

Arduino Timer1 Class

Overview

TimerOne 과 관련된 Register를 초기화한다.

Pin과 PWM 주기를 설정한다.

PWM은 핀과 주기를 동시에 설정

: pinMode(TIMER1_A_PIN, OUTPUT);

: setPwmDuty(pin, duty);

setPwmDuty는 주기만 바꾼다

```
#include <TimerOne.h>
#define LEDPIN 13
bool on_off = false;
void setup() {
    // put your setup code here, to run once:
    Timer1.initialize();
    Timer1.setPeriod(1000000); //1hz
    Timer1.attachInterrupt(ledTunOnOff);
    pinMode(LEDPIN,OUTPUT);
}
void ledTunOnOff(){
    on_off = !on_off;
    if(on_off){
        digitalWrite(LEDPIN,HIGH);
    }
    else{
        digitalWrite(LEDPIN,LOW);
    }
}
void loop() {
    // put your main code here, to run repeatedly:
}
```

Properties

```
static unsigned short pwmPeriod;
static unsigned char clockSelectBits;
```

Header

TimerOne_h_

: 사용자 정의 헤더

Arduino.h

: 아두이노 내부 헤더

config\known_16bit_timers.h
: 사용자 정의 헤더

Function

`_BV()`
: <avr/sfr_defs.h>에 정의되어 있는 compiler macro
: #define _BV(bit)(1<<(bit))
`void initialize(unsigned long microseconds=1000000)`
`void setPeriod(unsigned long microseconds)`
`void setPwmDuty(char pin, unsigned int duty)`
`void pwm(char pin, unsigned int duty)`
`void pwm(char pin, unsigned int duty, unsigned long microseconds)`
`void disablePwm(char pin) __attribute__((always_inline))`
`void attachInterrupt(void (*isr)())`
`void attachInterrupt(void (*isr)(), unsigned long microseconds)`
`void detachInterrupt()`

Identifier

`F_CPU`
: 아두이노 내부 클럭 속도 (16Mhz)

`TIMER1_RESOLUTION`
: Timer1 is 16bit (65536UL)

`__AVR__`
: 아트멜 AVR 8비트 RISC 단일칩 마이크로 컨트롤러

`__attribute__`
: Unix/Linux 환경의 GCC 컴파일러는 `__attribute__`라는 속성 옵션을 사용
: GCC 컴파일러에게 추가적인 에러체킹을 지시

`__attribute__((always_inline))`
: 컴파일러가 함수의 characteristics에 관계없이 인라인 함수로 동작
: 인라인 함수는 함수의 코드를 복제해서 넣는다.
: 함수 호출 과정이 없어서 속도가 좀 더 빠르다 .
: 코드가 복제되므로 함수를 많이 사용하면 실행 파일의 크기가 커진다.

Code

TimerOne.h

```

/*
 * Interrupt and PWM utilities for 16 bit Timer1 on ATmega168/328
 * Original code by Jesse Tane for http://labs.ideo.com August 2008
 * Modified March 2009 by Jérôme Despatis and Jesse Tane for ATmega328 support
 * Modified June 2009 by Michael Polli and Jesse Tane to fix a bug in setPeriod()
which caused the timer to stop
 * Modified April 2012 by Paul Stoffregen - portable to other AVR chips, use inline
functions
 * Modified again, June 2014 by Paul Stoffregen - support Teensy 3.x & even more AVR
chips
 *
 *
 * This is free software. You can redistribute it and/or modify it under
 * the terms of Creative Commons Attribution 3.0 United States License.
 * To view a copy of this license, visit
http://creativecommons.org/licenses/by/3.0/us/
 * or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco,
California, 94105, USA.
 *
 */

#ifndef TimerOne_h_
#define TimerOne_h_

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#include "config/known_16bit_timers.h"

#define TIMER1_RESOLUTION 65536UL // Timer1 is 16 bit

// Placing nearly all the code in this .h file allows the functions to be
// inlined by the compiler. In the very common case with constant values
// the compiler will perform all calculations and simply write constants
// to the hardware registers (for example, setPeriod).

class TimerOne
{
public:
    //*****
    // Configuration
    //*****
    void initialize(unsigned long microseconds=1000000) __attribute__((always_inline))
    {
        TCCR1B = _BV(WGM13); // set mode as phase and frequency correct pwm,
stop the timer
        TCCR1A = 0; // clear control register A
        setPeriod(microseconds);
    }
    void setPeriod(unsigned long microseconds) __attribute__((always_inline)) {
        const unsigned long cycles = (F_CPU / 2000000) * microseconds;
        if (cycles < TIMER1_RESOLUTION) {
            clockSelectBits = _BV(CS10);

