Freescale Semiconductor
Application Note

Document Number: AN3579
Rev. 0, 11/2007

Enabling an MCU for Touch Sensing with Proximity Sensor Software

by: Eduardo Muriel Hernandez
Ulises Corales
RTAC Americas

1 Introduction

This application note describes how to enable an MCU for touch sensing with a properly designed layout and a simple software module. This free software is compatible with any S08, RS08, and V1 Freescale microcontroller. The flexibility of this software and use of minimal hardware make this a solid base for simple touch sensing applications. This document gives details of hardware setup and software porting considerations. This application can be used in conjunction with the KITPROXIMITYEVM for a complete evaluation set.

Contents

1 Introduction 1
2 General Description. 2
2.1 System description 2
2.2 Proximity Application Description 4
3 Proximity Module 9
3.1 General Description. 9
3.2 Proximity Initialization Function 9
3.3 Electrode Sampling Function. 9
3.4 Update Electrode Status Function. 11
4 Hardware Setup. 15
4.1 Porting the Application to RS08, S08 and ColdFire V1 16
4.2 Configuring Electrodes and Buzzer Location 17
4.3 Timer Module. 18
5 Graphical User Interface 19
5.1 GUI Setup 19
5.2 Capacitance Bars 19
5.3 Threshold Detection 19
6 Conclusions 20
6.1 Proximity Software Reusability 20

2 General Description

2.1 System Description

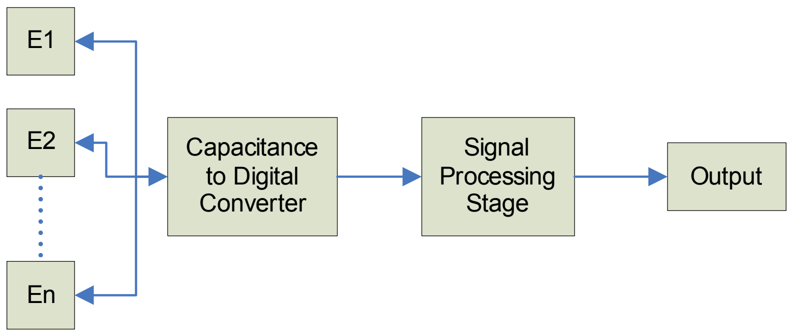


Figure 1. Block Diagram

The main components of the system are:

- Electrodes — These are the user interface and provide the inputs to the system.
- Capacitance to digital converter — This provides digital values proportional to the input capacitance of each electrode.
- Signal processing stage — This is where the measurements are processed and determines whether the electrode is touched or not.
- Output — This is a buzzer. Here the user can have feedback of the processes inside the system.

2.1.1 Electrodes

The electrodes are conductive material areas in the PCB that serve as plates of a capacitor. To charge these capacitors a pull-up resistor is connected to each of them forming a simple RC circuit.

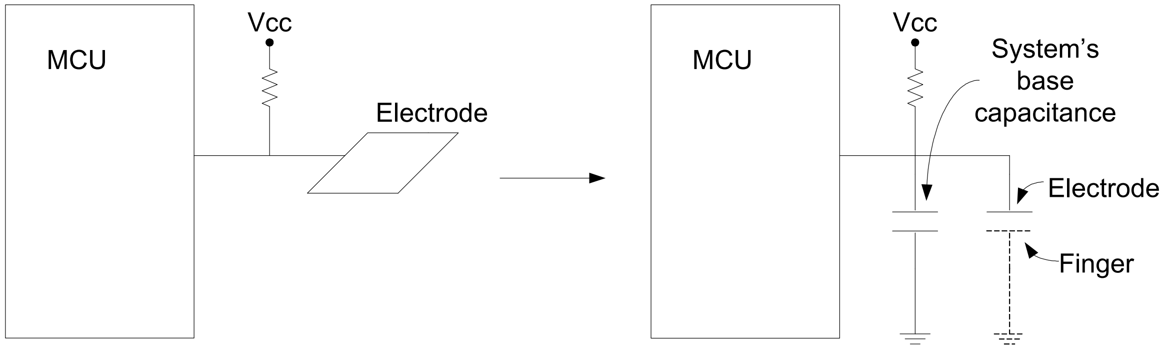


Figure 2. Electrodes Equivalent Circuit

2.1.2 Capacitance to Digital Converter

The function of the RC circuit is to transform capacitance into time. This transformation is determined by the time constant of the circuit (Equation 1).

Eqn. 1

$$\tau = RC$$

With R as a constant any capacitance variation linearly modifies the time it takes for the capacitor to reach a reference voltage. This time can be measured with a timer but a means to determine when the reference voltage has been reached is needed. Any MCU's input pin integrates a comparator to determine the logic level of its input, therefore the pin's threshold can be used as a reference voltage.

The capacitance to digital converter provides a value that is present in the input pin. The capacitance value is then increased when an electrode is touched. This must not be considered an accurate capacitance measurement. These values are meant to be processed and compared to determine the absence or presence of an object in proximity to the electrode.

2.1.3 Signal Processing Stage

The samples provided by the capacitance to digital converter must go through signal processing for proper interpretation. These samples are not accurate measurements but the capacitance variations can be detected easily with these values. This is exactly what is needed for touch detection. In this stage a variety of algorithms may be implemented to process the measurement values and improve the object detection. For example, filters to reduce noise effects and debouncing mechanisms to avoid false detections.

2.1.4 System Output

After the capacitance to digital converter output values are processed and a touch is detected the system must be capable of giving feedback of detection. In the described application a buzzer provides this feedback emitting a fixed frequency sound when a touch detection occurs. Any other outputs can be implemented.

2.2 Proximity Application Description

Figure 3 shows the system components described in the previous section that are implemented in the evaluation software.

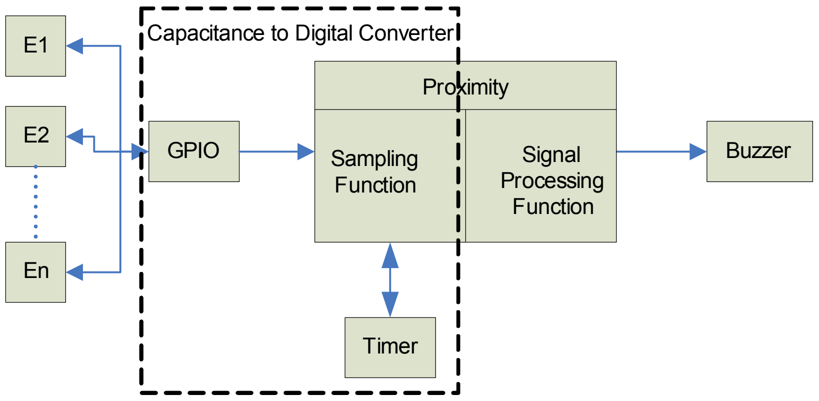


Figure 3. Application Block Diagram

Each module has specific functions:

- GPIO module — Manages everything referred to the MCU's I/O lines. This module provides the interface with the electrodes and the system output, in this case a buzzer.
- Timer module — Manages all the hardware timer module functions used in the application. For example, timer clocks and prescaler configuration, counter start, and stop and reset. The timer's ISR is also located in this module.
- Proximity module — This is the core of the application and interfaces with the GPIO and timer modules to obtain capacitance measurements. This module processes the capacitance values and evaluates if an electrode has been touched.
- Buzzer module — Provides functions to control the buzzer that produces sound when a touch has been detected.

