ANLPS206

MIC7400 Digital Interfacing

Introduction

This application note covers the operation of the MIC7400 evaluation board available from Micrel, which is controlled using the I²C Programmer PC application, also from Micrel.

Specific code examples are provided pseudo-C where helpful. These examples are based on the availability of a standard library for accessing the I²C interface of the target hardware. This example code and descriptions are based on the assumption that the library uses the function prototypes described in Table 1.

Table 1. C Programming Prototypes used in this Document

`uint8_t i2cRead(uint8_t regAddress)`	Will read one byte of data from the I²C register address *regAddress* and return it.
`void i2cWrite(uint8_t regAddress, uint8_t data)`	Will write one byte of data stored in data to the I²C register address *regAddress*.

The standard used throughout this document, the datasheet and the GUI is to label each regulators, sequencing and related functions starting at 1,instead of the commonly used digital standard of starting numbering at 0. The six regulators are therefore referred to as REG1 to REG6, and the sequencing states as SQ1 to SQ6.Square braces [ ] are used to indicate a bit or collection of bit of a register or field. For example, REG1 [0] indicates bit zero of REG1, and REG2 [2:0] indicates the three least significant bits of REG2. ***Bold Italic*** is used to represent a register name. *Italic* alone is used to represent an external hardware signal.

Connecting and Configuring the MIC7400 Evaluation Board

Figure 1 shows the overview of the MIC7400 evaluation board and highlights the digital connections and jumpers. Figure 2 shows top and bottom views of the dongle illustrating the location of the configuration options and Figure 3 shows the orientation of the dongle when connecting to the EVB. Note that for correct operation, I²C bus pull-ups must be present either on the EVB via TP14, or on the dongle via R11 and R12.

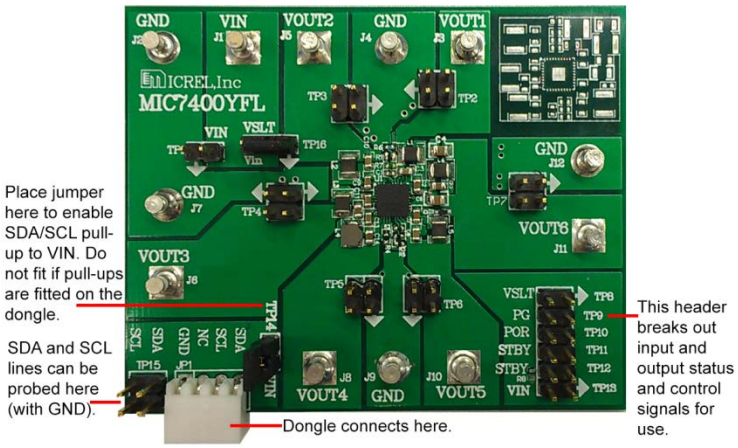


Figure 1. Digital Connections to the MIC7400 Evaluation Board

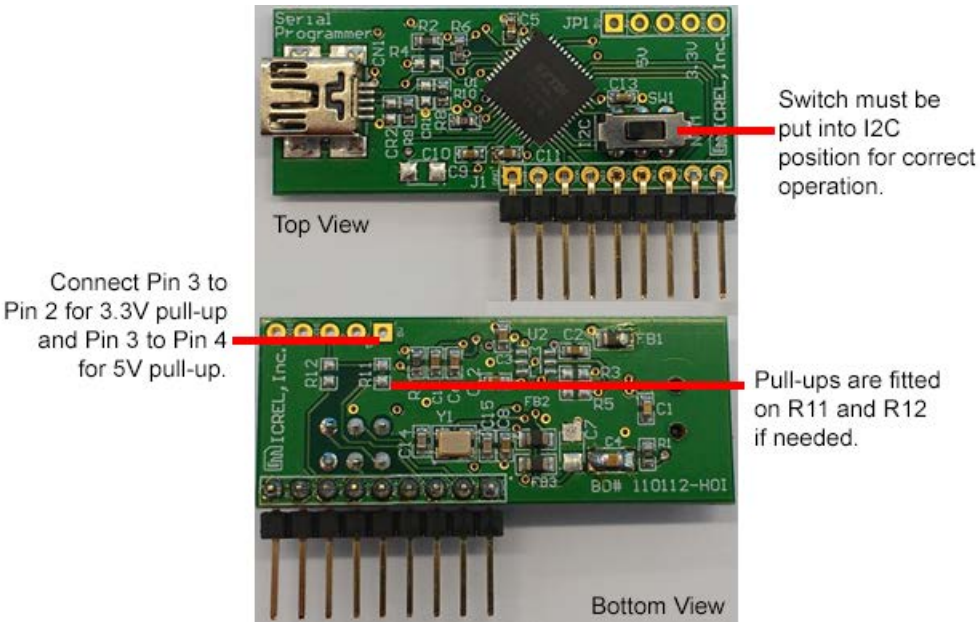


Figure 2. MICUSB Dongle

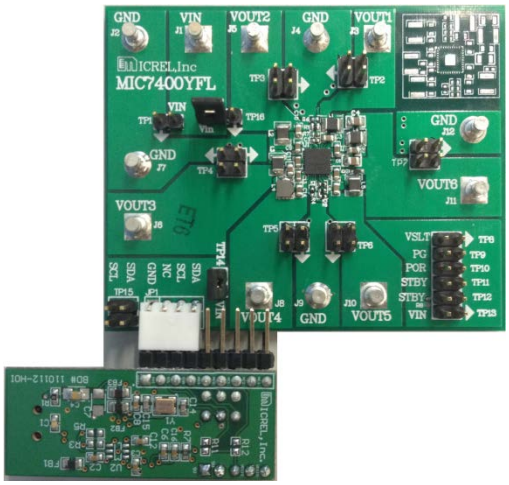


Figure 3. Connecting the Dongle to the MIC7400

MIC7400 Registers

The MIC7400 contains a total of 36 user-programmable registers, arranged in two groups of status and control registers. An additional five command registers are also available to allow the programming of internal EEPROM and I^2C control regulators. Refer to the MIC7400 datasheet for a full list of the register functions.

Status Register

There are three volatile status register (address 0x00 – 0x02) that provide a real time indication of the current status of MIC7400. See the *Determining the Status of the MIC7400* section for more details on how to read and interpret the status codes.

Table 2. Status Register Map

Register Address	Register Name	Register Bit Fields							
		7	6	5	4	3	2	1	0
0x00h	PGOOD1-6_REG			PGOOD6	PGOOD5	PGOOD4	PGOOD3	PGOOD2	PGOOD1
0x01h	STATUS_REG	EEPWRITE	EEPREAD				CALIB	CONFIG	READY
0x02h	FAULT_REG	OT		REG6OC	REG5OC	REG4OC	REG3OC	REG2OC	REG1OC

PGOODx is set when regulator x is within regulation, above 91% of the target value.

EEPWRITE and ***EEPREAD*** indicate an error when respectively writing or reading non-volatile EEPROM memory.

CONFIG indicates that the volatile register contents have been committed to non-volatile EEPROM memory.

READY indicates that the EEPROM setting have been loaded into volatile memory and the IC is ready.

OT is set when the IC temperature exceeds its safe operating value. All regulators are disabled to avoid damage.

REGxOC is set when regulator x has exceeds its current limit set by either the ***ILIMIT_x_REG*** or the ***STBY_ILIMIT_x_REG*** register. When the ***REGxOC*** bit is set, the regulator output is automatically disabled and ***PGOODx*** flag is cleared (unless masked). The ***ENx*** bit must be first cleared before the ***REGxOC*** bit can be cleared.

Control Register

Registers 0x03 - 0x23 are EEPROM-backed volatile registers that store the MIC7400 configuration data. At start up, each of the control registers is loaded from the non-volatile EEPROM to its default setting. The default settings can be factory-programmed at start up or reconfigured by the user either in the pre-production or during run-time evaluation and debugging. Once programmed to non-volatile memory, the settings become the default for the associated register. See the *Adjustable Current Limit* section for more details on programming and re-programming the default settings.

