JonnyMac Posts: **7,170**
2020-12-17 - 11:37:34 edited 2020-12-17 - 12:20:20 Flag

> Do we need to initialize the pins, I heard someone say that with the new Pin(functions) the pins are automatically set to outputs?

It may have been me during one of my prosthelyzing posts. The pinX() functions in Spin2 actually do three three things:
1) Select the correct IO registers (A or B)
2) Make pin(s) inputs or outputs as required
3) Write to or read from the desired pin(s).

To make a pin high in the P1, we would do this:

```
  dira[pin] := 1
  outa[pin] := 1
```

In the P2, this handles it:

```
  pinhigh(pin)
```

We only have 32 pins on the P1, but 64 on the P2. While we *can* directly access the IO registers, I don't recommend it because programs change. When I used to design products for EFX-TEK, I would never connect external circuitry to the Propeller in my schematic. When the PCB layout guy was finished, he had decided the best pins to use for the layout -- all I had to do was update my pin constants in the listing to move test code to the final product. This happens all the time (pins moving), which is one (of many) reasons why embedding pin #s into our code is a bad idea.

In the P1, we could do this:

```
  outa[MSB..LSB] := value
  dira[MSB..LSB] := %1111
```

In the P2, we do it like this:

```
  pinwrite(LSB addpins (MSB-LSB), value)
```

There is a caveat, however. In the P1, we can also do this:

```
  outa[LSB..MSB] := value
```

...which will reverse the output order of the lower four bits in value. How can we do this in the P2?

```
  pinwrite(LSB addpins (MSB-LSB), value rev (MSB-LSB))
```

Okay, this is not quite as friendly as the P1 version, but -- thankfully -- is an infrequent requirement.
It might be tempting to do this:

```
  outa.[LSB..MSB] := value
```

...but this doesn't work. The rightmost value in the .[ ] syntax is used as the LSB.

> Are the old pin registers (dira, outa, ina) necessary?

In Spin2, I don't see any need for direct IO register access.

> Is everything I did wrong, or are there compiler issues? Mainly I am curious why many pin initialization worked (or at least compiled) like outa[7] and outa.[7], and others did not.

Things have changed from the P1 to the P2. This syntax:

```
  outa[7]
```

...is actually referring to register $003 in the cog. Why? In P2 syntax, **outa** is just another register and anything in [ ] is an offset. The address of **outa** is $1FC. Add 7 and you get $203 which is an illegal address because the max is $1FF, hence it wraps around to $003.

I think you see now why I have become a bit adamant about NOT using IO registers in Spin2 code -- let the Spin2 instructions do the work for you.

> Also the difference on using ' or ' ' for comments. It worked on the P1, but I am not sure it works on the P2. The new ide displays those comments in 2 different shades?

Single and double apostrophes are both legal line-comment starters. The difference is in the view when using Propeller tool. There is a radio-button a the top marked Documentation. In this mode, only the comments with double apostrophes or double block comments {{ }} will be displayed.

> One other question. In many other P1 programs the pin#'s were passed from the demo object. Is there any reason why I should not do this (pass the pins) with a P2 program?

It is my (very strong) opinion that one should always craft objects that accept pins from the calling applications. Again, pins move from application to application, and hard-wiring pins is a bad idea. The exceptions, of course, are the fixed pins for programming and application memory (EE or flash). This is why my code calls out fixed IO pins versus application IO pins.

Code Reviews:
-- **Blink_v01** : The syntax you're using causes the IO registers to be treated as long arrays, not a single array of 32 bits. Again, use the pinX() functions.
-- **Blink_v02** : Mixed syntax with IO registers that Spin2 doesn't support.
-- **Blink_v03** : Same issue with registers in setup(). The main() method never runs because it is not called. Have a look at my template; setup() is called from main().
-- **Blink_v04** : Again, main() is never called. Note: Spin is not like C that looks for main(). In Spin, the first method runs (this is why I place main() at the top).
-- **Blink_v05** : Again, main() is never called.

Important notes:
-- It is best NOT to use IO registers directly in Spin2; the underlying code for the high-level instructions is very efficient and fast.
-- In Spin, the first method in the listing runs and controls the program. The code does not care about its name, nor go looking for anything with a specific name.

Side note. This blinker code works, but is tedious to write versus using the new Spin2 functions for IO.

```
  dirb.[56 & $1F] := 1

  repeat
    outb.[56 & $1F] ^= 1
    waitms(100)
```

It's worse if you want to do multiple pins manually. Note the use of addbits versus addpins due to the nature of the syntax.

```
  dirb.[(56 & $1F) addbits 3] := %1111

  repeat
    outb.[(56 & $1F) addbits 3] := getrnd()
    waitms(100)
```

To access bits with in a register we have to use the .[] notation. Registers are only 32 bits, so we have to use & $1F with the **B** IO registers. Again tedious. Don't do it.

Final comment about pin groups. They only work within one register. What happens if you do this?

```
  BUS = 30 addpins 7
```

The Propeller will create a group that will impact pins 30, 31, 0, 1, 2, 3, 4, 5 -- which is probably not what we want. I was just writing multi-channel LED driver for the P2 that looks for these boundary issues. It's something to be careful with.