# AN12546
# 一种基于 LPC804 微控制器的使用五通道 CapTouch 模块实现更多触摸电极的方法

版本 0 — 2019 年 7 月 30 日　　应用笔记

目录

## 1 总览

LPC804 旨在通过简单的硬件和易于使用的软件为客户提供超低成本的 MCU 解决方案。片上集成了一个电容式触摸模块（以下简称为“ CapTouch”）提供触摸感应界面的功能，以改善人机交互体验。

LPC804 上的 CapTouch 模块硬件仅支持五个通道。通常，具有多达五个通道的设备可以满足一些简单的用例。但是，如果原始 CapTouch 模块需要更多通道，则 LPC804 可能不支持该用法。 即使使用 LPC845，CapTouch 模块仅能支持最多九个通道。

但是，LPC804 上的开关矩阵（SWM）模块可以将 CapTouch 功能重新映射到几乎所有的 GPIO 引脚，而 CapTouch 功能是 LPC845 上引脚的固定功能。 通过适当的软件和 CapTouch 的设置，LPC804 在不同的时间段使用不同的引脚，因此它可以支持的通道数量超过了硬件所限制的数量。例如，12 个或更多的普通拨号键盘通道。

本应用笔记说明了 CapTouch 和 SWM 如何用于支持超出硬件限制的更多通道。

## 2 MCU 模块和硬件板

以下各节简要介绍了 CapTouch 和 SWM 模块。

### 2.1 CapTouch 模块

LPC800 系列 MCU 上的 LPC CapTouch 模块是一个互电容式触摸接口，用于检测触摸事件。用于 LPC CapTouch 的传感器的设计只是 PCB 上的导电迹线，仅带有一个用于测量的附加电容器。每个通道有多个 X 引脚（X0，X1，X2，... Xn），并共享一组 Y 信号（YH 和 YL）。为了感测通道，从相应的 X 引脚发出连续的脉冲，以通过电极将能量传递到 Y。这些脉冲驱动 CapTouch 模块内部状态机自动控制的“充放电”和“重新充放电”过程。硬件比较器用于测量 YH 引脚上的电压电平，并检查累积的能量是否足以超过指示的阈值。累积期间的周期数是当前通道的检测值，并保存在 CapTouch 模块的 TOUCH 寄存器中。然后它重置并继续对下一个通道进行相同的处理。最后，它在扫描完通道序列之后完成扫描。

如果在扫描过程中启用了这些通道（或用户手册中的所谓“轮询”），则当前通道号将由硬件自动递增。

注意

这里的“通道”是 CapTouch 模块内的原始 CapTouch 通道。在 LPC845 上，CapTouch 通道与固定引脚相连，因此板上的 CapTouch 通道只是模块 CapTouch 通道。但是，在 LPC804 上，CapTouch 通道可以通过 SWM 模块重新映射到各种 不同的引脚上。虽然一个模块的 CapTouch 通道一次只能映射到一个引脚上，但是在整个应用程序的生命周期中它可以通过时分复用映射到多个引脚上。

### 2.2 SWM 模块

LPC SWM 模块（开关矩阵）通过分两组来管理所有功能引脚：一组是可变功能引脚，另一组是固定功能引脚。

开关矩阵以高度灵活的方式控制每个数字或模拟/数字混合引脚的功能。它允许将许多功能（例如 USART，SPI，CTimer，电容式触摸和 I2C 功能等）连接到任何非电源或接地引脚上。

可以通过开关矩阵启用或禁用需要专用引脚的功能。这些功能称为固定引脚功能，不能移至其他引脚。仅当禁用固定引脚功能时，才能将其他任何可移动功能分配给该引脚。

在 LPC845 上，CapTouch 引脚被分配到固定功能组中，而 SWM 无法将其信号重新映射到各个引脚，因此本文中扩展 CapTouch 通道数的方法对此无效。但是对于 LPC804，CapTouch 引脚被分配到灵活可变功能组中。此功能为支持扩展 CapTouch 通道数的方法奠定了良好的基础，因此可以通过软件扩展硬件的功能。

## 2.3 电路板和连接

为了验证本文中的解决方案，专门设计了一个触摸面板。在“LPC 触摸面板 v1.1”板中，上面有 12 个电极。所有布局都在同一侧，这为其他组件（例如 LED）提供了布局的机会。触摸面板如 图 1 所示。

