UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Chapter 5
Real Time Interfacing
(Timer, Counter and Interrupt)

Introduction

- Use of Timers and interrupts are always unnoticed because the low level hardware operation is **hidden** by the Arduino API.
- Many Arduino functions uses timers, eg: **delay(), millis(), micros();** PWM functions **analogWrite()**, **tone()** and **noTone()**, whereas the **Servo library** uses timers and interrupts.
- In general most tasks can be accomplished without resorting to interrupts but the use of interrupt service routines can clean up the code and add another dimension to the program for:
  - Provide a **fast response** to external inputs and user interface.
  - **Free up** main loop for other things instead of sitting in delay functions.
  - Provide **accurate timing** in conjunction with hardware timers.

Timer Interface

- A timer, or counter is like a clock, and can be used to measure time events.
- The timer can be programmed by some **special registers**. We can configure the pre-scaler for the timer, or the mode of operation and many other things.
- The ATmega328 microchips pin compatible and only differ in the size of internal memory. Both have 3 timers, called **Timer0, Timer1 and Timer2**. Timer0 and Timer2 are **8 bit timer**, where Timer1 is a **16 bit timer**.
- The most important difference between 8bit and 16bit timer is the **timer resolution**. 8bits means 256 values (two to the power of 8) where 16bit means 65536 values (two to the power of 16) which is much higher resolution.

Timer Interface

- The Arduino Mega series is based on the Atmel AVR ATmega1280 or the ATmega2560. They are almost identical to previous chips but only differs in memory size. Both chips have **6 timers**.

- First 3 timers (Timer 0, Timer1 and Timer2) are identical to the ATmega168/328. **Timer3, Timer4 and Timer5** are all **16bit** timers.

- All timers depends on the Arduino **system clock**. Normally the system clock is **16MHz**,

- The timer hardware can be configured with some special timer registers. In the Arduino firmware, all timers were configured to a 1kHz frequency and interrupts are generally enabled.

Timer Interface

- Timer0: **8bit timer** and used for the **timer functions**, like delay(), millis() and micros(). If Timer0 registers is being changed, this may influence the Arduino timer function.

- Timer1: **16bit timer**. The **Servo library** uses Timer1 on Arduino Uno (Timer5 on Arduino Mega).

- Timer2: **8bit timer** like Timer0. the **tone()** function uses Timer2.

- Timer3, Timer4, Timer5: Timer 3,4,5 are only available on Arduino Mega boards. These timers are all 16bit timers.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Timer Registers

Timer behavior can be change through the timer register setting: The most important timer registers are:

- **TCCRx** - Timer/Counter Control Register. The **pre-scaler** can be configured here.
- **TCNTx** - Timer/Counter Register. The **actual** timer value is stored here.
- **OCRx** - Output Compare Register
- **ICRx** - Input Capture Register (only for 16bit timer)
- **TIMSKx** - Timer/Counter Interrupt Mask Register. To enable/disable timer interrupts.
- **TIFRx** - Timer/Counter Interrupt Flag Register. Indicates a pending timer interrupt.
- \

Timer Registers - TCCRx

TCCR1A

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TCCR1B

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

UTM UNIVERSITI TEKNOLOGI MALAYSIA

Timer Registers – TCCR1B csxx

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clkI/O/1 (No prescaling)
0	1	0	clkI/O/8 (From prescaler)
0	1	1	clkI/O/64 (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	clkI/O/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Timer Registers – TCCR1A&B wgmxx

Table 16-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Timer Registers – TIMSK0

15.9.6 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:3 – Reserved**

These bits are reserved bits in the ATmega48A/PA/88A/PA/168A/PA/328/P and will always read as zero.

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter Interrupt Flag Register – TIFR0.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

Timer Example

To calculate the timer frequency (for example **2Hz** using **Timer1**):

1. CPU frequency 16Mhz for Arduino
2. Maximum timer counter value (256 for 8bit, 65536 for 16bit timer). Choose a pre-scaler, eg, 256
3. Divide CPU frequency through the chosen pre-scaler (16000000 / 256 = 62500)
4. Divide result through the desired frequency (62500 / 2Hz = 31250)
5. Verify the result against the maximum timer counter value (31250 < 65536 success) if fail, choose bigger pre-scaler.

