NETMF for STM32 (F4 Edition)

Tour d'Horizon

Cuno Pfister & Beat Heeb

www.oberon.ch

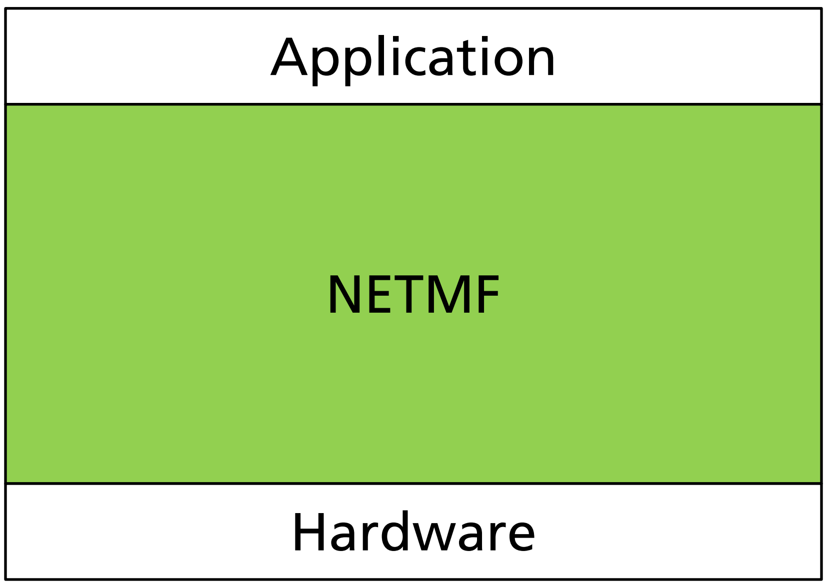
Overview

1. Architecture Qualities
2. Hardware
3. Solutions
4. Bootstrap
5. System Assemblies
6. CLR
7. PAL
8. HAL

1. Architecture Qualities

- Decomposing software systems into smaller components is driven by the desired *architecture qualities*
  - What are the main components of NETMF?
  - Which architecture qualities have led to this decomposition of NETMF?

NETMF as a Platform

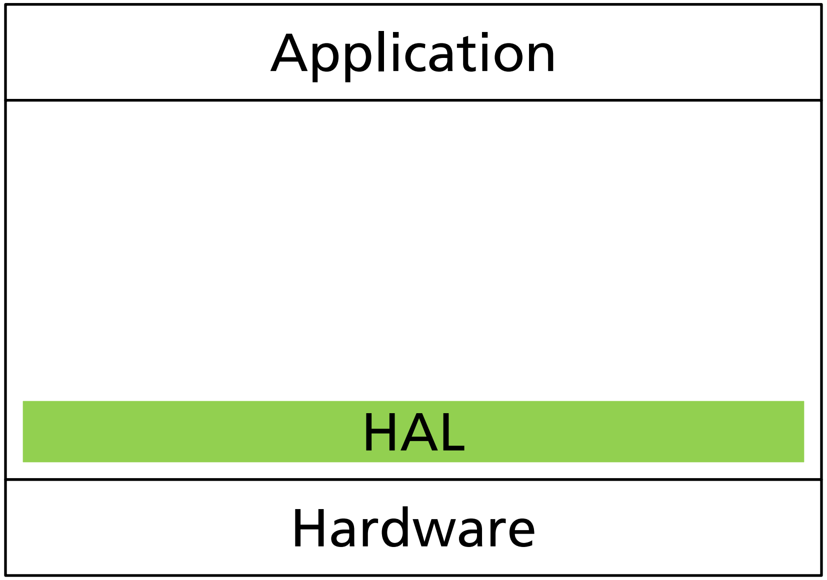


One C# or Visual Basic application runs at a time

The *.NET Micro Framework* (NETMF) is a «bootable runtime», i.e., no separate operating system is required

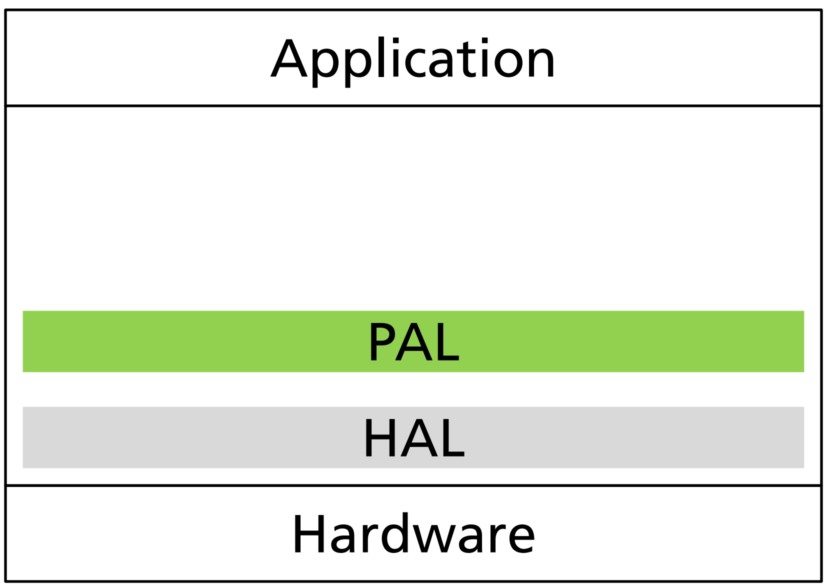
32-bit processor starting at < $10

Hardware Independence



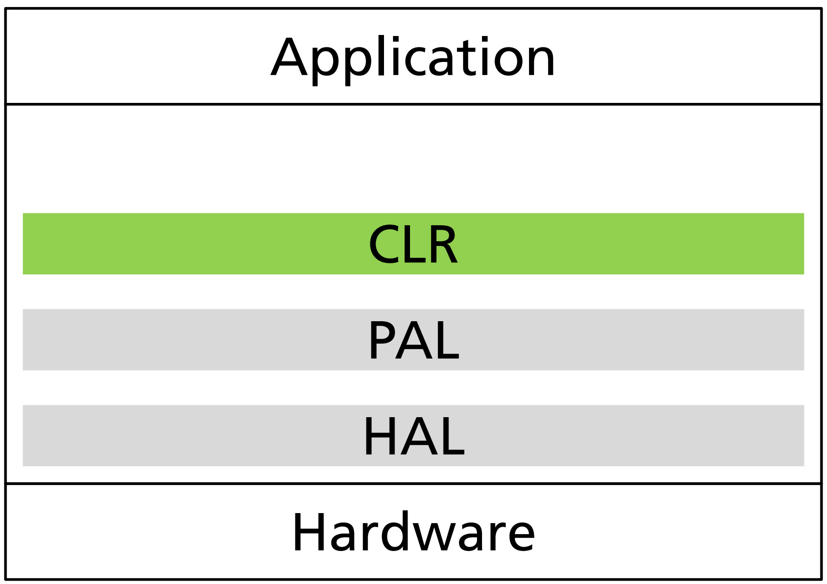
The *Hardware Abstraction Layer* (HAL) makes NETMF more portable by concentrating all device-dependent code in this layer. It consists of a collection of device drivers written in C/C++.