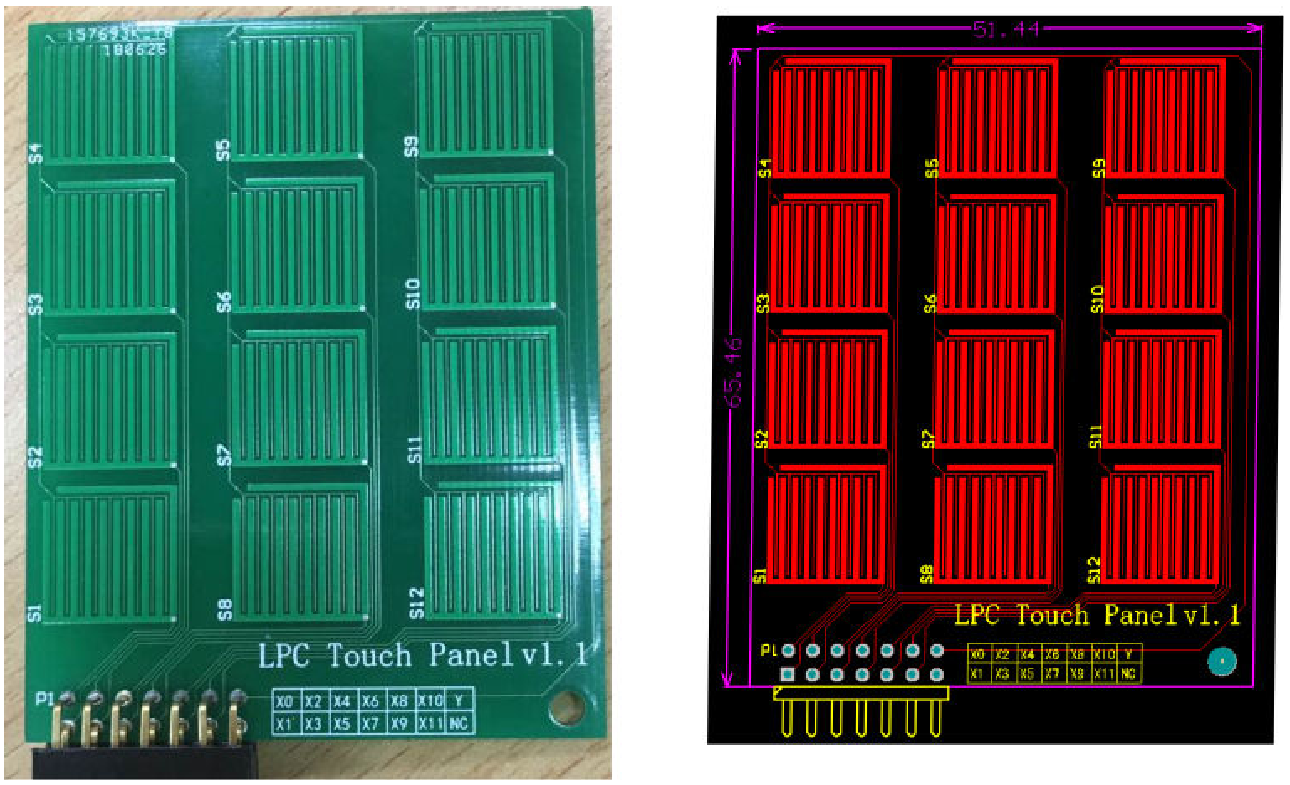
