

HDQ Protocol Implementation with MSP430

Andreas Dannenberg

MSP430

Introduction

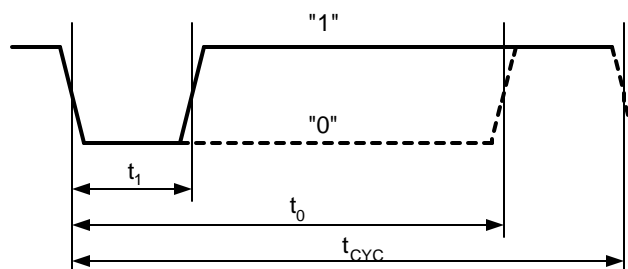
Most battery monitor ICs from TI, such as the bq2019 and bq26500, include a single-wire serial data interface (HDQ bus). Host controllers can use this interface to access various on-chip registers to read-out battery capacity, voltage, and other parameters. The purpose of this application note is to present a solution for interfacing HDQ-enabled battery gas gauge devices with the MSP430 microcontroller family.

HDQ Basics

The HDQ bus is a master-slave bus system using a simple one-wire, asynchronous, bi-directional, serial interface with a bit-rate of about 5-Kbit/s. The bus line is driven by open-collector devices and therefore requires an external pull-up resistor. The relatively slow bit rate is sufficient for reading out as well as setting registers in battery monitor ICs. The host may only need to communicate at infrequent intervals to update the user with the latest runtime computation, thus minimizing communication and saving power.

Data is always transmitted bit-by-bit in blocks of 8-bits with the LSB first. The bits are encoded as shown in Figure 1. Every bit always starts with a high-to-low transition of the HDQ bus line. The signal returns to high after time t_1 if the bit is a one and returns to high after time t_0 if the bit is a zero. The bit cycle time t_{CYC} is typical in the range of 190 μ s. Please see the datasheet of an HDQ enabled gas gauge device for the exact timing specifications [1].

Figure 1. HDQ Bit Timing



The protocol is command-based and data is transferred in blocks of 2 bytes. The first byte is always sent by the host (master) and contains the client register address (7-bit). It also contains the R/W-bit, which determines if the next byte is sent by host to the client (R/W = 1) or read from the client by the host (R/W = 0).

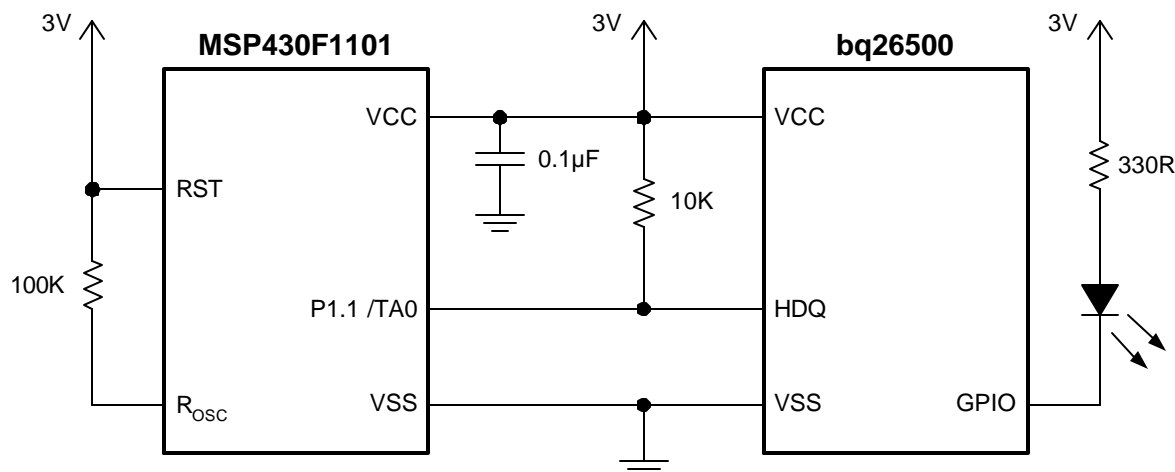
MSP430 HDQ Implementation

This section describes how an MSP430 microcontroller can be interfaced with HDQ-enabled devices. As an example, the interface to the bq26500 single-cell Li-Ion and Li-Pol battery gas-gauge IC is presented. The basic demo application reads out the on-chip temperature sensor and provides feedback about temperature changes using an LED.

Hardware Description

Figure 2 shows a typical HDQ system consisting of an MSP430F1101 used as a host controller and a bq26500 gas-gauge IC used as a slave device. The selected MSP430 device is a 20-pin, low-cost MSP430 family member. More information on this device can be found in the device data sheet [2]. Any other MSP430 derivatives could be used if more functionality is required on the application side.

