ENG20009
Engineering Technology Inquiry Project

Unit Convenor : Dr Rifai Chai

Email: rchai@swin.edu.au

Phone: 9214 8119

Office: EN606B

Swinburne University of Technology

Seminar 6 – Interrupts

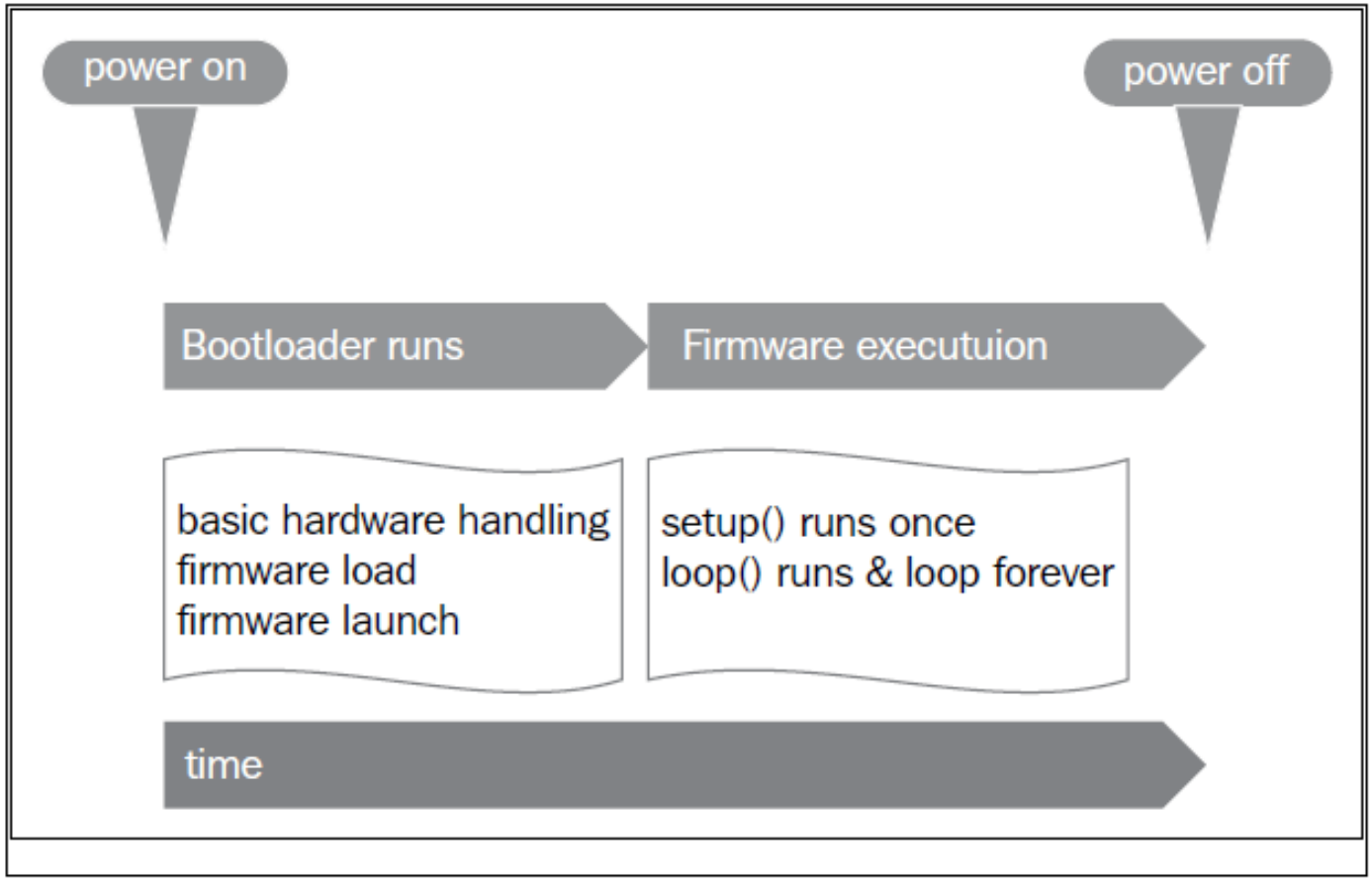
Topics:

1. Interrupt Overview
2. Arduino Interrupt
3. External Interrupt – Hardware interrupt
4. Internal Interrupt – Timer Interrupt
5. Examples interrupt in Arduino Due

1. Interrupt Overview

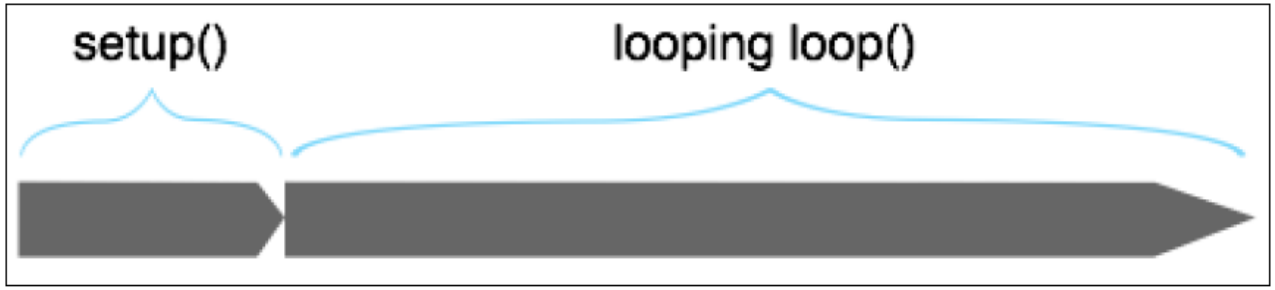
- Delay Concept: without even realizing it, you've already used the delay() core function and haven't realized it.
- Delaying an Arduino program can be done using the delay() and delayMicroseconds() functions directly in the loop() function.
- Both functions drive the program to make a pause. The only difference is that you have to provide a time in millisecond to delay() and a time in microseconds to delayMicroseconds().
- What does the program do during the delay? Nothing. It waits. When you call delay or delayMicroseconds in a program, it stops its execution for a certain amount of time.
- The following figure illustrating what happens when we power on our Arduino:

1. Interrupt Overview



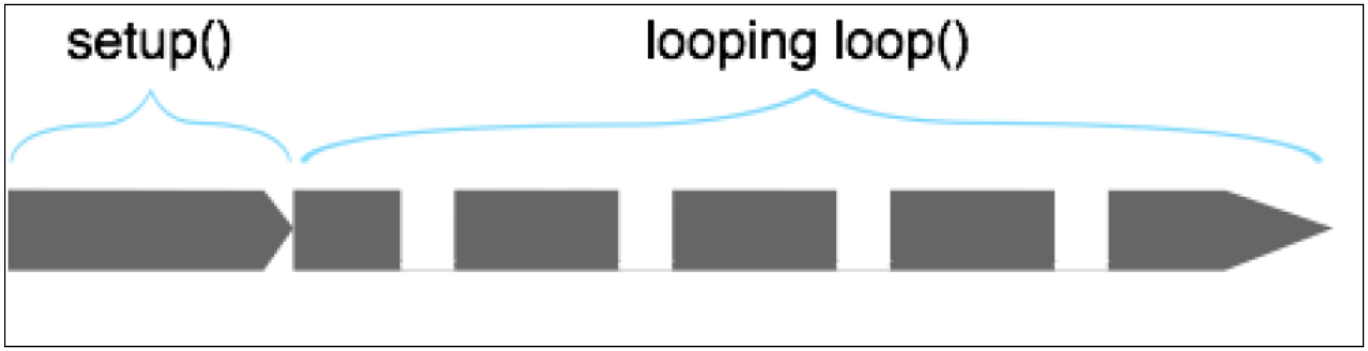
1. Interrupt Overview

- Figure to show the firmware execution itself, which is the part that we will work with, in the next rows:



The firmware life cycle with the main part looping

- Accepting the fact that when setup() stops, the loop() function begins to loop, everything in loop() is continuous. Now look at the same things when delays happen:



The firmware life cycle with the main part looping and breaking when delay() is called

1. Interrupt Overview

- The whole program breaks when delay() is called. The length of the break depends on the parameter passed to delay().
- We can notice that everything is done sequentially and in time. If a statement execution takes a lot of time, Arduino's chip executes it, and then continues with the next task.
- In that very usual and common case, if one particular task (statements, function calls, or whatever) takes a lot of time, the whole program could be hung and produce hiccup; consider the user experience.
- Imagine that concrete case in which you have to read sensors, flip-flop some switches, and write information to a display at the same time. If you do that sequentially and you have a lot of sensors, which is quite usual, you can have some lag and slowdown in the display of information because that task is executed after the other one in loop().

1. Interrupt Overview

- Pooling Concept: If I wanted to create a code that reads inputs, and performs something when a particular condition would be verified with the value of these inputs, I would write this pseudo-code:

```
setup()
- initialize things

loop()
- ask an input value and wait for it until it is available
- test this input according to something else
- if the test is true perform something else, loop to the beginning
```

1. Interrupt Overview

- The interrupt handler concept: Polling is nice but a bit time consuming, as we just figured out. The best way would be to be able to control when the processor would have to deal with inputs or outputs in a smarter way.
- Imagine our previously drawn example with many inputs and outputs. Maybe, this is a system that has to react according to a user action. Usually, we can consider the user inputs as much slower than the system's ability to answer.
- This means we could create a system that would interrupt the display as soon as a particular event would occur, such as a user input. This concept is called an event-based interrupt system.
- The interrupt is a signal. When a particular event occurs, an interrupt message is sent to the processor. Sometimes it is sent externally to the processor (hardware interrupt) and sometimes internally (software interrupt).

1. Interrupt Overview

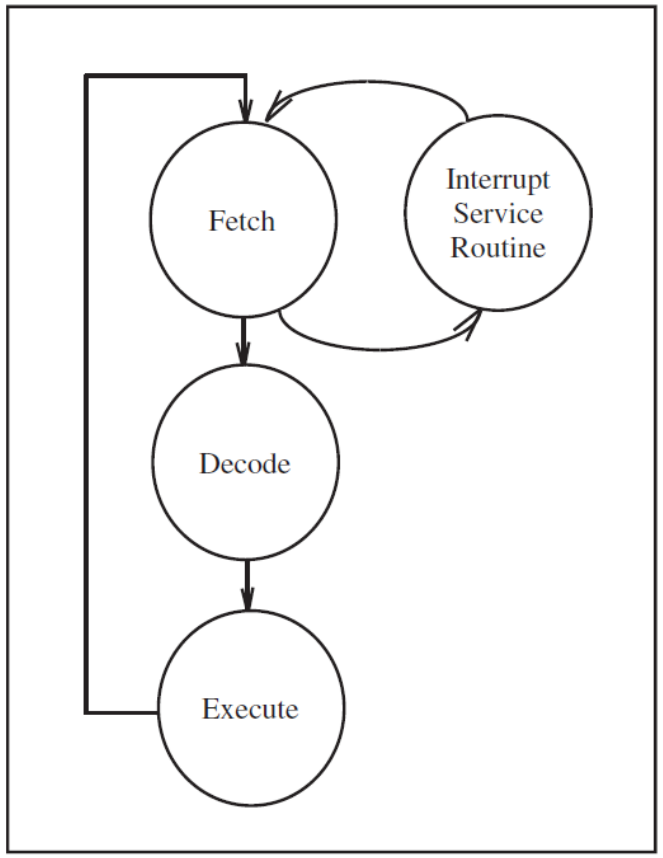
- The interrupt handler is a routine that handles the interrupt by doing something. For instance, on the move of the mouse, the computer operating system (commonly called the OS) has to redraw the cursor in another place.
- It would be time consuming to let the processor itself test each millisecond whether the mouse has moved, because the CPU would be running at 100 percent utilization. It seems smarter to have a part of the hardware for that purpose. When the mouse movement occurs, it sends an interrupt to the processor, and this later redraws the mouse.
- In the case of our installation with a huge number of inputs and outputs, we can consider handling the user inputs with an interrupt. We would have to implement what is called an Interrupt Service Routine (ISR), which is a routine called only when a physical world event occurs, that is, when a sensor value changes or something like that.

1. Interrupt Overview

- A microcontroller normally executes instructions in an orderly fetch-decode-execute sequence as dictated by a user-written program.
- However, the microcontroller must be equipped to handle unscheduled (although planned), higher priority events that might occur inside or outside the microcontroller. To process such events, a microcontroller requires an interrupt system.
- The interrupt system onboard a microcontroller allows it to respond to higher priority events. Appropriate responses to these events may be planned, but we do not know when these events will occur.
- When an interrupt event occurs, the microcontroller will normally complete the instruction it is currently executing and then transition program control to interrupt event specific tasks.

1. Interrupt Overview

- These tasks, which resolve the interrupt event, are organized into a function called an interrupt service routine (ISR). Each interrupt will normally have its own interrupt specific ISR. Once the ISR is complete, the microcontroller will resume processing where it left off before the interrupt event occurred.



1. Interrupt Overview

- So, Interrupts are a way to respond immediately to external signals without having to spend a lot of time looking for changes.
- Imagine you are at home, and you are waiting for an important parcel. This parcel will be delivered to your letter box without requiring a signature. The chances are that the postman will not knock on your door. You want to get your hands on it as soon as possible, so you go outside to look at the letter box frequently. It isn't there, so you wait for 10 minutes or so before having another look. You have to decide when to stop working (if you can actually work at all) before looking again, choosing a time that suits you. In computer terms, this continual checking for an event is previously mentioned known as polling.

1. Interrupt Overview

- Interrupts are different. A few days later, you wait for another parcel; only this time the parcel requires a signature, so the delivery man knocks on your door. This gives you a little more freedom. Because you don't have to waste time by looking inside the letter box every few minutes, you can get some work done.
- The delivery man will knock on your door to let you know that he has arrived, and at that time you can stop working for a few minutes to get your parcel. The downside to this is that you have to react quickly; if the delivery man does not get an answer quickly, he will go away. This situation is analogous to an interrupt.

1. Interrupt Overview

- Interrupts are a technique to let the processor continue working while waiting for an external event. It might not occur at all, in which case the main program continues, but if an external signal is received, the computer interrupts the main program and executes another routine, known as an Interrupt Service Routine, or ISR. ISRs are designed to be fast, and you should spend as little time as possible inside an ISR. When servicing an interrupt, some functions will not continue to work; delay() and millis() will not increment in interrupt context.