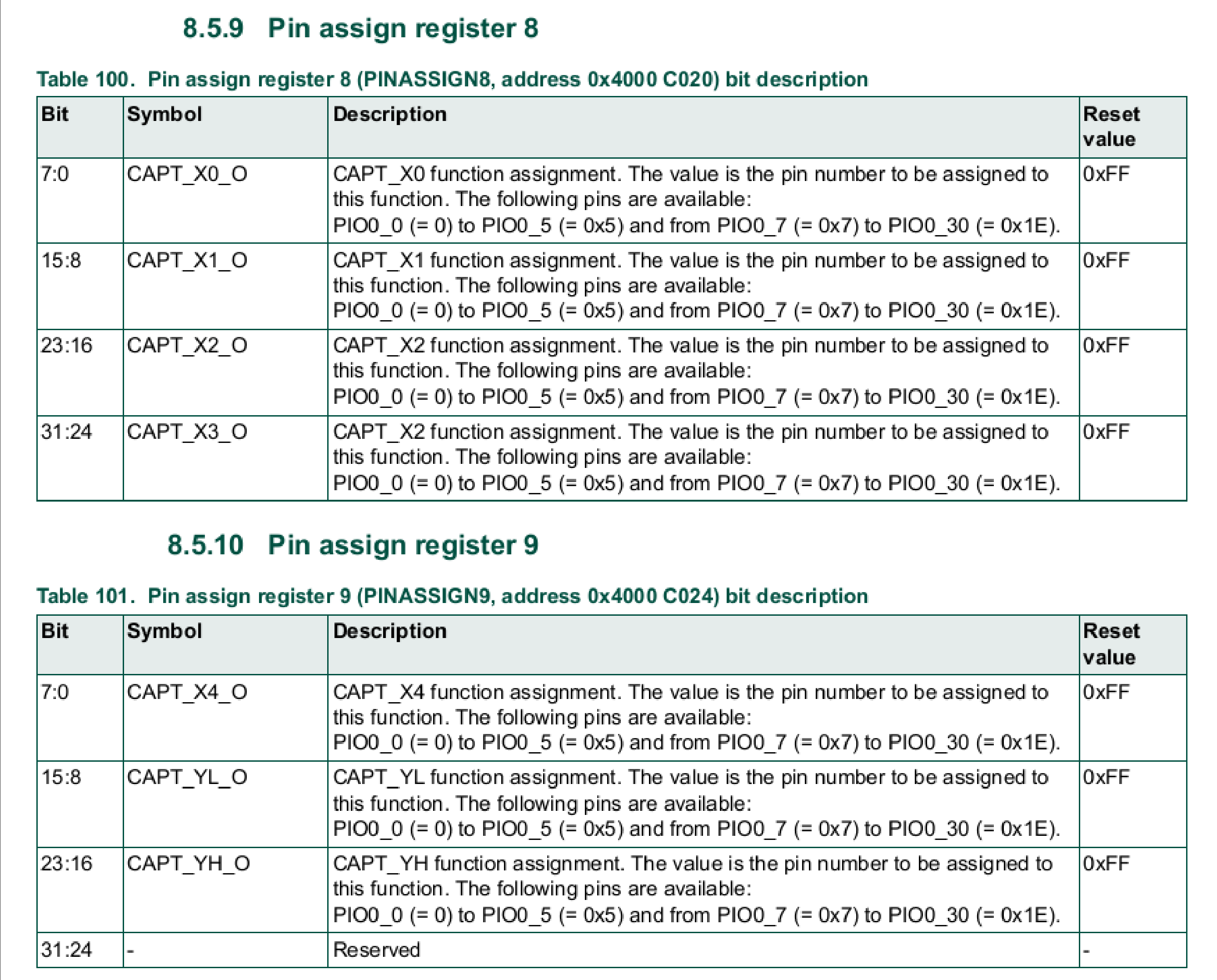