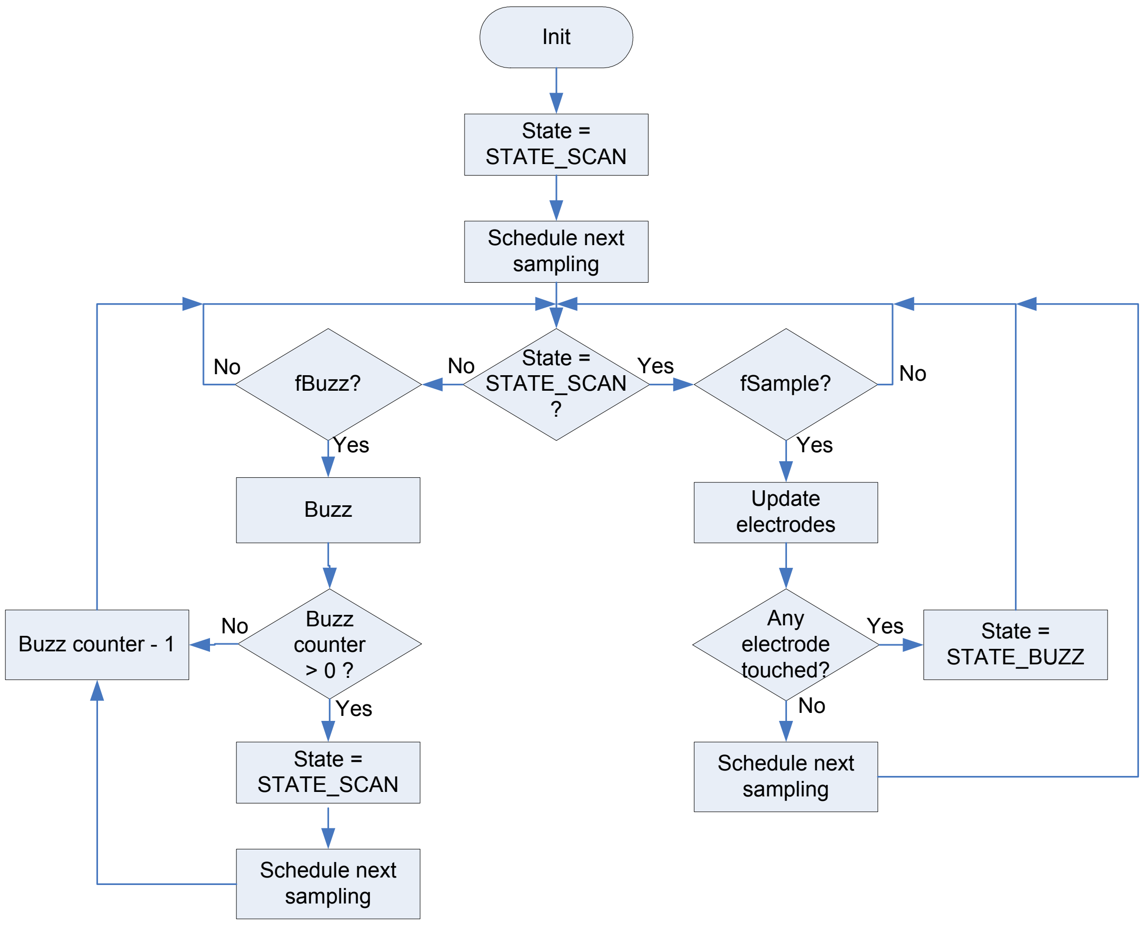


Figure 4. Flow Chart

The proximity application first initializes all necessary modules, then schedules the first sampling task and enters the scan state where one or all electrodes are periodically sampled. After the sampling period has elapsed the function in charge of sampling and processing is called and updates the status (touch/untouched) of the selected electrode. The application then evaluates if any electrode has been detected as touched and if it has, it changes the state to buzz. If no electrode has been touched the next sample is scheduled. During the buzz state a sound is emitted through a buzzer for 10 ms. After this period is completed the next sample is scheduled and the application returns to the scan state.

2.2.1 Proximity Software Architecture

For this application, the software architecture model shown in Figure 5 is implemented.

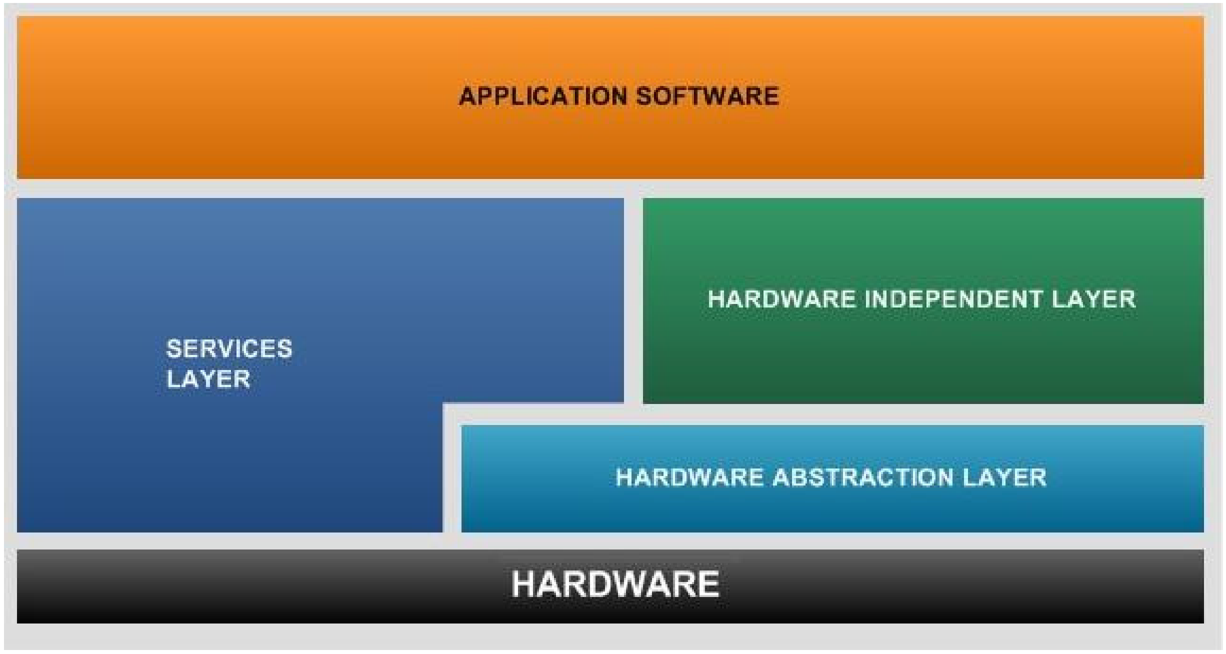


Figure 5. Software Architecture Model

This model is composed of different layers:

- Hardware Layer (HL) — The hardware layer defines the interfaces from the upper layers to the hardware resources, such as peripherals, configuration registers or any other hardware dependent resource. No functional feature is defined into this layer. This layer is implemented in the header file provided by CodeWarrior. For example the MC9S08QE128.h.
- Hardware Abstraction Layer (HAL) — The hardware abstraction layer is defined as the collection of software components that make direct access to hardware resources. Peripheral drivers are implemented in this layer. An example is, software modules that manage peripherals configuration, and peripheral events handling. The access to hardware is done through the HL interface.
- Hardware Independent Layer (HIL) — The hardware independent layer groups are all software modules that do not manage the microcontroller's resources but could make use of them to execute a specific task or function. These are groups that belong to this layer:
  - Algorithms that do not require specific hardware. For example, filters, mathematical functions, sorting, and searching algorithms.
  - Drivers for external resources that require hardware resources and interface with the hardware through the HAL. For example, a DAC communicating through I^2C or SPI.

— Peripherals that are not present in the MCU but are constructed by software using available hardware. For example, serial ports using GPIO lines and timers.

