Embedowe herezje - C# na mikrokontrolerach?

Michalak Przemysław

Gdańsk Embedded Meetup

06-12-2022

Mała autoprezentacja:

- komercyjnie w embedded ~8lat
  - branże:
    - IoT
    - pirotechnika
    - automotive
    - urządzenia medyczne
    - telekomunikacja
    - automatyka domowa
    - automatyka budynkowa
- aktualnie - embedded developer w jednym z działów w iSMA Controlli

Ogólny spoilerowy plan prezentacji

- garść informacji o C# i .NET Framework
- nanoFramework - teoria "ogólna"
- dlaczego "herezje"?
- dlaczego C# - plusy i minusy
  - wykorzystanie biznesowe
- nanoFramework - teoria niskopoziomowa
  - co wchodzi w skład nanoFrameworka - toolset
    - "quickStart" z punktu widzenia developera C#
  - komunikacja pomiędzy kodem C# a C/C++ (native)
    - na przykładzie dodania nowego API
- podsumowanie/wady

Mikrokontroler? C#?

- Pytania wstępne:
  - programowanie w C# - PC/web
  - mikrokontrolery w C#?
  - języki wyższego poziomu na mikrokontrolerach?

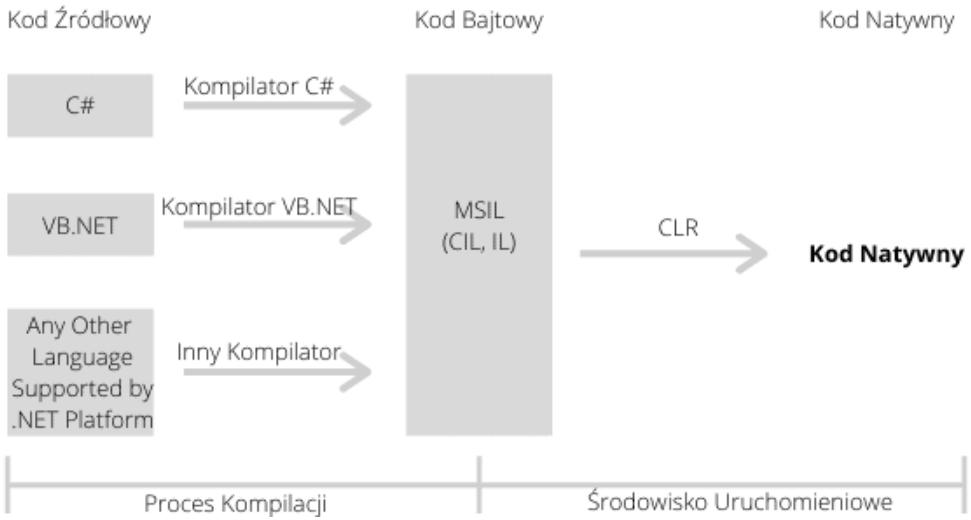
C#

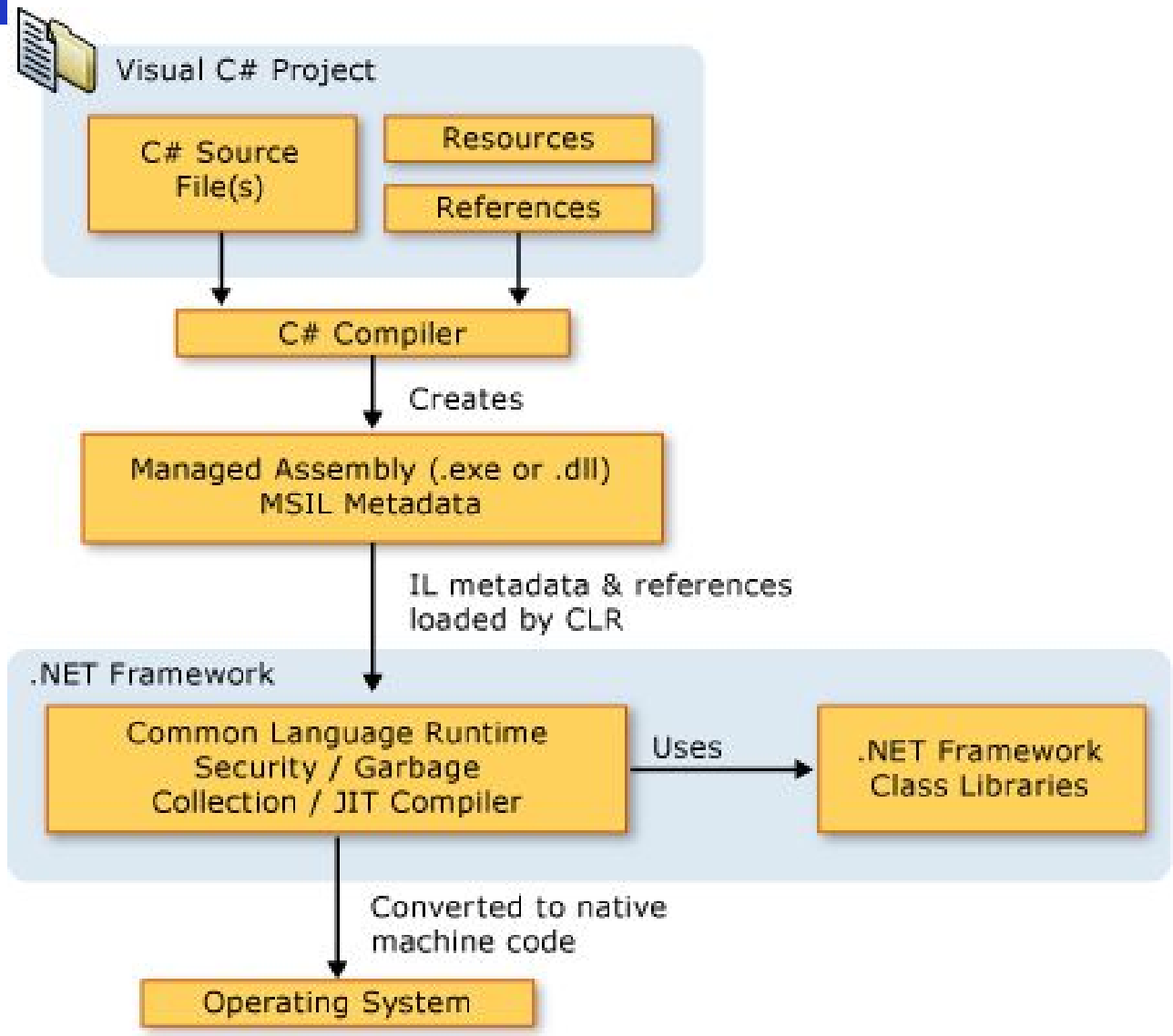
Najważniejsze cechy języka C#:

- w pełni obiektowy;
- stworzony i rozwijany przez Microsoft
- Garbage Collector działający automatycznie;
- dostęp do standardowych bibliotek;
- właściwości i zdarzenia (Properties, Events);
- delegaty oraz zarządzanie zdarzeniami (Delegates, Events Management);
- łatwość używania typów generycznych (Generics);
- kompilacja warunkowa;
- wielowątkowość;
- **LINQ** i wyrażenia lambda (lambda Expressions);
- DLL - Dynamic Link Libraries

C# - jak to działa?

