



How to exchange data buffers with the coprocessor



Contents

1 Article purpose	3
2 Introduction	4
3 Example of context description	5
4 Example of static architecture for exchanging data buffers	6
5 Cortex-M firmware	8
6 Linux user land application	9
7 Linux drivers	10
7.1 rpmsg_tty driver	10
7.2 rpmsg_sdb driver	10
8 Dynamic view	12
9 Results	15
10 Source code	16
11 Key messages	17
12 Usage	18
13 References	19



1 Article purpose

This article gives an example of high-rate transfers of data chunks from the Arm® Cortex®-M core to the Arm® Cortex®-A core.



2 Introduction

Relying on a logic analyzer sample, this article describes the mechanism and the software implemented to perform high-rate transfers. In this example, the Cortex-M core is used to perform continuously:

- real-time operations
- simple data algorithm (masking bit)
- transfer of the resulting data to the cortex-A

Depending on the frequency sampling, the data is transferred using:

- either [direct buffer exchange mode](#)
 - TTY RMsg channel for control and data transfer
 - sampling frequency is less than or equal to 5 MHz
- or [indirect buffer exchange mode](#)
 - transfer using DDR buffers requiring:
 - contiguous memory allocation in DDR memory
 - Cortex-M awareness of the physical address and size of the memory buffers
 - mmaping of buffers to enable Linux® user land application access to them
 - [rpmsg_sdb](#) (shared data buffer) Linux driver, developed to take care of DDR constraints. For details on this buffer exchange mechanisms, refer to the "[rpmsg_sdb driver](#)" chapter
 - TTY RMsg channel for control (especially to exchange references to the buffers between the processors)
 - sampling frequency is more than 5 MHz



3 Example of context description

Let us implement a logic analyzer running on the STM32MP1 discovery kit.

From the user interface, press the START button to start the logic analyzer sampling. The logic analyzer samples GPIO PORT E bits 8 to 14, which are present on the Arduino connector. They correspond to 7 bits. The 8th bit will be reset by M4 algorithm.

The number of received data is displayed on the screen as bytes and megabytes.



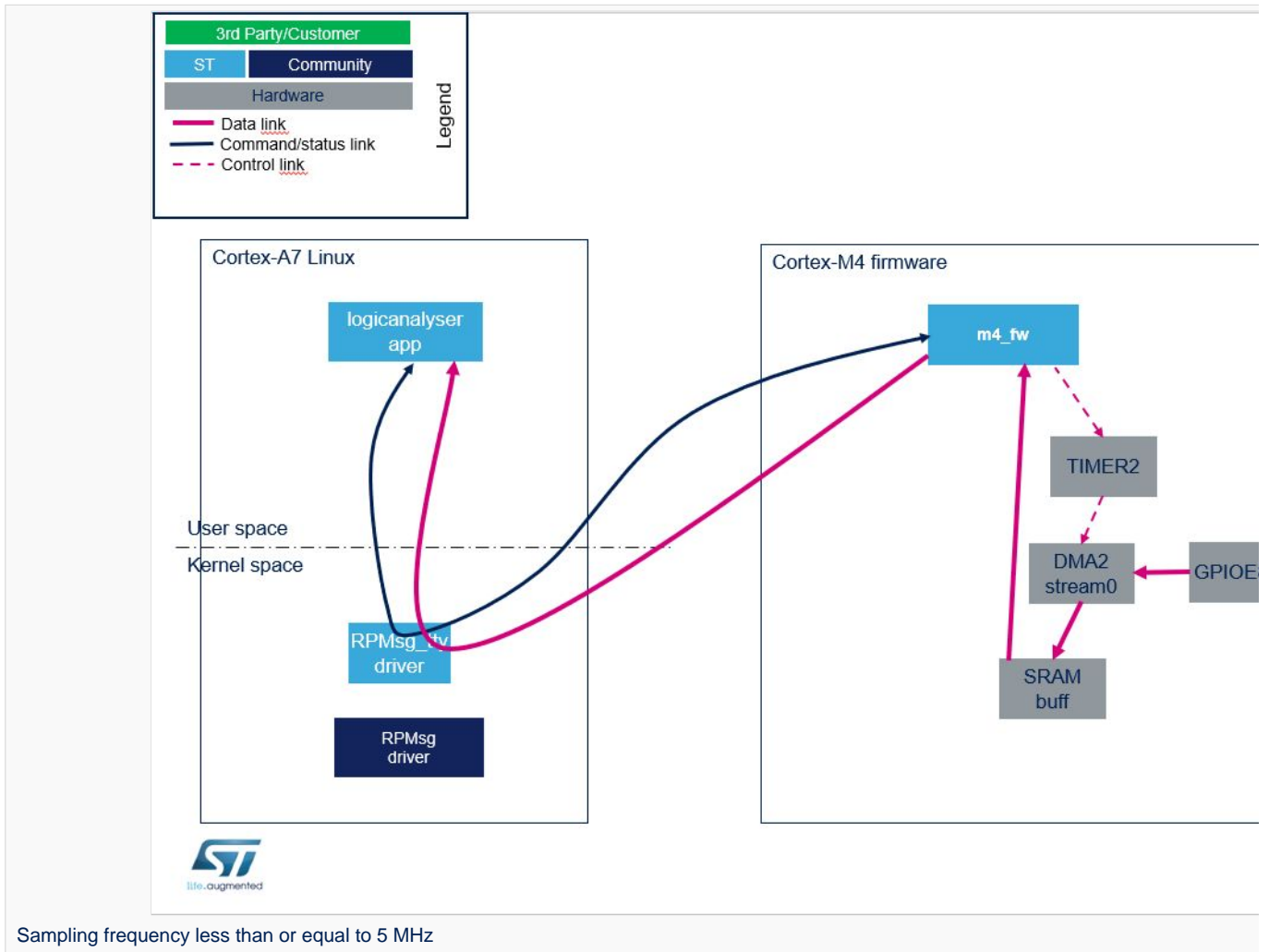
4 Example of static architecture for exchanging data buffers