Because HIL software modules do not contain any hardware specific information they can easily be ported to other platforms providing if needed, the appropriate HAL interface.

- Services Layer (SL) — The services layer is defined as the collection of software components that provide basic services for modules in other layers that require them. This includes timing management for task scheduling, memory management, system management, and power management. For example, watchdog, system clocks configuration, and low power modes.
- Application Layer — All application-specific software is included in the application layer. All other layers implement more abstract functions while the application layer integrates them to create application specific functions. For example the softeware components required to manage a user interface are contained in the HIL and HAL layers but the task to be executed depending on the input is decided in the application layer.

Following this software architecture model the modules of the system must be accommodated in their respective layers. It is clear that a timer and GPIO pins are the only MCU hardware resources needed for touch sensing. They interact directly with the hardware and belong in the HAL. The proximity and buzzer modules make use of HAL components. They provide support for resources external to the MCU and therefore belong in the HIL. The MCU clocks initialization is part of the SL. The resulting application model is shown in Figure 6.

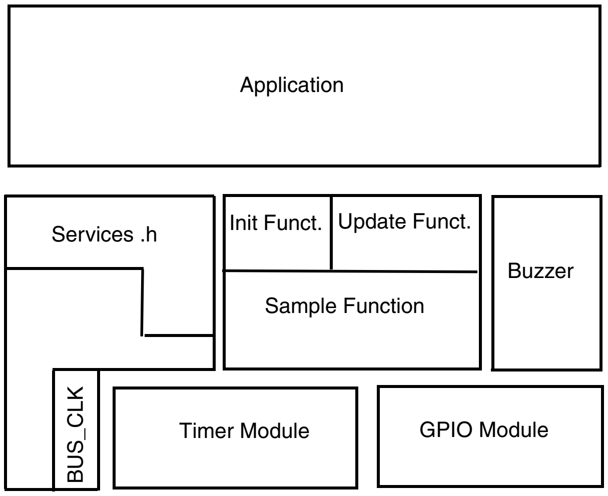
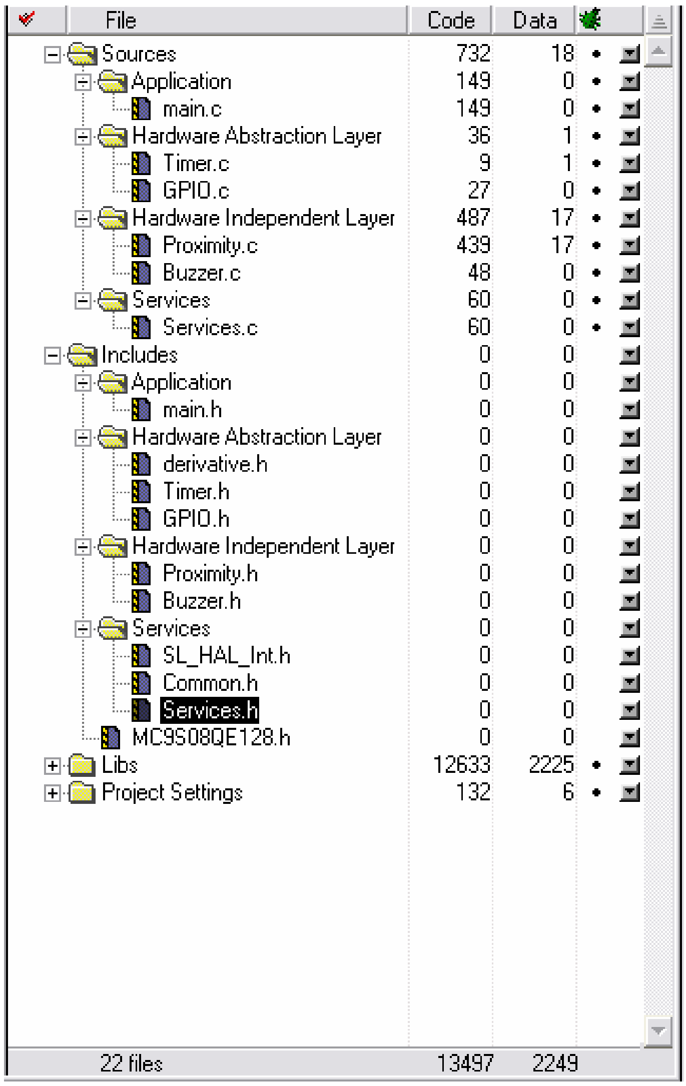


Figure 6. Proximity Software Architecture

Each of these modules are composed by a code and a header file. The header files have two sections, one named public and the other named private. This separation is for clarification because everything defined in the file is actually public. However, these restrictions must be respected by the user to improve code portability and reusability. The interfaces of each module are contained in the public section of each file and are explained later in the document



File	Code	Data	
Sources	732	18	•
Application	149	0	•
main.c	149	0	•
Hardware Abstraction Layer	36	1	•
Timer.c	9	1	•
GPIO.c	27	0	•
Hardware Independent Layer	487	17	•
Proximity.c	439	17	•
Buzzer.c	48	0	•
Services	60	0	•
Services.c	60	0	•
Includes	0	0	
Application	0	0	
main.h	0	0	
Hardware Abstraction Layer	0	0	
derivative.h	0	0	
Timer.h	0	0	
GPIO.h	0	0	
Hardware Independent Layer	0	0	
Proximity.h	0	0	
Buzzer.h	0	0	
Services	0	0	
SL_HAL_Int.h	0	0	
Common.h	0	0	
Services.h	0	0	
MC9S08QE128.h	0	0	
Libs	12633	2225	•
Project Settings	132	6	•
22 files	13497	2249	

Figure 7. Proximity Files Structure

2.2.2 Services Layer

The SL provides basic system services. This includes clocks and timer initialization. Three files belong to this layer:

- SL_HAL_Int.h. — This file is the interface between the SL and the HAL. The bus clock frequency is defined here and is needed for timer configuration and for the MCU clock module configuration (ICG or ICS for supported microcontrollers).
- Services.h. — All the clock modules configuration parameters are defined here as private. This is managed in the SL and no other module can make use of these values. Basic scheduler functionalities are provided as public and configure the timer to interrupt at a certain period. A flag is also set to signal this event. This can be used by the application or the HIL modules to schedule

tasks. The MCU_Init function prototype is also declared here and must only be called by the application.

- Services.c. — The microcontroller initialization and clocks modules initialization functions are located in this file. The application must call the MCU_Init function to initialize the necessary modules for system operation. This function initializes the clocks and timer modules with the Clocks_Init and TIMER_CONFIGURE functions.

3 Proximity Module

3.1 General Description

The proximity module is the core of the application. This module manages all the sampling processes for each electrode. It processes the measurement data and evaluates whether an electrode must be reported as touched or untouched. It makes use of the GPIO and timer modules. The GPIO is for interfacing with the electrodes and the timer module is to measure the capacitance. The proximity module belongs to the HIL and provides the application with a public variable to report the electrodes status and an array to perform and store the average of the previous samples per electrode.

Table 1 shows the three functions that perform the above mentioned tasks.

Table 1. Proximity Functions Table

Function Name	Input Parameters	Return Value
Proximity Init	None	Status code
Sample Electrode	Port pointer, bit mask, buffer	Status code
Update Electrode Status	Electrode identifier	Status code

3.2 Proximity Initialization Function

This function sets the initial state of averages by sampling every electrode and storing the reading as the current average value. This reduces the settling time to 0 and avoids false detections that can occur through the settling process. If a sample is invalid the average is set to 0xFF to avoid false detections while the average stabilizes with future valid samples. It receives no arguments and returns a status code reporting any possible malfunction.