Table 3. Control Register Map

Register Address	Register Name	Register Bit Fields							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0x03h	STBY_CTRL_REG		STBY_MODEB	STBY6	STBY5	STBY4	STBY3	STBY2	STBY1
0x04h	EN_REG	SAVE1		EN6	EN5	EN4	EN3	EN2	EN1
0x05h	OUT1_REG	OUT1[5:0]							
0x06h	OUT2_REG	OUT2[5:0]							
0x07h	OUT3_REG	OUT3[5:0]							
0x08h	OUT4_REG	OUT4[5:0]							
0x09h	OUT5_REG	OUT5[5:0]							
0x0Ah	OUT6_REG	OUT6[5:0]							
0x0Bh	STBY_OUT1_REG	SB_OUT1[5:0]							
0x0Ch	STBY_OUT2_REG	SB_OUT2[5:0]							
0x0Dh	STBY_OUT3_REG	SB_OUT3[5:0]							
0x0Eh	STBY_OUT4_REG	SB_OUT4[5:0]							
0x0Fh	STBY_OUT5_REG	SB_OUT5[5:0]							
0x10h	STBY_OUT6_REG	SB_OUT6[5:0]							
0x11h	SEQ1_REG	REGxSQ1[5:0]							
0x12h	SEQ2_REG	REGxSQ2[5:0]							
0x13h	SEQ3_REG	REGxSQ3[5:0]							
0x14h	SEQ4_REG	REGxSQ4[5:0]							
0x15h	SEQ5_REG	REGxSQ5[5:0]							
0x16h	SEQ6_REG	REGxSQ6[5:0]							
0x17h	DELAY_CTRL_REG	PORDEL[7:3]					STDEL[2:0]		
0x18h	SS1-2_REG	SS_SPEED		REG2SS [5:3]			REG1SS[2:0]		
0x19h	SS3-4_REG			REG4SS [5:3]			REG3SS[2:0]		
0x1Ah	SS5-6_REG			REG6SS [5:3]			REG5SS[2:0]		
0x1Bh	ILIMIT_1-2_REG	REG2CL[7:4]					REG1CL[3:0]		
0x1Ch	ILIMIT_3-4_REG	REG4CL[7:4]					REG3CL[3:0]		
0x1Dh	ILIMIT_5-6_REG	REG6CL[7:4]					REG5CL[3:0]		
0x1Eh	STBY_ILIMIT_1-2_REG	SB2CL[7:4]					SB1CL[3:0]		
0x1Fh	STBY_ILIMIT_3-4_REG	SB4CL[7:4]					SB3CL[3:0]		

Table 3. Control Register Map (Continued)

Register Address	Register Name	Register Bit Fields							
		7	6	5	4	3	2	1	0
0x20h	STBY_ILIMIT_5-6_REG	SB6CL[7:4]				SB5CL[3:0]			
0x21h	PORUP_REG PGOOD_MASK123	PGOOD_MASK3	PGOOD_MASK2	PGOOD_MASK1	PORUP[4:0]				
0x22h	PORDN_REG PGOOD_MASK456	PGOOD_MASK6	PGOOD_MASK5	PGOOD_MASK4	PORDN[4:0]				
0x23h	PULLDN1-6_REG	PULL6	PULL6C		PULLD5	PULLD4	PULLD3	PULLD2	PULLD1

Command Register

The command registers are a group of special registers that cannot be read or written directly. Instead, issuing a write command to any of the command addresses with no data field will result in the core logic issuing the associated command. Any additional data filed will be ignored. Figure 4 shows the correct format for issuing a command and Table 4 describes the user commands that are available.

Note:

Using the *SAVECONFIG* command will result in the current contents of the volatile registers being written to internal EEPROM. This operation is not reversible.

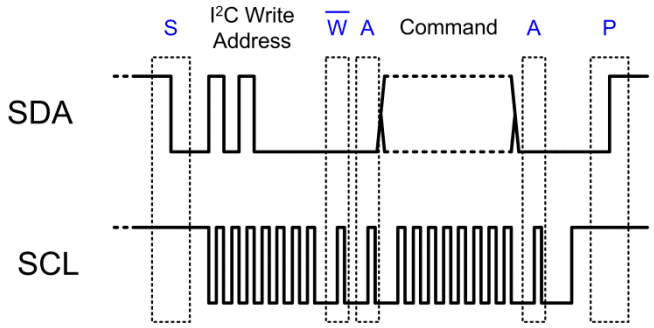


Figure 4. Command Frame Format

Table 4. Command Registers

Command Address	Command Name	Command Description
0x66h	SAVECONFIG	Save shadow register configuration data into EEPROM registers 03’h through 23’h.
0x6Ah	RESET	Reload normal mode default voltage and current limit settings and enable the regulator to normal mode with no soft-start, no sequence, no delay. Then clears the STANDBY register.
0x6Bh	RELOAD	Reload all data from EEPROM data into the shadow registers. No other actions are performed including soft-start sequencing and delay.
0x6Ch	REBOOT	Turns all regulators OFF, reloads EEPROM data into shadow registers, and then re-sequences the regulators with the programmed soft-start and sequence delays.
0x6Dh	SEQUENCE	Turns all regulators OFF, restarts the sequencer including soft-start and sequence delays.

Digital Interfacing with the MIC7400

Figure 5 shows typical digital interfacing requirements for the MIC7400. In this example, the host controller is powered by the MIC7400 with separate IO and CORE voltages at different voltage levels. In such a case, the EEPROM registers are used to start up the MIC7400 in a default state, with the correct sequencing as required. After startup, the power-on-reset (POR) pin, with its delay, is used to bring the controller out of reset. The PG is configured internally to indicate to the host controller that all power supplies are up and that the system can be started. The following sections describe each of the digital connections.

Note:

The MIC7400 has two external NC pins. For correct operation, these pins should be left floating and not shorted to VIN or GND.

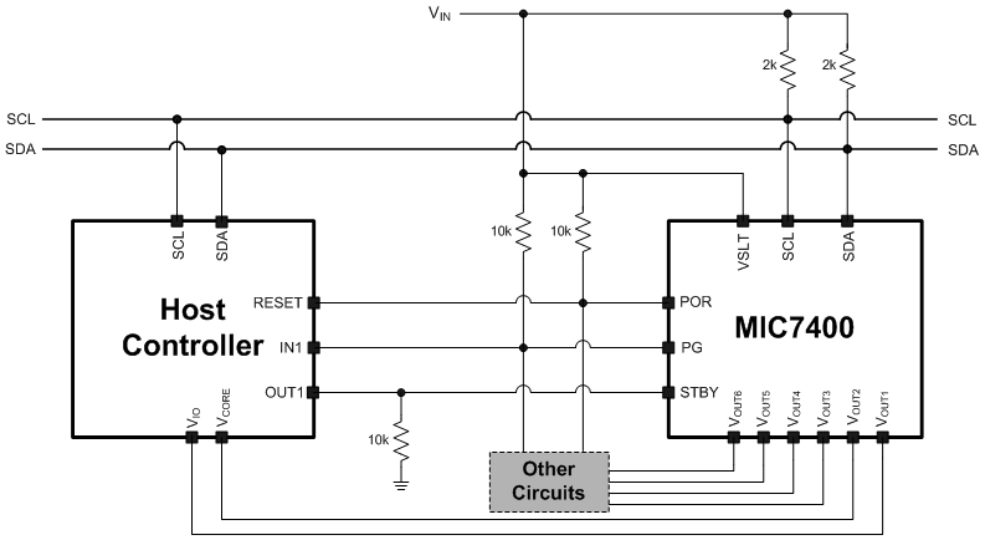


Figure 5. Interfacing the MIC7400 Digital IO to an External Controller

VSLT

The *VSLT* pin is a logic input that is used to adjust the internal POR comparator voltage range by a fixed 1V step. Table 5 shows the ranges for *VSLT* input. For POR voltage range between 2.25V to 3.25V, the pin should connect to ground. For POR voltage range between 3.25V to 4.25V, *VSLT* should be connected to VIN. See the section about *POR* for a detailed description of how *VSLT* is used by the POR comparator to generate the POR signal.

Table 5. POR VIN Range for VSLT

VSLT Input	POR Range	
	Min. (V)	Max. (V)
High (VSLT ~ VIN)	3.25	4.25
Low (VSLT ~ GND)	2.25	3.25

STBY – Wake-from-Standby Input

The *STBY* acts as an externally controlled wake-from-standby mode signal. The *STBY* pin should be connected to ground or VIN with a 100kΩ resistor and should not be left floating. If external control is required by the application, the *STBY* pin should be connect to an external driver at the controller.

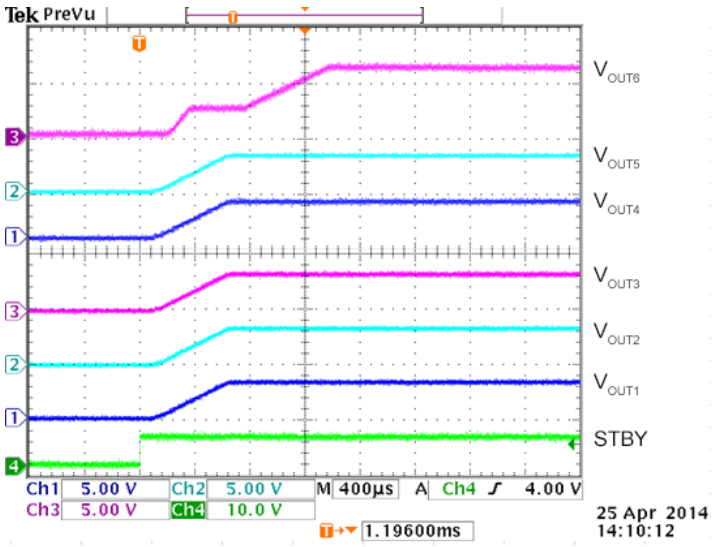


Figure 6. STBY Pulse, Wake-up Command

The rising edge of the *STBY* signal is interpreted as a wake-up command. If the MIC7400 is in ***STANDBY*** mode on the rising edge of *STBY*, the device will be automatically programmed into ***NORMAL*** mode (or by setting ***STBY_CTRL_REG*** register to logic 0), and voltage and current limit levels will transit from their ***STANDBY*** values to their ***NORMAL*** values immediately. No sequencing is performed although soft-start delays are applied to the transitions. Figure 6 illustrates the immediate transition from ***STANDBY*** to ***NORMAL*** mode on the positive edge of *STBY*.

