FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

T569 –SISTEMAS DE TEMPO REAL

Aula 12b

Prof. Marcelo Sousa

Agenda

- Queue Sets
- Event Groups
- TSI
- TSS

Queue Sets

- Introduzidos na versão 7.4
- Usados quando uma tarefa precisa receber múltiplos eventos de outras tarefas

Queue Sets

- Se estes eventos tiverem características diferentes (uns apenas notificam, outros enviam um byte, outros enviam um buffer de dados, etc), você não pode usar apenas uma Queue ou semáforo para receber estes eventos.

Queue Sets

- Padrão de projeto comum usado nestes casos é criar uma estrutura para representar estes eventos contendo o seu ID e um ponteiro para os dados do evento:

```
typedef struct
{
    int id;
    void *data;
} AppEvent;
```

- Dentro da tarefa, bloqueamos em uma Queue esperando pelo evento, tratando o evento recebido de acordo com seu ID:

```
void vTaskEvent(void * pvParameters)
{
    AppEvent event;
     for(;;)
    {
        xQueueReceive(eventQueue, &event, portMAX_DELAY); /* wait event */

        switch(event.id) {       /* handle event */

            case EVENT_KEY:
                processEventKey(event.data);
                break;

            case EVENT_PRINTER:
                processEventPrinter();
                break;

            case EVENT_DISPLAY:
                processEventDisplay();
                break;
             [...]
        }
    }
}
```

Queue Sets

- Com **Queue Sets**, é possível fazer com que uma tarefa possa ficar ***blocked*** esperando por múltiplos semáforos e/ou Queues ao mesmo tempo

- Para usar esta funcionalidade, é necessário:
  - Criar uma ***Queue Set*** com a função **xQueueCreateSet**();
  - Agrupar semáforos e Queues com a função **xQueueAddToSet**();
  - Bloquear esperando pela recepção de um destes elementos com a função **xQueueSelectFromSet**().

Queue Sets

- Neste caso, o exemplo ficaria:

```
void vTaskEvent(void * pvParameters)
{
    xQueueSetMemberHandle xActivatedMember;
    xQueueSetHandle xQueueSet;
    unsigned char key;

    /* create queue set */
    xQueueSet = xQueueCreateSet(EVENT_MAX);

    /* add queues to queue set */
    xQueueAddToSet(xQueueKey, xQueueSet);

    /* add semaphores to queue set */
    xQueueAddToSet(xSemaphoreDisplay, xQueueSet);
    xQueueAddToSet(xSemaphorePrinter, xQueueSet);

    for( ;; )
```

Queue Sets

- (continuação)

```
for( ;; )
    {   /* wait event */
        xActivatedMember = xQueueSelectFromSet(xQueueSet, portMAX_DELAY);

        /* handle event */
        if(xActivatedMember == xQueueKey) {
            xQueueReceive(xActivatedMember, &key, 0);
            processEventKey(key);
        }
        else if(xActivatedMember == xSemaphoreDisplay) {
            xSemaphoreTake(xActivatedMember, 0);
            processEventDisplay();
        }
        else if(xActivatedMember == xSemaphorePrinter) {
            xSemaphoreTake(xActivatedMember, 0);
            processEventPrinter();
        }
        [...]
    }
}
```

Queue Sets

- Entretanto, a primeira solução tem as seguintes vantagens:
  - A implementação com **Queue Sets** usa mais memória RAM, já que precisamos criar uma queue ou semáforo para cada tipo de evento;
  - O código fica maior, ocupando mais espaço em flash;
  - Consome mais ciclos de CPU, já que o fato do kernel precisar varrer uma queue set leva mais tempo do que varrer uma simples Queue.

- A principal motivação para o uso deste recurso da API é a migração de aplicações para o FreeRTOS, já que outros OS têm uma função que bloqueia em múltiplos objetos do kernel.

Low-power Tickless RTOS

- Nas aplicações de baixo consumo de energia, o sistema deve ficar no modo de menor consumo possível caso nenhuma outra tarefa do sistema esteja pronta para execução.
- Entretanto, o sistema continua acordando a cada tick do sistema.
- Uma solução é usar o Low-power tickless RTOS. Com esta funcionalidade habilitada, o kernel desliga a interrupção do tick durante todo o tempo em que não existir nenhuma tarefa pronta para execução, mantendo a aplicação o máximo de tempo possível em modo de baixo consumo de energia.

Queue Sets

- Pode ser criado com a função xEventGroupCreate():

```
EventGroupHandle_t xEventGroup;

  /* Tenta criar o grupo */
  xEventGroup = FRTOS1_xEventGroupCreate();

  /* Grupo criado com sucesso? */
  if( xEventGroup == NULL )
  {
      /* Erro ao criar o Event group por falta de heap. */
  }
  else
  {
      /* Event group criado com sucesso. */
  }
  [...]
```

FreeRTOS versão 8.0.0

- Liberada dia 19/02/2014

- Introduziu os Event Groups.
  - Agrupa bits (Event Bits) em grupos (Event Groups), que podem ser usados como mecanismo de sincronização (notificação) entre tarefas.