1. Interrupt Overview

- Basically, there are two kinds of interrupt:
  - An internal interrupt is usually caused by a timer. For example, the millis function uses one of the internal interrupts to trigger an action every millisecond (that of updating the counter).
  - An external interrupt is hardware-based. It usually occurs on a digital input. For example, a robot might use an interrupt for a ledge detector. When approaching a cliff or stair, the edge detector will signal the CPU of impending doom. If the CPU does not respond immediately, the robot might tumble down.

2. Arduino Uno Interrupt

https://content.arduino.cc/assets/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

12.4 Interrupt Vectors in ATmega328 and ATmega328P

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

2. Arduino Mega Interrupt

https://content.arduino.cc/assets/ATmega640-1280-1281-2560-2561-Datasheet-DS40002211A.pdf

14.1 Interrupt Vectors in ATmega640/1280/1281/2560/2561

Table 14-1. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016(3)	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI, STC	SPI Serial Transfer Complete
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator
30	$003A	ADC	ADC Conversion Complete

2. Arduino Mega Interrupt

Table 14-1. Reset and Interrupt Vectors (Continued)

Vector No.	Program Address(2)	Source	Interrupt Definition
31	$003C	EE READY	EEPROM Ready
32	$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	$0048	USART1 RX	USART1 Rx Complete
38	$004A	USART1 UDRE	USART1 Data Register Empty
39	$004C	USART1 TX	USART1 Tx Complete
40	$004E	TWI	2-wire Serial Interface
41	$0050	SPM READY	Store Program Memory Ready
42	$0052(3)	TIMER4 CAPT	Timer/Counter4 Capture Event
43	$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	$005C(3)	TIMER5 CAPT	Timer/Counter5 Capture Event
48	$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	$0066(3)	USART2 RX	USART2 Rx Complete
53	$0068(3)	USART2 UDRE	USART2 Data Register Empty
54	$006A(3)	USART2 TX	USART2 Tx Complete
55	$006C(3)	USART3 RX	USART3 Rx Complete
56	$006E(3)	USART3 UDRE	USART3 Data Register Empty
57	$0070(3)	USART3 TX	USART3 Tx Complete

2. Arduino Interrupt – Digital Pins for Interrupts

BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno, Nano, Mini, other 328-based	2, 3
Uno WiFi Rev.2, Nano Every	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21 (**pins 20 & 21** are not available to use for interrupts while they are used for I2C communication)
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR Family boards	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Nano 33 IoT	2, 3, 9, 10, 11, 13, A1, A5, A7
Nano 33 BLE, Nano 33 BLE Sense	all pins
Due	all digital pins
101	all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with **CHANGE**)

3. Arduino External Interrupt

BOARD	INT.0	INT.1	INT.2	INT.3	INT.4	INT.5
Uno	2	3				
Ethernet	2	3				
Leonardo	3	2	0	1	7	
Mega2560	2	3	21	20	19	18

- All Arduinos have interrupts; most use interrupts internally for serial communication or for timing counters. Some Arduinos have more user-programmable interrupts.
- Table above shows which interrupts are available on which pins for different models.
- The Arduino Due is different. It has highly advanced interrupt handling and can effectively be programmed to interrupt on every digital pin.

3. Arduino External Interrupt

attachInterrupt()

- This function specifies which routine to call when a specified interrupt is received.

```
attachInterrupt(interrupt, ISR, mode)
```

- This function attaches a function to the interrupt number interrupt, depending on the status of the pin. The mode specifies the pin state to trigger the interrupt.
- Valid states are LOW, CHANGE, RISING, FALLING, or HIGH. ISR names the function you want to run. The ISR can be any function you write, but it cannot have parameters and cannot return information.
- The Arduino Due has a slightly different prototype, as shown here:

```
attachInterrupt(pin, ISR, mode) // Arduino Due only!
```

3. Arduino External Interrupt

- Here again the details of the Interrupt Mode:
  - LOW: No current is applied to the interrupt pin.
  - CHANGE: The current changes, either between on and off or between off and on.
  - RISING: The current changes from off to on at 5 V.
  - FALLING: The current changes from on at 5 V to off.
- For example, to detect when a button attached to an interrupt pin has been pressed, we could use the RISING mode. Or, for example, if you had an electric trip wire running around your garden (connected between 5 V and the interrupt pin), then you could use the FALLING mode to detect when the wire has been tripped and broken.
- The delay() and Serial.available() functions will not work within a function that has been called by an interrupt.

3. Arduino External Interrupt

detachInterrupt()

- This function detaches a previously attached interrupt handler from attachInterrupt(). Interrupts on this ID will now be ignored. All other interrupts remain in place. It requires the interrupt ID to function.

```
detachInterrupt(interrupt);
```

- This function is again slightly different for the Arduino Due; the Due requires the pin number to be specified, not the interrupt ID.

```
detachInterrupt(pin); // Arduino Due only!
```

3. Arduino External Interrupt

noInterrupts()

- noInterrupts() temporarily disables interrupt handling. This is useful when you are in an interrupt handler and do not want to be disturbed by other interrupts.
- It does have a down side; some system functions require interrupts, mainly communication. Do not disable all interrupts just because your code does not require user-made interrupt handlers. Disable interrupts only when there is timing-critical code being performed.

```
// Normal code
noInterrupts();
// Time critical code
interrupts();
// Normal code
```

Interrupts()

- interrupts() re-enables all interrupts. You do not need to reconfigure interrupt handlers; all interrupts will be reconfigured as they were before calling noInterrupts().

3. Arduino External Interrupt

- Examples for configure interrupt:

```
attachInterrupt(0, function, mode);
attachInterrupt(1, function, mode);
```

Here, 0 is for digital pin 2, 1 is for digital pin 3, function is the name of the function to call when the interrupt is triggered, and mode is one of the four modes that triggers the interrupt.

- Examples Activating or Deactivating Interrupts:

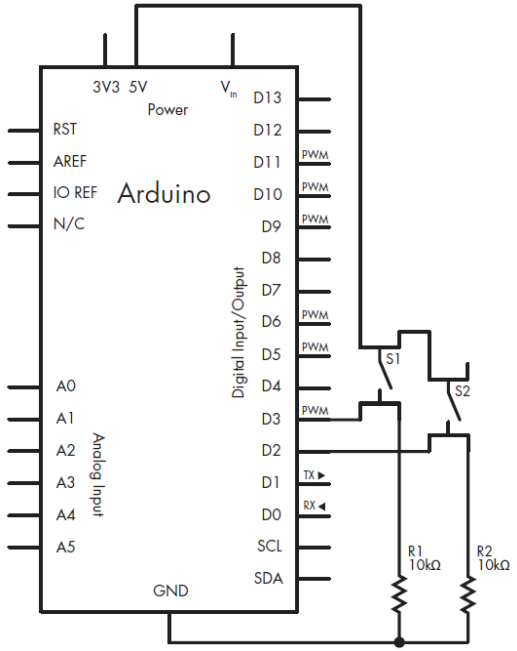
```
noInterrupts(); // deactivate interrupts
```

```
interrupts(); // reactivate interrupts
```

Interrupts work quickly and they are very sensitive, which makes them useful for time-critical applications or for “emergency stop” buttons on projects.