Note:

When initiating a wake-up command, the standby mode register will be cleared. The result is that the MIC7400 will enter normal operating mode, but also that all of the regulator's standby enable bits will be cleared. Whenever waking up from standby using this method, the standby enable bits must be reprogrammed if required.

POR

The *POR* pin is an active-high open-drain output that can be used as an external *POR* reset signal for other devices when power is first applied or during reset. The output is open drain and should be connected to VIN with a nominal 100kΩ pull-up resistor. Figure 7 shows a simplified model of the *POR* output generation, and its relationship to the VIN and the configuration registers ***PORUP***, ***PORDN***, and ***PORDEL***.

At startup, the output is deactivated and pulled up to VIN by an external resistor. When VIN passes the threshold setting indicated by VSLT pin and the ***PORUP*** register setting, an internal timer is started and counts to the value indicated by the ***PORDEL*** register setting. After the timer expires, the output N-channel is enabled and the POR output is pulled-down. Subsequently, if VIN drops below the value indicated by the VSLT pin and ***PORDN*** register setting, then the output N-channel is immediately disabled with no delay, resulting in the POR output being pulled up to VIN via external resistor.

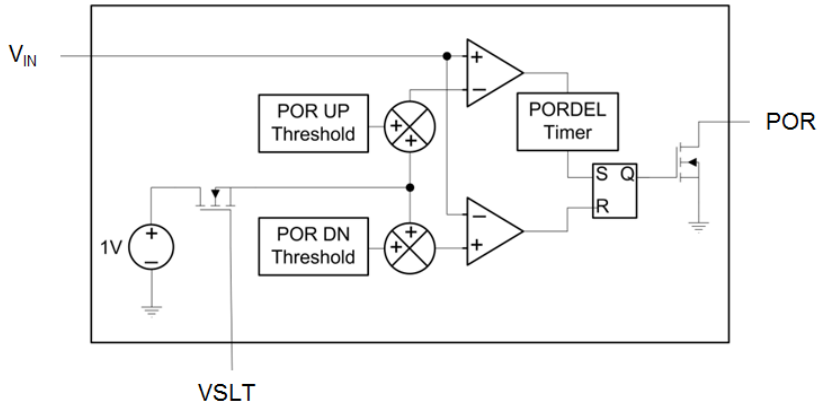
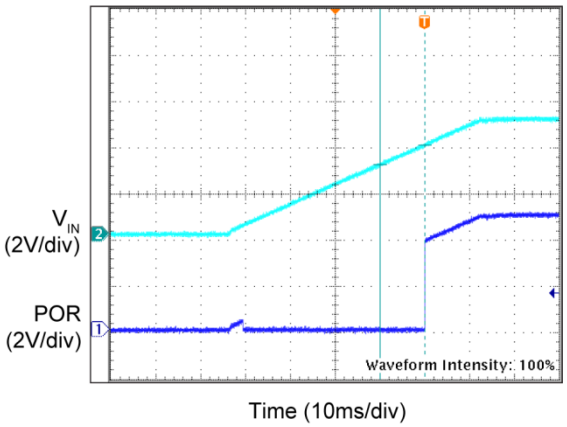


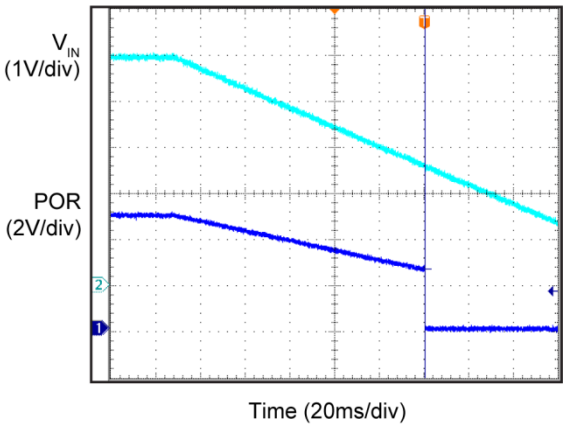
Figure 7. Simplified Model of POR Generation

Figure 8 shows an example of the POR delay output for ***PORDEL*** = 10ms, ***PORUP*** = 3.0V, and ***PORDN*** = 2.6V with VIN ramping up and down for VSLT low and high. Note that the delay is only present on the positive ramp, but not on the negative ramp.

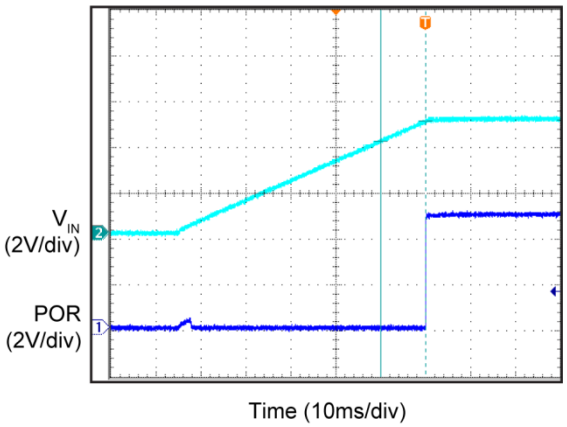
POR Rising



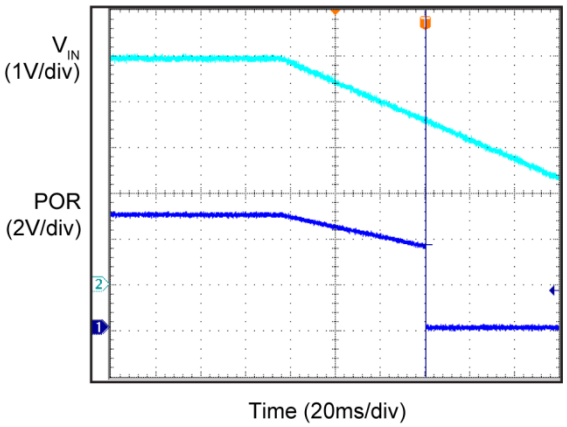
POR Falling



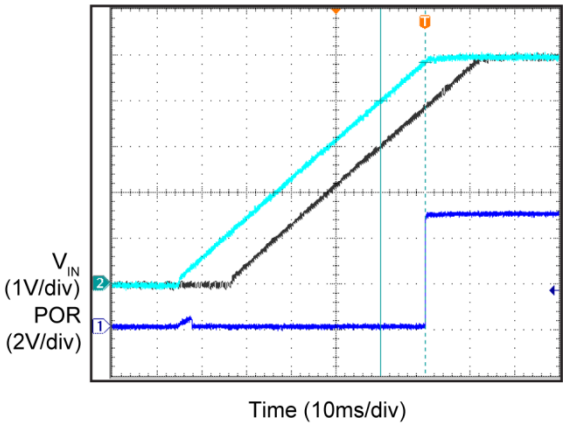
VSLT = LOW



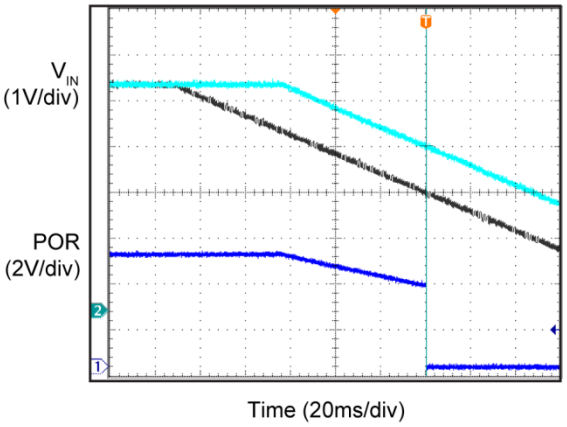
VSLT = LOW



VSLT = HIGH



VSLT = HIGH



VSLT = BOTH; High in Cyan, Low in Black

VSLT = BOTH; High in Cyan, Low in Black

Figure 8. POR Output for *PORDEL* = 10ms, *PORUP* = 3.0V, and *PORDN* = 2.6V

PGOOD

The power good (PG) pin is an open drain output that is used to indicate the status of selected internal regulator PG signals. The pin should be pulled up to VIN by a nominal 100kΩ resistor. PG is pulled low internally at startup, and is released to be pulled high via the pull up when all internal PGOOD signals are or become high. Internal PGOOD signals will go high either when within regulation, or if the ***PGOOD_MASK*** register bit is high. There is a falling edge deglitch time of 50µs to prevent false triggering on output voltage transient.

