AN12546

A Method of Enabling More Touch Pads With 5-ch CapTouch Module on LPC804 MCU

Rev. 0 — 30/07/2019 Application Note

Contents

1 Overview

LPC804 is aimed to provide the ultra-low cost MCU solution for customer, with simple hardware and easy-of-use software. There is a Capacitive Touch module (would be called "CapTouch" for short later) that provides the ability of touch sensing interface, to improve the experience of human & machine interaction.

The CapTouch module on LPC804 supports only 5 channels by hardware. Normally, the device with up to 5 channels can fulfill some simple use cases. But if more channels are required by the original CapTouch module, the LPC804 might not support the usage. Even by the LPC845, the CapTouch module support up to 9 channels.

However, the SWM module on LPC804 can remap the CapTouch function to almost all the GPIO pins, while the CapTouch feature was the fixed function for pins on LPC845. With suitable software and the settings to CapTouch, the LPC804 use the different pins at different time slice, then it can support more channels than the hardware limited count by software. For example, 12 channels of normal dialing keyboard, or more.

This application note illustrates the usage of CapTouch and SWM to support more channels beyond the hardware limitation.

2 MCU modules & Hardware boards

A brief review about the CapTouch and SWM module is provided in the following sections:

2.1 CapTouch module

LPC CapTouch module on LPC800 series MCU is a capacitive touch interface to detect the touch event. The design of the sensor for LPC CapTouch is just the conductive trace on PCB, with only one additional capacitor for measurement. There are multiple X pins (X0, X1, X2, ... Xn) for each channel and sharing one group of Y signals (YH and YL). To sense a channel, there are successive pulses from the responding X pin to transfer the energy to Y through the touch pad. These pulses drive the process of "charging" and "recharging" automatically controlled by the state machine inside the CapTouch module. A hardware comparator is used to measure the voltage level on the YH pin, and check if the accumulated energy is enough to cross the indicated threshold. The count of cycles during the accumulation is the sensing value for current channel and kept in the TOUCH register of CapTouch module. Then it resets and continue to do the same work to next channel. Finally, it finishes the scan by the end of the channel.

If, these channels are enabled during the scan (or the so-called "poll" in User Manual document), the current channel number is increased one by one automatically by hardware.

NOTE

Here the "channel" is the original CapTouch channel within the CapTouch module. On the LPC845, the CapTouch channels were bonded with the fixed pins, so the board CapTouch channel is just the module CapTouch channel. However, on the LPC804, the CapTouch channels can remap to various board pins through the SWM module. Then, one module CapTouch channel is mapped to one board pin at one time but time division multiplexed to multiple board pins in the whole application's life cycle.

2.2 SWM module

LPC SWM module (Switch Matrix) manages all the functional pins with two groups: one is flexible functional pins, the other is fixed functional pins.

The switch matrix controls the function of each digital or mixed analog/digital pin in a highly flexible way. It allows the connections from many functions like the USART, SPI, CTimer, Capacitive Touch, and I2C functions to any pin that is not power or ground.

Functions that need specialized pads can be enabled or disabled through the switch matrix. These functions are called fixed-pin functions and cannot move to other pins. Only when a fixed-pin function is disabled, any other movable function can be assigned to this pin.

On LPC845, the CapTouch pins are assigned into the fixed functional group, while the SWM cannot remap their signals to various pins, and then the way of extending CapTouch channel count in this paper does not take effect on it. But for LPC804, the CapTouch pins are assigned into the flexible functional group. This feature lay a good foundation to support the method of extend CapTouch channel count, so that the software can beyond the hardware limitation.

2.3 Board and Connections

To verify the solution in this paper, a special touch panel board is created. In the "LPC Touch Panel v1.1" board, there are 12 touch pads on it. All the layouts are on the same side, which provides the opportunity of other layouts for additional components, for example, the LEDs. The Touch Panel board can be showed in Figure 1.