3. Creating Stopwatch – No interrupt

- Our stopwatch will use two buttons: one to start or reset the count and another to stop counting and show the elapsed time. The sketch will continually check the status of the two buttons. When the start button is pressed, a millis() value will be stored, and when the second button is pressed, a new millis() value will be stored. The custom function displayResult() will convert the elapsed time from milliseconds into hours, minutes, and seconds. Finally, the time will be displayed on the Serial Monitor.



3. Creating Stopwatch – No interrupt

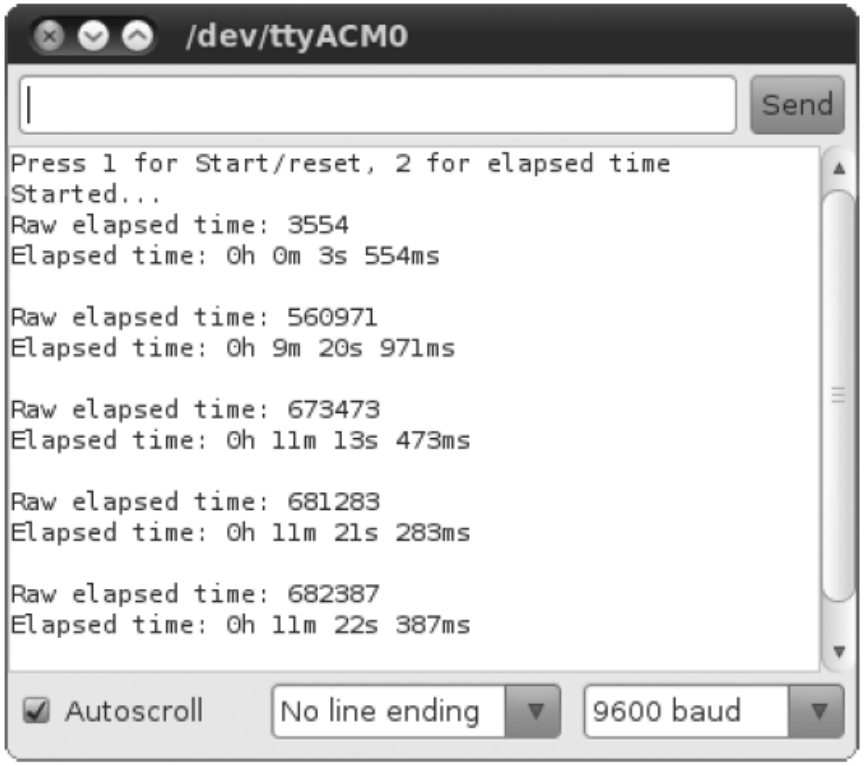
❑ Program/sketch:

```
unsigned long start, finished, elapsed;

void setup()
{
  Serial.begin(9600);
❶ pinMode(2, INPUT); // the start button
  pinMode(3, INPUT); // the stop button
  Serial.println("Press 1 for Start/reset, 2 for elapsed time");
}


void loop()
{
❸ if (digitalRead(2) == HIGH)
  {
    start = millis();
    delay(200); // for debounce
    Serial.println("Started...");
  }
❹ if (digitalRead(3) == HIGH)
  {
    finished = millis();
    delay(200); // for debounce
    displayResult();
  }
}
```

```
void displayResult()
{
  float h, m, s, ms;
  unsigned long over;

❷ elapsed = finished - start;

  h    = int(elapsed / 3600000);
  over = elapsed % 3600000;
  m    = int(over / 60000);
  over = over % 60000;
  s    = int(over / 1000);
  ms   = over % 1000;
  Serial.print("Raw elapsed time: ");
  Serial.println(elapsed);
  Serial.print("Elapsed time: ");
  Serial.print(h, 0);
  Serial.print("h ");
  Serial.print(m, 0);
  Serial.print("m ");
  Serial.print(s, 0);
  Serial.print("s ");
  Serial.print(ms, 0);
  Serial.println("ms");
  Serial.println();
}
```

3. Creating Stopwatch – No interrupt

- The basis for our stopwatch is simple. At ❶, we set up the digital input pins for the start and stop buttons. At ❸, if the start button is pressed, then the Arduino notes the value for millis() that we use to calculate the elapsed time once the stop button is pressed at ❹. After the stop button is pressed, the elapsed time is calculated in the function displayResult() at ❷ and shown in the Serial Monitor window.



3. Monitoring Interrupt pins

- Our example will blink the built-in LED every 500 milliseconds, during which time both interrupt pins will be monitored. When the button on interrupt 0 is pressed, the value for micros() will be displayed on the Serial Monitor, and when the button on interrupt 1 is pressed, the value for millis() will be displayed.
- This sketch will blink the onboard LED as shown in void loop() at ❸. When interrupt 0 is triggered, the function displayMicros() at ❶ will be called; or when interrupt 1 is triggered, the function displayMillis() at ❷ will be called. After either function has finished, the sketch resumes running the code in void loop. Open the Serial Monitor window and press the two buttons to view the values for millis() and micros().