Event Group

- Pode ser criado com a função xEventGroupCreate():

```
EventGroupHandle_t xEventGroup;

  /* Tenta criar o grupo */
  xEventGroup = FRTOS1_xEventGroupCreate();

  /* Grupo criado com sucesso? */
  if( xEventGroup == NULL )
  {
      /* Erro ao criar o Event group por falta de heap. */
  }
  else
  {
      /* Event group criado com sucesso. */
  }
  [...]
```

Event Group

- Nº de bits (flags) do grupo será 8 se configUSE_16_BIT_TICKS for definido em 1

- Serão 24 bits se configUSE_16_BIT_TICKS for 0

Event Group

- Espera pelo bit é definida com a função xEventGroupWaitBits ():

```
EventBits_t xEventGroupWaitBits(
                                const EventGroupHandle_t xEventGroup,
                                const EventBits_t uxBitsToWaitFor,
                                const BaseType_t xClearOnExit,
                                const BaseType_t xWaitForAllBits,
                                TickType_t xTicksToWait );
```

Event Group

- A tarefa fica no estado *blocked* esperando por uma mudança em um dos bits com a função xEventGroupWaitBits():

```
#define BIT_2 (1 << 2)

void task1(void *pvParameters)
{
    xEventGroup = (EventGroupHandle_t) pvParameters;
    EventBits_t uxBits;

    for (;;) {
        /* Aguarda o bit ser setado */
        uxBits = FRTOS1_xEventGroupWaitBits(xEventGroup, BIT_2, pdTRUE,
                                            pdFALSE, portMAX_DELAY);

        /* bit setado, faça algo! */
    }
}
```

Exemplo de xEventGroupWaitBits()

```
#define BIT_0( 1 << 0 )
#define BIT_4( 1 << 4 )

void aFunction( EventGroupHandle_t xEventGroup )
{
EventBits_t uxBits;
const TickType_t xTicksToWait = 100 / portTICK_PERIOD_MS;

    /* Wait 100ms max for either bit 0 or bit 4 to be set. Clear bits before exiting */
    uxBits = FRTOS1_xEventGroupWaitBits(
                xEventGroup,   /* The event group being tested. */
                BIT_0 | BIT_4, /* The bits within the event group to wait for. */
                pdTRUE,        /* BIT_0 and _4 should be cleared before returning. */
                pdFALSE,       /* Don't wait for both bits, either bit will do. */
                xTicksToWait );/* Wait a maximum of 100ms for either bit to be set. */
```

Cont. do exemplo:

```
    if( ( uxBits & ( BIT_0 | BIT_4 ) ) == ( BIT_0 | BIT_4 ) )
    {
        /* xEventGroupWaitBits() returned because both bits were set. */
    }
    else if( ( uxBits & BIT_0 ) != 0 )
    {
        /* xEventGroupWaitBits() returned because just BIT_0 was set. */
    }
    else if( ( uxBits & BIT_4 ) != 0 )
    {
        /* xEventGroupWaitBits() returned because just BIT_4 was set. */
    }
    else
    {
        /* xEventGroupWaitBits() returned because xTicksToWait ticks passed
        without either BIT_0 or BIT_4 becoming set. */
    }
}
```

Event Group

- Outra tarefa pode setar um bit com a função xEventGroupSetBits():

```
void task2(void *pvParameters)
{
    xEventGroup = (EventGroupHandle_t) pvParameters;
    EventBits_t uxBits;

    for (;;) {

        /* Espera por um pacote de dados da rede */
        rx_package();

        /* Trata o pacote */
        handle_package();

        /* Notifica que recebeu pacote */
        uxBits = FRTOS1_xEventGroupSetBits(xEventGroup, BIT_2);
    }
}
```

Diferença entre Event Groups e Queue Sets

- Várias tarefas podem ser “acordadas” com um único bit de um Event Group
- Apenas uma tarefa será “acordada” quando o evento acontecer
- Event Group é mais leve

Cortex M e o FreeRTOS

- http://www.freertos.org/RTOS-Cortex-M3-M4.html

Arquivo de config.

- http://www.freertos.org/a00110.html

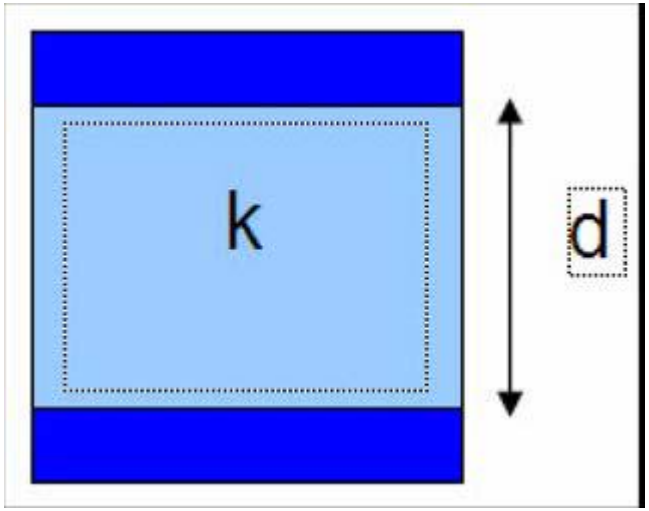
Evolução do FreeRTOS:

- http://www.freertos.org/History.txt

Touch Sensing Interface

Capacitância numa PCB

- $C = (k\varepsilon_0 A) / d$



- k é a constante dielétrica do material que separa as placas condutoras
- ε_0 é a permissividade do espaço livre (8.85 × 1012 F/m)

