Integration Guide

AD7147 Button Firmware

Introduction

The following guide attempts to describe the software integration process required to implement a simple 4 button capacitance sensor application. The AD7147 Button firmware configures the device and then reads the Interrupt Status Registers from the AD7147 when a sensor is activated.

AD7147 Configuration

On power-up the AD7147 registers must be configured to allow the part to function correctly in the application. This configuration step is done in the main system software; once the AD7147 register map is initialised an external host interrupt must then be configured. Touching a button will then cause an interrupt to fire which in turn causes software to jump to an Interrupt Service Routine (ISR) where the status of each button will be read from the AD7147. The following flowchart describes the AD7147 and ISR configuration process.

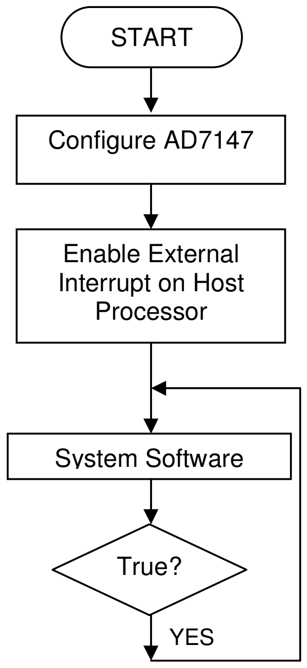


Fig.1

The AD7147 register configuration is done in two stages; firstly all 12 stages must be initialised with the sensor configuration including CIN connection, initial offset and threshold sensitivity information, there are 8 registers for each of the 12 stages. The second stage of the register configuration is to program the Bank 1 registers of the AD7147; these registers contain the power modes, environmental calibration and interrupt configuration settings.
Once these registers have been initialised a single read of the Interrupt Status registers is performed to clear the INT pin in case an interrupt was detected during the register configuration process, the part is then ready to respond to sensor activations immediately.

The AD7147 register configuration information is usually contained in an "AD7147 Config.h" file provided by ADI. The complete register configuration takes approximately 6.5mS with a 400kHz I2C interface as shown in Fig.2.

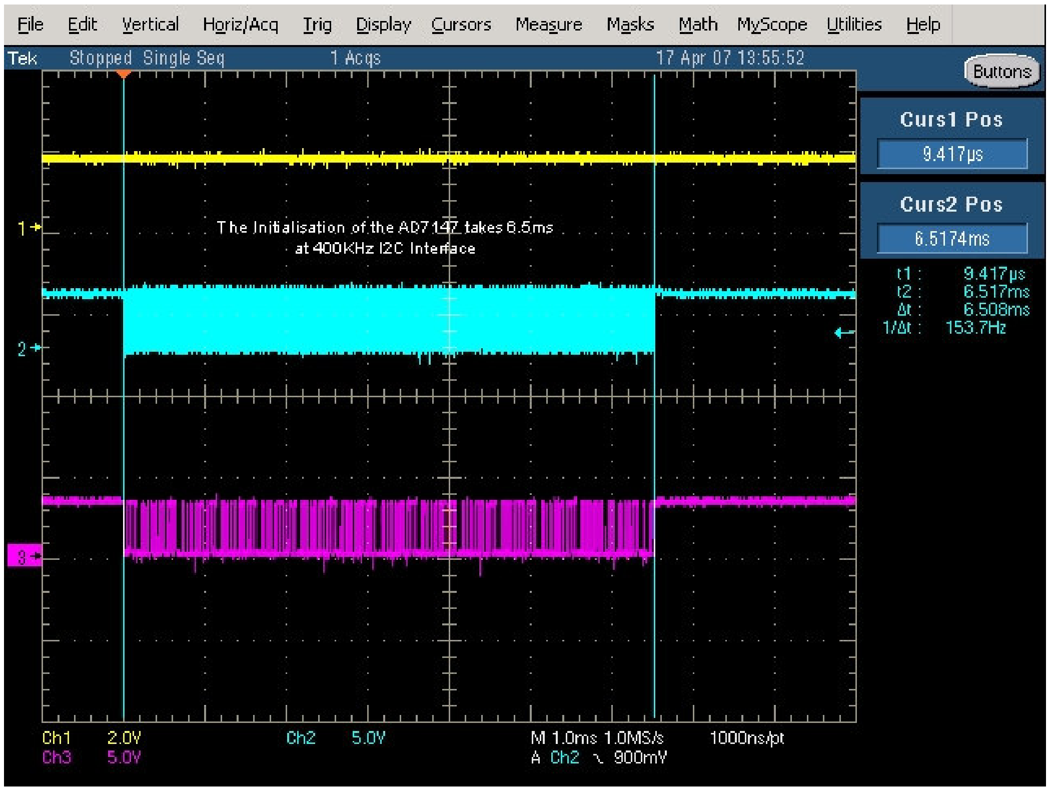
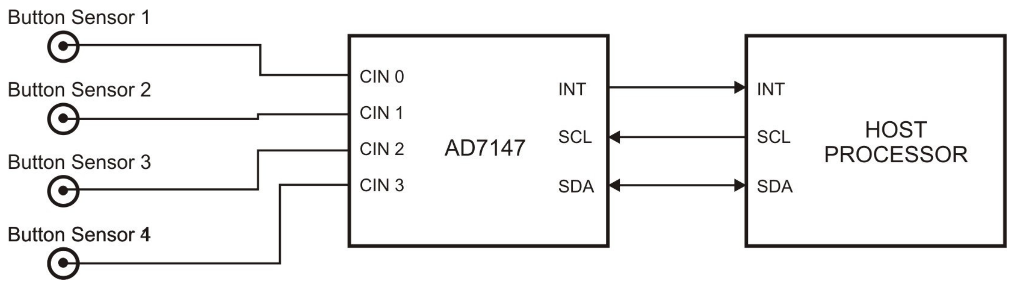

