


```

////////////////////////////////////
//
//  EXAMPLE: Modify the Fast Charge Current Limit
//
//
//  int userInput = 0; //integer variable to hold user input for this example.
//  int result = 0;    //integer variable to hold result of write operation.
//
//  cout << "Example: Modifying the Fast Change Current Limit" << endl;
//  cout << "This example will modify the contents of REG02." << endl << endl;
//
//  Reg02Val=ReadReg(DevID,Reg02Add,bytes2read);
//  cout << "The current setting of REG02 is: 0x" << hex << Reg02Val << endl << endl;

//
//  cout << "Please enter a value between ICHG_MIN(500) and ICHG_MAX(4532):" ;
//  cin  >> userInput;
//  result = bqSetFASTCHRG(userinput);
//  if (result == (-1) )
//      cout << "invalid entry" << endl;
//  else if (result == 0)
//      cout << "I2C Write error"  << endl;
//  else
//      cout << "I2C Write Succeeded"  << endl;
//
//  Reg02Val=ReadReg(DevID,Reg02Add,bytes2read);
//  cout << endl << "The new fast charge current setting in Reg02 is: 0x" << hex <<
Reg02Val << endl;
//
//
//
////////////////////////////////////

////////////////////////////////////
//
//  EXAMPLE: Simple code examples
//
//
//  //Change the fast charge timer to the 20hour setting
//  bqSetFastChgTimer(CHGTIMER_20h);
//
//  //Disable the safety timer setting
//  bqEnTIMER(DISABLE);
//
//  //Change the I2C watchdogtimer limit setting to 80 seconds
//  bqSetWatchDog(WatchDog_80s);
//
//
//
////////////////////////////////////

return 0;
}

// *****Start of Functions Related to I2C Functionality*****
*****
int ReadReg( int addr, int reg, int bytes2read)
{
    //***** I2C Function Initialization *****
    ****
    TIBusAdapters Adapters;
    SMBusAdapter::BusSpeed BusSpeed;

```



```

*****/
    int success;
    int RegVal;

    if((enable != ENABLE) & (enable != DISABLE))
        return -1;
    else
    {
        RegVal = enable << ENHIZ_LSHIFT;
        Reg00Val = Reg00Val & ENHIZ_MASK;
        Reg00Val = RegVal | Reg00Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg00Val;
        success = WriteReg(DevID,Reg00Add,data);

        return success;
    }
}

int bqSetVINDPM(int vdpm)
{
/*****
* bqSetVINDPM:
*
* Accepted Inputs: VINDPM_MIN >= vreg <= VINDPM_MAX
*
* Returns:
*     -1: Invalid Setting
*     0: I2C Write Fail
*     1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
* NOTE: Accepted values are determine by VINDPM_MIN,
* VINDPM_MAX variables defined in .h file. If invalid voltage*
* is detected regulation voltage will be kept as it is.
*****/

    int code = 0;
    int vregbits = 0;
    int success;

    if((vdpm < VINDPM_MIN) | (vdpm > VINDPM_MAX))
        //Invalid vreg value
        return -1;
    else
    {
        code = ((vdpm - VINDPM_MIN)/VINDPM_STEP);
        vregbits = code << VINDPM_LSHIFT;
        Reg00Val = Reg00Val & VINDPM_MASK;
        Reg00Val = vregbits | Reg00Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg00Val;
        success = WriteReg(DevID,Reg00Add,data);

        return success;
    }
}

int bqSetIINDPM(int code)

