DMA – ØVELSER OG JOURNAL

E4ISD2

Janus Bo Andersen (JA67494)

Februar 2020

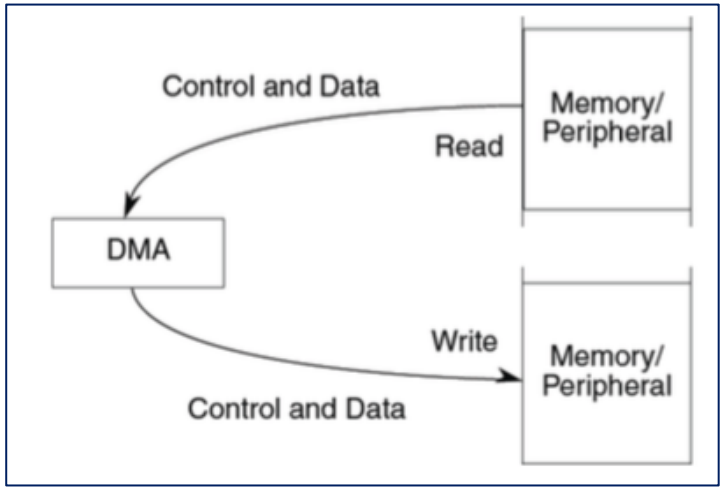
DMA – WHAT'S THE DEAL?
MEMORY-KOPIMASKINE

DMA: Hardware-styret kopiering af data i hukommelse med lav processorinteraktion:

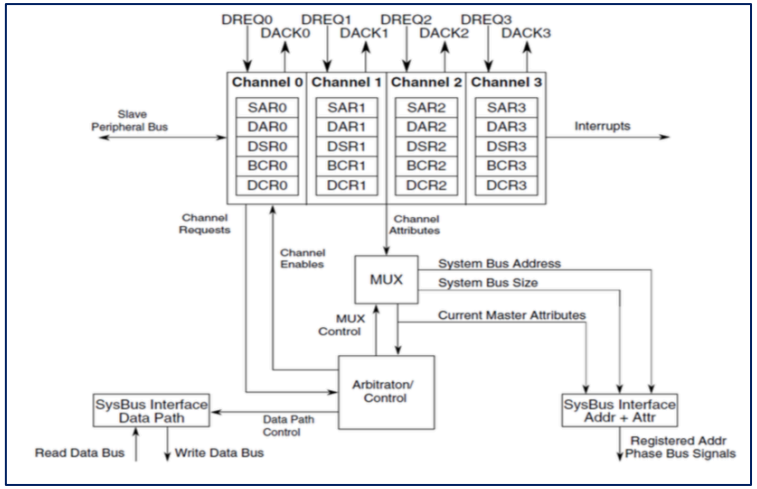
- **Figur 1:** DMA er en peripheral, der kan foretage kopieringsarbejde 'i baggrunden' mellem forskellige memoryområder
- **Figur 2**: DMA arbejder uafhængigt af processor, men tager kontrol med bussen vha. arbitration/control (crossbar switch).
  - Har 4 kanaler, der hver kan kopiere data fra/til angivne registre (SAR til DAR).
  - Holder styr på hvor meget, der stadig mangler at blive kopieret (BCR).
  - Kan køre i enten cycle-stealing mode (et item kopieres per event) eller burst mode (alt på en gang).
  - Startes enten via software eller via event fra anden peripheral (indstilles i DCR)
    - Hvis anden periph.: Trigger source vælges vha. DMAMUX (ej vist).

Fordele ved DMA:

- Mere responsivt / performant system
- Frigiver CPU-tid til noget andet: Sparer softwareoperationer, og CPU forbliver tilgængelig til andre operationer
- Fleksibelt. Kan kopiere mellem memoryområder, der fx repræsenterer forskellige peripherals (DAC, osv.)
  - Fx som signalgenerator er det en væsentlig mere stabil / deterministisk metode til at sikre stabil periodetid, m.v.
- Kan afvikles periodisk baseret fx på event fra en timer peripheral, så CPU'en kan forblive i low-power mode -> sparer energi, osv.



Figur 1. Memory-kopimaskinen



Figur 2. DMA arbejder uafhængigt, har 4 kanaler

ØV1: PROFILING
OPG. OG OPSÆTNING

```
15  #define ARR_SIZE (1)                                    // Size of memory copy
16  #define DBG_PIN (1)                                     // Use PTB1 for measuring time
17  #define MASK(x) (1UL << x)                              // Masking function
18  #define TOGGLE_DEBUG_PIN() FPTB->PTOR = 1UL << DBG_PIN  // Macro to Fast toggle PTB1
```

Øvelse 1 uge 5: Profiling på "memcpy"

Formål:

- Afgøre om **SW**- eller **DMA**-kopiering er mest fordelagtigt, og hvordan / hvorfor / hvornår.

Metode. Sammenligne...

