Integration Guide

AD7147

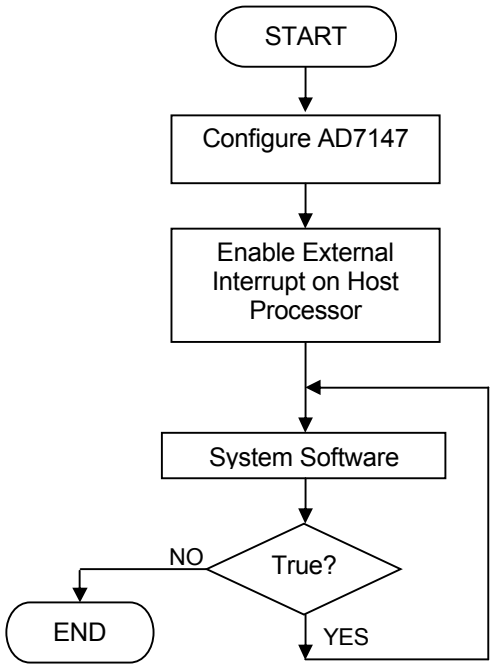
Scroll wheel Firmware

Introduction

The following guide attempts to describe the software integration process required to implement a scrollwheel sensor application.
The AD7147 Scrollwheel firmware configures the device and then reads the Interrupt Status Registers, the ADC results and the Ambient values for each stage of the sensor 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 any area of the scrollwheel will then cause an interrupt to fire which in turn causes software to jump to an Interrupt Service Routine (ISR) where registers will be read from the AD7147 and the absolute position will be computed.
The following flowchart describes the AD7147 and ISR configuration process.



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.1.

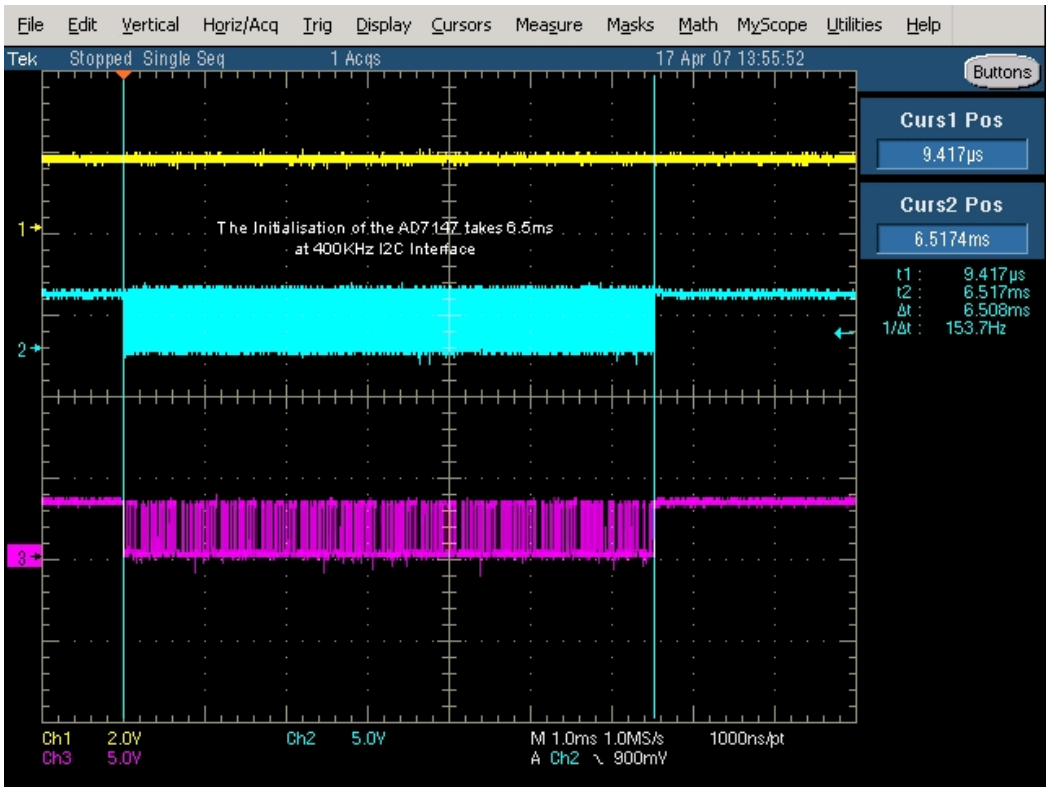


Fig.1 AD7147 Register Configuration

Scrollwheel Configuration

In the following "AD7147 Scrollwheel Config.h" file, a scrollwheel sensor made out of 8 sensors is connected to CIN0, CIN1… CIN7. The inputs are configured to connect to the positive input of Stages 0 to 11 respectively, all other CIN inputs are not connected and bank 1 and bank 2 registers are initialised as follows:

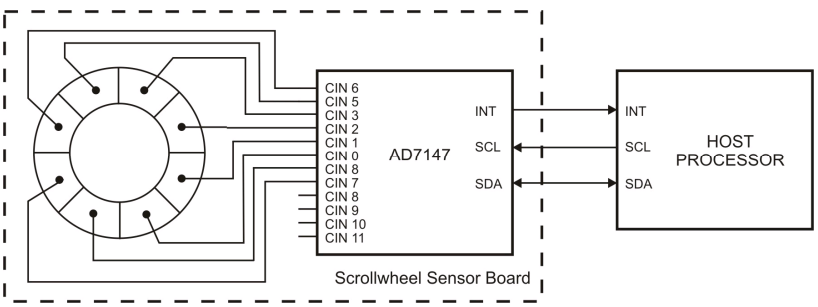


Fig.2 AD7147 Connection Diagram

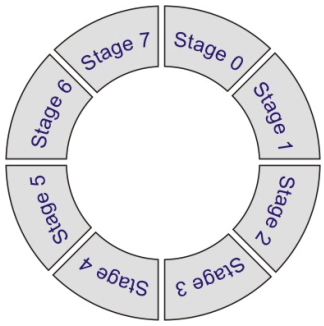


Fig.3 Mapping of the stages on the scrollwheel

```
//----------------------
//Function declarations
//----------------------
void ConfigAD7147(void);


//---------------------
//Function definitions
//---------------------
void ConfigAD7147(void)
{
   WORD xdata StageBuffer[8];


   //========================
   //= Stage 0 CIN3(+) S1   =
   //========================
   StageBuffer[0]=0xFFBF;   //Register  0x80
   StageBuffer[1]=0x1FFF;   //Register  0x81
   StageBuffer[2]=0x0100;   //Register  0x82
   StageBuffer[3]=0x2121;   //Register  0x83
   StageBuffer[4]=4000;     //Register  0x84
   StageBuffer[5]=4000;     //Register  0x85
   StageBuffer[6]=4250;     //Register  0x86
   StageBuffer[7]=4250;     //Register  0x87
   WriteToAD7147(STAGE0_CONNECTION, 8, StageBuffer, 0);

   //========================
   //= Stage 1 - CIN2(+) S2 =
   //========================
   StageBuffer[0]=0xFFEF;   //Register  0x88
   StageBuffer[1]=0x1FFF;   //Register  0x89
   StageBuffer[2]=0x0100;   //Register  0x8A
   StageBuffer[3]=0x2121;   //Register  0x8B
   StageBuffer[4]=4000;     //Register  0x8C
   StageBuffer[5]=4000;     //Register  0x8D
   StageBuffer[6]=4250;     //Register  0x8E
   StageBuffer[7]=4250;     //Register  0x8F
   WriteToAD7147(STAGE1_CONNECTION, 8, StageBuffer, 0);


   //========================
   //= Stage 2 - CIN1(+) S3 =
   //========================
```

```
StageBuffer[0]=0xFFFB;   //Register  0x90
StageBuffer[1]=0x1FFF;   //Register  0x91
StageBuffer[2]=0x0100;   //Register  0x92
StageBuffer[3]=0x2121;   //Register  0x93
StageBuffer[4]=4000;     //Register  0x94
StageBuffer[5]=4000;     //Register  0x95
StageBuffer[6]=4250;     //Register  0x96
StageBuffer[7]=4250;     //Register  0x97
WriteToAD7147(STAGE2_CONNECTION, 8, StageBuffer, 0);

//=========================
//= Stage 3 - CIN0(+) S4 =
//=========================
StageBuffer[0]=0xFFFE;   //Register  0x98
StageBuffer[1]=0x1FFF;   //Register  0x99
StageBuffer[2]=0x0100;   //Register  0x9A
StageBuffer[3]=0x2121;   //Register  0x9B
StageBuffer[4]=4000;     //Register  0x9C
StageBuffer[5]=4000;     //Register  0x9D
StageBuffer[6]=4250;     //Register  0x9E
StageBuffer[7]=4250 ;    //Register  0x9F
WriteToAD7147(STAGE3_CONNECTION, 8, StageBuffer, 0);

//=========================
//= Stage 4 - CIN8(+) S5 =
//=========================
StageBuffer[0]=0xFFFF;  //Register  0xA0
StageBuffer[1]=0x1FFB;  //Register  0xA1
StageBuffer[2]=0x0100;  //Register  0xA2
StageBuffer[3]=0x2121;  //Register  0xA3
StageBuffer[4]=4000;     //Register  0xA4
StageBuffer[5]=4000;     //Register  0xA5
StageBuffer[6]=4250;     //Register  0xA6
StageBuffer[7]=4250;     //Register  0xA7
WriteToAD7147(STAGE4_CONNECTION, 8, StageBuffer, 0);

//=========================
//= Stage 5 - CIN7(+) S6 =
//=========================
StageBuffer[0]=0xFFFF;   //Register  0xA8
StageBuffer[1]=0x1FFE;   //Register  0xA9
StageBuffer[2]=0x0100;   //Register  0xAA
StageBuffer[3]=0x2121;   //Register  0xAB
StageBuffer[4]=4000;     //Register  0xAC
StageBuffer[5]=4000;     //Register  0xAD
StageBuffer[6]=4250;     //Register  0xAE
StageBuffer[7]=4250;     //Register  0xAF
WriteToAD7147(STAGE5_CONNECTION, 8, StageBuffer, 0);

//=========================
//= Stage 6 - CIN6(+) S7 =
//=========================
StageBuffer[0]=0xEFFF;//Register  0xB0
StageBuffer[1]=0x1FFF;//Register  0xB1
StageBuffer[2]=0x0100;//Register  0xB2
StageBuffer[3]=0x2121;   //Register  0xB3
StageBuffer[4]=4000;     //Register  0xB4
StageBuffer[5]=4000;     //Register  0xB5
StageBuffer[6]=4250;     //Register  0xB6
StageBuffer[7]=4250;     //Register  0xB7
WriteToAD7147(STAGE6_CONNECTION, 8, StageBuffer, 0);
```

```
//=========================
//= Stage 7 - CIN5(+) S8 =
//=========================
StageBuffer[0]=0xFBFF;   //Register  0xB8
StageBuffer[1]=0x1FFF;   //Register  0xB9
StageBuffer[2]=0x0100;   //Register  0xBA
StageBuffer[3]=0x2121;   //Register  0xBB
StageBuffer[4]=4000;     //Register  0xBC
StageBuffer[5]=4000;     //Register  0xBD
StageBuffer[6]=4250;     //Register  0xBE
StageBuffer[7]=4250;     //Register  0xBF
WriteToAD7147(STAGE7_CONNECTION, 8, StageBuffer, 0);

//============================
//= Stage 8 - NO CONNECTION =
//============================
StageBuffer[0]=0xFFFF;   //Register  0xC0
StageBuffer[1]=0x3FFF;   //Register  0xC1
StageBuffer[2]=0x0000;   //Register  0xC2
StageBuffer[3]=0x2626;   //Register  0xC3
StageBuffer[4]=3000;     //Register  0xC4
StageBuffer[5]=3000;     //Register  0xC5
StageBuffer[6]=4000;     //Register  0xC6
StageBuffer[7]=4000;     //Register  0xC7
WriteToAD7147(STAGE8_CONNECTION, 8, StageBuffer, 0);

//============================
//= Stage 9 - NO CONNECTION =
//============================
StageBuffer[0]=0xFFFF;   //Register  0xC8
StageBuffer[1]=0x3FFF;   //Register  0xC9
StageBuffer[2]=0x0000;   //Register  0xCA
StageBuffer[3]=0x2626;   //Register  0xCB
StageBuffer[4]=3000;     //Register  0xCC
StageBuffer[5]=3000;     //Register  0xCD
StageBuffer[6]=4000;     //Register  0xCE
StageBuffer[7]=4000;     //Register  0xCF
WriteToAD7147(STAGE9_CONNECTION, 8, StageBuffer, 0);

//============================
//= Stage 10 - NO CONNECTION =
//============================
StageBuffer[0]=0xFFFF;   //Register  0xD0
StageBuffer[1]=0x3FFF;   //Register  0xD1
StageBuffer[2]=0x0000;   //Register  0xD2
StageBuffer[3]=0x2626;   //Register  0xD3
StageBuffer[4]=3000;     //Register  0xD4
StageBuffer[5]=3000;     //Register  0xD5
StageBuffer[6]=4000;     //Register  0xD6
StageBuffer[7]=4000;     //Register  0xD7
WriteToAD7147(STAGE10_CONNECTION, 8, StageBuffer, 0);

//============================
//= Stage 11 - NO CONNECTION =
//============================
StageBuffer[0]=0xFFFF;   //Register  0xD8
StageBuffer[1]=0x3FFF;   //Register  0xD9
StageBuffer[2]=0x0000;   //Register  0xDA
StageBuffer[3]=0x2626;   //Register  0xDB
StageBuffer[4]=3000;     //Register  0xDC
```

```
   StageBuffer[5]=3000;     //Register  0xDD
   StageBuffer[6]=4000;     //Register  0xDE
   StageBuffer[7]=4000;     //Register  0xDF
   WriteToAD7147(STAGE11_CONNECTION, 8, StageBuffer, 0);


   //-------------------------------------------------------------//
   //--------------------- Bank 1 Registers ----------------------//
   //-------------------------------------------------------------//

   //Initialisation of the 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]=0x0000; //Register 0x05
   AD7147Registers[STAGE_HIGH_INT_EN]=0x0000; //Register 0x06
   AD7147Registers[STAGE_COMPLETE_INT_EN]=0x0001; //Register 0x07
   WriteToAD7147(AMB_COMP_CTRL0, 6, AD7147Registers, AMB_COMP_CTRL0);

   //Enable calibration on 8 stages
   AD7147Registers[STAGE_CAL_EN]=0x00FF; //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, 3, AD7147Registers,
                  STAGE_LOW_LIMIT_INT);//Registers 0x08, 0x09 and 0x0A
}
```

