Feb 8.2022

# TrackPad detect software utility

The purpose of this software is to provide a means for the MSP430 to detect if a TrackPad is connected prior to performing the normal calibration after a POR. If a TrackPad is detected, normal calibration is performed. If a TrackPad is not connected, the software remains in a while-loop. This prevents the CapTIvate library firmware from becoming stuck in an endless calibration loop and returns control to the MCU.

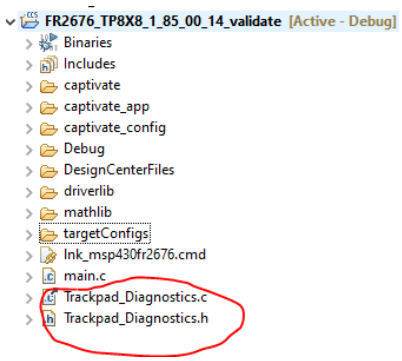
# Description

The software captures a known good calibration from one of the TrackPad elements then perform an absolute capacitance measurement on that element. Both the calibration and absolute measurements are stored in FRAM. On subsequent POR cycles the software use these values and compare the results of the absolute capacitive measurement to see if the TrackPad is connected. When a TrackPad is connected, the absolute capacitive measurement will have some value, which depends on the capacitance of the system being tested (ex. 9000). When a TrackPad is not connected, the same measurement will result in a very small number ~ 0 or even negative.

The following can be performed on a single development unit, but should be close to what is being manufactured in production. The results collected from this unit can then be used for all the production units.

# Instructions

1. Add the two files, TrackPad_Diagnostics.c, and TrackPad_Diagnostics.h to your project



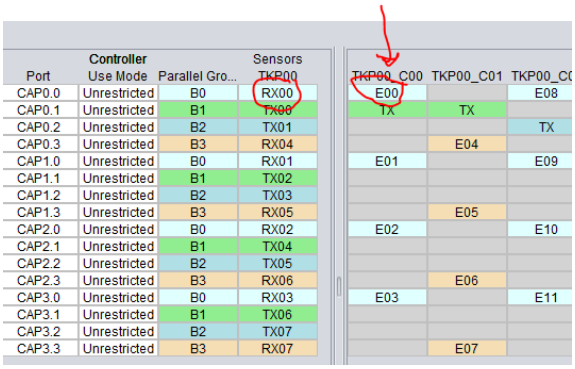
2. Modifications to file CAPT_App.c - modify as shown here. Be sure to place correctly as shown. Note, the purpose of the while-loop is to catch when a TrackPad is not connected and allows the MCU to respond to host commands. You will need to replace this with whatever functionality you need.

```
83 void CAPT_appStart(void)
84 {
85     //
86     // Initialize the user interface.
87     // This function call enables the CapTIvate peripheral,
88     // initializes and enables the capacitive touch IO.
89     //
90     MAP_CAPT_initUI(&g_uiApp);
91
92 #if (TRACKPAD_DIAGNOSTICS == ENABLED)
93     if(Touchpad_isConnected(&TKP00, TEST_CYCLE, TEST_ELEMENT) == false)
94             while(1);
95 #endif
96
97     //
98     // Load the EMC configuration, if this design has
99     // noise immunity features enabled.  This function call
100    // associates an EMC configuration with the EMC module.
101    //
102 #if (CAPT_CONDUCTED_NOISE_IMMUNITY_ENABLE==true)
103    MAP_CAPT_loadEMCConfig(&g_EMCConfig);
104 #endif
105
106    //
107    // Calibrate the user interface.  This function establishes
108    // coarse gain, fine gain, and offset tap tuning settings for
109    // each element in the user interface.
110    //
111    MAP_CAPT_calibrateUI(&g_uiApp);
112
113 #if (TRACKPAD_DIAGNOSTICS == DISABLED)
114    storeTestElementCalibration(&TKP00, 0,0);
115 #endif
116
117    //
118    // Setup Captivate timer
```

3. Add the following #include “TrackPad_Diagnostic.h” to the top of CAPT_App.c as shown.

```
39
40 #include "captivate.h"
41 #include "Trackpad_Diagnostics.h"
```

4. Open the CapTIvate Design Center project for your TrackPad sensor and locate RX0 in the controller view. It should be assigned to element E00 and is located in cycle 0.

