

TUGAS SISTEM MIKROPROSESOR (Timer-Interrupt driven)

Nama : Gabriel Sebastian Hidayat

NRP : 5103018003

Software bantu : - gcc-avr (compile)
- avr-objcopy (mengubah object file hasil compile → hex file)
- KiCAD (schematic editor)
- simulavr (simulasi program mikrokontroler AVR)
- GTKWave (menampilkan gelombang output hasil simulasi program)

Kendala Proteus

- Berkaitan dengan schematic editor dan simulator pada Proteus, saya mengalami kendala karena Proteus saya tidak bisa dijalankan (OS: Linux Pop-OS 20.04, saya gunakan 'wine' Windows program loader agar bisa menjalankan Proteus), tetapi karena suatu error, sekarang Proteus-nya tidak bisa saya launch dengan wine.
- Alternatif: menggunakan **KiCAD** yang merupakan software open source all platform (namun hanya schematic editor tanpa simulator kompleks) (<https://kicad.org>)
- Untuk simulasi menggunakan 'simulavr' (<https://github.com/Traumflug/simulavr>)

Proses Design

Diketahui:

- $F_{CPU} = 5 \text{ MHz}$
- Delay yang diinginkan:
 - 30 ms
 - 30 ms, namun phase shifted 90°
- Mode operasi timer CTC mode (TOP value: OCR1A)

Menentukan prescaler:

- Berdasarkan tabel berikut:

Prescaler	F_{COUNTER} ($F_{\text{CPU}} / \text{prescaler}$)	T_{COUNTER}	Max delay (OCR1A = 65535)
1	5 MHz	0.2 us	13.107 ms
8	625 KHz	1.6 us	104.856 ms
64	78.125 KHz	12.8 us	838.848 ms
128	39.062 KHz	25.6 us	1.667696 s
256	19.531 KHz	51.2 us	3.355392 s
512	9.765 KHz	102.4 us	6.710784 s
1024	4.882 KHz	204.8 us	13.421568 s

Maka yang paling cocok digunakan adalah dengan prescaler 8, dengan mempertimbangkan prescaler sekecil mungkin agar artifact start-up time yang disebabkan penggunaan prescaler seminimal mungkin.

Menentukan TCNT1 agar menghasilkan delay 30ms saat compare match A terjadi:

$$OCR1A = \frac{30 \times 10^{-3} s}{1.6 \times 10^{-6} s} = 18750$$

Menentukan OCR1B agar compare match B terjadi 15 ms SEBELUM compare match A:

- Bisa langsung dengan membagi nilai OCR1A dengan 2 (30ms / 2 = 15ms)

$$OCR1B = \frac{18750}{2} = 9375$$

Perlu ditekankan, di desain ini compare match B akan terjadi SEBELUM compare match A. Maka, event sequence yang terjadi secara garis besar adalah sebagai berikut:

1. Timer di start
2. pada count ke 9375 (15ms mark), terjadi compare match B, pin PD1 di toggle
3. timer berlanjut menghitung naik
4. pada count 18750 (30ms mark), terjadi compare match A, pin PD0 di toggle, dan timer di clear
5. siklus berulang

