Synaptics RMI4 Specification: NDA

Synaptics Confidential – Distributed Under NDA

PN: 511-000405-01 Rev. D

New in this revision

Function or Section	*Details*	*See Page…*
Function $01	Added new query registers: F01_RMI_Query42, 43, 44	48
	Added new control registers: F01_RMI_Ctrl06, 07, 08	55
Function $1A	New Function $1A: 0-D capacitive button sensors	61
Function $11	Added information queries (queries 12 to 26)	102 to 103
	Added jitter filter registers	102 and 125 to 126
	Added 8-bit W support registers	102 and 138
	Added pen registers	113 and 117
	Added large object detection	121
	Add gapless finger reporting run-time and tuning parameters	122 to 123 and 138
	Added Chiral Scroll registers	121 and 135
	Add registers for 2-D adjustable transmitter and receiver	126
Function $30	Added mechanical mouse button register	157
Function $31	New Function $31: LED	162
Function $34	Added Sensor ID command	169
Function $54	Added Force Fast Relaxation register	183
	Added test report types	195
	Added registers for multi-metric firmware noise mitigation	192 and 201

Copyright

Trademarks

Synaptics, the Synaptics logo, ClearPad and TouchPad are trademarks of Synaptics Incorporated.

All other brand names or trademarks are the property of their respective owners.

Notice

Conventions used in this document

The table below describes the documentation conventions. These conventions are used in all Synaptics technical literature.

Term	*Meaning*
$	Hexadecimal numbers are marked with a leading '$' sign: The number $7FF is equal to 2047 decimal.
—	Bits shown as "— " in register diagrams are equivalent to bits marked *Reserved.*
italics	*Italicized* words introduce a term described in the adjacent text or in the Glossary.
Reserved	*Reserved* is used to signify a bit or bit-field not currently used in any (published) way.
`Courier`	Courier font is used for text to be entered on a command line or in a program, or for text output from a device.
	This "caution" icon is used to indicate information about something bad that might happen if the guidelines for usage are not followed, or if care is not taken.

Contents

1. Introduction

This document defines a register-oriented protocol for use in Synaptics® embedded products. The overall protocol is known as RMI: the Register Mapped Interface. RMI uses a "register map" model that is convenient and familiar to host system developers.

The basic goals of RMI are:

1. To support a large and varied product line, with an emphasis on forward-, backward-, and cross-compatibility and consistency among Synaptics products;
2. To employ industry-standard I²C SMBus, and SPI-based interfaces, following the familiar and easy-to-use "register" model that is commonly found in devices with these interfaces; and
3. To be easy to document, understand, and use from the perspective of implementers of RMI drivers and systems incorporating RMI devices. RMI is designed so that any given RMI product can be documented concisely. For example, the numbering of functions and data sources is elaborate when considering the RMI protocol as a whole, but in each specific RMI device the resulting register map is straightforward and easy to use.

Each RMI product uses a particular physical interface (I²C, SMBus, or SPI) to access a particular register set tailored to the product. But RMI itself is a platform protocol that ties together the common aspects of all physical interfaces and all register maps of Synaptics' various embedded products.

1.1. Conventions used in RMI documentation

Bits within a byte, register, or other quantity are numbered with bit 0 as the least significant bit of the register or quantity. Ranges of bits are denoted *n*:*m* for the field of bits numbered *n* down to *m*, inclusive. For example, bits 7:4 comprise the most significant four bits of an 8-bit register.

All signed quantities in RMI are expressed in two's complement hexadecimal notation, where the most significant bit is taken as a sign bit. For example, a signed 8-bit byte is $00 to encode 0, $7F to encode +127, $80 to encode –128, and $FF to encode –1. A signed 6-bit register field would be $00 to encode 0, $1F to encode +31 (the largest value that can be encoded in a signed 6-bit field), $20 to encode –32 (the smallest value that can be encoded), and $3F to encode –1.

2. RMI structure

RMI, the Register Mapped Interface, is a communications interface for use with Synaptics modules. RMI is built upon industry-standard physical interfaces. Initially, RMI offers a choice of I²C, SMBus, or SPI. RMI communications involve two entities: The *host,* typically the main system processor, is the master. The *device,* typically a chip or module supplied by Synaptics, is a slave.

For systems with multiple hosts or multiple devices, RMI relies on the arbitration and addressing mechanisms of the underlying physical interface. For example:

- In SPI-based RMI systems with multiple devices, the host might generate a separate SSB signal for each device.
- In I²C or SMBus-based RMI systems with multiple hosts, the hosts might use bus arbitration and Repeated-Start transactions to negotiate safe shared access to a device.

2.1. RMI functions

RMI defines a standard set of *functions* that define features such as 2-D TouchPad™ sensors and brightness-controlled LEDs. The features and capabilities of an RMI device arise from the set of RMI functions that are included in the device. A particular RMI-based product will include or omit each possible function depending on the specific needs of the product. Certain RMI functions may only be supported on specific Synaptics touch controllers. For example, an RMI function to allow in-system reprogramming of Flash will only be available on Flash-based touch controllers. Each function is largely independent of any other functions that might be present in the same device.

Functions are consistently defined among all devices that include them. For example, Function $01 always corresponds to general device control features, and the registers associated with Function $01 are defined in a consistent way in all RMI devices that include Function $01. For devices that do not require Function $01, the registers associated with Function $01 are unimplemented.

Some RMI functions are completely universal, with a fixed definition that is identical in any product that includes them. Other RMI functions represent more general capabilities, with parameters that may be chosen differently from one product to another. For example, a function that supports 1-D Sensor operation may involve one linear strip sensor in one product, and two closed-loop sensors in another. Still other RMI functions may define flexible capabilities, such as the ability to specify the functionality of a device via run-time configuration methods. In general, each RMI function defines the set of:

- The control registers that configure its operation modes,
- The data registers that report information back to the host,
- The interrupt sources that signal changes in the contents of its data registers,
- The command registers that implement function-specific commands, and
- The query registers that describe its specific capabilities.

RMI functions are not required to implement all of these registers. The documentation for each RMI function defines the set of registers implemented by that function, as well as their contents.

2.1.1. Function numbers

RMI functions are identified by an informal name and a standardized identifying number. Function numbers are used to identify the capabilities supported by a device. The identifying number for an RMI function is an 8-bit integer in the range $01–$FF. The standard function numbers are shown in Table 1.

Table 1. RMI functions

Function	Purpose	See page
$01	RMI Device Control	45
$1A	0-D capacitive button sensors	61
$05	Image Reporting	68
$07	Image Reporting	75
$08	BIST	82
$09	BIST	87
$11	2-D TouchPad sensors	93
$19	0-D capacitive button sensors	141
$30	GPIO/LEDs	148
$31	LEDs	162
$34	Flash Memory Management	163
$36	Auxiliary ADC	174
$54	Test Reporting	176

2.2. Registers

RMI is defined in terms of a set of logical *registers* that appear within a register address map. The host communicates with the device by reading and writing the device's registers through physical interface transactions.

All registers in RMI are 8 bits wide. Quantities larger than a byte are held in several consecutive registers which are typically read or written as a group.

Certain multi-byte quantities may *require* that the host must read or write them as a group, called a *coherent region*. The general rules regarding the read and write access of a coherent region are discussed in section 2.6.

Registers are identified by 16-bit addresses. Where '$' signifies hexadecimal notation, the register addresses range from $0000 to $FFFF. Each address in this range potentially identifies one byte of register data. Only a few of the 65536 potential addresses are actually implemented; other addresses are marked *reserved*.

The register address space is divided into *pages* of related registers. Each page consists of 256 registers in the range $*xx*00–$*xx*FF. RMI devices typically export their entire customer-oriented register set within the first page of register addresses, covering the address range $0000-$00FF. See section 2.3 for details on the register address map organization.

Although in principle an RMI device can do anything in response to a read or a write of any register, RMI defines these standard types of registers:

- Query registers,
- Control registers,
- Data registers, and
- Command registers.

2.2.1. Query registers

Query registers are read-only registers that allow the host driver to determine what kind of RMI device is attached and what features it includes. Most hosts in embedded systems will never read the query registers at all, but platform host drivers and diagnostic tools may find these registers useful.

Query registers typically report constant data read from ROM on the device. Query registers may also report information that can change dynamically, based on how a device might have been configured.

The host should not write to a query register, but in any case writes to query registers are ignored.

Reserved query bits typically read as '0', but the host should ignore reserved query bits for forward compatibility with future RMI devices that might use these bits for additional query information.

2.2.2. Control registers

Control registers allow the host to initialize the device and control its functions. A control register generally looks like a readable and writable RAM location: The host can write data to the register, and the host can read the current contents of the register back without side effects.

Control registers generally do not change except when explicitly written by the host. However, this is merely a guideline and not a strict requirement. An RMI product or function may define control-like registers that change for other reasons. An example of this would be that a control register also reports status information, where the status information might change spontaneously.

Each control register has a defined reset value. When the device resets for any reason, all the control registers revert to their reset values. Flash-based products may allow for in-system changes to these default reset values. Consult the product-specific documentation to see which control registers allow flash-settable defaults.

Writing to a control register generally affects some aspect of the device's operation, either immediately or at a later time. Some control registers may also have side effects upon writing. Any such side effects are described in the documentation for the register.

A control register typically holds some *parameter* that configures the device. Parameters may have different sizes. The smallest parameter would be represented as a single bit. Larger parameters could fill an entire register.

Parameters larger than the size of a single register are allowed to span multiple registers. Sometimes the fixed properties of a particular device, such as the number of strip sensors in a 1-D function, are also referred to in RMI as *parameters*. Some parts of a control register may be marked *reserved* or "—", and some whole registers in a group of control registers may be *reserved*.

Reserved bits normally reset to '0'; these bits may harmlessly be written to '0', but the behavior of the device is undefined if the host writes a reserved bit to '1'. For example, some reserved control bits may activate undocumented or proprietary features of the device when written to '1', and those undocumented features may change without notice from one version of the product to another.

Similarly, some possible settings of a control register may be marked *reserved,* and the behavior of the device is undefined if the host sets a register to a reserved value.

Some control registers or parts of control registers are implemented only in some versions or models of a device. When a control register is *unimplemented*, it resets to a suitable value reflecting the fixed behavior that is implemented in its place (for example, a fixed sensitivity setting, or a fixed '0' enable bit for an unimplemented mode). At the implementation's discretion, an unimplemented register or register bit may be treated as a query register (writes are ignored regardless of the data written), or as a control register that does nothing (writes change the contents of the register but have no other effect on the device). In all cases, writes to unimplemented control bits are harmless.

Similarly, some possible settings of a control register may be unimplemented in some versions or models of a device; writing a control register to a setting that is unimplemented on the device has an undefined effect, but the effect is generally harmless and most often corresponds to one of the implemented settings of the register.

2.2.2.1. Device configuration

The complete set of parameters (both fixed and adjustable) contained in the control registers of an RMI device is together called the *configuration* of the device. A device can be *configured* at run-time by writing each control register in the device with the appropriate configuration data.

The amount of run-time configuration required will be specific to both the product and the application it is being used for. Some RMI products used in certain applications may not require any run-time configuration.

Flash-based products may contain their default device configuration in a special area of the Flash that can be updated without requiring a full firmware update. This would allow the configuration to be updated in-system. For more information, consult the product-specific documentation.

2.2.3. Data registers

Data registers report sensor readings and other input data to the host. Data registers are typically read-only in nature. Data registers may also support special semantics such 'read/clear' where a certain bit or bits within the data register might be automatically cleared after the register is read. Data registers can generate interrupt requests at certain times or at a certain rate. The special structure and behavior of RMI data registers are detailed in section 2.7.

Data registers may be defined to contain status information that might change spontaneously during device operation. Spontaneous changes to the status fields in a data register are capable of generating interrupts, while spontaneous changes to the status fields in a control register are not.

Some parts of some data registers are marked *reserved*; host software should always ignore these bits. RMI devices typically report '0' in all *reserved* data register bits, but the host should not rely on this.

2.2.4. Command registers

Command registers are read/write registers that allow the host to perform discrete commands or to signal discrete events on the device. Each bit in a command register corresponds to a possible command. Command register bits are special in that the host can only write them from '0' to '1'. Writing a '0' to a command bit leaves the state of the bit untouched by the write operation. Writing a '1' to a command bit issues the command. The command bit automatically clears to '0' when the command completes.

Some commands may complete instantaneously; their bits will never read as '1'. Other commands may take some time to complete. A host may either poll the command register to see when the command is complete or it may proceed immediately knowing that the command will execute in due course.

Certain command register bits may also be set to '1' by the device itself. RMI does not define what happens if a '1' is written to a command bit that is still '1' from a previous posting of the same command. The behavior depends on the particular command and function. For this reason, the host should never use a read/modify/write operation to write a bit or bits in a command register.

It is acceptable to write a '1' to one command bit while other command bits are already '1' because of previously-posted, still-pending commands. The result is that several commands will be posted at once. The order in which the device executes these pending commands is implementation-defined, and may not be the same as the order in which the commands were posted. In situations where the order of command execution is significant, the host should read and wait until the command register becomes $00 before writing a new command.

A command bit that causes an RMI device to reset will be irregular in that the RMI host interface will go "off the air" when the reset command is posted. After posting a reset command, it is not meaningful to poll the command register to wait for completion of the command, nor to post another command at the same time as a reset command is pending.

Unused bits in a command register are marked *reserved* or *unimplemented.* The host must never write a '1' to a reserved or unimplemented command bit. For example, some reserved command bits may activate undocumented or proprietary features of the device when written to '1' and these undocumented or proprietary features are subject to change without notice.

Reserved or unimplemented command register bits, and wholly reserved or unimplemented command registers, are not required to behave as described in this section when written to '1'. Writing $00 to a command register has no effect, and is harmless.

2.3. Register map organization

An RMI device always contains one or more RMI functions. Each of the RMI functions present in a device define the set of control, command, data, and query registers required to implement their own specific functionality. The comprehensive register map for a given device is formed by organizing the registers defined by each function that it implements into a single, orderly register map. The organization process may be product dependent, but will always be deterministic in nature.

2.3.1. Register address pages

All registers defined by a particular RMI device are placed within a general 16-bit RMI register address space. This 16-bit RMI register map is divided into pages of 256 registers per page. The register page address defines the upper eight bits of the 16-bit RMI address. The register page offset, or more simply, the offset defines the lower eight bits of the 16-bit RMI address. A single page can contain the registers associated with one or more RMI functions.

Register pages are defined to be self-contained:

- All registers defined by a particular RMI function must live on a single page.
- RMI forbids reading or writing a sequence of registers that goes past the last address in a page. The effects of attempting to read or write beyond the end of a page are undefined.

By convention, RMI products will make every effort to place all of the customer-facing RMI functions on page $00. From a customer's point of view, this means that a typical RMI device will essentially appear to have an 8-bit address space. Proprietary RMI functions like diagnostic or manufacturing mode functions will typically not live in page $00.

2.3.2. Global registers

Global registers are defined to appear at the same offset within every page. These registers perform the same basic function regardless of which page they exist in. The Page Select register and the query registers collectively known as the Page Description table are global registers.

2.3.2.1. Page Select register

RMI registers are always addressed with a unique 16-bit address. However, not all physical layers are capable of supplying the full 16 bits of register address information in a single transfer. RMI defines that at a minimum, all physical layer implementations must be capable of supplying the low-order 8 bits of the full 16-bit RMI address. The Page Select register is used to supply any high-order address bits that the physical layer is incapable of sending. For example, SMBus physical layer transfers are defined to supply the low-order 8 bits of address information as part of each transfer.

For SMBus transfers, the Page Select register supplies the high-order 8 address bits of the register address. In contrast, SPI physical layer transfers supply the low order 15 bits of the 16 RMI address bits, so the Page Select register need only supply the most significant bit of the 16-bit RMI address. For SPI, bits 6 through 0 of the Page Select register are ignored.

Table 2. Page Select register

Offset	7	6	5	4	3	2	1	0
$xxFF	RMI address bits 15:8							

The Page Select register is defined to exist at the same page offset on every RMI address page. This means that the Page Select register can always be accessed, regardless of its current contents. The default page offset for the Page Select register is product-specific, although typically it is at offset $FF.

The default value of the Page Select register typically is $00.

2.3.3. Register grouping within an RMI function

The organization of a device's register map occurs at a few different levels. At the lowest level, RMI functions are always designed so that the various kinds of registers (control, data, command, and query) associated with a function are always grouped together in like-kind groups.

For example, all the data registers defined by a particular RMI function are defined as a single group of data registers with sequentially increasing address offsets.

The order of the registers within each like-kind group is strictly defined by the function's specification. This means that by knowing the start offset for a particular kind of register group associated with a particular function, the offsets of all of the other like-kind registers associated with that function can be determined.

2.3.4. Function grouping within a page

If a page contains more than one RMI function, then the like-kind groups of registers defined by each function are typically grouped together to form like-kind metagroups. For example, all groups of data registers defined by the functions on a page are grouped together in a single data metagroup. This approach allows a host to read the entire set of data registers located on a page in a single physical layer transfer, or to write a complete configuration to all control registers on a page in a single physical layer transfer.

The order of the groups within a metagroup is unspecified. Typically, the order of the metagroups is the data register group first (lowest page offsets), followed by the control register group, then the command register group, followed by the query register group. Usually there is no empty space between the metagroups.

2.3.5. Page Description table

The mechanisms describing the organization of registers within a function, and the organization of functions within a page, are sufficient to implement RMI products. However, the result of the organization processes is necessarily product-specific. While a typical customer will choose to write a product-specific driver for the particular product, there may also be situations where a single host may be required to interface to a variety of RMI devices.

RMI defines a discovery mechanism that allows an interested host to discover what RMI functionality is present in any particular RMI device. Not only that, the discovery mechanism supplies enough information to enable a host to reconstruct a complete view of the RMI register address map for any given RMI device.

The discovery mechanism is page-based. For devices that support the discovery mechanism, each page with at least one RMI function contains a set of special read-only registers within a Page Description table.

If a host reads the contents of the Page Description table on a particular address page, it can discover:

- what RMI functions exist on that page
- the starting page offsets for each kind of register block associated with each RMI function on that page
- the number of interrupt sources associated with each RMI function on that page
- the version of each RMI function on that page

By scanning for Page Description tables in the general RMI register map, a host can reconstruct the entire RMI register address map, along with the interrupt bit assignments in the RMI Interrupt Status and Interrupt Enable registers. RMI devices that are severely ROM constrained are permitted to omit the Page Description table.

The Page Description table is defined as a general properties query register, followed by an array of Function Descriptors. The contents of the Page Description table are stored from high page offsets down towards lower page offsets, starting at page offset PdtTop.

The PdtTop typically has a value of $EF. Products that place the top of their Page Description table somewhere other than address offset $EF will define the new location in their product-specific documentation. A pictorial view of the relationship between the Page Description table and the RMI register set in a device can be found in section 2.4.

Because the value $00 does not belong to the range of permissible RMI function numbers, the end of the Page Description table is marked by finding a Function Descriptor containing an RMI function number of $00.

2.3.5.1. Query register 0: Page Description table properties

This register describes some basic information regarding the Page Description table itself.

Table 3. Page Description table Properties Query register

Offset	7	6	5	4	3	2	1	0
PdtTop-0	—	NonStd PSR	—	—	—	—	—	—

The bits of this register are defined as follows:

NonStdPSR (QueryBase+0, bit 6)

When '1', this bit indicates that the Page Select register is located somewhere other than its standard location at page offset $FF. Under those circumstances, the location of the Page Select register is defined in the product-specific documentation.

The rest of the bits in this register are *reserved* for future use.

2.3.5.2. Query registers 1-N: Function Descriptors

The Function Descriptor query registers live in the Page Description table, starting at the address offset PdtTop-1.

These registers are defined in such a fashion that a host can discover exactly what RMI functions live on that page, where to find the various kinds of registers defined by each of those functions, and the interrupt source count for those functions. This is enough information to reconstruct the complete RMI-compliant address map for any RMI device.

Table 4. Function Descriptor registers

Offset	7	6	5	4	3	2	1	0
FuncDescriptor-0	Function Number							
FuncDescriptor-1	—	Function Version		—	—	Interrupt Source Count		
FuncDescriptor-2	Data Base							
FuncDescriptor-3	Control Base							
FuncDescriptor-4	Command Base							
FuncDescriptor-5	Query Base							

The first register in the Function Descriptor defines the RMI function number that is implemented within the current page of the address space. A function number with the value $00 indicates the end of the function descriptor array.

The second register in the Function Descriptor contains some miscellaneous information about the function:

- The *Function Version* field is typically '0'. A non-zero value indicates that the definition of the function has changed in some fashion that is not backwards compatible with the function's original specification. An updated specification will define the changes between function versions.
- The *Interrupt Source Count* defines the number of interrupt source bits that this RMI function needs to allocate within the RMI Interrupt Enable and Interrupt Status registers. The value 7 is reserved to indicate "more than 6 interrupt sources."

Interrupt source bits are allocated by scanning all pages in a device that contain Page Description tables (PDTs), starting at page $00xx. As each PDT is encountered, the interrupt source bits are allocated in the order that the Function Descriptors are encountered within the Function Descriptor table, starting from the Function Descriptor placed at the highest address in the table. In particular: the first Function Descriptor encountered that defines a non-zero count for its number of interrupt sources is assigned bit positions starting with bit 0 in the Interrupt Status and the Interrupt Enable registers. Subsequent descriptors that allocate additional interrupt sources are allocated at the first available least-significant interrupt source bit.

For example: An RMI device contains three hypothetical functions, Function $10, Function $20, and Function $30, appearing in the function descriptor table in that sequence. Assume that Function $10 defines two interrupt sources.

Function $20 defines zero interrupt sources, and Function $30 defines three interrupt sources. Function $10 gets processed first, so it allocates the first available least-significant interrupt bits: bits 0 and 1. Function $20 gets processed next, but does not allocate any interrupt request bits. Function $30 gets processed last. Since interrupt bits 0 and 1 have already been allocated, Function $30 allocates the least-significant interrupt bits that are still available: bits 2, 3, and 4.

This means that the Interrupt Status register and Interrupt Enable registers for that product appear as:

Table 5. Sample Device Control register

Offset	7	6	5	4	3	2	1	0
InterruptEnable Reg	---	---	---	F30 IEN2	F30 IEN1	F30 IEN0	F10 IEN1	F10 IEN0

If there are more interrupt bits allocated than will fit into a single register, then additional registers are allocated to hold them. This means that a complex RMI device may contain multiple Interrupt Enable and Interrupt Status registers. Devices that define multiple interrupt registers always define the additional registers at subsequent locations in the register map. Multiple registers of the same type are also known as *replicated registers*, and are described in section 2.3.6.

The third through sixth registers in the Function Descriptor define the base address offsets on the current page for each of the four kinds of registers groups. The PDT only supplies the base address for the different kinds of registers. The interpretation of the register space after the base address must be done in accordance with the definition of each function as described in the RMI specification. In particular, the contents of an RMI function query register may alter the interpretation of the register space.

In terms of base addresses, PDT entries should be considered to be totally separate from each other. For example, an RMI device may implement two functions that support one command register each.

The PDT entry for Function 1 might start the command register at offset $10, while the subsequent PDT for Function 2 is allowed to place its command register at offset $20. But a driver scanning this PDT should not make any inferences regarding Function 1, such as assuming that it implements $20 - $10 or the $10 command registers.

If the RMI specification for a particular function does not actually define a particular kind of register, the base address contents for that register kind is meaningless and should be ignored.

2.3.6. Replicated registers

RMI allows for the replication of either single registers or blocks of related registers in the register map. The replicated block may vary in size, but the registers or register blocks within are all identical in form. The replication count can always be determined from information contained in one or more query registers. Because of that, a host can properly account for all replicated areas while reconstructing a device register map using the RMI discovery mechanisms.

2.3.6.1. Replicated single registers

The simplest case of replication is where a singular register gets duplicated one or more times in the address map. These simple replicated registers are marked with a ".*" in their description in this specification. For example, RMI Function $11 (2-D) contains a set of replicated Sensor Mapping registers defined as 'F11_2D_Ctrl12.*'. The specification for F11_2D_Ctrl12.* states that the number of replicated registers can be determined from the *MaximumElectrodes* field in F11_2D_Query4. Therefore, if a particular product defines *MaximumElectrodes* to be the value 3, three F11_2D_Ctrl12 registers will be defined by the product.

In the register map for that product, those three replicated registers will appear as:

```
F11_2D_Ctrl12.0   Sensor Mapping Control
F11_2D_Ctrl12.1   Sensor Mapping Control
F11_2D_Ctrl12.2   Sensor Mapping Control
```

In the device register map, each of the replicated registers gets a replication suffix appended to the basic register description number, starting at '.0'.

The replication formula can be more complex than the simple example above. For example, every RMI product contains at least one F01_RMI_Data1.* Interrupt Status register. The actual number of Interrupt Status registers implemented by an RMI device can be calculated by counting the total number of Interrupt Sources in the device:

```
InterruptStatusRegisterCount = trunc((NumberOfInterruptSources + 7)/ 8)
```

If the number of interrupt sources in a particular product was 10, the interrupt status register count would be (10+7)/8 or 2. In the register map for that product, those two replicated registers would appear as:

```
F01_RMI_Data1.0   Interrupt Status
F01_RMI_Data1.1   Interrupt Status
```

There may be instances where the replication count is 1, or even 0. If the interrupt status count in the previous example was only 3, then the register map for that product would show a single Interrupt Status register as F01_RMI_Data1.0.

2.3.6.2. Replicated register blocks

There are times when a related block of registers needs to be replicated. For example, the RMI F$11 function defines a block of five registers (F11_2D_Data1 through F11_2D_Data5).

These five registers are used to specify the various pieces of information regarding the absolute position associated with a single finger. If the 2-D sensor is capable of supporting more than one finger (that is, if the *NumberOfFingers* query in F11_2D_Query1 is greater than 0), the block of per-finger reporting registers will be replicated for all subsequent fingers.

For example, if a particular device reports *NumberOfFingers* as '1' (meaning 1+1 or two fingers being reported), then there are two blocks of replicated finger registers:

```
F11_2D_Data1.0          X Position 11:4
F11_2D_Data2.0          Y Position 11:4
F11_2D_Data3.0          X Position 3:0, Y Position 3:0
F11_2D_Data4.0          WX, WY
F11_2D_Data5.0          Z
F11_2D_Data1.1          X Position 11:4
F11_2D_Data2.1          Y Position 11:4
F11_2D_Data3.1          X Position 3:0, Y Position 3:0
```

```
F11_2D_Data4.1          WX, WY
F11_2D_Data5.1          Z
```

The replication suffix is appended to each register in the block being replicated, and gets incremented for each subsequent block. In the example above, the '.0' replication suffix indicates the replicated block associated with the first finger, while '.1' indicates the block associated with the second finger.

2.4. Register map organization summary

As mentioned earlier, all RMI devices typically place all customer-centric registers on page $00. Customers should be able to consider that an RMI device contains a single page of addresses. Proprietary RMI functions that define manufacturing or diagnostic functions are usually placed somewhere other than page $00.

Each page in the RMI address space is organized as a self-contained unit, organized as follows:

- The Page Select register is placed at a consistent address in each page, as described in 2.3.2.1.
- The registers implemented by the set of RMI functions present on the page are organized according to the rules in 2.3.3 and 2.3.4.
- A Page Description table is [optionally] placed on each page containing at least one RMI function.

A sample register map is shown in Figure 1, describing page $00 of a hypothetical RMI device. By scanning the Page Description table downwards starting from register $EF, the host can determine that there are three RMI functions on the page (F$11, F$22, F$33) and that there were a total of three interrupts defined, so there must be exactly one Interrupt Status register and one Interrupt Mask register.

Note: For the purposes of this example, the functions F$11, F$22, and F$33 and their registers are 'imaginary', in that they are not intended to bear any resemblance to a real F$11, F$22, or F$33 that might get defined at some later date.

In this example, the F$11 data register 0 implements an Interrupt Status register. The bits within that register would be assigned as follows:

1. F$11 is the first function encountered in the Page Description table. Because it defines a single interrupt source, that interrupt source would be assigned bit 0 in the Interrupt Status register.
2. F$22 is encountered next but defines no interrupt sources, so no bits are allocated for F$22 in the Interrupt Status register.
3. F$33 is encountered last. It defines two interrupt sources, so it would be assigned bits 1 and 2 in the Interrupt Status register.

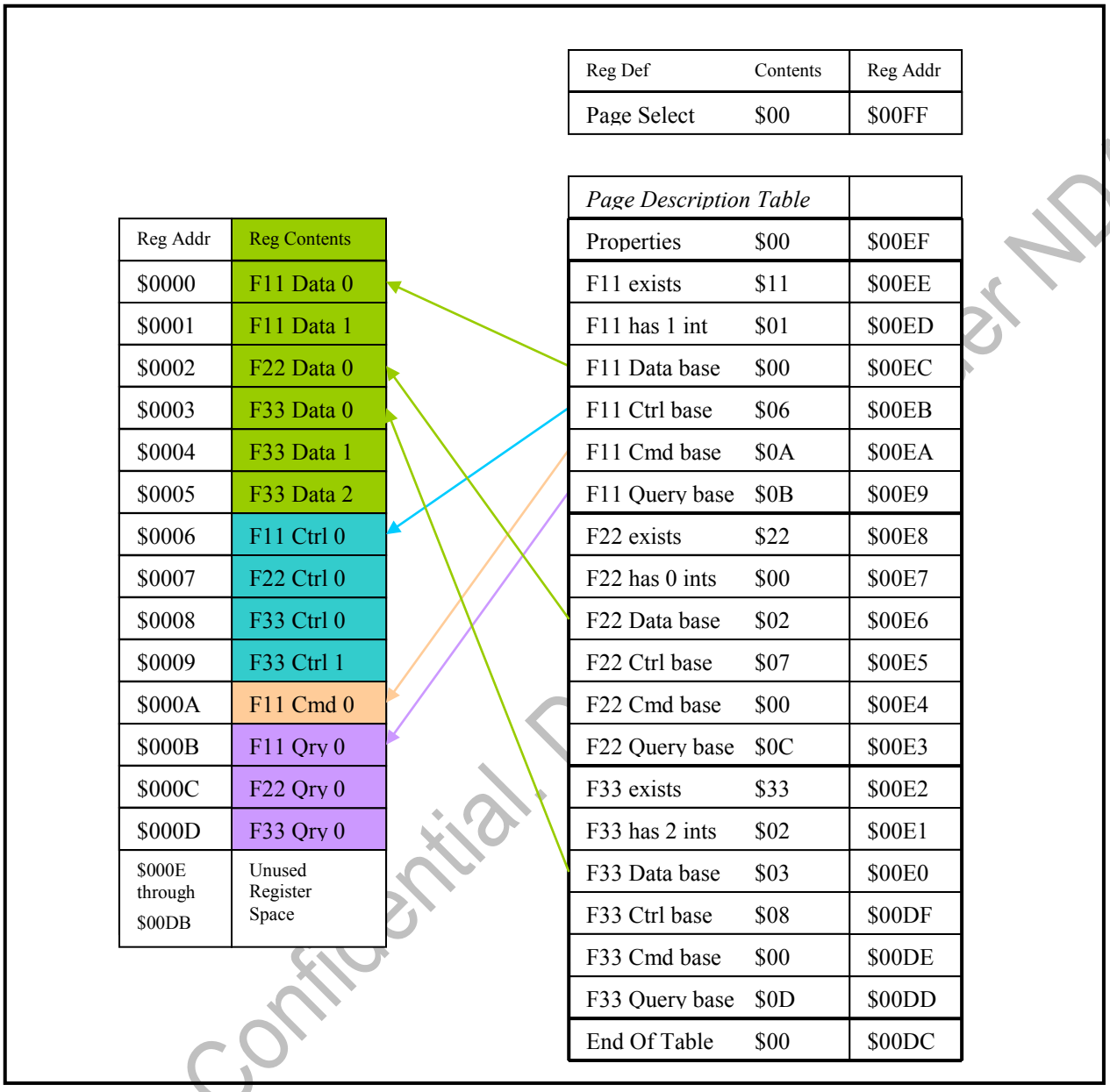

Figure 1. Sample register map

Note: Each Page Description table always contains base pointers for all four kinds of registers. For functions that do not define certain kinds of registers, a base pointer corresponding to an unimplemented kind of register is meaningless, and its value should be ignored.

In the example above, if the specification for a hypothetical Function $22 were to state that Function $22 defined no command registers, then the value returned by reading the Function $22 Command Base register should be ignored.

Important: The Page Description table only supplies the base address; to determine the number of registers, register usage, and other information, you must use queries.

2.5. RMI physical layer operations

This section summarizes the basic operations that are supported by every RMI physical layer.

2.5.1. Writing registers

The host may write to any *N* consecutive device registers starting at any register address *R* in a single transfer. The write transaction always "succeeds" as far as the RMI protocol is concerned. The value of *N* can be as small as 1 in order to support certain physical layer protocols like SMBus. The physical layer may also define an upper limit on the value of *N*.

A single register write operation may not span more than one register page. In other words, the *N* consecutive registers covered by a given write operation must have addresses that differ only in the low 8 bits. It is an error if the host performs a write operation that spans multiple pages; the effect is undefined and implementation-dependent.

As a special case, a write operation to a command register that causes a device to reset must write only to that command register, and must only set the command bit that initiates the reset operation. Most physical layers define a holdoff time after the Reset command before the host can again reliably access the RMI interface.

2.5.2. Reading registers

The host may read from any *N* consecutive device registers starting at any register address *R*. The read transaction always "succeeds" as far as the RMI protocol is concerned.

As with the write operations, the high and low limits on the value of *N* may be dependent on the physical layer implemented by a device. A single register read operation may not span more than one register page (in the sense defined above in section 2.5.1).

Preferably, the physical layer will allow the host to choose to adjust dynamically the number *N* of registers that it reads based on the first few data bytes it has read. However, RMI never *requires* the host to perform a read operation of non-fixed size, because some hosts lack this ability (for example, because of restrictions in their OS drivers).

2.5.3. Packet registers

A packet register is an RMI register with a special property: It holds many bytes of data rather than just one. Packet registers are sometimes called "FIFO registers" in RMI documentation. The figure below illustrates a segment of an RMI register map with a 7-byte-deep packet register at address 0x27.

The contents of a packet register can only be accessed by performing a block-read or block-write operation, which begins at the packet-register's address.

To read the packet register at address 0x27 in the figure below, perform a block read of seven bytes starting at address 0x27:

```
Host:     READ (0x27, 7)
Device:   0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC, 0xDE
```

Register 0x25	Register 0x26	Register 0x27	Register 0x28	Register 0x29	Register 0x2A	Register 0x2B
0xAA	0xBB	0x12	0xCC	0xDD	0xEE	0xFF
		0x34				
		0x56				
		0x78				
		0x9A				
		0xBC				
		0xDE				

Figure 2. RMI register map segment

If a packet register is not addressed directly, writes to it have no effect and reads from it return only a single 0x00 byte. For example, a block read of seven bytes starting at address 0x25 will return only a single '0x00' byte from register 0x27:

```
Host:     READ (0x25, 7)
Device:   0xAA, 0xBB, 0x00, 0xCC, 0xDD, 0xEE, 0xFF
```

A packet-register transaction always accesses the packet register starting from its first byte. For example, two consecutive three-byte reads from address 0x27 will not return the packet register's first six bytes; rather, they will return two copies of the packet register's first three bytes:

```
Host:     READ (0x27, 3)
Device:   0x12, 0x34, 0x56
Host:     READ (0x27, 3)
Device:   0x12, 0x34, 0x56 [not "0x78, 0x9A, 0xBC"]
```

Reading or writing beyond the last byte of a packet register has undefined results.

2.5.4. Signaling attention and interrupts

The device may signal the host for attention. From the host's point of view, the attention signal acts like a traditional interrupt signal. The host responds to the attention interrupt by reading the appropriate device registers to determine the reason for the attention request.

2.5.5. Robust operation

Every physical layer should provide for robust system operation in the presence of spontaneous resets at any time by either the device or the host.

Spontaneous resets on the device side are a fact of life in many capacitive touch sensor designs. Touch sensors, by their nature, are more exposed than most electronics to disruption by ESD (electrostatic discharges) during operation, particularly if the sensor area is framed by an open bezel instead of being under solid, uninterrupted plastic. Synaptics chips are designed to ensure that any such disruption results in a clean reset of the chip.

The RMI protocol is designed to allow the system to recover gracefully in the event of such a spontaneous device reset. Together, these designs allow robust touch sensing even in ESD-prone or otherwise error prone environments, as described in section 2.7.3.

Spontaneous resets on the host side are also a fact of life in embedded systems. If the host restarts suddenly, it may not have had time to shut down the peripherals in an orderly way. An RMI device should be prepared to accept a "reset" command no matter what was happening earlier, even if a previous RMI physical layer transaction was interrupted partway through, and even if a special command or mode was already in progress on the device.

2.6. Register coherence

Depending on the needs of the individual RMI functions, groups of registers may be defined as being *coherent* for the purposes of reading and/or writing. There are many reasons why registers may be grouped into a coherent region. Typically, a particular set of data being reported may span more than one register. In that case, defining the group of registers to be coherent indicates that they belong to the same snapshot in time. For example, consider a timer that reports a 10-bit timer count in two RMI registers, the two most-significant bits in a register at address *X*, and the eight least-significant bits in another register at address *X+1*. Because the timer count may be incrementing while the registers are being read, the following sequence of events could occur:

1. Timer count is $1FF.
2. Host reads the most significant Timer count register at addr *X*: $01.
3. Timer count increments to $200.
4. Host reads the least significant Timer count register at addr *X+1*: $00.

At this point, the host reconstructs the timer count from the two register reads. Because the two reads were not coherent in time, the host will believe that the timer count is $100 when the real timer count is $200.

To solve these problems, RMI allows functions to define groups of registers as being *coherent regions*. These coherent regions are treated specially by the RMI device so that the information they either report (for reads) or accept (for writes) has the correct temporal significance. In the example above, an RMI device would treat the timer as a coherent region.

The coherent sequence of events would go as follows:

1. Timer count is $1FF.
2. Host reads the most significant Timer count register from a coherent region at addr *X*: $01. The RMI function saves all the registers belonging to this coherent region in a special buffer. The saved count is $1FF.
3. Timer count increments to $200.
4. Host reads the least significant Timer count register from a coherent region at addr *X+1*. Because the second read is also from the same coherent region, the RMI device reads from the special buffer instead of the current Timer Count: $FF.

The host sees the value $1FF, which represents the time at which the host started reading the Timer count.

2.6.1. Write access into coherent regions

A set of write accesses to a write-coherent region is *committed* when the last register in the region is written. Writing that last register triggers an atomic update of all the registers in the coherent region.

If a single transfer happens to write into two or more adjacent coherent regions, multiple commit operations will occur during the course of the transfer as the last register in each region is written.

Most write-coherent regions require that the host write every register in the region as part of every write access to the region. However, the individual RMI functions are free to define write access methods that permit a host to write a meaningful subset of the registers in their coherency regions. Even so, any permitted register subset must always include the last register in the region; otherwise, the commit operation will not occur. These special access restrictions are defined in the function-specific sections of this document.

2.6.2. Read access into coherent regions

Read accesses into a coherent region are essentially 'invisible' to a host. During a read transfer, the first read access anywhere inside a coherent region cause a snapshot to be taken of all the registers that belong to the region. Subsequent register reads to the same region at sequentially increasing register addresses are serviced from the snapshot data, meaning that the subsequent register reads come from the same moment in time as the first read into the region.

A snapshot is invalidated (discarded) as soon as a register transfer occurs at any address other than a sequentially increasing address within the same region. This means that if a host repeatedly polls the same address within a region, a new snapshot is acquired for each poll operation. Once the host is satisfied with the results of the poll operation, subsequent reads at increasing addresses from within the coherent region are supplied from the final coherent snapshot. RMI functions typically place important status information that might be used for polling purposes at the start of a coherent region.

RMI functions that implement register read-access coherent regions describe their special access restrictions in the function-specific portions of this document.

2.7. Data reporting

An RMI device may have any number of *data sources*. In general, a data source corresponds to one independent sensor or input device, but in some cases a group of similar sensors (such as an array of buttons) are combined to form a single data source. A data source always defines one or more data registers.

2.7.1. Interrupt requests

RMI uses *interrupt requests* to signal whether or not a data source has information likely to be of interest to the host. Exactly what constitutes an interrupt, and when and at what rate new data will arrive, depend entirely on the kind of input device involved. The specification for each RMI function defines the exact rules for signaling interrupt requests for each of its data sources.

Each data source maintains an independent interrupt request state bit. The Interrupt Status register, described in section 4.2.2, reports the set of interrupt request bits in the device.

The Interrupt Request bit for a data source is '1' if the data source has an interrupt request, or '0' if there is no interrupt request for the source. Once a source has an interrupt request and its Interrupt Request bit has changed to '1', the bit remains at '1' until it is read, or (in some products) the host takes other actions that are documented to clear the interrupt request state of the data source.

