MAX77654
Programmer's Guide

UG7075; Rev 0; 08/19

Abstract

The MAX77654 is an integrated power supply solution includes a charger, single-inductor-multiple-output (SIMO) regulator, and linear regulators among other features. While the MAX77654 data sheet lists the electrical characteristics and full register map, this guide gives general advice for programming individual blocks in the MAX77654.

Table of Contents

List of Tables

OTP Options

For different part numbers, certain registers can have different reset values. For example, one OTP option sets SBB1 to 1.8V output by default. Refer to the *Part Number Decode* section of the MAX77654 data sheet for the full table of different OTP options and their reset values.

Register Reset Conditions

There are different types of resets depending on either the voltage at the SYS pin or the voltage at the CHGIN pin.

System Power-On Reset

As mentioned in the MAX77654 data sheet, the device is held in reset while $V_{SYS} < V_{POR}$. All registers reset to their respective reset values while in System Power-On Reset.

System Reset

Most of the registers reset to their reset values when a System Reset occurs. A System Reset is caused by any of the following events (also mentioned in Transitions 0A and 0C in *Table 6* in the MAX77654 data sheet):

- UVLO/OVLO/OTLO
- Manual Reset
- Software Off Command
- Software Cold Reset Command
- Factory Ship Mode Command
- Watchdog Timer Expire

The ERCFLAG register does not reset when a System Reset occurs; it resets only upon System Power-On Reset. This register contains various flags indicating which event caused the System Reset.

Charger Power-On Reset

The CNFG_CHG_G.USBS bit retains its value until $V_{CHGIN} < 1.8V$.

Charger Power OK

The STAT_CHG_A and STAT_CHG_B registers (registers with status bits related to charging) and the CNFG_CHG_B.ICHGIN_LIM[2:0] bit field reset when $V_{CHGIN} < V_{CHGIN\_UVLO}$ or suspending CHGIN (CNFG_CHG_G.USBS = 1).

Interrupts and Status Registers

The interrupt registers (INT_GLBL0, INT_GLBL1, and INT_CHG) have flags indicating if a fault or event happened in the past, and the status registers (STAT_GLBL, STAT_CHG_A, and STAT_CHG_B) give current information on the device state. For information on each interrupt and status bit, refer to the Register Map section in the MAX77654 data sheet.

Interrupts

When an interrupt register is read, the interrupt flags in that register are cleared.

Interrupts can be configured to pull nIRQ LOW when they are set or, in other words, unmasked. This is typically used as an external interrupt for a host controller. After detecting the interrupt, the host controller then reads the appropriate interrupt registers to determine which event occurred. To unmask the interrupts, set the appropriate bits in the registers: INTM_GLBL0, INTM_GLBL1, and INT_M_CHG. Interrupt flags in the registers continue updating even if they are masked.

INT_CHG Register

The CHGIN_I, CHG_I, and THM_I interrupt flags in the INT_CHG register indicate that the respective status bit field has changed. After detecting that these flags have set, check the appropriate status register to determine the new status.

For example, if a charger is connected to CHGIN, the STAT_CHG_B.CHGIN_DTLS[1:0] bit field updates and the CHGIN_I interrupt flag sets. Or, if the charger experiences a fault while charging the battery, the STAT_CHG_B.CHG_DTLS[3:0] bit field updates and the CHGIN_I interrupt flag sets.

ERCFLAG

This register contains special interrupt flags that retain their values even after a System Reset. These flags, such as SYSUVLO and SFT_OFF_F, indicate which event caused a System Reset. If the device does not power up, unexpectedly resets, or unexpectedly powers down, read from the ERCFLAG register to identify the cause.

Note that, unlike other interrupt flags, the interrupt flags in the ERCFLAG register cannot be configured to pull nIRQ LOW when set.

Example Pseudocode

```
// On power-up, check if any faults occurred that caused a reset.
int slave = 0x48;
ercflag_type erc = i2c.read(slave, 0x05);
if (erc.tovld || erc.sysovlo || erc.sysuvlo){
    //…
}

//…

// Set up charger insert or removal to assert the nIRQ pin.
i2c.write(slave, 0x07, 0b0000 0100); // Unmask the CHGIN_I flag.
//…
void nirq_handler(){
    // Read the INT_CHG register to see which interrupt occurred.
    int_chg_type interrupts = i2c.read(slave, 0x01);
    if (interrupts.chgin){ // CHGIN_DTLS[1:0] changed.
        //…
    }
    //…
}
```

Global Management

Configuring nEN

Since nEN triggers the power-up sequence, choosing an OTP with the most appropriate nEN settings is recommended. If custom settings are required, do the following:

- Write to CNFG_GLBL.nEN_MODE to choose between push-button (0) or slide-switch (1) mode.
- Write to CNFG_GLBL.PU_DIS to choose between a strong (0) or weak (1) internal nEN pullup resistor.
- Write to CNFG_GLBL.DBEN_nEN to configure the debounce time.
- Write to CNFG_GLBL.TMRST to configure the manual reset time.