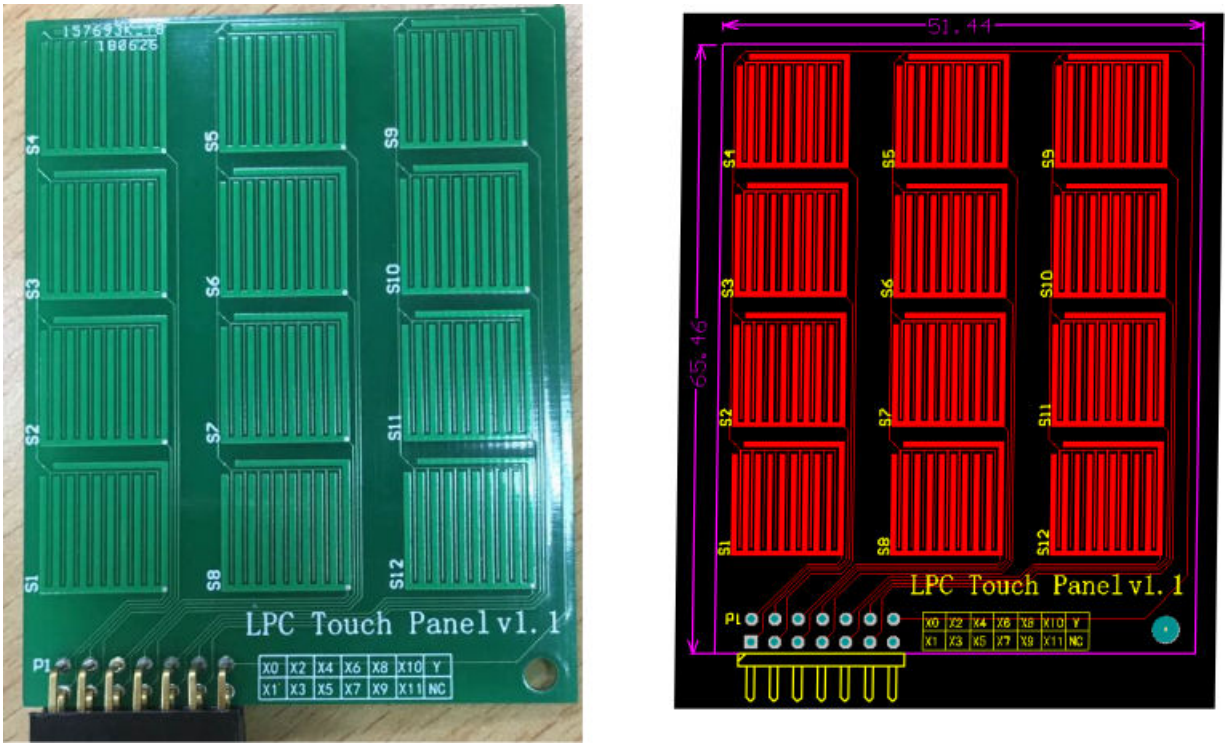

Figure 1. CapTouch pins in SWM module in UM document

As the User Manual said, all the pins in PIO0 port except PIO0_6 and PIO0_31 can be used as CapTouch function through the SWM module. See Figure 2.

8.5.9 Pin assign register 8

Table 100. Pin assign register 8 (PINASSIGN8, address 0x4000 C020) bit description

Bit	Symbol	Description	Reset value
7:0	CAPT_X0_O	CAPT_X0 function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_5 (= 0x5) and from PIO0_7 (= 0x7) to PIO0_30 (= 0x1E).	0xFF
15:8	CAPT_X1_O	CAPT_X1 function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_5 (= 0x5) and from PIO0_7 (= 0x7) to PIO0_30 (= 0x1E).	0xFF
23:16	CAPT_X2_O	CAPT_X2 function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_5 (= 0x5) and from PIO0_7 (= 0x7) to PIO0_30 (= 0x1E).	0xFF
31:24	CAPT_X3_O	CAPT_X2 function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_5 (= 0x5) and from PIO0_7 (= 0x7) to PIO0_30 (= 0x1E).	0xFF

8.5.10 Pin assign register 9

Table 101. Pin assign register 9 (PINASSIGN9, address 0x4000 C024) bit description

Bit	Symbol	Description	Reset value
7:0	CAPT_X4_O	CAPT_X4 function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_5 (= 0x5) and from PIO0_7 (= 0x7) to PIO0_30 (= 0x1E).	0xFF
15:8	CAPT_YL_O	CAPT_YL function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_5 (= 0x5) and from PIO0_7 (= 0x7) to PIO0_30 (= 0x1E).	0xFF
23:16	CAPT_YH_O	CAPT_YH function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_5 (= 0x5) and from PIO0_7 (= 0x7) to PIO0_30 (= 0x1E).	0xFF
31:24	-	Reserved	-

Figure 2. CapTouch pins in SWM module in UM document

The pins lead out from LPC804 chip (TSSOP24) to the touch panel interface. See Figure 3.