The example of data buffer exchange includes:

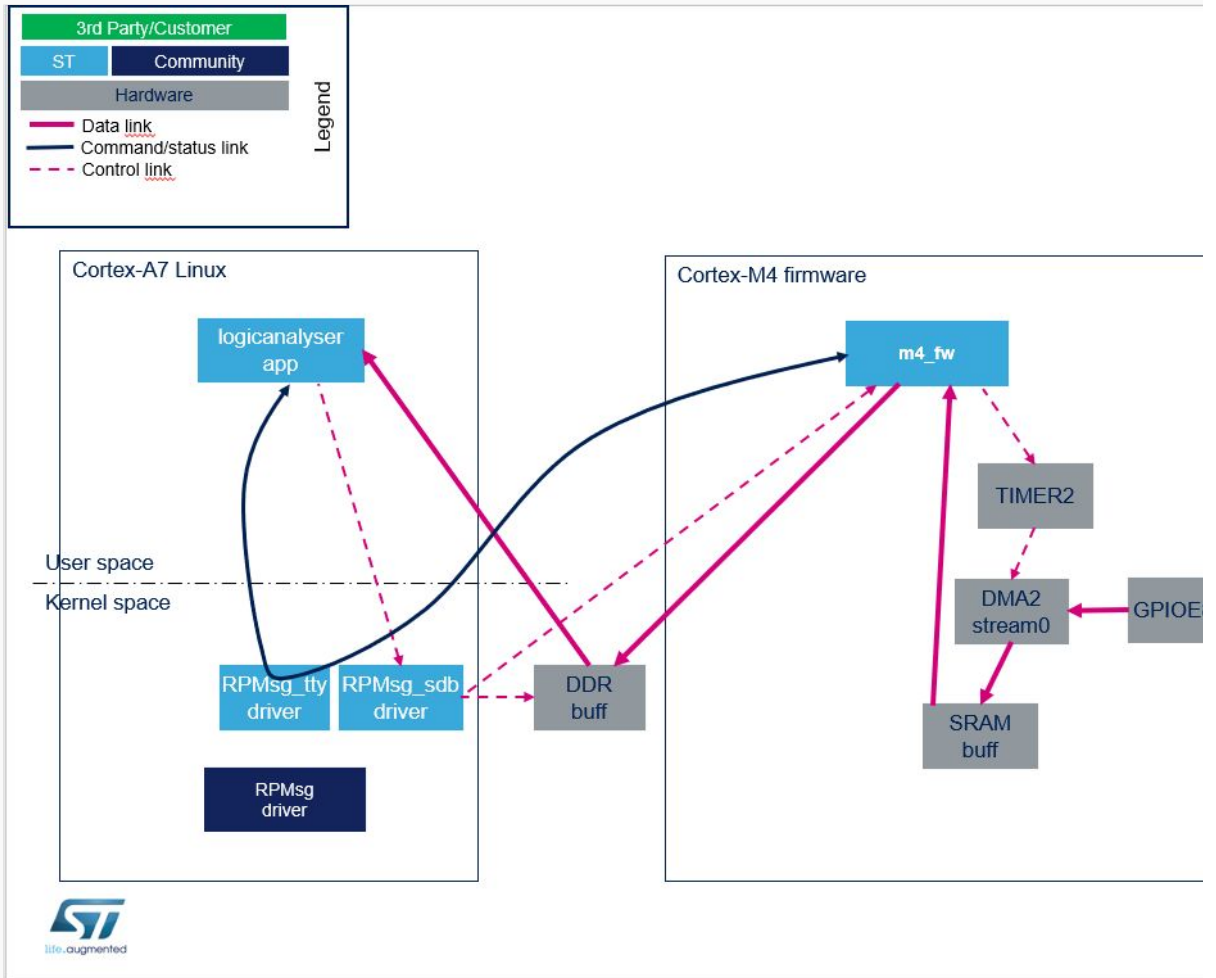
- a Cortex-M firmware
- a Linux user land application
- a Linux rpsmsg_tty driver
- a Linux rpsmsg_sdb (shared data buffer) driver

