The Professional Group of System LSI - **ATLab. Inc**

Digital Contact Controller
ATA5009 User Manual

Table of Contents

1. Overview

The ATA5009 uses time-domain multiplexing (TDM) architecture which shares one *pulse generation unit* to measure capacitance variations of all sensor input channels to achieve a cost effective design goal and adopts active pulse-pass architecture called *CapScope-P™* to improve noise immunity by automatically calibrating offset capacitance and gradual environmental capacitance changes of sensor input channels. It also contains two different low pass filters and a down sampler to suppress various noises. Its widened dynamic range which compensates for offset capacitance in the sensor input channels eliminates troublesome hardware tuning. The capacitance acquired by human touch is represented to 10-bit digital values to increase sensitivity resolution.

In addition to these major changes in technology, the ATA5009 provides a hard macro pipeline *position finding unit* which can calculate very high resolution of interpolated coordinates of up to ten thousand times per second. These new benefits make the ATA5009 suitable for 2D touch screen applications and/or high resolution scroll applications. To significantly reduce power consumption, the ATA5009 provides not only an optimized internal voltage regulator but also a smart power management scheme which has three operation modes – active, idle and sleep modes.

The ATA5009 has a total of 21 sensor input channels, and each of which can be configured to be either a sensor input or a general-purpose output. All the previously existing functions such as APIS, AIC, etc. still remain in the ATA5009.

The ATA5009 comes with 32-pin QFN. For other package type, please contact through dcc@atlab.co.kr.

2. Features Summary

- Patented Pulse-Pass TDM architecture
- 21 channels configurable to be sensor inputs, general-purpose outputs, or ADC
  - Capacitive sensing up to 1 fF resolution, current sensing up to 300 pA resolution
  - Current driving capability for general-purpose outputs: 14mA sink current and 8mA source current at VDDH 3.3V
  - 9-bit ADC with 3.4 kHz sampling, 15-bit ADC with 100 Hz sampling
- Current driving capability for general-purpose outputs: 20mA sink current or 3mA source current
- I^2C and SPI interfaces with the host MCU
  - I^2C Speed: 100KHz and 400KHz, I^2C Slave address: 0x64 or 0x66 (two IDs supported by the pin option)
  - SPI speed: 4MHz
- Wide Dynamic range (varies by the selection of the internal I/O resistor)
  - 3.2pF for human touch detection in the condition of 35K ohm internal I/O resistor selected
  - 79pF for offset capacitance compensation in the condition of 35K ohm internal I/O resistor selected
- High Sensitivity resolution (varies by the selection of the internal I/O resistor)
  - 1.1fF (min), 3.1fF (typ.), 10.8fF (max.)
- Various outputs available
  - 1-bit Touch output (APIS output) up to120Hz of report rate
  - 13-bit Position output up to 6000Hz of report rate
  - 10-bit Capacitance output up to 6000Hz of report rate
  - 8-bit Position Strength output up to 6000Hz of report rare
- Two types of interrupts: GINT for general-purpose and for touch detection or position change
- Wakeup from IDLE power saving state by touch operation
- Programmable registers to characterize applications
- Configurable AIC™ (Automatic Impedance Calibration)
- Two different modes for APIS™ (Adjacent Pattern Interference Suppression)
  - APIS mode 1: generates the strongest output
  - APIS mode 2: generates all outputs beyond their pre-defined threshold limits
- Hard macro pipeline position finding unit to generate 1D or 2D interpolated coordinates

3. Electrical Characteristics

Table 3-1. Electrical Characteristics

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
	ABSOLUTE MAXIMUM RATINGS					
Tstg	Storage Temperature		-40		90	℃
Topr	Operating Temperature		-35		85	℃
VDDH	IO Power Supply Voltage		1.6	3.3	5.5	V
VDDL	Core Power Supply Voltage		1.65	1.8	1.95	V
ESD	HBM			±5		KV
	MM			±200		V
	RECOMMENDED OPERATING CONDITIONS					
Toprr	Operating Temperature		-30	25	80	℃
Vddp	Power Supply Voltage (VDDH)		1.65		4.0	V
Vddc	Power Supply Voltage (VDDL)		1.7	1.8	1.9	V
Tr_i	Digital Input Rising Time	Input Pin			5	ns
Tf_i	Digital Input Falling Time	Input Pin			5	ns
	AC ELECTRICAL SPECIFICATIONS (Typical values at Ta=25℃ and VDDH=3.3V)					
fsys	System Clock		1.8	2	2.2	MHz
fs	Touch Input sampling frequency	When System Clock is 2MHz	3.4		12	KHz
Dr_c	Coordinate output Data Rate	When System Clock is 2MHz, maximum fs	24	200	6000	Hz
Dr_t	Touch output Data Rate	When System Clock is 2MHz	34		120	Hz
Stch	Touch Sensitivity		1.1	3.1	10.8	fF
Rs_i	Sensor Input Resistance		10	35	100	KΩ
Coff	Allowable Offset Capacitance for Channel to GND	When Max Rs_i = 35K ohm			79	pF
Tr_o	Output Rising Time	Load = 100pF		50	60	ns
Tf_o	Output Falling Time	Load = 100pF		50	60	ns
	DC ELECTRICAL SPECIFICATIONS (Typical values at Ta=25℃ and VDDH=3.3V and VDDH = 1.8V)					
Idd_ae	Supply Current (Active mode)	When using 1.8V VDDL, Fs = 3.4KHz	70	115	170	μA
Idd_ie	Supply Current (Idle mode)			40		μA
Idd_se	Supply Current (Sleep mode)			30		μA
Vil	Digital Input Low Voltage				0.25xVDDH	V
Vih	Digital Input High Voltage		0.7xVDDH			V
Vol	Digital Output Low Voltage	When VDDH=3.3V Isink = 14mA	0.15xVDDH			V
		When VDDH=1.8V Isink = 2mA				
Voh	Digital Output High Voltage	When VDDH=3.3V,Isource = 8mA			0.8 x VDDH	V
		When VDDH=1.8V,Isource = 2mA				
Idrsnk	GPO, I2C, INT Driving(Sink) Current	VDDH=3.3V	- 14			mA
		VDDH =1.8V	- 2			mA
Idrsrc	GPO, I2C, INT Driving (Source) Current	VDDH=3.3V			8	mA
		VDDH =1.8V			2	mA

4. Package & Pin Description

4.1. Ordering Information

Product Code	Package Type	Package Dimension	Pin Pitch	Num. of Sensor Inputs or Num. of Digital Outputs
ATA5009DB-32N	32QFN	4mm x 4mm	0.4mm	21

Table 4-1. Ordering Information

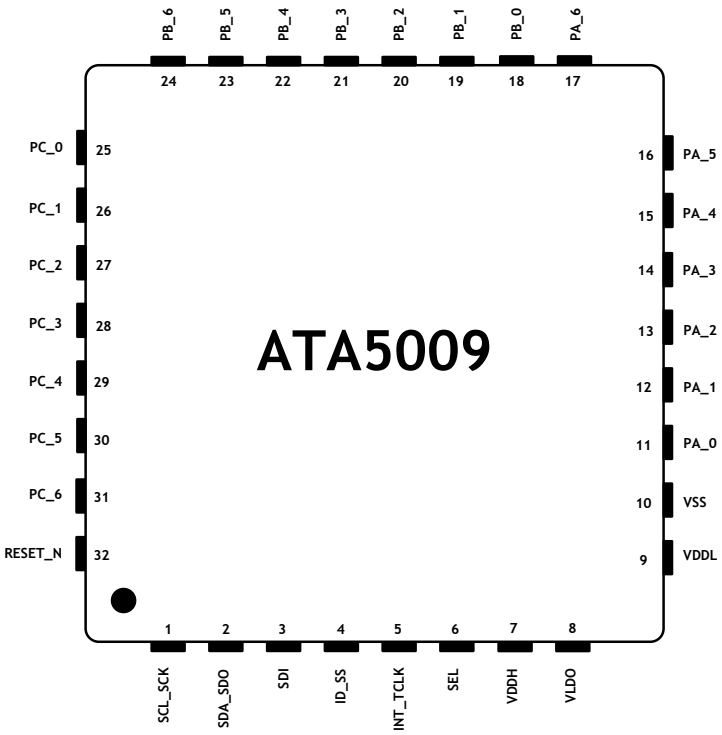


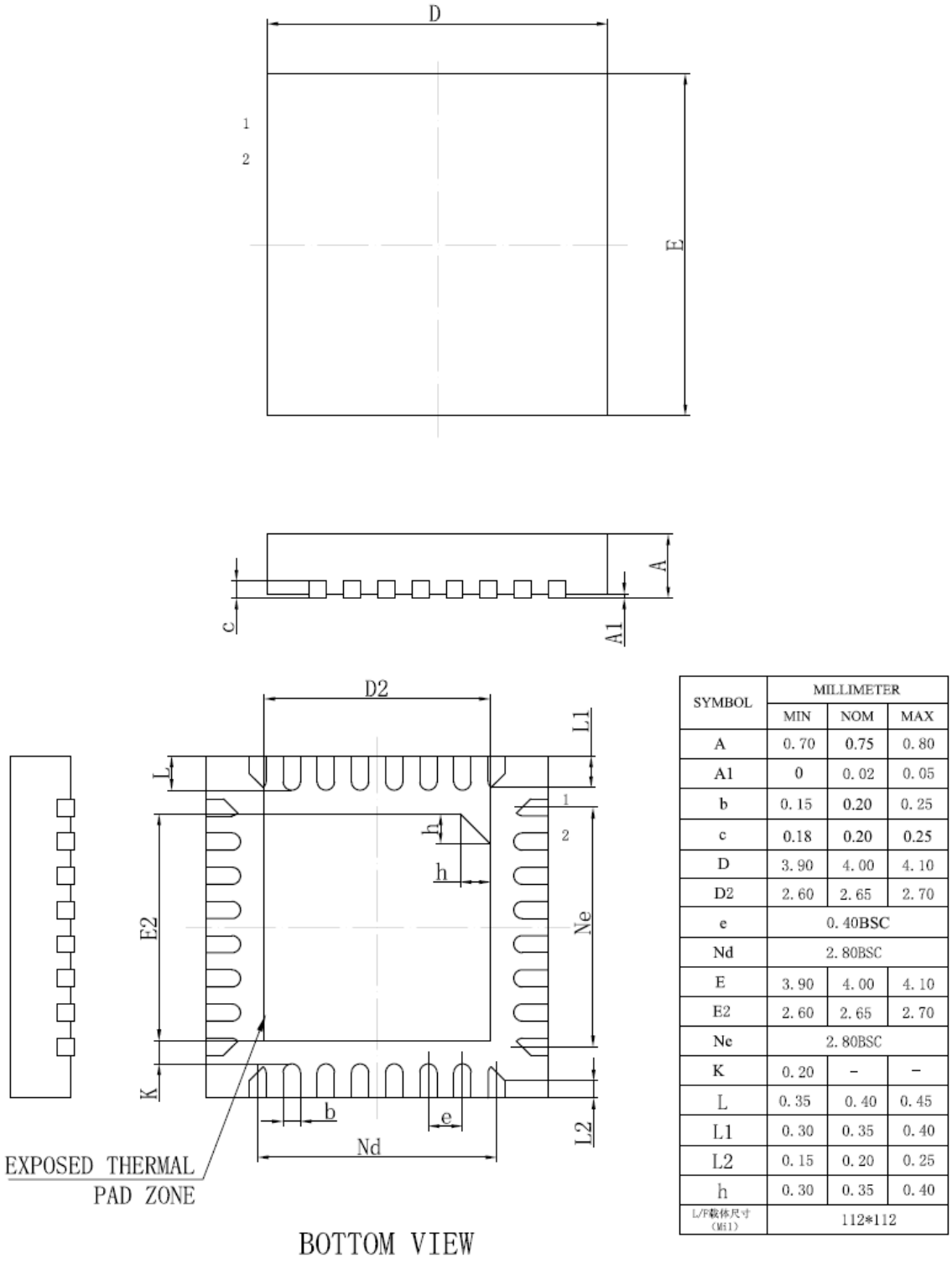
Figure 4-1. ATA5009_32QFN Package

4.2. Pin Description

Pin number	Name	Width	IO	Description
32	RESET_N	1	I	Reset, active LOW
5	INT_TCLK	1	IO	Interrupt Output or Test Clock Input
6	SEL	1	I	Host Communication: I^2C or SPI
11..17	PA	7	IO	7 Ports for Sensor Input or GPO
18..24	PB	7	IO	7 Ports for Sensor Input or GPO
25..31	PC	7	IO	7 Ports for Sensor Input or GPO
1	SCL_SCK	1	I	I^2C CLK or SPI CLK from the Host MCU
2	SDA_SDO	1	IO	Bi-directional I^2C Data from/to the Host MCU or SPI Data out to Host MCU
3	SDI	1	I	SPI Data input from Host MCU
4	ID_SS	1	I	I^2C Chip ID setting or SPI Slave Select from Host MCU
7	VDDH	1	P	IO power (1.8V, 3.3V, or 5.0V)
8	VLDO	1	O	1.8V LDO output
9	VDDL	1	P	1.8V Core Power
10	VSS	1	P	Ground

Table 4-2. 32QFN Pin Descriptions

4.3. Package Dimensions



SYMBOL	MILLIMETER		
	MIN	NOM	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
b	0.15	0.20	0.25
c	0.18	0.20	0.25
D	3.90	4.00	4.10
D2	2.60	2.65	2.70
e	0.40BSC		
Nd	2.80BSC		
E	3.90	4.00	4.10
E2	2.60	2.65	2.70
Ne	2.80BSC		
K	0.20	–	–
L	0.35	0.40	0.45
L1	0.30	0.35	0.40
L2	0.15	0.20	0.25
h	0.30	0.35	0.40
L/F载体尺寸 (MS1)	112*112		

5. Tuning System

5.1. Hardware

The tuning system helps the developer to tune the target touch board with various parameter adjustments that determine the performance of the target touch board. The tuning system consists of a sample target touch board, a USB I/F board which transmits data between the PC and the target touch board, and a PC where the tuning program runs as shown in Figure 5-1. Tuning system allows the developer to view all necessary parameters for tuning and transfers the desired values of parameters to the ATA5009 attached on or connected to the target touch board.

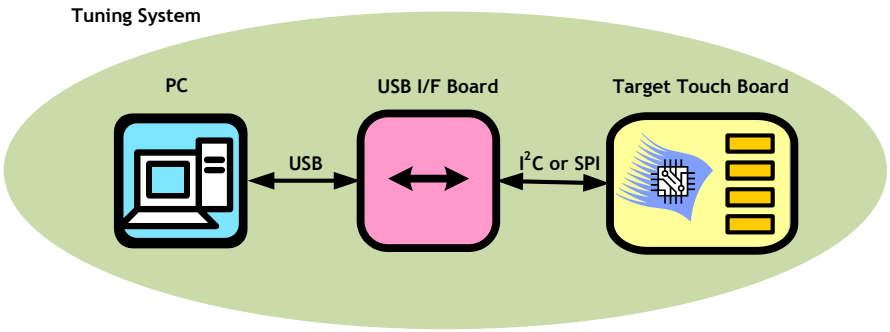


Figure 5-1. Conceptual Diagram of Tuning System

5.1.1. Application Board

The application board to develop typically has an ATA5009, Touch Screen panel, and the application's system board which interfaces with the ATA5009 as shown in Figure 5-2.

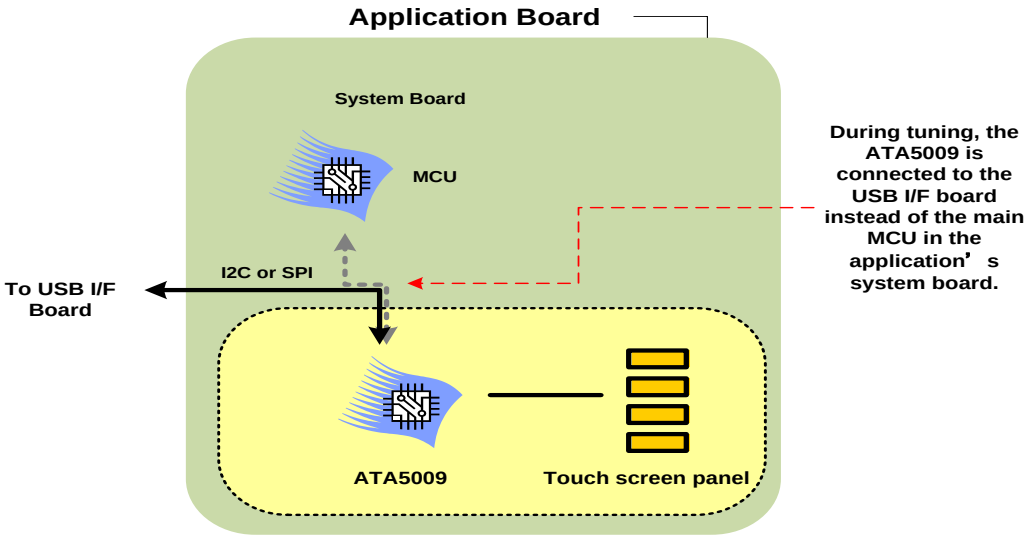


Figure 5-2. Diagram of an Application board

The ATA5009 can be located at either the touch screen panel or application's system board. During tuning the touch screen panel, the connection of I2C or SPI to the system main MCU should be cut and connected to the USB I/F board in the tuning system. To make this work easier, 0 ohm resistors or jumpers can be used on the I^2C or SPI lines.

5.1.2. PC and USB I/F Board

PC and USB I/F board behave like an MCU on the application board to tune the ATA5009. The USB I/F board has an 8bit MCU which transfers commands from PC to the ATA5009 or reads data from the ATA5009 and delivers them to the PC by switching the protocol between USB and I^2C/SPI.

Development-support-kit:

Development-support-kit as shown in Figure 5-3 consists of one USB I/F board, one sample touch board where the ATA5009 is attached, one USB extension cable, one 10-pin I^2C/SPI cable, and one CD including the ATA5009 tuning setup program which should be installed on the PC before starting tuning, Datasheet, and this user manual. After completing software installation, users can send commands to the MCU in the USB I/F board by PC tuning program and eventually change the values of internal registers of the ATA5009 on the sample touch board. Various output data of ATA5009 such as touch data, (x,y) coordinates or capacitance variation data will be transferred to the PC through the USB I/F board for display on the tuning program. The MCU in the USB I/F board controls the ATA5009 via I^2C or SPI interface to access internal registers of the ATA5009.

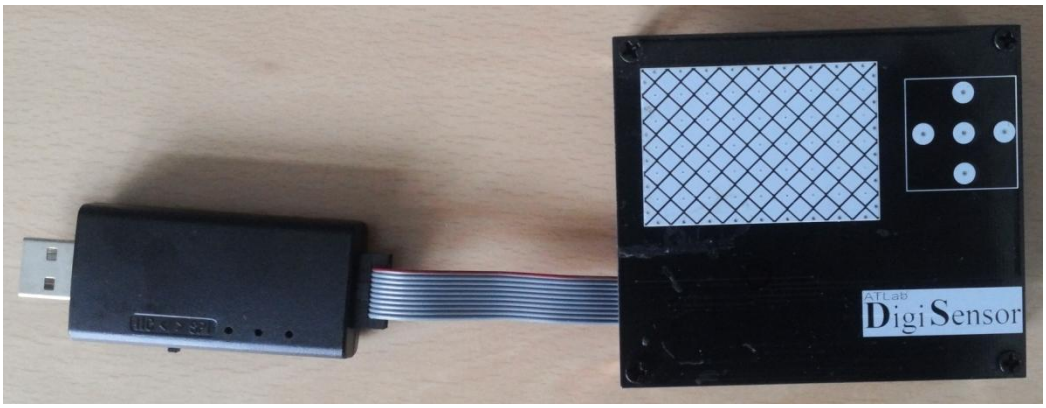


Figure 5-3. Development-support-kit

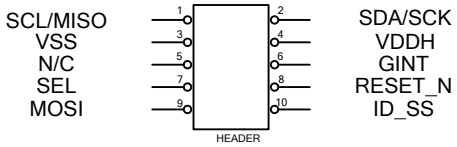
5.1.3. Connecting a sample touch board to USB I/F Board

Before connecting a sample touch board.

Before starting tuning the ATA5009, you need to connect the USB I/F board to the PC. The PC also requires MCU device driver program to control the Development system. Once you install the tuning setup program included in the Development-support-kit, an '.inf' file is created in the ../Inf folder. After finishing this installation procedure, the PC automatically detects corresponding device driver program when USB I/F board is connected. You can also assign it manually. After the installation is completed and the PC has recognized the Development-system, you will see the message, 'ATLab. Silabs320 board'. You are now ready to control the ATA5009 with the ATA5009 Tuning Program installed on the PC.

Connecting a sample touch board to the USB I/F board

The sample touch board is connected to the USB I/F board via an I^2C or SPI cable. This cable is composed of 10 pins and the pin assignments are shown in Figure 5-4.



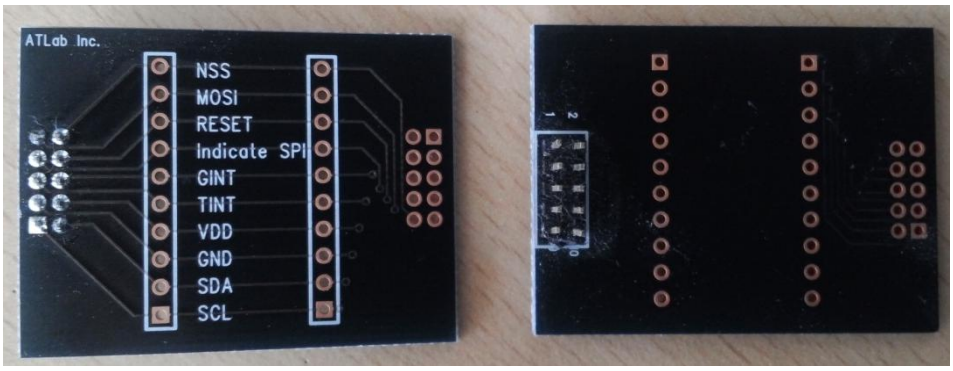


Figure 5-4. Pin Assignments of a 10-pin connector.

As like as the sample board, the 10-pin connector (or equivalent test posts) that should be on customer application board helps to decide register values during software/hardware development, to apply board testing during production, and to analyze fault causes.

10-pin connecter		ATA5009		Comment
pin number	Signal name	Pin number	PIN name	
1	SCL	1	SCL_SCK	I^2C Clock
2	SDA	2	SDA_SDO	I^2C Data
3	VSS	10	VSS	Ground-
4	VDDH	7	VDDH	I/O voltage supply
5	-	-		-
6	GINT	5	INT_TCLK	Interrupt Pin
7	SEL	6	SEL	Selection pin of I^2C or SPI, This pin should be grounded.
8	RESET_N	32	RESET_N	If user wants to give a RESET signal from MCU, need to connect this pin.
9		-	SDI	Don't care but, SDI pin should be Vdd or Gnd
10	ID_SS	4	ID_SS	I^2C Chip ID Selection VSS=0x64, VDD=0x66

Table 5-1. I^2C port configuration Table

10-pin connecter		ATA5009		Comment
pin number	Signal name	Pin number	PIN name	
1	MISO	2	SDA_SDO	SPI Data output
2	SCK	1	SCL_SCK	SPI Clock
3	VSS	10	VSS	Ground-
4	VDDH	7	VDDH	I/O voltage supply
5	-	5	-	-
6	GINT	5	INT_TCLK	Interrupt Pin
7	SEL	6	SEL	Selection pin of I^2C or SPI This pin should be VDDH.
8	RESET_N	32	RESET_N	If user wants to give a RESET signal from MCU, need to connect this pin.
9	MOSI	3	SDI	SPI Data input
10	ID_SS	4	ID_SS	SPI device selection (active low)

Table 5-2. SPI port configuration Table

5.2. Tuning Software

5.2.1. User Interface

When tuning software is installed, tuning software is located to “c:\ATLab\ATA5008 Tuning Viewer Xpress”. After the tuning kit shown in Figure 5.3 is connected to PC USB port, Window OS will find USB driver automatically. If PC does not find necessary USB driver, then please install USB driver manually. The USB driver is stored at in “c:\ATLab\ATA5008 Tuning Viewer Xpress\USB Driver”. Note that ATA5009 uses the same UI as ATA5008.

Figure 5-5. Main UI Screen

If you have previous tuning data, then select “Import Register File” as shown in Figure 5-6. It is important to notice that your intermediate tuning register values should save by selecting “Export Register File” for further tuning step.

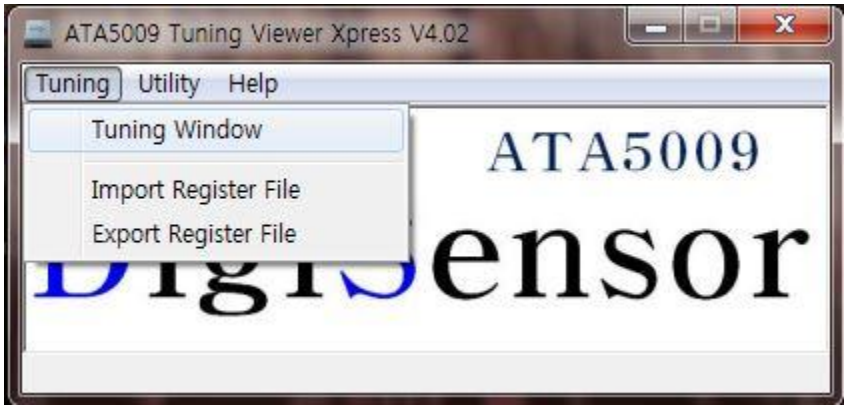


Figure 5-6. Import/Export/Tuning Window Menu

When selecting “Tuning Window” menu under “Tuning”, imported register values are displayed in Configuration Registers UI. The Configuration Registers UI enables to change whole register values. After register modification is made, click ”Write All” button and “Monitor” button. Then, either/both “Position Window” or/and “PValue Window” are displayed.

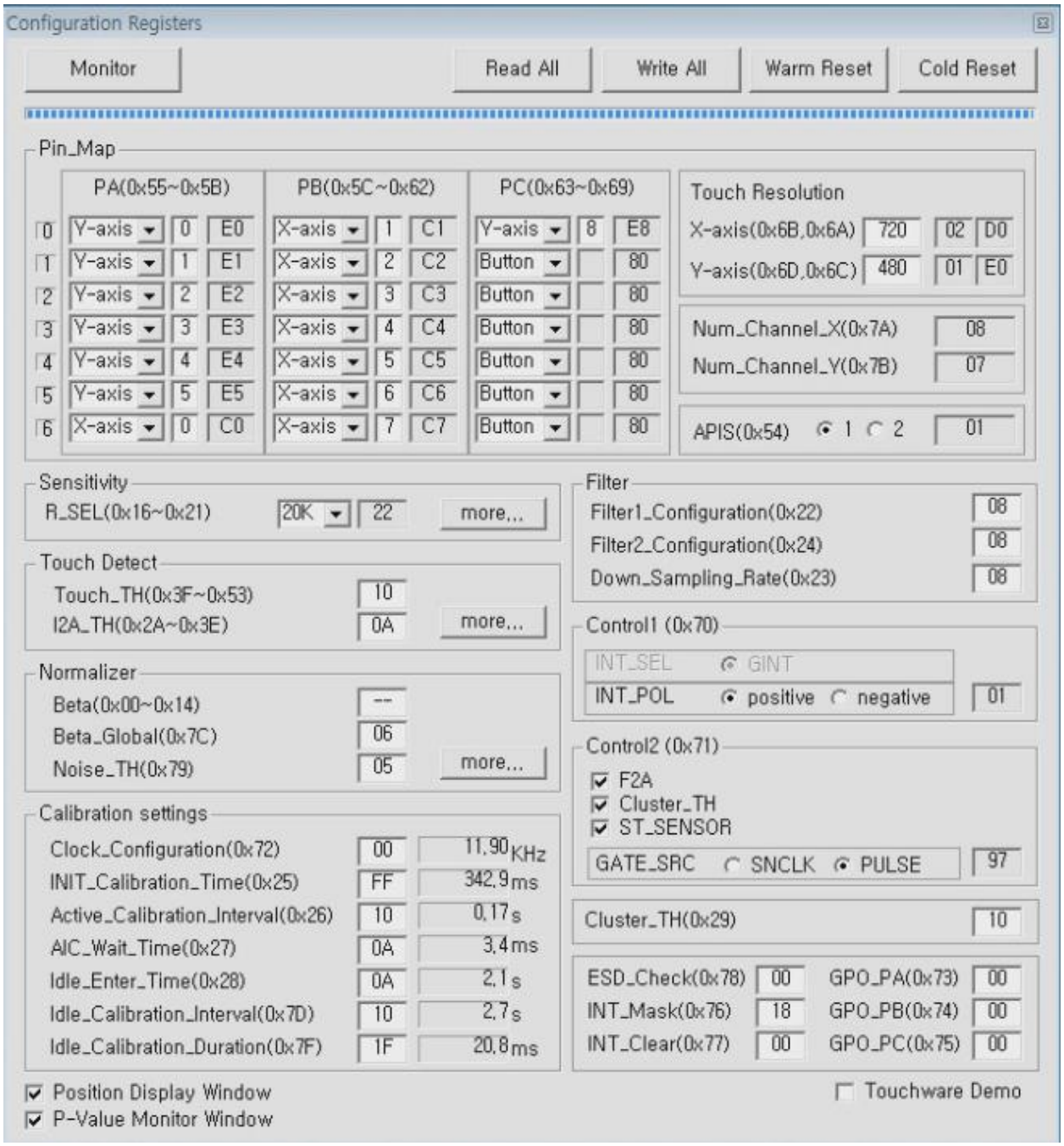


Figure 5-7. Configuration Registers UI

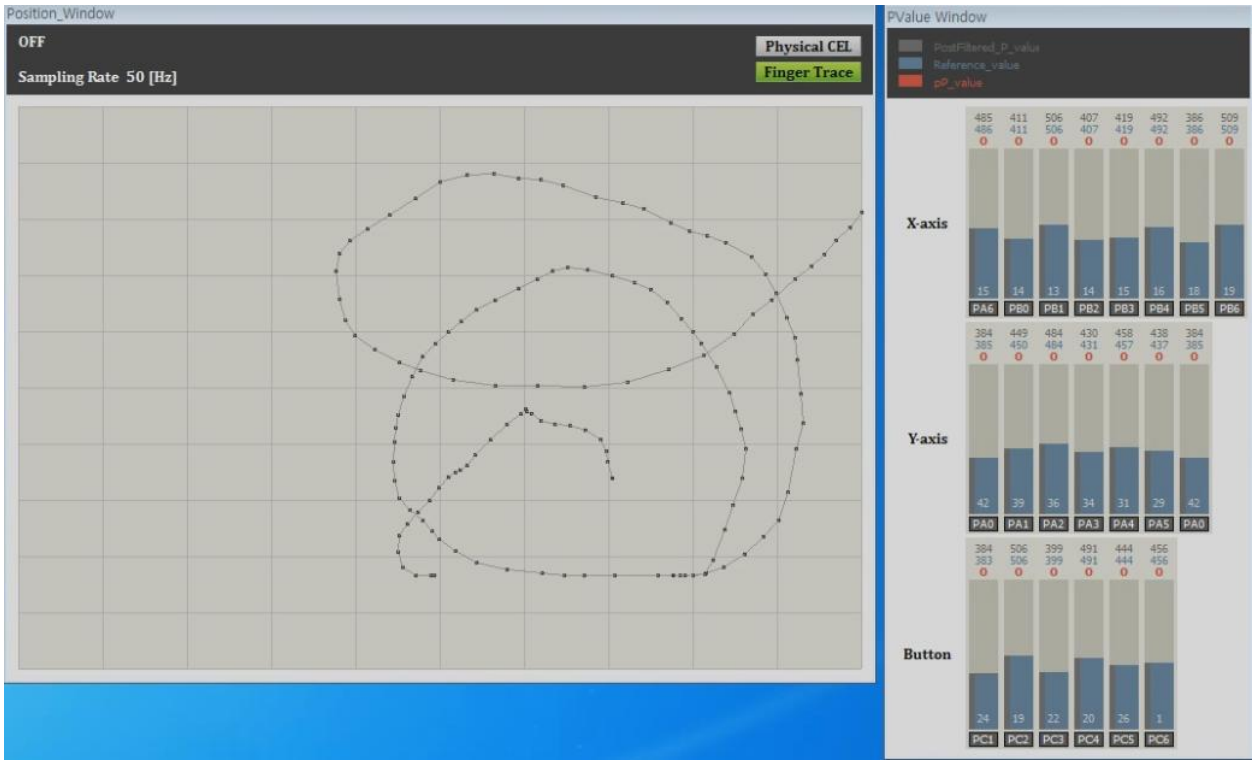


Figure 5-8. UI of Position Window and PValue Window

These UIs are selected by your own application. There are Position Window and Pvalue Window. You may select necessary UI(s). Figure 5-7 and 5-8 are example UIs, which is set to 8-channel for X-axis, 7-channel for Y-axis, and 6-channel for buttons.

