External Interrupts on MPIDE part 2: setIntVector()

by JayWeeks on October 23, 2015

Table of Contents

Author:JayWeeks
I build robots out of boxes! I love teaching what I've learned and seeing people add their own ideas to what they've learned. Nothing excites me more than seeing a student really take an idea and run with it!

Intro: External Interrupts on MPIDE part 2: setIntVector()

In my previous I'ble I taught you a very simple way to get external interrupts running on the DP32, uC32, and WF32*. Interrupts, however, are really complicated things, and while it's nice to have a simple function that takes care of everything for you, like attachInterrupt(), sometimes it's useful to have a little more control.

In this tutorial, I'm not only going to show you a different, more powerful way to set up external interrupts, I'm also going to teach you a little more about them and how they work.

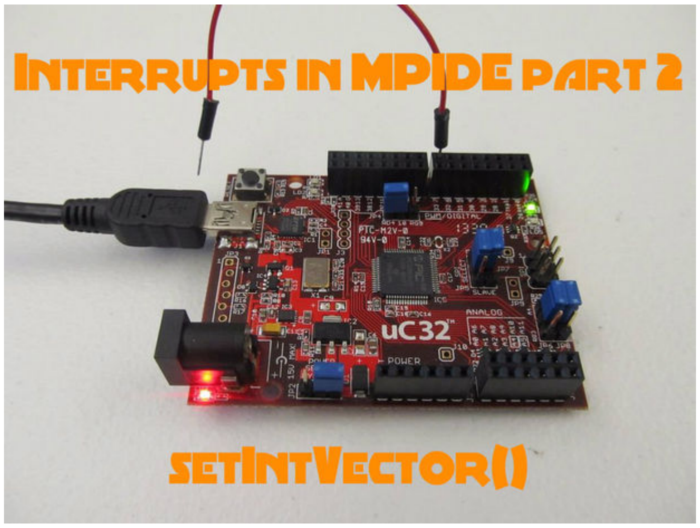
Let's get started!

* That tutorial will work with the other chipKIT boards as well, these are just the ones I showcased.

~~~~~
For more Instructables on building cheap robots, please check out the For Cheap Robots collection!

For more things that I've done, you can check out my profile page!

For more info from Digilent on the Digilent Makerspace, check out the Digilent blog!



## Step 1: Why Complicate Things?

In a little bit, you're going to see how much more complicated this method of using interrupts really is. Once you're used to interrupts, they're not so scary, but when you're starting out these can be very confusing. So why complicate things?

There are three reasons.

The first has to do with debouncing. I talked about debouncing in my previous tutorial, so I won't rehash it here. Very soon I'll be posting a tutorial that uses this more complicated method of using interrupts to debounce my input in an elegant and efficient way.

The second reason to use this method is speed. The attachInterrupt() function is slow (or so I've heard), and this method should be faster.

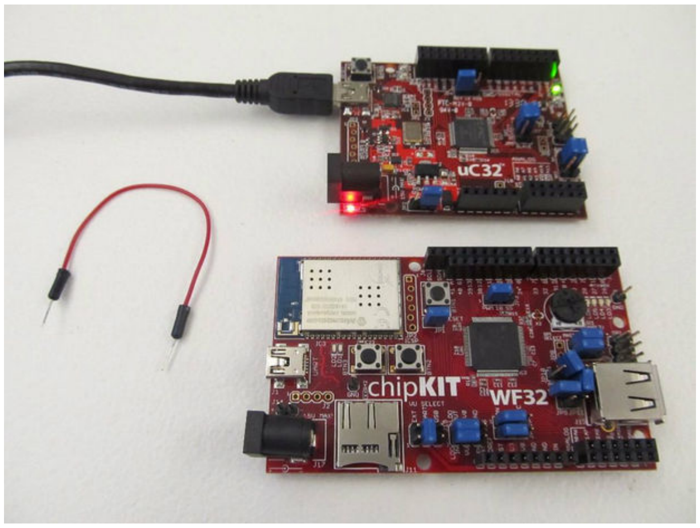
Finally, this method of using interrupts will help you understand more deeply what interrupts are and how they work.

## Step 2: What You'll Need

As with part 1, you're not going to need much for this tutorial, just a uC32 or WF32, a bit of wire, MPIDE, and an appropriate cable for programming your board.

"What?!" I hear you shout, sputtering in shock. "What about the DP32?" Yes, my friend I am shocked as well to be working on an I'ble that doesn't feature my stalwart companion, but for some reason the DP32 doesn't work with this particular code. I'm still trying to figure out why.

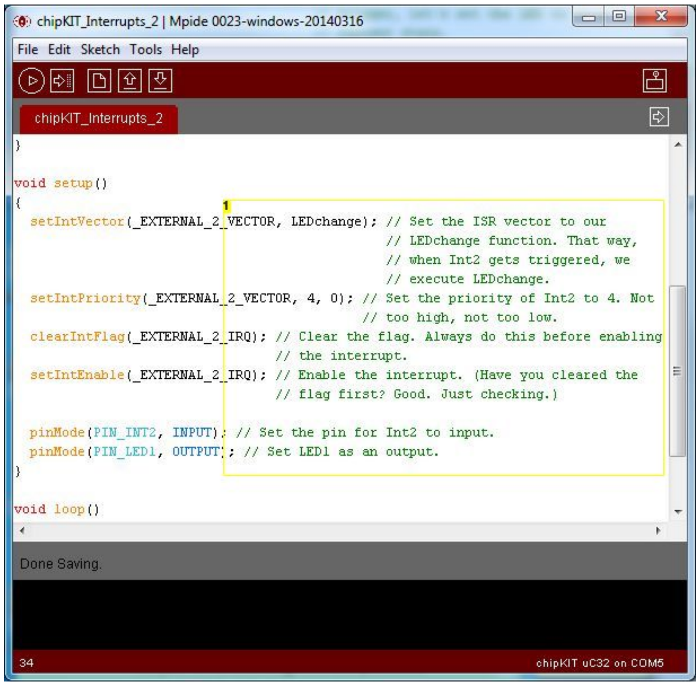
However! The uC32 and WF32 are still both very fine boards, and I'm happy to feature them!
~~~~~



