IQS316 Design Guide
IQ Switch® - ProxSense™ Series

Multi-channel Integrated Proximity Sensor with Micro-Processor Core

This design guide provides a description of the communication interface between the master and the IQS316 controller. The Memory Map of the IQS316 is provided in this document, followed by a description of each register and instruction. The IQS316 communicates in I^{2}C or SPI mode, both using a Memory Mapped structure. The last section of this document is dedicated to an example implementation and provides example code.

Contents

1 Memory Map

1.1 General Memory Map Structure

A general I^{2}C and SPI Memory Map is defined so that all ProxSense devices can use a standard framework. The general mapping is shown below.

Table 1.1 IQS316 Memory Mapping

Address	Access	Size(Bytes)	
00H-0FH	R	16	**Device Information**

Address	Access	Size(Bytes)	
10H-30H	R	32	**Device Specific Data**

Address	Access	Size(Bytes)	
31H-34H	R	4	**Proximity Status Bytes**

Address	Access	Size(Bytes)	
35H-38H	R	4	**Touch Status Bytes**

Address	Access	Size(Bytes)	
39H-3CH	R	4	**Halt Bytes**

Address	Access	Size(Bytes)	
3DH-41H	R	4	**Active Bytes (indicate cycle)**

Address	Access	Size(Bytes)	
42H-82H	R	64	**Current Samples**

Address	Access	Size(Bytes)	LTAs
83H-C3H	R/W	64	

Address	Access	Size(Bytes)	Device Settings
C4h-FDh	R/W	64	

* Note 'FE' and 'FF' are reserved for other functions in communication.

1.2 IQS316 Memory Map

1.2.1 Device Information

Address									
00H		**Product Number**							
Access	Bit	7	6	5	4	3	2	1	0
R	Value	27 (Decimal)							

Address									
01H		**Version Number**							
Access	Bit	7	6	5	4	3	2	1	0
R	Value	Variable							

1.2.2 Device Specific Data

Address		XY Info 1 (UI_FLAGS0)							
10H	Bit	7	6	5	4	3	2	1	0
Access	Value	SHOW_RESET	MODE_INDICATOR	~	~	~	ATI_BUSY	RESEED_BUSY	NOISE
R									

1.2.3 Proximity Status Bytes

Only the proximity status of the channels relating to the current group is available here.

Address		Proximity Status (Group dependant)							
31H	Bit	7	6	5	4	3	2	1	0
Access	Name	If Group = 0				CH3	CH2	CH1	CH0
R	Name	If Group = 1				CH7	CH6	CH5	CH4
	Name	If Group = 2				CH11	CH10	CH9	CH8
	Name	If Group = 3				CH15	CH14	CH13	CH12
	Name	If Group = 4				CH19	CH18	CH17	CH16

1.2.4 Touch Status Bytes

Only the touch status of the channels relating to the current group is available here.

Address		Touch Status (Group dependant)							
35H	Bit	7	6	5	4	3	2	1	0
Access	Name	If Group = 0				~	~	~	~
R	Name	If Group = 1				CH7	CH6	CH5	CH4
	Name	If Group = 2				CH11	CH10	CH9	CH8
	Name	If Group = 3				CH15	CH14	CH13	CH12
	Name	If Group = 4				CH19	CH18	CH17	CH16

*Note: This byte is not used for Group 0 (Prox Mode)

1.2.5 Halt Bytes

Only the filter halt status of the channels relating to the current group is available here.

Address
39H
Access
R

	Halt Status (Group dependant)							
Bit	7	6	5	4	3	2	1	0
Name	If Group = 0				CH3	CH2	CH1	CH0
Name	If Group = 1				CH7	CH6	CH5	CH4
Name	If Group = 2				CH11	CH10	CH9	CH8
Name	If Group = 3				CH15	CH14	CH13	CH12
Name	If Group = 4				CH19	CH18	CH17	CH16

1.2.6 Active Bytes

The group number is given here.

Address
3DH
Access
R

	Group Number							
Bit	7	6	5	4	3	2	1	0
Value	Variable (0-4)							
Note	Indicates which group's data is currently available							

1.2.7 Current Samples

The Current Samples of the current group are available here.

Address
42H
Access
R

	Current Sample CH0 / CH4 / CH8 / CH12 / CH16							
Bit	7	6	5	4	3	2	1	0
Value	Variable (HIGH byte)							
Note	CH0 (Group0) / CH4 (Group1) / CH8 (Group2) / CH12 (Group3) / CH16 (Group4)							

Address		Current Sample CH0 / CH4 / CH8 / CH12 / CH16							
43H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW byte)							
R	Note	CH0 (Group0) / CH4 (Group1) / CH8 (Group2) / CH12 (Group3) / CH16 (Group4)							

Address		Current Sample CH1 / CH5 / CH9 / CH13 / CH17							
44H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (HIGH byte)							
R	Note	CH1 (Group0) / CH5 (Group1) / CH9 (Group2) / CH13 (Group3) / CH17 (Group4)							

Address		Current Sample CH1 / CH5 / CH9 / CH13 / CH17							
45H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW byte)							
R	Note	CH1 (Group0) / CH5 (Group1) / CH9 (Group2) / CH13 (Group3) / CH17 (Group4)							

Address		Current Sample CH2 / CH6 / CH10 / CH14 / CH18							
46H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (HIGH byte)							
R	Note	CH2 (Group0) / CH6 (Group1) / CH10 (Group2) / CH14 (Group3) / CH18 (Group4)							

Address		Current Sample CH2 / CH6 / CH10 / CH14 / CH18							
47H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW byte)							
R	Note	CH2 (Group0) / CH6 (Group1) / CH10 (Group2) / CH14 (Group3) / CH18 (Group4)							

Address		Current Sample CH3 / CH7 / CH11 / CH15 / CH19							
48H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (HIGH byte)							
R	Note	CH3 (Group0) / CH7 (Group1) / CH11 (Group2) / CH15 (Group3) / CH19 (Group4)							

Address		Current Sample CH3 / CH7 / CH11 / CH15 / CH19							
49H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW byte)							
R	Note	CH3 (Group0) / CH7 (Group1) / CH11 (Group2) / CH15 (Group3) / CH19 (Group4)							

1.2.8 Long-Term Averages and Thresholds

The Long-Term averages, and each individual channels thresholds, of the current group are available here to read AND overwrite.

Address		Long-Term Average CH0 / CH4 / CH8 / CH12 / CH16							
83H	Bit	7	6	5	4	3	2	1	0
Access	Value	Touch Threshold		Prox Threshold		Variable (HIGH byte)			
R/W	Note	CH0 (Group0) / CH4 (Group1) / CH8 (Group2) / CH12 (Group3) / CH16 (Group4)							

Address		Long-Term Average CH0 / CH4 / CH8 / CH12 / CH16							
84H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW byte)							
R/W	Note	CH0 (Group0) / CH4 (Group1) / CH8 (Group2) / CH12 (Group3) / CH16 (Group4)							

Address		Long-Term Average CH1 / CH5 / CH9 / CH13 / CH17							
85H	Bit	7	6	5	4	3	2	1	0
Access	Value	Touch Threshold		Prox Threshold		Variable (HIGH byte)			
R/W	Note	CH1 (Group0) / CH5 (Group1) / CH9 (Group2) / CH13 (Group3) / CH17 (Group4)							

Address		Long-Term Average CH1 / CH5 / CH9 / CH13 / CH17							
86H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW byte)							
R/W	Note	CH1 (Group0) / CH5 (Group1) / CH9 (Group2) / CH13 (Group3) / CH17 (Group4)							

Address		Long-Term Average CH2 / CH6 / CH10 / CH14 / CH18							
87H	Bit	7	6	5	4	3	2	1	0
Access	Value	Touch Threshold		Prox Threshold		Variable (HIGH byte)			
R/W	Note	CH2 (Group0) / CH6 (Group1) / CH10 (Group2) / CH14 (Group3) / CH18 (Group4)							

Address		Long-Term Average CH2 / CH6 / CH10 / CH14 / CH18							
88H	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW byte)							
R/W	Note	CH2 (Group0) / CH6 (Group1) / CH10 (Group2) / CH14 (Group3) / CH18 (Group4)							

Address		Long-Term Average CH3 / CH7 / CH11 / CH15 / CH19							
89H	Bit	7	6	5	4	3	2	1	0
Access	Value	Touch Threshold		Prox Threshold		Variable (HIGH byte)			
R/W	Note	CH3 (Group0) / CH7 (Group1) / CH11 (Group2) / CH15 (Group3) / CH19 (Group4)							

Address		Long-Term Average CH3 / CH7 / CH11 / CH15 / CH19							
8AH	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW byte)							
R/W	Note	CH3 (Group0) / CH7 (Group1) / CH11 (Group2) / CH15 (Group3) / CH19 (Group4)							

1.2.9 Device Settings

It is attempted that the commonly used settings are situated closer to the top of the memory block. Settings that are regarded as more 'once-off' are placed further down.

