How to exchange data buffers with the coprocessor

Stable: 07.01.2021 - 10:21 | Revision: 25.10.2020 - 09:09

A quality version of this page, accepted on *7 January 2021*, was based off this revision.

Contents

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
- offload of a heavy data algorithm
- transfer of the resulting data flow to DDR buffers via DMA.

Such kind of implementation requires :

- 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.

A specific Linux driver, rpmsg_sdb (shared data buffer), has been developed to take care of such constraints.
For details on the buffer exchange mechanisms, refer to the how to exchange large data buffers with the coprocessor - principle article.

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 12, which are present on the Arduino connector. They correspond to 5 bits. The 3 remaining bits in each data byte are used to implement a packing algorithm.

After packing, the logic analyzer saves data in a binary file system named "date-time".dat, where date-time is the date and time of the system when the START button is pressed.

4 Example of static architecture for exchanging large data buffers

The example of large data buffer exchange includes:

- A Cortex-M firmware
- A Linux user land application
- A Linux rpmsg_sdb (shared data buffer) driver
- A Linux rpmsg_tty driver

File:How2ELDBArchi.jpg

800px

In the figure above, the numbers indicate the chronological order of data flows.

5 Cortex-M firmware

The Cortex-M firmware is responsible for:

- receiving messages containing the physical address and size of DDR buffer(s), from the Linux rpmsg_sdb driver
- receiving a command Start/Stop sampling (including sampling frequency) through the TTY RPMsg channel, from the Linux application
- On start request:
  - sampling the data at the requested sampling frequency
  - filtering and packing data
  - transferring via DMA the packed data to the DDR buffer by packet of 1024 bytes
  - informing the Cortex-A user interface (through the TTY RPMsg 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 web user interface.

It allows controlling:

- the sampling frequency
- the start / stop of the sampling.

The user interface displays statistics, including:

- the number of packed data received by the user interface
- the number of unpacked data decompressed by the user interface
- the number of packed data written by the user interface in the file system.

The user interface saves the packed data in a binary file.

7 Linux drivers

- The rpmsg_sdb Linux driver is responsible for the shared buffer management.
- The rpmsg_tty driver is used to communicate (transport commands and status/events) between the Cortex-M firmware and the Cortex-A user land application.

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 the rpmsg_tty driver for Cortex-M firmware control.
- It opens the rpmsg_sdb driver, then uses rpmsg_sdb IOCTL interface to allocate and mmap 3 buffers of 1Mbyte in DDR memory.

When the START button is pressed, the application sends the sampling command to the Cortex-M firmware (including the sampling frequency), and creates a "date-time.dat" binary file that is used to store the data sample in mass storage.

When the STOP button is pressed, an overrun data error or a file system full error occurs, the application sends the stop command to the Cortex-M firmware and finalizes the "date-time.dat" binary file.

When the Cortex-M firmware sends a "buffer full" signal via the rpmsg_sdb driver, the application unpacks data, writes the packed data in "date-time.dat" file, and updates the statistic information.

File:Howtobigdatamsc.jpg
800px

9 Results

Depending on the type of data available on GPIOE, the transfer rate between the Cortex-M and the Cortex-A core varies from 1 Mbyte per second (repetition factor of 7) to 8 Mbytes per second (repetition factor of 0).

In the later use case, the Cortex-M CPU is able to compress data at a rate up to 64 Mbits per second (8 Mbytes per second). This corresponds to the maximum rate that can be achieved.

10 Limitation

The limitation is due to data packing, as shown in the figure below:

File:LA perf.png

800px

On this oscilloscope snapshot, a GPIO is set at the beginning of the packing algorithm, and reset at the end of the algorithm. So, 109.5 µs are spent to pack 1024 bytes of data at a sampling frequency of 8 MHz. Increasing the frequency to 9 MHz would cross the limit : 9MHz => 111µs.

11 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

12 Usage

Please follow README.md of 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 :

File:Screenshot.jpg

800px

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.)

TeleTYpewriter

Remote Processor Messaging

Central processing unit