- Kopiering vha. **DMA** vs. **software**.
- Hastigheder måles med Analog Discovery

Opsætning:

- Kopiér varierende antal 32-bit words.
- Hukommelsesområder og DMA init..
- En FAST GPIO pin til debug-signal.
- Optimeret kode (-O3).

Software-kopiering

```
/*
 * @brief   software data copy, toggles PTB1 high while copying
 */
void test_sw_copy(void) {
    uint32_t *ps, *pd;

    uint16_t i; // for counting up array index

    // Initialize array with some data
    for (i = 0; i < ARR_SIZE; i++) {
        s[i] = i;   //source initialized to some number
        d[i] = 0;   //destination initialized to zero
    }

    TOGGLE_DEBUG_PIN();     //toggle high while copying

    ps = s; // set pointers equal to first elem in arrays
    pd = d;

    for (int i = 0; i < ARR_SIZE; i++) {
        *pd++ = *ps++;  // increment ptr, dereference value, copy
    }

    TOGGLE_DEBUG_PIN();     //toggle low when done copying
}
```

DMA-kopiering

```
/*
 * @brief   initialize DMA to copy 32bit->32bit and increment addresses.
 * DMA Mux is not enabled.
 */
void init_dma(void) {

    SIM->SCGC7 |= SIM_SCGC7_DMA_MASK;       // enable clock for DMA
    DMA0->DMA[0].DCR = DMA_DCR_SINC_MASK |  // increment source addr
                       DMA_DCR_SSIZE(0) |   // source data size is 32-bit
                       DMA_DCR_DINC_MASK |  // increment dest addr
                       DMA_DCR_DSIZE(0);    // dest data size is 32-bit
}
```

```
/*
 * @brief   Copy data words via DMA channel 0, 32-bit words
 * @param   source  source address
 * @param   dest    destination address
 * @param   count   number of words to copy
 */
void copy_32bit(uint32_t * source, uint32_t * dest, uint32_t count) {

    // Set source and destination memory addresses in SAR and DAR
    DMA0->DMA[0].SAR = DMA_SAR_SAR( (uint32_t) source );    //cast source addr ptr to 32 bit num
    DMA0->DMA[0].DAR = DMA_DAR_DAR( (uint32_t) dest );      //cast dest addr ptr to 32 bit num

    // Init byte count register (BCR), 4 bytes per word to copy (32-bit)
    DMA0->DMA[0].DSR_BCR = DMA_DSR_BCR_BCR(count * 4);

    // Clear DONE flag
    DMA0->DMA[0].DSR_BCR &= ~DMA_DSR_BCR_DONE_MASK;         // Must be zero in order to start

    TOGGLE_DEBUG_PIN(); //Set DBG PIN high during transfer

    // Software start of transfer
    DMA0->DMA[0].DCR |= DMA_DCR_START_MASK;                 // Write 1 to START to begin

    //Busy polling until DONE goes high
    while ( !(DMA0->DMA[0].DSR_BCR & DMA_DSR_BCR_DONE_MASK ) ) {
        // Zzzz!
    }

    TOGGLE_DEBUG_PIN(); //Set DBG PIN low as transfer is done
}
```

```
/*
 * @brief   wrapper function to test the DMA copy
 */
void test_dma_copy(void) {

    uint16_t i; // for counting up array index

    // Initialize array with some data
    for (i = 0; i < ARR_SIZE; i++) {
        s[i] = i;   //source initialized to some number
        d[i] = 0;   //destination initialized to zero
    }

    // Begin to copy from s to d
    copy_32bit(s, d, ARR_SIZE);
}
```

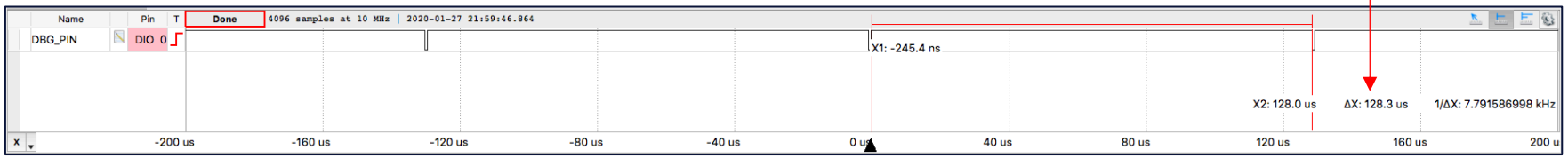
ØV1: PROFILING
KOPIERING MED SOFTWARE

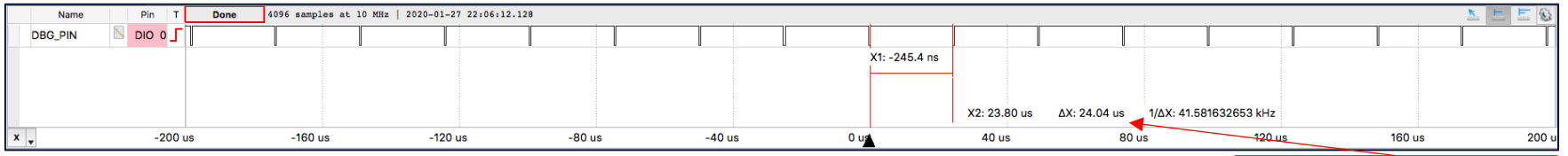
Software-kopiering:

- Kopierer 1, 256 og 512 elementer.
- Optimering er vigtigt! (-O3)
  - Faktor 5 forbedring!
- SW-kopiering skalerer lineært:
  - ca. 90-94 ns per 32-bit element, der kopieres,
  - dvs. ca. 5 clock cycles per 32-bit element.
- Ikke høj overhead/setup cost i cycles for SW-kopiering.