Address		UI Settings 0 (UI_SETTINGS0)							
C4H	Bit	7	6	5	4	3	2	1	0
Access	Name	RESEED	ATI_MODE	PROX RANGE	TOUCH RANGE	FORCE PROX MODE	FORCE TOUCH MODE	ND	0
R/W	Default	0	0	1	0	0	0	1	0

Address		Power Settings (POWER_SETTINGS)							
C5H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	SLEEP	MAIN_OSC	LP1	LP0
R/W	Default	~	~	~	~	0	0	0	0

Address		ProxSense Module Settings 1 (PROX_SETTINGS_1)							
C6H	Bit	7	6	5	4	3	2	1	0
Access	Name	CXVSS	ZC_EN	HALT1	HALT0	AUTO_ ATI	CXDIV2	CXDIV1	CXDIV0
R/W	Default	1	0	0	1	0	0	1	0

Address		ProxSense Module Settings 2 (PROX_SETTINGS_2)							
C7H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	SHIELD_ EN	STOP_ COMMS	ACK_ RESET	SKIP_ CONV	ACF_ DISABLE	LTN_ DISABLE	WDT_ DISABLE
R/W	Default	~	0	0	0	0	0	0[Note 1]	1

Note1: The LTN filter has a limitation, it is default ON, but is recommended that this feature be disabled by the user (setting the bit).

Address			ATI Multiplier C (ATI_MULT1)							
C8H	Bit		7	6	5	4	3	2	1	0
Access	Name	If Group = 0	CH3		CH2		CH1		CH0	
R/W		If Group = 1	CH7		CH6		CH5		CH4	
		If Group = 2	CH11		CH10		CH9		CH8	
		If Group = 3	CH15		CH14		CH13		CH12	
		If Group = 4	CH19		CH18		CH17		CH16	
	Default		1	1	1	1	1	1	1	1

Address		ATI Multiplier I (ATI_MULT2)							
C9H	Bit	7	6	5	4	3	2	1	0
Access	Name	CH3	CH2	CH1	CH0	If Group = 0			
R/W		CH7	CH6	CH5	CH4	If Group = 1			
		CH11	CH10	CH9	CH8	If Group = 2			
		CH15	CH14	CH13	CH12	If Group = 3			
		CH19	CH18	CH17	CH16	If Group = 4			
	Default	0	0	0	0	~	~	~	~

Address			ATI Compensation Setting (ATI_C0)							
CAH	Bit		7	6	5	4	3	2	1	0
Access	Value	If Group = 0	CH0							
R/W		If Group = 1	CH4							
		If Group = 2	CH8							
		If Group = 3	CH12							
		If Group = 4	CH16							

Address			ATI Compensation Setting (ATI_C1)							
CBH	Bit		7	6	5	4	3	2	1	0
Access	Value	If Group = 0	CH1							
R/W		If Group = 1	CH5							
		If Group = 2	CH9							
		If Group = 3	CH13							
		If Group = 4	CH17							

Address			ATI Compensation Setting (ATI_C2)							
CCH	Bit		7	6	5	4	3	2	1	0
Access	Value	If Group = 0	CH2							
R/W		If Group = 1	CH6							
		If Group = 2	CH10							
		If Group = 3	CH14							
		If Group = 4	CH18							

Address			ATI Compensation Setting (ATI_C3)							
CDH	Bit		7	6	5	4	3	2	1	0
Access	Value	If Group = 0	CH3							
R/W		If Group = 1	CH7							
		If Group = 2	CH11							
		If Group = 3	CH15							
		If Group = 4	CH19							

Address		Shield Settings (SHLD_SETTINGS)							
CEH	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	~	SHLD2	SHLD1	SHLD0
R/W	Default	0	0	0	0	0	0	0	0

* Note this byte will be ignored if SHIELD_EN (PROX_SETTINGS_2<6>) is set (ie if automated shield is selected).

Address		Unused (keep 00H)							
CFH	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	0	0	0	0	0	0
R/W	Default	~	~	0	0	0	0	0	0

Address		Cx Configuration (CX_CONFIG)							
D0H	Bit	7	6	5	4	3	2	1	0
Access	Name	CX_GPIO_1	CX_GPIO_0	~	~	Prox Mode Group Selection			
						GROUP4	GROUP3	GROUP2	GROUP1
R/W	Default	0	0	~	~	1	1	1	1

Address		DEFAULT_COMMS_POINTER							
D1H	Bit	7	6	5	4	3	2	1	0
Access	Default	10H (Beginning of Device Specific Data)							
R/W									

Address		Individual Channel Disable (CHAN_ACTIVE0)							
D2H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH3	CH2	CH1	CH0
R/W	Default	~	~	~	~	0	0	1	1

*Note: Only group 0 and 1 are default on, this is because with more than 2 channels active, the AC Filter isn't sampled at the optimal frequency, and thus is less effective.

Address		Individual Channel Disable (CHAN_ACTIVE1)							
D3H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH7	CH6	CH5	CH4
R/W	Default	~	~	~	~	1	1	1	1

Address		Individual Channel Disable (CHAN_ACTIVE2)							
D4H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH11	CH10	CH9	CH8
R/W	Default	~	~	~	~	1	1	1	1

Address		Individual Channel Disable (CHAN_ACTIVE3)							
D5H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH15	CH14	CH13	CH12
R/W	Default	~	~	~	~	1	1	1	1

Address		Individual Channel Disable (CHAN_ACTIVE4)							
D6H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH19	CH18	CH17	CH16
R/W	Default	~	~	~	~	1	1	1	1

Address		Individual Channel Reseed (CHAN_RESEED0)							
D7H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH3	CH2	CH1	CH0
R/W	Default	~	~	~	~	0	0	1	1

Address		Individual Channel Reseed (CHAN_RESEED1)							
D8H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH7	CH6	CH5	CH4
R/W	Default	~	~	~	~	1	1	1	1

Address		Individual Channel Reseed (CHAN_RESEED2)							
D9H	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH11	CH10	CH9	CH8
R/W	Default	~	~	~	~	1	1	1	1

Address		Individual Channel Reseed (CHAN_RESEED3)							
DAH	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH15	CH14	CH13	CH12
R/W	Default	~	~	~	~	1	1	1	1

Address		Individual Channel Reseed (CHAN_RESEED4)							
DBH	Bit	7	6	5	4	3	2	1	0
Access	Name	~	~	~	~	CH19	CH18	CH17	CH16
R/W	Default	~	~	~	~	1	1	1	1

Address		Auto ATI Target							
DCH	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (HIGH Byte)							
R/W	Default	04H (giving a Target value of = 1024 decimal)							

Address		Auto ATI Target							
DDH	Bit	7	6	5	4	3	2	1	0
Access	Value	Variable (LOW Byte)							
R/W	Default	00H (giving a Target value of = 1024 decimal)							

Address		I/O Port							
DEH	Bit	7	6	5	4	3	2	1	0
Access	Name	GPIO_7	GPIO_6	GPIO_5	GPIO_4	GPIO_3	GPIO_2	GPIO_1	GPIO_0
R/W		I/O's can be read, or set/cleared here.							

Address		I/O Tris							
DFH	Bit	7	6	5	4	3	2	1	0
Access	Name	GPIO_7	GPIO_6	GPIO_5	GPIO_4	GPIO_3	GPIO_2	GPIO_1	GPIO_0
R/W	Default	1	1	1	1	1	1	1	1

1.3 Memory Map Description

1.3.1 Device Information

Product Number

The product number for the IQS316 is 27 (decimal).

Version Number

The version number of the device ROM can be read in this byte.

1.3.2 Device Specific Data

XY Info1 (UI_FLAGS0)]

Bit 7: ***SHOW_RESET****:* This bit can be read to determine whether a reset occurred on the device since the ***ACK_RESET*** bit has been set. The value of ***SHOW_RESET*** can be set to '0' by writing a '1' in the ***ACK_RESET*** bit in the **PROX_SETTINGS_2** byte.