3.3 Electrode Sampling Function

This function is private of this module and performs the actual measurement of the capacitance in the selected electrode pin. This is done by making use of the GPIO and timer modules. The input arguments are a pointer to the port where the desired electrode is connected, a mask of the specific pin where the electrode is found, and a pointer to a variable where the measurement value must be stored. The function returns a status code indicating whether the measurement is valid.

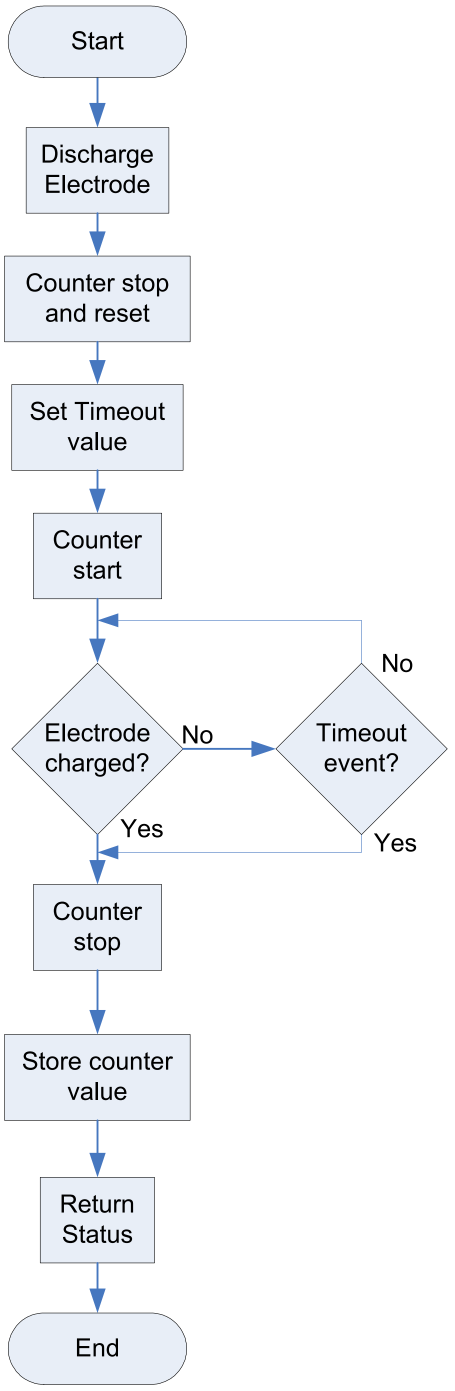


Figure 8. Sample Electrode Function Flow Chart

In Figure 8, first, the selected electrode is discharged to set its initial state near to 0 V. This is done by configuring an electrode's pin as an output and setting it to a logic 0 causing the previous charge of the capacitor to return to ground. Next, the measurement counter is stopped in case it is running, then reset and started again.After, the electrode's pin is configured as input. This allows the capacitor to recharge, and its value is constantly polled. This begins the measurement process with the counter running and the capacitor charging, although there is a delay between the counter start and the charge start the difference is always the same, therefore the measured value is not affected. After the capacitor voltage reaches a certain level the pin value is detected as a logic 1. After this happens, the counter is stopped and its value stored in the specified memory location.

If for any reason the capacitor never reaches the threshold value, a timeout event can stop the pin polling. This timeout event is signaled by the timer ISR through a software flag and the timeout value is configured when the measurement counter is started. The timeout flag is polled along with the pin value ensuring that the CPU is not deadlocked in a measurement forever. If a timeout event occurs the function returns a timeout status code indicating the reading is not valid and the timeout flag is cleared. If the measurement is valid the function returns an ok status code.

NOTE

When using the proximity module in any application it is important to keep in mind that the CPU is deadlocked for the time the measurement takes. This time depends on the base capacitance of the system. The worst case being the configured timeout value (75 µs for this application). Also, if another interrupt occurs during the pin polling the sample may not be valid and the user must implement a method to detect it.

3.4 Update Electrode Status Function

This function is the interface for the application layer. It must be called by the application to refresh the electrodes status. It performs the averaging of samples and compares the new sample with the previous average to determine if a specific electrode has been touched. The SampleElectrode function is called within the update function to obtain the new sample for the desired electrode. The only input parameter is the electrode to be updated that can also be set to the SAMPLE_ALL constant that drives the function to sample all the electrodes one by one.

