



Design of Prototyping Control Unit for BLDC/PMSM Drives and Piezoelectric Actuators

Květoslav Belda¹  · Pavel Píša^{1,2}  · Štěpán Pressl^{1,2} 

Received: 21 November 2025 / Accepted: 13 January 2026
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2026

Abstract

This paper addresses the challenges of creating a suitable control unit for experimental research and rapid control prototyping for drive systems in mechatronics and robotics. These systems are integrated into modern transport machines and smart production lines that form advanced cyber-physical factories. The research uses open-software and open-hardware solutions. The proposed concept deals with control and power hardware, the NuttX RTOS operating system and the pysimCoder text and block programming environment. The concept is mainly intended for modern, high dynamic brushless direct current and permanent magnet synchronous motor drives and innovative piezoelectric actuators. The paper involves design of custom printed circuits boards, and discusses the used microcontroller, power stage and necessary interfaces for peripherals. The paper also discusses a novel approach to remote tuning and firmware updates including generated code of controller applications, which was integrated into pysimCoder and NuttX RTOS as part of the development. The features of the developed unit are demonstrated by measured data obtained from real experiments with permanent magnet synchronous motor BLWR233D-36V-4000 and piezoelectric actuator PL140.

Keywords BLDC/PMSM drives · Motion control · Mechatronic systems · PysimCoder · Real-time generated code · NuttX RTOS · Power supply · Piezoelectric bender actuators

Introduction

Common industrial and home applications such as robot manipulators, production lines and machines, unmanned vehicles and others depend on controlling their motion. Consequently, motion controllers are widely used in industry. The market offer ranges from controllers working only with a specific actuator (motor) with closed firmware to more affordable and versatile controllers.

In initial stages, for fast application design, generic prototyping platforms are needful for developing or researching new control algorithms. Involved controllers are required to enable rapid control prototyping using model-based design, capable of running the designed control algorithms [1]. In this paper, we propose a modular and scalable, yet relatively low-cost, embedded solution with a strong emphasis on running control applications built using model-based design tools that generate real-time code.

This paper extends results presented in [2]. Introduction parts discuss the existing solutions and main design parameters. The paper focuses on hardware design and implementation towards developing novel control strategies (Sects. 2 and 3). Subsequently, the adjustment to the NuttX RTOS alongside the pysimCoder suite are presented (Sect. 4). The proposed solution allows easy connection of this unit from both Linux and Windows. Due to the required computations and low latencies in the control applications, the real-time properties of the NuttX RTOS are analysed as well. Finally, specific control applications are demonstrated in pysimCoder (Sect. 5). It also includes a new interface for external sensors, e.g. for sensing the deflection of piezoelectric actuators.

Květoslav Belda
belda@utia.cas.cz

Pavel Píša
pisa@fel.cvut.cz

Štěpán Pressl
pressste@fel.cvut.cz

¹ Department of Adaptive Systems, Institute of Information Theory and Automation, The Czech Academy of Sciences, Pod Vodárenskou věží 4, 182 00 Prague 8, Czech Republic

² Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Karlovo náměstí 13, 120 00 Prague 2, Czech Republic

Existing Tools and Platforms

Currently, there are specific solutions available on the market, such as the ODrive [3], VESC [4] and SOLO [5] controllers that focus on the control of Brushless Direct Current and Permanent Magnet Synchronous Motor (BLDC/PMSM) drives. These controllers show a number of similarities:

- (i) using mostly the same interfaces, such as UART, SPI, I2C, CAN, and USB; in addition to providing advanced and user-friendly rapid prototyping in general; and Ethernet interface offer, enabling a universal input/output interface useful for convenient remote data logging and configuration, even from around the world;
- (ii) stand-alone API in each controller to configure control algorithms; on the proposed platform our intention is to provide a non-restrictive environment where it would be possible to execute generated code of control algorithms, instead of creating your own application-specific API; finally
- (iii) the lack of more than three phases per axis, making control of stepper motors or other types of actuators not possible; our goal is to bridge this undesirable similarity/feature.

Several solutions for *Rapid Control Prototyping* already exist, such as Speedgoat [6] and dSPACE [7]. However, these systems are expensive and closely tied to MATLAB/Simulink. In contrast, we propose a unit that uses an open and freely available code generation tool, *pysimCoder* generating code for the open-source NuttX RTOS, offering a unique, low-cost solution for real-time rapid control prototyping.

Proposed Platform Specification

The presented platform design is guided to be fully open source and open hardware based, to remain maintainable and technologically sustainable for every user. To ensure universality for wide use, many communication interfaces are needed to enable communication with sensors, connection to a microcontroller, and logging purposes, such as Ethernet, I²C, SPI, USB, CAN, etc., as well as functional parts such as *Pulse Width Modulation*, *Analog-to-Digital Converter* and *Timer* peripherals for the specific design of *field oriented control* of BLDC/PMSM drives. Because the generated code increases the computational overhead for such tasks, it is necessary to choose a sufficiently powerful microcontroller with a powerful core.

Our work focuses primarily on controlling BLDC/PMSM, Stepper motors (even with feedback sensors) and brushed DC motors, especially for robotic applications [8]. However, the platform should also be applicable for special actuators e.g. piezoelectric and others in mechatronics [9]. The controller should be able to drive at least two stepper motors, therefore eight controllable phases are needed. Eight phases are also enough for BLDC/PMSM motors with more than three phases. With a higher number of phases, a used motor can be still operated when one or more phases are failed. For these advantages, the multi-phase BLDC motor is taken a good candidate for energy saving or military and space applications [10].

The controller is intended as a generic prototyping platform with a large number of interfaces, size and weight are not an issue. Once the control algorithms are designed and validated on this platform, the hardware can be redesigned to meet certain criteria (size, weight, safety requirements or spatial compliance), while maintaining the same algorithms and functionality. Despite all the proposed features, we strive for an affordable solution within the motion controllers, such as the ODrive.

Finally, to make the generated code portable across different platforms, a *real-time operating system* (RTOS) is considered. It provides a runtime environment for multi-threaded applications, network stacks, and a unified API for accessing hardware-dependent peripherals. In this paper, NuttX RTOS is chosen as a suitable operating system and *pysimCoder* as an experimental but open source code generation tool, which is able to generate NuttX-compatible code from block diagrams and remotely tune its internal parameters over the network using the Silicon Heaven protocol [11].

Hardware Design

Hardware (HW) design is inspired by the LX-RoCon controller [12]. One of the design rules is to divide the motion controller into a *Micro-Controller Unit (MCU)* board and a *Power stage* board as two printed circuit boards (PCBs). The complete motion control unit then consists of these two interconnected boards. This design rule allows to swap the less complex *Power stage* board for a different one while keeping the more complex *MCU* board the same. The rule has proven useful based on experience gained during the design. Since the *MCU* board accommodates most of the used connectors, it is placed above the power board which also serves as the 5 V power supply for the *MCU* board as well. HW has undergone several modifications and fixes during the preparation of this article. The specific changes will be explicitly mentioned.

Interfaces and Peripherals

For the specification, design and development of a motion control unit, the features for selecting a microcontroller, intended peripherals, and their use cases can be summarized as follows: TTL UART for Simple console user interface; RS232, RS422 or RS485 for communication with other devices, sensor data acquisition; (Q)SPI for Ext. SPI memory, sensor data acquisition; I²C for Ext. I²C memory, sensor data acquisition; Counter for position measurement from quadrature encoders; JTAG or SWD for program flashing and debugging; CAN for communication with other devices; Ethernet for data logging, remote access, *pysimCoder* parameter tuning; SD slot for data logging; PWMs for Control of power switches; ADCs for current measurement for motor *field oriented control* (FOC) and GPIOs for reading from hall sensors.

The motion controller should be able to simultaneously drive at least two stepper motors, as it is intended for stationary robot topologies of the SCARA type or Cartesian robot and mobile robots with axles of two independently driven wheels. As for motors, a stepper motor needs 4 half-bridge switches. So a total of 8 half-bridge switches are needed. BLDC/PMSM requires 3 half-bridge switches to be controlled and a DC brushed motor requires 2 half-bridge switches (forming a H-bridge). Choosing 8 half-bridge switches is enough for controlling 2 BLDC/PMSM and 4 DC brushed motors as well. Furthermore, at least 8 PWM outputs for power switches and at least 8 ADC channels for current measurement are necessary with two counter peripherals for encoder pulse counting and all the communication interfaces in paragraph above. Finally, all components should be complying with the industrial temperature range of at least -40 to 85°C .

Unit Microcontroller

A suitable microcontroller proceeds from an analysis of the features above. As a suitable pattern, a Microchip family SAM E70/S70/V70/V71 was selected. Specifically, the microcontroller *ATSAMV71Q21B* [13], was chosen. Similar microcontroller alternatives are e.g. series STMicroelectronics STM32H7 or the NXP imxRT. According to the microcontroller name, the control unit has the label *SaMoCon*. In the newer hardware revision, the microcontroller was replaced with a newer and cheaper pin compatible PIC32CZ2051CA70 microcontroller. Furthermore, these MCUs will be abbreviated as SAMV71 or PIC32CZ. The selected microcontroller core is ARM Cortex M7 equipped with support for single-precision and double-precision floating point unit, making it feasible to use alongside *pysimCoder* generated code with *double* 64-bit type in

calculations. The core is running at 300 MHz and features 2 MB of Flash memory, alongside 384 kB (512 kB in case of PIC32CZ) of SRAM.

The given microcontroller is equipped with all necessary peripherals, i.e. Ethernet, two PWM peripherals featuring 8 channels in total with complementary outputs with configurable dead zones, 2 *Analog Front End Controller* (AFEC) peripherals each having up to 11 AD channels at disposal and 4 TC (Timer/Counter) peripherals with encoder counter logic. Besides that, CAN peripherals, the HSMCI peripheral for SD interfacing and many UARTs, SPIs, TWIHSs (Microchip I²C) are present too.

Hardware Implementation

In the initial phase of prototyping the power stage board, Infineon IFX007T half-bridge switches [14] were selected, see Fig. 1. Since the maximum rated voltage is 40 V, this first power board has a nominal rating of 24 V. To reduce the component cost, a common ground is shared with the MCU board, as no galvanic isolators are needed. IFX007T is an all-in-one switch featuring two N power-switching MOSFETs alongside gate drivers. It includes built-in high and low-side current limitations alongside over-temperature protection. This makes switching very easy as it only requires one PWM control signal (IN) and an inhibitor input (INH) controlled by a GPIO.

