Poster Abstract: I^2C Considered Wasteful

Saving Energy with Host-Controlled Pull-Up Resistors

Birte Friesel
birte.friesel@uos.de
Osnabrück University
Osnabrück, Germany

Olaf Spinczyk
olaf@uos.de
Osnabrück University
Osnabrück, Germany

ABSTRACT

The Inter-Integrated Circuit (I^2C) bus is frequently used to connect sensors and actuators to cyber-physical systems. It is designed around always-on pull-up resistors, which transform valuable electric energy into heat whenever a 0-signal is sent or received. Using a software I^2C implementation which disables pull-ups when possible, we decrease the energy demand of I^2C transmissions at the cost of additional CPU time. On a low-power MSP430FR5969 microcontroller, we observe 10 to 50 % lower whole-system energy usage per transmission compared to conventional software I^2C. An advantage over hardware I^2C modules is only apparent at bus clocks below 10 kHz.

KEYWORDS

embedded systems design, energy models, energy saving, i2c bus

ACM Reference Format:

Birte Friesel and Olaf Spinczyk. 2019. Poster Abstract: I^2C Considered Wasteful: Saving Energy with Host-Controlled Pull-Up Resistors. In *The 18th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2019) (IPSN '19), April 16–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3302506.3312606

1 INTRODUCTION

The Inter-Integrated Circuits (I^2C) bus is one of the most common interfaces for communication between system components. It connects a master (usually a microcontroller) with up to several dozen slave devices (sensors, actuators, and/or storage) by means of two bidirectional wires: serial data (SDA) and serial clock (SCL) [2].

Transmissions are byte-oriented and clocked by the bus master; clock frequency is limited to 100 kHz in standard mode. Each transmission starts with a 7-bit slave address plus read/write bit, followed by an arbitrary number of data bytes. After each byte (both address and data), there is a ninth clock cycle providing an ACK/NAK slot which is filled by the communication partner.

On the wire, SDA and SCL are pulled HIGH by pull-up resistors (R_p) connected to VCC (see Fig. 1). A HIGH signal is sent by doing nothing, and a LOW signal by pulling the line to GND, causing a current to flow through the corresponding pull-up. Therefore, in addition to the energy spent in CPU and peripheral device, each I^2C transmission consumes energy in the pull-ups. Although this has been noted before [1], to our knowledge, no attempt at minimizing pull-up usage has been published so far. Yet, especially for battery-less energy harvesting devices, every saved microjoule counts.

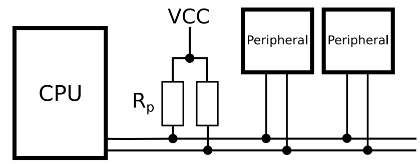

Figure 1: A typical I^2C bus layout.

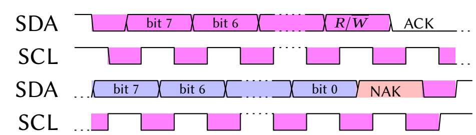

Figure 2: Master-controlled periods during an I^2C write (tinted blue/magenta) and read (tinted red/magenta).

2 REDUCING TRANSMISSION COST

During each clock cycle, SCL is low for half the time and SDA is either high (1-bit/NAK) or low (0-bit/ACK). The amount of energy lost to pull-up resistors in this time range depends on resistance R_p, bus voltage U, and bus frequency f:

$$E_{SCL,*} = \frac{U^2}{2R_p f}; \quad E_{SDA,0} = \frac{U^2}{R_p f}$$

A transmission with n data bytes uses $9n + 9$ clock cycles. It contains n ACK signals and up to $8(n + 1)$ 0-bits in address and data, giving a total transmission cost in the following range:

$$E_{SCL} = (9n + 9) \cdot E_{SCL,*}$$

$$n \cdot E_{SDA,0} \le E_{SDA} \le (9n + 8) \cdot E_{SDA,0}$$

As shown in Fig. 2, SCL is always controlled by the master[1], and SDA for 89 % of the time for I^2C writes ($8n + 8$ cycles) and up to 50 % of the time for reads ($n + 8$ cycles). In the general case, the master is responsible for the majority of energy lost in pull-ups.

We therefore propose connecting the pull-ups to the bus master instead of VCC and disabling them while it is transmitting a low signal on the corresponding line. This way, no energy is lost to pull-ups during this time: E_{SCL} becomes zero, and E_{SDA} is reduced by up to 89 % depending on data direction and the amount of 0-bits.

[1] Unless a slave device employs *clock stretching*, which causes additional pull-up energy consumption. It can often be remedied by lowering the bus frequency.

IPSN '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6284-9/19/04.
https://doi.org/10.1145/3302506.3312606

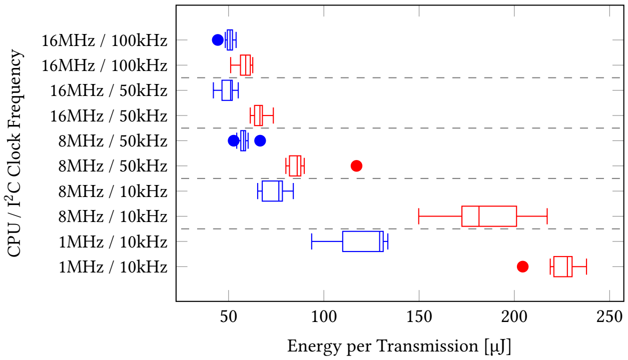

Figure 3: Software I^{2}C with (blue/top) and without (red/bottom) pull-up control. Each transmission consists of a 1-byte write followed by a 2-byte read.