I^2C Communication, Bus Capacitance, and Maximum Frequency

The I^2C communication clock and data lines, SCL and SDA respectively, are fully compliant with the I^2C specification for HS mode and designed to work at speeds up to 3.4Mbps. Both signals are normally pulled up to V_{IN} by a suitable resistor (nominally 2kΩ). In HS master devices, the SCL pull-up is replaced by a current source to provide faster clock rising edges. Further, the clock is adjusted to 33% duty cycle (33% high) to allow more time during the setup and hold phase.

The pull-up resistor value, along with bus capacitance (C_{BUS}) on the data line, forms an RC delay that determines how quickly the clock and data signals can rise. To account for this, the I^2C specification also provides a maximum bus speed that is dependent on C_{BUS}. The maximum clock frequency is linearly scaled between 3.4MHz at 100pF and 1.7MHz at 400pF. For example, for C_{BUS} <300pF, the maximum clock frequency is 2.833MHz. To ensure suitable drive current where current source pull-ups are not available, the pull-up resistors should be suitably scaled depending on the desired operating frequency of the bus master. For correct operation, it should be possible to pull the SDA line from 0V to 0.7 × V_{IN} after 2/3 of a clock period. The percentage rise can be calculated from the following equation:

$$V_{IN} = 1 - e^{-\frac{2}{3fRC}} \qquad \text{Eq. 1}$$

Note that for 3.4Mbps operation with 2kΩ pull-ups and 66% duty cycle, the bus capacitance must be less than 80pF to allow time to charge the capacitance to 0.7 × V_{DD} threshold to guarantee logic 1 is received at the slave.

Minimizing Quiescent Current

The *POR* pin will consume current when active, approximately I = V_{IN}/R2, which is about 33µA for a V_{IN} of 3.3V through a 100kΩ resistor. Where this quiescent current is a concern, the pull-up resistance can be increased at the expense of a slower fall time of the output. Note that for noise immunity it is not recommended to exceed 1MΩ on the pull-up resistor. Because the *POR* output is not used internally, if the signal is not required by the other circuitry, then it may be possible to set internal POR threshold higher than V_{IN} (e.g., to its maximum value [VSLT = V_{IN}, ***PORUP*** = 0x1F]) such that the signal is never activated.

The PG pin will consume current when external PG is not pulled low, approximately I = V_{IN}/R1, which is about 33µA for a VIN of 3.3V through a 100kΩ resistor. The pull-up current is only active when power is not good (e.g., at startup), but the pull-up resistance can be increased at the expense of longer rise time. Note that for noise immunity it is not recommended to exceed 1MΩ on the pull-up resistor. Because the PG output is not internally, unnecessary current drain can be avoided by setting ***PGOOD_MASK*** bits of all regulators that are either not used or are not needed in the power good monitoring.

MIC7400 Modes and Operation

The MIC7400 has three modes of operation: Startup, Normal, and Standby modes.

The MIC7400 is factory-programmed to a standard configuration. The configuration information is stored in internal EEPROM and loaded upon startup or reboot. As such, no initial configuration is necessary beyond setting the EEPROM values. It is possible via the I^2C interface to reconfigure the volatile memory in any mode to control the behavior of the MIC7400.

Startup Mode

Startup mode is entered by recycling V_{IN} such that the UVLO threshold is passed, or by issuing a REBOOT command via the I^2C interface.

In startup mode, each regulator is initially disabled and all settings are reloaded from EEPROM to their default values. The startup sequence then proceeds by sequentially enabling each regulator according to the ***SEQ*** registers and ramping up to their programmed NORMAL mode voltage using soft-start settings. After each regulator completes its soft start, the internal PGOOD flag is set, which can be read out via ***PGOOD*** registers. Once all regulators have completed their soft start (or else the soft-start flag has been masked out as described in the *Adjustable Output Voltage* section), then the external PG pull-down is deactivated. The MIC7400 then proceeds to normal operating mode.

Normal and Standby Modes

In normal operating mode, voltage outputs and current limits are set according to ***OUTx*** and ***REGxCL*** registers. In standby mode, these are set by ***SB_OUTx*** and ***SBxCL*** registers. The modes offer two independent sets of current limit voltage output settings to be used and switched between by control of the ***STBY_MODEB*** bit field. Changing the ***STBY_MODEB*** bit field will automatically switch between normal and standby mode settings. The feature is designed for scenarios in which the device may go into a low-power standby state with different power supply requirements, such as disabling unused supplies or adjusting the voltage to a processor core running at a reduced clock speed.

Either standby or normal mode can be programmed to default at startup by updating non-volatile EEPROM memory. When transitioning between normal and standby mode or vice-versa, the new current limit and voltage settings are applied automatically and voltage transitions are made between the modes using the soft-start settings. If any of the regulators have the ***PGOOD_MASK*** bit set, then the *PG* output will go low while the voltage soft-starts to its new value. If the ***PGOOD_MASK*** bits are not set, then that regulator will not affect the *PG* output pin.

The I^2C interface can be used to enter normal or standby mode by programming the ***STBY_MODEB*** register bit. When writing the ***STBY_MODEB*** bit field, it is important to use a read-modify-write routine to avoid disrupting the current state of the ***STBYx*** bits that are stored in the same register. The example of code below shows how to program into and out of standby mode using the I^2C interface.

```
uint8_t reg = i2cRead(0x03);
/* This will enable standby mode */
reg = reg | 0x40;
i2cWrite(0x03, reg);
/* This will disable standby mode */
reg = reg & 0xBF;
i2cWrite(0x03, reg);
```

It is also possible to exit from standby mode by using the rising edge of the STBY input pin. This operation is directly equivalent to programming the ***STBY_MODEB*** bit to 0 via the I^2C interface, and results in the same register change. Refer to the *STBY – Wake-from-Standby Input* section for more details of using the STBY pin to exit standby mode.

Table 6. Normal and Standby Mode Registers

Register Address	Register Name	Register Bit Fields							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0x00h	PGOOD1-6_REG			PGOOD6	PGOOD5	PGOOD4	PGOOD3	PGOOD2	PGOOD1
0x03h	STBY_CTRL_REG		STBY_MODEB	STBY6	STBY5	STBY4	STBY3	STBY2	STBY1
0x04h	EN_REG	SAVE1		EN6	EN5	EN4	EN3	EN2	EN1
0x05h	OUT1_REG	OUT1[5:0]							
0x06h	OUT2_REG	OUT2[5:0]							
0x07h	OUT3_REG	OUT3[5:0]							
0x08h	OUT4_REG	OUT4[5:0]							
0x09h	OUT5_REG	OUT5[5:0]							
0x0Ah	OUT6_REG	OUT6[5:0]							
0x0Bh	STBY_OUT1_REG	SB_OUT1[5:0]							
0x0Ch	STBY_OUT2_REG	SB_OUT2[5:0]							
0x0Dh	STBY_OUT3_REG	SB_OUT3[5:0]							
0x0Eh	STBY_OUT4_REG	SB_OUT4[5:0]							
0x0Fh	STBY_OUT5_REG	SB_OUT5[5:0]							
0x10h	STBY_OUT6_REG	SB_OUT6[5:0]							
0x1Bh	ILIMIT_1-2_REG	REG2CL[7:4]				REG1CL[3:0]			
0x1Ch	ILIMIT_3-4_REG	REG4CL[7:4]				REG3CL[3:0]			
0x1Dh	ILIMIT_5-6_REG	REG6CL[7:4]				REG5CL[3:0]			
0x1Eh	STBY_ILIMIT_1-2_REG	SB2CL[7:4]				SB1CL[3:0]			
0x1Fh	STBY_ILIMIT_3-4_REG	SB4CL[7:4]				SB3CL[3:0]			
0x20h	STBY_ILIMIT_5-6_REG	SB6CL[7:4]				SB5CL[3:0]			
0x21h	PORUP_REG PGOOD_MASK123	PGOOD_MASK3	PGOOD_MASK2	PGOOD_MASK1	PORUP[4:0]				
0x22h	PORDN_REG PGOOD_MASK456	PGOOD_MASK6	PGOOD_MASK5	PGOOD_MASK4	PORDN[4:0]				

Determining the Status of the MIC7400

PGOOD

Each regulator has a user accessible internal power good ***PGOOD1-6_REG*** register bit to indicate the current state of that regulators output. These bits can be used via polling in the user application to provide advanced or extended sequencing, such as sequencing only a subset of regulators after initial startup has completed. The following code shows an example of manual sequencing of firstly regulator 2, followed by regulators 4 and 6 simultaneously.

```
uint8_t errs;                          // Used to monitor errors
uint8_t enables = 0x00;                // Start with all regulators disabled
uint8_t sequence_state = 0;            // The current sequence state
do {
        switch (sequence_state)
        {
        case 0: /* Start regulator 2 */
                enables = enables | 0x02;
                i2cWrite(0x04, enables);
                sequence_state = sequence_state + 1;
                break;
        case 1: /* Wait for PGOOD on regulator 2 */
                if (i2cRead(0x00) & 0x02)
                {
                        sequence_state = sequence_state + 1;
                }
                break;
        case 2: /* Start regulators 4 and 6 */
                enables = enables | 0x28;
                i2cWrite(0x04, enables);
                sequence_state = sequence_state + 1;
                break;
        case 3: /* Wait for PGOOD on regulator 4 and 6 */
                if ((i2cRead(0x00) & 0x28) == 0x28)
                {
                        sequence_state = sequence_state + 1;
                }
                break;
        case 4: /* All regulators running */
                break;
        }
        /* Read the OC register so we can exit on problem */
        errs = i2cRead(0x02);
} while ((!errs) && (sequence_state < 4));
```

Current Limit

Each regulator in the MIC7400 has its own adjustable current limit. If the current limit is exceeded, the regulator output will automatically be disabled and the ***REGxOC*** limit flag register bit is set. If ***PGOOD_MASKx*** is not set, then the ***PGOODx*** is cleared and the external *PG* pin will also go low. Note that the enable register bit (***ENx*** or ***STBYx*** depending on the current mode) is not cleared automatically and the ***REGxOC*** bit cannot be cleared without first setting the enable register bit low.