Configuring the Main Bias

The device can be configured to draw lower quiescent current by changing the main bias power mode. However, the tradeoff for lower quiescent current is slower response time to load changes. Write to the CNFG_GLBL.SBIA_LPM bit to set the bias in low (1) or normal (0) power mode. SBIA_LPM = 1 is a request for the device to be in low-power mode. The device will automatically transition to normal power mode as necessary.

The bias is automatically turned on if any regulator is enabled or the device is woken up. To force the system to enable the bias, set CNFG_GLBL.SBIA_EN = 1. To check if the bias is on, read STAT_GLBL.BOK.

Software Control and Factory Ship Mode

While nEN is the hardware method used to enable or disable the device, the host controller can command the device to power off or enter factory ship mode with the CNFG_GLBL.SFT_CTRL[1:0] bit field.

Example Pseudocode

```
// Force the system to enable the bias and change the bias to low-power mode.
int slave = 0x48;
cnfg_glbl = i2c.read(slave, 0x10);
cngf_glbl |= 0b0011 0000; // Set just the SBIA_LPM and SBIA_EN bits.
i2c.write(slave, 0x10, cnfg_glbl);

//…

// Send the device to factory ship mode.
i2c.write(slave, 0x10, 0b0000 0011)
```

GPIOs

Configuring GPIO Mode

GPIOs can be configured as standard GPIOs or to operate in alternate mode. The GPIO alternate modes are:

- GPIO0: Active-High Output of the SBB2 FPS Slot
- GPIO1: SBB2 Enable
- GPIO2: Bias Low-Power Mode Enable

To configure GPIOx as a standard GPIO, set CNFG_GPIOx.ALT_GPIOx = 0. Set ALT_GPIOx = 1 to configure GPIOx to operate in alternate mode. Refer to the *General-Purpose Input Output (GPIO)* section of the MAX77654 data sheet for more details about GPIO alternate modes.

Configuring as an Input

Set CNFG_GPIOx.DIR = 1 to set GPIOx as an input and read CNFG_GPIOx.DI to get the current state of GPIOx. To add a 30ms input debouncer, set CNFG_GPIOx.DBEN_GPI = 1.

Configuring as an Output

Set CNFG_GPIOx.DIR = 0 to set GPIOx as an output and write to CNFG_GPIOx.DO to change the GPIOx state. Use CNFG_GPIOx.DRV to configure the output as a push-pull (1) or open-drain (0).

GPIO Rising/Falling Events

The INT_GLBLy.GPIx_R and INT_GLBLy.GPIx_F interrupt flags indicate whether GPIOx rose or fell, respectively. Note that this flag updates whether GPIOx is configured as an input or output.

Example Pseudocode

```
// Configure GPIO0 as a push-pull output that goes HIGH when a certain feature is enabled.
// Configure GPIO1 as SBB2 enable (alternate mode).
int slave = 0x48;
i2c.write(slave, 0x11, 0b0000 0100); // GPIO0
i2c.write(slave, 0x12, 0b0010 0000); // GPIO1
//…
enable_feature();
i2c.write(slave, 0x11, 0x0C); // Toggle GPIO0 HIGH.
```

Watchdog Timer

A watchdog timer is available to automatically reset or turn off the PMIC after a period of time has expired. To use the watchdog timer, do the following:

1. Set the watchdog period with CNFG_WDT.WDT_PER[5:4].
2. Set CNFG_WDT.WDT_MODE to configure whether the device powers off (0) or resets (1) when the timer expires.
3. Enable the watchdog timer by setting CNFG_WDT.WDT_EN = 1.

   Note: If CNFG_WDT.WDT_LOCK = 1, the watchdog timer cannot be disabled.

Clear the watchdog timer periodically by setting CNFG_WDT.WDT_CLR = 1. If the watchdog is not cleared within the programmed watchdog period, the watchdog timer expires.

