I2C or SPI - Which is better?

PropGuy2 Posts: **122**

January 2013 edited March 2013 in General Discussion **0**

When using the Propeller USB project board which is better / easier to use: I2C or SPI ? I have used both and each has advantages / disadvantages - But I would like to use one or the other to reduce the code size of my project(s) and still get reasonable performance. Does the Propeller chip hardware and / or Spin language favor I2C or SPI ?

Comments

13 Comments sorted by Date Added Votes

Leon Posts: **7,546**

January 2013 edited January 2013 **0**

SPI is faster and the software is a lot simpler. I2C uses fewer pins - only two even if you have several devices to interface.

Leon Heller
G1HSM

Mark_T Posts: **1,522**

January 2013 edited January 2013 **0**

Its not a very meaningful question - either can be better for a particular purpose. They have many aspects so there is no
simple "better than" relation that can be defined. I2C allows many devices to share a fixed 2 pin bus, SPI allows
simpler software/hardware support and can go faster, but takes 3+N pins to talk to N devices (and many devices

are "not quite SPI").

SPI is bad in that without pull-up resistors on every CS line you can't guarantee clean power-up behaviour, yet
pull-up resistors are not specified and hardly ever used. I2C specifies pull-up resistors on both SDA and SCL
but not everyone suggests the same values (or even uses them, Parallax). Doing I2C properly can mean 5V
and 3.3V devices can share the same bus in some circumstances, SPI needs level translation.

SRLM Posts: **5,032**

January 2013 edited January 2013 **0**

> Mark_T wrote: »
> ...(and many devices are "not quite SPI").

And to expand on what Mark_T said, many I2C devices are "not quite I2C". For I2C, all devices share the following concepts*:
1. Start and stop signals
2. First byte after a start is an address, with the lowest bit indicating read/write
3. For each slave, if the first byte isn't "my" address then ignore everything until stop.
4. Each byte is acknowledged (either by the slave or master).
*For the normal 7 bit addressing in single master, without all those I2C extensions that I've never used...

After that, it's a free for all. Most devices follow the "register" concept, with a very similar series of bytes to do an operation. I call this the ST I2C standard. But some devices deviate, such as:
1) MS5611 barometer (it doesn't use a register protocol).
2) EEPROMs (it uses a 2 byte address, and 128 byte pages).
3) PCF8523 (it requires a stop condition before the repeated start condition when reading bytes).

I can't speak for SPI, since I haven't really used it that much.

http://srlm.io

bee_man Posts: **109**

January 2013 edited January 2013 **0**

I prefer I2C since the prop already has 2 pins dedicated to that use. As said earlier, If you use SPI it is 2+N pins plus the 2 I2C for the EEPROM, so you are limited to the number of devices you can use. To answer your question, I would say the hardware favors I2C.

LoopyByteloose Posts: **12,537**

January 2013 edited January 2013 **0**

From the new learner's point of view, SPI is the older and more generic interface. I2C is owned by Phillips. They are both related to simple shift register output and input. Try to consider the group as one topic - synchronous serial i/o.

So if you want to read or learn assembler code, start with SPI where the input and output wires are not shared. Then you can read the I2C and learn to deal with all the problems of toggling our i/o pin from output to input. That is also why SPI is simply faster... less to manage.

You are always going to run into the odd device that refuses to pay an I2C royalty and requires SPI. And it might even create its own modified SPI that uses one i/o pin, but does not follow I2C standards. So one needs to learn to be comfortable with both.

Hwang Xian Shen, Puddleby-on-the-Marsh.

All things considered, I can live and thrive without Microsoft products. LINUX is just fine.

Bits Posts: **414**

February 2013 edited February 2013 **0**

> bee_man wrote: »
> I prefer I2C since the prop already has 2 pins dedicated to that use. As said earlier, If you use SPI it is 2+N pins plus the 2 I2C for the EEPROM, so you are limited to the number of devices you can use. To answer your question, I would say the hardware favors I2C.