- Słowniczek
  - aktualnie CIL - Common Intermediate Language ~"managed code" - ~"CIL bytecode"
  - CLR - Common Language Runtime - środowisko uruchomieniowe (CoreCLR openSource od 2015, aktualnie to dotnet/runtime)





.NET nanoFramework

nano
FRAMEWORK

nanoFramework

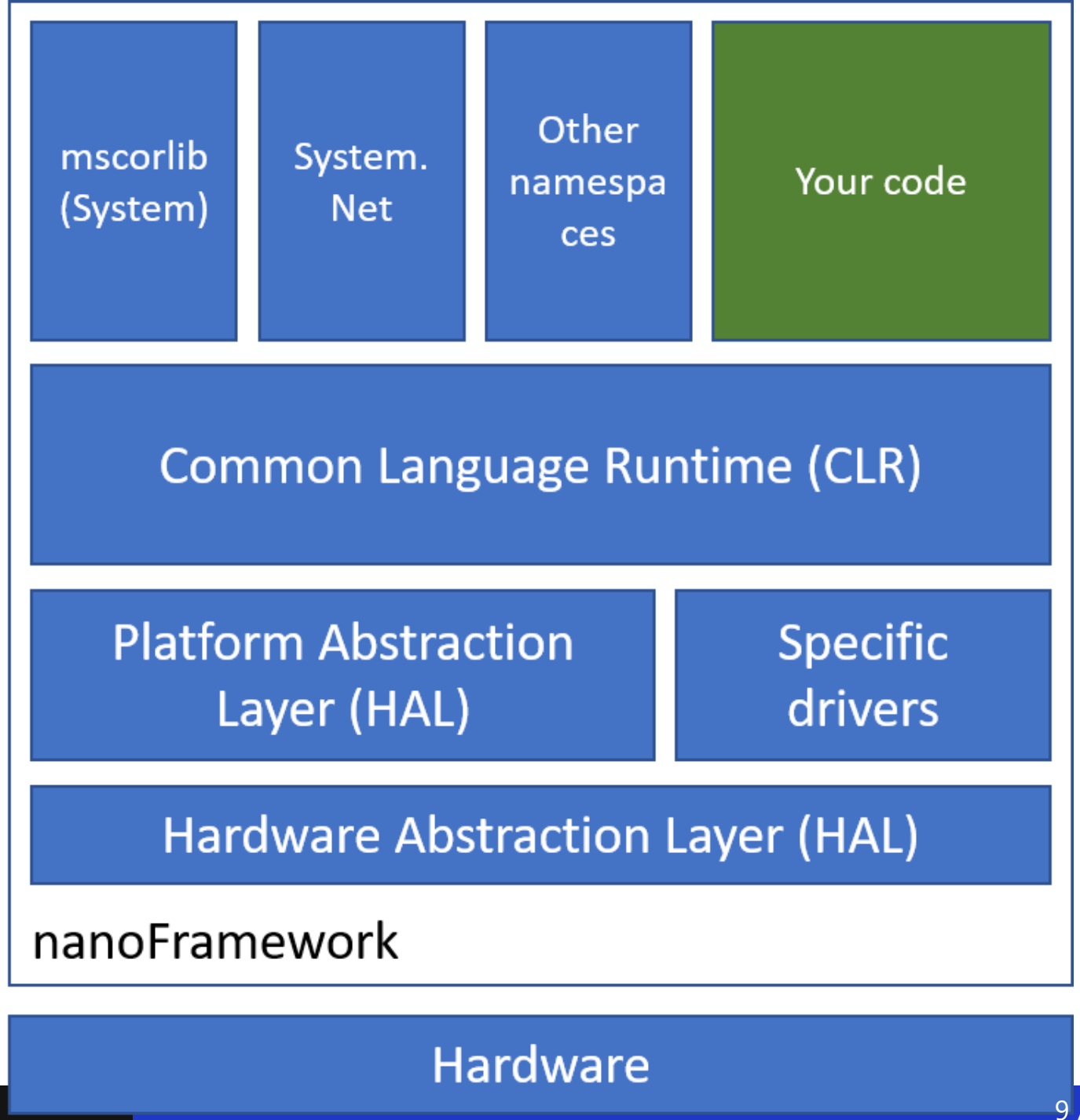
Here are some of its unique features:

- Can run on resource-constrained devices with as low as 256kB of flash and 64kB of RAM.
- Runs directly on bare metal. Currently there is support for ARM Cortex-M and Xtensa LX6 and LX7 cores.
- Supports common embedded peripherals and interconnects like GPIO, UART, SPI, I2C, USB, networking.
- Provides multithreading support natively.
- **Support for energy-efficient operation such as devices running on batteries.**
- Support for Interop code allowing developers to easily write libraries that have both managed (C#) and native code (C/C++).
- No manual memory management because of its simpler mark-and-sweep garbage collector.
- Execution constrains to catch device lockups and crashes.

> Jest trochę płynna granica między zaletami nF i C#.

architektura nF

- gdzie "Your code" dotyczy programistów C#
- programiści niskopoziomowi pracują na warstwie poniżej CLR'a
- mscorlib - (Microsoft Common Object Runtime Library)
- dokumentacja



Czyli "da radę" uruchomić C# na mikrokontrolerze.

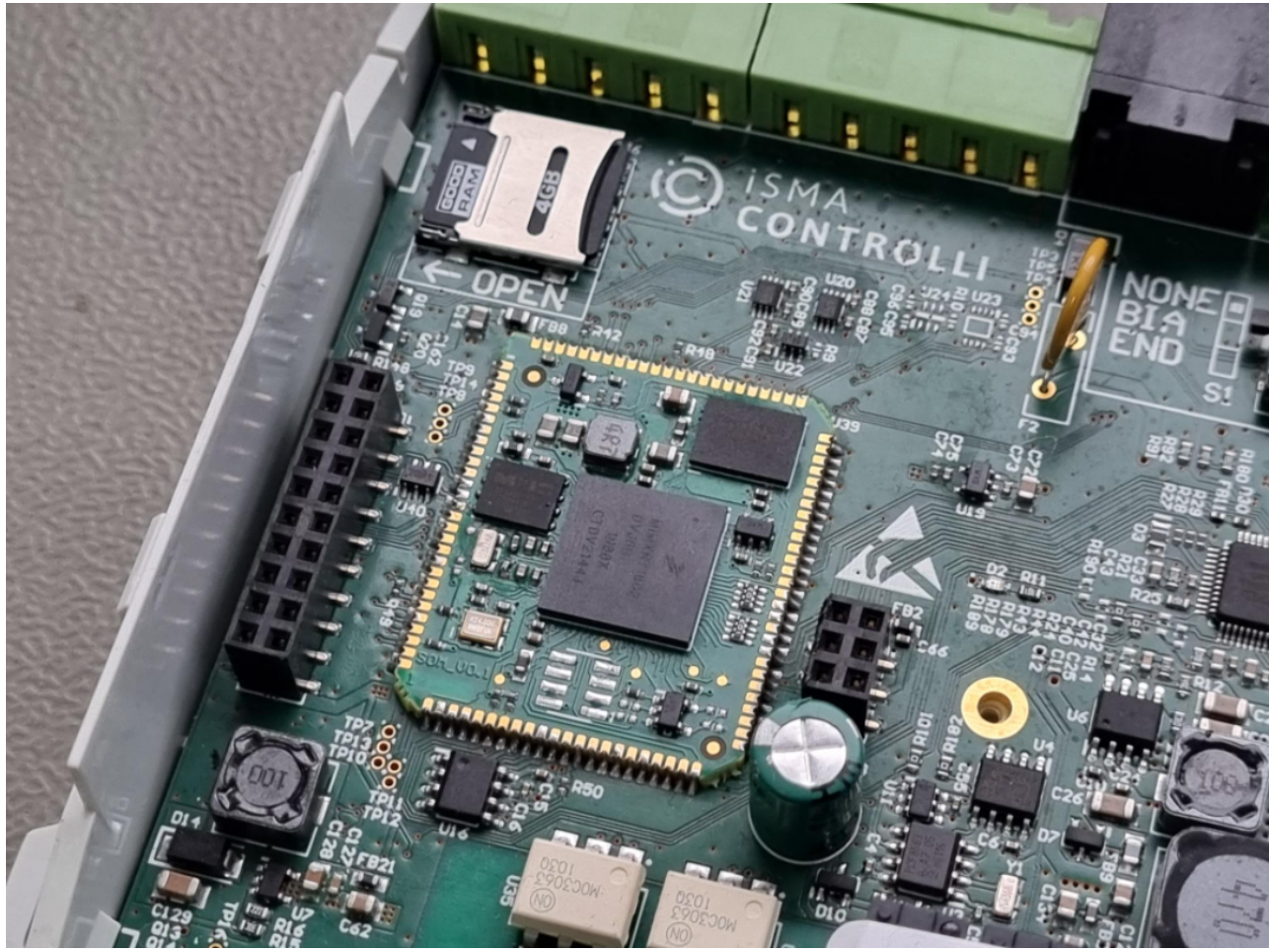
Wspierane platformy 1/2

Oficjalnie:

- Espressif ESP32 series
  - Espressif ESP32 series
  - Espressif ESP32-S2 series
  - Espressif ESP32-C3 series
- OrgPal boards
  - OrgPal PalThree (STM32F769BIT)
- STMicroelectronics boards
  - NUCLEO64_F091RC
  - STM32F429I_DISCOVERY
  - STM32F769I_DISCOVERY
- **NXP boards - port iSMA Controlli**
  - NXP i.MX_RT1060_EVK
  - iSMA Controlli SOM - w przyszłości -> fotka

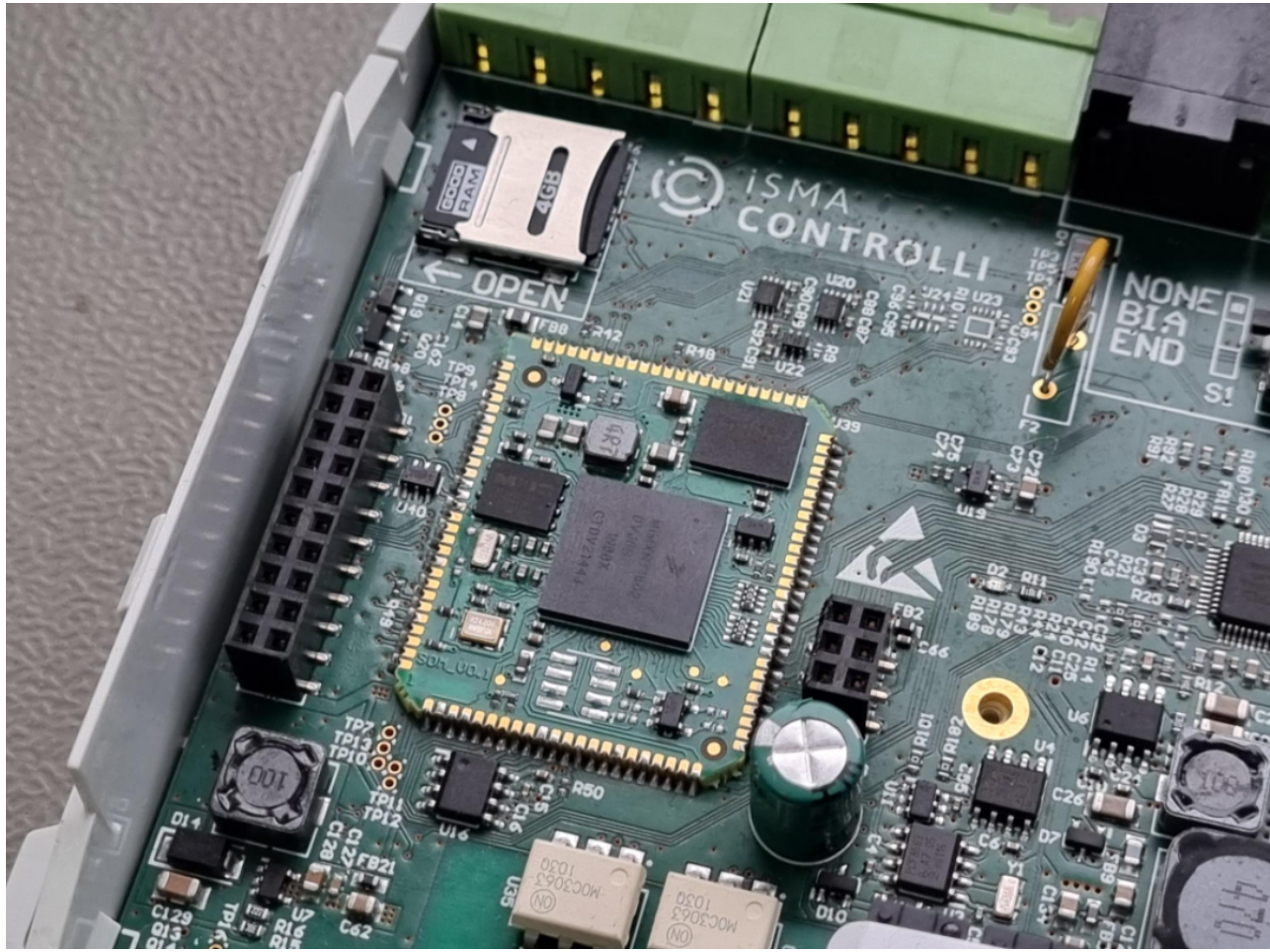
Każdy target ma tabelkę ze wspieranymi bilbiotekami.