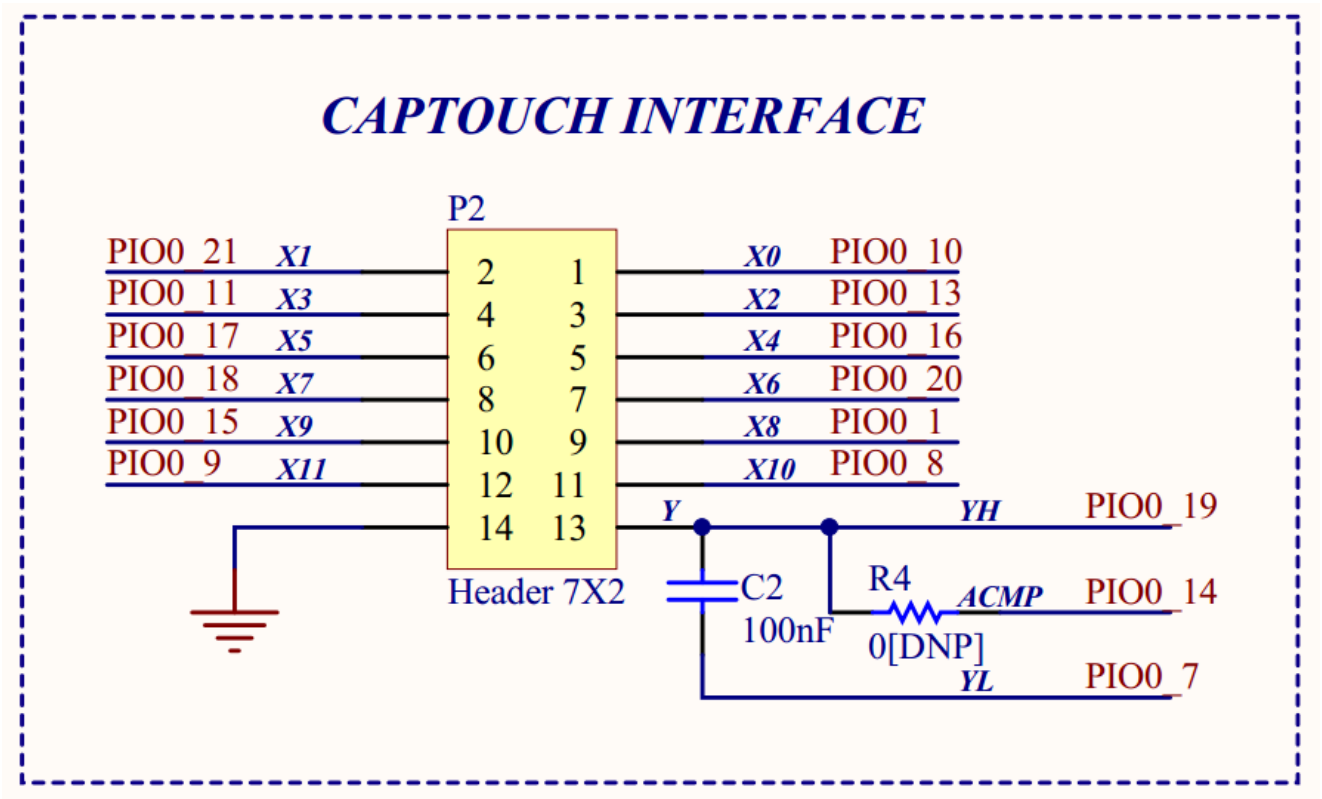

Figure 3. LPC804 pins for CapTouch interface in schematic

The signals are connected through the touch interface. The connections are listed in Table 1.

Table 1. CapTouch Pin Assignments

CapTouch Pin Assignments		
Functional Pin	**LPC804 Pin**	**Description**
UART_TX	PIO0_4	Comm Port
UART_RX	PIO0_0	Comm Port
CAPT_X0	PIO0_10	Touch X
CAPT_X1	PIO0_21	Touch X
CAPT_X2	PIO0_13	Touch X
CAPT_X3	PIO0_11	Touch X
CAPT_X4	PIO0_16	Touch X
CAPT_X5	PIO0_17	Touch X
CAPT_X6	PIO0_20	Touch X

Table continues on the next page...

Table 1. CapTouch Pin Assignments (continued)

CapTouch Pin Assignments		
CAPT_X7	PIO0_18	Touch X
CAPT_X8	PIO0_1	Touch X
CAPT_X9	PIO0_15	Touch X
CAPT_X10	PIO0_1	Touch X
CAPT_X11	PIO0_8	Touch X
CAPT_YH	PIO0_9	Touch YH
CAPT_YL	PIO0_7	Touch YL

Finally, there are LPC804 Touch Core board v1.0 and LPC Touch Panel board v1.1 connected as shown in Figure 4.