The data registers always report meaningful data whether or not the interrupt request condition is present. Most often, an interrupt request state of '0' implies that the data registers are unchanged since the last time the host read the data.

However, some data sources may define a more selective "interrupt" criterion, so that certain "insignificant" data changes may occur without causing interrupt request to be asserted. Very simple hosts could even read the data registers by periodic polling, observing the data but completely ignoring the Interrupt Request bits and the attention signal.

The post-reset default state of the bits in the Interrupt Status and Interrupt Enable registers is product-specific. Consult the product-specific documentation for details.

2.7.2. Attention signal

The host in an RMI system is in charge of all transactions with the device. When the device has an interrupt request to report to the host, it uses an *attention* mechanism to alert the host that it is time to read the data registers. Typically, a host system connects the attention signal to an interrupt input. The attention signal is merely advisory; the host is free to read the data registers at any time. The form of the attention signal depends on the physical interface; for example, see section 3.3.4 for a description of the attention signal for RMI-on-SPI.

Synaptics can supply RMI devices in which the attention signal is active-high or active-low. The attention polarity is a build-time option, and is not configurable at run-time.

The attention signal is said to be *asserted* if it is in the state that alerts the host, (that is, high for active-high polarity or low for active-low polarity). The attention signal is in the *de-asserted* state when it is not asserted.

The attention signaling model is compatible with both level-triggered and edge-triggered interrupt inputs on the host.

Every RMI device defines at least one Interrupt Status register that includes up to 8 Interrupt Enable bits, one bit per data source. For devices with more than 8 data sources, enough extra Interrupt Status registers are defined to hold all the Interrupt Enable bits defined by all the data sources present.

The attention signal is asserted whenever at least one interrupt source has a ‘1’ in its bit of the interrupt enable mask and its Interrupt Request bit is also ‘1’. The attention signal may become asserted or de-asserted if the host writes to the control registers to change the Interrupt Enable bits.

Most data sources continue to work, and continue to operate their Interrupt Request bits, even if their Interrupt Enable bit is ‘0’. The Interrupt Enable bit merely controls whether interrupt requests on a source will send an attention signal to the host.

The attention signal is de-asserted by reading all of the Interrupt Status registers in a device. This means that a host driver should process the interrupt handlers for all interrupt sources that are reporting ‘1’ when the Interrupt Status is read.

The host can disable attention by setting all the Interrupt Enable bits to ‘0’, thereby forcing the assertion signal to its de-asserted level. This gives the host a way to “disable interrupts” on the device side. For example, in a system where several devices’ attention pins have been logically OR’d together to drive a host interrupt pin, the host might wish to disable interrupts from some of the devices while remaining sensitive to other devices. The attention signal is always asserted after device reset; see section 2.7.3.

2.7.3. Spontaneous resets

RMI is designed to be robust in the presence of spontaneous resets of RMI devices. Several properties of RMI are designed to work together to allow the host to reliably detect when the device has spontaneously reset:

- Every RMI function that implements an interrupt source can be configured to either assert or de-assert both its Interrupt Request and Interrupt Enable bits upon reset.
- Every RMI device contains a Device Control function, such as RMI Function $01. An RMI Device Control function is always configured to assert both its Interrupt Request and Interrupt Enable bits upon reset.

- The assertion of any Interrupt Request and its corresponding Interrupt Enable causes the attention interrupt signal to become asserted after a reset.
- The attention signal prompts the host to read Interrupt Status register in order to determine the source of the interrupt.
- When the host reads the Interrupt Status register, it will find a '1' in the DevStatus interrupt request flag. This prompts the host to read the Device Status register.
- The Device Status register indicates that the device has lost its configuration because of a spontaneous reset.

The host should respond to a spontaneous device reset by fully reinitializing the device. Depending on the nature of the overall system, the host may even wish to use a spontaneous reset in one RMI device as a cue to reinitialize other parts of the system.

Hosts that operate the device in its default configuration, without ever writing to any control registers, will not necessarily be able to detect a spontaneous reset because the DevStatus interrupt request flag will always be '1'. However, these hosts need not be cognizant of spontaneous resets; from their point of view, the resetting device will simply pause briefly in its reporting of sensor activity and then resume normal operation.

3. Standard RMI physical layers

RMI may be implemented atop a variety of physical interfaces:

- RMI on I²C: See section 3.1.
- RMI on SMBus: See section 3.2.
- RMI on four-wire SPI: See section 3.3.

3.1. I²C physical interface

Synaptics RMI-on-I²C devices are suitable for connecting directly to an industry-standard I²C host interface. This section describes the I²C physical layer. The RMI-on-I²C interface has been developed using version 2.1 of the *I²C Bus Specification*, dated January 2000. This document can be found at http://www.nxp.com/. The remainder of this section assumes that the reader has familiarity with the *I²C Bus Specification* document.

3.1.1. I²C transfer protocols

To communicate with an RMI-on- I²C device, a host needs to be able to:

1. Read one or more RMI registers starting from some RMI register address.
2. Write one or more RMI registers starting from some RMI register address.

The I²C bus specification imposes no limit to the number of registers that the host can read in a single transfer. However, RMI does not permit a physical layer transfer to cross a page boundary, so the practical limit is 256.

The I²C bus specification imposes no limit on how long a transfer can take, or how slowly a bus master is allowed to clock the bus. To meet this specification, Synaptics RMI devices will not impose any timeouts during any I²C transfer.

3.1.1.1. I²C transfer terminology

The following terms are used in the definition of the I²C transfer protocols.

Table 6. I²C transfer protocol terms

Term	*Definition*
S	Indicates an I²C “Start” condition
P	Indicates an I²C “Stop” condition
Sr	Indicates an I²C “Repeated Start” condition
A	Indicates an I²C “ACK” bit
N	Indicates an I²C “NAK” bit
SlaveAddr	The 7-bit Slave Address field in an I²C header byte
Wr	The 1-bit ‘write’ field in an I²C header byte (a Write always has the value 0)
Rd	The 1-bit ‘read’ field in an I²C header byte (a Read always has the value 1)

3.1.2. RMI register addressing

In order to minimize bus traffic, only the low 8 bits of the full 16-bit RMI register addresses is specified by a read or write transfer. The high 8 bits of the address are supplied by the Page Select register (see section 2.3.2.1).

3.1.3. Block read operations

The Block Read operation allows a host to read one or more RMI registers starting from a specified RMI address. The device starts reading from the RMI address specified by the read operation, and continues to send registers from consecutively incrementing RMI addresses until the host finally NAKs the transfer.

Figure 3 is an example of a Block Read operation in which the host reads one RMI register from address N.

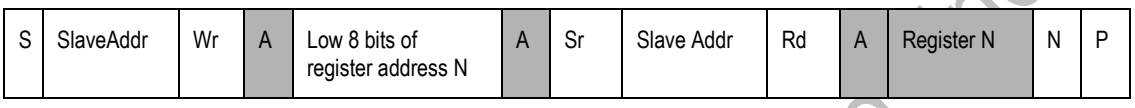

Figure 3. Block Read operation example: host reads one register

Figure 4 is an example of a Block Read operation in which the host reads four consecutive RMI registers starting from address N.

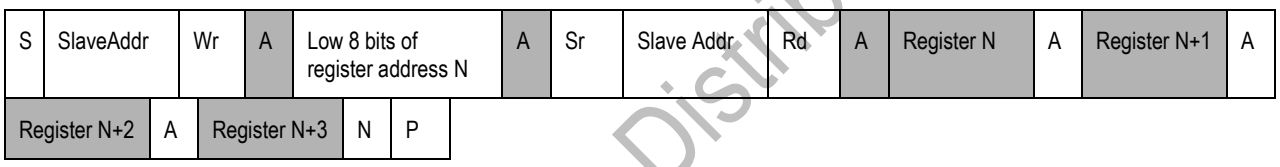

Figure 4. Block Read operation example: host reads four registers

Do not perform an I²C read operation that is not preceded by an I²C write of the low-order 8 bits of the RMI register address. The result of doing so is undefined.

3.1.3.1. Repeated starts

The Repeated Start separating the write of the RMI register address from the read of the register data ensures correct operation on an I²C bus that supports multiple bus masters. For a host bus that either does not require multi-master support or cannot generate Repeated Start events, Synaptics RMI-on-I²C devices will permit a host to replace a Repeated Start with a Stop event followed by a Start event.

3.1.4. Block write operations

The Block Write operation allows a host to write one or more RMI registers starting at a specified RMI address. The device starts writing data to the RMI address specified by the write operation, and continues to write registers to consecutively incrementing RMI addresses as long as the host keeps sending data. An RMI device will acknowledge (ACK) every byte that it receives.

Figure 5 is an example of a Block Write operation in which the host writes data to a single RMI register at address N.

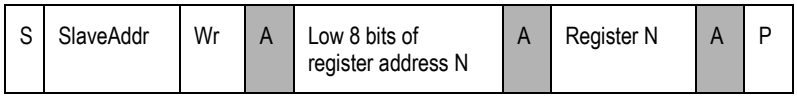

Figure 5. Block Write operation example: host writes to a single register

Figure 6 is an example of a Block Write operation in which the host writes three consecutive RMI registers starting with the register at address N.

S	SlaveAddr	Wr	A	Low 8 bits of register address N	A	Register N	A	Register N+1	A	Register N+2	A	P

Figure 6. Block Write operation example: host writes to three registe

3.1.5. I²C protocol compliance

The Synaptics I²C interface is designed to comply with the basic I²C protocol as described in the *I²C Bus Specification,* Version 2.1 by Philips. Conforming to this specification ensures the following:

- Synaptics modules correctly recognize and respond to Start events, Repeated Start events, and Stop events.
- Synaptics devices properly generate SCL "clock stretching" as a slave device.
- Synaptics modules support the 7-bit addressing mode.

3.1.5.1. Addressing modes

Synaptics devices do not support the 10-bit addressing extension to I²C. Synaptics Master-Slave modules master their transmissions to a host device using 7-bit addresses.

Synaptics modules can co-exist on the same I²C bus with other devices that support the 10-bit extended addressing mode. In addition, while Synaptics Slave-Only modules have 7-bit addresses, they can respond as slave device to a host/master that has a 10-bit address.

3.1.5.2. Data rate and clock stretching

Synaptics I²C devices can maintain either the "Fast Mode" data rate in the *I²C Bus Specification* at 400K bits per second, or the "Normal Mode" data rate of 100K bits per second while a byte is being clocked over the bus. In either case, a Synaptics RMI device may be required to perform an I²C Clock-Stretch operation in between the bytes of a transfer.

3.2. SMBus physical interface

Synaptics RMI-on-SMBus devices are suitable for connecting directly to an industry-standard SMBus host interface. This section describes the SMBus physical layer. The RMI-on-SMBus interface has been developed using version 2.0 of the *System Management Bus (SMBus) Specification*, dated August 3, 2000. This document can be found at http://www.smbus.org/.

SMBus represents a layer on top of a standard I²C interface. The main contribution of the SMBus layer is to define a set of standardized I²C transaction sequences (called SMBus 'transfer protocols') that are used to read or write data to a SMBus device. The SMBus transaction protocols supported by the Synaptics RMI-on-SMBus interface are defined in section 3.2.2.

3.2.1. RMI-on-SMBus addressing

RMI defines a 16-bit RMI register address space. When RMI is implemented using the SMBus physical layer, the size of the SMBus *Command Code* effectively limits the size of a register address to 8 bits. The Page Select register (see section 2.3.2.1) is used to supply the upper 8 bits of the 16-bit RMI address.

3.2.2. SMBus transfer protocols

To communicate with an RMI-on-SMBus device, a host needs to be able to do the following things:

1. Read one or more RMI registers starting from some RMI register address.
2. Write one or more RMI registers starting from some RMI register address.

RMI-on-SMBus supports the standard SMBus transfer protocols to read and write bytes, and to read and write words. Because all RMI registers are 8-bit registers, reading or writing a word really means to read or write a pair of sequential RMI byte-wide registers. The following subsections describe the SMBus read and write commands, using the notation found in the *System Management Bus (SMBus) Specification*, Version 2.0 of August 3, 2000.

3.2.2.1. SMBus transfer terminology

The following terms are used in the definition of the SMBus transfer protocols.

Table 7. SMBus transfer protocol terms

Term	Definition
S	Indicates an SMBus “Start” condition
P	Indicates an SMBus “Stop” condition
Sr	Indicates an SMBus “Repeated Start” condition
A	Indicates an SMBus “ACK” bit
N	Indicates an SMBus “NAK” bit
SlaveAddr	The 7-bit Slave Address field in an SMBus header byte
Wr	The 1-bit ‘write’ field in an SMBus header byte (a Write always has the value 0)
Rd	The 1-bit ‘read’ field in an SMBus header byte (a Read always has the value 1)

3.2.2.2. SMBus Byte Write

The SMBus *Byte Write* command writes a byte to a single register in an RMI-on-SMBus device. In its general form, the SMBus *Byte Write* command has this format:

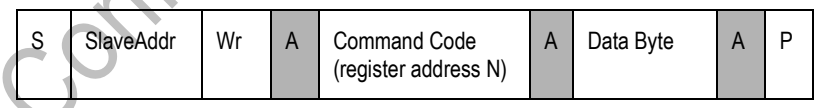

Figure 7. SMBus Byte Write command example

- The *Command Code* contains the low 8 bits of the address of the RMI register to be written. The Page Select register supplies the high 8 bits of the address. The result is a 16-bit RMI address ‘N’.
- The *Data Byte* is the value to write to the RMI register at address ‘N’.

3.2.2.3. SMBus Byte Read

The SMBus *Byte Read* command reads the contents of single register in an RMI-on-SMBus device. In its general form, the SMBus *Byte Read* command has this format:

S	SlaveAddr	Wr	A	Command Code (register address N)	A	Sr	Slave Addr	Rd	A	Data Byte	N	P

Figure 8. SMBus Byte Read command example

- The *Command Code* contains the low 8 bits of the address of the RMI register to be written. The Page Select register supplies the high 8 bits of the address. The result is a 16-bit RMI address 'N'.
- The *Data Byte* is the value that was read from the RMI register at address 'N'.
- All SMBus Read operations terminate with a NAK bit followed by a STOP bit.

3.2.2.4. SMBus Word Write

The SMBus *Word Write* command writes a pair of bytes to a sequential pair of registers in an RMI-on-SMBus device. In its general form, the SMBus *Word Write* command has this format:

S	SlaveAddr	Wr	A	Command Code (register address N)	A	Data Byte 0	A	Data Byte 1	A	P

Figure 9. SMBus Word Write command example

- The *Command Code* contains the low 8 bits of the address of the first RMI register to be written. The Page Select register supplies the high 8 bits of the address. The result is a 16-bit RMI address 'N'.
- *Data Byte 0* is written to the RMI address 'N'.
- *Data Byte 1* is written to the RMI address 'N'+1.

3.2.2.5. SMBus Word Read

The SMBus *Word Read* command reads the contents of a sequential pair of registers in an RMI-on-SMBus device. In its general form, the SMBus *Word Read* command has this format:

S	SlaveAddr	Wr	A	Command Code (register address N)	A	Sr	Slave Addr	Rd	A	Data Byte 0	A	Data Byte 1	N	P

Figure 10. SMBus Word Read command example

- The *Command Code* contains the low 8 bits of the address of the first RMI register to be written. The Page Select register supplies the high 8 bits of the address. The result is a 16-bit RMI address 'N'.
- *Data Byte 0* is the contents of the register at the RMI address 'N'.
- *Data Byte 1* is the contents of the register at the RMI address 'N'+1.
- All SMBus Read operations terminate with a NAK bit followed by a STOP bit.

3.2.3. Multi-register block read/write operations

To improve bus utilization, Synaptics RMI-on-SMBus devices permit special multi-register read and write operations as an extension to the SMBus specification. If a host performs a standard Byte Read operation but simply keeps reading more bytes, the device will continue to send registers from consecutively incrementing RMI addresses until the host finally NAKs the transfer.

Figure 11 is an example of a multi-register Read operation, in which the host reads four consecutive RMI registers starting from the specified register address 'N'.

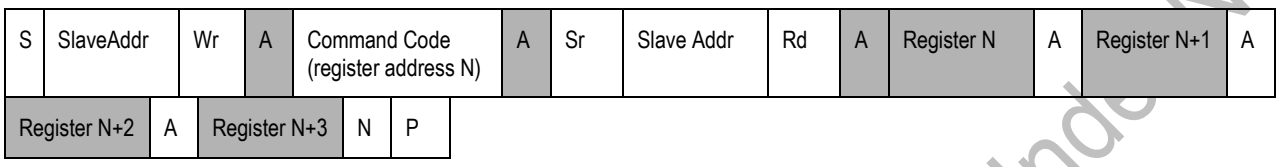

Figure 11. Multi-register Read operation example

In similar fashion, if the host performs a standard Byte Write operation but simply keeps writing more bytes, the RMI-on-SMBus device will write the extra bytes to subsequent register addresses.

Figure 12 is an example of a multi-register Write operation, in which the host writes three consecutive RMI registers starting with register address N.

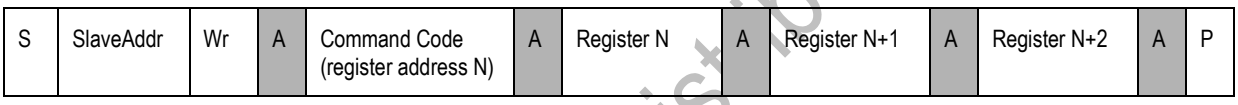

Figure 12. Multi-register Write operation example

3.2.4. Repeated starts

For the Read Byte and Read Word transfer protocols, the SMBus specification requires that a Repeated Start event must be used to separate the writing of the Command Code byte from the reading of the data byte(s). The Repeated Start ensures correct operation in host systems that support multiple bus masters. For host systems that either do not require multi-master support or cannot generate Repeated Start events, Synaptics RMI-on-SMBus devices permit a host to replace a Repeated Start with a Stop event followed by a Start event.

3.2.5. SMBus compliance

The SMBus Specification Version 2.0 describes a wide variety of features. Not all of these features are required to be supported for a particular device to be considered to be SMBus-compliant. Full information on the SMBus can be found in the document titled *System Management Bus (SMBus) Specification*, Version 2.0 of August 3, 2000. This document can be found at http://www.smbus.org/.

The SMBus is described in terms of *layers*. Synaptics SMBus compliance issues are dealt with on a layer-by-layer basis.

3.2.5.1. Layer 1: physical layer

In general, the SMBus physical layer looks like a standard I²C physical layer interface. To solve certain problems inherent in a typical I²C interface, SMBus imposes some additional restrictions on the I²C interface that it uses as its physical layer. These restrictions typically have to do with the timing and length of the transfers that are permitted.

The important restrictions imposed by the SMBus specification on a Synaptics RMI slave device are:

- 10 KHz minimum bus operating frequency,
- 25 mSec (min), 35 mSec (max) clock-low timeout period, and
- 500 mSec (max) Time in which a device must be operational after power-on reset.

Note: Synaptics RMI-on-SMBus devices do not support the SMBus SMBALERT# signal. This means that Synaptics SMBus devices will not respond to the SMBus Alert Response Address. Synaptics devices support a general purpose Attention signal (ATTN) that can either be used as an interrupt input to a host processor or as an input that the host can poll. As an order-time option, the ATTN signal can be configured as being either active high or active low.

3.2.5.2. Layer 2: data link layer

Synaptics RMI-on-SMBus devices implement the Data Link layer as described in the *SMBus Specification*. Section 4.3.3 of the *SMBus Specification* Version 2.0 defines that SMBus devices must implement "clock-low extending" (also known as I²C "clock-stretching"). In particular, the *SMBus Specification* V2.0 defines that *all* devices on a SMBus (both masters and slaves) must be able to tolerate both periodic and random clock stretching.

Synaptics RMI-on-SMBus slave devices can tolerate both random and periodic clock-stretching imposed by any other device sharing the SMBus.

Synaptics SMBus slave devices stretch the clock for short periods of time in random fashion, but they never stretch the clock long enough to violate the 10 KHz (min) bus transfer frequency.

3.2.5.3. Layer 3: SMBus network layer

The SMBus Specification defines eleven different command protocols that can be used to transfer data. The specification states that a slave device does not need to support all eleven protocols in order to be SMBus compliant.

Synaptics RMI-on-SMBus devices support the following four SMBus transfer protocols:

- Byte Read, Byte Write
- Word Read, Word Write

For increased transfer efficiency, Synaptics RMI-on-SMBus devices extend the SMBus protocol by supporting special multi-register reads and writes (see section 3.2.3).

Note: Synaptics SMBus devices do not support the ARP functionality to assign bus addresses. Synaptics SMBus devices implement a fixed, 7-bit I²C addressing mechanism. The 7-bit I²C slave address for a given Synaptics SMBus device is fixed at the time that the product is ordered. It is the implementer's responsibility to choose a SMBus address that will not conflict with other SMBus devices in their system. If desired, RMI-on-SMBus modules can be ordered with an address-strapping option. This allows a host system to select a module's I²C address among two or more preconfigured I²C addresses by strapping module IO pins.

Synaptics devices do not support the SMBus *Packet Error Check* (*PEC*) byte. Hosts should not expect Synaptics devices to generate a *PEC* byte during read operations. Hosts should not send *PEC* bytes to a Synaptics device during write operations.

3.3. SPI physical interface

This section describes the RMI-on-SPI physical layer.

3.3.1. SPI signals

RMI on SPI uses the industry-standard four-wire SPI interface. The SPI signals include:

- SSB, a device-select signal driven by the host. In some SPI systems, this signal is known as Slave Select (SS) or Chip Select (CS). SSB is an active-low signal that goes low when an RMI transaction is in progress.
- SCK, a clock signal driven by the host. Several clocking conventions are supported, as described in section 3.3.2.
- MOSI (master out / slave in), a data signal driven by the host.
- MISO (master in / slave out), a data signal driven by the RMI device. The device drives MISO only when SSB is low; when SSB is high, the device floats its MISO pin. This allows multiple RMI devices to be connected with SCK, MOSI, and MISO all tied in parallel, using separate SSB wires to address the various devices.
- ATTN, an *optional* attention signal driven by the RMI device. This pin is not present on devices that use the SRQ mechanism to signal attention (see section 3.3.4).
- RESET, an *optional* reset signal driven by the host. If a device provides a RESET pin, the pin is an active-low input with a pull-up resistor on the RMI device. This allows RESET to be left unconnected when not needed.

3.3.2. SPI clocking

"SPI" is actually a loosely defined family of standard interfaces. The clock polarity and clock phase (often denoted CPOL and CPHA) vary from one SPI system to another.

CPOL defines the idle level of SCK between transactions. If CPOL=0, SCK is low between transactions; if CPOL=1, SCK is high between transactions.

CPHA defines on which SCK edge the MOSI data and MISO data are sampled by their respective receivers. If CPHA=0, data is sampled when SCK leaves its idle level; if CHPA=1, data is sampled when SCK returns to its idle level.

The usual configuration for RMI devices is CPOL=1 and CPHA=1: The device samples MOSI, and expects the host to sample MISO, on the rising edge of SCK.

RMI devices can also be supplied with CPOL=0 and CPHA=1: The device samples MOSI, and expects the host to sample MISO, on the falling edge of SCK.

Configurations with CPHA=0 are not supported.

RMI always transmits each byte most-significant-bit first, following the convention of most chips' SPI interfaces. RMI always changes both MISO and MOSI on the same clock edge, and it always samples both MISO and MOSI on the same (opposite) clock edge.

3.3.3. SPI transaction format

The host (the SPI master) drives the SSB pin high between transactions and low during a transaction (see Figure 13and Figure 14). While SSB is high, the RMI device (the SPI slave) floats its MISO pin and ignores the MOSI and SCK pins; this allows multiple devices to share the MISO, MOSI, and SCK pins provided that each device receives a separate SSB signal. While SSB is low, the device drives the MISO pin, and it samples MOSI and changes MISO in response to the clock waveform on SCK. SSB edges delimit a transaction. The host is not allowed to tie SSB low permanently. It must fall to begin a transaction and rise to complete the transaction.

During the first two bytes (16 SCK pulses) after the fall of SSB, the host transmits an *address word* on MOSI, most significant byte first. In the address word, bit 15 is '1' for a read transaction and '0' for a write transaction. Bits 14:0 hold the register address *R*. During the first two bytes of the address word, the device transmits undefined data on MISO.

The SPI physical layer is only capable of transferring a 15-bit address. If an RMI device is constructed that requires access to the full 16-bit address space, the addresses above $7FFF will have to be accessed via setting the Page Select register (see section 2.3.2.1).

For a read transaction, in subsequent bytes (groups of 8 SCK pulses), the device transmits on MISO the contents of consecutive registers starting from the addressed register, and MOSI is ignored. It is permissible for SSB to be driven high between the second and third bytes of a read transaction (between the "address write" and "data read" portions of the transaction), but it is recommended to hold SSB low for the entire transaction if possible.

For a write transaction, in subsequent bytes, the host transmits on MOSI the write data for consecutive registers starting from the addressed register, and the device transmits undefined data on MISO. During a write transaction that writes several consecutive registers, the writing action to each register occurs as the transfer of the data byte for the register completes (except for a very few multi-byte quantities that are written only when the final byte is written, as described in section 2.1).

Note: This addressing mechanism is different from that of RMI-on-SMBus, but it is more consistent with the types of mechanisms most often used on SPI devices.

The transaction ends when the host raises SSB. If the host raises SSB during or after either byte of the address word, no transaction occurs. If the host raises SSB during a write transaction when only a fraction of the 8 bits of a data byte have been transmitted, the register corresponding to that data byte is not written (but any registers written earlier in the transaction will already be committed).

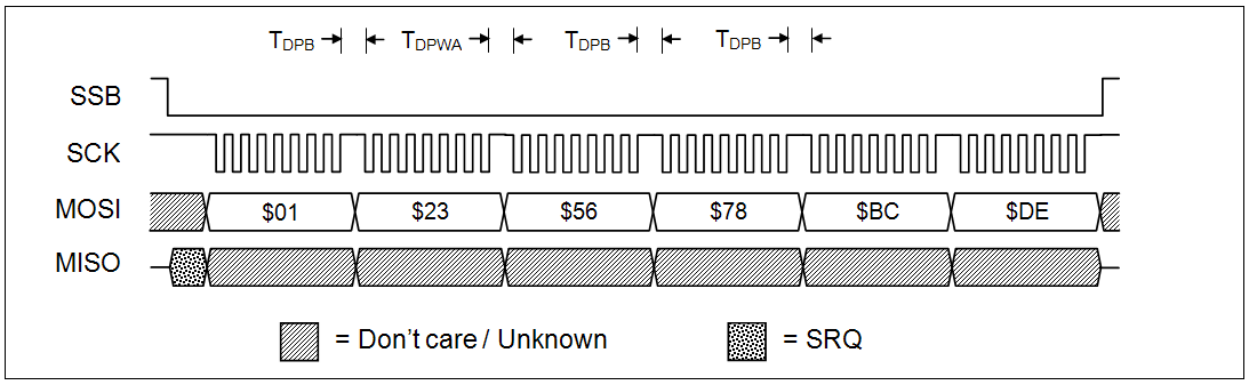

Figure 13. RMI-on-SPI write transaction
(assuming CPOL = 1 and CPHA = 1)

Note: The Host writes $56, $78, $BC, and $DE to registers $0123–$0126.

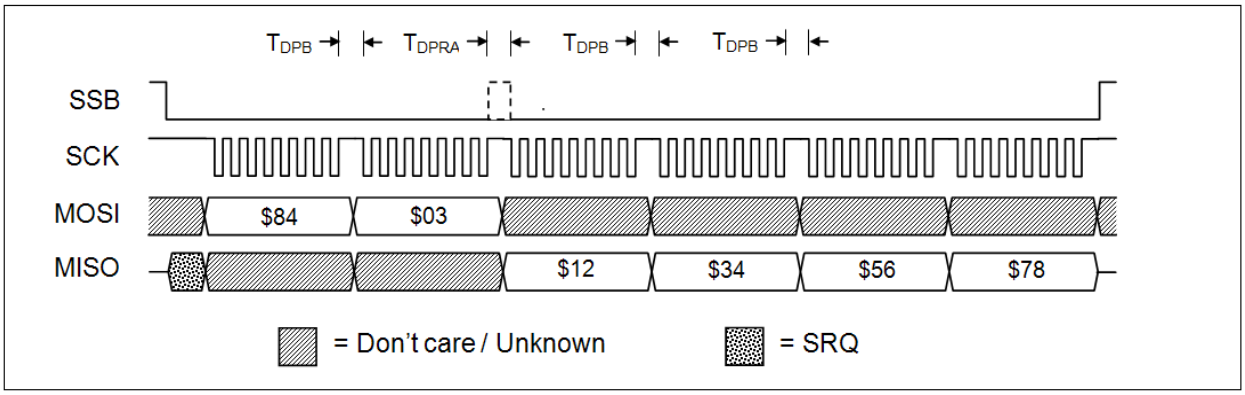

Figure 14. RMI-on-SPI read transaction
(assuming CPOL = 1 and CPHA = 1)

Notes:

- The Host reads $12, $34, $56, and $78 from registers $0403–$0406.
- It is recommended that SSB remain low for the entire transaction, but it is permissible for SSB to drive high between the "address write" and "data read" portions of the transaction.
- See the product datasheet (for example, the *ClearPad 3000 Platform Datasheet*) for the minimum duration for T_{DPWA}, T_{DPRA}, and T_{DPB}.

3.3.4. SPI attention mechanism

Synaptics can offer RMI devices using either of two additional attention mechanisms. One mechanism uses a non-standard extension of the SPI interface called a "service request" (SRQ) bit. The other uses a separate ATTN pin to hold the attention signal.

Note: The option for a separate attention pin accommodates hosts that cannot handle interrupts and MISO signals on the same host pin. It also may help in case RMI device interfaces must be implemented in non-Synaptics chips that cannot generate an SRQ-like bit on MISO.

As shown in Figure 13 and Figure 14 above, in the SRQ option, the RMI device drives the attention signal onto MISO as soon as SSB falls.

The SRQ attention signal is "live" on MISO in the interval between the fall of SSB and the first SCK edge: The host may lower SSB, leave SCK at its idle level, and watch MISO using an interrupt to wait for an interrupt request report from the device. After the first SCK edge, it is undefined whether MISO continues to follow the "live" attention state, or freezes at the state of the attention signal at the time of the SCK edge.

Note: The SRQ signal may be "live," but its behavior is relatively simple: The only change to MISO that can possibly occur during the SRQ period is one transition from the inactive to the active attention level, because attention, once asserted, can be deasserted only by a host transaction that reads the data registers or writes the Interrupt Enable bits. For RMI devices implemented using Synaptics' SPI-compatible chips, MISO will freeze after the first SCK edge.

Synaptics is also able to supply RMI devices that transmit the attention signal on a fifth ATTN pin. In this option, ATTN can be ordered either as an active-high push-pull output pin, or as an active-low open-drain pin for which the host must supply an external pull-up resistor. The latter option allows multiple RMI devices' ATTN pins to be merged in a wired-OR configuration. Because MISO is a fully driven push-pull output, the ATTN attention mechanism is better suited than SRQ to multi-device RMI systems.

4. Function $01: RMI device control

Function $01 implements a set of registers suitable as the foundation for controlling a large family of RMI products. Every RMI product requires a device control function to be present.

4.1. Function $01: query registers

Table 8. Function $01 query registers

Name	7	6	5	4	3	2	1	0
F01_RMI_Query0	Manufacturer ID							
F01_RMI_Query1	Has Product Properties2	Has Adjustable Doze Holdoff	Has Adjustable Doze	Has Charger Input	Has SensorID	—	Non Compliant	Custom Map
F01_RMI_Query2	CustomerSpecificProductFamily							
F01_RMI_Query3	ProductSpecificFirmwareRevision							
F01_RMI_Query4	MonthLow[2:0]			Year				
F01_RMI_Query5	CP2	CP1	Day					MonthHigh [bit 3]
F01_RMI_Query6	WaferLotID0LSB							
F01_RMI_Query7	WaferLotID0MSB							
F01_RMI_Query8	WaferLotID1LSB							
F01_RMI_Query9	WaferLotID1MSB							
F01_RMI_Query10	WaferLotID2LSB							
F01_RMI_Query11	—	ProductID, character1						
F01_RMI_Query12	—	ProductID, character2						
F01_RMI_Query13	—	ProductID, character3						
.	—	.						
.	—	.						
.	—	.						
F01_RMI_Query18	—	ProductID, character8						
F01_RMI_Query19	—	ProductID, character9						
F01_RMI_Query20	—	ProductID, character10						
F01_RMI_Query21	—							
F01_RMI_Query22	SensorID							
.	—	.						
.	—	.						
F01_RMI_Query42	—					HasGuest	Has MultiPhy	HasDS4Queries
F01_RMI_Query43.0	—				Length			
F01_RMI_Query43.1	—				HasMaskRevQuery	Has Reset Query	Has Packrat Query	Has Package IDQuery
F01_RMI_Query43.2	—					Has ATTN Control	HasSPI Control	HasI²C Control
F01_RMI_Query44	ResetPinNumber				—	PullUp Enabled	Reset Polarity	Reset Enabled

4.1.1. F01_RMI_Query0: manufacturer ID query

This register reports the identity of the manufacturer of the RMI device. Synaptics RMI devices report a Manufacturer ID of $01.

4.1.2. F01_RMI_Query1: product properties query

This byte contains bits that describe whether the RMI product has various optional properties.

Each property bit is '1' if the product has the associated property or '0' if the product does not have the associated property. Reserved property bits report as '0', but they may report as '1' in devices that comply with a future version of RMI.

CustomMap (F01_RMI_Query1, bit 0)

When this bit reports as '1', at least one custom, non RMI-compatible register exists in the register address map for this device. RMI defines no other information about the custom register implementation.

NonCompliant (F01_RMI_Query1, bit 1)

When this bit reports as '1', the device implements a register map that is not compliant with the RMI specification.

Reserved (F01_RMI_Query1, bit 2)

Reserved.

HasSensorID (F01_RMI_Query1, bit 3)

When this bit reports as'1', the *SensorID* query register (F01_RMI_Query22) exists.

HasChargerInput (F01_RMI_Query1, bit 4)

When this bit reports as '1', the *ChargerConnected* bit (F01_RMI_Ctrl0, bit 5) exists.

HasAdjustableDoze (F01_RMI_Query1, bit 5)

When this bit reports as '1', registers F01_RMI_Ctrl2 and F01_RMI_Ctrl3 exist.

HasAdjustableDozeHoldoff (F01_RMI_Query1, bit 6)

When this bit reports as '1', the *ChargerConnected* bit (F01_RMI_Ctrl0, bit 5) exists .

HasProductProperties (F01_RMI_Query1, bit 7)

F01_RMI_Query1 bit 7 is used to indicate the presence of F01_RMI_Query42, *ProductProperties2*.

The *CustomMap* and *NonCompliant* bits can be interpreted in conjunction with each other.

For example, if *CustomMap* is set, and *NonCompliant* is clear, the device implements both a complete RMI-compliant register set in addition to one or more custom registers.

If both *CustomMap* and *NonCompliant* are set, a device does not implement a fully RMI compliant register set, and it also implements one or more custom registers.

If *CustomMap* is clear, and *NonCompliant* is set, the device implements a non-compliant RMI register map that does not contain any custom registers. This might happen if a device were to implement a subset of RMI.

4.1.3. F01_RMI_Query2: customer-specific product family

Standard Synaptics RMI products define the contents of this register in their product-specific documentation. For custom products, the customer can define the meaning and content of these registers when the product is ordered.

CustomerSpecificProductFamily (F01_RMI_Query2)

In bootloader mode, this field will report the Family register as defined by the customer during the lockdown process.

In UI mode, this field is reserved.

4.1.4. F01_RMI_Query3: product-specific firmware revision

Standard Synaptics RMI products define the contents of this register in their product-specific documentation. For custom products, the customer can define the meaning and content of these registers when the product is ordered.

ProductSpecificFirmwareRevision (F01_RMI_Query3)

In bootloader mode, this field will report the Revision register as defined by the customer during the lockdown process.

In UI mode, this field will report the specific revision of the Design Studio 4 image itself. For example, if the ProductID register indicated that this was DS4 firmware, release 1.1, this field would be able to report the revision number of release 1.1.

4.1.5. F01_RMI_Query4 through F01_RMI_Query10: device serialization queries

These seven registers optionally record the individual identity of the device. Some RMI devices are not serialized at the factory; for unserialized devices, all of these registers report $00.

The various Device Serialization queries are defined as follows:

Date (F01_RMI_Query4; F01_RMI_Query5, bits 5:0)

The date-code registers record the date on which the module was manufactured. The actual interpretation is up to the manufacturer, but RMI diagnostic tools display this field assuming the bits are divided into year, month, and day fields.

The year code is a number from 1 to 31 to indicate years 2001–2031.

The month code is a number from 1 to 12 to indicate the months January through December. The day code is a number from 1 to 31 to indicate the day of the month. The day field can be 0 to indicate “unknown day of the month;” the day and month fields can both be 0 to indicate “unknown day of the year;” the entire 16 bits can be zero to indicate “unknown manufacturing date.”

WaferLotID (F01_RMI_Query6 through F01 _RMI_Query10)

The wafer-lot ID registers record the lot number of the wafer from which the module’s touch controller was produced.

4.1.6. F01_RMI_Query11 through F01_RMI_Query20: product ID queries

These 10 bytes (F01_RMI_Query11 through F01_RMI_Query20) report the identity of the particular RMI device or product as an array of 7-bit ASCII characters.

These registers form a string that identifies the product. Strings shorter than 10 characters always start at the first of these query registers (F01_RMI_Query11), and the unused characters at the end of the string will report as $00.

The contents of the Product ID string is product-specific, and the actual contents will be described in the product-specific documentation.

In bootloader mode, this field will report the ProductID registers as defined by the customer during the lockdown process.

In UI mode, this field will report the ProductID of the Design Studio 4 image itself, such as:

DS4 R1.1 indicates Design Studio 4 firmware, release 1.1

4.1.7. F01_RMI_Query21: reserved

This register is reserved.

4.1.8. F01_RMI_Query22: sensor ID

This register exists only if *HasSensorID* (F01_RMI_Query1, bit 3) is set to '1'.

The contents of this register identify the sensor attached to the device. The value is device-dependent and is described in the product-specific specification.

4.1.9. F01_RMI_Query23 through F01_RMI_Query41: reserved

These registers are reserved.

4.1.10. F01_RMI_Query42: product properties

HasDS4Queries (F01_RMI_Query42, bit 0)

When this bit reports as '1', it indicates the presence of the replicated register block F01_RMI_Query43.*. The length of the block will be defined by the first register in that replicated block, F01_RMI_Query43.0.

HasMultiPhy (F01_RMI_Query42, bit 1)

When this bit reports as '1', multiple physical communications interfaces are supported.

HasGuest (F01_RMI_Query42, bit 2)

When this bit reports as '1', a "guest" device is supported.

Reserved (F01_RMI_Query42, bits 7:3)

Reserved.

4.1.11. F01_RMI_Query43.*: DS4 query 0 through DS4 query 2

This block of query registers describes various aspects of device operation that are specific to Design Studio 4 (DS4) compatible features.

4.1.12. F01_RMI_Query43.0: DS4 queries 0

Length (F01_RMI_Query43.0, bits 3:0)

The length is defined as the length of the remaining Query43.* register block, not including this register. For example, a length that reports as 11 would indicate that in addition to F01_RMI_Query43.0, there would be 11 additional registers (F01_RMI_Query43.1 through F01_RMI_Query43.11).

Note: For forward compatibility, a host should not make any assumptions about any registers in this block that they are not aware of. For example, if a host knows the definition of F01_RMI_Query43.1 through F01_RMI_Query43.3, but the length reports as 5, then for the purposes of reconstructing the RMI address map, the host should allocate F01_RMI_Query43.4 and .5, but not make assumptions about what those registers do.

Reserved (F01_RMI_Query43.0, bits 7:4).

Reserved.

4.1.13. F01_RMI_Query43.1: DS4 queries 1

If F01_RMI_Query43.0 reports a length of at least 1 additional register, then this register exists.

HasPackageIDQuery (F01_RMI_Query43.1, bit 0)

If this bit reports as '1', the package ID query data will be accessible from inside the ProductID query registers.

If an RMI read operation is issued that starts accessing the ProductID query registers starting with the fourth-last register (F01_RMI_Query17), the device will return the package ID information in the following format:

F01_RMI_Query17:	Package ID low byte
F01_RMI_Query18:	Package ID high byte
F01_RMI_Query19:	Package ID revision low byte
F01_RMI_Query20:	Package ID revision high byte

For example: if a read starting at F01_RMI_Query17 returned the bytes [$81, $0C, $01, $00], the package ID would be interpreted as $0C81.0001, or "3201 revision 1".

HasPackratQuery (F01_RMI_Query43.1, bit 1)

If this bit reports as '1', the packrat query data will be accessible from inside the ProductID query registers.