The half-bridge also has an IS (current-sense) output whose current is proportional to the current flowing through the high-side transistor to the load. Subsequently, the high-side current can be determined through measurement of the voltage drop across the connected resistor. The IS pin also serves as a fault indication in case of over-current or high temperature. IS output current characteristics are shown in Fig. 2.

Although the high-side current can be measured using the IS output, it was decided to measure the current of the transistor low-side using a shunt connected to ground. However, the IS high-side value was used as an indicator of average current by connecting a parallel capacitor, and the indicator of a fault.

As shown in the Fig. 2, the fault output current is bigger than the expected current during normal operation. This way, a fault voltage drop can be compared with a reference on a comparator. Therefore the comparator with open drain/collector outputs was chosen for easy interconnection with other comparators. For more accurate measurement of the motor winding current, a shunt is used on the low side.

The scheme for current measurement and fault generation is shown in Fig. 3. If the voltage drop across resistor R_{is} is greater than V_{ref} , the comparator output decreases.

Fig. 4 Upper control MCU board with peripherals [2]

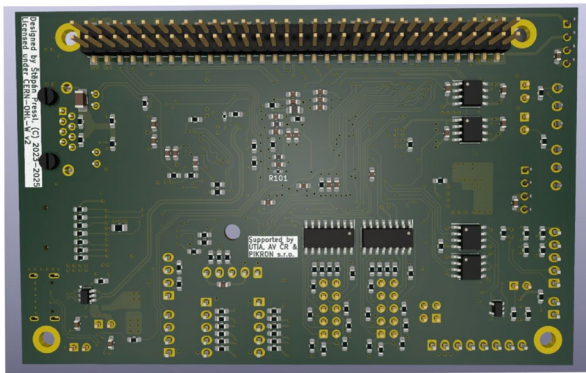
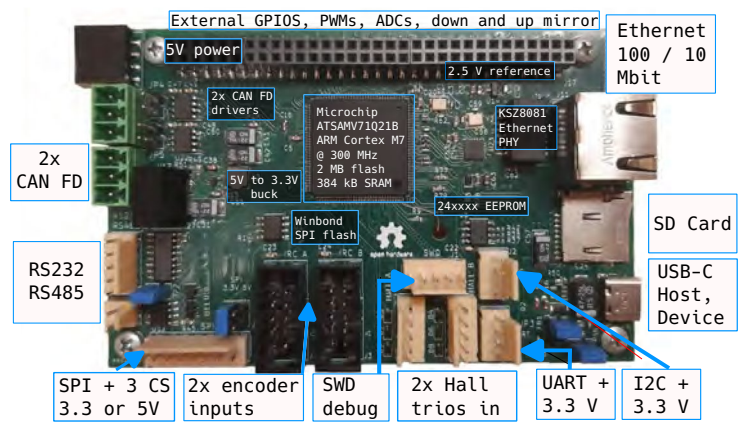


Fig. 5 Optimised PCB rear contact layout

$$V_{LS} = \frac{R_4}{R_3} R_s I_{RS} + V_{Bias}, \tag{2}$$

where I_{IS} is the current sourced by the IFX007T IS pin and I_{RS} is the current passing through the R_S shunt, which is in fact the low-side current.

MCU Board

Let us consider the upper control *MCU board*, with all the peripherals shown in Fig. 4. The connection of two boards is placed on one of the longer sides of the PCB. The optimised design of PCB rear contact layout is shown in Fig. 5. Extra GPIOs, SPI, and I²C are useful for future extensions. There is another connector on top of the MCU board with the same mirrored pinout as the lower interconnection which can be used for expanding boards placed above the MCU board and as an oscilloscope probing place. The following sections will focus on peripherals connected to separate peripheral connectors on the MCU board, referred to as *main peripherals*, while peripherals connected to the interconnect/expansion connector will be referred to as the *extra peripherals*.

Table 1 Routed peripherals [2]

SaMoCon peripheral	SAMV7 peripheral
2× CAN	MCAN0, MCAN1
SPI main & extra	SPI0, SPI1
TTL Console	UART3
RS232/RS485	USART2
Ethernet	GMAC in RMI mode
USB-C	HSUSB
2× IRC	TC[0,2] (Timer/Counter)
SD Card	HSMCI
H PWM out, Motor A	PWM0 CH0-3
L PWM out, Motor A	PWM0 CH0-3
H PWM out, Motor B	PWM1 CH0, CH1, (CH2), CH3
L PWM out, Motor B	PWM1 CH0-3
8 ADC channel, Motor A	AFEC0
8 ADC channel, Motor B	AFEC1
3 extra ADC channels	AFEC1
2× 3 Hall inputs	GPIOs on the same PIO
I ² C main & extra	TWIHS0, TWIHS1

Microcontroller Pinout

First, let us note that there exist many pin configurations due to the existence of alternative functions of microcontroller pins. However, due to the large number of peripherals required, conflicts began to occur when two peripheral outputs or inputs were on the same pin. See Table 1 for routed peripherals and the corresponding SAMV71 peripheral which provides sufficient insight into the expected number of pins.

Each IFX007T half-bridge is controlled by one PWM H (high-side) signal and one GPIO pin. Even though we need 8 PWM H outputs, we also route all remaining PWM L (low-side) outputs, because promising alternate gate drivers require complementary PWM outputs. In the case of IFX007T, the L output can be configured as a simple output pin. In the first revision, PWM1 CH2 could not be routed due to pin collisions with the HSMCI peripheral. This limitation was resolved in the newer revision by introducing jumper resistors, which allow selecting either PWM1

CH2 operation (with the SD card interfaced via the slower SPI mode) or full SDIO mode without the complementary PWM operation.

MCU-Board Voltage Supplies

All components on this PCB are powered from the 5 V or 3.3 V rail. MCU board is designed to be powered from the power board through the interconnection. The 5V rail can also be supplied from the USB-C connector. A buck regulator (Texas Instruments TPS562207) is used to convert 5V to 3.3 V. To ensure electrical safety, the CAN and RS232/RS485 peripherals are galvanically isolated, so 5 V to 5 V small power supplies are used. Also, a stable 2.5 V reference (Microchip MCP1525) was added for analog measurements.

Ethernet

A Physical Layer (PHY) component was connected to the microcontroller GMAC peripheral. The Microchip KSZ8081 IC was used, working in the RMII mode with specific bit rates up to 100 Mb/s. The design uses the recommendations from the datasheet [15] with compatible magnetics. Communication activity is indicated by two LED. The Ethernet interface is powerful feature allowing access to the unit from anywhere in the world. A related proposed optimised firmware update using the advantage of Ethernet interface is mentioned in Sect. 4.1.2.

USB

The USB-C pins CC1 and CC2 are connected to the Texas Instruments TUSB321 integrated circuit that negotiates host or device operation, given voltage levels at certain pins. The data pins are connected to the HSUSB SAMV71 peripheral. The board was designed to act as a host or a device, using an AP2171W power switch. If the controller acts

as a host, the enable pin (EN) is set high, enabling the power switch. If over-current happens, an open collector FLG output is used to indicate an error. If connected to a USB power source, the current passes through a Schottky diode with a low voltage drop and bypasses the USB power switch, see Fig. 6.

Other Communication Interfaces

The MCU board contains two CAN-FD peripherals with MCP2562FD transceivers. The board features a circuit that either functions as the RS232 or the RS485 interface. The RS232 interface uses a MAX232 converter and the RS485 uses the ST485 converter. CAN and RS interfaces are both isolated by the ADuM1201AR isolators.

The board is also equipped with main and extra routed TWIHS (I²C) interfaces. The main TWIHS peripheral has a Microchip 24AA64 EEPROM connected and the signals are routed to a 4 pin Molex connector alongside 3.3 V and GND. There is an another separate 4-pin Molex connector for a basic LVTTL console user interface, with 3.3 V, GND and RX and TX signals.

Sensing of Physical Quantities

The board is equipped with two connectors for incremental encoders and triple digital Hall BLDC motor sector sensors. The optical encoder connector is compatible with the HEDL encoder series with a 2×5 pin socket. The HEDL series connector incorporates differential two optical phases, a differential zero-cross index, grounding and a 5 V power output too.

The differential RS-422 encoder signals are converted to LVTTTL standard (3.3 V) by the Texas Instruments AM26LV32 receiver. The so-called *mark* signal is also used, which acts as another user-defined useful synchronisation signal. The decoder wiring scheme of the circuit is shown in Fig. 7.

Fig. 6 USB power components [2]

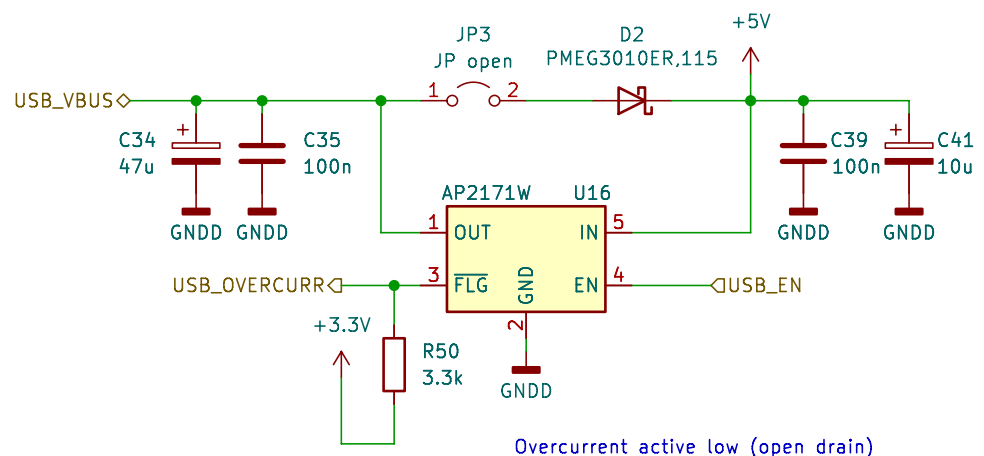


Fig. 7 Routed optical encoder with HEDL series pinout [2]

