Direct Memory Access

ECE 362
https://engineering.purdue.edu/ee362/

© Rick
Reading Assignment

- Textbook, Chapter 19, Direct Memory Access, pp. 469 – 480.
  - Textbook, Chapter 15.3, PWM Output, pp. 384 – 395
Future Reading Assignment

- Textbook, Chapter 22, Serial Communication Protocols, pp. 527 – 598
  - It’s a long chapter.
  - Let’s first look at Section 22.3, SPI, pp. 568–577.
  - Don’t worry so much about the USB section.
    - Read that only if you’re curious.
    - Not much we can do with that.
    - Other books are better for understanding USB.
Remember setting up the timer?

- In lab 5, you configured a timer and then issued an inline assembly WFI instruction.
- This was to illustrate that the CPU wasn’t doing anything (except during the timer ISR).
- The timer worked autonomously—all by itself.
Copying memory autonomously

• We can configure the microcontroller to copy words of memory from a source to a destination.

• We don’t need for the CPU to issue loads and stores to do this. It can be "asleep."

• This is traditionally called direct memory access (DMA).
  - Historically, it was used by peripherals to access memory "directly" without needing the CPU to explicitly load and store memory and peripheral words.

• DMA is much like a coprocessor that reads and writes memory locations, and increments (or decrements) its registers.
DMA Controller

- Multiple independent "channels".
  - Really this means it's just sets of src/dest registers.
  - Priorities
  - Different transfer sizes (byte, halfword, word)
  - Circular buffers
  - Up to 65535 data per request.
Don’t always transfer to/from mem

- DMA can transfer memory-to-peripheral.
- DMA can transfer peripheral-to-memory.
- DMA can transfer peripheral-to-peripheral.

- By "peripheral," often we mean any "I/O register"
DMA Block Diagram

- Turn on clock with RCC->AHBENR
- Configure channel registers.
- Enable.
- Hands-free after that.
DMA Registers

- Each DMA channel has 4 registers:
  - CMAR: Channel Memory Address Register
  - CPAR: Channel Peripheral Address Register
  - CNDTR: Channel Number of Data Register
  - CCR: Channel Configuration Register
### DMA Register Map

#### Table 34. DMA register map and reset values

| Offset | Register | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|--------|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0x00   | DMA_ISR  |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x04   | DMA_IFCR |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x08   | DMA_CCR1 |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C   | DMA_CNDTR1|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x10   | DMA_CPAR1|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x14   | DMA_CMAR1|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|        |          |     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

### Controller Registers

- **DMA_ISR**
- **DMA_IFCR**
- **DMA_CCR1**
- **DMA_CNDTR1**
- **DMA_CPAR1**
- **DMA_CMAR1**

### Per Channel Registers

- **MEMSEL**
- **PL[1:0]**
- **MSIZE[1:0]**
- **PSIZE[1:0]**
- **MINC**
- **PINC**
- **EIRC**
- **DIR**
- **TEIE**
- **HTIE**
- **TCIE**
- **EN**
- **PA[3:0]**
- **MA[31:0]**

The table shows the reset values for each register and the offsets for different types of registers.
DMA Controller Registers

- DMA_ISR: Interrupt Status Register
- 4 status flags for each channel:
  - GIF: Global Interrupt Flag
  - TCIF: Transfer Complete Flag
  - HTIF: Half Transfer Flag
  - TEIF: Transfer Error Flag