File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Tek Stopped Single Seq 1 Acqs 17 Apr 07 13:55:52
Buttons
Curs1 Pos 9.417µs
Curs2 Pos 6.5174ms
t1 : 9.417µs
t2 : 6.517ms
Δt : 6.508ms
1/Δt : 153.7Hz
The Initialisation of the AD7147 takes 6.5ms at 400KHz I2C Interface
Ch1 2.0V Ch2 5.0V M 1.0ms 1.0MS/s 1000ns/pt
Ch3 5.0V A Ch2 ↘ 900mV

Fig.2 AD7147 Register Configuration

Example - 4 Button Configuration

In the following AD7147 Config.h example, 4 buttons connected to CIN0, CIN1, CIN2 and CIN3 are configured to connect to the positive input of Stages 0 to 3 respectively, all other CIN inputs are not connected and bank 1 and bank 2 registers are initialised as follows,



```
//--------------------
//General Definitions
//--------------------

//Error Recovery - Stage 0 to 3, Any Low limit interrupt on
//these bits will force calibration

#define POWER_UP_INTERRUPT                    0x000F


//-----------------------
//Function declaration
//-----------------------
void ConfigAD7147(void);


//----------------------
//Function definition
//----------------------
void ConfigAD7147(void)
{

//============================================
//============ Bank 2 Registers ============
//============================================

WORD xdata StageBuffer[8];
//=======================
//= Stage 0 - CIN0 (+) =
//=======================
StageBuffer[0]=0xFFFE;  //Register 0x80
StageBuffer[1]=0x1FFF;  //Register 0x81
StageBuffer[2]=0x0000;  //Register 0x82
StageBuffer[3]=0x2626;  //Register 0x83
StageBuffer[4]=600;     //Register 0x84
StageBuffer[5]=500;     //Register 0x85
StageBuffer[6]=600;     //Register 0x86
StageBuffer[7]=600;     //Register 0x87
WriteToAD7147(STAGE0_CONNECTION, 8, StageBuffer, 0);


//=======================
//= Stage 1 - CIN1 (+) =
//=======================
StageBuffer[0]=0xFFFB;  //Register 0x88
StageBuffer[1]=0x1FFF;  //Register 0x89
StageBuffer[2]=0x0000;  //Register 0x8A
StageBuffer[3]=0x2626;  //Register 0x8B
StageBuffer[4]=950;     //Register 0x8C
StageBuffer[5]=800;     //Register 0x8D
StageBuffer[6]=950;     //Register 0x8E
StageBuffer[7]=950;     //Register 0x8F
WriteToAD7147(STAGE1_CONNECTION, 8, StageBuffer, 0);

//=======================
//= Stage 2 - CIN2 (+) =
//=======================
StageBuffer[0]=0xFFEF;  //Register 0x90
StageBuffer[1]=0x1FFF;  //Register 0x91
StageBuffer[2]=0x0000;  //Register 0x92
StageBuffer[3]=0x2626;  //Register 0x93
StageBuffer[4]=1400;    //Register 0x94
```

```
StageBuffer[5]=1200;    //Register 0x95
StageBuffer[6]=1400;    //Register 0x96
StageBuffer[7]=1400;    //Register 0x97
WriteToAD7147(STAGE2_CONNECTION, 8, StageBuffer, 0);

//=====================
//= Stage 3 - CIN3 (+) =
//=====================
StageBuffer[0]=0xFFBF;  //Register 0x98
StageBuffer[1]=0x1FFF;  //Register 0x99
StageBuffer[2]=0x0000;  //Register 0x9A
StageBuffer[3]=0x2626;  //Register 0x9B
StageBuffer[4]=2000;    //Register 0x9C
StageBuffer[5]=1800;    //Register 0x9D
StageBuffer[6]=2000;    //Register 0x9E
StageBuffer[7]=2000;    //Register 0x9F
WriteToAD7147(STAGE3_CONNECTION, 8, StageBuffer, 0);

//==========================
//= Stage 4 - Not Connected =
//==========================
StageBuffer[0]=0xFFFF;  //Register 0xA0
StageBuffer[1]=0x1FFF;  //Register 0xA1
StageBuffer[2]=0x0000;  //Register 0xA2
StageBuffer[3]=0x2626;  //Register 0xA3
StageBuffer[4]=2700;    //Register 0xA4
StageBuffer[5]=2500;    //Register 0xA5
StageBuffer[6]=2700;    //Register 0xA6
StageBuffer[7]=2700;    //Register 0xA7
WriteToAD7147(STAGE4_CONNECTION, 8, StageBuffer, 0);

//==========================
//= Stage 5 - Not Connected =
//==========================
StageBuffer[0]=0xFFFF;  //Register 0xA8
StageBuffer[1]=0x1FFF;  //Register 0xA9
StageBuffer[2]=0x0000;  //Register 0xAA
StageBuffer[3]=0x2626;  //Register 0xAB
StageBuffer[4]=3350;    //Register 0xAC
StageBuffer[5]=3000;    //Register 0xAD
StageBuffer[6]=3350;    //Register 0xAE