Stan na 12.2022 - oficjalna lista



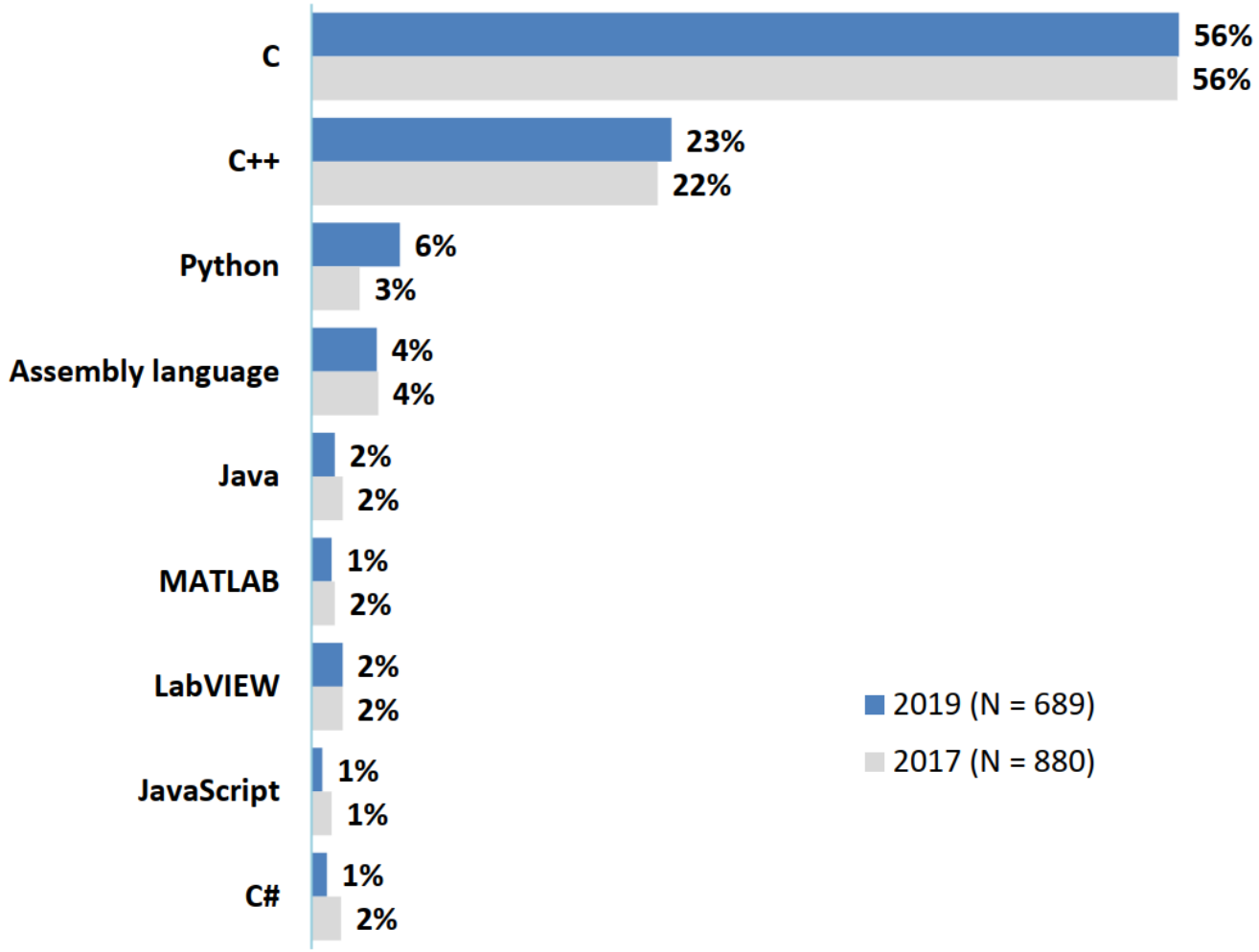
Wspierane platformy 2/2

- submoduł głównego repozytorium
- ~17 platform
  - MBN_QUAIL -> STM32F427VG
  - GHI FEZ CERB40-I -> STM32F405RG
  - IngenuityMicro Electron -> STM32F411CE
  - IngenuityMicro Oxygen -> STM32F411CE
  - WeAct F411CE -> STM32F411CE
  - ST Nucleo64 F411RE -> STM32F411RE
  - ST_STM32F411_DISCOVERY -> STM32F411VE
  - ST Nucleo144 F412ZG -> STM32F412ZG
  - ST STM32F4DISCO -> STM32F407VGT6
  - ST Nucleo144 F439ZI -> STM32F439ZI
  - TI_CC1352P1_LAUNCHXL_868 TI -> CC1352
  - TI_CC1352P1_LAUNCHXL TI -> CC1352
  - NETDUINO3_WIFI -> STM32F427VI
  - Lilygo TWatch -> Esp32
- WiP - POSIX support (Linux/Nuttx/FreeRTOS posix)
- WiP - Raspberry Pi Pico RP2040 - blog microhobby



Dlaczego "herezje"?

- cytat ~"python i wszystkie inne języki interpretowane/wysokiego poziomu na mikrokontrolerach to zwykłe herezje" - "stara szkoła embedded"
- AspenCore - 2019 Embedded Markets Study link
- inny raport Jet Brains raport 2021 - jednak wliczając toole narzędziowe/testowe (1 miejsce python 28%)



Dlaczego "herezje"?

- wpis z bloga Qt: Embedded Software Programming Languages: Pros, Cons, and Comparisons of Popular Languages
  - wady języków wysokiego poziomu:
    - wydajność/narzut na interpreter
    - rozmiar wygenerowanych/dodatkowych binarek
    - **nie mogą być używane w systemach real-time**
    - **brak standardów przemysłowych**

Dlaczego C# w embedded komercyjnie?

- Z punktu widzenia biznesowego:
  - "nie odkrywamy koła na nowo" - możliwość skorzystania z bibliotek .NET
    - czyli też całe community C#/.NET
    - community nanoFrameworka na discordzie - 2574/~250 online
  - wykorzystując możliwości C# kod jest czytelniejszy->bardziej utrzymywalny niż zaawansowane projekty w C
  - szersza pula programistów do zatrudnienia
  - framework dla unitTestów logiki biznesowej w C# "out of the box" - powszechnie używane/ustandaryzowane narzędzia w C#
  - krótszy Time-To-Market bez uszczerbku na jakości projektu
  - DLL - Dynamic Link Libraries - firmowy use-case
  - wersjonowanie pakietów/bibliotek, menager pakietów NuGet
- Dodatkowe koszty w sprzęcie?
  - wydajność/dodatkowe zasoby - kwestia wymagań jakie ma spełniać urządzenie
    - ale istnieje możliwosc możliwość "mieszania" podejść
  - wbudowane wykorzystywanie koprocesorów jak FPU dla floatów

Zastosowania komercyjne

Advanced Application Controllers

Zone Controllers

Light & Blind Controllers

Wall Panels

Metering Gateway

Wireless Modbus RTU

Wireless Gateways

Multiprotocol I/O Modules

Modbus I/O Modules

użycie DLL w projekcie

- domyślnie urządzenia mają być nie tylko konfigurowalne, ale także rozbudowywalne przez klientów
- rozbudowane kontrolery automatyki budynkowej można przyrównać do sterowników PLC
- udostępnienie opracowanego przez nas Frameworka dla klientów
- uruchamianie dodatkowych zewnętrznych aplikacji/rozszerzeń pobranych np. z karty pamięci

QuickStart z punktu widzenia programisty .Net

Niezbędne narzędzia

- ogólne:
  - IDE -> Microsoft Visual Studio 2019/2022
  - .Net Framework 6.0+
- wydane przez nanoFramework
  - nanoFirmwareFlasher (nanoff) - pobierane przez .Net 6.0
    - CLI obsługujące wgrywanie binarek na urządzenie; obsługuje
      - port COM dla ESP
      - port COM dla ST w trybie DFU
      - JTAG (ST-Link) dla STM32
  - .NET nanoFramework Extension
    - wgrywanie i debugowanie z poziomu VS
    - podgląd wgranych pakietów na urządzeniu
    - wersje pakietów natywnych

Pierwsze wgrywanie

Za pomocą nanoff wgrywamy "nanoBooter" + "nanoCLR":

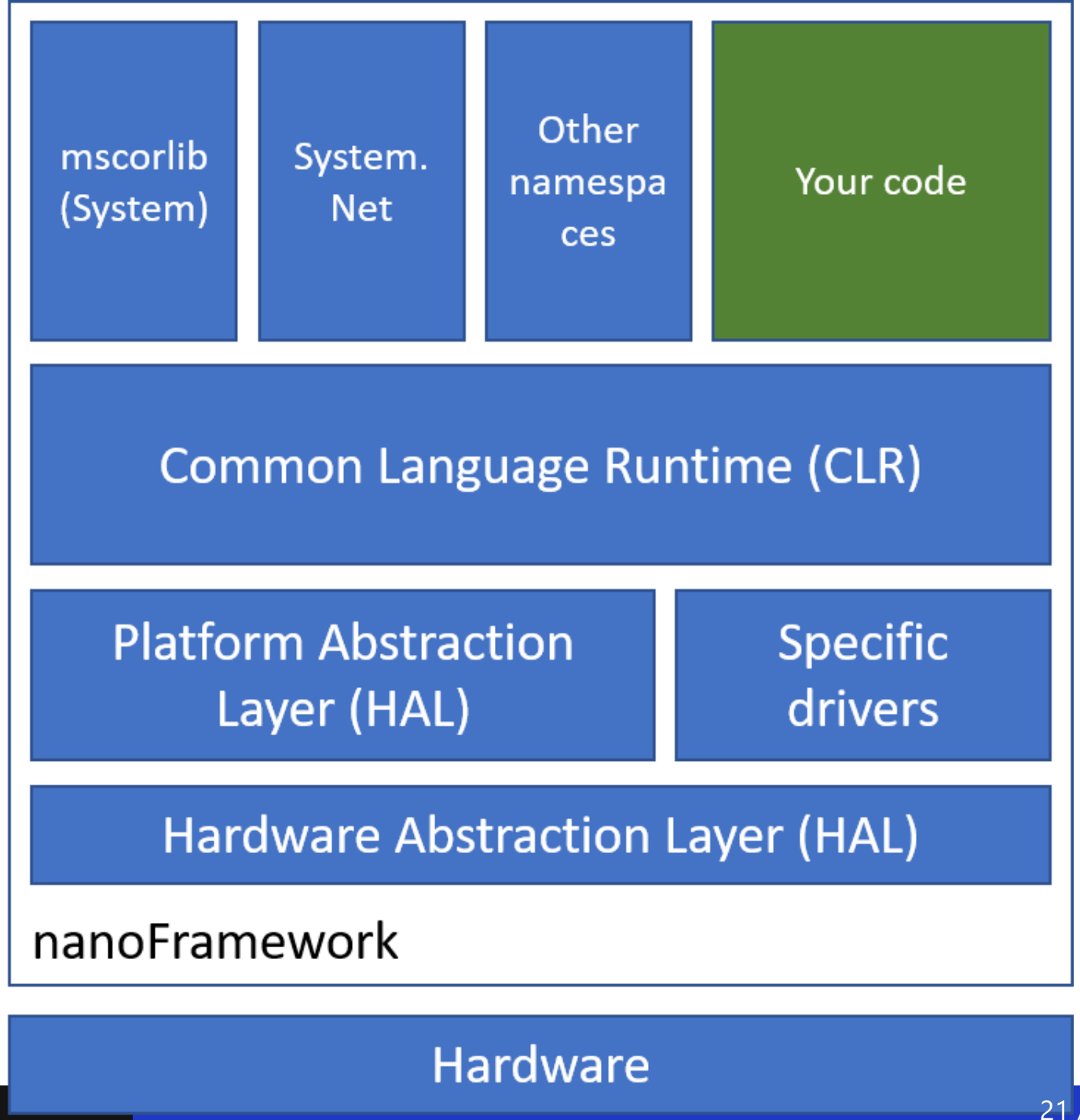
```
nanoff --platform esp32 --serialport COM31 --update --baud 115200
```