OS Independence



The *Platform Abstraction Layer* (PAL) is a rudimentary embedded operating system. It provides the minimal set of services necessary for executing .NET code. It is written in C/C++ and need not be modified when porting NETMF to different hardware.
It needs modifications only if NETMF is ported onto an existing operating system, which is not a relevant scenario for us here.

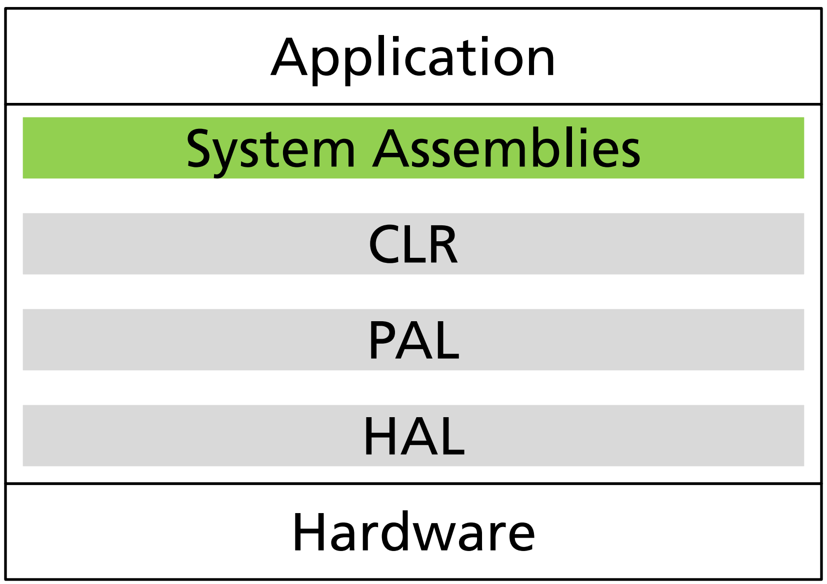
Reliability and Security



The *Common Language Runtime* (CLR) executes *Common Intermediate Language* (CIL) instructions, supports multi-threading, exceptions, allows debugging in Visual Studio, and manages memory (including automatic garbage collection). It is written in C/C++.

The CLR «manages» code in the sense that it guarantees memory and type safety of application code (e.g., no buffer overruns, no dangling pointers), even if the code contains accidental errors (reliability) or intentional errors (security).

Know-How, Tool and Code Reusability



System assemblies form a class library, partially written in C# and partially in C/C++, implements a (small) subset of the full .NET API, plus some special classes for the interaction with peripheral devices via GPIOs, analog inputs, PWMs, UARTs, I2C or SPI buses, etc. This allows reuse of existing C# know-how, of the Visual Studio tools, and to some degree also of actual C# code.

Overview

1. Architecture Qualities
2. Hardware
3. Solutions
4. Bootstrap
5. System Assemblies
6. CLR
7. PAL
8. HAL

2. Hardware

- Instruction Set Architectures
- Cores
- Microcontroller Chips
- Boards

ARMv7 Architectures

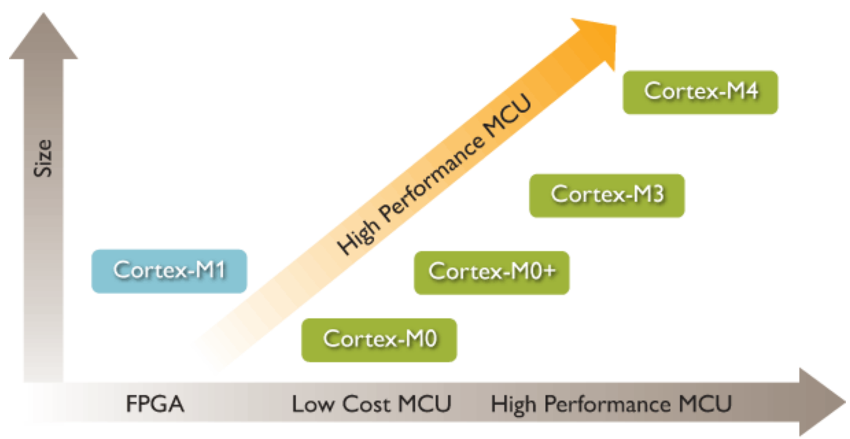
- ARMv7A
  - **A**pplication processors, high performance, for smartphones, tablets, servers etc.
- ARMv7R
  - **R**ealtime processors, medium performance, for automative apps etc.
- ARMv7M ← this is our focus!
  - **M**icrocontrollers for low-cost applications

ARMv7M Architecture

- Thumb2 Instruction Set
  - 16/32 bit instructions → high code density
- No MMU, typically no FPU and Caches
  - Optional Memory *Protection* Unit, 32-bit FPU
- Interrupt Controller
  - Flexible but complicated

ARM Cortex-Mx Cores

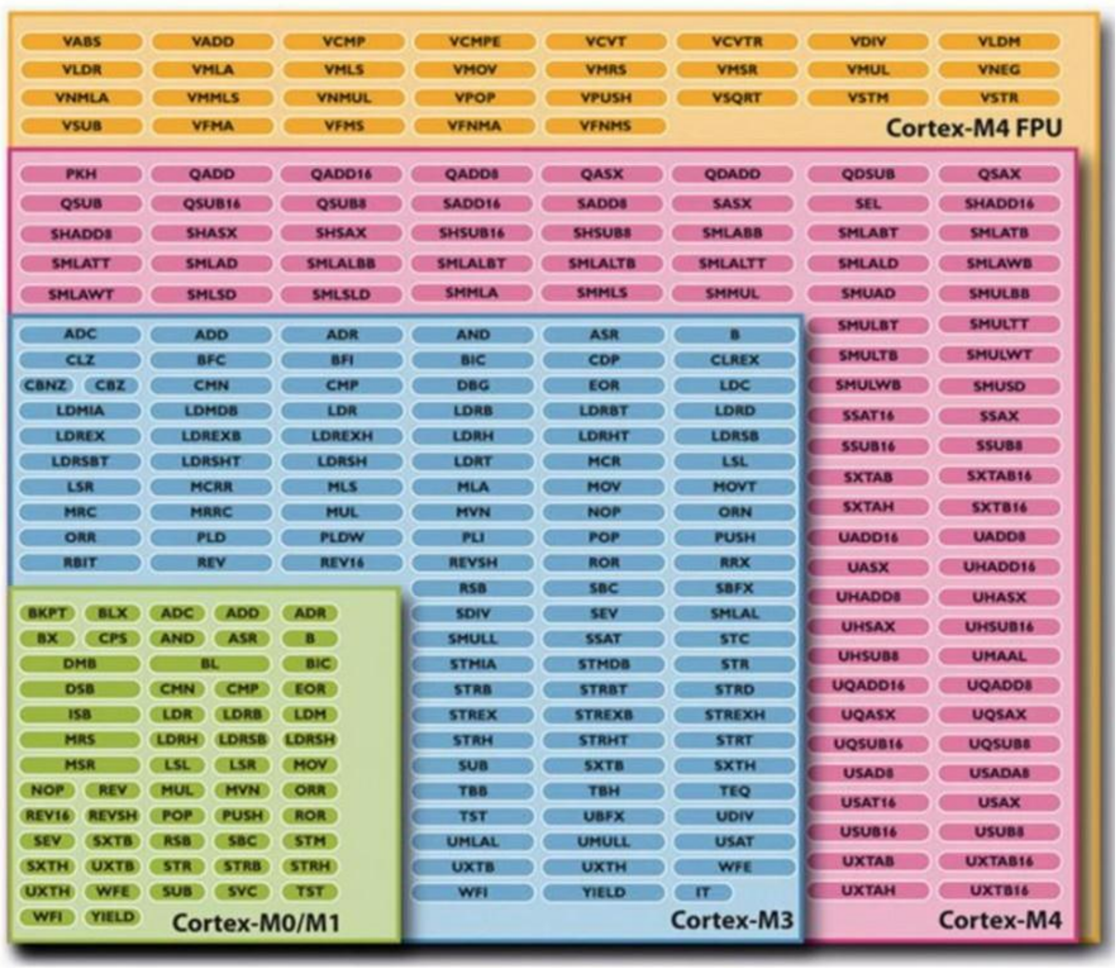
- A Core is an *Implementation* of an Architecture (as a design, not yet as silicon)
- Different ARMv7M Cores are called *Cortex* and numbered from M0 to M4
  - Higher numbers = higher performance



