Dmaengine overview

Stable: 11.02.2019 - 13:21 / Revision: 10.01.2019 - 10:44

SUMMARY
This article provides basic information about the DMA engine and how STM32 DMA, DMAMUX and MDMA drivers are plugged into it.

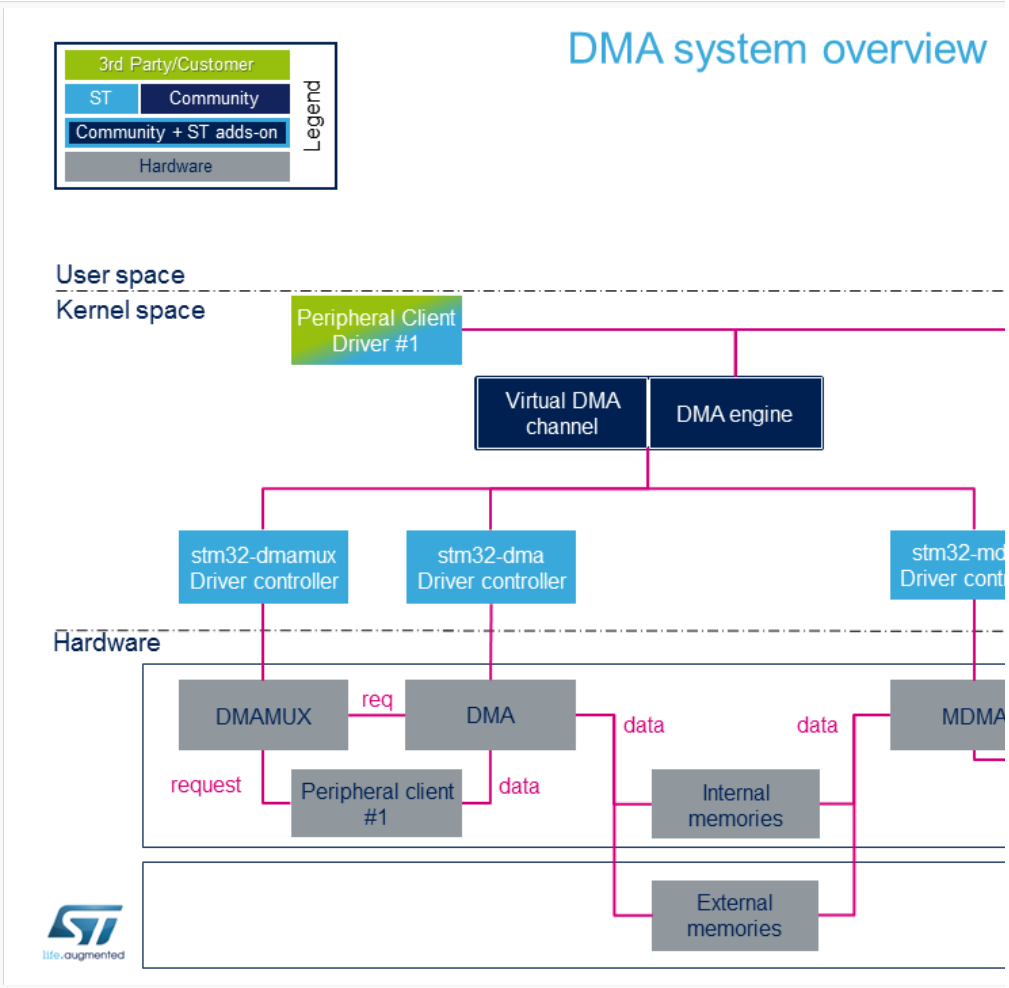
Contents

1 Framework purpose

This article provides basic information about the DMA framework. However it is worth browsing the Kernel documentation related to **DMA concept**[1].

The direct memory access (DMA) is a feature that allows some hardware subsystems to access memory independently from the central processing unit (CPU).
The DMA can transfer data between peripherals and memory or between memory and memory.

2 System overview



DMA Engine

2.1 Component description

- **Peripheral DMA client drivers**:

DMA clients are drivers that are mapped on the **DMA API**[2].

- **DMA engine**:

The DMA engine is the engine core on which all clients rely.
Refer to **DMA provider**[1] for useful information on DMA internal behaviour.

- **Virtual DMA channel support**:

The virtual DMA channel support manages virtual DMA channels and DMA requests queues. This layer is no used by DMA clients.

- **STM32 xDMA driver**:

The STM32 xDMA driver is used to develop the DMA engine API.

- **STM32 DMAMUX driver**:

The STM32 DMAMUX driver request multiplexer allows routing DMA request lines between the device peripherals and the DMA controllers.

- **DMAMUX, DMA and MDMA IP controller**:

This is the STM32 DMA controller that handles data transfers between peripherals and memories or memory and memory connected to the same bus.

DMAMUX (DMA request router): DMAMUX internal peripheral
DMA: DMA internal peripheral
MDMA : MDMA internal peripheral

- **Peripheral clients**:

Peripheral clients are peripherals where at least one DMA request line is mapped on DMAMUX.