- Mulighed for yderligere optimering
  - Smallere typer: med fx uint8_t eller uint16_t kan processor håndtere hhv. 4 og 2 tal i én registeroperation.

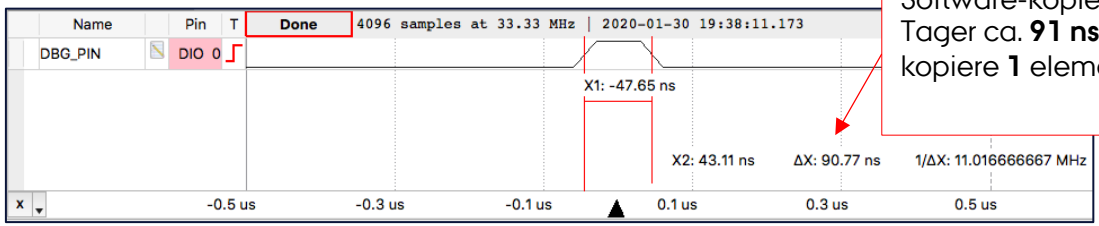
Software-kopiering uden optimering:
Tager ca. **128 µs** at kopiere **256** elementer...



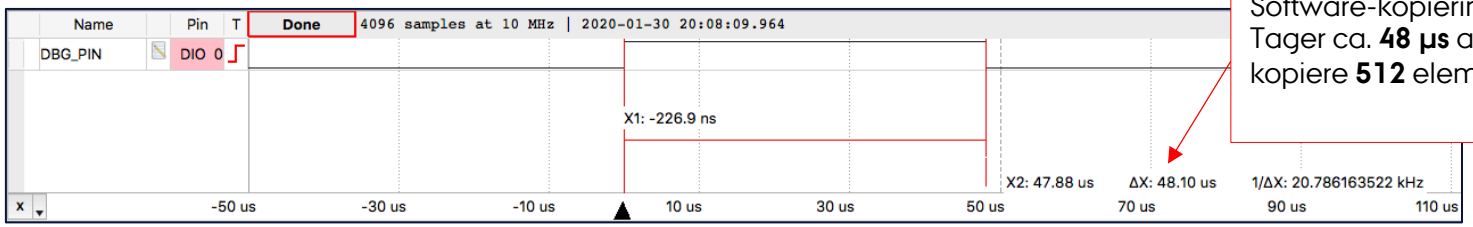


Software-kopiering med O3 optimering:
Tager ca. **24 µs** at kopiere **256** elementer.

(Matcher ikke bog s. 258)



Software-kopiering (O3):
Tager ca. **91 ns** at kopiere **1** element.



Software-kopiering (O3):
Tager ca. **48 µs** at kopiere **512** elementer.

ØV1: PROFILING
KOPIERING MED DMA

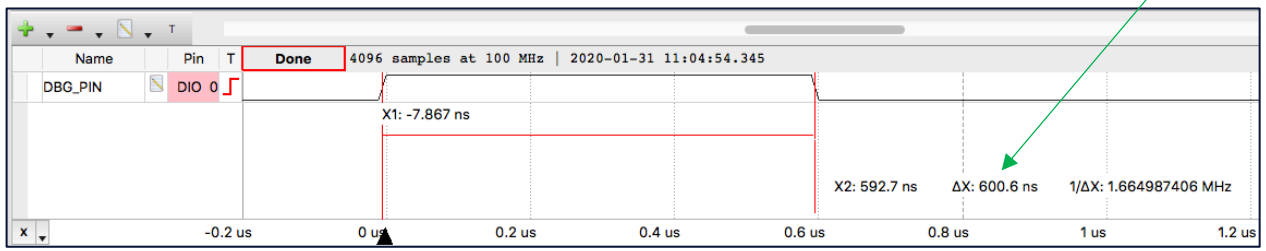
DMA-kopiering

- 256 og 512 elementer tager hhv. 11 µs og 22 µs.
  - Dvs. ca. 43 ns per 32-bit element
- 1 element tager ca. 600 ns:
  - Der er altså væsentlig setup/startup tid (overhead).
    - Ca. 560 ns setup cost.

