Welcome to this presentation of the STM32G4 Direct Memory Access controller (DMA). It covers the main features of the DMA controller module, enhanced by the new DMA request multiplexer (DMAMUX) module.
The main application benefit of the DMA is to off-load the CPU for data transfers, from any memory-mapped source towards any memory-mapped destination.

STM32G4 DMA features:
- 2 DMA controllers. For each DMA controller it is possible to do:
  - Programmable block transfers with 8 concurrent channels, (each of which is independently configurable)
  - Programmable channel-based priorities
  - Data transfers via the AHB master port (connected to the bus matrix)
- There is also a DMA request router (DMAMUX), with:
  - Programmable request source selection: either from a peripheral in DMA mode or from a trigger and then internally generated
  - Synchronization mode: from a synchronization
input (hardware event) with a DMAMUX request counter

- Requests chaining: with the DMAMUX request counter to generate an event that is an input trigger or synchronization to another request/channel
<table>
<thead>
<tr>
<th>DMAMUX features</th>
<th>DMAMUX</th>
<th>DMA features</th>
<th>DMA1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of peripheral requests</td>
<td>115</td>
<td>Number of channels</td>
<td>8</td>
</tr>
<tr>
<td>Number of request generators</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of triggers</td>
<td>21</td>
<td>DMA features</td>
<td>DMA2</td>
</tr>
<tr>
<td>Number of synchronizations</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of output DMA requests</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

There are 115 peripheral requests and 4 DMAMUX request generators.
There are 21 triggers and synchronization inputs.
There are 16 DMA channels/requests.
Let's focus on the DMA controller. Each channel of the DMA controller is independently configurable:

- A channel can be assigned to a DMA hardware request from a peripheral in peripheral-to-memory, or memory-to-peripheral data transfers.
- Alternatively, a channel is assigned to a software request in memory-to-memory data transfers.
- A channel is programmed with a priority level.
- A channel is programmed for a number of data transfers at a block level.

The software can control a channel via the separated interrupts and/or flags upon programmable events such as a block transfer complete, and/or a half-block transfer complete, and/or a transfer error. A faulty channel is automatically disabled in case of a bus access error.
A channel is programmed for a number of data transfers at a block level with:

- Independent source and destination data size: 8 bit, 16 bit, or 32 bit
- Independent source and destination start address
- Independent source and destination address increment: contiguously incremented or at fixed address
- Programmable amount of data to be transferred within a block
  - Up to 65,535 source data
  - Automatically decremented by hardware

- In Circular Buffer mode (continuous data transfer from/to a peripheral), when a block transfer is completed:
  - The following programmed information is automatically reloaded by hardware:
    - Amount of data to be transferred within a block
    - Source and destination start addresses

In Circular Buffer mode (continuous data transfer from/to a peripheral), when a block transfer is completed:
- The programmed amount of data to be transferred within a block is automatically reloaded by hardware
- As well as the source and destination start addresses
In Memory-to-memory mode, a block transfer starts as soon as the channel is enabled in this mode (no hardware request).

Whereas in peripheral-to-memory, and memory-to-peripheral modes:
- A block transfer starts as soon as both 1) the channel is enabled and 2) the peripheral sends a DMA hardware request
- A DMA hardware request identifies a (single) DMA data transfer
- Each DMA hardware request is paced and granted by the DMA when each data is successfully transferred to the destination

In any mode
- Channel arbitration is reassessed between every data transfer

In Memory-to-memory mode, a block transfer starts as soon as the channel is enabled (there is no hardware request).