ARM Cortex-M4 Core

- Currently Highest End ARMv7M Core
  - Faster than Cortex-M3, e.g., 32-bit FPU
- Adds Specialized Instructions for Digital Signal Processing (DSP) Algorithms
  - Not relevant for NETMF
- Memory Protection Unit
  - Not relevant for NETMF

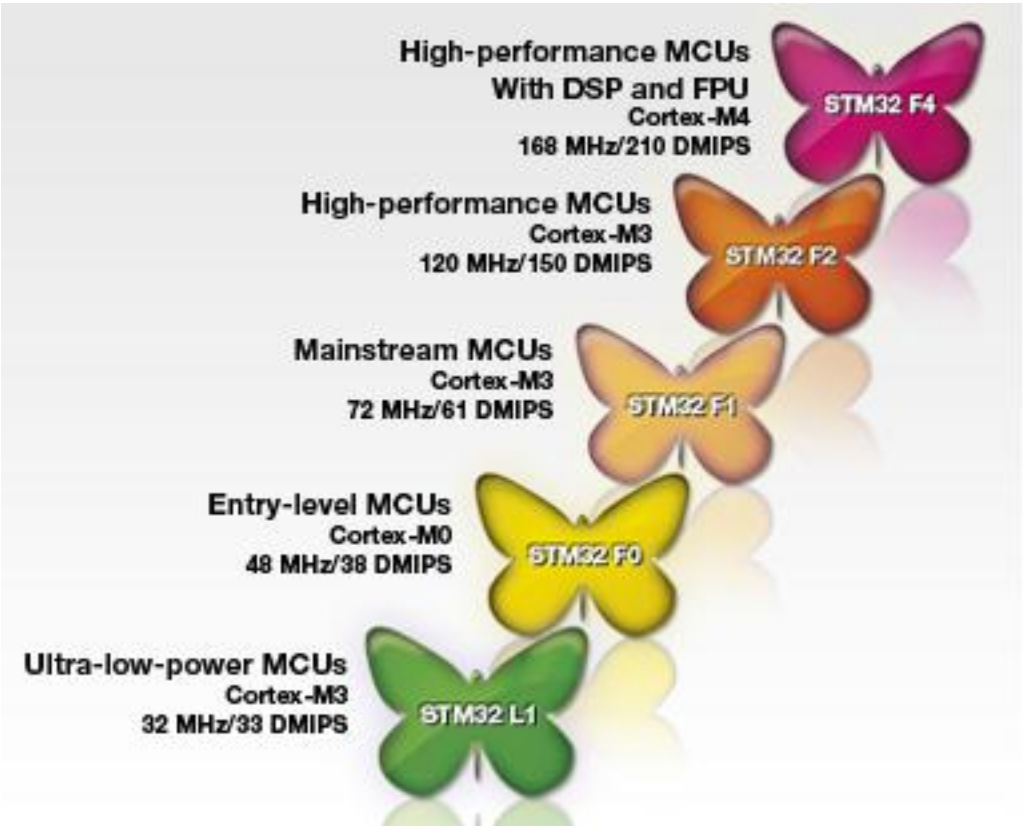
ARMv7M Instruction Subsets



Instruction subset differences are largely transparent when working with the ARM/Keil C compiler (compiler switches), except for the DSP instructions

Source:
www.elektroniknet.de

STM32 by STMicroelectronics



Required for NETMF
- $\geq$ 32 KB RAM
- $\geq$ 256 KB Flash
- STM32F101RC or better

Not suited for NETMF

STM32F4

- High End Microcontroller Family
  - Cortex-M4 core
  - 168 MHz
  - 192 KB RAM
  - 512 – 1024 KB Flash
  - e.g., STM32F407VG

