Direct Memory Access

ECE 362
https://engineering.purdue.edu/ece362/

Reading Assignment

- Textbook, Chapter 19, Direct Memory Access, pp. 469 – 480.

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.
    - Your development board has no USB interface.
      - There are some USB provisions in the STM32F091, but would take much work.
    - Other books are better for understanding USB.

CPU stops… work goes on

- If you configured a timer, and execute a WFI instruction, the CPU will wake up on an interrupt.
- If you set up PWM output (and don’t enable interrupts) and execute a WFI instruction, the timer will continue to produce an output, but the CPU will remain effectively "asleep."
  - The timer works 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 *co-processor* that reads and writes memory locations, and increments its registers.

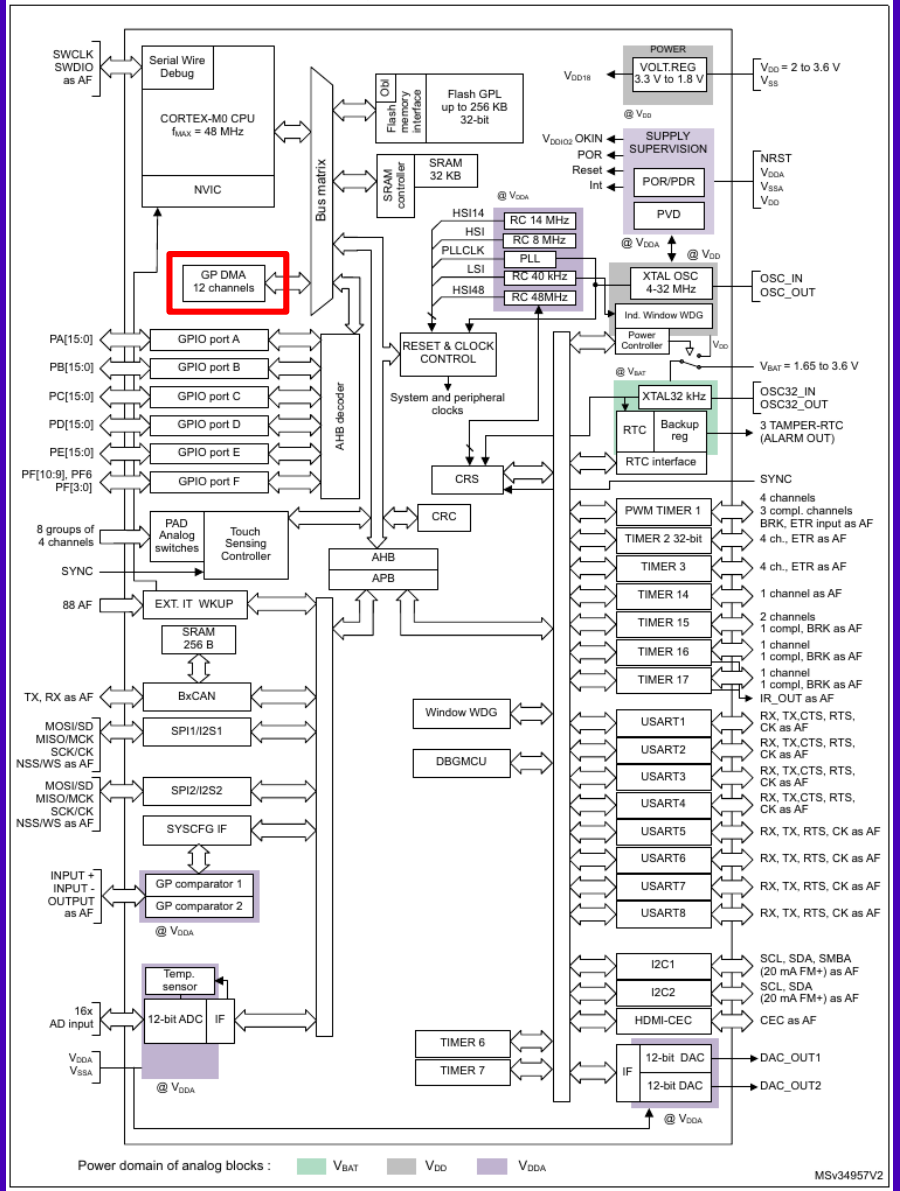
Discussed Early in FRM

- Chapter 7: RCC
- Chapter 8: CRS (Clock Recovery System)
- Chapter 9: GPIO
- Chapter 10: SYSCFG
- Chapter 11: DMA
- Chapter 12: Interrupts
- Chapter 13: ADC

Why have we put off talking about this so long?