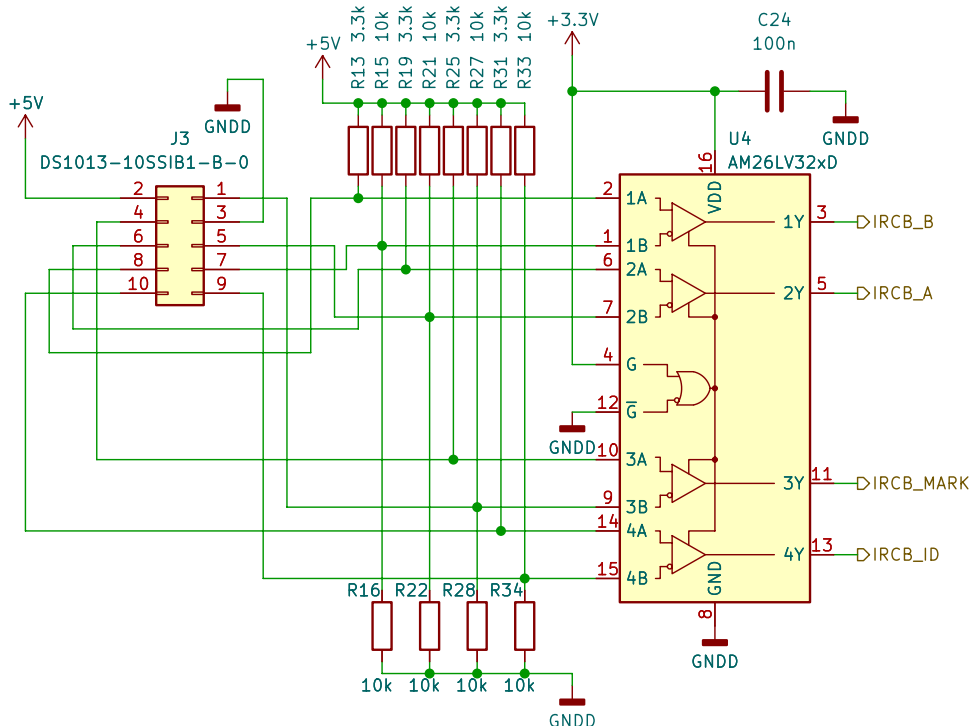
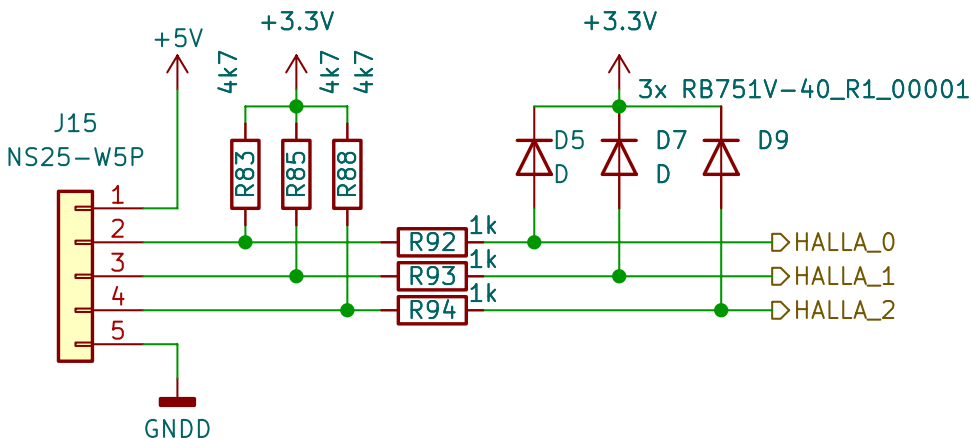


Fig. 8 Open drain/collector output: the output is 3.3 V or GND due to the pull-up to 3.3 V. Push-pull: the voltage limitation is done by shorting the output to 3.3 V through diodes. The shorting current is limited by series resistors [2]



As the voltage level for the inverted signals in the disconnected state is defined by the voltage divider, an encoder with single-ended outputs can be connected to the positive inputs. All inputs have a pull-up in case of open drain/collector outputs. The decoded signals from the receiver are connected to the SAMV71 TC0 and TC2 (*Timer/Counter*) peripherals.

The Hall outputs can be read by a GPIO pin. Our implementation uses a standard 5-pin Molex connector (5 V power, GND, and 3 Hall outputs). In general, the outputs can be either push-pull or open-drain. However, a 5 V push-pull output could potentially destroy the microcontroller, which means that precautions must be taken. The way the Hall inputs are routed. It is shown in Fig. 8.

Board Interconnection Pinout

The boards are connected using a standard 2×32 header pin connector with a pitch of 2.54 mm, see Fig. 5. The MCU board contains a pin connector header facing down to the power board and a socket facing up. The pinout of these connectors is the same, only mirrored, and is SMD type, which allows us to route traces in the inner layers.

Specific pins provide a 5 V supply voltage and the power digital ground to power the MCU board. Generally, the power board may contain 3.3 V powered circuits, so we need to route 3.3 V too, as well as the 2.5 V reference.

The bottom board may also contain analog circuitry so we route the analog ground as well. Analog and digital ground

connections are made on the MCU board near the ADC reference and ground pins.

All 16 complementary PWM outputs must be routed along the 16 ADC channels (sensing high and low sides of the IFX007) for two motors. One more I²C, GPIOs and SPI or 3 extra ADC channels are also routed.

Power Stage Board

As discussed in the analysis, the power board also serves for voltage conversion to 5 V. The rated current per phase is at least 10 A continuous, with a possible peak current of 20 A.

The board must perform current measurement and amplification. These analog signals are then processed by the MCU board located above. The power board is shown in Fig. 9.

The power to the half-bridge switches is routed on the top side of the PCB, while the outputs are routed on the back. The IFX007's exposed solder pad output is connected to a large copper surface, on which a heat sink can be placed to improve thermal dissipation of the switches.

Power Components

The power board is equipped with a 3-pin power connector, with one pin being a common ground pin, the others are the positive voltage rails, one for the MCU board, and the other for the motors. Each positive rail has an over-voltage protection diode. The implemented buck regulator IC is AOZ1284, used according to the recommendations in the datasheet. The total power consumption of the MCU board is assumed around 0.5 A, the USB power switch of the MCU board can handle a maximum continuous current of 1 A, and we suppose that the expansion board will

consume another 1 A. The IC provides up to 4 A, which is sufficient for our requirements.

All analog circuits powered from 5 V or 3.3 V (provided by the MCU board) are first filtered by an LC filter. The power rail for the IFX007 switches contains decoupling electrolytic capacitors.

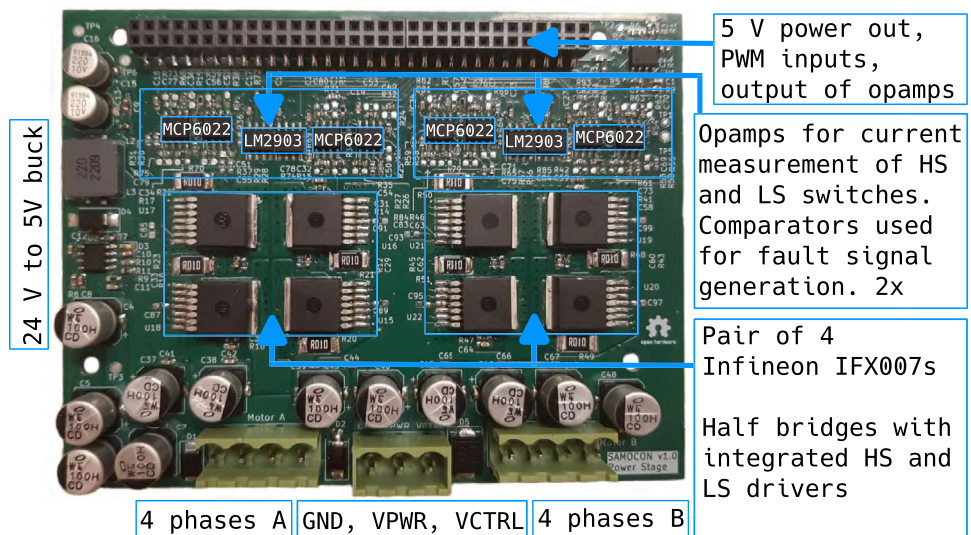
Low Side Shunt Current Measurement

Let formula (2) be considered and $V_{Bias} = 1.25 V$ at the centre of the voltage reference. This voltage is obtained by choosing a 1:1 voltage divider with a buffer to the lower output impedance of the divider. R_s was chosen to be 10 m Ω because higher resistance values would lead to big power losses. Such power loss at 10 A is 1 W, hence a 3 W 2512 resistor has been chosen. To use the whole ADC bandwidth, we amplify the voltage by at least 10 \times . This can be achieved by setting $R_3 = 1 k\Omega$ and $R_4 = 10 k\Omega$.

Current-Sense Measurement

Since the IFX007T IS (current-sense) output indicates a fault, we need to set the threshold value first which should equal to the point where $I = 20 A$. Referring to Table 11 in [14], the maximum analog sense current in fault condition is $I_{IS(lim)} = 6.1 mA$. The voltage drop across R_{IS} must certainly not exceed 3.3 V to avoid damage to the operational amplifier powered from the 3.3 V rail. We choose a resistor $R_{IS} = 510 \Omega$ from the E24 series. The maximum voltage drop is $V_{IS(lim)} = 3.111 V < 3.3 V$. The differential current sensing ratio in the static on state is on average [14] $\frac{dI_L}{dI_S} = 19,500$. Then, a threshold value for the comparator at $I = 20A$ can be computed as follows:

Fig. 9 Power stage board [2]



$$V_{th} = \frac{20}{19,500} 510 \text{ V} = 0.52 \text{ V.} \quad (3)$$

Since the differential ratio changes significantly and also has a non-negligible offset, we set $V_{th} = 0.655 \text{ V}$ using a divider made out of $2.2 \text{ k}\Omega$ and $6.2 \text{ k}\Omega$ resistors. This compare value is again amplified by a buffer to lower the output impedance.