W tym przypadku zostanie dobrana odpowiednia wersja w zależności od użytego modułu pamięci na nim itp.

Tak przygotowane urządzenie będzie widoczne w VS2019 w pluginie NanoFrameworkowym w oknie "deviceExplorer".

co wgraliśmy?

- do sprawdzenia w "deviceExplorer" w Visualu



Embedowy Hello World - Blink

Repozytorium z przykładami - Samples

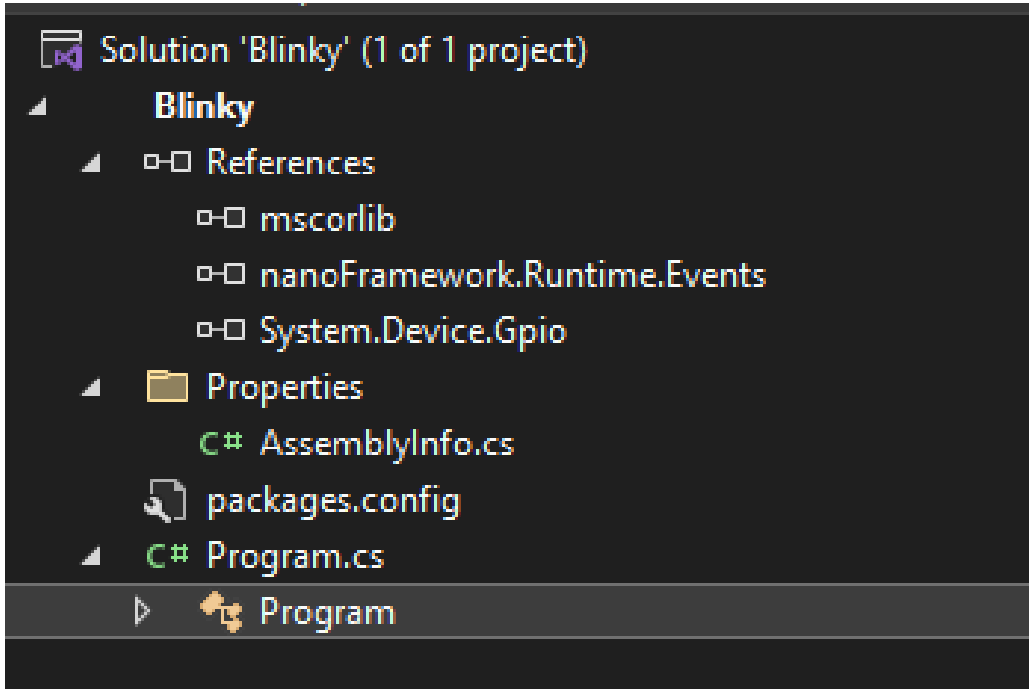
```
using System.Device.Gpio;
using System;
using System.Threading;
namespace Blinky
{
    public class Program
    {
        private static GpioController s_GpioController;
        public static void Main()
        {
            s_GpioController = new GpioController();

            GpioPin led = s_GpioController.OpenPin(2, PinMode.Output);

            led.Write(PinValue.Low);

            while (true)
            {
                led.Toggle();
                Thread.Sleep(125);
            }
        }

        static int PinNumber(char port, byte pin)
        {
            if (port < 'A' || port > 'J')
                throw new ArgumentException();
            return ((port - 'A') * 16) + pin;
        }
    }
}
```



Proces wgrywania

Logi z procesu wgrywania aplikacji:

```
1>------ Deploy started: Project: Blinky, Configuration: Debug Any CPU ------
1>Getting things ready to deploy assemblies to .NET nanoFramework device: ESP32 @ COM4.
1>Adding Blinky v1.0.8375.12076 (644 bytes) to deployment bundle
1>Adding System.Device.Gpio v1.1.22.60169 (5868 bytes) to deployment bundle
1>Adding mscorlib v1.12.0.4 (31832 bytes) to deployment bundle
1>Deploying 3 assemblies to device... Total size in bytes is 38344.
1>Deployment successful!
========== Deploy: 1 succeeded, 0 failed, 0 skipped ==========
========== Elapsed 00:03.026 ==========
```

Mapa pamięci urządzenia

Na przykładzie ESP32 w wersji 4MB.

```
++++++++++++++++++++++++++++++++
++        Memory Map          ++
++++++++++++++++++++++++++++++++
  Type     Start       Size
++++++++++++++++++++++++++++++++
  RAM   0x3ffe49ac  0x0001b000
  FLASH 0x00000000  0x00400000



++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++                   Flash Sector Map                             ++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  Region     Start      Blocks   Bytes/Block    Usage
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      0    0x00010000       1      0x1A0000     nanoCLR
      1    0x001B0000       1      0x1F0000     Deployment
      2    0x003C0000       1      0x040000     Configuration


++++++++++++++++++++++++++++++++++++++++++++++++++
++              Storage Usage Map                ++
++++++++++++++++++++++++++++++++++++++++++++++++++
  Start        Size (kB)           Usage
++++++++++++++++++++++++++++++++++++++++++++++++++
  0x003C0000    0x040000 (256kB)    Configuration
  0x00010000    0x1A0000 (1664kB)   nanoCLR
  0x001B0000    0x1F0000 (1984kB)   Deployment
```

Rozbieżność/brak bibliotek

Wszystkie wgrywane pakiety są weryfikowane pod kątem zgodności API.

```
Resolving.
Link failure: some assembly references cannot be resolved!!

Assembly: Blinky (1.0.8375.12076) needs assembly 'System.Device.Gpio' (1.1.22.60169)
Error: a3000000
The program '[1] .NET nanoFramework application: Managed' has exited with code 0 (0x0).
Assembly: System.Device.Gpio (1.1.22.60169) needs assembly 'nanoFramework.Runtime.Events' (1.11.1.42088)
Waiting for debug commands...
```

