## **QTouch Design Guide**

Atmel provides a QTouch library for the design of capacitive sensors using the QTouch technology. Implementation in Atmel Studio requires downloading QTouch libraries from the Atmel website. This enables the user to create a QTouch Executable Project in Atmel Studio 6 and use the built-in library functions. Following is the link to register and download the QTouch Library 5.0

The basic algorithm for designing a QTouch based sensor is :

 Configure sensors as keys/rotors/sliders using : void qt\_enable\_key (channel\_t channel, aks\_group\_t aks\_group,threshold\_t detect\_threshold, hysteresis\_t detect\_hysteresis)

void qt\_enable\_rotor (channel\_t from\_channel, channel\_t to\_channel, aks\_group\_t aks\_group, threshold\_t detect\_threshold ,hysteresis\_t detect\_hysteresis, resolution\_t angle\_resolution, uint8\_t angle\_hysteresis)

void qt\_enable\_slider (channel\_t from\_channel, channel\_t to\_channel, aks\_group\_t aks\_group, threshold\_t detect\_threshold, hysteresis\_t detect\_hysteresis, resolution\_t position\_resolution, uint8\_t position\_hysteresis)

- 2) Set the touch parameters for the library like the threshold level, detect integration values that set the number of counts the signal level should differ from the reference level in order to be registered as a touch, Maximum ON duration etc.
- 3) Initialize sensing using *qt\_init\_sensing()* to calibrate the channels and prepare the sensors for capacitive touch. It is a library defined function.
- 4) Initialize the timer ISR to run periodically and determine the time for a capacitive measurement.
- 5) Repeatedly call *qt\_measure\_sensors()* for capacitive measurements and to update the status flags. Library data structure qt\_measure\_sensors() takes the current time in milliseconds as the parameter and updates 10 flags.
- 6) Check if repeated measurements are required (QTLIB\_BURST\_AGAIN flag set to 1). If required, then measure sensors again.
- 7) Check the *qt\_touch\_status[]* to see if any sensors are in detect. If yes, then perform desired action. Else, repeat from step 5.

*qt\_touch\_status* is a sub-field of the data structure *qt\_touch\_lib\_measure\_data\_t*. The above data structure consists of three variables :

Fields	Туре	Comment
channel_signals	uint16_t	The measured signal on each channel.
channel_references	uint16_t	The reference signal for each channel.
qt_touch_status	qt_touch_status_t	The state and position of the configured sensors

*qt\_touch\_status* is a variable of the data structure *qt\_touch\_status\_t*.

Fields	Comment
sensor_states[]	For Sensor, the sensor_states. Bit "n" = state of nth sensor : Bit Value 0 - indicates the sensor is not in detect Bit Value 1 - indicates the sensor is in detect
rotor_slider_values[]	Rotors angles or slider positions if rotors and sliders are used. These values are valid when sensor states shows that the corresponding rotor or slider is in detect

The flow chart below is from the Atmel QTouch Library User Guide that gives a step-by-step approach to the design process.



## **VISUAL GUIDE WITH SCREENSHOTS :**

**Step 1 :** In Atmel Studio 6, in the Start Page choose New Project. Create a GCC QTouch Executable Project and name it. Click OK. As an example, we named the project here LED\_Blink.

Recent Templates	S	ort by: Default		Search Installed Templates
Installed Templates	5	GCC C Executable Project	C/C++	Type: C/C++ Creates an AVR 8-bit OTouch C project
AtmelBoards UserBoards		GCC C QTouch Executable Project	C/C++	
Assembler Atmel Studio Sol	lution	GCC C Static Library Project	C/C++	
		GCC C++ Executable Project	C/C++	
		GCC C++ Static Library Project	C/C++	
Name:	LED_Blink			
Location:	C:\Users\Kedari Rec	ldy\Documents\Atmel Studio		Browse
Solution name:	LED_Blink			Create directory for solution

**Step 2 :** Select the sensors you want to add in the project. The order in which the sensors are selected is important. The wizard allows a pin configuration only in increasing order. For example, a configuration of the sort Slider 0-2, Rotor 5-3 is not allowed. In this example, we chose 1 of each sensor. Select QTouch Technology and the device used. Click Next.

