Lab 4: Introducing STM32 Nucleo boards

Objectives

1. Get familiar with STM32 Nucleo-32 boards software and Hardware Development
2. Use the Nucleo-32 UART to transmit and receive data
3. Use the Nucleo-32 UART to communicate with the Pololu motor controller
4. Use the Pololu TReX Motor Controller (DMC01) to control the Dagu Robotic Platform

Introduction

Nucleo-32 boards

The STM32 Nucleo-32 boards are from a well-known semiconductor manufacturing company named **STMicroelectronics**. They provide an affordable and flexible way for users to try out new concepts and build prototypes with STM32 microcontrollers, choosing from the various combinations of performance, power consumption and features including:

- STM32 microcontrollers in 32-pin packages
- Three LEDs: USB communication LED (LD1), Power LED (LD2), User LED (LD3)
- Reset push-button
- Board expansion Arduino™ Nano connector (i.e. compatible with Arduino Nano standard to give full access to all Arduino compatible shields)
- Flexible board power supply options: ST-LINK USB VBUS or External sources
- On-board ST-LINK/V2-1 debugger/programmer
- Support of a wide choice of Integrated Development Environments (IDEs) including IAR™ EWARM, Keil® MDK-ARM, GCC-based IDEs, Arm® Mbed™

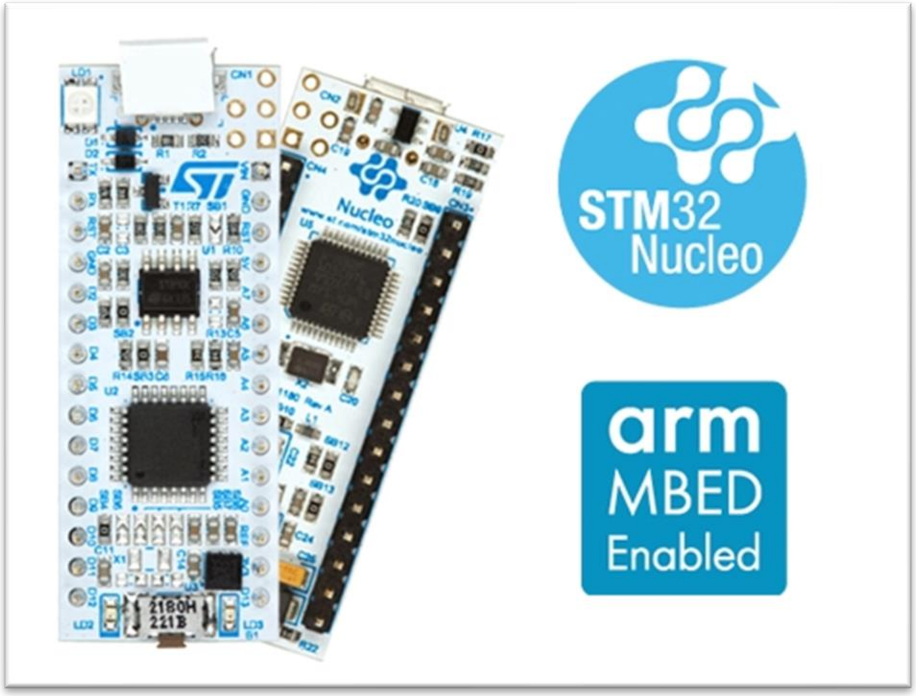

Figure 1

The Arduino™ Nano V3 connectivity support allows the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields.