Cortex-M4

no hardware support for encryption

Ethernet support

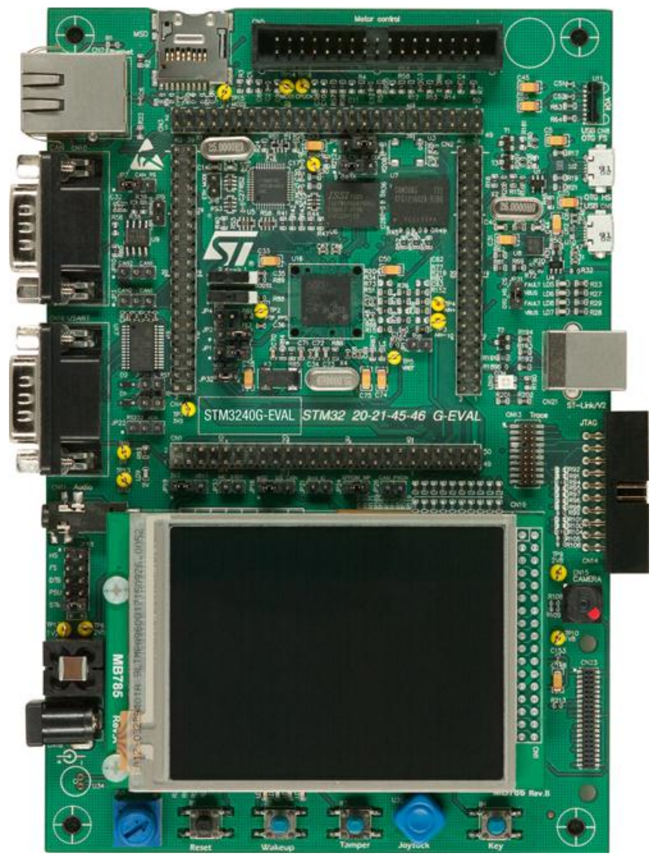
100 pin package

1 MB Flash

Overview here and more infos here

STMicroelectronics

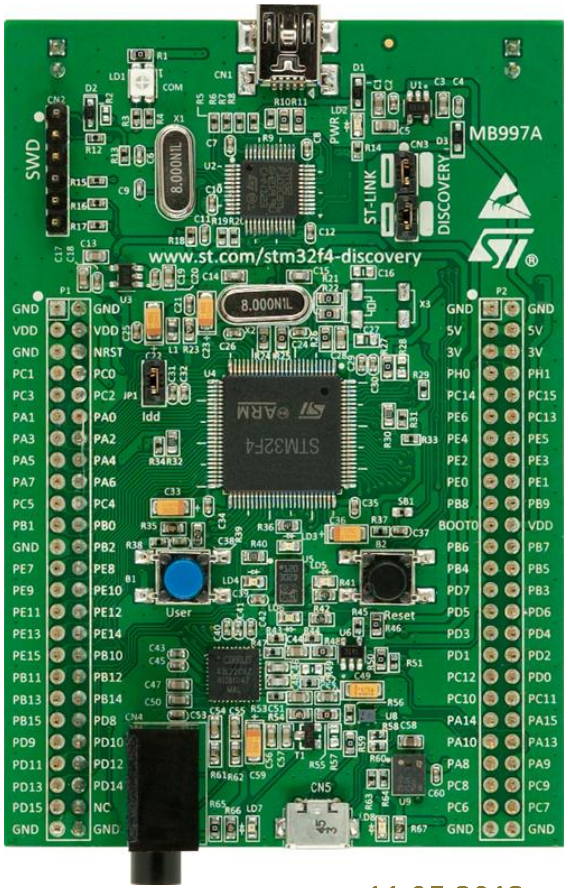
- STM3240G-Eval Evaluation Board
  - Breakout board for STM32F407IG, with connectors, external RAM, LCD display, etc.



- *NETMF for STM32*
  - Available, limited peripheral support (e.g., not for LCD)

STMicroelectronics

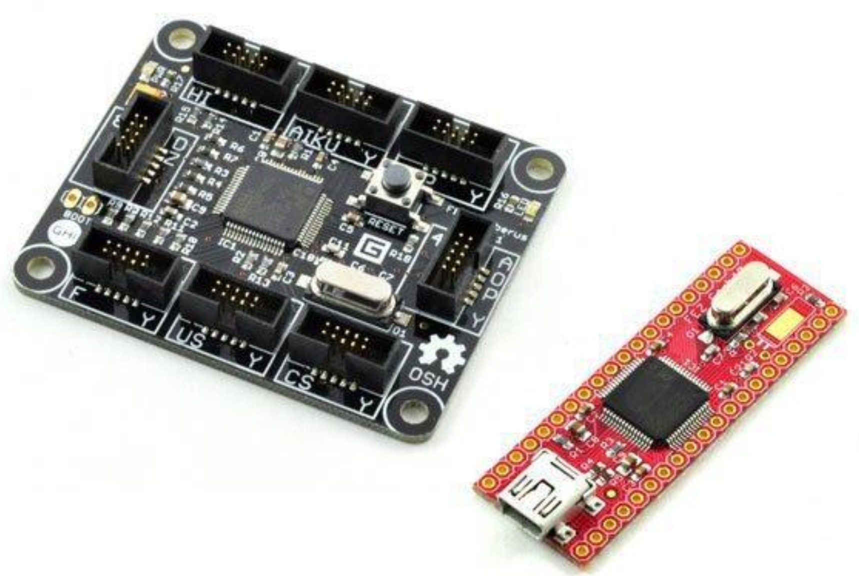
- Low-Cost STM32F4DISCOVERY Kit
  - STM32F407VG, audio DACs, etc.
- *NETMF for STM32*
  - Available



Note the separate ST-LINK debugging subsystem in the top third of the board, with its own STM32F1 microcontroller

GHI Electronics

- FEZ Cerberus
  - .NET Gadgeteer compatible open source hardware
  - Firmware will be based on *NETMF for STM32 (F4 Edition)*
  - STM32F405RG
- FEZ Cerb40
  - Breakout board for STM32F405RG
  - Same firmware



SecretLabs

- Netduino Go
  - Firmware is derived from Oberon microsystems' *NETMF for STM32 (F1 Edition)*
  - STM32F405RG
- *NETMF for STM32 (F4 Edition)*
  - Could be used as a firmware alternative

Mountaineer Boards

- Designed by CSA Engineering and Oberon microsystems

Coming soon...

NETMF for STM32

- NETMF runs on some STM32F1 Chips
  - *NETMF for STM32* available since Oct. 2011
- NETMF runs on almost all STM32F2 Chips
  - Drivers had to be extended, rewritten (USB) or added (Ethernet)
- NETMF runs on all STM32F4 Chips
  - Same as F2 edition, except for recompilation and larger RAM (identical peripherals to F2)
- NETMF could even run on some STM32**L**1 Chips
  - But CLR not optimal for ultra low-power applications