The packrat ID is a 24-bit binary number that uniquely identifies the FW build for any firmware that is built by Synaptics. Any change that results in a new FW will also result in a new Packrat ID.

If an RMI read operation is issued that starts accessing the ProductID query registers starting with the third-last register (F01_RMI_Query18), the device will return the 24-bit packrat ID information in the following format:

F01_RMI_Query18:	Packrat ID low byte
F01_RMI_Query19:	Packrat ID mid byte
F01_RMI_Query20:	Packrat ID high byte

For example, if a read starting at F01_RMI_Query18 returned the bytes [$56, $34, and $12], the packrat ID would be $123456, or 1,193,046 in decimal.

Note: When in bootloader mode, the packrat will be reported for the bootloader itself. When in UI mode, the packrat will be reported for the UI image.

HasResetQuery (F01_RMI_Query43.1, bit 2)

If this field reports as '1', the register F01_RMI_Query44 will exist.

HasMaskRevQuery (F01_RMI_Query43.1, bit 3)

If this field reports as '1', the silicon mask revision number will be reported.

This query will be supported by the firmware only with other Design Studio 4 queries, such as PackageID and PackratID.

Reserved (F01_RMI_Query43.1, bits 7:4)

Reserved.

4.1.14. F01_RMI_Query43.2: DS4 queries 2

If F01_RMI_Query43.0 reports a total length of at least 2 additional registers, then this register exists.

HasI^2CControl (F01_RMI_Query43.2, bit 0)

If this field reports as '1', the register F01_RMI_Ctrl6 will exist.

HasSPIControl (F01_RMI_Query43.2, bit 1)

If this field reports as '1', the register F01_RMI_Ctrl7 will exist.

HasATTNControl (F01_RMI_Query43.2, bit 2)

If this field reports as '1', the register F01_RMI_Ctrl8 will exist.

Reserved (F01_RMI_Query43.2, bits 7:3)

Reserved.

4.1.15. F01_RMI_Query44: reset query

If the *HasResetQuery* field of F01_RMI_Query43.1 reports as '1', this register exists.

Control of the Reset functionality is performed via the F34 'lockdown' function. This query register exists to allow a host to *determine* the current state of the reset functionality, not *define* it.

Warning: The lockdown assignment mentioned for the fields below is permanent. Once a lockdown is performed, the functional choices defined by the lockdown process cannot be changed.

ResetEnabled (F01_RMI_Query44, bit 0)

If this bit reports as '1', the hardware reset pin functionality has been enabled for this device.

ResetPolarity (F01_RMI_Query44, bit 1)

If this bit reports as '0', it means that the reset state is active low. A '1' means that the reset state is active high.

Note: Not all Touch Controllers will support a configurable reset active state. For Touch Controllers that can support defining the reset active state, the actual assignment will be made via the lockdown process. Regardless of whether the reset active state is definable or not, this bit will report the actual active state for the given Touch Controller.

PullUpEnabled (F01_RMI_Query44, bit 2)

If this bit reports as '1', it indicates that a built-in weak pull up has been enabled on the Reset pin. A '0' means that no pull-up is present.

Reserved (F01_RMI_Query44, bit 3)

Reserved.

ResetPinNumber (F01_RMI_Query44, bits 7:4)

This field represents which GPIO pin number has been assigned the reset functionality.

Note: Not all Touch Controllers will support a configurable pin number for the reset pin. For Touch Controllers s that can support defining the specific pin, the pin assignment will be made via the lockdown process. Regardless of whether the reset pin assignment is definable or not, this bit field will report the actual GPIO pin number that is assigned to the reset pin function.

4.2. Function $01: control registers

Table 9. Function $01control registers

Name	7	6	5	4	3	2	1	0
F01_RMI_Ctrl0	Configured	ReportRate	Charger Connected	—	—	NoSleep	SleepMode	
F01_RMI_Ctrl1	Int Enable7	Int Enable6	Int Enable5	Int Enable4	Int Enable3	Int Enable2	IntEnable1	IntEnable0
F01_RMI_Ctrl2	DozeInterval							
F01_RMI_Ctrl3	WakeupThreshold							
F01_RMI_Ctrl5	DozeHoldoff							
F01_RMI_Ctrl6	—	I²CSlaveAddress						
F01_RMI_Ctrl7	—						SPIMode	
F01_RMI_Ctrl8	ATTNPinNumber				—	ATTN PullUp State	ATTN Active State	ATTN Enabled

4.2.1. F01_RMI_Ctrl0: device control register

The Device Control register contains bits that control the pace of processing in the device, and the conditions under which it can enter low-power states.

The bits of this register are defined as follows:

SleepMode (F01_RMI_Ctrl0, bits 1:0)

This field controls power management on the device. This field affects all functions of the device together. Below are descriptions of all possible sleep modes.

SleepMode = '00': Normal Operation

In this state, the device automatically and invisibly switches between full operation and a "doze" state in which finger sensing happens at a reduced rate (typically a few tens of milliseconds). The doze sensing rate is chosen so that almost any human finger action, such as rapid tapping, will still perform well.

This setting merely authorizes the device to doze when it is able. Products may be able to doze only under certain conditions, and may remain fully awake, for example, when a finger is present or when LED ramp animations are active.

Most RMI devices can be left in the Normal Operation setting at all times.

SleepMode = '01': Sensor Sleep

This state fully disables touch sensors and similar "analog" inputs on the device. All touch sensors report the "not touched" state regardless of any finger presence. Digital inputs, such as mechanical buttons attached to GPI pins, continue to operate even in the Sensor Sleep state.

Setting the Sleep Mode field to Sensor Sleep constitutes a request to the device to enter the sleeping state. The device may continue to operate in a higher-power state for one or two more report periods before it goes to sleep.

All RMI devices support the Sensor Sleep state at least to the degree of forcing the touch sensors to the "not touched" state.

In many devices, the Sensor Sleep state conserves additional power. In devices that do not support this deeper sleeping mode, the Sensor Sleep state is identical to the Normal Operation mode except for forcing the "not touched" state. Even then, this mode may help to conserve overall system power by interrupting the host less often.

SleepModes = '10, 11': Reserved

These Sleep Mode encodings are reserved. Specific products may define sleep modes corresponding to these encodings which are particular to that product. Products implementing a reserved encoding describe its particular operational effects in their product-specific documentation.

The reset state of the Sleep Mode bits default to the Normal Operation state.

NoSleep (F01_RMI_Ctrl0, bit 2)

When set to '1', this bit disables whatever sleep mode may be selected by the Sleep Mode field, and forces the device to run at full power without sleeping. The reset state is '0', meaning that the current Sleep Mode is enabled.

Reserved (F01_RMI_Ctrl0, bits 4:3)

These bits are reserved for definition in future versions of the RMI protocol.

ChargerConnected (F01_RMI_Ctrl0, bit 5)

When this bit is set to '1', the touch controller employs a noise-filtering algorithm designed for use with a connected battery charger. When this bit is set to '0', the touch controller employs a noise-filtering algorithm designed for use while a charger is not connected.

ReportRate (F01_RMI_Ctrl0, bit 6)

This field sets the report rate for the device. It applies in common to all functions on the device that have a natural report rate.

Many RMI functions divide time into *report periods* that occur at a *report rate,* roughly analogous to the "packet rate" of a mouse. If there are several such functions on an RMI device that work in terms of report periods, all the functions schedule their operation to the common report period of the device.

The encoding of the Report Rate field is largely device-dependent. The RMI standard does not require any particular encoding, although the value '0' corresponds to the device-preferred report rate. The reset value of the Report Rate field is '0'. If supported by a device, the value '1' will define a non-standard product-dependent report rate. RMI functions that work in terms of report periods assert interrupt requests, and therefore also the attention signal, at most once per report period.

Usually, report periods happen at a steady rate. Some conditions may cause the report period in progress to be canceled and a new report period started. This will not result in any loss of data, but it will add a visible irregularity to the steady rate of reports. For example, depending on the device implementation, writes to some control registers will cancel and restart the report period.

Some RMI functions do not schedule their activity in terms of report periods; these are known as *asynchronous* functions. Asynchronous RMI functions may assert an interrupt request on any schedule depending on the needs of the function. If an RMI device contains only asynchronous functions, its Report Rate field is unimplemented and resets to '0' (as described in section 2.2.1).

Configured (F01_RMI_Ctrl0, bit 7)

An RMI device may need to be configured (see section 2.2.2.1) after a device reset event.

Important: A host should write this bit to '1' **before** beginning the configuration process in order to clear the *Unconfigured* status bit in the Device Status register. Once the configuration is complete, the host should verify that the Unconfigured bit in the Device Status register is still reporting as '0' (configured). This method guarantees that a host would properly detect a situation where an RMI device might unexpectedly reset during the configuration process for any reason.

Writing the Configured bit to '0' has no effect. Reading the Configured bit always returns a '0'. The Unconfigured bit in the Device Status register always reports the actual configuration state of an RMI device.

4.2.2. F01_RMI_Ctrl1.*: interrupt enable register

The Interrupt Enable control register determines which interrupt sources are able to participate in the decision to assert the attention interrupt signal. Any function in an RMI device that defines interrupt sources is assigned bits in the Interrupt Enable register. If the RMI device defines more interrupt sources than will fit in a single Interrupt Enable register, a sufficient number of additional Interrupt Status registers are also defined at sequential addresses in the register map. Any unassigned Interrupt Enable bits in the register are treated as reserved bits. Every bit assigned to an Interrupt Enable register has a matching assignment made to the corresponding Interrupt Status register.

Each bit of this register controls whether the corresponding interrupt source asserts attention when it has an interrupt request. Bit *n* of this register is '1' if the interrupt request on data source *n* should assert attention, or '0' if the interrupt request on source *n* should not affect the attention signal. See section 2.7.2. Setting this field to all '0' bits effectively disables the attention interrupt signal altogether.

Depending on the implementation, the effect on the attention signal of a change to the Interrupt Enable bits may be either immediate or deferred until the next report period.

Unimplemented bits in the Interrupt Enable register always report as '0'.

RMI Function $01 is designed to typically assert an attention interrupt upon reset, as described in section 2.7.3. Under unusual circumstances, certain products may be configured so that the reset state of the bits in the Function $01 Interrupt Enable register does not generate an interrupt; consult the product-specific documentation for details in those cases.

4.2.3. F01_RMI_Ctrl2: doze interval register

DozeInterval (F01_RMI_Ctrl2)

The Doze Interval is the time period that the device sleeps between finger-activity checks. The doze interval is specified in units of 10 milliseconds. For example, a value of 3 indicates a nominal 30 milliseconds between wakeup checks.

The maximum value is 1500 ms (150 in the register). Anything above that will be treated as 1500 ms. This is because the firmware wakes up every 1500 ms to run relaxation and adjust the baseline for any environmental changes. Setting this register to 0 also has the same effect. Under these conditions, the transmitters are never driven in low power; they sleep for 1500 ms, then wake up and run relaxation (and look for fingers).

4.2.4. F01_RMI_Ctrl3: wakeup threshold register

WakeupThreshold (F01_RMI_Ctrl3)

The Wakeup Threshold is the amplitude of the disturbance to the background capacitance that will cause the device to wake from dozing. The threshold is specified in units of femtofarads.

Low values for the wakeup threshold may cause the device to wake up unnecessarily when in the presence of noise, thus wasting power. High values for the wakeup threshold may cause the device to miss light touches or taps.

4.2.5. F01_RMI_Ctrl4: does not exist

F01_RMI_Ctrl4 does not exist.

4.2.6. F01_RMI_Ctrl5: doze holdoff register

DozeHoldoff (F01_RMI_Ctrl5)

The device only dozes while no fingers are present on the sensor. The Doze Holdoff register contains the length of the delay between the last finger lift and the first doze cycle.

0.1-second units: 0x00 = 0 seconds, 0xFF = 25.5 seconds.

4.2.7. F01_RMI_Ctrl6: I²C control

If the *Has I²CControl* field of F01_RMI_Query43.2 reports as '1', then this register exists.

Note: At the device level, control of the I²C functionality occurs in two ways:

1) For the bootloader, the functional assignments occur via the F34 'lockdown' process
2) For the UI image, the functional assignments occur via this control register

Note: This field is can only be updated by reflashing the device configuration. Writes to the RAM-based register will be ignored.

Warning: Assignments made via F34 lockdown should match the assignments made to this register. If they do not match, the device will have a different behavior in bootloader mode than in UI mode. This is inconvenient at best, and should be avoided.

I²CSlaveAddress (F01_RMI_Ctrl6, bits 6:0)

This field defines the current I2C slave address that the device will respond to.

Reserved (F01_RMI_Ctrl6, bit 7)

Reserved.

4.2.8. F01_RMI_Ctrl7: SPI control

If the *HasSPICtrl* field of F01_RMI_Query43.2 reports as '1', then this register exists.

Note: At the device level, control of the SPI functionality occurs in two fashions:

1) For the bootloader, the functional assignments occur via the F34 'lockdown' process
2) For the UI image, the functional assignments occur via this control register

Note: This field is can only be updated by reflashing the device configuration. Writes to the RAM-based register will be ignored.

Warning: Assignments made via F34 lockdown should match the assignments made to this register. If they do not match, the device will have a different behavior in bootloader mode than in UI mode. This is inconvenient at best, and should be avoided.

SPIMode (F01_RMI_Ctrl7 bits 1:0)

This field defines the SPI mode used to communicate with the device:

- '00' corresponds to SPI Mode 0
- '01' corresponds to SPI Mode 1
- '10' corresponds to SPI Mode 2
- '11' corresponds to SPI Mode 3

Note: Not all SPI modes are supported by all ASICs. Consult the product data sheet for a particular ASIC to find out which SPI modes it supports.

Reserved (F01_RMI_Ctrl7, bits 7:2)

Reserved.

4.2.9. F01_RMI_Ctrl8: attention control

If the *HasAttnCtrl* field of F01_RMI_Query43.2 reports as '1', this register exists.

Note: At the device level, control of the ATTN functionality occurs in two fashions:

1) For the bootloader, the functional assignments occur via the F34 'lockdown' process
2) For the UI image, the functional assignments occur via this control register

Note: For the UI, this field is can only be updated by reflashing the device configuration. Writes to the RAM-based register will be ignored.

Warning: Assignments made via Function $34 lockdown should match the assignments made to this register. If they do not match, the device will have a different behavior in bootloader mode than in UI mode. This is inconvenient at best, and should be avoided.

ATTNEnabled (F01_RMI_Ctrl8 bit 0)

If this bit reports as '1', the ATTN pin functionality has been enabled for this device.

ATTNActiveState (F01_RMI_Ctrl8 bit 1)

If this bit reports as '0', it means that the ATTN state is active low. A '1' means that the ATTN state is active high.

ATTNPullUpState (F01_RMI_Ctrl8 bit 2)

If this bit reports as '1', it indicates that a built-in weak pull up has been enabled on the ATTN pin. A '0' means that no pull up is present.

Reserved (F01_RMI_Ctrl8, bit 3)

Reserved.

ATTNPinNumber (F01_RMI_Ctrl8 bits 7:4)

This field represents which GPIO pin number has been assigned the ATTN functionality.

4.3. Function $01: data registers

Table 10. Function $01 data registers

Name	7	6	5	4	3	2	1	0
F01_RMI_Data0	Unconfigured	Flash Prog	—	—	StatusCode			
F01_RMI_Data1.*	Int Request7	Int Request6	Int Request5	Int Request4	Int Request3	Int Request2	Int Request1	Int Request0

4.3.1. F01_RMI_Data0: device status register

The Device Status register reports events of interest to the host regarding the general status of the RMI device. Any change to the Device Status register causing it to become non-zero is indicated to the host by asserting the DevStat interrupt request bit in the Interrupt Status register.

The bits of this register are defined as follows:

StatusCode (F01_RMI_Data0, bits 3:0)

The Status Code field reports the most recent device status event. If several status conditions arise simultaneously, the actual status code reported is implementation-dependent.

Status conditions created by host actions, such as invalid configurations of control bits, may be reported instantly or they might not be reported until a few milliseconds after the offending register was written (for example, error conditions might not be checked until the next report period). Similarly, if new data is written to the control registers to correct the error condition, it may take a few milliseconds for the Status Code to clear.

RMI defines the following standard status codes:

Code $00: No Error.

Code $01: Reset occurred.

This code is reported if no other status event has occurred since the last time the device was reset. This error code is cleared when the Configured bit in control register 0 is written to '1'.

Code $02: Invalid Configuration.

This error signals a problem with the general configuration of the device, not specific to any one function. Many RMI devices do not implement this error.

Code $03: Device Failure.

This error signals a hardware problem with the device, not specific to any one function. Many RMI devices do not implement this error. Other devices might, for example, signal this status code if the firmware fails a program memory self-check.

Products containing RMI function $34 define the following status codes:

Code $04: Configuration CRC Failure.

This error signals that the configuration of the device failed a program memory self-check and therefore normal operation of the device is not possible.

Code $05: Firmware CRC Failure.

This error signals that the firmware failed a program memory self-check and therefore normal operation of the device is not possible.

Code $06: CRC In Progress.

This error signals that the firmware is in the process of testing either the configuration area or the firmware area.

Codes $07–$0F: Reserved.

These status codes are reserved for definition by future versions of RMI.

Reserved (F01_RMI_Data0, bits 5:4)

These bits are reserved for definition in future versions of the RMI protocol.

FlashProg (F01_RMI_Data0, bit 6)

The *FlashProg* bit defines the current device operating mode. The *FlashProg* flag is set if the normal operation of the device is suspended because the device is in a Flash Programming enabled state.

This can be the result of the issuance of a Function $34 Flash Program Enable command, or if there has been an error with the startup of the device that prevents its normal operation. The *Status Code* field can be read to give the reason for this bit being set. When in this mode the normal operation of the device is not possible; see section 13 for more information about this mode.

A ‘0’ indicates the device is operating in User Interface (UI) mode. A ‘1’ indicates the device is operating in Flash Programming mode, also known as *bootloader mode*. Devices that do not contain Function $34 (Flash Programming) will always report ‘0’.

Unconfigured (F01_RMI_Data0, bit 7)

The Unconfigured flag reports as ‘1’ if the device loses its configuration for any reason. The Unconfigured flag can be cleared by writing the *Configured* bit of F01_RMI_Ctrl0 to a ‘1’.

4.3.2. F01_RMI_Data1.*: interrupt status register

The Interrupt Status register reports which of the set of possible interrupt sources are actively requesting interrupt service from the host. The RMI attention interrupt (see section 2.7.2) is asserted by the RMI device if any *Int Request* bit in the Interrupt Status register is ‘1’, and the corresponding *Int Enable* bit in the corresponding Interrupt Enable register is also ‘1’.

Note: A ‘.*’ indicates a variable-sized register block. Every product contains at least one Interrupt Status register. The number of Interrupt Status registers implemented by an RMI device can be calculated by counting the total number of Interrupt Sources in the device:

```
InterruptStatusRegisterCount = trunc((NumberOfInterruptSources + 7) / 8)
```

Any function in an RMI device that defines interrupt sources is assigned bits in the Interrupt Status register. If the RMI device defines more interrupt sources than will fit in a single Interrupt Enable register, enough additional Interrupt Status registers are also defined by Function $01 to hold the other *Int Enable* bits. If more than one Interrupt Status register is defined, the subsequent registers will be defined at sequential addresses after the first Interrupt Status register.

The assignment of bits in the Interrupt Status registers to the set of interrupt sources in an RMI device is always identical to the assignment of the bits in the Interrupt Enable registers. This means that the number of Interrupt Status registers is always identical to the number of Interrupt Enable registers.

The act of reading an Interrupt Status register clears all the *Int Request* bits within it. The reset state of the implemented bits in the Interrupt Status register is product-specific.

Because every RMI device contains a Device Control function, and the Device Control function always defines one interrupt source (the DevStatus interrupt source), there will always be at least one Interrupt Status register in every RMI device.

4.3.3. Function $01: interrupt source

The Data registers defined by with Function $01 are associated with a single interrupt source, called the *DevStatus* interrupt. The DevStatus interrupt request is asserted whenever the RMI device has experienced some unusual event that requires the host's immediate attention.

Any product that includes Function $01 allocates a DevStatus interrupt request bit in the Interrupt Status register (see section 4.3.2), and a DevStatus interrupt enable bit in the Interrupt Enable register (see section 4.2.2). The position of the allocated interrupt bit within those registers is product-specific; consult the product-specific documentation to determine their location.

The reset state of the DevStatus interrupt request bit and the DevStatus interrupt enable bit is product-specific. However, in a typical device, the default reset state for both of these bits is '1'. This means that after a reset, a DevStatus interrupt will be pending in the Interrupt Status register, and enabled in the Interrupt Enable register, thus asserting the host attention interrupt. This ensures that any reset event will assert the DevStatus interrupt to report a loss of configuration.

4.4. Function $01: command registers

Table 11. Function $01 command registers

Name	7	6	5	4	3	2	1	0
F01_RMI_Cmd0	—	—	—	—	—	—	—	Reset

4.4.1. F01_RMI_Cmd0: device command register

The Device Command register is used to issue special commands to an RMI device. For general guidelines on the use and operation of command registers, see section 2.2.4.

The bits of this register are defined as follows:

Reset (F01_RMI_Cmd0, bit 0)

Writing a '1' to this bit causes the device to reset exactly as if its RESET pin had been pulled low.

The device's host interface (SMBus or SPI pins) may not operate for a certain amount of time T_{RESC} (about 1 ms) after a reset command or a low level on the RESET pin. This delay is much shorter than the delay T_{POR} before the host interface begins operating after a power-on reset. The T_{RESC} delay begins at the end of the write transaction that writes to this register (for example, at the rise of SSB in the case of RMI on SPI).

Note: The device asserts attention as soon as it is capable of responding to an operation on its physical interface.

Reserved (F01_RMI_Cmd0, bits 7:1)

These command bits are reserved.

5. Function $1A: 0-D capacitive button sensors

Function $1A implements capacitive button (0-D) sensing.

5.1. Function $1A: query registers

Table 12. Function $1A query registers

Name	7	6	5	4	3	2	1	0
F1A_0D_Query0	—					MaxButtonCount		
F1A_0D_Query1	Has Filter Strength	Has Strongest Button Hysteresis	Has Release Threshold	Has PerButton Threshold	HasTxRx Mapping	Has MultiButton Select	Has Interrupt Enable	Has General Control

5.1.1. F1A_0D_Query0: maximum button count

This register is defined as follows:

MaxButtonCount (F1A_0D_Query0, bits 2:0)

MaxButtonCount specifies the maximum number of buttons supported by the device. The value of the register is one less than the actual number of buttons supported. Function $1A supports a maximum of 8 buttons.

Reserved (F1A_0D_Query0, bits 7:3)

Reserved.

5.1.2. F1A_0D_Query1: button features

The bits in this register are defined as follows:

HasGeneralControl (F1A_0D_Query1, bit 0)

If this bit reports as ‘1’, register F1A_0D_Ctrl0 exists.

HasInterruptEnable (F1A_0D_Query1, bit 1)

If this bit reports as ‘1’, replicated register F1A_0D_Ctrl1.* exists.

HasMultiButtonSelect (F1A_0D_Query1, bit 2)

If this bit reports as ‘1’, replicated register F1A_0D_Ctrl2.* exists.

HasTxRxMapping (F1A_0D_Query1, bit 3)

If this bit reports as ‘1’, replicated registers F1A_0D_Ctrl3.* and F1A_0D_Ctrl4.* exist.

HasPerButtonThreshold (F1A_0D_Query1, bit 4)

If this bit reports as ‘1’, replicated register F1A_0D_Ctrl5.* exists.

HasReleaseThreshold (F1A_0D_Query1, bit 5)

If this bit reports as ‘1’, register F1A_0D_Ctrl6 exists.

HasStrongestButtonHysteresis (F1A_0D_Query1, bit 6)

If this bit reports as ‘1’, register F1A_0D_Ctrl7 exists.

HasFilterStrength (F1A_0D_Query1, bit 7)

If this bit reports as ‘1’, register F1A_0D_Ctrl8 exists.

5.2. Function $1A: control registers

These registers control the operation of the capacitive buttons.

5.2.1. Calculating the number of control registers

Registers F1A_0D_Ctrl3 through F1A_0D_Ctrl5 are replicated registers. The number of these registers depends on *MaxButtonCount*:

F1A_0D_Ctrl3 Count = *MaxButtonCount* + 1

F1A_0D_Ctrl4 Count = *MaxButtonCount* + 1

F1A_0D_Ctrl5 Count = *MaxButtonCount* + 1

For example, when *MaxButtonCount* = 7 (as shown in Table 13):

F1A_0D_Ctrl3 Count = 8

F1A_0D_Ctrl4 Count = 8

F1A_0D_Ctrl5 Count = 8

5.2.2. Function $1A: control registers example

Table 13 shows an example map of the Function $1A control registers for a device which supports a maximum of 8 buttons.

Table 13. Function $1A control register example, MaxButtonCount = *7 (supporting 8 buttons)*

Name	7	6	5	4	3	2	1	0
F1A_0D_Ctrl0	—	—	—	—	FilterMode		MultiButtonReporting	
F1A_0D_Ctrl1.0	IntEnBtn7	IntEnBtn6	IntEnBtn5	IntEnBtn4	IntEnBtn3	IntEnBtn2	IntEnBtn1	IntEnBtn0
F1A_0D_Ctrl2.0	MultiBtn7	MultiBtn6	MultiBtn5	MultiBtn4	MultiBtn3	MultiBtn2	MultiBtn1	MultiBtn0
F1A_0D_Ctrl3.0	TransmitterBtn0							
F1A_0D_Ctrl4.0	ReceiverBtn0							
F1A_0D_Ctrl3.1	TransmitterBtn1							
F1A_0D_Ctrl4.1	ReceiverBtn1							
F1A_0D_Ctrl3.2	TransmitterBtn2							
F1A_0D_Ctrl4.2	ReceiverBtn2							
F1A_0D_Ctrl3.3	TransmitteBtn3							
F1A_0D_Ctrl4.3	ReceiverBtn3							
F1A_0D_Ctrl3.4	TransmitteBtn4							
F1A_0D_Ctrl4.4	ReceiverBtn4							
F1A_0D_Ctrl3.5	TransmitteBtn5							
F1A_0D_Ctrl4.5	ReceiverBtn5							
F1A_0D_Ctrl3.6	TransmitteBtn6							
F1A_0D_Ctrl4.6	ReceiverBtn6							
F1A_0D_Ctrl3.7	TransmitteBtn7							
F1A_0D_Ctrl4.7	ReceiverBtn7							
F1A_0D_Ctrl5.0	ThresholdBtn0							
F1A_0D_Ctrl5.1	ThresholdBtn1							
F1A_0D_Ctrl5.2	ThresholdBtn2							

Name	7	6	5	4	3	2	1	0
F1A_0D_Ctrl5.3	ThresholdBtn3							
F1A_0D_Ctrl5.4	ThresholdBtn4							
F1A_0D_Ctrl5.5	ThresholdBtn5							
F1A_0D_Ctrl5.6	ThresholdBtn6							
F1A_0D_Ctrl5.7	ThresholdBtn7							
F1A_0D_Ctrl6	ButtonReleaseThreshold							
F1A_0D_Ctrl7	StrongestButtonHysteresis							
F1A_0D_Ctrl8	FilterStrength							

5.2.3. F1A_0D_Ctrl0: general control

The fields in this register are defined as follows:

MultiButtonReporting (F1A_0D_Ctrl0, bits 1:0)

This 2-bit field describes how buttons in the MultiButton group (see F1A_0D_Ctrl2) will be reported.

00: Unrestricted buttons

The user can touch the buttons in any combination, and every touched button's *Btn*n bit (in the F1A_0D_Data0.* registers) will be set to 1.

01: Reserved

10: Strongest button only

The user is expected to touch only one button in the MultiButton group at a time. If multiple buttons in the group are touched, only the *Btn*n bit corresponding to the button with the strongest finger signal will be set to 1.

11: First button only

The user is expected to touch only one button in the MultiButton group at a time. When a button is touched, its *Btn*n bit will be set to 1. Until that button is released, no other *Btn*n bits will be set even if additional buttons in the group are touched.

FilterMode (F1A_0D_Ctrl0, bits 3:2)

Capacitive buttons are filtered by the firmware to reduce the effects of electrical noise. The strength of the filter can be adjusted using F1A_0D_Ctrl08. Two filters are available:

00: Standard filter

The standard Design Studio 4 filter provides the best balance between button responsiveness and noise rejection. This filter should be used in most cases.

01: Debounce filter

If the environment has unusually high levels of electrical noise, use the debounce filter. The debounce filter provides a higher level of noise rejection, at the cost of increased time to report button touches and releases. This latency may reduce firmware capability to detect very fast button taps. The debounce filter should be used only when the standard filter is not suitable for the particular use case.

10, 11: Reserved

5.2.4. F1A_0D_Ctrl1.*: button interrupt enable

The bits in this register determine whether the ATTN interrupt is generated in response to button touches and releases.

The register contains one bit per button, defined as follows:

IntEnBtn0 (F1A_0D_Ctrl1.0, bit 0)

Controls the behavior of the ATTN interrupt in response to Button 0.

When this bit is set to 1, an ATTN interrupt is generated whenever the Btn0 bit changes state. When this bit is set to 0, an ATTN interrupt is not generated when the Btn0 bit changes state.

Additional *IntEnBtn*n bits control the behavior of the ATTN interrupt in response to their corresponding *Btn*n bits.

5.2.5. F1A_0D_Ctrl2.*: multi button group selection

The bits in this register specify which buttons are members of the MultiButton group. The register contains one bit per button, defined as follows:

MultiBtn0 (F1A_0D_Ctrl2.0, bit 0)

Specifies whether Button 0 is affected by the *MultiButtonReporting* setting. When this bit is set to 1, Button 0 is a member of the MultiButton group and is therefore affected by the *MultiButtonReporting* setting. When this bit is set to 0, Button 0 is not a member of the MultiButton group and so is not affected by the *MultiButtonReporting* setting.

Additional *MultiBtn*n bits specify whether their corresponding buttons are members of the MultiButton group.

5.2.6. F1A_0D_Ctrl3.* and F1A_0D_Ctrl4.*: electrode mapping

Each capacitive button is located at the intersection of a transmitter (Tx) electrode and a receiver (Rx) electrode. These registers specify the location of each button. There is one pair of registers per button, defined as follows:

TransmitterBtn0 (F1A_0D_Ctrl3.0)

Specifies the transmitter electrode which corresponds to Button 0.

ReceiverBtn0 (F1A_0D_Ctrl4.0)

Specifies the receiver electrode which corresponds to Button 0. A value of 0xFF in this field indicates that Button 0 is disabled.

To disable Button 0, set *TransmitterBtn*0 to 255 and set *ReceiverBtn0* to 255.

Additional pairs of *TransmitterBtn*n and *ReceiverBtn*n registers specify the locations of their corresponding buttons.

5.2.7. F1A_0D_Ctrl5.*: button touch threshold

These registers adjust button sensitivity. There is one register per button, defined as follows:

ThresholdBtn0 (F1A_0D_Ctrl5.0)

Button 0 Touch Threshold. The value is unsigned, from 0 to 255.

Additional *ThresholdBtn*n registers adjust the threshold of their corresponding buttons.

5.2.8. F1A_0D_Ctrl6: release threshold

This register is defined as follows:

ButtonReleaseThreshold (F1A_0D_Ctrl6)

This register sets the release threshold for all buttons. It is the percentage of the touch threshold at which a button is determined to not be pressed. The percentage is stored as a 0.8 unsigned fixed-point value.

5.2.9. F1A_0D_Ctrl7: strongest-button hysteresis

This register is defined as follows:

StrongestButtonHysteresis (F1A_0D_Ctrl6)

This register has an effect only when *MultiButtonReporting* is set to "Strongest Button Only" mode, and it only affects buttons in the MultiButton group. It specifies the amount by which the finger signal of a newly-touched button must exceed that of the currently-reported button in order for the new button to be reported instead of the current button. When this field is set to a low value, a new button will be reported if its finger signal is only slightly stronger than that of the currently-reported button. When this field is set to a high value, a new button will not be reported unless its finger signal is much stronger than that of the currently-reported button.

5.2.10. F1A_0D_Ctrl8: filter strength

This register is defined as follows:

FilterStrength (F1A_0D_Ctrl8)

This register affects the strength of the filter selected in *FilterMode (F1A_0D_Ctrl0 bits 3:2)*. Each filter mode may use this register differently, but increasing the value of this register always increases the amount of filtering performed on 0D buttons.

5.3. Function $1A: data registers

This register reports the state of the capacitive buttons. Table 14 shows an example map of the Function $1A data registers for a device which supports a maximum of 6 buttons.

Table 14. Function $1A data register example, MaxButtonCount *= 5 (supports 6 buttons)*

Name	7	6	5	4	3	2	1	0
F1A_0D_Data0	—	—	Btn5	Btn4	Btn3	Btn2	Btn1	Btn0

5.3.1. F1A_0D_Data0: button state

The bits in this register indicate the touched/released state of each button. The register contains one bit per button, defined as follows:

Btn0 (F1A_0D_Data0, bit 0)

When this bit reports as ‘1’, button 0 is touched. When this bit reports as ‘0’, button 0 is not touched.

Btn1 (F1A_0D_Data0, bit 1)

When this bit reports as ‘1’, button 1 is touched. When this bit reports as ‘0’, button 1 is not touched.

Btn2 (F1A_0D_Data0, bit 2)

When this bit reports as ‘1’, button 2 is touched. When this bit reports as ‘0’, button 2 is not touched.

Btn3 (F1A_0D_Data0, bit 3)

When this bit reports as ‘1’, button 3 is touched. When this bit reports as ‘0’, button 3 is not touched.

Btn4 (F1A_0D_Data0, bit 4)

When this bit reports as ‘1’, button 4 is touched. When this bit reports as ‘0’, button 4 is not touched.

Btn5 (F1A_0D_Data0, bit 5)

When this bit reports as ‘1’, button 5 is touched. When this bit reports as ‘0’, button 5 is not touched.

Reserved (F1A_0D_Data0, bits 7:6)

Reserved.

5.4. Function $1A: interrupt source

Function $1A has one interrupt source. Function $1A asserts an interrupt request whenever both of these conditions are met:

A *Btn*n bit changes from ‘0’ to ‘1’ or from ‘1’ to ‘0’.

The corresponding *IntEnBtn*n bit is set to ‘1’.

5.5. Function $1A: command registers

Function $1A implements no command registers.

6. Function $05: Image reporting

Function $05 provides Image Reporting functions that allow direct visibility into image sensing. Only some devices support this feature.

It is the intent for Function $05 Image Reporting to provide direct access to low-level capacitance data before processing and interpretation (to derive finger or palm status, for example) for testing and debugging purposes.

Controls for specific modes of operation are not contained within the Image Reporting function. For example:

- Controls for algorithms used in image data processing and interpretation.
- Scaling sensitivities.

The image data reported by Function $05 is a matrix of signed integer values, corresponding to each intersection in the grid of transmitter and receiver electrodes used for transcapacitive sensing. It does not map that to an X, Y coordinate space. It also reports button capacitances on an additional last row (or rows).

Operation in normal reporting modes after or during the use of Function $05 is not provided, and may require a reset command to return the sensor to normal operation.

6.1. Function $05: query registers

These query registers describe the available electrodes, modes, and number of registers available in this product.

Table 15. Function $05 query registers

Name	7	6	5	4	3	2	1	0
F05_AD_Query0	—		NumberOfReceiverElectrodes					
F05_AD_Query1	—		NumberOfTransmitterElectrodes					
F05_AD_Query2	—							
F05_AD_Query3	Has16Bit Delta	—						
F05_AD_Query4	SizeOfF05ImageWindow (bytes)							
F05_AD_Query5	—							

6.1.1. F05_AD_Query0: number of receiver electrodes

The bits of this register are defined as follows:

NumberOfReceiverElectrodes (F05_AD_Query0, bits 5:0)

This 6-bit field reports the number of sensor electrodes available in the design.

Reserved (F05_AD_Query0, bits 7:6)

Reserved.

6.1.2. F05_AD_Query1: number of transmitter electrodes

NumberOfTransmitterElectrodes (F05_AD_Query1, bits 5:0)

This 6-bit field reports the number of drive electrodes available in the design.

6.1.3. F05_AD_Query2: reserved

6.1.4. F05_AD_Query3: has 16-bit delta image

Reserved (F05_AD_Query3, bits 6:0)

Reserved.

Has16bitDeltaImage (F05_AD_Query3, bit 7)

This bit field reports the size of the delta image when *ReportMode* = 2.

‘0’: a 1-byte delta image will be reported.

‘1’: a 2-byte delta image will be reported.

6.1.5. F05_AD_Query4: size of F05 image window

SizeOfF05ImageWindow (F05_AD_ Query4)

This field indicates the number of register bytes mapped in the F$05 image data registers – *F05_AD_Data2*. The value is 2 * *NumberOfReceiverElectrodes*.

For example: A device configured with 20 receivers has an image line size of 20 words (40 bytes), so the value of this query register and the size of the *F05_AD_Data2* image window will be 40 (decimal).

6.1.6. F05_AD_Query5: reserved

This register is reserved.

6.2. Function $05: control registers

Table 16. Function $05 control registers

Name	7	6	5	4	3	2	1	0
F05_ADI_Ctrl0	—			NoAutoCal	—			
F05_AD_Ctrl1	—							
F05_AD_Ctrl2	—							
F05_AD_Ctrl3	—							
F05_AD_Ctrl4	—							
F05_AD_Ctrl5	—							

6.2.1. F05_AD_Ctrl0: no automatic calibration

Reserved (F05_AD_Query3, bits 3:0)

Reserved.

NoAutoCal (F05_AD_Ctrl0, bit 4)

When set, this bit prevents normal AutoCalibration. AutoCalibration allows the sensor to adjust its operation to variations in temperature and environmental conditions and should not normally be disabled. If it is disabled then the delta image may become inaccurate and a *ForceZero* (in *F05_AD_Cmd0*) or *Reset* (*F01_RMI_Cmd0*) command may be required. To return to normal operation this bit should be cleared or a *Reset* command issued.

Initial calibration will normally take place after a reset (power-on reset or *Reset* command) is completed. Both of these events are accompanied by a normal reset interrupt event, which can be used to monitor AutoCalibration events.

Reserved (F05_AD_Query3, bits 7:5)

Reserved.

6.2.2. F05_AD_Ctrl1 to F05_AD_Ctrl5: reserved

These registers are reserved.

6.3. Function $05: data registers

Table 17. Function $05 data registers

Name	7	6	5	4	3	2	1	0
F05_AD_Data0	—							
F05_AD_Data1	ReportMode		ReportIndex					
F05_AD_Data2.*	ReportData							

6.3.1. F05_AD_Data0: reserved

This register is reserved.

6.3.2. F05_AD_Data1: report index and report mode

The bits of this register are defined as follows:

ReportIndex (F05_AD_Data1, bits 5:0)

The Baseline Image Line (*ReportMode*=1) field specifies the line number (transmitter electrode number, 0 through *NumberOfTransmitterElectrodes*) of the image data.

ReportMode (F05_AD_Data1, bits 7:6)

Specifies the report mode for the data in the ReportData registers.

ReportMode = 0, no image data is available, normal operation with finger reporting in F$11 and F$19. After selecting this mode, issue a *ForceZero* (in F05_AD_Cmd0) or *Reset* (F01_RMI_Cmd0) command.

ReportMode = 1, baseline capacitance image data is available. When in this mode, the *ReportIndex* field selects the line number of the image visible within the F05_AD_Data2 registers. Each pixel of the image is two bytes, low order byte first (lower addressed register). The word will represent a signed 16-bit capacitance value in the range -32.768 to 32.767 pF. Each unit represents 1 fF (0.001 pF). Typically, the image data will be between $0000 and $0F00 (0.000 pF and 3.750 pF) at each pixel, and a $FFFF value will indicate invalid data.

ReportMode = 2, delta image data is available. A delta image is the capacitance image after subtracting the most recently captured baseline. When in this mode, the *ReportIndex* field selects the line number of the image visible within the F05_AD_Data2 registers. Each pixel of the image is typically one signed byte, representing a signed 8-bit number (-128 to 127) in twos-complement notation. When the *Has16bitDeltaImage* is set then a signed 16-bit value (-32768 to 32767) is reported as two bytes, low order byte first (lower addressed register). Typically, the delta image data will be near to 0, except where there is user input (fingers); on those pixels the data will be positive.

ReportMode = 3 is reserved.

6.3.3. F05_AD_Data1: report data

ReportData (F05_AD_Data2.)*

Contains one line of the image. Each 1- or 2-byte value contains the baseline or delta image pixel data for the corresponding receiver number and the transmitter selected by *ReportIndex*.

6.4. Function $05: interrupt sources

The *GetImage* bit will remain set until the interrupt has occurred and then will be cleared automatically when the interrupt is asserted. Further image capture will then be stalled until the *GetImage* command bit is set again.

