

met problem in touch panel driver

Source:

<http://www.tech-archive.net/Archive/WindowsCE/microsoft.public.windowsce.platbuilder/2004-02/0804.html>

From: Lilo (*anonymous_at_discussions.microsoft.com*)

Date: 02/22/04

Date: Sun, 22 Feb 2004 06:01:06 -0800

Hi All:

I met a bit problem when I tried to port a touch panel driver in PB41, I migrated the touch driver which provided by MS, sample touch panel driver. my touch controller is ADS7846, and control by XScale SSP port. GPIO3 is the ads7846's IRQ pin. I modified the driver as below tchpdd.cpp show. under debug mode, as the OS boot, when OS load touch.dll and implement some code later, the system died, no any messages out put again. the messages is show if red font.

From the message and my driver, what cause the problem? when I removed the touch panel driver and build the system again, the system run well.

Anybody can give me some advice? Thank you very much!

PB output message:

4294795314 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: >>> Loading module touch.dll at address 0x03360000-0x0336A000 (RW data at 0x01F66000-0x01F66674)

Loaded symbols for

H:\WINCE410\PUBLIC\PISCE\RELDIR\INTEL_DBPXA250_DEV_PLATFORM_INTEL_SUPPLIED_ARMV4IR

4294795461 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: calibrating 0 point set

4294795463 PID:23bd55c2 TID:23c8f0ce *****OEMInterruptDone:

SYSINTR_TOUCH_CHANGEDpanel.

4294796553 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: OEMInterruptEnable Handle.

4294796556 PID:23bd55c2 TID:23c8f0ce *****OEMInterruptEnable: SYSINTR_TOUCHpanel.

4294796559 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: OEMInterruptEnable Handle.

4294796561 PID:23bd55c2 TID:23c8f0ce *****OEMInterruptEnable:

SYSINTR_TOUCH_CHANGEDpanel.

4294796569 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: TOUCH:TouchPanelpGetPriority - RegOpenKeyEx(\Drivers\BuiltIn\Touch) failed 2, using default thread priorities

4294796573 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: calibrating 5 point set

4294796579 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Maximum Allowed Error 5:

4294796581 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Calibration Results:

4294796583 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Screen => Mapped

4294796586 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: (120, 160) => (120, 160)

4294796588 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: (48, 64) => (48, 64)

4294796590 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: (48, 256) => (48, 256)

4294796592 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: (192, 256) => (192, 256)

4294796594 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: (192, 64) => (192, 64)

4294796596 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Maximum error (square of Euclidean distance in screen units) = 0

4294796604 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Oom Low Pages: 0020 Bytes: 00020000

microsoft.public.windowsce.platbuilder: met problem in touch panel driver

4294796606 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Oom Critical Pages: 0014 Bytes: 00014000
4294796609 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Oom Low Block Pages: 0004 Bytes: 00004000
4294796611 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Oom Critical Block Pages: 0002 Bytes: 00002000
4294796613 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Oom App Wait 8000 (dec ms)
4294796615 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: Oom AutoOom 0
4294796624 PID:23bd55c2 TID:23c8f0ce 0x83bd1bf0: UpdateIdleTimeoutFromRegistry
4294797768 PID:c3c8adce TID:e3c336d6 0x83c52400: *TCBTimeout: ClearIdleTimerReset
4294797775 PID:c3c8adce TID:e3c336d6 0x83c52400: TCBTimeout: setting fTCBTimerOn to 0
4294797777 PID:c3c8adce TID:e3c336d6 0x83c52400: TCBTimeout: NOT Restarting TCBTimer