Informacje o bibliotekach w aplikacji/na urządzeniu

```
Assemblies:
  Blinky, 1.0.8375.12076
  System.Device.Gpio, 1.1.22.60169
  mscorlib, 1.12.0.4

Native Assemblies:
  mscorlib v100.5.0.17, checksum 0x004CF1CE
  nanoFramework.Runtime.Native v100.0.9.0, checksum 0x109F6F22
  nanoFramework.Hardware.Esp32 v100.0.7.3, checksum 0xBE7FF253
  nanoFramework.Hardware.Esp32.Rmt v100.0.3.0, checksum 0x0A915860
  nanoFramework.Device.OneWire v100.0.4.0, checksum 0xB95C43B4
  nanoFramework.Networking.Sntp v100.0.4.4, checksum 0xE2D9BDED
  nanoFramework.ResourceManager v100.0.0.1, checksum 0xDCD7DF4D
  nanoFramework.System.Collections v100.0.1.0, checksum 0x2DC2B090
  nanoFramework.System.Text v100.0.0.1, checksum 0x8E6EB73D
  nanoFramework.Runtime.Events v100.0.8.0, checksum 0x0EAB00C9
  EventSink v1.0.0.0, checksum 0xF32F4C3E
  System.IO.FileSystem v1.0.0.0, checksum 0x3AB74021
  System.Math v100.0.5.4, checksum 0x46092CB1
  System.Net v100.1.5.0, checksum 0x5BAB8CB3
  System.Device.Adc v100.0.0.0, checksum 0xE5B80F0B
  System.Device.Dac v100.0.0.6, checksum 0x02B3E860
  System.Device.Gpio v100.1.0.6, checksum 0x097E7BC5
  System.Device.I2c v100.0.0.1, checksum 0xFA806D33
  System.Device.I2s v100.0.0.1, checksum 0x478490FE
  System.Device.Pwm v100.1.0.4, checksum 0xABF532C3
  System.IO.Ports v100.1.6.1, checksum 0xB798CE30
  System.Device.Spi v100.1.2.0, checksum 0x3F6E2A7E
  System.Device.Wifi v100.0.6.4, checksum 0x1C1D3214
  Windows.Storage v100.0.2.0, checksum 0x954A4192
```

Rozbudowany przykład - webserver GPIO

```
using System;
using System.Threading;
using System.Diagnostics;
using nanoFramework.Networking;
using nanoFramework.WebServer;
using System.Device.Wifi;

namespace nanoFramework.WebServer.GpioRest
{
    public class Program
    {
        private static string MySsid = "2G-Vectra-WiFi-80E0BE"; private static string MyPassword = "internety112272g";

        private static bool _isConnected = false;

        public static void Main()
        {
            Debug.WriteLine("Hello from a webserver!");
            try
            {
                int connectRetry = 0;

                Debug.WriteLine("Waiting for network up and IP address...");
                bool success;
                CancellationTokenSource cs = new(60000);

                success = WifiNetworkHelper.ConnectDhcp(MySsid, MyPassword, requiresDateTime: true, token: cs.Token);

                if (!success)
                {
                    Debug.WriteLine($"Can't get a proper IP address and DateTime, error: {WifiNetworkHelper.Status}.");
                    if (WifiNetworkHelper.HelperException != null)
                    {
                        Debug.WriteLine($"Exception: {WifiNetworkHelper.HelperException}");
                    }
                    return;
                }

                using (WebServer server = new WebServer(80, HttpProtocol.Http, new Type[] { typeof(ControllerGpio) }))
                {
                    server.Start();

                    Thread.Sleep(Timeout.Infinite);
                }

            }
            catch (Exception ex)
            {
                Debug.WriteLine($"{ex}");
            }
        }
    }
}
```

Niezbędne pakiety:

```
using System;
using System.Threading;
using System.Diagnostics;
using nanoFramework.Networking;
using nanoFramework.WebServer;
using System.Device.Wifi;
```

Utworzenie serwera i wystartowanie:

```
using (WebServer server = new WebServer(80, HttpProtocol.Http, new Type[] { typeof(ControllerGpio) }))
{
    server.Start();

    Thread.Sleep(Timeout.Infinite);
}
```

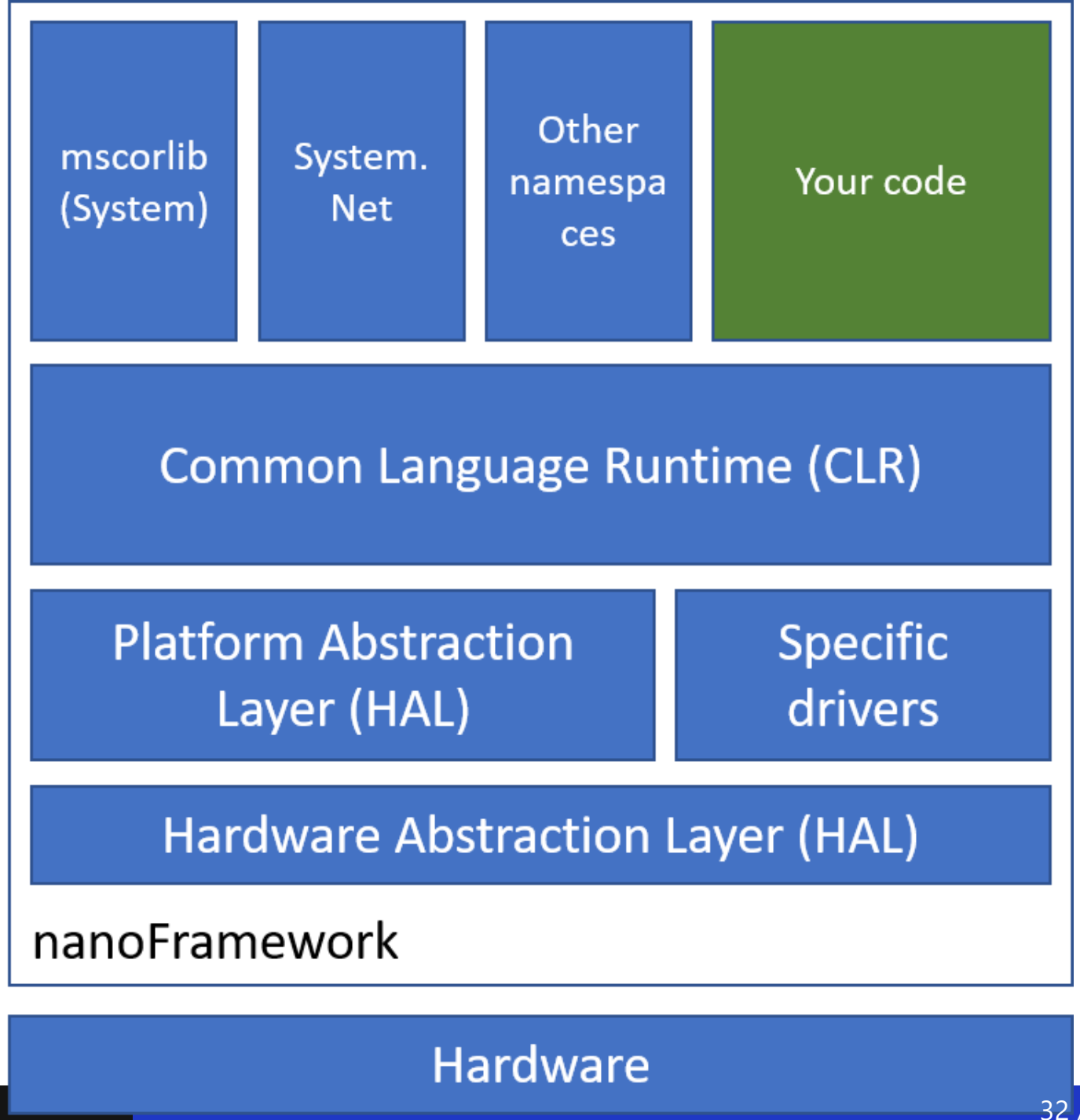
zajętość programu

```
1>Done building project "WebServer.GpioRest.nfproj".
2>------ Deploy started: Project: WebServer.GpioRest, Configuration: Debug Any CPU ------
2>Getting things ready to deploy assemblies to .NET nanoFramework device: ESP32 @ COM4.
2>Adding WebServer.GpioRest v1.0.0.0 (2916 bytes) to deployment bundle
2>Adding nanoFramework.WebServer v1.1.0.0 (8856 bytes) to deployment bundle
2>Adding Windows.Storage.Streams v1.14.19.64023 (6388 bytes) to deployment bundle
2>Adding System.IO.FileSystem v1.1.15.5532 (9796 bytes) to deployment bundle
2>Adding nanoFramework.System.Collections v1.4.0.3 (4096 bytes) to deployment bundle
2>Adding System.Device.Gpio v1.1.22.60169 (5868 bytes) to deployment bundle
2>Adding nanoFramework.System.Text v1.2.22.3995 (5828 bytes) to deployment bundle
2>Adding System.Net v1.10.38.33445 (20852 bytes) to deployment bundle
2>Adding System.Threading v1.1.8.6695 (3884 bytes) to deployment bundle
2>Adding System.Device.Wifi v1.5.37.9881 (7244 bytes) to deployment bundle
2>Adding nanoFramework.Runtime.Events v1.11.1.42088 (3412 bytes) to deployment bundle
2>Adding System.IO.Streams v1.1.27.27650 (6748 bytes) to deployment bundle
2>Adding System.Net.Http v1.5.61.54415 (30564 bytes) to deployment bundle
2>Adding mscorlib v1.12.0.4 (31832 bytes) to deployment bundle
2>Adding Windows.Storage v1.5.24.1018 (5324 bytes) to deployment bundle
2>Deploying 15 assemblies to device... Total size in bytes is 153608.
2>Deployment successful!
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Elapsed 00:09.385 ==========
```