Knowledge is Valuable

- It is difficult to understand how DMA works
- Not many people understand
- Therefore, this understanding is valuable



DMA Controller

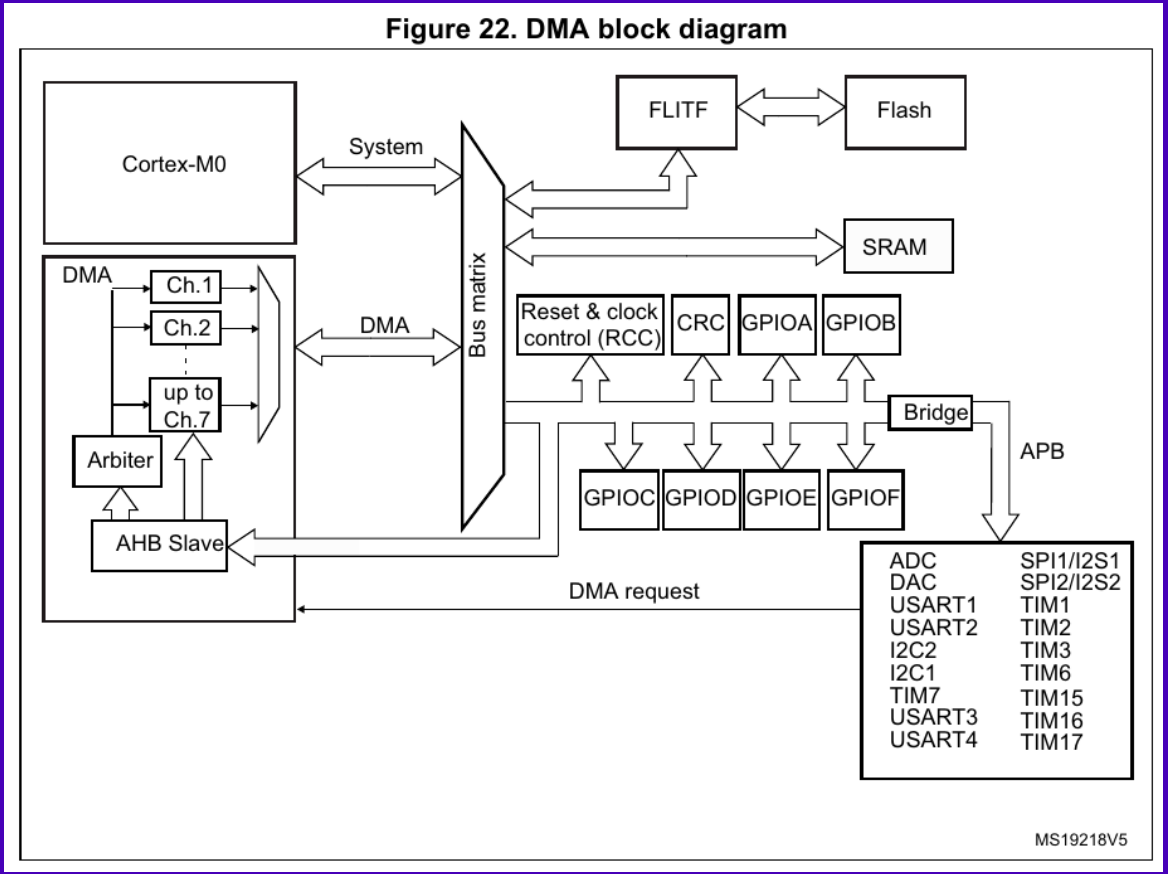
- Multiple independent "channels".
  - Really this means it's just sets of src/dest registers.
  - Priorities
  - Different transfer sizes (byte,halfword,word)
  - Circular buffers
  - Up to 65535 data per request.