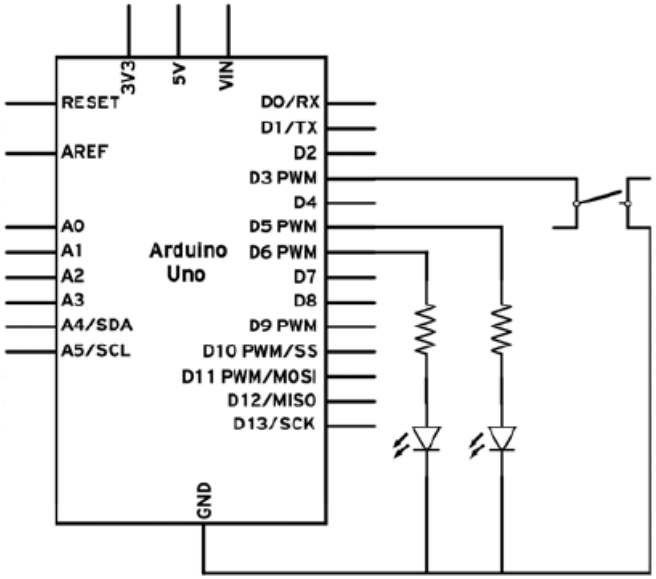
3. Monitoring Interrupt pins

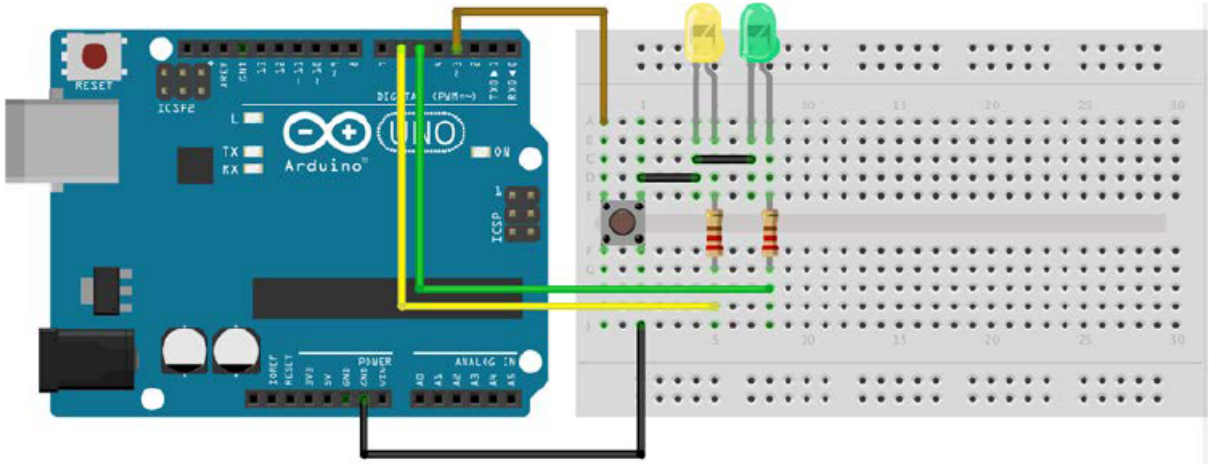
```
#define LED 13
void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  attachInterrupt(0, displayMicros, RISING);
  attachInterrupt(1, displayMillis, RISING);
}
```

```
❶ void displayMicros()
  {
    Serial.write("micros() = ");
    Serial.println(micros());
  }

❷ void displayMillis()
  {
    Serial.write("millis() = ");
    Serial.println(millis());
  }

❸ void loop()
  {
    digitalWrite(LED, HIGH);
    delay(500);
    digitalWrite(LED, LOW);
    delay(500);
  }
```

3. Monitoring Interrupt pins to fade two LEDs

- In this example, we will fade two LEDs using PWM and while that is happening, we will be able to select which LED is fading using the button connected to an interrupt.





3. Monitoring Interrupt pins to fade two LEDs

- Program/Sketch:

```
int LED1 = 5;
int LED2 = 6;

// Set a variable which we can change in the interrupt function
volatile int currentLED = LED1;

void setup(){
  // Set the button pin as an input with PULL UP resistor
  pinMode(3, INPUT_PULLUP);
  // Attach an interrupt to that pin which corresponds to interrupt 1
  // It will trigger when the input signals is FALLING
  attachInterrupt(1, changeLED, FALLING);
}

// Function that is being triggered by the interrupt
void changeLED(){
  if (currentLED == LED1) currentLED = LED2;
  else currentLED = LED1;
}

void loop(){
  // Fade In
  for (int i = 0; i < 256; i++){
    analogWrite(currentLED, i);
    delay(10);
  }

  // Fade In
  for (int i = 255; i > 0; i--){
    analogWrite(currentLED, i);
    delay(10);
  }
}
```

3. Monitoring Interrupt pins to fade two LEDs

The first important difference we see is the volatile variable type:

```
volatile int currentLED = LED1;
```

The volatile variable type is a directive to the compiler. It tells to store the variable in easily accessible RAM as it will be accessed during execution. Any variable we change during an interrupt-attached function has to be volatile, otherwise weird things will happen.

In the `setup()` function, we attach a function to interrupt 1. We call the `changeLED()` function and it will trigger when there is a falling edge in the signal on the interrupt pin. For more about trigger types, take a look at the *There's more...* section of this recipe.

```
attachInterrupt(1, changeLED, FALLING);
```

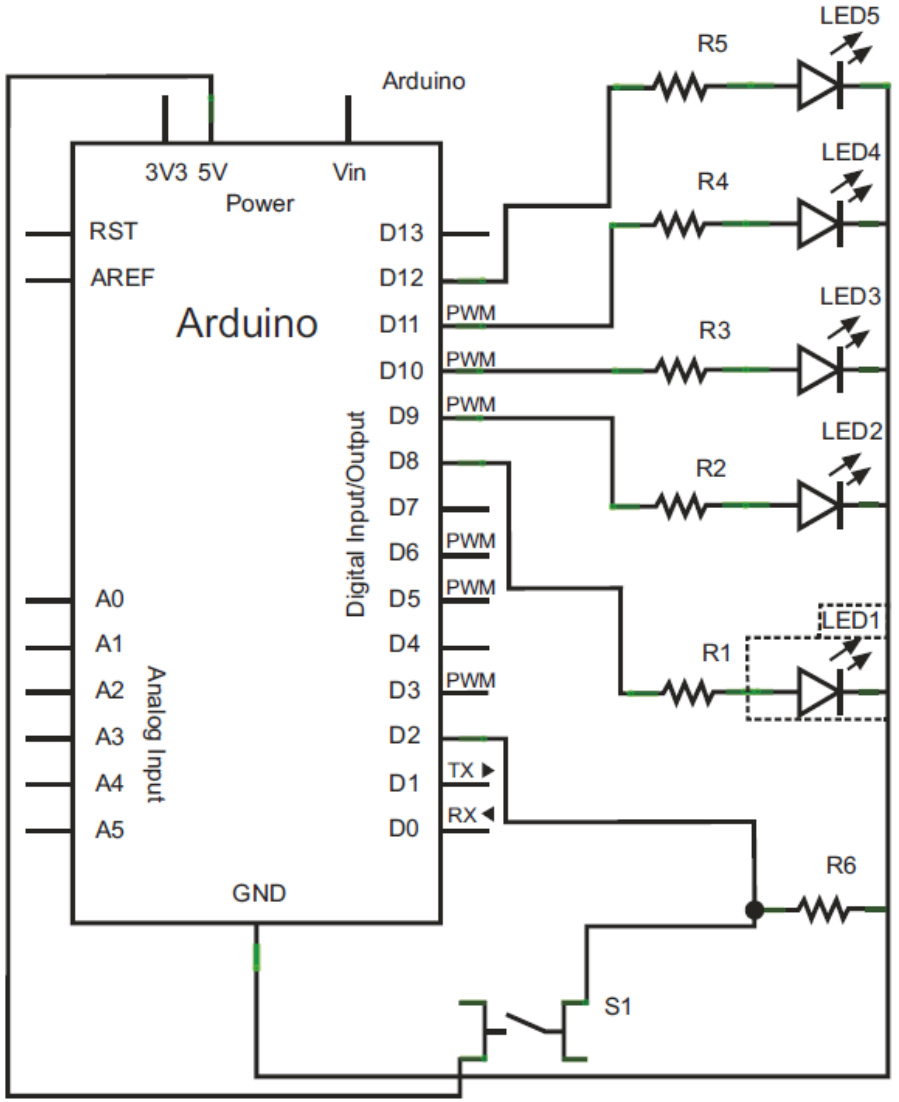
Now let's explore the `changeLED` function. Any function attached to an interrupt cannot return anything (it has to be a `void` function) and cannot have any arguments. Any variable modified inside an attached function has to be volatile.

```
void changeLED(){
  if (currentLED == LED1) currentLED = LED2;
  else currentLED = LED1;
}
```

In this function, we invert the current value of the `currentLED` variable. If it's `LED1` it becomes `LED2` and so on. This function can execute at any time a `FALLING` edge is detected on the interrupt pin.

3. Arduino Interrupt: control the LEDs with a push button

- Schematic of an Arduino connected to five LEDs controlled by a push button.