Step 3: Diving Right In

Download the Interrupts_2 example code and load it onto your board.

Now, with a wire plugged into pin 7 on either the uC32, or WF32, touch the other wire to ground, you should see LED1 switch on and off. Neat! But how does it work? Let's pull it apart!


chipKIT_Interrupts_2 | Mpide 0023-windows-20140316
File Edit Sketch Tools Help
chipKIT_Interrupts_2
1
Done Saving.
34
chipKIT uC32 on COM5

```
}

void setup()
{
  setIntVector(_EXTERNAL_2_VECTOR, LEDchange); // Set the ISR vector to our
                                               // LEDchange function. That way,
                                               // when Int2 gets triggered, we
                                               // execute LEDchange.
  setIntPriority(_EXTERNAL_2_VECTOR, 4, 0); // Set the priority of Int2 to 4. Not
                                            // too high, not too low.
  clearIntFlag(_EXTERNAL_2_IRQ); // Clear the flag. Always do this before enabling
                                 // the interrupt.
  setIntEnable(_EXTERNAL_2_IRQ); // Enable the interrupt. (Have you cleared the
                                 // flag first? Good. Just checking.)

  pinMode(PIN_INT2, INPUT); // Set the pin for Int2 to input.
  pinMode(PIN_LED1, OUTPUT); // Set LED1 as an output.
}

void loop()
```

Image Notes
1. All this crazy nonsense will make sense in a bit. I promise.

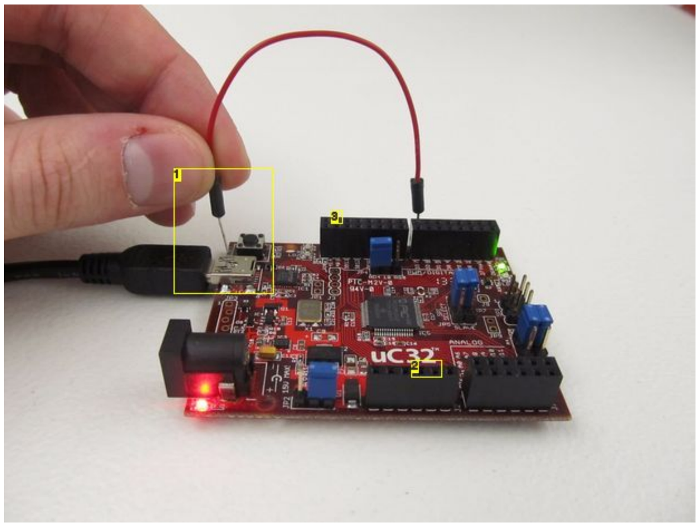


Image Notes
1. Personally I find it easier to tap the casing for the USB than to use the ground pins.
2. You can also use these two pins.
3. This is one of several ground pins on this board.