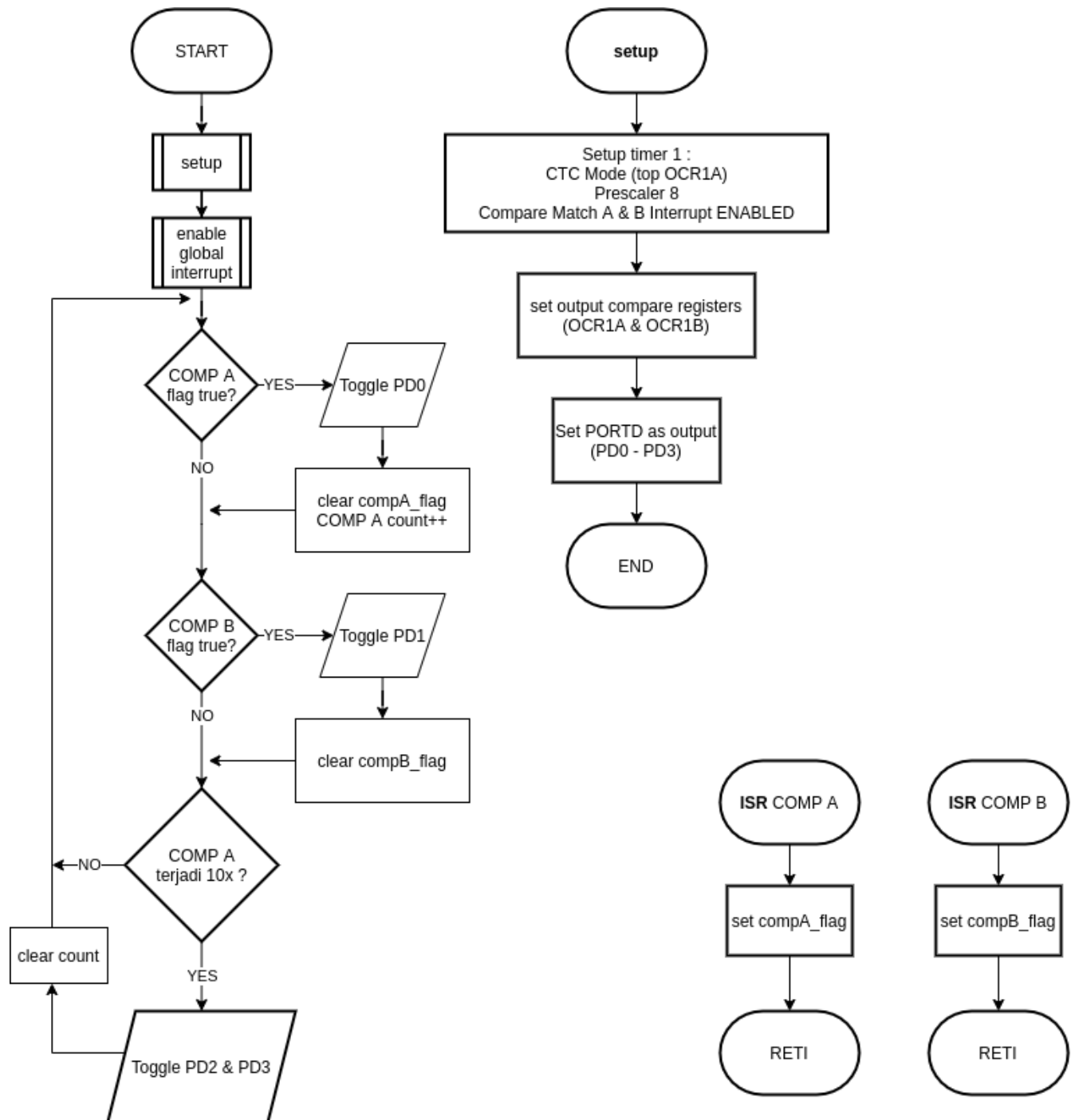
Maka, compare match B pertama terjadi 15ms setelah timer di start, namun compare match B seterusnya memiliki waktu efektif compare match 30 ms ($\frac{1}{2}$ compA time + 15 ms compB time). Dengan demikian, pada PD0 & PD1 akan mengeluarkan dua buah square wave yang memiliki frekuensi dan duty cycle identik, namun berbeda fase sebesar 90°.

Ada sedikit penambahan juga, untuk setiap 10x compare match A, pin PD2 dan PD3 akan di toggle. Idealnya ini akan terjadi setiap 300ms.

