TS2000 Haptics TS2000 V 1.30

Copyright © 2010-2012 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory		Pins (per External IO)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C20066A, CY8C20336H, CY8C20346H, CY8C20446H, CY8C22345H						
CY8C22x45H	1 or 2	0	0	1465[2]	38-175[3]	2
CY8C20xx6H	1 or 2[1]	0	0	1592[2]	43-180[3]	2

Note

1. Timer resource usage, for CY8C20xx6H devices only.
2. The presented API memory value is for worst case. Used flash depends on TS2000 configuration selected and the type of compiler used.
3. Consumed RAM significantly depends on how many and what effects were enabled in the TS2000 parameters.

Features and Overview

- Haptics effects based on industry proven Immersion TouchSense 2000 Haptics Effect Library
- Selection of up to 14 different effects
- Selection of two actuator models
- Improved user accuracy of CapSense® buttons due to tactile feedback
- Up to 3.3-V operation
- Simple API to play haptic effects

The TS2000 User Module enables haptic feedback effects based on Immersion TouchSense 2000 technology when added to a project. Haptics is a tactile sensation effect that lets the equipment user know that a touch event has been detected. Input accuracy and user satisfaction with the equipment is improved with haptics.

Figure 1. TS2000 Haptics Block Diagram. One Digital Block Configuration, Interrupt and Polled Update Method

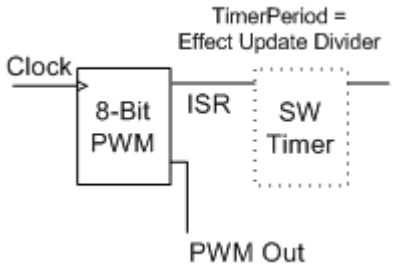


Figure 2. TS2000 Haptics Block Diagram: Two Digital Blocks Configuration, Interrupt and Polled Update Method

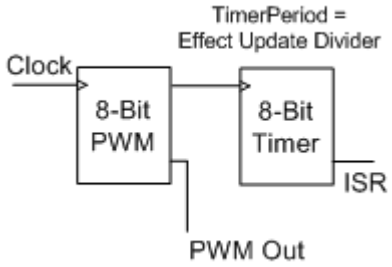


Figure 3. TS2000 Haptics Block Diagram. One Digital Block Configuration, Direct Update Method Option

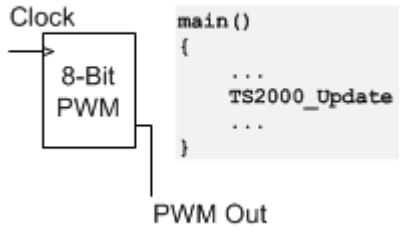


Figure 4. TS2000 Haptics Block Diagram: CY8C20xx6H Configuration, Interrupt and Polled Update Method

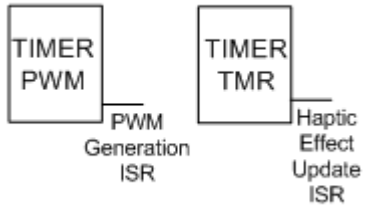
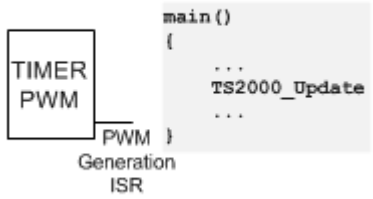


Figure 5. TS2000 Haptics Block Diagram: CY8C20xx6H Configuration, Direct Update Method Option



Quick Start Guide

1. Select and place the user modules that require dedicated pins (for example, I^2C, CSD2X, or CSD-AUTO). Assign ports and pins as necessary.
2. Place the TS2000 User Module. If the CY8C22x45H device is used, select either the “1-Block” option or the “2-Block” option. The “2-Block” option uses more chip resources, but minimizes the CPU loading of the TS2000 User Module. The CY8C20xx6H device has only one configuration: “2-Timer”.
3. Select the actuator that will be used for haptic vibrations.
4. Enter the port number and pin number of the PWM output that will be connected to in the IN-signal of the actuator drive circuit. If the CY8C22x45H device is used, route CompareOut line in the same way PWM is routed. Select the required Row_N_Output_M, connect it to GlobalOutOdd_x/GlobalOutEven_x, and connect the required pin.

   Note that using P1[0] and P1[1] is not recommended, because they are used by I^2C.
5. Enter the port number and pin number of the amplifier enable signal that will be connected to the SHDN# signal of the actuator drive circuit.
6. Select the effects that are to be used in the project. Disable effects to reduce the flash usage of the project. Disabled effects cannot be called in firmware.
7. Generate the application and switch to the Application Editor.
8. Adapt the sample code to call the required effects.
9. Program the PSoC device on the target board with the hex generated by PSoC Designer.

Functional Description

The basic operation of the TS2000 User Module is to give the actuator drive circuit a specific PWM signal that is periodically updated. By giving different sequences of PWM values, the actuator produces different effects.