Figure 2. MSP430 HDQ Example Schematic



To generate the HDQ bus timing, the Timer_A module of the MSP430 is used. In particular, the capture/compare block 0 (CCR0) is used for generating and sampling the bus signals with hardware support. The associated MSP430 pin, TA0, is brought out via port pin P1.1 and connected to the slave device. A 10-kohm pull-up resistor is provided as the HDQ bus is operated in high-impedance mode. Additionally, a 100-kohm resistor between the MSP430 R_{osc} pin and VCC is connected to stabilize the internal DCO clock generator and achieve a system clock with a very low temperature and voltage drift. However, in a user's application, other ways of providing a stable clock to the MSP430 could be used such as an external 32-kHz watch crystal which would also enable the MSP430 to implement an ultra-low-power real-time clock function.

For demonstration purposes, the bq26500 device is used as a LED-driver only. The GPIO pin is switched to output direction and operated as an open-drain output. Basic HDQ communication functionality can now be shown using this LED. Please refer to the bq26500 data sheet for detailed application information on using this battery gas-gauge IC [1].

Software Description

The demonstration software for this application report consists of the communication library HDQ.c / HDQ.h and an application using this library (bq26500_LED.c). The communication library is also provided in assembly language with the equivalent function (HDQ.s43). It makes use of the same C calling conventions as HDQ.c. By removing HDQ.c and adding HDQ.s43 to the project file, the assembly version of the code can be used. Only one of these files must be included in the project, otherwise the build results in symbol conflicts. The Assembly version ISR is executed faster (11% to 48%, depending on the function) than the C version even with the highest optimizer settings. This is mainly due to the use of efficient calculated branches in the Assembly interrupt service function opposed to the full implementation of a C language switch/case statement.

HDQ Communication Library

This library offers complete communication to an HDQ enabled device, such as the bq26500 battery gas gauge IC. The entire host communication is implemented in software and is supported by the Timer_A hardware module that can be found on all MSP430 devices. It also uses interrupts and low-power modes to minimize current consumption. Due to the Timer_A support, interrupt latency introduced by other interrupts that are being serviced and are blocking the CPU is not critical. The Timer_A0_ISR() function must be serviced before the next capture or compare event occurs.

The software library can be adapted to meet the requirements of a particular application. For example, the user is free to select a different clock source or frequency for sourcing the Timer_A module. In this case, the timing constants as defined in the header file must to be modified. For convenience, all the timing constants used are derived from one timer clock frequency definition ClkFreq. The entire bit timing can be re-calculated by changing this definition.

By adding either the C (HDQ.c) or Assembly (HDQ.s43) code file to the project and including the header file HDQ.h into a program, the following three functions can be called:

```
void HDQSetup(void)
```

Calling this function performs a basic setup and that is needed for the data transfer. It sets port pin P1.1 used for the HDQ bus line connection to input and enables the Timer_A.CCR0 function for this pin.

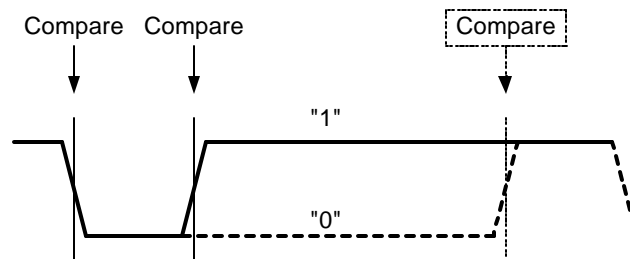
```
void HDQWrite(unsigned char Addr, unsigned char Data)
```

The HDQWrite() function is used to write data to a device that is connected over the HDQ bus. HDQ slave devices are usually internally organized as a block of registers. The parameter Addr contains the 7-bit address (0 – 127) of the register to write to, and the Data parameter the 8-bit data to transfer. After starting the Timer_A module, an HDQ break condition is sent by calling the local function HDQBreak(). This is to ensure that the slave device HDQ engine will be reset for the case the chip has already seen a start of communication due to contact bouncing during battery insertion.

In the next step, the 7-bit register address is transmitted to the slave device with the R/W bit set to one. This will make the slave device receive the second byte (data byte) after a short delay. The local function HDQBasicWrite() is used for sending out data words.

Figure 3 shows how the Timer_A is used to generate the HDQ signal waveform. The CCR0 block is used in compare mode only and setup to toggle the HDQ signal on every compare event. An interrupt is generated for each compare event. The ISR function increments then the CCR0 register by the number of timer counts until the next signal transition is needed. This way, bit-by-bit is transferred via timing that is hardware generated. The count values that determine the cycle time (bit time) and the point of the low-to-high transition are all defined in the module header file.

