Zephyr Community Training - Day 3

This session covers Inter-Process Communication (IPC) in Zephyr RTOS. Like other real-time operating systems, Zephyr provides kernel objects (primitives) for robust, thread-safe data exchange and synchronization.

The core goal of these labs is two-fold:

1. Classical IPC: Work with the main Zephyr tools for passing data, like Message Queues, FIFOs, and Pipes.
2. Modern Modularity (Zephyr Bus): Learn how the Zephyr Bus (zbus) uses a channel-based, publish/subscribe architecture. This high-level mechanism enhances modularity, flexibility, and simplifies design by decoupling software components.

By day's end, participants will better use Zephyr's IPC tools and know how to apply zbus to build scalable, loosely coupled embedded applications.

Instructor

Rodrigo Peixoto - rodrigopex@citrinio.com or rodrigopex@gmail.com

3-Day Zephyr Community Training Partners

The 3-Day Free Zephyr Community Edition Training is brought to you in partnership with AC6 and Beningo. Each day is brought to you from one of the partners:

Training realised by:

Day 1 - Roy Jamil

Day 2 - Jacob Beningo

Day 3 - Rodrigo Peixoto

General instruction ,

All the exercises are based on the concept of a system in which data is produced by a Producer thread, analyzed by a Filter thread, and consumed by a Consumer thread. The data flow between threads differs in each exercise, but the underlying idea remains the same. The only variation is that, in the zbus exercise, there are two Consumer threads instead of one.

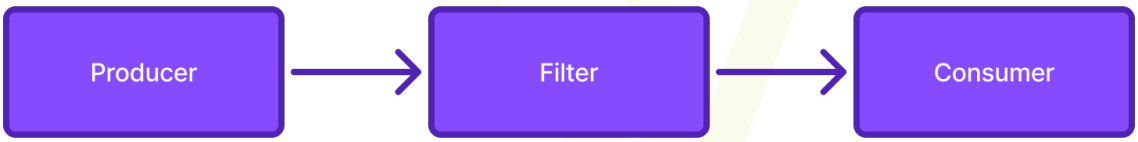

Figure 1: Basic system data flow.

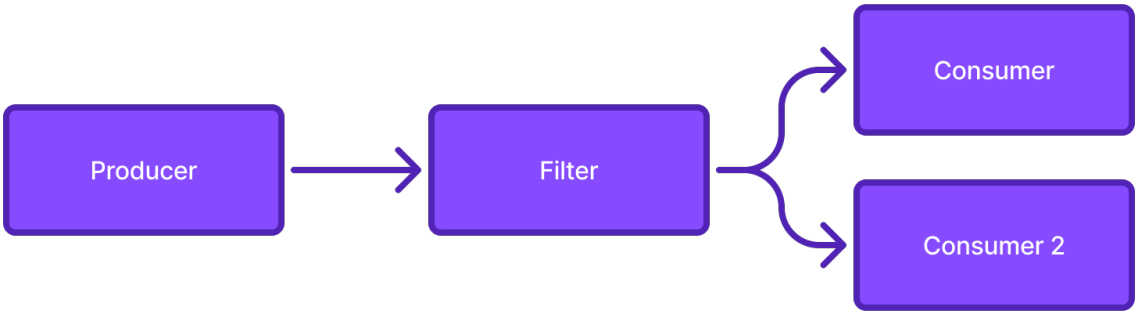

Figure 2: System data flow variation with 2 consumers.

To ensure you have the latest exercises, update the workspace by right-clicking in the workspace in the Zephyr Workbench tab and clicking on the Update button.

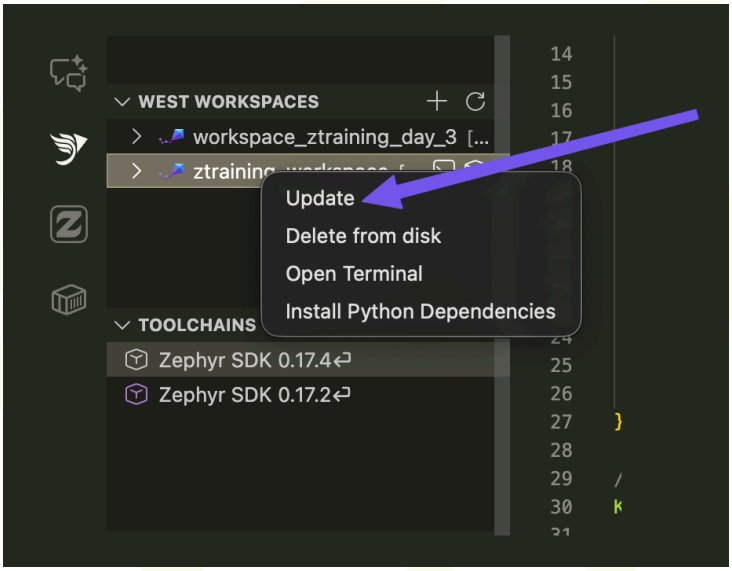

Figure 3: Example of replacements to be solved.

