Hello, and welcome to this presentation of the enhanced direct memory access controller, or eDMA, module for Kinetis K series MCUs. In this session you'll learn about the eDMA and the DMA multiplexer, also known as DMAMUX, their main features and the application benefits of leveraging these functions.
In this presentation we’ll cover:

- An overview of the module
- The on-chip interconnections and inter-module dependencies
- Software configuration
- An example use case
- And a few frequently asked questions
Let’s first begin with an overview of the module.
eDMA Module Overview

• The enhanced direct memory access (eDMA) controller is a module capable of performing complex data transfers with minimal intervention from a host processor.

• The eDMA module works in conjunction with the direct memory access multiplexer (DMAMUX), which routes DMA sources, called slots, to any of the DMA channels.
eDMA and DMAMUX Features

**eDMA Features**

- 16 or 32 channels, depending on the Kinetis device
- Each channel has its own transfer control descriptor (TCD)
  - Supports two-deep nested transfers called minor and major loops
- Fixed-priority and round-robin channel arbitration

**DMAMUX Features**

- One or two muxes, depending on number of DMA channels
- DMA routing is typically static, but the mux can be reprogrammed
- Two modes of operation
  1. Normal mode
  2. Periodic triggering mode available on first 4 channels

---

eDMA and DMAMUX Features

**eDMA Features**

- The eDMA controller offers 16 or 32 channels, which are available to perform complex data transfers with minimal CPU intervention. The number of channels is dependent on the Kinetis device. To find out exactly how many channels are available on your MCU, please refer to the reference manual.

- Each channel has its own transfer control descriptor (TCD) that supports two-deep nested transfer operations called minor and major loops, respectively. The TCDs store all the needed information to make the transactions, such as the source and destination addresses and the transfer size. Each TCD needs 32-bytes, which are stored in local memory. A channel can be activated either by software, by another channel through channel linking or by another peripheral through hardware requests.

- Fixed-priority and a round-robin schemes are allowed for channel arbitration. The channels can report through an interrupt request when the half and/or the total of the major loop is completed, which is useful to manage double buffers.

**DMAMUX Features**

- Kinetis devices contain one or two multiplexers, depending on the amount of channels. For 16 channel devices, one DMAMUX is available and for 32 channel devices, 2 instances of DMAMUX are available.
The eDMA module is partitioned into two major components:

1. **eDMA engine**
2. **Transfer-control descriptor (TCD) local memory**

The eDMA engine performs:
- The source and destination address calculations, and
- Data movement operations

The transfer-control descriptor local memory contains the transfer-control descriptors for each eDMA channel.
DMAMUX Block Diagram

The DMAMUX (DMA multiplexer) routes the DMA sources to any of the DMA channels.

Note: Do not set multiple channels with the same DMA source.

DMAMUX Block Diagram

The DMA multiplexer routes the DMA sources to the DMA channels.

Please note that a DMA source should be configured to use only one channel.
The TCD contains all the information about the data movement:
- Source address, address increment, and transfer size
- Destination address, address increment, and transfer size
- Number of bytes to transfer
- Number of major loops to execute

Each TCD can follow a two-deep nested loop (major and minor transfer loops).

TCDs can be processed in a chain so that the eDMA will automatically load the next TCD.

The eDMA uses a 32-byte transfer control descriptor.

---

eDMA Transfer Control Descriptor (TCD)

The TCD contains all the information about the data movement. This includes the source and destination addresses, the address increment after each transfer and the transfer size. The TCD also includes the number of bytes to transfer and the number of major loops to execute.

Each TCD can be a two-deep nested loop such as major and minor transfer loops.

The TCDs can be processed in a chain so that the eDMA will automatically load the next TCD. This is called channel linking.

Each TCD needs 32-bytes to store all this information.
This table shows the memory map of a TCD. It contains all the registers that the eDMA needs to accomplish the transaction. The TCD is responsible for channel activation, address path, data path, and control for source and destination.

<table>
<thead>
<tr>
<th>eDMA Offset</th>
<th>TCDn Register Name</th>
<th>Abbreviation</th>
<th>Width (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Source Address</td>
<td>TCDn_SADDR</td>
<td>32</td>
</tr>
<tr>
<td>0x04</td>
<td>Transfer Attributes</td>
<td>TCDn_ATTR</td>
<td>16</td>
</tr>
<tr>
<td>0x05</td>
<td>Signed Source Address Offset</td>
<td>TCDn_SOFF</td>
<td>16</td>
</tr>
<tr>
<td>0x08</td>
<td>Minor Byte Count</td>
<td>TCDn_NRVTFS</td>
<td>32</td>
</tr>
<tr>
<td>0x0C</td>
<td>Last Source Address Adjustment</td>
<td>TCDn_SLAST</td>
<td>32</td>
</tr>
<tr>
<td>0x10</td>
<td>Destination Address</td>
<td>TCDn_DADDR</td>
<td>32</td>
</tr>
<tr>
<td>0x14</td>
<td>Current Minor Loop Link, Major Loop Count</td>
<td>TCDn_CITER</td>
<td>16</td>
</tr>
<tr>
<td>0x16</td>
<td>Signed Destination Address Offset</td>
<td>TCDn_DOFF</td>
<td>16</td>
</tr>
<tr>
<td>0x18</td>
<td>Last Destination Address Adjustment/Scatter Gather Address</td>
<td>TDDn_DLAST_SGA</td>
<td>32</td>
</tr>
<tr>
<td>0x1C</td>
<td>Beginning Minor Loop Link, Major Loop Count</td>
<td>TCDn_BITER</td>
<td>16</td>
</tr>
<tr>
<td>0x1E</td>
<td>Control and Status</td>
<td>TCDn_CSR</td>
<td>16</td>
</tr>
</tbody>
</table>
Nested Loop Example

This example demonstrates two-deep nested loops.

