

AN-1460 Emulating the PowerWise Interface Using GPIOs and Software

ABSTRACT

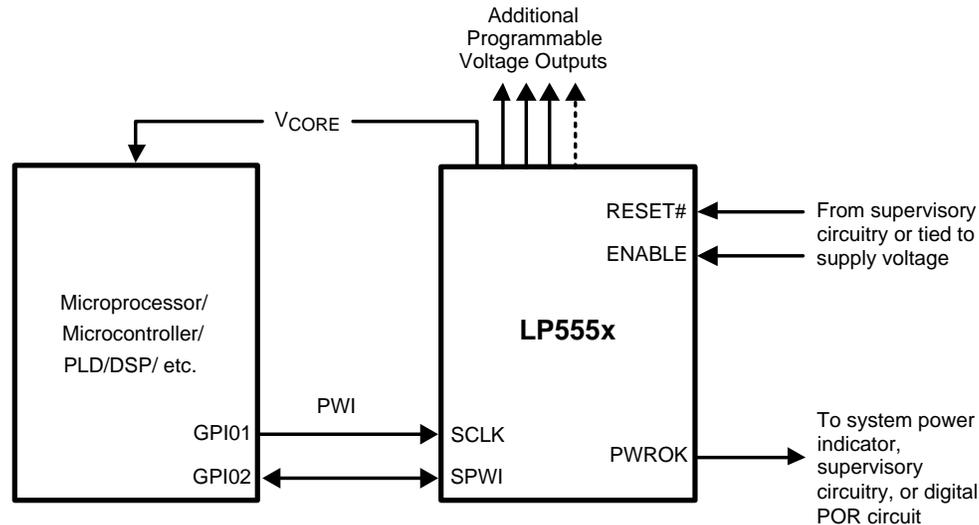
PowerWise Interface (PWI) compatible Energy Management Units (EMUs) from Texas Instruments are flexible, highly integrated, and digitally programmable system power managers that can be used in a variety of applications. Controlling or programming the EMU in the system is achieved using the PWI open-standard (www.pwistandard.org). The method for using the PWI1.0 connected EMU in systems with a processor containing a hardware PWI master like the Advanced Power Controller (APC) is obvious (http://www.ti.com/ww/en/analog/power_management/powerwise-avs.shtml). Using the PWI connected EMU in a system where the processor does not contain a hardware-based PWI master is straight forward using a GPIO port and a PWI protocol software driver.

Contents

1	Introduction	2
2	PWI Discussion	2
3	PWI Emulation	2
4	Software For PWI Emulation	3

1 Introduction

This application report describes a strategy for emulating a PWI master using general purpose input/output pins – GPIOs – and a PWI master software driver. This strategy can be deployed with a variety of processors, DSPs, and PLDs. Example C code for such a driver is included later in this document. The code presented has been verified using TI's COP8 microcontroller and the LP5550 PowerWise EMU, but should be easily portable to any processor architecture. This application applies specifically to PWI1.0, but a similar scheme could be used for PWI2.0 functionality.



2 PWI Discussion

The PWI 1.0 specification provides for a single-master, single-slave point-to-point bus. The point to point nature of the specification greatly simplifies the emulation of the PWI master.

Digital communication over the PWI is managed with a 2-wire serial interface. The PWI data line, SPWI, is bidirectional. From the master viewpoint the SPWI line is driven by the master for all command and data write frames, and driven by the slave (the EMU) for data read frames. The PWI master always drives the bus clock, SCLK. The PWI physical layer is implemented as a push-pull architecture. There are very weak bus pull-down resistors, or bus holders, to maintain low signal levels on the bus during turn around cycles, and at power-on.