5.2.2. Tuning by Configuration Registers UI PValue Window

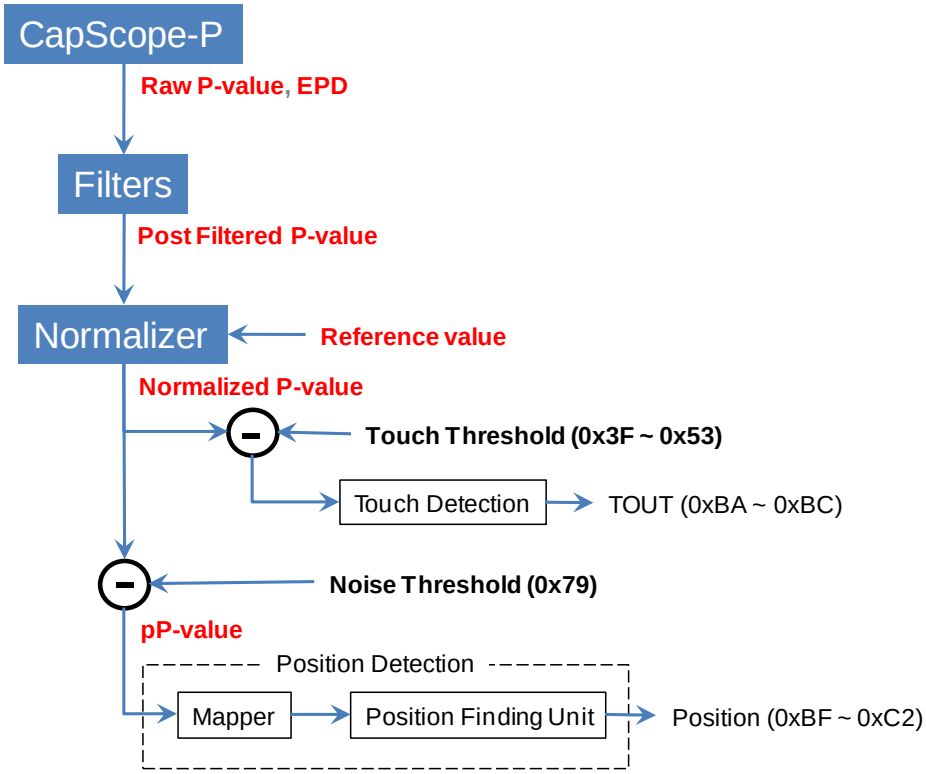


Figure 5-9. Internal Parameters and Monitoring Parameters

Among ATA5009 internal parameters, few parameters are displayed in PValue Window for parameter setting purpose. Capacitance sensing block (CapScope in Section 8-1) generates Raw P-value and EPD value. Here, EPD value is matched to offset capacitance values. The PostFiltered P-value is generated from Raw P-Value through Filter block). Reference value is generated by AIC (Automatic Impedance Calibration) process.

In the PValue Window, EPD, PostFiltered P-value, and Reference value are shown in white colored number inside dark blue box, gray colored narrow bar, and dark blue bar, respectively. Normalized P-value is generated by subtracting PostFiltered P-value with Reference value. The pP-value in red color is used to decide touch on/off and/or touch position.

By monitoring values in PValue Window, you can optimize register values. In no noise condition, the PostFiltered P-value is changed by +1 or -1. If variation of the PostFiltered P-value is larger than 1 at no-touch state, then there is some noises. Please minimize the variation by hardware changes. After hardware is fixed, please optimize Filter register values of 0x22, 0x23, and 0x24 in the Configuration Registers UI.

EPD value related to offset capacitance is also useful to judge hardware validity. If all channel are uniformly formed, then the EPD values are also uniform. If there is grounded plate to shield noises from main board(s), then EPD values become large.

Reference value is updated at every AIC operation that compensates environment changes. You may set the Calibration settings inside Configuration Registers UI.

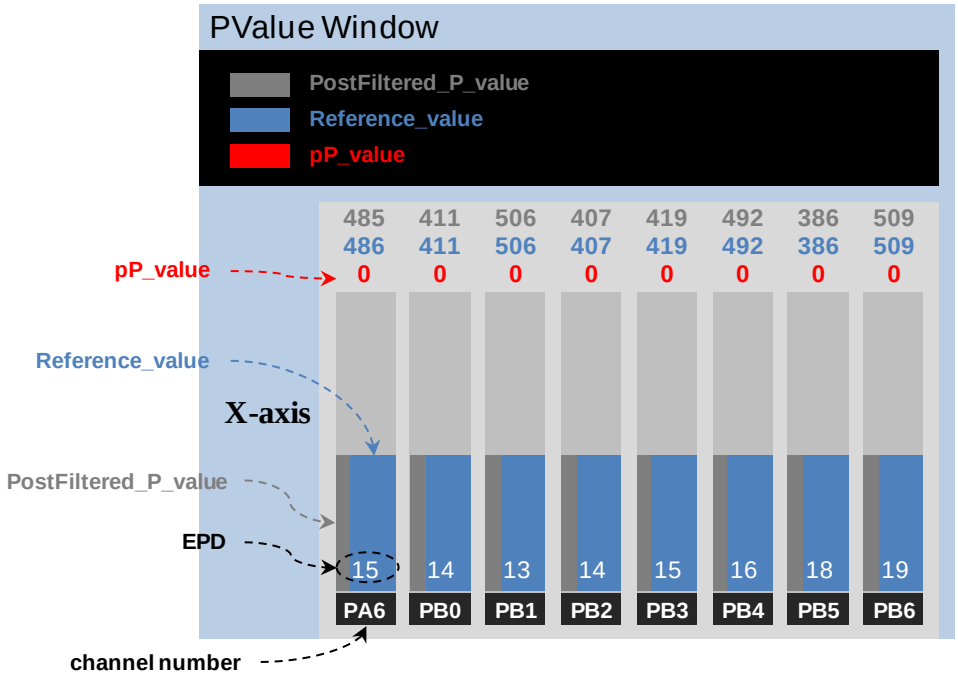


Figure 5-10. Enlargement of the PValue Window

When a specific channel is touched, box color of the channel number is changed to green. Note that the touch signal is only displayed when the channel is defined to “Button”, not X/Y-axis. Touch sensitivity is also monitored with the pP-value. Then, R_SEL registers (0x16 ~ 0x21) and Noise_TH (0x79) are selected in the Configuration Registers UI.

5.3. Tuning Flow chart

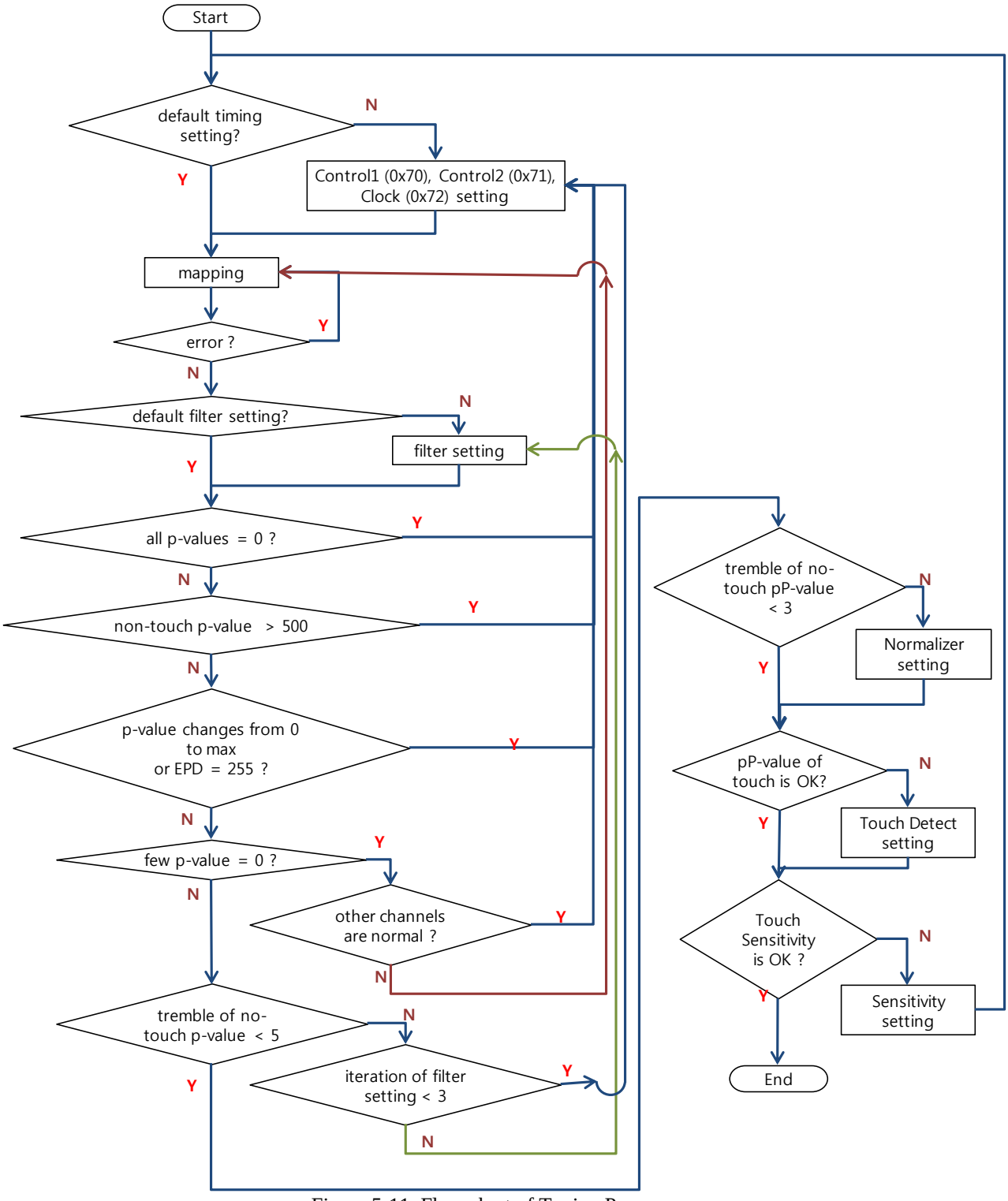


Figure 5-11. Flow chart of Tuning Process

It is important to notice that default value of ST_SENSOR bit[2] of Control2 register is “stop sensor operation”. If your application uses single ATA5009, then please **set ST_SENSOR of Control2(0x71)** in Figure 5-7.

5.4. Register lists related with each tuning process

5.4.1. Mapping registers

Address	R/W	Register name	Comments
0x55~0x69	RW	Pin_Map_PA0 ~ Pin_Map_PC6	Values depend on the user application
0x7a	RW	NUM_Channel_X	Values depend on the user application
0x7b	RW	NUM_Channel_Y	Values depend on the user application

5.4.2. Filters setting

Address	R/W	Register name	Comments
0x22	RW	Filter1_Configuration	Typically set to 0x01
0x23	RW	Down_Sampling_Rate	Value depends on the user application
0x24	RW	Filter2_Configuration	Modify the value during tuning if necessary

5.4.3. Normalizer setting

Address	R/W	Register name	Comments
0x00~0x14	RW	Beta_PA0 ~ Beta_PC6	Modify the value during tuning if necessary
0x7c	RW	Beta_Global	Modify the value during tuning if necessary
0x79	RW	Noise_Threshold	Modify the value during tuning if necessary

5.4.4. TDU setting

Address	R/W	Register name	Comments
0x2a~0x3e	RW	I2A_TH_PA0 ~ I2A_TH_PC6	Modify the value during tuning if necessary
0x3f~0x53	RW	Touch_TH_PA0 ~ Touch_TH_PC6	Modify the value during tuning if necessary

5.4.5. Advanced setting

Address	R/W	Register name	Comments
0x16~0x21	RW	R_SEL_PA0 ~ R_SEL_PC6	For sensitivity change of Position application
0x70	RW	CTRL_1	Control options
0x71	RW	CTRL_2	Control options
0x72	RW	CLK_CONF	Modify the value during tuning if necessary
0x25	RW	INIT_CAL	Typically set to 0xFF
0x26	RW	ACAL_INTV	Modify the value during tuning if necessary
0x27	RW	AIC_Wait_Time	Typically set to 0xFF
0x28	RW	A2I_Time	Modify the value during tuning if necessary
0x29	RW	Cluster_Threshold	Activate and define only for 2D application
0x7d	RW	ICAL_INTV	Modify the value during tuning if necessary
0x7e	RW	I2A_flag_number	Typically set to 0x01
0x7f	RW	ICAL_DUR	Modify the value during tuning if necessary

6. Application Circuit and PCB Guidelines

6.1. Application Circuit

6.1.1. 32QFN Application Circuit – I2C Interface

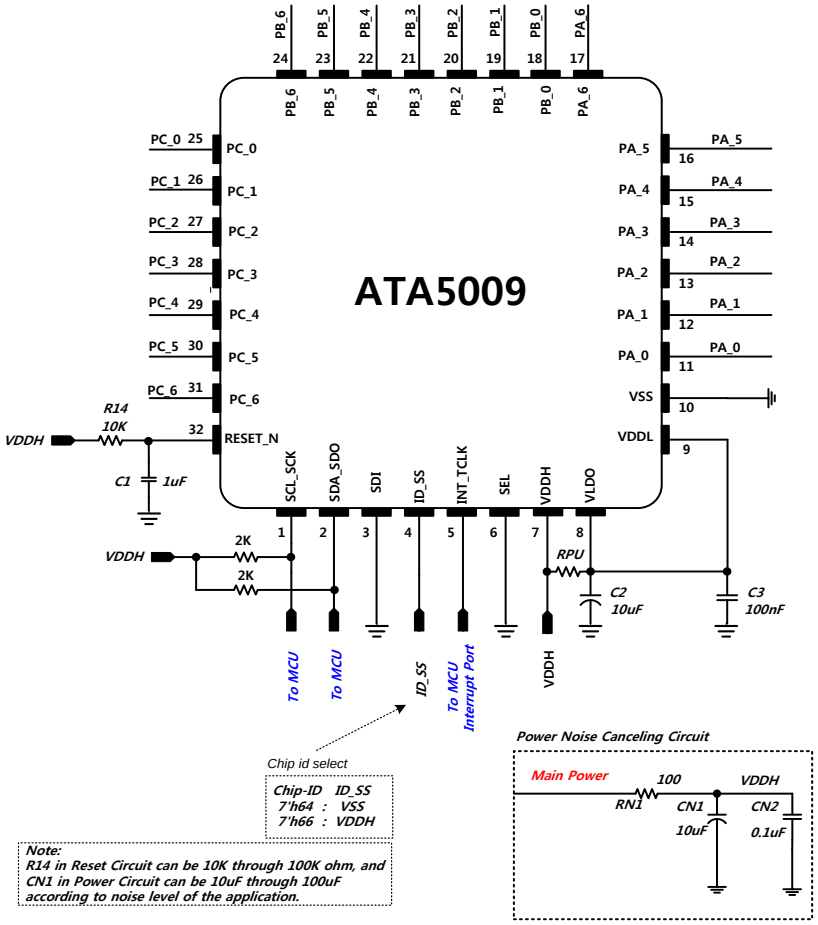


Figure 6-1. ATA5009_32QFN I²C Application Circuit

Operating mode	VDDH	*RPU*	Idle Current	Sleep Current	Remark
Use external 1.8V	2.1V ~ 5.0V	Open	22 µA	1.9 µA	set Addr. 0x81 and Data 0xFA
Use internal LDO w/o power saving mode	2.1V ~ 5.0V	Open	65 µA	40.0 µA	Internal LDO is always ON.
Use internal LDO w/ power saving mode	3.3V	50KΩ	33 µA	13.2 µA	See note 1 & 2
	5.0V	240KΩ	33 µA	13.5 µA	
	2.1V ~ 5.0V	Open	22 µA	1.9 µA	Need hard reset. See note 2 & 3

Note

1. *RPU* is recommended under the condition of maximum operating temperature of 80℃. If another maximum operating temperature is used, please ask ATLab for finding a proper *RPU* value. Generally speaking, for higher temperature than 80℃, lower *RPU* is required and higher sleep current is generated.
2. The internal LDO with power saving mode is made by setting Addr. 0x81 and Data 0x03.
3. When *RPU* is open at internal LDO with power saving mode, you have to make hard reset and re-write initial registers at the time of waking up from sleep mode.

6.1.2. 32QFN Application Circuit – SPI Interface

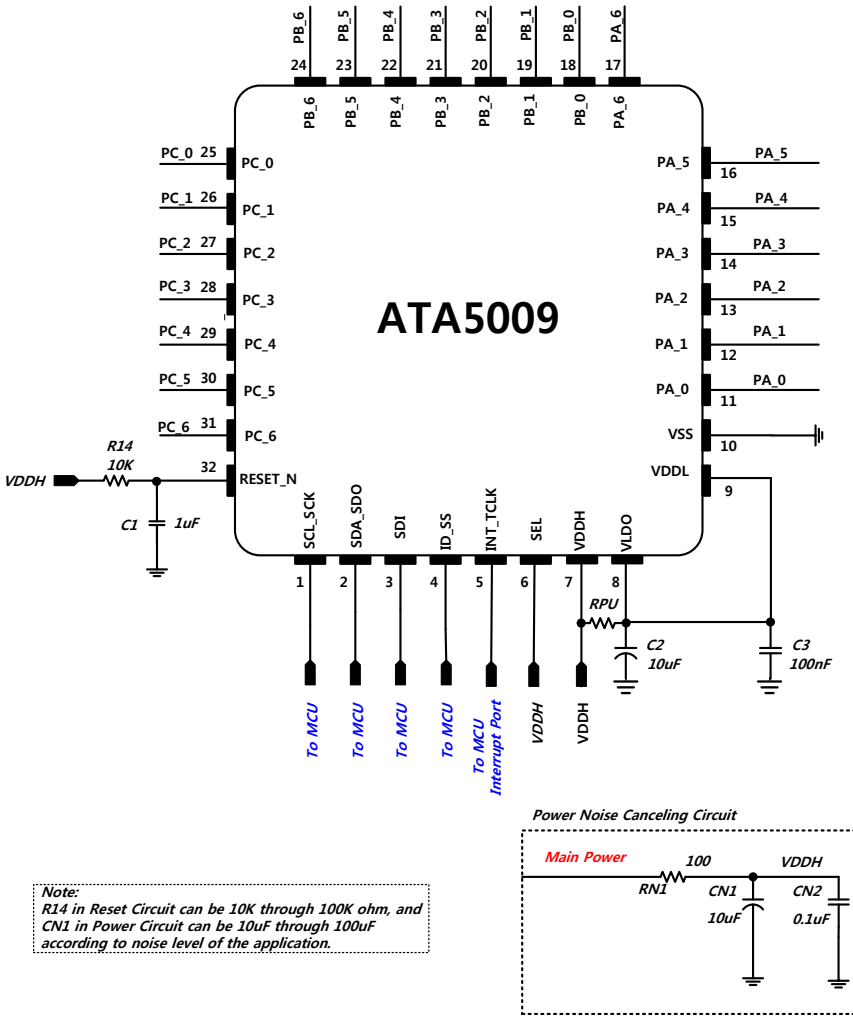


Figure 6-2. ATA5009_32QFN SPI Application Circuit

Operating mode	VDDH	*RPU*	Idle Current	Sleep Current	Remark
Use external 1.8V	2.1V ~ 5.0V	Open	22 µA	1.9 µA	set Addr. 0x81 and Data 0xFA
Use internal LDO w/o power saving mode	2.1V ~ 5.0V	Open	65 µA	40.0 µA	Internal LDO is always ON.
Use internal LDO w/ power saving mode	3.3V	50KΩ	33 µA	13.2 µA	See note 1 & 2
	5.0V	240KΩ	33 µA	13.5 µA	
	2.1V ~ 5.0V	Open	22 µA	1.9 µA	Need hard reset. See note 2 & 3

Note

1. *RPU* is recommended under the condition of maximum operating temperature of 80℃. If another maximum operating temperature is used, please ask ATLab for finding a proper *RPU* value. Generally speaking, for higher temperature than 80℃, lower *RPU* is required and higher sleep current is generated.
2. The internal LDO with power saving mode is made by setting Addr. 0x81 and Data 0x03.
3. When *RPU* is open at internal LDO with power saving mode, you have to make hard reset and re-write initial registers at the time of waking up from sleep mode.

6.1.3. Notes for application circuit

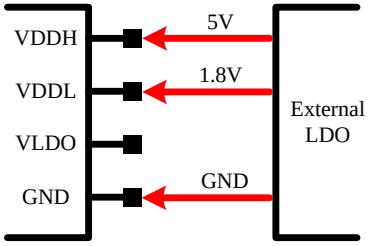
- The voltage of VDDH can be supplied between 1.8V and 5.0V.
- The application circuits depicted above uses an internal voltage regulator which supplies 1.8V to the core of the ATA5009.
- The reset circuit depicted in the application circuits is designed base on Power on Reset. If MCU wants to control RESET, connect RESET pin directly to MCU GPIO and delete R14 and C1.
- For the tuning purpose, zero-ohm resistors can be added optionally on two I2C lines. When tuning is required through ATLab's tuning kit, I2C connection between the host MCU and ATA5009 must be cut by eliminating zero-ohm resistors to avoid I2C bus collision initiated by both the host MCU and the tuning kit.
- 2K pull-up resistors in I2C interface can be omitted if the Host controller already uses them.
- 21 channels can be configured either sensor input pins or general purpose output ports. Sensor input pins also can be configured to (x,y) coordinate, touch sensing, current sensing, and ADC input pins.
- For the application to use I2C interface, The ATA5009 supports two different I^2C chip ID addresses to avoid address collision. If ID_SS pin is connected to LOW, its chip ID is 7'h64, If HIGH, it is 7'h66. However, 7'h65 or 7'h67 is reserved for internal purpose when chip ID is configured to 7'h64, or 7'h66, respectively.
- The ATA5009 supports three operation modes called Active mode, Idle mode, and Sleep mode. The power transition between Active mode and Idle mode is automatically performed by the ATA5009. However, it goes into Sleep mode by sending command through I^2C/SPI interface.

6.2. Guidelines for Power Connection

Make sure RESET_N(high voltage), SDA_SDO pull-up, SCL_SCK pull-up voltage are VDDH, otherwise leakage current may increased through P/N diode.

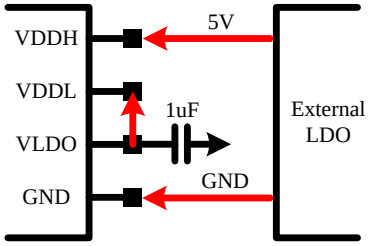
Case A.

VDDH: External 5V
VLDO: External 1.8V



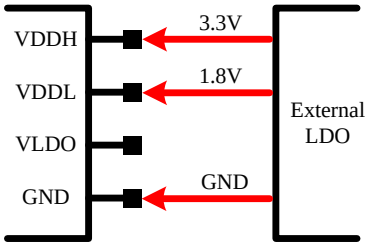
Case B.

VDDH: External 5V
VLDO: Internal LDO 1.8V



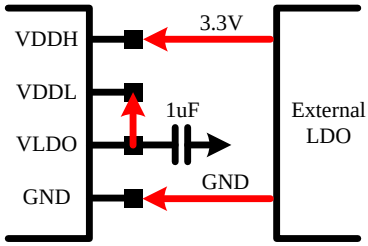
Case C.

VDDH: External 3.3V
VLDO: External 1.8V (Internal LDO off: Register Control)



Case D.

VDDH: External 3.3V
VLDO: Internal LDO 1.8V



Case E.

VDDH: External 1.8V
VLDO: External 1.8V

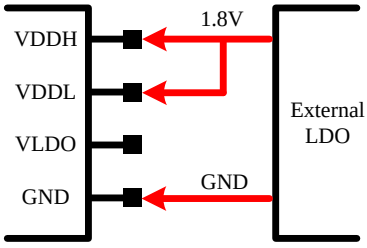


Figure 6-5. Various Power Cases

6.3. Guidelines for Power Sequence

To initialize the ATA5009 properly, please refer to the Power Sequence below when the power is given initially during boot-up. If the reset transition time during power on does not follow the time sequence below, the Internal oscillator would not operate normally. The Power Sequence is shown in the following example.

Power Connection

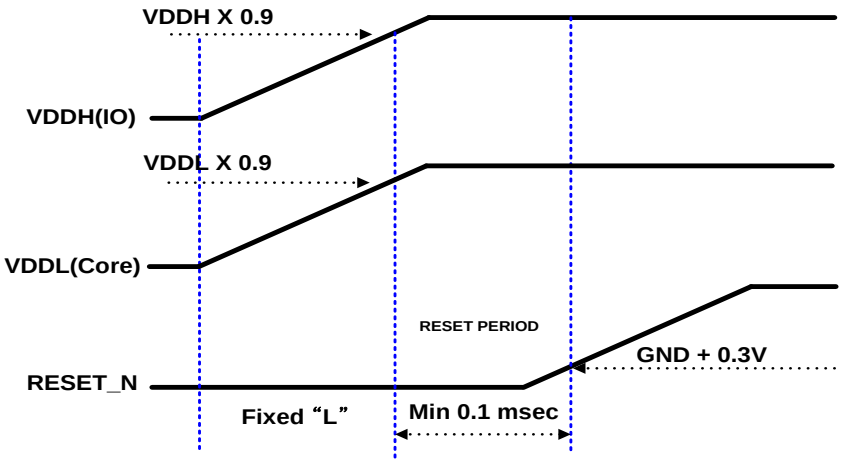


Figure 6-6. Power-on Sequence

In order to delay RESET_N transition about 0.1msec than VDDH transition, 10KΩ resistor and 1uF capacitor should be attached on RESET_N pin. Please see the typical application circuits described in the previous chapter. Also note that pulse width of RESET_N which is active low and generated by MCU must be longer than 10 msec to be valid RESET signal.

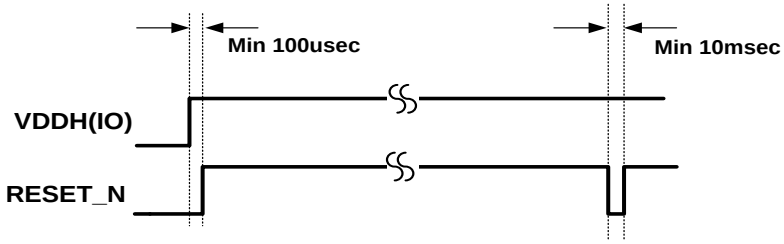


Figure 6-7. RESET_N Timing

6.4. Guidelines for Designing Touch Pads and PCBs

The important task in PCB design is to draw sensor lines to reduce influence from internal and/or external noise sources. The types of noise sources and suggested design guidelines are described next.

6.4.1. The Noise Influence by Other Chips

In touch module, we recommend that only the ATA5009 be mounted without any other chips because other chips can cause noise signals when controlling components such as LCD or Buzzer, etc.

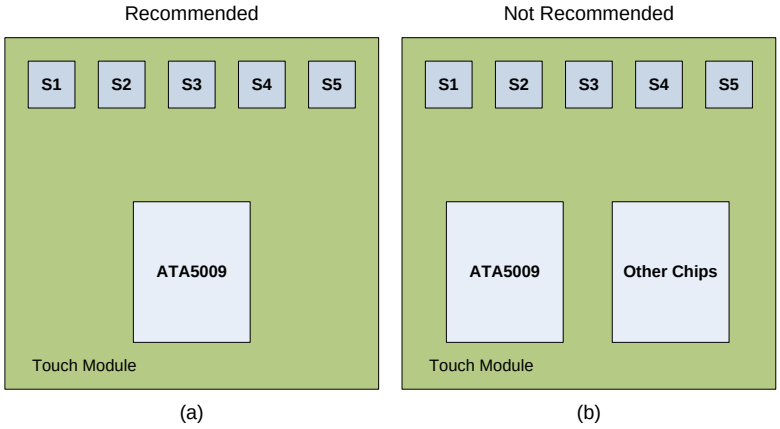


Figure 6-8. A Layout Example of Noise Influence

6.4.2. Cross Coupling Capacitance

Noise signal can be generated between sensor input lines. If sensor lines are not only very close to each other but also placed in parallel, they can become noise sources to each other. In order to avoid this, it is recommended to design sensor input lines as shown in Figure (a). Enlarge line spacing and make parallel portions as short as possible.

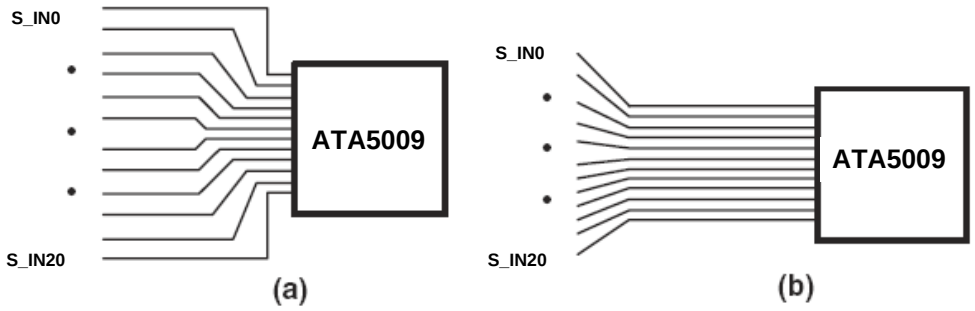


Figure 6-9. A Layout Example of Sensor Input Lines

6.4.3. Disposition of Data Lines and Sensor Input Lines

Figure 6-10(a) shows a problem caused by overlapping sensor input lines with data lines. For example, the capacitance generated by power lines with characteristics of consistent voltage output will not deeply affect sensor input lines. However, the capacitance generated by data lines fluctuating high and low voltage output will make sensor input lines unstable. Thus, the data lines in the front panel application should be placed closer to the connector in order to avoid undesirable influence on sensor input lines. Another important thing in layout design is that sensor input lines should be placed on the opposite side of data output lines. Finally, since overlapping data lines with sensor touch pads will be worse than overlapping data lines with sensor input lines, it is recommended that sensor pads should be apart from data lines.