We mainly work with NUCLEO-L432KC board. NUCLEO-L432KC microcontroller uses ARM®32-bit Cortex®-M4 CPU which can be clocked up to 80 MHz, and features 256KB Flash and 64KB SRAM. The microcontroller has plenty of peripherals such as GPIOs, Timers, USART, I2C, SPI, ADC, …

For NUCLEO-L432KC board pinout refer to the figures below and to https://os.mbed.com/platforms/ST-Nucleo-L432KC/

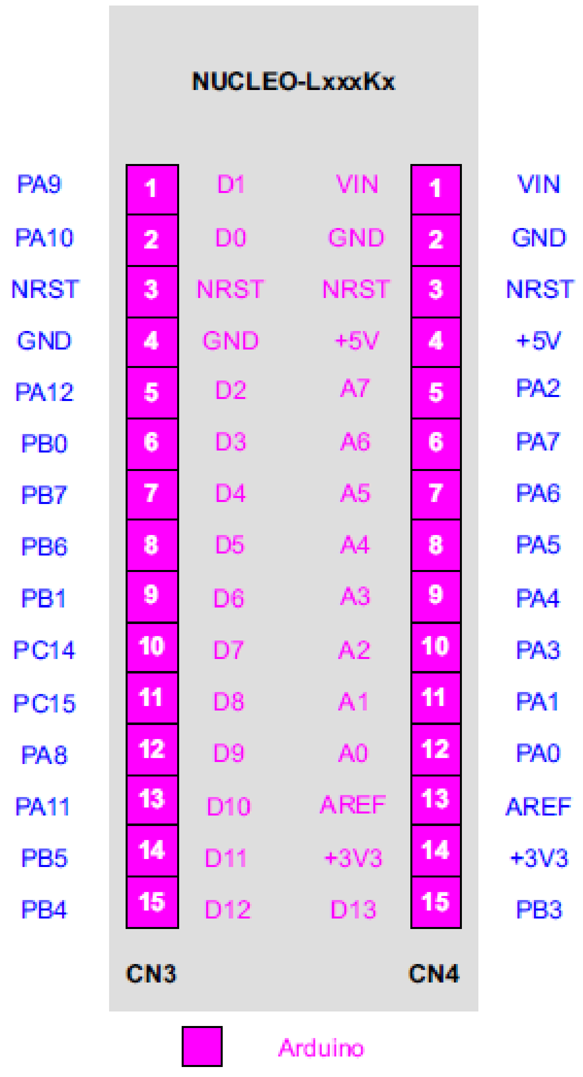

Figure 2. Nucleo-L432KC pin assignment

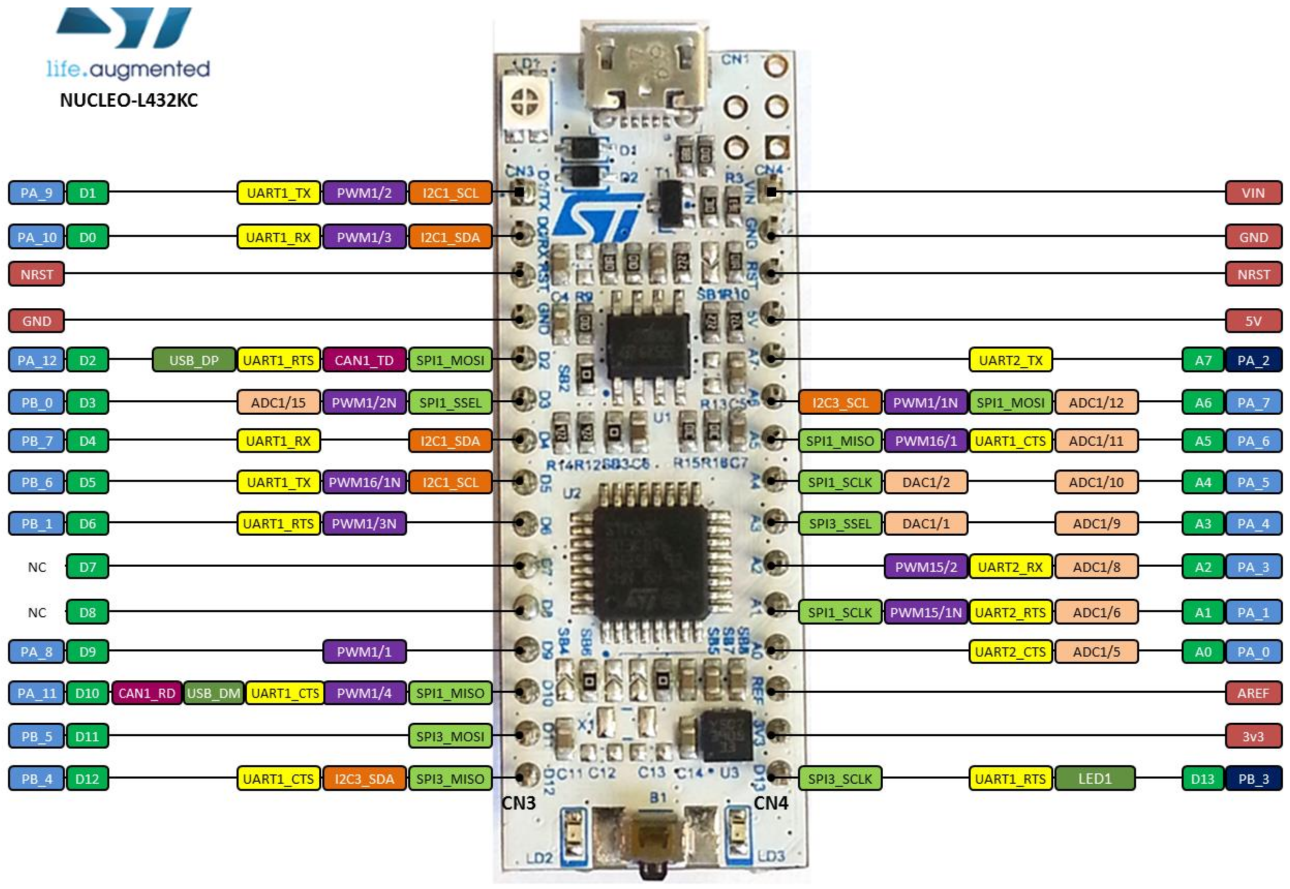
life.augmented
NUCLEO-L432KC
PA_9 D1 UART1_TX PWM1/2 I2C1_SCL
PA_10 D0 UART1_RX PWM1/3 I2C1_SDA
NRST
GND
PA_12 D2 USB_DP UART1_RTS CAN1_TD SPI1_MOSI
PB_0 D3 ADC1/15 PWM1/2N SPI1_SSEL
PB_7 D4 UART1_RX I2C1_SDA
PB_6 D5 UART1_TX PWM16/1N I2C1_SCL
PB_1 D6 UART1_RTS PWM1/3N
NC D7
NC D8
PA_8 D9 PWM1/1
PA_11 D10 CAN1_RD USB_DM UART1_CTS PWM1/4 SPI1_MISO