Figure 3. Transmit Bit Timing

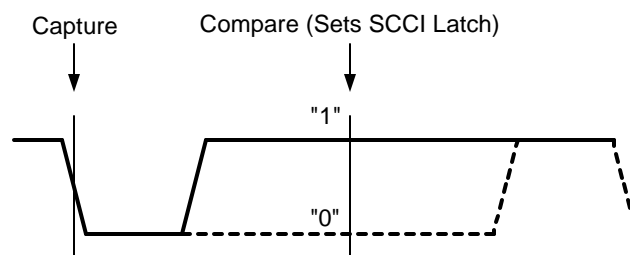


On completion of the communication the Timer_A module is disabled to conserve power. It can also be re-configured at this point if required for different functions. Note that by limitations of the HDQ protocol (e.g., compared to an I²C communication) it cannot be determined whether the slave device has received the data successfully.

```
unsigned int HDQRead(unsigned char Addr)
```

Using the function for reading a register is similar to the register write access. After sending an HDQ break condition, the address *Addr* for the slave device register to be accessed is sent out by calling the local function `HDQBasicWrite()`. By having the R/W bit in the address byte cleared, the slave device will start transmitting the contents of the addressed register by pulling low the now high-impedance HDQ bus line according to the HDQ timing specification. The Timer_A CCR0 block is used in capture mode to wait for the falling signal edge, and then switched over to compare mode. In compare mode it latches the state of the HDQ bus line at the center of the bit times for zero and one: $t_{\text{Sample}} = (t_0 + t_1) / 2$ using the SCCI latch. The SCCI latch feature of the Timer_A avoids the latency of the CPU to execute the ISR. Please refer to [3] for detailed information on the Timer_A operation.

Figure 4. Receive Bit Timing



After eight bits have been read, the Timer_A module is disabled and the function will return the read-out value. To avoid a potential software-lockup at this point (for the case that no communication to the slave device can be established), the Timer_A CCR1 block is used in compare mode to provide a time-out mechanism. It aborts the reception if no start edge could be received within t_{TO} . In this case, the `HDQRead()` function will return 0xffff to indicate the application that there was a communication failure. This is a unique value and will not occur during normal communication.

Application Example

The application example is intended to demonstrate the usage of the HDQ communication example and requires the hardware shown in Figure 2 to function. One of the features of the used bq26500 gas-gauge IC is an internal temperature sensor that can be read out using the HDQ interface. The temperature is measured periodically and stored into the internal on-chip registers `TEMPH` and `TEMPL` with a resolution of 0.25K. For a detailed description of the bq26500 features please refer to the device data sheet [1].

When the program `bq26500_LED.c` is executed, the MSP430 will do an initial temperature read-out of the bq26500 IC and store the result into the local variable `FirstTEMPL`. Prior to the initial read-out, the program will wait for a short duration to give the bq26500 time to do its first temperature measurement and initialize its internal registers properly. This delay is only needed in the case the bq26500 and the MSP430 are powered up at the same time as it is done in the example (Figure 2).

The temperature is read-out periodically every 1s using the WDT and compared to the initial temperature value FirstTEMPL. If there was a change in temperature of at least 1K (which equals four TEMPL counts), the GPIO open-drain output of the bq26500 is activated and the connected LED will illuminate. As soon as the temperature returns to the FirstTEMPL +/- 1K window, the LED will be switched off.

The watchdog-timer is operated in interval mode and will generate an interrupt every 32768 SMCLK counts. As this clock has a frequency of 2MHz with the used oscillator configuration, the WDT interrupt service function is called every 1/60th second. It decrements a counter that is used to implement a low-power Delay() function. When the counter decrements to zero, the low-power mode LPM0 is deactivated and the program execution resumes with the next instruction after the Delay() function.

The MSP430 is in low-power mode LPM0 most of the time. This mode has been chosen as it leaves the internal DCO oscillator running which is needed in this setup to generate the interval timing used for HDQ communication and also for sourcing the WDT. Other low-power modes such as LPM3 that could reduce the current consumption further require an additional external 32-kHz watch crystal.

For every HDQ read operation that is performed, the result of the HDQRead() function is compared with 0xffff to check if a time-out has occurred. In this case, the software will proceed and read the data again. Depending on the user's application, an additional error message could be generated.

References

1. *bq26500 Single-Cell Li-Ion and Li-Pol Battery Gas Gauge IC Data Sheet* (SLUS567)
2. *MSP430C11x1, MSP430F11x1A Mixed Signal Microcontroller Data Sheet* (SLAS241)
3. *MSP430x1xx Family User's Guide* (SLAU049)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Copyright © 2004, Texas Instruments Incorporated