Hands-On Workshop: Build Your First Zephyr Application on i.MX RT

Maureen Helm, NXP
Thea Aldrich, Linux Foundation

June 2019 | Session #AMF-SOL-T3639
Agenda

• Introduce the Zephyr Project
• Review High-level Software Features and Hardware Support
• Set up a Development Environment
• Hands-on: Build, Flash, and Debug an Application
Zephyr Project Introduction

What is the Zephyr Project? Why should I use it?
Zephyr Project

- Open source real time operating system
- Vibrant Community participation
- Built with safety and security in mind
- Cross-architecture with growing developer tool support
- Vendor Neutral governance
- Permissively licensed - Apache 2.0
- Complete, fully integrated, highly configurable, modular for flexibility, better than roll-your-own
- Product development ready with LTS
- Certification ready with Auditable

Zephyr OS

3rd Party Libraries
Application Services
OS Services
Kernel
HAL

Open Source, RTOS, Connected, Embedded Fits where Linux is too big

THE LINUX FOUNDATION PROJECTS
Architecture

- Highly Configurable, Highly Modular
- Cooperative and Pre-emptive Threading
- Memory and Resources are typically statically allocated
- Integrated device driver interface
- Memory Protection: Stack overflow protection, Kernel object and device driver permission tracking, Thread isolation
- Bluetooth® Low Energy (BLE 4.2, 5.0) with both controller and host, BLE Mesh
- Native, fully featured and optimized networking stack

Fully featured OS allows developers to focus on the application
NXP Board Support

- **i.MX RT Series (Cortex M7)**
  - RT1015 EVK
  - RT1020 EVK
  - RT1050 EVK
  - RT1060 EVK
  - RT1064 EVK

- **i.MX 6/7 Series (Cortex M4 subsystem)**
  - UDOO Neo Full
  - Colibri iMX7
  - WaRP7

- **Kinetis Series (Cortex M4, M0+)**
  - FRDM-K64F
  - FRDM-KW41Z
  - FRDM-KL25Z
  - TWR-KE18F
  - Hexiwear

- **LPC Series (Cortex M4, M0+, M33)**
  - LPCXpresso54114
  - LPCXpresso55S69 (coming soon)

[https://docs.zephyrproject.org/latest/boards/index.html](https://docs.zephyrproject.org/latest/boards/index.html)
NXP Board Support

• **Upstream**
  - Contributed and maintained by NXP and the community
  - NXP active in upstream working groups

• **Built upon MCUXpresso SDK**
  - SDK bare metal drivers and CMSIS device headers contributed upstream
  - Shim drivers adapt SDK interfaces to Zephyr interfaces
  - Maximizes code reuse

• **Tested on hardware in NXP board farm**
Long Term Support (LTS) Release

- Product-focused release will receive bug fixes and maintain stable APIs for two years
- Extended stabilization period enabled more testing and bug fixing prior to release
- Baseline for auditable version of Zephyr

- Released in Apr 2019 (Zephyr v1.14.0)
- Supports over 160 board configurations across 8 architectures
- Contributions from 250 developers

- Hands-on exercises in this workshop use the LTS release
Zephyr Project Governance

Goal: Separate business decisions from meritocracy, technical decisions

**Governing Board**
- Decides project goals
- Sets business, marketing and legal decisions
- Prioritizes investments and oversees budget
- Oversees marketing such as PR/AR, branding, others
- Identifies member requirements

**Technical Steering Committee**
- Serves as the highest technical decision body consisting of project maintainers and voting members
- Sets technical direction for the project
- Coordinates X-community collaboration
  - Sets up new projects
  - Coordinates releases
  - Enforces development processes
  - Moderates working groups
- Oversees relationships with other relevant projects

**Community**
- Code base open to all contributors, need not be a member to contribute.
- Path to committer and maintainer status through peer assessed merit of contributions and code reviews
- Ecosystem enablement
Zephyr Project Membership

February 2016

Integer
NXP
Synopsys

May 2019

Intel
Nordic Semiconductor
NXP
Oticon
antmicro
Cloud Foundry
Linaro
Linino.org
SiFive
Synopsys
Texas Instruments

and others….
Zephyr Development Environment

What tools do I need? How do I install them on my PC?
Development Environment Introduction

- Zephyr applications can be developed on Windows, Linux, or macOS host operating systems
- CMake and Python enable portability across host operating systems
- Detailed instructions are documented in the [Getting Started Guide](#)

- Major components:
  - Python 3: Script interpreter and packages
  - CMake/Ninja/Make: Build system
  - Device Tree Compiler: Compiles device tree hardware descriptions
  - Toolchain: gcc for Arm, RISC-V, x86, etc.
  - Debug/Flash Tools: J-Link, pyOCD, OpenOCD, etc.
  - West: Custom tool for repository management, build/flash/debug assistance, and image signing
  - Zephyr Git repositories: The source code!