Hardware Interrupt Configuration

In the scrollwheel application firmware, ADI developed a routine that configures the AD7147 either in End of Conversion interrupt mode or Threshold interrupt mode. On power up the AD7147 is configured in End of Conversion Interrupt mode. This mode is enabled by configuring the 3 interrupt enable registers as follow:

```
AD7147Registers[STAGE_LOW_INT_EN] = 0x0000;   //Register 0x05
AD7147Registers[STAGE_HIGH_INT_EN] = 0x0000;  //Register 0x06
AD7147Registers[STAGE_COMPLETE_INT_EN]=0x0001;//Register 0x07

AD7147Registers[STAGE_COMPLETE_INT_EN] = 0x0001;
```

This configures the Stage Complete Interrupt Enable Register 0x07 Bit <0> so that the INT pin will be asserted on completion of each stage 0 ADC conversion, all the stages will also be converted on each interrupt cycle. The STAGE_0_COMPLETE_STATUS_INT bit in the STAGE_COMPLETE_LIMIT_INT register (address 0x0A) is set automatically by the AD7147 when it has completed the conversion of all stages. When this bit is set, the interrupt pin is asserted which causes software to jump to an Interrupt Service Routine where the 3 interrupt status registers at address 0x08, 0x09 and 0x0A are read to clear the hardware interrupt.

In the End of Conversion interrupt mode the frequency of the interrupt depends on bit 8 and bit 9 of the Power Control Register (referenced as `AD7147Registers[PWR_CONTROL]` in the code at address 0x00). These bits control the decimation rate.

Bit 8	Bit 9	Decimation rate	Frequency of the Interrupt for 12 conversion stages
0	0	256	36ms
0	1	128	18ms
1	0	64	9ms
1	1	64	9ms

Table.1 Interrupt Frequency

In order to minimise the interrupt frequency to the host, ADI implemented a software routine in the ISR which changes the AD7147 interrupt mode depending on whether the user is contacting the sensor or not. When the user touches the sensor, the firmware configures the AD7147 in End of Conversion interrupt mode. When the user lifts off the sensor, the firmware configures the AD7147 in Threshold mode shortly after lifting off.

This is the code that implements this functionality:

```
/************************/
/* Change interrupt mode */
/************************/
if (((AD7147Registers[STAGE_HIGH_LIMIT_INT] &
      POWER_UP_INTERRUPT)!=0x0000) ||
     ((AD7147Registers[STAGE_LOW_LIMIT_INT] &
      POWER_UP_INTERRUPT)!=0x0000))
{
    //Configure the AD7147 in End of Conversion Interrupt mode
    if (AD7147Registers[STAGE_COMPLETE_INT_EN]==0x0000)
    {
        AD7147Registers[STAGE_LOW_INT_EN] &= 0xF000;
        AD7147Registers[STAGE_HIGH_INT_EN] &= 0xF000;
        AD7147Registers[STAGE_COMPLETE_INT_EN]=0x0001;
        WriteToAD7147(STAGE_LOW_INT_EN, 3, AD7147Registers,
                      STAGE_LOW_INT_EN);
    }
    InterruptCounterForBtnIntMode=NUMBER_OF_INTS_BEFORE_BTN_INT_MODE;
}
else
{
    //Configure the AD7147 in threshold interrupt mode
    if (InterruptCounterForBtnIntMode>0)
        InterruptCounterForBtnIntMode--;
    if (AD7147Registers[STAGE_HIGH_INT_EN]==0x0000 &&
        InterruptCounterForBtnIntMode==0)
    {
        AD7147Registers[STAGE_LOW_INT_EN] |= POWER_UP_INTERRUPT;
        AD7147Registers[STAGE_HIGH_INT_EN] |= 0x01FF;
        AD7147Registers[STAGE_COMPLETE_INT_EN]=0x0000;
```

```
        WriteToAD7147(STAGE_LOW_INT_EN, 3, AD7147Registers,
                      STAGE_LOW_INT_EN);
    }
}
```