StageBuffer[7]=3350;    //Register 0xAF
WriteToAD7147(STAGE5_CONNECTION, 8, StageBuffer, 0);

//==========================
//= Stage 6 - Not Connected =
//==========================
StageBuffer[0]=0xFFFF;  //Register 0xB0
StageBuffer[1]=0x1FFF;  //Register 0xB1
StageBuffer[2]=0x0000;  //Register 0xB2
StageBuffer[3]=0x2626;  //Register 0xB3
StageBuffer[4]=650;     //Register 0xB4
StageBuffer[5]=500;     //Register 0xB5
StageBuffer[6]=650;     //Register 0xB6
StageBuffer[7]=650;     //Register 0xB7
WriteToAD7147(STAGE6_CONNECTION, 8, StageBuffer, 0);

//==========================
//= Stage 7 - Not Connected =
//==========================
StageBuffer[0]=0xFFFF;  //Register 0xB8
StageBuffer[1]=0x1FFF;  //Register 0xB9
StageBuffer[2]=0x0000;  //Register 0xBA
```

```
StageBuffer[3]=0x2626;  //Register 0xBB
StageBuffer[4]=1150;    //Register 0xBC
StageBuffer[5]=1000;    //Register 0xBD
StageBuffer[6]=1150;    //Register 0xBE
StageBuffer[7]=1150;    //Register 0xBF
WriteToAD7147(STAGE7_CONNECTION, 8, StageBuffer, 0);

//============================
//= Stage 8 - Not Connected =
//============================
StageBuffer[0]=0xFFFF;  //Register 0xC0
StageBuffer[1]=0x1FFF;  //Register 0xC1
StageBuffer[2]=0x0000;  //Register 0xC2
StageBuffer[3]=0x2626;  //Register 0xC3
StageBuffer[4]=1800;    //Register 0xC4
StageBuffer[5]=1600;    //Register 0xC5
StageBuffer[6]=1800;    //Register 0xC6
StageBuffer[7]=1800;    //Register 0xC7
WriteToAD7147(STAGE8_CONNECTION, 8, StageBuffer, 0);

//============================
//= Stage 9 - Not Connected =
//============================
StageBuffer[0]=0xFFFF;  //Register 0xC8
StageBuffer[1]=0x1FFF;  //Register 0xC9
StageBuffer[2]=0x0000;  //Register 0xCA
StageBuffer[3]=0x2626;  //Register 0xCB
StageBuffer[4]=2400;    //Register 0xCC
StageBuffer[5]=2200;    //Register 0xCD
StageBuffer[6]=2400;    //Register 0xCE
StageBuffer[7]=2400;    //Register 0xCF
WriteToAD7147(STAGE9_CONNECTION, 8, StageBuffer, 0);

//=============================
//= Stage 10 - Not Connected =
//=============================
StageBuffer[0]=0xFFFF;  //Register 0xD0
StageBuffer[1]=0x1FFF;  //Register 0xD1
StageBuffer[2]=0x0000;  //Register 0xD2
StageBuffer[3]=0x2626;  //Register 0xD3
StageBuffer[4]=3400;    //Register 0xD4
StageBuffer[5]=3200;    //Register 0xD5
StageBuffer[6]=3400;    //Register 0xD6
StageBuffer[7]=3400;    //Register 0xD7
WriteToAD7147(STAGE10_CONNECTION, 8, StageBuffer, 0);


//=============================
//= Stage 11 - Not Connected =
//=============================
StageBuffer[0]=0xFFFF;  //Register 0xD8
StageBuffer[1]=0x1FFF;  //Register 0xD9
StageBuffer[2]=0x0000;  //Register 0xDA
StageBuffer[3]=0x2626;  //Register 0xDB
StageBuffer[4]=4400;    //Register 0xDC
StageBuffer[5]=4200;    //Register 0xDD
StageBuffer[6]=4400;    //Register 0xDE
StageBuffer[7]=4400;    //Register 0xDF
WriteToAD7147(STAGE11_CONNECTION, 8, StageBuffer, 0);

//===========================================
//=========== Bank 1 Registers ===========
//===========================================
//Initialisation of Bank 1 Registers

AD7147Registers[PWR_CONTROL]=0x00B2;  //Register 0x00
WriteToAD7147(PWR_CONTROL, 1, AD7147Registers, PWR_CONTROL);

AD7147Registers[AMB_COMP_CTRL0]=0x3230;  //Register 0x02
AD7147Registers[AMB_COMP_CTRL1]=0x0419;  //Register 0x03
AD7147Registers[AMB_COMP_CTRL2]=0x0832;  //Register 0x04
AD7147Registers[STAGE_LOW_INT_EN]=POWER_UP_INTERRUPT; //0x05
AD7147Registers[STAGE_HIGH_INT_EN]=0x000F; //Register 0x06
AD7147Registers[STAGE_COMPLETE_INT_EN]=0x0000; //Register 0x07
WriteToAD7147(AMB_COMP_CTRL0, 6, AD7147Registers,
AMB_COMP_CTRL0);

//Enable calibration on 4 stages
AD7147Registers[STAGE_CAL_EN]=0x000F;  //Register 0x01
WriteToAD7147(STAGE_CAL_EN, 1, AD7147Registers, STAGE_CAL_EN);

//Read High and Low Limit Status registers to clear INT pin
ReadFromAD7147(STAGE_LOW_LIMIT_INT, 2, AD7147Registers,
STAGE_LOW_LIMIT_INT); //Registers 0x08 & 0x09
}
```