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:
  - CCR: Channel Configuration Register
  - CMAR: Channel Memory Address Register
  - CPAR: Channel Peripheral Address Register
  - CNDTR: Channel Number of Data 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	Res.	Res.	Res.	Res.	TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5	TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	DMA_IFCR	Res.	Res.	Res.	Res.	CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	CHTIF6	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5	CTEIF4	CHTIF4	CTCIF4	CGIF4	CTEIF3	CHTIF3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	DMA_CCR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MEM2MEM	PL [1:0]		MSIZE [1:0]		PSIZE [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	DMA_CNDTR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	NDT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	DMA_CPAR1	PA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	DMA_CMAR1	MA[31:0]																															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

controller registers

per channel 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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
				r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	CTEIF 7	CHTIF 7	CTCIF7	CGIF7	CTEIF6	CHTIF6	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5
				w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTEIF 4	CHTIF 4	CTCIF 4	CGIF4	CTEIF 3	CHTIF 3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

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

0x08	**DMA_CCR1**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MEM2MEM	PL [1:0]		MSIZE [1:0]		PSIZE [1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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?

DMA Example Code

```
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->CNDTR = 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_MSIZE;   // 00: 8 bits
    DMA1_Channel1->CCR &= ~DMA_CCR_PSIZE;   // 00: 8 bits
    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(;;);
}
```

DMA ISR code

```
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

DMA Channel Configuration sequence code example

```
/* 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 tranfer 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) */
```

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

A.7.9 DMA one-shot mode on ADC

```
/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC - DMACFG is kept at 0
       for one shot mode */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA tranfer to be performs
       on DMA 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 = NUMBER_OF_ADC_CHANNEL; /* (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) */
```

- 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

```
/* (1) Enable the peripheral clock on DMA */
/* (2) Enable DMA transfer on ADC and circular mode */
/* (3) Configure the peripheral data register address */
/* (4) Configure the memory address */
/* (5) Configure the number of DMA tranfer to be performs
       on DMA channel 1 */
/* (6) Configure increment, size, interrupts and circular mode */
/* (7) Enable DMA Channel 1 */
RCC->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
ADC1->CFGR1 |= ADC_CFGR1_DMAEN | ADC_CFGR1_DMACFG; /* (2) */
DMA1_Channel1->CPAR = (uint32_t) (&(ADC1->DR)); /* (3) */
DMA1_Channel1->CMAR = (uint32_t)(ADC_array); /* (4) */
DMA1_Channel1->CNDTR = NUMBER_OF_ADC_CHANNEL; /* (5) */
DMA1_Channel1->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_0 | DMA_CCR_PSIZE_0
                    | DMA_CCR_TEIE | DMA_CCR_CIRC; /* (6) */
```

- 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 is TSEL?
- Which DMA channel?
  - Or channels?

DAC_CR TSEL field

- Note: We didn't have a dual DAC on STM32F051

Bits 5:3 **TSEL1[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).

14.8.1 DAC control register (DAC_CR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	DMAUDRIE2	DMAEN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]			TEN2	BOFF2	EN2
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	DMAUDRIE1	DMAEN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

A.8.7 DMA on DAC with timer7

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);
```

- Do you see the DMA setup in step (2)? No?
- Neither do I.

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->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 tranfer 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->AHBENR |= RCC_AHBENR_DMA1EN; /* (1) */
DMA1_Channel3->CPAR = (uint32_t) (&(DAC->DHR12RD)); /* (2) */
DMA1_Channel3->CMAR = (uint32_t)signal_data; /* (3) */
DMA1_Channel3->CNDTR = SIGNAL_ARRAY_SIZE; /* (4) */
DMA1_Channel3->CCR |= DMA_CCR_MINC | DMA_CCR_MSIZE_1 | DMA_CCR_PSIZE_1
                    | DMA_CCR_TEIE | DMA_CCR_CIRC; /* (5) */
DMA1_Channel3->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.