PB_5 D11 SPI3_MOSI
PB_4 D12 UART1_CTS I2C3_SDA SPI3_MISO
CN3
CN4
VIN
GND
NRST
5V
UART2_TX A7 PA_2
I2C3_SCL PWM1/1N SPI1_MOSI ADC1/12 A6 PA_7
SPI1_MISO PWM16/1 UART1_CTS ADC1/11 A5 PA_6
SPI1_SCLK DAC1/2 ADC1/10 A4 PA_5
SPI3_SSEL DAC1/1 ADC1/9 A3 PA_4
PWM15/2 UART2_RX ADC1/8 A2 PA_3
SPI1_SCLK PWM15/1N UART2_RTS ADC1/6 A1 PA_1
UART2_CTS ADC1/5 A0 PA_0
AREF
3v3
SPI3_SCLK UART1_RTS LED1 D13 PB_3

Figure 3

The ST-LINK debugger/programmer

Nucleo boards do not require any separate probe for programming and debugging as they integrate the ST-LINK debugger/programmer. The ST-LINK/V2 is an in-circuit debugger/programmer for the STM8 and STM32 microcontrollers. The single wire interface module (SWIM) and the JTAG/serial wire debugging (SWD) interfaces facilitate the communication with any STM8 or STM32 microcontroller operating on an application board.

