IHM002).

Conclusions

Conclusions

- Zephyr allows **focusing on application development**
- Zephyr gives access to **powerful and modern tools**: Devicetree, Kconfig, CMake
- Zephyr comes with ***batteries included***, e.g. CMSIS-DSP
- Zephyr represents a **cultural-shift** on how development is done in the embedded industry
- **Zero Latency Interrupts (ZLI)** allow **bare-metal like performance**
- Access to **vendor HALs** is a **key feature** to speed up driver development
- Zephyr has a **great and supportive community**!

THANK YOU!

Questions?

https://github.com/teslabs/spinner-zds-2021

https://github.com/teslabs/spinner

https://teslabs.github.io/spinner