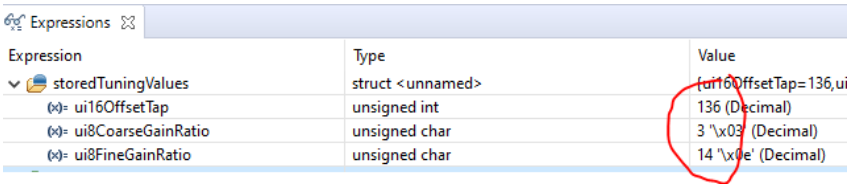


| Controller | | | Sensors | | | |
|---|---|---|---|---|---|---|
| Port | Use Mode | Parallel Gro... | TKP00 | TKP00_C00 | TKP00_C01 | TKP00_C0 |
| CAP0.0 | Unrestricted | B0 | RX00 | E00 | | E08 |
| CAP0.1 | Unrestricted | B1 | TX00 | TX | TX | |
| CAP0.2 | Unrestricted | B2 | TX01 | | | TX |
| CAP0.3 | Unrestricted | B3 | RX04 | | E04 | |
| CAP1.0 | Unrestricted | B0 | RX01 | E01 | | E09 |
| CAP1.1 | Unrestricted | B1 | TX02 | | | |
| CAP1.2 | Unrestricted | B2 | TX03 | | | |
| CAP1.3 | Unrestricted | B3 | RX05 | | E05 | |
| CAP2.0 | Unrestricted | B0 | RX02 | E02 | | E10 |
| CAP2.1 | Unrestricted | B1 | TX04 | | | |
| CAP2.2 | Unrestricted | B2 | TX05 | | | |
| CAP2.3 | Unrestricted | B3 | RX06 | | E06 | |
| CAP3.0 | Unrestricted | B0 | RX03 | E03 | | E11 |
| CAP3.1 | Unrestricted | B1 | TX06 | | | |
| CAP3.2 | Unrestricted | B2 | TX07 | | | |
| CAP3.3 | Unrestricted | B3 | RX07 | | E07 | |

5. In TrackPad_Diagnostics.h, set TEST_CYCLE, TEST_ELEMENT and TRACKPAD_DIAGNOSTICS as shown below.

```
#define TEST_CYCLE                  0
#define TEST_ELEMENT                0
#define ENABLED                     true
#define DISABLED                    false
#define TRACKPAD_DIAGNOSTICS        DISABLED
```

6. Build the project and program the MCU in debug mode. Make sure a TrackPad is connected.
7. Start the MCU and allow the CapTIvate library to perform a normal calibration.
8. Halt the debugger and add the following to the expressions window in CCS



Expressions

| Expression | Type | Value |
|---|---|---|
| storedTuningValues | struct <unnamed> | {ui16OffsetTap=136,ui |
| ui16OffsetTap | unsigned int | 136 (Decimal) |
| ui8CoarseGainRatio | unsigned char | 3 '\x03' (Decimal) |
| ui8FineGainRatio | unsigned char | 14 '\x0e' (Decimal) |

9. In file TrackPad_Diagnostics.c copy these values to the variable ‘storedTuningValues’ as shown. These are the specific normal calibration values for element 0 in cycle 0, which is going to be our test element for detecting the TrackPad.

```
#pragma PERSISTENT(storedTuningValues)
//#pragma LOCATION(storedTuningValues, 0x1800)
tCaptivateElementTuning storedTuningValues =
{
    .ui16OffsetTap = 136,
    .ui8CoarseGainRatio = 3,
    .ui8FineGainRatio = 14
};
#pragma PERSISTENT(iq16ExpectedCapSize)
_iq16 iq16ExpectedCapSize = 0;
```

10. In TrackPad_Diagnostics.h, modify TRACKPAD_DIAGNOSTICS as shown.

```
#define TEST_CYCLE                  0
#define TEST_ELEMENT                0
#define ENABLED                     true
#define DISABLED                    false
#define TRACKPAD_DIAGNOSTICS        ENABLED
```

11. Rebuild the project, program the MCU in debug mode. Set a breakpoint in function Touchpad_isConnected() as shown.

```
bool Touchpad_isConnected(tSensor *pSensor, uint8_t ui8Cycle, uint8_t ui8Element)
{
    uint32_t ui32EstCapSize;
    _iq16 iq16MeasuredCapSize;
    uint8_t i;

    loadTestElementCalibration(&TKP00,ui8Cycle, ui8Element);

    // Estimate the cap size 8 times and take an average across the 8 samples
    ui32EstCapSize = 0;
    for (i=0; i<8; i++)
    {
        ui32EstCapSize += Grip_estElementCapacitance(pSensor, ui8Cycle, ui8Element);
    }
    iq16MeasuredCapSize = _IQ16(ui32EstCapSize >> 3);

    __nop();

```

12. Run the MCU. When the program halts, add to the expressions window iq16MeasuredCapsize.

| Name | Type | Value |
|---|---|---|
| (x)= i | unsigned char | 8 '\x08' |
| (x)= iq16MeasuredCapSize | long | 8894.0 (Q-Value(16)) |

13. Copy that value and place in TrackPad_Diagnostics.c as shown here:

```
#pragma PERSISTENT(storedTuningValues)
//#pragma LOCATION(storedTuningValues, 0x1800)
tCaptivateElementTuning storedTuningValues =
{
     .ui16OffsetTap = 136,
     .ui8CoarseGainRatio = 3,
     .ui8FineGainRatio = 14
};
#pragma PERSISTENT(iq16ExpectedCapSize)
_iq16 iq16ExpectedCapSize = 8894.0;
```

14. Rebuild the project and program the MCU in debug mode. Run the MCU with the TrackPad connected. After a few seconds confirm the TrackPad is working normally. Halt the debugger.

15. Disconnect the TrackPad. Reset the MCU and start again in debug mode. After a few seconds halt the debugger. The code should be stuck in the while-loop as shown.

```
90      MAP_CAPT_initUI(&g_uiApp);
91
92  #if (TRACKPAD_DIAGNOSTICS == ENABLED)
93      if(Touchpad_isConnected(&TKP00, TEST_CYCLE, TEST_ELEMENT) == false)
94              while(1);
95  #endif
```

You are done. You can use these values for the production units as the test element calibration and absolute capacitive measurements will be very similar from unit to unit.