- **Memories**:

Memories can be either internal (e.g. SRAM, RETRAM or BCKRAM) or external (DDR).

2.2 APIs description

Please refer to **DMA Engine API Guide**[3] for a clear description of the DMA framework API.

In addition, going through **Dynamic API**[4] provides insight on the DMA memory allocation API. The client has to rely on this API to properly allocate DMA buffers so that they are processed by the DMA engine without any trouble.

The document **Dynamic DMA mapping Guide**[5] can be read in conjunction with the previous one.
It presents some examples and usecases.

3 Configuration

3.1 Kernel Configuration

The DMA engine and driver are enabled throughout menu config (see Menuconfig or how to configure kernel):

For DMA:

```
Device Drivers ->
    [*] DMA Engine support ->
        [*] STMicroelectronics STM32 DMA support
```

For DMAMUX:

```
Device Drivers ->
    [*] DMA Engine support ->
        [*] STMicroelectronics STM32 dma multiplexer support
```

For MDMA

```
Device Drivers ->
    [*] DMA Engine support ->
        [*] STMicroelectronics STM32 master dma support
```

3.2 Device Tree configuration

The DT configuration can be done using the STM32CubeMX.

Refer to the following articles for a description of the DT configuration:

- For DMA: DMA device tree configuration
- For DMAMUX: DMAMUX device tree configuration
- For MDMA: MDMA device tree configuration

4 How to trace and debug the framework

4.1 How to trace

Through menuconfig, enable DMA engine debugging and DMA engine verbose debugging (including STM32 drivers):

```
Device Drivers ->
    [*] DMA Engine support ->
        [*] DMA Engine debugging
        [*]   DMA Engine verbose debugging (NEW)
```

4.2 How to debug

4.2.1 devfs

sysfs entry can be used to browse for available DMA channels.

More information can be found in sysfs.

The following command lists all the registered DMA channels:

```
Board $> ls /sys/class/dma/
dma0chan0   dma0chan13  dma0chan18  dma0chan22  dma0chan27  dma0chan31  dma0chan8  dma1cha
dma0chan1   dma0chan14  dma0chan19  dma0chan23  dma0chan28  dma0chan4   dma0chan9  dma1cha
dma0chan10  dma0chan15  dma0chan2   dma0chan24  dma0chan29  dma0chan5   dma1chan0  dma1cha
dma0chan11  dma0chan16  dma0chan20  dma0chan25  dma0chan3   dma0chan6   dma1chan1  dma1cha
dma0chan12  dma0chan17  dma0chan21  dma0chan26  dma0chan30  dma0chan7   dma1chan2  dma1cha
```

Each channel is expanded as follows:

```
Board $> ls -la /sys/class/dma/dma0chan0/
total 0
drwxr-xr-x  3 root root    0 Jun  7 21:22 .
drwxr-xr-x 34 root root    0 Jun  7 21:22 ..
-r--r--r--  1 root root 4096 Jun  9 13:11 bytes_transferred
lrwxrwxrwx  1 root root    0 Jun  9 13:11 device -> ../../../58000000.dma
-r--r--r--  1 root root 4096 Jun  9 13:11 in_use
-r--r--r--  1 root root 4096 Jun  9 13:11 memcpy_count
drwxr-xr-x  2 root root    0 Jun  9 13:11 power
lrwxrwxrwx  1 root root    0 Jun  9 13:11 subsystem -> ../../../../../../class/dma
-rw-r--r--  1 root root 4096 Jun  7 21:22 uevent
```

device indicates which DMA driver manages the channel.

echoing **in_use** indicates whether the channel has been allocated or not.

```
Board $> cat /sys/class/dma/dma0chan0/in_use
1
```

4.2.2 Debugfs

debugfs entries are available. They are documented in *Part III - Debug drivers use of the DMA-API*[4].

4.2.3 dmatest

dmatest can be used to validate or debug DMA engine and driver without using client devices. This module is more a test than a debug module. It performs a memory-to-memory copy using standard DMA engine API.

For details on how to use this kernel module, refer to [6].

5 Source code location

DMA: drivers/dma/stm32-dma.c
MDMA: drivers/dma/stm32-mdma.c
DMAMUX: drivers/dma/stm32-dmamux.c

DMA engine:

- Engine: drivers/dma/dmaengine.c
- Virtual channel support: drivers/dma/virt-dma.c

6 To go further

Very useful documentation can be found at https://www.kernel.org/doc/html/v4.19/driver-api/dmaengine/index.html.

7 References

1. ↑ [1.01.1] DMA provider
2. ↑ DMA API
3. ↑ DMA Engine API Guide
4. ↑ [4.04.1] Documentation/DMA-API.txt Dynamic DMA mapping using the generic device
5. ↑ Documentation/DMA-API-HOWTO.txt Dynamic DMA mapping Guide
6. ↑ driver-api/dmaengine/dmatest.html

Direct Memory Access

Central processing unit

Application programming interface

Doubledata rate (memory domain)

Device Tree

Device File System (See https://en.wikipedia.org/wiki/Device_file#DEVFS for more details)

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