Capacitância numa PCB

- Considerações:
  - O objeto a ser detectado (o dedo) forma uma das placas do capacitor, enquanto a outra placa é formada pelo eletrodo na PCB.
  - Quanto maior o objeto a ser detectado, maior será a placa do capacitor correspondente e maior será a mudança na capacitância
  - Eletrodos maiores têm maior sensibilidade

Capacitância numa PCB

- Considerações:
  - Espessura do dielétrico tem um impacto direto na sensibilidade. Quando mais próximo estiver o dedo do eletrodo, maior a mudança na capacitância
  - Capacitâncias parasitas no circuito do eletrodo se somam e podem dificultar a detecção da aproximação do objeto

Ver documentação

- AN3863: Designing Touch Sensing Electrodes
  - Electrical Considerations and Recommended Layout Patterns

- KLQRUG: Kinetis L Peripheral Module – Quick Reference (A Compilation of Demonstration Software for Kinetis L Series Modules)
  - Chapter 10 – Touch Sense Input (TSI) Module

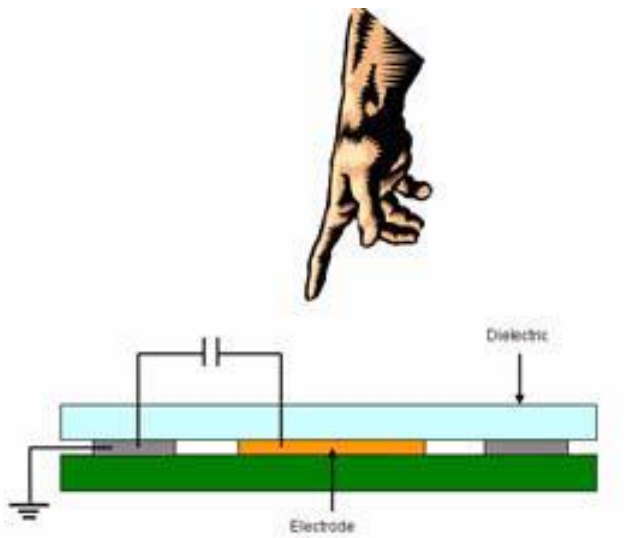
Sensor de toque

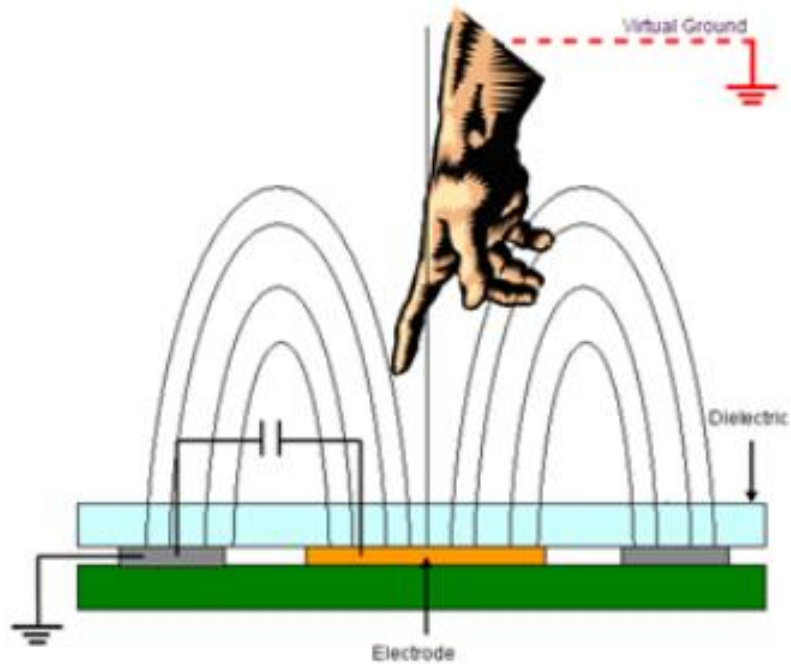
- Solução de sensor de toque inclui uma superfície plana coberta com um isolante