...only 5 channels on STM32F05...
DMA Controller Registers

- **DMA_IFCR**: Interrupt Flag Clear Register
  - 4 bits to clear all the previous status flags
  - Write a ‘1’ to clear the corresponding flag.

```
...only 5 channels on STM32F05...
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CTEIF4</td>
<td>CHTIF4</td>
<td>CTCIF4</td>
<td>CGIF4</td>
</tr>
<tr>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
<td>w</td>
</tr>
</tbody>
</table>
There is no DMA_CSELR

- Your textbook mentions a DMA_CSELR (Channel Select Register)
  - We don’t have that on an STM32F05x.
  - It does exist on an STM32F09x.
Configuring DMA

- Put the source or destination memory address in DMA_CMARx.
- Put address of the **Data Register** of the source or destination peripheral in DMA_CPARx.
- Put the number of data elements in DMA_CNDTR.
  - (not the number of bytes – number of elements)
- Configure DMA_CCRx.
Configuring DMA_CCRx

- **EN**: enable the DMA transfer
- **TCIE**: Transfer Complete Interrupt Enable
- **HTIE**: Half Transfer Interrupt Enable  (Why would we want such a thing?)
- **TEIE**: Transfer Error Interrupt Enable
- **DIR**: Data Transfer Direction (0: Read from peripheral, 1: Read from memory)
- **CIRC**: Circular Mode  (What is this?)
- **PINC**: Peripheral Increment Mode
- **MINC**: Memory Increment Mode
- **PSIZE**: Peripheral Size (00: 8-bit, 01: 16-bit, 10: 32-bit)
- **MSIZE**: Memory Size (00: 8-bit, 01: 16-bit, 10: 32-bit)
- **PL**: Channel Priority Level (00: Low, 01: Medium, 10: High, 11: Very high)
- **MEM2MEM**: Memory to Memory Mode

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Simplest Example: Mem-to-Mem

- It's also a contrived example.
  - Maybe not a good reason to do this other than illustration of how DMA works.

- Use DMA channel 1 to:
  - Read bytes from memory and write the bytes to a different part of memory.
  - Interrupt when complete.
  - CMAR is source address
  - CPAR is destination address
  - CNDTR is number of bytes
  - How should we set the CCR?
const char src[32] = "This is a string of characters.";

int main(void)
{
    char dst[32] = { 0 };
    RCC->AHBENR |= RCC_AHBENR_DMA1EN;
    DMA1_Channel1->CCR &= ~DMA_CCR_EN;  // Make sure DMA is turned off
    DMA1_Channel1->CMAR = (uint32_t)src;  // Copy from address in CMAR
    DMA1_Channel1->CPAR = (uint32_t)dst;  // Copy to address in CPAR
    DMA1_Channel1->CNDDR = sizeof src;  // Copy this many data.
    DMA1_Channel1->CCR |= DMA_CCR_DIR;  // Read from "memory"
    DMA1_Channel1->CCR |= DMA_CCR_TCIE;  // Interrupt when done.
    DMA1_Channel1->CCR &= ~DMA_CCR_HTIE;  // And not when half done.
    DMA1_Channel1->CCR &= ~DMA_CCR_MINC;  // Increment CMAR as we copy
    DMA1_Channel1->CCR &= ~DMA_CCR_PINC;  // Increment CPAR as we copy
    DMA1_Channel1->CCR |= DMA_CCR_MEM2MEM;  // Memory-to-memory copy
    DMA1_Channel1->CCR &= ~DMA_CCR_PL;  // Priority: 00: Low
    NVIC->ISER[0] = 1<<DMA1_Channel1_IRQn;  // Enable the interrupt.
    DMA1_Channel1->CCR |= DMA_CCR_EN;  // Engage!

    asm("wfi");
    for(;;);
}
void DMA1_Channel1_IRQHandler(void) {
    if (DMA1->ISR & DMA_ISR_GIF1) { // If the Global Interrupt flag set...
        DMA1->IFCR |= DMA_IFCR_CGIF1; // Clear the flag by writing to it.
    }
    if (DMA1->ISR & DMA_ISR_TCIF1) { // If the Transfer Complete flag set...
        DMA1->IFCR |= DMA_IFCR_CTCIF1; // Clear the flag by writing to it.
    }
    if (DMA1->ISR & DMA_ISR_HTIF1) { // If the Half Transfer flag set...
        DMA1->IFCR |= DMA_IFCR_CHTIF1; // Clear the flag by writing to it.
    }
    if (DMA1->ISR & DMA_ISR_TEIF1) { // If the Transfer Error flag set...
        DMA1->IFCR |= DMA_IFCR_CTEIF1; // Clear the flag by writing to it.
    }
}
Other examples

• Appendix A has lots of examples. Some are good.
  – A.5.1 DMA Channel Configuration
  – A.7.9 DMA one-shot mode on ADC
  – A.7.10 DMA circular mode on ADC
  – A.8.1 DMA on DAC with timer6/7 trigger.
  – A.8.7 DMA on DAC with timer7 trigger.
  – A.8.12 DMA memory-to-DAC with timer7 trigger.
A.5.1 DMA Channel Config

- Initialize peripheral (not shown)
- DMA channel init.
- MSIZE: 16-bit
- PSIZE: 16-bit
- Why Channel 1?

DMA Channel Configuration sequence code example