Note that there is a direct correspondence between the sampling frequency and the data flow rate: thus, a sampling frequency of 8 MHz means a data flow rate of 8 MB/s (megabytes per second).

Data flow when sampling frequency is less than or equal to 5 MHz: the direct buffer exchange mode



Data flow when sampling frequency is more than 5 MHz: the indirect buffer exchange mode (a.k.a. large data buffers exchange)



Sampling frequency more than 5 MHz



5 Cortex-M firmware

The Cortex-M firmware is responsible for:

- receiving a command giving of the number of DDR buffers through the TTY RMsg channel, from the Linux application.
- receiving messages containing the physical address and size of DDR buffer(s), from the Linux rmsg_sdb driver; These DDR buffers are always allocated, during the initialisation step, even if they are only used when the frequency sampling is more than 5 MHz (see below).
- receiving a command Start/Stop sampling (including sampling frequency) through the TTY RMsg channel, from the Linux application.
- On start request:
 - sampling the data at the requested sampling frequency thanks to DMA2 stream0 from GPIOE to SRAM buffers
 - masking and transferring data buffers from SRAM to Linux application
 - thanks to TTY over RMSG buffer by packet of 256 bytes if the frequency is less than or equal to 5 MHz
 - thanks to copy to DDR buffer by packet of 1024 bytes, if the frequency sampling is more than 5 MHz, and informing the Cortex-A user interface (through the SDB RMsg channel) when a DDR buffer of 1 Mbyte is filled, and roll to next DDR buffer



6 Linux user land application

The Cortex-A Linux application includes a GTK user interface.

It allows controlling:

- the sampling frequency
- the start / stop of the sampling
- the data to be sampled thanks to "Set data" notch UI widget

The user interface displays:

- statistics: the number of data received by the user interface, as bytes and Mbytes
- the first data of every new received megabyte



7 Linux drivers

7.1 rpmsg_tty driver

The `rpmsg_tty` driver^[1] simulates a serial link for communication between the Cortex-M firmware and the Cortex-A user land application. See also the [Linux RPMsg framework overview](#) article for information about the Linux Remote Processor Messaging (RPMsg) framework.

7.2 rpmsg_sdb driver

The RPMsg shared data buffer (SDB)^[2] driver **example** is in charge of:

- allocating large buffers in contiguous memory (DDR) and memory mapping them (mmap) for use by an application
- implementing the RPMsg service to share buffer information (address, size) with the coprocessor
- sending events to a Linux application (relying on the `eventfd` interface) when buffers are available (on RPMsg message reception)

No kernel configuration is needed. The `rpmsg_sdb` driver is proposed as module and can be installed using the associated Yocto recipe^[2]. No device tree declaration is needed. The `rpmsg_sdb` driver is registered as an RPMsg driver. It is probed when the remote processor creates the "rpmsg-sdb-channel" service.

The `rpmsg_sdb` driver exposes a `"/dev/rpmsg_sdb"` sysfs that offers an interface to allocate and manage the shared buffers.