图 1. 支持 12 个按键的 LPC804 触摸面板

如用户手册所述，除 PIO0_6 和 PIO0_31 外，PIO0 端口中的其它引脚都可以通过 SWM 模块用作 CapTouch 功能。见 图 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 | - |

图 2. 用户手册中 SWM 模块中的 CapTouch 引脚

LPC804 芯片（TSSOP24）引脚到触摸面板的接口如 图 3 所示。

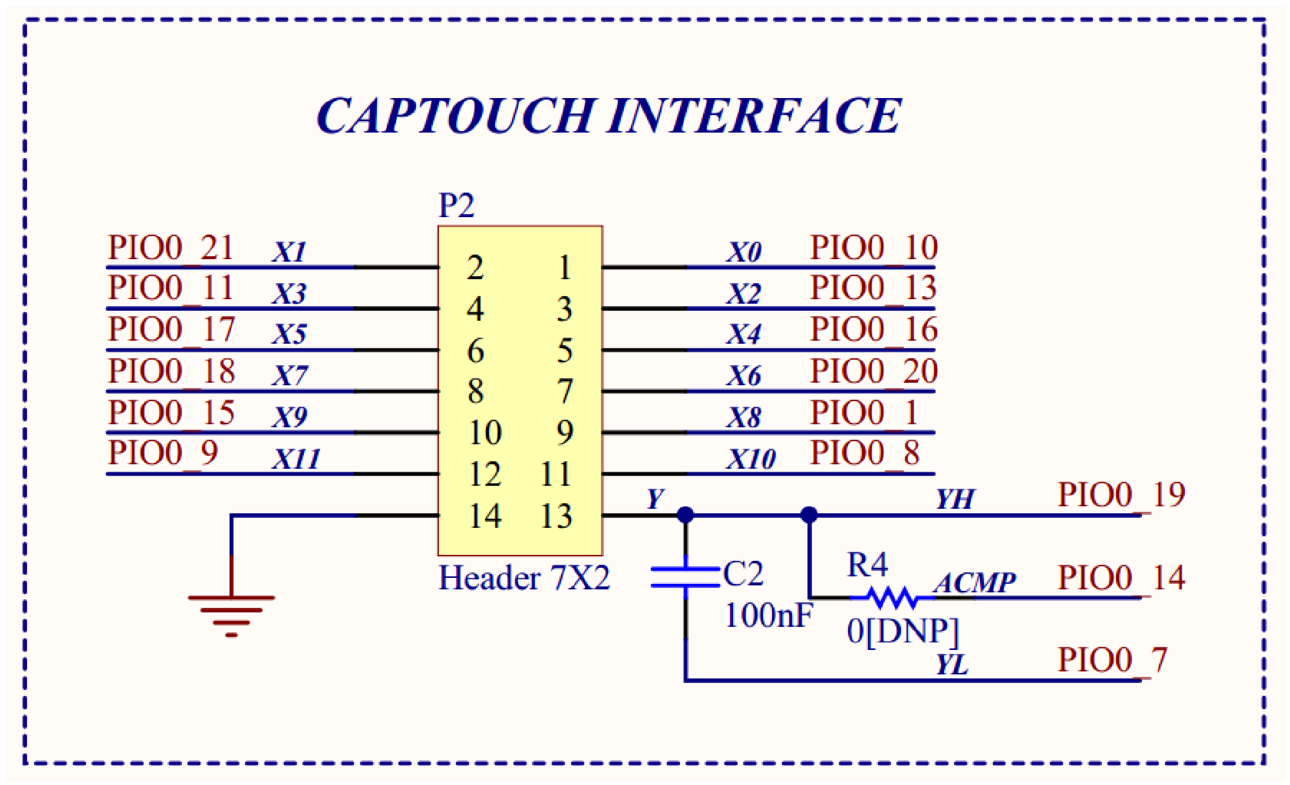


图 3. CapTouch 接口上的 LPC804 引脚分配图

表 1 列出了 LPC804 引脚连接情况。

表 1. CapTouch 引脚分配

| 功能引脚 | LPC804 引脚 | 描述 |
|---|---|---|
| UART_TX | PIO0_4 | 通讯端口 |
| UART_RX | PIO0_0 | 通讯端口 |
| CAPT_X0 | PIO0_10 | 触摸通道 X0 |
| CAPT_X1 | PIO0_21 | 触摸通道 X1 |
| CAPT_X2 | PIO0_13 | 触摸通道 X2 |
| CAPT_X3 | PIO0_11 | 触摸通道 X3 |
| CAPT_X4 | PIO0_16 | 触摸通道 X4 |
| CAPT_X5 | PIO0_17 | 触摸通道 X5 |
| CAPT_X6 | PIO0_20 | 触摸通道 X6 |
| CAPT_X7 | PIO0_18 | 触摸通道 X7 |
| CAPT_X8 | PIO0_1 | 触摸通道 X8 |

*下页继续...*

表 1. CapTouch 引脚分配（续上页）

| 功能引脚 | LPC804 引脚 | 描述 |
|---|---|---|
| CAPT_X9 | PIO0_15 | 触摸通道 X9 |
| CAPT_X10 | PIO0_1 | 触摸通道 X10 |
| CAPT_X11 | PIO0_8 | 触摸通道 X11 |
| CAPT_YH | PIO0_9 | 触摸信号 YH |
| CAPT_YL | PIO0_7 | 触摸信号 YL |