- [Zephyr SDK](#) provides toolchains and some debug/flash tools for Linux only
Windows: Command Prompt, WSL, or VM?

- **Windows Command Prompt**: Requires manual toolchain installation, but can use debug/flash tools like J-Link and pyOCD. **Recommended for new developers**

- **Windows Subsystem for Linux (WSL)**: Can use Zephyr SDK toolchains and sanitycheck, but does not support debug/flash tools like J-Link and pyOCD. **Not recommended**

- **Linux Virtual Machine (VM)**: Can use Zephyr SDK toolchains, sanitycheck, and debug/flash tools like J-Link and pyOCD; but requires installing a virtual machine. **Recommended for experienced developers**
Windows: Install Chocolatey and Packages

• Open an administrator command prompt

• Install Chocolatey package manager
  - Similar to apt on Ubuntu

• Disable global confirmation
  > choco feature enable -n allowGlobalConfirmation

• Use Chocolatey to install CMake
  > choco install cmake --installargs 'ADD_CMAKE_TO_PATH=System'

• Use Chocolatey to install dependencies
  > choco install git python ninja dtc-msys2 gperf
Windows: Bootstrap West and Clone Zephyr Repos

- Open a normal command prompt

- Bootstrap west
  > pip3 install west

- Clone the Zephyr git repositories
  > cd %userprofile%
  > west init --mr v1.14.0 zephyrproject
  > cd zephyrproject
  > west update

- Install python dependencies
  > pip3 install -r zephyr/scripts/requirements.txt
Windows: Install Toolchain and Flash/Debug Tools

• Install **GNU Arm Embedded** toolchain
  - Use Windows ZIP instead of Windows Installer. This will allow you to define an installation path without spaces
  - Skip this step if you already have MCUXpresso IDE installed

• Install **J-Link** flash/debug tools with Windows installer
  - Required for i.MX RT and LPC boards, optional for Kinetis boards
  - Skip this step if you already have MCUXpresso IDE installed

• Create `zephyrrc.cmd` file in `%userprofile%` directory
  ```
  set ZEPHYR_TOOLCHAIN_VARIANT=gnuarmemb
  set GNUARMEMB_TOOLCHAIN_PATH=C:\nxp\MCUXpressoIDE_10.3.1_2233\ide\tools
  set PATH=%PATH%;C:\Program Files (x86)\SEGGER\JLink_V642b
  ```
Install Eclipse IDE Plugins

• **Install** Eclipse IDE for C/C++ Developers
  - Skip this step if you already have MCUXpresso IDE installed

• **Install** GNU MCU Eclipse plug-ins
  - From the **Help** menu, select **Eclipse Marketplace**
  - Search for “gnu mcu eclipse” and click **Install**
Hands-On Exercises
The Fun Part!
Hands-On Overview

• **Exercise #1: Blinky**
  - Build and flash a simple application
  - Examine application source code and build artifacts

• **Exercise #2: Eclipse IDE Debugging**
  - Generate and import an Eclipse IDE project
  - Create and launch a debug configuration

• **Exercise #3: Display and Graphics with LittlevGL Integration**
  - Build and flash an LCD application

• **Exercise #4: Configuration and Memory Footprint**
  - Examine flash/ram footprint with \texttt{rom\_report} and \texttt{ram\_report}
  - Change the configuration and rebuild
Exercise #1: Blinky

Build and flash a simple application

Examine application source code and build artifacts
Build and Flash Blinky

- Open a normal command prompt

- Set up the build environment
  > cd %userprofile%\zephyrproject\zephyr
  > zephyr-env.cmd

- Build the blinky sample application
  > west build -b mimxrt1050_evk -d build\blinky samples\basic\blinky

- Flash it to the board
  > west flash -d build\blinky

- See the LED blinking
Blinky Application Source Code

- `samples\basic\blinky\src\main.c`

- Same application source code works on many different boards, not just i.MX RT1050-EVKB

- Standard GPIO interface APIs
  - `gpio_pin_configure()` and `gpio_pin_write()`

- Standard LED macros generated from device tree
  - `LED0_GPIO_CONTROLLER` and `LED0_GPIO_PIN`
i.MX RT1050-EVK Board Device Tree

- boards\arm\mimxrt1050_evk\mimxrt1050_evk.dts

- Defines board hardware components such as LEDs, sensors, and external memories
  - LED node defines GPIO instance and pin
  - Memory nodes define SDRAM and Hyperflash sizes
  - Chosen node selects UART instance for console

- Includes SoC device tree
i.MX RT1050 SoC Device Tree

- dts\arm\nxp\nxp_rt.dtsi

- Defines SoC peripheral addresses, interrupts, and device driver labels

- Clocks properties used by peripheral drivers to configure UART, I2C baud rates
Exercise #2: Eclipse IDE Debugging