0 = No reset has occurred since last cleared

1 = Reset has occurred

Bit 6: ***MODE_INDICATOR****:* Indicates current mode of charging

0 = Currently in Prox Mode

1 = Currently in Touch Mode

Bit 5:3: ***Unused***

Bit 2: ***ATI_BUSY:*** Status of automated ATI routine

0 = Auto ATI is not busy

1 = Auto ATI in progress

Bit 1: ***RESEED_BUSY:*** Global Channel Reseed Status

0 = Reseed is not busy

1 = Reseed is currently taking place

Bit 0: ***NOISE:*** This bit indicates the presence of noise interference.

0 = Current cycle has not detected the presence of noise

1 = Current cycle has detected the presence of noise

1.3.3 Proximity Status Bytes

Proximity Status

The proximity status of the channels relating to the current group can be read here. The current group can be determined by reading the **Group Number** register. The channels and group numbers relate as shown in Table 1.2.

Table 1.2 Channel data available

Current Group Number	Channels available
0	CH0 / CH1 / CH2 / CH3
1	CH4 / CH5 / CH6 / CH7
2	CH8 / CH9 / CH10 / CH11
3	CH12 / CH13 / CH14 / CH15
4	CH16 / CH17 / CH18 / CH19

1.3.4 Touch Status Bytes

Touch Status

The touch status of the channels relating to the current group can be read here. The current group can be determined by reading the **Group Number** register. The channels and group numbers relate as shown in Table 1.2.

1.3.5 Halt Bytes

Halt Status

The halt status of the channels relating to the current group can be read here. The current group can be determined by reading the **Group Number** register. The channels and group numbers relate as shown in Table 1.2.

1.3.6 Active Bytes

Group Number

The group number that can be read in this byte indicates which group's data is currently available. Group 0 is the Prox Mode group, and Group 1-4 are the Touch Mode groups.

1.3.7 Current Samples

The Current Samples for the current group can be read in their respective addresses. The HIGH bytes and LOW bytes are found in consecutive addresses.

1.3.8 Long-Term Averages & Touch/Prox Thresholds

The LTA values for the current group can be read in their respective addresses. The HIGH bytes and LOW bytes are found in consecutive addresses.

The first four bits (high nibble) of each LTA HIGH Byte is the Prox and Touch Thresholds for the respective channel. Care must be taken when overwriting a LTA that the required settings are also included in the HIGH byte.

LTA HIGH Byte

Bit 7-6: ***Touch Threshold:*** The value of these two bits, together with the global Touch Range bit determines the Touch Threshold, as shown in Table 1.3.

Bit 5-4: ***Prox Threshold:*** The value of these two bits, together with the Prox Range bit determines the Prox Threshold, as shown in 0.

Bit 3-0: ***LTA<11:8>:*** The upper 4 bits of the LTA.

Table 1.3 Touch Threshold Values

Touch Threshold <1:0>	TOUCH_RANGE = 0	TOUCH_RANGE = 1
	Touch Threshold:	
00	1/32	4/16
01	1/16	6/16
10	2/16	8/16
11	3/16	10/16

Table 1.4 Prox Threshold Values

Prox Threshold <1:0>	PROX_RANGE = 0	PROX_RANGE = 1
	Prox Threshold	
00	2	8
01	3	16
10	4	20
11	6	30

LTA LOW Byte

Bit 7-0: ***LTA<7:0>:*** The lower byte of the LTA.

1.3.9 Device Settings

UI Settings 0 (UI_SETTINGS0)

Bit 7: ***RESEED:*** Reseed the LTA filter. This can be used to adapt to an abrupt environment change, where the filter is too slow to track this change. Note that with the Short and Long Halt selections, an automatic Reseed will be performed when the halt time has expired, thus automatically adjusting to the new surroundings.

0 : Do not reseed

1 : Reseed (this is a global reseed)

Bit 6: ***ATI_MODE:*** This selects which mode to perform the auto ATI routine on, and the ***AUTO_ATI*** enable bit initiates the routine.

0 : Automated ATI will apply to Prox-Mode channels

1 : Automated ATI will apply to Touch-Mode channels

Bit 5: ***PROX RANGE*****:** Selects between two Prox threshold sets. The range is a global setting and applies to all channels; whereby each channel can then individually be setup to a custom threshold value within this selected range.

0 = Lower range threshold set

1 = Higher range threshold set

Bit 4: ***TOUCH RANGE*****:** Selects between two touch threshold sets. The range is a global setting and applies to all channels; whereby each channel can then individually be setup to a custom threshold value within this selected range.

0 = Lower range threshold set

1 = Higher range threshold set

Bit 3: ***FORCE PROX MODE*****:** Force charging to Prox Mode. If this bit is set, automatic transitions between Prox and Touch Mode are overwritten.

0 = Normal Operation

1 = Only Prox Mode charging

Bit 2: ***FORCE TOUCH MODE*****:** Force charging to Touch Mode. If this bit is set, automatic transitions between Prox and Touch Mode are overwritten. Note: this bit takes precedence over Bit3.

0 = Normal Operation

1 = Only Touch Mode charging

Bit 1: ***ND*****:** Noise Detection Enable. This setting is used to enable the on-chip noise detection circuitry. With noise detected, the noise affected samples will be ignored, and have no effect on the Prox, touch or LTA calculations. The ***NOISE*** bit will appropriately be set as indication of the noise status.

0 = Disable noise detection

1 = Enable noise detection

Bit 0: ***Internal:*** This bit should always keep the value of 0

Power Settings (POWER_SETTINGS)

Bit 7:4:	***Unused***
Bit 3:	***SLEEP*: This bit puts the IC in SLEEP mode. Sleep is entered after termination of the communication window. No processing is done in the sleep state. This function is available in both SPI and I^{2}C. In SPI, to wake the device from sleep, the /SS line is pulled low, thus selecting the device, whereby waking it from sleep. Communication with the device is then immediately resumed. In I^{2}C, to wake the device, the master simply is required to begin communication with the device. In both cases, when the IC is woken from sleep, the firmware returns to the same communication window that was last used to put the device to sleep, thus no new sample data is available. Note that if the IC has been in SLEEP for a considerable time, it is recommended to reseed the channels, if no interaction is assumed. 0 : No effect 1 : Puts device in sleep mode
Bit 2:	***MAIN_OSC*: Select the frequency of the main oscillator 0 = 8MHz 1 = 4MHz (not recommended)
Bit 1-0:	***LP*: Low Power (LP) options 00 = Normal Power 01 = LP1 ~100ms charging 10 = LP2 ~ 200ms charging 11 = LP3 ~ 300ms charging

ProxSense Module Settings 1 (PROX_SETTINGS_1)

Bit 7:	***CXVSS*: Ground Cx channels when inactive. The default and recommended setting is grounded. The result is illustrated by means of an example. If for instance Group 1 is charging, all surrounding sensing lines not part of Group 1 are grounded, and thus in a defined state. If the Cx's are set to float, then their state is unknown, and the sensors influence each other greatly, which is not ideal. 0 = Cx's float 1 = Cx's grounded
Bit 6:	***ZC_EN*: Enable zero-cross (ZC) triggered conversions. An input signal must be connected to the ZC_IN I/O to synchronise the charging to. This is occasionally used in high AC noise applications, whereby synchronising the charging to the AC, the noise is reduced. This input allows the timing of the conversions to be accurately controlled. Possibly the conversions can be sliced between noise events to keep the samples noise free. 0 = No Zero-Cross signal implemented

1 = Conversions synchronised to ZC_IN

Bit 5-4:	***HALT*: LTA Filter Halt selections**
	00 = Short (LTA filter halts for ~20 seconds, then reseeds)
	01 = Long (LTA filter halts for ~40 seconds, then reseeds)
	10 = Never (LTA filter never halts)
	11 = Always (LTA filter is halted permanently)
Bit 3:	***AUTO_ATI***: Enable the automated ATI routine. By enabling this bit, the device will perform an automated ATI routine on the selected groups (selected by ATI_MODE), and will attempt to reach the target setup in AUTO-ATI Target
	0 = No action
	1 = Begin auto ATI routine
Bit 2-0:	***CXDIV[2:0]***: Selection bits for charge transfer frequency

Table 1.5 Charge transfer frequency

MAIN_OSC = 4MHz		MAIN_OSC = 8MHz	
CXDIV	**Conversion Frequency**	**CXDIV**	**Conversion Frequency**
000	2MHz	000	4MHz
001	1MHz	001	2MHz
010	500kHz	010	1MHz
011	250kHz	011	500kHz
100	125kHz	100	250kHz
101-111	62.5kHz	101-111	125kHz

The charge transfer frequency is a very important parameter. Dependant on the design application, the device frequency must be optimised. For example, if keys are to be used in an environment where steam or water droplets could form on the keys, a higher transfer frequency improves immunity. Also, if a sensor antenna is a very large object/size, then a slower frequency must be selected since the capacitance of the sensor is large, and a slower frequency is required to allow effective capacitive sensing on the sensor.