Overview

1. Architecture Qualities
2. Hardware
3. Solutions
4. Bootstrap
5. System Assemblies
6. CLR
7. PAL
8. HAL

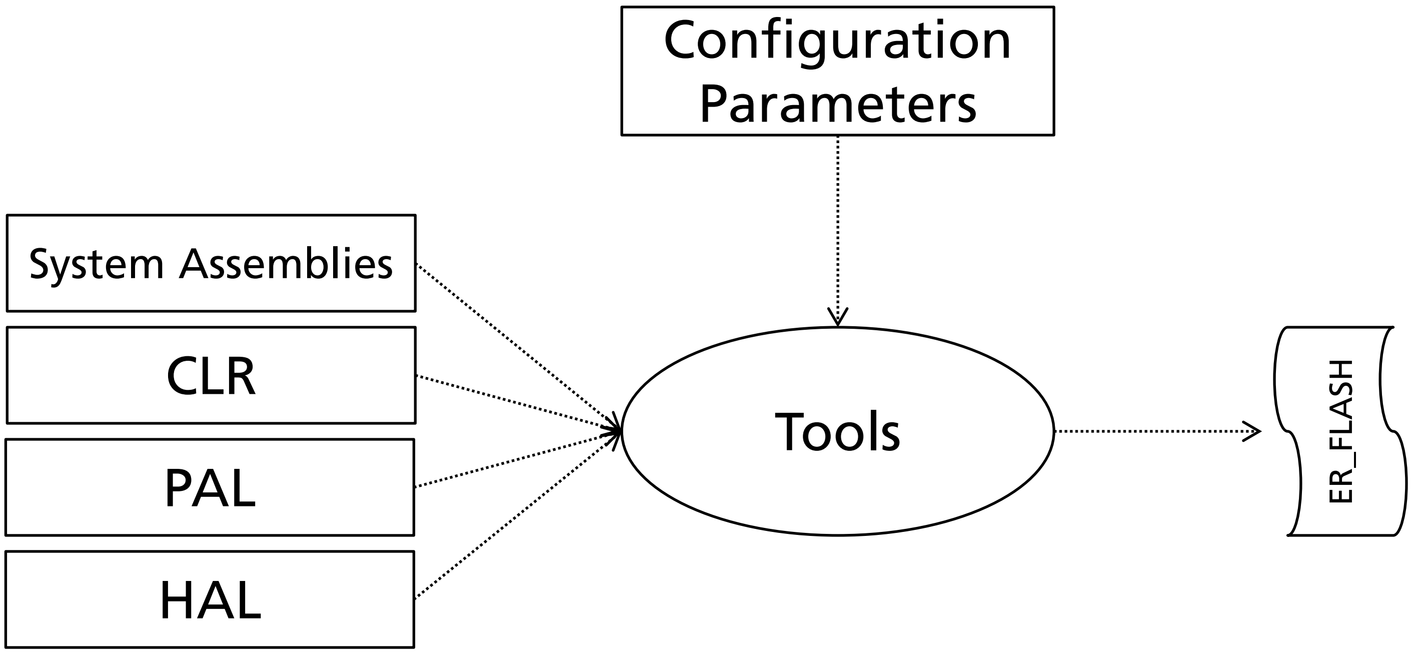
3. Solutions

- A *Solution* is one possible firmware configuration for a particular device
  - Usually there is one reference solution for a particular device, but there could be more than one, optimized for different purposes
    - e.g., one with and another one without Ethernet/TCP support
  - *C:\MicroFrameworkPK_v4_2\Solutions*

Creating New Solutions

- Select and Configure NETMF Components
  - Select components
  - Configure memory map
  - Configure peripheral drivers
- Build Firmware Image
  - Generate *ER_FLASH* file