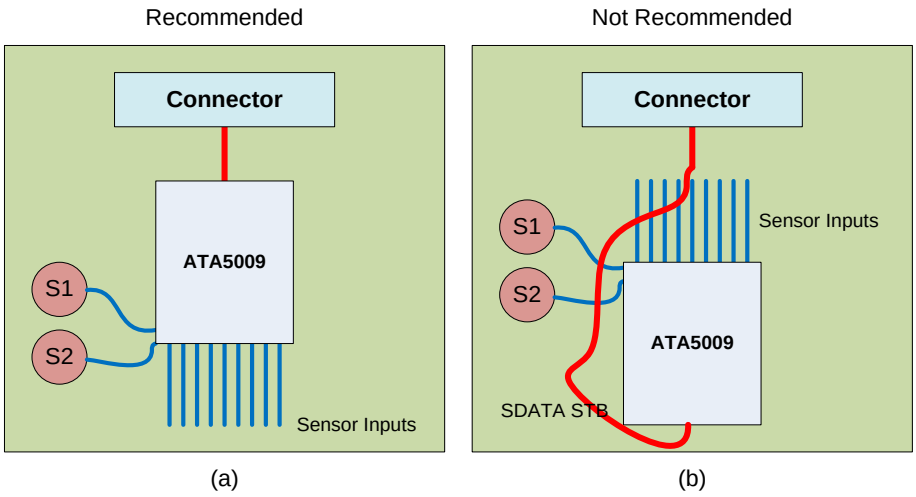


Figure 6-10. A Layout Example of Sensor Input Lines

6.4.4. LCD Control Signal Noise

If PCB which includes LCD control lines is located nearby touch PCB, it could be a noise source even if it is not on the same PCB. Therefore, you need to design PCB like Figure 6-11(a), which will be less affected by LCD control signal. Any kinds of pulse-type signals should be apart from the ATA5009 as far as possible.

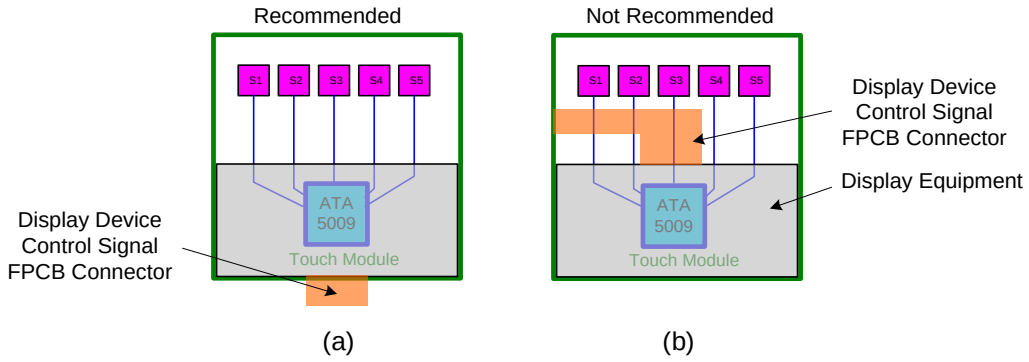


Figure 6-11. An Example of LCD Control Signal Noise

6.4.5. Charge Sharing

The sensitivity of the ATA5009 will be decreased if GND pattern is located close to the sensor input pads and lines because an electrical field generated by GND patterns will attenuate the strength of capacitance generated by the finger touch. This will decrease the sensitivity of the sensor input as shown in Figure 6-12(a). Although the GND pattern is used to reduce the interference of the lines, make sure to keep the GND pattern away from the sensor input pads as far as possible.

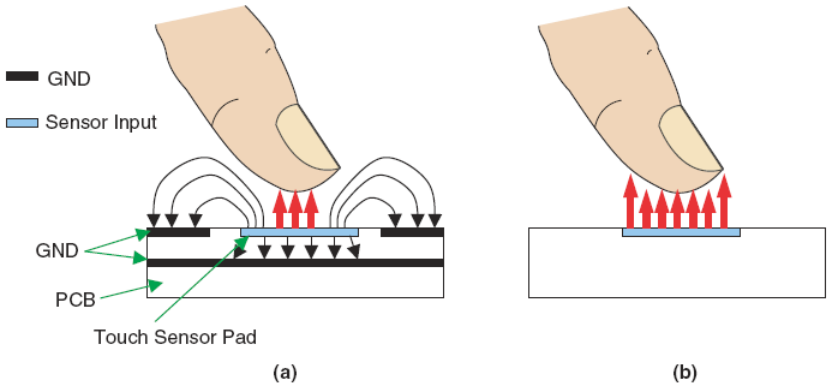


Figure 6-12. Charge Sharing by GND Pattern

6.4.6. Mismatch in Each Sensor Input

For a normal AIC function, each sensor input capacitance of the system should be within 140pF.

6.4.7. Large Sensor Input Pad

If a touch pad is bigger than 10mm x 10mm, it will become very sensitive to external environmental change. As a result, input impedance during no-touch could be unstable. In order to avoid this situation, it is recommended to use a pad layout as shown in Figure 6-13(b), which is exactly the same pad size as shown in Figure 6-13(a), but it will eliminate the problem by reducing real surface area.

Figure 6-13. Alternative Pad Patterning for a Large Sized Sensor Pad

6.4.8. Large touch pad size or long touch sensing line by shield

When touch pad size is very large or when sensing line is long, sensitivity is too high to control R_SEL registers (0x17 ~ 0x21). Note that ATA5009 detects any electro-magnetic field change with a relative low frequency (~ 100 KHz). For an example, metal protecting screen of electric fan is used as the sensing pad. Because of the low sensing frequency, total size of the protecting screen becomes the sensing pad size. Another example is to use a long sensing line. This is similar to use a long connecting wire between touch sensing pad and ATA5009. If your application should use a long connection line, then please make the connection line in shield.

If you want to decrease sensitivity, please decrease the sensitivity control resistor R3 as much as possible. But, the smallest value of R3 is not enough including the above cases. There is another way to decrease the sensitivity as shown in Figure 6-14. A capacitor CIN is added in serial to the touch sensor pad.

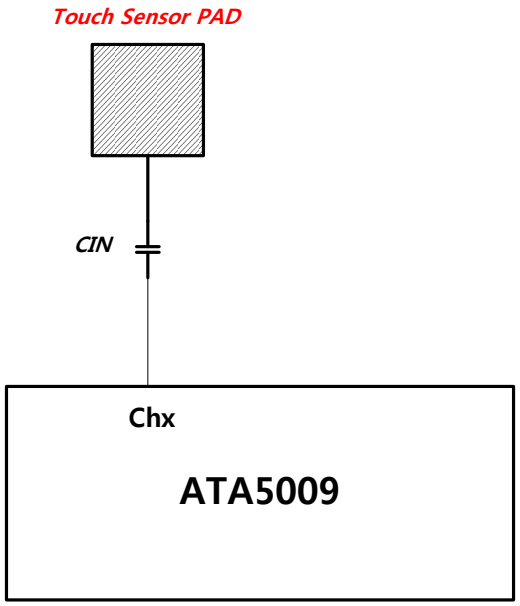


Figure 6-14. Decreasing Sensitivity by Adding a Capacitor

6.5. Guidelines for designing two dimensional PCBs and ITOs

6.5.1. Two dimensional PCBs

It will be supported separate files under NDA condition.

6.5.2. Two dimensional ITOs

It will be supported separate files under NDA condition.

6.6. Guidelines to use the sensor input channel as ADC

All sensing channel can be used as ADC by setting button mode. Figure 6-15 shows one example for 9-bit. Note that ATA5009 has 8-bit EPD and 10-bit P-value. Since there are overlap ranges between EPD and P-value, the maximum ADC resolution is about 15-bit. Figure 6-15 and 6-16 show about 9-bit ADC performance. ADC input range and resolution are programmable by setting register values, too. ATLab will provide supported seprate files. Please contact via dcc@atlab.co.kr.

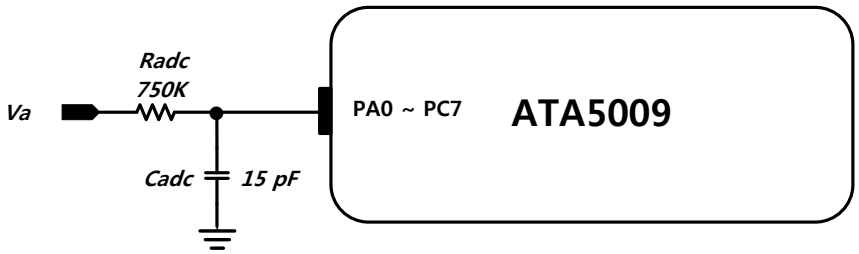


Figure 6-15. Application circuit for ADC

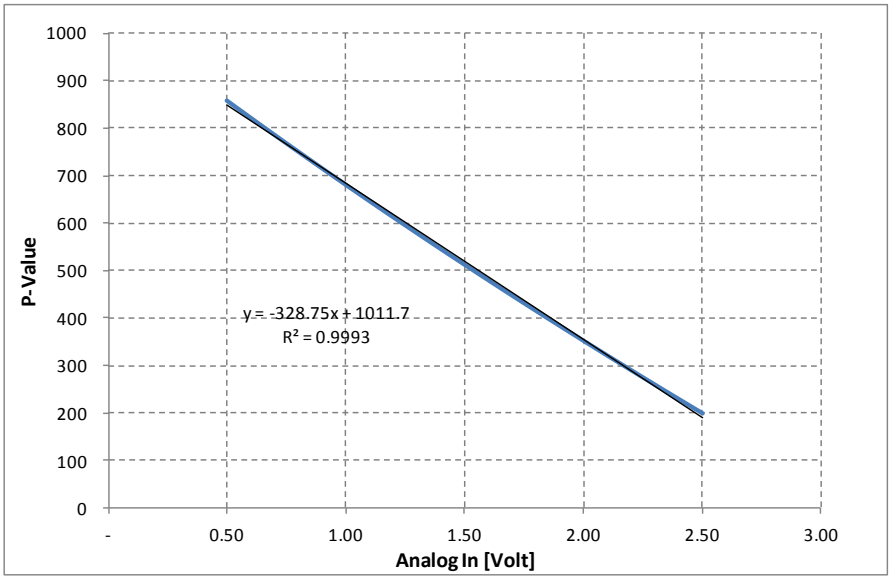


Figure 6-16. ADC Performance

7. Communication

7.1. Communication Protocol of I²C Bus

7.1.1. Basic Transfer Form of I2C Bus

ATA5009 follows all regulations defined by I²C specifications and it works in slave mode. Basic I²C communication protocol is composed of START → CHIP_ID → DATA → STOP and other various formats to be explained later. Since I²C is based on 8-bit serial communication, the host sends 8 clocks to the slave device and additional clock (the 9th) is sent as ACK/NAK for the purpose of checking communication error. If ACK/NAK bit is 'L', it means data transition was normal. After checking ACK is 'L', the host can send/receive another data or terminate data transition by a STOP signal. If ACK/NAK bit is 'H', some abnormal data transition occurred. The host must terminate the transition immediately by sending a STOP signal.

In general, the host initiates data transition by sending START signal to make SDA 'L' and sends 7-bit CHIP_ID with clocks. CHIP_ID is used to designate which device must react to the current host command. The 8th bit data indicates Read/Write operation and the 9th bit is ACK/NAK. Finally, the host will terminate data transition by sending a STOP signal to make both SDA and SCL 'H'. Therefore, SCL (clock line) and SDA (data line) remain in 'H' during none data transition.

The basic I²C data transition is shown in Figure 7-1, Figure 7-2, and Figure 7-3..

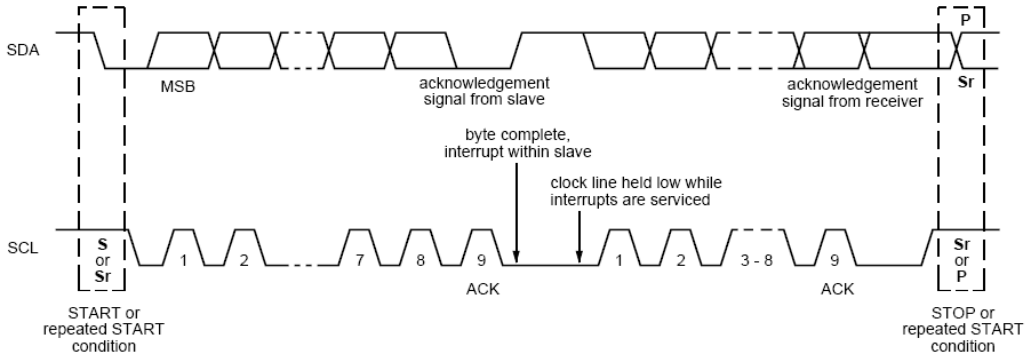


Figure 7-1. Basic Transfer Form of I²C Bus

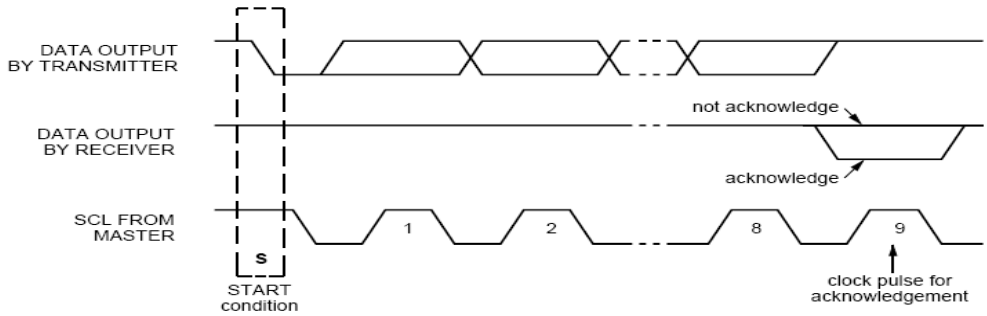


Figure 7-2. Waveform of ACK/NAK (the 9th bit)

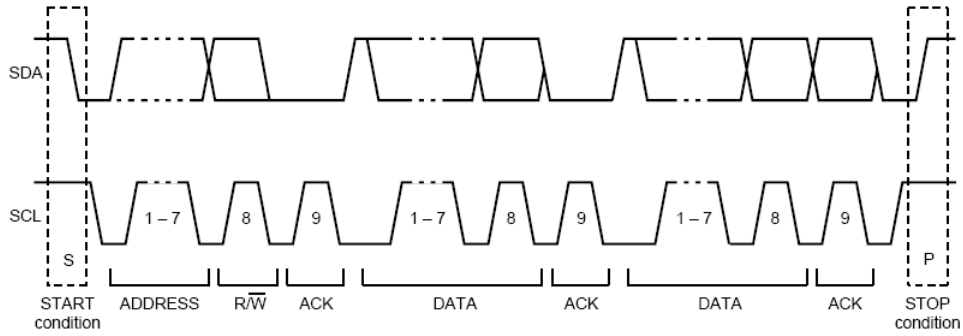


Figure 7-3. Continuous Data Transfer

7.1.2. Single Read Mode

The ATA5009 has more than 100 internal registers which consist of read/write registers, read-only registers and write-only registers. All these registers can be accessed by I^2C interface. Since each register has its own unique address, the host must send the write operation first to designate which register is to be read. The data format is shown below:

START → CHIP_ID(7) → WRITE(1) → ACK(1) → Register Address(8) → ACK(1) → **Repeated START** →CHIP_ID(7) → READ(1) → ACK(1) → DATA(8) → ACK(1) → STOP,

Where black-colored is driven by the host, red-colored is driven by the ATA5009 as a slave device, and (n) denotes the number of bits. Figure 7-4 describes the operation of single read mode.

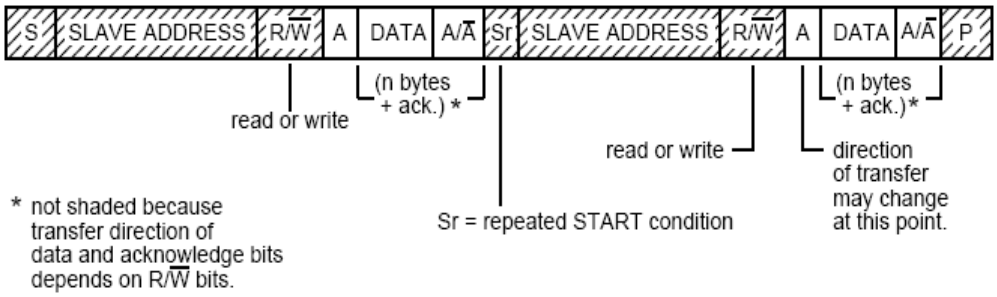


Figure 7-4: Data Format of Single Read Mode Using Repeated START

In the ATA5009, The data format below is also supported.

START → CHIP_ID(7) →WRITE(1) → ACK(1) → Register Address(8) → ACK(1) → STOP
START → CHIP_ID(7) →READ(1) → ACK(1) → DATA(8) → ACK(1) → STOP

7.1.3. Single Write Mode

Single Write Mode is not as complicated as Single Read Mode because data direction is always from the host to the ATA5009. The data format is shown below:

START → CHIP_ID(7) → WRITE(1) → ACK(1) → Register Address(8) → ACK(1) → DATA(8) → ACK(1) → STOP.

Where black-colored is driven by the host, red-colored is driven by the ATA5009 as a slave device, and (n) denotes the number of bits. Figure 7-5 describes the Single Write Mode.

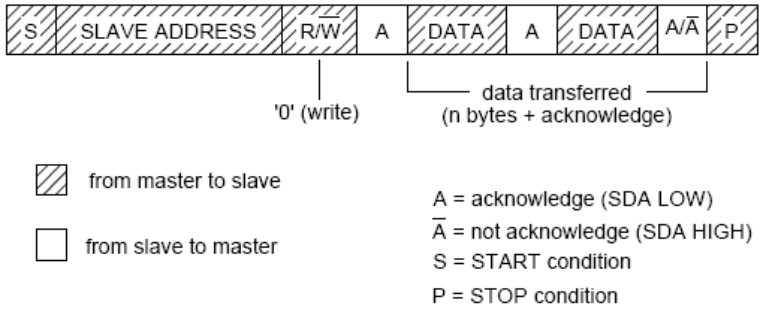


Figure 7-5. Data Format of Single Write Mode

In the ATA5009, The data format below is also supported.

START → CHIP_ID(7) → WRITE(1) → ACK(1) → Register Address(8) → ACK(1) → STOP
START → CHIP_ID(7) → WRITE(1) → ACK(1) → DATA(8) → ACK(1) → STOP

7.1.4. Burst Read Mode

In order to accelerate data transition, Burst Mode operation can be used. An important point with respect to this Burst Mode is that the addresses of registers to access must be consecutive. Register address is automatically incremented. In the case of Burst Read operation data transition format is shown below:

START → CHIP_ID(7) → WRITE(1) → ACK(1) → Starting Address of Register(8) → ACK(1) → **Repeated START** → CHIP_ID(7) → READ(1) → ACK(1) → DATA1(8) → ACK(1) → DATA2(8) →ACK(1) → … → DATAn(8) → ACK(1) → STOP.

7.1.5. Burst Write Mode

The Burst Write Mode is almost same as the Burst Read Mode except the data direction. The data format of Burst Write Mode is as below:

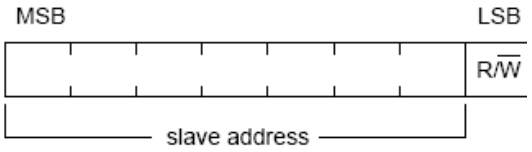
START → CHIP_ID(7) → WRITE(1) → ACK(1) → Starting Address of Register(8) → ACK(1) →DATA1(8) → ACK(1) → DATA2(8) → ACK(1) → … → DATAn(8) → ACK(1) → STOP.

7.1.6. Configuring Slave Address

The ATA5009 supports two different I²C addresses to avoid the bus conflict. As shown in Figure 7-6, when ID_SS pin is LOW, the I²C address is assigned with 0x64 and 0x65. Here, 0x65 is reserved for touchpad applications. Upon requests, ATLab will provide information further.

B6	B5	B4	B3	B2	B1	B0	Hex
1	1	0	0	1	0	0	64

(a)



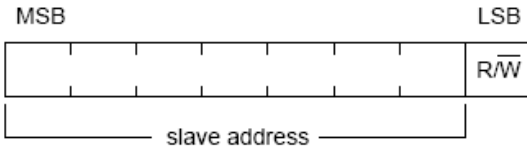
(b)

Figure 7-6. (a) I²C address (ID_SS=0), (b) Format of Slave Address with 7 bits

Similarly as shown in Figure 7-7, when ID_SS is HIGH, the I2C address is assigned with 0x66 and 0x67. Here, 0x67 reserved for touchpad applications. Upon requests, ATLab will provide information further.

B6	B5	B4	B3	B2	B1	B0	Hex
1	1	0	0	1	1	0	66

(a)



(b)

Figure 7-7. (a) I²C address (ID_SS=1), (b) Format of Slave Address with 7 bits

7.2. Communication Protocol of SPI Bus

The Serial Peripheral Interface Bus called SPI is a synchronous serial data link that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) signals. Sometimes SPI is called a "four wire" serial bus.

7.2.1. Interface

The SPI bus specifies four logic signals.

- SCLK — Serial Clock (output from master)
- MOSI — Master Output, Slave Input (output from master)
- MISO — Master Input, Slave Output (output from slave)
- NSS — Slave Select (active low; output from master)

Here, pin name of each SPI signal is that SCLK → SCL_SCK, MOSI → SDI, MISO → SDA_SDO, NSS → ID_SS, respectively.

7.2.2. Operation

The SPI bus can operate with a single master device and with one or more slave devices including ATA5009. If a single ATA5009 is used, the NSS signal may be fixed to logic low. Some slaves including multiple ATA5009 devices require the falling edge (high → low transition) of the slave select to initiate an action. With multiple slave devices, an independent NSS signal connection is required from the master for each slave device.

Most slave devices have tri-state outputs so their MISO signal becomes high impedance ("disconnected") when the device is not selected. Devices without tri-state outputs can't share SPI bus segments with other devices; only one such slave could talk to the master, and only its chip select could be activated.

7.2.3. Data Transmission

To begin communication with the ATA5009, the master should configure the clock (MCU) first using a frequency less than or equal to the maximum frequency that ATA5009 slave SPI device supports. The frequencies are commonly in the range of 500 KHz to 2 MHz. The master then pulls the ‘Slave Select’ low to access a slave SPI chip. During each SPI clock cycle, a full duplex data transmission occurs:

- the master sends a bit on the MOSI line and the slave gets it from the same line
- the slave sends a bit on the MISO line and the master gets it from the same line

Transmissions normally involve two shift registers whose length is 16-bits, one in the master and one in the slave, and they are connected in a ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave exchange register values. Then, each device takes that value and performs SPI operation such as writing it to the memory. If there are more data to exchange, the shift registers are loaded with new data and the process repeats.

Transmissions may involve any number of clock cycles. When there is no more data to transmit, the master stops toggling its clock. After that, normally it deselects the slave. The slave on the bus that is not currently activated by its ‘Slave Select’ line must disregard the input clock and MOSI signals, and must not drive MISO signals as well.

7.2.4. SPI Packet

The ATA5009 defines the Packet Structure of SPI as follows.

S	R/W	A/D	'0' (1 bit)	ADDRESS or Data (8-bit)	Reserved (3-Bit)	E

Data Type	Data Width	Description
Start	MOSI[15]	If this bit is '1', then it is a start of the Packet
Read or Write	MOSI[14]	This bit signifies the Read or Write type. 1: Read, 0: Write
Address/Data	MOSI[13]	This bit signifies the next 9-bit are Address or Data. '1': Address, '0': Data.
Reserved	MOSI[12]	This bit should be '0' always. "1" is used to access the other hidden registers.
Address or Data	MOSI[11:4]	These bits signifies Address or Data depending on MOSI[13]
Reserved	MOSI[3:1]	Don't care ('1' or '0')
End	MOSI[0]	For every SPI data transmission this bit should be '1' which signifies the End of the SPI packet. If not, SPI slave (ATA5009) discard the receiving data

Table 7-1. SPI packet

7.2.5. SPI Timing Diagram

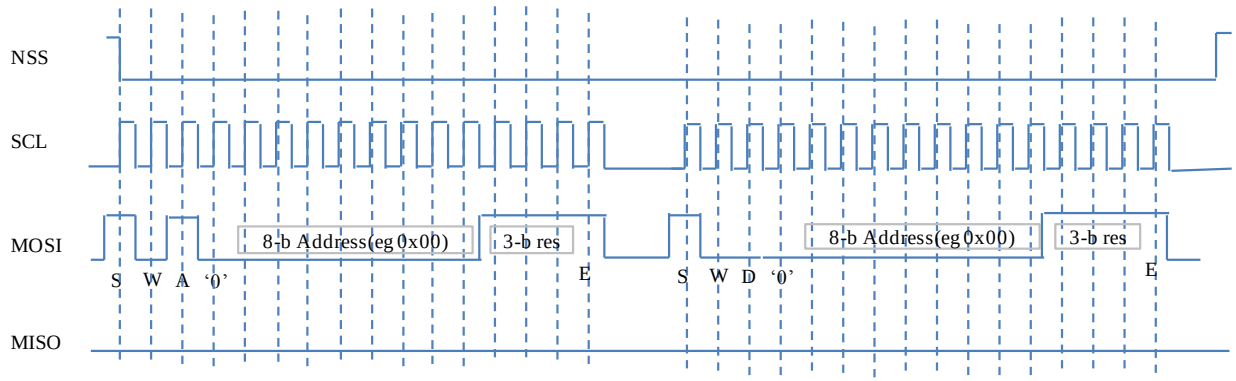


Figure 7-8: Single Data Write Cycle on SPI

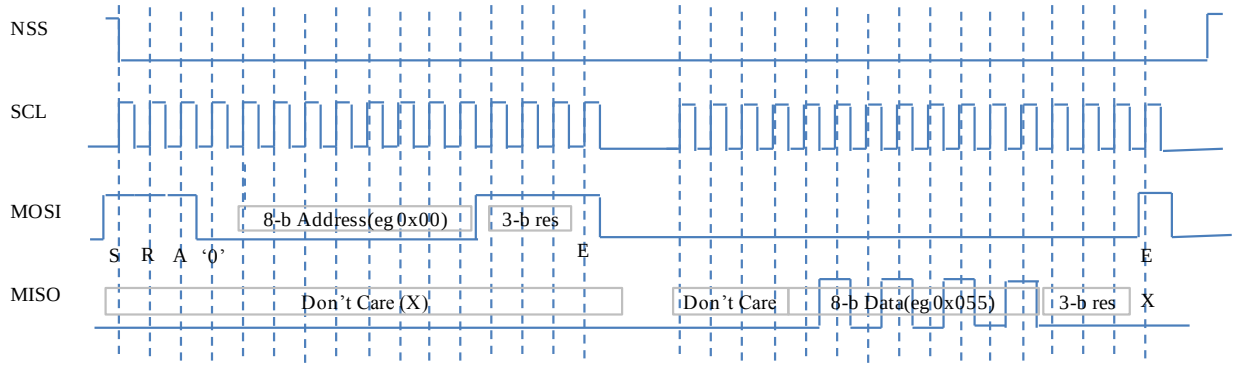


Figure 7-9: Single Data Read Cycle on SPI

7.2.6. SPI DC Characteristic

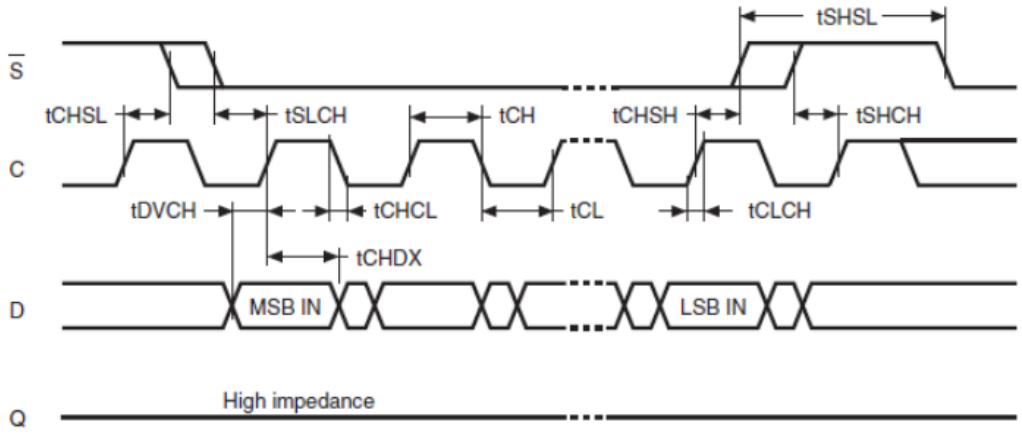
Symbol	Parameters	Min.	Max.	Units
VIL	Input Low Voltage	0.5	<VDDH	V
VIH	Input High Voltage	VDDH		V
VOL	Output Low voltage	VDDH-0.5		V
VOH	Output High Voltage		VDDH+0.5	V

Characterized value not tested in production.

7.2.7. SPI AC Characteristic

Symbol	Parameters	Min.	Max.	Units
fC	Clock Frequency	D.C	2	MHz
tSLCH	S active setup time	300		ns
tSHCH	S not active setup time	300		ns
tSHSL	S deselect time	400		ns
tCHSH	S active hold time	300		ns
tCHSL	S not active hold time	300		ns
tCH(1)	Clock high time	400		ns
tCL(1)	Clock low time	400		ns
tCLCH(2)	Clock rise time		4	us
tCHCL(2)	Clock fall time		4	us
tDVCH	Data in setup time	100		ns
tCHDX	Data in hold time	100		ns
tSHQZ(2)	Output Disable time		400	ns
tCLQV(3)	Clock low to output valid		400	ns
tCLQX	Output hold time	0		ns
tQLQH	Output rise time		400	ns
tQHQL	Output fall time		400	ns

7.2.8. SPI Serial Input Timing



8. Architecture

8.1. Block Diagram

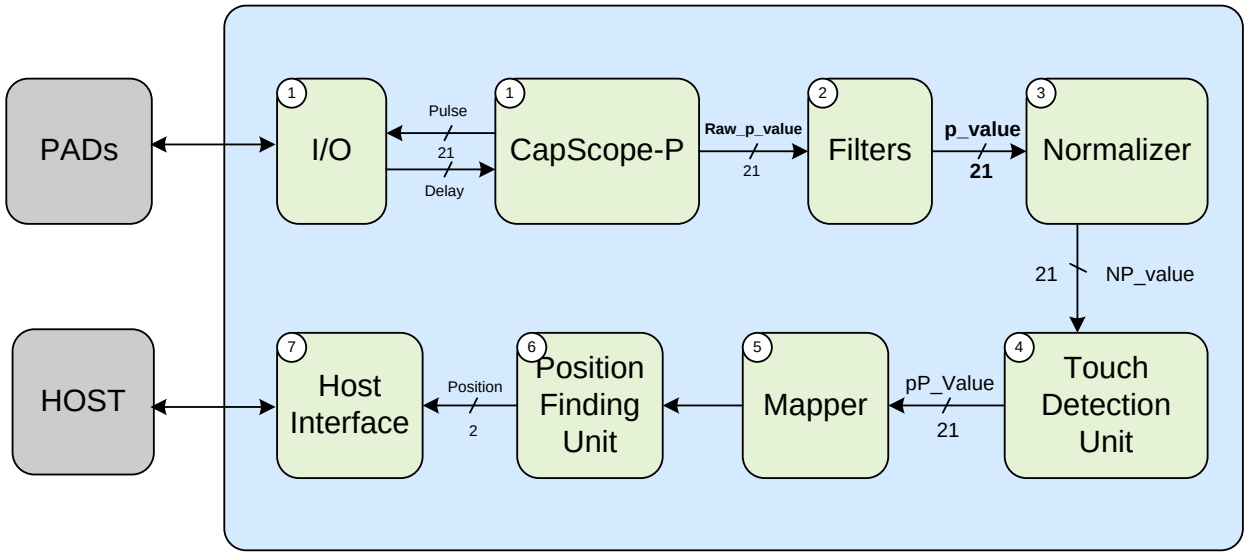


Figure 8-1. ATA5009 Block Diagram

8.1.1. PAD + I/O + Capscope-P

This block converts capacitance incoming through the sensor input pins to the digitized values. Sensor clock (typically 250KHz) is applied to this block and shared by all 21 channels. Offset capacitance and sensing capacitance induced by a finger generate RC time delay through I/O pad. CapScope-P measures this time delay and converts it to the digitized value called 'Raw_P_Value'.