- O circuito do sensor processa os sinais para identificar se ocorreu um toque

Sensor de toque

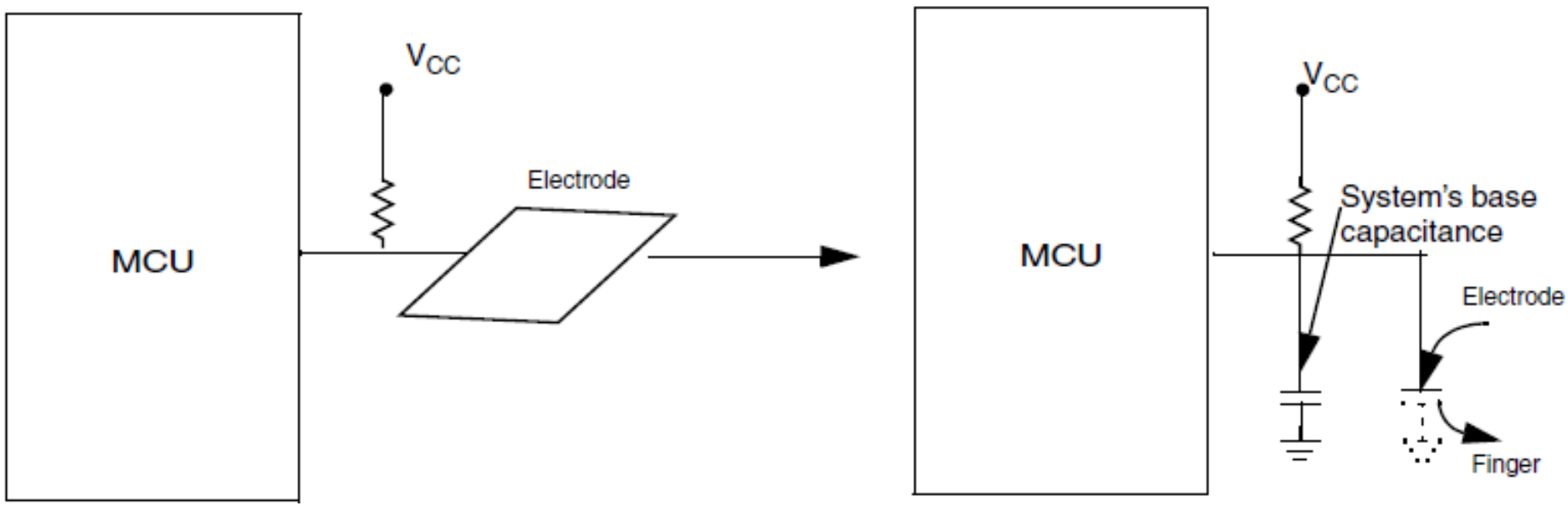
- Quando o sistema mede as mudanças de capacitância no circuito para identificar o toque, o sistema é chamado de **sensor capacitivo de toque**





Circuitos de detecção de toque

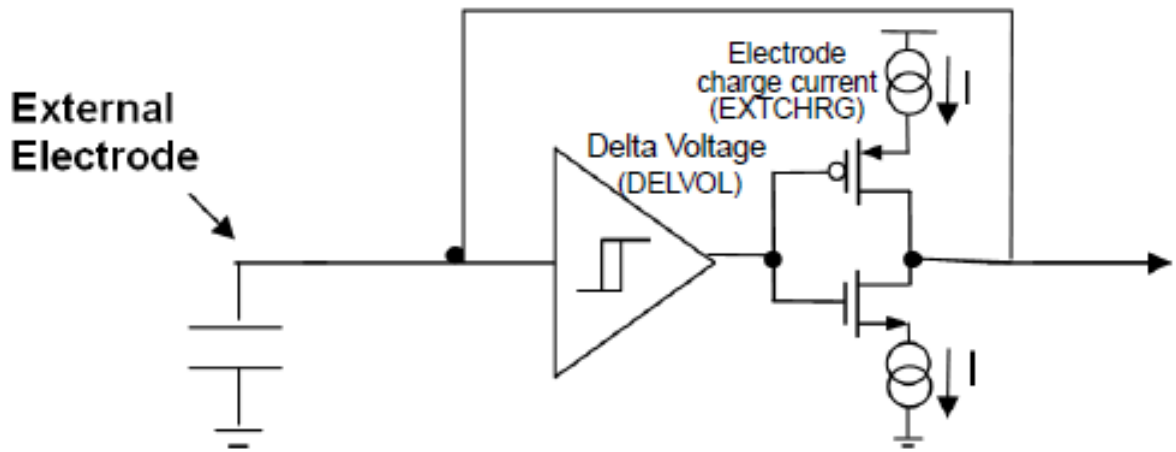
- Método RC – mede o tempo de carga do capacitor através de um resitor de alto valor 1MΩ
  - Método simples, mas não é tão robusto



Touch Sensing Interface

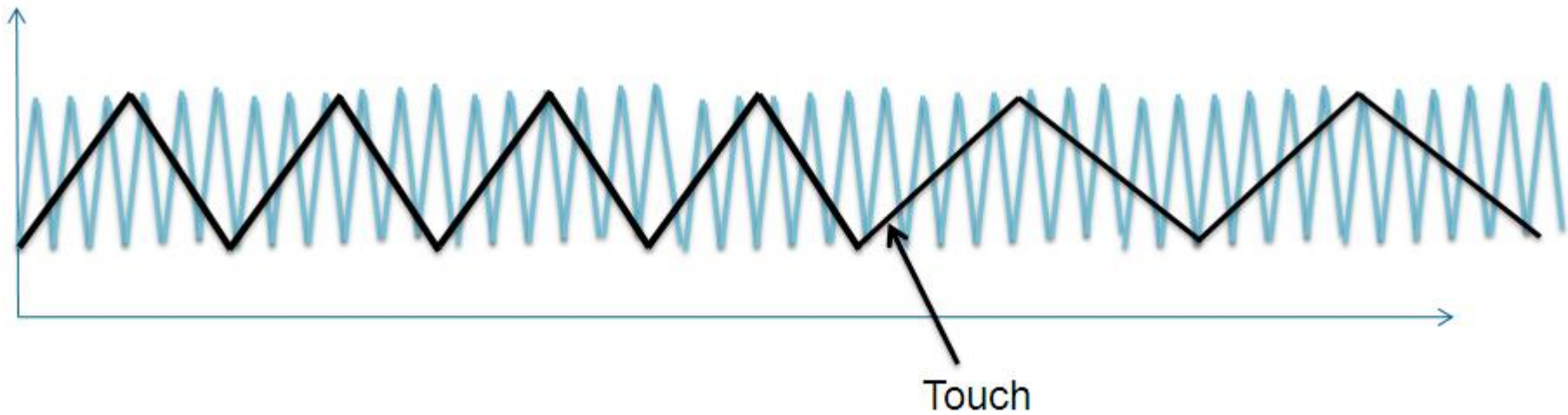
- Método da carga e descarga
  - Cria uma onda triangular, que apresenta um determinado valor pico a pico ou delta

$$F_{elec} = \frac{I}{2 * C_{elec} * V}$$



Touch Sensing Interface

- Utiliza um oscilador interno como referência
  - Oscilador interno com capacitor fixo, para acompanhar os mesmos desvios de temperatura e tensão