最后，将 LPC804 触摸核心板 v1.0 和 LPC 触摸面板 v1.1 如 图 4 连接起来。

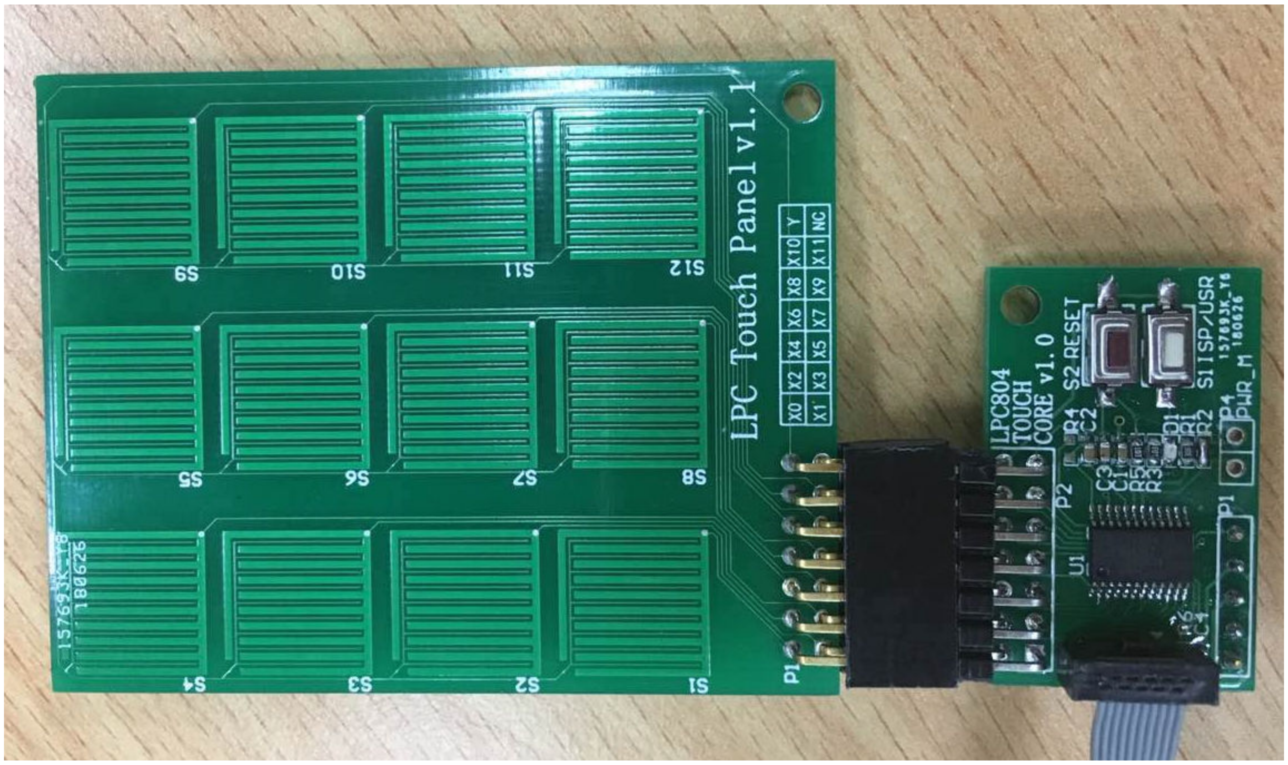


图 4. LPC804 触摸核心板和 LPC 触摸面板的连接图

## 3 软件和算法

实现思路是通过软件将多个短的硬件扫描序列（最多五个硬件通道）组合成一个较长的序列（在本文的示例程序中，最多 12 个通道）。每个通道获取单个感测值的方式与五个通道的使用情况相同，而所有感测值均保存在内存中。但是对于短扫描序列，软件使用索引来记录当前的短序列的编号（短序列由 CapTouch 硬件自动扫描），并通过 SWM 切换到不同的 CapTouch 功能引脚作为下一个短的硬件扫描序列。短序列完成后，它将产生硬件 POLL_DONE 中断标志。然后，软件更新序列索引，更新 CapTouch 的引脚，并开始新的短序列作为整个长序列的一部分。

初始化设置后，当发生 YES_TOUCH / NO_TOUCH 或 POLL_DONE 事件时，会触发 CapTouch 中断。每次触发 CapTouch 中断时，都会执行 CapTouch 的中断服务程序。这是所有 CapTouch 常见用法。其示意图如 图 5 所示。

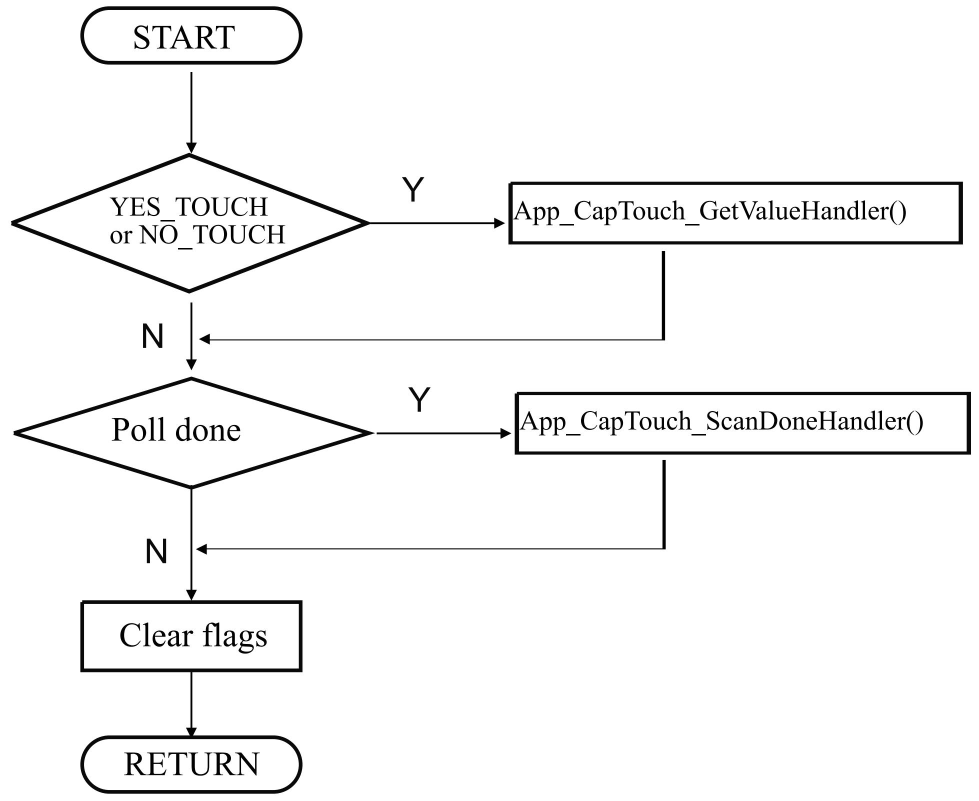

START
YES_TOUCH or NO_TOUCH
Y
App_CapTouch_GetValueHandler()
N
Poll done
Y
App_CapTouch_ScanDoneHandler()
N
Clear flags
RETURN