```c
/* The following example is given for the ADC. It can be easily ported on
any peripheral supporting DMA transfer taking of the associated channel
to the peripheral, this must check in the datasheet. */
/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA transfer to be performs on channel 1 */
/* (6) Configure increment, size and interrupts */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_DMAEN; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t)(ADC_array); /* (4) */
DMA1_Channel1->CNDTR = 3; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
| DMA_CCR_TEIE | DMA_CCR_TCIE ; /* (6) */
DMA1_Channel1->CCR |= DMA_CCR_EN; /* (7) */
/* Configure NVIC for DMA */
/* (1) Enable Interrupt on DMA Channel 1 */
/* (2) Set priority for DMA Channel 1 */
NVIC_EnableIRQ(DMA1_Channel1_IRQn); /* (1) */
NVIC_SetPriority(DMA1_Channel1_IRQn, 0); /* (2) */
```
Why would you do this?

- So you don’t have to wait on the ADC and pull the values with the CPU.
- Channel 1 again?
A.7.10 DMA circular mode on ADC

- Continually scan ADC.
  - The CIRC flag means that the DMA will restart.
  - Most recent values will always be fresh in the ADC_array[].
  - Channel 1?
A.8.1 DMA on DAC with TIM6/7

Independent trigger without wave generation code example

/* (1) Enable the peripheral clock of the DAC */
/* (2) Enable DMA transfer on DAC ch1 and ch2,
   enable interrupt on DMA underrun DAC ch1 and ch2,
   enable the DAC ch1 and ch2,
   select TIM6 as trigger by keeping 000 in TSEL1
   select TIM7 as trigger by writing 010 in TSEL2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN;  /* (1) */
DAC->CR |= DAC_CR_TSEL2_1 | DAC_CR_DMAUDRIE2 | DAC_CR_DMAEN2
         | DAC_CR_TEN2  | DAC_CR_EN2
         | DAC_CR_DMAUDRIE1 | DAC_CR_DMAEN1 | DAC_CR_BOFF1
         | DAC_CR_TEN1  | DAC_CR_EN1;  /* (2) */
DAC->DHR12R1 = DAC_OUT1_VALUE;  /* Initialize the DAC value on ch1 */
DAC->DHR12R2 = DAC_OUT2_VALUE;  /* Initialize the DAC value on ch2 */

• What is underrun?
• What are TIM6 and TIM7?
• What is TSEL?
• Which DMA channel?
  - Or channels?
Timers 6 & 7

- Relax. They’re simple.
  - They’re called the basic timers.
  - Why didn’t we start with these in the first place?
DAC_CR TSEL field

- Note: We don’t have a dual DAC on STM32F051

Bits 5:3 TSEL[2:0]: DAC channel1 trigger selection
- These bits select the external event used to trigger DAC channel1.
  - 000: Timer 6 TRGO event
  - 001: Timer 3 TRGO event
  - 010: Timer 7 TRGO event
  - 011: Timer 15 TRGO event
  - 100: Timer 2 TRGO event
  - 101: Reserved
  - 110: EXTI line9
  - 111: Software trigger

Note: Only used if bit TEN1 = 1 (DAC channel1 trigger enabled).
Simultaneous trigger without wave generation code example

/* (1) Enable the peripheral clock of the DAC */
/* (2) Enable DMA transfer on DAC ch1 for both channels,
  enable the DAC ch1 and ch2,
  select TIM7 as trigger by writing 010 in TSEL1 and TSEL2 */
RCC->APB1ENR |= RCC_APB1ENR_DACEN; /* (1) */
DAC->CR |= DAC_CR_TSEL1_1 | DAC_CR_TEN2 | DAC_CR_EN2
  | DAC_CR_TSEL2_1 | DAC_CR_TEN1 | DAC_CR_EN1; /* (2) */
/* Initialize the dual DAC value */
DAC->DHR12RD = (uint32_t)((2048 << 16) + 2048);

A.8.7 DMA on DAC with timer7

- Do you see the DMA setup in step (2)?
- Neither do I.
- Does our microcontroller have one of this timer?
A.8.12 DMA mem-to-DAC with TIM7

*/ (1) Enable DMA transfer on DAC ch1 for both channels, enable interrupt on DMA underrun DAC, enable the DAC ch1 and ch2, select TIM7 as trigger by writing 010 in TSEL1 and TSEL2 */
DAC-&gt;CR |= DAC_CR_TSEL1_1 | DAC_CR_TEN2 | DAC_CR_EN2 |
| DAC_CR_TSEL2_1 | DAC_CR_DMAUDRIE1 | DAC_CR_DMAEN1 |