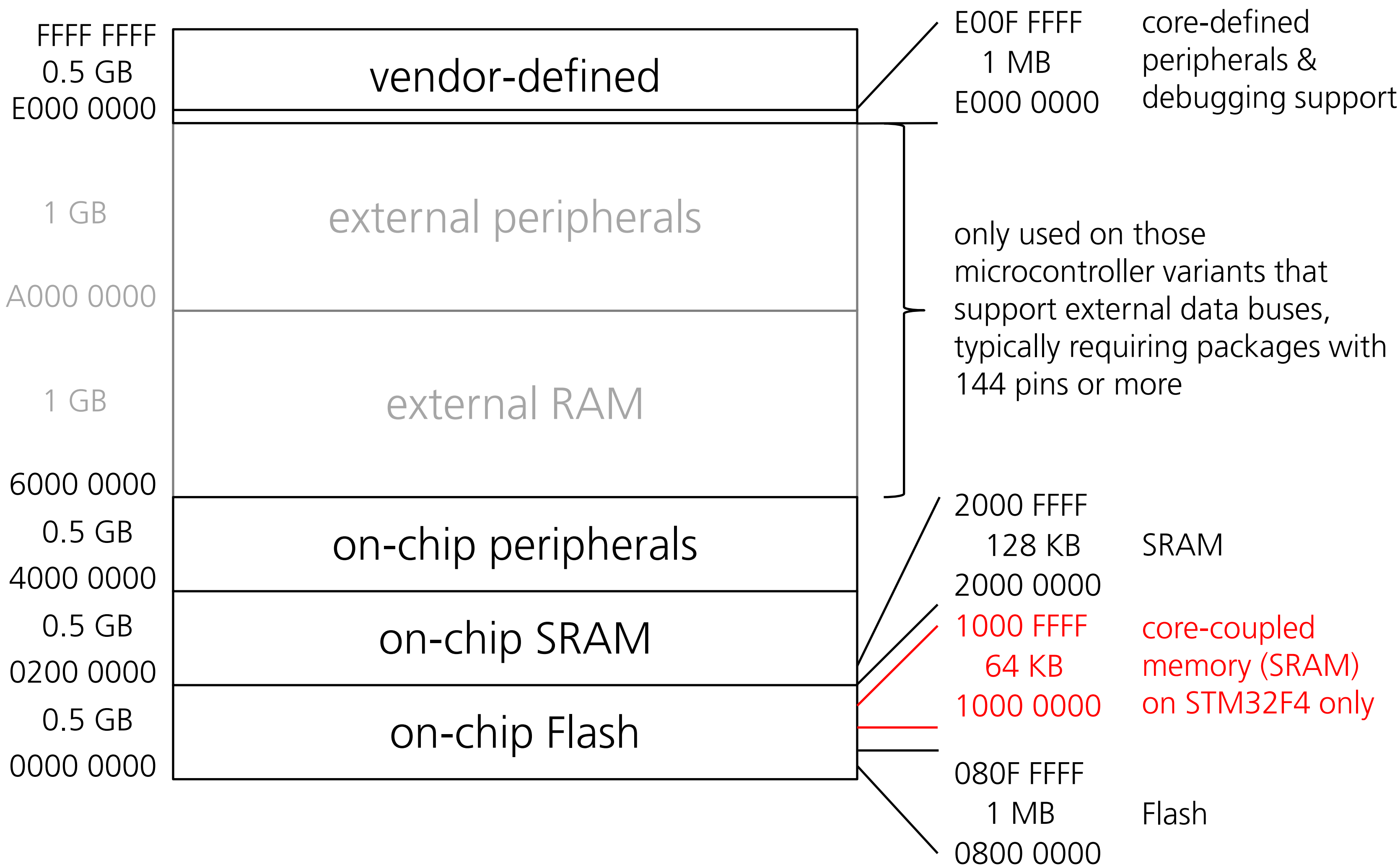
Build Firmware Image



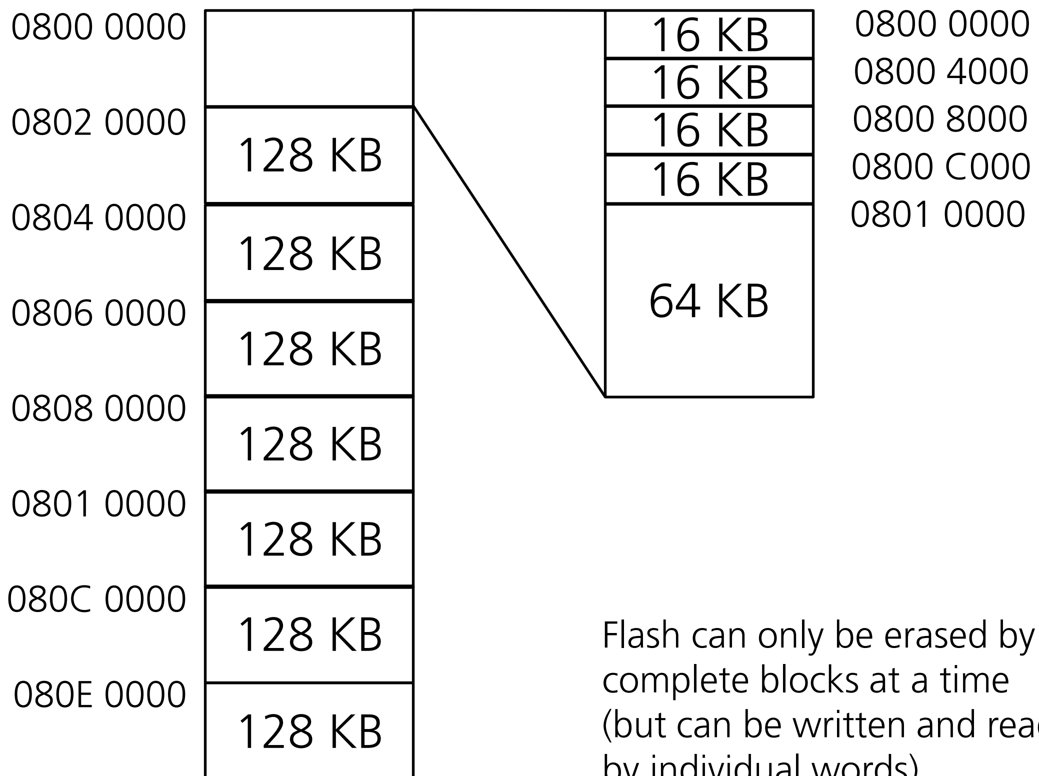
Select Components

- System Assemblies
  - Select assemblies that you want to provide
- CLR
  - Use entire CLR
- PAL Drivers
  - Use drivers for all supported OS functions
  - Use empty «stub» drivers for unsupported OS functions
- HAL Drivers
  - Use suitable drivers for all supported hardware
  - Use empty «stub» drivers for unsupported hardware

Cortex-M Memory Map

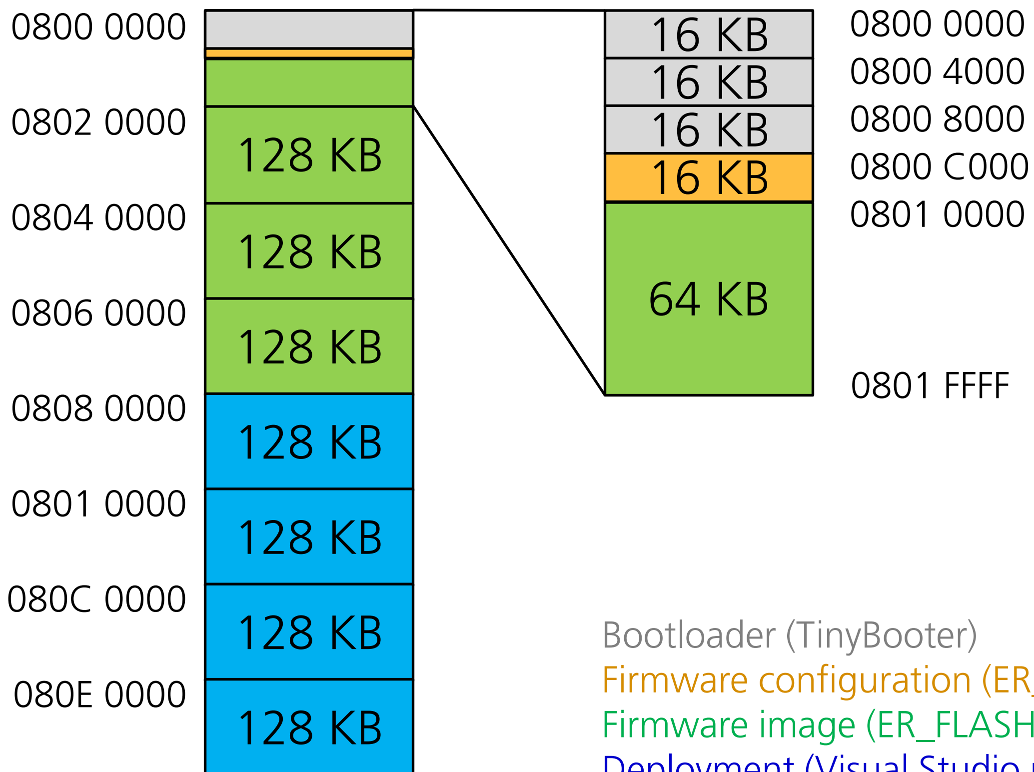


STM32F4 Flash Blocks



Flash can only be erased by complete blocks at a time (but can be written and read by individual words)

Example of Flash Usage



Bootloader (TinyBooter)
Firmware configuration (ER_CONFIG)
Firmware image (ER_FLASH)
Deployment (Visual Studio project)

Example Memory Map

Address Range	Type	~Size	Content	Comments
0800 0000 – 0800 BFFF	Flash	48 KB	Bootloader	TinyBooter (native code)
0800 C000 – 0800 FFFF	Flash	16 KB	Firmware config	MAC address, IP address, etc.
0801 0000 – 0807 FFFF	Flash	448 KB	Firmware image	NETMF (native & managed code)
0808 0000 – 080F FFFF	Flash	512 KB	Deployment	Application (managed code)
1000 0000 – 1000 3FFF	RAM	16 KB	Stack (NETMF)	Growing towards lower addresses
1000 4000 – 1000 FFFF	RAM	48 KB	Global variables and some buffers	UARTs, USB, etc.
2000 0000 – 2001 DFFF	RAM	120 KB	Heap	Application (objects & thread stacks)
2001 E000 – 2001 FDFF	RAM	7 KB	Buffers	Ethernet
2001 FE00 – 2001 FFFF	RAM	0.5 KB	Reserved	Interrupt handler table

This is a Flash and RAM memory map for the *MEM*.
We don’t provide space for Extended Weak References, as they are marginally usable in practice.