Example Pseudocode

```
// Enable and set a 64-second watchdog timer that shuts off the device if expired.
int slave = 0x48;
i2c.write(slave, 0x17, 0b0010 0010);
//…
void timer_handler(){ // Every X seconds, clear the watchdog timer.
   i2c.write(slave, 0x17, 0b0010 0110);
}
```

Charger

First, refer to the battery data sheet to find the appropriate charge current, charge voltage, prequalification voltage, etc. that should be used.

Configuring Prequalification

Set the prequalification voltage and current with CNFG_CHG_C.CHG_PQ[2:0] and CNFG_CHG_B.I_PQ, respectively. Note that I_PQ is a percentage of the fast-charge current.

Configuring Fast-Charge

Write to CNFG_CHG_E.CHG_CC[5:0] to set the fast-charge current and CNFG_CHG_G.CHG_CV[5:0] to set the fast-charge voltage. Calculate the values with the following equations:

$$CHG\_CC[5:0] = CC / 7.5mA - 1$$

$$CHG\_CV[5:0] = (CV - 3.600V) / 0.025V$$

If CHG_CC[5:0] > 0x27, then CC = 300mA. If CHG_CV[5:0] > 0x28, then CV = 4.600V.

Note that the SYS voltage must be at least 200mV higher than the CV.

If a safety timer for fast-charge is required, set the timer period with CNFG_CHG_E.T_FAST_CHG[1:0].

During the constant-current (CC) state, the device itself may heat up. If the die junction temperature reaches a certain threshold, the charge current is reduced. This threshold can be programmed with CNFG_CHG_D.TJ_REG[2:0].

Configuring Termination and Top-Off

In constant voltage (CV) state, the charge current drops as the battery voltage rises. After the current drops to the termination current, charging is done. Program the termination current with the CNFG_CHG_C.I_TERM[1:0] bit field.

After charging is done, the battery voltage drops as the battery relaxes. The MAX77654 can "top off" the battery, if required. Set the top-off timer with CNFG_CHG_C.T_TOPOFF[2:0]. Note that holding the battery at its peak charge voltage for a long time can lead to reduced lifetime. Again, refer to the battery data sheet for the battery manufacturer's recommendations.

Example Pseudocode

```
// Charge a 4.35V, 45mAh battery at 1C. Set a 3-hour safety timer.
int slave = 0x48;
i2c.write(slave, 0x21, 0b0000 0000); // Disable the charger.
i2c.write(slave, 0x24, 0b0001 0101); // Set CC = 45mA and 3-hour safety timer.
i2c.write(slave, 0x26, 0b0111 1000); // Set CV = 4.35V.
i2c.write(slave, 0x22, 0b1110 0000); // Set I_TERM = 5% and CHG_PQ = 3.0V.
i2c.write(slave, 0x21, 0b0000 0001); // Set I_PQ = 10% and enable the charger.
```

Battery Temperature Monitoring

To monitor the battery temperature, set CNFG_CHG_F.THM_EN = 1 to enable the thermistor function. Then, if the host controller wishes to read the voltage at the THM pin, switch the AMUX channel to output THM voltage as described in the **Analog Multiplexer** section.

JEITA

Enable the thermistor as described previously.

To set the JEITA thresholds, write to the THM_HOT[1:0], THM_WARM[1:0], THM_COOL[1:0], and THM_COLD[1:0] bit fields in the CNFG_CHG_A register to set the hot, warm, cool, and cold thresholds, respectively. Note that these bit fields set voltage thresholds at the THM pin. The temperature that corresponds to each voltage threshold depends on the β value of the thermistor. Refer to the *Register Map* section in the MAX77654 data sheet for temperature thresholds when β = 3380K. Refer to the battery data sheet for the battery manufacturer's recommendations of temperature thresholds.

There are separate bit fields to set charging current and charging voltage while the battery is warm or cool. Write to CNFG_CHG_F.CHG_CC_JEITA[5:0] to set the JEITA charging current

and CNFG_CHG_H.CHG_CV_JEITA[5:0] to set the JEITA charging voltage. The equations are the same as for the non-JEITA settings.

Example Pseudocode

```
// Set JEITA thresholds: hot = 50C, warm = 40C, cool = 10C, cold = 0C.
// At warm and cool temperatures, charge up to 4.2V at 22.5mA.
int slave = 0x48;
i2c.write(slave, 0x20, 0b0101 1010); // Set the JEITA thresholds.
i2c.write(slave, 0x27, 0b0110 0000); // Set JEITA CV = 4.2V.
i2c.write(slave, 0x25, 0b0000 1010); // Enable the thermistor and set JEITA CC = 22.5mA.
```

Charger Input

Ensure the charger input current limit is large enough to support system load current, regulator load current, and battery charge current. Note that if the charger input current limit is reached, the device enters supplement mode. Refer to the MAX77654 data sheet for more details.