When the sensor is not being touched, the AD7147 is configured in threshold mode so that no hardware interrupt are generated by the AD7147. When the sensor is touched a bit will be set in the Stage High Limit Interrupt Status register (address 0x09) and a hardware interrupt will be generated. As soon as a bit is set in this register, we clear bit 0 to 11 in the High and Low Limit Interrupt Enable registers and set bit 0 in the Stage Complete Interrupt Enable Register (address 0x07).

```
AD7147Registers[STAGE_LOW_INT_EN] &= 0xF000;
AD7147Registers[STAGE_HIGH_INT_EN] &= 0xF000;
AD7147Registers[STAGE_COMPLETE_INT_EN]=0x0001;
WriteToAD7147(STAGE_LOW_INT_EN,3,AD7147Registers,STAGE_LOW_INT_EN);
```

When the user lifts off the sensor, the firmware waits for 8 End of Conversion interrupts and changes the AD7147 back into threshold interrupt mode by writing to the following registers:

```
AD7147Registers[STAGE_LOW_INT_EN] |= POWER_UP_INTERRUPT;
AD7147Registers[STAGE_HIGH_INT_EN] |= 0x00FF;
AD7147Registers[STAGE_COMPLETE_INT_EN]=0x0000;
WriteToAD7147(STAGE_LOW_INT_EN,3,AD7147Registers,STAGE_LOW_INT_EN);
```

`POWER_UP_INTERRUPT` has the value `0x01FF`.

`AD7147Registers[STAGE_HIGH_INT_EN] |= 0x01FF;`
This configures the High Limit Interrupt Enable Register 0x06 Bits 0-7; 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.

`AD7147Registers[STAGE_LOW_INT_EN] |= POWER_UP_INTERRUPT;`
This configures the Low Limit Interrupt Enable Register 0x05 Bits 0-7; 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] = 0x01FF;
```

In this configuration file, the STAGE_CAL_EN bits are set to 0x01FF, this enables the environmental calibration and adaptive threshold logic from stage 0 to stage 8 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 following diagram shows the interrupt sequence used in the scrollwheel application.

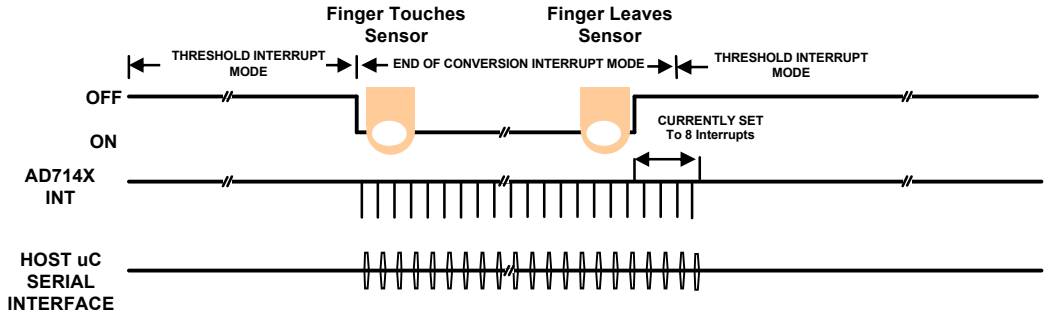


Fig.4 Interrupt Sequence

The interrupt service routine is executed every time the AD7147 generates a hardware interrupt on the INT pin. This interrupt is generated in the following conditions:

- When the user touches the sensor
- When a recalibration event is required
- Every 9ms, 18ms or 36ms when the AD7147 is operating in End of Conversion interrupt mode

In End of Conversion interrupt mode, the AD7147 continuously generates hardware interrupts while the sensor is being touched. As shown in Fig.4 the ISR is called every 9ms, 18ms or 36ms depending on the decimation rate.

The INT pin on the AD7147 is cleared by a read of the Interrupt status registers, 0x08, 0x09 and 0x0A. However, on the very first interrupt, the firmware will write to registers 0x05, 0x06 and 0x07 to change the AD7147 from Threshold interrupt mode to End of Conversion interrupt mode.

Once the user lifts off the sensor software will write again to registers 0x05, 0x06 and 0x07 to change the AD7147 back to Threshold Interrupt mode.

Recalibration

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 (at address 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()

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.5

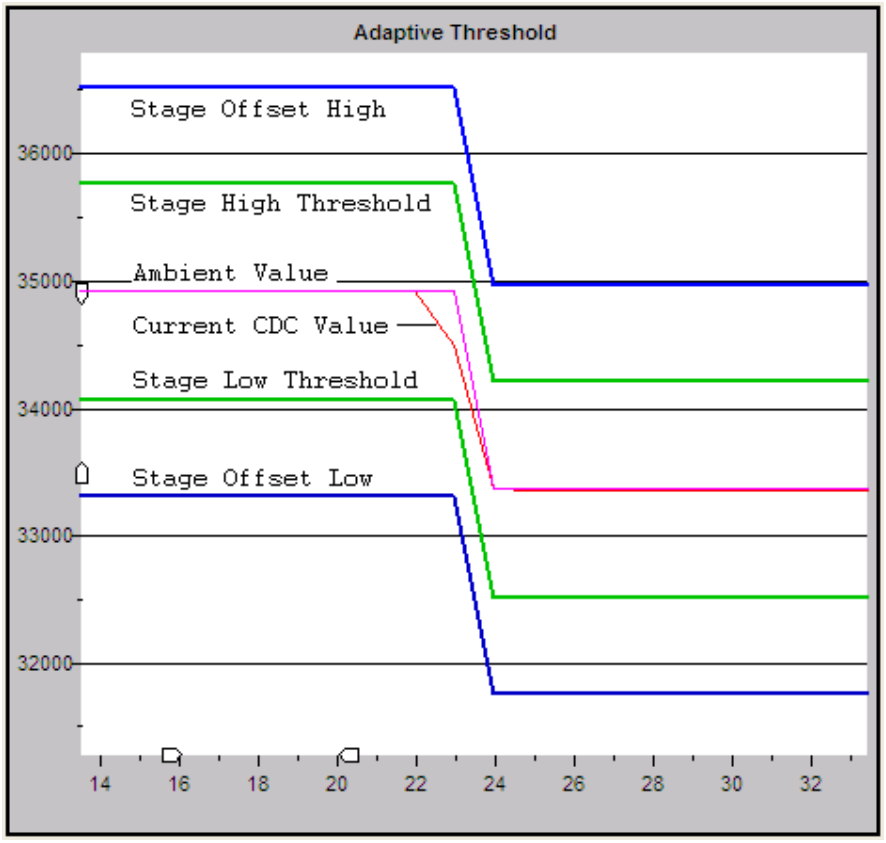


Fig.5 Recalibration

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.

Scroll wheel Algorithm

The following pseudo code describes the scrollwheel processing code implemented in the Interrupt Service Routine.