Konklusion:

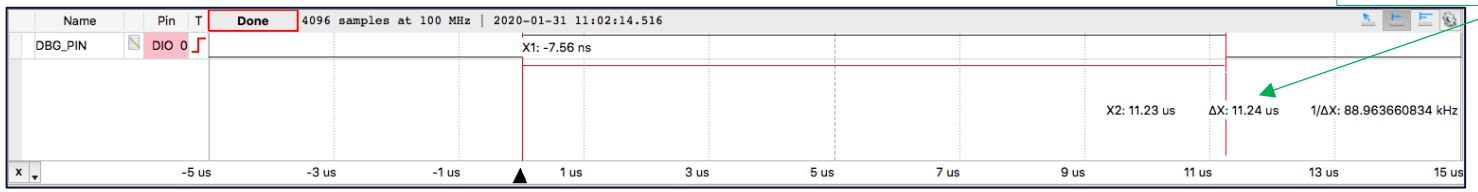
- => DMA er dobbelt så hurtigt per element som software-kopiering for større datamængder:
- ≥ 10 elementer: Brug DMA
- < 10 elementer: Brug SW

DMA-kopiering (O3): Tager ca. **600 ns** at kopiere **1** element.

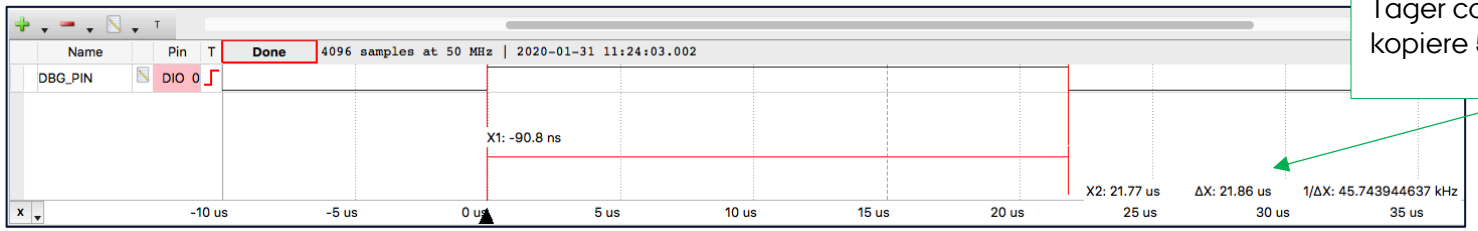


DMA-kopiering (O3): Tager ca. **11 µs** at kopiere **256** elementer.

Matcher bog s. 259!



DMA-kopiering (O3): Tager ca. **22 µs** at kopiere **512** elementer.



ØV1: VERIFIKATION AF KOPIERING OK! (APPENDIKS SLIDE)

Adresser på source og destination (32-bit registre)

Destination hh. *før* og *efter* kopiering

- SW: 2 registre per omgang i loopet.
- DMA: Alt på én gang, selvfølgelig

Kopiering udføres optimeret via writeback ops. (!)

Software-kopiering

Name	Type	Value
(x)= i	uint16_t	<optimized out>
ps	uint32_t *	0x1ffff888 <s>
pd	uint32_t *	0x1ffff088 <d>

0x1ffff088 <Traditional>

```
0x1FFFF088  0000000000000000  ........
0x1FFFF090  0000000000000000  ........
0x1FFFF098  0000000000000000  ........
0x1FFFF0A0  0000000000000000  ........
0x1FFFF0A8  0000000000000000  ........
0x1FFFF0B0  0000000000000000  ........
0x1FFFF0B8  0000000000000000  ........
0x1FFFF0C0  0000000000000000  ........
0x1FFFF0C8  0000000000000000  ........
0x1FFFF0D0  0000000000000000  ........
0x1FFFF0D8  0000000000000000  ........
0x1FFFF0E0  0000000000000000  ........
0x1FFFF0E8  0000000000000000  ........
```

0x1ffff088 <Traditional>

```
0x1FFFF088  00000000  ....
0x1FFFF08C  00000001  ....
0x1FFFF090  00000002  ....
0x1FFFF094  00000003  ....
0x1FFFF098  00000004  ....
0x1FFFF09C  00000005  ....
0x1FFFF0A0  00000006  ....
0x1FFFF0A4  00000007  ....
```

```
 32             for (int i = 0; i < ARR_SIZE; i++) {
00000518:    movs    r3, r1
0000051a:    movs    r1, #128        ; 0x80
0000051c:    movs    r2, r0
0000051e:    lsls    r1, r1, #4
00000520:    adds    r4, r3, r1
 33                *pd++ = *ps++;  // increment ptr, dereference
00000522:    ldmia   r3!, {r0, r1}
00000524:    stmia   r2!, {r0, r1}
00000526:    cmp     r4, r3
00000528:    bne.n   0x522 <test_sw_copy+46>
```

DMA-kopiering

Name	Type	Value
(x)= count	uint32_t	512
dest	uint32_t *	0x1ffff088 <d>
source	uint32_t *	0x1ffff888 <s>