DMA Request Mapping

Family Reference Manual
Page 202

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 30. Summary of the DMA requests for each channel on STM32F03x, STM32F04x and STM32F030x8STM32F05x devices

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC	ADC(1)	ADC(2)	-	-	-
SPI	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX
USART	-	USART1_TX(1)	USART1_RX(1)	USART1_TX(2) USART2_TX	USART1_RX(2) USART2_RX
I2C	-	I2C1_TX	I2C1_RX	I2C2_TX	I2C2_RX
TIM1	-	TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_CH3 TIM1_UP
TIM2	TIM2_CH3	TIM2_UP	TIM2_CH2	TIM2_CH4	TIM2_CH1
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	TIM3_CH1 TIM3_TRIG	-
TIM6 / DAC	-	-	TIM6_UP DAC_Channel1	-	-
TIM15	-	-	-	-	TIM15_CH1 TIM15_UP TIM15_TRIG TIM15_COM
TIM16	-	-	TIM16_CH1(1) TIM16_UP(1)	TIM16_CH1(2) TIM16_UP(2)	-
TIM17	TIM17_CH1(1) TIM17_UP(1)	TIM17_CH1(2) TIM17_UP(2)	-	-	-

1. DMA request mapped on this DMA channel only if the corresponding remapping bit is cleared in the SYSCFG_CFGR1 register. For more details, please refer to *Section 10.1.1: SYSCFG configuration register 1 (SYSCFG_CFGR1) on page 173*.
2. DMA request mapped on this DMA channel only if the corresponding remapping bit is set in the SYSCFG_CFGR1 register. For more details, please refer to *Section 10.1.1: SYSCFG configuration register 1 (SYSCFG_CFGR1) 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 if you want to use all four at the same time with DMA?
  - You don't.
- What are all these TIM things?

Request Map for STM32F07

Table 31. Summary of the DMA requests for each channel on STM32F07x devices (continued)

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
I2C	Reserved	I2C1_TX(1)	I2C1_RX(1)	I2C2_TX	I2C2_RX	I2C1_TX(2)	I2C1_RX(2)
TIM1	Reserved	TIM1_CH1(1)	TIM1_CH2(1)	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_CH3(1) TIM1_UP	TIM1_CH1(2) TIM1_CH2(2) TIM1_CH3(2)	Reserved
TIM2	TIM2_CH3	TIM2_UP	TIM2_CH2(1)	TIM2_CH4(1)	TIM2_CH1	Reserved	TIM2_CH2(2) TIM2_CH4(2)
TIM3	Reserved	TIM3_CH3	TIM3_CH4 TIM3_UP	TIM3_CH1(1) TIM3_TRIG(1)	Reserved	TIM3_CH1(2) TIM3_TRIG(2)	Reserved
TIM6 / DAC	Reserved	Reserved	TIM6_UP DAC_Channel1	Reserved	Reserved	Reserved	Reserved
TIM7 / DAC	Reserved	Reserved	Reserved	TIM7_UP DAC_Channel2	Reserved	Reserved	Reserved
TIM15	Reserved	Reserved	Reserved	Reserved	TIM15_CH1 TIM15_UP TIM15_TRIG TIM15_COM	Reserved	Reserved
TIM16	Reserved	Reserved	TIM16_CH1(1) TIM16_UP(1)	TIM16_CH1(2) TIM16_UP(2)	Reserved	TIM16_CH1(3) TIM16_UP(3)	Reserved
TIM17	TIM17_CH1(1) TIM17_UP(1)	TIM17_CH1(2) TIM17_UP(2)	Reserved	Reserved	Reserved	Reserved	TIM17_CH1(3) TIM17_UP(3)