The haptic system consists of physical, electrical, and firmware components:

Physical

An actuator or Eccentric Rotating Motor (ERM) gives the vibrations used for tactile feedback. The actuator selection and placement depends on the weight and geometry of the device. Refer to the Immersion website on physical guidelines for the haptics design:

http://www.immersion.com/products/touchsense-tactile-feedback/2000-series/index.html

Electrical

TS2000 haptics performance is heavily characterized with the circuit as given in the Features and Overview section. Use the components shown for optimal performance.

Firmware

The firmware to produce the correct effects has been simplified with the TS2000 User Module. To produce the required effects, call the TS2000_1_PlayEffect(BYTE *EffectData) API.

Recommended Reading

The typical use for a haptic design is to give feedback for nonmechanical CapSense sensors. The following documents are recommended reading before implementing a CapSense design:

- *CY8C20x66 Series PSoC Mixed Signal Array Technical Reference Manual, sections:*
  - *CapSense System*
- *Charting Tool to Debug CapSense Applications* – AN2397
- Getting started with CapSense
- CY8C20xx6A/A/H Design Guide

External Component Selection Guidelines

The two supported actuators are the Sanyo NRS-2574i and the Jinlong Z6DL2A017000B. For device masses of 150g or less, use the NRS-2574i. Otherwise, use the Z6DL2A017000B actuator. The actuator drive circuit (TPA6205A) must use a 3.3V supply voltage.

CPU Loading

For optimal CPU load and resource usage, five configurations of the TS2000 User Module are available. Three of these configurations use standard PSoC digital blocks and are used for CY8C22x45H devices. The other configurations use Timer resources, as presented for CY8C20xx6H.

The TS2000 User Module for CY8C22x45H devices has three options. Two-block configuration provides fully-HW PWM generation and gives the 5 ms timer (for haptic-effects generation) only HW. This configuration has the lowest CPU load.

One-block configuration also generates the PWM with a HW block, but the 5 ms timer is realized in FW. As a result, the CPU load is a bit higher.

The third configuration does not provide automatic 5 mS timer. You must provide all timing checking and call the TS2000_bUpdate every 5 mS.

The same functionality is for second option of selection wizard in CY8C20xx6H devices.

In CY8C20xx6H devices both the PWM and the 5-ms timer are generated in FW. The two 16-bit Timer resources are used to generate these signals. Timer_PWM ISR gives the PWM signal generation with resulting output frequency at about 23 kHz. The Timer_TMR ISR is used to generate 5-ms intervals for haptic-effects generation.

The duration of the Timer_PWM ISR is approximately 90 clock cycles and is depicted in TS2000_1.inc as “TS2000_1_ISR_DUR”. It is important to take into account the IMO clock speed and duration of the ISR when designing other ISRs in the system. For example, to prevent another ISR from being missed, ensure that the additional ISR is not triggered twice in the duration of the TS2000 ISR. It is strongly recommended to use CPU clock equal to IMO (24 MHz).

Note that when a haptic effect is being played, the CPU is not blocked. In other words, the project code continues to execute while the effect is being played. Moreover, the periodic TS2000 ISR only occurs during the duration of the haptic effect. When a haptic effect is not playing, there is no CPU loading caused by the TS2000 User Module.

DC and AC Electrical Characteristics

Table 1. Power Supply Requirements

Parameter	Min	Typical	Max	Unit	Test Conditions and Comments
Actuator Circuit V_{DD}[1]	-	3.30	3.47	V	3.3 V + 5%

Note

1. According to the Immersion recommended operating range.

Placement

This user module can be placed in three configurations:

CY8C20xx6H devices have two configuration, which uses two PSoC resources: Timer0 and Timer2. One of these resources generates the PWM signal, and the Haptics Update ISR is generated by the other. For direct update method there is no 5-ms Timer presented.

CY8C22x45H devices have three configurations available: two with one-block and one with two-block. The two-block configuration uses the second digital block to lower the load of the CPU, but consumes extra digital blocks. The one-block configuration saves the digital block, but uses a software timer resulting in a higher CPU load.

When placing TS2000 User Module, the wizard enables you to select one of the three or two possible configurations.

To change the user module configuration, call the wizard again and select the necessary configuration. To access the Selection Wizard, right-click on any digital block of the TS2000 User Module in the Workspace Explorer, and select “Selection Options” as shown in Figure 6.