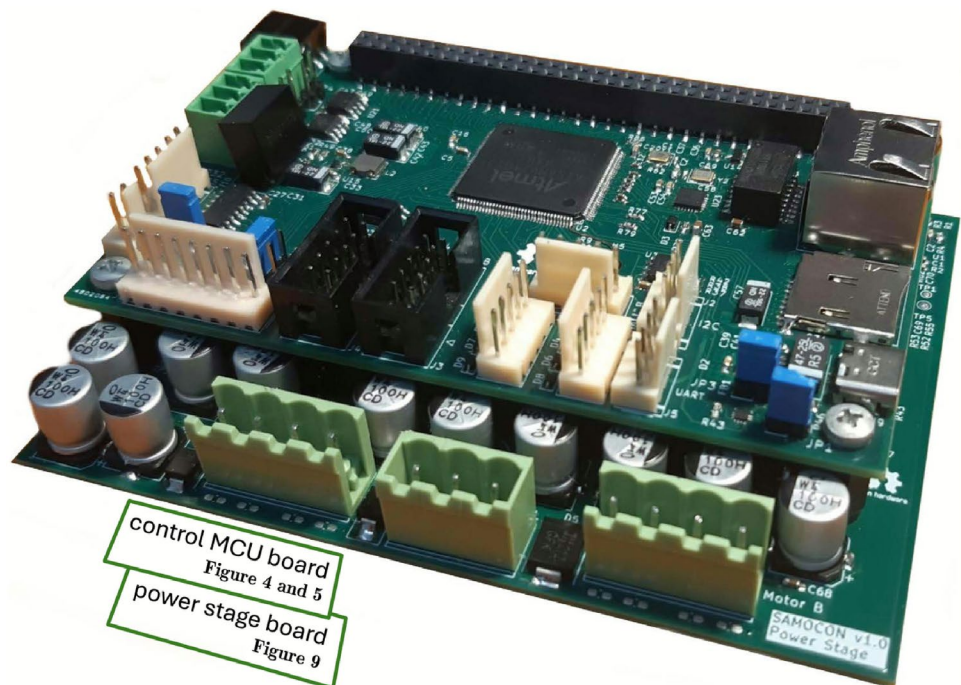
The last value to set is C_{is} . This value should be in the order of hundreds of pF, so that only high-frequency noise is filtered. Unfortunately, during testing, we noticed a high fault rate at the comparator output due to the fast voltage spikes on R_{is} , which were always caused after the PWM signal was switching off. This behaviour is resolved by higher capacity $C_{is} = 22 \text{ nF}$, which is a compromise between spike suppression and fast voltage rise.

Analog Components

The Used operational amplifiers are Microchip MCP6022, low offset and rail-to-rail operational amplifiers. The used comparator is Texas Instruments LM2903, a dual differential comparator with open-drain outputs. This means all comparators from 4 half bridges can be connected together. If at least one switch faults, the line goes low.

The complete motion control unit is shown in Fig. 10, assembled from MCU and power boards. This concludes the HW design and implementation issue.

Fig. 10 Connected MCU and power stage boards forming the controller



Software Implementation

This section briefly introduces NuttX and its adaptation to *SaMoCon* hardware and the *pysimCoder* suite of tools for developing *pysimCoder* applications.

NuttX is an open-source real-time operating system (RTOS) written in C language and released under Apache 2.0 license. The project NuttX was introduced by Gregory Nutt in 2007, [16]. Currently, the project is maintained on GitHub [17]. The main governing standards in NuttX are Portable Operating System Interface (POSIX) and ANSI standards [18]. It should be noted that OpenVela OS based on NuttX has passed the PSE52 Realtime Controller 1003.13-2003 System compliance certification [19]. However, *SaMoCon* is intended for research experiments and rapid prototyping; therefore, compliance with safety certifications, such as IEC 61508, has not been considered at this stage.

As NuttX strives to comply with POSIX standards [20], all interaction with the microcontroller peripherals is done by accessing the peripherals' registered `/dev` files, for instance an AD converter may be registered as `/dev/adc0`. Files are accessed using system calls such as `open`, `close`, `read` or `write`. Since different peripherals classes (ADC, encoder, PWM, etc.) do not share many similarities with each other, device specific `ioctl` calls are commonly used to change basic device parameters.

The project directory structure is similar that of the *Linux Kernel* [21], as is the configuration and building process. The project must first be configured by selecting `Kconfig` options and then built using `make` or `CMake`.

NuttX Startup and Operation on Control Unit

A custom BSP (*board support package*) is required to build NuttX. The `samv71-xult` BSP for the *SAMV71 Xplained Ultra Evaluation Kit* was used as a base. Several source files have been added to initialize and register the peripherals, the most important are AD converters (`/dev/adc0, 1`), PWM (`/dev/pwm0, 1`), quadrature decoders of rotary encoders (`/dev/qe0, 1`) and GPIOs for reading HALL signal and activating IFX007T INH. Furthermore, the NuttX build needs to be configured so that all peripherals work properly, i.e. the console driver for output to the SAMV71 UART3 peripheral; SAMV71 GMAC peripheral for communication with the KSZ8081 PHY and enabling the UDP/IP and TCP/IP, along with auxiliary applications such as `ping` or `telnet`. It was also necessary to configure the SAMV71 quadrature decoder, PWM and ADC drivers. Changes have been made to the SAMV71 quadrature decoder and PWM drivers, which have also made it to the GitHub mainline repository.

By default, a NuttX configuration uses a periodic timer interrupt that controls all system timing. The timer is provided by architecture-specific code that calls to NuttX at a rate controlled by `CONFIG_USEC_PER_TICK`. The default value of `CONFIG_USEC_PER_TICK` is 10^4 microseconds which corresponds to a timer interrupt rate of 100 Hz ([22], Tickless OS).

As all POSIX time functions in NuttX (e.g. `clock_gettime`, `clock_nanosleep`) are dependent only on the interrupt rate, the usage of the Tickless OS mode is desirable because it introduces a *free-running timer*, where each counter increment corresponds to the smallest

measurable time slice. If time slice is small, high precision delays can be made, useful in a fast periodic sampling of the `pysimCoder`-generated real-time applications. Another solution is to increase the interrupt rate, but such a configuration could introduce a lot of potential context switching overhead.

As mentioned and discussed in [2] (Sect. 5: `pysimCoder` Applications), all NuttX builds using the *Tickless OS* mode were incapable of yielding sampling frequencies above 1 kHz, making the model-based design impossible for more complicated systems. Fortunately, we can utilise a timer registered as a device (e.g. `/dev/timer1`). Using NuttX-specific `ioctl` calls, the timer can be configured so that the `poll` call on the timer device file descriptor stops blocking due to the set `POLLIN` flag every configured period. Therefore, polling the timer device's file descriptor can be used to clock the sampling frequency of the real-time application. Such a method is currently used in `pysimCoder`.

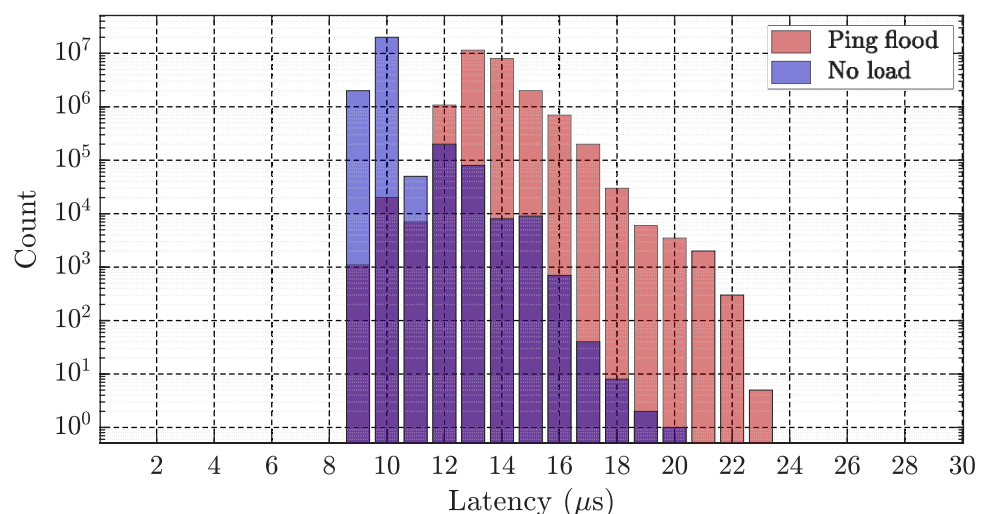
NuttX Realtime Benchmark

To prove that NuttX is suitable for use in control applications, an equivalent of Linux `rt-tests` `cyclictest` utility was written to test its realtime capabilities. The simple description of this benchmark utility is described in [23] (4.1.1). The `cyclictest` utility measures the thread latency by polling the device timer and measuring the time since the last timeout using the NuttX-specific `ioctl` calls. The port source code can be found here [24].

The Fig. 11 shows the histogram latencies measured using this command:

```
cyclictest -p 150 -T /dev/timer1 -m 1 -n 1 -h 50 -D 3600 -i 50.
```

Fig. 11 The thread latency histogram for two scenarios: no load and ping flood



The test thread priority was set to 150 (higher than the priority of other tasks), the test ran for 1 h and the thread was supposed to wake up every 50 microseconds. Two test scenarios were performed, one without load and one with flooding the target hardware (using the `ping -f <target-ip>` command) from a PC on the local network.

Figure 11 shows that ping-flood causes higher thread wake-up latencies. Despite the high load, the worst measured latency was 23 microseconds. Therefore, the sampling rates are substantially higher than 1 kHz. This proves that NuttX is a suitable OS for use with model-based applications where higher sampling rates are required.

Firmware Update Over the Network

For *Rapid Control prototyping*, it is essential to have a practical firmware updater. To meet this requirement, we have developed an updater that uses the `nxboot` bootloader [25] and the Silicon Heaven (SHV) infrastructure [26], a protocol that enables *remote procedure calls* (RPC) over TCP/IP [27].

The bootloader utilises recovery and update partitions, residing on an external SPI Windows Flash. The update application implements a *file node* in the context of Silicon Heaven protocol [28] with these remotely called methods: `write`, `stat` and `crc`. The `stat` method returns a list of the size of the file, the page size of the flash memory (for efficient write accesses), and the maximum size of one write command. Each `write` command that is received contains the raw data to be written and a file offset to write data to.

This method is used to write data to the update partition. The `crc` method returns the calculated IEEE 802.3 Cyclic Redundancy Check (CRC) value over the specified range in the file. Besides the file node methods, a `reset` method is implemented that restarts the device.

The firmware update process is straightforward. A separate thread is created to establish connection with the Silicon Heaven broker. The user with a new firmware image connects to the Silicon Heaven broker, performs the `stat` call on the target device, and then performs multiple writes using `write` calls. After all successful calls, the user performs a `crc` call over the whole file range (bytes [0, size - 1]) and if the calculated CRC value is correct, the user calls the `reset` method. After the restart, the `nxboot` bootloader checks the image in the update partition, copies the firmware located in the MCU flash to the recovery partitions, checks the image in the update partition, and flashes the MCU flash. If the application boots and functions correctly, the user (or an automated script) must confirm the

image, otherwise `nxboot` restores MCU Flash firmware from the recovery partition at the next boot as mentioned.