I would not state this so bluntly - the propeller has no dedicated i2c pins that I am aware of seeing how any one of the pins work as an i2c protocol.

I also prefer the i2c over SPI, but really depends on a multitude of factors.

[h=2]I would love to change the world, but they won't give me the source code![/h]

Duane C. Johnson Posts: **955**

February 2013 edited February 2013 **0**

Hi Bee

> bee_man wrote: »
> I prefer I2C since the prop already has 2 pins dedicated to that use. As said earlier, If you use SPI it is 2+N pins plus the 2 I2C for the EEPROM, so you are limited to the number of devices you can use. To answer your question, I would say the hardware favors I2C.

Technically the prop doesn't have "dedicated" I2C pins.
However, most use Pins[28..29] for use with EEProms.
So, this is like no extra pins are consumed for I2C devices other than those already used.

Duane J

User Name Posts: **1,298**

February 2013 edited February 2013 **0**

On a recent project, I implemented a fast SPI and fast I2C port on the same Prop. My experience was exactly as Leon mentioned above. The SPI was done in no time, was really simple, and transferred data very quickly. With I2C, the spec seemed ambiguous with respect to when to sample the data line. Had to write the code and then empirically determine what clocking kept the slave happy. Seemed a little rinky-dink.

Never far from a ski area, NW USA.

davejames Posts: **3,873**

February 2013 edited February 2013 **0**

...interesting that this discussion pops up when I'm going through an evaluation of the two protocols in prep for building another customer training class.

I'm familiar with I2C, but not SPI so I dug up the following that may help PropGuy2. In fact, on "page" 6 (toward the end), there is a topic "I2C vs SPI: Is There A Winner?"

Enjoy!

http://www.byteparadigm.com/kb/article/AA-00255/0/Introduction-to-SPI-and-IC-protocols.html

Well-written documentation requires no explanation.

localroger Posts: **2,744**

February 2013 edited February 2013 **0**

I have done PASM drivers for both interfaces, including an I2C slave.

SPI is faster but gives no indication that anything is wrong until a whole block of data has been transferred. SPI can be faster because it was designed for a fundamentally different challenge than I2C; SPI was never intended to transmit signals over distance, while I2C was designed to withstand some capacitance in wires taking the signal from the processor controlling your DVD player to the display panel in front.

I2C gives instant warning that something isn't right as the receiving device

must ACK (by pulling the signal line down itself) after each byte is transmitted. (Some drivers ignore this but it's good practice to check, and impossible with SPI.) It's already been mentioned that I2C needs only two pins for a bus serving many devices; SPI needs a separate enable pin for each device although the clock and signal pins can serve as a multi-device bus.

Unless you're programming in PASM the speed difference is not a factor; you can't max out either interface in Spin. (It could be a factor if you're using an object written by someone else.) Unless you're implementing an external memory it's unlikely that the speed difference will be a factor even then. I do consider it an advantage that two Prop pins are intended (if not required) for use by I2C; as with the programming serial pins it means that nearly every design ends up with I2C implemented on the EEPROM pins, and once the Prop is finished loading the EEPROM further use is a natural extension.

PropGuy2 Posts: **122**

March 2013 edited March 2013 **0**

So - I have finished my project and I have to say SPI is definitely the way to go. I used several SPI chips, ADC and RTC, along with the DataLogger module. No problems - but read(understand) and then code to the Spec sheet(s) chip select and clock pulse sequence, for the device you are using.

Kudos to SPI code examples by Beau Schwabe (Parallax) Study this code and you will be a SPI expert in no time.

robin jack Banned Posts: **4**

March 2013 edited March 2013 **0**

I think SPI is better for you because its a prettey easy and much simpler according to my experience on the other end I think I2C is also fine but comparing to SPI this is not good but any way you have a both option so I think you should tray SPI first. Its my personal suggestion for you.

make your color

Sign In or Register to comment.