| DAC_CR_TEN1 | DAC_CR_EN1; /* (1) */

*/ (1) Enable the peripheral clock on DMA */
*/ (2) Configure the peripheral data register address */
*/ (3) Configure the memory address */

*/ (4) Configure the number of DMA transfer to be performs on channel 3 */
*/ (5) Configure increment, size (32-bits), interrupts, transfer from memory to peripheral and circular mode */

*/ (6) Enable DMA Channel 3 */
RCC-&gt;AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
DMA1_Channel3-&gt;CPAR = (uint32_t)(&amp;DAC-&gt;DHR12RD)); /* (2) */
DMA1_Channel3-&gt;CMAR = (uint32_t)signal_data; /* (3) */
DMA1_Channel3-&gt;CNCTR = SIGNAL_ARRAY_SIZE; /* (4) */
DMA1_Channel3-&gt;CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_1 | DMA_CCR_PSIZE_1 |
| DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel3-&gt;CCR |= DMA_CCR_EN; /* (6) */

- DMA setup done here.
- Notice this example explicitly uses DMA channel 3! Why?
- How do you choose which DMA channel to use?
  - Use the one you’re required to use.
  - Using the wrong one does nothing.
11.3.7 DMA request mapping

DMA controller

The hardware requests from the peripherals (TIMx, ADC, DAC, SPI, I2C, and USARTx) are simply logically ORed before entering the DMA. This means that on one channel, only one request must be enabled at a time.

The peripheral DMA requests can be independently activated/de-activated by programming the DMA control bit in the registers of the corresponding peripheral.

*Table 30* and *Table 31* list the DMA requests for each channel.
DMA Request Mapping Table

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>Channel 1</th>
<th>Channel 2</th>
<th>Channel 3</th>
<th>Channel 4</th>
<th>Channel 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>ADC(1)</td>
<td>ADC(2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPI</td>
<td></td>
<td>SPI1_RX</td>
<td>SPI1_TX</td>
<td>SPI2_RX</td>
<td>SPI2_TX</td>
</tr>
<tr>
<td>USART</td>
<td></td>
<td>USART1_TX(1)</td>
<td>USART1_RX(1)</td>
<td>USART1_TX(2)</td>
<td>USART1_RX(2)</td>
</tr>
<tr>
<td>I2C</td>
<td></td>
<td>I2C1_TX</td>
<td>I2C1_RX</td>
<td>I2C2_TX</td>
<td>I2C2_RX</td>
</tr>
<tr>
<td>TIM1</td>
<td></td>
<td>TIM1_CH1</td>
<td>TIM1_CH2</td>
<td>TIM1_CH4</td>
<td>TIM1_CH3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TIM1_TRIG</td>
<td>TIM1_COM</td>
<td>TIM1_UP</td>
</tr>
<tr>
<td>TIM2</td>
<td>TIM2_CH3</td>
<td>TIM2_UP</td>
<td>TIM2_CH2</td>
<td>TIM2_CH4</td>
<td>TIM2_CH1</td>
</tr>
<tr>
<td>TIM3</td>
<td></td>
<td>TIM3_CH3</td>
<td>TIM3_CH4</td>
<td>TIM3_CH1</td>
<td>TIM3_TRIG</td>
</tr>
<tr>
<td>TIM6 / DAC</td>
<td></td>
<td></td>
<td>TIM6_UP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DAC_Channel1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIM15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIM16</td>
<td></td>
<td></td>
<td>TIM16_CH1(1)</td>
<td>TIM16_UP(1)</td>
<td>TIM16_CH1(2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TIM16_UP(2)</td>
<td>TIM16_UP(2)</td>
<td></td>
</tr>
<tr>
<td>TIM17</td>
<td>TIM17_CH1(1)</td>
<td>TIM17_UP(1)</td>
<td>TIM17_CH1(2)</td>
<td>TIM17_UP(2)</td>
<td></td>
</tr>
</tbody>
</table>

1. DMA request mapped on this DMA channel only if the corresponding remapping bit is cleared in the SYSCFG_CFRG1 register. For more details, please refer to Section 10.1.1: SYSCFG configuration register 1 (SYSCFG_CFRG1) on page 173.

2. DMA request mapped on this DMA channel only if the corresponding remapping bit is set in the SYSCFG_CFRG1 register. For more details, please refer to Section 10.1.1: SYSCFG configuration register 1 (SYSCFG_CFRG1) on page 173.

- e.g., You may only use DMA channel 3 with SPI1_TX.