ProxSense Module Settings 2 (PROX_SETTINGS_2)

Bit 7:	***Unused***
Bit 6:	***SHIELD_EN***: Automatic shield implementation. Each group will have a shield setup automatically on the two shield outputs, according to Table 1.6.

Table 1.6 Automated Shield Channels

Group	SHLD_A	SHLD_B
0	CxA0	CxB0
1	CxA0	CxB0
2	CxA1	CxB1
3	CxA2	CxB2
4	CxA3	CxB3

0 = Shield is set by SHIELD_SETTING byte

1 = Shield is automatically loaded according to Table 1.6

Bit 5: ***STOP_COMMS*****:** Skip the SPI/I^{2}C communication window. This can be used if the master does not want to service the IQS316 every charge cycle. Normal operation of the IC continues, and only the communication window is bypassed. Only when the master initiates, or when a Prox is sensed, will the communication be resumed.

0 = Normal Communication

1 = Communication aborted until Prox detected, or master forces a resume

Bit 4: ***ACK_RESET*****:** Acknowledge 'SHOW_RESET'.

0 = Nothing

1 = Clear the flag **SHOW_RESET** (send only once)

Bit 3: ***SKIP_CONV*****:** Don't perform conversion. This can be used, for example if settings for all the groups are to be written. The current groups' settings can be completed, and the communication window can then be terminated. The device then loads the next groups' data (without performing a conversion), and the next communication window is available. Stepping through all the groups can this be done without the need to wait for a conversion to complete.

0 = Normal operation

1 = Skip conversions (Load next groups data and return to communication window)

Bit 2: ***ACF_DISABLE*****:** Disable the AC Filter on Group 0.

0 = AC Filter is enabled

1 = AC Filter is disabled

Bit 1: ***LTN_DISABLE*****:** Disable the LTN Filter on Group 0.

0 = LTN Filter is enabled

1 = LTN Filter is disabled (recommended due to device limitation)

Bit 0: ***WDT_DISABLE*****:** Device watchdog timer (WDT) disable.

0 = Enabled

1 = Disabled

ATI Multiplier C (ATI_MULT1)

The ATI Multiplier and ATI Compensation bits allow the controller to be compatible with a large range of sensors, and in many applications with different environments. ATI allows the user to maintain a specific sample value on all channels. The ATI Multiplier parameters would produce the largest changes in sample values and can be thought of as the high bits of ATI. The ATI Compensation bits are used to influence the sample values on a smaller scale to provide precision when balancing all channels as close as possible to the target. The ATI Multiplier parameters are further grouped into two parameters namely ATI Multiplier C and ATI Multiplier I. ATI multiplier I consists of a single bit and has the biggest effect on the sample value and can be considered as the highest bit of the ATI parameters.

The ATI_MULT1 byte contains the ATI Multipliers C settings for all channels of the current group. Each channel has two ATI Multiplier C bits where the value of '11' would provide the highest CS value and the value of '00' would provide the lowest.

ATI Multiplier I (ATI_MULT2)

The ATI Multiplier I bit is the ATI bit which would make the largest adjustment to the sample value. The ATI_MULT3 byte contains the ATI Multiplier I settings for all the channels in the current group, where a value of '1' would produce the largest sample value and a value of '0' would produce the smallest sample value.

ATI Compensation Settings

The ATI Compensation parameter can be configured for each channel in a range between 0-255 (decimal). The ATI compensation bits can be used to make small adjustments of the sample values of the individual channels.

Shield Settings (SHLD_SETTINGS)

If the **SHIELD_EN** bit is set, the value written to the **SHLD_SETTINGS** register is simply ignored. Otherwise the shield can be manually configured here.

The **SHLD_SETTINGS** byte is used to enable or disable the two active shields. Bit 0-2 control which sensor lines are to be shielded on **SHLD_A** and **SHLD_B**. By default the shields are disabled with **SHLD_SETTINGS** = 0. Manual configuration is implemented as shown in Table 1.7.

Table 1.7 SHLD_A and SHLD_B configuration

SHLD_SETTINGS<2:0>	SHLD_A input connected to	SHLD_B input connected to
000	SHLDL off	SHLDR off
001	CxA6	CxB6
010	CxA5	CxB5
011	CxA4	CxB4
100	CxA3	CxB3
101	CxA2	CxB2
110	CxA1	CxB1
111	CxA0	CxB0

Cx Configuration (CX_CONFIG)

Bit 7: ***CX_GPIO_1*:** Cx or I/O selection.

0 = CxA7, CxA6, CxB7 and CxB6 are used as sensor lines

1 = GPIO_7, GPIO_6, GPIO_5 and GPIO_4 are implemented as I/O’s

Table 1.8 Upper Nibble of I/O Port Selection

CX_GPIO_1 Selection	CxA7 / GPIO_7 function	CxA6 / GPIO_6 function	CxB7 / GPIO_5 function	CxB6 / GPIO_4 function
0	CxA7	CxA6	CxB7	CxB6
1	GPIO_7	GPIO_6	GPIO_5	GPIO_4

Bit 6: ***CX_GPIO_0*:** Cx or I/O selection.

0 = CxA5, CxA4, CxB5 and CxB4 are used as sensor lines

1 = GPIO_3, GPIO_2, GPIO_1 and GPIO_0 are implemented as I/O’s

Table 1.9 Lower Nibble of I/O Port Selection

CX_GPIO_1 Selection	CxA5 / GPIO_3 function	CxA4 / GPIO_2 function	CxB5 / GPIO_1 function	CxB4 / GPIO_0 function
0	CxA5	CxA4	CxB5	CxB4
1	GPIO_3	GPIO_2	GPIO_1	GPIO_0

Bit 3-0: ***PM_CX_SELECT:*** Groups who's Cx's are included in Prox Mode charging

0 = Group included

1 = Group not included

In this register, a selection of groups 4-1 is made to determine which sensor lines will be used during Prox Mode charging (Group 0). Each bit therefore represents four sensor lines to be added or removed from Group 0.

*Note that at least two groups have to be set for this selection.

Table 1.10 Sensor Line Selection for Prox Mode

CX_CONFIG bit	CH0	CH1	CH2	CH3
0 (Group 1 channels)	CxA0	CxB0	CxA4	CxB4
1 (Group 2 channels)	CxA1	CxB1	CxA5	CxB5
2 (Group 3 channels)	CxA2	CxB2	CxA6	CxB6
3 (Group 4 channels)	CxA3	CxB3	CxA7	CxB7

To help illustrate this, an example is provided. If bit 0 and 2 are set in CX_CONFIG, the channels used in Prox Mode are shown in Table 1.11. It can be seen that the Proximity channel 0 (CH0) consists of the two sensor lines CxA0, and CxA2. And similarly the CH1 to CH3's sensor lines can be noted. This example thus has 8 of the 16 sensor lines also providing Proximity input. The other 8 have no influence on the Prox Mode channels.

Table 1.11 PM_CX_SELECT example

CX_CONFIG	CH0	CH1	CH2	CH3
CX_CONFIG= 04H	CxA0, CxA2	CxB0, CxB2	CxA4, CxA6	CxB4, CxB6

It can be seen that if all 4 bits are set, all 16 of the Cx sensor lines are antenna inputs for the Prox Mode. It is recommended that if the design has any sensor buttons close to noise sources (negative influence on proximity), that these can be chosen to fall in the same group, which can then be excluded from Prox Mode by means of the PM_CX_SELECT register.

Default Comms Pointer

The value stored in this register will be loaded into the Comms Pointer at the start of a communication window. For example, if the design only requires the Proximity Status information each cycle, then the *Default Comms Pointer* can be set to ADDRESS '31H'. This would mean that at the start of each communication window, the comms pointer would already be set to the Proximity Status register, simply allowing a READ to retrieve the data, without the need of setting up the address.

Individual Channel Disable

Each channel can be individually disabled in these registers. Note that the current group number has no influence on these registers as each channel disable register has a unique address.

Individual Channel Reseed

Each channel can be individually reseeded in these registers. Note that the current group number has no influence on these registers as each channel reseed register has a unique address.

Auto-ATI Target

The automated ATI target can be set in these two consecutive registers. These registers are used for the Prox Mode, as well as the Touch Mode ATI targets. The selection between which of these modes to Auto-ATI, is set by **ATI_MODE** in **UI_SETTINGS0<6**>.