- open/close: get/release file descriptor

```
int fd;
fd= open('/dev/rpmsg_sdb');
close(fd);
```

- mmap: allocate and map memories

```
void *buff0_id, *buff1_id;

buff0_id = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, 0);
buff1_id = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE, fd, 0);
```

- `RPMMSG_SDB_IOCTL_SET_EFD` ioctl: register event for a buffer

```
typedef struct
{
    int bufferId, eventfd;
} rpmsg_sdb_ioctl_set_efd;

int efd[NB_BUF];
rpmsg_sdb_ioctl_set_efd q_set_efd;

for (i=0;i<NB_BUF;i++){
    /* Create the eventfd, and sent it to kernel driver, for notification of buffer full */
    efd[i] = eventfd(0, 0);
```



```

q_set_efd.bufferId = i;
q_set_efd.eventfd = efd[i];

ioctl(mFdSdbRpmMsg, RMSG_SDB_IOCTL_SET_EFD, &q_set_efd)
}

```

- RMSG_SDB_IOCTL_GET_DATA_SIZE ioctl: get the size of a buffer

```

typedef struct
{
    int bufferId;
    uint32_t size;
} rmsg_sdb_ioctl_get_data_size;

rmsg_sdb_ioctl_get_data_size q_get_data_size;

q_get_data_size.bufferId = i; // i is the buffer index

ioctl(fd, RMSG_SDB_IOCTL_GET_DATA_SIZE, &q_get_data_size);

```

- Manage event

```

while (1) {
    ret = poll(fds, NB_BUF, TIMEOUT * 1000);
    if (ret < 0) {
        perror("poll()");
    } else if (ret) {
        printf("Data is available now.\n");
    } else if (ret == 0) {
        printf("No data within five seconds.\n");
    }
    for (j=0; j<NB_BUF; j++){
        if (fds[j].revents & POLLIN) {
            /* Event received for the buffer j: New data is available for buffer j */
        }
    }
}
}

```

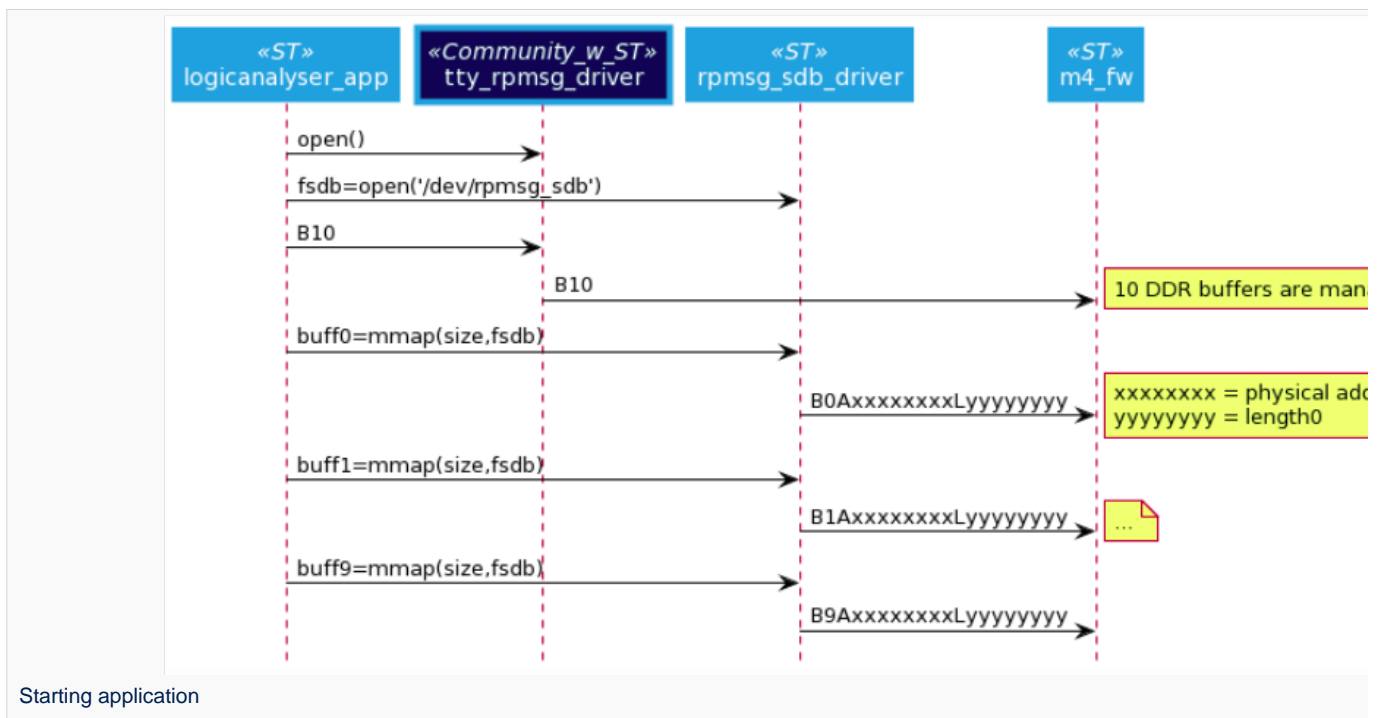


8 Dynamic view

At startup, the Linux application performs the following actions:

- It loads the `rpmsg_sdb.ko` module.
- It loads the Cortex-M firmware, then starts it.
- It opens a `rpmsg_tty` channel for Cortex-M firmware control.
- It opens a `rpmsg_tty` channel for Cortex-M firmware trace debug.
- It opens the `rpmsg_sdb` driver, then uses `rpmsg_sdb` IOCTL interface to allocate and `mmap` 10 buffers of 1 Mbyte in DDR memory.

When the user presses User button2, the Linux application starts.



When the START button is pressed, the application sends the sampling command to the Cortex-M firmware (including the sampling frequency).

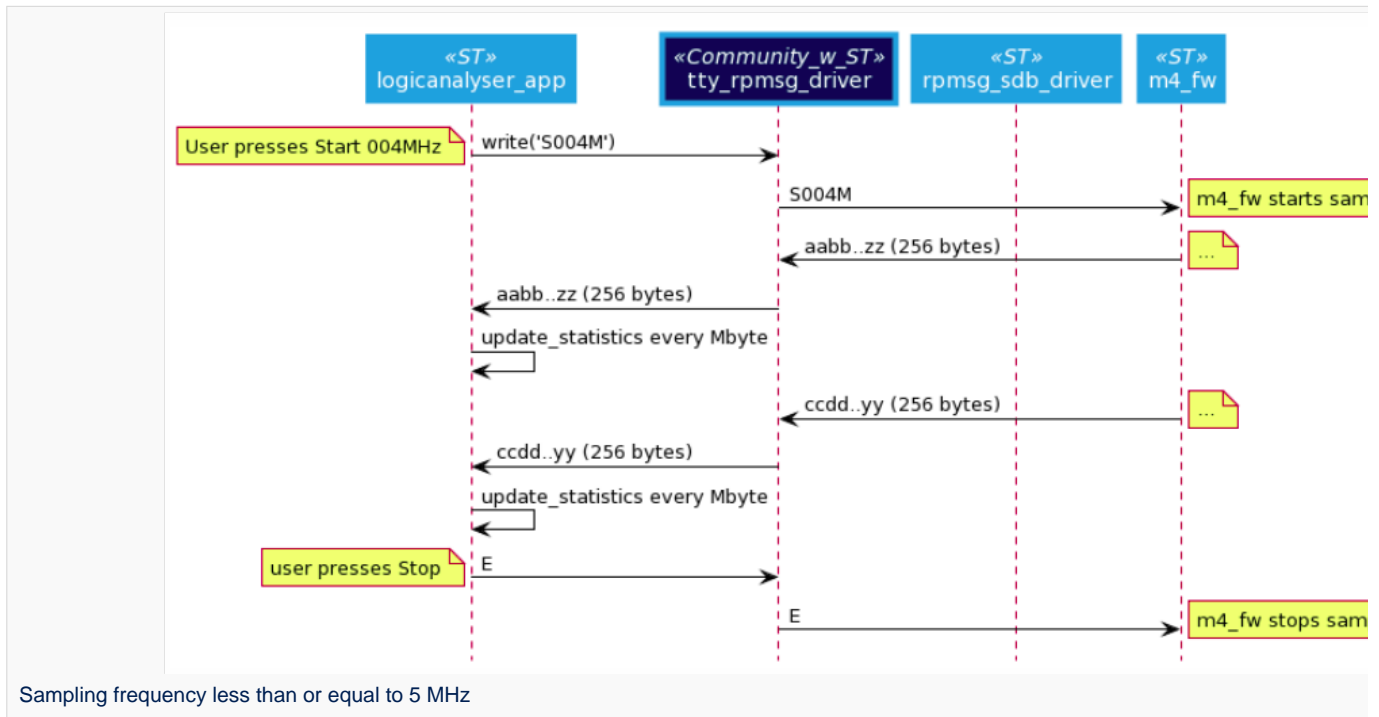
Case 1: user selects a frequency sampling of 4 MHz => case of TTY buffers (frequency scaling less than or equal to 5 MHz)

- When the Linux application receives a data buffer over TTY it checks if a new Mbyte has been fully received, and in this case it updates the statistics information.