No interrupts are generated from F$05 unless the *GetImage* bit has been set to request an image capture *and* the F$05 interrupt is enabled in the F01_RMI_Ctrl1 register(s).

6.5. Function $05: command registers

The command register bits automatically clear to zero when the requested operation has completed. They may be polled to test for command completion. It is necessary to poll *ForceZero* because no interrupt is generated when that command completes.

Table 18. Function $05 command registers

Name	7	6	5	4	3	2	1	0
F05_AD_Cmd0	—		ForceZero	—		GetImage	—	

6.5.1. F05_AD_Cmd0

The bits of this register are defined as follows:

Reserved (F05_AD_Cmd0, bits 1:0)

Reserved.

GetImage (F05_AD_Cmd0, bit 2)

Setting this bit requests an image to be captured for reading from F$05 data registers. The host should wait for the resulting interrupt before attempting to read the data. In order to ensure coherency of the image during the reading process, reporting of a new image is stalled until the *GetImage* bit is again set to '1'.

Reserved (F05_AD_Cmd0, bits 4:3)

Reserved.

ForceZero (F05_AD_Cmd0, bit 5)

Setting this bit requests a new baseline image to be taken causing the delta image to be forced to zero (for all pixels). The host should wait (poll) for the *ForceZero* bit to clear before attempting to next read from *ReportData.* The next baseline read from *ReportData* is from the frame following the forced zero of the delta image.

Reserved (F05_AD_Cmd0, bits 7:6)

Reserved.

6.6. Reading images with Function $05

This section describes the process for reading image data with Function $05. When reading image data, the following rules should be kept in mind:

- When image reporting is enabled, do not read any RMI registers outside of Function $05 that are part of a coherent group. If this is done, the currently captured image snapshot will be corrupted and should be discarded.
- Once an image has been captured no new image will be reported until *GetImage (F05_AD_Cmd0, bit 2)* is again set to '1'.
- If maintaining the frame rate is more important than the coherency of the report data, the next image capture can be started (with *GetImage*) before reading all the data for the current image, at the risk of overwriting data currently in the image.

To read an image, follow the instructions below:

1. Disable all other interrupts so that only the image reports can cause ATTN to be asserted. To do this, set the F$05 interrupt enable bit in F01_RMI_Ctrl1.*n* registers to '1', and set all others bits in those registers to '0'.

   Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), skip this step.

2. Enable image reporting by setting *ReportMode (F05_AD_Data1, bits 7:6)* to '01' or '10'.
3. Request an image capture by writing '1' to *GetImage (F05_AD_Cmd0, bit 2)*. When the resulting interrupt occurs its source can be determined by reading interrupt status in F01_RMI_Data1. This will clear the interrupt.
4. At this point the entire image is available and some or all of the profile data can be read by writing *ReportIndex (F05_AD_Data1, bits 5:0)* and reading from *ReportData (F05_AD_Data2.*)*. After each line is read, the *ReportIndex* field should be incremented and the next line read, until all desired lines have been read.
5. Request the next image report by writing '1' to *GetImage (F05_AD_Cmd0, bit 2)*.

To restore normal reporting mode, reset the device.

7. Function $07: Image Reporting

Function $07 provides Image Reporting functions that allow direct visibility into image sensing for larger sensors. Only some devices support this feature.

It is the intent for Function $07 Image Reporting to provide direct access to low-level capacitance data before processing and interpretation (for example, to derive finger or palm status) for testing and debugging purposes.

The image data reported by Function $07 is a matrix of signed integer values, corresponding to each intersection in the grid of transmitter and receiver electrodes.

Normal operation of functions other than Function $01 is not supported while Function $07 is being used. After using Function $07, the device must be reset in order to return it to normal operation.

7.1. Function $07: query registers

The query registers describe the available electrodes, modes, and number of registers available in this device.

Table 19. Function $07 query registers

Name	7	6	5	4	3	2	1	0
F07_AD_Query0	NumberOfReceiverElectrodes							
F07_AD_Query1	NumberOfTransmitterElectrodes							
F07_AD_Query2	—							
F07_AD_Query3	Has16-Bit DeltaImage	—						
F07_AD_Query4	SizeOfF07ImageWindow							
F07_AD_Query5	—							

7.1.1. F07_AD_Query0: number of receiver electrodes

The bits of this register are defined as follows:

NumberOfReceiverElectrodes (F07_AD_Query0)

This field reports the number of sensor electrodes available in the design.

7.1.2. F07_AD_Query1: number of transmitter electrodes

NumberOfTransmitterElectrodes (F07_AD_Query1)

This field reports the number of drive electrodes available in the design.

7.1.3. F07_AD_Query2: reserved

7.1.4. F07_AD_Query3: has 16-bit delta image

Reserved (F05_AD_Query3, bits 6:0)

Reserved.

Has16BitDeltaImage (F07_AD_Query3, bit 7)

This bit field reports the size of the delta image available, and therefore whether *ReportMode* = 2 or *ReportMode* = 3 should be used to read delta images.

'0': a 1-byte delta image will be reported (use *ReportMode* = 2)

'1': a 2-byte delta image will be reported (use *ReportMode* = 3)

7.1.5. F07_AD_Query4: size of F07 image window

SizeOfF07ImageWindow (F07_AD_ Query4)

This field indicates the number of register bytes mapped in the F$07 image data registers – *F07_AD_Data2*. The value is 2 * *NumberOfReceiverElectrodes.*

For example: A device configured with 20 receivers has an image line size of 20 words (40 bytes), so the value of this query register and the size of the *F07_AD_Data2* image window will be 40 (decimal).

7.1.6. F07_AD_Query5: reserved

This register is reserved.

7.2. Function $07: control registers

Table 20. Function $07 control registers

Name	7	6	5	4	3	2	1	0
F07_ADI_Ctrl0	—			NoAutoCal	—			
F07_AD_Ctrl1	—							
F07_AD_Ctrl2	—							
F07_AD_Ctrl3	—							
F07_AD_Ctrl4	—							
F07_AD_Ctrl5	—							

7.2.1. F07_AD_Ctrl0: no automatic calibration

Reserved (F07_AD_Ctrl0, bits 3:0)

Reserved.

NoAutoCal (F07_AD_Ctrl0, bit 4)

When set, this bit prevents normal AutoCalibration. AutoCalibration allows the sensor to adjust its operation to variations in temperature and environmental conditions and should not normally be disabled. If it is disabled then the delta image may become inaccurate and a *ForceZero* (in F07_AD_Cmd0) or Reset (F01_RMI_Cmd0) command may be required. To return to normal operation this bit should be cleared or a *Reset* command issued.

Initial calibration will normally take place after a reset (power-on reset or *Reset* command) is completed. Both of these events are accompanied by a normal reset interrupt event, which can be used to monitor AutoCalibration events.

Reserved (F07_AD_Ctrl0, bits 7:5)

Reserved.

7.2.2. F07_AD_Ctrl1 to F07_AD_Ctrl5: reserved

These registers are reserved.

7.3. Function $07: data registers

Table 21. Function $07 data registers

Name	7	6	5	4	3	2	1	0
F07_AD_Data0	—					ReportMode		
F07_AD_Data1	ReportIndex							
F07_AD_Data2.*	ReportData							
F07_AD_Data3	FIFOIndexLow							
F07_AD_Data4	FIFOIndexHigh							
F07_AD_Data5	FIFOData							

7.3.1. F07_AD_Data0: report mode

The bits of this register are defined as follows:

ReportMode (F07_AD_Data0, bits 2:0)

Specifies the report mode for the data in the ReportData and FIFOData registers.

ReportMode = 0, no image data is available, normal operation with finger reporting in F$11 and F$19. After selecting this mode, issue a *ForceZero* (in F07_AD_Cmd0) or *Reset* (F01_RMI_Cmd0) command.

ReportMode = 1, baseline capacitance image data is available. When in this mode, data can be read either line-by-line from the F07_AD_Data2 registers, or pixel-by-pixel by repeated reads from the F07_AD_Data5 register. Each pixel of the image is two bytes, low order byte first (lower addressed register). The word will represent a signed 16-bit capacitance value in the range -32.768 to 32.767 pF. Each unit represents 1 fF (0.001 pF). Typically, the image data will be between $0000 and $0F00 (that is, 0.000 pF and 3.750 pF) at each pixel, and a $FFFF value will indicate invalid data.

ReportMode = 2, delta image 8-bit data is available. A delta image is the capacitance image after subtracting the most recently captured baseline. When in this mode, data can be read either line-by-line from the F07_AD_Data2 registers, or pixel-by-pixel by repeated reads from the F07_AD_Data5 register. Each pixel of the image is one signed byte, representing a signed 8-bit number (-128 to 127) in twos-complement notation. This mode only returns valid data when *Has16BitDeltaImage* is ‘0’. Typically, the delta image data will be near to 0, except where there is a user input (fingers) and on those pixels the data will be positive.

ReportMode = 3, delta image 16-bit data is available. A delta image is the capacitance image after subtracting the most recently captured baseline. When in this mode, data can be read either line-by-line from the F07_AD_Data2 registers, or pixel-by-pixel by repeated reads from the F07_AD_Data5 register.

Each pixel of the image is a signed 16-bit value (-32768 to 32767) and reported as two bytes, low order byte first (lower addressed register). This mode only returns valid data when *Has16BitDeltaImage* is ‘1’. Typically, the delta image data will be near to 0, except where there is a user input (fingers) and on those pixels the data will be positive.

ReportMode = 4 through 7 are reserved.

Reserved (F07_AD_Data0, bits 7:3)

Reserved.

7.3.2. F07_AD_Data1: report index

The bits of this register are defined as follows:

ReportIndex (F07_AD_Data1)

This field specifies the line number (transmitter electrode number, 0 through *NumberOfTransmitterElectrodes-1*) of the image data to be read from *ReportData*.

7.3.3. F07_AD_Data2: report data

The bits of this register are defined as follows:

ReportData (F07_AD_Data2.)*

Contains one line of the image. Each 1- or 2-byte value contains the baseline or delta image pixel data for the corresponding receiver number and the transmitter selected by *ReportIndex*.

7.3.4. F07_AD_Data3 and F07_AD_Data4: FIFO index

FIFOIndexLow (F07_AD_Data3)

FIFOIndexHigh (F07_AD_Data4)

Contains the index number of the pixel data byte which can be read from *FIFOData*. These registers are set to zero when a new image is ready for reading, and auto-incremented on reads of *FIFOData*. They may also be written by a host directly in order to look up data for a specific pixel.

7.3.5. F07_AD_Data5: FIFO data

FIFOData (F07_AD_Data5)

This packet register contains data for one pixel of the image. As an alternative to writing *ReportIndex* and reading each line from *ReportData*, a host may just read *FIFOData* repeatedly. Once an image is available for reading, *FIFOData* contains the first pixel. A subsequent read returns data for the next pixel in the first line. After a line has been read, the next read returns data for the first pixel on the second line and so on.

For 16-bit data, each pixel requires two reads: the first to obtain low-byte data and the second to obtain high-byte data.

Important: This register must be read by specifying its exact address, so that its address will not be automatically incremented upon repeated reads. Repeated reads will automatically read the next element in the FIFO (packet register). For additional details, see section 2.5.3.

7.4. Function $07: interrupt sources

The Image Reporting data source can assert an interrupt request at the end of every report period (frame). Interrupts are enabled and reported in Function $01.

When an image report is requested by writing '1' to the *GetImage* bit in the F07_AD_Cmd0 register, an interrupt is generated once the image capture is complete and the data is available to be read. The *GetImage* bit will remain set until the interrupt has occurred and then will be cleared automatically when the interrupt is asserted. No additional images will be captured until the *GetImage* command bit is set again.

No interrupts are generated from F$07 unless the *GetImage* bit has been set to request an image capture *and* the F$07 interrupt is enabled in the F01_RMI_Ctrl1 register(s).

7.5. Function $07: command registers

The command register bits automatically clear to zero when the requested operation has completed. The register bits may be polled to test for command completion. It is necessary to poll *ForceZero* because no interrupt is generated when that command completes.

Table 22. Function $07 command registers

Name	7	6	5	4	3	2	1	0
F07_AD_Cmd0	—		Force Zero	—		GetImage	—	

7.5.1. F07_AD_Cmd0: get image and force zero

The bits of this register are defined as follows:

Reserved (F07_AD_Cmd0, bits 1:0)

Reserved.

GetImage (F07_AD_Cmd0, bit 2)

Writing this bit requests an image (baseline or delta) to be captured for reading from F$07 data registers. The host should wait for the resulting interrupt before attempting to read the data. In order to ensure coherency of the image during the reading process, another image will not be captured until the *GetImage* bit is again set to '1'. Only one image is captured each time *GetImage* is set.

Reserved (F07_AD_Cmd0, bits 4:3)

Reserved.

ForceZero (F07_AD_Cmd0, bit 5)

Setting this bit requests a new baseline image to be taken, forcing the delta image to zero (for all pixels). The host should wait (poll) for the *ForceZero* bit to clear before attempting to read from *ReportData* or *FIFOData*.

Reserved (F07_AD_Cmd0, bits 7:6)

Reserved.

7.6. Reading images with Function $07

This section describes the process for reading image data with Function $07. When reading image data, follow these rules:

- When image reporting is enabled, do not read any RMI registers outside of Function $07 that are part of a coherent group. If you do this, the currently captured image snapshot will be corrupted and should be discarded.
- After an image has been captured, no new image will be reported until *GetImage* (F07_AD_Cmd0, bit 2) is again set to '1'.
- If maintaining the frame rate is more important than the coherency of the report data, the next image capture can be started (by setting *GetImage*) before reading all the data for the current image, at the risk of overwriting data currently in the image.

To read an image, follow the instructions below:

1. Disable low-power/doze mode by writing the *NoSleep* bit in F01_RMI_Ctrl0.
2. Disable all other interrupts so that only the image reports can cause ATTN to be asserted. To do this, set the F$07 interrupt enable bit in the F01_RMI_Ctrl1.*n* registers to '1', and set all others bits in those registers to '0'.

   Note: If the device does not have an ATTN line, or if you prefer to ignore it and poll (not recommended), skip this step.
3. Disable *SlaveN* bits in F01_RMI_Ctrl2.n (leave the *Master* bit set).
4. Enable image reporting by writing *ReportMode* (F07_AD_Data1, bits 2:0*)* to 1, 2, or 3.
5. Request an image capture by writing '1' to *GetImage* (F07_AD_Cmd0, bit 2). When the resulting interrupt occurs, its source can be determined by reading the interrupt status from F01_RMI_Data1. This will clear the interrupt.
6. At this point the entire image is available to be read. To read the image using the *ReportData* registers:

   a. Write a '0' into the *ReportIndex* field to point the ReportData registers at the first line of the image.

   b. Read one line of image data from *ReportData* (F07_AD_Data2.*). After reading each line, increment the *ReportIndex* field. Repeat this until all lines have been read.

   Optionally, to read the image using the *FIFOData* register:

   - Perform a block read from the FIFOData register until all pixels have been read.
7. Request the next image report by writing '1' to *GetImage (*F07_AD_Cmd0, bit 2). Repeat this until the desired number of images have been read.
8. Reset the device to restore normal reporting mode.

8. Function $08: Built-in self-test

Function $08 implements controls for the Built-In Self-Test (BIST) functions for testing, characterization, and analysis of a system and its touch controller. There are two BIST functions, Function $08 and Function $09. Only one of these functions is applicable for a device; it is dependent upon the sensor chip. See the device Product Specification for information on which BIST function is used.

8.1. Typical BIST usage scenario

The BIST function is typically used to test devices after assembly. The basic functionality is not intended as a complete test or replacement for module testing. BIST can be run after the device has powered on and begun reporting after POR (PowerOnReset).

The BIST control limit registers default to the correct values for the test, but they can be changed by re-flashing the device with new Firmware.

The command RunBIST (F08_BIST_Cmd0, bit 0) executes TestNumber=0 by default, runs a complete BIST, and reports the first failing sub-test (for example, *Test1*).

In a typical case, running the default BIST command should result in a Passing ($00) result in the F08_BIST_Data1 register. In the case of a failing non-zero result, the output of the data register indicates the failed sub-test number. The failing sub-test result can be loaded into the Test Number Control register F08_BIST_Data0 to determine the cause of the fault. Running *RunBIST* (F08_BIST_Cmd0) again, with the *TestNumberControl* register properly loaded with the sub-test number, produces a result in the BIST data registers that indicates an electrode that failed the BIST sub-test.

8.2. Function $08: query registers

The various BIST queries provide the mechanism for configuring and reading the BIST controls (limits), commands (tests), and data (results) variables. The BIST limits may be configurable and set to defaults by Flash-backed registers or set in Flash on a per-product basis. Additional control limit registers are necessary for additional BIST tests.

Table 23. Function $08 query registers

Name	7	6	5	4	3	2	1	0
F08_BIST_Query0	LimitRegisterCount							
F08_BIST_Query1	HostTest En	—						

The locations of the control, data, and command registers for basic BIST tests are calculated from the F08_BIST_Query0 register. Any additional BIST tests and adjustments to the BIST register placements can be calculated based on F08_BIST_Query1.

8.2.1. F08_BIST_Query0: limit register count

The bits of this register are defined as follows:

Limit Register Count (F08_BIST_Query0)

Indicates the number of control registers used to set the limits for BIST testing capabilities.

8.2.2. F08_BIST_Query1: host test enable

Reserved (F08_BIST_Query1, bits6:0)

Reserved.

HostTestEn (F08_BIST_Query1, bit 7)

Indicates that BIST host-level testing is available.

8.3. Function $08: control registers

The control registers determine the limits for each of the implemented BIST tests. Depending on the configuration, a device can have one or two defined tests. The example below describes a Function $08 configuration with two tests.

Table 24. Function $08 Limit control registers

Name	7	6	5	4	3	2	1	0
F08_BIST_Ctrl0	Test1LimitLow							
F08_BIST_Ctrl1	Test1LimitHigh							
F08_BIST_Ctrl2	Test1LimitDiff							
F08_BIST_Ctrl3	Test2LimitLow							
F08_BIST_Ctrl4	Test2LimitHigh							
F08_BIST_Ctrl5	Test2LimitDiff							

8.3.1. F08_BIST_Ctrl0 through F08_BIST_Ctrl5

The bits of these registers are defined as follows:

Test1LimitLow (F08_BIST_Ctrl0)
Controls the acceptance Low BIST Limit for Test1.

Test1LimitHigh (F08_BIST_Ctrl1)
Controls the acceptance High BIST Limit for Test1.

Test1LimitDiff (F08_BIST_Ctrl2)
Controls the acceptance Difference BIST Limit for Test1.

Test2LimitLow (F08_BIST_Ctrl3)
Controls the acceptance Low BIST Limit for Test2.

Test2LimitHigh (F08_BIST_Ctrl4)
Controls the acceptance High BIST Limit for Test2.

Test2LimitDiff (F08_BIST_Ctrl5)
Controls the acceptance Difference BIST Limit for Test2.

8.4. Function $08: data registers

Function $08's data result registers are allocated as necessary for the tests defined by design capability. The data registers are filled upon the completion of a command. Reported data values may change, depending on the configuration of the control limit and the command executed. Future unimplemented extended BIST capabilities may add additional data registers.

The *TestNumberControl* register determines which BIST tests are run. If '0', all implemented tests are run.

Table 25. Function $08 data registers

Name	7	6	5	4	3	2	1	0
F08_BIST_Data0	TestNumberControl							
F08_BIST_Data1	OverallBISTResult							
F08_BIST_Data2	TestResult							

8.4.1. F08_BIST_Data0: test number control

TestNumberControl (F08_BIST_Data0)

Controls the test (or tests) run by the *BIST* command. Test capabilities are defined and documented on a design basis. Additional tests may be available, but not all devices will have all tests or necessarily sequential test numbers. '3' through '255' are reserved test numbers. By default:

- If '0', the complete BIST test executes each of the one or more BIST sub-tests in order until there is a failure. Reports the first failing test in the Test Result data register.
- If '1', then the *RunBIST* (F08_BIST_Cmd0) command executes the BIST sub-test 1 using the limits set by the Test1 limit control registers. Reports the failing electrode and the failing control limit in the Test Result data register.
- If '2', then the *RunBIST* (F08_BIST_Cmd0) command executes the BIST sub-test 2 using the limits set by the Test2 limit control registers. Reports the failing electrode and the failing control limit in the Test Result data register.

8.4.2. F08_BIST_Data1: overall BIST result

The Overall BIST Result register is '0' for success, or indicates the number of the first failing test.

Overall BIST Result (F08_BIST_Data1)

Reports the overall result of the BIST command executed:

- If '0', all requested BIST tests have passed.
- If '1', Test 1 has failed and F08_BIST_Data2 will indicate the failing electrode number.
- If '2', Test 2 has failed and F08_BIST_Data2 will indicate the failing electrode number.

'3' through '255' are reserved.

8.4.3. F08_BIST_Data2: test result

If the Overall BIST Result is non-zero, then the *TestResult* register contains test specific error information or the default complete test.

For example, if Test1 failed, the *TestResult* register would indicate which limit test failed and an electrode number associated with the failure, as shown below.

Table 26. Function $08 Test Result (Test1)

Name	7	6	5	4	3	2	1	0
F08_BIST_Data2	Limit Failure Code		—	Failing Electrode Number				

Failing Electrode Number (F08_BIST_Data2, bits 4:0)

Reports an electrode that fails the test. The value of this register is meaningful only if F08_BIST_Data1 is non-zero.

Limit Failure Code (F08_BIST_Data2, bits 7:6)

Reports which limit test fails the test:

00 = No failure.
01 = Low Limit failed.
10 = High Limit failed.
11 = Difference Limit failed.

8.5. Function $08: interrupt source

The BIST data source asserts an interrupt request at the end of any report period in which a command register bit has been cleared.

8.6. Function $08: command registers

The BIST command register provides the mechanism for executing the built-in self tests. Test times vary; test completion is indicated by the automatic clearing of the *RunBIST* bit and the assertion of the BIST interrupt request.

Table 27. Function $08 command register

Name	7	6	5	4	3	2	1	0
F08_BIST_Cmd0	—							RunBIST

8.6.1. F08_BIST_Cmd0: run BIST

RunBIST (F08_BIST_Cmd0, bit 0)

Runs the BIST using the test number and limits set in the data and control registers.

Reserved (F08_BIST_Cmd0, bits 7:1)

Reserved.

9. Function $09: BIST

Function $09 implements controls for the Built-In Self-Test (BIST) functions for testing, characterization, and analysis of a system and its touch controller. There are two BIST functions, Function $08 and Function $09. Only one of these functions is applicable for a device; it is dependent upon the sensor chip. See the device Product Specification for information on which BIST function is used.

9.1. Typical BIST usage scenario

The BIST function is typically used to test devices after assembly. The basic functionality is not intended as a complete test or replacement for module testing. BIST can be run after the device has powered on and begun reporting after POR (PowerOnReset).

The BIST control limit registers default to the correct values for the test, but they can be changed by re-flashing the device with new Firmware.

The command *RunBIST* (F09_BIST_Cmd0, bit 0) executes TestNumber=0 by default, runs a complete BIST, and reports the first failing sub-test (for example, Test1).

In a typical case, running the default BIST command should result in a Passing ($00) result in the F09_BIST_Data1 register. In the case of a failing non-zero result, the output of the data register indicates the failed sub-test number. The failing sub-test result can be loaded into the Test Number Control register F09_BIST_Data0 to determine the cause of the fault. Running *RunBIST* (F09_BIST_Cmd0) again, with the *TestNumberControl* register properly loaded with the sub-test number, produces a result in the BIST data registers that indicates an electrode that failed the BIST sub-test.

9.2. Function $09: query registers

The various BIST queries provide the mechanism for configuring and reading the BIST controls (limits), commands (tests), and data (results) variables. The BIST limits may be configurable and set to defaults by flash-backed registers or set in flash on a per-product basis. Additional control limit registers are necessary for additional BIST tests.

Table 28. Function $09 query registers

Name	7	6	5	4	3	2	1	0
F09_BIST_Query0	LimitRegisterCount							
F09_BIST_Query1	HostTestEn	Internal Limits	—	—	—	ResultRegisterCount		

The locations of the control, data, and command registers for basic BIST tests are calculated from the F09_BIST_Query0 register. Any additional BIST tests and adjustments to the BIST register placements can be calculated based on F09_BIST_Query1.

9.2.1. F09_BIST_Query0: limit register count

Limit Register Count (F09_BIST_Query0)

Indicates the number of control registers used to set the limits for BIST testing capabilities.

9.2.2. F09_BIST_Query1

Result Register Count (F09_BIST_Query1, bits 2:0)

Indicates the number of result registers used to report BIST test results.

Reserved (F09_BIST_Query1, bits 5:3)

InternalLimits (F09_BIST_Query1, bit 6)

Indicates that BIST internal limits are available.

HostTestEn (F09_BIST_Query1, bit 7)

Indicates that BIST host-level testing is available. Test Limit control registers (for example, F09_BIST_Ctrl0 through F09_BIST_Ctrl5) can be updated by the host before the BIST command is run and those values will be used to determine failures.

9.3. Function $09: control registers

The control registers determine the limits for each of the implemented BIST commands.

Table 29. Function $09 Limit control registers

Name	7	6	5	4	3	2	1	0
F09_BIST_Ctrl0	Test1LimitLo							
F09_BIST_Ctrl1	Test1LimitHi							
F09_BIST_Ctrl2	Test1LimitDiff							
F09_BIST_Ctrl3	Test2LimitLo							
F09_BIST_Ctrl4	Test2LimitHi							
F09_BIST_Ctrl5	Test2LimitDiff							

9.3.1. F09_BIST_Ctrl0 through F09_BIST_Ctrl5: limit control

These registers are defined as follows:

Test1LimitLo (F09_BIST_Ctrl0)
Controls the acceptance Low BIST limit for Test1.

Test1LimitHi (F09_BIST_Ctrl1)
Controls the acceptance High BIST limit for Test1.

Test1LimitDiff (F09_BIST_Ctrl2)
Controls the acceptance Difference BIST limit for Test1.

Test2LimitLo (F09_BIST_Ctrl3)
Controls the acceptance Low BIST limit for Test2.

Test2LimitHi (F09_BIST_Ctrl4)
Controls the acceptance High BIST limit for Test2.

Test2LimitDiff (F09_BIST_Ctrl5)
Controls the acceptance Difference BIST limit for Test2.

9.4. Function $09: data registers

Function $09's data result registers are allocated as necessary for the tests defined by design capability. The data registers are filled upon the completion of a command. Reported data values may change, depending on the configuration of the control limit and the command executed. Future unimplemented extended BIST capabilities may add additional data registers. The number of Test Result data registers is indicated in the *ResultRegisterCount* (F09_BIST_Query1, bits 2:0) field.

The *TestNumberControl* register determines which BIST tests are run. If '0', all implemented tests are run.

Table 30. Function $09 data registers

Name	7	6	5	4	3	2	1	0
F09_BIST_Data0	TestNumberControl							
F09_BIST_Data1	OverallBISTResult							
F09_BIST_Data2	TestResult1							
F09_BIST_Data3	TestResult2							

9.4.1. F09_BIST_Data0: test number control

TestNumberControl (F09_BIST_Data0)

Controls the test (or tests) run by the BIST command. Test capabilities are defined and documented on a design basis. Additional tests may be available (beyond 0, 1 and 2), but not all devices will have all tests or necessarily sequential test numbers. '3'–'255' are reserved test numbers. By default:

- If '0', the complete BIST test executes each of the one or more BIST sub-tests in order until there is a failure. Reports the first failing test in the Test Result data register.
- If '1', then the *RunBIST* (F09_BIST_Cmd0) command executes the BIST sub-test 1 using the limits set by the Test1Limit control registers. Reports the failing electrodes and the failing control limit in the Test Result data registers.
- If '2', then the *RunBIST* (F09_BIST_Cmd0) command executes the BIST sub-test 2 using the limits set by the Test2Limit control registers. Reports the failing electrodes and the failing control limit in the Test Result data registers.

9.4.2. F09_BIST_Data1: overall BIST result

The *OverallBISTResult* register is '0' for success, or indicates the number of the first failing test.

OverallBISTResult (F09_BIST_Data1)

Reports the overall result of the BIST command executed:

- If '0', all requested BIST tests have passed.
- If '1', Test 1 has failed and F09_BIST_Data2 and F09_BIST_Data3 indicate the failing electrode numbers.
- If '2', Test 2 has failed and F09_BIST_Data2 and F09_BIST_Data3 indicate the failing electrode numbers.

'3' through '255' are reserved.

9.4.3. F09_BIST_Data2 and F09_BIST_Data3: test results 1 and 2

When the *ResultRegisterCount (*F09_BIST_Query1, bits 2:0) field is 2, multiple data registers provide the failure results.

For example, if Test1 failed, the Test Result data registers would indicate which limit test failed and the corresponding transmitter and receiver electrodes associated with the failure, as shown below.

Table 31. Function $09 Test Results, Test1 example

Name	7	6	5	4	3	2	1	0
F09_BIST_Data2	LimitFailureCode		ReceiverNumber					
F09_BIST_Data3	TransmitterNumber							

Receiver Number (F09_BIST_Data2, bits 5:0)

Reports an electrode that fails the test. The value of this register is meaningful only if F09_BIST_Data1 is non-zero. Indicates the first failing receiver number.

Limit Failure Code (F09_BIST_Data2, bits 7:6)

Reports which limit test fails:

‘00’ = No failure.
‘01’ = Low limit failed.
‘10’ = High limit failed.
‘11’ = Difference limit failed.

Transmitter Number (F09_BIST_Data2)

Reports an electrode that fails the test. The value of this register is meaningful only if F09_BIST_Data1 is non-zero. Indicates the first failing transmitter number.

9.5. Function $09: interrupt source

The BIST data source asserts an interrupt request at the end of any report period in which a command register bit has been cleared.

9.6. Function $09: command registers

The BIST command register provides a mechanism for executing the built-in self tests. Test times vary; test completion is indicated by the automatic clearing of the *RunBIST* bit and the assertion of the BIST interrupt request.

Table 32. Function $09 command register

Name	7	6	5	4	3	2	1	0
F09_BIST_Cmd0	—	—	—	—	—	—	—	RunBIST

9.6.1. F09_BIST_Cmd0: run BIST

RunBIST (F09_BIST_Cmd0)

Runs the BIST using the test number and limits set in the data and control registers.

10. Function $11: 2-D sensors

Function $11 implements two-dimensional touch position sensors, such as the Synaptics TouchPad™ or ClearPad™ products. RMI has three potential data sources for 2-D devices: absolute, relative and gestures:

- The absolute data source reports the positions of one or more fingers in the native X-Y coordinates of the 2-D sensor.
- The relative data source reports the delta X-Y coordinates.
- The gesture data source reports detection of finger gestures, and shares an interrupt request bit and an interrupt enable bit with the absolute data source.

10.1. Function version

In June 2011, the Function Version field in the Function $11 Page Description Table entry incremented from '0' to '1', changing the definition of this function in a way that is not backwards compatible with the function's earlier specification.

Two register blocks are affected by this change: F11_2D_Ctrl12 and F11_2D_Ctrl13.

- In Function $11 **version 0**: Registers F11_2D_Ctrl12.* and F11_2D_Ctrl13.* exist.
- In Function $11 **version 1**: Registers F11_2D_Ctrl12.* and F11_2D_Ctrl13.* do not exist.

The description of Function $11 in this document is accurate for any device whose function version is set to '1'. To determine the function version, refer to section 2.3.5.2 on page 24.

For a description of Function $11 version '0', see PN 511-000136-01.

10.2. Number of 2-D sensors

The *NumberOfSensors* field of the F11_2D_Properties register expresses how many distinct 2-D sensors are present in the device. When more than one 2-D sensor is present in the device, each operates independently of the other. Each sensor reports its data in separate data registers with separate interrupt request and interrupt enable bits. Similarly each 2-D sensor's properties are described by a separate set of queries, and each is configured by a separate set of control registers.

The grouping of query, control, data and command registers in the register map for multiple 2-D sensors is analogous to the grouping of RMI functions. For example, the data registers for the first 2-D sensor is followed by the data registers for the next. The order of the grouping of multiple sensors cannot be changed.

10.3. Function $11: query registers

A device may contain more than one distinct 2-D sensor. As a result, there are two classes of Function $11 queries:

1. Queries that apply to all 2-D sensors in a device.
2. Queries that apply to a particular 2-D sensor within a device.

The per-device 2-D query registers are placed first in the register map, followed by one block of per-sensor query registers for each 2-D sensor supported by the device.

Table 33. Function $11 query registers

Name	7	6	5	4	3	2	1	0
F11_2D_Query0	—	—	Has Query12	Has Query11	HasQuery 9	NumberOfSensors		
F11_2D_Query1	Configurable	Has Sensitivity Adjust	Has Gestures	HasAbs Mode	HasRel Mode	NumberOfFingers		
F11_2D_Query2	—	NumberOfXElectrodes						
F11_2D_Query3	—	NumberOfYElectrodes						
F11_2D_Query4	—	MaximumElectrodes						
F11_2D_Query5	HasJitter Filter	Has Large Object Suppression	Has Bending Correct	Has Dribble	Has Adjust Hyst	Has Anchored Finger	AbsoluteDataSize	
F11_2D_Query6	—	—	—	—	—	—	—	—
F11_2D_Query7	HasChiral Scroll	HasPinch	HasPress	HasFlick	HasEarly Tap	Has DoubleTap	HasTap andHold	Has Single Tap
F11_2D_Query8	Has MultiFinger ScrollInertia	Has Multi FingerScroll EdgeMotion	HasMulti Finger Scroll	Individual Scroll Zones	HasScroll Zones	HasTouch Shapes	Has Rotate	Has Palm Detect
F11_2D_Query9	HasPen Hover AndEdge Filters	HasPen Hover Discrimination—	Has Contact Geometry	HasTwo Pen Thresholds	Has SuppressonPalm Detect	HasLarge Object Sensitivity	Has Finger Proximity	HasPen
F11_2D_Query10	—	—	—	NumberOfTouchShapes				
F11_2D_Query11	Has Drumming Correction	HasTxRx Clip	Has Segment-ation Aggressiveness	Has Default Finger Width	HasPitch Info	HasW Tuning	Has Algorithm Selection	HasZTuning
F11_2D_Query12	—	—	Has Physical Properties	Has General Information2	Has2D Adjustable Mapping	Has 8-bitW	Has Gapless Finger Tuning	Has Gapless Finger
F11_2D_Query13	—	JitterFilterType		JitterAverageWindowSize				
F11_2D_Query14	Has Advanced Driver	MouseButtons		ClickPadProperties		IsClear	LightControl	

Name	7	6	5	4	3	2	1	0
F11_2D_Query15	XSensorSizeLSB							
F11_2D_Query16	XSensorSizeMSB							
F11_2D_Query17	YSensorSizeLSB							
F11_2D_Query18	YSensorSizeMSB							
F11_2D_Query19	XBezelOriginOffsetLSB							
F11_2D_Query20	XBezelOriginOffsetMSB							
F11_2D_Query21	YBezelOriginOffsetLSB							
F11_2D_Query22	YBezelOriginOffsetMSB							
F11_2D_Query23	XBezelOppositeOffsetLSB							
F11_2D_Query24	XBezelOppositeOffsetMSB							
F11_2D_Query25	YBezelOppositeOffsetLSB							
F11_2D_Query26	YBezelOppositeOffsetMSB							

The fields in these registers are defined in the following sections.

10.3.1. F11_2D_Query0: per-device query registers

RMI Function $11 defines a single per-device query register. The fields in the register are defined as follows:

NumberOfSensors (F11_2D_Query0, bits 2:0)

This 3-bit field describes the number of 2-D sensors supported by Function $11. It is zero-based, so a value of ‘0’ means one sensor, ‘1’ means two sensors, and so on.

Note: The number of sensors indicated by this field influences the size and layout of a particular device’s Function $11 register map.

HasQuery9 (F11_2D_Query0, bit 3)

If *HasQuery9* reports as ‘1’, then the F11_2D_Query9 register exists.

HasQuery11 (F11_2D_Query0, bit 4)

If *HasQuery11* reports as ‘1’, then the F11_2D_Query11 register exists.

HasQuery12 (F11_2D_Query0, bit 5)

If *HasQuery12* reports as ‘1’, then the F11_2D_Query12 register exists.

This bit must be set to ‘1’ when *Has2DAdjustableMapping* (F11_2D_Query12, bit 3), *Has8BitW* (F11_2D_Query12, bit 2), or *HasGaplessFinger* (F11_2D_Query12, bit 0) are true.

Reserved (F11_2D_Query0, bits 7:6)

Reserved.

10.3.2. F11_2D_Query1 through F11_2D_Query12: per-sensor query registers

Each 2-D sensor has its own set of query registers, used to describe its own particular characteristics. The fields in these registers are defined in the following sections.

10.3.3. F11_2D_Query1: general sensor information

NumberOfFingers (F11_2D_Query1, bits 2:0)

This 3-bit field describes the maximum number of fingers the 2-D sensor supports.

Table 34. Code indicates number of fingers supported

Encoding	Maximum Number of Fingers Reported
000	1
001	2
010	3
011	4
100	5
101	10
110	Reserved
111	Reserved

Note: Because the field defines a maximum finger count, a 10-finger maximum allows the device to report any number of fingers up to and including 10.

HasRelMode (F11_2D_Query1, bit 3)

If *HasRelMode* reports as '1', then the sensor supports relative mode, and the relative mode data registers are present in the register map. The relative mode data registers indicate finger movement in the form of position deltas since the last report.

HasAbsMode (F11_2D_Query1, bit 4)

If *HasAbsMode* reports as '1', then the sensor supports absolute mode, and absolute mode data registers are present in the register map. The absolute mode data registers provide the finger positions in the form of absolute positions.

HasGestures (F11_2D_Query1, bit 5)

If *HasGestures* reports as '1', the sensor supports gesture processing. The gesture data source interprets specific finger movement events over short periods of time to provide the host with information regarding higher-level user inputs such as tapping, pinching, flicking, or pressing.

HasSensitivityAdjust (F11_2D_Query1, bit 6)

If *HasSensitivityAdjust* reports as '1', the sensor supports a global sensitivity adjustment. This allows a host to make small adjustments to the overall sensitivity of the pad. If this bit reports as '1', then register F11_2D_Ctrl14 (Sensitivity Adjust) will appear in the register map; otherwise, it will not be present.

Configurable (F11_2D_Query1, bit 7)

If *Configurable* reports as '1', the sensor supports configuration.

10.3.4. F11_2D_Query2 and F11_2D_Query3: number of X and Y electrodes

NumberOfXElectrodes (F11_2D_Query2, bits 6:0)

NumberOfYElectrodes (F11_2D_ Query3, bits 6:0)

The *NumberOfXElectrodes* and *NumberOfYElectrodes* queries describe the maximum number of electrodes the 2-D sensor supports on each axis.

10.3.5. F11_2D_Query4: maximum electrodes

MaximumElectrodes (F11_2D_Query4, bits 6:0)

'0' is a valid value for this register, even if F11_2D_Query2 and F11_2D_Query3 are non-zero.

10.3.6. F11_2D_Query5: absolute data source

This query register is present only if *HasAbsMode* in F11_2D_Query1 reports as '1'.

AbsDataSize (F11_2D_Query, bits 1:0)

This two bit field describes the type of data reported by the absolute data source. From this, the total number of absolute data registers can be calculated.

- *'00'* – The absolute position data source will report X, Y, Z and W data using a total of five data registers. This block of registers will be replicated for each finger present in the sensor.
- *'01', '10', '11'* – *Reserved*.

HasAnchoredFinger (F11_2D_Query5, bit 2)

If *HasAnchoredFinger* reports as '1', then the sensor supports the high-precision second finger tracking provided by the F11_2D_Ctrl1 manual tracking and motion sensitivity options. This is a two finger mode, and is only available when the *NumberOfFingers* query reports that two or more fingers are supported. If a first finger touches and remains steady, then a second finger can be tracked with high precision. If the first finger moves, then the reported positions of both the first and second fingers may degrade.

HasAdjHyst (F11_2D_Query5, bit 3)

Z hysteresis is the difference between the finger *release threshold* and the *touch threshold*. If Z falls below the release threshold (the smaller of the two values), it is interpreted as a finger not touching the sensor; if Z rises above the touch threshold, it is interpreted as a finger touching the sensor.

If *HasAdjHyst* reports as '1', the hysteresis setting for the sensor can be changed. Register F11_2D_Ctrl14 provides a field for adjusting the default hysteresis setting.

HasDribble (F11_2D_Query5, bit 4)

If *HasDribble* reports as '1', the sensor supports the generation of dribble interrupts, which may be enabled or disabled with the *Dribble* bit (F11_2D_Ctrl0, bit 6).

Typically, interrupts are generated (ATTN is asserted) when a finger arrives, lifts, or moves. The exact criteria for generating interrupts vary from one device to the next. Interrupts are generated only while the criteria are met.

For example, if the device is configured to generate interrupts only while the finger is moving, the device stops generating interrupts the instant the finger stops moving.