Figure 6. Accessing the Wizard

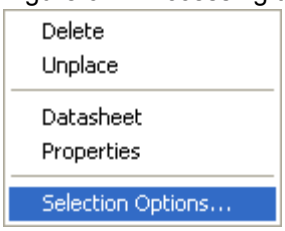


Figure 7. TS2000 User Module Configurations Selection, CY8C22x45H Devices

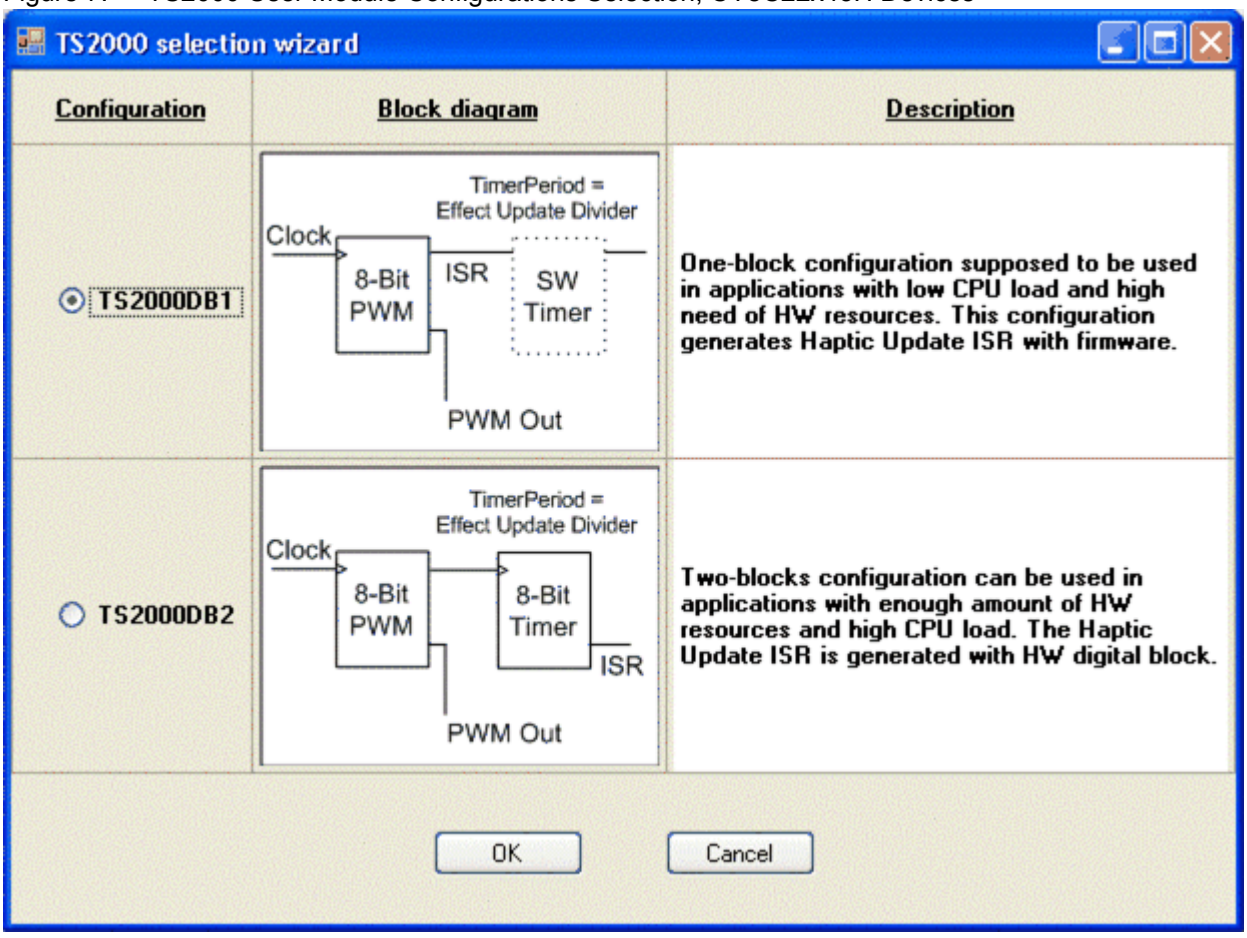
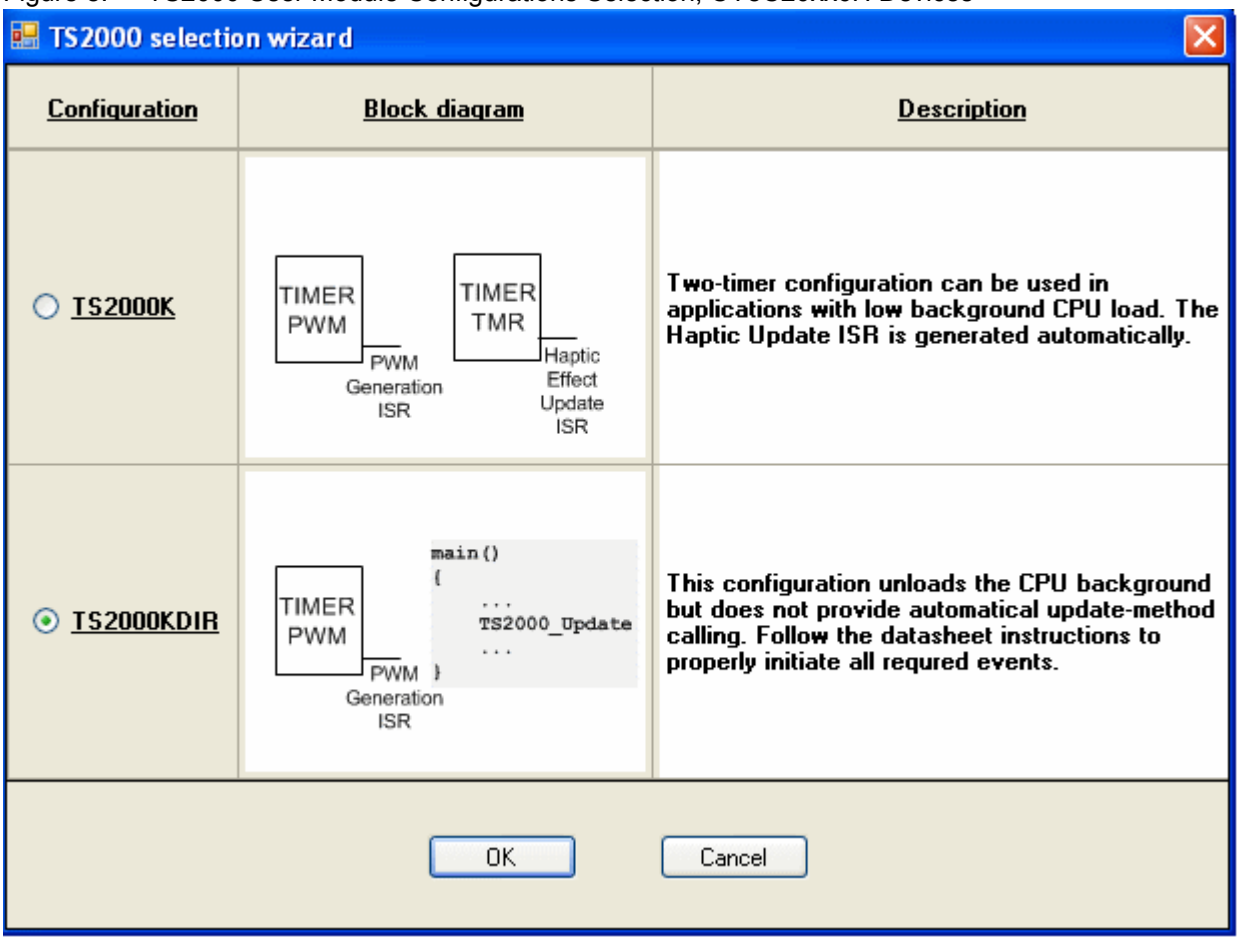