The ST-LINK/V2-1 programming and debugging tool is integrated in the STM32 Nucleo-32 board. The embedded ST-LINK/V2-1 supports the SWD interface only for STM32 devices. The **features** supported by the ST-LINK/V2-1 **include debug port, Virtual Com port and Mass storage interface on USB**.

The ST-LINK/V2-1 requires a dedicated USB driver, which, for Windows® 7, 8 and 10, can be found at https://www.st.com/en/development-tools/stsw-link009.html

The STM32Cube

To help developing code faster for STM32, STM created the STM32Cube software stack that includes several components such as the STM32CubeMX. STM32CubeMX is a **graphical software configuration tool** that allows the **generation of C initialization code** using graphical wizards. That code can be used in various development environments like Keil µVision. The tool is a free download from STM website. STM32CubeMX has following features:

- Pin out-conflict solver

- A clock-tree setting helper
- A power-consumption calculator
- Utility to perform MCU peripheral configuration like GPIO pins, USART, etc.,
- Utility to perform MCU peripheral configuration for middleware stacks like USB, TCP/IP, etc.

Also, STM provides the STM32CubeL4 that contains the LL and HAL library for the STM32L4 family. These libraries are complementary and cover a wide range of application requirements.

- The STM32Cube **hardware abstraction layer (HAL)** is an STM32 abstraction layer embedded software, ensuring maximized portability across the STM32 portfolio. The HAL offers high-level and feature-oriented APIs with a high-portability level to simplify the user application implementation. These hide the MCU and peripheral complexity from the end-user.
- The **Low Layer APIs (LL)** offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The LL offers low-level APIs at register level, with better optimization but less portability. These require deep knowledge of the MCU and peripheral specifications.

Experiment 1:

Follow the tutorial provided on BB for creating a project for the nucleo board using STM32CubeMX and µVision (*Programming STM32 Nucleo boards using STM32CubeMX & Keil.pdf*). In this experiment, we are going to blink the on-board user LED. The user LED is a green LED connected to Arduino Nano signal D13 corresponding to the STM32 I/O PB3.

Experiment 2:

In this experiment, we are going to use the UART to send a text message to a PC running a terminal emulator. The PC is connected to the nucleo board using a USB-to-TTL module or FTDI cable (USB-UART Bridge). The software shall be built using the STM32L4 HAL.
Notes:
- Use the *HAL_UART_Transmit()* API.
- UART2 is directly connected to the PC via the SWD interface of the ST-LINK debugger, i.e. its TX and RX can get passed through the USB virtual com port.

Experiment 3:

Use the UART to receive characters from a PC (running a terminal emulator) to the STM32 board. The terminal emulator should take different inputs to do one of the following: switch the on-board user LED on solid, flash the LED 5 times, or switch the LED off. Program the board to receive UART data through polling.

Experiment 4:

Rewrite the program of the previous experiment using interrupts rather than polling.
Notes:
- Check the polling mode and Interrupt mode IO operations in the pdf doc "Description of STM32L4 HAL and low-layer drivers".
- Enable the specified UART interrupt in the main function using:
  HAL_UART_Receive_IT(&huart2, &rxdata, 1); // For receiving one byte at a time
- Add this function to process the received data inside:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
 if (huart->Instance == USART2) // Check if the interrupt is from your UART instance
 {
  // Process the received character (rxdata)