Setting akhir timer adalah sebagai berikut:

- Timer 1, mode CTC → **set WGM12 @TCCR1B**
- prescaler 8 → **set CS11 @TCCR1B**
- CTC mode dengan top value OCR1A, interrupt driven → **set OCIE1A @TIMSK**
(**interrupt vector: TIMER1_COMPA_vect**)
- Tambahan interrupt compare match B dengan register OCR1B → **set OCIE1B @TIMSK**
(**interrupt vector: TIMER1_COMPB_vect**)

Flow Charts



Source Code

```
// Tugas Pemrograman 8 (Timer Interrupt)
// Name: Gabriel Sebastian
// NRP : 5103018003

#include <avr/io.h>
#include <avr/interrupt.h>

// SIMULAVR -----
#include <simulavr_info.h>
SIMINFO_DEVICE("atmega8");
SIMINFO_CPUFREQUENCY(5000000);

// MACROS -----
#define F_CPU 5000000UL

// VARIABLE DEFINITION
uint8_t compA_flag = 0;
uint8_t compB_flag = 0;
uint8_t compA_count = 0;
uint8_t count_limit = 10;

// MAIN PROGRAM -----
void setup(){

    // Setup Timer1 as compare match event
    // prescaler 8 (tick = 1.6 us)
    // timer1 set in CTC mode (WGM12 @TCCR1B)
    // interrupts: compare match A & B (OCIE1A & OCIE1B @TIMSK)
    // Compare match A to toggle PD0 every 30ms
    // Compare match B to toggle PD1 15ms BEFORE compare match A occurs
    // resulting in 90 phase shifted waveform in PD1
    // every 10 compare match A, toggle PD2 & PD3 (ideally every 300 ms)
    TCCR1A = 0;
    TCCR1B |= (1 << WGM12) | (1 << CS11);
    TIMSK |= (1 << OCIE1B) | (1 << OCIE1A);
    OCR1A = 18750;
    OCR1B = 9375;

    // setup PORTD as outputs
    DDRD = 0x0F;
    PORTD = 0;
}

void main(void){

    setup();

    // enable global interrupt bit
    sei();
```

```

while(1){
    if(compA_flag){
        PORTD ^= (1 << PD0);
        compA_flag = 0;
        compA_count++;
    }

    if(compB_flag){
        PORTD ^= (1 << PD1);
        compB_flag = 0;
    }

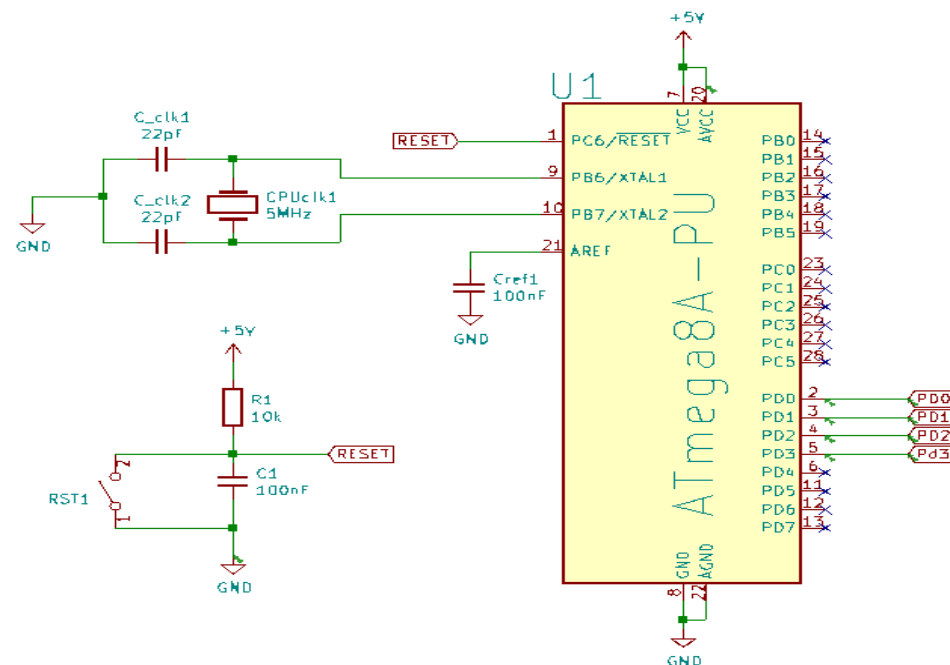
    // if compare match A occurs 10x (ideally 300ms),
    // toggle PD2 & PD3, then reset compare match A count
    if(compA_count >= count_limit){
        PORTD ^= ((1 << PD2) | (1 << PD3));
        compA_count = 0;
    }
}

// INTERRUPT ROUTINE -----
ISR(TIMER1_COMPA_vect){
    compA_flag = 1;
}

ISR(TIMER1_COMPB_vect){
    compB_flag = 1;
}