For example, if the Prox Mode channels must be tuned to sample values = 800, and the Touch Mode channels to sample values = 400, the following steps are taken:

Step 1:	Set *Auto ATI Target* to 800
Step 2:	Select Prox Mode for ATI by clearing ATI_MODE bit (UI_SETTINGS0<6> = 0)
Step 3:	Start Auto-ATI procedure by setting AUTO-ATI bit (PROX_SETTINGS<3>)
Step 4:	Wait for Prox Mode ATI to complete, which is when ATI_BUSY bit clears (UI_FLAGS0<2> = 0)
Step 5:	Set *Auto ATI Target* to 400
Step 6:	Select Touch Mode for ATI by setting ATI_MODE bit (UI_SETTINGS0<6> = 1)
Step 7:	Start Auto-ATI procedure by setting AUTO-ATI bit (PROX_SETTINGS<3>)

I/O Port and Tris

When setup to be used as I/O's (CX_GPIO_1 and CX_GPIO_0 settings), the data direction can be set in the **I/O Tris** register as shown in Table 1.12.

Table 1.12 Tris Configuration

Tris bit <7:0>	I/O configuration
0	Output
1	Input / Tri-state

If setup as outputs, the state of the I/O's can then be set in the register as shown in Table 1.13

Table 1.13 I/O Outputs

Port bit <7:0>	**I/O status**
0	Output LOW
1	Output HIGH

If setup as inputs, the status of the I/O's can be read from the register.

2 General Implementation hints

When implementing the communication interface with the IQS316, please refer to the IQS316 datasheet for a detailed description of the SPI and I^{2}C communication. This section contains some general guidelines and hints regarding the communication interface.

2.1 Communication window

Upon implementing either SPI or I^{2}C it is important to note the difference in the working of the communication window.

2.1.1 SPI Communication window

When communicating via SPI, the communication window will remain open until a new conversion command is received (FE written to IQS316 in 'address' time-slot). While the communication window is open the master may initiate and terminate as many read and write communication sessions as required.

2.1.2 I^{2}C Communication window

When communicating via I^{2}C, the communication window will automatically close when a STOP bit is received by the IQS316. The IQS316 will then proceed to start with a new conversion and the READY line will be pulled low until the new conversion is complete.

Note that there is no command via I^{2}C to initiate a new conversion. To perform multiple read and write commands, the repeated start function of the I^{2}C must be used to stack the commands together.

2.2 Startup Procedure

After sending initial settings to the IQS316, it is important to execute a reseed. It is suggested to execute an estimated 24 conversions after initial settings before calling for a reseed, to allow the system to stabilize. This translates to a startup time of approximately 350ms.

2.3 General I^{2}C Hints

2.3.1 I^{2}C Pull-up resistors

When implementing I^{2}C it is important to remember the pull-up resistors on the data and clock lines. 4.7kΩ is recommended, but for lower clock speeds bigger pull-ups will reduce power consumption.

2.3.2 MCLR

Suggested implementation is to have the IQS316 and the pull-up resistors connect to the power supply of the device. The MCLR pin should then be used to reset the IQS316. Remember to hold the MCLR low until master setup has been done.

2.3.3 Reset Device while using I^{2}C

When a reset occurs, some care needs to be taken to ensure that the IQS316 restarts correctly. The reset pin needs to be LOW before the IQS316 can be initialised, else the master will read a ready signal prematurely. To accomplish this without any delays, define the ready pin on the master as an output and pull it LOW. Then, redefine it as an input line just before initializing the IQS316.

3 Sample implementation

A minimalist implementation of the IQS316 is described in this section. This implementation sets the thresholds of the IQS316 and retrieves Prox and touch data. For this implementation a PIC18F4550 was used as the master device. The data may be used as the user prefers, an output section is therefore not included in this section.

Communication between the master and the IQS316 was done in SPI. Libraries for I^{2}C are also available. For an explanation of the I^{2}C protocol, refer to the IQS316 datasheet.

3.1 Overview

This implementation initiates communication between the master (PIC18F4550) and the IQS316. The master sends commands to configure the IQS316. Once the configuration is completed, the program enters an infinite loop.

In each loop cycle:

- A command is sent to the IQS316 to start new conversion.
- The master waits until the conversion is completed. (Ready signal)
- Data is read from IQS316.
- If a PROX or TOUCH is registered, a routine is executed to process the data retrieved.
- Additional communication with IQS316 if required (sending new commands).

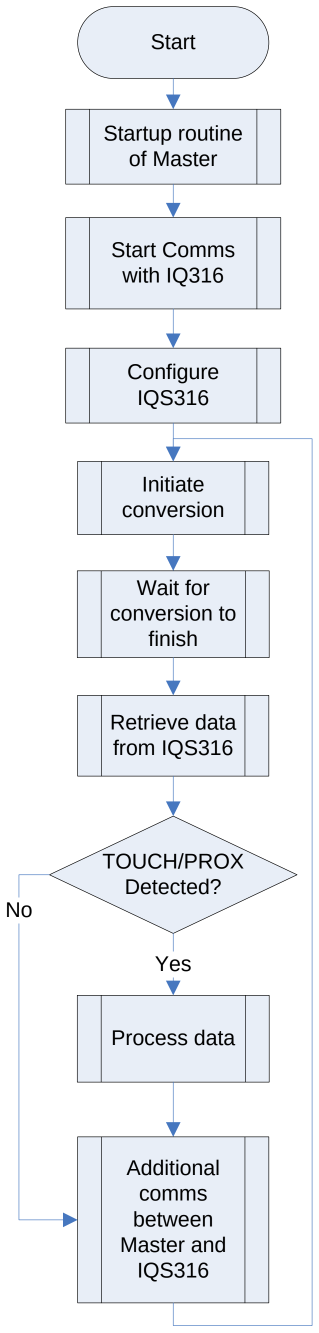
Start
Startup routine of Master
Start Comms with IQ316
Configure IQS316
Initiate conversion
Wait for conversion to finish
Retrieve data from IQS316
TOUCH/PROX Detected?
No
Yes
Process data
Additional comms between Master and IQS316

Figure 3.1 Overview

3.1.1 Communications:

The master initiates communication with the IQS316. For a detailed description of the communication protocol refer to the IQS316 datasheet.

Writing to IQS316: The master initiates communication by writing a zero (00H) to the IQS316. Next the address to write to is sent to the IQS316. The byte sent after the address will be written to that address.

Another address can now be sent to the IQS316. Communication is terminated by sending FFH instead of an address.

E.g. write 35H to address 12H:

Master writes 00H to IQS316. (Initiates comms in write mode, FFH returned)

Master writes 12H to IQS316. (Setup address, returns 00H)

Master writes 35H to IQS316. (35H stored at address 12H, 01H returned)

Master writes FFH to IQS316. (end write cycle, 00H returned)

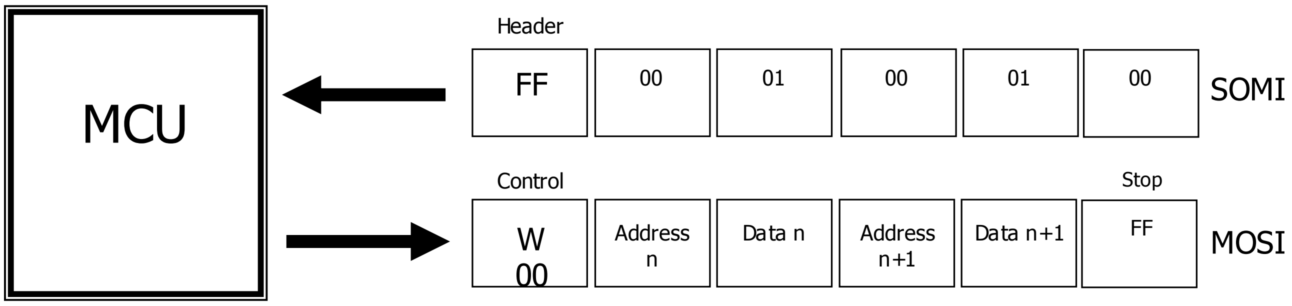

Figure 3.2 SPI write sequence

Additionally, if the master writes FEH to the IQS316, a new conversion will be initiated and the communication window will be terminated.

Reading from the IQS316: The master initiates communication by writing a one (01H) to the IQS316. During each communication cycle (one byte transmitted and received) the data stored at the location indicated by the address pointer will be sent to the master. The address pointer value in turn is replaced by the data sent to the IQS316 by the master. However, upon receiving FEH from the master the address pointer is simply incremented. The default value of the address pointer is 10H. The master ends this transfer by writing FFH to the IQS316.