-		D	1	0 • • • •	-			Device Information
Sensors		Button	⊥ 💽 Wheel	U Slider U	•			Device Name:
Technology			uch OMatrix					Family
								Speed
Device		Family	All	•	Search	for device	2	Vec
Name	Variant	App /	All	Data Memory(Bytes)	EEPROM(Bytes)	OTouch	OMatrix	Ports
AT0011581287	64	121	megaAVR	65024	4096	v	~	Datasheets
AT900361267	40/44	121	tinyAVR	16294	4096	~		QTouch Library Information
ATmaga164DA	40/44	16	USB AVR	1024	512	~	~	QTouch
ATmoga169DA	29/22	16	XIVIEGA	1024	512	~	~	Max Wheels/Sliders
ATmega100PA	10/11	22		2048	1024	~	~	Max Channels
ATmoga229D	29/22	22		2040	1024	~	~	Code Memory used 0
ATmega320F	28/22	1		512	256	~	~	Data Memory used 0
ATmega6///D	10/11	65		1096	2048	~	~	QMatrix
ATmega88PA	28/32	8		1024	512	~	~	Max Wheels/Sliders
ATtiny167	20/32	16		512	512	~	~	Max Channels
ATtiny44A	14/20	4		256	256	~	~	Code Memory used 0
ATtiny461A	20/32	4		256	256	~	~	Data Memory used
ATtiny48	32	4		256	64	~	~	QTouch Library Help
ATtiny48	28	4		256	64	~	~	Atmel Studio Supported Tools
ATtiny84	14/20	8		512	512	~	~	
ATtiny861A	20/32	8		512	512	~	~	
ATtiny88	32	8		512	64	~	~	
ATtiny88	28	8		512	64	~	~	
ATxmega128A1	100	139		8192	2048	~	~	
ATxmega16A4	44	20		2048	1024	~	~	
ATxmega16D4	44	20		2048	1024	~	*	
ATxmega32A4	44	36		4096	1024	~	~	
	4.4	26		4006	1024	~	~	

**Step 3 :** Select each sensor from the dropdown and assign a Sensor ID and define the size of each. The sensor ID assigned here will be used throughout to refer to the particular sensor.

Button 0 <b< th=""><th>utton0&gt;</th><th></th><th></th></b<>	utton0>		
Button0 Pr	operties	Project Id: UXC Project Name: default	
Physical F	ïrmware	_	
Sensor Id	0		
Name	Button0		
Height(mm	) 25 <u> </u>		
	Reset		
	Reset		

For the Wheel and Slider, the orientation has to be selected in addition to the dimensions. In the example, a clockwise orientation is chosen for the Wheel and horizontal orientation with LeftToRight direction is chosen for the Slider.

Wheel 0 < Wheel 0>	•		
Wheel0 Propertie	es	Project Id: 0xC Project Name: default	
Physical Firmura	-		
Firmwa	re		
Sensor Id	1		
Name	Wheel0		
Height(mm)	75		
Width(mm)	75		
Direction	Clockwise		
Inner radius(mm)	50 2		
Zero position	0		
2			
_			
	Reset		

Slider 0 <slide< th=""><th>er0&gt; 🔻</th><th></th><th></th></slide<>	er0> 🔻		
Slider0 Prop	erties	Project Id: 0xC Project Name: default	
Physical Fi	rmware		
Sensor Id	2		
Name	Slider0		
Height(mm)	25		
Width(mm)	75		2
Orientation	Horizontal		
Direction	Horizontal		
	Reset		

The sensor IDs are chosen as 0 for button, 1 for the wheel and 2 for the Slider.

**Step 4 :** Next the ports used have to be selected. The SNSK port is connected to the series resistor and SNS port is connected through a capacitor to the series resistor. The sensors can be split over multiple ports (interport) or within the same port (intraport). Therefore, if the number of pins are sufficient, the SNSK and SNS ports can be one port. Some of the sensors can be moved to a new port pair SNSK2 and SNS2. In this case, we used port A as the SNSK1 and port B as the SNS. Click Next.



**Step 5 :** In this step, the exact Port pins have to be assigned to each sensor (as connected in hardware design). The pin numbers HAVE to be assigned in increasing order only. No sensor can have its channels connected to pins in decreasing order as mentioned before. However, it is not necessary that Port B pin 0 has to be connected to Port A pin 0; it can be any pin of A.