Table 7. Current Limit Settings and Fault Registers

Register Address	Register Name	Register Bit Fields							
		7	6	5	4	3	2	1	0
0x00h	PGOOD1-6_REG			PGOOD6	PGOOD5	PGOOD4	PGOOD3	PGOOD2	PGOOD1
0x02h	FAULT_REG	OT		REG6OC	REG5OC	REG4OC	REG3OC	REG2OC	REG1OC
0x03h	STBY_CTRL_REG		STBY_MODEB	STBY6	STBY5	STBY4	STBY3	STBY2	STBY1
0x04h	EN_REG	SAVE1		EN6	EN5	EN4	EN3	EN2	EN1
0x1Bh	ILIMIT_1-2_REG	REG2CL[7:4]				REG1CL[3:0]			
0x1Ch	ILIMIT_3-4_REG	REG4CL[7:4]				REG3CL[3:0]			
0x1Dh	ILIMIT_5-6_REG	REG6CL[7:4]				REG5CL[3:0]			
0x1Eh	STBY_ILIMIT_1-2_REG	SB2CL[7:4]				SB1CL[3:0]			
0x1Fh	STBY_ILIMIT_3-4_REG	SB4CL[7:4]				SB3CL[3:0]			
0x20h	STBY_ILIMIT_5-6_REG	SB6CL[7:4]				SB5CL[3:0]			
0x21h	PORUP_REG PGOOD_MASK123	PGOOD_MASK3	PGOOD_MASK2	PGOOD_MASK1	PORUP[4:0]				
0x22h	PORDN_REG PGOOD_MASK456	PGOOD_MASK6	PGOOD_MASK5	PGOOD_MASK4	PORDN[4:0]				

The following code example shows how to monitor the ***REGxOC*** bits and how to detect an over current condition and re-enable the regulator when found.

```
/* Read the OC FAULT status register */
uint8_t oc_reg = i2cRead(0x02) & 0x3F;
/* If we get a fault, restart all faulting regulators */
if (oc_reg)
{
        /* Determine the mode and the register to write */
        uint8_t stby = i2cRead(0x03) & 0x40;
        uint8_t wr_reg = stby ? 0x03 : 0x04;
        /* Disable the regulator, clear the fault and restart the regulator */
        uint8_t en_state = i2cRead(wr_reg);   // Read the current status of the enables
        en_state = en_state & ~oc_reg;        // Mask out OC Regulator enable bits
        i2cWrite(wr_reg, en_state);           // Disable the regulator(s)
        i2cWrite(0x02, 0x00);                 // Clear all OC flags
        en_state = en_state | oc_reg;         // Mask in the enable bits
        i2cWrite(wr_reg, en_state);           // Enable the regulators
}
```

Overtemperature

The overtemperature detection will shut down the entire regulator and set the ***OT*** bit when detected. The detection of overtemperature conditions and the resetting of the associated flag are handled in the same way as the current limits described in the *Current Limit* section.

Table 8. Overtemperature Registers

Register Address	Register Name	Register Bit Fields							
		7	6	5	4	3	2	1	0
0x02h	FAULT_REG	OT		REG6OC	REG5OC	REG4OC	REG3OC	REG2OC	REG1OC

MIC7400 Features

Adjustable Output Voltage

Each regulator can be programmed to one of two output voltages depending on the mode of operation. In normal mode, the ***OUT1 – OUT6*** registers determine the output voltage, and in standby mode the ***SB_OUT1 – SB_OUT6*** determine the output voltage (see the *MIC7400 Modes and Operation* section for more details on operation modes). The register values can be changed at any time whether active or not. When a regulator output is already on, the MIC7400 will transit to the new value using digital soft-start ramp.

Each of ***OUT1 – OUT5*** (buck regulators) are configurable between 0.800V and 3.300V in 50mV steps. The minimum output voltage of 0.800V is represented by register 0x00 and the maximum 3.300V is represented by a register value above 0x32. For boost regulator ***OUT6***, the output voltage ranges between 7.0V and 14.0V in 200mV steps. The minimum output voltage of 7.0V is represented by a register value 0x23 and the maximum 14.0V is represented by a register value above 0x00.

Table 9. Output Voltage Control Registers

Register Address	Register Name	Register Bit Fields							
		7	6	5	4	3	2	1	0
0x03h	STBY_CTRL_REG		STBY_ MODEB	STBY6	STBY5	STBY4	STBY3	STBY2	STBY1
0x04h	EN_REG	SAVE1		EN6	EN5	EN4	EN3	EN2	EN1
0x05h	OUT1_REG			OUT1[5:0]					
0x06h	OUT2_REG			OUT2[5:0]					
0x07h	OUT3_REG			OUT3[5:0]					
0x08h	OUT4_REG			OUT4[5:0]					
0x09h	OUT5_REG			OUT5[5:0]					
0x0Ah	OUT6_REG			OUT6[5:0]					
0x0Bh	STBY_OUT1_REG			SB_OUT1[5:0]					
0x0Ch	STBY_OUT2_REG			SB_OUT2[5:0]					
0x0Dh	STBY_OUT3_REG			SB_OUT3[5:0]					
0x0Eh	STBY_OUT4_REG			SB_OUT4[5:0]					
0x0Fh	STBY_OUT5_REG			SB_OUT5[5:0]					
0x10h	STBY_OUT6_REG			SB_OUT6[5:0]					

Soft-Start and Digital Voltage Scaling

The MIC7400 includes a configurable soft-start function with digital voltage scaling to limit inrush current when initially charging the output capacitors or when transitioning from one voltage to another. The soft-start function allows individual control of the rise and fall time of each output voltage via internal DAC with 50mA steps for OUT1 to OUT5 and 200mV steps for OUT6. The soft-start is applied at startup, whenever a regulator output voltage is changed by setting associated ***OUTx*** or ***SB_OUTx*** register, at reset, at reboot, at resequence, when enabling a regulator, or when switching between standby and normal modes of operation with different voltage settings. The only exception is when a regulator is disabled, in which case no soft-start is applied and the voltage goes immediately to zero at a rate determined by the load of any internally selected pull-down. The soft-start ramp step time, t_{RAMP}, is determined from each regulator by ***REGxSS*** register setting. Additionally, setting ***SS_SPEED*** to 1 will double t_{RAMP} to allow an extra-long ramp time. The total transition time is the number of steps multiplied by t_{RAMP}.

Table 10. Soft-Start and Digital Voltage Scaling Registers

Register Address	Register Name	Register Bit Fields							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0x03h	STBY_CTRL_REG		STBY_ MODEB	STBY6	STBY5	STBY4	STBY3	STBY2	STBY1
0x04h	EN_REG	SAVE1		EN6	EN5	EN4	EN3	EN2	EN1
0x05h	OUT1_REG	OUT1[5:0]							
0x06h	OUT2_REG	OUT2[5:0]							
0x07h	OUT3_REG	OUT3[5:0]							
0x08h	OUT4_REG	OUT4[5:0]							
0x09h	OUT5_REG	OUT5[5:0]							
0x0Ah	OUT6_REG	OUT6[5:0]							
0x0Bh	STBY_OUT1_REG	SB_OUT1[5:0]							
0x0Ch	STBY_OUT2_REG	SB_OUT2[5:0]							
0x0Dh	STBY_OUT3_REG	SB_OUT3[5:0]							
0x0Eh	STBY_OUT4_REG	SB_OUT4[5:0]							
0x0Fh	STBY_OUT5_REG	SB_OUT5[5:0]							
0x10h	STBY_OUT6_REG	SB_OUT6[5:0]							
0x17h	DELAY_CTRL_REG	PORDEL[7:3]					STDEL[2:0]		
0x18h	SS1-2_REG	SS_SPEED		REG2SS [5:3]			REG1SS[2:0]		
0x19h	SS3-4_REG			REG4SS [5:3]			REG3SS[2:0]		
0x1Ah	SS5-6_REG			REG6SS [5:3]			REG5SS[2:0]		