```

```

        pwmPeriod = cycles;
    } else
    if (cycles < TIMER1_RESOLUTION * 8) {
        clockSelectBits = _BV(CS11);
        pwmPeriod = cycles / 8;
    } else
    if (cycles < TIMER1_RESOLUTION * 64) {
        clockSelectBits = _BV(CS11) | _BV(CS10);
        pwmPeriod = cycles / 64;
    } else
    if (cycles < TIMER1_RESOLUTION * 256) {
        clockSelectBits = _BV(CS12);
        pwmPeriod = cycles / 256;
    } else
    if (cycles < TIMER1_RESOLUTION * 1024) {
        clockSelectBits = _BV(CS12) | _BV(CS10);
        pwmPeriod = cycles / 1024;
    } else {
        clockSelectBits = _BV(CS12) | _BV(CS10);
        pwmPeriod = TIMER1_RESOLUTION - 1;
    }
    ICR1 = pwmPeriod;
    TCCR1B = _BV(WGM13) | clockSelectBits;
}

//*****
// Run Control
//*****
void start() __attribute__((always_inline)) {
    TCCR1B = 0;
    TCNT1 = 0;           // TODO: does this cause an undesired interrupt?
    resume();
}

void stop() __attribute__((always_inline)) {
    TCCR1B = _BV(WGM13);
}

void restart() __attribute__((always_inline)) {
    start();
}

void resume() __attribute__((always_inline)) {
    TCCR1B = _BV(WGM13) | clockSelectBits;
}

//*****
// PWM outputs
//*****
void setPwmDuty(char pin, unsigned int duty) __attribute__((always_inline)) {
    unsigned long dutyCycle = pwmPeriod;
    dutyCycle *= duty;
    dutyCycle >>= 10;
    if (pin == TIMER1_A_PIN) OCR1A = dutyCycle;
#ifdef TIMER1_B_PIN
    else if (pin == TIMER1_B_PIN) OCR1B = dutyCycle;
#endif
#ifdef TIMER1_C_PIN
    else if (pin == TIMER1_C_PIN) OCR1C = dutyCycle;
#endif
}

void pwm(char pin, unsigned int duty) __attribute__((always_inline)) {
    if (pin == TIMER1_A_PIN) { pinMode(TIMER1_A_PIN, OUTPUT); TCCR1A |=
_BV(COM1A1); }

```

```

        #ifdef TIMER1_B_PIN
        else if (pin == TIMER1_B_PIN) { pinMode(TIMER1_B_PIN, OUTPUT); TCCR1A |=
        _BV(COM1B1); }
        #endif
        #ifdef TIMER1_C_PIN
        else if (pin == TIMER1_C_PIN) { pinMode(TIMER1_C_PIN, OUTPUT); TCCR1A |=
        _BV(COM1C1); }
        #endif
        setPwmDuty(pin, duty);
        TCCR1B = _BV(WGM13) | clockSelectBits;
    }
    void pwm(char pin, unsigned int duty, unsigned long microseconds)
    __attribute__((always_inline)) {
        if (microseconds > 0) setPeriod(microseconds);
        pwm(pin, duty);
    }
    void disablePwm(char pin) __attribute__((always_inline)) {
        if (pin == TIMER1_A_PIN) TCCR1A &= ~_BV(COM1A1);
        #ifdef TIMER1_B_PIN
        else if (pin == TIMER1_B_PIN) TCCR1A &= ~_BV(COM1B1);
        #endif
        #ifdef TIMER1_C_PIN
        else if (pin == TIMER1_C_PIN) TCCR1A &= ~_BV(COM1C1);
        #endif
    }

    //*****
    // Interrupt Function
    //*****
    void attachInterrupt(void (*isr)()) __attribute__((always_inline)) {
        isrCallback = isr;
        TIMSK1 = _BV(TOIE1);
    }
    void attachInterrupt(void (*isr)(), unsigned long microseconds)
    __attribute__((always_inline)) {
        if (microseconds > 0) setPeriod(microseconds);
        attachInterrupt(isr);
    }
    void detachInterrupt() __attribute__((always_inline)) {
        TIMSK1 = 0;
    }
    static void (*isrCallback)();

private:
    // properties
    static unsigned short pwmPeriod;
    static unsigned char clockSelectBits;

#elif defined(__arm__) && defined(CORE_TEENSY)

#if defined(KINETISK)
#define F_TIMER F_BUS
#elif defined(KINETISL)
#define F_TIMER (F_PLL/2)
#endif