QTouch Project Builder	pins - Page 4 o	f 6										
	Sensor	Channel		SNS			SN	SK				
			Port		Pin	Port		Pin	Γ	í.		
	Wheel0	0	В	0	•	A	0	-	*			
	Wheel0	1	В	1	•	A	1	-	1	1		
	Wheel0	2	В	2	•	A	2	•	1	1		
	Slider0	3	В	3	•	A	3	•	1	l l		
	Slider0	4	В	4	•	A	4	•	1	N		
	Slider0	5	В	5	•	A	5	•	1	45		
	Button0	6	В	6	•	A	6	•	1	l l		
									•			
			Reset									
Help					< B	ack	Next	>	F	inish	Can	cel

**Step 6 :** On clicking Next, the user is given an option to enable the debug interface and power optimization. The Debug interface enables the application to output measurement values to I/O pins, which can be used by a USB bridge to view the output on Hawkeye or QTouch Studio. It is disabled for this example.

The power optimization is used to reduce the power usage of the system by about 40%. However, this turns of the spread spectrum feature of QTouch, which is used for better electromagnetic compatibility. We disabled this feature in the example.

Configure QDebug Inte	Frankla ODahura Interf			
	Select Debug Interface	SW impler	mented SPI -	Touch data to Q Touch Analyzer)
	Name	Port	Pin	
	SPI_BB_SS	A -	7 -	
	SPI_BB_SCK	B +	7 👻	
	SPI_BB_MOSI	C	· •	
	SPI_BB_MISO	C *	1 -	
	Power Analyzer Disa	ble 🔻 (T	his option is ava	ilable only for selected devices.)
Extension Parameters				
Delay Cycles	1 Delay c Possibl - For R - For R	ycles that det e values: 1 to s = 1KOhm, " s = 100KOhn	termine the cap 255 CPU Clock Typical pulse tin n, Typical pulse	scitance charge pulse width. cycles (nops) le is 1μs dime is 4 μs
Power Optimization	Disable - Used to	o reduce the	power consume	d by the library.
			N	

**Step 7 :** On clicking Next, a summary is generated. Click Finish. The wizard generates the code with separate files for setting up the timer, configuring the touch sensors, initializing the MCU system clock etc. There is also a main.c file that provides the user with an infinite loop to perform desired actions. The first step now is to configure the sensors using the function  $config_sensors()$  in the file LED\_Blink.qtdgn  $\rightarrow$  touch.c

The user can change the resolution of the slider/rotor from 8-bit to lower. The hysteresis can also be modified.



**Step 8 :** The QTouch parameters can also be changed in the above file. The function  $qt\_set\_parameters()$  sets default values for the parameters. The user can change these values depending on the functionality and sensitivity desired. Refer to the end of this file for a description of what each parameter indicates.

Step 9 : The ISR is initialized in the function  $init\_timer\_isr()$  in src  $\rightarrow$  QTouch  $\rightarrow$  init\\_mcu\\_atmega1284.c

The timer is initialized to run at 50ms. This time can be changed if desired by the user depending on how often the touch measurement wants to be made.

ELD_Blink - AtmelStudio		
File Edit View VAssistX Project Build Debug Tools Window Help		
: 🚼 🕶 🏥 🖅 💕 🛃 🎒 👗 🦾 📇 🕫 - 🗠 - 💭 + 🖳 🔚 🔍 🛛 🗰 🖉 Debug 🛛 - 🕅 🖉 DEBUG_INTERFACE_	🗟 🛃 🕾 🖉	[清清]]][2]]][][2]][][2]][2]][2][2][2][2][2][
: 🗑 🖾 🐼 🍋 🔓 🍓 🚓 🔏 📲 🗰 🗉 🔷 🗉 🕨 💷 🔷 🗉 🕨 😂 🖼 📜 🖆 🖅 🖅 Hex 📑 📲 💭 🖾 🗐 🚳 📮 🛗 🗄	🗄 🚠 📮 😳 🛲 ATmega1284P 🥤 No too	ol selected 🖕
init_mcu_atmega1284.c ×	-	Solution Explorer 👻 म 🗙
⇔ BRC •	• ( <b>*</b> Go	B 3
<pre>Bijf defined (_ATmega1284P_) Bijf defined(_QTOUCH) B/* Name : init_timer_isr Purpose : configure timer ISR to fire regularly ************************************</pre>		Solution LD, Blink (Lproject)  Solution LD, Blink  Solution LD, B
100 % - 4	*	🔍 ASF Explorer 💀 Solution Explorer
🗱 Error List 🔳 Output		0.120 0.20 1
keady	En 114	C0130 Ch 30 IN

The user can change *qt\_measurement\_period\_msec* macro defined as 50 in the above case to any value desired.