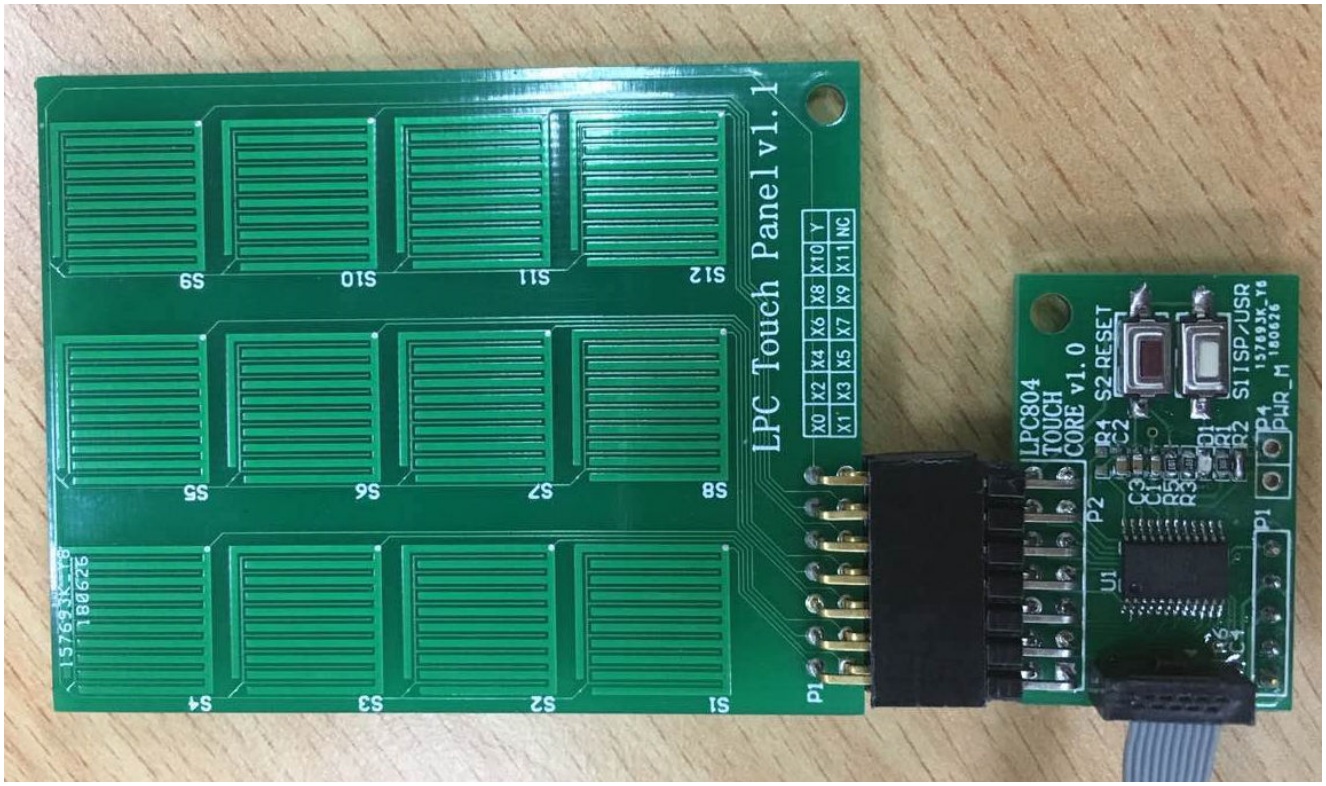

Figure 4. LPC804 Touch Core and LPC Touch Panel boards

3 Software & Algorithm

The idea is to use more than one short hardware-poll sequence (with up to 5 hardware channels) to assemble a longer sequence (up to 12 channels in this case) by software. The way of getting single sensing value for each channel is same as it is for 5 channels use case, while all the sensing values are kept in the memory. But for the short scan sequence, the software uses an index to record the current shorter sequence (execute automatically by CapTouch hardware), and switch to use different group of pins as CapTouch function with SWM. Once the short sequence is done, it asserts the hardware POLL_DONE interrupt flag. Then, the

software update the sequence index, update the pins for CapTouch, and start a new short sequence as a part of whole long sequence.

The CapTouch interrupt is triggered when the YES_TOUCH/NO_TOUCH or POLL_DONE event happens, due to the settings of initialization. The interrupt service routine for CapTouch is executed every time the CapTouch interrupt is triggered. This is the common one for all the CapTouch usage. Its diagram is shown as in Figure 5.

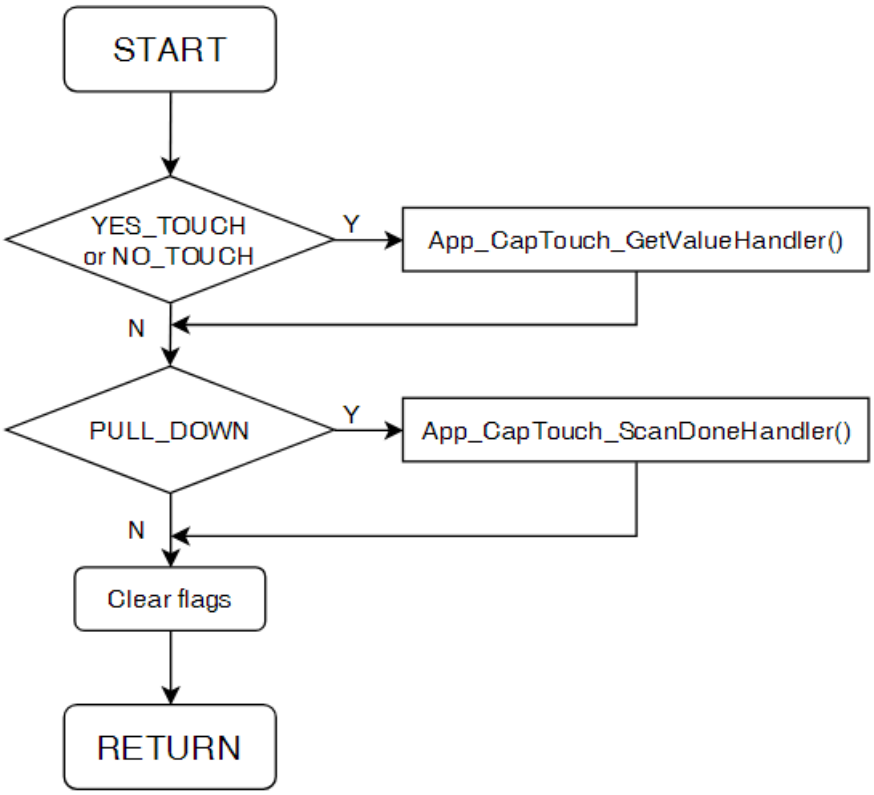

Figure 5. Diagram of CapTouch ISR

However, the best tricks are the implementation of the processing channel/sequence scan done event in App_CapTouch_GetValueHandler() / App_CapTouch_ScanDoneHandler() functions.