```

```

{
/*****
* bqSetIINDPM: Changes input current limit, actual current
*               is the lesser of the I2C and ILIM settings
*
* Accepted Inputs: IINLIM_100MA, IINLIM_150MA, IINLIM_500MA
* IINLIM_900MA, IINLIM_1200MA, IINLIM_1500MA, IINLIM_2000MA,
* IINLIM_3000MA
*
* Returns:
*      -1: Invalid Input
*       0: I2C Write Fail
*       1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*
*****/

    int success;
    int RegVal;

    if((code != IINLIM_100MA) & (code != IINLIM_150MA)& (code != IINLIM_500MA)& (code != IINLIM_900MA)
        & (code != IINLIM_1200MA)& (code != IINLIM_1500MA)& (code != IINLIM_2000MA)&
        (code != IINLIM_3000MA))
        return -1;
    else
    {
        RegVal = code << IINDPM_LSHIFT;
        Reg00Val = Reg00Val & IINDPM_MASK;
        Reg00Val = RegVal | Reg00Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg00Val;
        success = WriteReg(DevID,Reg00Add,data);

        return success;
    }
}

////////////////////////////////////
//-----//
//              REG 1              //
//-----//
////////////////////////////////////

int bqRstREG()
{
/*****
* bqRstREG:Resets Register Settings
* Accepted Inputs: None
*
* Returns:
*      0: I2C Write Fail
*      1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*
*****/

    int success;
    int RegVal;

```

```

    RegVal = RESET << RESETREG_LSHIFT ;
    Reg01Val = Reg01Val & RESETREG_MASK;
    Reg01Val = RegVal | Reg01Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg01Val;
    success = WriteReg(DevID,Reg01Add,data);

    return success;
}

int bqRstWatchDog()
{
    /*****
    * bqRstWatchDog:Resets WatchDog Timer
    * Accepted Inputs: None
    *
    * Returns:
    *     0: I2C Write Fail
    *     1: I2C Write Success
    *
    * Can be modified to send Ack bit as the success code
    *****/
    int success;
    int RegVal;

    RegVal = RESET << RESETWATCHDOG_LSHIFT ;
    Reg01Val = Reg01Val & RESETWATCHDOG_MASK;
    Reg01Val = RegVal | Reg01Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg01Val;
    success = WriteReg(DevID,Reg01Add,data);

    return success;
}

int bqSetCHGCONFIG(int code)
{
    /*****
    * bqSetCHGCONFIG: Charger Configuration: Disable, Charge
    *                 Battery, or OTG
    *
    * Accepted Inputs: DISABLE, CHARGE_BATTERY, OTG
    *
    * Returns:
    *     -1: Invalid Input
    *     0: I2C Write Fail
    *     1: I2C Write Success
    *
    * Can be modified to send Ack bit as the success code
    *****/
    int success;
    int RegVal;

    if((code != DISABLE) & (code != CHARGE_BATTERY) & (code != OTG))

```

```

        return -1;
    else
    {
        RegVal = code << CHGCONFIG_LSHIFT;
        Reg01Val = Reg01Val & CHGCONFIG_MASK;
        Reg01Val = RegVal | Reg01Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg01Val;
        success = WriteReg(DevID,Reg01Add,data);

        return success;
    }
}

int bqSetSYSMIN(int vlimit)
{
/*****
* bqSetVINDPM:
*
* Accepted Inputs: SYSMIN_MIN >= vlimit <= SYSMIN_MAX
*
* Returns:
*     -1: Invalid Setting
*     0: I2C Write Fail
*     1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
* NOTE: Accepted values are determine by SYSMIN_MIN,
* SYSMIN_MAX variables defined in .h file. If invalid voltage*
* is detected regulation voltage will be kept as it is.
*****/

    int code = 0;
    int regbits = 0;
    int success;

    if((vlimit < SYSMIN_MIN) | (vlimit > SYSMIN_MAX))
        //Invalid vreg value
        return -1;
    else
    {
        code = ((vlimit - SYSMIN_MIN)/SYSMIN_STEP);
        regbits = code << SYSMIN_LSHIFT;
        Reg01Val = Reg01Val & SYSMIN_MASK;
        Reg01Val = regbits | Reg01Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg01Val;
        success = WriteReg(DevID,Reg01Add,data);

        return success;
    }
}

int bqSetOTGILIM(int code)
{
/*****
* bqEnHIZ:Disable or enable the high impedance mode
* Accepted Inputs: BOOSTLIM_500mA, BOOSTLIM_1300mA
*****/