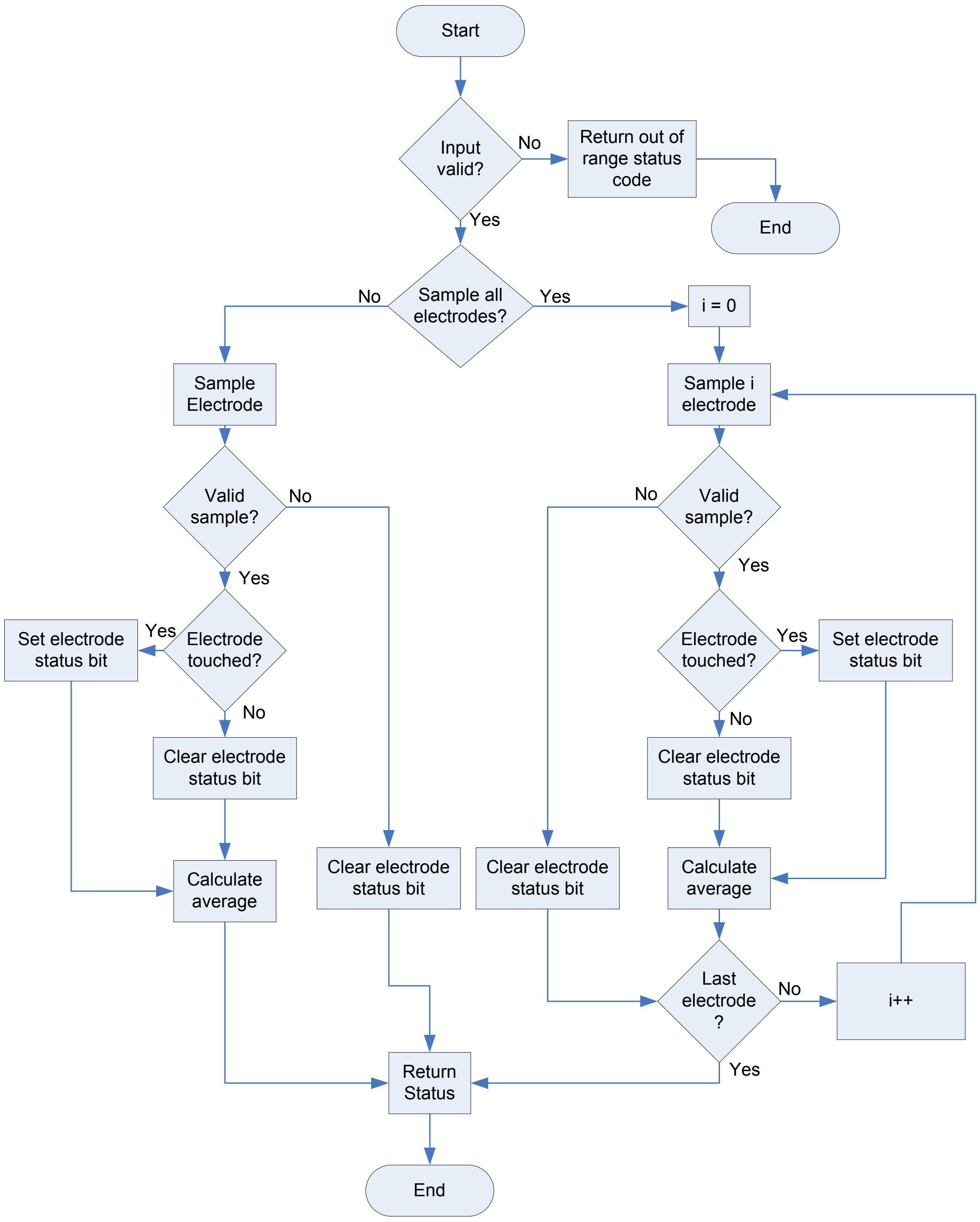


Figure 9. Electrode Status Update Function Flow Chart

In Figure 9, the input parameter is first validated to avoid any system misbehavior. If it is not valid an out of range status code is returned and the function exits.When the parameter is an electrode symbol (E1, E2… En) the sampling function is called to obtain the current capacitance of that electrode. Before processing this sample the status code is returned by the sampling function and checked to ensure that the reading is valid. If not, the electrode is reported as untouched and the same status code is returned by the update function. If the sample is valid the difference between the current sample and the average of the previous samples are compared to a threshold value to determine if the variation is significant enough to consider the electrode touched. The corresponding bit in the gesElectrodeStatus variable is then set or cleared to reflect the result of the previous comparison. Finally, the average is calculated to integrate the new sample and an ok status code is then returned. If the input parameter is the SAMPLE_ALL constant the whole process described is done for every electrode in the same function call.

3.4.1 Average

Any average acts as a low pass filter and helps to reduce the effects of noise. In this module a function that acts as an average is implemented (Equation 2). This approach does not require implementing a buffer to store the previous samples and avoids the need of integration of these samples every time, although it does have a slower response. Because the data is also processed with integer values a considerable steady state error is present. This error represents no problem if the system's sensitivity is good. If more precision is needed the average value can be calculated using fixed point values. Depending on the base capacitance of the system this can be done with eight bits or it may require to extend the values to sixteen bits.

Eqn. 2

$$Avg = Avg + \frac{(Sample - Avg)}{N}$$

N is a constant equal to the number of samples averaged. N = 8 for an 8 samples average.

Considering that a real average is defined by Equation 3 and substituting this in Equation 2, Equation 4 is obtained.

Eqn. 3

$$Avg = \frac{\sum_{i=0}^{N-1} Sample[i]}{N}$$

Eqn. 4

$$Avg = \frac{\sum_{i=0}^{N-1} Sample[i]}{N} + \frac{(Sample[N] - \frac{\sum_{i=0}^{N-1} Sample[i]}{N})}{N} = \frac{(N-1)\sum_{i=0}^{N-1} Sample[i]}{N^2} + \frac{Sample[N]}{N}$$

For a signal in a constant steady state where all the previous samples have the same value, Equation 4 can be expressed as:

Eqn. 5

$$Avg = \frac{N(N-1)Sample}{N^2} + \frac{Sample[N]}{N} = \frac{(N-1)Sample + Sample[N]}{N}$$

Expressing Equation 5 in summations:

Eqn. 6

$$Avg = \frac{\sum_{i=1}^{N-1} Sample[i] + \sum_{i=N-1}^{N} Sample[i]}{N} = \frac{\sum_{i=1}^{N} Sample[i]}{N}$$

Note that the result in Equation 6 is almost equal to Equation 3 which is the original average, but in Equation 6 the limits of the summation have been shifted. By observing the consequence of the process it is determined that the operations in Equation 2 subtracts a sample from the average. This sample is considered to be the oldest because all the samples are equal. The new sample is intergrated to the previous average with its respective weight.

This works fine for steady states with minimal variation as seen in Figure 10. When the signal changes level it takes longer for this function than the real average to reach the new value.

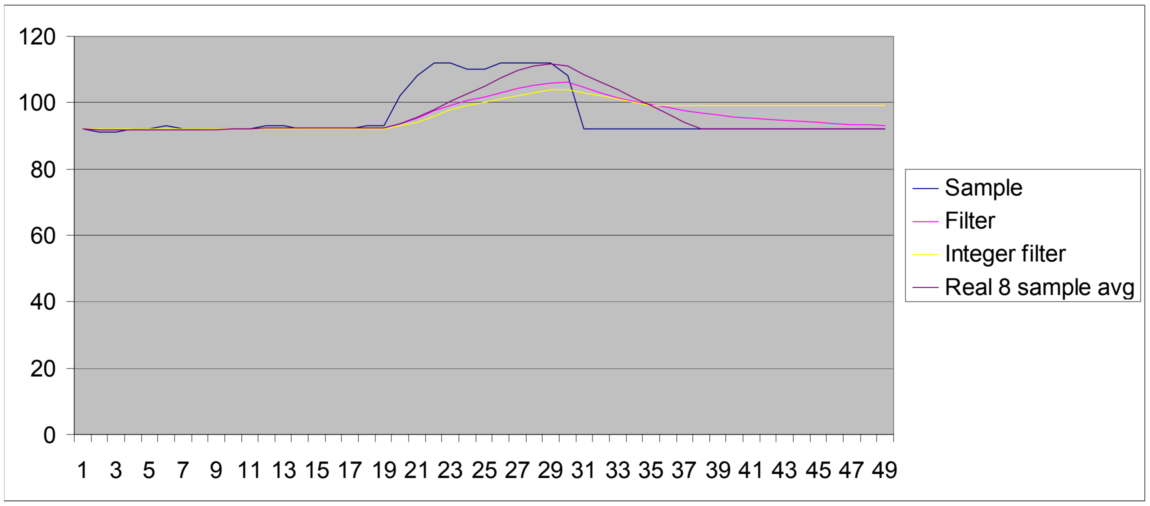


Figure 10. Average Plot

3.4.2 Threshold Comparison

The average values are used as a reference for comparison with every new sample to determine if the electrode is touched. This is done by subtracting the average value to the sample and comparing this difference to a predetermined threshold. If the difference exceeds the threshold value the status of the electrode is reported as touched. If it is lower than the threshold value the status is set to untouched. Figure 11 shows the difference between the samples and average of the plot in Figure 10. Every time this difference is higher than the threshold value, which is 10 for this example, the electrode is reported as touched.

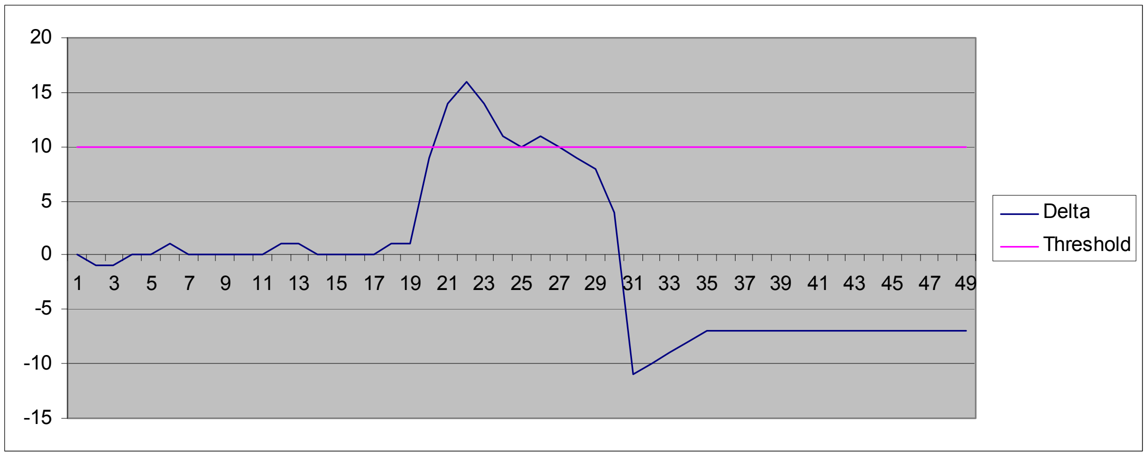


Figure 11. Threshold Comparison

3.4.3 Electrode Status Register

The proximity module provides the gesElectrodeStatus variable to report the status of every electrode. Each bit of the variable represents an electrode with the LSB corresponding to the first electrode (E1). This variable is an ELEC_STAT_REG and as default an unsigned eight bits. If more than eight electrodes are desired for an application the ELEC_STAT_REG must be redefined to support sixteen bits. The definition of this type along with many others is done in the proximity header file.