tchpdd.cpp

```
//-----  
// defines  
//-----  
#define REV_2A 0x2a0  
#define REV_1B 0x2a  
  
// The Differences in the definition of the two variables in the MDD for the two  
// OS's has been incorporated here.  
#ifndef USE_MERLIN  
const DWORD gIntrTouch = SYSINTR_TOUCH;  
const DWORD gIntrTouchChanged = SYSINTR_TOUCH_CHANGED;  
#endif  
  
#ifndef USE_TALISKER  
DWORD gIntrTouch = SYSINTR_TOUCH;  
DWORD gIntrTouchChanged = SYSINTR_TOUCH_CHANGED;  
#endif  
  
//-----  
// Global Variables.  
//-----  
  
static TOUCHPANEL_POINT_SAMPLES rgPointSamples;  
static TOUCH_PANEL_SAMPLE_FLAGS PrevStateFlags;  
static void TouchPanelPowerOff(BOOL);  
  
static unsigned int ResetCaps; //HACK: If 0x2a0 it's rev 2a, if its 0x2a then its rev 1b  
  
// The MDD requires a minimum of MIN_CAL_COUNT consecutive samples before  
// it will return a calibration coordinate to GWE. This value is defined  
// in the PDD so that each OEM can control the behaviour of the touch  
// panel and still use the Microsoft supplied MDD. Note that the extern "C"  
// is required so that the variable name doesn't get decorated, and  
// since we have an initializer the 'extern' is actually ignored and  
// space is allocated.
```

```
extern "C" const int MIN_CAL_COUNT = 25;

INT CurrentSampleRateSetting = 0; // Low sample rate setting

volatile INTC_REGS *v_pICReg = NULL;
volatile OST_REGS *v_pOSTReg = NULL;
volatile GPIO_REGS *v_pGPIOReg = NULL;
volatile SSP_REGS *v_pSSPReg = NULL;
PDRIVER_GLOBALS v_pDriverGlobals = NULL;

static unsigned nextExpectedInterrupt;

// Global flag to indicate whether we are in power handler routine, so
// we know to avoid system calls.
static BOOL bInPowerHandler = FALSE;

// AC 97 functions and other functions used by the touch driver.

#ifdef __cplusplus
extern "C" {
#endif

void msWait(int);
void usWait(int);
PVOID VirtualAllocCopy(unsigned size,char *str,PVOID pVirtualAddress);
//short int ReadAC97(BYTE , unsigned short int * , BYTE );
//short int WriteAC97(BYTE , unsigned short int, BYTE );
//short int RawAC97Write(BYTE , unsigned short int,BYTE);
//short int InitAcLink(volatile GPIO_REGS *, BOOL, BYTE );
//short int DeAllocAcLink(BYTE DevId);
//short int DeInitAcLink(volatile GPIO_REGS *, BOOL, BYTE);
//short int AC97GpioUnConfigure(volatile GPIO_REGS *, BOOL, BYTE );
//short int AC97GpioConfigure(volatile GPIO_REGS *, BOOL, BYTE );
BOOL StartClock(unsigned int,BOOL);
BOOL StopClock(unsigned int,BOOL);
extern void DriverSleep(DWORD, BOOL);

#ifdef __cplusplus
}
#endif

//-----

static void PddpTouchPanelDeallocateVm()
{
    if (v_pDriverGlobals)
    {
        VirtualFree(v_pDriverGlobals,DRIVER_GLOBALS_PHYSICAL_MEMORY_SIZE,MEM_RELEASE);
        v_pDriverGlobals = NULL;
    }
}
```

```

if (v_pICReg)
{
    VirtualFree((void *)v_pICReg,0x400,MEM_RELEASE);
    v_pICReg = NULL;
}
if (v_pOSTReg)
{
    VirtualFree((void *)v_pOSTReg,0x400,MEM_RELEASE);
    v_pOSTReg = NULL;
}
if (v_pGPIOREg)
{
    VirtualFree((void *)v_pGPIOREg,0x400,MEM_RELEASE);
    v_pGPIOREg = NULL;
}
if (v_pSSPReg)
{
    VirtualFree((void *)v_pSSPReg,0x400,MEM_RELEASE);
    v_pSSPReg = NULL;
}
}

//-----

INT evaluateSample(USHORT val0,USHORT val1,USHORT val2,int maxError,INT *sample)
{
    LONG diff0,diff1,diff2;
    INT retval=TouchSampleIgnore;

    if ((val0 < MAX_ADC_VAL) && (val1 < MAX_ADC_VAL) && (val2 < MAX_ADC_VAL))
    {
        // Calculate the absolute value of the differences of the sample
        diff0 = val0 - val1;
        diff1 = val1 - val2;
        diff2 = val2 - val0;
        diff0 = diff0 > 0 ? diff0 : -diff0;
        diff1 = diff1 > 0 ? diff1 : -diff1;
        diff2 = diff2 > 0 ? diff2 : -diff2;

        if (diff0 < diff1)
            *sample=(ULONG)(val0 + ((diff2 < diff0) ? val2 : val1));
        else
            *sample=(ULONG)(val2 + ((diff2 < diff1) ? val0 : val1));

        *sample>>=1;

        if ((diff0 < maxError) && (diff1 < maxError) && (diff2 < maxError))
            retval = TouchSampleValidFlag ;
    }
}
#ifdef DBGPOINTS1
if (retval == TouchSampleIgnore)