Touch Sensing Interface

- Recursos
  - Suporta até 16 eletrodos externos
  - Detecção das alterações de capacitância dos eletrodos com limiares superior e inferior programáveis
  - Gatilho de varredura configurável por hardware ou software
  - Detecção automática da capacitância do eletrodo através de todos os modos de operação
  - Capacidade de acordar o MCU dos modos low power

Touch Sensing Interface

- Características
  - Não precisa de componentes externos
  - Arquitetura de 1 pino por eletrodo
  - Imune às variações de temperatura de voltagem
  - Número de varreduras pode ser ajustado para tempo de resposta mais rápido ou maior resolução, até 4096 ciclos

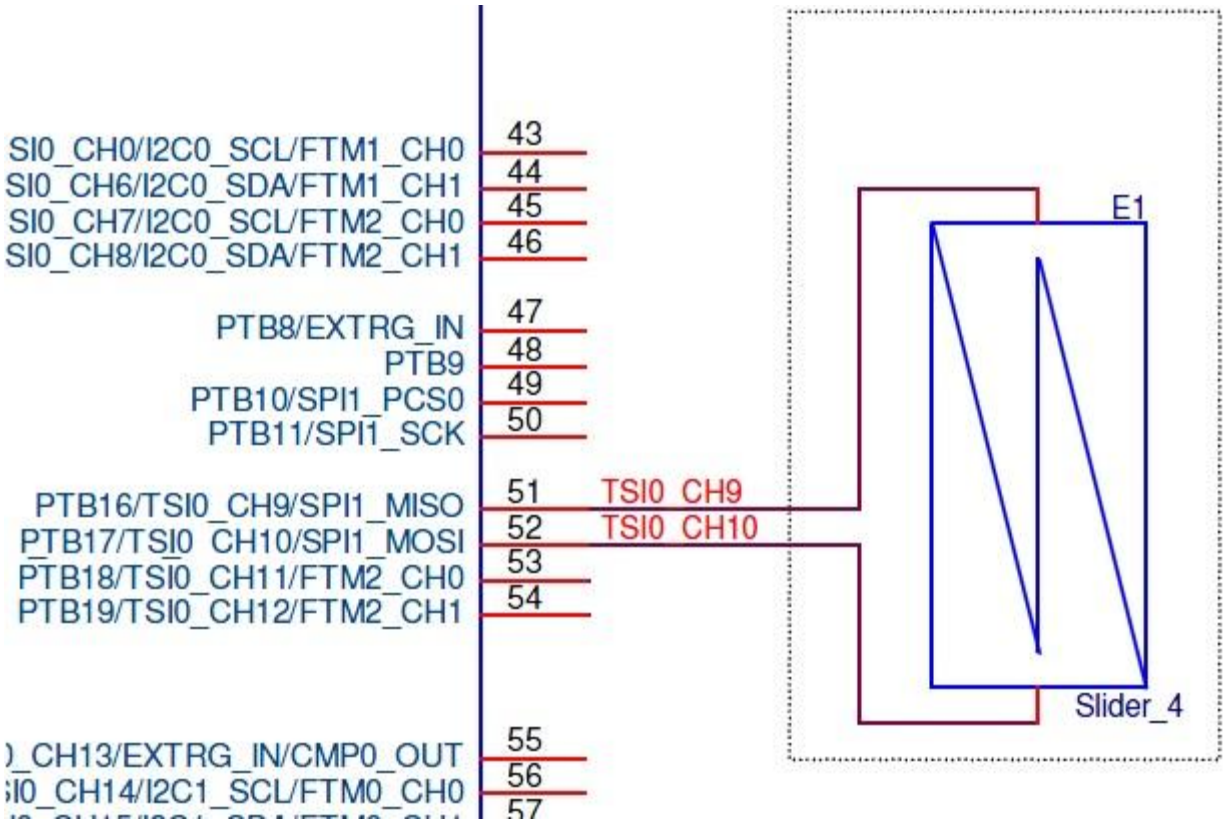
Touch Sensing Software

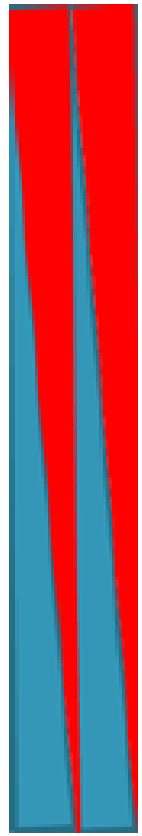
Touch-Sensing Software V. 3.1

- Suporta Kinetis K e L, S08, ColdFire V1 e Coldfire+
- Aplicações de example para placas Tower, KwikStik e Freedom
- Integração com MQX
- Integração com Processor Expert
- Advanced Filtering and Integrating Detection (AFID)
- TSI Noise mode
- Tolerante a água
- Eletrodos de proximidade e de blindagem
- Decodificador analógico

