NETMF for STM32

MFUpdate

Cuno Pfister, Beat Heeb, Pascal Spörri
with special thanks to Mark Munte
www.oberon.ch

Overview

1. MFUpdate
2. Build System
3. NMF Files
4. Managed Code API
5. Native Code API
6. Extended TinyBooter
7. Summary

Requirements

- Remote updates
  - Firmware (C/C++ code, system assemblies)
  - Application (application assemblies)
  - Data (e.g., configuration, keys)
- Extensibility dimensions
  - Protocols, storage types, optional backups, compression, security

Microsoft MFUpdate

- Lightweight framework with
  - Managed API that allows to implement most of an update mechanism in C#
    - Implement your own custom update strategies, fully within C# and Visual Studio
  - Native API that allows to implement the rest in C++
    - Microsoft provides some standard implementations as starting points

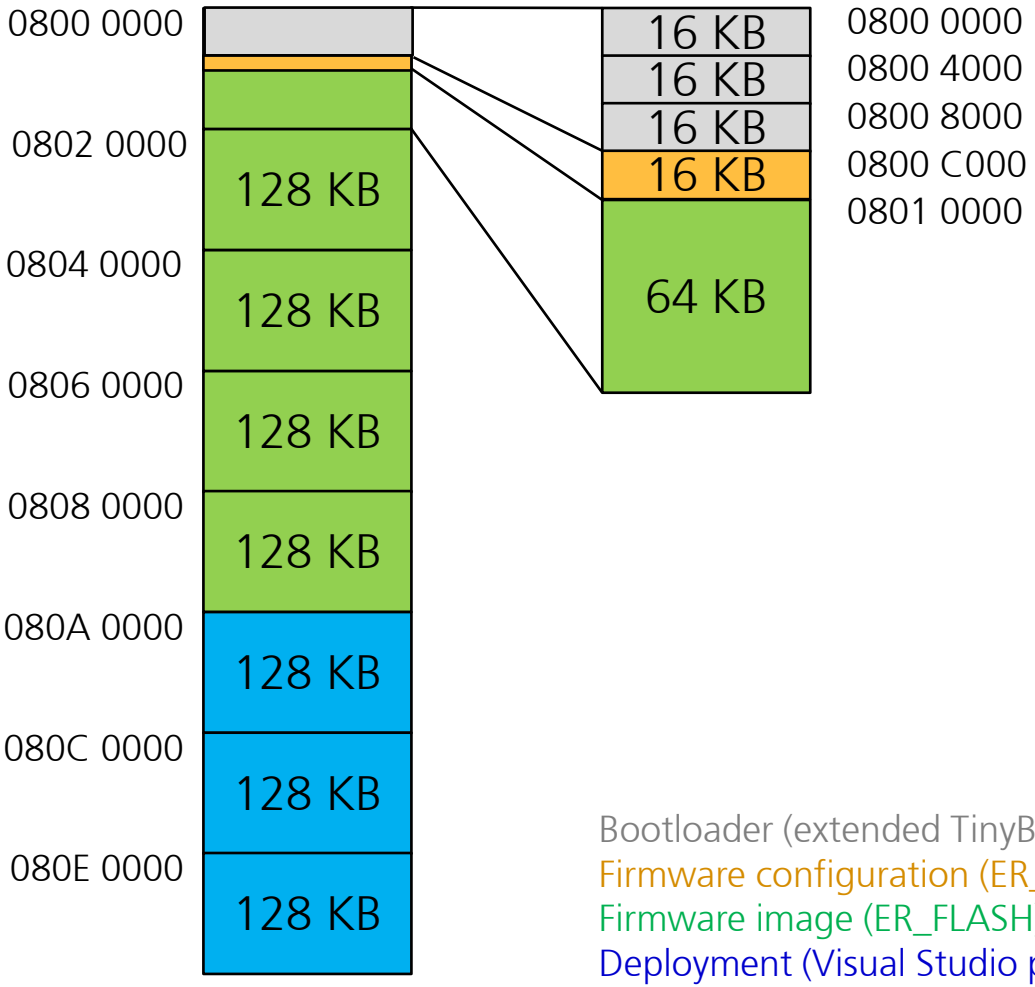
Firmware Updates

- Robust
  - Buffered on Flash storage
    - e.g., serial Flash chip on Mountaineer mainboards
  - Incremental buffering supported
- Flexible
  - .nmf file contains any binary code
    - HAL/PAL/CLR/system assemblies («firmware»)
    - But could also be application assemblies!
    - Or anything else, e.g., security keys

Other Update Types

- Assembly Updates and Key Updates
  - Uses the CLR only
    - Boot loader is not involved
    - *.dat file type instead of *.nmf
  - These update types are not discussed here
    - We use Firmware Updates for *any* kind of update (even for pure application updates)!