```

```

{
    ERRORMSG(1,(TEXT("Bad evaluateSample: 0x%x 0x%x 0x%x\r\n"),val0,val1,val2));
    if (diff0 > maxError)
        ERRORMSG(1,(TEXT("Diff0 too large 0x%x\r\n"),diff0));
    if (diff1 > maxError)
        ERRORMSG(1,(TEXT("Diff1 too large 0x%x\r\n"),diff1));
    if (diff2 > maxError)
        ERRORMSG(1,(TEXT("Diff2 too large 0x%x\r\n"),diff2));
}
#endif

```

```

return(retval);
}

```

```

//-----

void enableTouchTimerInterrupt(unsigned timeout)
{
    v_pOSTReg->osmr1 = v_pOSTReg->oscr + timeout;
    //allows a match between OSMR[1] and the OS Timer to assert interrupt bit M1 in the OSSR
    *(unsigned int *)&v_pOSTReg->oier |= OIER_E1;
    //enable the interrupt for OSMR1
    *(unsigned int *)&v_pICReg->icmr |= ICMR_OSMR1;
    // OS timer match register 1 has matched the OS timer counter
    *(unsigned int *)&v_pOSTReg->ossr = OSSR_M1;
}

```

```

//-----

BOOL setUcb1x00TouchInterruptMode()
{
    unsigned int tcr=0;

    v_pGPIOReg->GFER_x |= GPIO_3;
    //if (ResetCaps==REV_2A)
    // tcr = TSMY_GND | TSPY_GND | TSMX_POW | TSPX_POW; // set interrupt mode
    //else
    // tcr = TSMY_GND | TSPY_GND; // set interrupt mode

    //return(WriteAC97(UCB_TCH_CR, *(unsigned short *)&tcr, DEV_TOUCH));
    return(TRUE);
}

```

```

//-----

BOOL PddpEnablePenInterrupt(BOOL irqMode)
{
    // unsigned int ier;

    if(irqMode==PEN_DOWN) // if next expected interrupt is pen down

```

```
{ //ier = TSPX_INT;
//WriteAC97(UCB_FE_IE,ier,DEV_TOUCH); // enable falling edge interrupts.
//setUcb1x00TouchInterruptMode();
v_pGPIOReg->GFER_x |= GPIO_3;
}
else
{ // Enable timer interrupt for drawing
enableTouchTimerInterrupt(TOUCH_TIMER_INCREMENT);
}

return (TRUE);
}
BOOL sampleADC(USHORT *sample,unsigned axis)
{
unsigned int adccr=0;
// unsigned int adcDataReg;

BOOL noError=FALSE;

//adccr = (axis == UCB_ADC_X ? ADC_INPUT_TSPY : ADC_INPUT_TSPX) | ADC_ENA ;
v_pSSPReg->ssdr = axis;
*sample =(unsigned short) v_pSSPReg->ssdr;
//if (WriteAC97(UCB_ADC_CR,*(&adccr, DEV_TOUCH))
//{
// adccr |= ADC_START;
// if(WriteAC97(UCB_ADC_CR,*(&adccr, DEV_TOUCH))
// {
// do { // wait for sample completion
// ReadAC97(UCB_ADC_DATA,(unsigned short *) &adcDataReg,DEV_TOUCH);
// } while(!TEST(adcDataReg,ADC_DATA_VAL));
//
// *sample=TEST(adcDataReg,ADC_DATA) ;
// // disable adc
// CLEAR(adccr,ADC_ENA);
// WriteAC97(UCB_ADC_CR,*(&adccr,DEV_TOUCH);
noError=TRUE;
// }
//}
return(noError);
}
```

```
USHORT getTouchCoordinate(unsigned axis)
{
unsigned int tcr=0;
BOOL noError=FALSE;
USHORT sample=MAX_ADC_VAL;
```

```
//Bias the ADC, set position mode and choose the correct axis.
```