E.g. read address 15H and 16H:

Master writes 01H to IQS316. (Initiates comms in read mode, FFH returned)

Master writes 15H to IQS316. (set pointer to 15H, data stored at current pointer address returned)

Master writes FEH to IQS316 (pointer incremented, data stored at 15H returned)

Master writes FFH to IQS316. (end read cycle, data stored at 16H returned)

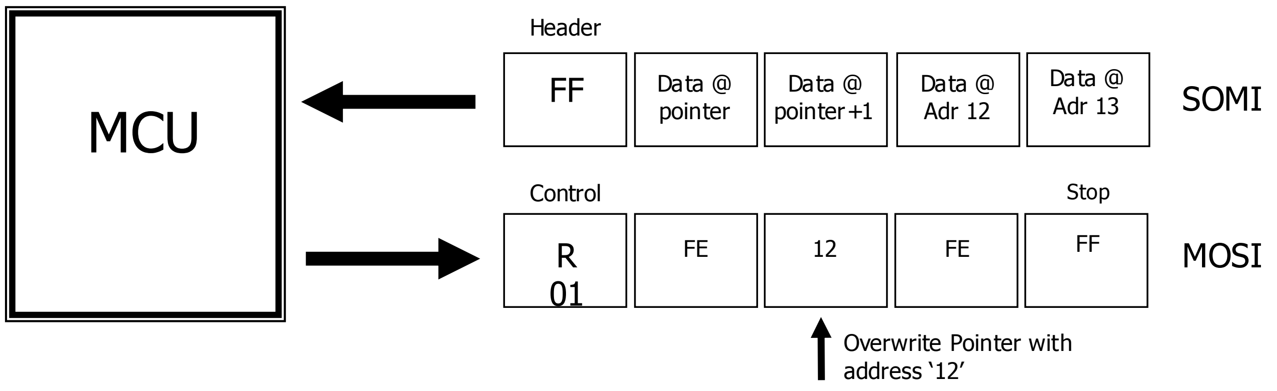

Figure 3.3 SPI read sequence

Please Note: The internal address pointer is only reset to the default value (10H) when a new conversion is called. It is not reset when switching between read and write routines.

3.1.2 Data Retrieval:

The IQS316 continuously checks group 0 until a PROX is detected. The IQS316 now checks groups 1-4 for PROX/TOUCH conditions. These are stored in the status registers during each group. If no TOUCH or PROX conditions occur for ~4 seconds, the IQS316 returns to only checking group 0. Group 1-4 are occasionally charged to keep the LTAs updated.

For a detailed explanation of the working of the different Charge Transfer Modes of Operation, refer to the ProxSense Module section of the datasheet.

To retrieve data from the IQS316, the group number has to be checked first. If the group number is zero, no PROX/TOUCH has been detected. Otherwise, all four group numbers has to be checked for the various PROX/TOUCH conditions. New conversions have to be initialized to obtain results from each group.

The following flow diagram illustrates an example of a data retrieving algorithm:

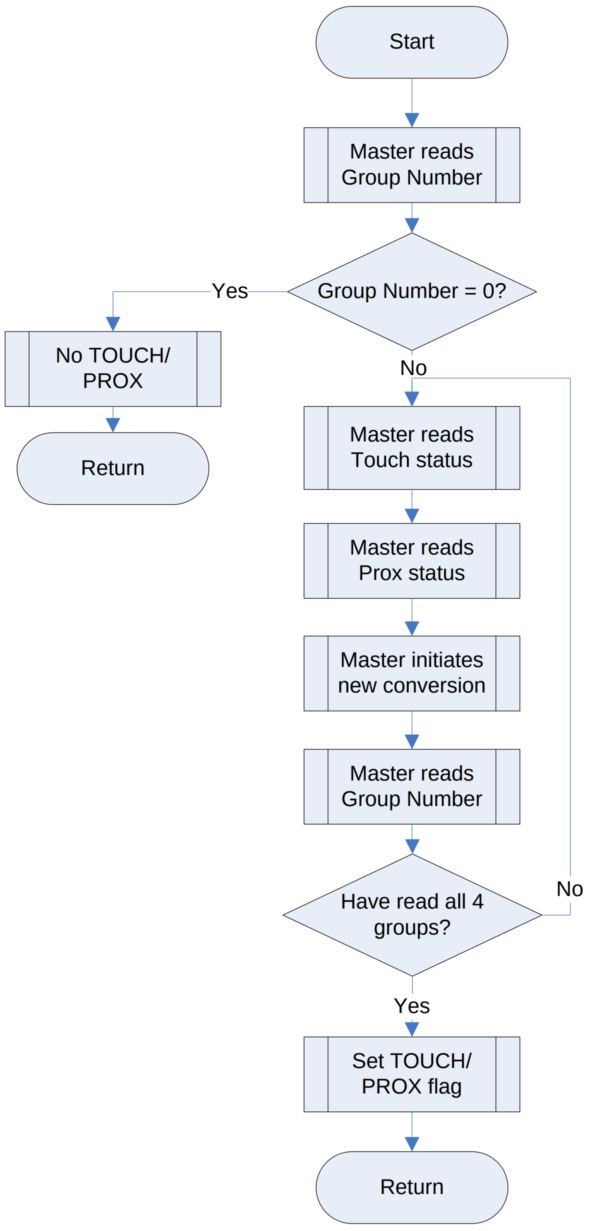

Figure 3.4 Data Retrieval

3.1.3 Data Processing:

If no TOUCH/PROX has been detected, no data processing needs to be done. If the TOUCH/PROX flag is set during the data retrieval routine, a data processing routine may be called.

This routine will read the PROX and TOUCH data of the 16 channels, as stored in a global data structure by the data retrieval routine and perform the required routines.

(This will depend on the specific application, whether to switch on LEDs, to perform calculations, etc.)

3.2 Functions

3.2.1 Main Function

The Main function sets up the hardware by writing all required initialization data to the controller. After initialization the function runs the infinite loop to retrieve data from the IQS316 and to process the data in the case that a Prox is detected.

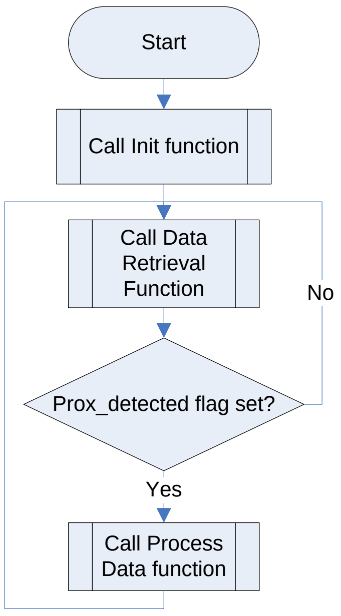

Figure 3.5 Main Function

3.2.2 Init

The init function executes commands setting up the system. The ports and registers of the PIC18F4550 are set first. The output pin connected to the IQS316 MCLR is set to high. The setup of the SPI is done by calling the Comms_init function. Once the SPI communication is initialized, commands are sent to the IQS316 via the SPI to set up the IQS316. This is done by calling the IQS316_Settings function. It is advised to execute all other hardware initialization routines before initializing the IQS316, as other hardware may cause environmental conditions for the IQS316 to change.

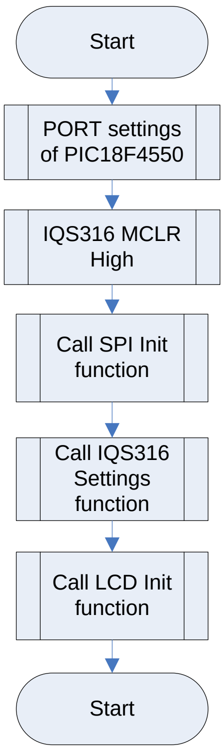

Figure 3.6 Init Function

3.2.3 Comms_init

The Comms_Init function sets the registers in the PIC18F4550 to start the SPI communication.

Once the IQS316 MCLR is released, the Comms_init function performs a short routine to ensure that the IQS316 is at the start of a communication session.

3.2.4 CommsIQS316_send

The CommsIQS316_send function is a basic function called by all other SPI communication functions. This function waits for the ready signal from the IQS316 before starting the communication. Next, the slave select (SS) line to the IQS316 is pulled LOW. The communication is done by clearing the interrupt flags and writing the data to be sent to the IQS316 in the data buffer. A loop waits for the communication to be completed and the received data is read from the data buffer and returned by the function.