Hardware Interrupt Configuration

```
AD7147Registers[STAGE_LOW_INT_EN] = POWER_UP_INTERRUPT;
AD7147Registers[STAGE_HIGH_INT_EN] = 0x000F;
AD7147Registers[STAGE_COMPLETE_INT_EN] = 0x0000;

AD7147Registers[STAGE_HIGH_INT_EN] = 0x000F;
```

This configures the High Limit Interrupt Enable Register 0x06 Bits 0-3; if any of these bits are set, then a hardware interrupt is generated on the INT pin when a sensor is activated and a high limit threshold is exceeded on the AD7147. A second interrupt is generated when the user lifts off the sensor as shown in Fig.3

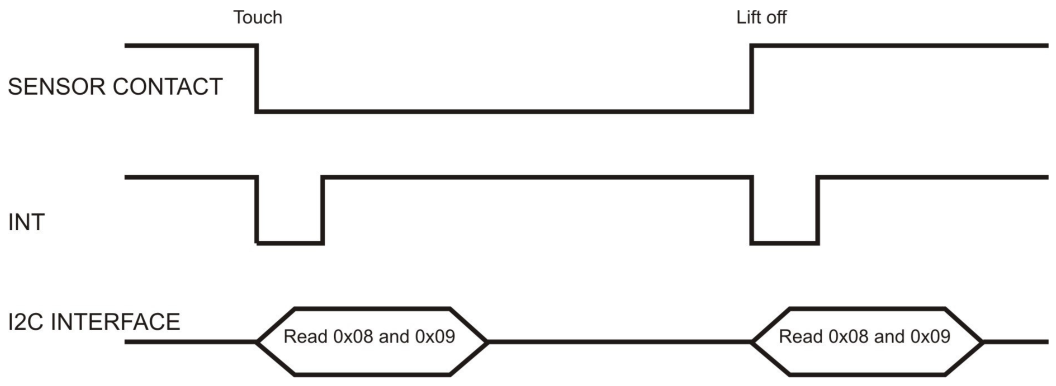


Fig.3 Interrupt Sequence

```
AD7147Registers[STAGE_LOW_INT_EN] = POWER_UP_INTERRUPT;
```

This configures the Low Limit Interrupt Enable Register 0x05 Bits 0-3; this will only generate a hardware interrupt if an error needs to be corrected by a forced recalibration on the AD7147. There are two sources of error that may need to be corrected:

1. If the sensor is touched when the part is powered up, the initial sensor thresholds calculated by the AD7147 will be incorrect, when the user lifts off the sensor a low limit threshold will be asserted and software must then recalibrate the part in the interrupt service routine
2. The second error that may occur is when the sensor value drifts below the low limit threshold due to excessive temperature or humidity errors. In this case the recalibration function in the interrupt service routine also needs to be called.

```
AD7147Registers[STAGE_COMPLETE_INT_EN] = 0x0000;
```

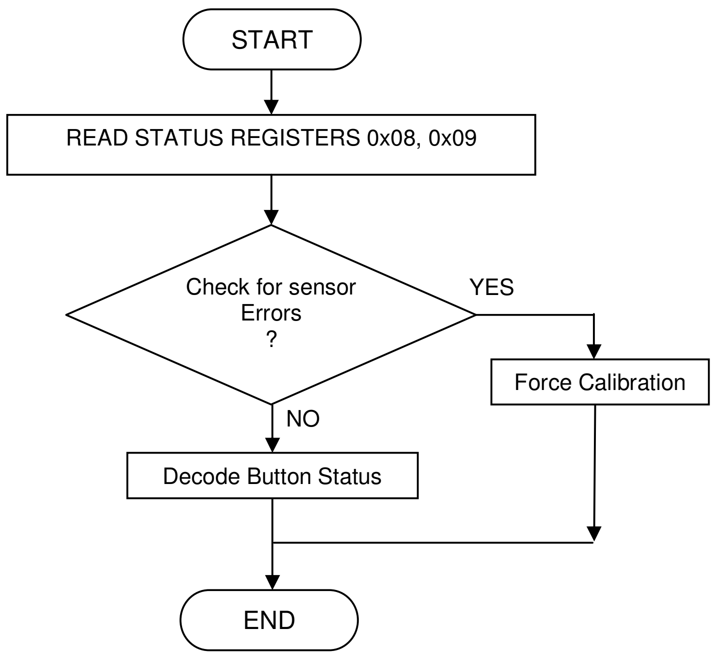
If this Register were set to 0x0001 then hardware interrupts would be asserted after each Stage0 conversion regardless of whether we are touching the sensor or not. CDC results would then be available for all 12 stages.

```
AD7147Registers[STAGE_CAL_EN] = 0x000F;
```

In this configuration file, the STAGE_CAL_EN bits are set to 0x000F, this enables the environmental calibration and adaptive threshold logic on stage 0 stage 1, stage 2 and stage 3 only as these are the only stages used in this application. Any unused or un-configured stages should not be enabled in the STAGE_CAL_EN register. E.g. When using an 8 channel device such as the AD7148; a maximum of 8 calibration stages should only be configured in the STAGE_CAL_EN register.