  // Re-enable the receive interrupt for the next byte
  HAL_UART_Receive_IT(&huart2, &rxdata, 1);
 }
}
```

The Pololu TReX Motor Controller

A versatile DC motor controller (shown on **Figure 4**) designed to seamlessly blend autonomous and human control of small- and medium-sized robots. The TReX can control two bidirectional motors through an asynchronous serial (RS-232 or TTL). The serial port has a default baud rate of 19,200 bps and can accept motor command packets to control the connected motors. The TReX Compact protocol command packet is simply:

command byte (with MSB set), any necessary data bytes

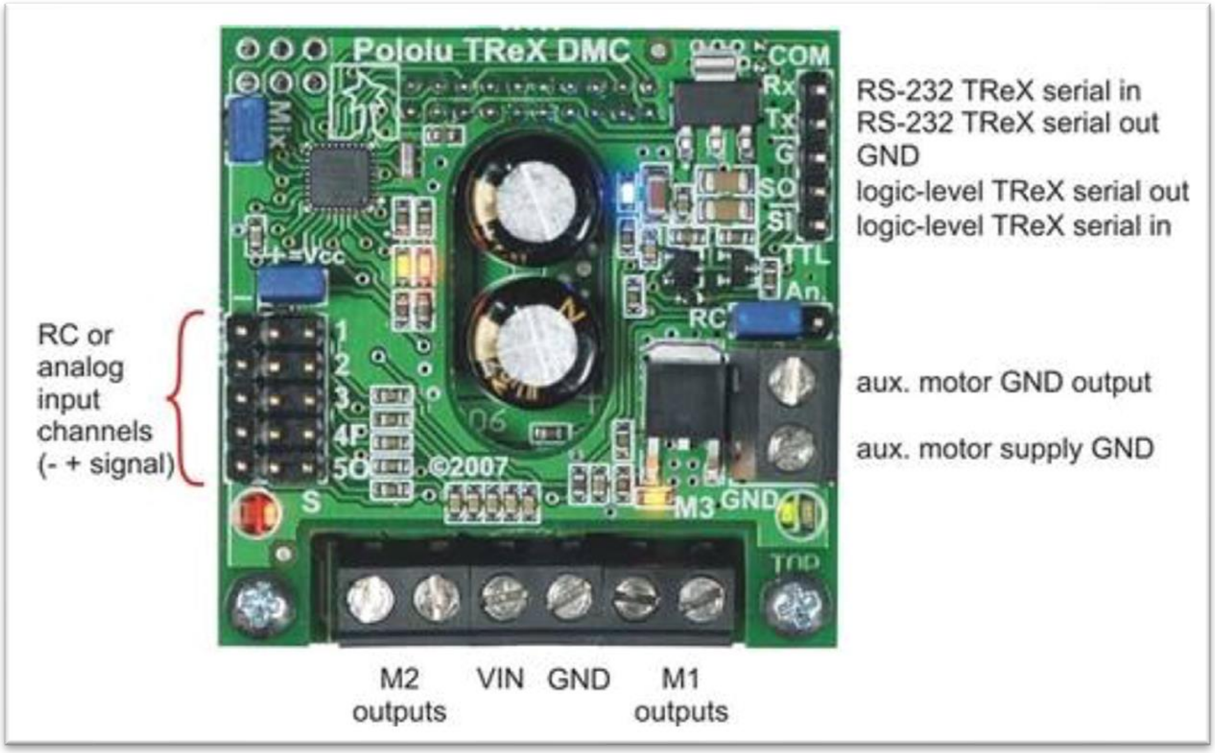

Figure 4. Pololu TReX Motor Controller

The command packets consist of a single command byte followed by any data bytes that the command requires. Command bytes always have their most significant bits set (i.e. range from 0x80 – 0xFF) while data bytes always have their most significant bits cleared (i.e. range from 0 – 0x7F). For example:

0xC0 – 0xC3: set motor 1
0xC8 – 0xCB: set motor 2

These commands take one data byte and returns nothing. It immediately sets the speed of the specified motor equal to the data byte (0-127) and the motor direction based on the two least significant bits of the command byte. The direction bits work as follows:

- 00 = brake low (command 0xC0/0xC8)
- 01 = reverse (command 0xC1/0xC9)
- 10 = forward (command 0xC2/0xCA)
- 11 = brake low (command 0xC3/0xCB)

Dagu Wild Thumper 4WD Chassis

Rugged, 4-wheel-drive chassis (shown on **Figure 5**) with independent suspension for each of its spiked 120mm-diameter wheels. Each wheel is connected to a DC motor for 4-wheel-drive operation. To control the vehicle, we are going to connect the motors to the TReX motor controller. The two left motors will be connected to the controller “M1 outputs” port and the two right motors will be connected to the “M2 outputs” port. The motor controller receives commands from the STM32 nucleo board using an asynchronous serial link and communicates it with Dagu.