```

Skematik Rangkaian



Proses Simulasi

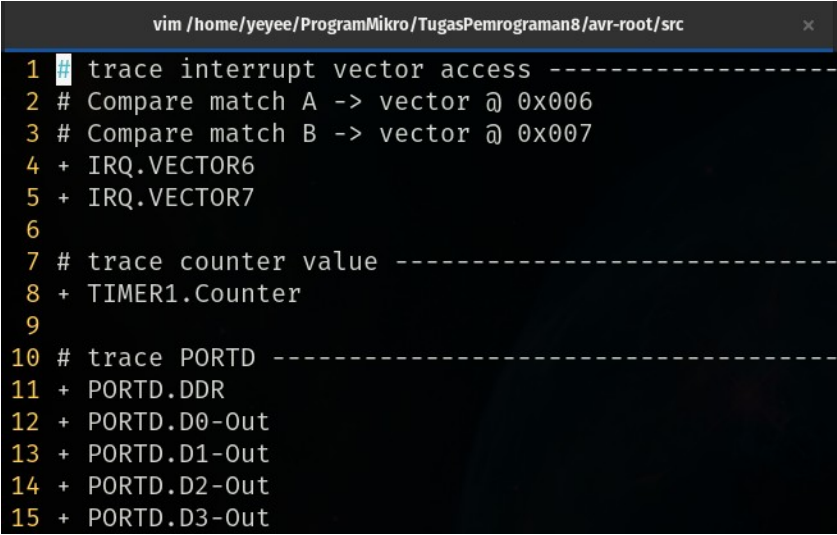
- Pertama, harus menambahkan 3 baris perintah ini pada file **main.c** agar bisa mensimulasikan dengan **simulavr** :

```
// DEBUG -----  
#include <simulavr_info.h>  
SIMINFO_DEVICE("atmega8");  
SIMINFO_CPUFREQUENCY(5000000);
```

- Dokumentasi & manual simulavr di github sangat minim sehingga maksud dari baris perintah ini kurang spesifik, namun secara garis besar, asumsinya adalah kemungkinan besar 3 baris perintah ini untuk mengintegrasikan simulavr ke dalam program utama (**main.c**) agar bisa memantau (tracing) perubahan-perubahan nilai register selama program berjalan dengan cara mengikutsertakan header file **simulavr_info.h**.

- Selanjutnya program bisa di compile dengan tambahan 3 baris perintah tersebut untuk menghasilkan object file dengan format **.elf** (*Executable Linkable Format*). (dengan beberapa setting option compiler yang rumit)

- Selanjutnya membuat daftar sinyal / register yang ingin dipantau simulavr selama proses simulasi, daftar register yang akan dipantau disimpan pada file dengan format file txt (**signal-to-trace.txt**) , yang memiliki format khusus (yang bisa terbaca simulavr). Isi dari file tersebut adalah sebagai berikut:



```
1 # trace interrupt vector access -----  
2 # Compare match A -> vector @ 0x006  
3 # Compare match B -> vector @ 0x007  
4 + IRQ.VECTOR6  
5 + IRQ.VECTOR7  
6  
7 # trace counter value -----  
8 + TIMER1.Counter  
9  
10 # trace PORTD -----  
11 + PORTD.DDR  
12 + PORTD.D0-Out  
13 + PORTD.D1-Out  
14 + PORTD.D2-Out  
15 + PORTD.D3-Out
```