Interrupt Service Routine (ISR)

The interrupt service routine is executed every time the AD7147 generates a hardware interrupt on the INT pin. This interrupt is generated when the user touches the sensor and lifts off from the sensor or when a recalibration event is required. The AD7147 does not continuously generate hardware interrupts in this mode while the sensor is being touched. As shown in Fig.2 the ISR is only called twice, once for a touch and a second time when the user leaves the sensor, therefore software always knows the current state of the sensors while not having to continuously service hardware interrupts. The INT pin on the AD7147 is cleared by a read of the Interrupt status registers, 0x08 and 0x09.



This flowchart above and the following example code describe the button processing code implemented in the Interrupt Service Routine. As soon as a button is touched the INT pin on the AD7147 is asserted which causes a hardware interrupt on the host processor to call the Interrupt Service Routine. Once the ISR is called the first thing that is done is a read from the Low Limit and High Limit status registers, these registers contain the current sensor status.

Recalibration

Next the code checks if a recalibration of the sensors is necessary in case an error has occurred as described below. If a low limit status bit is set which signifies an error then the ForceCalibration() function is called which forces a recalibration of the sensors by writing to the Forced_CAL bit<14> in the Ambient Compensation Control register, 0x02. A simple way to test the functionality of this code is to physically touch the sensor while power is applied to the part and the sensors are configured by the ConfigAD7147(void) function, when the use lifts off the sensor this software routine should recalibrate the sensors to the untouched value allowing the sensor to function correctly afterwards as shown in Fig.4