Case 2: user selects a frequency sampling of 8 MHz => case of copy to DDR buffers (frequency scaling more than 5 MHz)

- When the Cortex-M firmware sends a "buffer full" signal via the `rpmsg_sdb` driver, Linux application updates the statistics information.

When the STOP button is pressed, the application sends the stop command to the Cortex-M firmware.



Linux application stops.

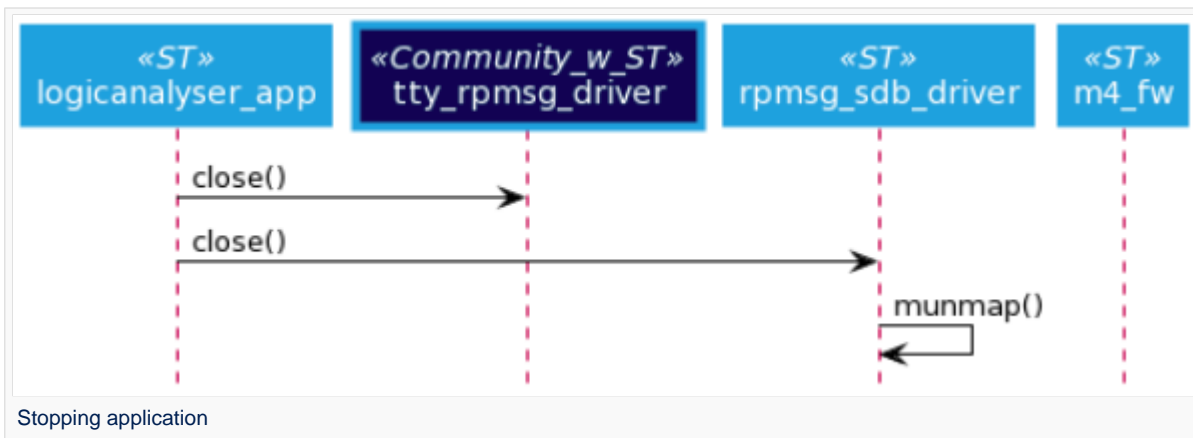
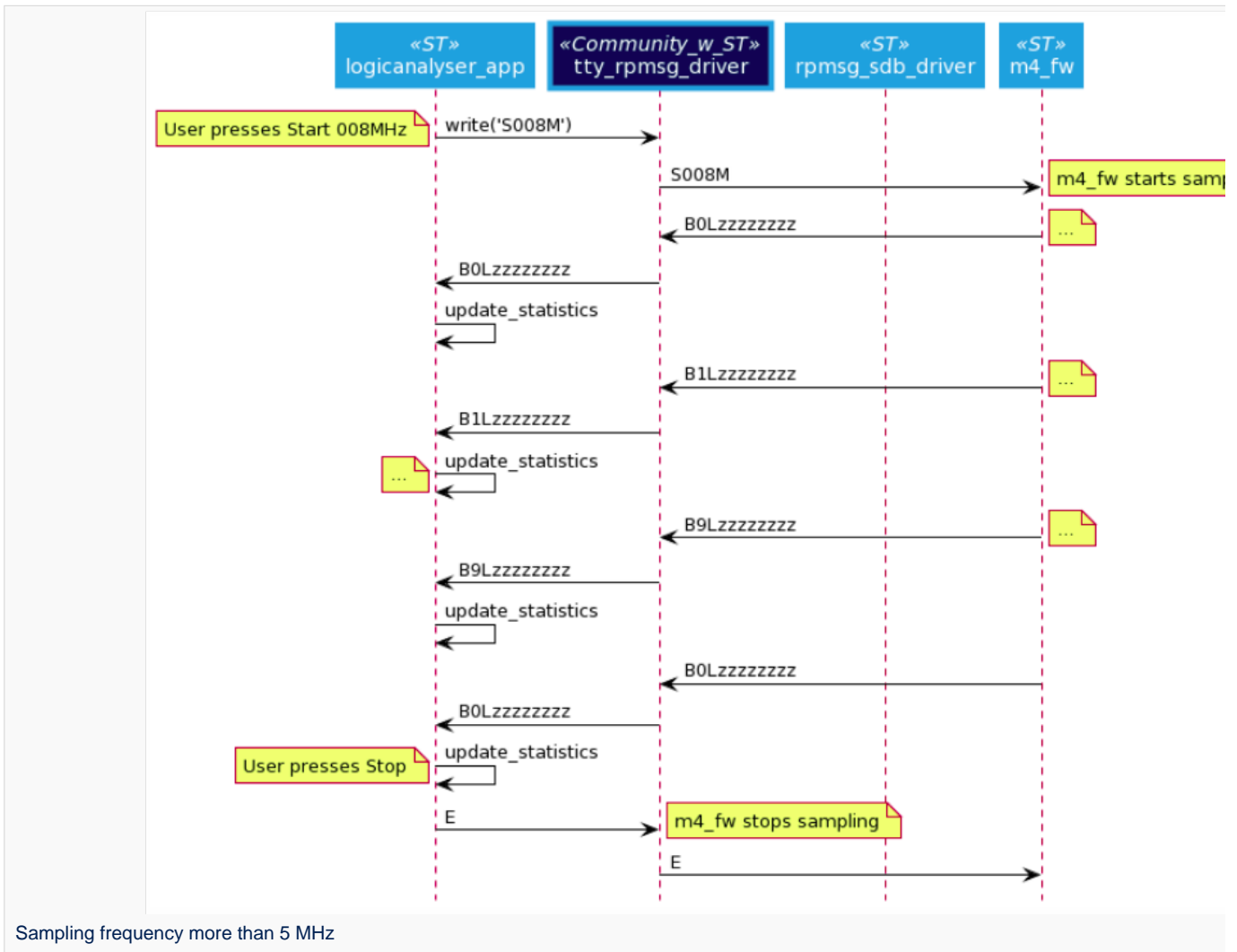
Summary of the RPMsg messages exchanged between the processors through the `rpmsg_sdb` driver:

- Information about the number of shared buffers is sent to the Cortex-M. The message is structured in a string with following format: **"Bx"**, where **x** is the number of shared buffers.
- Information about the buffer allocated and mmaped is sent to the Cortex-M.
The message is structured in a string with following format: **"BxAyyyyyyyLzzzzzzzz"**, where:
 - **x**: buffer index (32 bits, decimal format, no leading zero)
 - **yyyyyyyy**: physical address of the buffer in DDR (32 bits, 8-digit hexadecimal format, leading zero)
 - **zzzzzzzz**: length of the buffer (32 bits, 8-digit hexadecimal format, leading zero)
- Buffer filled event is received from the Cortex-M.
When the Cortex-M4 has filled a buffer, it can inform the Linux application by sending an RPMsg with following string format: **"BxLzzzzzzzz"**, where:
 - **x**: buffer index (32 bits, decimal format, no leading zero)
 - **zzzzzzzz**: length of the buffer (32 bits, 8-digit hexadecimal format, leading zero)

In the source code example, 10 buffers of 1 Mbyte each are allocated for the exchange. 3 buffers is the minimum to guarantee the real time behavior of the application. If the number of buffer needs to be increased (more than 10), then the `rpmsg_sdb_driver`, the M4 firmware, and the Linux application must be modified, as the messaging relies on a single digit for the buffer index (e.g. `"BxLyyyyyyy"` => `"BxxLyyyyyyy"`).



How to exchange data buffers with the coprocessor





9 Results

The Cortex-M CPU performs a mask and copy data operation on 1024 bytes within 75.4 μ s; This implies a maximum frequency sampling of: $1 / (75.4e-6 / 1024) \Rightarrow 13.58$ MHz. This corresponds to the maximum frequency sampling that can be achieved. In order to let a margin, the maximum frequency sampling implemented in this example is set to 12 MHz.



On this oscilloscope snapshot, a GPIO is set at the beginning of the data masking and copying algorithm, and reset at the end of the algorithm. So, 75.4 μ s are spent to mask and copy 1024 bytes of data in DDR.



10 Source code

The source code corresponding to this use case is available as a Yocto layer at:

<https://github.com/STMicroelectronics/meta-st-stm32mpu-app-logicanalyser.git>

The firmware is included in the Yocto layer as an .elf file.

The source code of the Cortex-M firmware is available at:

<https://github.com/STMicroelectronics/logicanalyser>

For firmware compilation, please have a look into: [Developer Package for STM32CubeMP1](#)



11 Key messages

Cortex-M is capable to perform basic algorithm on high data flow. If a more complex data treatment is needed, the data rate must be adapted to be able to treat it on Cortex-M.

Code instrumentation and GPIO set/reset to measure data algorithm timing are highly recommended to check real time of the full system.

For a data rate less than or equal to 5 MB/s, TTY over RPMSG should be the preferred solution.