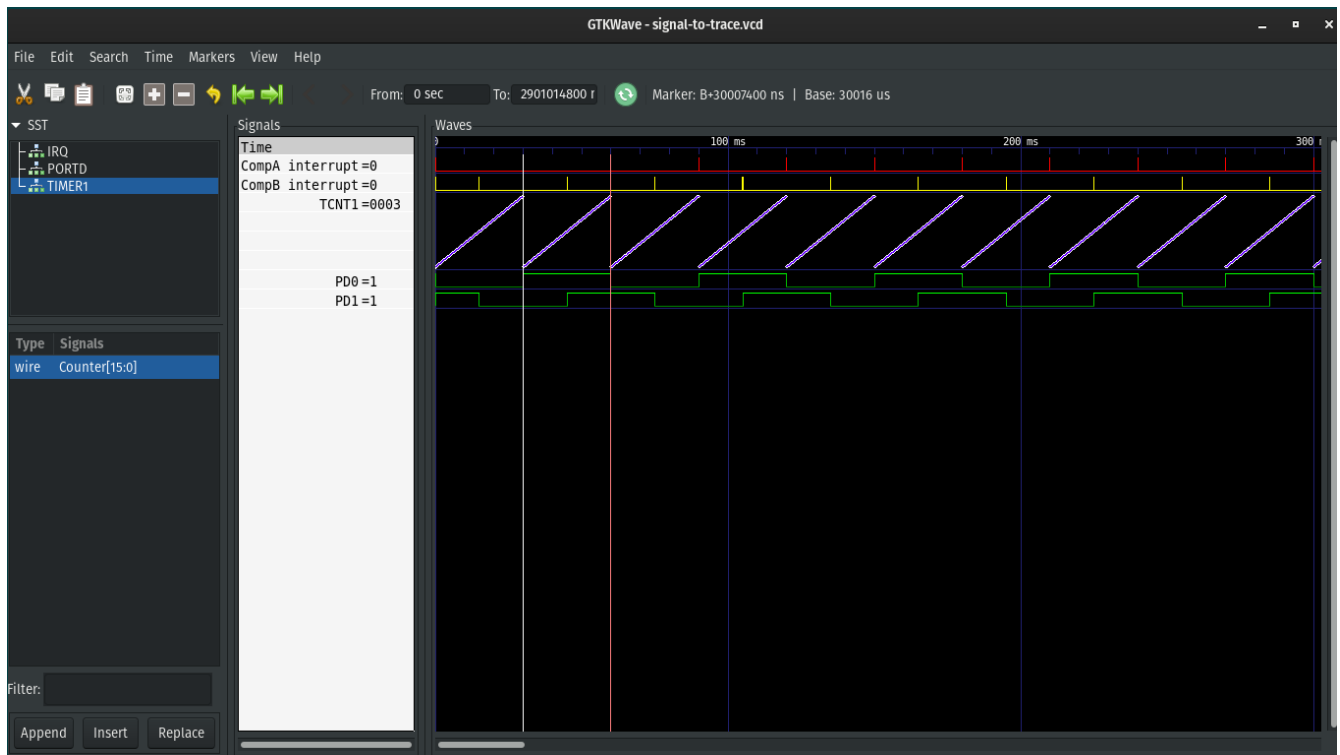
Gambar 3. Isi file *signal-to-trace.txt*

- Berdasarkan dokumentasi simulavr, simbol # menandakan baris tersebut adalah komentar, sedangkan simbol '+' menandakan register tersebut harus dipantau (tracing) perubahan nilainya saat simulasi.