3.4.4 Proximity Header File

The proximity.h file contains the public and private definitions for the proximity module. The public section is composed by the ELEC_STAT_REG definition, prototypes for the public functions, public variables definitions, the electrodes symbols, and the parameter N_ELECTRODES that define the number of electrodes supported by the module. If the electrode configuration is changed this parameter must be redefined to meet the actual configuration. The private section contains macro definitions for electrode and counter handling, and the timeout parameter defined as TIME_LIMIT. This parameter is expressed in counter units.

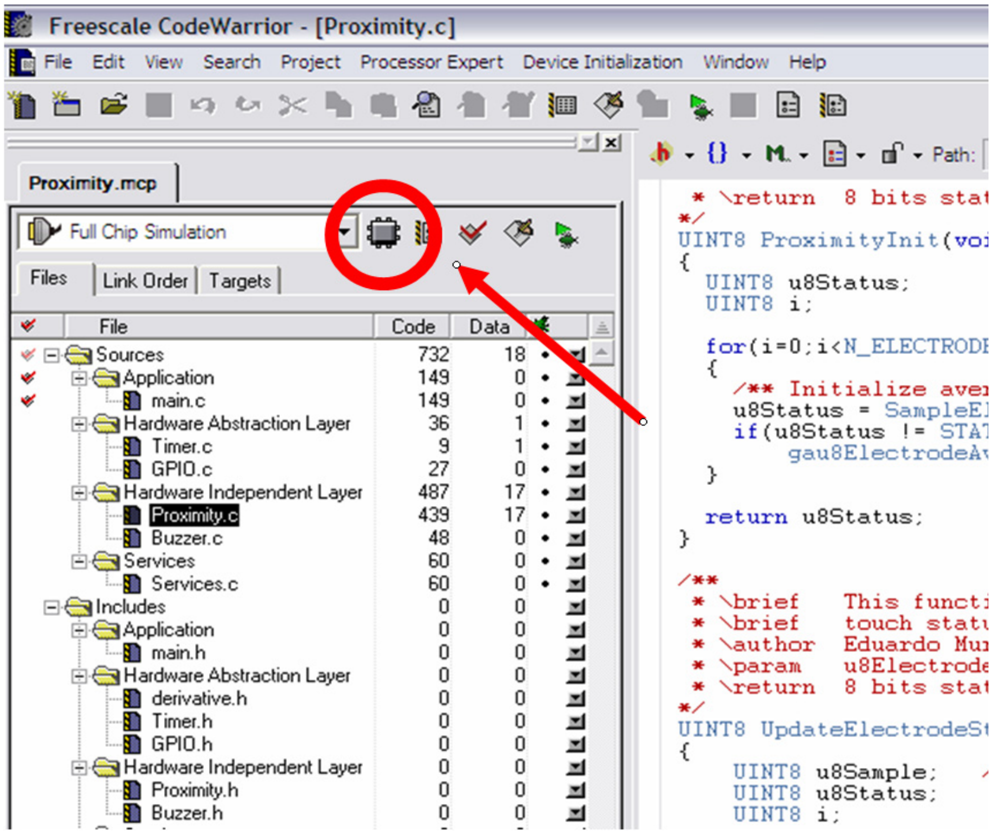
4 Hardware Setup

When setting up the software for a specific system implementation the first variable to be defined is the platform to be used. This determines the available hardware and the core capabilities. The RS08, S08 and V1 platforms are currently supported by the application.

4.1 Porting the Application to RS08, S08 and ColdFire V1

The implemented software architecture's main purpose is the code's portability. This allows for complete applications to be easily ported to different platforms and for individual modules to be reused in different applications. Taking advantage of this, all RS08, S08 and V1s MCUs have two different projects, one for the RS08 and another for the S08 and V1.

After the proper project is opened the specific MCU must be selected. The RS08 project is set by default for the RS08KA2, and the S08/V1 is set for the S08QE128. CodeWarrior provides a feature to change the target MCU, this feature is called, Change MCU/Connection, located in the Project menu or in the Make and Debug buttons area. This is shown in Figure 12.


Freescale CodeWarrior - [Proximity.c]
File Edit View Search Project Processor Expert Device Initialization Window Help
Path:
Proximity.mcp
Full Chip Simulation
Files
Link Order
Targets

File	Code	Data
Sources	732	18
Application	149	0
main.c	149	0
Hardware Abstraction Layer	36	1
Timer.c	9	1
GPIO.c	27	0
Hardware Independent Layer	487	17
Proximity.c	439	17
Buzzer.c	48	0
Services	60	0
Services.c	60	0
Includes	0	0
Application	0	0
main.h	0	0
Hardware Abstraction Layer	0	0
derivative.h	0	0
Timer.h	0	0
GPIO.h	0	0
Hardware Independent Layer	0	0
Proximity.h	0	0
Buzzer.h	0	0

```
 * \return   8 bits stat
*/
UINT8 ProximityInit(voi
{
   UINT8 u8Status;
   UINT8 i;

   for(i=0;i<N_ELECTRODE
   {
     /** Initialize aver
     u8Status = SampleEl
     if(u8Status != STAT
         gau8ElectrodeA
   }

   return u8Status;
}

/**
 * \brief    This functi
 * \brief    touch statu
 * \author   Eduardo Mu
 * \param    u8Electrode
 * \return   8 bits stat
*/
UINT8 UpdateElectrodeSt
{
     UINT8 u8Sample;
     UINT8 u8Status;
     UINT8 i;
```

Figure 12. Change MCU/Connection Function Location

When selecting this option, a window appears showing the list of available MCU families. The desired MCU must be selected as well as the proper connection method for the debugger.

NOTE

When using the RS08 project, no other S08/V1 MCU may be selected, and vice versa. The architectural differences between these two cores lead to the creation of different projects.