0x1ffff088 <Traditional>

```
0x1FFFF088  00000000  ....
0x1FFFF08C  00000000  ....
0x1FFFF090  00000000  ....
0x1FFFF094  00000000  ....
0x1FFFF098  00000000  ....
0x1FFFF09C  00000000  ....
0x1FFFF0A0  00000000  ....
0x1FFFF0A4  00000000  ....
0x1FFFF0A8  00000000  ....
0x1FFFF0AC  00000000  ....
0x1FFFF0B0  00000000  ....
0x1FFFF0B4  00000000  ....
0x1FFFF0B8  00000000  ....
0x1FFFF0BC  00000000  ....
```

0x1ffff088 <Traditional>

```
0x1FFFF088  00000000  ....
0x1FFFF08C  00000001  ....
0x1FFFF090  00000002  ....
0x1FFFF094  00000003  ....
0x1FFFF098  00000004  ....
0x1FFFF09C  00000005  ....
0x1FFFF0A0  00000006  ....
0x1FFFF0A4  00000007  ....
0x1FFFF0A8  00000008  ....
0x1FFFF0AC  00000009  ....
0x1FFFF0B0  0000000A  ....
0x1FFFF0B4  0000000B  ....
0x1FFFF0B8  0000000C  ....
0x1FFFF0BC  0000000D  ....
0x1FFFF0C0  0000000E  ....
0x1FFFF0C4  0000000F  ....
0x1FFFF0C8  00000010  ....
```

ØV2: DMA ANALOG WAVEFORM GEN.
OPGAVE OG OPSÆTNING

Øvelse 2 uge 5: Analog Waveform Gen vha. **ISR+DAC** vs. **DMA+DAC**

Formål:

- Analoge signaler vha. DAC – bedst drevet af ISR eller DMA?
- Måle forskelle i processorbelastning for de to metoder.
- Bekræfte at outputsignal er korrekt for begge metoder.

Metode:

- TPM+**ISR+DAC**:
  - TPM genererer 10 µs 'tick' og IRQ
  - ISR beregner og outputter trekantsignal via DAC.
- TPM+**DMA+DAC**:
  - TPM genererer 10 µs 'tick' og request via DMAMUX
  - Dvs. HW-trigger DMA til at kopiere fra forberegnet trekant-signal til DAC.
  - DMA i cycle-steal mode (1 transfer per request/tick).

Opsætning:

- GPIO, TPM, DAC, DMA initialiseres.
- Analogt signal fra DAC måles på PTE30.
- Benytter FAST GPIO til debug:
  - Digitalt TPM-ISR debug-signal på PTB1
  - Digitalt DMA-ISR debug-signal på PTB0.

TPM+ISR+DAC

```
20  #define DAC_POS (30)              // PTE30
21  #define DAC_RESOLUTION (4096)     // DAC code up to...
22  #define STEP_SIZE (16)            // Step size for DAC code
23  #define NUM_STEPS (512)           // DAC_RESOLUTION / STEP_SIZE
```

```
57 /*
58  * @brief   ISR for TPM0 to output via DAC
59  */
60 void TPM0_IRQHandler() {
61     static int change = STEP_SIZE;
62     static uint16_t out_data = 0;
63
64     TOGGLE_DEBUG_PIN();
65
66     // Clear the overflow flag by writing 1, man. p. 552
67     TPM0->SC |= TPM_SC_TOF_MASK;
68     NVIC_ClearPendingIRQ(TPM0_IRQn);
69
70     // ISR work here
71
72     // Configure the signal change per round
73     out_data += change;
74     if ( out_data < STEP_SIZE ) {
75         change = STEP_SIZE;
76     } else if (out_data >= DAC_RESOLUTION - STEP_SIZE) {
77         change = -STEP_SIZE;
78     }
79
80     // Output via the DAC, high bits first, then low bits
81     DAC0->DAT[0].DATH = DAC_DATH_DATA1(out_data >> 8);
82     DAC0->DAT[0].DATL = DAC_DATL_DATA0(out_data);
83
84     TOGGLE_DEBUG_PIN();
85 }
```

TPM+DMA+DAC

```
46      TPM0->SC = TPM_SC_DMA_MASK | // enable DMA flag, man. p. 552
47                 TPM_SC_PS(1);     // prescale by 2, man. p. 553
```