Using the same approach you use on other days, add the **`lab1_msgq`**, **`lab2_fifo`**, and **`lab3_zbus`** applications from the **`Day3/labs`** folder to your workspace. Next, carefully review the code and identify all locations marked with **`$$$`**. For each **`$$$`**, determine and replace it with the correct value or statement needed for the code to compile correctly. Make sure you understand what each replacement does by reading the code before making any changes. This will help.

Lab1: Data Passing - Message Queue

In this lab, you will build a simple application that introduces the Zephyr message queue. You will create a data passing flow between threads using a message queue.

Learning Objectives

By completing this lab, you will:

- Setup a message queue in Zephyr
- Generate random numbers in test environment

- Implement a data passing flow in C

```c
/*
 * Copyright (c) 2025 Citrinio
 *
 * SPDX-License-Identifier: Apache-2.0
 */
#include <zephyr/kernel.h>

/* Declare an external message queue to use that in this file */
extern struct k_msgq msgq_$$$$;

void $$$$$$$$_thread_entry(void *arg1, void *arg2, void *arg3)
{
    ARG_UNUSED(arg1);
    ARG_UNUSED(arg2);
    ARG_UNUSED(arg3);
```

Figure 4: Example of replacements to be solved.

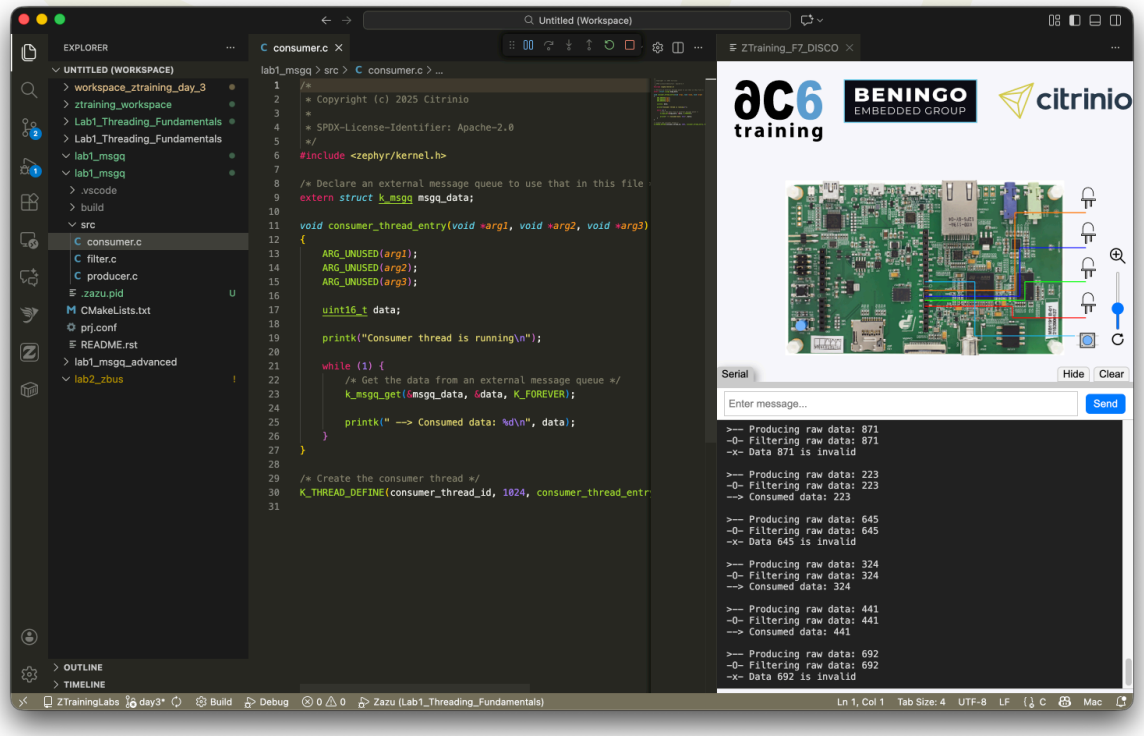

Figure 5: Expected results.