图 5. CapTouch 中断服务程序（ISR）流程图

但是，最好的方法是在 App_CapTouch_GetValueHandler（ ）/ App_CapTouch_ScanDoneHandler（ ）函数中处理通道/序列扫描完成事件。

在实现算法之前，应在项目中准备一些表和变量。

```
/* 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.
*/
```

从代码中可以看到整个扫描序列（较长的扫描序列）被分为 3 个较短的序列，可以直接由硬件 CapTouch 模块执行。 有 X 引脚掩码值表（cAppCapTouchGroupXMask []）和它们的设置功能表（fAppCapTouchSetGroupPins []）来启动短序列扫描。 还有一个表用于保留感测值（gAppCapTouchValues []）。 所有表都经过组织，可以用变量“ gAppCapTouchCurGroupIdx”进行索引，该变量连接较短的序列并将它们组合成较长的序列。 在 main（ ）函数中，通过启用 CapTouch 硬件模块的连续扫描模式，整个扫描工作从第一组开始。

```
int main(void)
{
...
/*
* CapTouch: Start from Group 0.
*/
App_CapTouch_Init();
gAppCapTouchCurGroupIdx = 0u;
fAppCapTouchSetGroupPins[gAppCapTouchCurGroupIdx]();
App_CapTouch_StartPollContinuous(cAppCapTouchGroupXMask[gAppCapTouchCurGroupIdx]);
while (1)
 {
 }
}
```

在 App_CapTouch_Init（ ）函数中，它设置了 CapTouch 模块的控制时序。 最重要的是，为通道扫描完成事件（YES_TOUCH 和 NO_TOUCH）和序列扫描完成事件（POLL_DONE）启用中断。 然后，一旦发生通道扫描完成事件，即可将通道的感应值保存在用户定义的内存中。 并且，一旦发生序列扫描完成事件，软件将在较长的整个序列中切换到下一组短序列。

配置 CapTouch 转换器时，一种策略是在硬件自动运行原连续模式时，将序列之间的时间间隔设置得尽可能长（间隔越短，响应速度越快，但功耗也就越大）。 但是，在本文的情况下，我们将原硬件序列扫描作为一次性序列进行（与 POLLNOW 模式不同，POLLNOW 一次只执行单个传感任务，但是一个扫描序列包含一系列连续传感 任务）。 一次性序列可在原连续模式下使用，但在序列完成后停止。 它由软件在 POLL_DONE 标志中断服务程序中控制。 然后，该软件将在必要时启动新序列。 这意味着，在这种情况下，实际上将忽略硬件序列时间间隔（最初在 LPC_CAPT-> POLL_TCNT [POLL]寄存器中设置），它将完全由软件控制。.

最后，让我们来看一下 App_CapTouch_GetValueHandler（ ）和 App_CapTouch_ScanDoneHandler（ ）函数。

```
/* 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]);
 }
```

由于每个组的变量和函数都列在表中，因此此处的代码看起来很简短。每个功能的工作都很简单明了：

- App_CapTouch_GetValueHandler（ ）将最新的感应值移到感应值表中的相应内存中。LPC_CAPT [TOUCH]寄存器保留最新的检测通道和其感应值的信息，这是此处算法所必需的。 有关 LPC_CAPT [TOUCH]寄存器的字段说明，请参见 图 6。

| 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. | | |

图 6. CapTouch 模块中的 TOUCH 寄存器

- App_CapTouch_ScanDoneHandler（ ）包括以下步骤：停止最近完成的序列；将组索引移至下一个；为下一个组准备引脚设置；最后启动对下一个组的扫描。参见 图 7 切换不同组 CapTouch 引脚的示意图。