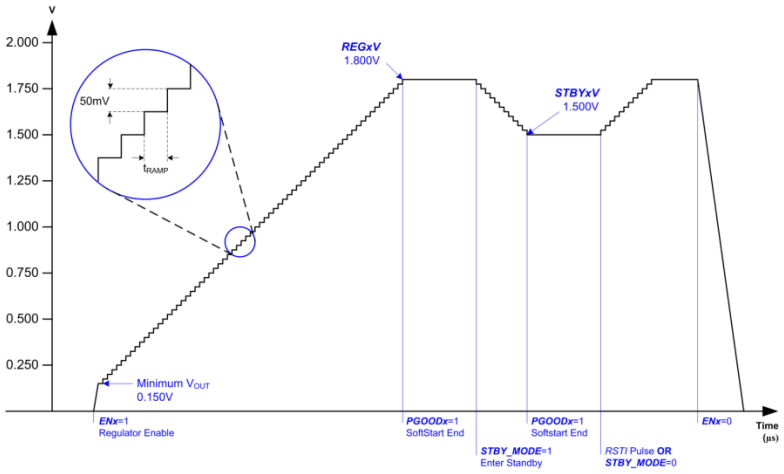


Figure 9. Buck Regulator Soft-Start Behavior for Different Events

Figure 9 shows the application of soft-start and DVS. There is a minimum voltage setting for each of the regulator outputs, which is determined by the minimum register setting. This is the first step used in the DAC output and is 150mV for buck and 1.4V for boost regulator. The boost startup waveforms need a little more explanation. The boost regulator minimum voltage is determined after pre-charge by VIN, because the inductor current will increase the switch pin voltage to overcome the V_F of the rectifying diode. When pre-charge is completed, the DAC will begin the soft-start at 1.4V, which appears as a delay in the start of the output ramp as the DAC voltage climbs to meet the current VIN level. Figure 10 shows the startup waveform for the boost regulator. If the soft-start ramp is interpolated, it can be shown to intersect V = 0 at the point corresponding to the end of the initial rise time.

After the output voltage reaches its set point, the ***PGOOD*** bit will be set. If the ***STBYx*** bit is set and the ***STBY_MODEB*** bit is also set, the regulator will soft-start down to the standby settings. The device can then be woken either by programming the ***STBY_MODEB*** bit back to 0 or by issuing a rising edge on *STBY*, at which point the output will ramp back to the normal mode settings. Disabling the regulator will result in the output voltage returning immediately to 0.

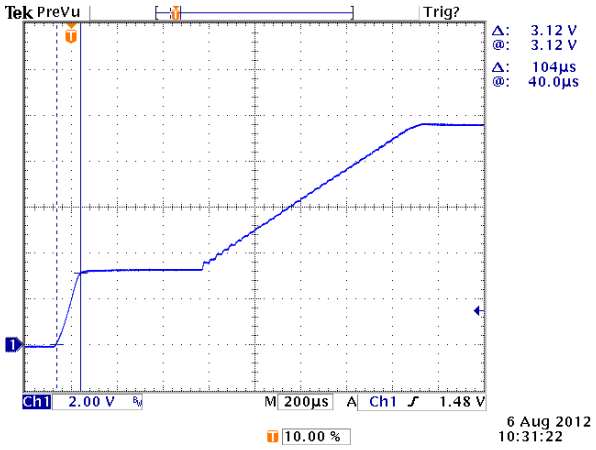


Figure 10. Boost Startup Waveform

Sequencing

The MIC7400 includes configurable sequence of regulator outputs that allow any regulator to wait for any or all of the other regulators to reach power good before starting up. The sequence is controlled by ***SEQx*** registers. Each of the ***SEQ1*** to ***SEQ6*** registers represents a single sequence state and any bits set in the sequence state will cause the regulator to start up in that state. For example, to start regulator 6 in sequence state 1, you would set bit 5 or ***SEQ1*** = 0b00100000. To start regulators 2 and 4 in sequence state 2, you would set bits 1 and 3 of ***SEQ2*** = 0b00001010. If a bit appears in more than one sequence state, it will default to using the earliest sequence state.

For a sequence state to complete and progress to the next state, all of the internal ***PGOOD*** settings for each of the regulators in that sequence state must go high. Note that in this case, if a regulator startup is interrupted before it can reach its output voltage, by an overcurrent event or otherwise, the sequencing will be halted as the ***PGOOD*** flag will not go high.

Each of the sequence states begins with a configurable 0ms to 7ms delay timer, controlled by ***STDEL***. The timer is started after all ***PGOOD*** signals of the previous state are high and must expire before the next sequence state is started. A sequence state can be used purely as a delay by not setting any bits within the sequence state register. ***STDEL*** does not have any effect following the final state as the sequencing is complete when the last regulator ***PGOOD*** is high.

Table 11. Sequencing Registers

Register Address	Register Name	Register Bit Fields							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0x00h	PGOOD1-6_REG			PGOOD6	PGOOD5	PGOOD4	PGOOD3	PGOOD2	PGOOD1
0x11h	SEQ1_REG			SEQ1[5:0]					
0x12h	SEQ2_REG			SEQ2[5:0]					
0x13h	SEQ3_REG			SEQ3[5:0]					
0x14h	SEQ4_REG			SEQ4[5:0]					
0x15h	SEQ5_REG			SEQ5[5:0]					
0x16h	SEQ6_REG			SEQ6[5:0]					
0x17h	DELAY_CTRL_ REG	PORDEL[7:3]					STDEL[2:0]		

Table 12. Example of Sequencing Bits Effects and Sequence Transitions

Register Name	Register Value	SEQ Bits						Effect upon Entering State	Transitions to Next State When…
		5	**4**	**3**	**2**	**1**	**0**		
SEQ1_REG1	0x01	0	0	0	0	0	1	REG1 will begin soft-start	PGOOD1 + STDEL
SEQ2_REG2	0x02	0	0	0	0	1	0	REG2 will begin soft-start	PGOOD2 + STDEL
SEQ3_REG3	0x04	0	0	0	1	0	0	REG3 will begin soft-start	PGOOD3 + STDEL
SEQ4_REG4	0x00	0	0	0	0	0	0	Nothing	STDEL
SEQ5_REG5	0x18	0	1	1	0	0	0	REG4 and REG5 will begin soft-start	PGOOD4 + PGOOD5 + STDEL
SEQ6_REG6	0x21	1	0	0	0	0	1	REG6 will begin soft-start	PGOOD6

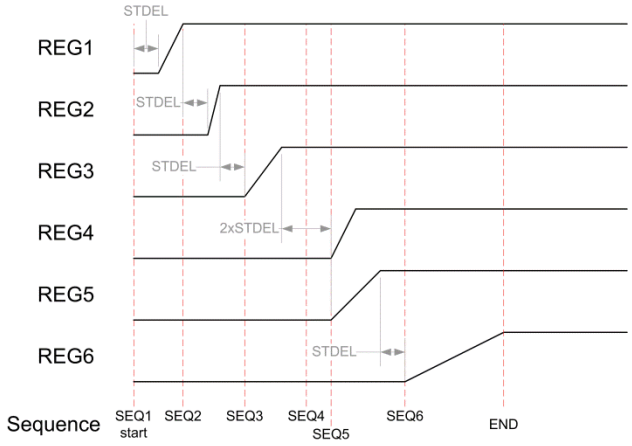


Figure 11. Regulator Sequencing Example

Consider the example sequencing configuration illustrated in Table 12 and the output shown in Figure 11, which shows a typical sequencing configuration. At startup, the regulators are all off and the controller initiates an initial delay according to ***STDEL*** before starting sequence state 1. As only ***SEQ1[0]*** is set, *REG1* will begin its soft-start ramp, until it reaches 90% of its final value, at which point ***PGOOD1*** will go high, starting sequence state 2. Sequence state 2 starts with an initial delay set by ***STDEL*** before beginning soft-start of *REG1*, setting ***PGOOD2*** high and progressing to the next state when the output voltage reaches 90% of its set value. The same applies to sequence state 3. In sequence state 4, as no bits are set, only the ***STDEL*** delay will be applied before immediately progressing to the next state. In state 5, there are two bits set, so after the initial delay both regulators will begin soft-start at the same time and, in this case, sequencing will not continue until both of the signals from regulator ***PGOOD4*** and ***PGOOD5*** are high. In the final state, again two bits are set, but in this case *REG1* has already started up and so the ***PGOOD1*** is already high and sequencing will complete when *REG6* reaches 91% of its set value.

PGOOD and Masking

The MIC7400 contains flexible power good generation and masking that allows all or any combination of the internal ***PGOOD*** signals to be combined to generate the external *PG* signal. The ***PGOOD*** signals can be separately masked by individual control bits.

By default, all of the individual ***PGOOD*** signals are used to generate the power good output for the regulator. The ***PGOOD_MASKx*** bit is used to disable the use of the PGOOD in the generation of the PG signal. If this bit is cleared, then the internal ***PGOODx*** of this regulator must go high in order for the output *PG* signal to go high (logical AND). If this is bit is set, then the ***PGOODx*** state of the regulator is not important in the generation of the external *PG* signal. The output remains dependent on the regulator PGOOD signal as long as ***PGOOD_MASKx*** bit is low, such that any disruption in the PGOOD signal will cause the external *PG* output to go low.