File Downloads

chipKIT_Interrupts_2.pde (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'chipKIT_Interrupts_2.pde']

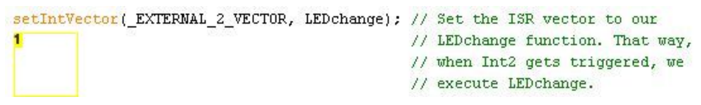
Step 4: SetIntVector()

We'll start with the setIntVector() function inside our void setup() function.

If you recall in part 1, part of attachInterrupt() to specify which function we wanted to execute when our interrupt got triggered. setIntVector() does exactly that, but it's a little different as you can see.

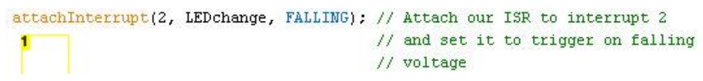
With attachInterrupt(), we specified the number of the external interrupt we wanted to use, but this code uses "_EXTERNAL_2_VECTOR". Obviously this refers to our external interrupt 2, but we could also use it to refer to other interrupts, and not even just our external ones.

Check out this example of using attachInterrupt() for internal interrupts, specifically the timer interrupts. It's also a great demonstration of why timer interrupts are useful and important.



```
setIntVector(_EXTERNAL_2_VECTOR, LEDchange); // Set the ISR vector to our
                                             // LEDchange function. That way,
                                             // when Int2 gets triggered, we
                                             // execute LEDchange.
```

Image Notes
1. This is the code from part 2.



```
attachInterrupt(2, LEDchange, FALLING); // Attach our ISR to interrupt 2
                                        // and set it to trigger on falling
                                        // voltage
```

Image Notes
1. This is the code from part 1. It's getting replaced by setIntVector(), along with some other functions.

Step 5: SetIntPriority(): What is Priority?

What in the world is an "interrupt priority" and why do we need to set it?

Let's say that the ISR (Interrupt Service Routine, if you recall from part 1) for Int1 is busy executing, but before it can finish, Int2 gets triggered, and tries to call it's own ISR (we'll call them ISR1 and ISR2, for convenience). What happens? Does ISR1 get put on pause while ISR2 executes, or does ISR2 have to wait until ISR1 is finished? The answer depends on the priority of Int1 and Int2.

Let's say that Int1 has a priority of 4, and Int2 has a priority of 5. In this case, because Int2 has a higher priority, ISR1 gets paused so ISR2 can execute. Then, once ISR2 is done, ISR1 resumes where it left off.

Let's switch it now so Int1 has a priority of 5, and Int2 has a priority of 4. In this case, because Int1 has the higher priority, ISR2 has to wait until ISR1 is done executing before it can start.

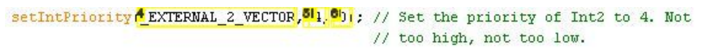
Priority levels range from 1 to 7, where 7 is the highest. Actually, there is also a priority level 0, but that is considered disabled, and will never execute. In the case of our program, we set the priority of Int2 to 4.

(Notice how we're still using "_EXTERNAL_2_VECTOR"? That'll be important in a bit.)

"Wait..." I can hear you thinking (using my mind-reading powers), "what happens if two interrupts with the same priority are triggered?"

Good question! In our previous examples, if both Int1 and Int2 have a priority of 4, then ISR2 will always wait for ISR1 to finish executing before it can start.

But wait! There's more!



```
setIntPriority(_EXTERNAL_2_VECTOR, 4, 0); // Set the priority of Int2 to 4. Not
                                          // too high, not too low.
```

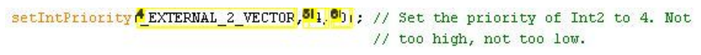
Image Notes
1. Interrupt priority.
2. Our interrupt vector.
3. Interrupt sub-priority.
4. Our interrupt vector.
5. Interrupt priority.
6. Interrupt sub-priority.