Program Example

- Test

Interrupt Interface

- The program running on a controller is normally in **sequence (In**struction by instruction)
- **An interrupt** is an external event that interrupts the running program and runs a **special interrupt service routine (ISR).**
- After the ISR has been finished, the running program is **continued** with the following instruction.

Interrupt Interface

- Interrupts can be enabled or disabled with the function **interrupts()** or **noInterrupts()** by setting or clearing bits in the Interrupt mask register **(TIMSKx)**. By default the interrupts are enabled.

- When an interrupt occurs, a flag in the interrupt flag register **(TIFRx)** is **being set.** This interrupt will be cleared automatically when **entering the ISR** or by **manually clearing** the bit in the interrupt flag register.

- The Arduino functions **attachInterrupt()** and **detachInterrupt()** can only be used for external interrupt pins.

Timer Interrupts

- A timer can generate different types of interrupts.
- The register and bit definitions can be found in the processor data sheet (Atmega328 or Atmega2560)
- The **suffix x** stands for the timer number (0..5), the **suffix y** stands for the output number (A,B,C), for example **TIMSK1** (Timer1 interrupt mask register) or **OCR2A** (Timer2 output compare register A).

Timer Interrupts Registers

- **Timer Overflow:** Timer overflow means the timer has reached is limit value. When a timer overflow interrupt occurs, the timer overflow bit **TOVx** will be set in the interrupt flag register **TIFRx**. When the timer overflow interrupt enable bit **TOIEx** in the interrupt mask register **TIMSKx** is set, the timer overflow interrupt service routine ISR **(TIMERx_OVF_vect)** will be called.

- **Output Compare Match:** When a output compare match interrupt occurs, the **OCFxy** flag will be set in the interrupt flag register **TIFRx**. When the output compare interrupt enable bit **OCIExy** in the interrupt mask register **TIMSKx** is set, the output compare match interrupt service ISR **(TIMERx_COMPy_vect)** routine will be called.

Timer Interrupts Registers

- **Timer Input Capture:**

  When a timer input capture interrupt occurs, the input capture flag bit **ICFx** will be set in the interrupt flag register **TIFRx**. When the input capture interrupt enable bit **ICIEx** in the interrupt mask register **TIMSKx** is set, the timer input capture interrupt service routine **ISR(TIMERx_CAPT_vect)** will be called.

External Interrupts

- The Arduino functions ***attachInterrupt()*** and ***detachInterrupt()*** can only be used for external interrupt pins. Two pins can be used as external interrupts: pins 2 and 3.

- The interrupt is enabled through the following line:
  - attachInterrupt (interrupt, function, mode)

    Eg: attachInterrupt(0, doEncoder, FALLING);

Shukri - UTM FKE 2015

Timer & Interrupt Example 1

```
#define ledPin 13

void setup()
{
pinMode(ledPin, OUTPUT);

// initialize Timer1
noInterrupts(); // disable all interrupts
TCCR1A = 0;
TCCR1B = 0;
TCNT1 = 0;

OCR1A = 31250; // compare match register 16MHz/256/2Hz
TCCR1B |= (1 << WGM12); // CTC mode
TCCR1B |= (1 << TOIE1); // 256 prescaler
TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
interrupts(); // enable all interrupts
}

ISR(Timer1_COMPA_vect) // timer compare interrupt service routine
{
digitalWrite(ledPin, digitalRead(ledPin) ^ 1); // toggle LED pin
}

void loop()
{
// your program here...
}
```

Blinking LED with compare match interrupt

This example uses the Timer1 in CTC mode and the compare match interrupt to toggle a LED. The timer is configured for a frequency of 2Hz. The LED is toggled in the interrupt service routine.

