Softwaretechnik, Mikrocontroller und fischertechnik

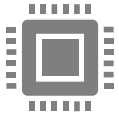
Stefan Falk

Lasst uns eine Lücke füllen

	fischertechnik-Controller	Controller mit nanoFramework	Arduino u. ä.
Entwicklungssystem	RoboPro	Microsoft Visual Studio 2017/2019	Einfache Systeme, nicht viel mächtiger als ein simpler Texteditor
Programmiersprache	RoboPro	C#	C
Eigenschaften	Komfortabel und „unkaputtbar“	Fast der volle .net-Umfang, sehr mächtig, sehr komfortabel, dafür Timing im ms-Bereich	Low-Level, sehr nah an der Hardware, relativ fehlerträchtig, dafür sehr schnell in der Ausführung
Zielgruppe	Einsteiger bis Experten	Leute, die komfortabel komplexe Software entwickeln und testen wollen mit demselben Komfort, den sie von PCs gewohnt sind	Masochisten ;-) die ganz tief unten wissen wollen, wie alles funktioniert, oder die Timing im µs-Bereich benötigen

Was ist nanoFramework?

Ein kostenloses Add-In in Microsoft Visual Studio 2017/2019



Firmware: standardkonforme .net-Runtime (CLR) für Mikrocontroller

Tools, um die Firmware auf verschiedenste Boards zu bringen

Ein vollständig quelloffenes Projekt als Nachfolger des Microsoft .net Micro Frameworks

https://nanoframework.net

https://github.com/nanoframework

Sehr freundliche und hilfsbereite Entwickler

https://discord.gg/gCyBu8T

Unterstützte Plattformen

- So ungefähr jedes Board mit einem 32-bit-µC drauf – Auswahl:

Target	Gpio	Spi	I2c	Pwm	Adc	Serial	OneWire	Events	SWO	Networking	Large Heap
ST_STM32F429I DISCOVERY	✔	✔	✔	✔	✔	✔	✔	✔	✔		✔
ST_NUCLEO64 F091RC	✔	✔	✔	✔		✔	✔	✔	✔		
ST_STM32F769I DISCOVERY	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
MBN_QUAIL	✔	✔	✔	✔		✔	✔	✔			
NETDUINO3_WIFI	✔	✔	✔	✔	✔	✔	✔	✔			
ESP32_WROOM_32	✔	✔	✔	✔	✔	✔	✔	✔		✔	

Was habe ich davon?

Softwaretechnisch aus dem Vollen schöpfen

.net/C#

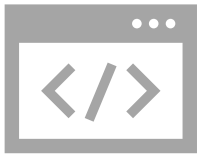
strikt typsicher

vollständig objektorientiert

automatische Garbage Collection

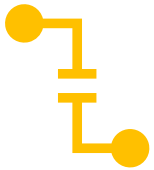
Multithreading

umfangreiches Standard-Framework



Entwicklungsystem vom Feinsten

Visual Studio als komfortables Entwicklungssystem mit IntelliSense, NuGet-Packages und was man sonst so als Desktop-Entwickler gewohnt ist

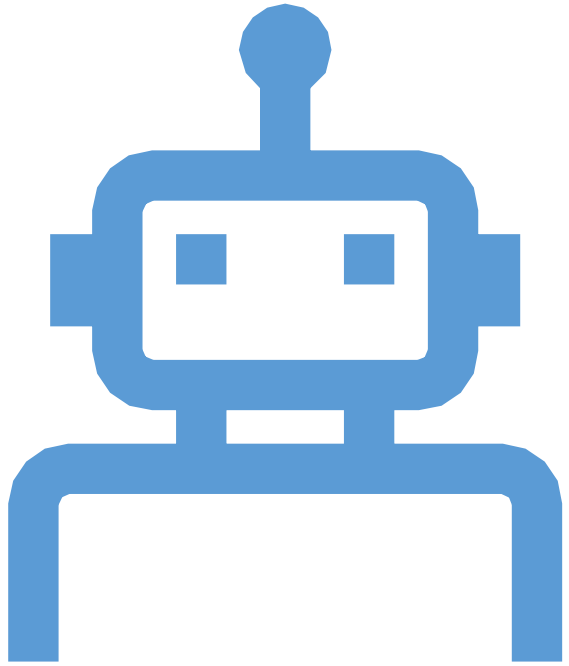


Traumhaftes Debugging

Haltepunkte, Variablen inspizieren und setzen, all die Debugging-Features von .net – live vom PC in den Mikrocontroller

Womit erkaufe ich mir das?

- Du brauchst einen Windows-PC
  - Aktuell kein Linux, kein Mac
  - Die kostenlose Visual Studio Community Edition genügt
- nanoFramework interpretiert die compilierte Intermediate Language (IL, der .net-Bytecode) zur Laufzeit
  - Du kannst damit kein Timing im µs-Bereich haben, sondern nur im ms-Bereich
  - Das ist immer noch viel schneller als das 10-ms-Raster der RoboPro-Controller
  - Für fast alle fischertechnik-Modelle mehr als schnell genug
  - Wenn es wirklich nötig ist, kannst Du selber nativen (C/C++)-Code einbetten für die Teile, die harte Echtzeit benötigen



nano Framework

Nun zeig‘ doch endlich!