Change the charger input current limit by writing to CNFG_CHG_B.ICHGIN_LIM[2:0]. Depending on the value of CNFG_SBB_TOP.ICHGIN_LIM_DEF, calculate the value with one of the following equations:

$$ICHGIN\_LIM\_DEF = 0: \quad ICHGIN\_LIM[2:0] = I_{CHGIN\_LIM} / 95mA - 1$$

$$ICHGIN\_LIM\_DEF = 1: \quad ICHGIN\_LIM[2:0] = 5 - I_{CHGIN\_LIM} / 95mA$$

If ICHGIN_LIM[2:0] > 0b100, the current limit is the same as if ICHGIN_LIM[2:0] = 0b100.

In the case of cheap chargers or long charging cables, the device can be programmed to maintain a minimum charger input voltage by decreasing the charging current. Program the minimum charger input voltage with CNFG_CHG_B.VCHGIN_MIN[2:0]. Calculate the value with the following equation:

$$VCHGIN\_MIN[2:0] = \left(V_{CHGIN\_MIN} - 4.0V\right) / 0.1V$$

There are some cases where the charger stays plugged in, but the application requires temporarily disconnecting the charger. To internally disconnect the charger from the system, set CNFG_CHG_G.USBS = 1. This creates the same condition as $V_{CHGIN} < V_{CHGIN\_UVLO}$.

Example Pseudocode

```
// Set the minimum charger input voltage = 4.5V and the charger input current limit = 285mA.
int slave = 0x48;
if (charger_inserted()){ // ICHGIN_LIM resets while there is no charger.
   //…
   i2c.write(slave, 0x21, 0b1010 1001); // Set VCHGIN_MIN, ICHGIN_LIM and enable charger.
}
```

System (SYS) Voltage

While the charger is plugged in, the SYS voltage is regulated. By default, V_{SYS} = 4.5V, but the SYS voltage can be programmed by writing to CNFG_CHG_D.VSYS_REG[4:0]. Calculate the value with the following equation:

$$VSYS\_REG[4:0] = (V_{SYS} - 4.1V) / 0.025V$$

If VSYS_REG[4:0] > 0x1C, V_{SYS} = 4.8V.

Analog Multiplexer

For low-precision estimates of charger-related voltages and currents (e.g., battery charge current or battery temperature), measure the voltage at the AMUX pin. To choose which voltage or current to output to the AMUX pin, write to the CNFG_CHG_I.MUX_SEL[3:0] bit field. Refer to the MAX77654 data sheet for conversion equations.

The full scale or range of the battery discharge current reading can be programmed with the CNFG_CHG_I.IMON_DISCHG_SCALE[3:0] bit field. This allows finer resolution if the discharge current is small.

Charger Block Status

Read from the STAT_CHG_A and STAT_CHG_B registers to get the current statuses of various functions in the charger block.

- STAT_CHG_A
  - VCHGIN_MIN_STAT: Indicates if the charger input voltage reached the minimum. See the **Charger Input** section.
  - ICHGIN_LIM_STAT: Indicates if the charger input reached the current limit. See the **Charger Input** section.
  - VSYS_MIN_STAT: Indicates if the SYS voltage reached the minimum. See the **System (SYS) Voltage** section.
  - TJ_REG_STAT: Indicates if the die junction temperature reached the programmed threshold. See the **Configuring Fast-Charge** section.
  - THM_DTLS[2:0]: Indicates if the battery is hot, cold, etc. See the **Battery Temperature Monitoring** section.
- STAT_CHG_B
  - CHG_DTLS[3:0]: Indicates the charger state. See the **Charger** section.
  - CHGIN_DTLS[1:0]: Indicates the state of the charger input. See the **Charger Input** section.
  - CHG: Indicates if the battery is being charged.
  - TIM_SUS: Indicates if the fast-charge safety timer is suspended due to a fault.

Single-Inductor-Multiple-Output (SIMO) Regulator

Configuring as a Buck or Buck-Boost

All SIMO channels can operate either in buck or buck-boost mode. The tradeoffs for operating in either mode are summarized in **Table 1**. If operating in buck mode, ensure that $V_{IN\_SBB} > V_{SBBx} + 0.7V$.