Figure 8. TS2000 User Module Configurations Selection, CY8C20xx6H Devices



Parameters and Resources

PWM InputClock

The parameter should be selected such that the PWM output frequency is higher than 22 kHz.

The input frequency should be between 5.6 MHz and 13 MHz for proper TS2000 module operation.

To check if the selected FClock is in proper range, use the following equation:

FPWM_Output = FClock / 255

If FPWM_Output is higher then 22 kHz, then the parameter is selected properly.

EffectUpdateDivider

The parameter should be selected such that the Haptic timer is generating 5-ms intervals with 10% error (that is, timer update rate of 200 Hz).

The EffectUpdateDivider is dependent on the FClock parameter. These two parameters must be configured at the same time. The following equation shows the relationship between these two values:

EffectUpdateDIvider = (FClock/(200Hz * 255)) - 1

Taking to account the valid range of PWM InputClock parameter, the EffectUpdateDivider should have a range of [109..255].

Example:

Clock selected as VC1 and VC1 = 4 and VC1 source is IMO 24 MHz.

As a result:

FClock = 6 MHz

Then:

EffectUpdateDivider = (6 MHz/(200 Hz*255)) - 1 = 117

EffectUpdateMethod

The effect update method specifies the way in which the Update function will appear. The API call must be repeated with 5-ms period.

After you select “Interrupt”, disregard the 5-ms interrupt generation for haptic update event. The only APIs you should use are TS2000_Start and TS2000_PlayEffect/TS2000_bPlayEffect.

When you select “Polled” then the user module will not call Update function for haptics with interrupt. You have to call TS2000_bUpdate API from main (or other functions) with an interval no longer than 5 ms.

Example 3 in the Sample Code sections shows how the second option should be realized in code.

Actuator

Select one of the two possible actuators. For devices less than 150 g, use the Sanyo NRS-2574i. The Jinlong Z6DL2A017000 is a stronger actuator and should be used for heavier devices.

For the schematic of electrical connections to PSoC, refer to the typical schematic in the actuator datasheet.

Compare Out

This parameter is used to route the PWM signal through the internal PSoC interconnect to an output pin, in a way similar to the simple PWM User Module.

PWM_Port, PWM_Pin

These parameters set the port and pin to connect to the “–IN” signal of the actuator drive circuit. Choose the port first and then select from the available pins.

AmplifierEnable_Port, AmplifierEnable_Pin

These parameters set the port and pin to connect to the SHDN# signal of the actuator drive circuit. Choose the port first and then select from the available pins.

Effects

Up to 14 different effects are available with the TS2000 User Module. The choices for each effect are “enable” and “disable”. Enabling an effect allows it to be called from the TS2000_1_PlayEffect() API. Disabling an effect reduces the flash usage of the project.

