```
// Exemplary c-code for the CaliPile
// Presence detection with host optimization
// The code does
// - improve the adoption speed of the CaliPile
// - determine the presence of a person

// After start-up of the host system
// The person must leave and enter the field-of-view once
// for a proper operation
// Once a (thermal) instability was recognized
// by heat-up or cool-down of the sensor
// during presence sensing,
// the host system will switch to safe mode, where
// presence sensing is not possible any more
// this will be indicated by a blinking of the LED

// the code was optimized for the Excelitas Demonstration Set
// in case of a fixed mounting of the bare sensor
// please adapt all settings like filters and thresholds
// to your application conditions

// this code must be adapted for each host system
// Excelitas is not liable for the code
// All rights belong to Excelitas but the code
// can be used and modified for any CaliPile application
// free of charge

// SMBus/I2C Rx Tx Buffer for register + eeprom content
unsigned char           SMB_buf[64];
// SMBUS Slave address = default 10
unsigned char           slave_address;

// timer flag is on to trigger once the host optimization procedure.
// This will lead to a faster resetting of the sensor at the power-on.
// on power-on you MUST write this configuration to initialize the
// sensor properly for the host optimization procedure
// thresholds, filter settings etc. must be optimized in the application
// every application is unique and required detailed understanding and/or testing
//unsigned char   register_write[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
//  0x8D,0x0D,15,30,30,0x09,0x04,20,0xFF,0x00,0,0}; // high sensitivity
unsigned char   register_write[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0x8B,0x0B,30,30,30,0x09,0x04,20,0xFF,0x00,0,0}; // low sensitivity

// 32 bit variables
unsigned long int TPobject;
unsigned long int TPambient;

// presence detection requires a stable environment
// once a person is in the field of view there is no way
// to distinguish between background and signal with
// a 1 channel sensor.
// Thus a stability requirement is needed for safe
```

```c
// absence detection.
unsigned long int TPambient_was = 0;

unsigned long int TPObjLP1;
unsigned long int TPObjLP2;
unsigned long int TPambLP3;

// those are states of the CaliPile
// used for presence detection or recognition
typedef enum Tintstatus {
    STATnointerrupt=0, STATinterrupt, STATabsence,
    STATpresence, STATresetting, STATsetting} intstatus;
#define MAXSTATUS         5

intstatus  current_status = STATnointerrupt;

// return value: (0) = Transmission OK
//               (1) = SA+W not acked
//               (2) = start address not acked
//               (3) = SA+R not acked
// parameters  : start adress = 0...31
//               length = no of bytes to read 1...32
char read_register(char start_adr, char length)
{
    char status;
    // here comes controller specific code for I2C communication
    return(status);
}

// return value: (0) = Transmission OK
//               (1) = SA not acked
//               (2) = start address not acked
//               (3) = SMB_wbyte(value) not acked
// parameters  : start address = 0...31
//               value  = 0 ... 255
char write_register(char adr, char value)
{
    char status;
    // here comes controller specific code for I2C communication
    return(status);
 }

// optimize the response of the sensor by
// setting filters to fast values once an
// interrupt was detected
// determine presence with the over-temperature feature
// call this procedure in case of all CaliPile interrupts
char presence_detection()
{
    char chipstatus;
    char interruptstatus;
    char interruptmask;
    char LPsettings;
    char mask;
    char i;
    unsigned int dTPamb;
    intstatus new_status;
    char under_temperature_soft;

    //LED = 1;

    // read the chip status and
    // interrupt status to reset the interrupt of the chip
    // if this method is called via an interrupt, make
    // sure the interrupt was not read previously
    // otherwise use the chip status which may
    // be not up-to-date depending on the readout speed

    read_register(0, 32);
    interruptstatus = SMB_buf[18]; // interrupt status + chip status
    chipstatus = SMB_buf[19];

    // simulate here the interrupt since interrupt pin is not used on this
    board
    // read the interruptmask
    interruptmask = SMB_buf[25];
    // compare interrupstatus with the mask like the sensor is doing it
    if ((interruptstatus & interruptmask) == 0)
    {
        return 0;
    }

    // initialize the status
    new_status = current_status;

    // ************ ambient temperature stability condition check *****************
    // determine the current ambient offset to previously registered one