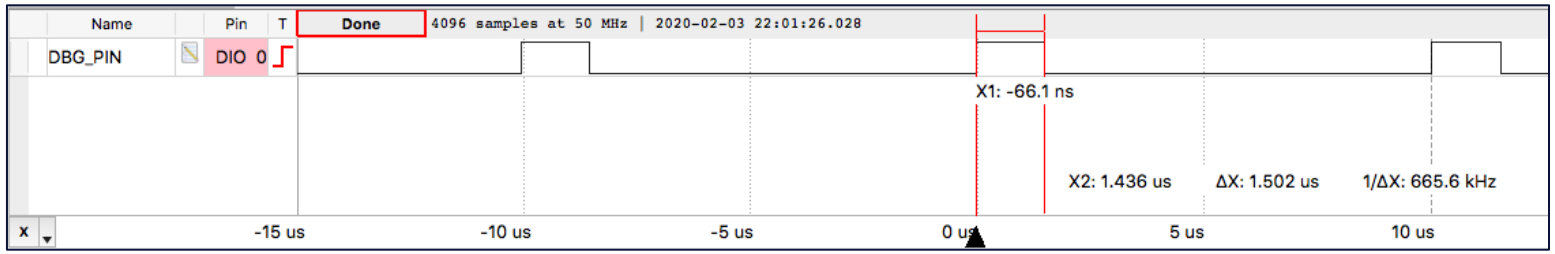
```
14 /*
15  * @brief   initialize DMA to copy 16 bit -> 16 bit and increment addresses.
16  *  DMA Mux is enabled.
17  */
18 void init_dma(uint16_t * source, uint32_t count) {
19
20     //Save in global variables
21     dma_source = source;    //pointer to beginning of triangle array
22     dma_byte_count = count * 2; //2 bytes per data item (16 bit data)
23
24     SIM->SCGC7 |= SIM_SCGC7_DMA_MASK;       // en clk for DMA
25     SIM->SCGC6 |= SIM_SCGC6_DMAMUX_MASK;    // en clk for DMAMUX
26
27     DMAMUX0->CHCFG[0] = 0;  //man. p. 340, disable during config
28
29     DMA0->DMA[0].DCR = DMA_DCR_SINC_MASK |  // increment source addr
30                        DMA_DCR_SSIZE(2) |   // source data size is 16-bit (p. 358)
31                        DMA_DCR_DSIZE(2) |   // dest data size is 16-bit (p. 359)
32                        DMA_DCR_EINT_MASK |  // en interrupt (p. 357)
33                        DMA_DCR_ERQ_MASK |   // en periph. request (p. 357)
34                        DMA_DCR_CS_MASK;     // cycle-steal mode (p. 358)
35
36     //Config DMAMUX peripheral request source (man. p. 340)
37     DMAMUX0->CHCFG[0] = DMAMUX_CHCFG_SOURCE(54);
38
39     //Config NVIC
40     NVIC_SetPriority(DMA0_IRQn, 2);
41     NVIC_ClearPendingIRQ(DMA0_IRQn);
42     NVIC_EnableIRQ(DMA0_IRQn);
43 }
```

```
45 /*
46  * @brief   Sets SAR, DAR, BCR. Clears DONE and enables channel thru DMAMUX.
47  */
48 void start_dma(void) {
49
50     //SAR and DAR config
51     DMA0->DMA[0].SAR = DMA_SAR_SAR((uint32_t) dma_source);  //from triangle_data
52     DMA0->DMA[0].DAR = DMA_DAR_DAR((uint32_t) &(DAC0->DAT[0])); //to DAC data field
53
54     DMA0->DMA[0].DSR_BCR = DMA_DSR_BCR_BCR(dma_byte_count); //bytes to be copied
55
56     //Get ready!
57     DMA0->DMA[0].DSR_BCR &= ~DMA_DSR_BCR_DONE_MASK; //clear DONE (necessary?)
58     DMAMUX0->CHCFG[0] |= DMAMUX_CHCFG_ENBL_MASK;    //ENBL to enable channel (p. 340)
59 }
60
61 /*
62  * @brief   ISR for DMA0, called when copy of complete triangle period performed.
63  */
64 void DMA0_IRQHandler(void) {
65     TOGGLE_DMA_ISR_PIN();
66
67     DMA0->DMA[0].DSR_BCR |= DMA_DSR_BCR_DONE_MASK; //write 1 to clear all status (p.356)
68     start_dma(); //begin again ;)
69
70     TOGGLE_DMA_ISR_PIN();
71 }
```