The 14 effects are:

- Strong Click
- Strong Click 60%
- Strong Click 30%
- Sharp Click
- Sharp Click 60%

- Sharp Click 30%
- Soft Bump
- Soft Bump 60%
- Soft Bump 30%
- Double Click
- Double Click 60%
- Triple Click
- Soft Fuzz
- Strong Buzz

The percentages associated with some effects indicate the intensity of that effect. For example, a Sharp Click has a stronger intensity than a Sharp Click 60%, which has a stronger intensity than a Sharp Click 30%.

Application Programming Interface

The Application Programming Interface (API) functions are given as part of the user module to allow you to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the include files.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the TS2000_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable and constant symbol. In the following descriptions the instance name has been shortened to TS2000 for simplicity.

Note

** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must also ensure their code observes this policy. Though some user module API functions may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

Entry Points are supplied to initialize the TS2000 User Module, start it and play the haptic effects. In all cases the instance name of the module replaces the TS2000 prefix shown in the following entry points. Failure to use the correct instance name is a common cause of syntax errors.

TS2000_Start

Description:

Initializes registers and enables the resources used by the TS2000 User Module. This function must be called before calling TS2000_PlayEffect().

C Prototype:

```
void  TS2000_Start(void)
```

Assembly:

```
lcall  _TS2000_Start
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

TS2000_PlayEffect

Description:

Plays one of the 14 haptic effects. The effect of interest must be enabled in the Chip View of PSoC Designer. Pass the name of the effect to play the required effect. For example, to play the StrongClick60 effect use TS2000_PlayEffect(TS2000_StrongClick60).

C Prototype:

```
void  TS2000_PlayEffect(BYTE *EffectData)
```

Assembly:

```
mov   A, > EffectData
push  A
mov   A, < EffectData
push  A
lcall  _TS2000_PlayEffect
add SP, -2
```

Parameters:

```
A  =>  Effect Data
```

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

TS2000_bPlayEffect

Description:

Plays one of the 14 haptic effects and returns the state of the user module. The effect of interest must be enabled in the Chip View of PSoC Designer. Pass the name of the effect to play the required effect.

For example, to play the StrongClick60 effect use: bStatus = TS2000_bPlayEffect(TS2000_StrongClick60)

C Prototype:

```
BYTE  TS2000_bPlayEffect(BYTE *EffectData)
```

Assembly:

```
mov   A, > EffectData
push  A
mov   A, < EffectData
push  A
lcall  _TS2000_bPlayEffect
add SP, -2
```

Parameters:

```
A  =>  Effect Data
```

Return Value:

1 - Requested effect not played. Current effect playing.

0 - Success

Side Effects:

See Note ** at the beginning of the API section.

TS2000_bUpdate

Description:

Provides haptic Update event processing. This API has to be called from main() routine or from other function with an interval not longer than 5 ms.

This call is required if the EffectUpdateMethod parameter is set to “Polled” or if direct update option is selected in the selection wizard.

C Prototype:

```
BYTE  TS2000_bUpdate(void)
```

Assembly:

```
lcall  _TS2000_bUpdate
```

Parameters:

None

Return Value:

1 - Effect Playing

0 - Effect Stopped

Side Effects:

See Note ** at the beginning of the API section.

Sample Firmware Source Code

Example 1

This code starts the user modules and shows an example of how to add haptics to a CapSense design based on CSD (CY8C20xx6H) for interrupt haptic update method.

Note The TS2000 User Module should be renamed from default TS2000_1 to TS2000; this also concerns the other user modules used in this example. The CSD User Module has to provide four buttons, and the DoubleClick60, SharpClick, SoftFuzz, and TripleClick effects must be enabled in the TS2000 User Module. Other settings may be left by default, except the settings corrected according to the DRC or Wizard warnings.

```
//-------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules

void main(void)
{
    BYTE bPress;

    M8C_EnableGInt; // Enable global interrupts

    CSD_Start();
    CSD_InitializeBaselines(); // Scan all sensors first time, init baseline
    CSD_SetDefaultFingerThresholds();

    TS2000_Start();

    while (1) // Loop forever
    {
        CSD_ScanAllSensors();
        CSD_UpdateAllBaselines();

        if(CSD_bIsAnySensorActive()) // If sensor press detected
        {
            if(CSD_bIsSensorActive(0) & bPress)    // if Row pressed
            {
                bPress = 0; // reset trigger

                if(CSD_bIsSensorActive(1))      // if button_1 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_DoubleClick60);
                }
                else if(CSD_bIsSensorActive(2)) // if button_2 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_SharpClick);
                }
                else if(CSD_bIsSensorActive(3)) // if button_3 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_SoftFuzz);
                }
                else if(CSD_bIsSensorActive(4)) // if button_4 of Row pressed
                {
```

```
                    TS2000_PlayEffect(TS2000_TripleClick);
                }
            }
        }
        else
        {
            bPress = 1; // set trigger if buttons were released
        }
    }
}
```

The equivalent code written in Assembly is:

```
;-----------------------------------------------------------------------------
; Assembly main line
;-----------------------------------------------------------------------------
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all user modules

export _main

AREA  bss (RAM,REL)

bPress::      BLK 1  ;