PysimCoder for User Real-Time Code Generation

PysimCoder is an open-source rapid prototyping application – environment consisting of block diagram editor and real time code generator for Python. It is designed by Professor Roberto Bucher from the University of Applied Sciences and Arts of Southern Switzerland, [29].

PysimCoder can run in any Linux environment or in Windows using Windows Subsystem for Linux 2 (WSL 2). WSL in general is a feature of Windows that allows the user to run a Linux environment on Windows machine, without the need for a separate virtual machine or dual booting. WSL is designed to provide a seamless and productive experience for developers who want to use both Windows and Linux at the same time. Specifically, WSL 2 is the default distro type when installing a Linux distribution. WSL 2 uses virtualization technology to run a Linux kernel inside of a lightweight utility virtual machine (VM). Linux distributions run as isolated containers inside of the WSL 2 managed VM. Linux distributions running via WSL 2 will share the same network namespace, device tree (other than `/dev/pts`), CPU/Kernel/Memory/Swap, `/init` binary, but have their own PID namespace, Mount namespace, User namespace, Cgroup namespace, and `init` process. WSL 2 increases file system performance and adds full system call compatibility in comparison to the WSL 1 architecture.

Although `pysimCoder`'s main purpose is to generate RT control code, it can also perform simple simulations combining continuous-time and discrete-time blocks. It uses an extended python-control library for the integration of methods as full or reduced state space observer, anti-windup mechanism and discrete linear quadratic regulator. In addition, `pysimCoder` offers a graphical editor with a code generator. This combination allows the user to design the desired application graphically using predefined blocks and then generate a C code that can be run on target hardware. Details and `pysimCoder` setup and compilation is in [11].

PysimCoder supports code generation for various platforms using custom APIs, like STMicroelectronic's STM32H7, Microchip SAMD21, or Linux. Simulations of control systems can be done if the code is generated for Linux. In the context of this paper, the ability of code generation for the POSIX-compatible NuttX RTOS is more important. As with NuttX, the generated code runs in a high-priority task, with a periodic control loop. The sampling period of the control loop can be set in `pysimCoder` GUI

Fig. 12 The control unit with a PMSM and an unused DC motor serving as a brake [2]

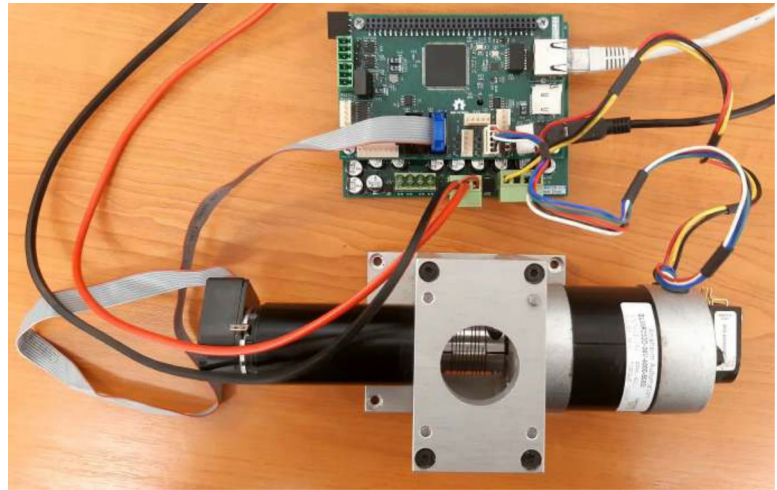


Table 2 Parameters: PMSM BLWR233D-36V-4000 [2]

Symbol	Description	Value
P	Rated power	95 W
R_S	Stator resistance	0.64 Ω
L_S	Stator inductance	0.0021 H
Ψ_M	PM rotor magnetic flux	0.0200077 Wb
B	Viscous friction coef	$\approx 0 \text{ kg m}^2 \text{ s}^{-1}$
p	Number of pole pairs	4
J	Moment of inertia	$12 \cdot 10^{-5} \text{ kg m}^2$

before the code generation. Runtime monitoring and tuning of model parameters that allow the user to display and edit parameters, inputs, and outputs of individual blocks, is solved using Silicon Heaven infrastructure (SHV) [26] and [11].

Leveraging the SHV infrastructure further, a program compiled from the newly generated code can be flashed onto the target platform (or confirmed), as described in 4.1.2, using a dedicated pop-up window in the pysicsCoder graphical user interface.

User Applications Using pysicsCoder

This section demonstrates control applications and their pysicsCoder diagrams. The behaviour of the control system can be remotely tuned using the Silicon Heaven protocol, for example using references or PID controller values. It is shown here that pysicsCoder is ready to create control applications, including those used for PMSM *field-oriented control*, piezoelectric actuator control and related position measurement, as well as logging capabilities. In this work, we focused more on the control of the piezoelectric actuator, the PMSM applications remain the same as in [2].

Control of PMSM Drives

The unit connected to PMSM is shown in Fig. 12. The parameters of the used PMSM are listed in Table 2. The PMSM has already been tested, see [30]. A diagram of the current PI controller of dq axes is presented, showing that pysicsCoder is capable of creating FOC applications. We present a simple non-vector PMSM control. The unit performance is designed up to 250 W, usual for drive in robot joints.

Current Controllers

To verify and tune the PI current control, a constant current synchronous mode is used, see diagram shown in Fig. 13. The rotor rotates synchronously by generating an angle from the integrating angular velocity (see integrator), acting only in the d -axis, while q -axis action is zero. The angular velocity must be set low. The measured currents i_a , i_b and i_c are acquired by reading the voltage drop on the current shunts with an AD converter. These readings are recalculated to obtain the i_q and i_d currents using the Clarke and Park transformations. The angle used in the Park transformation comes from the integrator.

These quantities are subtracted from the references and passed to inner PI controllers. Afterward, the Inverse Park and Inverse Clarke transformations are used to obtain the actions used to control the IFX007 switches. The angle used in the Inverse Park transformation is also from the integrator. The implementation in pysicsCoder is depicted in Fig. 14. In this figure, the ADC outputs are also passed to subtract and gain blocks, mapping an ADC number to the real currents using linear regression. The calibration procedure was done beforehand by fitting measured data using a custom Python script.

The output of the reference $i_{d,ref}$ and measured i_d and i_q currents is sent by using a NuttX UDP Socket to a given IP

Fig. 13 A simplified diagram of current controllers [2]

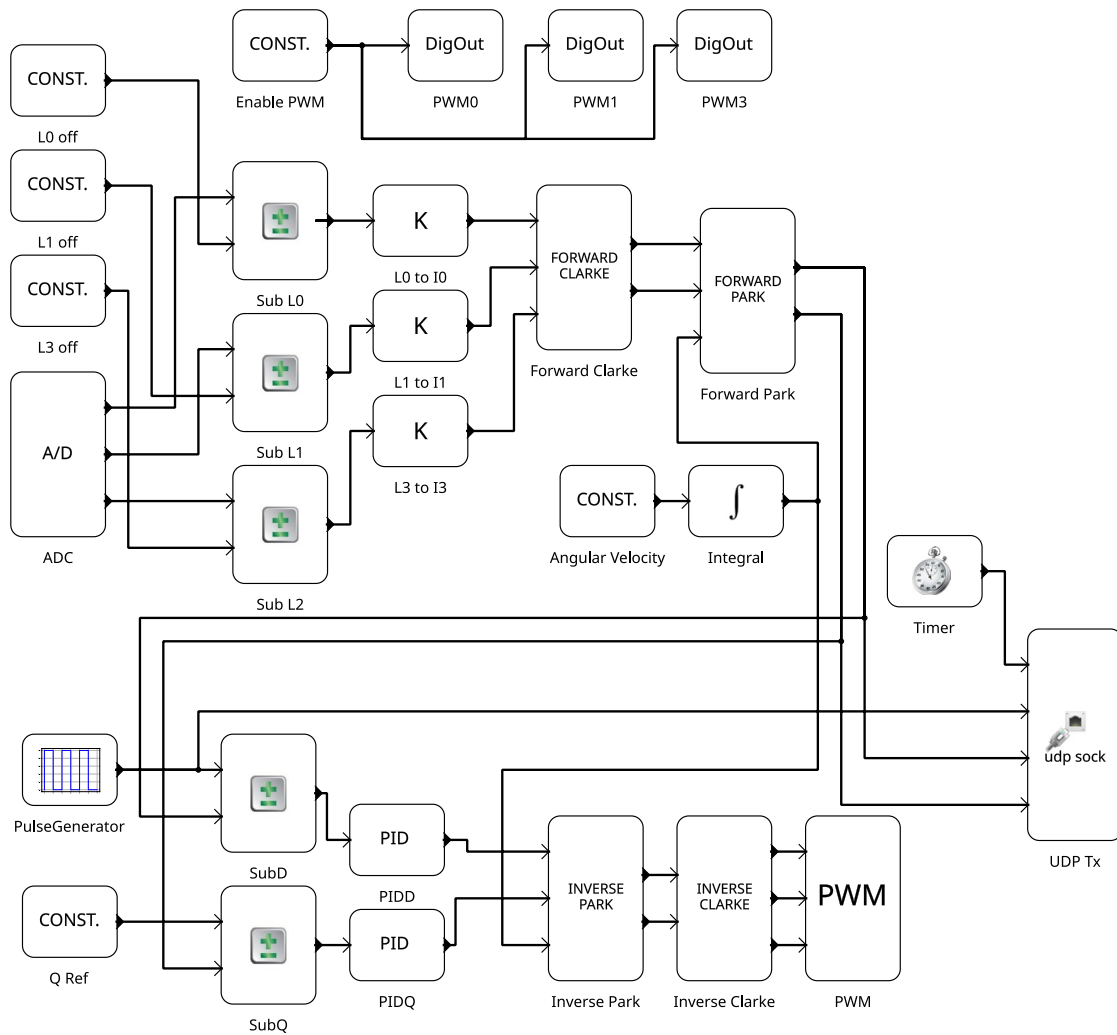
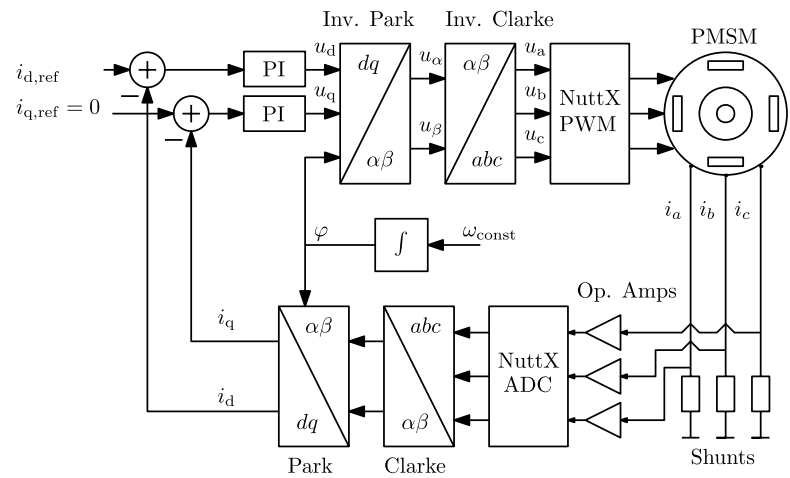


Fig. 14 PysimCoder diagram of dq current control [2]

address, which can be configured in the block itself. Using the Silicon Heaven protocol, the user can tune the constants of the PI controllers remotely while watching the changing waveforms. Fig. 15 presents the captured data visualisation.