**Step 10 :** In main.c, the system is initialized by disabling the JTAG pins and prescaling the system clock (if desired). The file qt\_asm\_tiny\_mega.s in src  $\rightarrow$  QTouch does this using assembly language. Next the function defined for touch initialization *touch\_init()* is called. This function sets up the Studio Masks which can be obtained from the QTouch studio by defining the pin configurations and type of sensors desired. It configures the sensors (*config\_sensors()*), sets up the touch parameters with default values. (*qt\_set\_parameters()*) and calibrates the sensors (*qt\_init\_sensing()*). The function for ISR initialization is also called. Now, the system is ready to measure data.



**Step 11 :** For measuring capacitive data, a function called *touch\_measure()* is called in an infinite loop. This function checks the flag *time\_to\_measure\_touch* which is set in the ISR every 50ms. Hence, it executes every 50ms and updates the status flags using the function  $qt_measure_sensors()$  and also updates the variable of the data structure  $qt_touch_lib_measure_data_t$ .

**Step 12 :** To check the status of the sensors and angle/position values, we define two macros as below :

include files #include aver/interrupt.h> #include (aver/interrupt.h> #define _delay_cycles(n) _builtin ave_delay_cycles(n) #define _enable_interrupt() sei()	
<pre>#include <avr io.h=""> #include <avr interrupt.h=""> #define _delay_cycles(n)builtin avr_delay_cycles(n) #defineonable_interrupt() _sei()</avr></avr></pre>	
<pre>finclude cavr/incrrupt.h&gt;</pre>	
<pre>#include cavr/interrupt.n&gt; #definedelay_cycles(n)builtin_avr_delay_cycles(n) #defineenable_interrupt() sei()</pre>	
<pre>#definedelay_cycles(n)bulitin_avr_delay_cycles(n) #defineenable_interrupt() sei()</pre>	
#defineenable_interrupt() sel()	
tipelude "touch and h"	
History to the	
mancadae couchtan	
manifast constants	
#/	
*/	
#define GET SENSOR STATE(SENSOR NUMBER) of measure data.ot touch status.sensor states[(SENSOR NUMBER/8)] & (1 << (SENSOR	NUMBER % 8
#define GET ROTOR SLIDER POSITION(ROTOR SLIDER NUMBER) at measure data at touch status, rotor slider values[ROTOR SLIDER N	UMBER1 T
	. 1
· /*	
type definitions	
*/	
*/	
*/ =/*	
*/ □/**/	
*/	
<pre>/**/ /*- /*- /*- /*- /*/ /*/ /*/ /*</pre>	
<pre>/*</pre>	
<pre>/*</pre>	
<pre>/*</pre>	

## GET\_SENSOR\_STATE(SENSOR\_NUMBER)

qt\_measure\_data is a variable of a library defined data structure qt\_touch\_lib\_measure\_data\_t. In the above macro, we intend to get the touch status of a particular sensor and hence, the  $qt_touch_status[]$  consists of the information desired. As discussed earlier,  $qt_touch_status[]$  in itself is a structure that returns status of sensors and positions/angles of the slider/rotor.

Therefore the macro *qt\_measure\_data.qt\_touch\_status.sensor\_status[...]* returns 1 if the particular sensor is in detect and returns 0 if it is not touched.

By checking the condition *if*(*GET\_SENSOR\_STATE*(*SENSOR\_NUMBER*)), we know if a particular sensor has been touched. SENSOR\_NUMBER here is the Sensor ID defined in the wizard. Accordingly, an action can be performed.

## GET\_ROTOR\_SLIDER\_POSITION(SENSOR\_NUMBER)

It is defined as *qt\_touch\_status.rotor\_slider\_values[....]* returns the position of the slider sensor or the angle of the rotor slider.

The SENSOR\_NUMBER here is not the same as the sensor ID used in the macro GET\_SENSOR\_STATE. This sensor number is assigned only to Rotors/Sliders in the order in which they are defined. The sensor\_number in this case is 0 for a rotor and 1 for a slider since the Slider was defined after the Rotor.

SENSOR	SENSOR ID	SENSOR_NUMBER for ROTOR_SLIDER_VALUES
Button	0	
Rotor	1	0
Slider	2	1

Therefore, a code as shown below is used to determine the sensors in detect and get the position or angle values on a scale of 0-255.