Request Map on STM32F09

- It’s complicated.
- Read about the DMA_CSELR (Channel Select Register) in your textbook or FRM.
  - We didn’t have that on an STM32F05x.
  - It does exist on an STM32F09x.
  - CSELR has one four-bit field per DMA channel to select a trigger source
  - Note that the term for it with CMSIS is RMPCR (Remap Control Register) and the symbols used for it try to hide some of the complexity that follows.
    - e.g.: `DMA1->RMPCR |= DMA_RMPCR1_CH2_ADC;` // map ADC to DMA1 ch2

Default Request Map on STM32F09

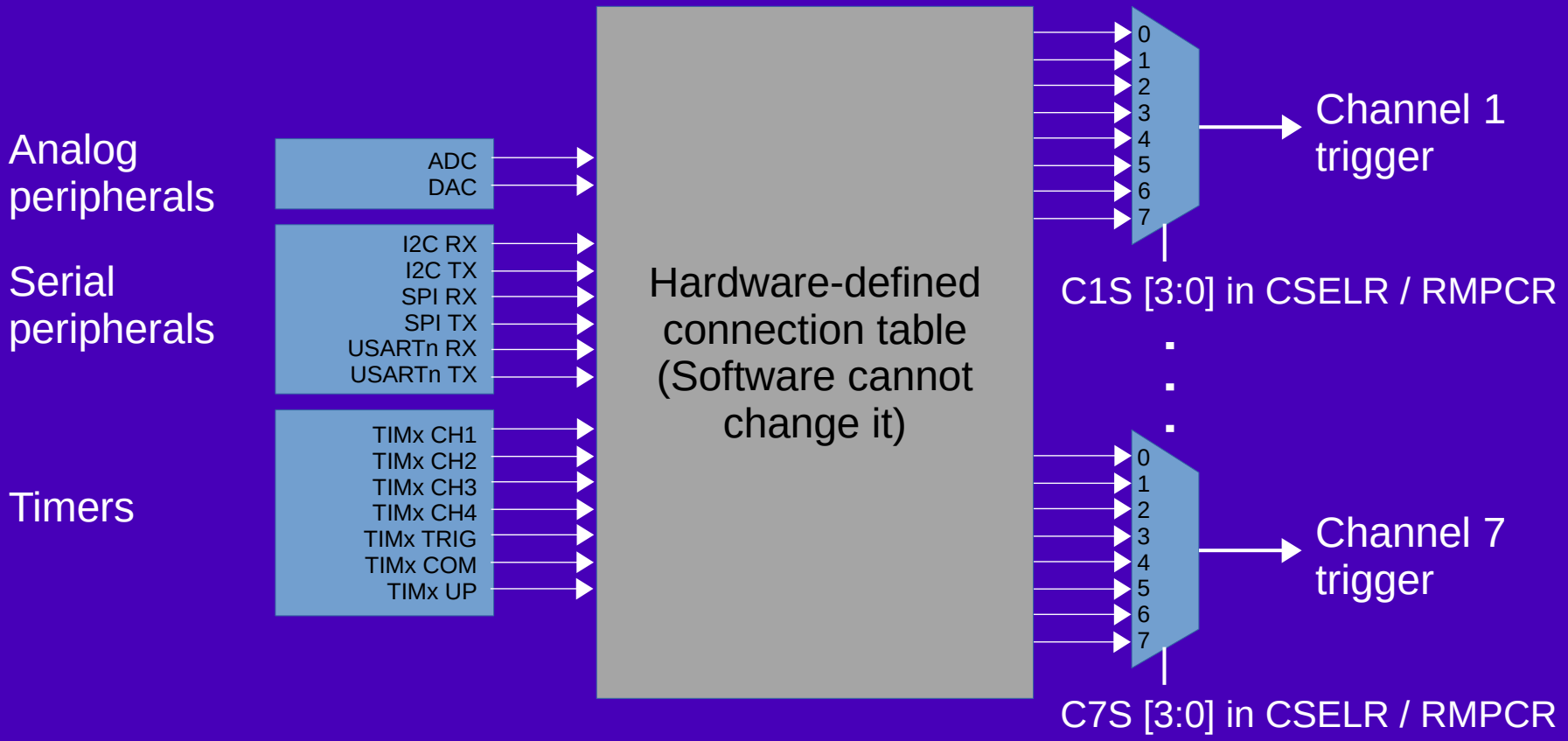
Table 32. Summary of the DMA1 requests for each channel on STM32F09x devices