Integracja C# i firmware

Zasięg prac

- do tej pory mieliśmy wpływ na sekcję "YourCode" i "Other namespaces"
- w celu wprowadzenia nowych funkcjonalności wymagających interakcji z HW/Firmware wpływ rozszerzamy o wszystkie warstwy poniżej Your code (oraz Other namespaces jako dostęp do nowego API)

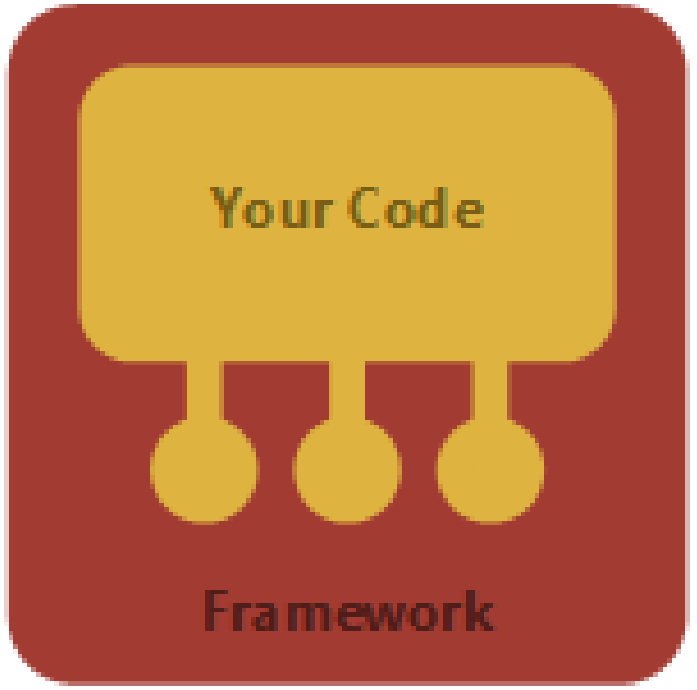


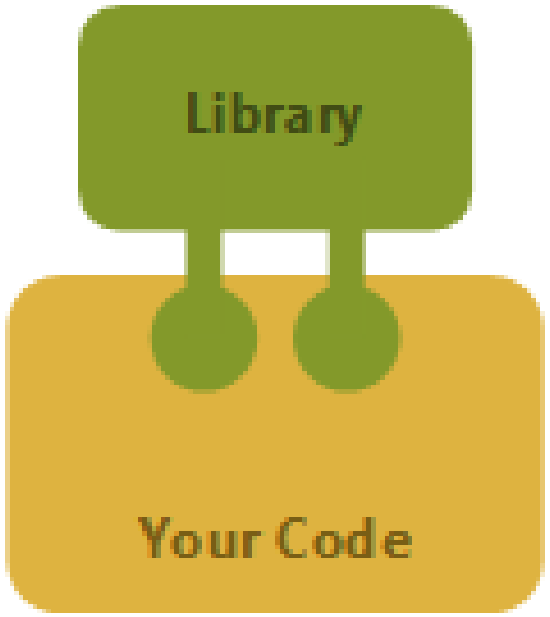
projekt nf-interpreter

- jest to wspólne repozytorium na projekty _nanoCLR oraz _nanoBooter.
- jest to dokładnie ten sam projekt, który wgrywaliśmy wcześniej korzystając z nanoff (FirmwareFlasher)
- link do repozytorium: nf-interpreter
  - build oparty o CMake + presety
  - proponowane IDE - VSC
  - dev containers - pełen build/debugowanie z wykorzystaniem dockera
    - bezproblemowa integracja z CI/CD

Framework vs. biblioteka

- w embeddedowych projektach mikrokontrolerowych zazwyczaj korzysta się z "bibliotek zewnętrznych"
- framework może wymagać dostosowania firmowego kodu/sposobu budowania projektu
- w przypadku frameworków openSource - można zawsze zaproponować i uzasadnić zmianę na forum





Kolejność uruchamiania aplikacji

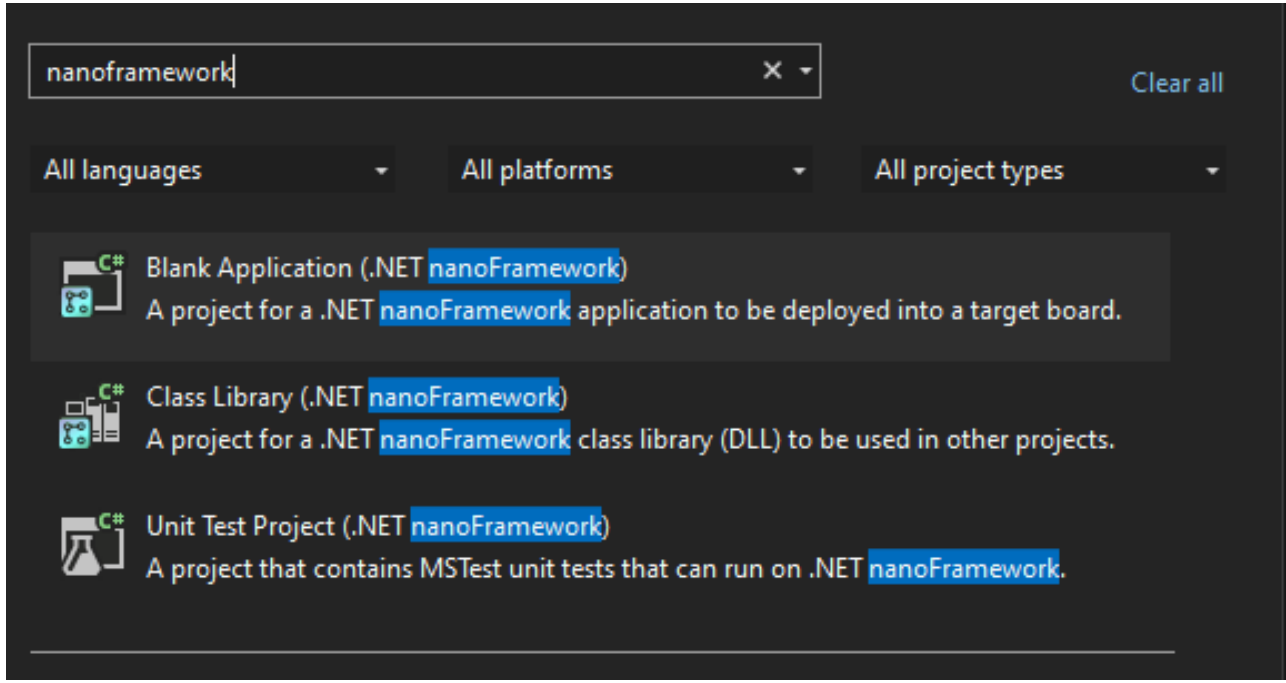
W jakiej kolejności startowane są aplikacje na urządzeniu:

- bootloader niskopoziomowy producenta układu ew. customowy firmowy bootloader
- nanoBooter
- aplikacja nanoCLR
  - konfiguracja pamięci, inicjalizacja peryferiów
  - konfiguracja RTOSa (nanoCLR nie działa w wersji "bare")
    - cały CLR jest uruchomiony jako pojedynczy wątek wykorzystywanego RTOSa

Na przykładzie z dokumentacji gdzie celem jest API do wyciągnięcia numeru seryjnego z mikrokontrolera STM32.