At the time of writing the first article [30], the sampling frequency was set to 1 kHz due to limitations of NuttX. After utilizing the device timer, we measured the maximum computation time for a single control action to be approximately

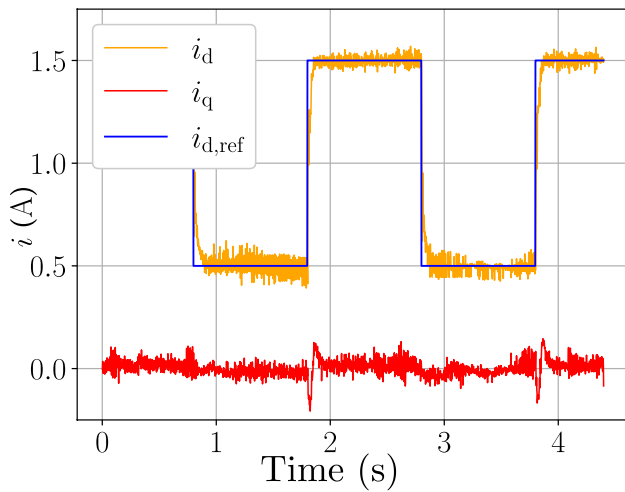


Fig. 15 The PI dq current control, $i_{q,ref} = 0$, [2] with average settling time 0.05 s and ripple 0.1 A

150 μ s. Consequently, the control loop frequency was set more conservatively to 5 kHz to accommodate possible latency deviations of the NuttX scheduler. Although much lower latencies were measured in 4.1.1, that benchmark did not include any additional computations. Moreover, the

generated code is highly generic due to the modularity of the connected blocks, which introduces further overhead. During testing, we also observed that computational time increased when adding the ADC NuttX block, caused by the capture of ADC data via the NuttX ADC driver.

Simple Closed-Loop Control

A simplified diagram of the closed-loop control is shown in Fig. 16. The difference between the target and the current incremental encoder count (NuttX-specific block) goes into the PID controller, setting the q -axis action for the highest torque while the d input is set to zero. The inverse Park transformation requires the rotor angle to be estimated. This is performed by the electrical angle estimation block using current IRC count and Hall sensors.

The realisation of this control diagram in pysimCoder is shown in Fig. 17. The *Hall To 6 Sectors* block reads the GPIOs (NuttX-specific) of the Hall outputs and transforms them into the 6 sectors (a simple integer). The *Encoder* block outputs the current encoder count (C) the encoder count of the last hit index (C_1) and the number of hit indexes.

Fig. 16 A simplified diagram of the closed-loop control [2]

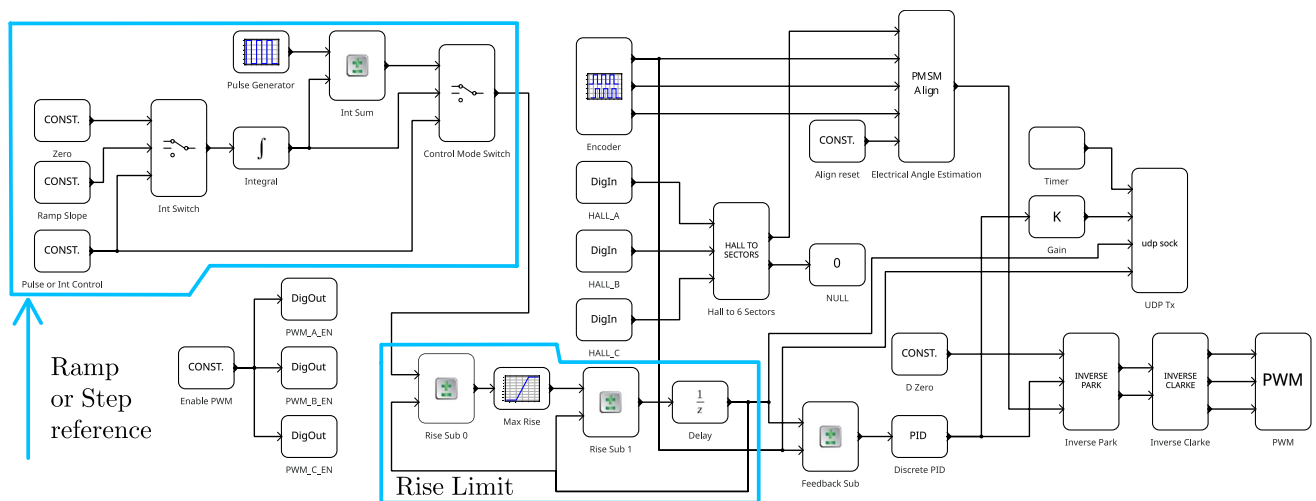
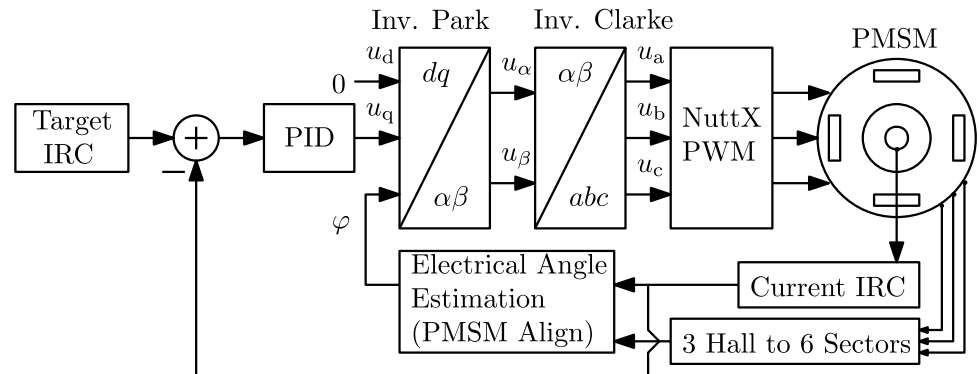


Fig. 17 PysimCoder of the closed-loop control diagram with a selectable reference [2]

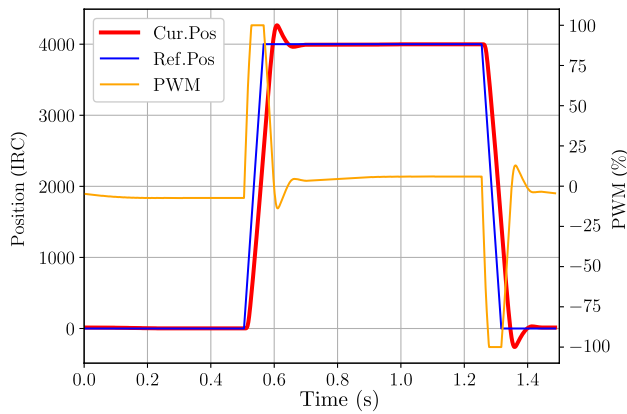


Fig. 18 The step response to 4000 encoder counts [2] with settling time 0.015 s

Table 3 Parameters: PL140-PIC251 [2]

Symbol	Description	Value
l_t	Length	40 mm
w	Width	11 mm
d	Thickness	0.55 mm
c	Capacitance	8.2 μ F
u_{rated}	Rated voltage range	± 30 V
z_f	Free deflection at u_{rated}	± 1 mm
f_b	Blocking force at u_{rated}	± 0.5 N
f_n	First natural frequency	160 Hz

The angle estimation is performed by the *PMSM Align* block. When the number of hit indexes is zero, the estimation is done only using Hall sensors, and the block outputs multiples of $\pi/3$. When a first index is hit, the following formula is used:

$$\varphi_{Est} = \frac{C - C_I}{C_{Turn}} 2\pi + \varphi_{Offset}, \tag{4}$$

where C_{Turn} denotes the number of encoder counts per one electrical turn and φ_{Offset} denotes an offset between the start of electrical angle and index. Values C_{Turn} and C_{Offset} must be set in the *PMSM Align* block.

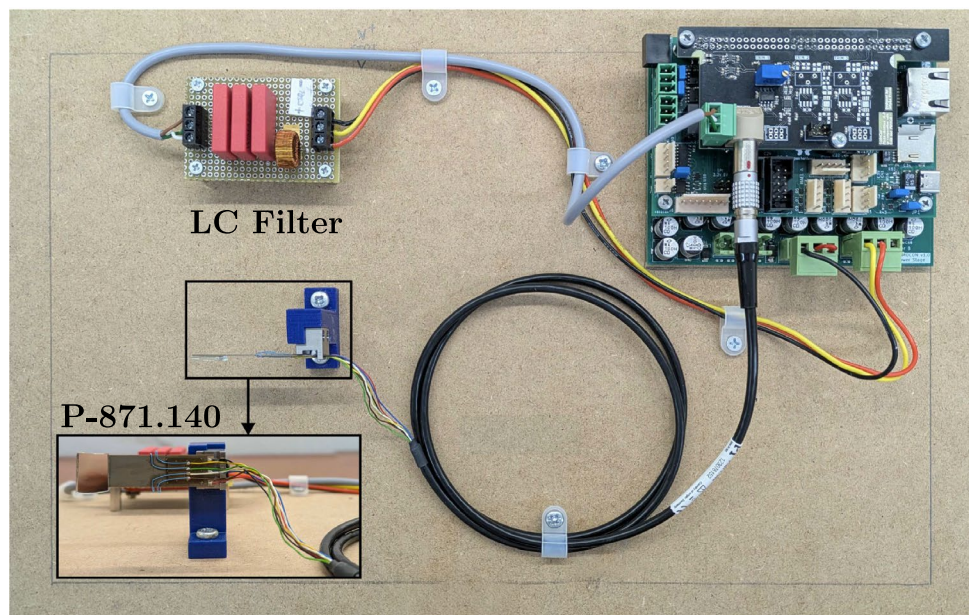
The diagram in Fig. 17 also features a selectable reference between a ramp and a step. The reference is chosen by setting the constant remotely using Silicon Heaven [11]. Furthermore, the rise time of the ramp is limited with a saturated sumator. Again, this diagram features a NuttX UDP Socket sending the PID action, the reference, and the actual position.