3. Arduino Interrupt: control the LEDs with a push button

```
volatile int state = LOW;
int ledArray[] = {8, 9, 10, 11, 12};
int count = 0;
int timer = 75;
int pause = 500;

void setup(){
  for (count=0;count<5;count++){
    pinMode(ledArray[count], OUTPUT);
  }
  attachInterrupt(0, ledOnOff, RISING);
}

void loop(){
  if (state) {
    for (count=0;count<5;count++){
      digitalWrite(ledArray[count], HIGH);
      delay(timer);
    }
    delay(pause);

    for (count=0;count<5;count++){
      digitalWrite(ledArray[count], LOW);
      delay(timer);
    }
    delay(pause);
  }
}
```

1 Sets up volatile variable

2 Initializes interrupt

3 Pauses code operation

3. Arduino Interrupt: control the LEDs with a push button

```
void ledOnOff() {
  static unsigned long lastMillis = 0;
  unsigned long newMillis = millis();
  if (newMillis - lastMillis < 50){
  }
  else {
    state = !state;
    lastMillis = newMillis;
  }
}
```

❹ Uses static variable

❺ Performs debounce check

- At the beginning of this code, you declare the state variable as volatile ❶. The volatile keyword is used for variables that can be altered by something outside the part of the sketch where it appears; one of the main uses of volatile is when using interrupts, as we're doing here.

3. Arduino Interrupt: control the LEDs with a push button

- The standard Arduino has two interrupts: interrupt 0 is attached to digital pin 2, and interrupt 1 is attached to digital pin 3.
- The Arduino Mega has an additional four interrupts: interrupt 2 is attached to digital pin 21, interrupt 3 is attached to digital pin 20, interrupt 4 is attached to digital pin 19, and interrupt 5 is attached to digital pin 18.
- The function attachInterrupt(interrupt, function, mode) takes three arguments. The first is interrupt, which can be set to 0 or 1; the second is the function to call, which must have no arguments and return nothing; and the third is the mode that generates the interrupt. This mode can have one of four values: LOW triggers whenever the pin is low; CHANGE triggers whenever the pin changes value; RISING triggers whenever the pin changes from LOW to HIGH; and FALLING triggers whenever the pin changes from HIGH to LOW.

3. Arduino Interrupt: control the LEDs with a push button

- In this sketch, you set the interrupt to RISING ❷.
- The interrupt will be triggered when the push button is pressed and the pin switches from LOW to HIGH.
- Another change in this sketch is that now the LEDs light up one after the other with a slight delay between each; then, when all the LEDs are lit, there is a slight pause ❸ and all the LEDs are switched off. The sequence then repeats. Pressing the push button stops the sequence; pressing it again restarts it.
- In this sketch, you counter the effect of the switch bouncing by using a static variable called lastMillis ❹..
- Static variables keep their values between calls to a function. The millis() function returns the number of milliseconds since the program started, and each time the interrupt-service routine is called, you assign the value of millis() to the variable newMillis. You then compare the value of newMillis to lastMillis ❺;

3. Arduino Interrupt: control the LEDs with a push button

- If the result is less than 50 milliseconds (the bounce period), you do nothing and return to the main sketch loop. If the value is greater than or equal to 50 milliseconds, you're outside the bounce period, meaning that the button has really been pressed again. In that case, you update the state variable and assign the value of newMillis to lastMillis before returning to the main sketch loop.
- Many people consider interrupts an advanced technique, but if you're careful, you should have no problem using them. During interruptservice routines, keep your sketch code as small as possible; this will help you avoid unexpected things happening in the rest of your sketch. Another caveat is that you can't use the delay function inside an interrupt-service routine.

4. Arduino Internal Interrupt – Timer Interrupt

- Timer interrupts allow you to perform a task at very specifically timed intervals regardless of what else is going on in your code. In this instructable I'll explain how to setup and execute an interrupt in Clear Timer on Compare Match or CTC Mode.
- Normally when you write an Arduino sketch the Arduino performs all the commands encapsulated in the loop() {} function in the order that they are written, however, it's difficult to time events in the loop(). Some commands take longer than others to execute, some depend on conditional statements (if, while...) and some Arduino library functions (like digitalWrite or analogRead) are made up of many commands. Arduino timer interrupts allow you to momentarily pause the normal sequence of events taking place in the loop() function at precisely timed intervals, while you execute a separate set of commands. Once these commands are done the Arduino picks up again where it was in the loop().

4. Arduino Internal Interrupt – Timer Interrupt

- Prescalers and the Compare Match Register

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/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.

4. Arduino Internal Interrupt – Timer Interrupt

- The Uno has three timers called timer0, timer1, and timer2. Each of the timers has a counter that is incremented on each tick of the timer's clock. CTC timer interrupts are triggered when the counter reaches a specified value, stored in the compare match register. Once a timer counter reaches this value it will clear (reset to zero) on the next tick of the timer's clock, then it will continue to count up to the compare match value again. By choosing the compare match value and setting the speed at which the timer increments the counter, you can control the frequency of timer interrupts.
- The Arduino clock runs at 16MHz, this is the fastest speed that the timers can increment their counters. At 16MHz each tick of the counter represents 1/16,000,000 of a second (~63ns), so a counter will take 10/16,000,000 seconds to reach a value of 9 (counters are 0 indexed), and 100/16,000,000 seconds to reach a value of 99.

4. Arduino Internal Interrupt – Timer Interrupt

- In many situations, you will find that setting the counter speed to 16MHz is too fast. Timer0 and timer2 are 8 bit timers, meaning they can store a maximum counter value of 255. Timer1 is a 16 bit timer, meaning it can store a maximum counter value of 65535. Once a counter reaches its maximum, it will tick back to zero (this is called overflow). This means at 16MHz, even if we set the compare match register to the max counter value, interrupts will occur every 256/16,000,000 seconds (~16us) for the 8 bit counters, and every 65,536/16,000,000 (~4 ms) seconds for the 16 bit counter. Clearly, this is not very useful if you only want to interrupt once a second.
- Instead you can control the speed of the timer counter incrementation by using something called a prescaler. A prescaler dictates the speed of your timer according the the following equation:

(timer speed (Hz)) = (Arduino clock speed (16MHz)) / prescaler

4. Arduino Internal Interrupt – Timer Interrupt

- So a 1 prescaler will increment the counter at 16MHz, an 8 prescaler will increment it at 2MHz, a 64 prescaler = 250kHz, and so on. As indicated in the tables above, the prescaler can equal 1, 8, 64, 256, and 1024. (I'll explain the meaning of CS12, CS11, and CS10 in the next step.)

- Now you can calculate the interrupt frequency with the following equation:

  interrupt frequency (Hz) = (Arduino clock speed 16,000,000Hz) / (prescaler * (compare match register + 1))

- the +1 is in there because the compare match register is zero indexed, rearranging the equation above, you can solve for the compare match register value that will give your desired interrupt frequency:

4. Arduino Internal Interrupt – Timer Interrupt

compare match register = [16,000,000Hz/ (prescaler * desired interrupt frequency)] - 1

- Remember that when you use timers 0 and 2 this number must be less than 256, and less than 65536 for timer1

  so if you wanted an interrupt every second (frequency of 1Hz):
  compare match register = [16,000,000 / (prescaler * 1)] -1
  with a prescaler of 1024 you get:
  compare match register = [16,000,000 / (1024 * 1)] -1
  = 15,624
  since 256 < 15,624 < 65,536, you must use timer1 for this interrupt.