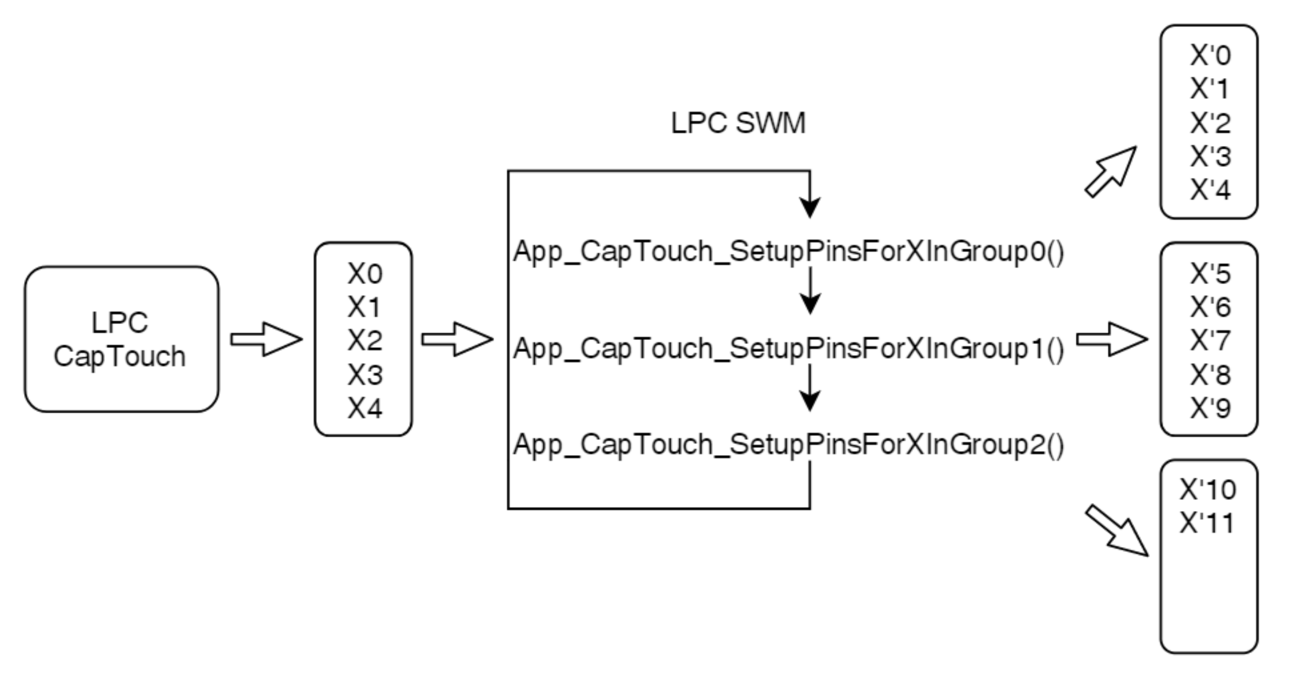

LPC SWM
LPC CapTouch
X0 X1 X2 X3 X4
App_CapTouch_SetupPinsForXInGroup0()
App_CapTouch_SetupPinsForXInGroup1()
App_CapTouch_SetupPinsForXInGroup2()
X'0 X'1 X'2 X'3 X'4
X'5 X'6 X'7 X'8 X'9
X'10 X'11