```
BEGIN
    Read Interrupt Status Registers at 0x008, 0x009, 0x00A
    //Read from AD7147
    FOREACH Stage
        Read ADC Values
        Read Ambient Values
        Read Max Average Values
    NEXT

    Increment Interrupt Counter

    IF Interrupt counter = 2 THEN
        //Initialisation
        Initialise scrollwheel algorithm
    ELSE IF Interrupt counter > 2 THEN
        //Recalibration
        IF Sensor Error is detected THEN
            Force Calibration
            Reset Interrupt Counter to reload the initialisation
        ELSE
            Update Scrollwheel
            Update Joypad
        END IF

        //Change of Interrupt mode
        IF Scrollwheel is touched AND AD7147 is in Threshold Interrupt Mode THEN
            Change AD7147 from Threshold Interrupt Mode to End of Conversion
            Interrupt mode.
            Reload Interrupt Counter for Changing interrupt mode
        ELSE IF Scrollwheel is not touched AND AD7147 is in End of Conversion Interrupt
                Mode THEN
            Decrement Interrupt Counter for Changing interrupt mode
            IF Counter for Changing interrupt mode = 0 THEN
                Change AD7147 from End of Conversion Interrupt mode to Threshold
                Interrupt Mode.
            END IF
        END IF
    END IF
END
```

Once the ISR is called the first thing that is done is to read the status registers from the AD7147.
On the second interrupt after power up, initialisation of some variables needs to be done for the scroll wheel algorithm to operate correctly. On the 3rd interrupt after power up, the scroll wheel algorithm can be processed.

First the code checks if a recalibration of the sensors is necessary in case an error has occurred as described above. 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()`" function. When the user lifts off the sensor this software routine should recalibrate the scroll wheel to the untouched value allowing the sensor to function correctly afterwards.
If an error is not detected then we process the scroll wheel algorithm and the change of interrupt mode as described previously if necessary. The scroll wheel algorithm is executed calling the function `UpdateScrollwheel()`".

The following pseudo code describes the functionality behind the "`UpdateScrollwheel ()`" function. This function is defined in the file "`AD7147 - Scrollwheel.c`".

```
BEGIN
    Detect touch errors on the scrollwheel

    Read 12 CDC results
    Read 12 Ambient values

    FOREACH Stage
        //Calculate a value representing the distance between the current value and the
        //ambient value
        IF Current ADC Value >  Ambient Value THEN
            Sensor value = Current ADC Value - Ambient Value
        ELSE
            Sensor value = 0
        END IF
    NEXT

    //Determine activation
    IF Any bit of the 8 stages is set in Stage High Limit Threshold register THEN
        Scrollwheel activated = TRUE
        Increment interrupt counter for the TAP
        IF interrupt counter > (T_MIN+T_MAX) THEN
            Clear No Touch Interrupt Counter
        END IF
    ELSE
        Scrollwheel activated = FALSE
        Reset activation variables
        Reset list box variables
        //Work out the tap
        Increment No Touch Interrupt Counter
        IF T_MIN > interrupt counter for the TAP > T_MAX AND
            No Touch Interrupt Counter > interrupt counter for the TAP THEN
            Set tapping bit for a few interrupts.
        END IF
    END IF

    IF Scrollwheel activated = TRUE THEN
```

```
        Find Sensor with highest response
        //Identify sensor besides the sensor with the highest response
        SELECT Sensor with highest response
            CASE 0
                Second sensor before sensor with highest response = 6
                Sensor before sensor with highest response = 7
                Sensor after sensor with highest response = 1
                Second sensor after sensor with highest response = 2
            CASE 1
                Second sensor before sensor with highest response = 7
                Sensor before sensor with highest response = 0
                Sensor after sensor with highest response = 2
                Second sensor after sensor with highest response = 3
            CASE 6
                Second sensor before sensor with highest response = 4
                Sensor before sensor with highest response = 5
                Sensor after sensor with highest response = 7
                Second sensor after sensor with highest response = 0
            CASE 7
                Second sensor before sensor with highest response = 5
                Sensor before sensor with highest response = 6
                Sensor after sensor with highest response = 0
                Second sensor after sensor with highest response = 1
            DEFAULT
                Second sensor before sensor with highest response =
                sensor with highest response - 2

                Sensor before sensor with highest response =
                            sensor with highest response - 1

                Sensor after sensor with highest response =
                            sensor with highest response + 1

                Second sensor after sensor with highest response =
                        sensor with highest response +2
        END SELECT
    END IF