    TPambient = SMB_buf[10];// & mask;
    TPambient <<= 8;
    TPambient |= SMB_buf[11];
    // divide it by 2 to get rom 16 bit to 15 bit PTAT resolution with a slope of
    172 counts/K
    TPambient >>= 1;
    // initialize the variable the first time
    if (TPambient_was == 0) TPambient_was = TPambient;
    if (TPambient_was > TPambient)
        dTPamb = TPambient_was - TPambient;
    else
        dTPamb = TPambient - TPambient_was;
    if (dTPamb > 30) // did the condition change by more than ~0.2K? Please optimize
    this condition for your application
    {
        if (current_status == STATpresence)
        {
```

```c
            // if the situation is not stable, the overtemperature limit is probably not correct
            new_status = STATresetting;
            // indicate with the LED
            for (i = 0; i < 3; i++)
            {
                LED = 0;
                DEBUGPIN = 0;
                delay_ms(100);
                LED = 1;
                DEBUGPIN = 1;
                delay_ms(100);
            }
        }
        TPambient_was = TPambient;
    }
    // ************ end of temperature stability condition check *****************

    // use the chip status to capture the current condition
    // interruptstatus = chipstatus;

    // initialize the chip for first time usage after power on
    if (current_status == STATnointerrupt) new_status = STATresetting;

    // entrance condition to presence condition
    // 1. Enter only when the new_status was not redefined by another condition
    // 2. must come from the absence condition
    // 3. must be identified as presence flag after interrupt
    // 4. must have positive sign on presence flag
    // use chip status not to miss a current condition which was not triggered
    if (new_status == current_status && current_status == STATabsence
        && (chipstatus & 0x08) != 0 && (chipstatus & 0x80) == 0)
        //&& (interruptstatus & 0x08) != 0 && (interruptstatus & 0x80) == 0)
    {
        new_status = STATpresence;
    }

    // entrance condition to set fast filter resetting
    // 1. Enter only when the new_status was not redefined by another condition
    // 2. must come from the absence condition
    // 3. must be identified as presence flag after interrupt
    // 4. must have negative sign on presence flag
    // use chip status not to miss a current condition which was not triggered
    if (new_status == current_status && current_status == STATabsence
        && (chipstatus & 0x08) != 0 && (chipstatus & 0x80) != 0)
        //&& (interruptstatus & 0x08) != 0 && (interruptstatus & 0x80) != 0)
    {
        new_status = STATresetting;
    }

    // entrance condition to enter the absence status
    // 1. Enter only when the new_status was not redefined by another condition
    // 2. Enter only when the current_status is resetting
    // 3. must have presence flag (filters catched up)
    // 4. Presence sign flag must indicate negative number
    // use interrupt status to look into the memory of the sensor
    if (new_status == current_status && current_status == STATresetting
        //&& (chipstatus & 0x08) != 0 && (chipstatus & 0x80) == 0
        && (interruptstatus & 0x08) != 0 && (interruptstatus & 0x80) == 0
        )
    {
        new_status = STATabsence;
    }

    // ********* check in software the under temperature condition ***********
    if (current_status == STATpresence)
    {
        // please note: using the internal under-temperature feature to generate a
        trigger
        // requires the person to approach the device first
        // to generate a signal which is above the threshold by 64 counts at least
        // otherwise the lamp will turn off again
        // this is due to the internal hysteresis
        // numbers can be compared on the µC from time to time in addition which is
        here reflected
        under_temperature_soft = 0;
        if (SMB_buf[1] < SMB_buf[28])
        {
            under_temperature_soft = 1;
        }
        if (SMB_buf[1] == SMB_buf[28] && SMB_buf[2] < SMB_buf[29])
        {
            under_temperature_soft = 1;
        }
    }

    // entrance condition to enter the resetting condition
    // 1. Enter only when the new_status was not redefined by another condition
    // 2. Enter only when the current_status is the presence condition
    // 4. Enter only when undertemperature was sensed or better
    // 4. Enter only when software undertemperature was sensed
    if (new_status == current_status && current_status == STATpresence
        //&& (chipstatus & 0x10) != 0  // use chip status which is the up to date
        condition not cleared in the interrupt register
        && under_temperature_soft != 0 // use in addition the software
        undeartemperature in case of weak signals below 64 counts
        )
    {
        new_status = STATresetting;
    }