Before implementing the algorithm, some tables and variables should be prepared in your project.

```
/* Total 12 channels. */
#define APP_CAPTOUCH_ALL_X_COUNT 12 /* 12 channels for all. */

/* X pin masks to launch the scan. */
#define APP_CAPTOUCH_GROUP_0_X_COUNT 5 /* 5 channels in group 0. */
#define APP_CAPTOUCH_GROUP_1_X_COUNT 5 /* 5 channels in group 1. */
#define APP_CAPTOUCH_GROUP_2_X_COUNT 2 /* 2 channels in group 2. */
#define APP_CAPTOUCH_GROUP_COUNT 3 /* totally 3 groups. */
const uint32_t cAppCapTouchGroupXCount[] =
{
    APP_CAPTOUCH_GROUP_0_X_COUNT,
    APP_CAPTOUCH_GROUP_1_X_COUNT,
    APP_CAPTOUCH_GROUP_2_X_COUNT
};

/* Channels mask for hardware sequence. */
```

```
#define APP_CAPTOUCH_GROUP_0_X_MASK ((1U << APP_CAPTOUCH_GROUP_0_X_COUNT) - 1U)
#define APP_CAPTOUCH_GROUP_1_X_MASK ((1U << APP_CAPTOUCH_GROUP_1_X_COUNT) - 1U)
#define APP_CAPTOUCH_GROUP_2_X_MASK ((1U << APP_CAPTOUCH_GROUP_2_X_COUNT) - 1U)
const uint32_t cAppCapTouchGroupXMask[] =
{
    APP_CAPTOUCH_GROUP_0_X_MASK,
    APP_CAPTOUCH_GROUP_1_X_MASK,
    APP_CAPTOUCH_GROUP_2_X_MASK,
};

/* Functions to set CapTouch pins. */
void App_CapTouch_SetupPinsForXInGroup0(void);
void App_CapTouch_SetupPinsForXInGroup1(void);
void App_CapTouch_SetupPinsForXInGroup2(void);
void (*fAppCapTouchSetGroupPins[])(void) =
{
    App_CapTouch_SetupPinsForXInGroup0,
    App_CapTouch_SetupPinsForXInGroup1,
    App_CapTouch_SetupPinsForXInGroup2
};

volatile uint32_t gAppCapTouchCurGroupIdx; /* to keep the current index of x pin group. */
volatile uint32_t gAppCapTouchValues[APP_CAPTOUCH_ALL_X_COUNT]; /* to keep sensing values in groups.
*/
```

According to the code, it can be seen that the whole scan sequence (the longer one) is divided into 3 shorter one, which can be executed by hardware CapTouch module directly. There is X pin mask value table (cAppCapTouchGroupXMask[]) and their setup function table (fAppCapTouchSetGroupPins[]) to launch the scan for shorter sequence. There is also a table to keep the sensing values (gAppCapTouchValues[]). All the tables are organized and can be indexed with the variable "gAppCapTouchCurGroupIdx", which connects the shorter sequences and assemble them to be the longer one. In the main() function, the whole scan work is started from the first group with enabling the continuous scan mode of CapTouch hardware module.

```
int main(void)
{

...

/*
* CapTouch: Start from Group 0.
*/

App_CapTouch_Init();

gAppCapTouchCurGroupIdx = 0u;
fAppCapTouchSetGroupPins[gAppCapTouchCurGroupIdx]();
App_CapTouch_StartPollContinuous(cAppCapTouchGroupXMask[gAppCapTouchCurGroupIdx]);

while (1)
    {
    }
}
```

In the App_CapTouch_Init() function, it setups the control timing of the working CapTouch module. And the most important is, to enable the interrupt for channel scan done event (YES_TOUCH and NO_TOUCH) and sequence scan done event (POLL_DONE). Then, once the channel scan done event happens, the channel's sensing value can be kept in the user-defined memory. And, once the sequence scan done event happens, the software would switch to the next group for next shorter sequence in the longer whole sequence.

When configuring the CapTouch converter, a strategy is to set the interval between sequence as longer as possible (shorter interval would have faster response but with more power consumption) when running the original continuous mode automatically by hardware. However, in the case of this paper right now, we make an original hardware sequence scan as a one-time sequence (it is different from the POLLNOW mode, POLLNOW executes the sensing task once a time but the sequence includes a series of successive sensing tasks). The one-time sequence works with original continuous mode but stops when the sequence is done. It is controlled by software in interrupt service routine when the POLL_DONE flag is on. Then the software would launch the new sequence later when necessary. That means, the hardware sequence interval (originally set in LPC_CAPT->POLL_TCNT[POLL] register) would be ignored actually in the case here, it would be totally controlled by software.