FRDM-KL25Z com TSS

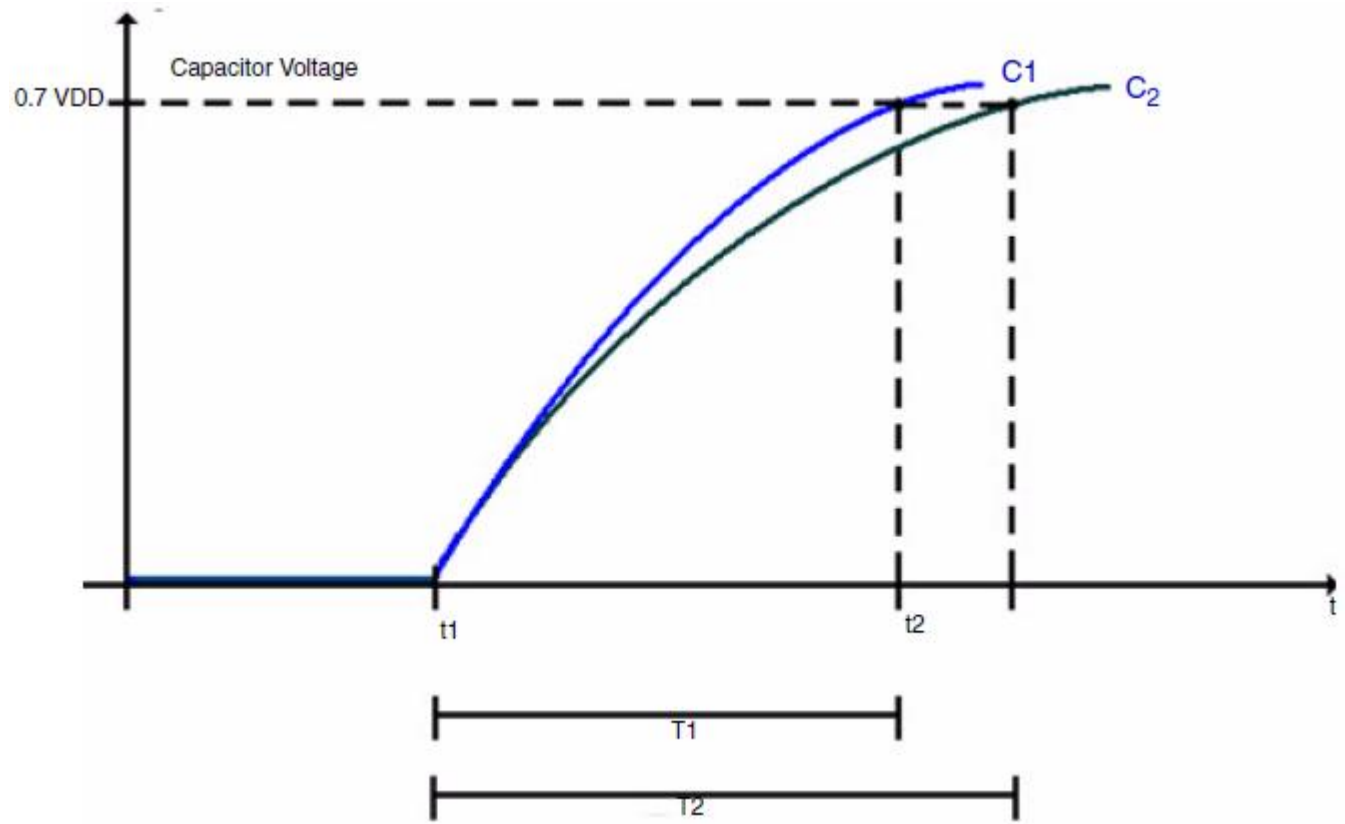
– Analog Slider Electrode



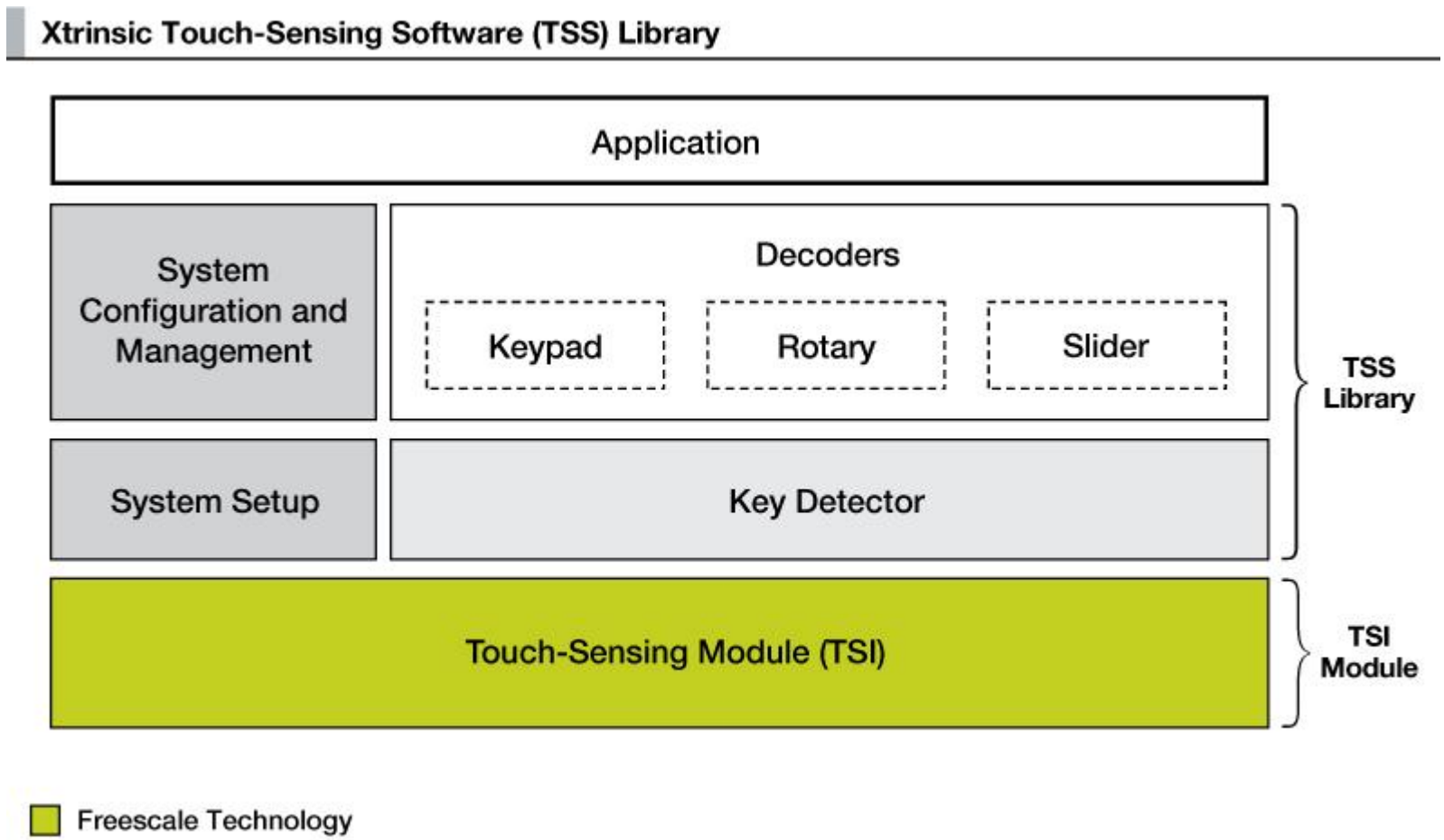


FRDM-KL25Z com TSS

- AN4637 mostra como usar o método GPIO



TSS com a TSI


Xtrinsic Touch-Sensing Software (TSS) Library
Application
System Configuration and Management
Decoders
Keypad
Rotary
Slider
System Setup
Key Detector
TSS Library
Touch-Sensing Module (TSI)
TSI Module
Freescale Technology

www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TSS&fsrch=1

TSS: Xtrinsic Touch-Sensing Software

Overview | Documentation | Downloads | Training & Support

Application Notes

Xtrinsic touch-sensing software (TSS) Suite, is an innovative touch-sensing software library offers differentiated features such as water tolerance, noise detection and touch detection algorithm for reduced false touches under electrical noise.