    //Calculate absolute position
    IF No touch errors were detected THEN
        Apply weights to sensor values
        Modify weights when scrolling around the 12.00 point
        Calculate Mean value
        IF transition from segment 7 to segment 0 around the 12.00 point is
          detected THEN
            Calculate position offset
            Calculate position ratio so that the number of positions ranges from 0 to 127
        END IF

        Position on scrollwheel = (Mean Value – Position offset) / Position ratio
        Update Position FIFO with new scrollwheel position

        IF transition from segment 7 to segment 0 around the 12.00 point is detected
             clockwise THEN
            Keep track of scrollwheel position at 12.00
        ELSE IF transition from segment 7 to segment 0 around the 12.00 point is detected
                 anticlockwise THEN
            Keep track of scrollwheel position at 12.00
        END IF
```

```
        //Calculate relative position
        IF Fast rate of change detected = FALSE AND Lift off detected = FALSE
            AND Multiple tap detected = FALSE THEN
            IF transition from segment 7 to segment 0 around the 12.00 point is
                detected clockwise THEN
                Relative displacement = 12.00 position + New scrollwheel Position
            ELSE IF transition from segment 7 to segment 0 around the 12.00
                    point is detected anticlockwise THEN
                Relative displacement = 12.00 position – (127 – New scrollwheel
                Position)
            ELSE
                Relative displacement = New scrollwheel Position –  Position from
                                        previous update in the list box
            END IF

            IF Absolute (Relative displacement) > Item resolution THEN
                IF Relative displacement >= 0 THEN
                    Generate a GO DOWN command
                ELSE IF Relative displacement < 0 THEN
                    Generate a GO UP command
                END IF
            END IF
        END IF

        //Clear tap if we're scrolling
        IF Movement has been detected THEN
            Cancel Tap
        END IF

