Using Interrupts in the Arduino IDE

Using an Interrupt is a great way to capture a triggered event. It ensures the event will be captured regardless of what other processes are running in the background. Internal Interrupts are enabled as a default, this is mainly used for communications.

In the Arduino IDE you can use external interrupts, by initializing them in the startup routine as seen below.

void setup()

{

attachInterrupt(0, interrupt, LOW);

InterruptTrigger = 0;

}

This is telling the arduino that interrupt 0 on the arduino board is going to be configured as an external interrupt, and it will be triggered on a low value. When a low value is seen by the interrupt it will go to a subroutine called “interrupt.”

Make sure you read which pins on your arduino board can be configured as external interrupt pins. In the example above, this would be pin 2 as found on the Uno board. This is the **first parameter** within the brackets. The **second parameter** is the code function to call when an interrupt occurs. This must be a routine that takes no parameters and returns no value (ie…. void()….). The **third parameter** is which state the interrupt will trigger on.

The selections are: 1) LOW = The input is at a LOW state
2) RISING = The input state changes from LOW to HIGH
3) FALLING = the input state changes from HIGH to LOW
4) CHANGE = The input state changed from HIGH to LOW or LOW to HIGH, i.e. has changed its state

Below is an example of subroutine I used for one of my projects. During this routine I’m

Changing a variable from 0 to 1 (using the routine as a trigger). It is extremely important to

declare this variable as a volatile variable. This ensures that the compiler uses RAM to store

the variable instead of a storage register. The storage register can be temporarily

inaccurate if they are being modified outside of the main thread. Because of this, timing

functions should not be used within the routine as well.

volatile int InterruptTrigger;

void interrupt()

{

 InterruptTrigger = 1;

}

If at any time throughout the code you wish to disable or remove an interrupt. Which was

previously configured in the setup routine, you can do so by using “noInterrupts(),” or

“detachInterrupt(interrupt\_number).” However once this is done, you must

re-enable the interrupts by using “interrupts();” or “attachInterrupt(parm1, parm2, parm3)”

(if it was an external interrupt) the interrupt if you wish to implement that functionality.

Please be cautious when enabling / disabling interrupts. Serial communications work using

Interrupts, so if they are disabled you may lose the capability to communicate with other

Devices, also timing functions can suffer when using interrupts, so use them wisely.

You can also set up the ISR yourself instead of using Arduino’s pre-defined Interrupt

routines. To do this look at the datasheet for the appropriate micro and configure the

function register for the interrupt you wish to use. Also remember to declare the interrupt

routine….. eg. ISR(INT0\_vect) //“interrupt routine for Int 0.”

If you have anything to add to this document please let a technician know.