3.2.5 CommsIQS316_Start_Write

The CommsIQS316_Start_Write function sends a Zero (00H) to the IQS316 to initialize a communication session, indicating that data will be written to the IQS316.

3.2.6 CommsIQS316_Start_Read

The CommsIQS316_Start_Read function sends a One (01H) to the IQS316 to initialize a communication session, indicating that data will be read from the IQS316.

3.2.7 CommsIQS316_Write

The CommsIQS316_Write function requires an address and a data byte as parameters. The address is sent to the IQS316, after which the data byte is also sent, using of the CommsIQS316_send function.

Note that the SPI channel has to be initialized with a CommsIQS316_Start_Write command in order for this function to execute correctly. Once communication is in write mode however, multiple CommsIQS316_Write commands may be executed consecutively.

3.2.8 CommsIQS316_Read

The CommsIQS316_Read function requires an address from which to read as parameter. The address is sent to the IQS316, which is placed in the Address Pointer. Next, FEH is sent to the IQS316. The byte returned during the cycle during which the FEH is sent, will be the data stored at the location indicated by the Address Pointer, which will be the address sent as parameter. This byte is returned to the master.

Note that the SPI channel has to be initialized with a CommsIQS316_Start_Read command in order for this function to execute correctly. Once communication is in read mode, the CommsIQS316_Read and CommsIQS316_Read_Next functions may be executed multiple times.

3.2.9 CommsIQS316_Read_Next

The CommsIQS316_Read_Next function sends FEH to the IQS316 by means of the CommsIQS316_send function. The data returned to the SPI buffer is returned. This function will read the data stored in the address indicated by the Address Pointer.

Note that the Address Pointer is automatically incremented after each read operation, unless it was set to a specific value during that read operation. The default value of the Address Pointer is 10H and the Address Pointer will reset to this value every time that the SPI communication channel is terminated and initiated.

3.2.10 CommsIQS316_Terminate

The CommsIQS316_terminate function sends FFH to the IQS316, which terminates the SPI communication with the IQS316. The IQS316 will read the next byte received to determine whether the next communication session will be in read or write mode.

3.2.11 CommsIQS316_Initiate_Conversion

A 00H and FEH are sent to the IQS316 consecutively. If no SPI communication session was active, this will send the command to the IQS316 to start with the next conversion of data, hence calculating new data.

3.2.12 Comms_Error

The Comms_Error function can be called from any of the SPI functions if an unexpected value is received. During developmental stages, this function may be used to indicate that an error has occurred during communication. In final stages it would probably be preferred to simply restart the system in the case that an error is detected.

3.2.13 IQS316_Settings

The IQS316_Settings function sends the values to the IQS316 to set the registers necessary for setting up the proper working of the IQS316.

The IQS316 does not require much configuration except for the thresholds. Each of the 16 channels has individual TOUCH and PROX thresholds. The thresholds can only be set while their respective groups are active.

To quickly move through all 4 groups, the SKIP_CONV bit is set. By initiating a new conversion the IQS316 now switches to the next group without calculating PROX and TOUCH data. After all the thresholds have been set, the SKIP_CONV bit is cleared again to allow for normal operation again.

A reseed is called and 24 new conversions are called to allow the system to settle before returning to the main function.

3.2.14 IQS316_New_Conversion

The IQS316_New_Conversion function retrieves data from the IQS316 after a conversion has been completed. The data is stored in the IQS316 global data structure to be easily accessible from anywhere in the program.

This function checks whether the IQS316 is in Prox mode or in touch mode by reading the group number. In the case that the group number is 0, the prox_detected flag is cleared and the routine exits.

If the group number is not 0, then the prox_detected flag is set and the PROX and TOUCH data of all the channels are retrieved and stored in the global data structure.

3.3 Sample Code

3.3.1 Global Variables

```
struct {
        unsigned char prox_detected;    //flag to indicate whether IQS316 is in prox mode or touch mode
        unsigned char prox4_11;                 //prox status of channels 4-11
        unsigned char prox12_19;                //prox status of channels 12-19
        unsigned char touch4_11;                //touch status of channels 4-11
        unsigned char touch12_19;               //touch status of channels 12-19
} IQS316;
```