ØV2: ANALOG WAVEFORM GEN.
ISR+DAC: OUTPUT OG PROCESSORTID

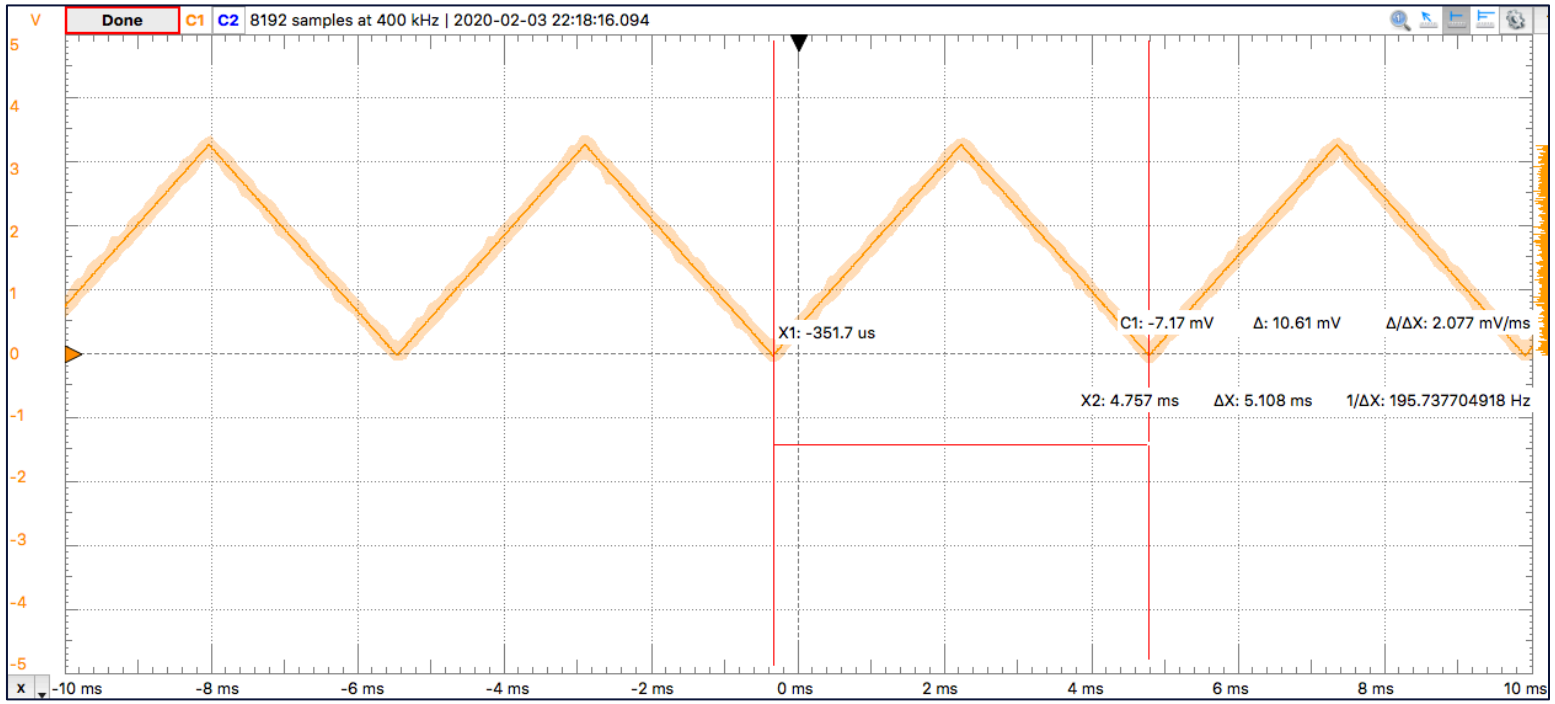
TPM laver 'tick' hver 10 µs

- ISR tager 1.5 µs
- = 15% processortid



Trekantsignal er næsten som ønsket:

- Periodetid, $T = 5.11$ ms.
- Frekvens, $f = 195.7$ Hz.



Target-beregninger:

- Niveauer i DAC-kode.: 4096
- Stepstørrelse: 16
- Trin for fuld periode: 512
- Tid per trin: 10 µs
- Periodetid: 5.12 ms
- Frekvens: 195.3 Hz.

ØV2: ANALOG WAVEFORM GEN.
DMA+DAC: OUTPUT OG PROCESSORTID

ISR kører én gang per trekant-periode

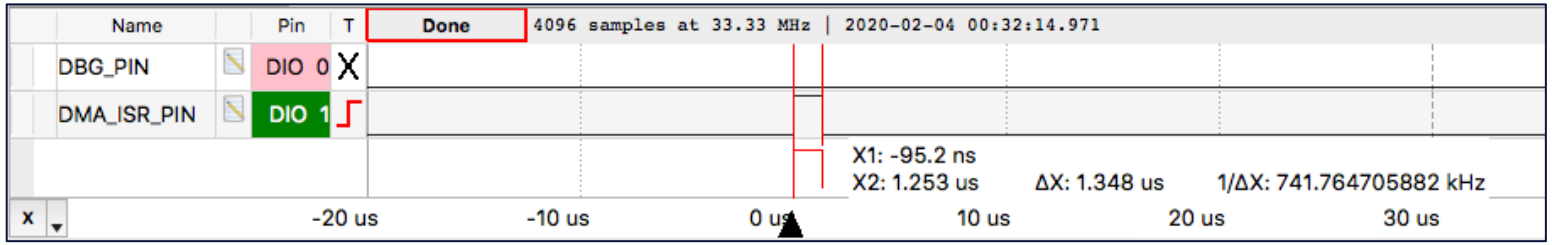
- Dvs. 1 gang per 5.14 ms, når DMA-ISR genstarter DMA.
- DMA ISR tager 1.3 µs
- < 0.01% processortid
- Tager dog tid på bussen (ikke målt).