3 PWI Emulation

Emulating the PWI master requires that the software driver controls the allocated GPIOs to create a direct-drive, always output SCLK pin, and a bi-directional SPWI pin. The operating frequency range for the PWI bus is 0Hz to 15 MHz, with the clock being present only during a data transaction. The fact that the bus is specified down to DC means that any device, operating at any frequency, can handle the PWI master task. This also allows a great deal of flexibility in implementing the driver because the EMU is not timing dependent beyond the mandatory set-up and hold times and the minimum pulse width of the SCLK.

The emulation driver should include support for the following PWI commands:

- Core Voltage Adjust
- Reset
- Sleep
- Shutdown
- Wakeup
- Register Write
- Register Read
- Synchronize

It is important to pay attention to the I/O voltage levels on the PWI bus. The LP555x EMUs will signal at the level of their I/O voltage regulator (i.e., the regulator associated with PWI register R7). This regulator is programmable and should be set to the same voltage as used by the PWI master if it does not directly drive the I/O ring of the master. If the I/O voltage regulator is used to power the I/O ring of the master, then no special care needs to be taken.

4 Software For PWI Emulation

The first functions that need to be implemented are the initialization, frame write, and frame read functions. It will be helpful to refer to the PWI specification, available at www.pwistandard.org, as you read through the frame write and read functions.

The initialization code is device-dependent, so only pseudo-code is provided here.

```

pwil_initialization(void)
{
    GPIO1_value = LO; //Make sure the SCLK line comes on driven low
    GPIO1_configuration = push-pull output;
    GPIO2_value = LO; //Make sure that the SPWI line comes on driven low
    GPIO2_configuration = push-pull output; //All transactions begin with SPWI as an output
#define SCLK GPIO1
#define SPWI GPIO2
#define SPWI_DIR GPIO2 Configuration
    pwil_synchronize(); //A PWI synchronize command should always be issued at POR
}
The following two functions implement the data link layer of the PWI.
byte pwil_write_frame(byte data)
{
    byte error = 0; //Return value
    byte bit = 0; //Used as data counter to serialize "data"
    SPWI_DIR = OUT; //Make sure we are driving the SPWI pin
    // START Bit
    SPWI = HI;
    pwil_clock_pulse(); //See the helper function later on in this code
    // Send the two reserved bits, 00b
    SPWI = LO;
    pwil_clock_pulse(); //Reserved 1
    pwil_clock_pulse(); //Reserved 2
    // Now we will do the payload bits
    for(bit = 0; bit < 8; bit++){
        SPWI = ((data & 0x80) ? HI : LO); //Transfer data out the MSB of data
        pwil_clock_pulse(); //Clock each bit out
        data <<= 1; //Slide the next MSB into place
    }
    // STOP Bit
    SPWI = LO;
    SCLK = HI; //Take the SCLK pin high
    SPWI_DIR = IN; // If this is a read command, the EMU will begin to drive SPWI on falling edge
    SCLK = LO; //Complete the clock cycle
    return(error); //Currently no error checking, any needed can be added
}
byte pwil_read_frame(byte* data)
{
    byte error = 0; //Return value
    byte bit = 0; //Used as data counter to handle serial-to-parallel conversion
    SPWI_DIR = IN; //Ensure that SPWI is an input
    // Look for START Bit from EMU
    if(SPWI == LO){
        error = 1; //No START Bit from slave, error code = 1
    }
    pwil_clock_pulse(); //Acknowledge the START Bit
    // Now ensure EMU drives 2 Reserved Bits, 00b
    if(SPWI == HI){
        error = 2; //Reserve Bit 1 not correct, error code = 2
    }
    pwil_clock_pulse(); //Clock in first Reserved Bit

```

```

if(SPWI == HI){
    error = 2; //Reserve Bit 2 not correct, error code = 2
}
pwil_clock_pulse(); //Clock in second Reserved Bit
for(bit = 0; bit < 8; bit++){
    (*data) = SPWI; //Grab the bit
pwil_clock_pulse(); //Tell EMU we got it
    (*data) <<= 1; //Make room for the next most significant bit
}
// Look for STOP Bit from EMU
if(SPWI == HI){
    error = 3; //No STOP Bit from slave, error code = 3
}
    pwil_clock_pulse();
    return(error); //Currently no error checking, any needed can be added
}