    if (new_status != current_status)
    {
        // configuration according to the new status
        if (new_status == STATresetting)
        {
            // make the background low pass follow the filtered signal nearly with
            nearly the same speed
            write_register(20, 0xCD);
            interruptmask = 0x08;
            write_register(25, interruptmask);
            // set now the presence threshold to a value of only 1 so that a sign
            change
            // will be triggered immediately
            write_register(22, 1);
        }

        if (new_status == STATabsence)
        {
            // restore default filter settings and the threshold
            write_register(20, register_write[20]);
            write_register(22, register_write[22]);
            // prepare the system to go to sleep and listen only to the positive
            presence interrupt
            // trigger on the presence feature only and put your host to sleep
            interruptmask = 0x08;
            write_register(25, interruptmask);
        }

        if (new_status == STATpresence)
        {
            interruptmask = 0;
            // setup for presence detection via overtemperature
            // store the last L1val to overtemperature for the overtemperature
            feature in case of absence detection
            // optionally LP2 can be used but may be less robust an absence
            misinterpretation due to a warm seat or similar
            // LP1 may give you a shorter distance but is more robust to sence your
            absence
            TPObjLP1 = 0;
            TPObjLP2 = 0;

            // copy first 16 bit of TPLP1 or TPLP2 to the TPOT

            TPObjLP1 = SMB_buf[5];
            TPObjLP1 <<= 8;
            TPObjLP1 |= SMB_buf[6];
            TPObjLP1 <<= 8;
            TPObjLP1 |= SMB_buf[7];
            TPObjLP1 >>= 4;

            mask = 0x0F;
            TPObjLP2 = (unsigned int)(SMB_buf[7] & mask);
            TPObjLP2 <<= 8;
            TPObjLP2 |= SMB_buf[8];
            TPObjLP2 <<= 8;
            TPObjLP2 |= SMB_buf[9];
            // use LP1 if you want a safer absence measurement
            // use LP2 if you want a better presence measurement
            TPOTthres = TPObjLP1 / 8; // 20 bits maped on 17 bits
            TPOTthres >>= 1; // last bit is not used
            // set now the current thereshold
            write_register(29, (char)TPOTthres);
            TPOTthres >>= 8; // shift the first 8 bits to the last position
            write_register(28, (char)TPOTthres);

            // store the current ambient temperature as a hint if the condition is
            stable
            TPambient_was = TPambient;

            // set the system to send an interrupt on an undertemperature event
            // make sure bit 4 in register 26 is set to 0 to notice an
            undertemperature event!
            interruptmask = 0x10;

            // enable the timer to check periodically the temperature stability
            condition
            // and make sure the timing of readout and real events did not skip one
            interrupt condition
            interruptmask |= 0x01;

            interruptmask |= 0x08;
            write_register(25, interruptmask);
        }

        // indication of new status
        if (new_status == STATpresence)
        {
            LED = 1;
            DEBUGPIN = 1;
        }
        else
        {
            LED = 0;
            DEBUGPIN = 0;
        }

        current_status = new_status;
        return 1;
    }
    return 0;
}

// optimize the response of the sensor by
// setting filters to fast values once an
// interrupt was detected
// call this procedure in case of all interrupts:
// returns 1 if an interrupt was handled
```

```
char    host_optimization()
{
    char chipstatus;
    char interruptstatus;
    char interruptmask;
    char LPsettings;
    char mask;
    char i;
    intstatus new_status;



    //LED = 1;

    // read the chip status and
    // interrupt status to reset the interrupt of the chip
    // if this method is called via an interrupt, make
    // sure the interrupt was not read previously
    // otherwise use the chip status which may
    // be not up-to-date depending on the readout speed

    read_register(0, 32);
    interruptstatus = SMB_buf[18]; // interrupt status + chip status
    chipstatus = SMB_buf[19];

    // simulate here the interrupt since interrupt pin is not used on this
    board
    // read the interruptmask
    interruptmask = SMB_buf[25];
    // compare interrupstatus with the mask like the sensor is doing it
    if ((interruptstatus & interruptmask) == 0)
    {
        return 0;
    }

    // initialize the status
    new_status = current_status;

    // use the chip status to capture the current condition
    // interruptstatus = chipstatus;