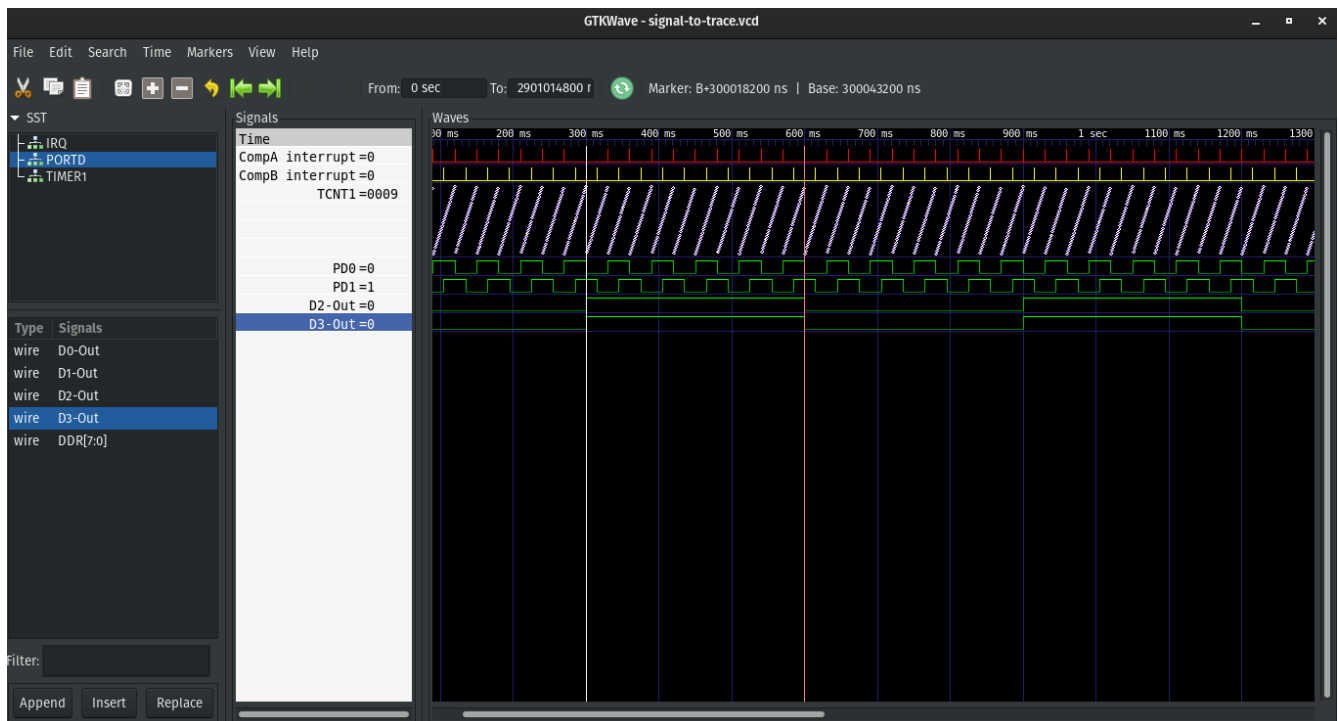
* nomor vektor interrupt diambil dari datasheet ATmega8

- Selanjutnya memanggil simulavr untuk menganalisa file **main.elf** hasil compile tersebut, dengan bantuan daftar **signal-to-trace.txt** , simulavr akan memberikan output berupa file **.vcd** (value change dump). File vcd ini merupakan representasi perubahan-perubahan nilai register yang terdaftar dalam list pemantauan tersebut. File .vcd ini bisa dibaca oleh GTKWave untuk memberikan tampilan visual timing diagram dari hasil simulasi.

Hasil Simulasi



Gambar 1. Overview #1 timing diagram dari sistem.



Gambar 2. Overview #2 with PD2 & PD3

Analisa Hasil Simulasi

Gambar 1:

Ketika timer 1 di start, maka counter akan menghitung naik dari 0 sampai TOP, disini TOP value adalah nilai OCR1A yang di set menjadi 19750, karena timer 1 di set menjadi CTC mode. Maka ketika nilai TCNT1 sama dengan OCR1A, akan terjadi 'Compare Match A' dan memanggil interrupt TIMER1_COMPA. Pada ISR interrupt ini, yang dilakukan hanya set flag bernama **compA_flag**, karena disini mengoptimalkan ISR memiliki proses yang sesingkat-singkatnya.