UTM UNIVERSITI TEKNOLOGI MALAYSIA

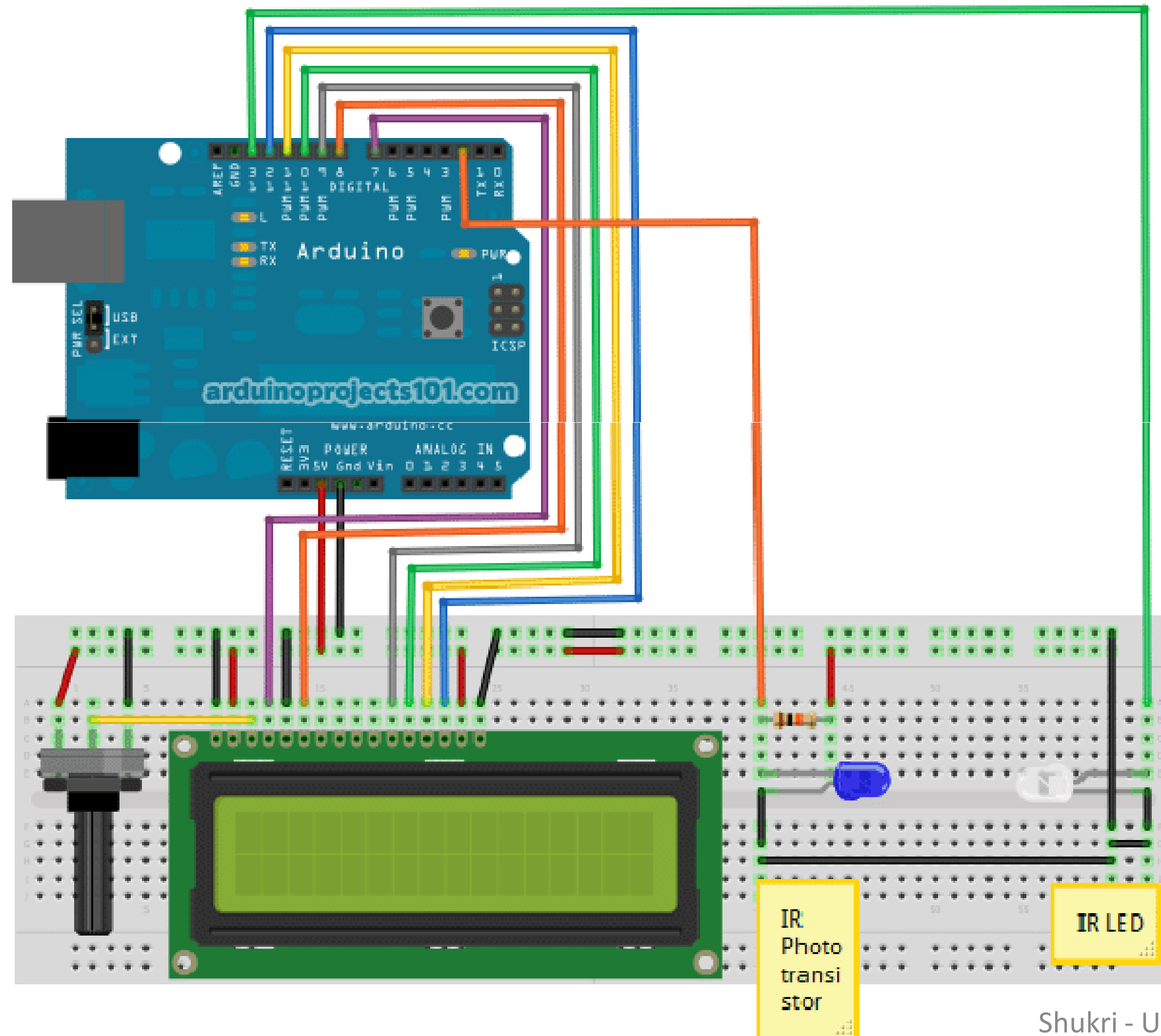
Timer & Interrupt Example 2

```
#define ledPin 13

void setup()
{
pinMode(ledPin, OUTPUT);

// initialize Timer1
noInterrupts(); // disable all interrupts
TCCR1A = 0;
TCCR1B = 0;

TCNT1 = 34286; // preload timer 65536-16MHz/256/2Hz
TCCR1B |= (1 << CS12); // 256 prescaler
TIMSK1 |= (1 << TOIE1); // enable timer overflow interrupt
interrupts(); // enable all interrupts
}

ISR(Timer1_OVF_vect) // interrupt service routine that wraps a
            //user defined function supplied by attachInterrupt
{
TCNT1 = 34286; // preload timer
digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}

void loop()
{
// your program here...
}
```

Blinking LED with timer overflow interrupt

Same example like before but now we use the timer overflow interrupt. In this case Timer1 is running in normal mode.
The timer must be pre loaded every time in the interrupt service routine.