microsoft.public.windowsce.platbuilder: met problem in touch panel driver

```
//tcr = TSC_BIAS_ENA | TSC_MODE_POSITION | ((axis==UCB_ADC_X) ? (TSPX_POW |
TSMX_GND) : (TSPY_POW | TSMY_GND));
//if(WriteAC97(UCB_TCH_CR,*(unsigned short *)&tcr, DEV_TOUCH))
//{
noError=sampleADC(&sample,axis); // first 2 is basically to prime the pump.
noError=sampleADC(&sample,axis); // gives better variation on Sandgate.
noError=sampleADC(&sample,axis);
//}

return(sample);
}

//
// DDSI Implementation
//

// @DOC EX_TOUCH_DDSI EXTERNAL DRIVERS TOUCH_DRIVER
extern "C" BOOL TouchDriverCalibrationPointGet(TPDC_CALIBRATION_POINT *pTCP)
{
INT32 cDisplayWidth = pTCP->cDisplayWidth;
INT32 cDisplayHeight = pTCP->cDisplayHeight;

int CalibrationRadiusX = cDisplayWidth/10;
int CalibrationRadiusY = cDisplayHeight/10;

switch (pTCP->PointNumber)
{
case 0: // Middle
pTCP->CalibrationX = cDisplayWidth/2;
pTCP->CalibrationY = cDisplayHeight/2;
break;

case 1: // Upper Left
pTCP->CalibrationX = CalibrationRadiusX*2;
pTCP->CalibrationY = CalibrationRadiusY*2;
break;

case 2: // Lower Left
pTCP->CalibrationX = CalibrationRadiusX*2;
pTCP->CalibrationY = cDisplayHeight - CalibrationRadiusY*2;
break;

case 3: // Lower Right
pTCP->CalibrationX = cDisplayWidth - CalibrationRadiusX*2;
pTCP->CalibrationY = cDisplayHeight - CalibrationRadiusY*2;
break;

case 4: // Upper Right
pTCP->CalibrationX = cDisplayWidth - CalibrationRadiusX*2;
pTCP->CalibrationY = CalibrationRadiusY*2;
}
```