To move the vehicle forward at 50% of its maximum speed (64_{10}), the STM32 sends the following command packets:

- 0xC2 0x40
- 0xCA 0x40

To move backward at 50% of its maximum speed, the STM32 sends the following command packets:

- 0xC1 0x40
- 0xC9 0x40

To turn the vehicle to the left or to the right, differential drive is utilized. To steer to the left, increase the speed of the right wheel above that of the left wheels ($v_r > v_l$). To steer to the right, make $v_l > v_r$. The steering speed is proportional to speed difference.

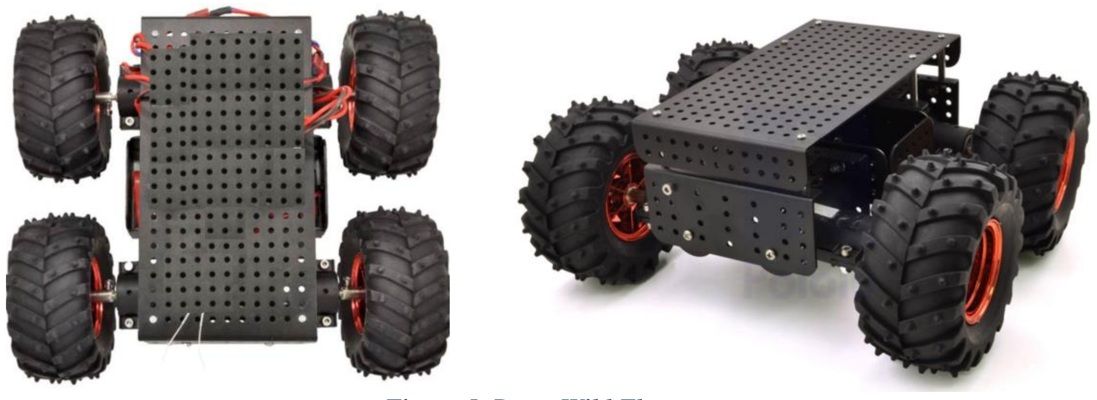

Figure 5. Dagu Wild Thumper

Lab Report [10 pts] (Group of two students submission)

1. [5 pts] Connect the nucleo board to the Dagu motor controller and send commands to move the DAGU vehicle as per the following sequence
   - Move forward for 2m
   - Make a right turn
   - Move forward for 2m
   - Make a left turn
   - Move forward for 2m

   Provide your code along with a small sized video of the Dagu Thumper moving in the specified sequence.

2. [2 pts] Develop an embedded application that gets input text from a smart phone over a UART to Bluetooth Bridge (HC-05 or HC-06 or HM-10 or JDY-08) and echoes it to a terminal emulator software running on the PC.
   Notes:
   - Use interrupts rather than polling.
   - Check the polling mode and Interrupt mode IO operations in the pdf doc "Description of STM32L4 HAL and low-layer drivers".
   - Provide your code along with a small sized video of the application running on the nucleo.

3. [3 pts] Combine your work of Q1 and Q2 to develop an embedded application that sends commands from the smart phone to move the DAGU vehicle.
   The command consists of a character followed by a number. The character indicates which direction the vehicle moves in (R for right, L for left, F for forward, B for backwards, S for Stop) and the number indicates how many meters to move (ranging from 0 to 5).
   Provide your code along with a small sized video of the application running on the nucleo.