Table 13. Example of Sequencing Bits Effect and Sequence Transitions

Register Address	Register Name	Register Bit Fields							
		7	6	5	4	3	2	1	0
0x00h	PGOOD1-6_REG			PGOOD6	PGOOD5	PGOOD4	PGOOD3	PGOOD2	PGOOD1
0x21h	PORUP_REG PGOOD_MASK123	PGOOD_ MASK3	PGOOD_ MASK2	PGOOD_ MASK1	PORUP[4:0]				
0x22h	PORDN_REG PGOOD_MASK456	PGOOD_ MASK6	PGOOD_ MASK5	PGOOD_ MASK4	PORDN[4:0]				

Pull-Down in Disable Mode

Each of the DC-to-DC regulators has an internal pull-down that can be used to discharge the regulator output when the regulator is disabled. In addition, the boost regulator (***REG6***) has a configurable setting to control the pull-down value. Setting any of the ***PULLD1-6_REG*** register bit fields will enable the pull-down on disable function for that regulator. The ***PULL6C*** bit field controls the pull-down level for the boost regulator according to Table 15. Note the values assume V_{IN} = 5.0V and lower V_{IN} values will scale proportionally.

Table 14. Pull-Down in Disable Mode Registers

Register Address	Register Name	Register Bit Fields							
		7	6	5	4	3	2	1	0
0x23h	PULLD1-6_REG	PULL6	PULL6C		PULLD5	PULLD4	PULLD3	PULLD2	PULLD1

Table 15. REG6 Pull-Down Current Setting

PD6C[1]	PD6C[0]	Pull-Down Current
0	0	148mA
0	1	111mA
1	0	74mA
1	1	37mA

Adjustable Current Limit

Each regulator has two independently adjustable current limit settings: one each for normal mode (***REGxCL***) and one each for standby mode (***SBxCL***). This allows the MIC7400 to automatically switch current limits in the respective mode without re-programming. All buck regulator current limits are programmable between 0.6A and 8.1A in 500mA steps. The current limit setting to register value is reversed, with 0x0 representing the maximum current limit (8.1A) and 0xF representing the minimum (0.6A). For the boost regulator (REG6), the current limit is programmable between 1.76A and 2.60A in 120mA steps, representing register settings 0x7 to 0x0.

Table 16. Adjustable Current Limit Registers

Register Address	Register Name	Register Bit Fields							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0x00h	PGOOD1-6_REG		PGOOD6	PGOOD5	PGOOD4	PGOOD3	PGOOD2	PGOOD1	0x00h
0x02h	FAULT_REG	OT		REG6OC	REG5OC	REG4OC	REG3OC	REG2OC	REG1OC
0x03h	STBY_CTRL_REG		STBY_MODEB	STBY6	STBY5	STBY4	STBY3	STBY2	STBY1
0x05h	OUT1_REG	OUT1[5:0]							
0x06h	OUT2_REG	OUT2[5:0]							
0x07h	OUT3_REG	OUT3[5:0]							
0x08h	OUT4_REG	OUT4[5:0]							
0x09h	OUT5_REG	OUT5[5:0]							
0x0Ah	OUT6_REG	OUT6[5:0]							
0x1Bh	ILIMIT_1-2_REG	REG2CL[7:4]				REG1CL[3:0]			
0x1Ch	ILIMIT_3-4_REG	REG4CL[7:4]				REG3CL[3:0]			
0x1Dh	ILIMIT_5-6_REG	REG6CL[7:4]				REG5CL[3:0]			
0x1Eh	STBY_ILIMIT_1-2_REG	SB2CL[7:4]				SB1CL[3:0]			
0x1Fh	STBY_ILIMIT_3-4_REG	SB4CL[7:4]				SB3CL[3:0]			
0x20h	STBY_ILIMIT_5-6_REG	SB6CL[7:4]				SB5CL[3:0]			

Programming the Internal EEPROM

The MIC7400 contains internal EEPROM registers for storing volatile configuration data such that the MIC7400 can be configured to suit customer applications. A copy of the register configuration can be stored to the EEPROM memory via I^2C commands to allow the device defaults to be programmed or re-programmed at any time using the Save Configuration command.

When writing data to EEPROM using the Save Configuration command, a period of time is required to allow the internal EEPROM to be written. During this time, no EEPROM commands can be issued. The user can poll the ***CONFIG*** bit of the EEPROM status register to determine if the operation has completed during this time. On successful completion, the ***CONFIG*** bit will go high. The code below illustrates polling for the CONFIG bit.

```
/* Write the SAVE_CONFIG command */
i2cWrite(0x66, 0);
while ((i2cRead(0x01) & 0x02) != 0x02)
       ; /* Just wait */
/* Contents are commited */
```

Note:

Using the SAVECONFIG command will result in the current contents of the volatile registers being written to internal EEPROM, overwriting the default or any previously saved values. This operation is non-reversible and should be used with caution.

Table 17. Status Registers

Register Address	Register Name	Register Bit Fields							
		7	6	5	4	3	2	1	0
0x01h	STATUS_REG	EEPWRITE	EEPREAD				CALIB	CONFIG	READY

Appendix A: An Overview of I²C

The MIC7400 registers are accessed using the industry standard I²C interface. This section contains background information on the I²C interface and its application on the MIC7400.

Start, Stop, and Restart Sequences

Command sequences are only issued by the bus master and accepted by a slave when the SCL line is high. Given this, there are only two commands that can be supported with the other pin alone, for negative edge (Start sequence or S) and positive edge (Stop sequence or P). The restart (Sr) sequence is issued by a start command (S) while the bus is not in the idle state. The purpose of the Sr bit is to allow a start command to be issued without the master relinquishing control of the bus. This also removes the need to issue an additional stop bit; instead a repeated start bit allows only a single bit to be transmitted without the master having to relinquish control of the bus. Figure 12 shows the signal sequences for S, P, and Sr, and Table 18 provides a brief overview of the command names and sequences, along with the phases typically used in I²C communication.

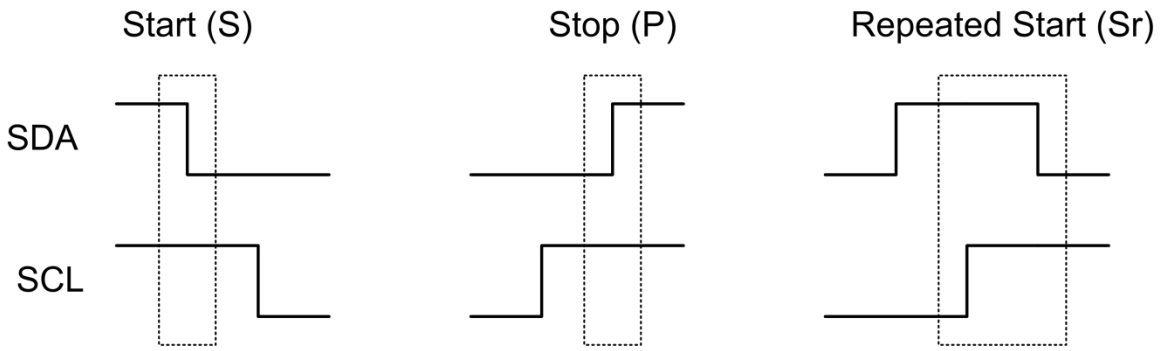


Figure 12. Start, Stop, and Restart Sequences

Table 18. I2C Sequences, Codes, and Communication Phases

Code	Command	Code	Command
S	Start	Sr	Repeated Start
P	Stop		
Code	**Phase**	**Code**	**Phase**
$\overline{W}$	Not write bit, SDA = 0	R	Read bit, SDA = 1
A	Ack bit, SDA = 0	$\overline{A}$	Not Ack bit, SDA = 1

Framing

I²C commands have specific framing sequences. These sequences rely on special start and stop conditions for framing, along with several special bit fields. Figure 13 shows the timing for an I²C write frame of a single byte and Figure 14 shows the corresponding framing for a read of a single byte. In both figures, the red lines denote the periods where the SDA line is controlled by the slave device (the MIC7400 in this case). Note that the slave address is only 7 bits and, taken together with the Read-not-Write bit, form an 8-bit word.

Figure 13. I²C Write Framing. Red Lines Indicate Slave Having Control of SDA.

For writing, additional bytes can be written within the same sequence by adding additional register address and data sequences before the stop bit.

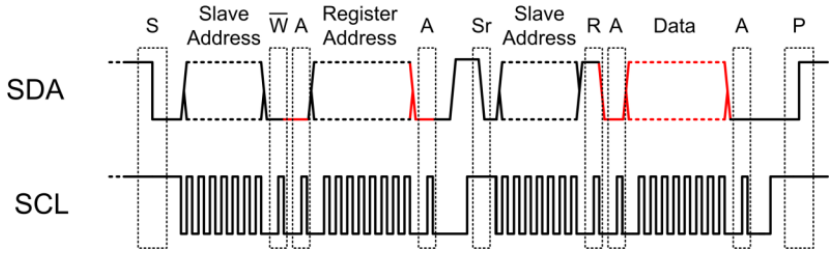


Figure 14. I²C Read Framing. Red Lines Indicate Slave Having Control of SDA.