        //Format position data
        Scrollwheel status = Touch error
        IF scrollwheel activated = TRUE
                IF Lift off detected = FALSE AND Touch Error = 0 THEN
                Scrollwheel status = Scrollwheel status OR new position OR
                Activation bit
            END IF
        ELSE
            Scrollwheel status = Scrollwheel status with activation bit cleared
            IF Tap detected =TRUE THEN
                Set tap bit
            ELSE
                Clear tap bit
            END IF
        END IF
    END IF //End IF No touch errors were detected THEN
END
```

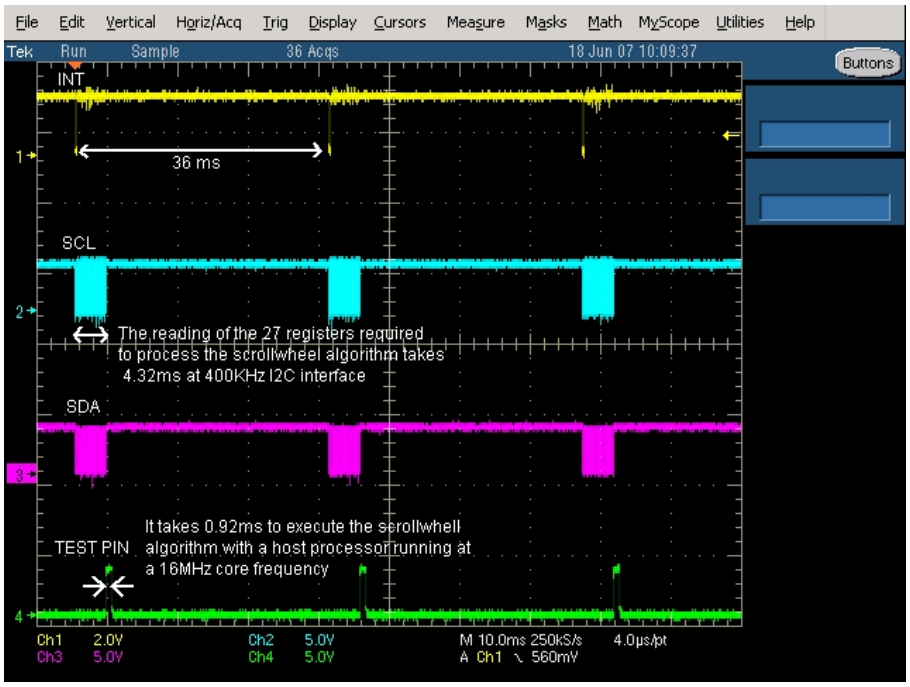


Fig.6 Communication timing between the AD7147 and host processor

As soon as the scrollwheel 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 sequence of reads from the AD7147. All registers are read back within 4.32ms using a 400KHz I2C interface, the algorithm to process position data takes a further 0.92ms with a host processor running at 16MHz core clock frequency. These are the registers being read.

- Low Limit, High Limit and Stage Complete status registers.
- ADC values for the first 8 stages
- Ambient values for the first 8 stages

Next, we look for touch errors. A touch error is registered when 2 fingers are detected or when a wide surface area of the sensor is in contact with user's hand. To detect 2 fingers, we look at the High Limit Interrupt Status register (at address 0x09) and search for the interrupt bits that are not contiguously set. An error registered due to a contact of a wide sensor area occurs when too many bits are set in the High Limit Interrupt Status register.

After checking for touch errors, we process the activation of the scrollwheel. The scrollwheel is activated if any of the 8 lower bits in High Limit Interrupt Status register is set.

If the scrollwheel is activated, we calculate the sensor response for all the segments of the scroll wheel. The sensor value is only calculated when the CDC result is above the ambient value. If not, the sensor value is 0.

The sensor value is defined as follow:

$$Sensor\_value = Current\_value - Ambient\_value$$

After calculating the response of the sensor for each stage, we keep track of the number of interrupts registered when touching and when the user does not touch the scrollwheel.
The counters used in these 2 operations are useful to determine a valid Tap on the scrollwheel.
When lifting off, we check if the number of interrupt when the user was touching is within certain bounds. If it is and if the number of interrupt registered when lifting off is greater that the number of interrupts when touching, then we register a valid tap.

$$Tap = \left(T\_MIN < TouchDownCounter < T\_MAX\right) AND \left(NoTouchCounter > TouchDownCounter\right)$$

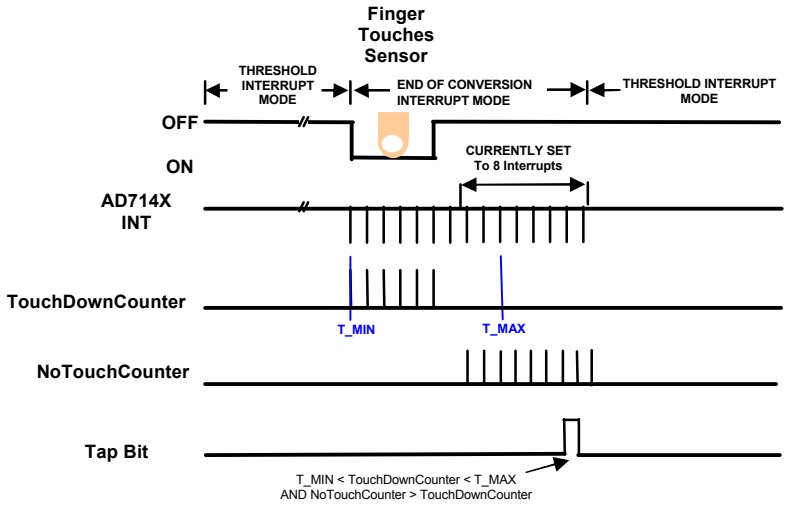


Fig.7 Tap timing

When the scroll wheel is activated, we compute the absolute position based on the sensor values. To calculate the position, we first determine the sensor that has the greatest response among the 8 sensors that constitutes the scrollwheel. Then we determined the 2 sensors on either sides of the sensor with the highest response and we apply weights to these sensors. The result of this computation gives us the mean value which defined by the following formula:

$$Mean = \frac{\sum_{i=Second\_sensor\_before\_the\_highest}^{i=Second\_sensor\_after\_the\_highest} Scaled\_sensor\_response \times Weight \times (i+3)}{\sum_{i=Second\_sensor\_before\_the\_highest}^{i=Second\_sensor\_after\_the\_highest} Scaled\_sensor\_response}$$

IMPROVING LINEARITY AT THE 127 - 0 CROSS OVER POINT

The formula used above is fine, but does not take into account that the sensor is round. Indeed, segment 7 is next to segment 0, and when applying the weights to each segment, the mean value will decrease before reaching the 127 to 0 cross over point.

To overcome this problem, we alter the set of data around the 127 to 0 cross over point. If the sensor segment 7 is the segment with the highest response, then we consider sensor segment 0 to be the one after and therefore apply to segment 0 a higher weight factor than segment 7. The same applies if sensor segment 0 is the segment with the highest response we apply to segment 7 a lower weight factor than segment 0.

THE MEAN OFFSET

After applying the above formula to the correct sensor values, the resulting mean value will range from 75 to 310 (depending on the pressure). This is showed on the following plot,

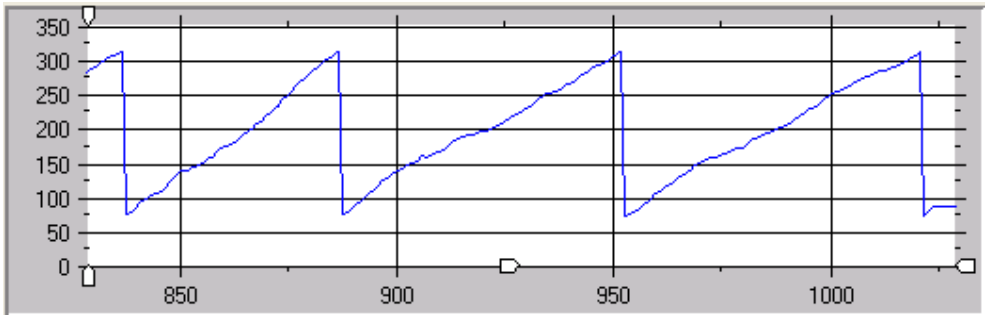


FIG.8 Response of the Mean value

In order to calculate this offset accurately, we must catch the 127 to 0 transition. To do so, we record the mean value at the moment where the sensor with the higher response goes from either Sensor 7 to Sensor 0 or from either Sensor 0 to Sensor 7. At that point, we can guarantee that the recoded mean value will be the smallest one. This smallest mean value becomes the offset that we will subtract to the mean value. Figure 9 shows the response of the mean value with the position offset subtracted.

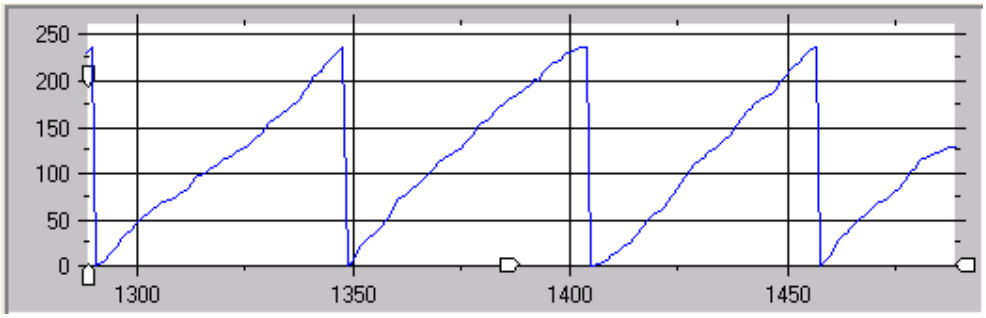


FIG.9 Response of the Mean value with the position offset subtracted

The mean value now ranges from 0 to 230 (depending on the pressure).

THE MEAN RATIO

Similarly to the offset, the position ratio is calculated when passing the 127 to 0 crossover point. If the user scrolls clockwise, then at the 127 to 0 crossover point, the position ratio is defined by the mean value without offset from the previous interrupt divided by 127. If the user scrolls anticlockwise then the position ratio is defined by the mean value without offset from the current interrupt divided by 127.

Once calculated and applied to the Mean value without offset, we can see that the absolute position ranges from 0 to 127 as showed in figure 10.

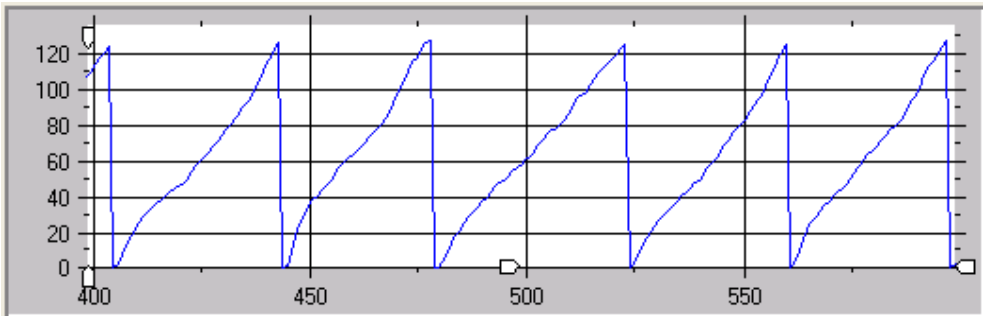


FIG.10 Response of the scrollwheel ranging from 0 to 127

CALCULATION OF THE RELATIVE POSITION

The calculation of the relative position is done by recording the absolute position on the first touch and then subtracting it from the latest absolute position. At the 127 to 0 crossover point, the relative position is recorded and if this latest is negative, then we keep decrementing the latest relative position. If positive, the relative position is incremented.

When the relative position is greater than the item resolution in the list box (defined as `DISPLAY_ITEMS_CONSTANT` in the code), the position on the first

touched is re-initialised to the current position and at the same time an update in the list box is done. If the relative position is negative then the firmware will generate a GO UP command. If the relative position is positive, the firmware will generate a GO DOWN command.

Joypad functionality with the scroll wheel sensor

With the 8 segment scroll wheel senor, it is possible to add extra functionality to the firmware to compute UP, DOWN, LEFT and RIGHT positions.
For this functionality, we just need to decode the High Limit Interrupt Status register when the sensor is touched.

THE DIRECTION BUTTON

The decoding of the direction is done with a look up table as shown below:

```
//Only look at the scrollwheel bits
switch (AD7147Registers[STAGE_HIGH_LIMIT_INT] & 0xFF)
{
   /All possible combinations for top button
   case 0x0001:
     DirectionButtonSet=1;
   break;
   case 0x0080:
     DirectionButtonSet=1;
   break;
   case 0x0081:
     DirectionButtonSet=1;
   break;
   case 0x0083:
     DirectionButtonSet=1;
   break;
   case 0x00C1:
     DirectionButtonSet=1;
   break;
   case 0x00C3:
     DirectionButtonSet=1;
   break;
   //All possible combinations for right button
   case 0x0002:
     DirectionButtonSet=2;
   break;
   case 0x0004:
     DirectionButtonSet=2;
   break;
   case 0x0006:
     DirectionButtonSet=2;
   break;
   case 0x0007:
     DirectionButtonSet=2;
   break;
   case 0x000D:
     DirectionButtonSet=2;
   break;
   case 0x000E:
     DirectionButtonSet=2;
```

```
   break;
   //All possible combinations for bottom button
   case 0x0008:
     DirectionButtonSet=3;
   break;
   case 0x0010:
     DirectionButtonSet=3;
   break;
   case 0x0018:
     DirectionButtonSet=3;
   break;
   case 0x001C:
     DirectionButtonSet=3;
   break;
   case 0x0038:
     DirectionButtonSet=3;
   break;
   case 0x003C:
     DirectionButtonSet=3;
   break;
   //All possible combinations for left button
   case 0x0020:
     DirectionButtonSet=4;
   break;
   case 0x0040:
     DirectionButtonSet=4;
   break;
   case 0x0060:
     DirectionButtonSet=4;
   break;
   case 0x0070:
     DirectionButtonSet=4;
   break;
   case 0x00E0:
     DirectionButtonSet=4;
   break;
   case 0x00F0:
     DirectionButtonSet=4;
   break;
   default:
     DirectionButtonSet=0;
   break;
}//End switch (AD7147Registers[STAGE_HIGH_LIMIT_INT])
```

All the possible combinations for a direction button are decoded.
On activation, if the user has not moved then we register a direction button after 16 interrupts (16 * 0.036 = 0.57 second). For this application, we decided to give priority to the scroll wheel on activation and not to the direction buttons. If a direction button is set and the user starts scrolling, then the direction buttons are cleared and the scroll wheel updates take over.

When a direction button is registered, we also create a command that can be used to control a menu list. This command consists of a bit that is permanently toggled until the user moves the finger or lift off the sensor.

THE CENTRE BUTTON

Unlike the direction buttons, we do not wait for 0.57 second to activate the centre button. This last is registered immediately on activation if no movement was registered.

Configuration of the firmware

There is very little to configure in the scrollwheel firmware. The only setting that may need to be altered is the "DISPLAY_ITEM_CONSTANT" definition. This definition is in the file "AD7147 Scrollwheel Config.h" and is set by default to 16. This means that scrolling one revolution, will allow the user to select 1 item among 16 in the list box.

The tables below show the data format returned by the "`ScrollwheelStatus`" and the "`JoypadStatus`" variables.

Bits	Description
0	Absolute positions ranging from 0 to 127
1	
2	
3	
4	
5	
6	
7	
8	0
9	0
10	Touch errors
11	
12	Go DOWN command
13	Go UP command
14	Tap
15	Activation

Table 2 – Scrollwheel Status

Bits	Description
0	DOWN button
1	UP button
2	LEFT button
3	RIGHT button
4	CENTRE button
5	Go DOWN command
6	Go UP command
7	Go LEFT command
8	Go RIGHT command
9	Unused
10	
11	
12	
13	
14	

Table 3 – Joypad Status

Code and data memory requirements

The scrollwheel firmware requires 8.42Kb of program memory and 455 bytes of RAM. These are the requirements to implement the AD7147 register configuration, recalibration and scrollwheel algorithm processing 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.