Step 6: SetIntPriority(): Sub-priority

Now, let's say that Int1 has a priority of 5, and Int2 is triggered with a priority of 4. Then, while Int2 is waiting for ISR1 to finish executing, suddenly Int3 is triggered, and it has a priority of 4 as well!

That means that now both Int2 and Int3 are waiting for ISR1 to finish, but when it does, which one executes first? That's where sub-priorities come in! If the priority of both interrupts in a list have the same priority, then the interrupt with the highest subpriority will execute first.

This is something that will probably rarely come up, but it's important stuff to know as you get more and more interrupts in your code! Make sure to keep your priorities straight!



```
setIntPriority(_EXTERNAL_2_VECTOR, 4, 0); // Set the priority of Int2 to 4. Not
                                          // too high, not too low.
```

Image Notes
1. Interrupt priority.
2. Our interrupt vector.
3. Interrupt sub-priority.
4. Our interrupt vector.
5. Interrupt priority.
6. Interrupt sub-priority.

Step 7: ClearIntFlag() and setIntEnable()

Up to this point, we've talked about interrupts being "triggered" and jumping straight to their ISRs, without paying much attention to how that jump happens.

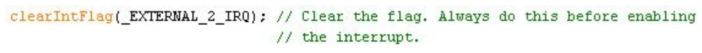
When an interrupt is triggered, a very specific bit in memory, one associated with that interrupt, is flipped from 0 to 1. This tells the chip that that interrupt has been triggered, and will need its ISR executed. This bit is called an interrupt "flag".

The thing about interrupt flags is that they can be triggered regardless of whether the interrupt is active or not. In fact, during normal operation of your board, you may have been triggering interrupt flags without knowing it. The only reason this doesn't execute an ISR is because the interrupt hasn't been enabled.

Before we enable an interrupt, we always clear it's flag. If we didn't it would be pretty much guaranteed that the ISR would be executed immediately after the interrupt was enabled. That's why clearIntFlag() is always* called before setIntEnable().

Finally, notice how both clearIntFlag() and setIntEnable() use "_EXTERNAL_2_IRQ" rather than "_EXTERNAL_2_VECTOR"? This is because you're referencing different places in memory. The "IRQ" stands for "Int Request", which is another way of referring to the interrupt flag.

* I'm sure there's some fringe scenario where you don't want to clear the flag before enabling the interrupt, but we're not going to worry about that for now.



```
clearIntFlag(_EXTERNAL_2_IRQ); // Clear the flag. Always do this before enabling
                               // the interrupt.
```

Step 8: Interrupt Service Routine

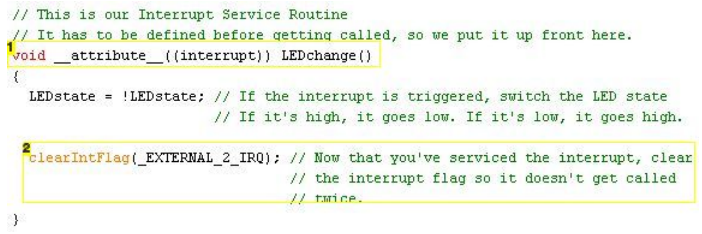
When we're using setIntVector(), we have to write our ISRs a little differently from the one we wrote for attachInterrupt().

Firstly, the ISR has to be defined before you can use it. That means putting it up front before the rest of your code.

Then, the actual declaration is different. This time, we have to specify that the function we are declaring is an interrupt, whereas before we could just declare any old function.

Finally, we absolutely have to remember to use clearIntFlat() at the end of this ISR. Normally, attachInterrupt() takes care of clearing the flag after the ISR has run, but we're doing everything ourselves with this code so we have to clear the flag ourself as well.