```

```

*
* Returns:
*     -1: Invalid Input
*     0: I2C Write Fail
*     1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*
*****/
int success;
int RegVal;

if((code != BOOSTLIM_500mA) & (code != BOOSTLIM_1300mA))
    return -1;
else
{
    RegVal = code << BOOSTLIM_LSHIFT;
    Reg01Val = Reg01Val & BOOSTLIM_MASK;
    Reg01Val = RegVal | Reg01Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg01Val;
    success = WriteReg(DevID,Reg01Add,data);

    return success;
}

}

////////////////////////////////////
//-----//
//                REG 2                //
//-----//
////////////////////////////////////

int bqSetFASTCHRG(int ichg)
{
/*****
* bqSetFASTCHRG:
*
* Accepted Inputs: ICHG_MIN >= ichg <= ICHG_MAX
*
* Returns:
*     -1: Invalid Setting
*     0: I2C Write Fail
*     1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
* NOTE: Accepted values are determine by ICHG_MIN,
* ICHG_MAX variables defined in .h file. If invalid voltage
* is detected regulation voltage will be kept as it is.
*****/

int code = 0;
int regbits = 0;
int success;

if((ichg < ICHG_MIN) | (ichg > ICHG_MAX))
    //Invalid vreg value
    return -1;
else
{
    code = ((ichg - ICHG_MIN)/ICHG_STEP);
    regbits = code << ICHG_LSHIFT;
    Reg02Val = Reg02Val & ICHG_MASK;

```



```

    Reg02Val = regbits | Reg02Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg02Val;
    success = WriteReg(DevID,Reg02Add,data);

    return success;
}

//-----
//          REG 3          //
//-----
//-----

int bqSetPRECHRG(int iprechg)
{
/*****
* bqSetPRECHRG:
*
* Accepted Inputs: PRECHG_MIN >= iprechg <= PRECHG_MAX
*
* Returns:
*   -1: Invalid Setting
*   0: I2C Write Fail
*   1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
* NOTE: Accepted values are determine by PRECHG_MIN,
* PRECHG_MAX variables defined in .h file. If invalid voltage
* is detected regulation voltage will be kept as it is.
*****/

    int code = 0;
    int regbits = 0;
    int success;

    if((iprechg < PRECHG_MIN) | (iprechg > PRECHG_MAX))
        //Invalid vreg value
        return -1;
    else
    {
        code = ((iprechg - PRECHG_MIN)/PRECHG_STEP);
        regbits = code << PRECHG_LSHIFT;
        Reg03Val = Reg03Val & PRECHG_MASK;
        Reg03Val = regbits | Reg03Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg03Val;
        success = WriteReg(DevID,Reg03Add,data);

        return success;
    }
}

int bqSetTERMCHRG(int iterm)
{
/*****
* bqSetTERMCHRG:
*
* Accepted Inputs: ITERM_MIN >= iterm <= ITERM_MAX
*****/

```

```

* Returns:
*   -1: Invalid Setting
*   0: I2C Write Fail
*   1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
* NOTE: Accepted values are determine by ITERM_MIN,
* ITERM_MAX variables defined in .h file. If invalid voltage
* is detected regulation voltage will be kept as it is.
*****/

int code = 0;
int regbits = 0;
int success;

if((iterm < ITERM_MIN) | (iterm > ITERM_MAX))
    //Invalid vreg value
    return -1;
else
{
    code = ((iterm - ITERM_MIN)/ITERM_STEP);
    regbits = code << ITERM_LSHIFT;
    Reg03Val = Reg03Val & ITERM_MASK;
    Reg03Val = regbits | Reg03Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg03Val;
    success = WriteReg(DevID,Reg03Add,data);

    return success;
}

//-----//
//-----//
//-----//
//-----//
//-----//

int bqSetChgVoltage(int vreg)
{
    /**
    * bqSetChgVoltage: Send battery regulation voltage in mV and
    * the function will calculate the closest value without
    * going above the desired value. Function will calculate
    * the I2C code and store it in vregbits. Reg03Val keeps track
    * of the overall register value as there are other features
    * that can be programmed on this register.
    * Accepted Inputs: VREGMIN >= vreg <= VREG_MAX
    *
    * Returns:
    *   -1: Invalid Regulation Voltage
    *   0: I2C Write Fail
    *   1: I2C Write Success
    *
    * Can be modified to send Ack bit as the success code
    * NOTE: Accepted values are determine by VREG_MAX, VREG_MIN
    * variables defined in .h file. If invalid voltage is
    * detected regulation voltage will be kept as it is.
    *****/

    int code = 0;
    int vregbits = 0;
    int success;