Whereas in peripheral-to-memory, and memory-to-peripheral modes:
- A block transfer starts as soon as both 1) the channel is enabled and 2) the peripheral sends a DMA hardware request
- A DMA hardware request identifies a (single) DMA data transfer
- Each DMA hardware request is paced and granted by the DMA when each data is successfully transferred to the destination

In any mode, channel arbitration is reassessed between every data transfer.
Each DMA channel can notify software with an interrupt triggered by any of 4 possible events:

- Half-block transfer completion
- Block transfer completion
- Transfer error
- Any of the 3 above events (i.e. global)
The DMAMUX has two main sub-blocks: the request multiplexer and the request generator.
The DMAMUX request multiplexer enables routing a DMA request from the peripherals to the DMA controllers.
The routing function is ensured by the programmable multi-channel DMA request multiplexer.
Each channel selects a unique DMA request, unconditionally or synchronously with events, from its DMAMUX synchronization inputs.
The DMAMUX may also be used as a DMA request generator from programmable events on its input trigger signals.
A DMA request multiplexer channel generates both a request to the DMA controller and an event that can be used as a synchronization input as well as a trigger input.
Do not confuse DMA request generator channels (0 to 3) with DMA request multiplexer channels (0 to 15).
For each multiplexer channel, there is a configuration register: DMAMUX_CxCR with

- Programmable input request selection (via DMAREQ_ID field).
  - For each request from a peripheral working in DMA mode, a DMAREQ_ID is assigned.
  - DMAREQ_ID = 0 corresponds to no DMA request selected.
- After having configured this channel and the DMA controller channel to which it is routed, the DMA channel can be enabled.
  - It is NOT allowed to configure two different channels with the same DMA request input.
For each multiplexer channel:
- A built-in DMA request counter is programmable (via the NBREQ field).
- A served DMA request decrements the programmed DMA request counter. At its underrun, the DMA request counter is automatically reloaded with the programmed value in NBREQ field.
- At its underrun, a DMAMUX event can be generated if enabled (via EGE field).
- 4 DMAMUX events (from Channels 0 to 3) are looped back and connected to the DMAMUX as trigger inputs and synchronization inputs. This allows request chaining for another different DMA channel, via synchronization and/or trigger.
For each multiplexer channel, there are 2 operating modes (as programmed via the SE field):

- Unconditional mode: Input request is output as is.
- Synchronized mode: A number of requests is grouped and delayed/synchronized.
When the request multiplexer channel is configured unconditionally (SE=0), the DMA request is transmitted as is and as paced by the DMA controller. When the DMA controller has served a data transfer, the DMA request is de-asserted and the built-in DMA request counter is decremented. At the counter underrun, if enabled via the EGE field, an event can be generated.
In Synchronous mode, additionally:

- The request is conditioned with:
  - A Programmable synchronization input selection (via SYNC_ID field)
  - A Programmable synchronization event: none/rising/falling/either edge (via SPOL field)
  - The single built-in request counter (via NBREQ field) that may be also used for event generation

- After the synchronization event:
  - Output DMA request is connected to the pending input request

- At the counter underrun:
  - DMA output request is disconnected from the multiplexer channel input

- A synchronization overrun flag (SOFx in DMAMUX_CSR) is reported:
  - If a new synchronization event occurs before the counter underrun
  - An interrupt is then generated if enabled (via SOIE field)
An interrupt is then generated if enabled (via SOIE field)
When the DMAMUX channel is configured in Synchronous mode, its behavior is as follows. The request multiplexer input (DMA request from the peripheral) can be pending, and it will not be forwarded on the DMAMUX request multiplexer output until the synchronization event is received. Then, the request multiplexer connects its input and output and all the peripheral requests will be forwarded. Each forwarded and granted DMA request decrements the request multiplexer counter (set at a defined programmed value). When the counter reaches zero, the connection between the DMA controller and the peripheral is cut, waiting for a new synchronization event. For each underrun of the counter, a request multiplexer can generate an optional event to synchronize and/or trigger a second DMAMUX request multiplexer channel. The same event can be used in some low-power scenarios.
to switch the system back to Stop mode without CPU intervention.
Synchronization mode can be used to automatically synchronize data transfers with a timer for example, or to condition transfers from any peripheral event that is mapped as a synchronization input. Additionally, a synchronization overflow can notify the software if a programmed number of DMA requests has not been completed between two synchronization events.
For each request generator channel:

- A DMA request can be generated, following a trigger event. And selected as input of a DMAMUX request multiplexer channel (via DMAREQ_ID field of the DMAMUX_CxCR).
- The request is generated, via the configuration register DMAMUX_RGxCR, if enabled (by GE field), with:
  - Programmable trigger input selection (via SIG_ID field).
  - Programmable trigger event: none/rising/falling/either edge (via GPOL field).
  - A built-in request counter (via GNBREQ field).
- A served DMA request decrements the programmed request counter.
- At its underrun:
  - Request counter is automatically reloaded with the programmed value in GNBREQ field.
  - Request generator stops generating a request.
- A trigger overrun flag (OFx in DMAMUX_RGSR) is reported.
  - If a new trigger event occurs before the counter underrun.
  - An interrupt is then generated if enabled (via OIE field).
• A trigger overrun flag (OFx in DMAMUX_RGSR) is reported:
  ❖ If a new trigger event occurs before the counter underrun
  ❖ An interrupt is then generated if enabled (via OIE field)
On a trigger event, a programmed number of DMA requests (GNBREQ+1) is generated. There may be a trigger overflow if two trigger events occur before the GNBREQ+1 requests and data transfers are completed.
This table shows the STM32G4 mapping of the DMAMUX request multiplexer inputs, for any channel. Assigning a request input is programmed by the DMAREQ_ID for any DMAMUX request multiplexer channel x (DMAMUX_CxCR register).

The same request input must not be mapped to two different channels.
This table shows the STM32G4 mapping of the trigger inputs and the synchronization inputs for any channel. Assigning a trigger input is programmed by the SIG_ID field of any DMAMUX request generator x (DMAMUX_RGxCR register). Assigning a synchronization input is programmed by the SYNC_ID field of any DMAMUX request multiplexer channel x (DMAMUX_CxCR register).
Each DMAMUX request generator channel can notify software of a trigger overrun.
Each DMAMUX request multiplexer channel can notify software of a synchronization overrun.

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request generator trigger overrun</td>
<td>Set when a trigger input overrun is detected</td>
</tr>
<tr>
<td></td>
<td>(before that, a programmed number of DMA requests created by the DMAMUX</td>
</tr>
<tr>
<td></td>
<td>request generator has been completed)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interrupt event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request multiplexer synchronization overrun</td>
<td>Set when a synchronization input overrun is detected</td>
</tr>
<tr>
<td></td>
<td>(before that, a programmed number of transmitted DMA requests or</td>
</tr>
<tr>
<td></td>
<td>generated DMAMUX events has been completed)</td>
</tr>
</tbody>
</table>
This table indicates the state of the DMA controller and DMAMUX according to the power mode. In Sleep and Low-power sleep modes, the DMA controller and the DMAMUX remain active and can be used for example to transfer UART or I2C received characters to memory, and afterwards to wake up the CPU.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Alive.</td>
</tr>
<tr>
<td>Low-power run</td>
<td>Alive.</td>
</tr>
<tr>
<td>Sleep</td>
<td>Alive. DMA/DMAMUX interrupts can wake up the CPU.</td>
</tr>
<tr>
<td>Low-power sleep</td>
<td>Alive. DMA/DMAMUX interrupts can wake up the CPU.</td>
</tr>
<tr>
<td>Stop 0/Stop 1</td>
<td>Clocked-off &amp; frozen. DMA/DMAMUX registers retention.</td>
</tr>
<tr>
<td>Standby</td>
<td>Powered-down. DMA/DMAMUX must be reinitialized after exiting Standby mode.</td>
</tr>
<tr>
<td>Shutdown</td>
<td>Powered-down. DMA/DMAMUX must be reinitialized after exiting Shutdown mode.</td>
</tr>
</tbody>
</table>