If you don't end your ISR with clearIntFlag() (or at least, don't use it somewhere in your code) then the flag for that interrupt will always be triggered, and the interrupt will just re-activate after it's finished, getting your code stuck in an infinite loop.



```
// This is our Interrupt Service Routine
// It has to be defined before getting called, so we put it up front here.
void __attribute__((interrupt)) LEDchange()
{
  LEDstate = !LEDstate; // If the interrupt is triggered, switch the LED state
                        // If it's high, it goes low. If it's low, it goes high.

  clearIntFlag(_EXTERNAL_2_IRQ); // Now that you've serviced the interrupt, clear
                                 // the interrupt flag so it doesn't get called
                                 // twice.
}
```

Image Notes
1. This is different from how we declared our ISR when we used attachInterrupt().
2. Gotta remember to clear the flag too.

Step 9: 1, 2, skip a few, 99, Where to Find Your Interrupts!

In my previous tutorial, I covered volatile variables and what's going on in the main loop, so if you're unfamiliar with those I'd recommend re-reading them there. That way I can skip right on ahead to where to find information on how to use interrupts other than Int2.

Working with chipKIT Interrupts (attached) is a really good guide on what interrupts are and how they work. If anything I've said here is confusing, you can probably get a better explanation there.

On page 7, they list all the vector names for the PIC32MX family of microcontrollers (these are what the DP32 and WF32 use). Starting on page 9, they list all the interrupt request names. Remember! Vector names are used with setIntVector() and setIntPriority(). Interrupt request names are used with clearIntFlag() and setIntEnable().

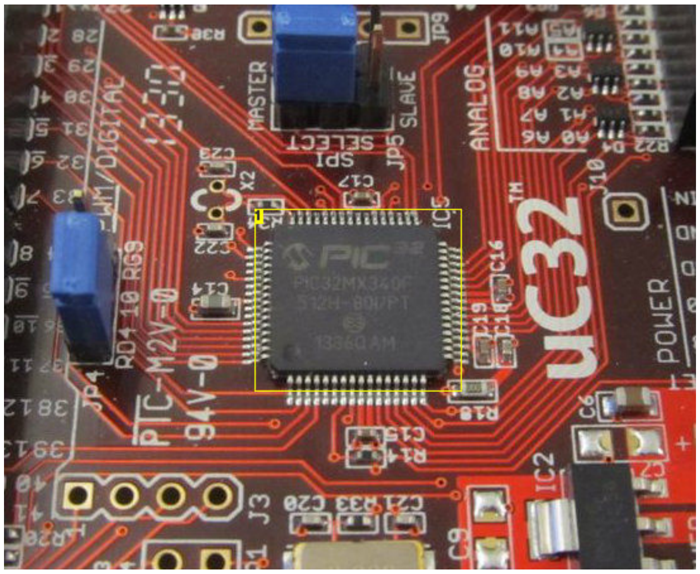


Image Notes
1. PIC32MX340F512H

File Downloads

Working with chipKIT Interrupts.pdf (261 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'Working with chipKIT Interrupts.pdf']

Step 10: Where to Go from Here?

Like I mentioned before, Interrupts Made Easy with chipKIT is a good example of using this more complicated method of controlling interrupts to set up a timer interrupt. (Strangely enough, this code does work with the DP32. I'll update this tutorial if I ever figure that one out.)

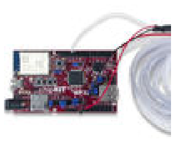
The Working with chipKIT Interrupts PDF I attached earlier is another great resource for understanding how this code works.

I'm also not done with these interrupt tutorials myself. Soon there will be two more tutorials that use debouncing to make these interrupts more usable for hardware purposes. Then I'll finally be able to show you this mysterious project involving my Metal Wheels for Cheap Robots tutorial that I've been hinting at for so long!

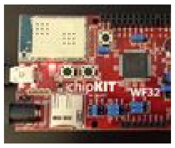
I'm super excited you guys!

Related Instructables

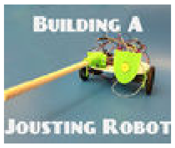
External Interrupts on MPIDE part 1: attachInterrupt() by JayWeeks


Using LabVIEW LINX and chipKIT WF32 to Control an LED Strip by Sudharsan Sukumar

Using the MPIDE Board-Defs by JayWeeks


Display Weather and Location Using chipKIT WF32 and LabVIEW by Sudharsan Sukumar

Behind You! Anti Creeper Alert System by Sudharsan Sukumar


Jousting Robot (Wiring Tutorial) by Sudharsan Sukumar

Comments