```

```

    if((vreg < VREG_MIN) | (vreg > VREG_MAX))
        //Invalid vreg value
        return -1;
    else
    {
        code = ((vreg - VREG_MIN)/VREG_STEP);
        vregbits = code << VREG_LSHIFT;
        Reg03Val = Reg03Val & VREG_MASK;
        Reg03Val = vregbits | Reg03Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg03Val;
        success = WriteReg(DevID,Reg03Add,data);

        return success;
    }
}

int bqSetBATLOWV(int setting)
{
    /*****
    * bqSetBATLOWV: BATLOWV setting 2.8V or 3.0V
    * Accepted Inputs: BATLOWV_2800mV, BATLOWV_3000mV
    *
    * Returns:
    *     -1: Invalid Input
    *     0: I2C Write Fail
    *     1: I2C Write Success
    *
    * Can be modified to send Ack bit as the success code
    *****/
    int success;
    int RegVal;

    if((setting != BATLOWV_2800mV) & (setting != BATLOWV_3000mV))
        return -1;
    else
    {
        RegVal = setting << BATLOWV_LSHIFT;
        Reg02Val = Reg02Val & BATLOWV_MASK;
        Reg02Val = RegVal | Reg02Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg02Val;
        success = WriteReg(DevID,Reg02Add,data);

        return success;
    }
}

int bqSetRECHRG(int setting)
{
    /*****
    * bqSetRECHRG: Battery Recharge Threshold setting 100mV or
    *               300mV
    * Accepted Inputs: VRECHG_100mV, VRECHG_300mV
    *
    * Returns:
    *****/

```

```

*      -1: Invalid Input      *
*      0: I2C Write Fail      *
*      1: I2C Write Success    *
*                               *
* Can be modified to send Ack bit as the success code      *
*                               *
*****/
    int success;
    int RegVal;

    if((setting != VRECHG_100mV) & (setting != VRECHG_300mV))
        return -1;
    else
    {
        RegVal = setting << VRECHG_LSHIFT;
        Reg02Val = Reg02Val & VRECHG_MASK;
        Reg02Val = RegVal | Reg02Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg02Val;
        success = WriteReg(DevID,Reg02Add,data);

        return success;
    }
}

////////////////////////////////////
//-----//
//          REG 5          //
//-----//
////////////////////////////////////

int bqEnTERM(int enable)
{
/*****
* bqEnTERM:Disable or enable Charge Termination      *
* Accepted Inputs: ENABLE, DISABLE                    *
*                                                       *
* Returns:                                             *
*      -1: Invalid Input      *
*      0: I2C Write Fail      *
*      1: I2C Write Success    *
*                               *
* Can be modified to send Ack bit as the success code      *
*                               *
*****/
    int success;
    int RegVal;

    if((enable != ENABLE) & (enable != DISABLE))
        return -1;
    else
    {
        RegVal = enable << ENTERM_LSHIFT;
        Reg05Val = Reg05Val & ENTERM_MASK;
        Reg05Val = RegVal | Reg05Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg05Val;
        success = WriteReg(DevID,Reg05Add,data);

        return success;
    }
}