Finally, let us come to see the App_CapTouch_GetValueHandler() and App_CapTouch_ScanDoneHandler() functions.

```
/* This function would be called when the channel scan is done. */
void App_CapTouch_GetValueHandler(void)
{
    uint32_t val = LPC_CAPT->TOUCH;
    gAppCapTouchGroupValues[gAppCapTouchCurGroupIdx][(val & TOUCH_XVAL) >> 12U] = val &
TOUCH_COUNT;
}

/* This function would be callbed when the sequence scan is done. */
void App_CapTouch_ScanDoneHandler(void)
{
    /* Disable the scan first. */
    App_CapTouch_PausePoll();

/* Move to the new group. */
gAppCapTouchCurGroupIdx = (gAppCapTouchCurGroupIdx + 1u) % APP_CAPTOUCH_GROUP_COUNT;
fAppCapTouchSetGroupPins[gAppCapTouchCurGroupIdx]();

/* Start the new sequence scan. */
App_CapTouch_StartPollContinuous(cAppCapTouchGroupXMask[gAppCapTouchCurGroupIdx]);
}
```

As the variables and functions for each group are itemized into tables, the code here looks brief. The work for each function is simple and clear:

- App_CapTouch_GetValueHandler() moves the most recent sensing value into the responding memory in the sensing value table. LPC_CAPT[TOUCH] register keeps the information of most recent sensing channel and value, which is necessary for the algorithm here. See Figure 6 for the fields' description of LPC_CAPT[TOUCH] register.

Bit	Symbol	Description	Reset value	Access
11:0	COUNT	Contains the count value reached at trigger or time-out.	0x0	R/O
15:12	XVAL	Contains the index of the X pin for the current measurement, or lowest X for a multiple-pin poll now measurement.	0x0	R/O
16	ISTOUCH	'1' if the trigger is due to a touch event, '0' if the trigger is due to a no-touch event.	0x0	R/O
17	ISTO	'1' if the measurement resulted in a time-out event, '0' otherwise.	0x0	R/O
19:18	-	Reserved.	-	-
23:20	SEQ	Contains the 4-bit sequence number, which increments at the end of each polling round.	0x0	R/O
30:24	-	Reserved.	-	-
31	CHANGE	Will be '1' for one bus clock at the end of each X measurement while the data are changing, otherwise '0'. Touch data read while this bit is '1' are invalid.		

Figure 6. TOUCH register in CapTouch module

- App_CapTouch_ScanDoneHandler() includes the steps of stopping the most recent done sequence, moving the group index to next, preparing the pins settings for next group, and launching the scan for next group finally. See to Figure 7 as the diagram of switching different group of CapTouch pins.