CxS [3:0]	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
0000	TIM2_CH3	TIM2_UP TIM3_CH3	TIM3_CH4 TIM3_UP	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	-	-
	-				TIM2_CH1	-	-
	-	-	-		TIM15_CH1 TIM15_UP TIM15_TRIG TIM15_COM	-	-
	ADC	-	TIM6_UP DAC_ Channel1	TIM7_UP DAC_ Channel2		-	-
	-	USART1_TX	USART1_RX	USART2_TX	USART2_ RX	USART3_ RX	USART3_TX
	-	-	-	-	-	USART4_ RX	USART4_TX
	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX	-	-
	-	I2C1_TX	I2C1_RX	I2C2_TX	I2C2_RX	-	-
	-	TIM1_CH1	TIM1_CH2	-	TIM1_CH3	-	-
	-	-	TIM2_CH2	TIM2_CH4	-	-	-
	TIM17_CH1 TIM17_UP	-	TIM16_CH1 TIM16_UP	TIM3_CH1 TIM3_TRIG	-	-	-

...continued

0001	ADC	ADC	TIM6_UP DAC_ Channel1	TIM7_UP DAC_ Channel2	-	-	-
0010	-	I2C1_TX	I2C1_RX	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
0011	-	SPI1_RX	SPI1_TX	SPI2_RX	SPI2_TX	SPI2_RX	SPI2_TX
0100	-	TIM1_CH1	TIM1_CH2	-	TIM1_CH3	TIM1_CH1 TIM1_CH2 TIM1_CH3	-
0101	-	-	TIM2_CH2	TIM2_CH4	-	-	TIM2_CH2 TIM2_CH4
0110	-	-	-	TIM3_CH1 TIM3_TRIG	-	TIM3_CH1 TIM3_TRIG	-
0111	TIM17_CH1 TIM17_UP	TIM17_CH1 TIM17_UP	TIM16_CH1 TIM16_UP	TIM16_CH1 TIM16_UP	-	TIM16_CH1 TIM16_UP	TIM17_CH1 TIM17_UP
1000	USART1_ RX	USART1_TX	USART1_RX	USART1_TX	USART1_ RX	USART1_ RX	USART1_TX
1001	USART2_ RX	USART2_TX	USART2_ RX	USART2_TX	USART2_ RX	USART2_ RX	USART2_TX

Illustration From Textbook



Why?

- You might ask, "Why would anyone create a microcontroller like this?"
  - You would not be alone.
  - There will be more things to make you ask "Why?" in the future as we look at other peripherals.
- Some microcontrollers have more flexible "fully-mappable" DMA channels where any peripheral can be associated with any DMA channel. (and similar things for I/O pins and alternate functions)
  - One reason STM32 does not is because it takes a lot of transistors to create a "crossbar switch" or something else able to flexibly route each trigger.

DMA2 on STM32F091

- The STM32F091 has a second DMA controller
  - It has a similar, but entirely separate request map, and CSELR / RMPCR.
  - See Table 33 in the FRM
  - Still not as good as having fully-mappable DMA channels, but at least it give more options

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: TRG0: 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.

Two Ways to use DMA with DAC

- Trigger the DAC and DMA (ch 3) with Timer 6
  - Set the UDE (update DMA enable) bit in TIM6_DIER
  - Set TIM6_MMS to 010 select the update event as trigger output (TRGO)
  - Set TSEL1 to 000 in DAC_CR (the default) to trigger on TIM6 TRGO
- Trigger the DAC with a timer (TIM2,3,6,7,15)
  - Set TSEL1 to value for TIM2,3,6,7,15
  - The DAC triggers DMA (ch 3) to transfer next value
  - Do this if you need to use Timer 6 for something else

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 (<10kHz), you’re probably better off doing so with an interrupt handler.