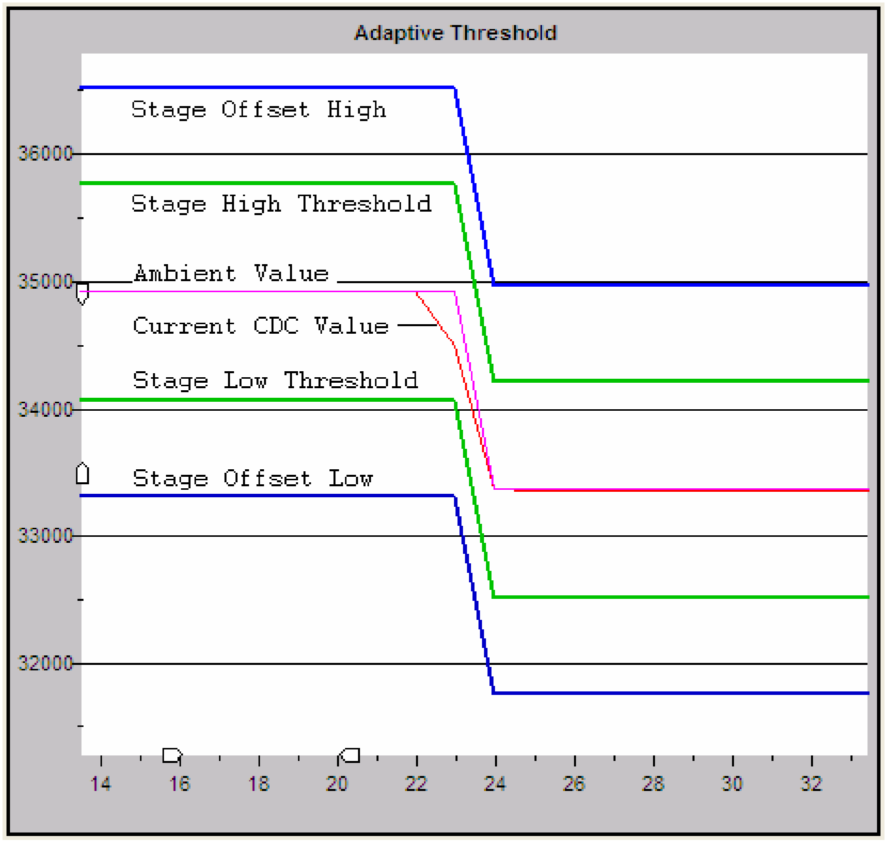


Fig.4

It is absolutely necessary to have the recalibration routine in your code, this software does two things. Firstly it recalibrates the sensor response if the user has been touching the sensor while the part is being configured after power up. If the user has been touching the sensor while the part is initialised the upper and lower sensor thresholds will be set at the incorrect levels around the touched sensor value, therefore without the recalibration code the high sensor threshold would never get set as it would always be higher than the touched sensor value. The recalibration code determines if a low threshold limit is exceeded as the user lifts off the sensor after part initialisation and then forces a recalibration of the sensor thresholds on chip so that they now centre around the untouched sensor value which is correct. When the user then touches the sensor again the high threshold will be exceeded as normal. Secondly the recalibration code corrects for various errors that may occur causing the sensor value to drift lower very quickly due to excessive temperature drifts, if the sensor value drifts below the low limit threshold the recalibration code will then re-centre the high and low thresholds around the current sensor value.

Button Decode

If an error was not detected then the button status is decoded in the DecodeButtons() function as shown in the example code below. The ISR is then completed and host software can read the button status variable and process the information accordingly.

```
//ServiceAD7147Isr();
//-------------------------------------------------------------------
// Function called by the AD7147 ISR. Anything that must be
// executed during the ISR needs to be done here
//-------------------------------------------------------------------
static int ServiceAD7147Isr(void)
{
//Read Low Limit and High Limit Status Registers
ReadFromAD7147(STAGE_LOW_LIMIT_INT, 2, AD7147Registers,
               STAGE_LOW_LIMIT_INT);

//Recover from errors if needed
if (((AD7147Registers[STAGE_LOW_LIMIT_INT] & POWER_UP_INTERRUPT)!=