When dribble is enabled, interrupts continue to be generated for a short while even after the interrupt-generating criteria are no longer met. These extra interrupts are called “dribble interrupts.”

HasBendingCorrection (F11_2D_Query5, bit 5)

If *HasBendingCorrection* reports as ‘1’, then registers F11_2D_Data28, F11_2D_Data36, and registers F11_2D_Ctrl52 through F11_2D_Ctrl57 exist.

HasLargeObjectSuppression (F11_2D_Query5, bit 6)

If *HasLargeObjectSuppression* reports as ‘1’, then registers F11_2D_Ctrl58 and F11_2D_Data28 exist.

HasJitterFilter (F11_2D_Query5, bit 7)

If *HasJitterFilter* reports as ‘1’, then registers F11_2D_Query13 and F11_2D_Ctrl73 through F11_2D_Ctrl76.* exist.

If *HasJitterFilter* reports as ‘0’, then registers F11_2D_Query13 and F11_2D_Ctrl73 through F11_2D_Ctrl76.* are not present.

10.3.7. F11_2D_Query6: relative data source

This query register is present only if *HasRelMode* in F11_2D_Query1 reports as ‘1’. This register defines no queries regarding the relative data source.

10.3.8. F11_2D_Query7 and F11_2D_Query8: gesture information

These query registers are present only if *HasGestures* in F11_2D_Query1 reports as ‘1’.

HasSingleTap (F11_2D_Query7, bit 0)

If *HasSingleTap* reports as ‘1’, a basic *single-tap* gesture is supported. A single-tap is defined to be a sequence of events where a finger lands on the sensor, stays in essentially the same location (defined by *Maximum Tap Distance*) for no more than a brief period of time (defined by *Maximum Tap Time*), and then leaves.

If *HasDoubleTap* or *HasTapAndHold* is true, a single-tap will not be reported for a short period of time after the finger leaves. The delay is necessary to distinguish a single-tap from the start of a tap-and-hold or double-tap gesture. If a host application desires to act on a tap event as soon as the finger lifts, it should use the early-tap report instead.

HasTapAndHold (F11_2D_Query7, bit 1)

If *HasTapAndHold* reports as ‘1’, a basic *tap-and-hold* gesture is supported. A tap-and-hold is defined to be a sequence of events where a finger lands on the sensor, stays in essentially the same location (defined by *Maximum Tap Distance*) for no more than a brief period of time (defined by *Maximum Tap Time*), leaves the sensor, then lands again in the same approximate location within a brief period time (a function of *Maximum Tap Time*), and remains touching the sensor for longer than *Maximum Tap Time*.

HasDoubleTap (F11_2D_Query7, bit 2)

If *HasDoubleTap* reports as ‘1’, a *double-tap* gesture is supported. A double-tap is defined to be a sequence of two taps separated by a brief time interval based on *Maximum Tap Time*. A double-tap is reported as soon as the finger leaves the sensor for the second time.

HasEarlyTap (F11_2D_Query7, bit 3)

If *HasEarlyTap* reports as '1', *early taps* are reported by the gesture data source. An early tap is reported as soon as the finger lifts for any tap event that could be interpreted as either a single tap or as the first tap of a double-tap or tap-and-hold gesture.

HasFlick (F11_2D_Query7, bit 4)

If *HasFlick* reports as '1', the 2-D sensor supports a *flick* gesture. A flick gesture is defined to be where a single finger leaves the sensor while moving faster than a specified speed (defined by *Minimum Flick Speed*) and beyond a specified distance (defined by *Minimum Flick Distance*).

HasPress (F11_2D_Query7, bit 5)

If *HasPress* reports as '1', the *press* gesture is supported. In a press gesture, a single finger touches the sensor and stays in essentially the same location for a moderate interval of time (defined by *Minimum Press Time*).

HasPinch (F11_2D_Query7, bit 6)

If *HasPinch* reports as '1', the sensor supports the two-finger *pinch* gesture. A pinch gesture is defined to be two fingers touching the pad and moving either towards each other or away from each other.

HasChiralScroll (F11_2D_Query7, bit 7)

When this bit reports as '1', registers F11_2D_Data14, F11_2D_Data15, F11_2D_Ctrl59, F11_2D_Ctrl60, and F11_2D_Ctrl79 exist.

F11_2D_Data8 bit 7 reports *ChiralScroll* and F11_2D_Ctrl10 bit 7 controls *ChiralScrollIntEn*.

HasPalmDetect (F11_2D_Query8, bit 0)

If *HasPalmDetect* reports as '1', the 2-D sensor notifies the host whenever a large conductive object such as a palm or a cheek touches the 2-D sensor.

Note: Most 2-D data reporting is inhibited while a palm is detected.

HasRotate (F11_2D_Query8, bit 1)

If *HasRotate* reports as '1', the sensor supports the two finger *rotate* gesture. A rotate gesture is defined as:

- exactly two fingers touching the pad,
- the two fingers maintaining approximately the same distance apart (if the fingers move together, the sensor may interpret the movement as a pinch gesture), and
- the fingers swiveling clockwise or counterclockwise on the sensor surface, as if the hand is turning a dial or knob.

HasTouchShapes (F11_2D_Query8, bit 2)

If *HasTouchShapes* reports as '1', *TouchShapes* are supported. A TouchShape is a fixed rectangular area on the sensor that behaves like a capacitive button.
If *HasTouchShapes* reports as '1', then the F11_2D_Query10 register is used by this device and replicated register F11_2D_Data13.* exists.

HasScrollZones (F11_2D_Query8, bit 3)

If *HasScrollZones* reports as '1', *ScrollZones* are supported. A ScrollZone is a narrow rectangular area on one of the four edges of the sensor that reports relative motion. The zones parallel to the Y axis are called "Y Left" (at the edge of the screen where X is at a minimum) and "Y Right" (at the edge where X is at a maximum). The zones parallel to the X axis are called "X Lower" (at the edge where Y is at a minimum) and "X Upper" (at the edge where Y is at a maximum).

IndividualScrollZones (F11_2D_Query8, bit 4)

If *IndividualScrollZones* reports as '1', there are four Scroll Motion data registers, one for each of the four ScrollZones. If *IndividualScrollZones* reports as '0', then there are only two Scroll Motion data registers.

Motion on either of the "X" ScrollZones is reported in the "X Lower Scroll Motion" data register, and motion on either of the "Y" ScrollZones is reported in the "Y Right Scroll Motion" data register.

HasMultiFingerScroll (F11_2D_Query8, bit 5)

If *HasMultiFingerScroll* reports as '1', MultiFinger Scrolling is supported. When multiple fingers land at the same time and move in unison, the MultiFinger Scrolling gesture bit is set and the amount of scrolling is reported.

HasMultiFingerScrollEdgeMotion (F11_2D_Query8, bit 6)

If *HasMultiFingerScrollEdgeMotion* reports as '1' and *HasMultiFingerScroll* reports as '1', the MultiFinger Scroll Edge Motion feature is supported. When edge motion is supported, F11_2D_Ctrl28, bit 2 enables the feature. The edge boundaries are product-specific and are fixed.

HasMultiFingerScrollInertia (F11_2D_Query8, bit 7)

If *HasMultiFingerScrollInertia* reports as '1' and *HasMultiFingerScroll* reports as '1', MultiFinger Scroll Inertia is supported. When inertia is supported, F11_2D_Ctrl28, bits 7:4 configure the feature.

10.3.9. F11_2D_Query9: advanced sensing features

This register identifies the sensor's advanced sensing features. It only exists if *HasQuery9* (F11_2D_Query0, bit 3) is set to '1'

HasPen (F11_2D_Query9, bit 0)

If *HasPen* reports as '1', detection of a stylus is supported and registers F11_2D_Ctrl20 and F11_2D_Ctrl21 exist.

HasProximity (F11_2D_Query9, bit 1)

If *HasProximity* reports as '1', detection of fingers near the sensor is supported and registers F11_2D_Ctrl22 through F11_2D_Ctrl26 exist.

HasPalmDetectSensitivity (F11_2D_Query9, bit 2)

If *HasPalmDetectSensitivity* reports as '1', the sensor supports the palm detect sensitivity feature and register F11_2D_Ctrl27 exists.

HasSuppressOnPalmDetect (F11_2D_Query9, bit 3)

If *HasSuppressOnPalmDetect* reports as '1', the sensor supports the palm detect suppression feature and register F11_2D_Ctrl27 exists.

HasTwoPenThresholds (F11_2D_Query9, bit 4)

If *HasTwoPenThresholds* reports as '1' and *HasPen* reports as '1', register F11_2D_Ctrl35 exists.

HasContactGeometry (F11_2D_Query9, bit 5)

If *HasContactGeometry* reports as '1,' then the sensor supports the use of contact geometry to map absolute X and Y target positions and registers F11_2D_Data18.* through F11_2D_Data27 exist.

10.3.10. F11_2D_Query10: TouchShape support

This register is present only if *HasTouchShapes* (F11_2D_Query8, bit 2) is set to '1'.

NumberOfTouchShapes (F11_2D_ Query10, bits 4:0)

Holds the number (minus one) of TouchShapes supported by the sensor.

10.3.11. F11_2D_Query11: advanced sensing features 1

This register is present only if *HasQuery11* (F11_2D_Query0, bit 4) is set to '1'.

HasZTuning (F11_2D_Query11, bit 0)

If *HasZTuning* reports as '1', the sensor supports Z tuning and registers F11_2D_Ctrl29 through F11_2D_Ctrl33 exist.

HasAlgorithmSelection (F11_2D_Query11, bit 1)

If *HasAlgorithmSelection* reports as '1', the sensor supports configuration of the position-interpolation and post-correction algorithm and register F11_2D_Ctrl34 exists.

HasWTuning (F11_2D_Query11, bit 2)

If *HasWTuning* reports as '1', the sensor supports Wx and Wy scaling and registers F11_2D_Ctrl36 through F11_2D_Ctrl39 exist.

HasPitchInfo (F11_2D_Query11, bit 3)

If *HasPitchInfo* reports as '1', the X and Y pitches of the sensor electrodes can be configured and registers F11_2D_Ctrl40 and F11_2D_Ctrl41 exist.

HasFingerSize (F11_2D_Query11, bit 4)

If *HasFingerSize* reports as '1', the default finger width settings for the sensor can be configured and registers F11_2D_Ctrl42 through F11_2D_Ctrl44 exist.

HasSegmentationAggressiveness (F11_2D_Query11, bit 5)

If *HasSegmentationAggressiveness* reports as '1', the sensor's ability to distinguish multiple objects close together can be configured and register F11_2D_Ctrl45 exists.

HasXYClip(F11_2D_Query11, bit 6)

If *HasXYClip* reports as '1', the inactive outside borders of the sensor can be configured and registers F11_2D_Ctrl46 through F11_2D_Ctrl49 exist.

HasFastFlick (F11_2D_Query11, bit 7)

If *HasFastFlick* reports as '1', the sensor can be configured to distinguish between a fast flick and a quick drumming movement and registers F11_2D_Ctrl50 and F11_2D_Ctrl51 exist.

10.3.12. F11_2D_Query12: advanced sensing features 2

HasGaplessFinger (F11_2D_Query12, bits 0)

When this bit reports as '1', F11_2D_Ctrl61 through F11_2D_Ctrl67 exist.

HasGaplessFingerTuning (F11_2D_Query12, bit 1)

When this bit and F11_2D_Query12 bit 0 are both '1', registers F11_2D_Ctrl66, and F11_2D_Data32 through F54_AD_Data34 exist.

Has8-bitW (F11_2D_Query12, bit 2)

If *Has8-bitW* reports as '1', replicated register F11_2D_Data35.* exists.

Has2DAdjustableMapping (F11_2D_Query12, bit 3)

When this bit reports as '1', registers F11_2D_Ctrl77 and F11_2D_Ctrl78 exist and replace F11_2D_Query2 through F11_2D_Query4 (that is, the host should ignore the values in F11_2D_Query2 through F11_2D_Query4 and instead use the values in F11_2D_Ctrl77 and F11_2D_Ctrl78).

HasGeneralInformation2 (F11_2D_Query12, bit 4)

When this bit reports as '1', register F11_2D_Query14 is present.

HasPhysicalProperties (F11_2D_Query12, bit 5)

When this bit reports as '1', registers F11_2D_Query15 through F11_2D_Query26 are present.

Reserved (F11_2D_Query12, bits 7:6)

Reserved.

10.3.13. F11_2D_Query13: jitter data

JitterAverageWindowSize (F11_2D_Query13, bits 3:0)

JitterFilterType (F11_2D_Query13, bits 6:4)

Reserved (F11_2D_Query13, bit 7)

Reserved.

10.3.14. F11_2D_Query14: general information 2

This register is present only if *HasGeneralInformation2* (F11_2D_Query12, bit 4) reports as '1'.

LightControl (F11_2D_Query14, bits 1:0)

00 = No Light Control
01 = LuxPad
10 = Dual-Mode LuxPad
11 = Reserved

IsClear (F11_2D_Query14, bit 2)

0 = Opaque
1 = Clear

ClickPadProperties (F11_2D_Query14, bits 4:3)

00 = Not a ClickPad
01 = Hinged ClickPad
10 = Uniform ClickPad
11 = Reserved

MouseButtons (F11_2D_Query14, bits 6:5)

00 = No Mouse Buttons
01 = Has Left Button only
10 = Has Left + Right Buttons
11 = Has Left + Right + Middle Buttons

HasAdvancedDriverGestures (F11_2D_Query14, bit 7)

When this bit reports as '1', advanced driver gestures are supported.

10.3.15. F11_2D_Query15 through F11_2D_Query18: sensor size

These registers are present only if *HasPhysicalProperties* (F11_2D_Query12, bit 5) reports as '1'.

XSensorSizeLSB (F11_2D_Query15)

XSensorSizeMSB (F11_2D_Query16)

YSensorSizeLSB (F11_2D_Query17)

YSensorSizeMSB (F11_2D_Query18)

These registers report the length of the sensor's axes, in millimeters expressed as unsigned fixed-point 12.4 numbers.

10.3.16. F11_2D_Query19 through F11_2D_Query22: bezel origin offset

These registers are present only if *HasPhysicalProperties* (F11_2D_Query12, bit 5) reports as '1'.

XBezelOriginOffsetLSB (F11_2D_Query19)

XBezelOriginOffsetMSB (F11_2D_Query20)

YBezelOriginOffsetLSB (F11_2D_Query21)

YBezelOriginOffsetMSB (F11_2D_Query22)

These registers report the position of the bezel's origin relative to the sensor's origin, in millimeters expressed as signed fixed-point 8.8 numbers.

10.3.17. F11_2D_Query23 through F11_2D_Query26: bezel opposite offset

These registers are present only if *HasPhysicalProperties* (F11_2D_Query12, bit 5) reports as '1'.

XBezelOppositeOffsetLSB (F11_2D_Query23)

XBezelOppositeOffsetMSB (F11_2D_Query24)

YBezelOppositeOffsetLSB (F11_2D_Query25)

YBezelOppositeOffsetMSB (F11_2D_Query26)

These registers report the position of the bezel corner opposite its origin relative to the sensor corner opposite its origin, in millimeters expressed as signed fixed-point 8.8 numbers.

10.4. Function $11: control registers

These registers control the operation of the 2-D sensors.

Table 35. Function $11 control registers

Name	7	6	5	4	3	2	1	0
F11_2D_Ctrl0	Report Beyond Clip	Dribble	Relative Ballistics	Relative PosFilter	AbsolutePosFilter	ReportingMode		
F11_2D_Ctrl1	Man Tracked Finger	ManTrack En	MotionSensitivity		PalmDetectThreshold			
F11_2D_Ctrl2	DeltaXPositionThreshold							
F11_2D_Ctrl3	DeltaYPositionThreshold							
F11_2D_Ctrl4	Velocity							
F11_2D_Ctrl5	Acceleration							
F11_2D_Ctrl6	SensorMaximumXPosition [7:0]							
F11_2D_Ctrl7	—	—	—	—	SensorMaximumXPosition [11:8]			
F11_2D_Ctrl8	SensorMaximumYPosition [7:0]							
F11_2D_Ctrl9	—	—	—	—	SensorMaximumYPosition [11:8]			
F11_2D_Ctrl10	Chiral Scroll IntEn	Pinch Interrupt Enable	Press Interrupt Enable	Flick Interrupt Enable	EarlyTap Interrupt Enable	Double Tap Interrupt Enable	TapAnd Hold Interrupt Enable	SingleTap Interrupt Enable
F11_2D_Ctrl11	—	—	—	MultiFinger Scroll Interrupt Enable	ScrollZone Interrupt Enable	Touch Shape Interrupt Enable	Rotate Interrupt Enable	PalmDetect Interrupt Enable
	Note: F11_2D_Ctrl12.* and F11_2D_Ctrl13.* do not exist in Function $11 version 1 (V1 is the current version of Function $11). See section 10.1 for details.							
F11_2D_Ctrl14	HysteresisAdjustment			SensitivityAdjustment (per sensor)				
F11_2D_Ctrl15	MaximumTapTime							
F11_2D_Ctrl16	MinimumPressTime							
F11_2D_Ctrl17	MaximumTapDistance							
F11_2D_Ctrl18	MinimumFlickDistance							
F11_2D_Ctrl19	MinimumFlickSpeed							
F11_2D_Ctrl20	—	—	—	—	—	—	PenJitter Filter Enable	PenDetect Interrupt Enable
F11_2D_Ctrl21	PenZLowerThreshold							
F11_2D_Ctrl22	—	—	—	—	—	—	Proximity JitterFilter Enable	Proximity Detect Interrupt Enabl
F11_2D_Ctrl23	ProximityDetectionZThreshold							
F11_2D_Ctrl24	ProximityDeltaXThreshold							
F11_2D_Ctrl25	ProximityDeltaYThreshold							
F11_2D_Ctrl26	ProximityDeltaZThreshold							
F11_2D_Ctrl27	—			Suppress OnPalm Detect	Palm-DetectSensitivity			

Name	7	6	5	4	3	2	1	0
F11_2D_Ctrl28	Multi-FingerScrollMomentum				—	Edge Motion Enable	Multi-FingerScrollMode	
F11_2D_Ctrl29	ZTouchThreshold							
F11_2D_Ctrl30	ZTouchHysteresis							
F11_2D_Ctrl31	SmallZThreshold							
F11_2D_Ctrl32.0	SmallZScaleFactorLSB [7:0]							
F11_2D_Ctrl32.1	SmallZScaleFactorMSB [15:8]							
F11_2D_Ctrl33.0	LargeZScaleFactorLSB [7:0]							
F11_2D_Ctrl33.1	LargeZScaleFactorMSB [15:8]							
F11_2D_Ctrl34	—		PositionPostCorrection			PositionInterpolation		
F11_2D_Ctrl35	PenZUpperThreshold							
F11_2D_Ctrl36	WxScaleFactor							
F11_2D_Ctrl37	WxOffset							
F11_2D_Ctrl38	WyScaleFactor							
F11_2D_Ctrl39	WyOffset							
F11_2D_Ctrl40.0	XPitch [7:0]							
F11_2D_Ctrl40.1	XPitch [15:8]							
F11_2D_Ctrl41.0	YPitch [7:0]							
F11_2D_Ctrl41.1	YPitch [15:8]							
F11_2D_Ctrl42.0	DefaultFingerWidthForX [7:0]							
F11_2D_Ctrl42.1	DefaultFingerWidthForX s [15:8]							
F11_2D_Ctrl43.0	DefaultFingerWidthForY [7:0]							
F11_2D_Ctrl43.1	DefaultFingerWidthForY [15:8]							
F11_2D_Ctrl44	—							Report MeasureSize
F11_2D_Ctrl45	SegmentationAggressiveness							
F11_2D_Ctrl46	ReceiverClipLSB [7:0]							
F11_2D_Ctrl47	ReceiverClipMSB [7:0]							
F11_2D_Ctrl48	TransmitterClipLSB [7:0]							
F11_2D_Ctrl49	TransmitterClipMSB [7:0]							
F11_2D_Ctrl50	MinimumDrummingSeparation							
F11_2D_Ctrl51	MaximumDrummingMovement							
F11_2D_Ctrl52	BendingDetectionThreshold							
F11_2D_Ctrl53	BendingFingerDetectionThreshold							
F11_2D_Ctrl54.0	PeakBendingCapacitanceLSB							
F11_2D_Ctrl54.1	PeakBendingCapacitanceMSB							
F11_2D_Ctrl55	SingleFingerPitchCorrection				MultipleFingerPitchCorrection			
F11_2D_Ctrl56	ReceiverAxisBendingCorrection							
F11_2D_Ctrl57	TransmitterAxisBendingCorrection							
F11_2D_Ctrl58	Inhibit ATTN	LargeObjectSize						

Name	7	6	5	4	3	2	1	0
F11_2D_Ctrl59	Chiral Start Mode	—	ChiralScrollStartAngle			ChiralScrollSpeed		
F11_2D_Ctrl60	—	ChiralScrollStartDistance						
F11_2D_Ctrl61	GaplessTouchThresholdFactor							
F11_2D_Ctrl62.0	DefaultFingerProfileCurvatureLSB							
F11_2D_Ctrl62.1	DefaultFingerProfileCurvatureMSB							
F11_2D_Ctrl63	HysteresisOfFingerProfileWidthVariation							
F11_2D_Ctrl64	HysteresisOfFingerProfileAmplitudeVariation							
F11_2D_Ctrl65	BorderReportingRejectionZone							
F11_2D_Ctrl66.0	WidthOfThe1stCoordinateOfHoveringRejectionCurveLSB							
F11_2D_Ctrl66.1	WidthOfThe1stCoordinateOfHoveringRejectionCurveMSB							
F11_2D_Ctrl66.2	AmplitudeOfThe1stCoordinateOfHoveringRejectionCurveLSB							
F11_2D_Ctrl66.3	AmplitudeOfThe1stCoordinateOfHoveringRejectionCurveMSB							
...	...							
F11_2D_Ctrl66.14	AmplitudeOfThe4thCoordinateOfHoveringRejectionCurveLSB							
F11_2D_Ctrl66.15	AmplitudeOfThe4thCoordinateOfHoveringRejectionCurveMSB							
F11_2D_Ctrl67	—					Enable Gapless Finger Tuning OnEdges	—	Enable Gapless Finger Tuning
F11_2D_Ctrl68.0	QuadraticCoefficentLSB							
F11_2D_Ctrl68.1	QuadraticCoefficentMSB							
F11_2D_Ctrl68.2	LineraCoefficentLSB							
F11_2D_Ctrl68.3	LineraCoefficentMSB							
F11_2D_Ctrl68.4	ConstantTermLSB							
F11_2D_Ctrl68.5	ConstantTermMSB							
F11_2D_Ctrl69	HysteresisPercentage							
F11_2D_Ctrl70	NumberOfPenTemporalFilterFrames							
F11_2D_Ctrl71	PenBorderReportingRejectionZoneSize							
F11_2D_Ctrl72	ZOffsetTolerance							
F11_2D_Ctrl73	—					JitterFilterStrength		
F11_2D_Ctrl74	JitterFreeRunFrames							
F11_2D_Ctrl75.0	LGMThresholdScale					SmallJitterXThreshold		
F11_2D_Ctrl75.1	—					SmallJitterYThreshold		
F11_2D_Ctrl75.2	SmallJitterZThreshold							
F11_2D_Ctrl76.0	LargeJitterXThreshold							
F11_2D_Ctrl76.1	LargeJitterYThreshold							
F11_2D_Ctrl76.2	LargeJitterZThreshold							
F11_2D_Ctrl77	NumberOf2DReceivers							
F11_2D_Ctrl78	NumberOf2DTransmitters							
F11_2D_Ctrl79	—					TriggerZoneWidth		

10.4.1. F11_2D_Ctrl0: general control

This register contains general control bits that affect 2-D reporting and operation. The fields are defined as follows:

ReportingMode (F11_2D_Ctrl0, bits 2:0)

Depending on the specific application, a host may desire to tailor the rate and nature of the reports generated by a 2-D sensor. For example, hosts that have a low bandwidth connection to an RMI device may wish to restrict the conditions under which the device generates reports, so as to reduce reporting bandwidth requirements. The reporting mode is a 3-bit field that describes the various reporting models.

ReportingMode = '000': *Continuous, when finger present*

The absolute data source interrupt is asserted for every reporting period during which one or more fingers are touching the 2-D sensor. A final interrupt is asserted after the last finger has been lifted.

ReportingMode = '001': *Reduced reporting mode*

In this mode, the absolute data source interrupt is asserted whenever a finger arrives or leaves. Fingers that are present but basically stationary do not generate additional interrupts unless their positions change significantly. In specific, for fingers already touching the pad, the interrupt is asserted whenever the change in finger position exceeds either *DeltaXPosThreshold* or *DeltaYPosThreshold*.

Note: The contents of the *PalmDetectThreshold* register are used in this mode.

ReportingMode = '010': *Finger-state change reporting mode*

In this mode, the absolute data source interrupt is asserted whenever a finger arrives or leaves. Changes in finger position while a finger is down do not generate interrupts, regardless of their magnitude.

Notes:

- The contents of the *DeltaXPosThreshold* and *DeltaYPosThreshold* registers are ignored in this mode.
- The contents of the *PalmDetectThreshold* register are ignored in this mode.

ReportingMode = '011': *Finger-presence change reporting mode*

In this mode, the absolute data source interrupt is asserted only when the number of fingers on the sensor changes to or from zero.

Note: The contents of the *DeltaXPosThreshold* and *DeltaYPosThreshold* registers are ignored in this mode.

ReportingMode = '111': *Continuous reporting mode*

In this mode, the absolute data source interrupt is asserted on every reporting period, regardless of finger presence, finger state changes, or finger position changes.

All other *ReportingMode* values are *reserved*.

AbsolutePosFilter (F11_2D_Ctrl0, bit 3)

When set to ‘1’, this control bit enables various filtering algorithms that reduce noise and jitter in the reported absolute position, at the expense of moderate (usually undetectable) lag in response to very fast finger motions. For products which support stylus detection, this bit is set to ‘0’ (disabled) by default and should not be changed.

Note: The *AbsolutePosFilt* bit has an effect only when *ReportingMode* = ‘000’ (continuous when finger present).

RelativePosFilter (F11_2D_Ctrl0, bit 4)

When set to ‘1’, this control bit enables filtering of the reported relative position changes. When set to ‘0’, relative position filtering is disabled.

RelativeBallistics (F11_2D_Ctrl0, bit 5)

When set to ‘1’, this control bit enables ballistics processing for the relative finger motion on the 2-D sensor. When set to ‘0’, ballistics processing is disabled.

Dribble (F11_2D_Ctrl0, bit 6)

When set to ‘1’, this control bit enables dribbling. When set to ‘0’, dribbling is disabled. This bit has no effect unless *HasDribble* (F11_2D_Query5, bit 4) reports as ‘1’.

ReportBeyondClip (F11_2D_Ctrl0, bit 7)

This bit has an effect only when *HasXYClip* reports as '1'.

When this bit is set to '1', fingers outside the active area specified by the *XClip* and *YClip* registers (F11_2D_Ctrl46 through F11_2D_Ctrl49) will be reported, but with reported finger position clipped to the edge of the active area.

When this bit is set to '0', fingers outside the active area will be ignored.

10.4.2. F11_2D_Ctrl1: palm and finger control

PalmDetectThreshold (F11_2D_Ctrl1, bits 3:0)

Specifies the threshold at which a wide finger is considered a palm. A value of 0 inhibits palm detection. The sensor inhibits its position reporting interrupts whenever either Wx or Wy is greater than or equal to the *PalmDetectThreshold*. This feature is only available when Reporting Mode is set to ‘001’.

MotionSensitivity (F11_2D_Ctrl1, bits 5:4)

This field specifies the threshold an anchored finger must move before it is considered no longer anchored. The settings are:

- 00 – Low motion sensitivity
- 01 – Medium motion sensitivity
- 10 – High motion sensitivity
- 11 – Infinite motion sensitivity

This control field is only available when the *HasAnchoredFinger* query reports as ‘1’. The measurements related to these settings are product-specific. The ranges for Low, Medium, and High sensitivity may overlap.

ManualTrackingEn (F11_2D_Ctrl1, bit 6)

This bit specifies whether the host or the device determines which finger is the tracked finger:

- If this bit is set to '0', the device chooses the tracked finger.
- If this bit is set to '1', the host chooses the tracked finger.

This control field is only available when the *HasAnchoredFinger* query reports as '1'.

ManuallyTrackedFinger (F11_2D_Ctrl1, bit 7)

This bit is only meaningful when *HasAnchoredFinger* reports as '1' and *ManualTrackingEn* (F11_2D_Ctrl1, bit 6) is set to '1'.

This bit indicates which finger is being tracked:

- If this bit is set to '0', Finger #0 is being tracked.
- If this bit is set to '1', Finger #1 is being tracked.

Finger #0 is the finger reported in the first set of finger data registers (F11_2D_Data1.0 through F11_2D_Data5.0) while Finger #1 is the finger reported in the second set of finger data (F11_2D_Data1.1 thru F11_2D_Data5.1).

10.4.3. F11_2D_Ctrl2 and F11_2D_Ctrl3: distance threshold

DeltaXPositionThreshold (F11_2D_Ctrl2)

DeltaYPositionThreshold (F11_2D_Ctrl3)

In ReportingMode '001' (Reduced Reporting), 2-D position update interrupts are inhibited unless the finger moves more than a certain threshold distance along either axis. These registers are used to define the thresholds for each axis.

A value of 0 for both registers means that any change in position will cause an interrupt, similar to reporting mode '000' (continuous, when finger present).

10.4.4. F11_2D_Ctrl4: velocity control

Velocity (F11_2D_Ctrl4)

When *RelBallistics* is set to '1', this register defines the velocity ballistic parameter applied to all relative motion events. If *RelBallistics* is set to '0', this register is ignored.

10.4.5. F11_2D_Ctrl5: acceleration control

Acceleration (F11_2D_Ctrl5)

When *RelBallistics* is set to '1', this register defines the acceleration ballistic parameter applied to all relative motion events. If *RelBallistics* is set to '0', this register is ignored.

10.4.6. F11_2D_Ctrl6 through F11_2D_Ctrl9: maximum X and Y position control

SensorMaxXPosLSB (F11_2D_Ctrl6)

SensorMaxXPosMSB (F11_2D_Ctrl7)

SensorMaxYPosLSB (F11_2D_Ctrl8)

SensorMaxYPosMSB (F11_2D_Ctrl9)

These 12-bit fields are used to define a sensor's maximum X and Y positions. X positions are reported in the range 1 through *SensorMaxXPos*-1; Y positions are reported in the range 1 through *SensorMaxYPos*-1. A reported X position of 0 or *SensorMaximumXPosition*, or a reported Y position of 0 or *SensorMaximumYPosition*, indicates that the position is outside of reportable range.

Note: Not all 2-D sensors are capable of sensing these extreme values. For example, if a sensor cannot recognize values outside its reportable range, then it will not report positions equal to 0 or *SensorMaximumX/YPosition*.

The units for the sensor position are arbitrary. A host may choose to have the maximum dynamic range by setting both of these fields to their maximum value: 4095. Hosts implementing 2-D clear sensors over an LCD display may choose to set these registers so that reportable positions match the number of pixels on an axis. Other choices, for example, allow a host to get positions reported in physical distance units like millimeters.

Notes: The two *SensorMaximumXPosition* registers (F11_2D_Ctrl6 and F11_2D_Ctrl7) are a coherent group: To change either register, both registers must be written (see section 2.6).

The two *SensorMaximumYPosition* registers (F11_2D_Ctrl8 and F11_2D_Ctrl9) are a coherent group: To change either register, both registers must be written (see section 2.6).

10.4.7. F11_2D_Ctrl10 and F11_2D_Ctrl11: gesture control

Each of these control registers is present only if at least one of the gestures it controls is supported (for example, F11_2D_Ctrl10 is only present if F11_2D_Query7 is non-zero, and F11_2D_Ctrl11 is only present if F11_2D_Query8 is non-zero). For a definition of each of the gestures referenced below, see the documentation for the corresponding query register F11_2D_Query1 (see section 10.3.3).

SingleTapIntEn (F11_2D_Ctrl10, bit 0)

Setting *SingleTapIntEn* to '1' enables a gesture interrupt in response to a single-tap gesture. A *single-tap* gesture is also sometimes referred to as a *tap* gesture.

TapAndHoldIntEn (F11_2D_Ctrl10, bit 1)

Setting *TapAndHoldIntEn* to '1' enables a gesture interrupt in response to a tap-and-hold gesture.

DoubleTapIntEn (F11_2D_Ctrl10, bit 2)

Setting *DoubleTapIntEn* to '1' enables a gesture interrupt in response to a double-tap gesture.

EarlyTapIntEn (F11_2D_Ctrl10, bit 3)

Setting *EarlyTapIntEn* to '1' enables a gesture interrupt in response to an early-tap gesture.

FlickIntEn (F11_2D_Ctrl10, bit 4)

Setting *FlickIntEn* to '1' enables a gesture interrupt in response to a flick gesture.

PressIntEn (F11_2D_Ctrl10, bit 5)

Setting *PressIntEn* to '1' enables a gesture interrupt in response to a press gesture.

PinchIntEn (F11_2D_Ctrl10, bit 6)

Setting *PinchIntEn* to '1' enables a gesture interrupt in response to a pinch gesture.

ChiralScrollIntEn (F11_2D_Ctrl10, bit 7)

Setting *ChiralScrollIntEn* to '1' enables a gesture interrupt in response to a chiral scroll gesture.

PalmDetIntEn (F11_2D_Ctrl11, bit 0)

Setting *PalmDetIntEn* to '1' enables a gesture interrupt in response to a palm detect gesture.

RotateIntEn (F11_2D_Ctrl11, bit 1)

Setting *RotateIntEn* to '1' enables a gesture interrupt in response to a rotate gesture.

TouchShapeIntEn (F11_2D_Ctrl11, bit 2)

Setting *TouchShapeIntEn* to '1' enables a gesture interrupt in response to a TouchShape touch or lift.

ScrollZoneIntEn (F11_2D_Ctrl11, bit 3)

Setting *ScrollZoneIntEn* to '1' enables a gesture interrupt in response to motion on a ScrollZone.

MultiFingerScrollIntEn (F11_2D_Ctrl11, bit 4)

Setting *MultiFingerScrollIntEn* to '1' enables a gesture interrupt in response to multi-finger scrolling.

Note: Gestures are always detected and reported in the gesture data registers, regardless of the setting of the corresponding interrupt enables in F11_2D_Ctrl10 and F11_2D_Ctrl11. The interrupt-enable registers only define which gestures will participate in generating ATTN; they do not actually mask reporting of the data bits.

10.4.8. F11_2D_Ctrl12.* does not exist

Important: The F11_2D_Ctrl12.* registers no longer exist.

10.4.9. F11_2D_Ctrl13.* does not exist

Important: The F11_2D_Ctrl13.* registers no longer exist.

10.4.10. F11_2D_Ctrl14: sensitivity adjustment

This register is only present if the *HasSensitivityAdjus*t bit in F11_Query1 reports as '1'.

SensitivityAdjustment (F11_2D_Ctrl14, bits 4:0)

This 5-bit signed field allows a host to alter the overall sensitivity of a 2-D sensor. As shipped, the default *SensitivityAdjustment* of 0 will correspond to the factory recommended overall sensitivity setting for the sensor.

A host may choose to make the sensor more or less sensitive than the factory setting by changing the value in this register. A positive value in this register will make the sensor more sensitive than the factory defaults, and a negative value will make it less sensitive.

HysteresisAdjustment (F11_2D_Ctrl14, bits 7:5)

This 3-bit unsigned field controls the sensor's Z hysteresis setting. The factory recommended setting is '0'. Changing this setting to a value higher than '0' will increase the hysteresis by two Z units per count such that the maximum setting of '7' corresponds to an increase in hysteresis of 14 Z units.

Warning: Setting this register to extreme values may render the sensor incapable of detecting touch or release events.

10.4.11. F11_2D_Ctrl15: maximum tap time

This control register is present only if *HasGestures* reports as '1' in the per-sensor query register F11_2D_Query1 (see section 10.3.3) and *HasEarlyTap*, *HasSingleTap*, *HasDoubleTap*, or *HasTapAndHold* reports as '1' in the per-sensor query register F11_2D_Query7.

MaximumTapTime (F11_2D_Ctrl15)

Determines the maximum duration of a tap, in 10-millisecond units. Maximum intertap-time (between the two taps of a double-tap gesture or between the tap and the hold of a tap-and-hold gesture) is proportional to this value.

10.4.12. F11_2D_Ctrl16: minimum press time

This control register is present only if *HasGestures* reports as '1' in the per-sensor query register F11_2D_Query1 (see section 10.3.3) and *HasPress* reports as '1' in the per-sensor query register F11_2D_Query7.

MinimumPressTime (F11_2D_Ctrl16)

The minimum duration required for stationary finger(s) to generate a press gesture, in 10-millisecond units.

10.4.13. F11_2D_Ctrl17: maximum tap distance

This control register is present only if *HasGestures* reports as '1' in the per-sensor query register F11_2D_Query1 (see section 10.3.3) and *HasEarlyTap*, *HasSingleTap*, *HasDoubleTap*, *HasTapAndHold*, or *HasPress* reports as '1' in the per-sensor query register F11_2D_Query7.

MaximumTapDistance (F11_2D_Ctrl17)

Determines the maximum finger movement allowed during a tap, in 0.1-millimeter units.

The maximum allowed inter-tap movement is proportional to these values, as is maximum movement allowed during a press.

10.4.14. F11_2D_Ctrl18: minimum flick distance

This control register is present only if *HasGestures* reports as '1' in the per-sensor query register F11_2D_Query1 (see section 10.3.3) and *HasFlick* reports as '1' in the per-sensor query register F11_2D_Query7.

MinimumFlickDistance (F11_2D_Ctrl18)

Determines the minimum finger movement for a flick gesture, in 1-millimeter units.

10.4.15. F11_2D_Ctrl19: minimum flick speed

This control register is present only if *HasGestures* reports as '1' in the per-sensor query register F11_2D_Query1 (see section 10.3.3) and *HasFlick* reports as '1' in the per-sensor query register F11_2D_Query7.

MinimumFlickSpeed (F11_2D_Ctrl19)

Determines the minimum finger speed for a flick gesture, in 10-millimeter/second units. Small values indicate lower speeds, large values indicate higher speeds.

10.4.16. F11_2D_Ctrl20: pen control

This control register is present only if *HasPen* reports as '1' in the per-sensor query register F11_2D_Query9.

PenDetectInterruptEnable (F11_2D_Ctrl20, bit 0)

Setting this bit to '1' enables an interrupt in response to stylus activity on the sensor.

PenJitterFilterEnable (F11_2D_Ctrl20, bit 1)

Setting this bit to '1' enables the stylus anti-jitter filter.

Note: Changes to this register do not take effect until the *ForceUpdate* command (F54_AD_Cmd0, bit 2) is executed.

10.4.17. F11_2D_Ctrl21: pen Z lower threshold

This control register is present only if *HasPen* reports as '1' in the per-sensor query register F11_2D_Query9.

PenZThreshold (F11_2D_Ctrl21)

This is the stylus-detection lower threshold. Smaller values result in higher sensitivity.

10.4.18. F11_2D_Ctrl22: proximity control

This control register is present only if *HasProximity* reports as '1' in F11_2D_Query9, bit 1.

ProximityDetectInterruptEnable (F11_2D_Ctrl22, bits 1:0)

Setting this bit to '1' enables an interrupt in response to proximity detection on the sensor.

ProximityJitterFilterEnable (F11_2D_Ctrl22, bit 2)

Setting this bit to '1' enables the proximity anti-jitter filter.

Reserved (F11_2D_Ctrl22, bit 7:3

Reserved.

10.4.19. F11_2D_Ctrl23: proximity detection Z threshold

This control register is present only if *HasProximity* reports as '1' in the per-sensor query register F11_2D_Query9.

ProximityZThreshold (F11_2D_Ctrl23)

This is the threshold for finger-proximity detection.

10.4.20. F11_2D_Ctrl24: proximity detection X threshold

This control register is present only if *HasProximity* reports as '1' in the per-sensor query register F11_2D_Query9.

ProximityDelta-XThreshold (F11_2D_Ctrl24)

In reduced-reporting modes, this is the threshold for proximate-finger movement in the direction parallel to the X-axis.

10.4.21. F11_2D_Ctrl25: proximity detection Y threshold

This control register is present only if *HasProximity* reports as '1' in the per-sensor query register F11_2D_Query9.

ProximityDelta-YThreshold (F11_2D_Ctrl25)

In reduced-reporting modes, this is the threshold for proximate-finger movement in the direction parallel to the Y-axis.

10.4.22. F11_2D_Ctrl26: proximity delta Z threshold

This control register is present only if *HasProximity* reports as '1' in the per-sensor query register F11_2D_Query9.

ProximityDelta-ZThreshold (F11_2D_Ctrl26)

In reduced-reporting modes, this is the threshold for proximate-finger movement in the direction parallel to the Z-axis.