STM32F4 Memory-Mapped I/O

Address Range	Interface Type	Comments
4000 0000 – 4000 23FF	Timer	Registers for timers 2..7, 12..14
4000 3800 – 4000 43FF	SPI	Registers for SPI 2..3
4000 4400 – 4000 53FF	USART	Registers for USART 2..5
4000 5400 – 4000 5FFF	I2C	Registers for I2C 1..3
4000 7400 – 4000 77FF	DAC	Registers for DAC
4001 0000 – 4000 07FF	Timer	Registers for timers 1, 8
4001 1000 – 4001 17FF	USART	Registers for USART 1, 6
4001 2000 – 4001 23FF	ADC	Registers for ADC 1..3
4001 3000 – 4001 33FF	SPI	Registers for SPI 1
4001 4000 – 4001 4BFF	Timer	Registers for timers 9..11
4002 0000 – 4002 23FF	GPIO	Registers for using GPIO A0..A15 to GPIO I0..I15
4002 3C00 – 4002 3FFF	Flash	Registers for programming the on-chip Flash
4002 8000 – 4002 93FF	Ethernet MAC	Registers for programming the on-chip Ethernet MAC
4004 0000 – 5003 FFFF	USB OTG	Registers for programming the on-chip USB interfaces

Pin Assignment

- Assign Peripheral Functions to Pins
  - All I/O pins of an STM32 can be configured as digital inputs or outputs (GPIO)
  - Some pins can alternatively be configured as other peripheral pins, e.g., as Tx of a UART
  - Complication:
    the same peripheral may be accessible through several pins
    - More possible configurations for larger packages

Solution Example

Flash	RAM	Binary file	Remarks
40	0	IO_Init_<sol>.lib	Initialize GPIO pins, external memory bus, etc.
250	0	usb_pal_config_<sol>.lib	Initialize USB endpoints, used protocols, etc.
148	68	STM32F2_blconfig_<sol>.lib	Configure Flash memory map
24	0	BlockStorage_AddDevices_<sol>.lib	Register Flash areas (only changed for external Flash)
28	12	DebuggerPort_SSL_config_stubs.lib	SSL is not supported (empty stub)
2	0	FS_Config_stubs.lib	File system is not supported (empty stub)
492	80	Total size in bytes	

This is a solution used for the
MUM

Where to Change?

- In C:\MicroFrameworkPK_v4_2\Solutions\<sol>\
  - platform_selector.h
  - <sol>.settings
  - DeviceCode\
    - Blockstorage\addDevices\Bl_addDevices.cpp
    - Init\IO_Init.cpp
    - USB\usb_config.cpp
  - TinyBooter\
    - if boot behavior should be modified, e.g. stay in TinyBooter upon key press
  - TinyCLR\
    - scatterfile_tinyclr_mdk.xml
    - TinyCLR.proj

Overview

1. Architecture Qualities
2. Hardware
3. Solution
4. **Bootstrap**
5. System Assemblies
6. CLR
7. PAL
8. HAL

4. Bootstrap

- TinyBooter → NETMF → Application

TinyBooter

- Processor starts TinyBooter after Reset
  - At the beginning of the Flash (0x0800 0000)
  - See *<sol>\TinyBooter\TinyBooterEntry.cpp*
  - TinyBooter either goes into a command interpreter, or
  - Starts the CLR
    - by looking for a Flash sector that starts with a CLR marker word
    - performs optional verification of CLR signature
    - C:\MicroFrameworkPK_v4_2\Application\TinyBooter\Commands.cpp

NETMF

- Start NETMF Firmware
  - Marker word is first instruction
  - Performs a second initialization of the hardware
  - NETMF either goes into a command interpreter, or
  - starts the managed application
    - by calling the *Main* method of the *Startup object* that had been set in the properties of the Visual Studio project
    - performs optional verification of application signature
    - NETMF firmware is a self-contained application, i.e., it could be started by any bootloader, not just TinyBooter – or directly after reset

Application

- Start Application

How to Deploy

- TinyBooter
  - Deploy *tinybooter.bin* using STMicroelectronics' *Flash Loader Demo* or its *ST-LINK* utility (for boards with ST-LINK)
- NETMF
  - Deploy *ER_CONFIG* and *ER_FLASH* using *MFDeploy*
- Application
  - Deploy a C# project using *Visual Studio*

Overview

1. Architecture Qualities
2. Hardware
3. Solutions
4. Bootstrap
5. System Assemblies
6. CLR
7. PAL
8. HAL

5. System Assemblies

- No Changes Needed!

System Assemblies Overview

Flash	RAM	Binary file	Remarks
69900	0	tinyclr_dat.obj	Consists of • mscorlib • Microsoft.SPOT.Native • Microsoft.SPOT.Hardware • Microsoft.SPOT.Hardware.PWM • Microsoft.SPOT.Hardware.SerialPort • Microsoft.SPOT.Net • System
69900	**0**		

These are the system assemblies as used for the *MUM*

Overview

1. Architecture Qualities
2. Hardware
3. Solutions
4. Bootstrap
5. System Assemblies
6. CLR
7. PAL
8. HAL

6. CLR

- No Changes Needed!
  - The CLR can be regarded as a black box…
  - but if you are interested:
    - CLR sources are in *C:\MicroFrameworkPK_v4_2\CLR*
    - For the bootstrap, see
      - *…\STM32\DeviceCode\CortexM3\TinyHal\FirstEntry.s*
      - *…\STM32\DeviceCode\STM32_Bootstrap\STM32_bootstrap.cpp*
      - *C:\MicroFrameworkPK_v4_2\DeviceCode\Initialization\tinyhal.cpp*

Common Language Runtime

- Interpreter
- Garbage collector
  - With heap compaction
  - Custom heap allows non-collected objects
- Light-weight C# threads
  - About 400 bytes per thread
  - Stack frames are threaded within heap
  - Interpreter reschedules every 20 ms

Sleep When Idle

- Typical NETMF applications wait for I/O events
- NETMF goes into a power-saving sleep mode until some event occurs
  - e.g., GPIO input has changed its state, *Socket.Receive* has new data available

I/O: Synch vs. Asynch

- APIs for the System Assemblies are mostly synchronous
  - e.g., blocking *Socket.Receive* operations
- API of the PAL is mostly asynchronous
  - Ethernet, WiFi, USB, USART, I2C, GPIO (int.)
  - Driver completes I/O upon interrupt
  - Driver signals completed I/O to interpreter
  - Interpreter schedules an application thread

Cooperative Multitasking

- PAL uses a single native thread
  - Thus a single C/C++ stack is sufficient (shared with the interrupts, see below)
- HAL drivers may use interrupts
  - All interrupts run at the same level
  - Interrupt handlers may run at most for 20 ms
  - Interrupt handlers may spawn «completions» if they need to do more work later on

Completions & Continuations

- Timed «Run To Completion» Tasks
  - Completions for critical tasks
    - Executed within timer interrupt handler
  - Continuations for non-critical tasks
    - Executed after the timeout has expired and the CLR is idle (i.e., has no thread to schedule)
  - In our drivers, we didn’t need any such tasks
    - Except for Ethernet: detect cable (completion), receive data (continuation)

CLR Example

Flash	RAM	Binary file	Remarks
22	13888	InteropAssembliesTable.lib	Method resolution caches, can be configured in size
164	0	tinyclr.obj	
149945	2798	tmp_tinyclr.lib	The actual language runtime (virtual machine)
150131	**16686**		

This is the CLR as used for the *MUM*

Overview

1. Architecture Qualities
2. Hardware
3. Solution
4. Bootstrap
5. System Assemblies
6. CLR
7. PAL
8. HAL

7. PAL

- No Changes Needed!
  - When porting NETMF onto the «bare metal», the PAL is not touched (but some functions may be stubbed)
  - A TCP/IP stack (lwIP) is contained in *C:\MicroFrameworkPK_v4_2\DeviceCode\pal\lwip*
  - PAL sources are contained in *C:\MicroFrameworkPK_v4_2\DeviceCode\pal*