met problem in touch panel driver

```
break;

default:
pTCP->CalibrationX = cDisplayWidth/2;
pTCP->CalibrationY = cDisplayHeight/2;
SetLastError(ERROR_INVALID_PARAMETER);
return FALSE;
}

return TRUE;
}
//-----

extern "C" BOOL DdsiTouchPanelGetDeviceCaps(INT iIndex,LPVOID lpOutput)
{

if (lpOutput == NULL)
{
ERRORMSG(1,(__TEXT("TouchPanelGetDeviceCaps: invalid parameter.\r\n")));
SetLastError(ERROR_INVALID_PARAMETER);
return FALSE;
}

switch (iIndex)
{
case TPDC_SAMPLE_RATE_ID:
{
TPDC_SAMPLE_RATE *pTSR = (TPDC_SAMPLE_RATE*)lpOutput;

pTSR->SamplesPerSecondLow = TOUCHPANEL_SAMPLE_RATE_LOW;
pTSR->SamplesPerSecondHigh = TOUCHPANEL_SAMPLE_RATE_HIGH;
pTSR->CurrentSampleRateSetting = CurrentSampleRateSetting;
}
break;

case TPDC_CALIBRATION_POINT_COUNT_ID:
{
TPDC_CALIBRATION_POINT_COUNT *pTCPC =
(TPDC_CALIBRATION_POINT_COUNT*)lpOutput;
pTCPC->flags = 0;
pTCPC->cCalibrationPoints = 5;
}
break;

case TPDC_CALIBRATION_POINT_ID:
return(TouchDriverCalibrationPointGet((TPDC_CALIBRATION_POINT*)lpOutput));

default:
ERRORMSG(1,(__TEXT("TouchPanelGetDeviceCaps: invalid parameter.\r\n")));
SetLastError(ERROR_INVALID_PARAMETER);
return FALSE;
}
```

```
}  
return TRUE;  
}
```

```
//-----  
  
BOOL DdsiTouchPanelSetMode(INT iIndex,LPVOID lpInput)  
{  
    BOOL ReturnCode = FALSE;  
  
    //RETAILMSG(1,(TEXT("DdsiTouchPanelSetMode(0x%x)\r\n"),iIndex));
```

```
    switch (iIndex)  
    {  
        case TPSM_SAMPLERATE_LOW_ID:  
        case TPSM_SAMPLERATE_HIGH_ID:  
            SetLastError(ERROR_SUCCESS);  
            ReturnCode = TRUE;  
            break;  
  
        default:  
            SetLastError(ERROR_INVALID_PARAMETER);  
            break;  
    }  
    return (ReturnCode);  
}
```

```
//-----  
// Function: AllocTouchPanelRegs  
//  
// Purpose: Allocates memory for all the registers used in the touch panel driver.  
// Returns: returns TRUE if it could allocate all the memory it needs.  
//  
//-----
```

```
BOOL AllocTouchPanelRegs()  
{  
    if (v_pDriverGlobals == NULL)  
    {  
        v_pDriverGlobals = (PDRIVER_GLOBALS)  
            VirtualAllocCopy(DRIVER_GLOBALS_PHYSICAL_MEMORY_SIZE,(char  
*)TEXT("TouchPanelEnable: DRIVER_GLOBALS"),  
                (PVOID)DRIVER_GLOBALS_PHYSICAL_MEMORY_START);  
        if (v_pDriverGlobals)  
        {  
            if (v_pICReg == NULL)  
            {  
                v_pICReg = (volatile INTC_REGS *)  
                    VirtualAllocCopy(0x400,(char *)TEXT("TouchPanelEnable: INTC_BASE_U_VIRTUAL"),  
                        (PVOID)INTC_BASE_U_VIRTUAL);  
                if (v_pICReg)
```

```

{
if (v_pGPIOReg == NULL)
{
v_pGPIOReg = (volatile GPIO_REGS *)
VirtualAllocCopy(0x400,(char *)TEXT("TouchPanelEnable: GPIO_BASE_U_VIRTUAL"),
(PVOID)GPIO_BASE_U_VIRTUAL);
if (v_pGPIOReg)
{
if (v_pOSTReg == NULL)
{
v_pOSTReg = (volatile OST_REGS *)
VirtualAllocCopy(0x400,(char *)TEXT("TouchPanelEnable: OST_BASE_U_VIRTUAL"),
(PVOID)OST_BASE_U_VIRTUAL);
if (v_pOSTReg)
{
if (v_pSSPReg == NULL)
{
v_pSSPReg = (volatile SSP_REGS *)
VirtualAllocCopy(0x400,(char *)TEXT("TouchPanelEnable: SSP_BASE_U_VIRTUAL"),
(PVOID)SSP_BASE_U_VIRTUAL);
}
}
}
}
}
}
}
}
}
}
}

if (!v_pSSPReg || !v_pOSTReg || !v_pGPIOReg || !v_pICReg || !v_pDriverGlobals )
{
PddpTouchPanelDeallocateVm();
DEBUGMSG(1,(TEXT("DdsiTouchPanelEnable(): Error %u\r\n"),GetLastError()));
return (FALSE);
}

return(TRUE);
}

BOOL DdsiTouchPanelEnable()
{
// DEBUGMSG(1,(TEXT("*****Entering DdsiTouchPanelEnable()\r\n")));

#ifdef PLAT_SANDGATE
// set the Board Control Register values
// set_BCRVal(BCR_AUD_POWER_ON, 0, FALSE);
// msWait(1000);
#endif
AllocTouchPanelRegs();