8.1.2. Filters.

Filters eliminate high frequency noises included in Raw_P_Value and create the filtered digitized value called 'P_Value' which still has offset capacitance existing in the channel.

8.1.3. Normalizer.

Normalizer extracts capacitance induced by the human behavior from P_Value and outputs the normalized digitized value called 'nP_Value'.

8.1.4. Touch Detection Unit (TDU).

With the given sensitivity threshold, TDU determines the touch output of each channel by comparing nP_Value variations with the sensitivity threshold values. TDU also generates 'pP_Value' called purified P-Value to calculate position data by PFU.

8.1.5. Channel Mapper.

Channel Mapper determines sensor input pins either to belong 1D scroll, 2D coordinate or Button elements and routes each pin's position.

8.1.6. Position Finding Unit (PFU)

PFU calculates the position of 1D or the coordinates of 2D.

8.2. PAD + I/O + CapScope-P

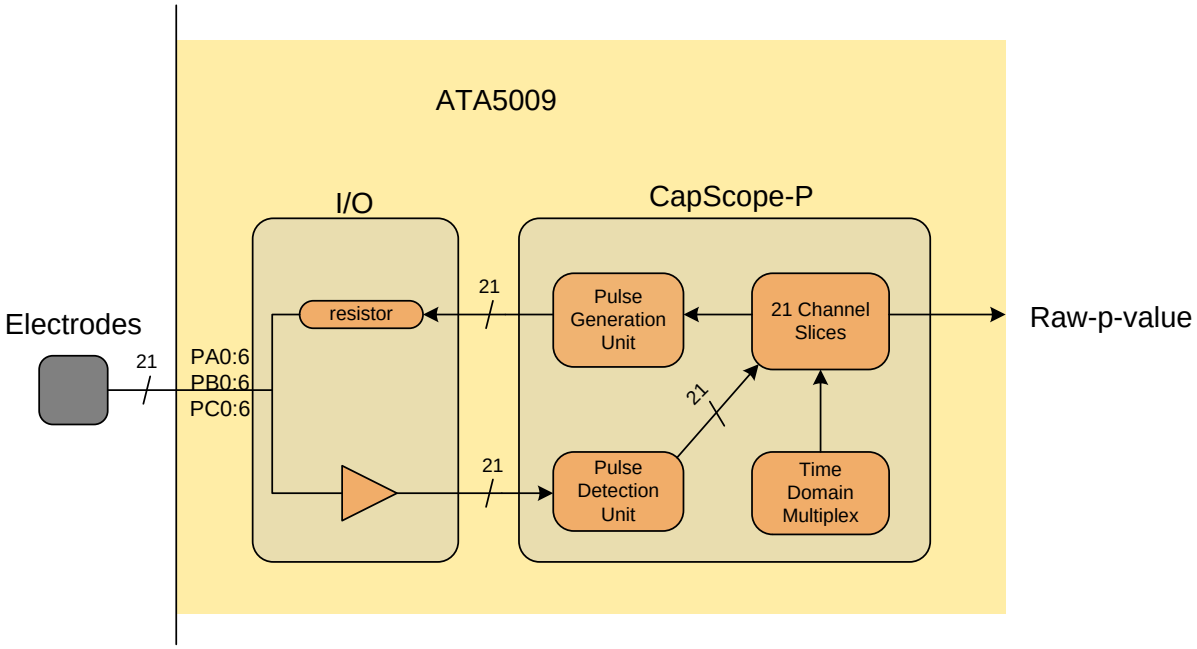


Figure 8-2. CapScope-P Core Internal Structure

There are four units in the CapScope-P block called Pulse Generation Unit, 21 Channel Slices, Time Domain Multiplex and Pulse Detection Unit. The Time Domain Multiplex (TDM) sends an enable signal sequentially to 21 Channel Slices (CS). The selected channel slice controls time to send/receive a defined width of pulse to/from pin PA0~PA6, PB0~PB6 and PC0~PC6 where touch electrodes are connected. Each channel slice becomes active when its enable signal goes high and calculates the amount of capacitance induced at the corresponding electrode by the method that Channel Slice sends the digitized value which determines the pulse width to the Pulse Generation Unit and receives information whether pulse exists or not from the Pulse Detection Unit. If PDU does not detect a pulse, then CS increases the digitized value for PGU to generate a wider pulse and transmits it to the electrode. On the other hand, if PDU detects a pulse, CS decreases the digitized value for PGU to make a narrower pulse. This digitized value is called Raw_P_Value and generated at every sensor clock period. Pulse Generation Unit (PGU) generates a pulse by receiving a digitized value which enables PGU to determine pulse width from the enabled channel slice. Pulse Detection Unit (PDU) determines whether a pulse exists or not on the selected channel by TDM.

There are I/O blocks between touch electrodes and CapScope-P. Each I/O block has a resistor which adjusts sensitivity resolution of a digitized value. In addition to the resistor, there is a buffer which makes a pulse passed R-C tank be logic High or Low under the condition of buffer's logic threshold.

The conversion formula between the capacitance existing on the electrode and the digitized value can be achieved by creating a circuit that the digitized value CS calculates is equal to the I/O RC time delay between the electrode and sensor input.

$$\text{I/O Time Delay} = 1.2 \times R \times C, \text{whre R is a resistor in I/O}$$

$$\text{Time Delay calculated by the ATA5009} = \text{Fine unit delay} \times N, \text{where N is a digitized value}$$

$$C = \frac{\text{Fine Unit delay} \times N}{1.2 \times R}$$

A way to create a pulse which is equal to IO time delay is based on the theory that I/O buffer has no output when PGU generates a narrower pulse than I/O Time delay but it has an output when PGU creates a wider pulse than IO Time Delay. With this theory, PGU generates a wider pulse if no signal is detected at PDU, vice versa. Meanwhile, PGU output becomes I/O Time Delay.

8.3. Filters

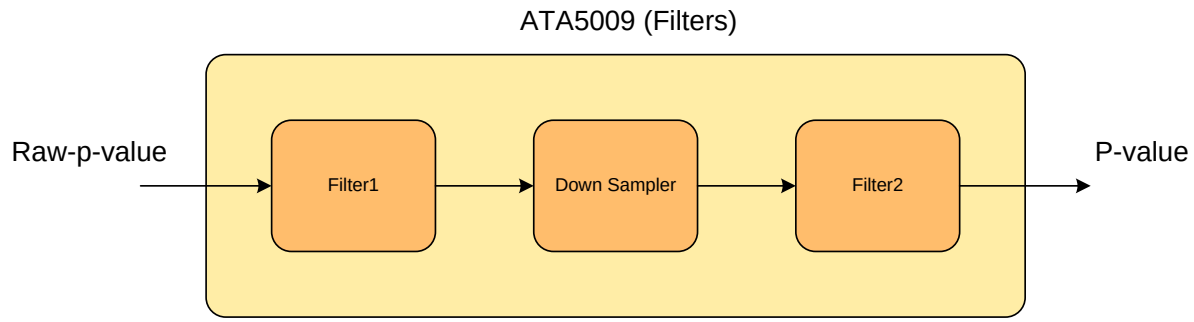


Figure 8-3. Filters Structure

Filter1 and Filter2 are 1st order IIR filters and its cutoff frequencies are 60 Hz ~ 6 KHz and 30 Hz ~ 3 KHz, respectively. 6 KHz is a default value of Filter1 and 3 KHz is a default value of Filter2. Cutoff frequencies of Filter1 and Filter2 are individually controllable by changing values of corresponding registers. To set smaller values to the registers makes cutoff frequencies more reduced.

The 1st order IIR filters are working based on the equation below.

$$Y[n] = \frac{coeficient}{32} X[n] + \left(1 - \frac{coeficient}{32}\right) Y[n-1]$$

Down Sampler located between Filter1 and Filter2 controls data rate of position data by controlling Filter2's cutoff frequency. Its default value is 12 and sets data rate of position data to 1000 times per second (1 KHz).

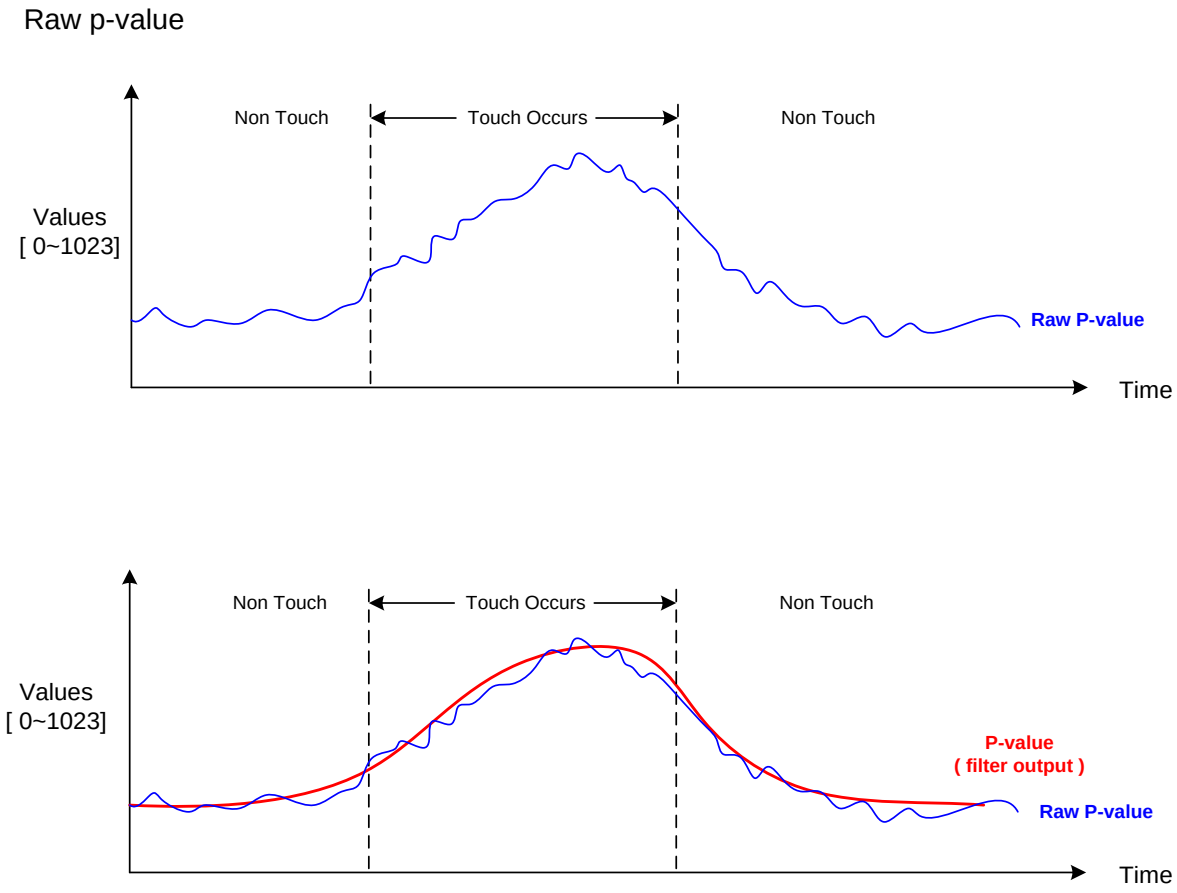


Figure 8-4. Filters effect.

Filters eliminate high frequency noises included in Raw-P-Value. In addition to these filters, the ATA5009 also provides the method to eliminate low frequency noises by adjusting the system clock. Low frequency noise appears by an aliasing effect when higher frequency noise than sampling frequency of the ATA5009 are aliased. Moreover, the ATA5009 is able to eliminate various noises with different frequencies and amplitudes. For example, high amplitude noises usually drops sensitivity of input channels so that the ATA5009 can handle these noises by adjusting sensitivity of each channel. The noises having similar frequencies with the sampling clock of the ATA5009 can be eliminated by adjusting the system clock of the ATA5009.

In the application, P-Values, the output of filters, should vary less than 5 or 6 codes under the condition of the perfect filters' setting during non-touch status when you monitor P-Values through the tuning view program. If fluctuation of P-Values is greater than 5 or 6 codes, the filter coefficients must be modified. In addition, adjusting the system clock or adding external RC filters to the sensor input lines can be required if adjustment of filter coefficients is not good enough.

Lower filter coefficient value(0x01~0x1f) make filter's cut-off frequency low. Then noise will be reduced and response will be delayed.

8.4. Normalizer

The input of Normalizer called P-Value includes various capacitances such as offset capacitance existing touch electrodes, PCB patterns, internal capacitance in the sensor input pins, remaining noise not filtered by the filters, the capacitance induced by human behavior (here after called sensing capacitance or nP-Value), and other capacitance by environmental change. The Normailizer has a role to extract sensing capacitance from P-Value as accurate as possible.

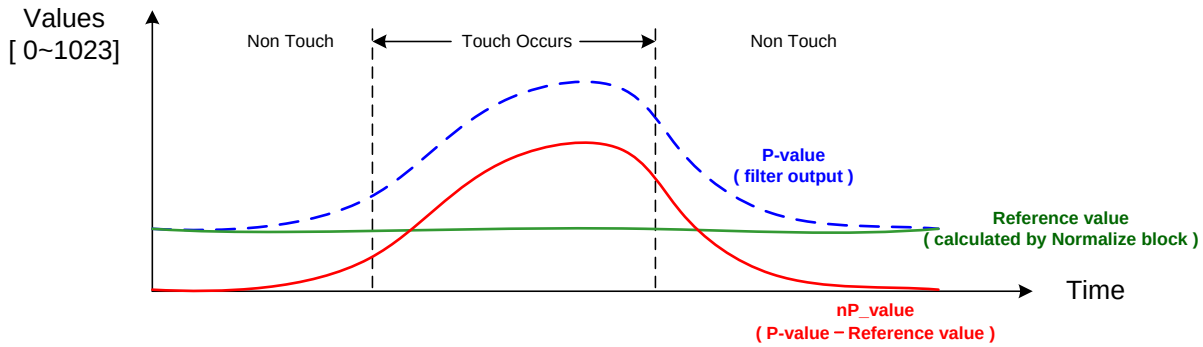


Figure 8-5. Normalizer effect.

In the Figure 8-5, the green horizontal line is called the reference value calibrated by the Normalizer. The reference value contains all noisy capacitances except the sensing capacitance. Therefore, nP-Value is obtained by subtracting the reference value from P-Value in each input channel. To get more accurate nP-Values, it is important to calibrate more accurate reference values. This process is called calibration. Calibration is one of major blocks in the ATA5009 and provides many internal registers to tune to get accurate reference values. The registers below are related with the Normalizer. To see more detail how to control these registers, please refer to the register section of this manual.

- Active calibration interval / Idle calibration interval
- Idle calibration duration
- Calibration waiting time
- Beta_PA0 ~ Beta_PC6 / Beta global

During calibration by the Normalizer, two criteria are used to check the reference update for each channel called Beta and Beta_global. Beta is assigned for each channel and defined at Beta_PA0 through Beta_PC6 registers. Beta defines the minimum variation of raw P-value generated by a human touch. If raw P-value variation is greater than Beta, the ATA5009 assumes that the corresponding channel is in touch condition and stops calibration to avoid fault calibration. If raw P-value variation is less than Beta, the ATA5009 assumes that the channel is not in touch condition and performs calibration when other calibration criteria are also met. Through the experiments under various environments, calibration is well performed when Beta is set to around 5 to 6 or about 50% to 80% of raw P-value variation under non-touch condition.

Beta_global is the first criterion for calibration to prevent a fault calibration. In the case of 1D or 2D applications where the touch electrodes are very closely located, touching a channel increases not only its own P-Value but also P-Values of neighbor electrodes. If increased P-Values of neighbor electrodes are less than their Beta, possibly fault calibration can be performed. Therefore, Beta_global is used when P-Value of at least one channel is greater than Beta_global to prevent any channel's fault calibration. The recommended Beta_global value is twice of average Beta value through maximum Alpha value. Note that "Alpha value" is equal to "touch threshold".

8.5. Touch Detection Unit.

Touch Detection Unit (TDU) has two roles. One is to determine whether channels are touched or not with given touch-threshold values defined at internal registers. If nP-Value of each channel is greater than its touch-threshold value, TDU generates touch output of that channel. The other is to create *'pP_Value'* and to transmit them to the Position Finding Unit. pP_Value is obtained by subtracting Noise Threshold from nP-Value. Noise Threshold values are defined at the registers.

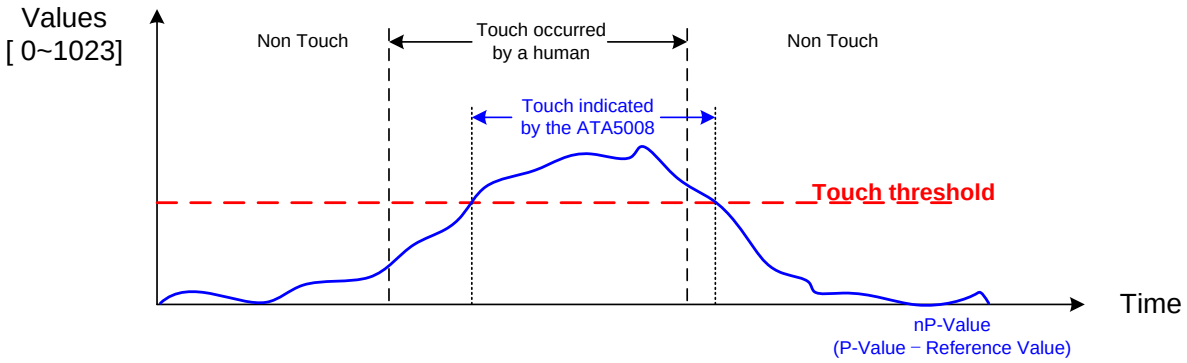


Figure 8-6. Touch-Threshold effect.

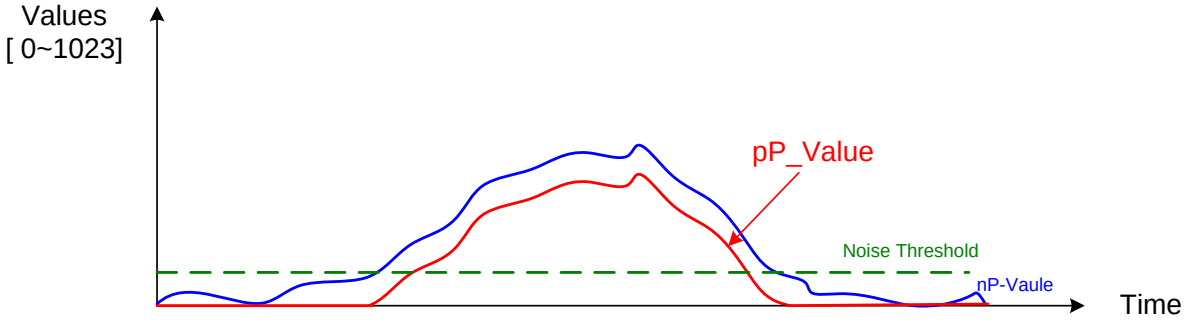


Figure 8-7. Noise Threshold effect.

Figure 8-6 shows the first role in the TDU where the ATA5009 detects touch when nP-Value overcomes Touch Threshold value. Figure 8-7 shows the second role in the TDU where the red colored line describes pP_Value. It is noticed that pP_Value becomes zero during nP-Value is less than Noise Threshold value.

Noise Threshold can eliminate potential noises within nP-Value unfiltered by the Normailizer and also reduce proximity effect. Providing pP_Value to Position Finding Unit enables PFU to calculate positions more accurately. Usually Noise Threshold values are given about twice of Beta values.

8.6. Mapper

The ATA5009 has twenty one input channels divided by three groups; PA0~PA6, PB0~PB6 and PC0~PC6. The mapper defines each channel as one of three functional categories: **Position**, **Button,** or **GPO** (General Purpose Output). The registers related with the Mapper are 0x55 through 0x69 and the format of Mapper registers is shown below. Here, '**Position**' is used for touchpad or scroll bar, '**Button**' is for capacitive sensing, current sensing, or ADC, and '**GPO**' is for logic-level output.

Bits	Width	Default Value	Description
7	1	0x1	Configures a channel to be either a sensor input or a digital output 0: The channel is configured as a digital output. 1: The channel is configured as a sensor input.
6	1	0x1	Assigns a sensor input to the position or button application. 0: The channel is used as an element of a button application. 1: The channel is used as an element of an 1D or 2D application. Only valid when bit7 is 1.
5	1	0x0	Declares a sensor input port to be used as an element of X or Y. 0: X, 1: Y Only valid when both bit6 & bit7 are 1.
4:0	5	0x00	Channel Mapping Number, 0~ 20. Only valid when both bit6 and bit7 are 1.

Table 8-1. Format of mapping registers

All channels should be mapped uniquely and any duplicated mapping causes an error. The table 8-2 below is an example of mapping where PA0~PA6 and PB0 are defined as elements of X axis, PB1~PB5 are defined as elements of Y axis, PC0~PC3 are defined as elements of button inputs, finally PB6 and PC4~PC6 are defined as elements of general purpose outputs.

channel	function	register address	register value
PA_0	X_7	0x55	0xC7
PA_1	X_6	0x56	0xC6
PA_2	X_5	0x57	0xC5
PA_3	X_4	0x58	0xC4
PA_4	X_3	0x59	0xC3
PA_5	X_2	0x5A	0xC2
PA_6	X_1	0x5B	0xC1
PB_0	X_0	0x5C	0xC0
PB_1	Y_0	0x5D	0xE0
PB_2	Y_1	0x5E	0xE1
PB_3	Y_2	0x5F	0xE2
PB_4	Y_3	0x60	0xE3
PB_5	Y_4	0x61	0xE4
PB_6	GPO	0x62	0x00
PC_0	Button	0x63	0x80
PC_1	Button	0x64	0x80
PC_2	Button	0x65	0x80
PC_3	Button	0x66	0x80
PC_4	GPO	0x67	0x00
PC_5	GPO	0x68	0x00
PC_6	GPO	0x69	0x00

Table 8-2. An Example of Registers' Channel Mapping

Figure 8-8 describes how the registers' mapping configuration is physically connected to the channels of the ATA5009.

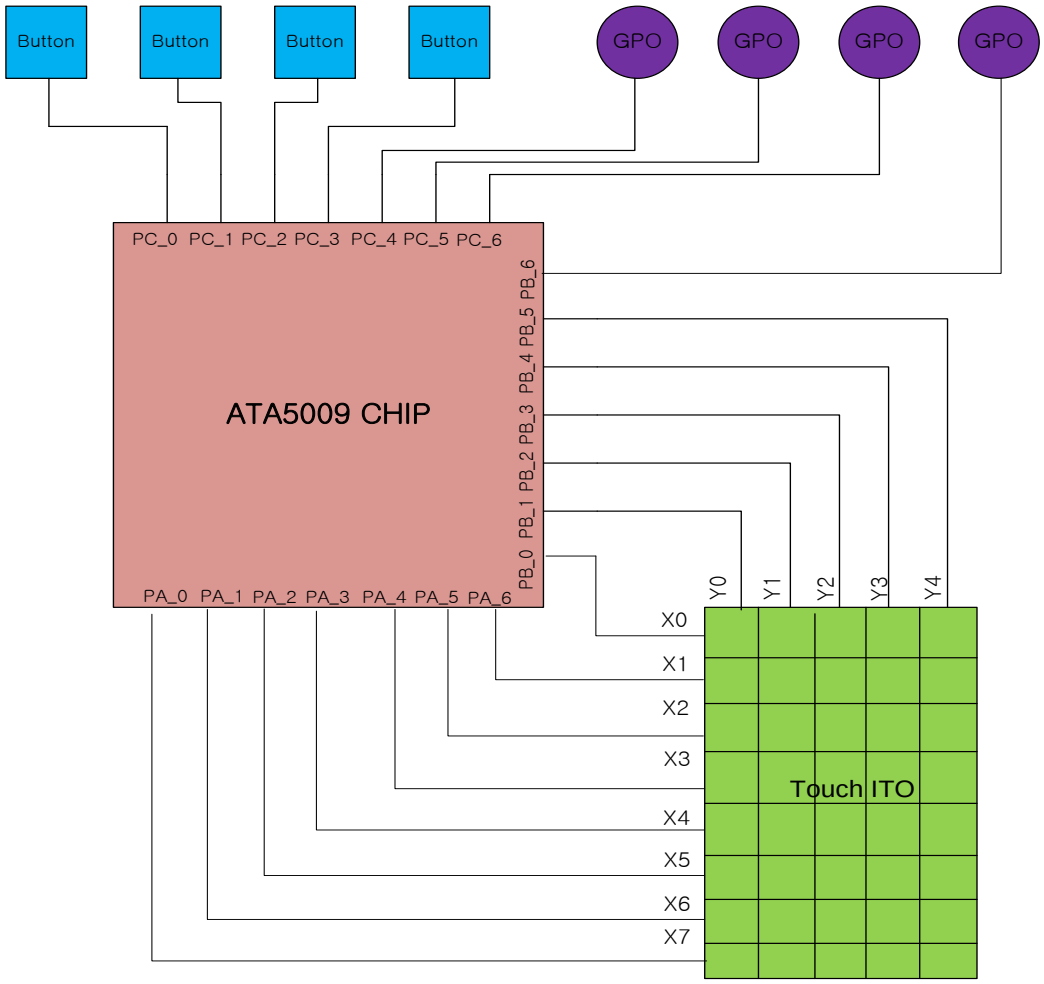


Figure 8-8. An Example of Physical Channel Mapping

8.7. Position Finding Unit

Position Finding Unit (PFU) calculates interpolated coordinates using 10-bit pP_Value generated by Touch Detection Unit. There are two independent clusters called X and Y to define where channels belong to. PFU only works when at least one channel is assigned as an element of a cluster. In order to create virtual coordinates with the limited number of physical channels, a target coordinate resolution is required. There are four registers to determine a target coordinate resolution called Resolution_X (0x6A, 0x6B), Resolution_Y (0x6C, 0x6D), Num_Channel_X (0x7A), Num_Channle_Y (0x7B). For example, if a target coordinate resolution is 320x320, both Resolution_X and Resolution_Y are assigned with the value of 320. The maximum value of both Resolution_X and Resolution_Y is 2047. Num_Channel_X and Num_Channel_Y define the number of physical channels assigned to X axis and Y axis, respectively.

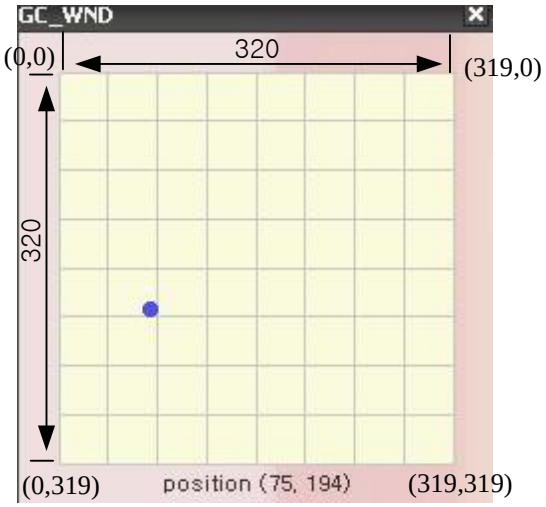


Figure 8-9. Interpolated 320x320 resolution on 8x8 2D Touch Screen

According to Figure 8-9, the interpolated coordinate calculated by PFU is (75,194) on 2D touch screen which has 8 physical X channels and 8 physical Y channels. When both X and Y clusters are defined in 2D touch screen, there can be a week touch in the edge area which can generate a miscalculated coordinates. To eliminate a week touch in the edge area, 'Cluster_Threshold (0x29)' is used. Cluster threshold is valid only when both X and Y clusters are defined. If only one cluster has elements and the other has no element, Cluster Threshold should not be used. The control of cluster threshold is done by bit4 of Control2 register (0x71).

ATA5009 supports single-touch. If there are more than 2-touch, then PFU generates average X/Y values from all touch positions. This feature enables to implement a touchpad gesture. You can implement by monitoring TOUT registers (0xBA, 0xBB, 0xBC) and by changing the absolute position to a relative position. When there is more than 1 touch output, then you can recognize multi-touch and determine gestures from pattern of touch outputs and change of the touch position. If need more information, please mail to dcc@atlab.co.kr.

8.8. Interrupt

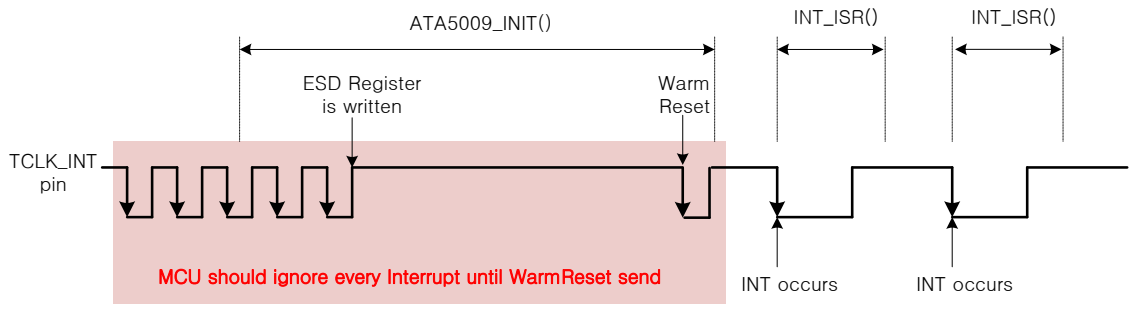
This chapter describes how to use interrupt signals from INT_TCLK pin by host MCU.

The interrupt output pin INT_TCLK can be configured into either TINT(touch strobe interrupt) or GINT (general purpose interrupts). INT_SEL bit(bit4) of CONTROL1(0x70) Register is used to configure the INT_TCLK pin. By default setting, the INT_SEL is *LOW* and the INT is configured to GINT.

For a recovery from a power failure (due to such as ESD) the INT_TCLK pin is periodically toggled at power up until the “ESD Power Failure Detection Register” is set by a host.

TINT is generated by change of either touch or position only.

INT_MASK is valid for GINT only. (not TINT)



8.8.1. TINT

Configuration Flow of Host startup

1. To use TINT set Bit4 of 0x70 to ‘High.
2. Determine interrupt polarity to be either falling edge or rising edge for host MCU to detect. It can be set by Bit1 of 0x70 register called Control1 register. If this bit is set to ‘L’, interrupt polarity is set to rising edge and vice versa.
3. Write 0xFF first to 0x78 register called ESD_Check register during MCU initializes the registers of the ATA5009. It will keep generating interrupt signal through INT_TCLK pin as the form of clock signal until 0xFF is written into ESD_Chcek register. All interrupts generated before ESD_Check register is set should be discarded by host MCU.
4. Write other registers with appropriate data during initialization.
5. Give Warm Reset (0xFF address) to the ATA5009. Be sure that all interrupts generated before giving Warm Reset should be ignored by host MCU. The interrupts after giving Warm Reset should be handled normal and interrupt service routine must be performed as the flow chart below in the Figure 8-10.
6. Interrupt signal automatically goes back to original level as soon as MCU reads appropriate registers. If button Application, MCU should read TOUT_PA~TOUT_BC (0xBA~0xBC), else if 2-D application, MCU should read POSITON registers (0xBF~0xC2). Figure 8-11 shows the change of interrupt polarity of TINT whose polarity is set to rising edge during initialization of the ATA5009.