Each DMA request will move the amount of bytes configured in the NBYTES (minor byte count) register. This corresponds to a minor loop. Once this is completed, another DMA request will transfer another minor loop. The CITER (major loop count) register will decrement each time a minor loop is transferred. When the CITER register is equal to zero, that means a major loop has been completed.

In other words, a minor loop contains NBYTES bytes and a major loop contains CITER minor loops.
Minor Loop Channel Linking Example

This example demonstrates minor loop channel linking.

Each DMA request will move the amount of bytes configured in the NBYTES (minor byte count) register like the last example. This corresponds to a minor loop. Once this is completed, a DMA request for the linked channel is generated. Another DMA request will transfer another minor loop and a new DMA request for the linked channel will be generated.
Major Loop Channel Linking Example

Finally, this example demonstrates major loop channel linking.

Major loop channel linking works exactly like minor loop channel linking, except the DMA request for the linked channel is done once the major loop is completed.
Periodic Triggering: Normal Operation Example

The DMA periodic trigger capability allows the system to schedule regular DMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the peripheral to the DMA until a trigger event has been seen.

There is one DMA request per peripheral request, but the DMA request is delayed until the trigger asserts. This helps ensure a fixed transfer frequency.

This image shows a normal operation example where the trigger is periodic. The DMA request is generated when both the peripheral request and the trigger are present.
Periodic Triggering: Ignored Trigger Example

Now let's go over an example where the trigger is ignored.

After the DMA request has been serviced, the peripheral will negate its request, effectively resetting the gating mechanism until the peripheral reasserts its request and the next trigger event is seen.

This means that if a trigger is seen, but the peripheral is not requesting a transfer, then that trigger will be ignored. The peripheral request is not active when the second trigger asserts. The peripheral has to wait until the third trigger before a new DMA request is asserted.
Next, let’s discuss on-chip interconnections and inter-module dependencies.
eDMA Interconnections

This diagram displays how other peripherals make requests to the DMAMUX, which is connected to the eDMA controller through the peripheral bridge. This process makes it possible for transactions to happen through the crossbar switch – mastering the data movement.
Now, let’s move on to discuss software configuration.
Kinetis SDK eDMA Driver Example

This is an example of loop transfers in the eDMA using Kinetis SDK.

In this example, we will use channel 1.

We will transfer eight samples of 1-byte length.

Two samples on each request.

We will need two interrupts on each loop, so we configure two TCDs. The interrupts help manage a ping pong buffer.

Step 1, Initialize the DMA module. The channel arbitration will be round-robin. Once the configuration structure is filled, we can call the EDMA_DRV_Init() function.

The 2nd step is to request a DMA channel through the EDMA_DRV_RequestChannel() function. In this case the request will be generated by GPIO PORT C.

The 3rd step is to configure the TCD. The loop transfer is configured through the EDMA_DRV_ConfigLoopTransfer() function.

The callback needs to be installed through the EDMA_DRV_InstallCallback() function.

Finally, the channel is started with the EDMA_DRV_StartChannel() function. The DMA is waiting for a request.

Additionally, a channel can be stopped through the EDMA_DRV_StopChannel() function.

And a channel can be released through the EDMA_DRV_ReleaseChannel() function.
eDMA Driver Example Results

Once the channel is started, the destination buffer is blank. When the first request is asserted, two bytes are transferred. The second request transfers two more bytes, which results in the first of the two interrupts. The third request copies the other two bytes. And finally, the fourth request moves the last two bytes – this generates the second interrupt.
Let's review two example use cases.
Audio Equalizer Example

Let's say we want to develop an audio equalizer using Kinetis MCUs.

For this application, we'll need a Kinetis device with USB support to communicate to a PC to acquire the original audio.

An audio codec is required to translate the audio data into sound on your speaker.

The Kinetis MCU will need to configure the audio codec. Let's assume communication between these devices is I2C.

Finally, use I2S to transfer audio data from the Kinetis MCU to the audio codec. The eDMA helps automate the data flow so the CPU will be free to process the audio data.
Audio Equalizer Implementation

The audio data is received from a USB into a buffer. We can use an eDMA channel to take this data and move it to a ping pong buffer dedicated for the audio samples.

A second DMA channel can be used to move the processed data from the ping pong audio buffer to the SAI TX (Transmit) FIFO. This eDMA will be triggered by the SAI TX FIFO watermark signal.

With this use case, the CPU can handle other tasks such the audio filter. There is no need to send audio data to the audio codec, since everything is done in the background with the eDMA.
Finally, let’s review some frequently asked questions.
**Question:** What do eDMA bus cycles look like?

**Answer:** eDMA bus cycles look like any other bus cycle. Slave device architecture (including the FlexBus) don’t treat DMA cycles differently than core requests, so the bus timing will be the same as if the core had requested a bus cycle of the same size.

**Question:** Can I assign a single DMA request source to multiple channels? For example, use SPI0 Rx to trigger DMA channels 0 and 1 simultaneously.

**Answer:** No, this can result in unpredictable behavior. In some cases such as when you aren’t reading from a FIFO that will update as it is read, you can use channel linking to get a similar effect. If you need two copies of data from a FIFO, then you could use one DMA channel to read from the FIFO and write the data to memory. Then, link to a second channel that makes a copy of the data in memory.
This concludes our presentation on the eDMA module for Kinetis MCUs.

For more information on eDMA, please visit the application notes listed here.

We also invite you to visit us on the web Freescale.com/Kinetis and check out our Kinetis community page.