For reading, multiple successive bytes can be read by repeating the last byte process.

High-Speed (HS) Mode at 3.4Mbps

The MIC7400 supports high speed I²C communications up to the I²C-specified 3.4MHz clock rate. HS mode is entered from the bus master by issuing a special one-byte address, bit pattern 0b0001XXXX (where X is 'don't care'). Thus all devices on the bus must then enter HS mode. Relinquishing control of the bus with a stop command will terminate the HS mode. The disadvantage is that the relatively slow HS code must be issued every time a command is requested. For multiple successive HS communications, the master can keep control of the bus by using repeated start sequences and inhibiting the generation of stop commands until the communication is complete (e.g., by just holding SCL low). When using this method, the master maintains control of the bus, effectively stopping any other device from mastering the I²C bus. The advantage is that the bus always stays in high speed mode, resulting in the fastest possible communication.

Note:

The MIC7400 does not actually respond to the HS mode command – instead it is ignored. High speed communication is possible even without the I²C HS command.

Entering High-Speed Mode

In order to enter high-speed mode, an HS code command sequence must be entered in F/S mode. This consists of a start or restart sequence followed by the HS word and a Not ACK, as shown in Figure 16. Notice that at the end of the sequence (9th bit) there is no stop or restart code. At this point, the I^2C bus is being driven by the master and is in HS mode. During this stage, no other I^2C master on the bus can arbitrate for control.

When the bus is in HS mode, the only way to exit the mode and return the bus to the idle (non-mastered) state is to issue a stop sequence. Because of this, is it possible, as shown in Figure 16, to issue a command starting with a restart sequence and return the bus to the idle state (SDA, SCL high) without issuing a stop sequence. This process can continue indefinitely and the master will continue to keep control of the bus. Note that subsequent transactions can continue at the full 3.4Mbps without needing to issue the slow HS command at F/S mode. This is how the GUI operates in HS mode. To exit HS mode, just issue a stop bit and revert to normal operation. The GUI will not do this; instead it will wait for the next communication and add the stop sequence to this. This will not cause any problems because high-speed mode will also work for lower data rates.

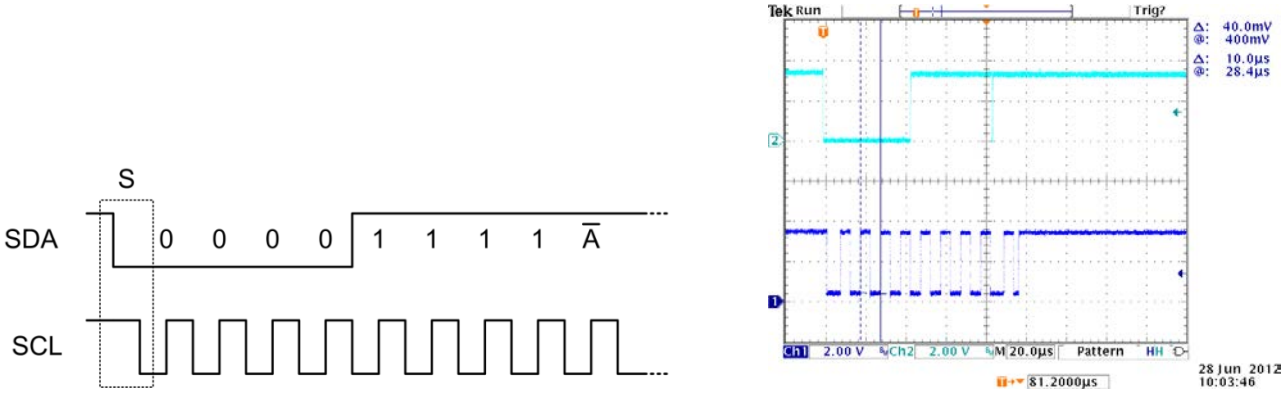


Figure 15. High-Speed Code Signaling

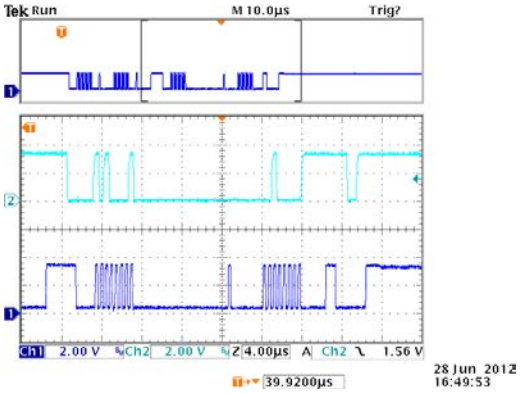


Figure 16. Subsequent HS Communications Without Stop Sequence

I²C Feature Support

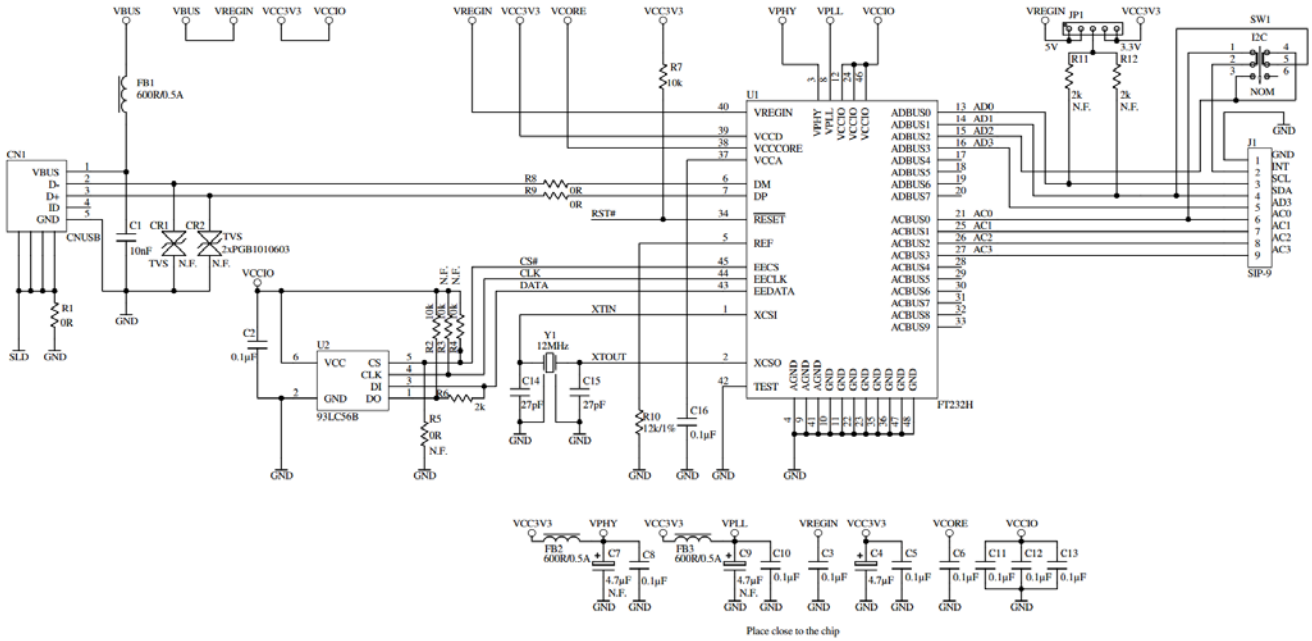
The full I²C protocol supports a number of additional features that are not necessary in every application. Table 19 lists the subset of features of the full I²C specification that are supported by the I²C slave device on the MIC7400.

Table 19. Support I²C Modes and Protocols

Feature	Support
Fast/Standard (F/S) mode	Full
High-Speed (HS) mode	Full, up to 3.4Mbps
7/10-bit addressing	7-bit addressing only
Bus arbitration	No support
Clock stretching	No support
General call address	No support
START byte	No support
Software reset	No support

Appendix B: Schematics

FT232H-based USB Dongle



MICREL, INC. 2180 FORTUNE DRIVE SAN JOSE, CA 95131 USA
TEL +1 (408) 944-0800 FAX +1 (408) 474-1000 WEB http://www.micrel.com

Micrel, Inc. is a leading global manufacturer of IC solutions for the worldwide high performance linear and power, LAN, and timing & communications markets. The Company's products include advanced mixed-signal, analog & power semiconductors; high-performance communication, clock management, MEMs-based clock oscillators & crystal-less clock generators, Ethernet switches, and physical layer transceiver ICs. Company customers include leading manufacturers of enterprise, consumer, industrial, mobile, telecommunications, automotive, and computer products. Corporation headquarters and state-of-the-art wafer fabrication facilities are located in San Jose, CA, with regional sales and support offices and advanced technology design centers situated throughout the Americas, Europe, and Asia. Additionally, the Company maintains an extensive network of distributors and reps worldwide.