Robustness and flexibility adds value to targeted Freescale silicon. The TSS 3.1 free software library supports a unified API for ARM®Cortex™-M0+, ARM® Cortex™ -M4, ColdFire+ and HCS08 MCU platforms and different capacitance measurement algorithms.

Features

- Support for Kinetis K, Kinetis L, S08, ColdFire V1, and Coldfire+
- Example applications for Tower, KwikStik and Freedom boards
- MQX integration
- Processor Expert integration
- Advanced Filtering and Integrating Detection (AFID)
- TSI Noise mode
- Water tolerant
- Proximity and shielding electrode(s)
- Analog decoder

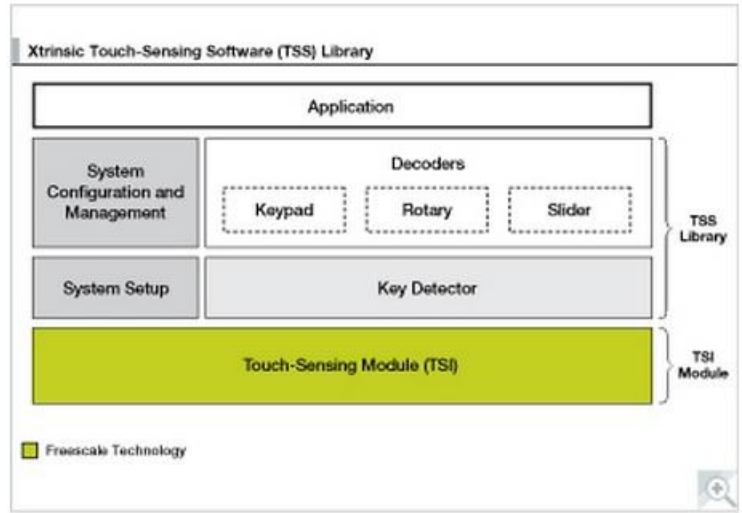
Related Software and Tools

TWRPI-ROTARY: Capacitive Touch Rotary Plug-In for the Tower System
FRDM-KE02Z40M: Freescale Freedom Development Platform for Kinetis KE02 MCUs
FRDM-KE04Z: Freescale Freedom Development Platform for the Kinetis KE04 MCUs