PAL Example

Flash	RAM	Binary file
24	0	heap_pal.lib
76	16	palevent_pal.lib
396	44	events_pal.lib
436	580	Buttons_pal.lib
468	4	tinycrt_pal.lib
486	0	native_double_pal.lib
810	0	COM_pal.lib
920	24	asyncproccall_pal.lib
1543	0	config_pal.lib
1716	44	Time_pal.lib
1956	20	blockstorage_pal.lib
2640	3600	usart_pal.lib
3046	66	usb_pal.lib
3841	140	system_initialization_hal.lib

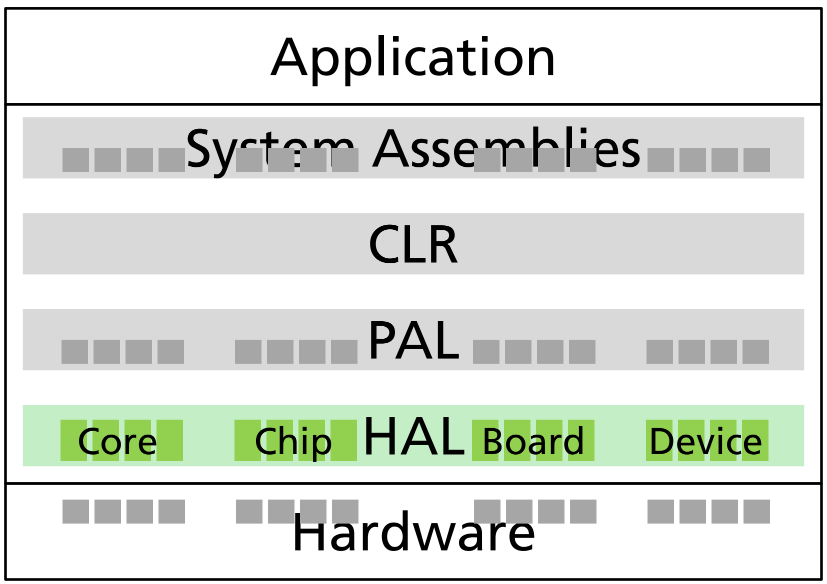
Flash	RAM	Binary file
4	0	ssl_pal_stubs.lib
8	0	piezo_pal_stubs.lib
12	0	Gesture_pal_stubs.lib
12	0	Ink_pal_stubs.lib
12	0	TimeService_pal_stubs.lib
12	0	fs_pal_stubs.lib
16	0	Watchdog_pal_stubs.lib
38	0	sockets_pal_stubs.lib
40	0	MFUpdate_PAL_stubs.lib
19250	**4554**	**Total size in bytes**

This is the PAL as used for the *MUM*

Overview

1. Architecture Qualities
2. Hardware
3. Solutions
4. Bootstrap
5. System Assemblies
6. CLR
7. PAL
8. HAL

8. HAL



We can distinguish between different types of HAL components, according to the kind of hardware components that they support. These are the so-called *drivers*. Usually there is a driver counterpart in the PAL for every HAL driver, often also in the system assemblies.

Core Support Components

- Core Initialization
  - Interrupt controller
- How about SysTimer?
  - This in-core timer is not used
    - NETMF needs a higher resolution «real-time» timer (ideally with 100 ns precision)
    - We combine two timers for 48 bit resolution
    - *C:\MicroFrameworkPK_v4_2\DeviceCode\Targets\Native\STM32\DeviceCode\STM32_Time*

Core Support Components

- Global Lock Mechanism
  - Use object constructors/destructors to enter/exit monitors
  - Implemented by disabling/enabling interrupts
- *C:\MicroFrameworkPK_v4_2\DeviceCode\Targets\Native\STM32\DeviceCode\CortexM3*
  - Also valid for Cortex-M4

Chip Support Components

- Drivers for on-chip peripherals
- Interrupts used by their respective drivers
  - Timer, GPIO, USART, I2C, USB, Ethernet
- *C:\MicroFrameworkPK_v4_2\DeviceCode\Targets\Native\STM32\DeviceCode\STM32_**
- F1 drivers can differ from F2/F4 drivers
  - F1 "XL" chips have newer peripherals as well

Board Support Components

- Drivers for on-board Peripherals
- *C:\MicroFrameworkPK_v4_2\Solutions*

Device Support Components

- Drivers for off-board Peripherals
- *C:\MicroFrameworkPK_v4_2\Solutions*
  - e.g., driver for Micron M25P64 serial Flash
- or *C:\MicroFrameworkPK_v4_2\DeviceCode\Drivers*
  - e.g., driver for Microchip ENC28J60 external Ethernet controller

HAL Example

Flash	RAM	Binary file
122	0	GlobalLock_hal_Cortex.lib*
412	356	TinyHal_Cortex.lib*
192	0	STM32F2_Power.lib**
200	0	STM32F2_Analog.lib**
204	0	STM32F2_IntC.lib**
224	0	STM32F2_bootstrap.lib**
402	52	STM32F2_Flash.lib**
476	8	STM32F2_time.lib**
752	0	STM32F2_PWM.lib**
879	12	STM32F2_SPI.lib**
984	8	STM32F2_I2C.lib**
1356	24	STM32F2_USART.lib**
1404	144	STM32F2_GPIO.lib**
1446	3502	STM32F2_USB.lib**

Flash	RAM	Binary file
2	0	cpu_watchdog_stubs.lib
2	0	cpu_cache_stubs.lib
4	0	cpu_prestackinit_stubs.lib
6	0	virtualkey_hal_stubs.lib
8	0	LargeBuffer_hal_stubs.lib
8	0	SimpleHeap_config_stubs.lib
8	0	SimpleHeap_stubs.lib
8	0	backlight_hal_stubs.lib
8	0	LargeBuffer_hal_stubs.lib
8	0	SimpleHeap_config_stubs.lib
8	0	SimpleHeap_stubs.lib
18	0	batterycharger_hal_stubs.lib
26	0	lcd_hal_stubs.lib
46	0	batterymeasurement_hal_stubs.lib
9189	4106	Total size in bytes

* Core support for Cortex-M3 / Cortex-M4
** Chip support for STM32F2 / STM32F4

This is the HAL as used for the *MUM*

CLib

Flash	RAM	Binary file
6346	0	fz_ws.l
8133	70	c_w.l
13104	0	m_ws.l
27583	**70**	**Total size in bytes**

CLib mainly used for formatted output. Could probably still be optimized.

Several versions available, e.g., one that does not use floating point numbers.

This is the CLib as used for the *MUM*

Total Size of NETMF

Flash	RAM	Binary file
69900	0	System Assemblies
27583	70	CLib
492	80	Solution
9189	4106	HAL
19250	4554	PAL
150131	16686	CLR
276545	**25496**	**Total size in bytes**

This is NETMF as used for the *MUM*

End of our Tour!