```

microsoft.public.windowsce.platbuilder: met problem in touch panel driver

```
//if(AllocTouchPanelRegs())
// InitAcLink(v_pGPIOReg,0,DEV_TOUCH);
//else
// return(FALSE);

//HACK: If the capabilities reads 0x2a0 it's rev 2a, if its 0x2a then its rev 1b
//ReadAC97(0x0, (unsigned short *) &ResetCaps, DEV_TOUCH);

// Setup pen down interrupts, but leave ints disabled until InterruptEnable().
DdsiTouchPanelPowerHandler(0);

//RETAILMSG(1,(TEXT("Exiting DdsiTouchPanelEnable()\r\n")));

return(TRUE);
}
```

//-----

```
VOID DdsiTouchPanelDisable()
{
//RETAILMSG(1,(TEXT("DdsiTouchPanelDisable()\r\n")));
PddpTouchPanelDeallocateVm(); // free up any resources
//DeInitAcLink(v_pGPIOReg, 0, DEV_TOUCH); // Clean up
}
```

//-----

```
LONG DdsiTouchPanelAttach()
{
return(1);
}
```

```
TOUCH_PANEL_SAMPLE_FLAGS SampleTouchScreen(INT *x,INT *y)
{
unsigned int i;
TOUCH_PANEL_SAMPLE_FLAGS TmpStateFlags;
INT TmpX = 0;
INT TmpY = 0;

for(i=0;i<NUMBER_SAMPLES_PER_POINT;i++)
{
rgPointSamples[i].XSample=getTouchCoordinate(UCB_ADC_X);
rgPointSamples[i].YSample=getTouchCoordinate(UCB_ADC_Y);
}
```

```
TmpStateFlags = TouchSampleDownFlag;
if (evaluateSample(rgPointSamples[0].XSample,rgPointSamples[1].XSample,rgPointSamples[2].XSample,
DELTA_X_COORD_VARIANCE,&TmpX) == TouchSampleIgnore)
{
TmpStateFlags |= TouchSampleIgnore;
DEBUGMSG(1,(TEXT("Invalid X sample\r\n")));
}
```

met problem in touch panel driver

```

}
else
{
    TmpStateFlags |=
evaluateSample(rgPointSamples[0].YSample,rgPointSamples[1].YSample,rgPointSamples[2].YSample,
    DELTA_Y_COORD_VARIANCE,&TmpY);
}

// DEBUGMSG(1,(TEXT("Filtered - SampleFlags: 0x%x X: 0x%x Y:
0x%x\r\n"),TmpStateFlags,TmpX,TmpY));

*x=TmpX;
*y=TmpY;

return(TmpStateFlags);
}

//-----

VOID DdsiTouchPanelGetPoint(TOUCH_PANEL_SAMPLE_FLAGS *pTipStateFlags,INT *pUncalX,INT
*pUncalY)
{
    static TOUCH_PANEL_SAMPLE_FLAGS PrevStateFlags = TouchSampleIgnore;
    unsigned InterruptType=SYSINTR_TOUCH;
    // unsigned int ier;

    *pTipStateFlags = TouchSampleIgnore;

    if(v_pDriverGlobals->tch.touchIrq)
    { v_pDriverGlobals->tch.touchIrq=0;
      v_pDriverGlobals->tch.timerIrq=0;
      InterruptType=SYSINTR_TOUCH;
        // handle pen down irq
        *pTipStateFlags=SampleTouchScreen(pUncalX,pUncalY);
        nextExpectedInterrupt=PEN_UP_OR_TIMER;
        //ier = TSPX_INT;
        //WriteAC97(UCB_INT_CS,ier,DEV_TOUCH); // clear the falling edge intr that just came in.
        //WriteAC97(UCB_FE_IE,0,DEV_TOUCH); // disable interrupts.
        v_pGPIOReg->GFER_x &= ~GPIO_3;
    }
    else if(v_pDriverGlobals->tch.timerIrq) // handle timer irq
    { v_pDriverGlobals->tch.timerIrq=0;
      v_pDriverGlobals->tch.touchIrq=0;
      InterruptType=SYSINTR_TOUCH_CHANGED;
        *pTipStateFlags=SampleTouchScreen(pUncalX,pUncalY);
        nextExpectedInterrupt=PEN_UP_OR_TIMER;
    }

    if (!penIsDown() && (nextExpectedInterrupt==PEN_UP_OR_TIMER))
    {
        *pTipStateFlags = TouchSampleValidFlag; // send pen up to mdd
    }
}