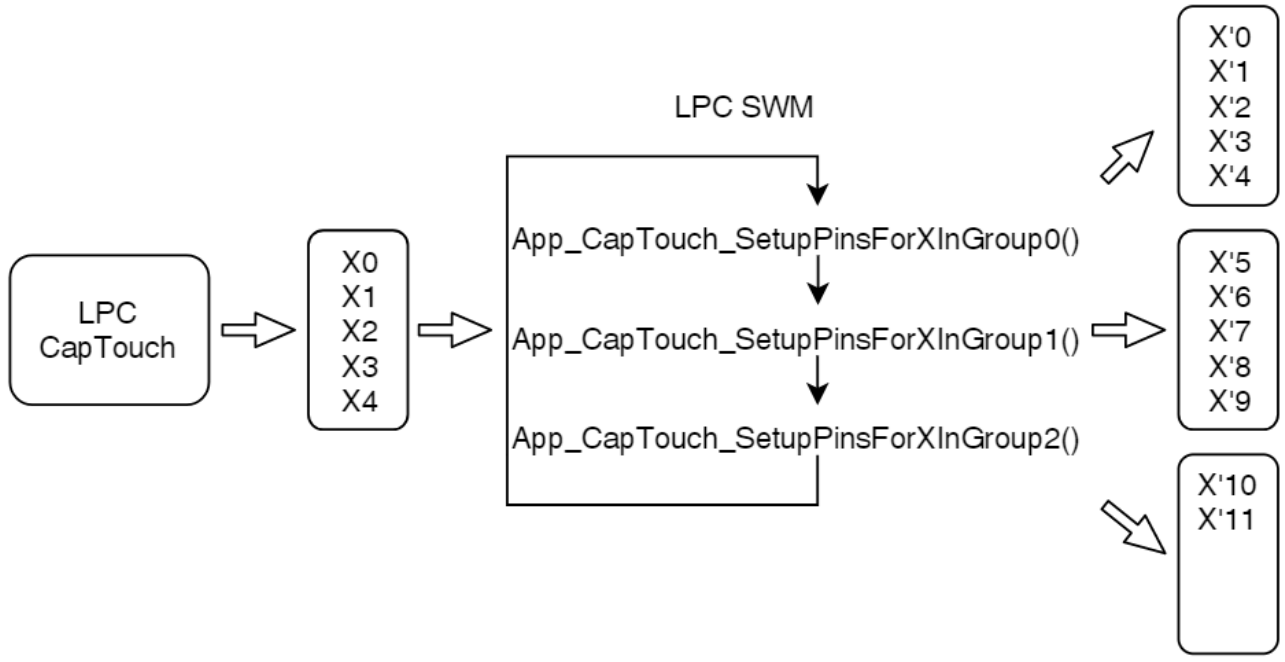

Figure 7. Diagram of switching various groups of CapTouch pins

To shorten the time inside the ISR, the function of setting pin for new group of CapTouch channel is only to remap the pins with SWM module. During the initialization of whole application, which runs only one time at the beginning of the project, it prepares the part of pin configuration in IOCON module.

```
void App_CapTouch_InitPins(void)
{
    /* YH & YL */
    ConfigSWM(CAPT_YH, P0_19);
```

```
        ConfigSWM(CAPT_YL, P0_7 );

        LPC_IOCON->PIO0_19 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_YH. */
        LPC_IOCON->PIO0_7 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_YL */


        /* Group 0, only for IOCON. */
        LPC_IOCON->PIO0_10 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X0. */
        LPC_IOCON->PIO0_21 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X1. */
        LPC_IOCON->PIO0_13 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X2. */
        LPC_IOCON->PIO0_11 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X3. */
        LPC_IOCON->PIO0_16 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X4. */


        /* Group 1, only for IOCON. */
        LPC_IOCON->PIO0_17 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X5. */
        LPC_IOCON->PIO0_20 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X6. */
        LPC_IOCON->PIO0_18 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X7. */
        LPC_IOCON->PIO0_1 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X8. */
        LPC_IOCON->PIO0_15 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X9. */


        /* Group 2, only for IOCON. */
        LPC_IOCON->PIO0_8 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X10. */
        LPC_IOCON->PIO0_9 = IOCON_PIO_MODE(0) | IOCON_PIO_RESERVED; /* CAPT_X11. */
    }
    void App_CapTouch_SetupPinsForXInGroup0(void)
    {
        ConfigSWM(CAPT_X0, P0_10);
        ConfigSWM(CAPT_X1, P0_21);
        ConfigSWM(CAPT_X2, P0_13);
        ConfigSWM(CAPT_X3, P0_11);
        ConfigSWM(CAPT_X4, P0_16);
    }

    void App_CapTouch_SetupPinsForXInGroup1(void)
    {
        ConfigSWM(CAPT_X0, P0_17);
        ConfigSWM(CAPT_X1, P0_20);
        ConfigSWM(CAPT_X2, P0_18);
        ConfigSWM(CAPT_X3, P0_1);
        ConfigSWM(CAPT_X4, P0_15);
    }

    void App_CapTouch_SetupPinsForXInGroup2(void)
    {
        ConfigSWM(CAPT_X0, P0_8);
        ConfigSWM(CAPT_X1, P0_9);
    }
```

4 Functional Verification

A Freemaster project is also created to monitor the sensing values of all the channels. In the GUI panel of Freemaster project, the waveforms of sensing values are shown with their related channel according to touch event. See Figure 8 for waveforms of X0 – X7.