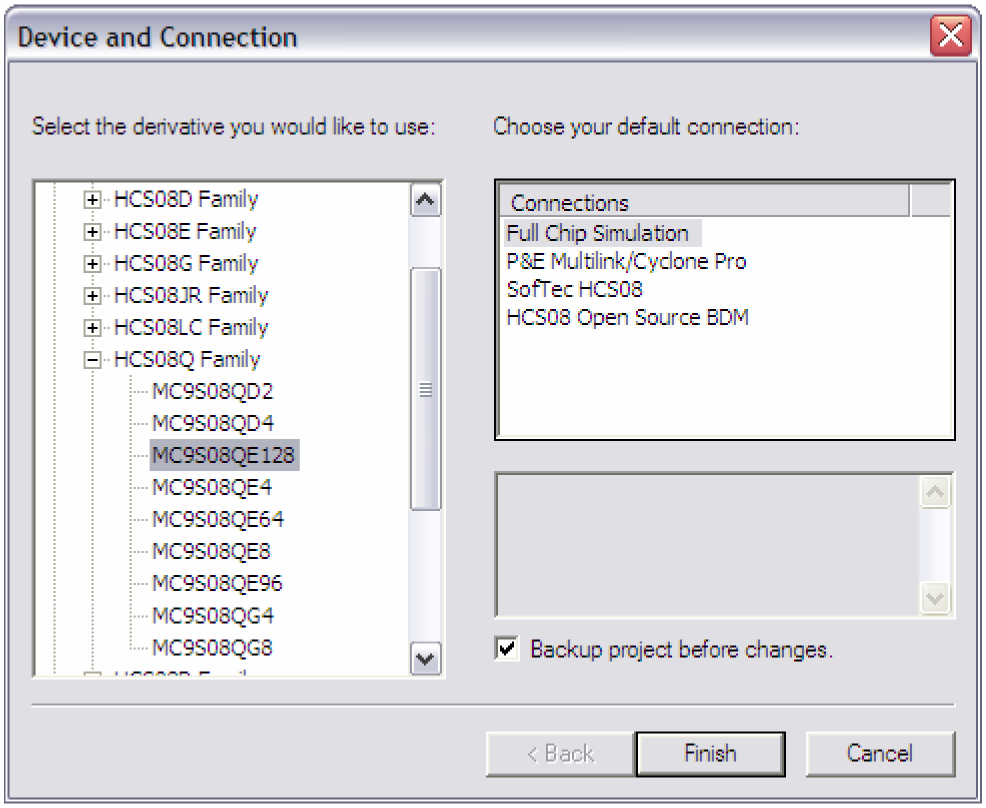


Figure 13. MCU/Connection Selection Window

4.2 Configuring Electrodes and Buzzer Location

The next hardware settings to be defined are the GPIO pins that are used for the electrodes and the output signal. Based on the hardware definition of the system the electrode pins and number of electrodes can be configured in the project. When using an MCU demo board in conjunction with the KITPROXIMITYEVM module the electrodes location can be found in the user guide. This location is defined in the project inside the GPIO module.

4.2.1 GPIO Module

To improve flexibility all the MCU pins used for the electrodes are gathered in an array of pins. This creates what can conceptually be seen as a virtual port dedicated to manage common signals for a specific function. This array has no size restriction and any GPIO pin can be included and allows for any electrodes and output configuration.

NOTE

Only input/output pins can be used for proximity sensing. Input or output only pins must not be connected to electrodes and as a consequence must not be part of the electrodes pin array.

These functions are contained in the GPIO module that belong to the HAL of the architecture. Each module is composed of a header and a code file. For this module the contents of each file are the following:

- Header file — Contains the required macros for the interface with upper layers as well as the type definition for the virtual ports.
- Code file — Contains the actual virtual ports definitions. Here is where modifications to electrodes and buzzer location must be done.

The virtual port type is a structure that contains a pointer to an MCU port and a mask for the chosen pin for that port. With this information any operation can be done to a specific pin and all the port pointer-pin pairs can be grouped in one array for easy access. This kind of structure is flexible because an index variable can be used for more dynamic pin access.

In the implemented module two virtual ports have been created: one for electrodes and another for the output buzzer. For example, if a visual feedback were to be implemented turning on an LED for every electrode the buzzer virtual port can be extended for the purpose of grouping all the output signals in one structure.

4.2.2 Buzzer Module

A basic buzzer driver module was created for this application. It uses a GPIO pin that is part of a virtual port defined in the GPIO module that sends a 10ms signal of a determined frequency. In the header file different frequency indexes have been defined as well as the corresponding interface macros and functions.

The interface is composed by the frequency indexes, the BUZZER_INIT() macro that configures the buzzer pin as output, the BUZZER_SOUND() macro that toggles the buzzer pin, and the BuzzerConfigure function that calculates the necessary values for the timing control to produce the desired frequency sound. This function receives the frequency index and a pointer to an eight bits variable to store the 10ms counter.

The BUZZER_OUTPUT value is located in the private section of the header file and selects the element of the virtual port that is used for the buzzer. In this section the timing values for the sound generation are also defined.

4.3 Timer Module

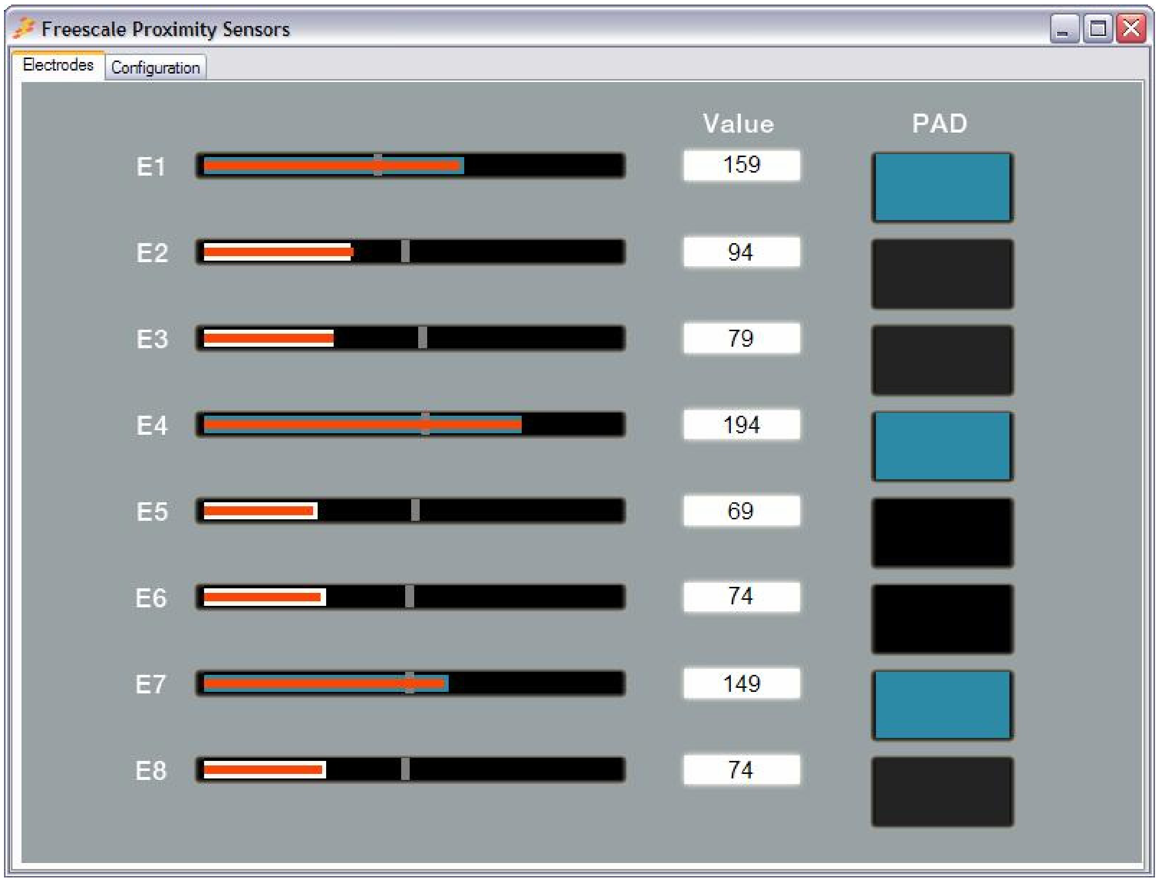
The timer module provides timing functions for this entire application. The TPM module is supported in the S08/V1 project, and the MTIM is used for the RS08. This module makes use of the services layer to obtain the MCU bus clock value to set the proper clock configuration for the desired timer clock frequency.

The public section of the header file contains the definition of a bits structure type that creates a flag that reports interrupt events to the upper layers and all the macros required for the interface with this HAL component. Even though the hardware modules used in the RS08 and S08/V1 are different the same interface is provided for compatibility. When a different hardware is used the modules that make use of this component do not need to be modified.