3.3.2 Functions

```
void main(void)
{
        init();

        while(1)
        {
                IQS316_New_Conversion();
                if (IQS316.prox_detected)
                {
                        IQS316_Process_Conversion();
                }
        }
}

void init(void)
{
        ADCON1 = 0x0F;                          //PORTA all digital operation
        TRISA = 0x01;                           //RA2,1 outputs, RA0 input.
        TRISD = 0x00;                           //configure PORTD for output
        LATD = 0x0F;                            //LEDS off
        TRISB = 0xFF;                           //PORTB for input
        INTCON2bits.RBPU = 0;

        LATB = 0xFC;

        Comms_init();

/*
        Place other functions responsible for hardware initialization here.
        E.g. LCD_Init();
*/
        IQS316_Settings();                      //execute after all hardware initializations

}

void Comms_init()
{
        unsigned char temp;

        SSPCON1 = 0x31;                         //enables SSP, clock set
        TRISB = TRISB | 0x01;                   //set TRISB<0>
        TRISB = TRISB & 0xFD;                   //clear TRISB<1>
        TRISC = TRISC & 0x7F;                   //clear SDO<7>

        LATA = LATA | 0x04;                     //pull RA2 (SPI_SELECT) high
        LATA = LATA | 0x02;                     //pull RA1 (Slave select high)
        LATA = LATA | 0x08;                     //MCLR High

        temp = CommsIQS316_send(0xFF);          //sequence to ensure correct comms initialization
        while (temp != 0xFF)
        {
                temp = CommsIQS316_send(0xFF);
        }
        temp = CommsIQS316_send(0xFF);

}

unsigned char CommsIQS316_send(unsigned char send_data)
{
        unsigned char temp;

        while (PORTAbits.RA0 == 0)              //wait for ready signal
        {}
        LATA = LATA & 0xFD;                     //pull SS line low
        PIR1bits.SSPIF = 0;                     //clear flag
        temp = SSPBUF;                          //perform read
        SSPBUF = send_data;                     //initiate transmission
        while (PIR1bits.SSPIF == 0)             //wait for interrupt flag
        {}
        temp = SSPBUF;
        LATA = LATA | 0x02;                 //release SS line
        return temp;
}


void IQS316_Settings(void)
{
        unsigned char start_num, temp;

        CommsIQS316_Start_Read();           //read group number
        start_num = CommsIQS316_Read(GROUP_NUM);
        CommsIQS316_Terminate();

        CommsIQS316_Start_Write();          //set the CONV_SKIP bit
        CommsIQS316_Write(PROX_SETTINGS_2, 0x09);
        CommsIQS316_Terminate();

        temp = start_num;

        do
        {
                switch (temp)
                {
                        case 0:
                                CommsIQS316_Start_Write();
                                CommsIQS316_Write(LTA_04_HI, 0x00);
                                CommsIQS316_Write(LTA_15_HI, 0x00);
                                CommsIQS316_Write(LTA_26_HI, 0x00);
                                CommsIQS316_Write(LTA_37_HI, 0x00);
                                CommsIQS316_Terminate();
                                break;
                        case 1:
                                CommsIQS316_Start_Write();
                                CommsIQS316_Write(LTA_04_HI, 0x00);
                                CommsIQS316_Write(LTA_15_HI, 0x00);
                                CommsIQS316_Write(LTA_26_HI, 0x00);
                                CommsIQS316_Write(LTA_37_HI, 0x00);
                                CommsIQS316_Terminate();
                                break;
                        case 2:
                                CommsIQS316_Start_Write();
                                CommsIQS316_Write(LTA_04_HI, 0x00);
                                CommsIQS316_Write(LTA_15_HI, 0x00);
                                CommsIQS316_Write(LTA_26_HI, 0x00);
                                CommsIQS316_Write(LTA_37_HI, 0x00);
                                CommsIQS316_Terminate();
                                break;
                        case 3:
                                CommsIQS316_Start_Write();
                                CommsIQS316_Write(LTA_04_HI, 0x00);
                                CommsIQS316_Write(LTA_15_HI, 0x00);
                                CommsIQS316_Write(LTA_26_HI, 0x00);
                                CommsIQS316_Write(LTA_37_HI, 0x00);
                                CommsIQS316_Terminate();
                                break;
                        case 4:
                                CommsIQS316_Start_Write();
                                CommsIQS316_Write(LTA_04_HI, 0x00);
                                CommsIQS316_Write(LTA_15_HI, 0x00);
```

```
                                        CommsIQS316_Write(LTA_26_HI, 0x00);
                                        CommsIQS316_Write(LTA_37_HI, 0x00);
                                        CommsIQS316_Terminate();
                                        break;
                        }

                        CommsIQS316_Initiate_Conversion();
                        CommsIQS316_Start_Read();
                        temp = CommsIQS316_Read(GROUP_NUM);
                        CommsIQS316_Terminate();
                } while (temp != start_num);                    //ensure all groups have been set.

                CommsIQS316_Start_Write();                      //clear CONV_SKIP bit
                CommsIQS316_Write(PROX_SETTINGS_2, 0x01);
                CommsIQS316_Terminate();

                for (temp = 0; temp <= 23; temp++)
                {
                        CommsIQS316_Initiate_Conversion();
                }
                //initial conversions to allow system to settle

                CommsIQS316_Start_Write();
                CommsIQS316_Write(UI_SETTINGS0, 0xA2);                  //set to 0xB2 for high touch thresholds
                CommsIQS316_Terminate();
                CommsIQS316_Initiate_Conversion();
/*
                Place other commands to set up the IQS316 here.
*/
}

void IQS316_New_Conversion(void)
{
                unsigned char temp_num, start_num, temp_touch, temp_prox;

                CommsIQS316_Initiate_Conversion();

                CommsIQS316_Start_Read();
                temp_num = CommsIQS316_Read(GROUP_NUM);
                CommsIQS316_Terminate();

                if (temp_num == 0)
                {
                        IQS316.prox_detected = 0;                       //clear flag
                }
                else
                {
                        IQS316.prox_detected = 1;                       //set flag
                        start_num = temp_num;
                        do
                        {
                                CommsIQS316_Start_Read();
                                temp_touch = CommsIQS316_Read(TOUCH_STAT);
                                temp_prox = CommsIQS316_Read(PROX_STAT);
                                CommsIQS316_Terminate();

                                switch(temp_num)
                                {
                                        case 1:
```

```
                                        IQS316.prox4_11 = IQS316.prox4_11 & 0xF0;
                                        IQS316.touch4_11 = IQS316.touch4_11 & 0xF0;

                                        IQS316.prox4_11 = IQS316.prox4_11 | (temp_prox & 0x0F);
                                        IQS316.touch4_11 = IQS316.touch4_11 | (temp_touch & 0x0F);
                                        break;
                                case 2:
                                        IQS316.prox4_11 = IQS316.prox4_11 & 0x0F;
                                        IQS316.touch4_11 = IQS316.touch4_11 & 0x0F;

                                        IQS316.prox4_11 = IQS316.prox4_11 | ((temp_prox & 0x0F) << 4);
                                        IQS316.touch4_11 = IQS316.touch4_11 | ((temp_touch & 0x0F) <<
4);
                                        break;
                                case 3:
                                        IQS316.prox12_19 = IQS316.prox12_19 & 0xF0;
                                        IQS316.touch12_19 = IQS316.touch12_19 & 0xF0;

                                        IQS316.prox12_19 = IQS316.prox12_19 | (temp_prox & 0x0F);
                                        IQS316.touch12_19 = IQS316.touch12_19 | (temp_touch & 0x0F);
                                        break;
                                case 4:
                                        IQS316.prox12_19 = IQS316.prox12_19 & 0x0F;
                                        IQS316.touch12_19 = IQS316.touch12_19 & 0x0F;

                                        IQS316.prox12_19 = IQS316.prox12_19 | ((temp_prox & 0x0F) <<
4);
                                        IQS316.touch12_19 = IQS316.touch12_19 | ((temp_touch & 0x0F)
<< 4);
                                        break;
                        }

                        CommsIQS316_Initiate_Conversion();
                        CommsIQS316_Start_Read();
                        temp_num = CommsIQS316_Read(GROUP_NUM);
                        CommsIQS316_Terminate();

                } while ((temp_num != 0) && (temp_num != start_num));  //ensure all groups are checked

        }
}

void IQS316_Process_Conversion (void)
{
/*
                Place code here to process data in the IQS316 structure.
*/
}

void CommsIQS316_Start_Write(void)
{
        unsigned char temp;
        temp = CommsIQS316_send(0x00);          //initialize comms in write mode
        if (temp != 0xFF)                       //check for unexpected return value
        {
                Comms_Error();
        }
}
```

```
void CommsIQS316_Start_Read(void)
{
        unsigned char temp;
        temp = CommsIQS316_send(0x01);          //initialize comms in read mode
        if (temp != 0xFF)                       //check for unexpected return value
        {
                Comms_Error();
        }
}

void CommsIQS316_Write(unsigned char write_addr, unsigned char data)
{
        CommsIQS316_send(write_addr);
        CommsIQS316_send(data);
}

unsigned char CommsIQS316_Read(unsigned char read_addr)
{
        unsigned char temp;
        CommsIQS316_send(read_addr);
        temp = CommsIQS316_send(0xFE);
        return temp;
}

unsigned char CommsIQS316_Read_Next(void)
{
        unsigned char temp;
        temp = CommsIQS316_send(0xFE);
        return temp;
}

void CommsIQS316_Terminate(void)
{
        CommsIQS316_send(0xFF);
}

void CommsIQS316_Initiate_Conversion(void)
{
        unsigned char temp;
        temp = CommsIQS316_send(0x00);
        if (temp != 0xFF)
        {
                Comms_Error();
        }
        CommsIQS316_send(0xFE);
}

void Comms_Error(void)
{
        /*
                Place error routine code here
        */
        while (1)
        {}
}
```

3.3.3 Constant Declarations

```
// Addresses in the IQS316Memory Map
```

```
#define PROD_NUM                0x00
#define VERSION_NUM             0x01

#define UI_FLAGS0               0x10

#define PROX_STAT               0x31
#define TOUCH_STAT              0x35
#define HALT_STAT               0x39
#define GROUP_NUM               0x3D

#define CUR_SAM_04_HI           0x42
#define CUR_SAM_04_LO           0x43
#define CUR_SAM_15_HI           0x44
#define CUR_SAM_15_LO           0x45
#define CUR_SAM_26_HI           0x46
#define CUR_SAM_26_LO           0x47
#define CUR_SAM_37_HI           0x48
#define CUR_SAM_37_LO           0x49

#define LTA_04_HI               0x83
#define LTA_04_LO               0x84
#define LTA_15_HI               0x85
#define LTA_15_LO               0x86
#define LTA_26_HI               0x87
#define LTA_26_LO               0x88
#define LTA_37_HI               0x89
#define LTA_37_LO               0x8A

#define UI_SETTINGS0            0xC4
#define POWER_SETTINGS          0xC5
#define PROX_SETTINGS_1         0xC6
#define PROX_SETTINGS_2         0xC7
#define PROX_CFG1               0xC8
#define CMT_SETTINGS            0xC9
#define ATIC0                   0xCA
#define ATIC1                   0xCB
#define ATIC2                   0xCC
#define ATIC3                   0xCD
#define SHLD_SETTINGS           0xCE
#define INT_CAL_SETTINGS        0xCF
#define PM_CX_SELECT            0xD0
#define DEFAULT_COMMS_PTR       0xD1
#define CHAN_ACTIVE0            0xD2
#define CHAN_ACTIVE1            0xD3
#define CHAN_ACTIVE2            0xD4
#define CHAN_ACTIVE3            0xD5
#define CHAN_ACTIVE4            0xD6
#define CHAN_RESEED0            0xD7
#define CHAN_RESEED1            0xD8
#define CHAN_RESEED2            0xD9
#define CHAN_RESEED3            0xDA
#define CHAN_RESEED4            0xDB
#define AUTO_ATI_TARGET_HI      0xDC
#define AUTO_ATI_TARGET_LO      0xDD

#define DIRECT_ADDR_RW          0xFC
#define DIRECT_DATA_RW          0xFD
```

PRETORIA OFFICE
Physical Address
160 Witch Hazel Avenue
Hazel Court 1, 1st Floor
Highveld Techno Park
Centurion, Gauteng
Republic of South Africa

Tel:	+27 12 665 2880
Fax:	+27 12 665 2883

PAARL OFFICE
Physical Address
109 Main Street
Paarl
7646
Western Cape
Republic of South Africa

Tel:	+27 21 863 0033
Fax:	+27 21 863 1512

Please visit www.azoteq.com for a full portfolio of the ProxSense™ Capacitive Sensors, Datasheets, Application Notes and Evaluation Kits available.
ProxSenseSupport@azoteq.com