```

```

}

int bqTERMSTAT(int enable)
{
/*****
* bqTERMSTAT: Matches ITERM or Indicates before actual
*               termination on STAT
* Accepted Inputs: TERMSTAT_ITERM, TERMSTAT_EARLY
*
* Returns:
*   -1: Invalid Input
*   0: I2C Write Fail
*   1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*****/
    int success;
    int RegVal;

    if((enable != TERMSTAT_ITERM) & (enable != TERMSTAT_EARLY))
        return -1;
    else
    {
        RegVal = enable << TERMSTAT_LSHIFT;
        Reg05Val = Reg05Val & TERMSTAT_MASK;
        Reg05Val = RegVal | Reg05Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg05Val;
        success = WriteReg(DevID,Reg05Add,data);

        return success;
    }
}

int bqSetWatchDog(int code)
{
/*****
* bqSetWatchDog:
*
* Accepted Inputs: DISABLE, WatchDog_40s, WatchDog_80s,
*               WatchDog_160s
*
* Returns:
*   -1: Invalid Input
*   0: I2C Write Fail
*   1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*****/
    int success;
    int RegVal;

    if((code != DISABLE) & (code != WatchDog_40s)& (code != WatchDog_80s)& (code !=
        WatchDog_160s))
        return -1;
    else
    {
        RegVal = code << WatchDog_LSHIFT;

```

```

    Reg05Val = Reg05Val & WatchDog_MASK;
    Reg05Val = RegVal | Reg05Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg05Val;
    success = WriteReg(DevID,Reg05Add,data);

    return success;
}

}

int bqEnTIMER(int enable)
{
/*****
* bqEnTIMER:Disable or enable Safety Timer Setting
* Accepted Inputs: ENABLE, DISABLE
*
* Returns:
* -1: Invalid Input
* 0: I2C Write Fail
* 1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*
*****/
    int success;
    int RegVal;

    if((enable != ENABLE) & (enable != DISABLE))
        return -1;
    else
    {
        RegVal = enable << ENTIMER_LSHIFT;
        Reg05Val = Reg05Val & ENTIMER_MASK;
        Reg05Val = RegVal | Reg05Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg05Val;
        success = WriteReg(DevID,Reg05Add,data);

        return success;
    }
}

int bqSetFastChgTimer(int code)
{
/*****
* bqSetFastChgTimer:
*
* Accepted Inputs: CHGTIMER_5h, CHGTIMER_8h, CHGTIMER_12h,
*                  CHGTIMER_20h
*
* Returns:
* -1: Invalid Input
* 0: I2C Write Fail
* 1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*
*****/

```

```

*****/

int success;
int RegVal;

if((code != CHGTIMER_5h) & (code != CHGTIMER_8h) & (code != CHGTIMER_12h) & (code != CHGTIMER_20h))
    return -1;
else
{
    RegVal = code << CHGTIMER_LSHIFT;
    Reg05Val = Reg05Val & CHGTIMER_MASK;
    Reg05Val = RegVal | Reg05Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg05Val;
    success = WriteReg(DevID,Reg05Add,data);

    return success;
}

}

////////////////////////////////////
//-----//
//                REG 6                //
//-----//
////////////////////////////////////
int bqSetBATCOMP(int resistor)
{
/*****
* bqSetBATCOMP:
*
* Accepted Inputs: BATCOMP_MIN >= resistor <= BATCOMP_MAX
*
* Returns:
*     -1: Invalid Setting
*     0: I2C Write Fail
*     1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
* NOTE: Accepted values are determine by BATCOMP_MIN,
* BATCOMP_MAX variables defined in .h file. If invalid
* voltage is detected regulation voltage will be kept as it
* is.
*****/

int code = 0;
int regbits = 0;
int success;

if((resistor < BATCOMP_MIN) | (resistor > BATCOMP_MAX))
    //Invalid vreg value
    return -1;
else
{
    code = ((resistor - BATCOMP_MIN)/BATCOMP_STEP);
    regbits = code << BATCOMP_LSHIFT;
    Reg06Val = Reg06Val & BATCOMP_MASK;
    Reg06Val = regbits | Reg06Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg06Val;
    success = WriteReg(DevID,Reg06Add,data);