AREA text(ROM,REL,CON)

_main:

   M8C_EnableGInt ; Enable Global Interrupts

; CSD UM initialization
   lcall CSD_Start             ; Start CSD UM
   lcall CSD_InitializeBaselines     ; Base line Initialization
   lcall CSD_SetDefaultFingerThresholds ;Set Finger Threshholds

   lcall _TS2000_Start     ; Start Haptics UM

.main_loop:
    lcall CSD_ScanAllSensors         ; Scan sensors
    lcall CSD_UpdateAllBaselines     ; Update Base lines

    lcall CSD_bIsAnySensorActive ; Sensor press detection
    cmp   A, 00h                ; If Sensor press not detected
    jz    .SetTrigger           ; jump out

    mov   A, 00h                ; test if Row pressed
    lcall CSD_bIsSensorActive

    RAM_PROLOGUE RAM_USE_CLASS_4
   RAM_SETPAGE_CUR >bPress
    and   A, [bPress]           ; if Row pressed and trigger released
    RAM_EPILOGUE RAM_USE_CLASS_4

    jz    .main_loop
```

```
    RAM_PROLOGUE RAM_USE_CLASS_4
    RAM_SETPAGE_CUR >bPress
    mov   [bPress], 00h         ; reset trigger
    RAM_EPILOGUE RAM_USE_CLASS_4

    mov   A, 01h                ; prepare to scan button_1 of Row
    lcall CSD_bIsSensorActive
    cmp   A, 00h                ; if button_1 of Row pressed
    jz    .TryButton2
    mov   A, > _TS2000_DoubleClick60 ; Get MSB part of  TS2000_DoubleClick60 address
    push  A
    mov   A, < _TS2000_DoubleClick60  ; Get LSB part of  TS2000_DoubleClick60 address
    push  A
    lcall _TS2000_PlayEffect ; initiate effect playing
    add SP, -2
    jmp   .main_loop

.TryButton2:
    mov   A, 02h                ; prepare to scan button_2 of Row
    lcall CSD_bIsSensorActive
    cmp   A, 00h                ; if button_2 of Row pressed
    jz    .TryButton3
    mov   A, > _TS2000_SharpClick ; Get MSB part of  TS2000_SharpClick address
    push A
    mov   A, < _TS2000_SharpClick ; Get LSB part of  TS2000_SharpClick address
    push A
    lcall _TS2000_PlayEffect ; initiate effect playing
    add SP, -2
    jmp   .main_loop

.TryButton3:
    mov   A, 03h                ; prepare to scan button_1 of Row
    lcall CSD_bIsSensorActive
    cmp   A, 00h                ; if button_3 of Row pressed
    jz    .TryButton4
    mov   A, > _TS2000_SoftFuzz ; Get MSB part of  TS2000_SoftFuzz address
    push A
    mov   A, < _TS2000_SoftFuzz ; Get LSB part of  TS2000_SoftFuzz address
    push A
    lcall _TS2000_PlayEffect     ; initiate effect playing
    add SP, -2
    jmp   .main_loop

.TryButton4:
    mov   A, 04h                ; prepare to scan button_1 of Row
    lcall CSD_bIsSensorActive
    cmp   A, 00h                ; if button_4 of Row pressed
    jz    .main_loop
    mov   A, > _TS2000_TripleClick ; Get MSB part of  TS2000_TripleClick address
    push A
    mov   A, < _TS2000_TripleClick ; Get LSB part of  TS2000_TripleClick address
    push A
    lcall _TS2000_PlayEffect ; initiate effect playing
    add SP, -2
```

```
    jmp   .main_loop

.SetTrigger:
    RAM_PROLOGUE RAM_USE_CLASS_4
    RAM_SETPAGE_CUR >bPress
    mov [bPress], 01h           ; set trigger if buttons were released
    RAM_EPILOGUE RAM_USE_CLASS_4

jmp .main_loop

.terminate:
jmp .terminate
```

Note This assembly code is compiler-dependent and may not build without errors on all C compilers.

Example 2

This code starts the user modules and shows an example of how to add haptics to a CapSense design based on CSD2X (CY8C22x45H) for polled haptic update method.

Note The TS2000 User Module should be renamed from default TS2000_1 to TS2000; this also concerns the other user modules used in this example. The CSD2X User Module has to provide four buttons, and the DoubleClick60, SharpClick, SoftFuzz, and TripleClick effects must be enabled in the TS2000 User Module. Other settings may be left by default, except the settings corrected according to the DRC or Wizard warnings.