```

```

public:
//*****
// Configuration
//*****
void initialize(unsigned long microseconds=1000000) __attribute__((always_inline))
{
    setPeriod(microseconds);
}
void setPeriod(unsigned long microseconds) __attribute__((always_inline)) {
    const unsigned long cycles = (F_TIMER / 2000000) * microseconds;
    if (cycles < TIMER1_RESOLUTION) {
        clockSelectBits = 0;
        pwmPeriod = cycles;
    } else
    if (cycles < TIMER1_RESOLUTION * 2) {
        clockSelectBits = 1;
        pwmPeriod = cycles >> 1;
    } else
    if (cycles < TIMER1_RESOLUTION * 4) {
        clockSelectBits = 2;
        pwmPeriod = cycles >> 2;
    } else
    if (cycles < TIMER1_RESOLUTION * 8) {
        clockSelectBits = 3;
        pwmPeriod = cycles >> 3;
    } else
    if (cycles < TIMER1_RESOLUTION * 16) {
        clockSelectBits = 4;
        pwmPeriod = cycles >> 4;
    } else
    if (cycles < TIMER1_RESOLUTION * 32) {
        clockSelectBits = 5;
        pwmPeriod = cycles >> 5;
    } else
    if (cycles < TIMER1_RESOLUTION * 64) {
        clockSelectBits = 6;
        pwmPeriod = cycles >> 6;
    } else
    if (cycles < TIMER1_RESOLUTION * 128) {
        clockSelectBits = 7;
        pwmPeriod = cycles >> 7;
    } else {
        clockSelectBits = 7;
        pwmPeriod = TIMER1_RESOLUTION - 1;
    }
    uint32_t sc = FTM1_SC;
    FTM1_SC = 0;
    FTM1_MOD = pwmPeriod;
    FTM1_SC = FTM_SC_CLKS(1) | FTM_SC_CPWMS | clockSelectBits | (sc &
FTM_SC_TOIE);
}

//*****
// Run Control
//*****
void start() __attribute__((always_inline)) {
    stop();
    FTM1_CNT = 0;
    resume();
}
void stop() __attribute__((always_inline)) {