```

This is a supporting function that toggles the SCLK GPIO pin. This could also be done with a macro, but is not here to aid clarity.

```

void pwil_clock_pulse(void)
{
    // SCLK assumed low upon entering
    SCLK = HI;
    /** INSERT SOME DELAY HERE TO EVEN OUT THE DUTY CYCLE AND ENSURE THAT YOU
    MEET THE MINIMUM PULSE WIDTH REQUIREMENTS OF THE SPEC. ***/
    SCLK = LO;
    return();
}

```

We now proceed to the register write and register read commands. A very bare-bones implementation could get by with nothing more than these two commands, but it is recommended that all PWI functionality be implemented.

```

void pwil_reg_write(byte register_number, byte write_data)
{
    byte error = 0; //Error message; how to process?
    /* Note that this function does not ensure whether the register requested is valid,
    but it does lop off the high nibble since there are only 15 registers available. */
    register_number &= 0x0F;
    error = pwil_write_frame(CMD_REG_WRITE | register_number); //Tell the EMU what we want
    error = pwil_write_frame(write_data); //Push the data to the register
    return();
}
byte pwil_reg_read(byte register_number)
{
    byte error = 0; //Error message, how to process?
    byte register_contents = 0; //The data that was read
    /* Note that this function does not ensure whether the register requested is valid,
    but it does lop off the high nibble since there are only 15 registers available. */
    register_number &= 0x0F;
    error = pwil_write_frame(CMD_REG_READ | register_number); //Tell the EMU what we want
    // Here we must handle the turn-around clock pulse
    pwil_clock_pulse();
    // Now read what the SPC is driving back
    error = pwil_read_frame(& register;_contents);
    // Send back what we got
    return(register_contents);
}

```

Below are the command functions for PWI. We start with the two most important, Core Voltage Adjust, and the Synchronize command. The synchronize command should be implemented and called at start-up to ensure that the EMU is synchronized with the master.

```

byte pwil_core_v_adjust(byte voltage)
{
    byte error = 0; //Error message
    if(voltage & 0x80){
        error = 1; //The MSB of R0 is reserved per PWI spec
    }
    else{
        pwil_write_frame(CORE_V_ADJ | voltage); //Command with MSB set indicates Core V Adjust
    }
}

```

```
    }
    return(error);
    }
void pwil_synchronize(void)
{
    /* The synchronize command is a series of 11 1's followed by a STOP bit
    byte bit = 0; //Used for the counter below
    SPWI_DIR = OUT; //Make sure we are driving the SPWI pin
    // START Bit
    SPWI = HI;
    CLK_PULSE;
    // Send the two reserved bits, 11b, note that SPWI is already logic high
    CLK_PULSE; //Reserved 1
    CLK_PULSE; //Reserved 2
    // Now we will do the payload bits, note that SPWI is already logic high
    for(bit = 0; bit < 8; bit++){
        CLK_PULSE; //Clock each bit out
    }
    // Stop Bit is a logic low
    SPWI = LO;
    CLK_PULSE;
    return(void);
    }
void pwil_reset(void) {
pwil_write_frame(CMD_RESET); //CMD_RESET = 0x10
return(void);
}
void pwil_sleep(void) {
pwil_write_frame(CMD_SLEEP); //CMD_SLEEP = 0x11
return(void);
}
void pwil_shutdown(void) {
pwil_write_frame(CMD_SHUTDOWN); //CMD_SHUTDOWN = 0x12
return(void);
}
void pwil_wakeup(void) {
pwil_write_frame(CMD_WAKEUP); //CMD_WAKEUP = 0x13
return(void);
}
}
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

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

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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 significant portions of TI 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation	www.ti.com/automotive
Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Video and Imaging	www.ti.com/video

TI E2E Community

e2e.ti.com