```
//-------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules

void main(void)
{
    BYTE bPress;

    M8C_EnableGInt; // Enable global interrupts

    CSD2X_Start();
    CSD2X_InitializeBaselines(); // Scan all sensors first time, init baseline
    CSD2X_SetDefaultFingerThresholds();

    TS2000_Start();

    while (1) // Loop forever
    {
        CSD2X_ScanAllSensors();
        CSD2X_UpdateAllBaselines();

        if(CSD2X_bIsAnySensorActive()) // If sensor press detected
        {
            if(CSD2X_bIsSensorActive(0) & bPress)    // if Row pressed
            {
                bPress = 0; // reset trigger
```

```
                if(CSD2X_bIsSensorActive(1))      // if button_1 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_DoubleClick60);
                }
                else if(CSD2X_bIsSensorActive(2)) // if button_2 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_SharpClick);
                }
                else if(CSD2X_bIsSensorActive(3)) // if button_3 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_SoftFuzz);
                }
                else if(CSD2X_bIsSensorActive(4)) // if button_4 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_TripleClick);
                }
            }
        }
        else
        {
            bPress = 1; // set trigger if buttons were released
        }
       TS2000_bUpdate();
    }
}
```

Example 3

This code starts the user modules and shows an example of how to add haptics to a CapSense design based on CSD2X (CY8C22x45H) for direct haptic update method.

Note The TS2000 User Module should be renamed from default TS2000_1 to TS2000; this also concerns the other user modules used in this example. The CSD2X User Module has to provide four buttons, and the DoubleClick60, SharpClick, SoftFuzz, and TripleClick effects have to be enabled in the TS2000 User Module, other settings may be left by default, except the settings corrected according to the DRC or Wizard warnings.

Custom settings of the Counter8 User Module parameters:

- Counter8_Clock = VC3

Custom settings of Global Resources used by Counter8 User Module:

- VC1 = 4
- VC3 = 150
- VC3 Source = VC1

```
//-------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------
#include <m8c.h>      // part specific constants and macros
#include "PSoCAPI.h" // PSoC API definitions for all user modules

void main(void)
{
    BYTE bPress;
```

```
BYTE bEffectState;
BYTE bTime;

M8C_EnableGInt; // Enable global interrupts

CSD2X_Start();
CSD2X_InitializeBaselines(); // Scan all sensors first time, init baseline
CSD2X_SetDefaultFingerThresholds();

TS2000_Start();
    // VC1 = 4
    // VC3 = 150 (VC3 Source = VC1)
    // Counter8_Clock = VC3
Counter8_Start();
Counter8_WritePeriod(255);

bTime = Counter8_bReadCounter();
bEffectState = 0; // reset at start because no effect requested

while (1) // Loop forever
{
    if(bEffectState)
    { // adjust the timeout value for proper effect playing
        if ( (BYTE)(bTime - Counter8_bReadCounter()) >= 99 )
        {
            bEffectState = TS2000_bUpdate();
            bTime = Counter8_bReadCounter();
        }
    }
    else
    {
        CSD2X_ScanAllSensors();
        CSD2X_UpdateAllBaselines();

        if(CSD2X_bIsAnySensorActive()) // If sensor press detected
        {
            if(CSD2X_bIsSensorActive(0) & bPress) // if Row pressed
            {
                bEffectState = 1;
                bPress = 0; // reset trigger
                if(CSD2X_bIsSensorActive(1)) // if button_1 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_DoubleClick60);
                }
                else if(CSD2X_bIsSensorActive(2)) // if button_2 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_SharpClick);
                }
                else if(CSD2X_bIsSensorActive(3)) // if button_3 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_SoftFuzz);
                }
                else if(CSD2X_bIsSensorActive(4)) // if button_4 of Row pressed
                {
                    TS2000_PlayEffect(TS2000_TripleClick);
```