… Demo!

Das Abstract I/O-Framework

softwaretechnisch noch eins draufsetzen

Mein Programm braucht einen Taster.

Wirklich? Nein! Es braucht nur irgendwas mit ja oder nein!

Also schreiben wir das Programm so, dass es nur „irgendwas mit ja oder nein“ bekommt.

Dann kann ich aber auch was anderes reinreichen, was „ja oder nein“ liefert!

- Einen Analogwert einer Lichtschranke, mit einem Schwellwert verglichen!
- Ein Software-Objekt, mit dem ich meinen Modell-Code vorab testen kann!

Die Idee

Das einfachste Programm der Welt

- Wenn und solange ich den Taster drücke, soll der Motor laufen
- Superprimitiver Ansatz:

```
while (true)
{
    if (TasterGedrueckt())
    {
        MotorEin();
    }
    else
    {
        MotorAus();
    }
}
```

Dasselbe Programm mit AbstractIO

```
// Merke: Dieses Programm wird sich heute nicht mehr ändern!

public static void Run(IBooleanInput button, IBooleanOutput motor)
{
    while (true)
    {
        motor.Value = button.Value;
    }
}
```

Aufruf mit abstrakten I/O-Objekten

```
Run(

    button: new Netduino3.DigitalInput(DigitalInputPin.D2),

    motor: shield.GetDcMotor(1)
            .MappedFromBoolean(falseValue: 0f, trueValue: 1f)

);
```

Lass den Motor weich anlaufen!

```
Run(

    button: new Netduino3.DigitalInput(DigitalInputPin.D2),

    motor: shield.GetDcMotor(1)
            .Smoothed(valueChangePerSecond: 0.5f, rampIntervalMs: 20)
            .MappedFromBoolean(falseValue: 0f, trueValue: 1f)

);
```

Eine Lampe soll den Lauf anzeigen

```
Run(

    button: new Netduino3.DigitalInput(DigitalInputPin.D2),

    motor: shield.GetDcMotor(1)
            .Smoothed(valueChangePerSecond: 0.5f, rampIntervalMs: 20)
            .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
            .Distributed(
                shield.GetDcMotor(2)
                .MappedFromBoolean(falseValue: 0f, trueValue: 1f))

);
```

Die Lampe soll blinken anstatt nur leuchten

```
Run(

    button: new Netduino3.DigitalInput(DigitalInputPin.D2),

    motor: shield.GetDcMotor(1)
            .Smoothed(valueChangePerSecond: 0.5f, rampIntervalMs: 20)
            .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
            .Distributed(
                shield.GetDcMotor(2)
                .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
                .BlinkedWhenTrue(onDurationMs: 500, offDurationMs: 500))

);
```

Weich blinken natürlich!

```
Run(

    button: new Netduino3.DigitalInput(DigitalInputPin.D2),

    motor: shield.GetDcMotor(1)
            .Smoothed(valueChangePerSecond: 0.5f, rampIntervalMs: 20)
            .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
            .Distributed(
                shield.GetDcMotor(2)
                .Smoothed(valueChangePerSecond: 2f, rampIntervalMs: 20)
                .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
                .BlinkedWhenTrue(onDurationMs: 500, offDurationMs: 500))

);
```

Analoge Lichtschranke anstatt Taster

```
Run(

    button: new Netduino3.AnalogAdcInput(AnalogInputPin.A1)
        .ScaleToRange(smallestValueMappedTo: 0f, largestValueMappedTo: 1f)
        .SchmittTrigger(threshold: 0.5f, hysteresis: 0.05f),

    motor: shield.GetDcMotor(1)
            .Smoothed(valueChangePerSecond: 0.5f, rampIntervalMs: 20)
            .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
            .Distributed(
                shield.GetDcMotor(2)
                .Smoothed(valueChangePerSecond: 2f, rampIntervalMs: 20)
                .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
                .BlinkedWhenTrue(onDurationMs: 500, offDurationMs: 500))

    );
```

Lichtschranke ist ok, aber andersrum bitte

```
Run(

    button: new Netduino3.AnalogAdcInput(AnalogInputPin.A1)
        .ScaleToRange(smallestValueMappedTo: 0f, largestValueMappedTo: 1f)
        .SchmittTrigger(threshold: 0.5f, hysteresis: 0.05f)
        .Invert(),

    motor: shield.GetDcMotor(1)
            .Smoothed(valueChangePerSecond: 0.5f, rampIntervalMs: 20)
            .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
            .Distributed(
                shield.GetDcMotor(2)
                .Smoothed(valueChangePerSecond: 2f, rampIntervalMs: 20)
                .MappedFromBoolean(falseValue: 0f, trueValue: 1f)
                .BlinkedWhenTrue(onDurationMs: 500, offDurationMs: 500))

    );
```

Große Änderungen am Modell ohne große Änderungen am Programm

Komfortable und wiederverwendbare Zusatzfunktionalität für alles mit bestimmten Datentypen

Änderung der Pin-Belegung mit minimalem Aufwand

Noch schnellere Entwicklung

Unit-Testbarkeit von Programmen für fischertechnik-Modelle – ohne Modell

Umstieg auf ein völlig anderes nanoFramework-Board mit minimalem Aufwand

https://github.com/steffalk/AbstractIO