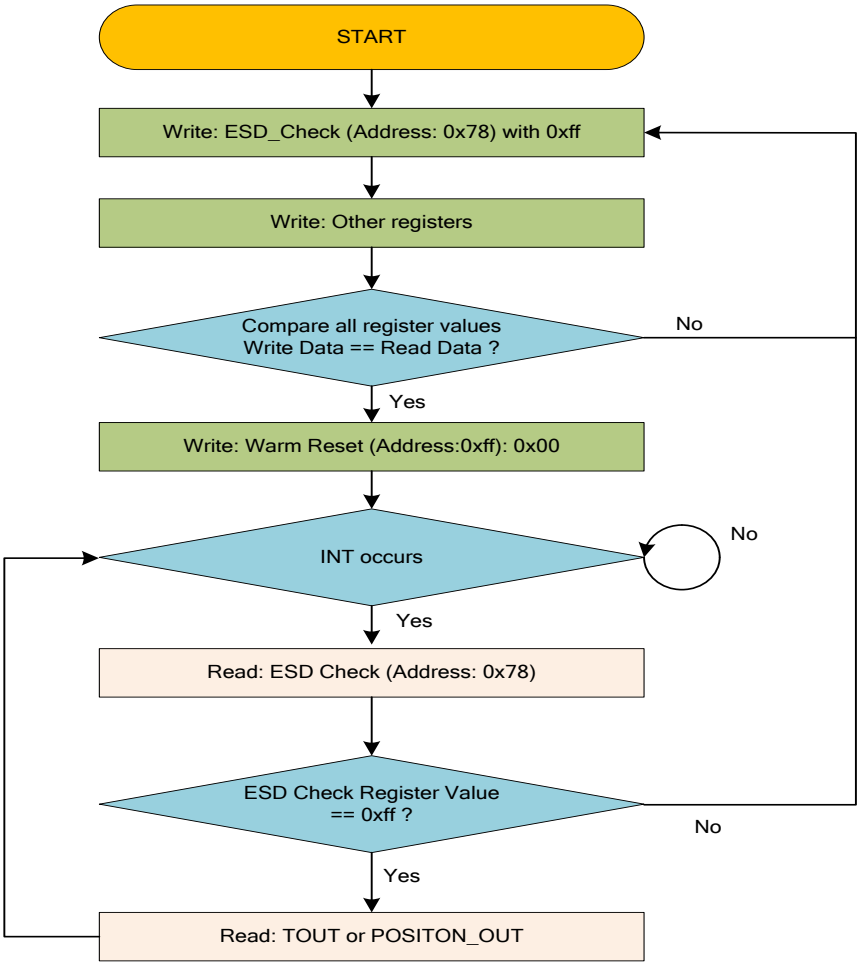


Figure 8-10: Flow Chart of TINT Service Routine

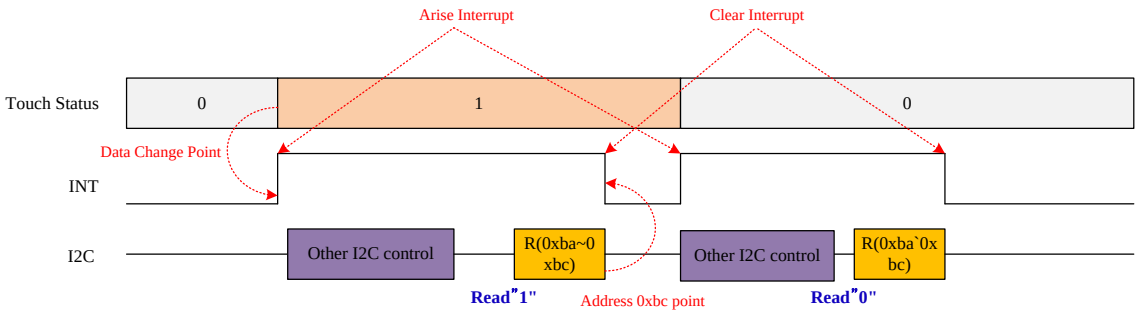


Figure 8-11: Example of TINT Signal

8.8.2. GINT

Configuration Flow of Host startup

1. To use GINT, set Bit4 of 0x70 to ‘LOW.
2. Determine interrupt polarity to be either falling edge or rising edge for host MCU to detect. It can be set by Bit1 of 0x70 register called Control1 register. If this bit is set to ‘L’, interrupt polarity is set to rising edge and vice versa.
3. Write 0xFF first to 0x78 register called ESD_Check register during MCU initializes the registers of the ATA5009. It will keep generating interrupt signal through INT_TCLK pin as the form of clock signal until 0xFF is written into ESD_Chcek register. All interrupts generated before ESD_Check register is set should be discarded by host MCU.
4. Write other registers with appropriate data during initialization.
5. Give Warm Reset (0xff) to the ATA5009. Be sure that all interrupts generated before giving Warm Reset should be ignored by host MCU. The interrupts after giving Warm Reset should be handled normal and interrupt service routine must be performed as the flow chart below in the Figure 8-12.
6. Interrupt signal GINT automatically goes back to original level when MCU performs below GINT service routine.
    a. MCU reads INT_PENDING_REG (0xed).
    b. MCU writes 0x1F to INT_MASK register (0x76).
    c. If INT_PENDING is 0x01, then MCU reads TOUT_PA~TOUT_PC (0xBA~0xBC),
    if INT_PENDING is 0x02, then MCU reads POSITON registers (0xBF~0xC2),
    if INT_PENDING is 0x03, then MCU reads TOUT_PA~TOUT_PC (0xBA~0xBC) and POSITON registers (0xBF~0xC2).
    d. MCU writes 0x1F to INT_CLEAR register (0x77).
    e. Host writes 0x18 to INT_MASK register (0x76).

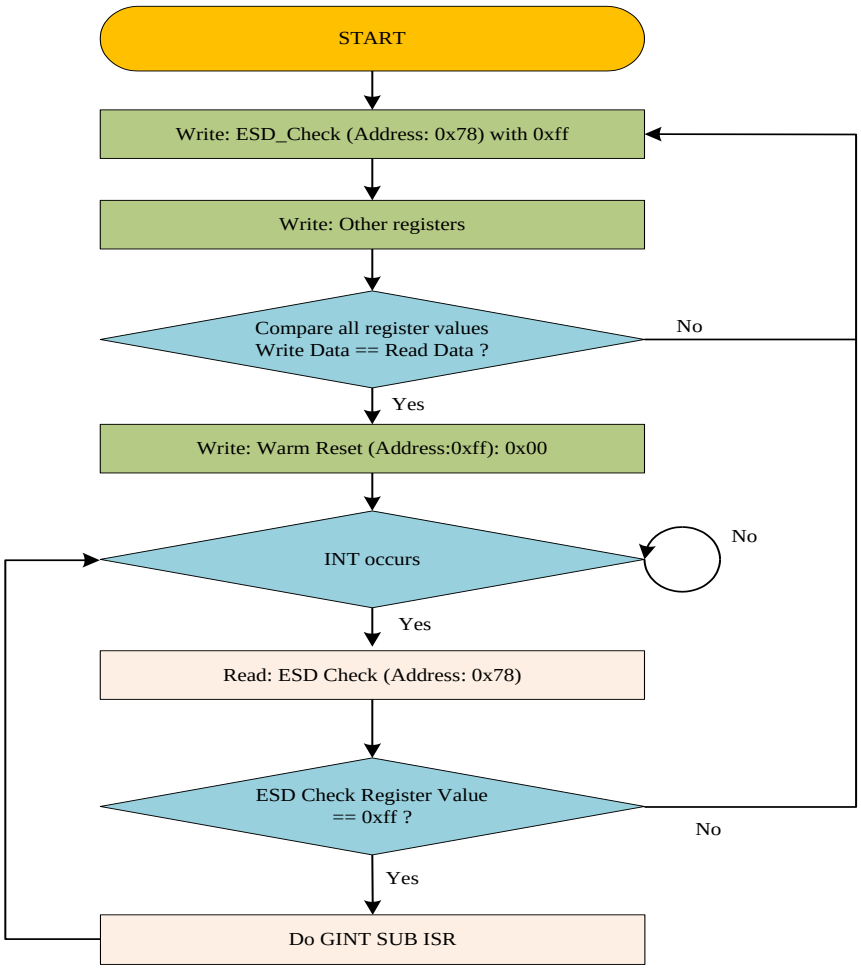


Figure 8-12: Flow Chart of GINT Service Routine

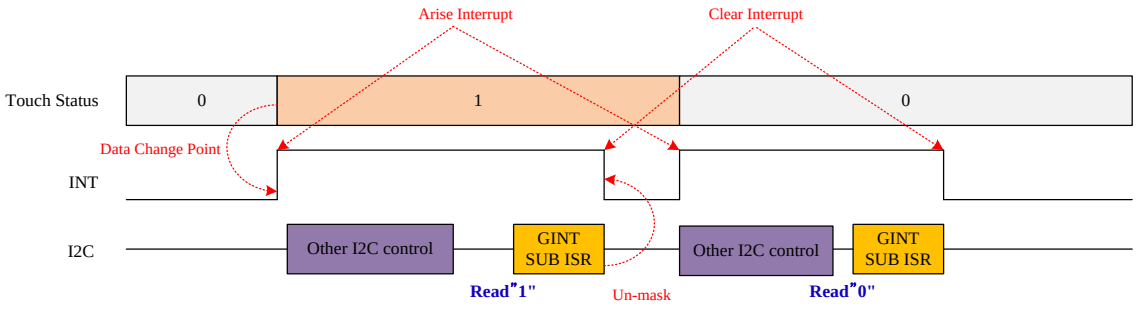


Figure 8-13: Example of GINT Signal

8.9. Power Management

The state diagram below in the Figure 8-14 shows the condition of ATA5009's power transition among three power states, Active mode, Idle mode and Sleep mode.

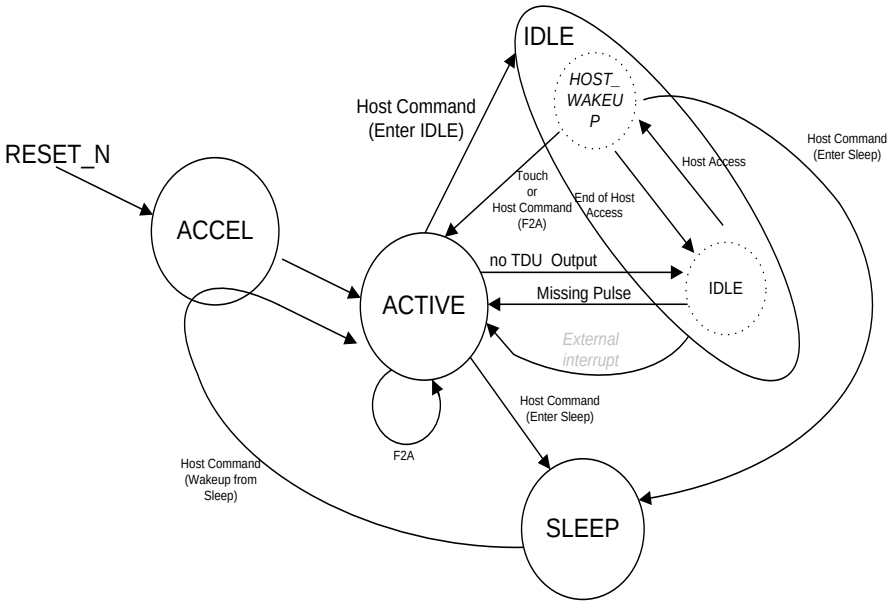


Figure 8-14. PMU (Power Management Unit) State Diagram

8.9.1. Condition of Active to Idle transition

a. When touch is not detected during Idle enter time.
b. After finishing Idle calibration

8.9.2. Condition of Idle to Active transition

a. When touch is detected in Idle mode
b. When Idle calibration is ready to start in Idle mode
c. When Force to Active command is received

8.9.3. Condition to enter Sleep mode

a. When Enter Sleep command is received no matter what current power state is.

8.9.4. Condition of Sleep to Active transition

a. When Wake-up command is received in Sleep mode

Please refer to Sleep and Wake up command in the section 9.2.3.

8.10. Reset

8.10.1. Hardware Reset (RESET_N pin)

a. Power On Reset (POR).
When host MCU has no more I/O pins to control RESET signal to the ATA5009, POR is another choice. In order to delay RESET_N transition about 0.1msec than VDDH transition, 10KΩ resistor and 1uF capacitor should be attached on RESET_N pin. It also eliminates noises to give unexpected false RESET signals to the ATA5009.

b. MCU-Reset
Whenever MCU wants to reset the ATA5009, MCU-Reset by I/O port is recommended. With 10K ohm resistor and 1uF capacitor on the RESET_N, MCU should drive active LOW more than 10ms to generate a valid Reset signal.

Please refer to the waveform of Reset signal in the section 6.3.

8.10.2. Software Reset

a. Warm Reset
Warm reset initializes all digital units except internal registers and Clock Generation Unit. It should be given to the ATA5009 to start chip operation after hardware reset. In addition, warm reset is required when internal register values are changed.

b. Cold Reset
Cold Reset initialize all the blocks in the ATA5009 including internal registers and CGU.

Please refer to Warm Reset and Cold Reset command in the section 9.2.3.

9. Registers

The addresses shown in the table are for I^2C and SPI interface.

9.1. Abbreviation

Abbreviated	Original
A2I	Active to Idle
ACAL	active calibration
APIS	Adjacent Pattern Interference Suppress
CAL	calibration
CLK	clock
COEF	coefficient
CONF	configuration
CTRL	Control
EPD	Extended Pulse Delay
HCAL	hold calibration
I2A	Idle to Active
ICAL	idle calibration
INIT	Initialize
INT	interrupt
INTV	interval
NUM	number
PA	Port A
PB	Port B
PC	Port C
PEND	pending
RES	resolution
TH	Threshold

9.2. Register List

9.2.1. Configuration Registers

Order	Address	R/W	Register Name	Description
1~21	0x00~0x14	RW	Beta_PA0 ~ Beta_PC6	Local criteria applied to each port to enable calibration
22	0x15	RW	EPD_rollbank (Reserved)	Do not change value, Test purpose only
23~34	0x16~0x21	RW	R_SEL_PA1_PA0 ~ R_SEL_PC6	To select I/O resistor to change sensitivity resolution
35	0x22	RW	Filter1_Configuration	Pre Filter Configuration
36	0x23	RW	Down_Sampling_Rate	Down Sampler's sampling rate control
37	0x24	RW	Filter2_Configuration	Post Filter Configuration
38	0x25	RW	INIT_Calibration_Time	To initialize calibration time
39	0x26	RW	AIC_Interval	To define calibration interval in Active mode
40	0x27	RW	AIC_Wait_Time	To suspend AIC during the period defined here
41	0x28	RW	Idle_Enter_Time	To define the time to enter Idle mode automatically
42	0x29	RW	Cluster _Threshold	To Activate it only for 2D application
43~63	0x2a~0x3e	RW	I2A_TH_PA0 ~ I2A_TH_PC6	Criteria to detect Touch in Idle mode
64~84	0x3f~0x53	RW	Touch_TH_PA0 ~ Touch_TH_PC6	Criteria to detect Touch in Active mode
85	0x54	RW	APIS	To define number of touch outputs
86~106	0x55~0x69	RW	Pin_Map_PA0 ~ Pin_Map_PC6	Pin Mapping for PA0 through PC6
107	0x6a	RW	Resolution_X_L	Touch Resolution of X axis Least Significant Byte
108	0x6b	RW	Resolution_X_H	Touch Resolution of X axis Most Significant Byte
109	0x6c	RW	Resolution_Y_L	Touch Resolution of Y axis Least Significant Byte
110	0x6d	RW	Resolution_Y_H	Touch Resolution of Y axis Most Significant Byte
111	0x6e	RW	X_Cluster_Gain (Reserved)	Do not change value, test purpose only
112	0x6f	RW	Y_Cluster _Gain (Reserved)	Do not change value, test purpose only
113	0x70	RW	Control1	Function control 1 (interrupt)
114	0x71	RW	Control2	Function control 2 (idle, 1D/2D, sensor control, auto_epd)
115	0x72	RW	Clock_Configuration	To change Sensor Clock speed
116	0x73	RW	GPO_PA	If any of PA6:0 are set to GPO, the corresponding bits in this register output through Port A
117	0x74	RW	GPO_PB	If any of PB6:0 are set to GPO, the corresponding bits in this register output through Port B
118	0x75	RW	GPO_PC	If any of PC6:0 are set to GPO, the corresponding bits in this register output through Port C
119	0x76	RW	INT_Mask	General Interrupt Mask register
120	0x77	RW	INT_Clear	General Interrupt Clear register
121	0x78	RW	ESD_Check	To check inter register collapse due to ESD MUST be set to 0xFF before start.
122	0x79	RW	Noise_Threshold	To suppress noise or proximity effect in TDU
123	0x7a	RW	NUM_Channel_X	To define number of channels in X position
124	0x7b	RW	NUM_Channel_Y	To define number of channels in Y position
125	0x7c	RW	Beta_Global	global criterion applied to all channels to enable calibration
126	0x7d	RW	Idle_Calibration_Interval	To define calibration interval in Idle mode
127	0x7e	RW	PDU_Latency (Reserved)	Do not change value, Test purpose only
128	0x7f	RW	Idle_Calibration_Duration	To define the time staying at Active mode for idle cal.
129	0x80	RW	EPD_Recover (Reserved)	Do not change value, Test purpose only
130	0x81	RW	LDO_control	LDO on/off enable control

9.2.2. Output Registers

Order	Address	R/W	Register Name	Description
131~172	0x90~0xb9	R	REF_PA0 ~ REF_PC6	Offset capacitances calibrated by Normalizer
173	0xba	R	TOUT_PA	APIS Touch output in PA
174	0xbb	R	TOUT_PB	APIS Touch output in PB
175	0xbc	R	TOUT_PC	APIS Touch output in PC
176	0xbd	R	Strength_X	Strength of X Position
177	0xbe	R	Strength_Y	Strength of Y Position
178	0xbf	R	Position_X_L	X Position Lease significant byte
179	0xc0	R	Position_X_H	X Position Most Significant byte
180	0xc1	R	Position_Y_L	Y Position Least Significant byte
181	0xc2	R	Position_Y_H	Y Position Most Significant byte
182~223	0xc3~0xec	R	pP_Value_PA0 ~ pP_Value_PC6	Purified P_Value by Normalizer and TDU These values become input of PFU.
224	0xed	R	INT_PEND	General Interrupt Pending

9.2.3. Command Registers

Following registers are asynchronous and you can write arbitrary data to a specific register to execute corresponding functions.

Order	Address	R/W	Register Name	Description
225	0xfa	W	EX_LDO	write ”00” when using external LDO application mode
226	0xfc	W	WAKEUP	write “00” then ATA5009 receive Wakeup command
227	0xfd	W	SLEEP	write “00” then ATA5009 receive Sleep command
228	0xfe	W	CRESET	write “00” then ATA5009 receive CRESET command
229	0xff	W	WRESET	write “00” then ATA5009 receive WRESET command

9.3. Configuration Register Description

9.3.1. Beta_PA0..Beta_PC6 (0x00 ~ 0x14)

Bits	Width	Default Value	Description
7:0	8	0x04	The second level of Beta for the channel PA0 through PC6. This criterion is tested when there is no channel whose raw P-value change is greater than "beta_global" which is the first level of Beta.

9.3.2. EPD_rollback (Reserved) (0x15)

Bits	Width	Default Value	Description
7:0	8	0x04	Do not change default value, test purpose only.

9.3.3. R_SEL_PA1_PA0 (0x16)

Bits	Width	Default Value	Description
7:4	4	0x02	I/O resistor selection bits for the sensitivity of Port A1. (See note 1)
3:0	4	0x02	I/O resistor selection bits for the sensitivity of Port A0. (See note 1)

(note 1)

0000	10K ohm	0100	30K ohm	1000	50K ohm	1100	70K ohm
0001	15K ohm	0101	35K ohm	1001	55K ohm	1101	80K ohm
0010	20K ohm	0110	40K ohm	1010	60K ohm	1110	90K ohm
0011	25K ohm	0111	45K ohm	1011	65K ohm	1111	100K ohm

R-sel setting is decided by an application. R-sel should make that pP-values are larger than 80 in touchpad applications, larger than 40 in scroll applications, and larger than 20 in button applications.

It is important to note that larger sensitivity setting is not always better than lower sensitivity. Especially 65K and 100K setting may cause large noises. So user should careful to use higher than 65K R-sel settings

9.3.4. R_SEL_PA3_PA2 (0x17)

Bits	Width	Default Value	Description
7:4	4	0x02	I/O resistor selection bits for the sensitivity of Port A3. (See note 1)
3:0	4	0x02	I/O resistor selection bits for the sensitivity of Port A2. (See note 1)

9.3.5. R_SEL_PA5_PA4 (0x18)

Bits	Width	Default Value	Description
7:4	4	0x02	I/O resistor selection bits for the sensitivity of Port A5. (See note 1)
3:0	4	0x02	I/O resistor selection bits for the sensitivity of Port A4. (See note 1)

9.3.6. R_SEL_PA6 (0x19)

Bits	Width	Default Value	Description
7:4	4	0x00	Reserved (Don't care)
3:0	4	0x02	I/O resistor selection bits for the sensitivity of Port A6. (See note 1)

9.3.7. R_SEL_PB1_PB0 (0x1A)
Refer to the description of R_SEL_PA1_PA0 register by substituting PB1 and PB0 for PA1 and PA0.

9.3.8. R_SEL_PB3_PB2 (0x1B)
Refer to the description of R_SEL_PA3_PA2 register by substituting PB3 and PB2 for PA3 and PA2.

9.3.9. R_SEL_PB5_PB4 (0x1C)
Refer to the description of R_SEL_PA5_PA4 register by substituting PB5 and PB4 for PA5 and PA4.

9.3.10. R_SEL_PB6 (0x1D)
Refer to the description of R_SEL_PA6 register by substituting PB6 for PA6.

9.3.11. R_SEL_PC1_PC0 (0x1E)
Refer to the description of R_SEL_PA1_PA0 register by substituting PC1 and PC0 for PA1 and PA0.

9.3.12. R_SEL_PC3_PC2 (0x1F)
Refer to the description of R_SEL_PA3_PA2 register by substituting PC3 and PC2 for PA3 and PA2.

9.3.13. R_SEL_PC5_PC4 (0x20)
Refer to the description of R_SEL_PA5_PA4 register by substituting PC5 and PC4 for PA5 and PA4.

9.3.14. R_SEL_PC6 (0x21)
Refer to the description of R_SEL_PA6 register by substituting PC6 for PA6.

9.3.15. Filter1_Configuration (0x22)

Bits	Width	Default Value	Description
7:6	2	0x0	Reserved (Don't care)
5	1	0x0	A control bit to bypass the pre-filter. 0: to use pre-filter 1: to bypass pre-filter
4:0	5	0x08	Pre-filter coefficient whose range is 1/32 through 31/32. Pre-filter coefficient = bit[4:0] / 32. Lower filter coefficient value make filter's cut-off frequency low. Then noise will be reduced and response will be delayed.

9.3.16. Down_Sampling_Rate (0x23)

Bits	Width	Default Value	Description
7:0	8	0x08	Controls Down sampler's output rate which eventually changes Position Data output rate. Data Rate$(\frac{\text{sample}}{\text{Sec}}) = \frac{\text{fs}}{\text{Down_Sampling_Rate}}$ if Down_Sampling_Rate < 2 , Data Rate $=\frac{\text{fs}}{2}$ If this register is changed, post filter coefficient also must be changed accordingly.

9.3.17. Filter2_Configuration (0x24)

Bits	Width	Default Value	Description
7:6	2	0x0	Reserved (Don't care)
5	1	0x0	A control bit to bypass the post filter. 0: use post filter 1: bypass post filter
4:0	5	0x08	Post-filter coefficient whose range is 1/32 through 31/32. Post-filter coefficient = bit[4:0] / 32. Lower filter coefficient value make filter's cut-off frequency low. Then noise will be reduced and response will be delayed.

9.3.18. INIT_Calibration_Time (0x25)

Bits	Width	Default Value	Description
7:0	8	0xFF	Initial calibration time control register. This is to make AIC delay after power-on. In other words, during power-on time, AIC must be disable. To prevent AIC during power-on duration, we have to set 0x25 register value long enough. $\text{time (ms)} = \frac{16 \times \text{init calibration time}}{\text{fs}}$

9.3.19. AIC Interval: Active_Calibration_Interval (0x26)

Bits	Width	Default Value	Description
7:0	8	0x80	This register determines AIC interval when the ATA5009 is in active mode. During calibration, P-values of all channels are scanned. ATA5009 checks whether raw P-value variation is smaller than beta value. If raw P-value variation is over beta value, then ignore current AIC activation and wait for the next AIC interval. And, after AIC, the reference value of the Normalizer is updated. The recommended interval is about 1 sec through 3 sec. By controlling 0x26 value, we can also make ATA5009 immune for water drop. It is important to notice that 0x26 is only one parameter that controls water performance. The conversion equation is: $\text{Active Calibration interval(s)} = \frac{128}{\text{fs}} \times \text{Active\_Calibration\_Interval}$

9.3.20. AIC Wait Time (0x27)

Bits	Width	Default Value	Description
7:0	8	0x20	Even after Touch Detection Unit determines that all sensor inputs have entered the touch-off status from the touch-on status in which at least any one channel had been touched, AIC is still not performed until the counter which loads the value of CAL_Wait_Time is cleared. The value recommended is between 30 msec to 50 msec. The conversion equation is: $\text{Calibration Wait Timel(ms)} = \frac{16000}{fs} \times \text{AIC\_Wait\_Time}$

9.3.21. Idle_Enter_Time (0x28)

Bits	Width	Default Value	Description
7:0	8	0x30	If time elapses in ACTIVE state without touch detection during the period defined at this register, the ATA5009 enters IDLE state by itself. The conversion equation is: $\text{Time to enter Idle mode (sec)} = \frac{2556}{fs} \times \text{Idle\_Enter\_Time}$

9.3.22. Cluster_Threshold (0x29)

Bits	Width	Default Value	Description
7:0	8	0x10	Define this register only for 2D application to eliminate week touches in the edge area. Get valid coordinates only when at least one of pP_Values in X cluster and at least one of pP_Values in Y cluster are greater or equal to this threshold value.

9.3.23. I2A_TH_PA0..I2A_TH_PA6 (0x2A..0x30)

Bits	Width	Default Value	Description
7:0	8	0x0A	This parameter decides the sensitivity of the PA Ports to wake up the ATA5009 in IDLE power saving state.

9.3.24. I2A_TH_PB0..I2A_TH_PB6 (0x31..0x37)

Bits	Width	Default Value	Description
7:0	8	0x0A	This parameter decides the sensitivity of the PB Ports to wake up the ATA5009 in IDLE power saving state.

9.3.25. I2A_TH_PC0..I2A_TH_PC6 (0x38..0x3E)

Bits	Width	Default Value	Description
7:0	8	0x0A	This parameter decides the sensitivity of the PC Ports to wake up the ATA5009 in IDLE power saving state.

9.3.26. Touch_TH_PA0 .. Touch_TH_PA6 (0x3F..0x45)

Bits	Width	Default Value	Description
7:0	8	0x10	Touch Detection Unit generates touch outputs when nP-Value of each PA port is greater than or equal to the corresponding register value.

9.3.27. Touch_TH_PB0 .. Touch_TH_PB6 (0x46..0x4C)

Bits	Width	Default Value	Description
7:0	8	0x10	Touch Detection Unit generates touch outputs when nP-Value of each PB port is greater than or equal to the corresponding register value.

9.3.28. Touch_TH_PC0 .. Touch_TH_PC6 (0x4D..0x53)

Bits	Width	Default Value	Description
7:0	8	0x10	Touch Detection Unit generates touch outputs when nP-Value of each PC port is greater than or equal to the corresponding register value.

9.3.29. APIS (0x54)

Bits	Width	Default Value	Description
7:2	6	0x00	Reserved (Don't care)
1:0	2	0x1	Select APIS mode. - [01] : enables APIS mode 1 (generates the strongest output) - [10] : enables APIS mode 2 (generates all outputs beyond their pre-defined threshold limits) Only one of two bits should be set to 1.

9.3.30. Pin_Map_PA0..Pin_Map_PA6 (0x55..0x5B)

Bits	Width	Default Value	Description
7	1	0x1	Configures a PA port to be either a sensor input or a digital output 0: The channel is configured as a digital output. 1: The channel is configured as a sensor input.
6	1	0x1	Assigns a sensor input to the position or button application. 0: The channel is used as an element of a button application. 1: The channel is used as an element of a 1D or 2D application. Only valid when bit7 is 1.
5	1	0x0	Declares a sensor input port to be used as an element of X or Y. 0: X, 1: Y Only valid when both bit6 & bit7 are 1.
4:0	5	0x00	Channel Mapping Number, 0~ 20. Only valid when both bit6 and bit7 are 1.

9.3.31. Pin_Map_PB0..Pin_Map_PB6 (0x5C..0x62)

Bits	Width	Default Value	Description
7	1	0x1	Configures a PB port to be either a sensor input or a digital output 0: The channel is configured as a digital output. 1: The channel is configured as a sensor input.
6	1	0x1	Assigns a sensor input to the position or button application. 0: The channel is used as an element of a button application. 1: The channel is used as an element of a 1D or 2D application. Only valid when bit7 is 1.
5	1	0x0	Declares a sensor input port to be used as an element of X or Y. 0: X, 1: Y Only valid when both bit6 & bit7 are 1.
4:0	5	0x00	Channel Mapping Number, 0~ 20. Only valid when both bit6 and bit7 are 1.

9.3.32. Pin_Map_PC0..Pin_Map_PC6 (0x63..0x69)

Bits	Width	Default Value	Description
7	1	0x1	Configures a PC port to be either a sensor input or a digital output 0: The channel is configured as a digital output. 1: The channel is configured as a sensor input.
6	1	0x1	Assigns a sensor input to the position or button application. 0: The channel is used as an element of a button application. 1: The channel is used as an element of a 1D or 2D application. Only valid when bit7 is 1.
5	1	0x0	Declares a sensor input port to be used as an element of X or Y. 0: X, 1: Y Only valid when both bit6 & bit7 are 1.
4:0	5	0x00	Channel Mapping Number, 0~ 20. Only valid when both bit6 and bit7 are 1.

9.3.33. Resolution_X_L (0x6A)

Bits	Width	Default Value	Description
7:0	8	0xFF	Least significant byte of the touch resolution of X axis for the touch screen

9.3.34. Resolution_X_H (0x6B)

Bits	Width	Default Value	Description
7:0	8	0x05	Most significant byte of the touch resolution of X axis for the touch screen

9.3.35. Resolution_Y_L (0x6C)

Bits	Width	Default Value	Description
7:0	8	0xFF	Least significant byte of the touch resolution of Y axis for the touch screen

9.3.36. Resolution_Y_H (0x6D)

Bits	Width	Default Value	Description
7:0	8	0x05	Most significant byte of the touch resolution of Y axis for the touch screen

9.3.37. X_Cluster_Gain (Reserved) (0x6E)

Bits	Width	Default Value	Description
7:0	8	0x10	Do not change value, test purpose only

9.3.38. Y_Cluster_Gain (Reserved) (0x6F)

Bits	Width	Default Value	Description
7:0	8	0x10	Do not change value, test purpose only

9.3.39. Control1 (0x70): Interrupt

Bits	Width	Default Value	Bit Name	Description
7:5	3	0x0		Reserved (Don't care)
4	1	0x0	INT_SEL	Interrupt selection bit 0: GINT, 1: TINT
3:2	2	0x0		Reserved (Don't care)
1	1	0x0	INT_POL	Interrupt polarity selection bit 0: rising edge, 1: falling edge
0	1	0x1	ACC_OP	Reserved for test mode only, should be 1 always.