Selain itu, register OCR1B digunakan untuk memicu interrupt kedua yaitu 'Compare Match B'. Compare match B di set pada 15 ms sebelum compare match A terjadi dengan cara set nilai OCR1B menjadi $\frac{1}{2}$ OCR1A, yaitu 9375. Karena hal ini, hasil gelombang dari compare match B akan mengalami pergeseran fase sebesar 90° terhadap compare match A, seperti terlihat pada **Gambar 2**.

Pada main loop, yang dilakukan hanya mengecek flag-flag yang dipicu oleh kedua ISR compare match tersebut. Apabila compare match A yang terjadi, maka toggle PD0. Apabila yang terpicu flag compare match B, maka yang di toggle adalah PD1.

Gambar 2:

Pada branch compare match A juga terdapat proses penghitungan berapa jumlah compare match A yang terjadi. Apabila compare match A telah terjadi sebanyak 10 kali (idealnya 300ms) maka pin PD2 & PD3 akan di toggle.

Gambar 1 menunjukkan bahwa program akan melompat (jump instruction) menuju interrupt vector untuk menjalankan ISR yang sesuai setiap kali compare match A / B terjadi.

- Interrupt COMPA (0x006) : di akses setiap ~30ms (30007400 ns)
- Interrupt COMPB (0x007) : di akses setiap ~15ms sebelum compare match A terjadi

Gambar 2 menunjukkan bahwa PD2 dan PD3 di toggle setiap: **300.018200 ms**

*perbedaan kedua garis ukur terlihat pada bagian '**Marker B**'

Evaluasi

- Sebelumnya mencoba memaksakan dalam ISR menggunakan > 2 instruksi misalnya seperti berikut:

```
ISR(.....){
    count++
    toggle pin PD1

    cek count >= 2 ? :
        toggle pin PD0
}
```

yang terjadi adalah pengecekan count tidak dijalankan. Apabila instruksi-instruksinya dibalik, misalnya count++ dibawah, maka instruksi count++ tersebut tidak dijalankan. Sehingga untuk mempersingkat waktu eksekusi ISR, pada ISR hanya dilakukan set flag, pengecekan flag tetap pada main loop. Kendala ini yang menjadi salah satu pertimbangan sehingga desain sistem menggunakan fitur 2 buah compare match pada timer 1.

Update Perbandingan Dengan Tugas Sebelumnya

- Sebelumnya menggunakan timer 2 mode CTC, pada tugas ini yang digunakan timer 1 (16-bit) mode CTC agar bisa mengakses lama delay yang sama, namun prescaler dari timer dibuat sekecil mungkin (meminimalisir efek artifact), dan agar dapat menerapkan 2 interrupt untuk 2 nilai compare match yang berbeda

- Pada tugas sebelumnya, fungsi untuk setup timer dilakukan setiap kali timer akan di start, pada tugas ini timer di setup di awal program dan akan berjalan sendiri sepanjang program dengan bantuan interrupt.

Tugas 7 (polling method)	Tugas 8 (interrupt driven)
<pre>void start_timer2_CTC(){ TIFR = (1 << OCF2); TCCR2 = (1 << CS22) (1 << CS21) (1 << CS20) (1 << WGM21); while((TIFR & (1 << OCF2)) != (1 << OCF2)); TCCR2 = 0; return; }</pre>	<p>Inisialisasi timer dipindah ke fungsi setup()</p> <pre>void setup(){ TCCR1A = 0; TCCR1B = (1 << WGM12) (1 << CS11); TIMSK = (1 << OCIE1B) (1 << OCIE1A); OCR1A = 18750; OCR1B = 9375; // setup PORTD as outputs DDRD = 0x0F; PORTD = 0; }</pre> <p>ISR dari compare match:</p> <pre>ISR(TIMER1_COMPA_vect){ compA_flag = 1; } ISR(TIMER1_COMPB_vect){ compB_flag = 1; }</pre>

- fungsi **start_timer2_CTC()** dihilangkan (karena tidak menggunakan timer 2)
- fungsi **my_delay_ms()** dihilangkan
- include header file untuk fungsi delay dihilangkan (**util/delay.h**)