10.4.23. F11_2D_Ctrl27: palm-detect parameters

This control register is present only if *HasPalmDetectSensitivity* or *HasSuppressOnPalmDetect* reports as '1' in the per-sensor query register F11_2D_Query9.

PalmDetectSensitivity (F11_2D_Ctrl27, bits 3:0)

This field is meaningful only if *HasPalmDetectSensitivity* reports as '1'.

This field is a signed 4-bit value which adjusts the palm-detect sensitivity. '0' is the factory default. When this value is small, smaller objects will be identified as palms; when this value is large, only larger objects will be identified as palms.

The range of acceptable values for this field is -7 to +7 (1110 binary to 0111 binary). -8 (1111 binary) is reserved.

SuppressOnPalmDetect (F11_2D_Ctrl27, bit 4)

This bit is meaningful only if *HasSuppressOnPalmDetect* reports as '1'.

When this bit is set to '1', all F$11 interrupts except PalmDetect are suppressed while a palm is detected.

While the PalmDetect bit (F11_2D_Data9, bit 0) reports as '1', the device will behave as though the F$11 Absolute data-source interrupt is disabled, the F$11 Relative data-source is disabled, and all F$11 gesture interrupts except PalmDetect are disabled.

Reserved (F11_2D_Ctrl27, bits 7:5)

Reserved.

10.4.24. F11_2D_Ctrl28: multi-finger scroll parameters

This control register is present only if *HasMultiFingerScroll* reports as '1' in the per-sensor query register F11_2D_Query8.

MultiFingerScrollMode (F11_2D_Ctrl28, bits 1:0)

This field is an unsigned 2-bit value which allows choice of multi-finger scroll mode and determines whether and how X or Y displacements are reported.

- 0, Locking: Movement is reported in only one axis. Once X-axis or Y-axis multi-finger motion is detected, movement is reported only along that axis until the fingers are lifted. Zero is reported for the opposite axis.
- 1, Locking with Suppression: Movement is reported in only one axis. As in Mode 0, movement is reported only along one axis until the fingers are lifted. However, if the opposite axis becomes the axis of greater motion, zero is reported for *both* axes until the original axis again becomes the axis of greater motion or the fingers are lifted.
- 2, Single Axis without Locking: Movement is reported in only one axis. Movement is reported along the current axis of greater motion.
- 3, Free: Movement is reported along both axes simultaneously.

MultiFingerScrollEdgeMode (F11_2D_Ctrl28, bit 2)

If *MultiFingerEdgeMode* reports as '1' and *HasMultiFingerScrollEdgeMotion* reports as '1', edge-motion scrolling is enabled.

Reserved (F11_2D_Ctrl28, bit 3)

This bit is reserved.

MultiFingerScrollInertiaMomentum (F11_2D_Ctrl28, bits 7:4)

This field is an unsigned 4-bit value which controls the length of time that scrolling continues after fingers have been lifted. It has an effect only if *HasMultiFingerScrollInertia* reports as '1'.

10.4.25. F11_2D_Ctrl29: z touch threshold

This control register is present only if *HasZTuning* (F11_2D Query11, bit 0) reports as '1'.

ZTouchThreshold (F11_2D_Ctrl29)

Specifies the finger-arrival Z threshold. Large values may cause smaller fingers to be rejected.

10.4.26. F11_2D_Ctrl30: z touch hysteresis

This control register is present only if *HasZTuning* (F11_2D Query11, bit 0) reports as '1'.

ZTouchHysteresis (F11_2D_Ctrl30)

Specifies the difference between the finger-arrival Z threshold *ZTouchThreshold (F11_2D_Ctrl29)* and the finger-departure Z threshold. The range is 0 to 255.

10.4.27. F11_2D_Ctrl31: small z threshold

This control register is present only if *HasZTuning* (F11_2D Query11, bit 0) reports as '1'.

SmallZThreshold (F11_2D_Ctrl31)

Specifies the threshold below which Z is scaled using *SmallZScaleFactor* (F11_2D_Ctrl32.*). Above the threshold, Z is scaled using *LargeZScaleFactor* (F11_2D_Ctrl33.*). The range is 0 to 255.

10.4.28. F11_2D_Ctrl32.0/32.1: small z scale factor

This control register is present only if *HasZTuning* (F11_2D Query11, bit 0) reports as '1'.

SmallZScaleFactorLSB (F11_2D_Ctrl32.0)

SmallZScaleFactorMSB (F11_2D_Ctrl32.10)

Specifies the scale factor used when Z is below *SmallZThreshold* (F11_2D_Ctrl31). This is an unsigned fixed-point number in Q1.15 format (range is 0-1.99997).

10.4.29. F11_2D_Ctrl33.0/33.1: large z scale factor

This control register is present only if *HasZTuning* (F11_2D Query11, bit 0) reports as '1'.

LargeZScaleFactorLSB (F11_2D_Ctrl33.0)

LargeZScaleFactorMSB (F11_2D_Ctrl33.1)

Specifies the scale factor used when Z is above *SmallZThreshold* (F11_2D_Ctrl31). This is an unsigned fixed-point number in Q1.15 format (range is 0-1.99997).

10.4.30. F11_2D_Ctrl34: algorithm selection

This control register is present only if *HasAlgorithmSelection* (F11_2D Query11, bit 1) reports as '1'.

PositionInterpolation (F11_2D_Ctrl34, bits 2:0)

Specifies the position-interpolation algorithm to be used:

- 000 – Default
- 001 – Algorithm 1
- 010 – Algorithm 2
- 011 – Algorithm 3
- 100-111 – Reserved

PositionPostCorrection (F11_2D_Ctrl34, bits 5:3)

Specifies the post-correction algorithm to be used:

- 000 – None
- 001 – Default
- 010-111 – Reserved

Reserved (F11_2D_Ctrl34, bits 7:6)

These bits are reserved.

10.4.31. F11_2D_Ctrl35: pen Z upper threshold

This control register is present only if *HasPen* (F11_2D Query9, bit 0) reports as '1' and *HasTwoPenThresholds* (F11_2D Query9, bit 4) reports as '1'.

PenZUpperThreshold (F11_2D_Ctrl35)

When F11_2D_Query9, bit 4 is set to 1, Ctrl35 is the Pen Z Upper Threshold.

10.4.32. F11_2D_Ctrl36: Wx scale factor

This control register is present only if *HasWTuning* (F11_2D Query11, bit 2) reports as '1'.

WxScaleFactor (F11_2D_Ctrl36)

Specifies the scale factor used to calculate *Wx*.

This register contains the scale factor used in converting raw W to reported W on the X-axis. It is an unsigned number in Q4.4 fixed-point format.

10.4.33. F11_2D_Ctrl37: Wx offset

This control register is present only if *HasWTuning* (F11_2D Query11, bit 2) reports as '1'.

WxOffset (F11_2D_Ctrl37)

Specifies the offset used to calculate *Wx*.

This register contains the offset used in converting raw W to reported W on the X-axis. It is a signed 8-bit number.

10.4.34. F11_2D_Ctrl38: Wy scale factor

This control register is present only if *HasWTuning* (F11_2D Query11, bit 2) reports as '1'.

WyScaleFactor (F11_2D_Ctrl38)

Specifies the scale factor used to calculate *Wy*.

This register contains the scale factor used in converting raw W to reported W on the Y-axis. It is an unsigned number in Q4.4 fixed-point format.

10.4.35. F11_2D_Ctrl39: Wy offset

This control register is present only if *HasWTuning* (F11_2D Query11, bit 2) reports as '1'.

WyOffset (F11_2D_Ctrl39)

Specifies the offset used to calculate *Wy*.

This register contains the offset used in converting raw W to reported W on the Y-axis. It is a signed 8-bit number.

10.4.36. F11_2D_Ctrl40.0/40.1: x pitch

These registers are present only if *HasPitchInfo* (F11_2D Query11, bit 3) reports as '1'. This register pair contains the pitch in millimeters for the X-axis in Q4.12 fixed-point format.

XPitchLSB (F11_2D_Ctrl40.0)

XPitchMSB (F11_2D_Ctrl40.1)

Specifies the pitch of the X electrodes on the sensor.

10.4.37. F11_2D_Ctrl41.0/41.1: y pitch

These control registers are present only if *HasPitchInfo* (F11_2D Query11, bit 3) reports as '1'.

YPitchLSB (F11_2D_Ctrl41.0)

YPitchMSB (F11_2D_Ctrl41.1)

Specifies the pitch of the Y electrodes on the sensor.

This register pair contains the pitch in millimeters for the Y-axis in Q4.12 fixed-point format.

10.4.38. F11_2D_Ctrl42.0/42.1: finger size on X axis

These control registers are present only if *HasFingerSize* (F11_2D Query11, bit 4) reports as '1'.

FingerSizeXLSB (F11_2D_Ctrl42.0)

FingerSizeXMSB (F11_2D_Ctrl42.1)

When only part of a finger is detected (as when a finger arrives from the edge of the sensor), an assumption must be made about the size of the finger. Writing to these registers sets the assumed size of the finger when measured along the X-axis, in product-specific units.

If *ReportMeasuredSize* (F11_2D_Ctrl44, bit 0) is set to '0', these registers contain the assumed size of the finger.

If *ReportMeasuredSize* (F11_2D_Ctrl44, bit 0) is set to '1', these registers contain the actual measured size of the finger on the sensor (for tuning purposes).

10.4.39. F11_2D_Ctrl43.0/43.1: finger size on Y axis

These control registers are present only if *HasFingerSize* (F11_2D Query11, bit 4) reports as '1'.

FingerSizeYLSB (F11_2D_Ctrl43.0)

FingerSizeYMSB (F11_2D_Ctrl43.1)

When only part of a finger is detected (as when a finger arrives from the edge of the sensor), an assumption must be made about the size of the finger. Writing to these registers sets the assumed size of the finger when measured along the Y-axis, in product-specific units.

If *ReportMeasuredSize* (F11_2D_Ctrl44, bit 0) is set to '0', these registers contain the assumed size of the finger.

If *ReportMeasuredSize* (F11_2D_Ctrl44, bit 0) is set to '1', these registers contain the actual measured size of the finger on the sensor (for tuning purposes).

10.4.40. F11_2D_Ctrl44: report measured size

ReportMeasuredSize (F11_2D_Ctrl44, bit 0)

When this bit is set to '0', registers F11_2D_Ctrl42 and F11_2D_Ctrl43 report the assumed finger size – the value previously written to those registers. When this bit is set to '1', registers F11_2D_Ctrl42 and F11_2D_Ctrl43 report the actual measured size of the first finger that is processed.

The measured size is reported in the same units in which the assumed finger size should be stored.

Note: This bit should only be set to '1' for tuning purposes.

Reserved (F11_2D_Ctrl44, bits7:1)

Reserved.

10.4.41. F11_2D_Ctrl45: segmentation aggressiveness

This control register is present only if *HasSegmentationAggressiveness* (F11_2D Query11, bit 5) reports as '1'.

SegmentationAggressiveness (F11_2D_Ctrl45)

Controls the sensor's ability to distinguish multiple objects placed very close to each other. Larger values allow close objects to be distinguished more reliably, but may cause the sensor to erroneously split large objects.

10.4.42. F11_2D_Ctrl46: receiver clip LSB

This control register is present only if *HasXYClip* (F11_2D Query11, bit 6) reports as '1'.

ReceiverClipLSB (F11_2D_Ctrl46)

Specifies the width of the inactive border on the left edge of the sensor, in units of 1/32 mm.

10.4.43. F11_2D_Ctrl47: receiver clip MSB

This control register is present only if *HasXYClip* (F11_2D Query11, bit 6) reports as '1'.

ReceiverClipMSB (F11_2D_Ctrl47)

Specifies the width of the inactive border on the right edge of the sensor, in units of 1/32 mm.

10.4.44. F11_2D_Ctrl48: transmitter clip LSB

This control register is present only if *HasXYClip* (F11_2D Query11, bit 6) reports as '1'.

TransmitterClipLSB (F11_2D_Ctrl48)

Specifies the width of the inactive border on the upper edge of the sensor, in units of 1/32 mm.

10.4.45. F11_2D_Ctrl49: transmitter clip MSB

This control register is present only if *HasXYClip* (F11_2D Query11, bit 6) reports as '1'.

TransmitterClipMSB (F11_2D_Ctrl49)

Specifies the width of the inactive border on the lower edge of the sensor, in units of 1/32 mm.

10.4.46. F11_2D_Ctrl50: minimum drumming separation

This control register is present only if *HasFastFlick* (F11_2D Query11, bit 7) reports as '1'.

MinimumDrummingSeparation(F11_2D_Ctrl50)

It can be difficult to distinguish between two "drumming" fingers (a finger lifting at one position followed immediately by a finger touching at another position) and a single finger performing a rapid flick gesture.

This register specifies the minimum distance, in millimeters, between two drumming fingers. A finger that moves less than this distance between reports will never be misreported as two drumming fingers.

10.4.47. F11_2D_Ctrl51: maximum drumming movement

This control register is present only if *HasFastFlick* (F11_2D Query11, bit 7) reports as '1'.

MaximumDrummingMovement (F11_2D_Ctrl51)

A finger which appears to move more than the Minimum Drumming Separation between two reports is also examined on the following report. Its movement on that third report determines whether it is a single finger performing a rapid flick or two drumming fingers. This register specifies the maximum distance, in millimeters, that a drumming finger may move on the third report. A finger which exceeds the Minimum Drumming Separation and then also exceeds the Maximum Drumming Movement will never be misreported as two drumming fingers.

10.4.48. F11_2D_Ctrl52: bending detection threshold

This control register is present only if *HasBendingCorrection* (F11_2D_Query5, bit 5) reports as '1'.

BendingDetectionThreshold (F11_2D_Ctrl52)

Sets the threshold for activation of Bending Correction. When set to '0', Bending Correction is always active; when set to 255 Bending Correction is disabled.

10.4.49. F11_2D_Ctrl53: bending finger detection threshold

This control register is present only if *HasBendingCorrection* (F11_2D_Query5, bit 5) reports as '1'.

BendingFingerDetectionThreshold (F11_2D_Ctrl53)

Sets the threshold for finger detection while Bending Correction is active.

10.4.50. F11_2D_Ctrl54.0/54.1: peak bending capacitance

This replicated register is only present if *HasBendingCorrection* (F11_2D_Query5, bit 5) reports as '1'.

PeakBendingCapacitanceLSB (F11_2D_Ctrl54.0)

PeakBendingCapacitanceMSB (F11_2D_Ctrl54.1)

10.4.51. F11_2D_Ctrl55: finger interaction and response

This control register is present only if *HasBendingCorrection* (F11_2D_Query5, bit 5) reports as '1'.

MultipleFingerPitchCorrection (F11_2D_Ctrl55, bits 3:0)

Describes the spread of finger signals across pixels for the purpose of multiple-finger bending correction.

SingleFingerPitchCorrection (F11_2D_Ctrl55, bits 7:4)

Describes the spread of finger signals across pixels for the purpose of single-finger bending correction.

Note: These two fields are usually set to equal values.

10.4.52. F11_2D_Ctrl56: receiver axis bending correction

This control register is present only if *HasBendingCorrection* (F11_2D_Query5, bit 5) reports as '1'.

ReceiverAxisBendingCorrection (F11_2D_Ctrl56)

Specifies the magnitude of the bending correction applied to the axis composed of receiver electrodes. Smaller values move the reported finger position closer to the center of the sensor; larger values move the reported finger position closer to the edges.

10.4.53. F11_2D_Ctrl57: transmitter axis bending correction

This control register is present only if *HasBendingCorrection* (F11_2D_Query5, bit 5) reports as '1'.

TransmitterAxisBendingCorrection (F11_2D_Ctrl57)

Specifies the magnitude of the bending correction applied to the axis composed of transmitter electrodes. Smaller values move the reported finger position closer to the center of the sensor; larger values move the reported finger position closer to the edges.

10.4.54. F11_2D_Ctrl58: large object suppression parameters

This control register is present only if *HasLargeObjectSuppression* (F11_2D_Query5, bit 6) reports as '1'. The fields in this register are defined as follows:

LargeObjectSize (F11_2D_Ctrl58, bits 6:0)

Defines the minimum size of large objects whose presence on the sensor should be ignored.

InhibitATTN (F11_2D_Ctrl58, bit 7)

When set to '1', the ATTN signal will be suppressed while a large object is present on the sensor and being ignored, although other objects on the sensor will continue to be tracked.

When set to '0', the ATTN signal will not be suppressed.

10.4.55. F11_2D_Ctrl59: chiral scroll parameters

This control register is present only if *HasChiralScroll* (F11_2D_Query7, bit 7) reports as '1'. The fields in this register are defined as follows:

ChrialScrollSpeed (F11_2D_Ctrl59, bits 2:0)

The Chiral Scroll speed is set as follows:

0 = zero motion
1 = slowest
7 = fastest.

ChiralScrollStartAngle (F11_2D_Ctrl59, bits 5:3)

The minimum angular progression the finger heading must take to trigger the chiral scroll gesture once the start distance has been achieved.

This is valid only when *ChiralStartMode* (F11_2D_Ctrl59, bit 7) is set to '0'.

Reserved (F11_2D_Ctrl59, bit 6)

Reserved.

ChiralStartMode (F11_2D_Ctrl59, bit 7)

When set to '1' a separate X and Y zone exist; when set to '0', no zones exist.

10.4.56. F11_2D_Ctrl60: chiral scroll distance

ChiralScrollDistance(F11_2D_Ctrl60, bits 6:0)

The Chiral Scroll Start Distance, in 1-millimeter units. This is the cumulative movement required to make indication of the chiral scroll gesture possible. This is only valid when *ChiralStartMode (F11_2D_Ctrl59, bit 7)* is set to '0'.

Reserved (F11_2D_Ctrl60, bit 7)

Reserved.

10.4.57. F11_2D_Ctrl61: gapless touch threshold factor

This register only exists when *HasGaplessFinger* (F11_2D_Query12, bit 0) reports as '1'.

GaplessTouchThresholdFactor (F11_2D_Ctrl61)

This register is used to set the fractional factor of the gapless touch threshold versus the finger touch threshold. The factor is expressed in the fixed-point 0.8 format, which has a range of 0 to 255/256 with a granularity of 1/256. For this register, value 0 means the factor is not set.

10.4.58. F11_2D_Ctrl62.0/62.1: default finger profile curvature

These registers only exist when *HasGaplessFinger* (F11_2D_Query12, bit 0) reports as '1'.

DefaultFingerProfileCurvatureLSB (F11_2D_Ctrl62. 0)

DefaultFingerProfileCurvatureMSB (F11_2D_Ctrl62.1)

When a finger approaches from corners, its curvature of profile cannot be estimated due to an incomplete touch footprint. This register is used to set a default curvature for such occasion.

10.4.59. F11_2D_Ctrl63: hysteresis of finger profile width variation

This register only exists when *HasGaplessFinger* (F11_2D_Query12, bit 0) reports as '1'.

HysteresisOfFingerProfileWidthVariation (F11_2D_Ctrl63)

Once a finger is reported, an extra fraction of sensor pitch is allowed at the demarcation of touching and hovering decision. This register expresses the fraction in the fixed-point 0.8 format, which has a range of 0 to 255/256 with a granularity of 1/256.

10.4.60. F11_2D_Ctrl64: hysteresis of finger profile amplitude variation

This register only exists when *HasGaplessFinger* (F11_2D_Query12, bit 0) reports as '1'.

HysteresisOfFingerProfileAmplitudeVariation (F11_2D_Ctrl64)

Once a finger is reported, an extra delta ADC offset in either a positive or a negative direction is allowed at the demarcation of touching and hovering decision. This register sets the delta ADC offset with a range of 0 to 255, which expresses hysteresis up to a range of -255 to 255.

10.4.61. F11_2D_Ctrl65: border reporting rejection zone

This register only exists when *HasGaplessFinger* (F11_2D_Query12, bit 0) report as '1'.

BorderReportingRejectionZone (F11_2D_Ctrl65)

When a finger is first detected in a zone near edges, the reporting is suppressed until the finger moves out of the zone and enters the central area of the sensor. This register specifies the zone by giving the width of the border in the unit of sensor pitch. It is in the fixed-point 1.7 format, which has a range of 0 to 511/256. Setting the value to 0 disables the rejection zone.

10.4.62. F11_2D_Ctrl66.0 through F11_2D_Ctrl66.15: hovering rejection curve

These registers only exist when *HasGaplessFinger* (F11_2D_Query12 bit 0) reports as '1'.

WidthOfThe1stCoordinateOfHoveringRejectionCurveLSB (F11_2D_Ctrl66.0)

WidthOfThe1stCoordinateOfHoveringRejectionCurveMSB (F11_2D_Ctrl66.1)

AmplitudeOfThe1stCoordinateOfHoveringRejectionCurveLSB (F11_2D_Ctrl66.2)

AmplitudeOfThe1stCoordinateOfHoveringRejectionCurveMSB (F11_2D_Ctrl66.3)

...

AmplitudeOfThe4thCoordinateOfHoveringRejectionCurveLSB (F11_2D_Ctrl66.14)

AmplitudeOfThe4thCoordinateOfHoveringRejectionCurveMSB (F11_2D_Ctrl66.15)

These registers specify four coordinates in the Width-Amplitude scatter plot to form a piecewise-linear curve separating touching and hovering objects. Each coordinate consists of a width in the fixed-point 1.15 format and amplitude in 16-bit unsigned integers.

The scatter plot is generated by reading F11_2D_Data32 through F11_2D_Data34. Each pair of data results in a dot in the plot. The amplitude of each dot is read from F11_2D_Data32. The width is the smaller one of the square root of the reciprocal of F11_2D_Data33 and F11_2D_Data34. The width and the curvatures are in different fixed-point formats.

10.4.63. F11_2D_Ctrl67: enable gapless finger tuning

This register only exists when *HasGaplessFingerTuning* (F11_2D_Query12, bit 1) reports as '1' and *HasGaplessFinger* (F11_2D_Query12, bit 0) reports as '1'.

EnableGaplessFingerTuning (F11_2D_Ctrl67, bit 0)

When this bit is set, F11_2D_Data32 through F11_2D_Data34 report amplitude and curvatures when a finger is detected on the sensor. When the bit is not set, F11_2D_Data32 through F11_2D_Data34 will not be updated.

Reserved (F11_2D_Ctrl67, bit 1)

Reserved.

EnableGaplessFingerTuningOnEdges (F11_2D_Ctrl67, bit 2)

When this bit is set, F11_2D_Data32 through F11_2D_Data34 report amplitude and curvatures only when a finger is detected on the sensor border within one sensor pitch to an edge. When the finger is more than one sensor pitch away from the edge, F11_2D_Data32 through F11_2D_Data34 will not be updated.

When the bit is not set, F11_2D_Data32 through F11_2D_Data34 are updated only when a finger is detected more than one sensor pitch away from the edge toward the central area.

Reserved (F11_2D_Ctrl67, bits 7:3)

Reserved.

10.4.64. F11_2D_Ctrl68.0/68.5: quadratic and linear coefficients

Each of these registers is set to its optimal value by Design Studio 4.

QuadraticCoefficientLSB (F11_2D_Ctrl68.0)

QuadraticCoefficientMSB (F11_2D_Ctrl68.1)

LinearCoefficientLSB (F11_2D_Ctrl68.2)

LinearCoefficientMSB (F11_2D_Ctrl68.3)

ConstantTermLSB (F11_2D_Ctrl68.4)

ConstantTermMSB (F11_2D_Ctrl68.5)

10.4.65. F11_2D_Ctrl69: hysteresis percentage

This register only exists when *HasPenHoverDiscrimination* (F11_2D_Query9 bit 6) reports as '1' and *HasPen* (F11_2D_Query9, bit 0) reports as '1'.

HysteresisPercentage (F11_2D_Ctrl69)

Once a pen is reported, an extra delta Cxy offset percentage in the negative direction is allowed at the demarcation of pen and hovering decision. The maximum percentage is 99. Setting 0 means that no hysteresis is allowed.

10.4.66. F11_2D_Ctrl70: number of pen temporal filter frames

This register exists if *HasPen* (F11_2D_Query9, bit 0) reports as '1' and *HasPenHoverAndEdgeFilters* (F11_2D_Query9 bit 7) reports as '1'.

NumberOfPenTemporalFilterFrames (F11_2D_Ctrl70)

Number of frames to hold off pen reporting. The valid range is 0 to 15. Setting 0 means that temporal filtering is disabled.

10.4.67. F11_2D_Ctrl71: pen border reporting rejection zone size

This register exists if *HasPen* (F11_2D_Query9, bit 0) reports as '1' and *HasPenHoverAndEdgeFilters* (F11_2D_Query9 bit 7) reports as '1'.

PenBorderReportingRejectionZoneSize (F11_2D_Ctrl71)

When a finger or pen arrives in a zone near the sensor edges, reporting of the finger or pen is suppressed until it moves out of the zone into the central area of the sensor. This register specifies the size of the zone by giving the width of the border in millimeters expressed by 4.4 fixed-point format. Setting value 0 means that the rejection zone is disabled.

10.4.68. F11_2D_Ctrl72: z offset tolerance

ZOffsetTolerance (F11_2D_Ctrl72)

10.4.69. F11_2D_Ctrl73: jitter filter strength

This register exists only if *HasJitterFilter (F11_2D_Query5, bit 7)* reports as '1'.

JitterFilterStrength (F11_2D_Ctrl73, bits 3:0)

The default value is 8. Decreasing this will make the jitter filter less effective, but more sensitive to motion. Increasing it will make the jitter filter more effective, but more less to motion.

Reserved (F11_2D_Ctrl73, bits 7:4)

Reserved.

10.4.70. F11_2D_Ctrl74: jitter free run frames

This register exists only if *HasJitterFilter (F11_2D_Query5, bit 7)* reports as '1'.

JitterFreeRunFrames (F11_2D_Ctrl74)

This is the number of transitional frames from steady to frozen. The larger the value is, the longer it takes to "freeze" the jitter position.

10.4.71. F11_2D_Ctrl75.0: small jitter X threshold

This register exists only if *HasJitterFilter (F11_2D_Query5, bit 7)* reports as '1'.

SmallJitterXThreshold (F11_2D_Ctrl75.0, bits 4:0)

This is the X jitter threshold when finger Z equals *F11_2D_CtrlCC.02*.

LGMThresholdScale (F11_2D_Ctrl75.0, bits 7:5)

This value will be multiplied to small jitter thresholds in Low Ground Mass (LGM). The larger jitter thresholds will be multiplied based on the ratio.

10.4.72. F11_2D_Ctrl75.1: small jitter Y threshold

This register exists only if *HasJitterFilter (F11_2D_Query5, bit 7)* reports as '1'.

SmallJitterYThreshold (F11_2D_Ctrl75.1, bits 4:0)

This is the Y jitter threshold when finger Z equals *F11_2D_CtrlCC.02*.

Reserved (F11_2D_Ctrl75.1, bits 7:5)

Reserved.

10.4.73. F11_2D_Ctrl75.2: small jitter Z threshold

This register exists only if *HasJitterFilter (F11_2D_Query5, bit 7)* reports as '1'.

SmallJitterZThreshold (F11_2D_Ctrl75.2)

When finger Z equals this, X and Y thresholds can be found in *F11_2D_CtrlCC.0/1*.

10.4.74. F11_2D_Ctrl76.0: large jitter X threshold

This register exists only if *HasJitterFilter (F11_2D_Query5, bit 7)* reports as '1'.

LargeJitterXThreshold (F11_2D_Ctrl76.0)

This is the X jitter threshold when finger Z equals *F11_2D_CtrlCC.02*.

10.4.75. F11_2D_Ctrl76.1: large jitter Y threshold

This register exists only if *HasJitterFilter (F11_2D_Query5, bit 7)* reports as '1'.

LargeJitterYThreshold (F11_2D_Ctrl76.1)

This is the Y jitter threshold when finger Z equals *F11_2D_CtrlCC.02*.

10.4.76. F11_2D_Ctrl76.2: large jitter Z threshold

This register exists only if *HasJitterFilter (F11_2D_Query5, bit 7)* reports as '1'.

LargeJitterZThreshold (F11_2D_Ctrl76.2)

When finger Z equals this, X and Y thresholds can be found in *F11_2D_CtrlCC.0/1*.

10.4.77. F11_2D_Ctrl77: number of 2-D receivers

This register is present only if *Has2DAdjustableMapping* (F11_2D_Query12, bit 3) reports as '1'.

NumberOf2DReceivers (F11_2D_Ctrl77)

This register specifies the number of receiver electrodes assigned to the 2-D sensor. The range is 1 to *NumberOfReceiverElectrodes* (F54_AD_Query0).

10.4.78. F11_2D_Ctrl78: number of 2-D transmitters

This register is present only if *Has2DAdjustableMapping* (F11_2D_Query12, bit 3) reports as '1'.

NumberOf2DTransmitters (F11_2D_Ctrl78)

This register specifies the number of transmitter electrodes assigned to the 2-D sensor. The range is 1 to *NumberOfTransmitterElectrodes* (F54_AD_Query1).

10.4.79. F11_2D_Ctrl79: chiral scroll parameters 2

This register is present only if *HasChiralScroll* (F11_2D_Query7, bit 7) reports as '1'.

TriggerZoneWidth (F11_2D_Ctrl79, bits 4:0)

This defines the width, in millimeters, of the Chiral Trigger Zone around the perimeter of the sensor.

Reserved (F11_2D_Ctrl79, bits 7:5)

Reserved.

10.5. Function $11: data registers

Function $11's data registers are divided into three groups: one which contains finger status and absolute position data, one for relative motion data, and one for gesture data. Within each group, the registers are time-coherent (see section 2.6), but the registers in one group are not guaranteed to be coherent with those in either of the other two.

10.5.1. Data register layout

The exact register layout depends on the specific features, as well as the number of fingers supported by the sensor. The register map construction rules are applied as follows:

1. One or more F11_2D_Data0 registers is placed first. The size of this register block can be calculated from the decoded *NumberOfFingers* field in F11_2D_Query1 (see section 10.3.3):

   ```
   F11_2D_Data0 register count = ceil(decoded NumberOfFingers/4)
   ```

2. If the *HasAbsMode* field of F11_2D_Query1 reports as '1', then the block of data registers that describes the absolute finger position for a single finger (F11_2D_Data1 through Data5) is placed next. This block of registers is replicated once for each of the fingers that the sensor supports, as described by the *NumberOfFingers* field in F11_2D_Query1 (see section 10.3.3).
3. If the *HasRelMode* field of F11_2D_Query1 reports as '1', then the pair of data registers that describes the relative finger motion for a single finger (F11_2D_Data6, 7) is placed next. This block of registers is replicated once for each of the fingers that the sensor supports, as described by the *NumberOfFingers* field in F11_2D_Query1 (see section 10.3.3).
4. If F11_2D_Query7 is non-zero, then F11_2D_Data8 is placed next.
5. If F11_2D_Query7 or F11_2D_Query 8 is non-zero, then F11_2D_Data9 is placed next.
6. If the *HasPinch* or *HasFlick* field of F11_2D_Query7 reports as '1', then F11_2D_Data10 is placed next.
7. If the *HasRotate* field of F11_2D_Query8 or the *HasFlick* field of F11_2D_Query7 reports as '1', then both the F11_2D_Data11 and F11_2D_Data12 registers are placed next.
8. If the *HasTouchShapes* field of F11_2D_Query8 reports as '1', then one or more F11_2D_Data13 registers is placed next. The size of this register block can be calculated from the *NumberOfTouchShapes* field in F11_2D_Query10:

   ```
   F11_2D_Data13  register count = 1 + floor(NumberOfTouchShapes/8)
   ```

9. If the *HasScrollZones* field of F11_2D_Query8 reports as '1', then F11_2D_Data14 and F11_2D_Data15 are placed next. If the *IndividualScrollZones* field of F11_2D_Query8 reports as '1', then F11_2D_Data16 and F11_2D_Data17 are placed next.
10. If the *HasContactGeometry* field of F11_2D_Query9 reports as '1', then the block of data registers that describes the contact geometry for a single finger (F11_2D_Data18 through F11_2D_Data27) is placed next. This block of registers is replicated once for each of the fingers that the sensor supports, as described by the *NumberOfFingers* field in F11_2D_Query1 (see section 10.3.3).

A two-finger example with individual Scroll Zones, 12 TouchShapes, contact geometry, and bending correction is shown in Table 36.

Table 36. Function $11 data register example (two fingers, scroll zones, 12 TouchShapes, contact geometry, and bending correction)

Name	7	6	5	4	3	2	1	0
F11_2D_Data0.0	—		—		FingerState1		FingerState0	
F11_2D_Data1.0	XPosition [11:4]							
F11_2D_Data2.0	YPosition [11:4]							
F11_2D_Data3.0	YPosition [3:0]				XPosition [3:0]			
F11_2D_Data4.0	Wy [3:0]				Wx [3:0]			
F11_2D_Data5.0	Z [7:0]							
F11_2D_Data1.1	XPosition [11:4]							
F11_2D_Data2.1	YPosition [11:4]							
F11_2D_Data3.1	YPosition [3:0]				XPosition [3:0]			
F11_2D_Data4.1	Wy [3:0]				Wx [3:0]			
F11_2D_Data5.1	Z [7:0]							
F11_2D_Data6.0	XDelta							
F11_2D_Data7.0	YDelta							
F11_2D_Data6.1	XDelta							
F11_2D_Data7.1	YDelta							
F11_2D_Data8	Chiral Scroll	Pinch	Press	Flick	EarlyTap	Double Tap	TapAnd Hold	SingleTap
F11_2D_Data9	GestureFingerCount			Multi Finger Scroll	Scroll Zone	Shape	Rotate	Palm Detect
F11_2D_Data10	PinchMotion / XFlickDistance							
F11_2D_Data11	RotateMotion / YFlickDistance							
F11_2D_Data12	FingerSeparation / FlickTime							
F11_2D_Data13.0	Shape7	Shape6	Shape5	Shape4	Shape3	Shape2	Shape1	Shape0
F11_2D_Data13.1	—	—	—	—	Shape11	Shape10	Shape9	Shape8
F11_2D_Data14	ChiralScrollX							
F11_2D_Data15	ChiralScrollY							
F11_2D_Data16	XUpperScrollMotion							
F11_2D_Data17	YLeftScrollMotion							
F11_2D_Data18.0	ContactGeometryTarget X [11:4]							
F11_2D_Data19.0	ContactGeometryTarget Y [11:4]							
F11_2D_Data20.0	ContactGeometryTargetY [3:0]; ContactGeometryTargetX [3:0]							
F11_2D_Data21.0	ContactGeometryWidth [11:4]							
F11_2D_Data22.0	ContactGeometryHeight [11:4]							
F11_2D_Data23.0	ContactGeometryHeight [3:0]; ContactGeometryWidth [3:0]							
F11_2D_Data24.0	ContactGeometryAngle							
F11_2D_Data25.0	ContactGeometryCenterX [11:4]							
F11_2D_Data26.0	ContactGeometryCenterY [11:4]							
F11_2D_Data27.0	ContactGeometryCenterY [3:0]; ContactGeometryCenterX [3:0]							
F11_2D_Data18.1	ContactGeometryTarget X [11:4]							
F11_2D_Data19.01	ContactGeometryTarget Y [11:4]							

Name	7	6	5	4	3	2	1	0
F11_2D_Data20.1	ContactGeometryTargetY [3:0]; ContactGeometryTargetX [3:0]							
F11_2D_Data21.1	ContactGeometryWidth [11:4]							
F11_2D_Data22.1	ContactGeometryHeight [11:4]							
F11_2D_Data23.1	ContactGeometryHeight [3:0]; ContactGeometryWidth [3:0]							
F11_2D_Data24.1	ContactGeometryAngle							
F11_2D_Data25.1	ContactGeometryCenterX [11:4]							
F11_2D_Data26.1	ContactGeometryCenterY [11:4]							
F11_2D_Data27.1	ContactGeometryCenterY [3:0]; ContactGeometryCenterX [3:0]							
F11_2D_Data28	—						Large Object Present	Bending
F11_2D_Data29	CxyLSB							
F11_2D_Data30	CxyMSB							
F11_2D_Data31	PenZ							
F11_2D_Data32.0	FingerProfileAmplitudeLSB							
F11_2D_Data32.1	FingerProfileAmplitudeMSB							
F11_2D_Data33.0	FingerProfileX-axisCurvatureLSB							
F11_2D_Data33.1	FingerProfileX-axisCurvatureMSB							
F11_2D_Data34.0	FingerProfileY-axisCurvatureLSB							
F11_2D_Data34.1	FingerProfileY-axisCurvatureMSB							
F11_2D_Data35.0	WyMSB [7:4]				WxMSB [7:4]			
F11_2D_Data35.1	WyMSB [7:4]				WxMSB [7:4]			
F11_2D_Data36	BendingMetric							
...	...							
F11_2D_Data69	PenHoverHysteresis							

10.5.2. Finger reporting

For sensors that support more than one finger, the data registers associated with a finger are determined at the time that the finger lands on the sensor. The data register block assigned to reporting that finger remains constant until the finger leaves. For example, consider the following sequence of events:

1. No fingers are present on the sensor.
2. A single finger lands on the sensor. This finger is defined to be the *primary finger*, or Finger0. The primary finger is reported using the first available block of absolute data registers, for example, Data1.0 through Data5.0.
3. With the primary finger still present, a second finger lands on the sensor, Finger1. This second finger gets assigned to the next available block of absolute data registers, for example, Data1.1 through Data5.1.
4. If the original finger were to lift while the second finger were to remain pressed, the second finger would become the primary finger, but that finger would remain being reported in the data registers to which it had originally been assigned; in the example above, this means that the primary finger is now reported in the Data1.1 through Data5.1 register block.
5. If another finger were to land while the second finger remained pressed, the new finger would be reported in the first available block of absolute data registers.

6. If the sensor uses Relative mode, the F11_2D_Data6.* and Data7.* registers are used to track relative movement of the fingers.

10.5.3. F11_2D_Data0.*: finger status data

The size of this register block can be calculated from *NumberOfFingers* field in F11_2D_Query1 (see section 10.3.3). The number of F11_2D_Data0 registers can be calculated as:

```
F11_2D_Data0_registerCount = ceil(NumberOfFingers/4)
```

The F11_2D_Data0 registers encode a 2-bit status field for each finger supported by the sensor. The *FingerStateN* fields are assigned one per finger, starting from least-significant bit-pairs to most-significant bit-pairs. A Function $11 sensor always contains at least one F11_2D_Data0 register. If a sensor supports more than four fingers, a second F11_2D_Data0 register will be present containing the status information for the remaining fingers.

FingerStateN

Each finger has two status bits to describe its state. The encoding of the bits is:

‘00’ – The finger is not present.

‘01’ – The finger is present and the positional information associated with it is accurate.

‘10’ – The finger is present, however the positional information associated with it may be inaccurate.

‘11’ – This encoding is reserved for product-specific usage. Consult the product-specific documentation for more details.

If the device has stylus or proximity capability (either *HasPen* or *HasProximity* is set to ‘1’), *FingerStateN* has different definitions:

FingerStateN

Each finger has two status bits to describe its state. The encoding of the bits is:

‘00’ – Neither a finger nor a stylus is present.

‘01’ – A finger is touching the sensor.

‘10’ – A stylus is touching the sensor.

‘11’ – A finger is near the sensor, but not touching it.

10.5.4. F11_2D_Data1.*: x position data (MSB)

These data registers report the most-significant bits of the absolute X position data. A host that is interested in minimizing bus bandwidth can read these registers and obtain a rough estimate of the finger position. To obtain a fully accurate position report, read F11_2D_Data3 also.

10.5.5. F11_2D_Data2.*: Y position data (MSB)

These data registers report the most-significant bits of the absolute Y position data. A host that is interested in minimizing bus bandwidth can read these two registers and obtain a rough estimate of the finger position. To obtain a fully accurate position report, read F11_2D_Data3 also.

10.5.6. F11_2D_Data3.*: X and Y position data (LSB)

This register is only present if the *AbsDataSize* field of F11_2D_Query5 reports as ‘00’ (see section 10.3.6).

This data register contains the least-significant bits for both the X and Y absolute position information. In combination with F11_2D_Data1 and F11_2D_Data2, full 12-bit X and Y position reports can be constructed.

When no finger is present on the sensor, the X and Y positions report the last known valid finger position.

The fields in this register are defined as follows:

X3:0 (F11_2D_Data3., bits 3:0)*

In conjunction with F11_2D_Data1, the combined 12-bit field reports the horizontal position of the finger on the pad, where the origin is on the left side of the pad.

Y3:0 (F11_2D_Data3., bits 7:4)*

In conjunction with F11_2D_Data2, the combined 12-bit field reports the vertical position of the finger on the pad, where the origin is on the lowest side of the pad.

10.5.7. F11_2D_Data4.*: finger width (W) data

This register is only present if the AbsDataSize field of F11_2D_Query5 reports as '00' (see section 10.3.6). Wx and Wy report as '0' when a stylus is detected.