Counter & Interrupt Example 3



Shukri - UTM FKE 2015

```
/*
 * Optical Tachometer
 *
 * Uses an IR LED and IR phototransistor to implement an optical tachometer.
 * The IR LED is connected to pin 13 and ran continually.
 * Pin 2 (interrupt 0) is connected across the IR detector.
 *
 * Code based on: www.instructables.com/id/Arduino-Based-Optical-Tachometer/
 * Coded by: arduinoprojects101.com
 */

int ledPin = 13;                // IR LED connected to digital pin 13
volatile byte rpmcount;
unsigned int rpm;
unsigned long timeold;

// include the library code:
#include <LiquidCrystal.h>
// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void rpm_fun()
 {
   //Each rotation, this interrupt function is run twice, so take that into consideration for
   //calculating RPM
   //Update count
      rpmcount++;
 }
```

```
void setup()
 {
   lcd.begin(16, 2);  // intialise the LCD

   //Interrupt 0 is digital pin 2, so that is where the IR detector is connected
   //Triggers on FALLING (change from HIGH to LOW)
   attachInterrupt(0, rpm_fun, FALLING);

   //Turn on IR LED
   pinMode(ledPin, OUTPUT);
   digitalWrite(ledPin, HIGH);

   rpmcount = 0;
   rpm = 0;
   timeold = 0;
 }

 void loop()
 {
   //Update RPM every second
   delay(1000);
   //Don't process interrupts during calculations
   detachInterrupt(0);
   //Note that this would be 60*1000/(millis() - timeold)*rpmcount if the interrupt
   //happened once per revolution instead of twice. Other multiples could be used
   //for multi-bladed propellers or fans
   rpm = 30*1000/(millis() - timeold)*rpmcount;
   timeold = millis();
    rpmcount = 0;

    //Print out result to lcd
    lcd.clear();
    lcd.print("RPM=");
    lcd.print(rpm);

    //Restart the interrupt processing
    attachInterrupt(0, rpm_fun, FALLING);
   }
```

Serial & Interrupt Example 4

```
/*
 Serial Event example

 When new serial data arrives, this sketch adds it to a String.
 When a newline is received, the loop prints the string and
 clears it. A good test for this is to try it with a GPS receiver
 that sends out NMEA 0183 sentences.
 Created 9 May 2011
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/SerialEvent

 */

String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;  // whether the string is
complete

void setup() {
  // initialize serial:
  Serial.begin(9600);
  // reserve 200 bytes for the inputString:
  inputString.reserve(200);
}
```

```
void loop() {
  // print the string when a newline arrives:
  if (stringComplete) {
    Serial.println(inputString);
    // clear the string:
    inputString = "";
    stringComplete = false;
  }
}

/*
  SerialEvent occurs whenever a new data comes in
the
 hardware serial RX.  This routine is run between each
 time loop() runs, so using delay inside loop can delay
 response.  Multiple bytes of data may be available.
 */
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

References

- http://arduino-info.wikispaces.com/Timers-Arduino
- http://duino4projects.com/arduino-rpm-counter-tachometer-code/
- http://playground.arduino.cc/Main/RotaryEncoders
- http://playground.arduino.cc/Code/Interrupts
- http://www.dave-auld.net/?option=com_content&view=article&id=107:arduino-interrupts&catid=53:arduino-input-output-basics&Itemid=107
- http://arduino-info.wikispaces.com/Timers-Arduino
- http://duino4projects.com/e-books/
- http://arduino.cc/en/Tutorial/SerialEvent