9.3.40. Control2 (0x71): Function control 2 (idle, 1D/2D, sensor control, auto_epd)

Bits	Width	Default Value	Bit Name	Description
7	1	0x0	F2A	0: Disable 1: Enable, to make power state Active mode always.
6	1	0x0	F2I	Reserved for test mode only. Should be 0 always.
5	1	0x0		Reserved (Don't care)
4	1	0x1	Cluster_TH	0: Disable Cluster_Threshold for 1D application 1: Enable Cluster_Threshold for 2D application
3	1	0x0	DEBUG	Reserved for test mode only. Should be 0 always.
2	1	0x0	ST_SENSOR	0: Sensor stops its operation. 1: Sensor starts its operation.
1	1	0x1	GATE_SRC	0: It will leave to discharge the remaining voltage of each sensing channel naturally. 1: It will force to discharge the remaining voltage of each sensing channel.
0	1	0x1	AUTO_EPD	Reserved for test purpose only. Should be 1 always.

9.3.41. Clock Configuration (0x72)

Bits	Width	Default Value	Description
7	1	0x0	Reserved (Don't care)
6:4	3	0x0	Sensor clock(Snclk) & sampling frequency(fs) control 000 : Snclk →250KHz , fs → 11.9KHz 001 : Snclk →200KHz , fs → 9.52KHz 010 : Snclk →167KHz , fs → 7.95KHz 011 : Snclk →143KHz , fs → 6.81KHz 100 : Snclk →125KHz , fs → 5.95KHz 101 : Snclk →100KHz , fs → 4.76KHz 110 : Snclk →83.3KHz , fs → 3.97KHz 111 : Snclk →71.4KHz , fs → 3.4KHz
3	1	0x0	Sensor clock frequency control for Idle mode 0: Snclk → 997 Hz, fs → 47 Hz 1 : Snclk → 488 Hz, fs → 23 Hz
2	1	0x0	Sensing signal control 0 : OPP (OPeration in Parallel)) mode (OSC Release) 1 : OPS (Operation in Sequence)) mode (NCS Low : OSC Release, NCS High : OSC Hold) Here, NCS is equal to "ID_SS" in pin name.
1	1	0x0	Multi Oscillator Control 0 : Normal scan mode (Main INT Oscillator X 1) 1 : Turbo scan mode (Main INT Oscillator X 2)
0	1	0x0	Reserved (Don't care)

9.3.42. GPO_PA (0x73)

Bits	Width	Default Value	Description
7	1	0x0	Reserved (Don't care)
6:0	7	0x00	General purpose output data When bit7 of each Pin_Map_PA0:6 is set to 0, corresponding bit of this register is valid and outputs to Port A

9.3.43. GPO_PB (0x74)

Bits	Width	Default Value	Description
7	1	0x0	Reserved (Don't care)
6:0	7	0x00	General purpose output data When bit7 of each Pin_Map_PB0:6 is set to 0, corresponding bit of this register is valid and outputs to Port B.

9.3.44. GPO_PC (0x75)

Bits	Width	Default Value	Description
7	1	0x0	Reserved (Don't care)
6:0	7	0x00	General purpose output data When bit7 of each Pin_Map_PC0:6 is set to 0, corresponding bit of this register is valid and outputs to Port C.

9.3.45. INT_Mask (0x76)

Bits	Width	Default Value	Description
7:5	3	0x0	Reserved (Don't care)
4	1	0x1	I2A (Idle to Active) interrupt 0: Unmasked 1: Masked
3	1	0x1	A2I (Active to Idle) interrupt 0: Unmasked 1: Masked
2	1	0x0	Recovery request failure interrupt 0: Unmasked 1: Masked
1	1	0x0	PFU(Position Finding Unit) interrupt 0: Unmasked 1: Masked
0	1	0x0	APIS interrupt 0: Unmasked 1: Masked

Note that this register setting is valid only in GINT mode operation.

9.3.46. INT_Clear (0x77)

After serving an interrupt request, the corresponding clear bit should be set to 1 to inform the ATA5009. The bit is automatically cleared later by the ATA5009 so that the host does not need to write 0 to the bit.

Bits	Width	Default Value	Description
7:5	3	0x0	Reserved (Don't care)
4	1	0x0	I2A(Idle to Active) interrupt
3	1	0x0	A2I (Active to Idle) interrupt
2	1	0x0	Recovery request failure interrupt
1	1	0x0	PFU (Position Finding Unit) interrupt
0	1	0x0	APIS interrupt

9.3.47. ESD_Check (0x78)

Bits	Width	Default Value	Description
7:0	8	0x00	This register should always be written to "0xFF" during boot-up. If this register is not initialized with 0xFF, ATA5009 will generate GINT to inform that the host starts power-on sequence again.

9.3.48. Noise_Threshold (0x79)

Bits	Width	Default Value	Description
7:0	8	0x0a	The threshold value to eliminate proximity effect of the touch screen during calculation of position data.

9.3.49. NUM_Channel_X (0x7A)

Bits	Width	Default Value	Description
7:5	3	0x0	Reserved (Don't care)
4:0	5	0x00	Number of channels assigned for X axis in the touch panel

9.3.50. NUM_Channel_Y (0x7B)

Bits	Width	Default Value	Description
7:5	3	0x0	Reserved (Don't care)
4:0	5	0x00	Number of channels assigned for Y axis in the touch panel

9.3.51. Beta_Global (0x7C)

Bits	Width	Default Value	Description
7:5	3	0x0	Reserved (Don't care)
4:0	5	0x08	First level Beta criterion which is applied to all touch channels.

9.3.52. Idle_Calibration_Interval (0x7D)

Bits	Width	Default Value	Description
7:0	8	0x13	This time interval is used when ATA5009 is in IDLE state wakes up the ATA5009 to be ACTIVE state to enable a calibration. The recommended interval is about 3 sec through 5 sec. The more often idle calibration is performed, the more power the ATA5009 consumes in the idle mode. $\text{Idle Calibration interval(s)} = \frac{8}{\text{fs(idle)}} \times \text{ Idle\_Calibration\_Interval}$

9.3.53. PDU_Latency (Reserved) (0x7E)

Bits	Width	Default Value	Description
7:0	8	0x1	Reserved for test purpose only. Should be 0x01 always.

9.3.54. Idle_Calibration_Duration (0x7F)

Bits	Width	Default Value	Description
7:0	8	0x1f	When the ATA5009 currently in IDLE state wakes up to be ACTIVE state for calibration, it remains in ACTIVE during this duration and goes back to IDLE state again when there is no touch detected. The recommended duration is 10ms to 30ms. $\text{Idle Calibration duration(ms)} = \frac{8}{\text{fs}} \times \text{ Idle\_Calibration\_duration.}$

9.3.55. EPD_Recover (Reserved) (0x80)

Bits	Width	Default Value	Description
7:0	8	0x32	Reserved for test purpose only. Should be 0x32 always.

9.3.56. LDO_Control (0x81)

Bits	Width	Default Value	Description
7	1	0x0	LDO usage (Priority1) 0: Internal LDO application 1: External LDO application
6:3	4	0x0	Don't care (Not used)
2	1	0x0	Pull up when LDO off period (Priority3 – Active when bit[0] = 1'b) 0: 3.3V application 1: 5V application
1	1	0x0	LDO on/off applying mode (Priority3 – Active when bit[0] = 1'b1) 0: LDO off in Sleep mode 1: LDO off in Idle mode and Sleep mode
0	1	0x0	LDO on/ off control (Priority2 – Active when bit[7] = 1'b1) 0: LDO always on 1: LDO on/off according to operation mode

(Note 2)

- Data[7] of 0x81 is just for monitoring purpose.
- Internal LDO is basically 'ON' when Power on. If you want to use external LDO, you have to write command of 0xFA. And if you want to use internal LDO again, reset the device.
- There are priorities in this register bits: [7] > [0] > [2], [1]. If higher priority bits are not enabled, lower priority bits are same as 'don't care'.
- Here are few cases: (See application circuit in details)
  - ✓ External LDO (1.8V) use : 0xfa
  - ✓ Internal LDO use and no power saving mode : default value
  - ✓ Internal LDO use and power saving mode 0x03

9.4. Output Register Description

9.4.1. REF_PA0_L..REF_PA6_L (0x90, 0x92, 0x94, 0x96, 0x98, 0x9A, 0x9C)

Bits	Width	Default Value	Description
7:0	8	0x00	Least significant byte of the reference values for Port PA0:6. These values are updated automatically by calibration of the Normalizer.

9.4.2. REF_PA0_H..REF_PA6_H (0x91, 0x93, 0x95, 0x97, 0x99, 0x9B, 0x9D)

Bits	Width	Default Value	Description
7:2	6	0x00	Reserved (Don't care)
1:0	2	0x0	Most significant byte of the reference values for Port PA0:6. These values are updated automatically by calibration of the Normalizer.

9.4.3. REF_PB0_L..REF_PB6_L (0x9E, 0xA0, 0xA2, 0xA4, 0xA6, 0xA8, 0xAA)

Bits	Width	Default Value	Description
7:0	8	0x00	Least significant byte of the reference values for Port PB0:6. These values are updated automatically by calibration of the Normalizer.

9.4.4. REF_PB0_H..REF_PB6_H (0x9F, 0xA1, 0xA3, 0xA5, 0xA7, 0xA9, 0xAB)

Bits	Width	Default Value	Description
7:2	6	0x00	Reserved (not used)
1:0	2	0x0	Most significant byte of the reference values for Port PB0:6. These values are updated automatically by calibration of the Normalizer.

9.4.5. REF_PC0_L..REF_PC6_L (0xAC, 0xAE, 0xB0, 0xB2, 0xB4, 0xB6, 0xB8)

Bits	Width	Default Value	Description
7:0	8	0x00	Least significant byte of the reference values for Port PC0:6. These values are updated automatically by calibration of the Normalizer.

9.4.6. REF_PC0_H..REF_PC6_H (0xAD, 0xAF, 0xB1, 0xB3, 0xB5, 0xB7, 0xB9)

Bits	Width	Default Value	Description
7:2	6	0x00	Reserved (not used)
1:0	2	0x0	Most significant byte of the reference values for Port PC0:6. These values are updated automatically by calibration of the Normalizer.

9.4.7. TOUT_PA (0xBA)

Bits	Width	Default Value	Description
7	1	0x0	Reserved (not used)
6:0	7	0x00	Touch status of PA6~PA0 when they are used as sensor inputs.

9.4.8. TOUT_PB (0xBB)

Bits	Width	Default Value	Description
7	1	0x0	Reserved (not used)
6:0	7	0x00	Touch status of PB6~PB0 when they are used as sensor inputs.

9.4.9. TOUT_PC (0xBC)

Bits	Width	Default Value	Description
7	1	0x0	Reserved (not used)
6:0	7	0x00	Touch status of PC6~PC0 when they are used as sensor inputs.

9.4.10. Strength_X (0xBD)

Bits	Width	Default Value	Description
7:0	8	0x00	X cluster touch strength

9.4.11. Strength_Y (0xBE)

Bits	Width	Default Value	Description
7:0	8	0x00	Y cluster touch strength

9.4.12. Position_X_L (0xBF)

Bits	Width	Default Value	Description
7:0	8	0x00	Least significant byte of X position value.

9.4.13. Position_X_H (0xC0)

Bits	Width	Default Value	Description
7	1	0x0	0: invalid 1: valid
6:5	2	0x0	Reserved (not used)
4:0	5	0x00	Most significant bits of X position value.

9.4.14. Position_Y_L (0xC1)

Bits	Width	Default Value	Description
7:0	8	0x00	Least significant byte of Y position value.

9.4.15. Position_Y_H (0xC2)

Bits	Width	Default Value	Description
7	1	0x0	0: invalid 1: valid
6:5	2	0x0	Reserved (not used)
4:0	5	0x00	Most significant bits of Y position value.

9.4.16. pP_Value_PA0_L..pP_Value_PA6_L (0xC3, 0xC5, 0xC7, 0xC9, 0xCB, 0xCD, 0xCF)

Bits	Width	Default Value	Description
7:0	8	0x00	Least significant byte of pP-Values of Port PA0:6.

9.4.17. pP_Value_PA0_H..pP_Value_PA6_H (0xC4, 0xC6, 0xC8, 0xCA, 0xCC, 0xCE, 0xD0)

Bits	Width	Default Value	Description
7:2	6	0x00	Reserved (not used)
1:0	2	0x0	Most significant bits of pP-Values of Port PA0:6.

9.4.18. pP_Value_PB0_L..pP_Value_PB6_L (0xD1, 0xD3, 0xD5, 0xD7, 0xD9, 0xDB, 0xDD)

Bits	Width	Default Value	Description
7:0	8	0x00	Least significant byte of pP-Values of Port PB0:6.

9.4.19. pP_Value_PB0_H..pP_Value_PB6_H (0xD2, 0xD4, 0xD6, 0xD8, 0xDA, 0xDC, 0xDE)

Bits	Width	Default Value	Description
7:2	6	0x00	Reserved (not used)
1:0	2	0x0	Most significant bits of pP-Values of Port PB0:6.

9.4.20. pP_Value_PC0_L..pP_Value_PC6_L (0xDF, 0xE1, 0xE3, 0xE5, 0xE7, 0xE9, 0xEB)

Bits	Width	Default Value	Description
7:0	8	0x00	Least significant byte of pP-Values of Port PC0:6.

9.4.21. pP_Value_PC0_H..pP_Value_PC6_H (0xE0, 0xE2, 0xE4, 0xE6, 0xE8, 0xEA, 0xEC)

Bits	Width	Default Value	Description
7:2	6	0x00	Reserved (not used)
1:0	2	0x0	Most significant bits of pP-Values of Port PC0:6.

9.4.22. 9-4-22 INT_PEND (0xED)

When General interrupt (GINT) is used, the host can notice which interrupt is currently pending for the service through this register.

Bits	Width	Default Value	Description
7:5	3	0x0	Reserved (not used)
4	1	0x0	I2A (Idle to Active) interrupt
3	1	0x0	A2I (Active to Idle) interrupt
2	1	0x0	Recovery request failure interrupt
1	1	0x0	PFU (Position Finding Unit) interrupt
0	1	0x0	APIS interrupt

9.5. Command Register Description

Following registers are asynchronous and you can write arbitrary data to a specific register to execute corresponding functions.

Number	Address	R/W	Register name	Description
225	0xfa	W	EX_LDO	write ”00” when using external LDO application mode
226	0xfc	W	WAKEUP	write “00” then ATA5009 receive Wakeup command
227	0xfd	W	SLEEP	write “00” then ATA5009 receive Sleep command
228	0xfe	W	CRESET	write “00” then ATA5009 receive CRESET command
229	0xff	W	WRESET	write “00” then ATA5009 receive WRESET command

Appendix

A-1 Sample source code

A-1-1 Initialization

In order to utilize the ATA5009 (denoted as 'sensor' here after) in mobile phones, MP3 players or other I^2C Host devices, its registers must be initialized through I^2C interface as soon as power is supplied. For example, if the system needs to initialize the sensor to activate APIS mode, the register whose name and address are APIS and 0x54 (in hexadecimal) must be initialized with an appropriate value. The registers of the sensor are explained more detail in the section 9. After initialization, warm reset must follow to apply current setting immediately and you can confirm its operation result by observing LCD on the USB board or the Tuning Viewer Program. Do not apply cold reset which will initialize all registers of the sensor with default values.

The registers with similar characteristics have consecutive addresses to use I^2C burst mode which may reduce I^2C access time significantly. For example, BETA registers of PA0 through PC6 starts at the address 0x00 and ends at 0x14. The difference between single mode and burst mode is explained by programming BETA registers as below.

Program Example (in ANSI-C):

Let's assume that I^2C write function is as below.

```
I2C_Write(BYTE chipid, BYTE address, BYTE size, BYTE *data_write); // I2C Write function
```

Then we can program in two ways:

Case 1: Single mode writing to BETA PA0 (0x00) through BETA PC6 (0x14)

```
int chip_id = 0x64;
int i;
BYTE Beta =4;
for (i=0; i<21; i++)
{
      I2C_Write(chip_id, i, 1, Beta);          // i = address, 1 = data size, Beta = data
}
```

In the above program, a single I^2C write function is repeated 21 times. After executing this program, all BETA registers have the same value of 4. In this case, data size is defined as 1.

Case 2: Burst mode writing to BETA PA0 (0x00) through BETA PC6 (0x14)

```
Int chip_id = 0x64;
BYTE Beta[21];
For(i=0; i<21; i++)
{
      Beta[i] = 4;
}
I2C_Write(chip_id, 0,21,Beta);     // 0 = start address, 21 = data size, Beta = data
```

The above program is an example of burst write. All BETA registers have the same value of 4, but the data size is now defined as 21.

From the program examples above, if you can use burst mode operation on the consecutive addressed registers of the ATA5009 as many as possible, it will reduce the access time of I^2C. Figure A-1shows a flowchart of I^2C communication when the host MCU initializes the ATA5009 during the boot up.

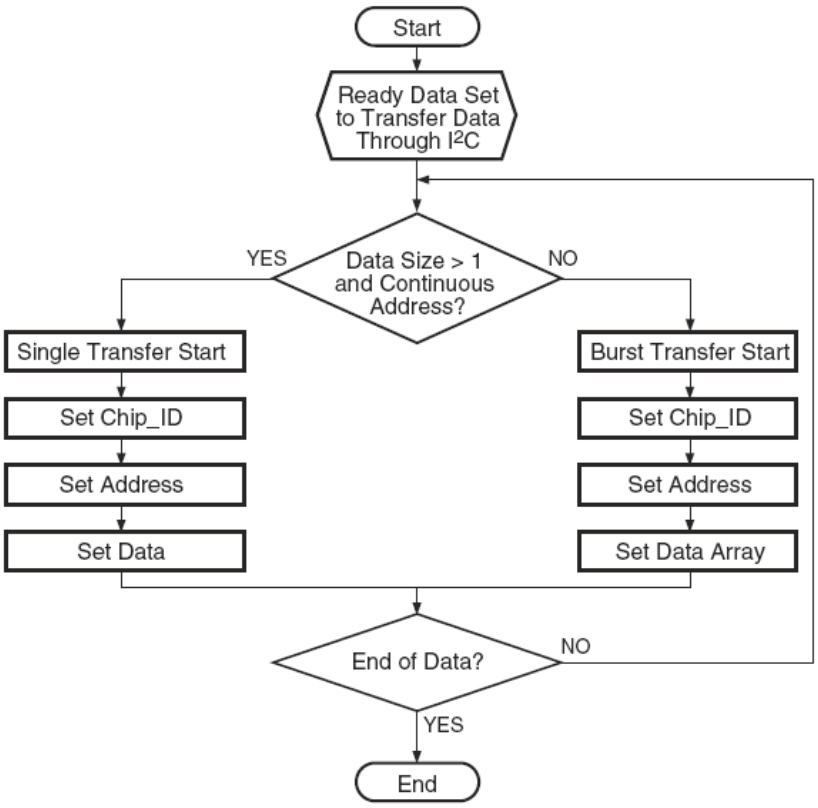


Fig A-1: Initialization of registers through I^2C by the Host MCU during the Boot-up

A-1-2 Sample Code for TINT using ATA5009 (I²C Control Mode)

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This is an example C-code to implement TINT such as 2-dimension ITO(9x6 diamond pattern) is connected to sensor inputs.
// Let's assume that
//          (1) TINT is used for position data
//
// Released by ATLab Korea, 20090202
// All rights are reserved.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#ifndef BYTE
        typedef unsigned char BYTE;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Global constants
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#define ATA5009_I2C_CHIPID                  0x64
#define ATA5009_REGS_NUM                    129

#define ATA5009_REG_WARM_RESET              0xFF
#define ATA5009_REG_ESD_CHECK               0x78
#define ATA5009_REG_POSITION                0xBF

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Init register set arrays...
// below initial value are some different to real application value. It is only example.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
const BYTE ATA5009_RegData[ATA5009_REGS_NUM]={
                                0x04,  // 0x00: beta_pa0            , default value = 0x04
                                0x04,  // 0x01: beta_pa1            , default value = 0x04
                                0x04,  // 0x02: beta_pa2            , default value = 0x04
                                0x04,  // 0x03: beta_pa3            , default value = 0x04
                                0x04,  // 0x04: beta_pa4            , default value = 0x04
                                0x04,  // 0x05: beta_pa5            , default value = 0x04
                                0x04,  // 0x06: beta_pa6            , default value = 0x04
                                0x04,  // 0x07: beta_pb0            , default value = 0x04
                                0x04,  // 0x08: beta_pb1            , default value = 0x04
                                0x04,  // 0x09: beta_pb2            , default value = 0x04
                                0x04,  // 0x0a: beta_pb3            , default value = 0x04
                                0x04,  // 0x0b: beta_pb4            , default value = 0x04
                                0x04,  // 0x0c: beta_pb5            , default value = 0x04
                                0x04,  // 0x0d: beta_pb6            , default value = 0x04
                                0x04,  // 0x0e: beta_pc0            , default value = 0x04
                                0x04,  // 0x0f: beta_pc1            , default value = 0x04
                                0x04,  // 0x10: beta_pc2            , default value = 0x04
                                0x04,  // 0x11: beta_pc3            , default value = 0x04
                                0x04,  // 0x12: beta_pc4            , default value = 0x04
                                0x04,  // 0x13: beta_pc5            , default value = 0x04
                                0x04,  // 0x14: beta_pc6            , default value = 0x04
                                0x04,  // 0x15: epd_rollback (R)    , default value = 0x04
                                0x55,  // 0x16: R_SEL_PA1_PA0       , default value = 0x22
                                0x55,  // 0x17: R_SEL_PA3_PA2       , default value = 0x22
                                0x55,  // 0x18: R_SEL_PA5_PA4       , default value = 0x22
                                0x05,  // 0x19: R_SEL_PA6           , default value = 0x02
                                0x55,  // 0x1a: R_SEL_PB1_PB0       , default value = 0x22
                                0x55,  // 0x1b: R_SEL_PB3_PB2       , default value = 0x22
                                0x55,  // 0x1c: R_SEL_PB5_PB4       , default value = 0x22
                                0x05,  // 0x1d: R_SEL_PB6           , default value = 0x02
                                0x55,  // 0x1e: R_SEL_PC1_PC0       , default value = 0x22
                                0x55,  // 0x1f: R_SEL_PC3_PC2       , default value = 0x22
```

```
0x55,  // 0x20: R_SEL_PC5_PC4          , default value = 0x22
0x05,  // 0x21: R_SEL_PC6              , default value = 0x02
0x02,  // 0x22: FIL1_CONF              , default value = 0x08
0x20,  // 0x23: Down_Sampling_Rate     , default value = 0x08
0x04,  // 0x24: FIL2_CONF              , default value = 0x08
0xff,  // 0x25: INIT_CAL               , default value = 0xFF
0x3c,  // 0x26: ACAL_INTV              , default value = 0x10
0xff,  // 0x27: AIC_Wait_Time          , default value = 0x0A
0xff,  // 0x28: Idle_Enter_TIME        , default value = 0x0A
0x14, // 0x29: Cluster_Threshold       , default value = 0x10
0x40,  // 0x2a: I2A_TH_pa0             , default value = 0x0A
0x40,  // 0x2b: I2A_TH_pa1             , default value = 0x0A
0x40,  // 0x2c: I2A_TH_pa2             , default value = 0x0A
0x40,  // 0x2d: I2A_TH_pa3             , default value = 0x0A
0x40,  // 0x2e: I2A_TH_pa4             , default value = 0x0A
0x40,  // 0x2f: I2A_TH_pa5             , default value = 0x0A
0x40,  // 0x30: I2A_TH_pa6             , default value = 0x0A
0x40,  // 0x31: I2A_TH_pb0             , default value = 0x0A
0x40,  // 0x32: I2A_TH_pb1             , default value = 0x0A
0x40,  // 0x33: I2A_TH_pb2             , default value = 0x0A
0x40,  // 0x34: I2A_TH_pb3             , default value = 0x0A
0x40,  // 0x35: I2A_TH_pb4             , default value = 0x0A
0x40,  // 0x36: I2A_TH_pb5             , default value = 0x0A
0x40,  // 0x37: I2A_TH_pb6             , default value = 0x0A
0x40,  // 0x38: I2A_TH_pc0             , default value = 0x0A
0x40,  // 0x39: I2A_TH_pc1             , default value = 0x0A
0x40,  // 0x3a: I2A_TH_pc2             , default value = 0x0A
0x40,  // 0x3b: I2A_TH_pc3             , default value = 0x0A
0x40,  // 0x3c: I2A_TH_pc4             , default value = 0x0A
0x40,  // 0x3d: I2A_TH_pc5             , default value = 0x0A
0x40,  // 0x3e: I2A_TH_pc6             , default value = 0x0A
0x1a,  // 0x3f: Touch_TH_pa0           , default value = 0x10
0x1a,  // 0x40: Touch_TH_pa1           , default value = 0x10
0x1a,  // 0x41: Touch_TH_pa2           , default value = 0x10
0x1a,  // 0x42: Touch_TH_pa3           , default value = 0x10
0x1a,  // 0x43: Touch_TH_pa4           , default value = 0x10
0x1a,  // 0x44: Touch_TH_pa5           , default value = 0x10
0x1a,  // 0x45: Touch_TH_pa6           , default value = 0x10
0x1a,  // 0x46: Touch_TH_pb0           , default value = 0x10
0x1a,  // 0x47: Touch_TH_pb1           , default value = 0x10
0x1a,  // 0x48: Touch_TH_pb2           , default value = 0x10
0x1a,  // 0x49: Touch_TH_pb3           , default value = 0x10
0x1a,  // 0x4a: Touch_TH_pb4           , default value = 0x10
0x1a,  // 0x4b: Touch_TH_pb5           , default value = 0x10
0x1a,  // 0x4c: Touch_TH_pb6           , default value = 0x10
0x1a,  // 0x4d: Touch_TH_pc0           , default value = 0x10
0x1a,  // 0x4e: Touch_TH_pc1           , default value = 0x10
0x1a,  // 0x4f: Touch_TH_pc2           , default value = 0x10
0x1a,  // 0x50: Touch_TH_pc3           , default value = 0x10
0x1a,  // 0x51: Touch_TH_pc4           , default value = 0x10
0x1a,  // 0x52: Touch_TH_pc5           , default value = 0x10
0x1a,  // 0x53: Touch_TH_pc6           , default value = 0x10
0x01,  // 0x54: APIS                   , default value = 0x01
0xe5,  // 0x55: Pin_MAP_pa0            , default value = 0xC0
0xe4,  // 0x56: PIN_MAP_pa1            , default value = 0xC0
0xe3,  // 0x57: PIN_MAP_pa2            , default value = 0xC0
0xe2,  // 0x58: PIN_MAP_pa3            , default value = 0xC0
0xe1,  // 0x59: PIN_MAP_pa4            , default value = 0xC0
0xe0,  // 0x5a: PIN_MAP_pa5            , default value = 0xC0
0xc0,  // 0x5b: PIN_MAP_pa6            , default value = 0xC0
0xc1,  // 0x5c: PIN_MAP_pb0            , default value = 0xC0
```

```
0xc2,  // 0x5d: PIN_MAP_pb1          , default value = 0xC0
0xc3,  // 0x5e: PIN_MAP_pb2          , default value = 0xC0
0xc4,  // 0x5f: PIN_MAP_pb3          , default value = 0xC0
0xc5,  // 0x60: PIN_MAP_pb4          , default value = 0xC0
0xc6,  // 0x61: PIN_MAP_pb5          , default value = 0xC0
0xc7,  // 0x62: PIN_MAP_pb6          , default value = 0xC0
0xc8,  // 0x63: PIN_MAP_pc0          , default value = 0xC0
0x00,  // 0x64: PIN_MAP_pc1          , default value = 0xC0
0x00,  // 0x65: PIN_MAP_pc2          , default value = 0xC0
0x00,  // 0x66: PIN_MAP_pc3          , default value = 0xC0
0x00,  // 0x67: PIN_MAP_pc4          , default value = 0xC0
0x00,  // 0x68: PIN_MAP_pc5          , default value = 0xC0
0x00,  // 0x69: Pin_Map_PC6          , default value = 0xC0
0xd0,  // 0x6a: Resolution_X_L       , default value = 0xFF
0x02,  // 0x6b: Resolution_X_H       , default value = 0x05
0xe0,  // 0x6c: Resolution_Y_L       , default value = 0xFF
0x01,  // 0x6d: Resolution_Y_H       , default value = 0x05
0x10,  // 0x6e: X_Cluster_Gain (R)   , default value = 0x10
0x10,  // 0x6f: Y_Cluster_Gain (R)   , default value = 0x10
0x13,  // 0x70: control1             , default value = 0x01
0x05,  // 0x71: control2             , default value = 0x13
0x00,  // 0x72: CLK_CONFIG           , default value = 0x00
0x00,  // 0x73: GPO_pa               , default value = 0x00
0x00,  // 0x74: GPO_pb               , default value = 0x00
0x00,  // 0x75: GPO_pc               , default value = 0x00
0x18,  // 0x76: INT_MASK             , default value = 0x18
0x00,  // 0x77: INT_CLEAR            , default value = 0x00
0xff,  // 0x78: ESD_Check            , default value = 0x00
0x07,  // 0x79: Noise_Threshold      , default value = 0x05
0x09,  // 0x7a: NUM_CH_X             , default value = 0x00
0x06,  // 0x7b: NUM_CH_Y             , default value = 0x00
0x07,  // 0x7c: BETA_GLOBAL          , default value = 0x06
0x46,  // 0x7d: ICAL_INTV            , default value = 0x10
0x01,  // 0x7e: PDU_Latency  (R)     , default value = 0x01
0x0e,  // 0x7f: ICAL_DUR             , default value = 0x1F
0x32,  // 0x80: epd_recover (R)      , default value = 0x32
0x00,  // 0x81: ldo_control          , default value = 0x00

};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Functions
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
extern char I2C_READ(BYTE chipid, BYTE addr, BYTE length, BYTE *data_read);
extern char I2C_WRITE(BYTE chipid, BYTE addr, BYTE length, BYTE *data_write);

void ATA5009_Init();
void ATA5009_TINT_ISR();
void ATA5009_ESD_Recovery();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: main
// Arguments:
//          IN        void
//          OUT       void
// Description: Main routine performs all configuration tasks, and waits for interrupt
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void main(void)
{
	// TODO: Add user's configuration tasks here, e.g. gpio, interrupt polarity

	// ATA5009 registers initialization & verification
	ATA5009_ESD_Recovery();

	while(1);	// waiting for interrupt
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_Init
// Arguments:
//		IN		void
//		OUT		void
//Description: This function sets up reset sequence of the touch sensor and writes initial register values.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_Init(void)
{
	BYTE i;
	BYTE ucData;

	/*
	   ATA5009 H/W Reset Pin Control
	   H/W reset pin is active low. Normal state needs high level with reset pin.
	   To make reset active, set '0' at least 1ms, recommendation is more than 10ms.
	   And get back to '1' to make normal state.

	   The code below is an example, GPIO_ATA5009_RESET is I/O port and connected to RESET_N pin.
	   You need to change with your own command.
	*/
	GPIO_ATA5009_RESET = 0;	// make reset low.
	Delay(2);			// suppose 1 is almost 1ms.
	GPIO_ATA5009_RESET = 1;	// release reset pin to high.