Wx (F11_2D_Data4., bits 3:0)*

Wy (F11_2D_Data4., bits 7:4)*

These fields report the estimated finger width as an unsigned integer, where 0 represents an extremely narrow finger and 15 represents an extremely wide contact such as a palm laid flat on the sensor. The ratio of *Wx* and *Wy* provides an estimate of the finger contact aspect ratio.

If the *FingerState* field for that finger corresponds to '10', then the W values for all the fingers on the sensor will be the same.

10.5.8. F11_2D_Data5.*: finger contact (Z) data

This register is only present if the AbsDataSize field of F11_2D_Query5 reports as '00' (see section 10.3.6). This register reports a fixed value when a stylus is detected.

Z (F11_2D_Data5.)*

This field reports the amount of finger contact or finger signal strength, which often serves as a rough estimate of finger pressure.

When Z = 0, the position cannot be measured and the X and Y Position registers are left unchanged. By default Z is taken as 0 whenever the device's built-in algorithms determine that no finger is present.

10.5.9. F11_2D_Data6.* and F11_2D_Data7.*: finger motion deltas

These registers are only present if the *HasRelMode* field of F11_2D_Query1 reports as '1' (see section 10.3.3). This block of registers is replicated once for each of the fingers that the sensor supports.

These registers report finger motion as signed, 8-bit values for each reported finger. The delta registers accumulate motion until the host reads the delta registers. The motion accumulators will clip to +127 or –128 if the delta motion exceeds the 8-bit range.

Important: The DeltaX and DeltaY registers *must* be read as a sequential pair. Reading these registers has the side effect of clearing them.

DeltaX (F11_2D_Data6.)*

This byte reports the amount of horizontal finger motion during a reporting period, where a positive *DeltaX* represents rightward motion (toward increasing absolute X positions).

DeltaY (F11_2D_Data7.)*

This byte reports the amount of vertical finger motion during a reporting period, where a positive *DeltaY* represents upward motion (toward increasing absolute Y positions).

10.5.10. F11_2D_Data8 and F11_2D_Data9: gesture data

F11_2D_Data8 is only present if the F11_2D_Query7 register is non-zero. F11_2D_Data9 is only present if either the F11_2D_Query7 or the F11_2D_Query8 registers are non-zero.

For a definition of each of the gestures referenced below, see the documentation for the corresponding query register F11_2D_Query1 (see section 10.3.3).

When a sensor recognizes a gesture, the appropriate gesture bit is set in one of these data registers. If the corresponding gesture interrupt enable bit is set (see section 10.4.7), an ATTN interrupt is generated. The ATTN interrupt must be processed in a timely fashion by the host, especially if the host desires to correlate the gesture with a position.

For example, if the host is interested in correlating a *single-tap* gesture with its corresponding absolute finger position, the host should read the position registers with minimal latency after detecting the tap. If the host is too slow to respond, a finger may arrive again at some other location, and the host will get a false indication of the position where the tap occurred.

The fields in these data registers are defined as follows:

SingleTap (F11_2D_Data8, bit 0)

If this bit is set, then a single-tap gesture has completed.
This bit is automatically cleared when it is read.

TapAndHold (F11_2D_Data8, bit 1)

If this bit is set, then a tap-and-hold gesture has completed.
This bit is automatically cleared when it is read.

DoubleTap (F11_2D_Data8, bit 2)

If this bit is set, then a double-tap gesture has completed.
This bit is automatically cleared when it is read.

EarlyTap (F11_2D_Data8, bit 3)

If this bit is set, then an early-tap gesture has been detected. This bit is automatically cleared when it is read.

An early-tap gesture is identical to a single-tap gesture except that the gesture is complete as soon as the finger lifts. In contrast, a single-tap gesture is not completed until enough finger-up time has passed to be sure that the tap is not part of a tap-and-hold or double-tap gesture.

Flick (F11_2D_Data8, bit 4)

If this bit is set, then a flick gesture has completed. This bit is automatically cleared when it is read. The X/Y distance and time associated with the flick gesture are reported in F11_2D_Data10 through F11_2D_Data12.

Press (F11_2D_Data8, bit 5)

If this bit is set, then a press gesture is active.

Note: This bit is not automatically cleared when it is read; it will remain set to 1 until the finger lifts or moves more than the maximum allowed press distance.

Pinch (F11_2D_Data8, bit 6)

If this bit is set, then a pinch gesture is either in progress or has completed.
This bit is automatically cleared when it is read, but it will be set again if the pinch gesture continues.

The change in distance between the two fingers is reported in F11_2D_Data10.

ChiralScroll (F11_2D_Data8, bit 7)

When this bit reports as '1', a chiral scroll gesture is either in progress or has completed. This bit is automatically cleared to '0' whenever the register is read, but it will report '1' again if the chiral scroll gesture continues.

PalmDetect (F11_2D_Data9, bit 0)

If this bit is set, then the palm detection gesture is active. A *palm* is defined to be any 'large' conductive object touching the 2-D sensor. An object is considered *large* if its width in either the X or Y axis exceeds the *PalmDetectThreshold* (F11_2D_Ctrl1, bits 3:0).

Notes:

- This bit is *not* automatically cleared when it is read; it will remain set to 1 until the palm lifts from the sensor.
- Most data reporting is inhibited if a palm is detected. This gesture notifies the host when this condition persists.

Rotate (F11_2D_Data9, bit 1)

If this bit is set, then a rotate gesture is either in progress or has completed. This bit is automatically cleared when it is read, but it will be set again if the rotate gesture continues.

The accumulated finger motion distance is reported in F11_2D_Data11 and instantaneous relative finger separation is reported in F11_2D_Data12.

Shape (F11_2D_Data9, bit 2)

If this bit is set, then a finger has either touched or lifted from a TouchShape. The touched/lifted status of each TouchShape is reported in the F11_2D_Data13.* registers.

Note: This bit is automatically cleared when it is read, but the F11_2D_Data13.* registers are unaffected by reading this bit.

ScrollZone (F11_2D_Data9, bit 3)

If this bit is set, then finger motion in a ScrollZone is either in progress or has completed.
This bit is automatically cleared when it is read, but it will be set again if the motion continues.

The accumulated finger motion distance is reported in F11_2D_Data14 through F11_2D_Data17.

MultiFingerScroll (F11_2D_Data9, bit 4)

If this bit is set, then a multi-finger scroll gesture is either in progress or complete. The accumulated finger motion distance is reported in F11_2D_Data14 and F11_2D_Data15.

Check the *GestureFingerCount (F11_2D_Data9, bits 7:5)* field to determine how many fingers were present when this gesture was detected.

GestureFingerCount (F11_2D_Data9, bits 7:5)

This field reports the actual number of fingers used by the reported gesture.

Table 37. Number of fingers used by the reported gesture

Encoding	Number of Fingers Reported
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	*Reserved*

For example, a two-finger press gesture would set the *Press* bit to '1', and also set the *GestureFingerCount* field to '1' (indicating two fingers present). A single-finger flick would set the Flick bit to '1', and also set the *GestureFingerCount* to '0' (indicating one finger).

10.5.11. F11_2D_Data10: pinch motion and X flick distance

F11_2D_Data10 is shared by pinch and flick gestures. This register is present if *HasPinch* field or the *HasFlick* field of F11_2D_Query7 reports as '1'.

- When the *Pinch* bit of F11_2D_ Data8 is set to '1', this register reports the change in distance between the two fingers since this register was last read. Negative values are reported when the fingers are moving closer together; positive values are reported when the fingers are moving apart. Units are in millimeters.
- When the *Flick* bit of F11_2D_ Data8 is set to '1', this register reports the distance of flick gesture in X direction. Negative values are reported when the finger moves in negative X direction; positive values are reported when the finger moves in positive X direction. Units are in millimeters.

Note: This register is cleared to '0' when read.

10.5.12. F11_2D_Data11: rotate motion and Y flick distance

F11_2D_Data11 is shared by the rotate and flick gestures. This register is present if the *HasFlick* field of F11_2D_Query7 or the *HasRotate* field of F11_2D_Query8 reports as '1'.

- When the *Flick* bit of F11_2D_ Data8 is set to '1', this register reports the distance of the flick gesture in Y direction. Negative values are reported when the finger moves in negative Y direction; positive values are reported when the finger moves in positive Y direction.
- When the *Rotate* bit of F11_2D_Data9 is set to '1', this register reports the accumulated distance of the rotate motion. The register is a signed quantity; clockwise motion is positive and counterclockwise motion is negative.

Notes:

- This register is cleared to ‘0’ when read.
- Units are in millimeters.

10.5.13. F11_2D_Data12: finger separation and flick time

This register is only present if the *HasFlick* field of F11_2D_Query7 reports as ‘1’ or the *HasRotate* field of F11_2D_Query8 reports as ‘1’. When the *Flick* bit of F11_2D_ Data8 is set to ‘1’, this register reports the total time of the flick gesture. Flick speed can be calculated in conjunction with X/ Y flick distance. Units are in 10-millisecond increments for flick and in millimeters for rotate.

Note: This register is cleared to ‘0’ when read.

10.5.14. F11_2D_Data13.*: TouchShape status

These registers are only present if the *HasTouchShapes* field of F11_2D_Query8 reports as ‘1’. Each bit of these registers reports the state of a TouchShape: 0 = not touched, 1 = touched.

10.5.15. F11_2D_Data14 and F11_2D_Data15: chiral scroll

When *ChiralStartMode* (F11_2D_Ctrl59 bit 7) is set to ‘1’, these two registers report chiral scrolling distances in the X and Y directions. When *ChiralStartMode* (F11_2D_Ctrl59 bit 7) is set to ‘0’, chiral scrolling distance is reported in *ChiralScrollX* while *ChiralScrollY* always reports 0x00. These registers are automatically cleared to zero after they are read.

10.5.16. F11_2D_Data16: x upper scroll motion

This register is only present if the *HasScrollZones* field of F11_2D_Query8 reports as ‘1’ and the *IndividualScrollZones* field of F11_2D_Query8 reports as ‘1’. When the *ScrollZone* bit of F11_2D_ Data8 is set to ‘1’, this register reports the distance of the motion on the “X Upper” ScrollZone. Negative values are reported when the finger moves in negative X direction; positive values are reported when the finger moves in positive X direction.

Note: This register is cleared to ‘0’ when read.

10.5.17. F11_2D_Data17: y left scroll motion

This register is only present if the *HasScrollZones* field of F11_2D_Query8 reports as ‘1’ and the *IndividualScrollZones* field of F11_2D_Query8 reports as ‘1’. When the *ScrollZone* bit of F11_2D_ Data8 is set to ‘1’, this register reports the distance of the motion on the “Y Left” ScrollZone. Negative values are reported when the finger moves in negative X direction; positive values are reported when the finger moves in positive Y direction.

Note: This register is cleared to ‘0’ when read.

10.5.18. F11_2D_Data18.*: contact geometry X target position (MSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as ‘1’.

ContactGeometryTargetX (F11_2D_Data18.)*

This data register reports bits 11:4 of the 12-bit contact geometry absolute X target position. If no finger is present on the sensor, the last known valid absolute X target position is reported.

10.5.19. F11_2D_Data19.*: contact geometry Y target positions (MSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

ContactGeometryTargetY (F11_2D_Data19.)*

This data register reports bits 11:4 of the 12-bit contact geometry absolute Y target position. If no finger is present on the sensor, the last known valid absolute Y target position is reported.

10.5.20. F11_2D_Data20.*: contact geometry X and Y target positions (LSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

This data register reports bits 3:0 of the 12-bit c contact geometry absolute X and Y target positions. If no finger is present on the sensor, the last known valid absolute X and Y target positions are reported.

The fields in this register are defined as follows:

- *TargetX3:0 (F11_2D_Data20, bits3:0)*
  Bits 3:0 of the absolute X (horizontal) contact geometry target position of the finger on the sensor.
- *TargetY3:0 (F11_2D_Data20, bits7:4)*
  Bits 3:0 of the absolute Y (vertical) contact geometry target position of the finger on the sensor.

10.5.21. F11_2D_Data21.*: contact geometry width (MSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

ContactGeometryWidth (F11_2D_Data21.)*

This data register reports bits 11:4 of the 12-bit contact geometry width. If no finger is present on the sensor, the last known valid width is reported.

10.5.22. F11_2D_Data22.*: contact geometry height (MSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

ContactGeometryHeight (F11_2D_Data22.)*

This data register reports bits 11:4 of the 12-bit contact geometry height. If no finger is present on the sensor, the last known valid height is reported.

10.5.23. F11_2D_Data23.*: contact geometry width and height (LSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

This data register reports bits 3:0 of the 12-bit contact geometry width and height. If no finger is present on the sensor, the last known valid width and height are reported.

The fields in this register are defined as follows:

- *WidthX3:0 (F11_2D_Data23, bits 3:0)*
  Bits 3:0 of the horizontal contact geometry width of the finger on the sensor.
- *HeightY3:0 (F11_2D_Data23, bits 7:4)*
  Bits 3:0 of the vertical contact geometry height of the finger on the sensor.

10.5.24. F11_2D_Data24.*: contact geometry angle

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

This data register reports the contact geometry angle of the finger on the sensor, in 2-degree units. For example, 32 degrees would be reported as '16'. The range is 0 to 179 (0 to 358 degrees). If no finger is present on the sensor, the last known valid angle is reported.

10.5.25. F11_2D_Data25.*: contact geometry X center position (MSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

ContactGeometryCenterX (F11_2D_Data25.)*

This data register reports bits 11:4 of the 12-bit contact geometry absolute X center position. If no finger is present on the sensor, the last known valid absolute X center position is reported.

10.5.26. F11_2D_Data26.*: contact geometry Y center position (MSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

ContactGeometryCenterY (F11_2D_Data26.)*

This data register reports bits 11:4 of the 12-bit contact geometry absolute Y center position. If no finger is present on the sensor, the last known valid absolute Y center position is reported.

10.5.27. F11_2D_Data27.*: contact geometry X and Y center positions (LSB)

This register is only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

This data register reports bits 3:0 of the 12-bit contact geometry absolute X and Y center positions. If no finger is present on the sensor, the last known valid absolute X and Y center positions are reported.

The fields in this register are defined as follows:

CenterX3:0 (F11_2D_Data27., bits 3:0)*

Bits 3:0 of the absolute X (horizontal) contact geometry center position of the finger on the sensor.

CenterY3:0 (F11_2D_Data27., bits 7:4)*

Bits 3:0 of the absolute Y (vertical) contact geometry center position of the finger on the sensor.

10.5.28. F11_2D_Data28: extended status

This register is present only if *HasBendingCorrection* (F11_2D_Query5, bit 5)) reports as '1'or *HasLargeObjectCorrection* (F11_2D_Query5, bit 6) reports as '1'.

Bending (F11_2D_Data28, bit 0)

When this bit reports as '1', physical bending of the sensor has been detected and is being corrected.

LargeObjectPresent (F11_2D_Data28, bit 1)

When this bit reports as '1', a large object has been detected on the sensor and is being ignored.

Reserved (F11_2D_Data28, bits 7:2)

Reserved.

10.5.29. F11_2D_Data29: Cxy LSB

CxyLSB (F11_2D_Data29)

10.5.30. F11_2D_Data30: Cxy MSB

CxyMSB (F11_2D_Data30)

10.5.31. F11_2D_Data31: pen Z

PenZ (F11_2D_Data31)

10.5.32. F11_2D_Data32.0/32.1: finger profile amplitude

These registers only exist when *HasGaplessFingerTuning* (F11_2D_Query12, bit 1) reports as '1' and *HasGaplessFinge*r (F11_2D_Query12, bit 0) reports as '1'.

FingerProfileAmplitudeLSB (F11_2D_Data32.0)

FingerProfileAmplitudeMSB (F11_2D_Data32.1)

While a finger is detected on the sensor, these registers report the amplitude of the unprojected Gaussian-fitted finger profile. When the finger is lifted from the sensor, these registers continue to hold the last-recorded value.

10.5.33. F11_2D_Data33.0/33.1: finger profile X-axis curvature

These registers only exist when *HasGaplessFingerTuning* (F11_2D_Query12, bit 1) reports as '1' and *HasGaplessFinge*r (F11_2D_Query12, bit 0) reports as '1'.

FingerProfileX-axisCurvatureLSB (F11_2D_Data33.0)

FingerProfileX-axisCurvatureMSB (F11_2D_Data33.1)

While a finger is detected on the sensor, these registers report the x-axis curvature of the unprojected Gaussian-fitted finger profile in the fixed-point 8.8 format. When the finger is lifted from the sensor, these registers continue to hold the last-recorded value.

10.5.34. F11_2D_Data34.0/34.1: finger profile Y-axis curvature

These registers only exist when *HasGaplessFingerTuning* (F11_2D_Query12, bit 1) reports as '1' and *HasGaplessFinge*r (F11_2D_Query12, bit 0) reports as '1'.

FingerProfileY-axisCurvatureLSB (F11_2D_Data34.0)

FingerProfileY-axisCurvatureMSB (F11_2D_Data34.1)

While a finger is detected on the sensor, these registers report the y-axis curvature of the unprojected Gaussian-fitted finger profile in the fixed-point 8.8 format. When the finger is lifted from the sensor, these registers continue to hold the last-recorded value.

10.5.35. F11_2D_Data35.* finger width MSB

These registers only exist when *Has8-bitW* (F11_2D_Query12 bit 2) reports as '1'.

WxMSB (F11_2D_Data35., bits 3:0)*

WyMSB (F11_2D_Data35., bits 7:4)*

10.5.36. F11_2D_Data36: bending metric

This register is only present if *HasBendingCorrection* (F11_2D_Query5, bit 5) reports as '1'.

BendingMetric (F11_2D_Data36)

Reports the strongest bending or finger-presence signal on the sensor.

10.5.37. F11_2D_Data37 through F11_2D_Data68: reserved

10.5.38. F11_2D_Data69: pen hover hysteresis

These registers only exist when *HasPenHoverDiscrimination* (F11_2D_Query9 bit 6) reports as '1' and *HasPen* (F11_2D_Query9, bit 0) reports as '1'.

PenHoverHysteresis (F11_2D_Data69)

Once a pen is reported, an extra delta Cxy offset percentage in the negative direction is allowed at the demarcation of pen and hovering decision. The maximum percentage is 99. Setting 0 means that no hysteresis is allowed.

10.6. Function $11: interrupt source

Function $11 defines an interrupt source for each data source that it contains. For example, an RMI device that contains a Function $11 absolute 2-D data source and a gesture source, but no 2-D relative data source, will contain an interrupt source for the absolute and gesture data sources, but not for the relative data source.

The absolute data source of a 2-D sensor asserts an ATTN interrupt request on every report period where the interrupt conditions are satisfied by the current *ReportingMode* setting in control register F11_2D_Ctrl0 (see section 10.4.1). The reporting modes are defined to span a range of application requirements from very low reporting rates to very high reporting rates.

The relative data source of a 2-D sensor asserts an ATTN interrupt request on every report period that adds a non-zero amount of motion to either the X or Y delta motion accumulator registers.

Note: Because interrupt request bits are 'sticky', an interrupt request will remain at '1' even if a later backward finger motion subtracts the deltas down to zero again by the time the host reads the deltas; therefore, the host should be prepared occasionally to see ($00, $00) deltas even when an interrupt request is reported.

The gesture data source of a 2-D sensor asserts an interrupt request during every report period when a gesture has been either been detected (as in the case of pinch, press, or palm detect), or has been completed (as in the case of early-tap, single-tap, tap-and-hold, double-tap, and flick). The gesture source shares an interrupt request bit and an interrupt enable bit with the absolute data source.

Note: To clear interrupts caused by the gesture data source, the gesture flags registers F11_2D_Data8 and F11_2D_Data9 must be read in addition to reading the device Interrupt Status register.

10.7. Function $11: command registers

Function $11 has the following command register defined.

Table 38. Function $11 command register

Name	7	6	5	4	3	2	1	0
F11_2D_Cmd0	—	—	—	—	—	—	—	ReZero

ReZero (F11_2D_Cmd0, bit 0)

Writing a '1' to this field causes all 2-D sensors to revert to their "not touched" state. The host should never need to issue a Rezero command under normal conditions, because Synaptics devices handle zeroing completely automatically.

11. Function $19: 0-D capacitive button sensors

Function $19 implements a 0-D sensing function, typically known as a capacitive button.

11.1. Function $19: query registers

Query registers 0 and 1 contains general query information regarding Function $19, button sensing.

Table 39. Function $19 query registers

Name	7	6	5	4	3	2	1	0
F19_Btn_Query0	—	—	—	—	—	Has Hysteresis Threshold	Has Sensitivity Adjust	Configurable
F19_Btn_Query1	—	—	—	ButtonCount				

11.1.1. F19_Btn_Query0: button features

The bits in register F19_Btn_Query0 are defined as follows:

Configurable (F19_Btn_Query0, bit 0)

If this field is ‘0’, the sensor mapping control registers are not configurable at run-time:

- The *ButtonCount* field represents the exact number of buttons that exist in the product. The sensor map control registers are read-only.

If this field is ‘1’, the button enables and sensor mappings are configurable at run-time:

- The *ButtonCount* field represents the maximum number of buttons that can be enabled at once. Button numbers will always be in the range [0 through (*ButtonCount*-1)].
- The sensor map control registers are read-write.

HasSensitivityAdjust (F19_Btn_Query0, bit 1)

If this field is ‘1’, the all-button sensitivity adjustment register F19_Btn_Ctrl5 exists.

HasHysteresisThreshold (F19_Btn_Query0, bit 2)

If this field is ‘1’, the hysteresis adjustment register F19_Btn_Ctrl6 exists.

Reserved (F19_Btn_Query0, bits 7:3)

Reserved.

11.1.2. F19_Btn_Query1: button count query

The bits in register F19_Btn_Query1 are defined as follows:

ButtonCount (F19_Btn_Query1, bits 4:0)

For configurable products (*Configurable* reports as ‘1’), the *ButtonCount* represents the maximum number of buttons that can be enabled at one time. For non-configurable products (*Configurable* reports as ‘0’), the *ButtonCount* represents the exact number of buttons that are supported in the product.

11.2. Function $19: control registers

These registers control the operation of the capacitive buttons.

Table 40. Function $19 control registers for an example of ButtonCount=4

Name	7	6	5	4	3	2	1	0
F19_Btn_Ctrl0	—	—	—	—	FilterMode		ButtonUsage	
F19_Btn_Ctrl1.0	—	—	—	—	IntEnBtn3	IntEnBtn2	IntEnBtn1	IntEnBtn0
F19_Btn_Ctrl2.0	—	—	—	—	SingleBtn3	SingleBtn2	SingleBtn1	Single Btn0
F19_Btn_Ctrl3.0	—	SensorMap_Btn0						
F19_Btn_Ctrl3.1	—	SensorMap_Btn1						
F19_Btn_Ctrl3.2	—	SensorMap_Btn2						
F19_Btn_Ctrl3.3	—	SensorMap_Btn3						
F19_Btn_Ctrl4.*	—							
F19_Btn_Ctrl5	—	—	—	SensitivityAdjustment				
F19_Btn_Ctrl6	—	—	—	—	HysteresisThreshold			

The table above shows a view of the control registers that would be present if the device reported a *ButtonCount* of 4. F19_Btn_Ctrl1, F19_Btn_Ctrl2, F19_Btn_Ctrl3, and F19_Btn_Ctrl4 are all replicated registers. This means that the number of these registers varies, depending upon how many buttons are defined for this product. For example, every product contains at least one button interrupt enable control register. The button enable bits are mapped into the button enable registers such that bit 0 of the first button enable register controls the enabling of button0, bit 1 controls button 1, and so on, with 8 button controls per register.

A product with 11 buttons would require two button interrupt enable control registers: F19_Btn_Ctrl1.0 and F19_Btn_Ctrl1.1.

11.2.1. Calculating the number of control registers

The layout of the F$19 Button control registers always follows the general form: one general control register, followed by some number of button enable control registers, followed by some number of single button participation registers, followed by some number of sensor map control registers, followed by a single InterferenceThreshold control register, and ending with a single diagnostic control register. The number of button interrupt enable control registers can be calculated from the *ButtonCount* query:

```
F19_ButtonInterruptEnableRegisterCount = trunc((ButtonCount + 7) / 8);
```

The number of single button participation control registers is always identical to the number of button enable control registers:

```
F19_ButtonParticipationRegisterCount = F19_ButtonEnableRegisterCount;
```

The number of sensor map control registers (F19_Btn_Ctrl3 through F19_Btn_Ctrl6 in the example above) is always identical to *ButtonCount*:

```
F19_SensorMapRegisterCount = ButtonCount
```

The number of *Reserved* control registers (F19_Btn_Ctrl6 through F19_Btn_Ctrl10 in the example above) is always identical to *ButtonCount*.

11.2.2. F19_Btn_Ctrl0: general control

The fields in this control register are defined as follows:

ButtonUsage (F19_Btn_Ctrl0, bits 1:0)

This 2-bit field provides guidance about how the capacitive buttons are expected to be used in typical operation of the device. The reset default is '00', *Unrestricted usage*. For devices with only one capacitive button, the *ButtonUsage* field is effectively ignored.

ButtonUsage = '00': Unrestricted usage.

This value indicates that the user may touch the buttons in any combination. If the user is touching multiple buttons, all of those buttons are reported.

ButtonUsage = '01': Reserved.

ButtonUsage = '10': Strongest button only

This 'single button reporting mode' indicates that the user is expected to touch only one capacitive button at a time. In this mode, only one of the buttons in the participation group can report as being pressed at any one time. If the finger is proximate to several buttons at the same time, the device will attempt to choose the button with the strongest finger signal.

ButtonUsage = '11': First button only.

This 'single button reporting mode' indicates that the user is expected to touch only one capacitive button at a time. In this mode, only one of the buttons in the participation group can report as being pressed at any one time. If the finger is proximate to several buttons at the same time, the device will attempt to choose the first button that was touched for as long as that button remains touched.

If the first button touched is lifted while other buttons are still touched, it is undefined which of the still-touched buttons will be reported next.

Button usages '10' and '11' are advisory in the sense that some devices might not fully implement the "strongest button" or "first button" rules. In such devices, button usages '10' and '11' may be treated the same. However, all Function $19 devices must ensure that no more than one button data bit is reported as '1' at the same time whenever the Button Usage field is set to '10' or '11'.

FilterMode (F19_Btn_Ctrl0, bit 2)

Capacitive buttons are always filtered to reduce the effects of electrical noise. Depending on the specific noise environment for a product, different filters may provide improved usability tradeoffs.

FilterMode = '00': Standard Filter

The majority of button products should use the standard filter. This filter is designed to produce the best balance of button responsiveness versus noise rejection.

FilterMode = '01': High Noise Rejection Filter

For systems subject to unusually high levels of electrical noise, this filter setting implements higher levels of noise rejection, but at the cost of increasing the time it takes to register button presses and releases, and decreasing the ability to detect fast button tap events. It should only be used when the standard filter mode is not suitable.

FilterMode = ’10’, ‘11’:

These filter modes are *reserved* for device-specific filtering choices. They may not be present in all RMI devices. Consult the product specification for more information.

11.2.3. F19_Btn_Ctrl1.*: button interrupt enable control

Every product contains at least one button interrupt enable control register. The exact number of these registers is product-specific. See section 11.2.1 for details on calculating the total number of control registers in a product that contains RMI F$19 Buttons.

The buttons in a product are numbered starting at 0, and continuing up to (*ButtonCount*-1). The button enable bits are mapped into the button enable registers such that bit 0 of the first button enable register controls the enabling of button0, bit 1 controls button1, and so on. If there are more than 8 buttons, then bit 0 of the next sequential button enable register controls button8, and so on.

Assigning a ‘1’ to a button interrupt enable bit means that if the corresponding bit in the button data register changes state, the F$19 button interrupt source (see section 11.4) generates an ATTN interrupt. Assigning a ‘0’ to a button interrupt enable bit means that the corresponding bit in the button data register is ignored as part of the F$19 ATTN interrupt generation process. This mechanism allows a host to do things like put the system into a low power mode where all buttons are ignored except for a single button that is defined to generate a wake-up ATTN interrupt.

11.2.4. F19_Btn_Ctrl2.*: single button participation control

Button modes ‘10’ (strongest button mode) and ‘11’ (first button mode) are referred to as *single button reporting modes* in that only one button is reported from a set of buttons that might be pressed at the same time. By default, the single button reporting modes select one button from among every possible button on the device. Certain products may find it advantageous to exclude certain buttons when calculating the result of a single button reporting mode. For example, a device might support buttons that allow a user to select from a class of related, but mutually exclusive operations like ‘fast forward’, ‘rewind’, ‘pause’, and ‘play’. It would be natural to allow a user to select only one of these buttons at a time.

However, the same device might also contain a ‘mute’ or ‘wireless’ button. From a human interface point of view, a user should be allowed to press the ‘wireless’ or ‘mute’ buttons at the same time as they are holding down the ‘fast forward’ button. In that case, it would be desirable to exclude the ‘wireless’ and ‘mute’ buttons from the set of buttons participating in the single button reporting modes.

The button participation control register allows a host to select which buttons are excluded from consideration in the single button reporting selection process. Assigning a ‘1’ to a button bit in this register means that the corresponding button is excluded from participating in the single button reporting modes, so the true state of the button will always be reported in the corresponding bit in a button data register. Assigning a ‘0’ to a button bit in this register indicates that button reports for that bit are subject to the outcome of the single button reporting modes.

Every product contains at least one button participation control register. The number of registers and the numbering of the bits within the registers is always identical to the button enable control registers.

11.2.5. F19_Btn_Ctrl3.*: sensor map control

Every product contains at least one sensor map control register. The exact number of these registers is product-specific. See section 11.2.1 for details on calculating the total number of control registers in a product that contains RMI Function $19 buttons.

The buttons in a product are numbered from 0 to *ButtonCount*-1. The number of sensor map control registers is always identical to *ButtonCount*. The set of sensor map registers is laid out such that the first sensor map control register applies to button 0, the next sensor map control register applies to button 1, and so on, up to the final sensor map control register which applies to button *ButtonCount*-1.

Each of the sensor map control registers conforms to the following layout:

SensorMap (bits 4:0)

This 5-bit field maps a particular sensor electrode to a particular capacitive button. Writing a value *N* to this field in a particular control register maps electrode *N* to the button associated with that control register. The numbering of the electrodes is based on the particular touch controller package. It is an undefined operation to assign a button to an electrode that does not exist.

If the *Configurable* bit reports as '1', the host can arbitrarily assign any electrode to a button by writing the electrode number into this field.

If the *Configurable* bit reports as '0', the mapping between electrodes and buttons is fixed and cannot be changed; the mapping registers are read-only.

Regardless of the state of the *Configurable* field, the default values for the sensor mapping control registers are always product-specific; consult the product's documentation for more details.

The example register map in Table 41 assumes a product that has 4 buttons and 28 touch control sensors, S0 through S27.

Table 41. Function $19 sensor map control registers for an example of ButtonCount=4

Name	7	6	5	4	3	2	1	0
F19_Btn_Ctrl0	—	—	—	—	—	—	ButtonUsage='10'	
F19_Btn_Ctrl1.*	—	—	—	—	IntEn Btn3=0	IntEn Btn2=1	IntEn Btn1=1	IntEn Btn0=1
F19_Btn_Ctrl2.*	—	—	—	—	Single Btn3=0	Single Btn2=0	Single Btn1=1	Single Btn0=1
F19_Btn_Ctrl3.0	—			SensorMap_Btn0=15				
F19_Btn_Ctrl3.1	—			SensorMap_Btn1=27				
F19_Btn_Ctrl3.2	—			SensorMap_Btn2=0				
F19_Btn_Ctrl3.3	—			SensorMap_Btn3=6				

In Table 41, the first sensor map control register, at F19_Btn_Ctrl3.0 corresponds to button0. The *SensorMap* field in that register contains the value 15, meaning that button0 is associated with electrode S15.

In the same fashion, button1 is associated with electrode S27, button2 is associated with electrode S0, and button3 is associated with electrode S6.

The example register map also indicates that the interrupt for button3 is '0' (disabled), so events on button3 will not generate interrupts, although the state of Button3 will still be reported in the data register.

For disabled buttons, the *SensorMap* value in the corresponding sensor map control register is unimportant. The *ButtonUsage* field is '10', representing the "strongest button" mode.

Register F19_Btn_Ctrl2 indicates that button0 and button1 are participating in the single button modes. If a user simultaneously touches all four buttons on this device:

- Either button0 or button1 is reported as being pressed, depending on which has the strongest signal.
- Button2 reports as being pressed because it is not part of the single button participation group.
- Button3 still reports as being pressed, even though its corresponding interrupt bit is not enabled.

11.2.6. F19_Btn_Ctrl4.*: reserved

These registers are reserved for Synaptics use. Do not alter the contents of these registers when updating the device flash configuration.

11.2.7. F19_Btn_Ctrl5: all-button sensitivity adjustment

The fields in this control register are defined as follows:

SensitivityAdjustment (F19_Btn_Ctrl5, bits 4:0)

This 5-bit field provides a global sensitivity adjustment to the contacting and releasing thresholds applicable to all buttons. The value is signed, from -16 to 15, with -16 the least sensitive threshold for all buttons and +15 the most sensitive threshold for all buttons.
The default adjustment is 0.

11.2.8. F19_Btn_Ctrl6: all-button hysteresis threshold

The fields in this control register are defined as follows:

HysteresisThreshold (F19_Btn_Ctrl6, bits 3:0)

This 4-bit field introduces hysteresis to button reporting when operating in the *Strongest Button Mode*. In that 'single button reporting mode', only the button with the strongest finger signal is reported at any one time. By applying *Hysteresis Threshold*, a new strongest button is selected only if it is stronger by the threshold amount than the currently reported one (or, the strength of the currently reported one falls to be less than another button by the threshold amount).

This mechanism avoids fast alternative reporting among multiple fingers when they have a close strength of signal. *HysteresisThreshold* ranges from 0 to 15, where 0 means disabled.
The default threshold is 0.

11.3. Function $19: data registers

Function $19 always has one data source.

Each bit in the capacitive button data registers is ‘1’ if the button is being touched or ‘0’ if the button is not being touched.

11.3.1. Calculating the number of data registers

The number of button-reporting data registers depends on the *ButtonCount* query:

```
F19_DataRegisterCount = trunc((ButtonCount + 7) / 8)
```

The data registers of a capacitive button sensor are shown in Table 42, with the example of a *ButtonCount* of 12.

Table 42. Function $19 capacitive button data registers (for example of 12 buttons)

Name	7	6	5	4	3	2	1	0
F19_Btn_Data0.0	Btn7	Btn6	Btn5	Btn4	Btn3	Btn2	Btn1	Btn0
F19_Btn_Data0.1	—	—	—	—	Btn11	Btn10	Btn9	Btn8

11.4. Function $19: interrupt source

Function $19 implements one interrupt source. Function $19 asserts an interrupt request whenever any button bit in any of its button data registers changes from a ‘0’ to a ‘1’ or from a ‘1’ to a ‘0’, and the corresponding button interrupt enable bit is a ‘1’. Because an interrupt request is a “sticky” bit, interrupt requests read as ‘1’ if any button bit has changed since the last time the data registers were read, even if the button bit has changed back to its previous value since that time. An interrupt request is asserted synchronously with the report rate of other capacitive sensors in the device.

11.5. Function $19: command registers

Function $19 implements a single command register.

Table 43. Function $19 command register

Name	7	6	5	4	3	2	1	0
F19_RMI_Cmd0	—							Rezero Command

RezeroCommand (F19_Btn_Cmd0, bit 0)

Writing a ‘1’ to this field causes all button sensors to revert to their “not touched” state. The host should never need to issue a Rezero command under normal conditions, because Synaptics devices handle zeroing completely automatically.

Reserved (F19_Btn_Cmd0, bits 7:1)

Reserved.

12. Function $30: GPIO/LED and Mechanical Buttons

Function $30 implements a group of general purpose input/output pins, as might be used for input buttons, LED control and so on. Each pin is configured as either an input/output (GPIO) or an adjustable-current output (LED). The adjustable-current LED pins can be programmed for a variety of waveforms. Although an adjustable-current output is referred to as an LED, there is nothing requiring the controlled component to be a light-emitting diode; in fact, simple LEDs that do not require accurate or current-limited brightness control may be implemented by the host as a GPIO output.

The same pin can be configured and controlled as either a GPIO or LED, which is the flexibility needed for a general-purpose input-output device. A custom product typically defaults each pin as an LED, a GPI, or a GPO depending on the pin's intended purpose, but the flexibility to change it remains.

12.1. Function $30: power management

Function $30 interacts with the power management features described in section 4.2.1. The device does not doze when any LED is active or when any LED is ramping down.

12.2. Function $30: query registers

Query registers 0 and 1 contain general query information regarding Function $30, GPIO/LED.

Table 44. Function $30 GPIO/LED query registers

Name	7	6	5	4	3	2	1	0
F30_GPIO_Query0		Has Mechanical Mouse Buttons	HasGpio Driver Control	Has Haptic	HasGpio	HasLed	Has Mappable Buttons	—
F30_GPIO_Query1	—	—	—	GpioLedCount				

12.2.1. F30_GPIO_Query0

The bits of this register are defined as follows:

Reserved (F30_GPIO_Query0, bit 0)
Reserved.

HasMappableButtons (F30_GPIO_Query0, bit 1)
The *HasMappableButtons* bit is '0' if a GPO/LED cannot be controlled by capacitive buttons. If the *HasMappableButtons* bit is '1' then the each GPIO/LED pin may be controlled by a capacitive button. The programming of which button controls a GPIO/LED is described in section 12.3.8. This bit affects the number of such registers: there is either zero total or one register per GPIO/LED.

HasLed (F30_GPIO_Query0, bit 2)
This bit is set when the function support basic LED operations. When this bit is '0', the registers associated with LED operation are not present in the register map.

HasGpio (F30_GPIO_Query0, bit 3)
This bit is set when the function support basic GPI and GPO operations. When this bit is '0', the registers associated with GPI and GPO operation are not present in the register map.

HasHaptic (F30_GPIO_Query0, bit 4)
This bit is set when the function supports haptic operations. If the *HasHaptic* bit is '1' then any capacitive button may trigger a timed on-off change of the GPIO/LED pin. The *HasHaptic* bit is '0' if the haptic timed output pulse option is not available.

HasGpioDriverControl (F30_GPIO_Query0, bit 5)
This bit is set when the function supports programming GPIO pin/driver characteristics. See section 12.3.7 for information on the GPIO/LED control registers that define pin programming.

HasMechanicalMouseButtons (F30_GPIO_Query0, bit 6)
When this bit reports as '1', register F30_GPIO_Ctrl10 exists.

Reserved (F30_GPIO_Query0, bit 7)
Reserved.

12.2.2. F30_GPIO_Query1: GPIO/LED count

GpioLedCount (F30_GPIO_Query1, bits 4:0)
The GPIO/LED Count reports the number of GPIO/LED pins available. Several types of Function $30 registers use this value to calculate their register space size.

12.3. Function $30: control registers

These registers control the operation of the GPIO/LED pins. A '.*' indicates a variable-sized register block. Every product contains at least one variable-sized register block. The bits are mapped into the registers such that, for example, bit 0 of the GPIO/LED Select register controls the enabling of pin0, bit 1 controls pin 1, and so on. See section 12.3.1 for a description of how to determine the exact number of control registers for a product. See section 12.7 for an example of a register layout using 11 GPIO/LED pins.

Table 45. Function $30 control registers

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl0.*	—	—	—	—	—	—	—	LedSel0
F30_GPIO_Ctrl1	—	—	Halted	Halt	—	—	—	Gpi Debounce
F30_GPIO_Ctrl2.0	—	—	—	—	—	—	—	Dir0
F30_GPIO_Ctrl3.0	—	—	—	—	—	—	—	Data0
F30_GPIO_Ctrl4.*	—	—	—	—	—	—	—	LedAct0
F30_GPIO_Ctrl5.0	RampPeriodA							
F30_GPIO_Ctrl5.1	RampPeriodB							
F30_GPIO_Ctrl6.* (GPIO control)	—	STRPU	—	STRPD	SPCTRL			—
F30_GPIO_Ctrl6.* (LED control)	—	STRPU	—	STRPD	SPCTRL			—
F30_GPIO_Ctrl7.*	Open Drain	Invert	Valid	CapacitiveButtonNumber				
F30_GPIO_Ctrl8.0	HapticEn7	HapticEn6	HapticEn5	HapticEn4	HapticEn3	HapticEn2	HapticEn1	HapticEn0
F30_GPIO_Ctrl8.1	—	—	—	—	—	Haptic En10	HapticEn9	HapticEn8
F30_GPIO_Ctrl9	HapticDuration							
F30_GPIO_Ctrl10	—						NumberOfMechanical MouseButtons	

12.3.1. Calculating the number of control registers

The GPIO/LED function has a large number of registers. Some of the control register areas have one bit per GPIO/LED (such as the F30 GPIO/LED Select registers) and some have an entire register per GPIO/LED.