4. Arduino Internal Interrupt – Timer Interrupt

```
1   //timer interrupts
2
3   //storage variables
4   boolean toggle1 = 0;
5
6   void setup(){
7
8     //set pins as outputs
9     pinMode(13, OUTPUT);
10
11  cli();//stop interrupts
12
13  //set timer1 interrupt at 1Hz
14    TCCR1A = 0;// set entire TCCR1A register to 0
15    TCCR1B = 0;// same for TCCR1B
16    TCNT1  = 0;//initialize counter value to 0
17    // set compare match register for 1hz increments
18    OCR1A = 15624;// = (16*10^6) / (1*1024) - 1 (must be <65536)
19    // turn on CTC mode
20    TCCR1B |= (1 << WGM12);
21    // Set CS12 and CS10 bits for 1024 prescaler
22    TCCR1B |= (1 << CS12) | (1 << CS10);
23    // enable timer compare interrupt
24    TIMSK1 |= (1 << OCIE1A);
25
26  sei();//allow interrupts
27
28  }//end setup
```

4. Arduino Internal Interrupt – Timer Interrupt

```
30  ISR(TIMER1_COMPA_vect){//timer1 interrupt 1Hz toggles pin 13 (LED)
31    if (toggle1){
32      digitalWrite(13,HIGH);
33      toggle1 = 0;
34    }
35    else{
36      digitalWrite(13,LOW);
37      toggle1 = 1;
38    }
39  }
40
41
42  void loop(){
43    //do other things here
44  }
45
46
47
```

- The program will shows the LED attached to pin 13 turning on for one second then turning off for one second (timer1 interrupt).

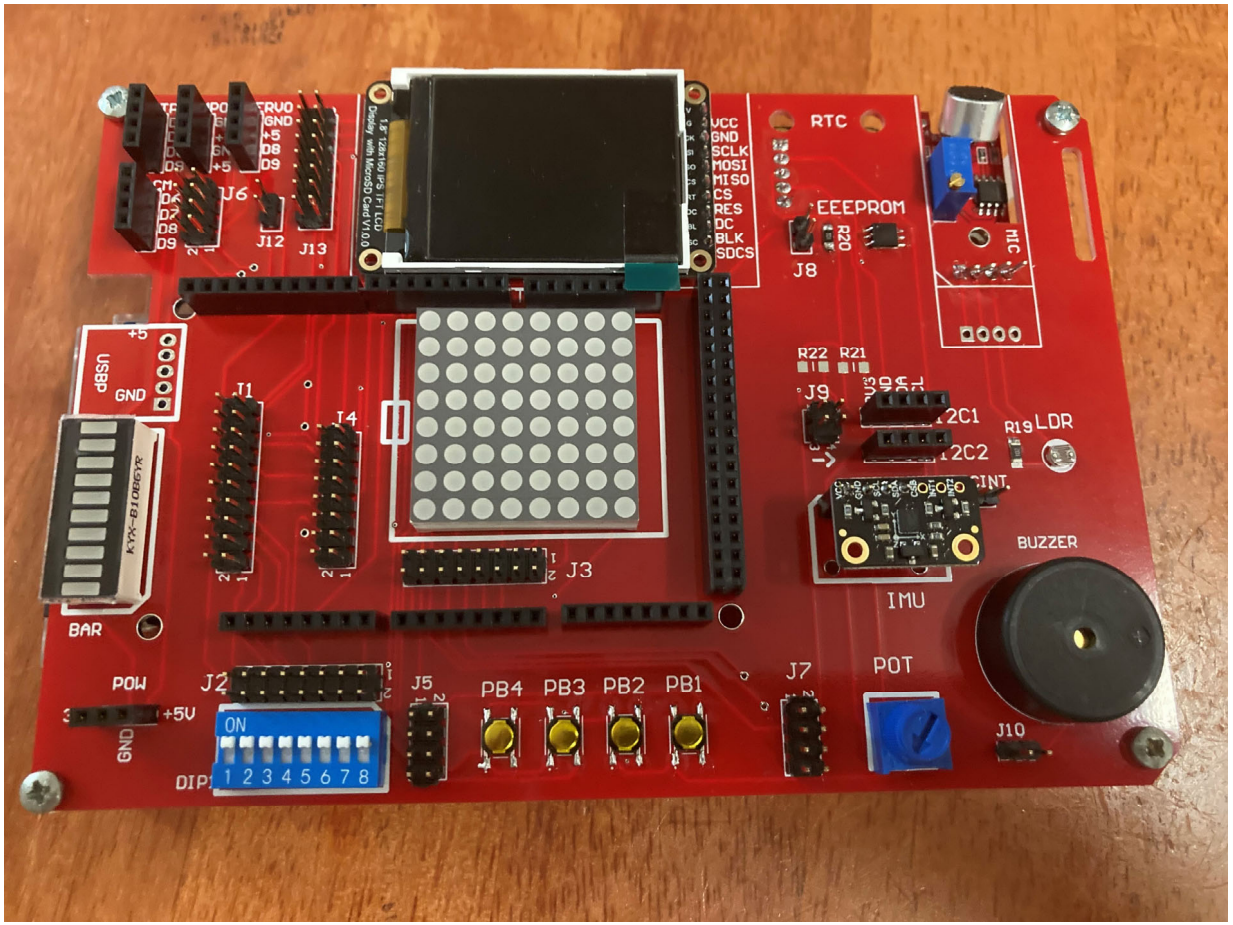
5. Examples interrupt in Arduino Zero and Arduino Due

- External/Hardware interrupt

Example code for hardware interrupt:

```
attachInterrupt(digitalPinToInterrupt(pin),ISR,mode);//recommended for arduino board
attachInterrupt(pin, ISR, mode) ; //recommended Arduino Due, Zero only
//argument pin: the pin number
//argument ISR: the ISR to call when the interrupt occurs;
   //this function must take no parameters and return nothing.
   //This function is sometimes referred to as an interrupt service routine.
//argument mode: defines when the interrupt should be triggered.
```

5. Examples interrupt in Arduino Zero and Arduino Due



- Apply hardware interrupt to PB1 (D2)
- Turn on/off with delay 500ms to one of LED in the LED Bar (D11).

5. Examples interrupt in Arduino Zero and Arduino Due

```
1  //Example - Exernal interrupt
2
3  #define LED 11
4  #define BUTTON_PIN 2
5  void setup() {
6    // put your setup code here, to run once:
7
8    Serial.begin(9600); // setup serial
9    pinMode(LED, OUTPUT); // setup output
10   pinMode(BUTTON_PIN, INPUT_PULLUP);
11   attachInterrupt(BUTTON_PIN,displaymic,LOW); //start the int.0   // opton 1 to configure hardware/external interupt
12  // attachInterrupt(digitalPinToInterrupt(21), displaymic,RISING); // option 2
13
14 }
15
16 //ISR - external interupt
17 void displaymic(){
18   Serial.write("micros = ");
19   Serial.println(micros());
20 }
21
22 void loop() {
23   // put your main code here, to run repeatedly:
24   // turn on/off LED
25   digitalWrite(LED, HIGH);
26   delay(500);
27   digitalWrite(LED, LOW);
28   delay(500);
29
30 }
31
```

5. Examples interrupt in Arduino Due

- Timer Interrupt
- The following is based on an Arduino forum entry at:
https://forum.arduino.cc/t/timer-interrupts-on-due/127643

- Below is the code to implement.
- startTimer sets the interrupt frequency and the pointer to the interrupt service routine (ISR).
- TC3_Handler is the actual ISR where you can place your code that will be executed with the highest possible precision.