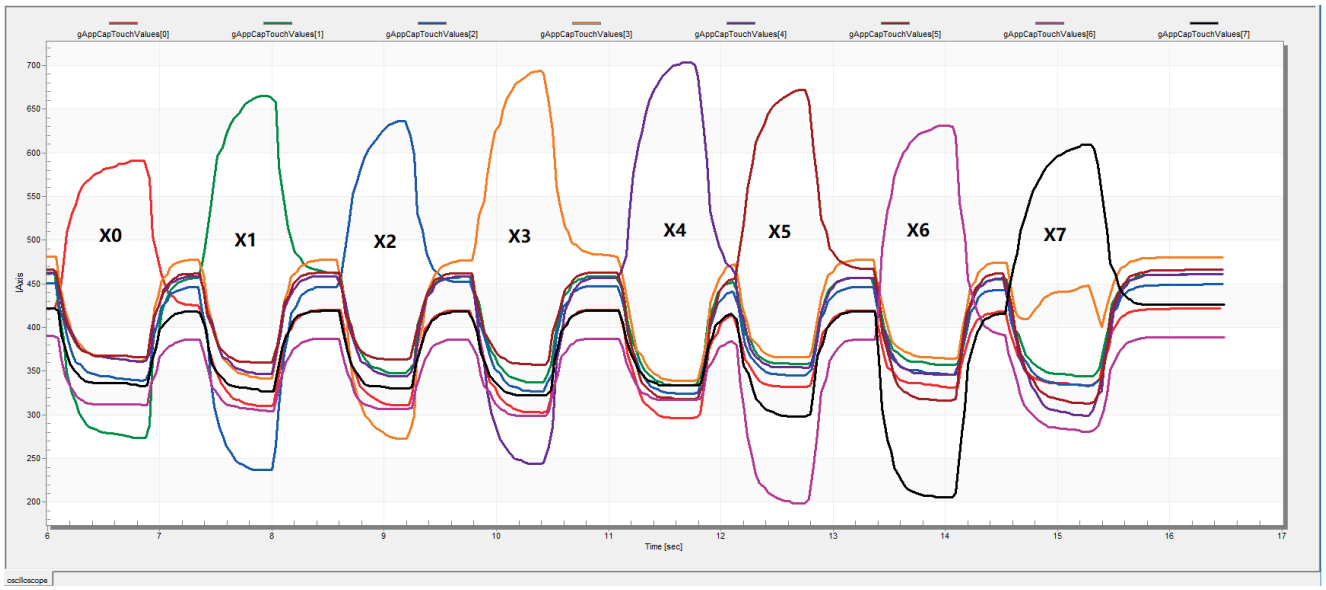

Figure 8. CapTouch waveforms of X0 - X7

Once any pad is touched, all the sensing values become low except for the touched one, and it increases and keeps the highest. As the Freemaster only supports up to 8 variables in a scope page by default, the more channels, the X8 - X11, are not shown in the same panel. To check the waveforms of other sensing values, another scope page is created to include X8 - X11. See Figure 9.

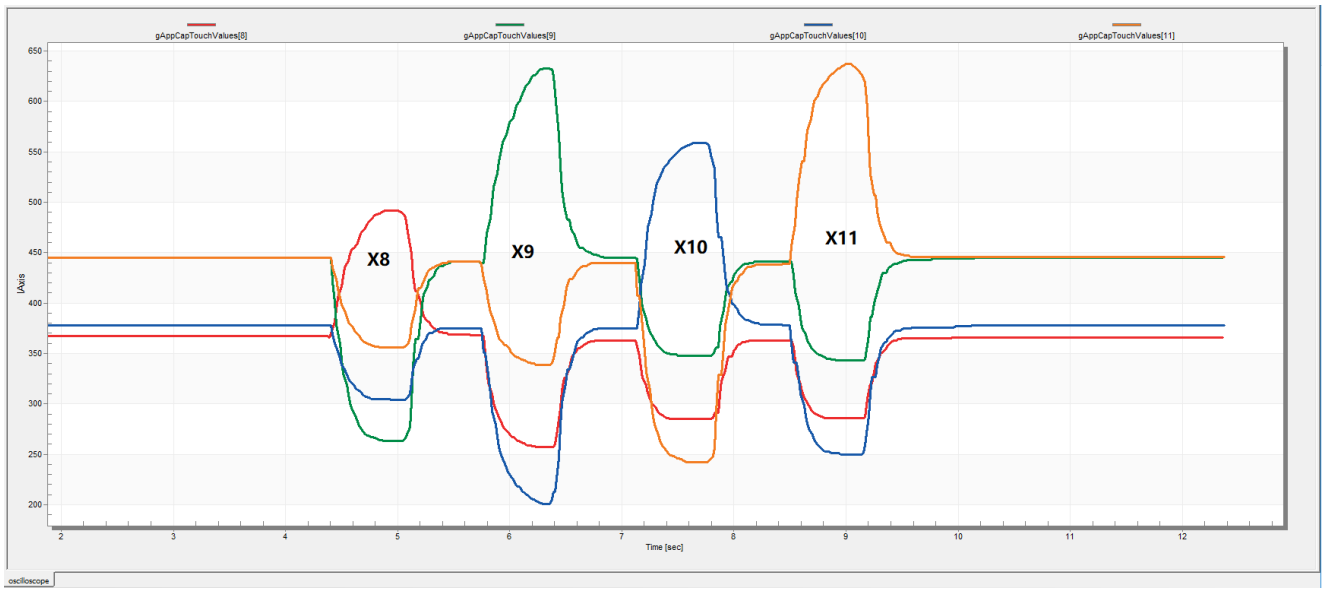

Figure 9. CapTouch waveforms of X8 - X11

According to these waveforms, the software method of enabling more channels beyond the CapTouch hardware limitation is proven to be available.

5 Conclusion

LPC804's CapTouch module is reduced to only 5 hardware channels. But, with the SWM module and suitable software, we create a method to break the limitation of channel count. The method is to switch mapping between CapTouch hardware channels and

IO pins with SWM, and combine the shorter hardware scan sequence into a whole longer one to include more channels simulated by software. After enabling this method, during the application development, user could use the sensing values as same as there were fetched directly from CapTouch hardware regardless its hardware limitation. Then, users can make the calibration, set up the threshold, detect the key value and process the filter as in the traditional development with the original CapTouch hardware. Even the hardware compare feature in CapTouch module is also compatible within this multiplexing method, when executing the shorter sequence, which is an easy way of using CapTouch module to reduce the work load of CPU.

6 Revision history

Table 2.

Revision history		
Rev	**Date**	**Description**
0	30/07/2019	Initial version

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 30/07/2019

Document identifier: AN12546

arm