STM32F407 Memory Map



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

STM32F407 Memory Map

Address Range	Type	Size	Content	Comments
0800 0000 – 0800 BFFF	Flash	48 KB	Bootloader	Extended TinyBooter (native code)
0800 C000 – 0800 FFFF	Flash	16 KB	Firmware config	MAC address, IP address, etc.
0801 0000 – 0809 FFFF	Flash	576 KB	Firmware image	NETMF (native & managed code)
080A 0000 – 080F FFFF	Flash	384 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

* Only for *Mountaineer Ethernet Mainboard*, USB mainboards have 127 KB heap instead

MFUpdate Elements

- Build system
- NMF files
- Managed code API
  - *Microsoft.SPOT.Update* namespace
  - Allows you to use HTTP, TFTP, USB sticks, ...
- Native code API
- Extended TinyBooter
  - Normal TinyBooter extended with update feature of Microsoft's MicroBooter

Overview

1. MFUpdate
2. **Build System**
3. NMF Files
4. Managed Code API
5. Native Code API
6. Extended TinyBooter
7. Summary

Managed vs. Native

- Managed build system
  - Packs managed code (.NET assemblies) into .nmf file
    - A .NET assembly may contain resources, e.g. strings, images or arbitrary blobs
- Native build system
  - Packs native code into .nmf file
  - Optionally append .NET assemblies
    - Makes sense for rarely changing code, to speed up deployment during debugging
  - Optionally append data sections

Microsoft BuildHelper

- BuildHelper utility
  - Creates .nmf files out of the application's .pe files, by compressing and appending them
  - Creates .nmf files out of the porting kit's .bin files, by compressing them
  - Make sure that you have *BuildHelper.exe*
    - %ProgramFiles%\Microsoft .NET Micro Framework\v4.3\Tools\BuildHelper.exe

Post-Build Script

- Add post-build script in Visual Studio
  - Add the script below in the *Post-build event command line* of the *Build Events* tab of the application project's properties

```
echo post-build started --------------------------
echo $(TargetDir)le
cd $(TargetDir)le
dir *.pe /B > pefiles.txt
"%ProgramFiles%\Microsoft .NET Micro Framework\v4.3\Tools\"MetaDataProcessor.exe -create_database pefiles.txt
$(ProjectName).dat
"%ProgramFiles%\Microsoft .NET Micro Framework\v4.3\Tools\"MetaDataProcessor.exe -dump_dat
$(ProjectName).dat
echo 0x080A0000 D Deploy > symdefs.txt
%ProgramFiles%\Microsoft .NET Micro Framework\v4.3\Tools\buildhelper -symdef symdefs.txt Deploy -
compress $(ProjectName).dat $(ProjectName).nmf
```

Overview

1. MFUpdate
2. Build System
3. **NMF Files**
4. Managed Code API
5. Native Code API
6. Extended TinyBooter
7. Summary

NMF Files

- Contains an update image
  - May be complete or partial update
  - May contain any mix of native code, managed code and data blocks
  - May contain several, non-contiguous regions
- Produced by tool chain
  - Visual Studio for managed code, then *BuildHelper.exe*
  - e.g. Keil tools for native code, then *BuildHelper.exe*
- Consumed by extended TinyBooter

File Format

- Concatenated list of regions
  - Start address of region
  - Compressed bytes of region content
    - Default algorithm is LZ77

Overview

1. MFUpdate
2. Build System
3. NMF Files
4. **Managed Code API**
5. Native Code API
6. Extended TinyBooter
7. Summary

Referenced Assemblies

- Add references to these DLLs
  - *Microsoft.SPOT.Update*
  - *MFUpdate*

abstract class MFUpdate

- Most important methods
  - *AddPacket*
    - Persist part of an update image
  - *ValidateUpdate*
    - Check consistency of update image
  - *InstallUpdate*
    - Reboot
    - Unpack and commit update image
    - Delete update image

class MFFirmwareUpdate

- Implementation of *MFUpdate* class
  - Adds *CurrentFirmwareVersion* to enable reflection on the currently installed firmware
    - Properties, e.g., *Version* number
    - Assemblies

Overview

1. MFUpdate
2. Build System
3. NMF Files
4. Managed Code API
5. **Native Code API**
6. Extended TinyBooter
7. Summary

Key Interfaces

- IUpdatePackage
  - Structure containing hooks to the extension interfaces (update provider, validation provider, storage provider, backup provider)
- IUpdateProvider
  - Most important methods
    - InitializeUpdate
    - InstallUpdate

Other Provider Interfaces

- IUpdateValidationProvider
  - Validate individual packets and complete image, e.g. using a CRC check
- IUpdateStorageProvider
  - Store packets of an update image
- IUpdateBackupProvider
  - Optional backup/restore mechanism

Standard Providers

- Update provider
  - For extended TinyBooter
- Validation provider
  - CRC checking
  - Optional SSL support
- Uses a block storage provider
  - Reserved *BLOCKTYPE_UPDATE* region

Block Storage Providers

- External block storage provider allows to use SPI Flash for *MFUpdate*, for storing an NMF image
  - Mountaineer boards have external Flash
    - 8 MB of serial SPI Flash
    - Uppermost 1 MB block reserved for *MFUpdate*
- Internal block storage provider allows to write decompressed image to internal Flash

Overview

1. MFUpdate
2. Build System
3. NMF Files
4. Managed Code API
5. Native Code API
6. **Extended TinyBooter**
7. Summary

Extended TinyBooter

- Extended with MicroBooter update feature
- Optional boot loader mode
  - Hold down USER button during RESET
  - Communicates with MFDeploy for download of update in SREC format (*.hex files)
- **Installs an update, if available**
- Starts CLR
  - which in turn starts application

Overview

1. MFUpdate
2. Build System
3. NMF Files
4. Managed Code API
5. Native Code API
6. Extended TinyBooter
7. Summary

MFUpdate Summary

Features / Quality Attributes	MFUpdate Solutions
Remote updates	Firmware Update provider, extended TinyBooter
Robustness	Intermediate storage (Firmware Updates), image verification, incremental installation
Firmware, not just application	Any memory region, including key storage
Bandwidth and storage space	Compressed update files
Security	SSL support (not available on Mountaineer boards)
Flexibility and memory footprint	Framework instead of fixed mechanism
Convenience	Visual Studio for most tasks
No vendor lock-in	Standard APIs, open source
Engineering services	Mountaineer Group companies