The sampling rate parameters were the same as for the *pysimCoder* current control scheme. A step response of PMSM with PWM, reference and positions is in Fig. 18.

Control of Piezoelectric Actuators

This section presents a setup with the SaMoCon controller controlling a piezoelectric actuator P-871.140 [31] and simple *pysimCoder* implementation. The actuator parameters are listed in Table 3, its material is ceramic PIC251. The purpose is to demonstrate simple applications of piezoelectric actuator control that can be quickly developed for research purposes. The experimental setup is shown in Fig. 19 and diagram of inputs and outputs of the actuator in *pysimCoder* is in Fig. 20.

Fig. 19 Experimental setup of the piezoelectric actuator and PCB for deflection measurement



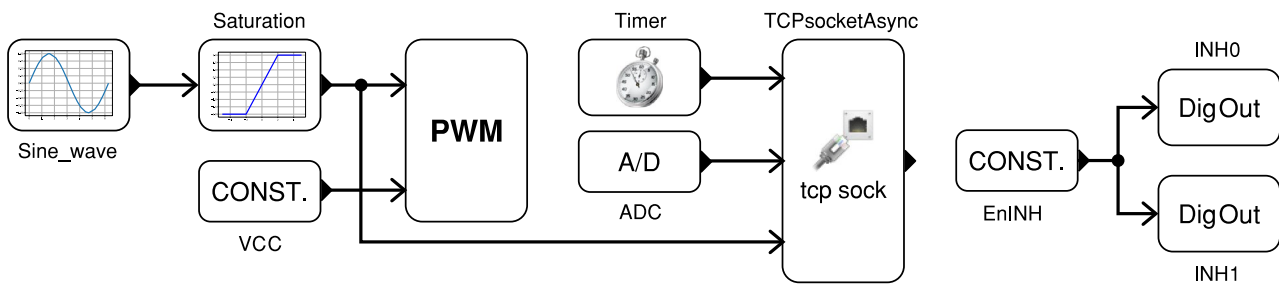


Fig. 20 The diagram with sine reference, and inputs and outputs of piezoelectric actuator

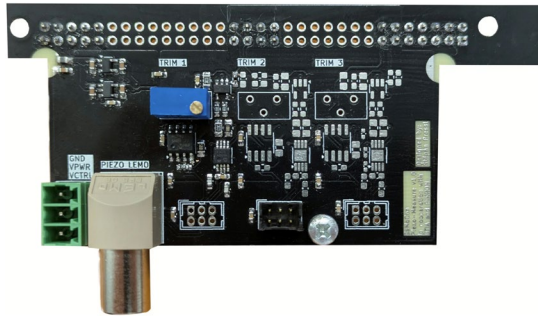


Fig. 21 The P-871.140 strain gauge sensor measurement and driver PCB board

Modulation of Control Actions

The piezoelectric actuator is controlled by a linear voltage. In our case, we have constructed an LC low-pass filter, turning the PWM voltage into a DC voltage which depends on the duty. Although the power board supplies 24 V at most, this voltage was used for initial tests, despite the rated voltage range of the actuator (Table 3). Performance is generally limited by the actuator warming, which affects its lifespan.

HW Realisation of Deflection Measurement

The deflection of the piezoelectric actuator was determined by an integrated *strain gauge sensor* (SGS). For that purpose, we designed an auxiliary PCB board. Since the piezoelectric actuator features a full-bridge strain gauge sensor, this PCB supplies the full-bridge with precisely defined voltage while utilizing a differential amplifier to measure the disproportional differential voltage.

The actuator datasheet [31] does not specify sensor parameters, but our measurements indicate that the resistance of one strain gauge sensor is 1.5 kΩ. Hence, we set the supply voltage of SGS to conservative 4.2 V. The supply voltage is provided by an MCP6022 operational amplifier, connected as a voltage follower of the 4.2 V reference. With such voltage, the current is within the operational range of the amplifier. For the differential amplifier, we selected the MCP6N16 instrumentation amplifier. The asymmetry of the full bridge was considered so that a multi-turn trimmer

could adjust the bias voltage of the instrumentation amplifier. The output of the instrumentation amplifier is again amplified and connected to the SAMV71 AFEC peripheral (AD converter). The amplification factor of the differential voltage is, in total, 300.

The PCB consists of three measuring circuits intended for piezoelectric actuators with custom-made strain gauge sensors. The first circuit is equipped with a LEMO-female connector, compatible with P-871.140. The LC filter output is connected to this board since the voltage for actuator control is in the same cable harness. The driver board (another PCB) for strain gauge sensor measurements is shown in Fig. 21.

Measurement Results

A sampling frequency of the model was set to 500 Hz as it provides adequate control of the system dynamics even with uncompensated nonlinear effects given by the hysteresis of the piezoelectric actuator. The presented measurements show raw data from the SAMV7 AD converters. The AD converters were configured to perform hardware averaging; thus all reads from the AD converter are 16 bit long.

The measured data are shown in Figs. 22, 23 and 24. The red dashed lines show the expected ADC value of the steady position before any tracking occurred. The measurements were carried out using three input signals: the sine waveform with two different amplitudes, ($\pm 5\%$ and $\pm 35\%$ deviation around the 50% PWM duty, which corresponds to 12 V after the LC filter, the steady position of the piezoelectric actuator) and a triangular waveform, whose PWM Duty ranges from 0 to 100%. Before any measurement took place, 50% PWM duty was applied (half of the operating range), then we read several ADC samples to get the value of the steady position.

All data were acquired using a pysimCoder TCP/IP block, continuously sending the current timestamp, the PWM duty, and the current ADC measurement to a target PC. The values of the PWM duty are scaled to the ADC readings to show the real deflection in the case of open-loop control.

Fig. 22 Sine waveform tracking. The PWM range is [45%, 55%], deflection $\pm 440 \mu\text{m}$

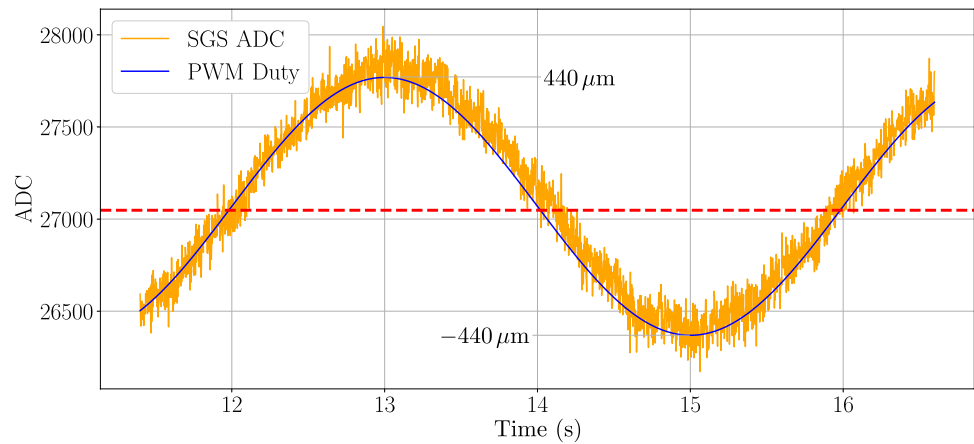


Fig. 23 Sine waveform tracking. The PWM range is [15%, 85%], deflection $\pm 640 \mu\text{m}$

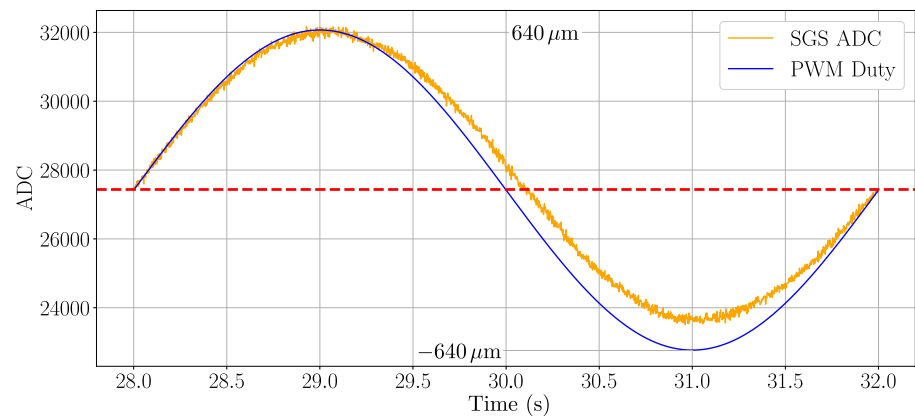
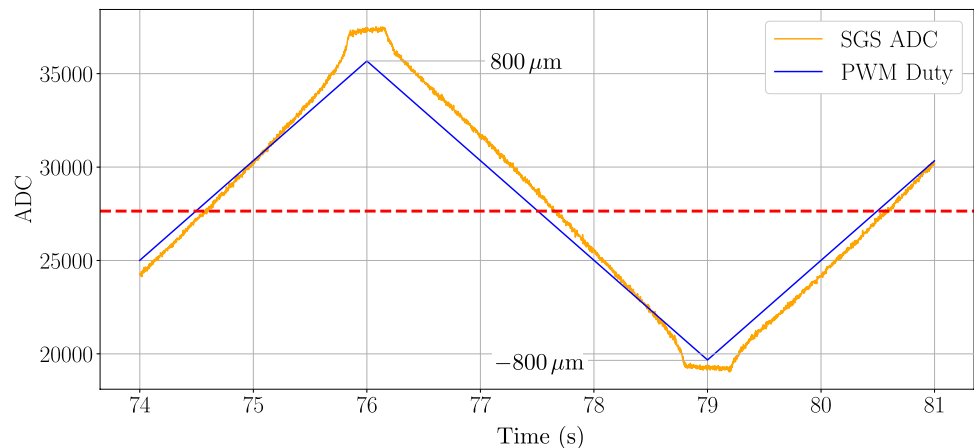


Fig. 24 Triangular waveform tracking. The PWM range is [0%, 100%], deflection $\pm 800 \mu\text{m}$



Conclusions

The paper addresses the design and development of an extensible prototyping unit with power stage up to 24 V determined for motion control of drives up to 200 W used in mechatronic and robotic applications [8, 32]. Here, modern BLDC/PMSM drives and piezoelectric actuators were considered. The selection of the microcontroller, communication interfaces, design of microcontroller boards and power stages, and SW implementation are described

including a description of the integration of expansion boards for specific applications such as external sensing and actuation.