Procedura dodania nowego API z dostępem do HW/Firmware:

- utworzenie nowego projektu zawierającego nową klasę z API
  - jest to projekt dostępny po wgraniu extension dla VS - "Class Library"
- jego wynikiem są pliki:
  - dll - służacy jako referencja dla innych projektów w C#
  - "stuby" - wygenerowane nagłówki fukcji
    - należy je uzupełnić o kod danej funkcjonalności i dodać do projektu nf-interpretera + do głównego CMake budującego projekt



Oznaczenie funkcji jako natywnych - nowa klasa

```
using System.Runtime.CompilerServices;

namespace NF.AwesomeLib
{
public class Utilities
{

private static byte[] _hardwareSerial;

public static byte[] HardwareSerial
{
  get
  {
    if (_hardwareSerial == null)
    {
        _hardwareSerial = new byte[12];
        NativeGetHardwareSerial(_hardwareSerial);
    }

    return _hardwareSerial;
  }
}

#region Stubs

  [MethodImpl(MethodImplOptions.InternalCall)]
  private static extern void NativeGetHardwareSerial(byte[] data);

#endregion stubs
}

}
```

zoom na kod

Getter pola "HardwareSerial" w C#

```
public static byte[] HardwareSerial
{
  get
  {
    if (_hardwareSerial == null)
    {
        _hardwareSerial = new byte[12];
        NativeGetHardwareSerial(_hardwareSerial);
    }

    return _hardwareSerial;
  }
}
```

W tego typu klasach logika biznesowa może być "mieszana" - zarówno w C# jak i wywołania natywne.

zoom na kod

Funkcja oznaczona atrybutem jako "wywołanie natywne":

```
[MethodImpl(MethodImplOptions.InternalCall)]
private static extern void NativeGetHardwareSerial(byte[] data);
```

inny przykład - wartość zwracana i 2 argumenty:

```
[MethodImpl(MethodImplOptions.InternalCall)]
private static extern double NativeSuperComplicatedCalculation(double value);
```

Wygenerowany plik konfiguracyjny do dodania do nf-interpretera:

```
#include "NF_AwesomeLib.h"

static const CLR_RT_MethodHandler method_lookup[] =
{
    NULL,
    NULL,
    Library_NF_AwesomeLib_NF_AwesomeLib_Math::NativeSuperComplicatedCalculation___STATIC__R8__R8,
    NULL,
    NULL,
    Library_NF_AwesomeLib_NF_AwesomeLib_Utilities::NativeGetHardwareSerial___STATIC__VOID__SZARRAY_U1,
};

const CLR_RT_NativeAssemblyData g_CLR_AssemblyNative_NF_AwesomeLib =
{
    "NF.AwesomeLib",
    0xEE794AFB,
    method_lookup,
    { 1, 0, 0, 0 }
};
```

Dodatkowy plik pośredni odpowiadający za weryfikację:

```
HRESULT
Library_NF_AwesomeLib_NF_AwesomeLib_Utilities::NativeGetHardwareSerial___STATIC__VOID__SZARRAY_U1(
  CLR_RT_StackFrame& stack )
{
    NANOCLR_HEADER(); hr = S_OK;
    {

        CLR_RT_TypedArray_UINT8 param0;
        NANOCLR_CHECK_HRESULT( Interop_Marshal_UINT8_ARRAY( stack, 0, param0 ) );

        Utilities::NativeGetHardwareSerial( param0, hr );
        NANOCLR_CHECK_HRESULT( hr );

    }
    NANOCLR_NOCLEANUP();
}
```

Wygenerowana wersja mocka zawiera zabezpieczenie:

```
void Utilities::NativeGetHardwareSerial( CLR_RT_TypedArray_UINT8 param0, HRESULT &hr )
{
    NANOCLR_HEADER();
    {
        NANOCLR_SET_AND_LEAVE(stack.NotImplementedStub());
    }
    NANOCLR_NOCLEANUP();
}
```

Finalna implementacja z wpisaniem wartości do otrzymanego bufora:

```
void Utilities::NativeGetHardwareSerial( CLR_RT_TypedArray_UINT8 param0, HRESULT &hr )
{
  if (param0.GetSize() < 12)
  {
    hr = CLR_E_BUFFER_TOO_SMALL;
    return;
  }
  memcpy((void*)param0.GetBuffer(), (const void*)0x1FFF7A10, 12);
}
```

Funkcje przekazujące referencje

```
CLR_RT_HeapBlock *writeSpanByte;
CLR_RT_HeapBlock_Array *writeBuffer;
uint8_t *writeData = NULL;
int16_t writeSize = 0;
int16_t writeOffset = 0;
writeSpanByte = stack.Arg1().Dereference();
if (writeSpanByte != NULL)
{
    // get buffer
    writeBuffer = writeSpanByte[SpanByte::FIELD___array].DereferenceArray();
    if (writeBuffer != NULL)
    {
        // Get the write offset, only the elements defined by the span must be written, not the whole
        // array
        writeOffset = writeSpanByte[SpanByte::FIELD___start].NumericByRef().s4;

        // use the span length as write size, only the elements defined by the span must be written
        writeSize = writeSpanByte[SpanByte::FIELD___length].NumericByRef().s4;
        writeData = (unsigned char *)writeBuffer->GetElement(writeOffset);
    }
}
```

Dereferencja napisu

```
const char* szText = stack.Arg1().RecoverString();
// You can well check if it's a valid non null string like any other heap element:
FAULT_ON_NULL(szText);
```

Etap wywołania funkcji natywnych - uwagi

- Operacje na danych wołanych z C# są miejscem wymagającyn najwięcej uwagi
- wymagają szczegółowej wiedzy o typach danych w C++
- otrzymujemy tam "wprost" dostęp do danych/tablic CLRa
- wymagane sekcje krytyczne w przypadku współdzielenia danych
- brak sprawdzania typów
- wymaga testów modułowych
- implementacja natywna powinna być traktowana jak przerwania w bare-metal - wykonywać się nieblokująco i możliwie krótko
  - sprawdza się tutaj design-pattern "Active Object" z kolejkami rozkazów
    - książki Miro Samek/Quantum Leap

Pominięte wcześniej zalety

- dostęp do pakietów nanoFramework.IoT.Device
- wsparcie dla IoT korzystających z chmury Microsoftu
  - powstają już devkity certyfikowane przez Microsoft/Azure z obsługą nanoFrameworka
  - IoT bindings - funkcjonalność chmury Microsoftu odnośnie "parowania" ze sobą wejść i wyjść urządzeń
- możliwość implementacji funkcjonalności na poziomie nf-interpretera lub C# - w zależności od wymagań projektowych
- możliwość kompilacji tylko wymaganych bibliotek natywnych za pomocą presetów CMake

Wady tego rozwiązania

- oprócz wcześniej wymienionych wynikających z cech języka (narzut na interpreter itp)
- ogólny próg wejścia w przypadku firmy nie mającej styczności z .Net
  - wymagana infrastruktura CI/CD dla budowania pakietów
  - problem w .Net "dependency hell" związany z wymaganymi wersjami pakietów - wymagane jest właśce projektowanie/organizacja bibliotek
- konieczność reorganizacji bibliotek/systemu budowania w celu dostosowania do frameworka
- nadal brak ugruntowanej pozycji na rynku - chociaż community rośnie
  - (maj 2022 - 2 miliony pobrań pakietu z NuGet)
  - optymistycznie - brak bugów związanych z CLRem
  - jak pomóc -> how to contribute

Linki

- nanoFramework - oficjalna strona
- nanoFramework - dokumentacja
- nanoFramework - github
- IoT Show: An introduction to .NET nanoFramework
- IoT Show: How to connect .Net nanoFramework to the Cloud