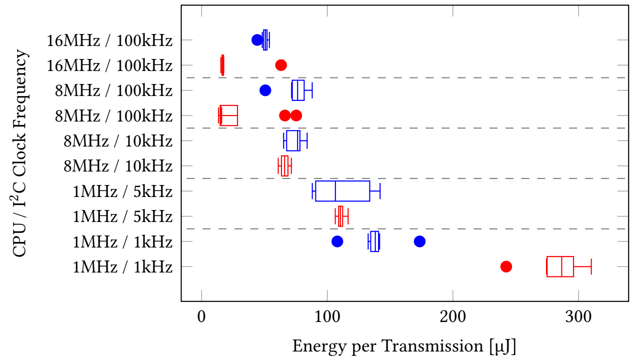

Figure 4: Software I^{2}C with (blue/top) and hardware I^{2}C without (red/bottom) pull-up control. Each transmission consists of a 1-byte write followed by a 2-byte read.

The only requirements are two additional master output pins (one per resistor) and an adjusted I^{2}C implementation. With proper timing, disabled pull-ups are transparent to slave devices and do not violate the I^{2}C specification.

3 EVALUATION

The net amount of energy saved by this approach depends on the microcontroller and I^{2}C implementation in use: although E_{SDA} and E_{SCL} are decreased, the energy consumed by the CPU is increased due to additional CPU cycles required for pull-up control.

I^{2}C can be implemented in software and in hardware. Software I^{2}C (also known as *bit-banging*) requires the CPU to wake up at least twice per clock cycle and read/set SDA and SCL as appropriate. If the microcontroller contains an I^{2}C hardware module, it can handle the protocol and send/receive single bytes autonomously; this typically requires just one wakeup per byte.

Adjusting a software implementation is easy: whenever toggling SDA/SCL between high impedance and output low, toggle the corresponding pull-up as well. This introduces almost no overhead. The functionality of hardware modules, on the other hand, cannot be altered. Unless developers are designing a custom microcontroller and willing to add this feature, pull-up control is only possible by switching to a less energy-efficient software implementation.

To confirm that master-controlled pull-ups are more efficient than conventional software I^{2}C, and assess whether software I^{2}C with pull-up control is more efficient than hardware I^{2}C with always-on pull-ups, we implemented all three variants on an ultra-low-power MSP430FR5969 microcontroller connected to an LM75 I^{2}C temperature sensor. We measured the energy consumed by the entire system (controller, pull-ups, and sensor) while varying bus clock, CPU frequency, and data direction.

Typical I^{2}C implementations use 1.8 V and pull-ups between 1 and 2.2 kΩ. Since our evaluation board only provides a fixed voltage of 3.6 V, we used 3.9 kΩ pull-ups, which consume nearly the same amount of power as 1 kΩ resistors at 1.8 V.

For software I^{2}C, results confirm that manual pull-up control reduces transmission cost by 10 % at 100 kHz and up to 50 % at lower clock rates, regardless of CPU frequency (see Fig. 3 for an excerpt). As the two software I^{2}C variants have nearly identical source code, we expect similar results on other microcontrollers. If software I^{2}C is used anyway, we therefore recommend implementing pull-up control.

In case of hardware I^{2}C, both theoretical and empirical results indicate that it is more efficient than either software I^{2}C implementation. For typical transmissions at 100 kHz, the amount of energy saved by pull-up control ($E_{SCL} + E_{SDA}$) is lower than the additional energy consumption caused by more frequent CPU wakeups.

In our case, this only changes at bus frequencies below 3 to 10 kHz (see Fig. 4), which are rarely used in practice. So, unless a slave device requires unusually low frequencies, hardware I^{2}C should be used where possible. However, as the efficiency of software I^{2}C depends on the efficiency of the microcontroller itself, this is not a general result.

4 CONCLUSION

We have presented a simple, but effective, method to decrease the energy lost in pull-ups during I^{2}C transmissions, giving a net energy saving of 10 % per transmission for typical software I^{2}C implementations. Implementing this method in hardware I^{2}C modules should reduce energy use even more, however, this is only feasible when designing custom hardware, e.g. when working with FPGAs.

Although the wide availability of energy-efficient hardware I^{2}C modules limits the immediate practical impact of this result to microcontrollers without hardware I^{2}C support (such as ESP8266 Wi-Fi SoCs), we think that it makes an important point. It pays off to analyze all components of an embedded system, even if they are several decades old and taken for granted. Energy saving may be possible using simple and, in retrospect, obvious adjustments.

REFERENCES

[1] A. K. Oudjida, M. L. Berrandjia, R. Tiar, A. Liacha, and K. Tahraoui. 2009. FPGA implementation of I2C & SPI protocols: A comparative study. In *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*. 507–510. https://doi.org/10.1109/ICECS.2009.5410881
[2] NXP Semiconductors. 2012. *I2C-bus specification and user manual.* Rev. 5.