Generate and import an Eclipse IDE project

Create and launch a debug configuration
Generate an Eclipse IDE Project

• Open a normal command prompt

• Set up the build environment
  > cd %userprofile%\zephyrproject\zephyr
  > zephyr-env.cmd

• Move to a directory outside the Zephyr tree. This is required only when generating Eclipse projects
  > cd %userprofile%

• Generate and build an Eclipse project for the hello_world application
  > west build -b mimxrt1050_evk %ZEPHYR_BASE%\samples\hello_world - -G"Eclipse CDT4 - Ninja"
Import the Eclipse IDE Project

- Open **MCUXpresso IDE**
- From the **File** menu, select **Import…**
- Select **Existing Projects into Workspace**
- Select **Next**
- Select **Browse** and navigate to your build directory
- Select **Finish**

**Warning:** Do not check **Copy projects into Workspace**
```c
/*
 * Copyright (c) 2012-2014 Wind River Systems, Inc.
 * SPDX-License-Identifier: Apache-2.0
 */

#include <ziel.h>
#include <lib/cprintf.h>

void main(void)
{
    printk("Hello World! %s\n", CONFIG_BOARD);
}
```
Create a New Debug Configuration

- From the Run menu, select Debug Configurations…

- Select GDB SEGGER J-Link-Debugging, and click the New button

- **Warning:** Do not select GDB SEGGER Interface Debugging
J-Link Debug Configuration: Main

- Select the **Main** tab and configure the following settings:
  - **Project:** `hello_world@build`
  - **C/C++ Application:** `zephyr/zephyr.elf`
J-Link Debug Configuration: Debugger

- Select the Debugger tab and configure the following settings:
  - Device name: MCIMXRT1052
  - GDB Client Executable name: C:\nxp\MCUXpressoIDE_10.3.1_2233\ide\tools\bin\arm-none-eabi-gdb.exe
  - Uncheck Allocate console for semihosting and SWO
J-Link Debug Configuration: Startup

- Select the **Startup** tab
- Uncheck **Enable semihosting**
- Uncheck **Enable SWO**
J-Link Debug Configuration: SVD Path

- Select the SVD Path tab and configure the following settings:

- SVD file path:
  C:\Users\NXPTraining\zephyrproject\zephyr\ext\hal\nxp\mcux\devices\MIMXRT1052\MIMXRT1052.xml

- Select Debug to start the debugger!
Open a Serial Terminal

- From the Window menu, select Show View->Terminal
- Select the Terminal tab in the bottom third of the window
- Select Open a Terminal
- Enter serial port settings as shown (COM number may be different)
Run the Application

- Select **Resume** to run the application

- See in the terminal:
  Hello World! mimxrt1050_evk
Exercise #3: Display and Graphics with LittlevGL Integration

Build and flash an LCD application
Build and Flash LittlevGL

• Open a normal command prompt

• Set up the build environment
  > cd %userprofile%\zephyrproject\zephyr
  > zephyr-env.cmd

• Build the LittlevGL sample application
  > west build -b mimxrt1050_evk -d build\lvgl samples\gui\lvgl

• Flash it to the board
  > west flash -d build\lvgl

• See “Hello world!” on the LCD
Exercise #4: Configuration and Memory Footprint

Examine flash/ram footprint with `rom_report` and `ram_report`

Change the configuration and rebuild
Examine Memory Footprint

• Open a normal command prompt

• Set up the build environment
  > cd %userprofile%\zephyrproject\zephyr
  > zephyr-env.cmd

• Move to the LittlevGL sample application build directory
  > cd build\lvgl

• Run reports to see flash and ram memory footprints
  > ninja rom_report
  > ninja ram_report
Change Configuration and Rebuild

• Open samples\gui\lvgl\prj.conf in a text editor and disable logging
  CONFIG_LOG=n

• Open a normal command prompt

• Set up the build environment
  > cd %userprofile%\zephyrproject\zephyr
  > zephyr-env.cmd

• Rebuild the LittlevGL sample application with the new configuration
  > west build -d build\lvgl -c

• Move to the LittlevGL sample application build directory and rerun reports
  > cd build\lvgl
  > ninja rom_report
  > ninja ram_report

<table>
<thead>
<tr>
<th>CONFIG_LOG</th>
<th>ROM (B)</th>
<th>RAM (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>80044</td>
<td>590628</td>
</tr>
<tr>
<td>N</td>
<td>72376</td>
<td>588425</td>
</tr>
<tr>
<td>Delta</td>
<td>7668</td>
<td>2203</td>
</tr>
</tbody>
</table>
Backup
References

- https://docs.zephyrproject.org/latest/boards/index.html
- https://docs.zephyrproject.org/1.14.0/getting_started/index.html#build-and-run-an-application
- https://docs.zephyrproject.org/1.14.0/application/index.html#eclipse-debugging