```

```

        return success;
    }
}

int bqSetVCLAMP(int vclamp)
{
/*****
* bqSetVCLAMP:
*
* Accepted Inputs: VCLAMP_MIN >= vclamp <= VCLAMP_MAX
*
* Returns:
*     -1: Invalid Setting
*     0: I2C Write Fail
*     1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
* NOTE: Accepted values are determine by VCLAMP_MIN,
* VCLAMP_MAX variables defined in .h file. If invalid voltage
* is detected regulation voltage will be kept as it is.
*****/

    int code = 0;
    int regbits = 0;
    int success;

    if((vclamp < VCLAMP_MIN) | (vclamp > VCLAMP_MAX))
        //Invalid vreg value
        return -1;
    else
    {
        code = ((vclamp - VCLAMP_MIN)/VCLAMP_STEP);
        regbits = code << VCLAMP_LSHIFT;
        Reg06Val = Reg06Val & VCLAMP_MASK;
        Reg06Val = regbits | Reg06Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg06Val;
        success = WriteReg(DevID,Reg06Add,data);

        return success;
    }
}

int bqSetTREG(int code)
{
/*****
* bqSetTREG:
*
* Accepted Inputs: TREG_60C, TREG_80C, TREG_100C, TREG_120C
*
* Returns:
*     -1: Invalid Input
*     0: I2C Write Fail
*     1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*****/

    int success;

```



```

    int RegVal;

    if((code != TREG_60C) & (code != TREG_80C)& (code != TREG_100C)& (code != TREG_120C))
        return -1;
    else
    {
        RegVal = code << TREG_LSHIFT;
        Reg06Val = Reg06Val & TREG_MASK;
        Reg06Val = RegVal | Reg06Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg06Val;
        success = WriteReg(DevID,Reg06Add,data);

        return success;
    }
}

////////////////////////////////////
//-----//
//                REG 7                //
//-----//
////////////////////////////////////

int bqEnDPDM(int enable)
{
    /*****
    * bqEnDPDM:Disable or enable D+/D- Detection
    * Accepted Inputs: ENABLE, DISABLE
    *
    * Returns:
    *     -1: Invalid Input
    *     0: I2C Write Fail
    *     1: I2C Write Success
    *
    * Can be modified to send Ack bit as the success code
    *****/
    int success;
    int RegVal;

    if((enable != ENABLE) & (enable != DISABLE))
        return -1;
    else
    {
        RegVal = enable << ENDPDM_LSHIFT;
        Reg07Val = Reg07Val & ENDPDM_MASK;
        Reg07Val = RegVal | Reg07Val;

        //Execute I2C Write Function
        array<Byte,1> ^data = gcnew array<Byte,1>(1);
        data[0]=Reg07Val;
        success = WriteReg(DevID,Reg07Add,data);

        return success;
    }
}

int bqEnTMR2X(int enable)
{
    /*****
    * bqEnTMR2X:Disable or enable 2x Extended Safety Timer
    * Accepted Inputs: ENABLE, DISABLE
    *
    *****/

```

```

* Returns:
*   -1: Invalid Input
*   0: I2C Write Fail
*   1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*
*****/
int success;
int RegVal;

if((enable != ENABLE) & (enable != DISABLE))
    return -1;
else
{
    RegVal = enable << EN2XTIMER_LSHIFT;
    Reg07Val = Reg07Val & EN2XTIMER_MASK;
    Reg07Val = RegVal | Reg07Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg07Val;
    success = WriteReg(DevID,Reg07Add,data);

    return success;
}

}

int bqOffBATFET(int enable)
{
/*****
* bqOffBATFET:Disable or enable 2x Extended Safety Timer
* Accepted Inputs: ENABLE, DISABLE
*
* Returns:
*   -1: Invalid Input
*   0: I2C Write Fail
*   1: I2C Write Success
*
* Can be modified to send Ack bit as the success code
*
*****/
int success;
int RegVal;

if((enable != ENABLE) & (enable != DISABLE))
    return -1;
else
{
    RegVal = enable << OFFBATFET_LSHIFT;
    Reg07Val = Reg07Val & OFFBATFET_MASK;
    Reg07Val = RegVal | Reg07Val;

    //Execute I2C Write Function
    array<Byte,1> ^data = gcnew array<Byte,1>(1);
    data[0]=Reg07Val;
    success = WriteReg(DevID,Reg07Add,data);

    return success;
}

}

```

```
////////////////////////////////////
//-----//
//                REG 8                //
//-----//
////////////////////////////////////

// *****End of Functions Related to 19x Functionality*****↵
// *****
```