	// Set ESD_Check  Register
	ucData = 0xFF;
	I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_ESD_CHECK, 1, &ucData);

	// Set Register values
	for(i=0; i< ATA5009_REGS_NUM; i++)
	{
	    ucData = ATA5009_RegData[i];
	    I2C_WRITE(ATA5009_I2C_CHIPID, i, 1, &ucData);
	}

	// Give a WARM Reset to activate all the new settings.
	I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_WARM_RESET, 1, 0x00);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_TINT_ISR
// Arguments:
```

```
//            IN          void
//            OUT         void
// Description: Whether the customer uses polling mode or interrupt mode,
//                  this code can be called to handle touch sensor's position data.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_TINT_ISR(void)
{
      BYTE ucData[4];
      unsigned int uiPositionX, uiPositionY;
      unsigned char ucPosValidX, ucPosValidY;

      // check ESD_Check register to recognize reset state.
      I2C_READ(ATA5009_I2C_CHIPID, ATA5009_REG_ESD_CHECK,1,ucData);
      if(ucData[0]!=0xFF)
      {
          ATA5009_ESD_Recovery();
          return;
       }

      // read position data (4Byte)
      I2C_READ(ATA5009_I2C_CHIPID , ATA5009_REG_POSITION, 4, ucData);

      // parsing raw data to valid & positionX, positionY
      if((ucData[1]&0x80)!=0)
          ucPosValidX =1;
      else
          ucPosValidX =0;

      if((ucData[3]&0x80)!=0)
          ucPosValidY =1;
      else
          ucPosValidY =0;

      if(ucPosValidX )
         uiPositionX = ((unsigned int)(ucData[1]<<8)|(unsigned int)(ucData[0]))&0x1FFF;
      if(ucPosValidY )
         uiPositionY = ((unsigned int)(ucData[3]<<8)|(unsigned int)(ucData[2]))&0x1FFF;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_ESD_Recovery
//Arguments:
//            IN          void
//            OUT         void
// Description: When ESD(Electro Static Discharge) or any other electric shock may occur malfunction of touch sensor.
//                  In this case, touch sensor's registers could be cleared, and this code should be called from reset state checking
//                  routine. Then this code call initialize touch sensor again and try to recover chip.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_ESD_Recovery(void)
{
      BYTE ucData;

      // call init function
```

```
    ATA5009_Init();

    // verify register values
    for(i=0; i<ATA5009_REGS_NUM; i++)
    {
        I2C_READ(ATA5009_I2C_CHIPID , i, 1, &ucData);

        if(ucData != ATA5009_RegData[i])
        {
            ucData = ATA5009_RegData[i];
            I2C_WRITE(ATA5009_I2C_CHIPID, i, 1, &ucData);
        }
    }

    // Give a WARM Reset to activate all the new settings.
    I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_WARM_RESET, 1, 0x00);
}
```

A-1-3 Sample Code for GINT using ATA5009 (I²C Control Mode)

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This is an example C-code to implement GINT such as 2-dimension ITO(9x6 diamond pattern) is connected to sensor inputs.
// Let's assume that
//          (1)GINT is used for position data
//
// Released by ATLab Korea, 20090202
// All rights are reserved.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#ifndef BYTE
        typedef unsigned char BYTE;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Global constants
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#define ATA5009_I2C_CHIPID                  0x64
#define ATA5009_REGS_NUM                    129

#define ATA5009_REG_WARM_RESET              0xFF
#define ATA5009_REG_ESD_CHECK               0x78

#define ATA5009_REG_GINT_PENDING            0xED
#define ATA5009_REG_GINT_MASK               0x76
#define ATA5009_REG_GINT_CLEAR              0x77

#define ATA5009_REG_TOUT                    0xBA
#define ATA5009_REG_POSITION                0xBF

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Init register set arrays...
// Below initial value are some different to real application value. It is only example.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
const BYTE ATA5009_RegData[ATA5009_REGS_NUM]={
                                0x04,  // 0x00: beta_pa0              , default value = 0x04
                                0x04,  // 0x01: beta_pa1              , default value = 0x04
                                0x04,  // 0x02: beta_pa2              , default value = 0x04
                                0x04,  // 0x03: beta_pa3              , default value = 0x04
                                0x04,  // 0x04: beta_pa4              , default value = 0x04
                                0x04,  // 0x05: beta_pa5              , default value = 0x04
                                0x04,  // 0x06: beta_pa6              , default value = 0x04
                                0x04,  // 0x07: beta_pb0              , default value = 0x04
                                0x04,  // 0x08: beta_pb1              , default value = 0x04
                                0x04,  // 0x09: beta_pb2              , default value = 0x04
                                0x04,  // 0x0a: beta_pb3              , default value = 0x04
                                0x04,  // 0x0b: beta_pb4              , default value = 0x04
                                0x04,  // 0x0c: beta_pb5              , default value = 0x04
                                0x04,  // 0x0d: beta_pb6              , default value = 0x04
                                0x04,  // 0x0e: beta_pc0              , default value = 0x04
                                0x04,  // 0x0f: beta_pc1              , default value = 0x04
                                0x04,  // 0x10: beta_pc2              , default value = 0x04
                                0x04,  // 0x11: beta_pc3              , default value = 0x04
                                0x04,  // 0x12: beta_pc4              , default value = 0x04
                                0x04,  // 0x13: beta_pc5              , default value = 0x04
                                0x04,  // 0x14: beta_pc6              , default value = 0x04
                                0x04,  // 0x15: epd_rollback (R)      , default value = 0x04
                                0x55,  // 0x16: R_SEL_PA1_PA0         , default value = 0x22
                                0x55,  // 0x17: R_SEL_PA3_PA2         , default value = 0x22
                                0x55,  // 0x18: R_SEL_PA5_PA4         , default value = 0x22
                                0x05,  // 0x19: R_SEL_PA6             , default value = 0x02
```

```
0x55,  // 0x1a: R_SEL_PB1_PB0          , default value = 0x22
0x55,  // 0x1b: R_SEL_PB3_PB2          , default value = 0x22
0x55,  // 0x1c: R_SEL_PB5_PB4          , default value = 0x22
0x05,  // 0x1d: R_SEL_PB6              , default value = 0x02
0x55,  // 0x1e: R_SEL_PC1_PC0          , default value = 0x22
0x55,  // 0x1f: R_SEL_PC3_PC2          , default value = 0x22
0x55,  // 0x20: R_SEL_PC5_PC4          , default value = 0x22
0x05,  // 0x21: R_SEL_PC6              , default value = 0x02
0x02,  // 0x22: FIL1_CONF              , default value = 0x08
0x20,  // 0x23: Down_Sampling_Rate     , default value = 0x08
0x04,  // 0x24: FIL2_CONF              , default value = 0x08
0xff,  // 0x25: INIT_CAL               , default value = 0xFF
0x3c,  // 0x26: ACAL_INTV              , default value = 0x10
0xff,  // 0x27: AIC_Wait_Time          , default value = 0x0A
0xff,  // 0x28: Idle_Enter_TIME        , default value = 0x0A
0x14, // 0x29: Cluster_Threshold       , default value = 0x10
0x40,  // 0x2a: I2A_TH_pa0             , default value = 0x0A
0x40,  // 0x2b: I2A_TH_pa1             , default value = 0x0A
0x40,  // 0x2c: I2A_TH_pa2             , default value = 0x0A
0x40,  // 0x2d: I2A_TH_pa3             , default value = 0x0A
0x40,  // 0x2e: I2A_TH_pa4             , default value = 0x0A
0x40,  // 0x2f: I2A_TH_pa5             , default value = 0x0A
0x40,  // 0x30: I2A_TH_pa6             , default value = 0x0A
0x40,  // 0x31: I2A_TH_pb0             , default value = 0x0A
0x40,  // 0x32: I2A_TH_pb1             , default value = 0x0A
0x40,  // 0x33: I2A_TH_pb2             , default value = 0x0A
0x40,  // 0x34: I2A_TH_pb3             , default value = 0x0A
0x40,  // 0x35: I2A_TH_pb4             , default value = 0x0A
0x40,  // 0x36: I2A_TH_pb5             , default value = 0x0A
0x40,  // 0x37: I2A_TH_pb6             , default value = 0x0A
0x40,  // 0x38: I2A_TH_pc0             , default value = 0x0A
0x40,  // 0x39: I2A_TH_pc1             , default value = 0x0A
0x40,  // 0x3a: I2A_TH_pc2             , default value = 0x0A
0x40,  // 0x3b: I2A_TH_pc3             , default value = 0x0A
0x40,  // 0x3c: I2A_TH_pc4             , default value = 0x0A
0x40,  // 0x3d: I2A_TH_pc5             , default value = 0x0A
0x40,  // 0x3e: I2A_TH_pc6             , default value = 0x0A
0x1a,  // 0x3f: Touch_TH_pa0           , default value = 0x10
0x1a,  // 0x40: Touch_TH_pa1           , default value = 0x10
0x1a,  // 0x41: Touch_TH_pa2           , default value = 0x10
0x1a,  // 0x42: Touch_TH_pa3           , default value = 0x10
0x1a,  // 0x43: Touch_TH_pa4           , default value = 0x10
0x1a,  // 0x44: Touch_TH_pa5           , default value = 0x10
0x1a,  // 0x45: Touch_TH_pa6           , default value = 0x10
0x1a,  // 0x46: Touch_TH_pb0           , default value = 0x10
0x1a,  // 0x47: Touch_TH_pb1           , default value = 0x10
0x1a,  // 0x48: Touch_TH_pb2           , default value = 0x10
0x1a,  // 0x49: Touch_TH_pb3           , default value = 0x10
0x1a,  // 0x4a: Touch_TH_pb4           , default value = 0x10
0x1a,  // 0x4b: Touch_TH_pb5           , default value = 0x10
0x1a,  // 0x4c: Touch_TH_pb6           , default value = 0x10
0x1a,  // 0x4d: Touch_TH_pc0           , default value = 0x10
0x1a,  // 0x4e: Touch_TH_pc1           , default value = 0x10
0x1a,  // 0x4f: Touch_TH_pc2           , default value = 0x10
0x1a,  // 0x50: Touch_TH_pc3           , default value = 0x10
0x1a,  // 0x51: Touch_TH_pc4           , default value = 0x10
0x1a,  // 0x52: Touch_TH_pc5           , default value = 0x10
0x1a,  // 0x53: Touch_TH_pc6           , default value = 0x10
0x01,  // 0x54: APIS                   , default value = 0x01
0xe5,  // 0x55: Pin_MAP_pa0            , default value = 0xC0
0xe4,  // 0x56: PIN_MAP_pa1            , default value = 0xC0
```

```
    0xe3,  // 0x57: PIN_MAP_pa2          , default value = 0xC0
    0xe2,  // 0x58: PIN_MAP_pa3          , default value = 0xC0
    0xe1,  // 0x59: PIN_MAP_pa4          , default value = 0xC0
    0xe0,  // 0x5a: PIN_MAP_pa5          , default value = 0xC0
    0xc0,  // 0x5b: PIN_MAP_pa6          , default value = 0xC0
    0xc1,  // 0x5c: PIN_MAP_pb0          , default value = 0xC0
    0xc2,  // 0x5d: PIN_MAP_pb1          , default value = 0xC0
    0xc3,  // 0x5e: PIN_MAP_pb2          , default value = 0xC0
    0xc4,  // 0x5f: PIN_MAP_pb3          , default value = 0xC0
    0xc5,  // 0x60: PIN_MAP_pb4          , default value = 0xC0
    0xc6,  // 0x61: PIN_MAP_pb5          , default value = 0xC0
    0xc7,  // 0x62: PIN_MAP_pb6          , default value = 0xC0
    0xc8,  // 0x63: PIN_MAP_pc0          , default value = 0xC0
    0x00,  // 0x64: PIN_MAP_pc1          , default value = 0xC0
    0x00,  // 0x65: PIN_MAP_pc2          , default value = 0xC0
    0x00,  // 0x66: PIN_MAP_pc3          , default value = 0xC0
    0x00,  // 0x67: PIN_MAP_pc4          , default value = 0xC0
    0x00,  // 0x68: PIN_MAP_pc5          , default value = 0xC0
    0x00,  // 0x69: Pin_Map_PC6          , default value = 0xC0
    0xd0,  // 0x6a: Resolution_X_L       , default value = 0xFF
    0x02,  // 0x6b: Resolution_X_H       , default value = 0x05
    0xe0,  // 0x6c: Resolution_Y_L       , default value = 0xFF
    0x01,  // 0x6d: Resolution_Y_H       , default value = 0x05
    0x10,  // 0x6e: X_Cluster_Gain (R)   , default value = 0x10
    0x10,  // 0x6f: Y_Cluster_Gain (R)   , default value = 0x10
    0x13,  // 0x70: control1             , default value = 0x01
    0x05,  // 0x71: control2             , default value = 0x13
    0x00,  // 0x72: CLK_CONFIG           , default value = 0x00
    0x00,  // 0x73: GPO_pa               , default value = 0x00
    0x00,  // 0x74: GPO_pb               , default value = 0x00
    0x00,  // 0x75: GPO_pc               , default value = 0x00
    0x18,  // 0x76: INT_MASK             , default value = 0x18
    0x00,  // 0x77: INT_CLEAR            , default value = 0x00
    0xff,  // 0x78: ESD_Check            , default value = 0x00
    0x07,  // 0x79: Noise_Threshold      , default value = 0x05
    0x09,  // 0x7a: NUM_CH_X             , default value = 0x00
    0x06,  // 0x7b: NUM_CH_Y             , default value = 0x00
    0x07,  // 0x7c: BETA_GLOBAL          , default value = 0x06
    0x46,  // 0x7d: ICAL_INTV            , default value = 0x10
    0x01,  // 0x7e: PDU_Latency  (R)     , default value = 0x01
    0x0e,  // 0x7f: ICAL_DUR             , default value = 0x1F
    0x32,  // 0x80: epd_recover (R)      , default value = 0x32
    0x00,  // 0x81: ldo_control          , default value = 0x00

};


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Functions
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
extern char I2C_READ(BYTE chipid, BYTE addr, BYTE length, BYTE *data_read);
extern char I2C_WRITE(BYTE chipid, BYTE addr, BYTE length, BYTE *data_write);

void ATA5009_Init();
void ATA5009_GINT_ISR();
void ATA5009_ESD_Recovery();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: main
```

```
// Arguments:
//          IN          void
//          OUT         void
// Description: Main routine performs all configuration tasks, and waits for interrupt
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void main(void)
{
        // TODO: Add user's configuration tasks here, e.g. gpio, interrupt polarity

        // ATA5009 registers initialization & verification
        ATA5009_ESD_Recovery();

        while(1);       // waiting for interrupt
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_Init
// Arguments:
//          IN          void
//          OUT         void
//Description: This function sets up reset sequence of the touch sensor and writes initial register values.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_Init(void)
{
        BYTE i;
        BYTE ucData;

        /*
           ATA5009 H/W Reset Pin Control
           H/W reset pin is active low. Normal state needs high level with reset pin.
           To make reset active, set '0' at least 1ms, recommendation is more than 10ms.
           And get back to '1' to make normal state.

           The code below is an example, GPIO_ATA5009_RESET is I/O port and connected to RESET_N pin.
           You need to change with your own command.
        */
        GPIO_ATA5009_RESET = 0;    // make reset low.
        Delay(2);                  // suppose 1 is almost 1ms.
        GPIO_ATA5009_RESET = 1;    // release reset pin to high.

        // Set ESD_Check  Register
        ucData = 0xFF;
        I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_ESD_CHECK, 1, &ucData);

        // Set Register values
        for(i=0; i< ATA5009_REGS_NUM; i++)
        {
            ucData = ATA5009_RegData[i];
            I2C_WRITE(ATA5009_I2C_CHIPID, i, 1, &ucData);
        }

        // Give a WARM Reset to activate all the new settings.
        I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_WARM_RESET, 1, 0x00);
```

```
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_GINT_ISR
// Arguments:
//          IN      void
//          OUT     void
// Description:
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_GINT_ISR(void)
{
    BYTE ucData[4];
    BYTE ucPending;
    unsigned int uiPositionX, uiPositionY;
    unsigned char ucPosValidX, ucPosValidY;

    // check ESD_Check register to recognize reset state.
    I2C_READ(ATA5009_I2C_CHIPID, ATA5009_REG_ESD_CHECK,1,ucData);
    if(ucData[0]!=0xFF)
    {
        ATA5009_ESD_Recovery();
        return;
     }

    // Read INT_PENDING_REG register
    I2C_READ(ATA5009_I2C_CHIPID, ATA5009_REG_GINT_PENDING, 1, &ucPending);

    // Write INT_MASK register to mask all interrupts
    ucData[0] = 0x1F;
    I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_GINT_MASK, 1, ucData);

    // GINT handler
    if(ucPending &0x01)
    {
        // APIS interrupt: read touch data (3Bytes)
        I2C_READ(ATA5009_I2C_CHIPID, ATA5009_REG_TOUT, 3, ucData);
    }
    if(ucPending &0x02)
    {
        // PFU interrupt: read position data (4Bytes)
        I2C_READ(ATA5009_I2C_CHIPID, ATA5009_REG_POSITION, 4, ucData);

        //parsing raw data to valid & positionX, position
        if((ucData[1]&0x80)!=0)
            ucPosValidX =1;
        else
            ucPosValidX =0;

        if((ucData[3]&0x80)!=0)
            ucPosValidY =1;
        else
            ucPosValidY =0;
```

```
        if(ucPosValidX!=0 && ucPosValidY!=0)
        {
           uiPositionX = ((unsigned int)(ucData[1]<<8)|(unsigned int)(ucData[0]))&0x1FFF;
           uiPositionY = ((unsigned int)(ucData[3]<<8)|(unsigned int)(ucData[2]))&0x1FFF;
        }
    }

    // Write INT_CLEAR register to clear all interrupts
    ucData[0] = 0x1F;
    I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_GINT_CLEAR, 1, ucData);

    // TODO: Enable interrupt by Host MCU

    // Write INT_MASK register to unmask all interrupst
    ucData[0] = 0x00;
    I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_GINT_MASK, 1, ucData);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_ESD_Recovery
//Arguments:
//          IN          void
//          OUT         void
// Description: When ESD(Electro Static Discharge) or any other electric shock may occur malfunction of touch sensor.
//              In this case, touch sensor's registers could be cleared, and this code should be called from reset state checking
//              routine. Then this code call initialize touch sensor again and try to recover chip.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_ESD_Recovery(void)
{
    BYTE ucData;

    // call init function
    ATA5009_Init();

    // verify register values
    for(i=0; i<ATA5009_REGS_NUM; i++)
    {
        I2C_READ(ATA5009_I2C_CHIPID , i, 1, &ucData);

        if(ucData != ATA5009_RegData[i])
        {
            ucData = ATA5009_RegData[i];
            I2C_WRITE(ATA5009_I2C_CHIPID, i, 1, &ucData);
        }
    }

    // Give a WARM Reset to activate all the new settings.
    I2C_WRITE(ATA5009_I2C_CHIPID, ATA5009_REG_WARM_RESET, 1, 0x00);
}
```

A-1-4 Sample Code for TINT using ATA5009 (SPI Control Mode)

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This is an example C-code to implement TINT such as 2-dimension ITO(9x6 diamond pattern) is connected to sensor inputs.
// Let's assume that
//          (1)TINT is used for position data
//
// Released by ATLab Korea, 20090202
// All rights are reserved.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#ifndef BYTE
        typedef unsigned char BYTE;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Global constants
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#define ATA5009_REGS_NUM                    129

#define ATA5009_REG_WARM_RESET              0xFF
#define ATA5009_REG_ESD_CHECK               0x78
#define ATA5009_REG_POSITION                0xBF

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Init register set arrays...
// below initial value are some different to real application value. It is only example.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
const BYTE ATA5009_RegData[ATA5009_REGS_NUM]={
                                …
                                /* ATA5009_RegData sample values are same as the arrays described in A.1.2 */
                                …
};


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Functions
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
char SPI_READ(BYTE chipid, BYTE addr, BYTE length, BYTE *data_read);
char SPI_WRITE(BYTE chipid, BYTE addr, BYTE length, BYTE *data_write);

void ATA5009_Init();
void ATA5009_TINT_ISR();
void ATA5009_ESD_Recovery();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: main
// Arguments:
//          IN          void
//          OUT         void
// Description: Main routine performs all configuration tasks, and waits for interrupt
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void main(void)
{
        // TODO: Add user's configuration tasks here, e.g. gpio, interrupt polarity

        // ATA5009 registers initialization & verification
        ATA5009_ESD_Recovery();
```

```
    while(1);      // waiting for interrupt
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_Init
// Arguments:
//          IN        void
//          OUT       void
//Description: This function sets up reset sequence of the touch sensor and writes initial register values.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_Init(void)
{
    BYTE i;
    BYTE ucData;

    /*
       ATA5009 H/W Reset Pin Control
       H/W reset pin is active low. Normal state needs high level with reset pin.
       To make reset active, set '0' at least 1ms, recommendation is more than 10ms.
       And get back to '1' to make normal state.

       The code below is an example, GPIO_ATA5009_RESET is I/O port and connected to RESET_N pin.
       You need to change with your own command.
    */
    GPIO_ATA5009_RESET = 0;   // make reset low.
    Delay(2);                 // suppose 1 is almost 1ms.
    GPIO_ATA5009_RESET = 1;   // release reset pin to high.

    // Set ESD_Check  Register
    ucData = 0xFF;
    SPI_WRITE(0, ATA5009_REG_ESD_CHECK, 1, &ucData);

    // Set Register values
    for(i=0; i< ATA5009_REGS_NUM; i++)
    {
        ucData = ATA5009_RegData[i];
        SPI_WRITE(0, i, 1, &ucData);
    }

    // Give a WARM Reset to activate all the new settings.
    SPI_WRITE(0, ATA5009_REG_WARM_RESET, 1, 0x00);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_TINT_ISR
// Arguments:
//          IN        void
//          OUT       void
// Description: Whether the customer uses polling mode or interrupt mode,
//                this code can be called to handle touch sensor's position data.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_TINT_ISR(void)
```

```
{
    BYTE ucData[4];
    unsigned int uiPositionX, uiPositionY;
    unsigned char ucPosValidX, ucPosValidY;

    // check ESD_Check register to recognize reset state.
    SPI_READ(0, ATA5009_REG_ESD_CHECK,1,ucData);
    if(ucData[0]!=0xFF)
    {
        ATA5009_ESD_Recovery();
        return;
     }

    // read position data (4Byte)
    SPI_READ(0 , ATA5009_REG_POSITION, 4, ucData);

    // parsing raw data to valid & positionX, positionY
    if((ucData[1]&0x80)!=0)
        ucPosValidX =1;
    else
        ucPosValidX =0;

    if((ucData[3]&0x80)!=0)
        ucPosValidY =1;
    else
        ucPosValidY =0;

    if(ucPosValidX )
       uiPositionX = ((unsigned int)(ucData[1]<<8)|(unsigned int)(ucData[0]))&0x1FFF;
    if(ucPosValidY )
       uiPositionY = ((unsigned int)(ucData[3]<<8)|(unsigned int)(ucData[2]))&0x1FFF;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_ESD_Recovery
//Arguments:
//          IN      void
//          OUT     void
// Description: When ESD(Electro Static Discharge) or any other electric shock may occur malfunction of touch sensor.
//              In this case, touch sensor's registers could be cleared, and this code should be called from reset state checking
//              routine. Then this code call initialize touch sensor again and try to recover chip.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_ESD_Recovery(void)
{
    BYTE ucData;

    // call init function
    ATA5009_Init();

    // verify register values
    for(i=0; i<ATA5009_REGS_NUM; i++)
    {
        SPI_READ(0 , i, 1, &ucData);
```

```
            if(ucData != ATA5009_RegData[i])
            {
                ucData = ATA5009_RegData[i];
                SPI_WRITE(0, i, 1, &ucData);
            }
        }

        // Give a WARM Reset to activate all the new settings.
        SPI_WRITE(0, ATA5009_REG_WARM_RESET, 1, 0x00);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: SPI_Read
// Arguments:
//          IN              chipid          :set to 0
//                          addr            :register address
//                          length          :read size
//          OUT             *data_read      :data from slave
// Description: SPI Reference code using Silabs MCU(C8051F320).
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
char SPI_Read(BYTE chipid, BYTE addr, BYTE length, BYTE *data_read)
{
        unsigned char i,j;
        unsigned short command;
        unsigned char dontcare;
        unsigned char ad;
        unsigned char sADDR[2];
        unsigned char sDATA[2];

        ad =1;
        command = (unsigned short)((0x7<<13)|((chipid&0x01)<<12)|((addr&0xff)<<4)|0xf);
        sADDR[0] = (unsigned char)(command>>8);
        sADDR[1] = (unsigned char)(command&0x00ff);

        NSSMD0 = 0;         // enable

        for(i=0; i<2; i++)
        {
           SPIF = 0;
           SPI0DAT       = sADDR[i];
           while ( !SPIF );                 // wait for data from transmit buffer to shift register
           dontcare = SPI0DAT;              // don't care
        }

        for(i=0; i<length; i ++)
        {
           if(i == length-1)
           {
              ad = 0;
              command = 0;
           }
           else
```

```
            command = (unsigned short)((0x3<<14)|((ad&0x01)<<13)|((id&0x01)<<12)|(((addr+1+i)&0xff)<<4)|0xf);

            sADDR[0] = (unsigned char)(command>>8);
            sADDR[1] = (unsigned char)(command&0x00ff);

            for(j=0; j<2; j++)
            {
               SPIF = 0;
               SPI0DAT = sADDR[j];
               while ( !SPIF );              // wait for data from transmit buffer to shift register
               sDATA[j] = SPI0DAT;           //don't care
            }
            data_read[i] = (unsigned char)((sDATA[0]<<4)|(sDATA[1]>>4));
        }
        NSSMD0   = 1;       // disable
        return 1;
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: SPI_Write
// Arguments:
//          IN              chipid          :set to 0
//                          addr            :register address
//                          length          :write size
//          OUT             *data_write     :data from master
// Description: SPI Reference code using Silabs MCU(C8051F320).
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
char SPI_Write(BYTE chipid, BYTE addr, BYTE length, BYTE *data_write)
{
        unsigned char ad;
        unsigned char i,j;
        unsigned short command;
        unsigned char sADDR[2];
        unsigned char sDATA[2];

        NSSMD0 = 0;             // enable
        for(i=0; i<length; i ++)
        {
            ad = 1;
            command = (unsigned short)((0x2<<14)|((ad&0x01)<<13)|((chipid&0x01)<<12)|(((addr+i)&0xff)<<4)|0xf);
            sADDR[0] = (unsigned char)(command>>8);
            sADDR[1] = (unsigned char)(command&0x00ff);

            for(j=0; j<2; j++)
            {
               SPIF = 0;
               SPI0DAT = sADDR[j];
               while ( !SPIF );         // wait for data from transmit buffer to shift register
            }

            ad = 0;
            command = (unsigned short)((0x2<<14)|((ad&0x01)<<13)|((id&0x01)<<12)|((data_write[i]&0xff)<<4)|0xf);
            sDATA[0] = (unsigned char)(command>>8);
```

```
        sDATA[1] = (unsigned char)(command&0x00ff);

        for(j=0; j<2; j++)
        {
          SPIF = 0;
          SPI0DAT       = sDATA[j];
          while ( !SPIF );          // wait for data from transmit buffer to shift register
        }
        SPIF = 0;
      }
      NSSMD0   = 1;      // disable
      return 1;
}
```

A-1-5 Sample Code for GINT using ATA5009 (SPI Control Mode)

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This is an example C-code to implement GINT such as 2-dimension ITO(9x6 diamond pattern) is connected to sensor inputs.
// Let's assume that
//          (1)GINT is used for position data
//
// Released by ATLab Korea, 20090202
// All rights are reserved.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#ifndef BYTE
        typedef unsigned char BYTE;
#endif

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Global constants
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#define ATA5009_REGS_NUM                    129

#define ATA5009_REG_WARM_RESET              0xFF
#define ATA5009_REG_ESD_CHECK               0x78

#define ATA5009_REG_GINT_PENDING            0xED
#define ATA5009_REG_GINT_MASK               0x76
#define ATA5009_REG_GINT_CLEAR              0x77

#define ATA5009_REG_TOUT                    0xBA
#define ATA5009_REG_POSITION                0xBF

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Init register set arrays...
// below initial value are some different to real application value. It is only example.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
const BYTE ATA5009_RegData[ATA5009_REGS_NUM]={
                                …
                                /* ATA5009_RegData sample values are same as the arrays described in A.1.2 */
                                …
};


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Functions
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
char SPI_READ(BYTE chipid, BYTE addr, BYTE length, BYTE *data_read);
char SPI_WRITE(BYTE chipid, BYTE addr, BYTE length, BYTE *data_write);

void ATA5009_Init();
void ATA5009_GINT_ISR();
void ATA5009_ESD_Recovery();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: main
// Arguments:
//          IN      void
//          OUT     void
// Description: Main routine performs all configuration tasks, and waits for interrupt
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void main(void)
```

```
{
    // TODO: Add user's configuration tasks here, e.g. gpio, interrupt polarity

    // ATA5009 registers initialization & verification
    ATA5009_ESD_Recovery();

    while(1);    // waiting for interrupt
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_Init
// Arguments:
//          IN      void
//          OUT     void
//Description: This function sets up reset sequence of the touch sensor and writes initial register values.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_Init(void)
{
    BYTE i;
    BYTE ucData;

    /*
       ATA5009 H/W Reset Pin Control
       H/W reset pin is active low. Normal state needs high level with reset pin.
       To make reset active, set '0' at least 1ms, recommendation is more than 10ms.
       And get back to '1' to make normal state.

       The code below is an example, GPIO_ATA5009_RESET is I/O port and connected to RESET_N pin.
       You need to change with your own command.
    */
    GPIO_ATA5009_RESET = 0;   // make reset low.
    Delay(2);                 // suppose 1 is almost 1ms.
    GPIO_ATA5009_RESET = 1;   // release reset pin to high.

    // Set ESD_Check  Register
    ucData = 0xFF;
    SPI_WRITE(0, ATA5009_REG_ESD_CHECK, 1, &ucData);

    // Set Register values
    for(i=0; i< ATA5009_REGS_NUM; i++)
    {
        ucData = ATA5009_RegData[i];
        SPI_WRITE(0, i, 1, &ucData);
    }

    // Give a WARM Reset to activate all the new settings.
    SPI_WRITE(0, ATA5009_REG_WARM_RESET, 1, 0x00);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_GINT_ISR
// Arguments:
//          IN      void
```

```
//            OUT       void
// Description: Whether the customer uses polling mode or interrupt mode,
//                   this code can be called to handle touch sensor's position data.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_GINT_ISR(void)
{
      BYTE ucData[4];
      BYTE ucPending;
      unsigned int uiPositionX, uiPositionY;
      unsigned char ucPosValidX, ucPosValidY;

      // check ESD_Check register to recognize reset state.
      SPI_READ(0, ATA5009_REG_ESD_CHECK,1,ucData);
      if(ucData[0]!=0xFF)
      {
            ATA5009_ESD_Recovery();
            return;
       }

      // Read INT_PENDING_REG register
      SPI_READ(0, ATA5009_REG_GINT_PENDING, 1, &ucPending);

      // Write INT_MASK register to mask all interrupts
      ucData[0] = 0x1F;
      SPI_WRITE(0, ATA5009_REG_GINT_MASK, 1, ucData);

      // GINT handler
      if(ucPending &0x01)
      {
          // APIS interrupt: read touch data (3Bytes)
          SPI_READ(0, ATA5009_REG_TOUT, 3, ucData);
      }
      if(ucPending &0x02)
      {
          // PFU interrupt: read position data (4Bytes)
          SPI_READ(0, ATA5009_REG_POSITION, 4, ucData);

          //parsing raw data to valid & positionX, position
          if((ucData[1]&0x80)!=0)
              ucPosValidX =1;
          else
              ucPosValidX =0;

          if((ucData[3]&0x80)!=0)
              ucPosValidY =1;
          else
              ucPosValidY =0;

          if(ucPosValidX!=0 && ucPosValidY!=0)
          {
             uiPositionX = ((unsigned int)(ucData[1]<<8)|(unsigned int)(ucData[0]))&0x1FFF;
             uiPositionY = ((unsigned int)(ucData[3]<<8)|(unsigned int)(ucData[2]))&0x1FFF;
          }
```

```
    }

    // Write INT_CLEAR register to clear all interrupts
    ucData[0] = 0x1F;
    SPI_WRITE(0, ATA5009_REG_GINT_CLEAR, 1, ucData);

    // TODO: Enable interrupt by Host MCU

    // Write INT_MASK register to unmask all interrupst
    ucData[0] = 0x00;
    SPI_WRITE(0, ATA5009_REG_GINT_MASK, 1, ucData);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: ATA5009_ESD_Recovery
//Arguments:
//          IN          void
//          OUT         void
// Description: When ESD(Electro Static Discharge) or any other electric shock may occur malfunction of touch sensor.
//              In this case, touch sensor's registers could be cleared, and this code should be called from reset state checking
//              routine. Then this code call initialize touch sensor again and try to recover chip.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ATA5009_ESD_Recovery(void)
{
    BYTE ucData;

    // call init function
    ATA5009_Init();

    // verify register values
    for(i=0; i<ATA5009_REGS_NUM; i++)
    {
        SPI_READ(0 , i, 1, &ucData);

        if(ucData != ATA5009_RegData[i])
        {
            ucData = ATA5009_RegData[i];
            SPI_WRITE(0, i, 1, &ucData);
        }
    }

    // Give a WARM Reset to activate all the new settings.
    SPI_WRITE(0, ATA5009_REG_WARM_RESET, 1, 0x00);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: SPI_Read
// Arguments:
//          IN          chipid          :set to 0
//                      addr            :register address
//                      length          :read size
//          OUT         *data_read      :data from slave
// Description: SPI Reference code using Silabs MCU(C8051F320).
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
char SPI_Read(BYTE chipid, BYTE addr, BYTE length, BYTE *data_read)
{
    unsigned char i,j;
    unsigned short command;
    unsigned char dontcare;
    unsigned char ad;
    unsigned char sADDR[2];
    unsigned char sDATA[2];

    ad =1;
    command = (unsigned short)((0x7<<13)|((chipid&0x01)<<12)|((addr&0xff)<<4)|0xf);
    sADDR[0] = (unsigned char)(command>>8);
    sADDR[1] = (unsigned char)(command&0x00ff);

    NSSMD0 = 0;       // enable

    for(i=0; i<2; i++)
    {
       SPIF = 0;
       SPI0DAT      = sADDR[i];
       while ( !SPIF );                 // wait for data from transmit buffer to shift register
       dontcare = SPI0DAT;              // don't care
    }

    for(i=0; i<length; i ++)
    {
       if(i == length-1)
       {
          ad = 0;
          command = 0;
       }
       else
          command = (unsigned short)((0x3<<14)|((ad&0x01)<<13)|((id&0x01)<<12)|(((addr+1+i)&0xff)<<4)|0xf);

       sADDR[0] = (unsigned char)(command>>8);
       sADDR[1] = (unsigned char)(command&0x00ff);

       for(j=0; j<2; j++)
       {
          SPIF = 0;
          SPI0DAT = sADDR[j];
          while ( !SPIF );              // wait for data from transmit buffer to shift register
          sDATA[j] = SPI0DAT;           //don't care
       }
       data_read[i] = (unsigned char)((sDATA[0]<<4)|(sDATA[1]>>4));
    }
    NSSMD0  = 1;      // disable
    return 1;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function name: SPI_Write
```

```
// Arguments:
//          IN                  chipid              :set to 0
//                              addr                :register address
//                              length              :write size
//          OUT                 *data_write         :data from master
// Description: SPI Reference code using Silabs MCU(C8051F320).
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
char SPI_Write(BYTE chipid, BYTE addr, BYTE length, BYTE *data_write)
{
    unsigned char ad;
    unsigned char i,j;
    unsigned short command;
    unsigned char sADDR[2];
    unsigned char sDATA[2];

    NSSMD0 = 0;             // enable
    for(i=0; i<length; i ++)
    {
       ad = 1;
       command = (unsigned short)((0x2<<14)|((ad&0x01)<<13)|((chipid&0x01)<<12)|(((addr+i)&0xff)<<4)|0xf);
       sADDR[0] = (unsigned char)(command>>8);
       sADDR[1] = (unsigned char)(command&0x00ff);

       for(j=0; j<2; j++)
       {
          SPIF = 0;
          SPI0DAT = sADDR[j];
          while ( !SPIF );          // wait for data from transmit buffer to shift register
       }

       ad = 0;
       command = (unsigned short)((0x2<<14)|((ad&0x01)<<13)|((id&0x01)<<12)|((data_write[i]&0xff)<<4)|0xf);
       sDATA[0] = (unsigned char)(command>>8);
       sDATA[1] = (unsigned char)(command&0x00ff);

       for(j=0; j<2; j++)
       {
          SPIF = 0;
          SPI0DAT       = sDATA[j];
          while ( !SPIF );          // wait for data from transmit buffer to shift register
       }
       SPIF = 0;
    }
    NSSMD0   = 1;      // disable
    return 1;
}
```