5. Interrupt Line Mapping in Arduino Zero

Peripheral Source	NVIC Line
EIC NMI – External Interrupt Controller	NMI
PM – Power Manager	0
SYSCTRL – System Control	1
WDT – Watchdog Timer	2
RTC – Real-Time Counter	3
EIC – External Interrupt Controller	4
NVMCTRL – Nonvolatile Memory Controller	5
DMAC - Direct Memory Access Controller	6
USB - Universal Serial Bus	7
EVSYS – Event System	8
SERCOM0 – Serial Communication Interface 0	9
SERCOM1 – Serial Communication Interface 1	10
SERCOM2 – Serial Communication Interface 2	11
SERCOM3 – Serial Communication Interface 3	12
SERCOM4 – Serial Communication Interface 4	13
SERCOM5 – Serial Communication Interface 5	14
TCC0 – Timer Counter for Control 0	15
TCC1 – Timer Counter for Control 1	16
TCC2 – Timer Counter for Control 2	17
TC3 – Timer Counter 3	18
TC4 – Timer Counter 4	19
TC5 – Timer Counter 5	20
TC6 – Timer Counter 6	21
TC7 – Timer Counter 7	22
ADC – Analog-to-Digital Converter	23
AC – Analog Comparator	24
DAC – Digital-to-Analog Converter	25
PTC – Peripheral Touch Controller	26
I2S - Inter IC Sound	27
AC1 - Analog Comparator 1	28
TCC3 - Timer Counter for Control 3	29

5. Examples interrupt in Arduino Due

```
// ---------------------------------------------------------------
// Timer Interrupt Setup for Arduino Due
// ---------------------------------------------------------------
// Reference: http://forum.arduino.cc/index.php?topic=130423.0
void TC3_Handler()
{
   TC_GetStatus(TC1, 0);
   // Your code here…
}
void startTimer(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t frequency)
{
   pmc_set_writeprotect(false);
   pmc_enable_periph_clk((uint32_t)irq);
   TC_Configure(tc, channel, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC
                                          |TC_CMR_TCCLKS_TIMER_CLOCK4);
   uint32_t rc = VARIANT_MCK/128/frequency; //128 because TIMER_CLOCK4 is selected
   TC_SetRA(tc, channel, rc/2); //50% high, 50% low
   TC_SetRC(tc, channel, rc);
   TC_Start(tc, channel);
   tc->TC_CHANNEL[channel].TC_IER=TC_IER_CPCS;
   tc->TC_CHANNEL[channel].TC_IDR=~TC_IER_CPCS;
   NVIC_EnableIRQ(irq);
}
```

5. Examples interrupt in Arduino Due

- To initialize a timer in an application:

```
#define TIMERFREQUENCY 1000 // Time = 1 sec / TIMERFREQUENCY
void setup()
{
   // Set up the timer interrupt
   // TC1 channel 0, the IRQ for that channel and the desired frequency
   startTimer(TC1, 0, TC3_IRQn, TIMERFREQUENCY);
}
void loop(){
}
```

5. Examples interrupt in Arduino Zero

- To initialize a timer 4 with Zero:

```
//https://forum.arduino.cc/t/timer-interrupt-on-arduino-zero-board/409166/6
boolean toggle1 = 0;

void setup() {
  Serial.begin(9600);
  pinMode(11, OUTPUT); // setup output

  PORT->Group[PORTA].DIRSET.reg = PORT_PA06;       // Set-up digital pin D8 as an output

  // Set up the generic clock (GCLK4) used to clock timers
  GCLK->GENDIV.reg = GCLK_GENDIV_DIV(1) |          // Divide the 48MHz clock source by divisor 1: 48MHz/1=48MHz
                     GCLK_GENDIV_ID(4);            // Select Generic Clock (GCLK) 4
  while (GCLK->STATUS.bit.SYNCBUSY);               // Wait for synchronization

  GCLK->GENCTRL.reg = GCLK_GENCTRL_IDC |           // Set the duty cycle to 50/50 HIGH/LOW
                      GCLK_GENCTRL_GENEN |         // Enable GCLK4
                      GCLK_GENCTRL_SRC_DFLL48M |   // Set the 48MHz clock source
                      GCLK_GENCTRL_ID(4);          // Select GCLK4
  while (GCLK->STATUS.bit.SYNCBUSY);               // Wait for synchronization

  // Feed GCLK4 to TC4 and TC5
  GCLK->CLKCTRL.reg = GCLK_CLKCTRL_CLKEN |         // Enable GCLK4 to TC4 and TC5
                      GCLK_CLKCTRL_GEN_GCLK4 |     // Select GCLK4
                      GCLK_CLKCTRL_ID_TC4_TC5;     // Feed the GCLK4 to TC4 and TC5
  while (GCLK->STATUS.bit.SYNCBUSY);               // Wait for synchronization

  TC4->COUNT16.CC[0].reg = 0xB71A;                 // Set the TC4 CC0 register as the TOP value in match frequency mode
  while (TC4->COUNT16.STATUS.bit.SYNCBUSY);        // Wait for synchronization

  //NVIC_DisableIRQ(TC4_IRQn);
  //NVIC_ClearPendingIRQ(TC4_IRQn);
  NVIC_SetPriority(TC4_IRQn, 0);    // Set the Nested Vector Interrupt Controller (NVIC) priority for TC4 to 0 (highest)
  NVIC_EnableIRQ(TC4_IRQn);         // Connect TC4 to Nested Vector Interrupt Controller (NVIC)

  TC4->COUNT16.INTFLAG.bit.OVF = 1;                       // Clear the interrupt flags
  TC4->COUNT16.INTENSET.bit.OVF = 1;                      // Enable TC4 interrupts

  TC4->COUNT16.CTRLA.reg |= TC_CTRLA_PRESCALER_DIV1024 |  // Set prescaler to 1024, 48MHz/1024 = 46.875kHz
                            TC_CTRLA_WAVEGEN_MFRQ |       // Put the timer TC4 into match frequency (MFRQ) mode
                            TC_CTRLA_ENABLE;              // Enable TC4
  while (TC4->COUNT16.STATUS.bit.SYNCBUSY);

}
```

5. Examples interrupt in Arduino Zero

```
44  void loop() {
45    // put your main code here, to run repeatedly:
46
47  }
48
49
50  void TC4_Handler()                                // Interrupt Service Routine (ISR) for timer TC4
51  {
52    // Check for overflow (OVF) interrupt
53    if (TC4->COUNT16.INTFLAG.bit.OVF && TC4->COUNT16.INTENSET.bit.OVF)
54    {
55      // Put your timer overflow (OVF) code here:
56      // ...
57   //   PORT->Group[PORTA].OUTTGL.reg = PORT_PA06;     // Toggle the D8 pin HIGH LOW
58     Serial.println("true");
59     if (toggle1){
60      digitalWrite(11,HIGH);
61      toggle1 = 0;
62    }
63    else{
64      digitalWrite(11,LOW);
65      toggle1 = 1;
66    }
67
68      TC4->COUNT16.INTFLAG.bit.OVF = 1;             // Clear the OVF interrupt flag
69    }
70  }
```

Commonwealth of Australia
Copyright Act 1968

Notice for paragraph 135ZXA (a) of the *Copyright Act 1968*

Warning

This material has been reproduced and communicated to you by or on behalf of Swinburne University of Technology under Part VB of the *Copyright Act 1968* (the ***Act***).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.