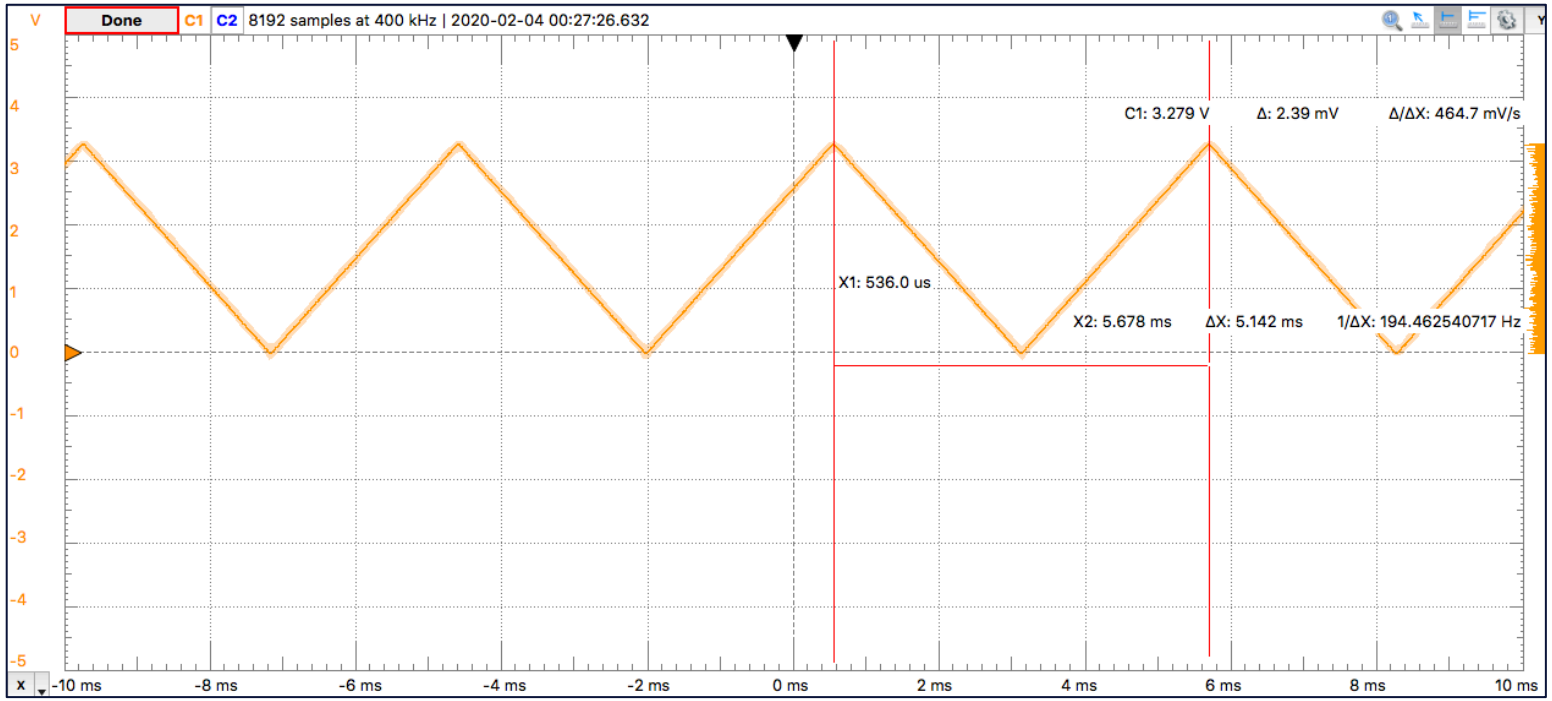
Trekantsignal er stadig som ønsket:

- Periodetid, $T = 5.14$ ms.
- Frekvens, $f = 194.5$ Hz.

Konklusion:

- Begge metoder OK!
- DMA giver *meget* bedre profil for brug af processor-ressourcer.





ØV2: ANALOG WAVEFORM GEN
MEMORY-CHECK (APPENDIKS)

Triangle-data:

- 16-bit = 2 byte per datapunkt.
  - Med 512 steps i "trekanten" skal der kopieres 1024 bytes via DMA.
- Init af værdier kan principielt bekræftes ved oscilloskop (forrige slides).
- Source data inspiceres:
  - 0x0000, 0x0010, … = 0, 16, 32, …

Name	Type	Value
source	uint16_t *	0x1ffff0a0 <triangle_...
count	uint32_t	512
count@entry	uint32_t	512

0x1ffff0a0 <Traditional>

```
0x1FFFF0A0  0000   ..
0x1FFFF0A2  0010   ..
0x1FFFF0A4  0020    .
0x1FFFF0A6  0030   0.
0x1FFFF0A8  0040   @.
0x1FFFF0AA  0050   P.
0x1FFFF0AC  0060   `.
0x1FFFF0AE  0070   p.
0x1FFFF0B0  0080   ..
0x1FFFF0B2  0090   ..
0x1FFFF0B4  00A0   ..
0x1FFFF0B6  00B0   ..
0x1FFFF0B8  00C0   ..
0x1FFFF0BA  00D0   ..
0x1FFFF0BC  00E0   ..
0x1FFFF0BE  00F0   ..
0x1FFFF0C0  0100   ..
```

AARHUS
UNIVERSITET