      0x0000) && ((AD7147Registers[STAGE_HIGH_LIMIT_INT] &
      POWER_UP_INTERRUPT) == 0x0000))
{
 ForceCalibration();
}
else
{
 ButtonStatus = DecodeButtons(AD7147Registers[STAGE_HIGH_LIMIT_INT]);
}


// ForceCalibration();
//-------------------------------------------------------------
// Function called by ServiceAD7147Isr when there is a touch on
//power up or when there is a problem with the //calibration.
//--------------------------------------------------------------
static void ForceCalibration(void)
{
 ReadFromAD7147(AMB_COMP_CTRL0, 1, AD7147Registers, AMB_COMP_CTRL0);
 AD7147Registers[AMB_COMP_CTRL0] |= 0x4000;
 WriteToAD7147(AMB_COMP_CTRL0, 1, AD7147Registers, AMB_COMP_CTRL0);
}


//DecodeButtons()
//--------------------------------------------------------------
// Function called by ServiceAD7147Isr. This function //retrieves the
// button being set based on the High Limit Status Register.
//--------------------------------------------------------------
static WORD DecodeButtons(const WORD HighLimitStatusRegister)
{
 WORD ButtonStatusValue=0;

 if ((HighLimitStatusRegister & 0x0001) == 0x0001)
    ButtonStatusValue |= 0x0001;

 if ((HighLimitStatusRegister & 0x0002) == 0x0002)
    ButtonStatusValue |= 0x0002;

 if ((HighLimitStatusRegister & 0x0004) == 0x0004)
    ButtonStatusValue |= 0x0004;

 if ((HighLimitStatusRegister & 0x0008) == 0x0008)
    ButtonStatusValue |= 0x0008;

 return (ButtonStatusValue);
}
```

Code and data memory requirements

The following table lists the program code and data memory required to implement the AD7147 register configuration, recalibration and button decoding on a host controller. These code sizes do not include any software I2C or SPI driver; it assumes the host processor will have a dedicated hardware driver. As an example to add a software I2C driver requires an additional 726 bytes of code memory and 16 bytes of data memory. Also, additional code memory may be required to perform specific functions based on button combinations etc.

Memory	**No. Buttons**					
	2	4	6	8	10	12
Code	0.574kB	0.737kB	0.903kB	1.069kB	1.232kB	1.398kB
Data	0.106kB	0.106kB	0.106kB	0.106kB	0.106kB	0.106kB