- What DMA channel must you use with the DAC?
- What DMA channel must you use with I2C1_RX?
- What DMA channel must you use with TIM3_CH4?
- What happens if you want to use all four at the same time?
  - You don't.
- What are all these TIM things?
Timer DMA requests

- TIMx_CHy: Capture/Compare on TIMx channel y.
- TIMx_UP: Counter update (when the counter wraps around to ARR or 0).
- TIMx_TRIG: Trigger event (counter start, stop, initialization or count by int/ext trigger)
- TIMx_COM: CC Control bits (CCxE, CCxNE, OCxM) have been updated
DMA Triggers

- Some peripherals have a natural means of initiating DMA transfers.
  - e.g. SPI read, write, UART, I2C
- Others are triggered directly by timers.

**DAC:** (DAC_CR TSEL1[2:0])
- 000: Timer 6 TRGO event
- 001: Timer 3 TRGO event
- 010: Timer 7 TRGO event
- 011: Timer 15 TRGO event
- 100: Timer 2 TRGO event
- 101: Reserved
- 110: EXTI line9
- 111: Software trigger

**ADC:** (ADC_CFGR1 EXTSEL[2:0])
- 000: TRGO: TIM1_TRGO
- 001: TRG1: TIM1_CC4
- 010: TRG2: TIM2_TRGO
- 011: TRG3: TIM3_TRGO
- 100: TRG4: TIM15_TRGO
- 101: TRG5: Reserved
- 110: TRG6: Reserved
- 111: TRG7: Reserved

TIM1_CC4 is enabled by setting bit CC4DE in TIM1_DIER.

TIMx_TRGO is enabled by setting MMS[2:0] in TIMx_CR2.
Shortcomings of DMA

- It’s nice to have things taken care of automatically, but DMA can’t do everything.
  - It can transfer peripheral values into memory, but it can’t modify them or do calculations on them.
  - If you wanted to use DMA to copy ADC to DAC, you could do it, but it won’t modify the signal.
    - Copying ADC to the serial port might be a useful thing to do though.
    - You might use one DMA channel to copy from ADC to memory, a second to copy memory to DAC, and regular computation to modify the memory in the middle.
  - If you want to operate on an individual peripheral register at relatively low speed (<50kHz), you’re probably better off doing so with an interrupt handler.
Application:

• Addressable RGB LED strip:
  https://www.sparkfun.com/products/12025

• 60 RGB LEDs per meter.
  - Each LED uses 8-bits to specify each of R, G, B.
  - 24-bit color selection per LED.

• "One-wire" serial control.
Unique Interface

- You don’t just send bits, but high/low patterns of particular lengths to indicate 1s and 0s.
- Send 24 bits like this per LED.

![Data transfer time chart](chart.png)

- Sequence chart:
  - 0 code
    - T0H
    - T0L
  - 1 code
    - T1H
    - T1L
  - RET code
    - Treset

![Waveform diagram](waveform.png)

- "0": 1.162us
- "1": 1.162us
- \(7 \times 0.166\) = 6MHz

\[
0.333\text{us} \\
0.667\text{us} \\
0.833\text{us} \\
0.500\text{us}
\]
Use PWM

- Configure timer so that prescaler output is 6MHz.
- ARR should be set to 7-1.
- Configure channel mode for PWM mode 1.
- Now the CCR can be set to 2 to represent a "0" and to 4 to represent a "1".
- How can we change the duty cycle for each period?
  - Ideally, we’d like to have a region of memory to represent 2- and 4-width high marks.
Use DMA with PWM

- Configure the timer so that an update event triggers a DMA transfer.
- In the DMA controller, set the CPAR address to the TIMx->CCRy register location.
- PSIZE will be 16 or 32 bits.
- MSIZE can be 8 bits.
- Set the OCyPE bit so that the CCRy is not updated until the current cycle is complete.
- Result: 24 bytes copied per LED: 16777216 colors.
Observations

- Modulation of a 6MHz signal is not something you could easily do with software.
- An interrupt service routine would add too much overhead in terms of saving and restoring registers.
- Using a DMA channel as a "coprocessor" is a very useful trick.