```

```

nextExpectedInterrupt = PEN_DOWN;
}

PddpEnablePenInterrupt(nextExpectedInterrupt);
InterruptDone(InterruptType);
RETAILMSG(1,(TEXT("x= %x y= %x\r\n"),*pUncalX,*pUncalY));
}
//-----
void DdsiTouchPanelPowerHandler(BOOL bOff)
{
// unsigned short int ier;
unsigned short int tcr=0;

//RETAILMSG(1, (TEXT("DdsiTouchPanelPowerHandler(0x%x)\r\n"),bOff));

// Set flag so we know to avoid system calls
bInPowerHandler = TRUE;

if (bOff)
{
    TouchPanelPowerOff(bOff);
}
else
{
// Make sure appropriate GPIO is setup for the CODEC.
//AC97GpioConfigure(v_pGPIOReg,bOff,DEV_TOUCH);
//set SSP port
v_pSSPReg->sscr0 = 0x0000;//disable SSE
    v_pSSPReg->sscr0 |= 0x082a;
        //0x000a // DSS :12 bit data size
        //0x0020 //FRF : NATIONAL MICROWIRE
        //0x0000 //ECS :on chip clock
        //0x0800 //SCR :17,200k
v_pSSPReg->sscr1 |= 0x0000;
        //0x0000 //RIE :0
        //0x0000 //TIE :0
        //0x0000 //LBM :0
        //0x0000 //SPO :0
        //0x0000 //SPH :0
v_pSSPReg->sscr0 |= 0x0080; //enable SSE
msWait(1000);

// reset the global variables and the state of the pen.
if(v_pDriverGlobals!=NULL)
{
    v_pDriverGlobals->tch.touchIrq=0;
    v_pDriverGlobals->tch.timerIrq=0;
}
nextExpectedInterrupt=PEN_DOWN;

    DriverSleep(1, bInPowerHandler); // need to wait when you resume.

```

```
//if (ResetCaps==REV_2A)
// tcr = TSMY_GND | TSPY_GND | TSMX_POW | TSPX_POW; // set interrupt mode
//else
// tcr = TSMY_GND | TSPY_GND; // set interrupt mode

//RawAC97Write(UCB_TCH_CR, tcr,DEV_TOUCH );
v_pGPIOReg->GFER_x |= GPIO_3;
// Give some time for write to complete.
DriverSleep(1, bInPowerHandler);

//ier = TSPX_INT;
//RawAC97Write(UCB_FE_IE,ier,DEV_TOUCH);

// RETAILMSG(1,(TEXT("*****Exit DdsiTouchPanelPowerHandler Done\r\n")));
}
bInPowerHandler = FALSE;
}

//-----
// Function: TouchPanelPowerOff
//
// Purpose: This will poweroff the Touch Panel, basically unconfigure the GPIO lines.
// Returns: void.
//
//-----

static void TouchPanelPowerOff(BOOL bOff)
{
//AC97GpioUnConfigure(v_pGPIOReg,bOff,DEV_TOUCH);
}

//-----
// Function: TouchPanelInitializeCursor
//
// Purpose: currently does nothing.
// Returns: TRUE
//
//-----

BOOL TouchPanelInitializeCursor()
{
return TRUE;
}
```