```
                    }
                    else
                    {
                        bEffectState = 0;
                    }
                }
            }
            else
            {
                bPress = 1; // set trigger if buttons were released
            }
        }
    }
}
```

Configuration Registers

The TS2000 User Module uses one or two digital PSoC blocks. Each block is personalized and parameterized through a set of registers. The set of registers used by the user module with brief descriptions are given in this section. Symbolic names for these registers are defined in the user module instance's C and assembly language interface files (the ".h" and ".inc" files).

CY8C22x45H Registers

- PWM8
  - Function Register, Bank 1: DxCxxFN

    This register defines the settings of the Digital Basic/Communications Type 'B' Block to be the PWM8 digital block of the TS2000 User Module.
  - Output Register, Bank 1: DxCxxOU

    This register is used to control the connection of the PWM8 digital block outputs to the available row interconnect.
  - Control0 Register, Bank 0: DxCxxCR0

    This register defines the settings of the Digital Basic/Communications Type 'B' Block to be the PWM8 digital block of the TS2000 User Module.
  - Period Register, Bank 0: DxCxxDR1

    This register defines the period of PWM-signal of the PWM8 digital block of the TS2000 User Module.
  - Compare Register, Bank 0: DxCxxDR2

    This register defines the pulse width of PWM-signal of the PWM8 digital block of the TS2000 User Module.

- Input Register, Bank 0: DxCxxIN

  This register is used to select the data and clock inputs for the PWM8 digital block of the TS2000 User Module.

- Timer8
  - Function Register, Bank 1: DxCxxFN

    This register defines the settings of the Digital Basic/Communications Type 'B' Block to be the Timer8 digital block of the TS2000 User Module.
  - Output Register, Bank 1: DxCxxOU

    This register defines the settings of the Digital Basic/Communications Type 'B' Block to be the Timer8 digital block of the TS2000 User Module.
  - Control0 Register, Bank 0: DxCxxCR0

    This register defines the settings of the Digital Basic/Communications Type 'B' Block to be the Timer8 digital block of the TS2000 User Module.
  - Period Register, Bank 0: DxCxxDR1

    This register defines the period of the Timer8 digital block of TS2000 User Module.
  - Compare Register, Bank 0: DxCxxDR2

    This register defines the settings of the Digital Basic/Communications Type 'B' Block to be the Timer8 digital block of the TS2000 User Module.
  - Input Register, Bank 0: DxCxxIN

    This register is used to select the data and clock inputs for the Timer8 digital block of the TS2000 User Module.

CY8C20xx6 Registers

This block can operate in Timer and TX mode. For more details refer to the Block Resources datasheet section.

- Timer_TMR
  - Configuration Register, Bank 0: PTx_CFG

    This register defines the settings of the Timer0/Timer2 resource to be the 16-bit Timer of the TS2000 user module.
  - Data Register 0, Bank 0: PTx_DATA0

    This register defines the LSB of period of the Timer_TMR resource of the TS2000 User Module.
  - Data Register 1: PTx_DATA1

    This register defines the MSB of period of the Timer_TMR resource of the TS2000 User Module.

- TIMER_PWM
  - Configuration Register, Bank 0: PTx_CFG

    This register defines the settings of the Timer0/Timer2 resource to be the 16-bit Timer of the TS2000 User Module and to generate PWM signal in firmware.
  - Data Register 0, Bank 0: PTx_DATA0

    This register defines the LSB of period of the Timer_PWM resource of the TS2000 User Module.
  - Data Register 1, Bank 0: PTx_DATA1

    This register defines the MSB of period of the Timer_PWM resource of the TS2000 User Module.

Version History

Version	Originator	Description
1.0	DHA	Initial version.
1.1	DHA	Added DRC for the following parameters: 1. Clock 2. CompareOut 3. Actuator Improved Double Click effect.
1.20	DHA	1. Implemented new API TS2000_bPlayEffect(), and added API function definition. 2. Changed _TS2000_PWM_ISR handler to correctly process PWM Output state. 3. Improved _TS2000_PWMWrite function. 4. Updated TS2000_ISR_DUR constant according to new ISR (TS2000KINT.asm). 5. Updated resource meter. 6. Corrected incorrect parameter name ("I2C_Port" is changed to "AmplifierEnable_Port"). 7. Updated user module to optimize memory usage and execution speed.
1.30	DHA	1. Added wizard help button and file. 2. Updated TS2000_bUpdated API. 3. Updated sample code in the user module datasheet.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Credits



TouchSense® Technology Licensed from immersion Corporation. Protected by one or more of the following patents:

U.S. Patents: 4823634, 4896554, 5184319, 5185561, 5220260, 5235868. 5389865. 5414337, 5459382, B1 5459382, 5482056,5513100,5559412,5576727, 5589854, 5592401, 5623582, 5629594, 5631881, 5676157, 5891898,5701140, 5721566, 5724264, 5731804, 5734373,5739811, 5767639,5769640, 5790108, 5805140, 5821920,5825308,5828197, 5831408, 5844392, 5872438, 5880714, 5889670, 5889672, 5907487, 5929607, 5930741, 5929846, 5956484, 5959613,5999168,6015473,6020875, 6020876, 6020967, 6024576,6028593,6037927, 6042555, 6046727,6050718. 6050962, 6057828, 6059506, 6061004, 6067077, 6078308,6078676, D427635, 6088017, 6088019, 6100874, 6101530, 6104158, 6104379, 6104382, 6106301, 6110130, 6125337, 6125385, 6128006, 6131097, 6134506, 6147674, 6148280, 6154198, 6154201,6161126, 6162190, 6166723,6169540, 6184868, 6191774. 6195592, 6201533, 6211861, 6215470, 8219032, 6219033, 6232891, 6243078, 6246390, 6252579, 6252583. 6256011, 6259382, 6271828, 6271833, 6275213 B1. 6278439, 6281651, 6285351, 6288705, 6292170, 5754023, "RE37374, 6292174, 6300936, 6300937, 6300938, 6304091, 6310605, 6317116, 6323837, 6342880. 6343349, 6348911, 6353850, 6353427, 6366272, 6366273, 6374255, 6380925, 6396232, 6400352, 6411276, 6413229, 6424333, 6424356, 6429846, 6428490.6433771, 6437771, 6448977. 6469692, 6470302, 6486872. 6497672, 6563487. 6564168, 6580417, 6636161, 6636197, 6639581, 6654000, 6661403, 6680729, 6683437, 8686901. 6686911, 6693622, 6693626, 6697043, 6697044, 6697048, 6697748, 6697086, 8703550, 6704001, 6704002, 6704683. 6705871, 6707443, 6715045, 6750877, 6762745. 6781569.

Copyright © 2010-2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.