A-1-6 Register setting example: touchpad plus 5-touch button

In this example, 8-ch for X-axis, 7-ch for Y-axis, 5-ch for buttons, and 1-ch for no-use are assigned. Demo video is available at https://www.youtube.com/watch?v=eMJWPEZjhSk.

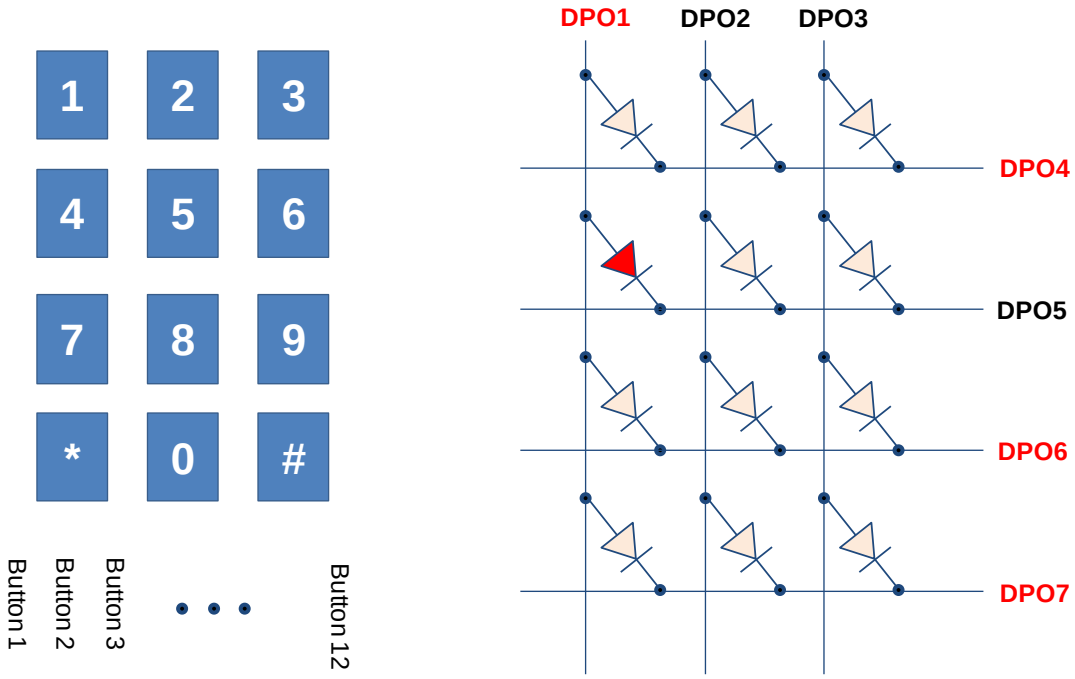
```
BYTE ATA5008_RegData[129]={
  0x04,  // 0x00: beta_pa0                  , default value = 0x04
    0x04,  // 0x01: beta_pa1                , default value = 0x04
    0x04,  // 0x02: beta_pa2                , default value = 0x04
    0x04,  // 0x03: beta_pa3                , default value = 0x04
    0x04,  // 0x04: beta_pa4                , default value = 0x04
    0x04,  // 0x05: beta_pa5                , default value = 0x04
    0x04,  // 0x06: beta_pa6                , default value = 0x04
    0x04,  // 0x07: beta_pb0                , default value = 0x04
    0x04,  // 0x08: beta_pb1                , default value = 0x04
    0x04,  // 0x09: beta_pb2                , default value = 0x04
    0x04,  // 0x0a: beta_pb3                , default value = 0x04
    0x00,  // 0x0b: beta_pb4                , default value = 0x04
    0x04,  // 0x0c: beta_pb5                , default value = 0x04
    0x04,  // 0x0d: beta_pb6                , default value = 0x04
    0x04,  // 0x0e: beta_pc0                , default value = 0x04
    0x04,  // 0x0f: beta_pc1                , default value = 0x04
    0x04,  // 0x10: beta_pc2                , default value = 0x04
    0x04,  // 0x11: beta_pc3                , default value = 0x04
    0x04,  // 0x12: beta_pc4                , default value = 0x04
    0x04,  // 0x13: beta_pc5                , default value = 0x04
    0x04,  // 0x14: beta_pc6                , default value = 0x04
    0x04,  // 0x15: epd_rollback(R)                 , default value = 0x04
    0x45,  // 0x16: R_SEL_PA1_PA0                   , default value = 0x22
    0x44,  // 0x17: R_SEL_PA3_PA2                   , default value = 0x22
    0x54,  // 0x18: R_SEL_PA5_PA4                   , default value = 0x22
    0x05,  // 0x19: R_SEL_PA6                       , default value = 0x02
    0x44,  // 0x1a: R_SEL_PB1_PB0                   , default value = 0x22
    0x44,  // 0x1b: R_SEL_PB3_PB2                   , default value = 0x22
    0x44,  // 0x1c: R_SEL_PB5_PB4                   , default value = 0x22
    0x04,  // 0x1d: R_SEL_PB6                       , default value = 0x02
    0x45,  // 0x1e: R_SEL_PC1_PC0                   , default value = 0x22
    0x44,  // 0x1f: R_SEL_PC3_PC2                   , default value = 0x22
    0x44,  // 0x20: R_SEL_PC5_PC4                   , default value = 0x22
    0x04,  // 0x21: R_SEL_PC6                       , default value = 0x02
    0x01,  // 0x22: FIL1_CONF                       , default value = 0x08
    0x24,  // 0x23: Down_Sampling_Rate              , default value = 0x08
    0x01,  // 0x24: FIL2_CONF                       , default value = 0x08
    0xff,  // 0x25: INIT_CAL                        , default value = 0xFF
    0x10,  // 0x26: ACAL_INTV                       , default value = 0x10
    0x0a,  // 0x27: AIC_Wait_Time                   , default value = 0x0A
    0x0a,  // 0x28: Idle_Enter_Time                 , default value = 0x0A
```

```
0x10,  // 0x29: Cluster_Strength_TH          , default value = 0x10
0x08,  // 0x2a: I2A_TH_pa0                   , default value = 0x0A
0x08,  // 0x2b: I2A_TH_pa1                   , default value = 0x0A
0x08,  // 0x2c: I2A_TH_pa2                   , default value = 0x0A
0x08,  // 0x2d: I2A_TH_pa3                   , default value = 0x0A
0x08,  // 0x2e: I2A_TH_pa4                   , default value = 0x0A
0x08,  // 0x2f: I2A_TH_pa5                   , default value = 0x0A
0x08,  // 0x30: I2A_TH_pa6                   , default value = 0x0A
0x08,  // 0x31: I2A_TH_pb0                   , default value = 0x0A
0x08,  // 0x32: I2A_TH_pb1                   , default value = 0x0A
0x08,  // 0x33: I2A_TH_pb2                   , default value = 0x0A
0x08,  // 0x34: I2A_TH_pb3                   , default value = 0x0A
0x08,  // 0x35: I2A_TH_pb4                   , default value = 0x0A
0x08,  // 0x36: I2A_TH_pb5                   , default value = 0x0A
0x08,  // 0x37: I2A_TH_pb6                   , default value = 0x0A
0x08,  // 0x38: I2A_TH_pc0                   , default value = 0x0A
0x08,  // 0x39: I2A_TH_pc1                   , default value = 0x0A
0x08,  // 0x3a: I2A_TH_pc2                   , default value = 0x0A
0x08,  // 0x3b: I2A_TH_pc3                   , default value = 0x0A
0x08,  // 0x3c: I2A_TH_pc4                   , default value = 0x0A
0x08,  // 0x3d: I2A_TH_pc5                   , default value = 0x0A
0x08,  // 0x3e: I2A_TH_pc6                   , default value = 0x0A
0x10,  // 0x3f: Touch_TH_pa0                 , default value = 0x10
0x10,  // 0x40: Touch_TH_pa1                 , default value = 0x10
0x10,  // 0x41: Touch_TH_pa2                 , default value = 0x10
0x10,  // 0x42: Touch_TH_pa3                 , default value = 0x10
0x10,  // 0x43: Touch_TH_pa4                 , default value = 0x10
0x10,  // 0x44: Touch_TH_pa5                 , default value = 0x10
0x10,  // 0x45: Touch_TH_pa6                 , default value = 0x10
0x10,  // 0x46: Touch_TH_pb0                 , default value = 0x10
0x10,  // 0x47: Touch_TH_pb1                 , default value = 0x10
0x10,  // 0x48: Touch_TH_pb2                 , default value = 0x10
0x10,  // 0x49: Touch_TH_pb3                 , default value = 0x10
0x10,  // 0x4a: Touch_TH_pb4                 , default value = 0x10
0x10,  // 0x4b: Touch_TH_pb5                 , default value = 0x10
0x10,  // 0x4c: Touch_TH_pb6                 , default value = 0x10
0x10,  // 0x4d: Touch_TH_pc0                 , default value = 0x10
0x10,  // 0x4e: Touch_TH_pc1                 , default value = 0x10
0x10,  // 0x4f: Touch_TH_pc2                 , default value = 0x10
0x10,  // 0x50: Touch_TH_pc3                 , default value = 0x10
0x10,  // 0x51: Touch_TH_pc4                 , default value = 0x10
0x10,  // 0x52: Touch_TH_pc5                 , default value = 0x10
0x10,  // 0x53: Touch_TH_pc6                 , default value = 0x10
0x01,  // 0x54: APIS                         , default value = 0x01
0xe0,  // 0x55: PIN_MAP_pa0                  , default value = 0xC0
0xe1,  // 0x56: PIN_MAP_pa1                  , default value = 0xC0
0xe2,  // 0x57: PIN_MAP_pa2                  , default value = 0xC0
0xe3,  // 0x58: PIN_MAP_pa3                  , default value = 0xC0
0xe4,  // 0x59: PIN_MAP_pa4                  , default value = 0xC0
0xe5,  // 0x5a: PIN_MAP_pa5                  , default value = 0xC0
0xc0,  // 0x5b: PIN_MAP_pa6                  , default value = 0xC0
0xc1,  // 0x5c: PIN_MAP_pb0                  , default value = 0xC0
0xc2,  // 0x5d: PIN_MAP_pb1                  , default value = 0xC0
0xc3,  // 0x5e: PIN_MAP_pb2                  , default value = 0xC0
0xc4,  // 0x5f: PIN_MAP_pb3                  , default value = 0xC0
0xc5,  // 0x60: PIN_MAP_pb4                  , default value = 0xC0
0xc6,  // 0x61: PIN_MAP_pb5                  , default value = 0xC0
0xc7,  // 0x62: PIN_MAP_pb6                  , default value = 0xC0
0xc8,  // 0x63: PIN_MAP_pc0                  , default value = 0xC0
0x80,  // 0x64: PIN_MAP_pc1                  , default value = 0xC0
0x80,  // 0x65: PIN_MAP_pc2                  , default value = 0xC0
```

```
    0x80,  // 0x66: PIN_MAP_pc3                  , default value = 0xC0
    0x80,  // 0x67: PIN_MAP_pc4                  , default value = 0xC0
    0x80,  // 0x68: PIN_MAP_pc5                  , default value = 0xC0
    0x80,  // 0x69: PIN_MAP_pc6                  , default value = 0xC0
    0x40,  // 0x6a: Resolution_X_L               , default value = 0xFF
    0x06,  // 0x6b: Resolution_X_M               , default value = 0x05
    0x84,  // 0x6c: Resolution_Y_L               , default value = 0xFF
    0x03,  // 0x6d: Resolution_Y_M               , default value = 0x05
    0x10,  // 0x6e: X_Cluster_Strength_gain(R)   , default value = 0x10
    0x10,  // 0x6f: Y_Cluster_Strength_gain(R)   , default value = 0x10
    0x01,  // 0x70: control1                     , default value = 0x01
    0x95,  // 0x71: control2                     , default value = 0x13
    0x00,  // 0x72: CLK_CONFIG                   , default value = 0x00
    0x00,  // 0x73: GPO_pa                       , default value = 0x00
    0x00,  // 0x74: GPO_pb                       , default value = 0x00
    0x00,  // 0x75: GPO_pc                       , default value = 0x00
    0x18,  // 0x76: INT_MASK                     , default value = 0x18
    0x00,  // 0x77: INT_CLEAR                    , default value = 0x00
    0x00,  // 0x78: ESD_Check                    , default value = 0x00
    0x08,  // 0x79: Noise_TH                     , default value = 0x05
    0x09,  // 0x7a: NUM_CH_X                     , default value = 0x00
    0x06,  // 0x7b: NUM_CH_Y                     , default value = 0x00
    0x01,  // 0x7c: BETA_GLOBAL                  , default value = 0x06
    0x10,  // 0x7d: ICAL_INTV                    , default value = 0x10
    0x01,  // 0x7e: PDU_Latency(R)               , default value = 0x01
    0x1f,  // 0x7f: ICAL_DUR                     , default value = 0x1F
    0x32,  // 0x80: epd_recover(R)               , default value = 0x32
   0x00,  // 0x81: ldo_control                   , default value = 0x00
};
```

A-2 Application Examples

A-2-1 3x4 touch button with individual backlight control by touch button



Here, 12-ch are set to button mode and 7-ch are set to GPO mode. Pin mapping is made by setting register 0x55 ~ 0x69. Please refer “Tuning Software” section that guide you how to set button mode and GPO mode.

As an example of button-4 touch, the corresponding touch signal (0xba ~ 0xbc) and interrupt signal of either TINT or GINT (0x70) are generated. And, external MCU can read out in a scheme of either Interrupt-based or signal polling-based. After MCU detects that Button 4 is touched, MCU makes DPO1, DPO4, DPO6, DPO7 logic-high and DPO2, DPO3, DPO5 logic-low. Then, LED corresponding Button 4 will turn on.

For simplicity, resistor to control LED brightness is omitted to the above circuit. If you use the same color LED, then the resistors can be added to DPO pins (DPO1, DPO2, DPO3). But, if you use different color LED, then resistor must be added with LED serially.

A-3 Frequently Asked Questions (FAQ)

A-3-1 Sensitivity

Q1. Parameters to control touch sensitivity
A1. There are two parameters of touch threshold (so called Alpha) and R_SEL.

Q2. How to decide R_SEL value?
A2. As R_SEL increase, all P-values (raw P, filtered P, nP, and pP-value) are also increasing. Notice that pP-value is made of nP-value minus noise threshold. For touch button application, nP-value is compared with touch threshold. Here, nP-value should be larger than 20. For touchpad application, pP-value is used to calculate position coordinate. Here, pP-value should be larger than 80. For more information, please refer to section 7-4 and 7-5.

Q3. How to decide touch threshold?
A3. After nP-value is measured at no-touch state, decide touch threshold as about twice of nP-value variation. Here, nP-value variation at no-touch state is decided by application circuits such as power noise or other device noise.

A-3-2 Electrical Noise Handling

Q1. Parameters to reduce electrical noise problem
A1. ATA5009 has internal noise filters (see section 7-3), touch threshold for button application, noise threshold for touchpad application.

Q2. How to decide internal noise filter parameters
A2. Depending upon noise sources, the filter parameters are determined by experiment. Generally speaking, Filter1 (0x22) is for high-frequency noise pre-processing, Down_Sampler and Filter2 are for mid-frequency noise filtering. Refer to section 7-3 more.

Q3. How to decide touch threshold and noise threshold
A3. After the filter parameters are determined, nP-value deviation is measured. Generally, touch threshold and noise threshold are about twice of the deviation value. It is important to notice that we have to consider the worst case noise situation.

Q4. Does the APIS mode help to reduce abnormal operation by noise?
A4. Yes. For button applications, APIS mode helps. For examples of multiple touches are detected, then the APIS mode 1 makes the highest probable output. This case is rarely occurred, but APIS mode will ensure higher reliable touch output.

Q5. How external MCU helps ? And what the external MCU should do?
A5. As every input device shall do, any faulty touch input should be detected and concealed. Here, statistic concealing algorithm such as majority vote algorithm is widely used. In addition to this concealment process, a special User Interface such as confirming step increases reliability a lot.

A-3-3 Automatic Impedance Calibration (AIC)

Q1. What is AIC?
A1. AIC operation is same as widely used calibrating operation. Since ATA5009 detects time-delay caused by any impedance change, ATLab names 'Impedance Calibration'.

Q2. Why does AIC need?
A2. Human input device (HID) should work in various environments that are wide operating temperature, hard vibrating situations, wide range RF noisy environments, etc. AIC operation makes HID operation stable. For an example of water spray, AIC operation prevents from faulty touch signal and provides normal touch signal.

Q3. Parameters for AIC
A3. There are 7 parameters to control AIC.
- 2 AIC decision parameters: beta (0x00 ~ 0x14), beta global (0x7C),
- 5 AIC timing parameters: Initializing calibration time (0x25),
  AIC interval (0x26), AIC wait time (0x27),
  Idle calibration interval (0x7D), Idle calibration duration (0x7F)

Q4. Why does beta use raw-P value?
A4. beta is used to detect peak-to-peak P value. When human finger approaches to touch sensing electrode, the raw-P value includes some coupled noise signal on finger. Thus, raw-P value above reference value is compared with beta. It is important to notice that raw-P value is better to detect object approaching than filtered value.

Q5. Difference between beta global (0x7C) and beta (0x00 ~ 0x14)
A5. beta global is applied to all channels. If raw-P value above reference value of any channel is above beta global, then AIC does not work. Thus, beta global must be decided by measuring peak-to-peak of raw-P value in a normal operating environment.
In addition to beta global, beta is applied to each channel. When a certain channel is placed to higher noisy location, beta of the channel should be higher than others. Except the special case, beta is equal to beta global.

A-3-4 Dynamic range and its application

Q1. What is maximum dynamic range?
A1. ATA5009 has two-step pulse width control schemes. One is coarse step (named as Extended Pulse Delay: EPD) step and the other is fine step. The coarse step is made of 8-bit resolution and the fine step is of 10-bit resolution. Since one coarse step is approximately 4-time of one fine step, the maximum dynamic range is about 16-bit.

Q2. What are benefits of the 16-bit dynamic range?
A2. Under a given 1-bit sensitivity, wider dynamic range allows operation with a larger offset capacitance. One example is touch operation in water. Please refer to videos in YouTube (search with "DigiSensor"). The other example is ground shield of a sensing electrode to make a stable touch under RF noises.
For touchpad application of ATA5009, the wide dynamic range produces 13-bit 2-D touch interpolation. With proper touch sensor pattern and pin-mapping (0x55 ~ 0x69), the mapped channels produce 2-D position with X-Y resolution setting (0x6A ~ 0x6D).

A-3-5 Automatic recovering by electric shock

Q1. What is automatic recovery?
A1. By external electric shock such as electric static charge on finger or electric power spike, register values inside ATA5009 may be changed. This electric shock shows up between touchpad and Vdd, because touchpad becomes a reference potential. When the electric shock occurs during I2C or SPI communication, wrong data may be read/write or wrong command may be executed. For a critical example, CRESET (Cold Reset) command may be miss-executed by the electric shock.

Q2. How does damage of electric shock detect?
A2. By monitoring ATA5009 register regularly, we can recognize whether the electric shock happens or not. ATA5009 provides ESD_Check register (0x78). Comparing with current value of ESD Check register value with default value is 0x00 , we can determine whether there is electric shock or not. It is important to notice that ATLab recommends to write 0xff to ESD_Check register (0x78) at cold reset or/and power boot.
Often, another register other than ES_Check register is damaged. Especially, damage of control1 register (0x70) and control2 register (0x71) are critical to operate normally.

Q3. Recommendation for automatic recovery?
A3. It is useful to compare as many registers as possible with between current values and written values. And, frequency of the comparison is executed also as often as possible.

A-3-6 Touchpad application

Q1. Self-Cap or Mutual-Cap?
A1. ATA5009 supports only self-cap

Q2. Does ATA5009 need any specific a touch pattern?
A2. ATA5009 includes position calculation engine on chip, which need diamond patterns. But, ATA5009 also supports any touch pattern by providing capacitive sensing outputs. Here, an external CPU needs to calculate positioning information from the capacitive sensing outputs.

Q3. What is maximum resolution?
A3. ATA5009 has 21-channels. By setting registers of 0xBF, 0xC0, 0xC1, and 0xC2, user can program the X-Y resolution.

Q4. Can ATA5009 support multiple chip operation for wider touchpad?
A4. Yes. With I2C interface, two ATA5009 chips are operating in parallel. With SPI interface, there is no theorectical limiting numbers by controlling ID_SS pin.

Q5. Can ATA5009 support multiple touches?
A5. Position outputs (0xBF, 0xC0, 0xC1, 0xC2) support a single touch case. For multi-touch operations, follow the next steps.

- Step 1. Monitor touch output of relevant channels of TOUT_PA/PB/PC (0xBA, 0xBB, 0xBC)
- Step 2. Judge multi-touch by checking gaps among X/Y touch outputs.
- Step 3. For multi-touch positions, then read corresponding pP_Values (0xD1 ~ 0xEC) and calculate each touch potion from the pP_Values

Q6. Can ATA5009 support touchpad gestures of multi-finger such as scrolls?
A6. If you need multi-touch gesture not multi-touch positions, then follow the next steps.

- Step 1. Monitor touch output of relevant channels of TOUT_PA/PB/PC (0xBA, 0xBB, 0xBC)
- Step 2. For gesture decisions, use the touch pattern
- Step 3. Use position output outputs (0xBF, 0xC0, 0xC1, 0xC2) for gesture size.

A-3-7 Temperature characteristics and Operation under abrupt temperature change

Q1. What are ATA5009 temperature characteristics?
A1. The p-value of ATA5009 is about - 2000 ppm/℃ temperature coefficient. The negative temperature coefficient (NTC) implies that p-value at a given capacitances is decreasing as temperature increases. For an example, this touch-on signal is produced with a given capacitance at room temperature, but the touch signal becomes off if temperature increases. Since AIC is not executed at the touch-on state, calibration against temperature does not work. Thus, please avoid rapid temperature changes during touch-on state. The temperature increasing should be within AIC period. Please optimize registers of Active_Calibration_Interval (0x26) and touch_TH (0x3F ~ 0x3E).

Q2. Can I keep touch-on during temperature changes?
A2. If your application is to keep touch-on state or touch-off state with a large temperature changes, then AIC does not work so that ATA5009 is not calibrated alone. There are two ways. One is to use temperature changing element(s) and the other is to implement an algorithm with an extra channel. For details, please ask to ATLab.

Q3. Can I use ATA5009 as a temperature sensor?
A3. Temperature coefficient of ATA5009 has ± 10% tolerance by lot. Thus, ATA5009 can use a secondary safety sensor, not primary sensor. For an example of oven, ATA5009 may use to prevent from over-heating.

Q4. Under abrupt temperature changes, can ATA5009 operate?
A4. Due to the temperature characteristics in A1 description, you may encounter abnormal symptoms in two cases of temperature-up at touch-on state and temperature-down at touch-off state.

- ✓ In the temperature-up case, touch-on signal at low temperature can become touch-off at the increasing temperature. Because p-value is decreasing as temperature increases, this symptom happens when p-value for touch-on is smaller than touch threshold value. General remedies are [1] increasing sensitivity (R_SEL registers) and [2] decreasing touch threshold value (Touch_TH registers).
- ✓ In the temperature-down case, touch-off signal at a high temperature can become touch-on at the decreasing temperature. The p-value is increased as temperature is decreasing. When the difference of p-value and reference value is larger than touch threshold value, then touch signal becomes ON. Here, the reference value is updated at every AIC. Therefore, general remedy is to apply AIC before abnormal symptom happens. Please reduce setting values of AIC interval (0x26) and AIC wait time (0x27) to be faster enough than temperature changing speed.

A-3-8 Reset

Q1. If VDDH is rising slower than R-C delay, then does Cold Rest by command work?

A1. Yes at internal LDO use application. When VDDH is rising slower than R-C delay of RESET_N, register values of ATA5009 are not reset. In this case, please apply Cold Reset command (0xfe).
If your application is to use external LDO, then please make sure that LDO output voltage must be smaller than VDDH.

Q2. What if ATA5009 is mal-functioned?

A2. There is little chance to be mal-function ATA5009. An excessive electric shock causes to change register values or/and to make communication fail. For examples of changes of Control1 register (0x70) and Control2 regitser (0x71) lead catastrophic states. Thus, please monitor key setting registers and make Hardware Rest (RESET_N pin).

A-3-9 How to make GPI (general purpose input) among touch inputs?

Q1. Can I use touch input pin to general purpose input?

A1. Any channel is defined by GPO (general purpose output), not GPI (general purpose input). However, touch input pin can be used by GPI by adding external parts. For just switch input application, please add serial capacitor between touch input pin and switch. ATA5009 detects capacitance changes by switch on/off. If you desire to use multiple levels GPI, then please ask for further helps to ATLab.

Revision History

Revision: 3.01
Updated: Jan. 11, 2017
- Page number is moved to right-end

Revision: 3.0
Updated: Dec. 21, 2016
- Add current sensing and ADC features

Revision: 2.5
Updated: Sept. 28, 2016
- Add application examples in appredix

Revision: 2.4
Updated: July 20, 2016
- Clarify uncertain content
- Update Fig. 5-9
- Delete 24-QFN

Revision: 2.3
Updated: Jun. 24, 2016
- Typesetting
- Change Figure 5-3

Revision: 2.2
Updated: May 10, 2016
- Add FAQ for temperature cycle test
- Adding index number in FAQ
- Change 0x26 register name

Revision: 2.1
Updated: Mar. 20, 2016
- Add FAQ for multi-touch and gesture

Revision: 2.0
Updated: Jan. 30, 2016
- Rearrange section and add software UI

Revision: 1.3
Updated: Jan. 5, 2016
- Adding FAQ of temperature

Revision: 1.2
Updated: Nov. 23, 2015
- Adding Register setting example: touchpad plus 5-touch button

Revision: 1.11
Updated: Nov. 11, 2015
- Typesetting

Revision: 1.1
Updated: Nov. 10, 2015
- Adding FAQ of automatic recovery
- Correct default value error of I2A_TH and Touch_TH

Revision: 1.0
Updated: Nov. 3, 2015
- Adding Tuning Software Section

Revision: 0.17
Updated: Sept. 25, 2015
- Update FAQ

Revision: 0.16
Updated: Sept. 21, 2015
- Update application circuits
- Add FAQ

Revision: 0.15
Updated: Aug 12, 2014
- modify Interrupt.

Revision: 0.14
Updated: JUN. 09, 2014
- 9-3-56 Added LDO control Register (Note 2)
- 5.3 Modified power connection

Revision: 0.13
Updated: JUN. 09, 2014
- Red letters → Black letters.
- Application circuit changed. (add RPU)
    - Type setting (5009)

Revision: 0.12
Updated: Mar. 11, 2014
- In chapter 7.2, the conversion formula is corrected.
- 3. Electrical Specifications is corrected.
- 5.2 Application Circuit is changed

Revision: 0.11
Updated: Mar. 06, 2014
- 2. Feature Summary is corrected.
- 3. Electrical Specifications is corrected.
- 5.2 Application Circuit is changed.

Revision: 0.1
Updated: Jan. 09, 2013
- Update from “ATA5008 User manual EN V0.1