Table 1. Buck vs. Buck-Boost Tradeoffs

Tradeoff	Buck Mode	Buck-Boost Mode
Maximum Output Current	500mA for 1.8V output and I_{P_SBBx} = 1A	300mA for 1.8V output and I_{P_SBBx} = 1A
Ripple	Higher	Lower
Efficiency	Higher	Lower

To configure channel SBBx to operate in buck mode, set CNFG_SBBx_B.OP_MODE = 1. Or, to operate in buck-boost mode, set OP_MODE = 0.

Setting Output Voltage

Set the output voltage of SBBx with CNFG_SBBx_A.TV_SBBx[6:0]. Calculate the value with the following:

$$TV\_SBBx[6:0] = (V_{SBBx} - 0.8V) / 0.050V$$

If TV_SBBx[6:0] > 0x5E, V_{SBBx} = 5.500V.

Optimizing Maximum Output Current, Ripple, and Efficiency

Use the CNFG_SBBx.OP_MODE and CNFG_SBBx.IP_SBBx[1:0] bit fields to adjust maximum output current, ripple, and efficiency. OP_MODE settings are explained in the **Configuring as a Buck or Buck-Boost** section. IP_SBBx[1:0] sets the inductor peak current limit (I_{P_SBBx}), or the maximum current that the inductor current rises to. **Table 2** summarizes how higher or lower peak current limits affect ripple, efficiency, etc. Refer to the MAX77654 data sheet for more details.

Table 2. Effects of Higher or Lower I_{P_SBBx}

Tradeoff	Lower I_{P_SBBx}	Higher I_{P_SBBx}
Maximum Output Current	Lower	Higher
Ripple	Lower	Higher
Efficiency	Higher (except for I_{P_SBBx} = 0.333A)	Lower

Checking for SIMO Timeouts

The INT_GLBL1.SBB_TO interrupt flag indicates if the SIMO regulator is experiencing a heavy load such that at least one of the output channels has fallen out of regulation.

Example Pseudocode

```
// Enable SBB2. Set SBB2 = 0.85V in buck mode with IP_SBB2 = 0.5A.
int slave = 0x48;
i2c.write(slave, 0x2D, 0b0000 0001); // Set SBB2 = 0.85V.
i2c.write(slave, 0x2E, 0b0110 1110); // Set buck mode, IP_SBB2, and enable.
```

LDOs/LSWs

Configuring as an LDO or Load Switch

LDO0 and LDO1 can function as either LDOs or load switches. To configure LDOx as a load switch, set CNFG_LDOx_B.LDOx_MODE = 1. Or, to configure LDOx as an LDO, set LDOx_MODE = 0 instead.

Setting Output Voltage

Set the output voltage of LDOx with CNFG_LDOx_A.TV_LDOx[6:0]. Calculate the value with the following:

$$TV\_LDOx[6:0] = (V_{LDOx} - 0.8V) / 0.025V$$

Checking for Dropout or Faults

To check if LDOx is in dropout, read the STAT_GLBL.DODx_S bit. The corresponding interrupt bit is INT_GLBL0.DODx_R.

To check if the LDOx voltage is out of regulation, read the INT_GLBL1.LDOx_F interrupt bit.

Example Pseudocode

```
// Enable LDO0 = 1.1V. Enable LDO1 as a load switch.
int slave = 0x48;
i2c.write(slave, 0x38, 0b0000 1100); // Set LDO0 = 1.1V.
i2c.write(slave, 0x39, 0b0000 1110); // Set LDO0 to LDO mode and enable.
i2c.write(slave, 0x3B, 0b0001 1110); // Set LDO1 to LSW mode and enable.
```

Power Sequencing and Enabling/Disabling Regulators

The SIMO and LDO regulators can be enabled, disabled, or placed in a power sequence slot. To enable or disable regulators, set CNFG_SBBx_B.EN_SBBx[2:0] or CNFG_LDOx_B.EN_LDOx[2:0] = 0x6 (enable) or 0x4 (disable). To set a regulator in a power sequence slot, write the slot number to EN_SBBx[2:0] or EN_LDOx[2:0]. Slot numbers range from 0 to 3.

Regulator Active Discharge

Each regulator has an active discharge resistor to quickly drop the voltage to 0V when the regulator is disabled. To enable a regulator's active discharge resistor, set CNFG_SBBx_B.ADE_SBBx or CNFG_LDOx_B.ADE_LDOx = 1.

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	08/19	Initial release	—