The developed hardware of control unit has been successfully tested. The tests were performed using code generated by the block-oriented graphical environment pycoder, running on the NuttX RTOS, and also using the Silicon Heaven infrastructure for remote parameter tuning and firmware updates. Changes to NuttX peripheral drivers for the given platform and the `cyclictest` NuttX

implementation [24] have been merged into the mainline repository by the NuttX maintainers.

In contrast to the first experiments in [2], we have successfully fixed the sampling issues in NuttX with a different approach involving the usage of a hardware timer peripheral. We also proved that NuttX is a feasible operating system for such tasks with a benchmarking utility, which the NuttX maintainers merged into the mainline repository. We also addressed several hardware issues and selected a newer, lower-cost PIC32CZ microcontroller for the updated MCU board.

Additionally, firmware updates using the Silicon Heaven protocol were integrated during the *Google Summer of Code 2025* event by Štěpán Pressl (one of the authors of this paper). With this integration, the Silicon Heaven protocol is officially supported in NuttX, and standalone firmware updates can be tested using the NuttX application: `shv_nxboot_updater`.

The advantage of the open-source environment `pysimCoder` for rapid application prototyping without the need to write code was demonstrated. It was proven in the simple PID PMSM control and the piezoelectric actuator control. Compared to fully professional commercial solutions for universal use, e.g. dSPACE, the presented solution is feasible with affordable components and free open source software.

Future work on the platform will focus on eliminating the voltage spikes observed during current measurements (as discussed in Sect. 3.2.3) by selecting appropriate capacitor values or redesigning the board with alternative half-bridge switches. Future work should also address optimizing the NuttX ADC driver to enable faster computations in the code generated by `pysimCoder` or the `pysimCoder` blocks overall.

Moreover, it should be emphasized that `pysimCoder` uses the `double` datatype for calculations. If any algorithms happen to be too computationally intensive when implemented in `pysimCoder`, they can be implemented in more optimized form using fixed point arithmetic as integers. Since `pysimCoder` generated code uses NuttX as an underlying runtime environment, integrating such custom implementations should be straightforward, as NuttX offers standardised driver and POSIX API. We should also strive for optimizing the NuttX RTOS in the future.

Hardware and `pysimCoder` model diagrams are available in the public repository:

<https://gitlab.fel.cvut.cz/otrees/motion/samocon>

Acknowledgements This work was supported by The Czech Academy of Sciences, Institute of Information Theory and Automation under project No. 23-04676J of the Czech Science Foundation: Controllable Gripping Mechanics: Modelling, Control and Experiments.

Author Contributions The authors contributed equally.

Funding Project No. 23-04676J, Controllable Gripping Mechanics: Modelling, Control and Experiments, the Czech Science Foundation.

Data Availability Data available in public repository at: <https://gitlab.fel.cvut.cz/otrees/motion/samocon>

Declarations

Conflict of interest No Conflict of interest.

Research Involving Human and/or Animals Not applicable. **Informed Consent** Not applicable.

References

1. Tachiquingutierrez RT, Lay-Ekuakille A, Chiffi C, Singh SP, Rao KS. Design, fabrication, and testing of a microelectronic controller for sensing and actuating in robotic neurorehabilitation. *IEEE Sens J.* 2023;23(16):18700–7. <https://doi.org/10.1109/JSEN.2023.3291956>.
2. Belda K, Piša P, Pressl Š. Motion control unit design for control prototyping of modern bldc/pmsm drives and piezo actuators. in Proceedings of the 21st international conference on informatics in control, automation and robotics, 2024;579–90. <https://doi.org/10.5220/0012999300003822>
3. ODrive Robotics: The ODrive Website. 2024; <https://odriverobotics.com/>. Accessed 20 Sept. 2024.
4. Vedder B. The VESC project website. <https://vesc-project.com/>; 2022. Accessed 20 Sept 2024.
5. SOLO Motor Controllers SRL: The SOLO Controllers Website. <https://www.solomotorcontrollers.com/>; 2024. Accessed 20 Sept. 2024.
6. Speedgoat GmbH: Speedgoat Rapid Control Prototyping. <https://www.speedgoat.com/solutions/testing-workflows/rapid-control-prototyping>; 2024. Accessed 20 Sept. 2024.
7. dSPACE GmbH: dSPACE Rapid Prototyping Systems. <https://www.dspace.com/en/pub/home/applicationfields/foo/prototyping-systems.cfm>; 2024. Accessed 22 Sept. 2024.
8. Záda V, Belda K. Structure design and solution of kinematics of robot manipulator for 3d concrete printing. *IEEE Trans Autom Sci Eng.* 2022;19(4):3723–34. <https://doi.org/10.1109/TASE.2021.3133138>.
9. Gao X, Yang J, Wu J, Xin X, Li Z, Yuan X, et al. Piezoelectric actuators and motors: materials, designs, and applications. *Adv Mater Technol.* 2020;5(1):1900716. <https://doi.org/10.1002/admt.201900716>.
10. Moon JJ, Lee W, Park SW, Kim JM. Fault tolerant control method of seven-phase bldc motor in asymmetric fault condition due to open phase. In: 2015 9th International conference on power electronics and ECCE Asia (ICPE-ECCE Asia), 2015;1591–6. <https://doi.org/10.1109/ICPE.2015.7167989>
11. Lenc M, Piša P, Bucher R. `pysimCoder`—Open-source rapid control prototyping for GNU/Linux and NuttX. in 2023 24th International Conference Process Control, 2023;102–7. <https://doi.org/10.1109/PC58330.2023.10217596>
12. PiKRON: Lx_RoCon — Robot Motion Controller. https://pikron.com/pages/products/motion_control/lx_rocon.html 2014; Accessed 10 July 2024.
13. Microchip, Technology.: Arm MCUs: SAM E70/S70/V70/V71 Family Data Sheet. <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU32/ProductDocuments/DataSheets>

- [/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527.pdf](#) 2023; Accessed 01 Oct 2023.
14. Infineon, Technologies.: High Current PN Half Bridge with Integrated Driver. https://www.infineon.com/dgdl/Infineon-IFX007-T-DS-v01_00-EN.pdf?fileId=5546d46265f064ff0166433484070b75; 2018. Accessed 10 Sept. 2023.
 15. Microchip, Technology.: 10BASE-T/100BASE-TX PHY with RMII Support. <https://ww1.microchip.com/downloads> 2019; Accessed 10 Oct 2023.
 16. ASF, Apache Software Foundation.: Apache NuttX, RTOS. <https://nuttx.apache.org/> 2023; Accessed 10 July 2024.
 17. ASF, Apache Software Foundation.: Repository of NuttX Source Code. <https://github.com/apache/nuttx> 2024; Accessed 10 July 2024.
 18. ASF, Apache Software Foundation.: NuttX Documentation. <https://nuttx.apache.org/docs/latest/> 2024; Accessed 10 July 2024.
 19. The Open Group, IEEE: POSIX™ Certification Register. <https://posix.opengroup.org/register.html> 2025; Accessed 20 Dec 2025.
 20. Austin Group: POSIX IEEE Std 1003.1-2024. <https://posix.opengroup.org/> 2024; Accessed 10 July 2024.
 21. Torvalds L. Linux kernel. <https://kernel.org/> 2024; Accessed 10 July 2024.
 22. ASF, Apache Software Foundation.: NuttX Documentation — System Time & Clock. https://nuttx.apache.org/docs/latest/reference/os/time_clock.html 2024; Accessed 10 July 2024.
 23. Reghenzani F, Massari G, Fornaciari W. The real-time linux kernel: a survey on preempt_rt. ACM Comput Surv. 2019. <https://doi.org/10.1145/3297714>.
 24. ASF, Apache Software Foundation.: The Cyclictest NuttX Port Source Code. <https://github.com/apache/nuttx-apps/blob/master/benchmarks/cyclictest/cyclictest.c> 2025; Accessed 09 March 2025.
 25. ASF, Apache Software Foundation.: NuttX Documentation — System Time & Clock. <https://nuttx.apache.org/docs/latest/applications/boot/nxboot/index.html> 2025; Accessed 09 March 2025.
 26. Elektroline: The Repository of Silicon Heaven Protocol Source Code. <https://github.com/silicon-heaven> 2024; Accessed 10 July 2024.
 27. Kumar D, Yadulla AR, Bhuvanesh A, Pawar P, Kasula VK, Keerthanadevi R. Hierarchical blockchain framework for node authentication in IoT networks: a comprehensive analysis. in 2025 International conference in advances in power, signal, and information technology (APSIT), Bhubaneswar, India, 2025; 1–6 <https://doi.org/10.1109/APSIT63993.2025.11086262>
 28. Elektroline: Silicon Heaven Protocol - File node documentation. <https://silicon-heaven.github.io/shv-doc/rpcmethods/file.html> 2025; Accessed 09 March 2025.
 29. Bucher R. pysimCoder: Block diagram editor and realtime code generator for Python. <https://github.com/robertobucher/pysimCoder> 2024; Accessed 10 July 2024.
 30. Belda K, Piša P. Explicit model predictive control of PMSM drives. in IEEE 30th International Symposium on Industrial Electronics (ISIE). Kyoto, Japan. 2021;1–6 <https://doi.org/10.1109/ISIE45552.2021.9576296>.
 31. PI, Physik Instrumente, GmbH.: P-871 PICMA® Piezo Bender Actuators. https://www.pi-usa.us/fileadmin/user_upload/pi_us/files/product_datasheets/P871_Piezo_Bimorph_Bender.pdf 2008; Accessed 08 Feb 2024.
 32. Belda K, Venkrbec L, Jirsa J. Modelling, control design and inclusion of articulated robots in cyber-physical factories. Actuators. 2025. <https://doi.org/10.3390/act14030129>.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.