It is not recommended to use several DMA2 streams to deal with high data rates: using a second DMA stream to transfer data to DDR does not work if both streams work at same high rate (trials at 6 MHz proved this); in this case a DMA stream0 error occurs.



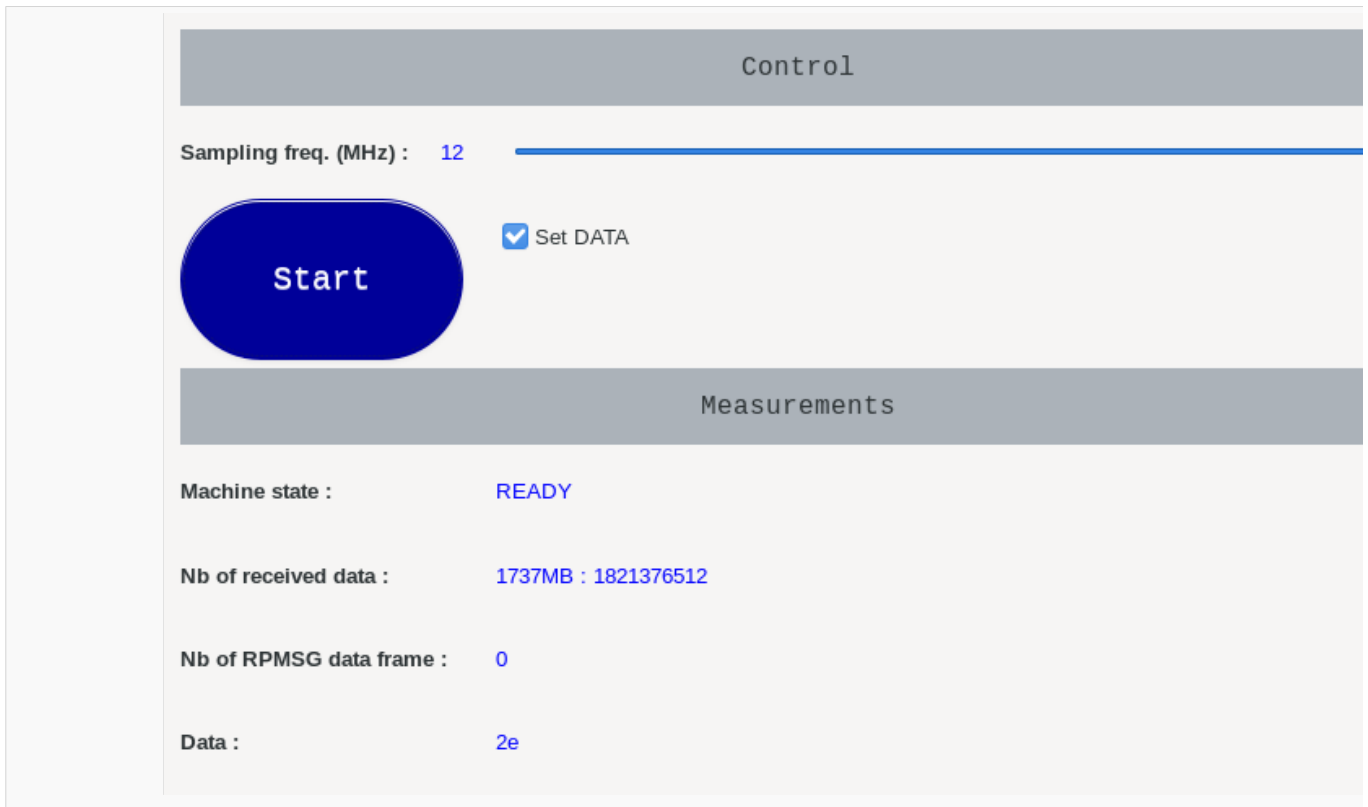
12 Usage

Please follow README.md of the Yocto layer to perform installation.

The **logicanalyser application** is launched/stopped by pressing User2 button of the STM32MP1 Discovery board.

Select the sampling frequency and click on Start to start the use case.

Snapshot view of user interface:





13 References

- drivers/rpmsg/rpmsg_tty.c
- 2.02.1 rpmsg-sdb-mod Yocto recipe.

Arm[®] is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere. 

Cortex[®]

TeleTYpewriter

Remote Processor Messaging

Doubledata rate (memory domain)

Linux[®] is a registered trademark of Linus Torvalds.

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

User Interface

System File System (See <https://en.wikipedia.org/wiki/Sysfs> for more details)

Central processing unit

Direct Memory Access