图 7. 切换各组 CapTouch 引脚的示意图

为了缩短中断服务程序的时间，为新的 CapTouch 通道组设置引脚的功能仅仅是通过 SWM 模块重新映射引脚。在整个应用程序的初始化期间（该过程仅在项目开始时运行一次），它通过 IOCON 模块来配置相应的引脚。

```
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 功能验证

在本例中，创建了一个 FreeMASTER 工程项目，用于观察所有通道的感测值。在 FreeMASTER 项目的 GUI 面板中，根据触摸事件显示感测值的波形及其相应的触摸通道号。X0 – X7 的波形请参见 图 8。

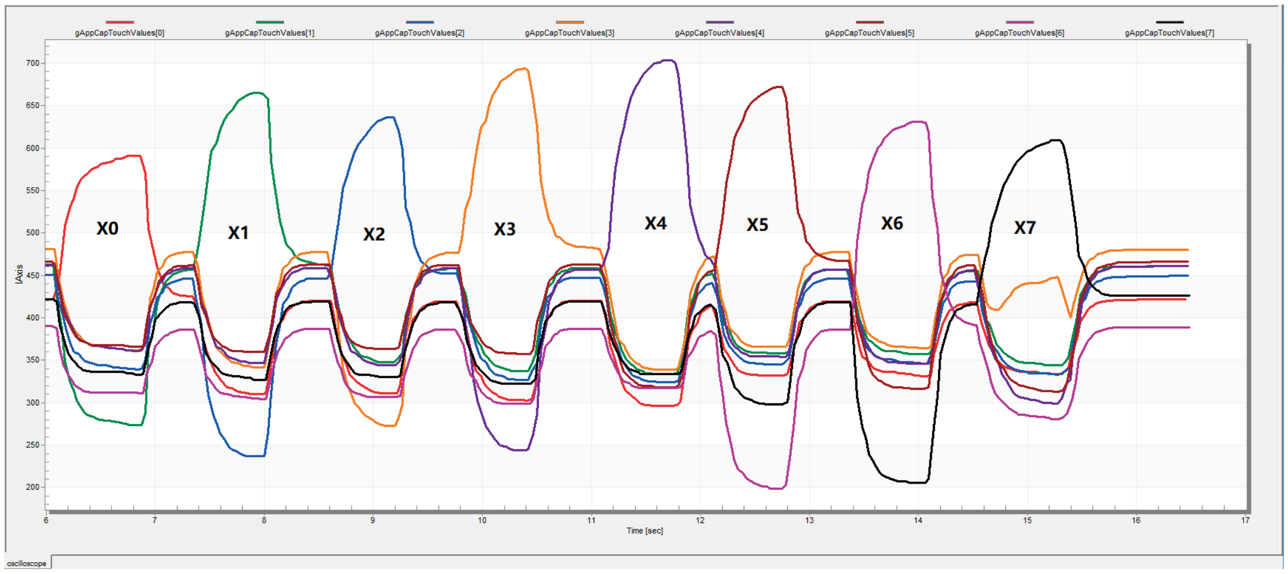


图 8. X0-X7 的 CapTouch 波形

当触摸到触摸板上的任何一个电极，除被触摸的电极以外，其它的感应值都变低。 由于默认情况下在显示页面中最多支持八个通道（新版 FreeMASTER 已经可以支持更多的通道了），因此更多的通道 X8-X11 不会显示在同一面板中。 为了检查其他感测值的波形，将创建另一个示波器页面以包含 X8-X11。参见 图 9。

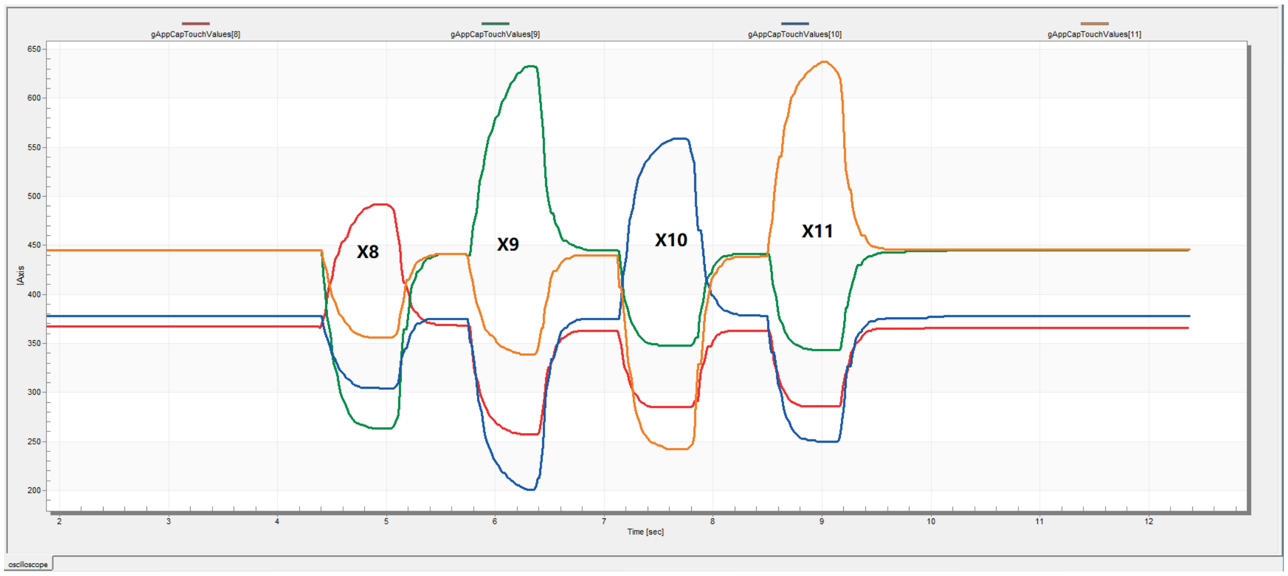


图 9. X8-X11 的 CapTouch 波形

根据这些波形，已证明存在可以支持超出 CapTouch 硬件限制的更多通道的软件方法。

## 5 总结

LPC804 的 CapTouch 模块仅支持五个硬件通道。但是，借助 SWM 模块和合适的软件，我们创建了一种方法来打破通道数量的限制。该方法是使用 SWM 在 CapTouch 硬件通道和 IO 引脚之间切换映射，并将较短的硬件扫描序列组合为更长的整体扫描序列，以包含更多由软件模拟的通道。启用此方法后，在应用程序开发期间，用户可以使用与直接从 CapTouch 硬件获取的值相同的感测值，而不受其硬件限制。然后，用户可以像使用原 CapTouch 硬件的传统开发一样进行校准，设置阈值，检测按键值并进

行滤波器处理。当执行较短的序列时，CapTouch 模块中的硬件比较功能也与这种多路复用方法兼容，这是使用 CapTouch 模块减少 CPU 工作量的简便方法。

## 6 修订记录

| 版本号 | 日期 | 说明 |
|---|---|---|
| 0 | 2019 年 7 月 30 日 | 初始发行 |

Date of release: 2019 年 7 月 30 日

Document identifier: AN12546