All the necessary definitions for the hardware module configuration can be found in the private section. In some microcontrollers more than one TPM module is available and the user can choose which one to use in the first subsection. Here, the base address of the module is taken from the TPMxSC register and the other module registers are defined later in the file as offsets of this base address. This allows the user to change modules by simply changing the TPMxSC to the status and control register of the desired TPM hardware module. The desired timer clock frequency must be configured in the TIMER_CLK parameter and the appropriate prescaler is automatically defined to achieve the nearest clock frequency. This is only true if the bus clock is used as the clock source. For this application it is a requirement.

5 Graphical User Interface

The proximity software GUI provides a powerful tool for system understanding configuration and debugging. This application enables the user to see real-time data of the capacitance measurement and to configure the desired threshold for touch detection in a graphic and more intuitive environment. It also provides a non-intrusive interface with the embedded application because it communicates with the MCU through the BDM interface. This avoids additional processing on the MCU software for communication.

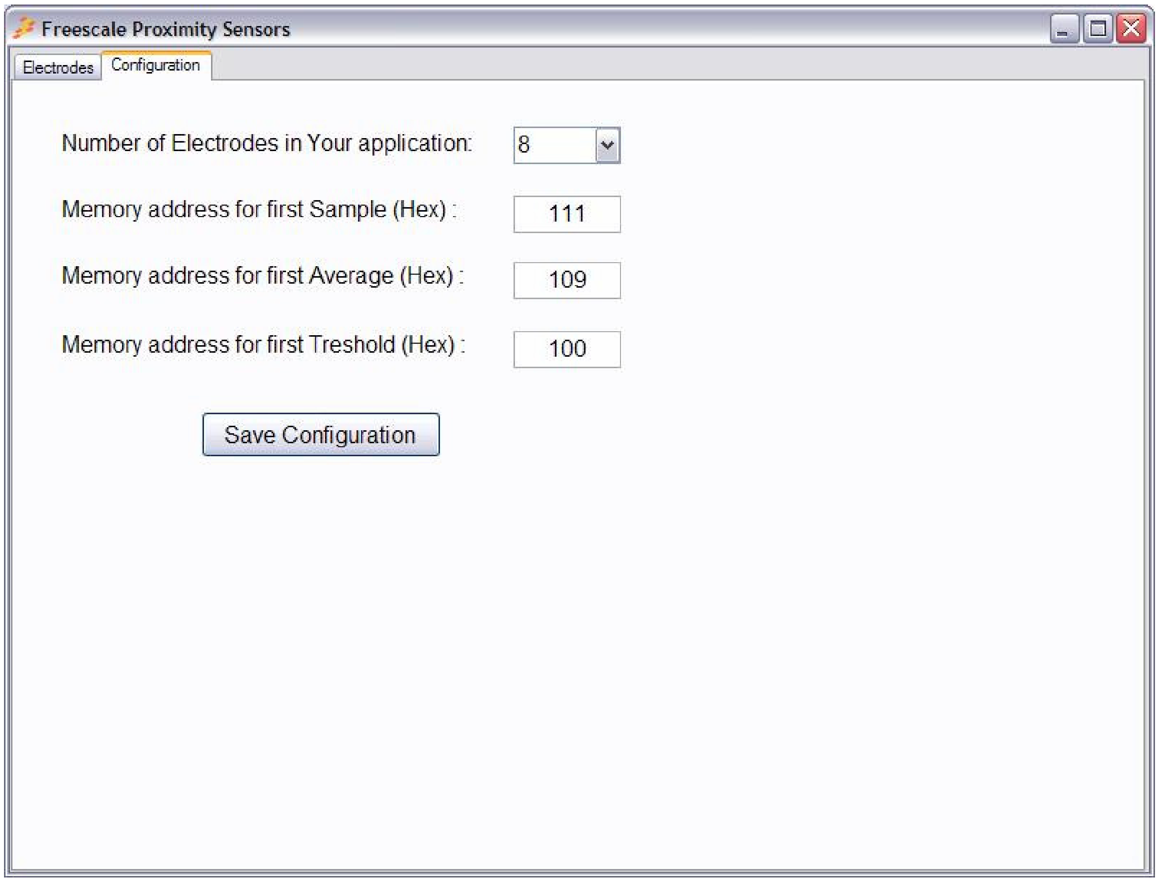


	Value
E1	159
E2	94
E3	79
E4	194
E5	69
E6	74
E7	149
E8	74

Figure 14. Application Front End

5.1 GUI Setup

This application reads the data values directly from RAM through the BDM interface without interfering with the CPU processing. The GUI must be provided with the memory addresses where the data is located. This is done by selecting the configuration option of the main windows and manually entering the addresses of the first elements of the data arrays and selecting the number of electrodes enabled in the system.



Number of Electrodes in Your application: 8

Memory address for first Sample (Hex) : 111

Memory address for first Average (Hex) : 109

Memory address for first Treshold (Hex) : 100

Figure 15. Data Configuration Option

5.2 Capacitance Bars

Figure 14 shows the bars that represent the capacitance present in every electrode. The orange bar represents the average value and the white bar represents the current sample value. The numeric value of the sample is shown in the box next to the bars. This allows the user to graphically observe the system's behavior, determine the base capacitance of the system, and detect the capacitance variations if an object is in proximity to the electrodes.

5.3 Threshold Detection

The threshold for touch detection is configurable with the sliders located in every electrodes bar. The user can observe the capacitance variations when touching an electrode and decide where to set its threshold. Every time the capacitance bar crosses the threshold marker the button next to the bar turns green.

NOTE

The GUI must be started without touching any electrode for proper threshold detection.

6 Conclusions

6.1 Proximity Software Reusability

The proximity module composed by proximity.c and proximity.h files is highly portable through platforms and applications because it has no direct hardware dependencies. If the user wants to integrate touch sensing capabilities in a different application the proximity module can be integrated into the project. The interface functions from the lower layers must be provided to this module. This includes the timer handling and the GPIO functions that the module makes use of. For the timer module the interface is composed by the elements in Table 2.

Table 2. Timer Module Interface

Macro	Description
TIMER_START()	Starts the timer counter used for measurement
TIMER_STOP()	Stops the counter
TIMER_RESET()	Sets the timer counter to 0
TIMER_GET_COUNT()	Returns an 8-bits counter value
TIMER_CONFIGURE()	Called once to initialize the peripheral
TIMER_SET_MOD(x)	Defines x as the counter value that triggers a timeout event

In addition to these macros a flag contained in a bits structure to signal the timeout event is needed and declared as frTimer_flags.Bits.Timeout. To use the proximity module in a microcontroller where no TPM is available the corresponding timer module must be created integrating all the mentioned functions. The hardware change then is transparent for the proximity module and no modifications are required.

For the GPIO module, Table 3 shows the elements of the interface. All macros receive a pointer to a port and a mask of the specific pin.

Table 3. GPIO Module Interface

Macro	Description
PIN_OUTPUT(x,y)	Configures the pin y of port *x as output
PIN_INTPUT(x,y)	Configures the pin y of port *x as input
PIN_SET(x,y)	Sets the pin y of port *x
PIN_CLEAR(x,y)	Clears the pin y of port *x
PIN_TOGGLE(x,y)	Toggles the pin y of port *x

The port pointers and the pins masks are recommended to be stored in the VIRTUAL_PORT type declared in the GPIO.h file of the provided application. The same macros and a structure like the vpPortx created in the GPIO.c file must be provided to the proximity module if a different GPIO module is used.

6.2 Freescale Proximity Solutions

The proximity software solution explained through this document is capable of touch sensing, but not suitable for real applications by itself. The level of data processing and the detection algorithm is not reliable enough for integration in a commercial product although the user may implement other algorithms over the proximity module. Freescale offers a variety of highly integrated proximity solutions that include external digital and analog sensors. These are robust designs that integrate validated algorithms for proximity sensing. For more information on the proximity line please visit www.freescale.com/proximity.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3579
Rev. 0
11/2007