    // initialize the chip for first time usage after power on
    //if (current_status == STATnointerrupt) new_status = STATresetting;

    // entrance condition to set fast filter setting
    // 1. Enter only when the new_status was not redefined by another condition
    // 2. must come from the no interrupt condition
    // 3. must be identified as presence flag after interrupt
    // 4. must have positive sign on presence flag
    // use chip status not to miss a current condition which was not triggered
    if (new_status == current_status && current_status == STATnointerrupt
        && (chipstatus & 0x08) != 0 && (chipstatus & 0x80) == 0)
        //&& (interruptstatus & 0x08) != 0 && (interruptstatus & 0x80) == 0)
    {
        new_status = STATsetting;
    }

    // entrance condition to set fast filter resetting
    // 1. Enter only when the new_status was not redefined by another condition
    // 2. must come from the no interrupt condition
    // 3. must be identified as presence flag after interrupt
    // 4. must have negative sign on presence flag
    // use chip status not to miss a current condition which was not triggered
    if (new_status == current_status && current_status == STATnointerrupt
        && (chipstatus & 0x08) != 0 && (chipstatus & 0x80) != 0)
        //&& (interruptstatus & 0x08) != 0 && (interruptstatus & 0x80) != 0)
    {
        new_status = STATresetting;
    }

    // entrance condition to enter the no interrupt status
    // 1. Enter only when the new_status was not redefined by another condition
    // 2. Enter only when the current_status is setting
    // 3. must have presence flag (filters catched up)
    // 4. Presence sign flag must indicate negative number
    // use interrupt status to look into the memory of the sensor
    if (new_status == current_status && current_status == STATsetting
        //&& (chipstatus & 0x08) != 0 && (chipstatus & 0x80) != 0
        && (interruptstatus & 0x08) != 0 && (interruptstatus & 0x80) != 0
        )
    {
        new_status = STATnointerrupt;
    }

    // entrance condition to enter the no interrupt status
    // 1. Enter only when the new_status was not redefined by another condition
    // 2. Enter only when the current_status is resetting
    // 3. must have presence flag (filters catched up)
    // 4. Presence sign flag must indicate positive number
    // use interrupt status to look into the memory of the sensor
    if (new_status == current_status && current_status == STATresetting
        //&& (chipstatus & 0x08) != 0 && (chipstatus & 0x80) == 0
        && (interruptstatus & 0x08) != 0 && (interruptstatus & 0x80) == 0
        )
    {
        new_status = STATnointerrupt;
    }

    if (new_status != current_status)
    {
        // configuration according to the new status
        if (new_status == STATsetting || new_status == STATresetting)
        {
            // make the background low pass follow the filtered signal nearly with
            nearly the same speed
            write_register(20, 0xCD);
            interruptmask = 0x08;
            write_register(25, interruptmask);
            // set now the presence threshold to a value of only 1 so that a sign
            change
            // will be triggered immediately
            write_register(22, 1);
        }

        if (new_status == STATnointerrupt)
        {
            // restore default filter settings and the threshold
            write_register(20, register_write[20]);
            write_register(22, register_write[22]);
            // prepare the system to go to sleep and listen only to the positive
            presence interrupt
            // trigger on the presence feature only and put your host to sleep
            interruptmask = 0x08;
            write_register(25, interruptmask);
        }

        // indication of new status
        if (new_status == STATresetting || new_status == STATsetting)
        {
            LED = 1;
            DEBUGPIN = 1;
        }
        else
        {
            LED = 0;
            DEBUGPIN = 0;
        }

        current_status = new_status;
        return 1;
    }
    return 0;
}

void main(void)
{
    // ...
    // reload SA from E2PROM
    general_call(0x04);
    // wait until the device is ready for communication
    delay_us(300);
    // write registers 20 to 29
    write_configuration();
    // check the correct settings
    read_configuration();
    while (1){
        // use either of both: presence_detection or host_optimization
        //while (presence_detection() != 0)
        {
            // repreat checking the sensor as long as the sensor keeps changing its
            configuration
        }

        while (host_optimization()!=0)
        {
            // repreat checking the sensor as long as the sensor keeps changing its
            configuration
        }
        // ...
    }
} // end main
```