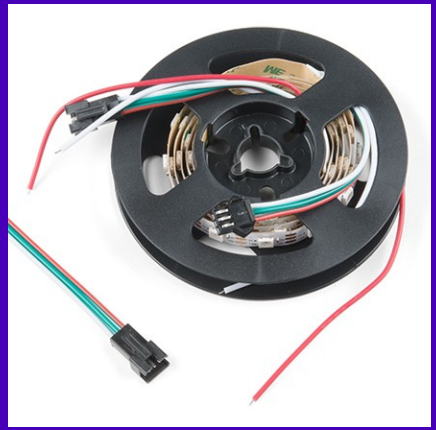
Application: Audio Generation

- High-quality audio generation is often too complicated to process one sample at a time driven by an interrupt. (too much overhead)
- Use a circular buffer that continually transfers at a high rate.
  - On "half transfer," invoke interrupt to refill the first half of the buffer.
  - On "transfer complete," invoke interrupt to refill the second half of the buffer.
  - For a buffer size of N samples, an interrupt happens only every N/2 samples.
- Students have made audio players that read a file from an SDcard (using DMA) and write half the buffer to the DAC.
  - Reaching the "half transfer" mark on the DMA transfer to the DAC causes the next chunk of music to be fetched into the first half of the buffer
  - Reaching the "transfer complete" mark on the DMA transfer to the DAC causes the next chunk of music to be fetched into the second half of the buffer

Application: LED strip

- 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.

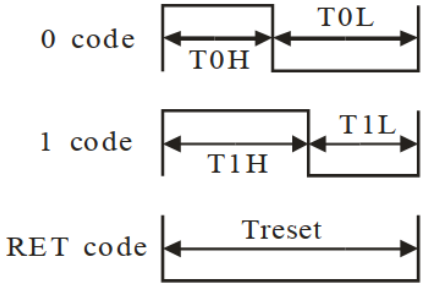


Unique Interface

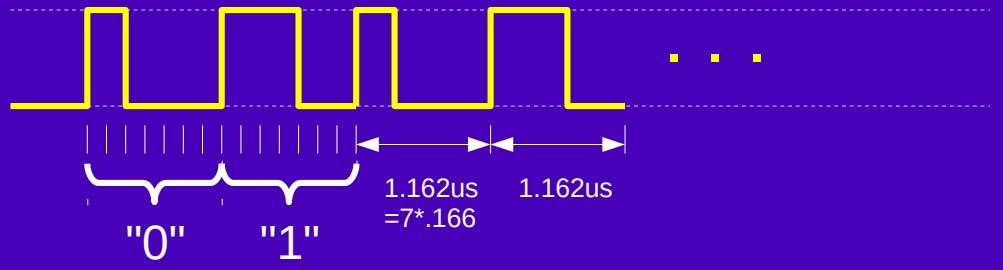
Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.333us	±150ns
T1H	1 code ,high voltage time	0.667us	±150ns
T0L	0 code , low voltage time	0.833us	±150ns
T1L	1 code ,low voltage time	0.500us	±150ns
RES	low voltage time	Above 50μs	

Sequence chart:



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



$.166us^{-1} = 6MHz$

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.
- CNDTR will be (24 bits/LED) * (# LEDs in chain)
- Set the OCyPE bit so that the CCRy is not updated until the current cycle is complete.
- Result: 24 bytes copied per LED:
  - to produce 24 logical 1/0 bits per LED for 16,777,216 different 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.

Interesting Stories

- Getting LED strips to work with PWM/DMA was an interesting trick.
  - Teams of students have done this for mini-projects.
  - One team was going to do this for a matrix of 512 24-bit RGB LEDs = 12,288 bytes.
    - The STM32F051 had only 8K of RAM.
    - They ended up writing a *timing loop in assembly language*.
    - I admit I didn't think they'd be able to do it.