More

Benefits

- Touch added to one-chip solution to reduce overall system cost and size
- Improved touch detection algorithm for reduced false touches under electrical noise
- Enables over 1,000 Freescale 8-bit and 32-bit MCUs as touch



Supported Devices

- MCF51JM: Flexis 32-bit V1 ColdFire USB Microcontroller
- ColdFire+ MCUs
- Touch Sensors
- S08P: 8-bit 5V EEPROM with TSI MCUs
- K1x Baseline MCUs
- K2x USB MCUs
- K30_100: Kinetis K30 Segment LCD 100 MHz MCUs
- K4x USB and Segment LCD MCUs
- K5x Measurement MCUs
- K6x Ethernet Crypto MCUs
- K70_120: Kinetis K70 Graphic LCD 120/150 MHz MCUs
- Kinetis L Series MCUs

Featured Documentation

AN3863: AN3863, Designing Touch Sensing Electrodes - Application Notes

Download Now

Download TSS 3.1
Xtrinsic touch-sensin software (TSS) Suite

Featured Training & Events

On-Demand Training
Kinetis L Series Tutorial - Touch-Sensing with Processor Expert
Convert Your Freescale 8-bit MCU into a Smart Touch Sensor

Featured Product

CRTouch
Enables resistive touch screens to handle basic gesture recognition

Featured Partners

Pounce Consulting

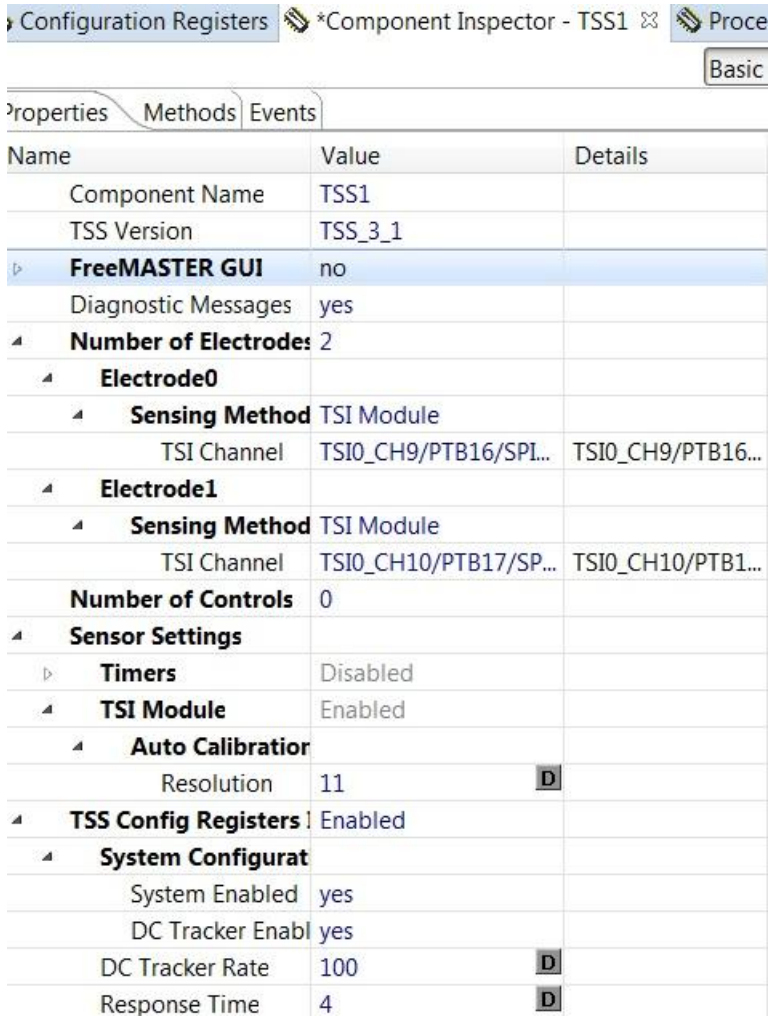
Featured Video

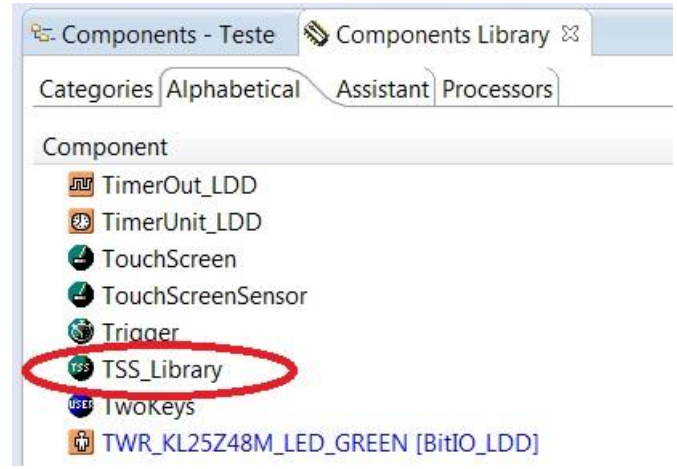
MCU-Based Capacitive Touch Sensing Demonstrations
(03:18 min)

Feedback

FRDM-KL25Z com TSS e TSI

- AN4330 mostra como usar o método TSI no TSS





Trabalho

- Implementar na FRDM-KL25Z uma aplicação no FreeRTOS com TSS no modo TSI