```

```

    FTM1_SC = FTM1_SC & (FTM_SC_TOIE | FTM_SC_CPWMS | FTM_SC_PS(7));
}
void restart() __attribute__((always_inline)) {
    start();
}
void resume() __attribute__((always_inline)) {
    FTM1_SC = (FTM1_SC & (FTM_SC_TOIE | FTM_SC_PS(7))) | FTM_SC_CPWMS |
FTM_SC_CLKS(1);
}

//*****
//  PWM outputs
//*****
void setPwmDuty(char pin, unsigned int duty) __attribute__((always_inline)) {
    unsigned long dutyCycle = pwmPeriod;
    dutyCycle *= duty;
    dutyCycle >>= 10;
    if (pin == TIMER1_A_PIN) {
        FTM1_C0V = dutyCycle;
    } else if (pin == TIMER1_B_PIN) {
        FTM1_C1V = dutyCycle;
    }
}
void pwm(char pin, unsigned int duty) __attribute__((always_inline)) {
    setPwmDuty(pin, duty);
    if (pin == TIMER1_A_PIN) {
        *portConfigRegister(TIMER1_A_PIN) = PORT_PCR_MUX(3) | PORT_PCR_DSE |
PORT_PCR_SRE;
    } else if (pin == TIMER1_B_PIN) {
        *portConfigRegister(TIMER1_B_PIN) = PORT_PCR_MUX(3) | PORT_PCR_DSE |
PORT_PCR_SRE;
    }
}
void pwm(char pin, unsigned int duty, unsigned long microseconds)
__attribute__((always_inline)) {
    if (microseconds > 0) setPeriod(microseconds);
    pwm(pin, duty);
}
void disablePwm(char pin) __attribute__((always_inline)) {
    if (pin == TIMER1_A_PIN) {
        *portConfigRegister(TIMER1_A_PIN) = 0;
    } else if (pin == TIMER1_B_PIN) {
        *portConfigRegister(TIMER1_B_PIN) = 0;
    }
}

//*****
//  Interrupt Function
//*****
void attachInterrupt(void (*isr)()) __attribute__((always_inline)) {
    isrCallback = isr;
    FTM1_SC |= FTM_SC_TOIE;
    NVIC_ENABLE_IRQ(IRQ_FTM1);
}
void attachInterrupt(void (*isr)(), unsigned long microseconds)
__attribute__((always_inline)) {
    if(microseconds > 0) setPeriod(microseconds);
    attachInterrupt(isr);
}
void detachInterrupt() __attribute__((always_inline)) {
    FTM1_SC &= ~FTM_SC_TOIE;
}

```

```

        NVIC_DISABLE_IRQ(IRQ_FTM1);
    }
    static void (*isrCallback)();

private:
    // properties
    static unsigned short pwmPeriod;
    static unsigned char clockSelectBits;

#undef F_TIMER

#endif
};

extern TimerOne Timer1;

#endif

```

TimerOne.cpp

```

/*
 * Interrupt and PWM utilities for 16 bit Timer1 on ATmega168/328
 * Original code by Jesse Tane for http://labs.ideo.com August 2008
 * Modified March 2009 by Jérôme Despatis and Jesse Tane for ATmega328 support
 * Modified June 2009 by Michael Polli and Jesse Tane to fix a bug in
setPeriod() which caused the timer to stop
 * Modified Oct 2009 by Dan Clemens to work with timer1 of the ATmega1280 or
Arduino Mega
 * Modified April 2012 by Paul Stoffregen
 * Modified again, June 2014 by Paul Stoffregen
 *
 * This is free software. You can redistribute it and/or modify it under
 * the terms of Creative Commons Attribution 3.0 United States License.
 * To view a copy of this license, visit
http://creativecommons.org/licenses/by/3.0/us/
 * or send a letter to Creative Commons, 171 Second Street, Suite 300, San
Francisco, California, 94105, USA.
 *
 */

#include "TimerOne.h"

TimerOne Timer1;                // preinstantiate

unsigned short TimerOne::pwmPeriod = 0;
unsigned char TimerOne::clockSelectBits = 0;
void (*TimerOne::isrCallback)() = NULL;

// interrupt service routine that wraps a user defined function supplied by
attachInterrupt
#ifdef __AVR__
ISR(TIMER1_OVF_vect)
{

```



```
    Timer1.isrCallback();
}

#elif defined(__arm__) && defined(CORE_TEENSY)
void ftm1_isr(void)
{
    uint32_t sc = FTM1_SC;
    #ifdef KINETISL
    if (sc & 0x80) FTM1_SC = sc;
    #else
    if (sc & 0x80) FTM1_SC = sc & 0x7F;
    #endif
    Timer1.isrCallback();
}

#endif
```

Reference

[Timer1 Clock Source - Developer Help \(microchipdeveloper.com\)](https://www.microchip.com/developerhelp?id=TIMER1_CLOCK_SOURCE)