Alternative execution (terminal)

You can also run the project in a terminal with the Zephyr environment set.

Build

Inside the `Day3` folder, run the command to build the project:

Console output

```
$ west build -p -b mps2/an385 lab1_msgq -- -DCONFIG_QEMU_ICOUNT=n

...supressed...

[132/132] Linking C executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:       13812 B         4 MB      0.33%
             RAM:        8608 B         4 MB      0.21%
        IDT_LIST:          0 GB        32 KB      0.00%
Generating files from /../ZTrainingLabs/Day3/solutions/build/zephyr/
zephyr.elf for board: mps2
```

Note

- The `--` allows you to add configuration at compile time.
- The **`-DCONFIG_QEMU_ICOUNT=n`** enables us to see QEMU execution in our time, not at full speed.

Run

To run the project, execute the following command:

Console output

```
$ west build -t run
-- west build: running target run
[0/1] To exit from QEMU enter: 'CTRL+a, x'[QEMU] CPU: cortex-m3
qemu-system-arm: warning: nic lan9118.0 has no peer
*** Booting Zephyr OS build v4.2.1 ***
Consumer thread is running
Filter thread is running
Producer thread is running

 >-- Producing raw data: 903
 -O- Filtering raw data: 903
 -x- Data 903 is invalid

 >-- Producing raw data: 12
 -O- Filtering raw data: 12
 --> Consumed data: 12

 >-- Producing raw data: 324
 -O- Filtering raw data: 324
 --> Consumed data: 324

 ...<continues>
```

To finish the execution, type **`ctrl+c`**.

Lab2: Data Passing - FIFO

In this lab, you will build a simple application that introduces the Zephyr FIFO. You will create a data passing flow between threads using a FIFO.

Learning Objectives

By completing this lab, you will:

- Setup a FIFO in Zephyr
- Generate random numbers in test environment
- Dynamic allocation
- Implement a data passing flow in C

prj.conf

- Set heap CONFIG_HEAP_MEM_POOL_SIZE to 1024

Exercise

Replace the **$$$** to what makes sense based on the comments.

Results should be like:

Console output

```
*** Booting Zephyr OS build v4.2.1 ***
Consumer thread is running
Filter thread is running
Producer thread is running

>-- Producing raw data: 146
-O- Filtering raw data: 146
--> Consumed data: 146

>-- Producing raw data: 203
-O- Filtering raw data: 203
--> Consumed data: 203

>-- Producing raw data: 864
-O- Filtering raw data: 864
-x- Data 864 is invalid

>-- Producing raw data: 189
-O- Filtering raw data: 189
--> Consumed data: 189

>-- Producing raw data: 473
-O- Filtering raw data: 473
--> Consumed data: 473

...<continues>
```

Lab3: Data Passing - ZBus

In this lab, you will build a simple application that introduces the Zephyr zbus. You will create a data passing flow between threads using zbus.

Learning Objectives

By completing this lab, you will:

- Use zbus
- Generate random numbers in test environment
- Implement a data passing flow in C

prj.conf

- Set heap CONFIG_HEAP_MEM_POOL_SIZE to 1024
- Enable CONFIG_ZBUS
- Enable CONFIG_ZBUS_MSG_SUBSCRIBER

Exercise

Replace the **$$$** to what makes sense based on the comments.

Results should be like:

,

Console output

```
*** Booting Zephyr OS build v4.2.1 ***
Timer producer is starting
MSG Subscriber consumer thread is running
Subscriber consumer thread is running

>-- Producing raw data: 343
-O- Filtering raw data: 343
--> MSub consumed data: 343
--> Sub consumed data: 343

>-- Producing raw data: 789
-O- Filtering raw data: 789
-x- Data 789 is invalid

>-- Producing raw data: 552
-O- Filtering raw data: 552
-x- Data 552 is invalid

>-- Producing raw data: 668
-O- Filtering raw data: 668
-x- Data 668 is invalid

>-- Producing raw data: 371
-O- Filtering raw data: 371
--> MSub consumed data: 371
--> Sub consumed data: 371

...<continues>
```

Bonus

All the exercises have an advanced version with different implementations exploring different approaches. See the **`Day3/solutions`** folder.