For the registers that have one bit per pin, the number of registers is calculated from the *GpioLedCount* Query as follows

```
F30_???_RegisterCount = trunc((GpioLedCount + 7) / 8);
```

Where there is one register per GPIO/LED, the number of registers is

```
F30_???_RegisterCount = GpioLedCount;
```

12.3.2. F30_GPIO_Ctrl0.*: GPIO/LED select

Every Function $30 usually contains at least one GPIO/LED Select register; this register is only present when both *HasLed* and *HasGpio* query bits are '1'. The '.*' indicates a variable-sized register block. See section 12.3.1 for a description of how to determine the exact number of control registers for a product.

The bits are mapped into the registers such that, for example, bit 0 of the GPIO/LED Select register controls the enabling of pin0, bit 1 controls pin 1, and so on. The reset default is '0'.

Each GPIO/LED pin is configurable at run time as either a GPIO or a LED. For each bit, if bit = 0 then the associated pin is GPIO; if bit = 1 then the associated pin is LED.

Caution: This register should typically be written before any other register. Otherwise there may be unexpected behavior. If a pin's type is changed by writing this register then any control registers associated with the pin are set to their default values; this means both the associated GPIO and LED register values are reset. For example, if a pin is changed from a LED to a GPIO then the pin's GPIO control registers will go to their default state (high impedance) and the pins LED registers such as pattern and brightness will go to their default state. This is because the GPIO and LED functionality shares resources. This register is almost never changed after it is initialized; this behavior is not a limitation.

12.3.3. F30_GPIO_Ctrl1: GPIO/LED general control

Every Function $30 contains one GPIO/LED General Control register. This register reset default is '0', and can be left at 0 in most applications.

GpiDebounce (F30_GPIO_Ctrl1, bit 0)

If this bit is set to '1', the device applies a mechanical-switch debouncing algorithm to all the GPI inputs.

Halt (F30_GPIO_Ctrl1, bit 4)

While this bit is set to '1', all LED ramping and animation is stopped and all GPIO input and output operations are stopped. Each LED holds at its current intensity, even if a ramp operation was active. When synchronized behavior is required – for example, when configuring several LEDs – this bit should be used. There is a delay after setting this bit until this function halts. The *Halted* bit should be monitored for this status.

Halted (F30_GPIO_Ctrl1, bit 5)

When this bit reports as '1', this function is halted.

12.3.4. F30_GPIO_Ctrl2.* and F30_GPIO_Ctrl3.*: GPIO Input/Output Mode control

Every Function $30 usually contains at least one GPIO Input/Output Mode register pair; these registers are only present if the *HasGpio* query bit is '1'. See section 12.3.1 for a description of how to determine the exact number of control registers for a product.

For each GPIO/LED there are two bits to specify the mode, Dir*N* and Data*N*. These bits are split between registers as shown in Table 45. The reset default of Dir*N* and Data*N* is typically '00' (high-impedance).

These registers control the input/output modes of pins configured as GPIO. For a pin programmed as an LED, writes are ignored and reads return undefined values. The Data*N* and Direction*N* bits together control the mode and output state of GPIO #*N*, as shown in Table 46.

Table 46 Function $30 GPIO control settings

DirN	DataN	State of GPIO #N
0	0	High-impedance state (typically input mode)
0	1	Pull-up resistor (typically input mode)
1	0	Driving digital ‘0’
1	1	Driving digital ‘1’

Writes to this mode register for pins that are the target of a capacitive button-to-GPIO mapping are ignored as described in section 12.3.8.

To write to both DirN and DataN without causing brief “glitches” on the output pins, perform the writes in this order:

1. When switching a pin's direction from input to output, write DataN followed by DirN.
2. When switching a pin's direction from output to input, write DirN followed by DataN.

Alternatively you can do the following:

1. Set the Halt bit (F30_GPIO_Ctrl1, bit 4) to 1.
2. Write the registers.
3. Clear the Halt bit to 0.

The state of an input is read through the GPI/LED data registers; see section 12.4 for more information.

12.3.5. F30_GPIO_Ctrl4.*: LED active control

Every Function $30 usually contains at least one LED Active register. This register is only present if the *HasLed* query bit is ‘1’. The ‘.*’ indicates a variable-sized register block. See section 12.3.1 for a description of how to determine the exact number of control registers for a product.

The bits are mapped into the registers such that, for example, bit 0 of the LED Active register represents the setting for LED 0, bit 1 describes LED 1, and so on. See section 12.3.1 for information on determining the exact sizes. The reset default is ‘0’.

Reading this register returns which LEDs are active. The exact on/off behavior is specified in the LED control registers; see section 12.3.7 for more information. For pins programmed as a GPIO, writes are ignored and reads return undefined values. Writes are also ignored and reads are undefined if the LED is the target of a Button-to-GPIO mapping as described in section 12.3.8.

Writing the LED Active *N* bit to ‘1’ sets LED#*N* to the Active state. The LED will turn on over a specified period of time or animate in a specified pattern. The amount of LED current corresponding to the “on” state for an LED is set by the per-LED Control registers. The data register indicates when a ramping operation is in progress; this applies to LEDs that turn on over a period of time.

Writing the LED Active *N* bit to ‘0’ sets LED #*N* to the inactive state. The LED will turn off either immediately or over a specified period of time. In RMI Function $30, the “off” state is always represented by zero sink current on the LED pin. The data register indicates when a ramping operation is in progress; this applies to LEDs that turn off over a period of time.

12.3.6. F30_GPIO_Ctrl5.*: LED ramp period control

Every Function $30 usually contains two or six Ramp Period registers, depending on the *ExtendedPatterns* query bit; this register is only present if the *HasLed* query bit is '1'. When the *ExtendedPatterns* bit is '0', there are two registers, and when it is '1', there are six registers. The reset default is '0' for each register.

These registers specify the ramp rates as used by the LED patterns specified in the next section. Each unit of a period specifies 10 milliseconds, so the range of a ramp period is 0 to approximately 2.5 seconds. $00 indicates an instantaneous transition and $FF is approximately 2.5 seconds.

12.3.7. F30_GPIO_Ctrl6.*: GPIO/LED control

Each GPIO/LED usually has a register that programs something about the PIN operation. The '.*' indicates that this is a variable-sized register; each LED is mapped to a register such that, for example, register 6.0 of the LED Active register block represents the settings for LED 0, register 6.1 describes LED 1, and so on. See section 12.3.1 for information on determining the exact sizes.

There are two situations when this register is preset:

- *HasLed* query bit is '1', or
- *HasLed* is '0' and *HasGpioDriverControl* is '1'.

The reset default is '0' for each register. Each individual register in this register block controls either an LED or a GPO, depending on whether the associated GPIO pin is programmed as an LED or a GPIO. This is typically selected with the GPIO/LED Select register; see section 12.3.2 for more information. The following subsections describe the two register layouts, one for a GPIO and one for LED.

12.3.7.1. F30_GPIO_Ctrl6.: GPIO control*

This is the register description for controlling a GPIO.

Reserved (F30_GPIO_Ctrl6., bit 0)*

Reserved.

SPCTRL (F30_GPIO_Ctrl6., bits 3:1)*

This field defines the output driver strength control. This 3-bit field controls the drive speed (or drive strength) of the GPIO output driver. A higher value for the field corresponds to higher drive strength.

STRPD (F30_GPIO_Ctrl6., bit 4)*

This field defines the strong pull-down strength:

- If '0', there is no effect.
- If '1', there is double the maximum uncontrolled GPIO sink current.

This bit is provided as an alternative to the controlled LED intensity control in situations where a current greater than 12 mA is desired and accuracy is not critical. The reason for the low accuracy is that the sink current is determined by the LED drop and series resistance of external circuit. Thus, in applications where the GPIO pull-down driver is used to sink current, the field that is a concatenation of STRPD and SPCTRL serves as a coarse control for the maximum uncontrolled sink current.

Reserved (F30_GPIO_Ctrl6., bit 5)*

Reserved.

STRPU (F30_GPIO_Ctrl6., bit 6)*

This field defines the strong pull-up enable:

- If '0', a weak resistive pull-up strength.
- If '1', a strong resistive pull-up strength.

The weak pull-up strength consumes less power. This is only meaningful if the GPIO Input/Output Mode specifies a pull-up resister.

Reserved (F30_GPIO_Ctrl6., bit 7)*

Reserved.

12.3.7.2. F30_GPIO_Ctrl6.*: LED control

This is the register description for controlling an LED.

These registers control the intensity and waveform/animation of the selected LED. There are two styles specified by the patterns: continuous animation while the LED is active and secondly waveforms that follow the LED active bit.

Brightness (F30_GPIO_Ctrl6., bits 3:0)*

The Brightness field indicates the target intensity level when the LED is enabled:

- $0 represents fully off,
- $1 - $E represent intermediate intensities evenly or approximately-evenly spaced (in units of human-perceived brightness) between fully off and fully on, and
- $F represents fully on.

Pattern (F30_GPIO_Ctrl6., bits 7:4)*

The Pattern field takes one of the following values:

Pattern = '0000': Rise and fall period A.

In this setting, when the LED is activated by setting the LED Active *N* bit in the LED Active register, the LED ramps to the intensity indicated by the Brightness field over a time determined by *RampPeriodA* of the LED Ramp Period register and then holds at that intensity. When the LED is deactivated by clearing the LED Active *N* bit, the LED ramps down to the "off" state over a time determined by *RampPeriodA* and then remains "off."

The ramp begins immediately after the write to the LED Active control registers or the Per-LED Control register, and may be unsynchronized (out of phase) with other ongoing ramps or animations. To ramp several LEDs synchronously, use the LedHalt bit of the GPIO/LED General Control register. The GPI/LED Data registers provide ramp busy status. The LED's ramp busy bit shows '1' after the LED active bit is written, and stays at '1' until the LED brightness reaches either the target level or off, depending on the direction of the ramp.

Pattern = '0001': Rise and fall period B.
This pattern is like pattern '0000', except that the *RampPeriodB* parameter determines the ramp rate instead of *RampPeriodA*.

Pattern = '0010': Rise period A, fast fall.
In this setting, when the LED is activated the LED ramps to the intensity indicated by the *Brightness* field over a time determined by *RampPeriodA* and then holds at that intensity. When the LED is deactivated, the LED switches immediately to the "off" state.

Pattern = '0011': Rise period B, fast fall.
This pattern is like pattern '0010', except that the *RampPeriodB* parameter determines the ramp rate instead of *RampPeriodA*.

Pattern = '0100': Fast rise, fall period A.
In this setting, when the LED is activated the LED switches immediately to the intensity indicated by the *Brightness* field and then holds at that intensity. When the LED is deactivated, the LED ramps down to the "off" state over a time determined by *RampPeriodA* and then remains "off."

Pattern = '0101': Fast rise, fall period B.
This pattern is like pattern '0100', except that the *RampPeriodB* parameter determines the ramp rate instead of *RampPeriodA*.

Pattern = '0110': Ramping animation.
In this setting, when the LED is activated the LED ramps to the intensity indicated by the *Brightness* field over a time determined by *RampPeriodA*, and then holds at that intensity for a time determined by *RampPeriodB*. The LED then ramps down to the "off" state over *RampPeriodA* and remains off for *RampPeriodB*. This cycle repeats continuously for as long as the LED is activated. When the LED is deactivated, the LED switches immediately to the "off" state.

Pattern = '0111': Pulsed animation.
In this setting, when the LED is activated the LED pulses between the "on" and "off" states. The LED switches immediately to the intensity indicated by the *Brightness* field, and then remains at the target intensity for a time determined by *RampPeriodB*. At that point, the LED switches immediately to the "off" state and remains "off" for a time determined by *RampPeriodA*. This cycle repeats continuously for as long as the LED is activated. When the LED is deactivated, the LED switches immediately to "off" state.

12.3.8. F30_GPIO_Ctrl7.*: button-to-GPIO mapping and control

Each GPIO/LED has one register that programs a capacitive button to control the GPO/LED output. If the *HasMappableButtons* query bit is '0' then these Button-to-GPIO Mapping and Control registers are not present in the register space.

The '.*' indicates a variable-sized register block. Every product contains at least one Button Mapping register. Each LED is mapped to a register such that, for example, register 7.0 of the Button Mapping register block represents the settings for LED 0, register 7.1 describes LED 1, and so on. See section 12.3.1 for information on determining the exact sizes. The reset default is '0' for each register.

The precise effect of the capacitive button on the GPIO/LED is defined by the *ButtonPolarity* and *OpenDrain* bits. For a mapping to an LED, the button state sets the corresponding LED Active register bit as described in section 12.3.5. For a mapping to a GPO the output follows the capacitive button.

CapacitiveButtonNumber (F30_GPIO_Ctrl7., bits 4:0)*

This is the capacitive button number that affects the GPIO/LED pin that corresponds to this register. A value of 0x1F specifies that any capacitive button affects the GPIO/LED. The precise meaning is that all the capacitive buttons are ORed together into a single virtual button which is used to affect the corresponding GPIO/LED.

Valid (F30_GPIO_Ctrl7., bit 5)*

When this bit is '1', the output associated with this GPIO/LED is controlled by the specified button.

Invert (F30_GPIO_Ctrl7., bit 6)*

When this bit is '1', the state of the button (finger present or finger absent) is inverted before affecting the GPIO/LED.

OpenDrain (F30_GPIO_Ctrl7., bit 7)*
For pins programmed as GPIO and controlled by a button, the pin's mode (see section 12.3.4 for a description of the input and output modes) is affected by the button state; the button state overrides bits in that register. Set this bit to '1' if the GPO is open-drain.

Example 1: Active low "Totem Pole" button – Invert*N* = 1, OpenDrain*N* = 0

Example 2: SPST button connect to GND, - Invert*N* = 0, OpenDrain*N* = 1

Note: This bit is only used if the corresponding pin is programmed as a GPIO. It is not used if it is programmed as an LED.

Note: The same button may affect more than one GPIO/LED; for example, consider a tri-color LED. It is guaranteed that all the affected GPIO/LED animations/waveforms start simultaneously.

12.3.9. F30_GPIO_Ctrl8.*: haptic enable control

If the *HasHaptic* bit in Query Register 0 is set to 1, then Function $30 contains at least one Haptic Enable register. See section 12.3.1 to determine the exact sizes.

Setting a bit in this register enables haptic output for the corresponding GPIO pin. If more than one bit is set, then the haptic pulse is output simultaneously to all enabled pins. The *CapacitiveButtonNumber* parameter for a GPIO enabled as a haptic output (see F30_GPIO_Ctrl7, bits 4:0) specifies which capacitive button to monitor. When multiple bits are set the corresponding *CapactiveButtonNumber*s should be the same.

When any button state bit goes high, a one-shot timer is triggered/re-triggered. The timer state controls the state of a GPIO or LED, depending on *LedSel*. The *ButtonPolarity* and *OpenDrain* bits can be set for each GPIO pin.

12.3.10. F30_GPIO_Ctrl9: haptic duration control

If the *HasHaptic* bit in Query Register 0 is set to '1', then Function $30 contains a HapticDuration register. The duration of the haptic output to the GPIO/LED pin is defined by the HapticDuration register.

Each unit of duration represents 10 milliseconds, so the range of a haptic output pulse is 0 to approximately 2.5 seconds. $00 disables the output, $01 is a 10 millisecond output pulse and $FF is approximately 2.5 seconds.

12.3.11. F30_GPIO_Ctrl10: mechanical mouse buttons

This register exists only if *HasMechanicalMouseButtons* (F30_GPIO_Query0 bit 6) reports as '1'.

NumberOfMechanicalMouseButtons (F30_GPIO_Ctrl10, bits 1:0)

This field reports the number of GPIO pins connected to mechanical mouse buttons.

00 = Reserved
01 = One GPIO pin
10 = Two GPIO pins
11 = Three GPIO pins

Reserved (F30_GPIO_Ctrl10, bits 7:2)

Reserved.

12.4. Function $30: data registers

Function $30 has a single data register type which is shared by GPI input data and LED ramping state information. The '.*' indicates a variable-sized register block. Every product contains at least one GPIO data register.

Table 47. Function $30 per-GPIO/LED data register

Name	7	6	5	4	3	2	1	0
F30_GPIO_Data0.*	—	—	—	—	—	—	—	GpiLed Data0

This register is used to read the state of the pins programmed as GPIs. For pins programmed as a GPI, the value in this register is the same as the value on the input pin. Pins programmed as GPO have undefined values.

For pins programmed as an LED, this register has an LED's "ReachedTarget" status. The value in this register is set to a '1' when a ramp up operation completes and has reached its maximum; it is set to a '0' when a ramp down operation completes and has reached its minimum. The specific behavior is seen in the following diagram. This allows the host to determine the state of a ramping operation. The "rampBusy" is LedActive xor ReachedTarget.

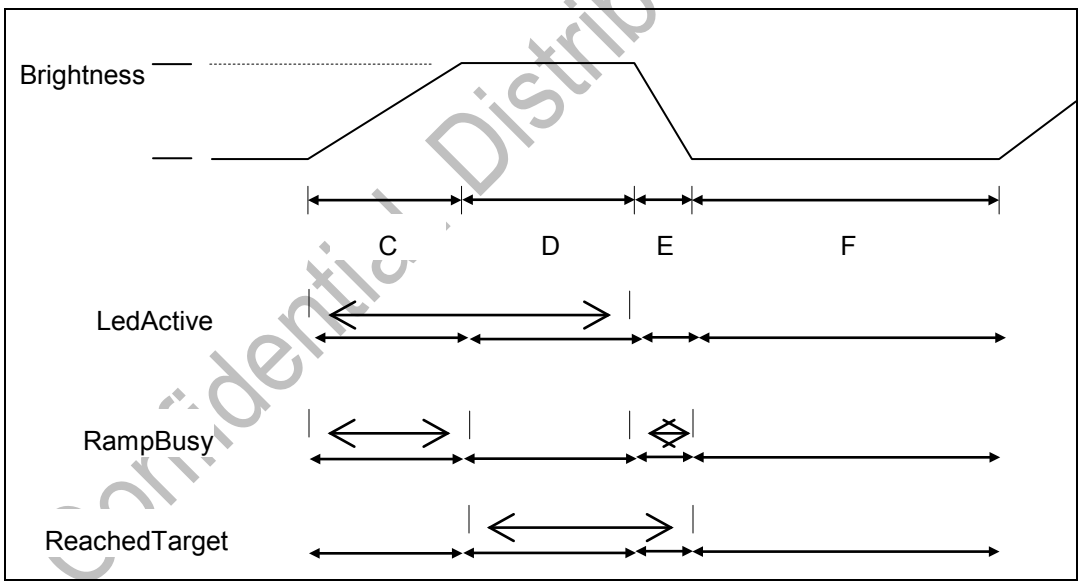

Figure 15. LED ramping status

12.5. Function $30: interrupt source

Function $30 implements one interrupt source. Function $30 asserts an interrupt request whenever any GPI input data bit in a GPI/LED Data register changes from a '0' to a '1' or from a '1' to a '0'. Because an interrupt request is a "sticky" bit, interrupt requests read as '1' if any button bit has changed since the last time the data registers were read, even if the button bit has changed back to its previous value since that time. An interrupt request is asserted synchronously with the report rate.

12.6. Function $30: command registers

Function $30 does not implement any command registers.

12.7. Function $30 example: complete control layout

Function $30 contains several variable-sized control registers. Every product contains at least one variable-sized register block. The bits are mapped into the registers such that, for example, bit 0 of the GPIO/LED Select register controls the enabling of pin 0, bit 1 controls pin 1, and so on. The following example shows a register layout using 11 GPIO/LED pins. For this example, *GpioLedCount* = 11, *HasMappableButtons* = 1, *HasLed* = 1, *HasGpio* = 1, *HasHaptic* = 1, *HasGpioDriverControl* = 0.

Table 48. Function $30 example: complete control layout

Name	7	6	5	4	3	2	1	0
F30_GPIO_Query0	—	—	HasGpio Driver Control	Has Haptic	HasGpio	HasLed	Has Mappable Buttons	—
F30_GPIO_Query1	—	—	—	GpioLedCount				
F30_GPIO_Ctrl0.0	Led Sel7	Led Sel6	Led Sel5	Led Sel4	Led Sel3	Led Sel2	Led Sel1	Led Sel0
F30_GPIO_Ctrl0.1	—	—	—	—	—	Led Sel10	Led Sel9	Led Sel8
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	Gpi Debounce
F30_GPIO_Ctrl2.0	Dir7	Dir6	Dir5	Dir4	Dir3	Dir2	Dir1	Dir0
F30_GPIO_Ctrl2.1	—	—	—	—	—	Dir10	Dir9	Dir8
F30_GPIO_Ctrl3.0	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0
F30_GPIO_Ctrl3.1	—	—	—	—	—	Data10	Data9	Data8
F30_GPIO_Ctrl4.0	LedAct7	LedAct6	LedAct5	LedAct4	LedAct3	LedAct2	LedAct1	LedAct0
F30_GPIO_Ctrl4.1	—	—	—	—	—	Led Act10	LedAct9	LedAct8
F30_GPIO_Ctrl5.0	RampPeriodA							
F30_GPIO_Ctrl5.1	RampPeriodB							
F30_GPIO_Ctrl5.2	RampPeriodC							
F30_GPIO_Ctrl5.3	RampPeriodD							
F30_GPIO_Ctrl5.4	RampPeriodE							
F30_GPIO_Ctrl5.5	RampPeriodF							
F30_GPIO_Ctrl6.0	Pattern				Brightness			
F30_GPIO_Ctrl6.1	Pattern				Brightness			
F30_GPIO_Ctrl6.2 .... F30_GPIO_Ctrl6.8	Pattern				Brightness			
F30_GPIO_Ctrl6.9	Pattern				Brightness			
F30_GPIO_Ctrl6.10	Pattern				Brightness			
F30_GPIO_Ctrl7.0	Open Drain	Button Polarity	Valid	Capacitive Button Number				
F30_GPIO_Ctrl7.1	Open Drain	Button Polarity	Valid	Capacitive Button Number				
F30_GPIO_Ctrl7.2 ...	Open Drain	Button Polarity	Valid	Capacitive Button Number				

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl7.10								
F30_GPIO_Ctrl8.0	Haptic En7	Haptic En6	Haptic En5	Haptic En4	Haptic En3	Haptic En2	Haptic En1	Haptic En0
F30_GPIO_Ctrl8.1	—	—	—	—	—	Haptic En10	Haptic En9	Haptic En8
F30_GPIO_Ctrl9	Haptic Duration							
F30_GPIO_Data0.0	GpiLed Data7	GpiLed Data6	GpiLed Data5	GpiLed Data4	GpiLed Data3	GpiLed Data2	GpiLed Data1	GpiLed Data0
F30_GPIO_Data0.1	—	—	—	—	—	GpiLed Data10	GpiLed Data9	GpiLed Data8

12.8. Function $30 examples: query bits

When a particular firmware is built, the query bits are fixed by the features included in the build. The subsections below describe some of the possible configurations.

12.8.1. Function $30 example: both GPIOs and LEDs

For this example, *HasLed* is '1', *HasGpio* is '1', and *HasGpioDriverControl* is '1'.

Table 49. Function $30 example: GPIOs and LEDs

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl0.*	LedSel7	LedSel6	LedSel5	LedSel4	LedSel3	LedSel2	LedSel1	LedSel0
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	Gpi Debounce
F30_GPIO_Ctrl2.*	Dir7	Dir6	Dir5	Dir4	Dir3	Dir2	Dir1	Dir0
F30_GPIO_Ctrl3.*	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0
F30_GPIO_Ctrl4.*	LedAct7	LedAct6	LedAct5	LedAct4	LedAct3	LedAct2	LedAct1	LedAct0
F30_GPIO_Ctrl5.*	Ramp PeriodA							
F30_GPIO_Ctrl6.*	Pattern-Brightness-for-LEDs or STRPU-STRPD-SPCTRL-for-GPIOs							

12.8.2. Function $30 example: LEDs only

For this example, *HasLed* is '1' and *HasGpio* is '0'.

Table 50. Function $30 example: LEDs

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	Gpi Debounce
F30_GPIO_Ctrl4.*	LedAct7	LedAct6	LedAct5	LedAct4	LedAct3	LedAct2	LedAct1	LedAct0
F30_GPIO_Ctrl5.*	RampPeriod*							
F30_GPIO_Ctrl6.*	Pattern				Brightness			

12.8.3. Function $30 example: GPIOs only, with driver control

For this example, *HasLed* is '0', *HasGpio* is '1', and *HasGpioDriverControl* is '1'.

Table 51. Function $30 example: GPIOs with driver control

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	Gpi Debounce
F30_GPIO_Ctrl2.*	Dir7	Dir6	Dir5	Dir4	Dir3	Dir2	Dir1	Dir0
F30_GPIO_Ctrl3.*	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0
F30_GPIO_Ctrl6.*	—	STRPU	—	STRPD	SPCTRL			—

12.8.4. Function $30 example: GPIOs only, without driver control

For this example, *HasLed* is '0', *HasGpio* is '1', and *HasGpioDriverControl* is '0'.

Table 52. Function $30 example: GPIOs without driver control

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	Gpi Debounce
F30_GPIO_Ctrl2.*	Dir7	Dir6	Dir5	Dir4	Dir3	Dir2	Dir1	Dir0
F30_GPIO_Ctrl3.*	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0

13. Function $31: LEDs

Function $31 implements adjustable-current LED outputs. Devices with up to twelve LEDs are supported.

13.1. Function $31: query registers

Table 53. Function $31 LED query registers

Name	7	6	5	4	3	2	1	0
F31_LED_Query0	—							Has Brightness
F31_LED_Query1	—				NumberOfLEDs			

HasBrightness (F31_LED_Query0, bit 0)

When this bit reports as '1', replicated registers F31_LED_Ctrl0.* exist.

NumberOfLEDs (F31_LED_Query1 bits 3:0)

This field reports the number of LED outputs provided by the device. The range is 1 through 12.

13.2. Function $31: control registers

These registers control the LED outputs.

13.2.1. F31_LED_Ctrl0.*: LED brightness controls

These replicated registers control the brightness of each LED. The number of replicated registers is equal to *NumberOfLEDs (F31_LED_Query1, bits 3:0)*.

Table 54. Function $31 LED Brightness Control registers

Name	7	6	5	4	3	2	1	0
F31_LED_Ctrl0.*	LED*n*Brightness							

Brightness (F31_LED_Ctrl0.)*

The *LED*n *Brightness* register controls the intensity of LED *n* (0 = off, 255 = maximum intensity).

13.3. Function $31: data registers

Function $31 has no data registers.

13.4. Function $31: interrupt source

Function $31 has no interrupt sources.

13.5. Function $31: command registers

Function $31 has no command registers.

14. Function $34: Flash memory management

Function $34 implements all functionality related to flash memory. Flash programming can be used to upgrade the user interface (UI) firmware in the field or to alter the configuration of the RMI device.

In addition to flash erase and programming operations, Function $34 provides fundamental integrity assurance for that flash memory and the firmware and configuration it contains. Function $34 provides the following features:

- Boot-time integrity check for the UI firmware and its configuration.
- Mechanism to bypass execution of UI firmware even if it passes the integrity check.
- Firmware and Configuration erase and reprogram function.
- Mechanism to recover from interrupted or failed erase/reprogram attempts.

14.1. Overview

14.1.1. Non-volatile memory organization

Figure 16 shows the basic storage methodology for the device firmware:

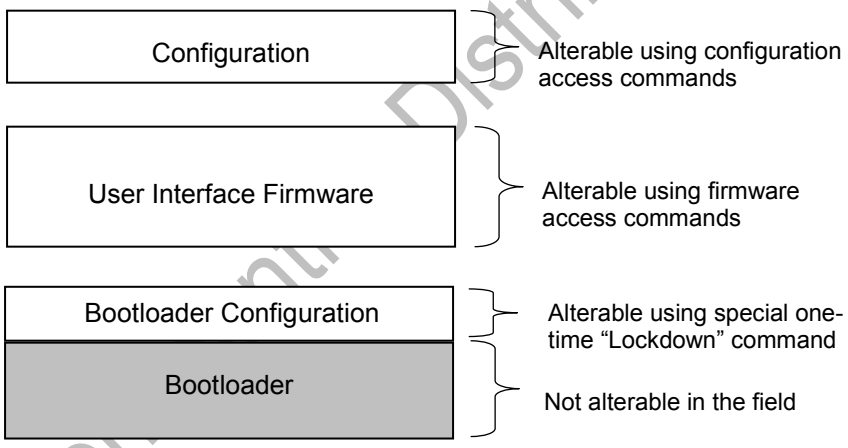

Figure 16. Non-volatile memory organization

14.1.1.1. Configuration

The configuration space stores the default values of the device’s control registers. The bootloader provides a mechanism to erase and reprogram this space. Because an existing configuration may not be valid for a new firmware revision, any update to the UI firmware should be followed by an update of the configuration space.

14.1.1.2. User interface firmware

The UI firmware space contains the firmware that implements the primary function of the device. The bootloader firmware provides a mechanism to erase and reprogram this space.
UI firmware images are provided by Synaptics in an encrypted form to ensure they can only be installed on an appropriate device.

It is not possible to erase the UI firmware space without also erasing the configuration space.

14.1.1.3. Bootloader

The bootloader checks the integrity of the UI firmware space and provides the ability to re-flash the UI or the configuration space.

To guarantee that a device can never be irrecoverably corrupted during a firmware update operation, the bootloader space is not field-programmable via Function $34. However, to implement the bootloader Lockdown command, the bootloader will permit a small, one-time change confined to the bootloader configuration space. Because the bootloader space is not field-erasable, the lockdown operation is permanent.

The size of these spaces is product specific and can be determined by querying the Function $34 registers.

14.1.2. Modality

Function $34 flash programming functions are mutually exclusive with most other RMI functions. Two modes determine which functions are available:

- **UI mode:** This is the default RMI mode, but in Function $34 only one command is operable in this mode: Enable Flash Programming, which is used to enter Flash Programming mode. A device is in UI mode when the *Flash Programming En* bit is '0'. A Function $01 Reset command is used to end Flash Programming mode.

  Note: It is possible for the touch controller itself to initiate entry to the Flash Programming mode as part of the recovery mechanism for programming failures due to interruption or other causes. See section 4.3.1 for more information on Flash CRC errors.

- **Bootloader mode:** All other Function $34 commands only function in this mode. A device is in Bootloader mode (also sometimes known as Flash Programming mode) when the *Flash Programming En* bit is '1'.

When Flash Programming is enabled, the device is in Bootloader mode and the normal operation of the device is disabled. Only the following subset of functions is available:

- *Page Description Tables*
  The Page Description tables accurately reflect the reduced functionality offered in Bootloader mode. Only Functions $01 and $34 are visible.

Important: The data in the Page Description tables may be different from the Page Description tables operating in UI mode. Hosts should re-scan the Page Description tables when entering or exiting Bootloader mode.

- *Page Select Register*
  The Page Select Register always selects Page $00.

Important: Writes to the Page Select Register are ignored.

- *Function $01: Control Register 1 – Interrupt Enable* register.
  Only the Function $34 interrupt enable bit is writable. All others are forced to 0, disabling those sources.

- *Function $01: Data Register 1 – Device Status* register.
  The *FlashProg* bit is set and the *Status Code* field indicates the reason that the bit is set.

- *Function $01: Data Register 1 – Interrupt Status* register
  Only the Function $34 interrupt source asserts.
- *Function $34:*
  All of the Function $34 registers function as documented in this specification.

14.2. Function $34: query registers

Table 55. Function $34 query registers

Name	7	6	5	4	3	2	1	0
F34_Flash_Query0	BootloaderID0							
F34_Flash_Query1	BootloaderID1							
F34_Flash_Query2	—					Has ConfigID	Unlocked	RegMap
F34_Flash_Query3	BlockSize [7:0]							
F34_Flash_Query4	BlockSize [15:8]							
F34_Flash_Query5	FirmwareBlockCount [7:0]							
F34_Flash_Query6	FirmwareBlockCount [15:8]							
F34_Flash_Query7	ConfigurationBlockCount [7:0]							
F34_Flash_Query8	ConfigurationBlockCount 15:8]							

14.2.1. F34_Flash_Query0 and F34_Flash_Query1: bootloader ID query

This register reports the unique 16-bit Bootloader ID. This ID is used to identify the bootloader revision. For example, a Version 3 bootloader would set *Bootloader ID 0* to ‘V’ and *Bootloader ID 1* to ‘3’.

14.2.2. F34_Flash_Query2: flash properties query

This byte contains bits that describe whether the RMI product has various optional properties.

Each property bit is ‘1’ if the product has the associated property or ‘0’ if the product does not have the associated property. Reserved property bits report as ‘0’, but they may report as ‘1’ in devices that comply with a future version of flash functions.

RegMap Query (F34_Flash_Query2, bit 0)

This bit always reports as 1.

Unlocked (F34_Flash_Query2, bit 1)

This command has no meaning in UI mode. In Bootloader mode, a device can be in either a locked or an unlocked state:

- A ‘1’ indicates that the device is in the generic, unlocked state. A device in this state is capable of being locked down.
- A ‘0’ in this bit location indicates that either the device has already been locked down, or that it is a legacy device. In either case, the generic aspects of device operation are fixed and cannot be locked down.

HasConfigId (F34_Flash_Query2, bit 2)

If *HasConfigId* reports as '1', registers F34_Flash_Ctrl0.0 throughF34_Flash _Ctrl0.3 exist.

Reserved (F34_Flash_Query2, bits 7:3)

Reserved.

14.2.3. F34_Flash_Query3 and F34_Flash_Query4: block size query

The Block Size indicates the number of bytes in one data block. When programming the firmware, the data should be broken into blocks of this size and each block programmed individually.

BlockSize (F34_Flash_Query3)

Bits 7:0.

BlockSize (F34_Flash_Query4)

Bits 15:8.

14.2.4. F34_Flash_Query5 and F34_Flash_Query6: firmware block count query

The *Firmware Block Count* indicates the number of blocks in a firmware image. *Block Size* * *Firmware Block Count* = total number of bytes in a firmware image area.

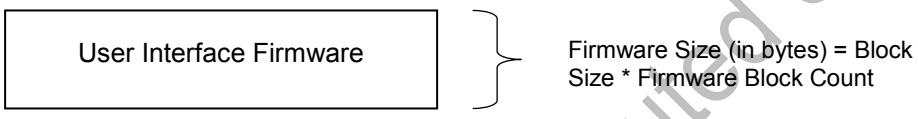

Figure 17. Firmware Size

FirmwareBlockCount (F34_Flash_Query5)

Bits 7:0.

FirmwareBlockCount (F34_Flash_Query6)

Bits 15:8.

14.2.5. F34_Flash_Query7 and F34_Flash_Query8: configuration block count query

The *Configuration Block Count* indicates the number of blocks in a configuration image. *Block Size* * *Configuration Block Count* = total number of bytes in a configuration image.

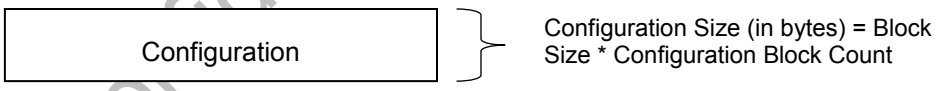

Figure 18. Configuration Size

ConfigurationBlockCount (F34_Flash_Query7)

Bits 7:0.

ConfigurationBlockCount (F34_Flash_Query8)

Bits 15:8.

14.3. Function $34: control registers

These control registers are present only if *HasConfigID* (F34_Flash_Query2, bit 2) reports as '1'.

Table 56. Function $34 control registers

Name	7	6	5	4	3	2	1	0
F34_Flash_Ctrl0.0	CustomerConfigurationID [7:0]							
F34_Flash_Ctrl0.1	CustomerConfigurationID [15:8]							
F34_Flash_Ctrl0.2	CustomerConfigurationID [23:16]							
F34_Flash_Ctrl0.3	CustomerConfigurationID [31:24]							

CustomerConfigurationID (F34_Flash_Ctrl0.0 through F34_Flash_Ctrl0.3)

These registers hold a customer-specified configuration identifier.

Note: The firmware does not use the data contained in these registers for any purpose. The information in these registers is strictly for use by the customer host system. The intent is that the customer can store data in these registers for the purposes of tracking changes to the RMI configuration space.

These registers can be read from either bootloader mode or UI mode. However, because the data in these registers is associated with the configuration space of the device, the data returned by these registers while the device is in bootloader mode should only be trusted if the host knows that the configuration space is valid.

14.4. Function $34: data registers

Function $34's data registers specify block numbers, report block data, and control all aspects of flash update operations.

Table 57. Function $34 data registers

Name	7	6	5	4	3	2	1	0
F34_Flash_Data0	BlockNumber [7:0]							
F34_Flash_Data1	BlockNumber [15:8]							
F34_Flash_Data2.*	BlockData							
F34_Flash_Data3	Program Enabled	FlashStatus			FlashCommand			

14.4.1. F34_Flash_Data0 and F34_Flash_Data1: block number registers

The Block Number registers are used to specify which Flash Block is accessed by flash commands. These registers should be written with a block number prior to issuing any of the following commands:

- Read Firmware Block
- Write Firmware Block
- Read Configuration Block
- Write Configuration Block

The Block Number registers automatically increment after each operation.

14.4.2. F34_Flash_Data2.*: block data registers

The Block Data registers are used to read or write the data for a block. F34_Flash_Data2 is a replicated register (see section 2.3.6 for a definition of these registers). Each implementation has at least one of these registers, but the exact count is implementation-specific. The value obtained from the F34_Flash_Query3,4 'block size' registers represents the exact number of F34_Flash_Data2 registers (number of registers = *n*, where *n* is Block Size).

The Block Data registers must be written with data prior to a block write operation. At the completion of a block read operation, the Block Data registers contain the results of the read operation.

Prior to executing an *Erase All*, *Erase Configuration* or *Enable Flash Programming* operation, the first two registers must be written with the Bootloader ID, which acts as a key to enable the operation.

14.4.3. F34_Flash_Data3: flash control/status register

This register is used to control all aspects of flash update operation.

Flash Command (F34_Flash_Data3, bits 3:0)

When this field is written the requested operation is performed. This field is not writable while the previous operation is still in progress – this field is only writable when the current value is $0. When a requested operation is completed this field is set back to $0 by the device and a Function $34 interrupt is asserted. All commands that can be used to read, erase, or program the flash can only be issued when the device is in a 'Flash Programming Enabled' state.

Table 58. Function $34 Flash command values

Value	Meaning	Available in UI Mode?	Available in Bootloader mode?	Requires key before issuing command?
$0	Idle – No command is active	Yes	n/a	n/a
$1	*Reserved*			
$2	Write Firmware Block	No	Yes	No
$3	Erase All	No	Yes	Yes
$4	Write Lockdown Block	No	Yes	No
$5	Read Configuration Block	No	Yes	No
$6	Write Configuration Block	No	Yes	No
$7	Erase Configuration	No	Yes	Yes
$8	Read Sensor ID	No	Yes	No
$9-$E	*Reserved*	No		No
$F	Enable Flash Programming	Yes	No	Yes

Flash Command $00: Idle

There are no Function $34 commands in process and the device is ready to receive a Function $34 command.

Flash Command $01: Reserved.

Flash Command $02: Write Firmware Block.

This command is issued after a Block Number is written to the Block Number registers and the data block is written to the Block Data registers. This command actually writes the block into the UI Firmware space. The UI Firmware space must have first been erased with command $03 prior to writing the firmware blocks. An attempt to write to a non-erased firmware block will result in an error 5 – "*Block Not Erased*."

Flash Command $03: Erase All.

This command clears both the UI Firmware space and the Configuration space. Use it before programming (writing) the new image with command $02. Because an existing configuration is likely invalid for a new firmware revision, this command also clears the configuration area, which should be programmed with the new configuration associated with the new firmware revision.

Important: Prior to executing this command, the host must write the Bootloader ID to the Block Data registers as a "key value" to reduce the possibility of an accidental erasure.

Flash Command $04: Write Lockdown Block.

This command writes the lockdown blocks. It is anticipated that the number of blocks to be written to lock a device down will be very small (about three blocks in total). The number of blocks is defined by the image file itself.

This command is only processed if the *Unlocked* bit is in the '1' (unlocked) state. The lockdown information cannot be erased once it is written.

The lockdown operation is a one-time affair. The changes caused by the lockdown operation will not take effect until the device is either reset or an 'Enable Flash Programming' command is issued.

The actual lockdown operation is performed after the final lockdown block is written and before the status code is returned. Lockdown blocks written before the final lockdown block may report "success" as they buffer the lockdown information. When the final block is written, the device will report the overall success or failure as shown in Table 59.

Table 59. Lockdown state and result code

State	*Result code*
Unlocked, and lockdown data is valid	0 ("Success")
Unlocked, but lockdown data is invalid	2 ("Flash Programming Not Enabled / Bad Command")
Unlocked, but block number is invalid	3 ("Invalid Block Number")
Locked, and block number is valid	4 ("Block Not Erased")
Locked, but block number is not valid	3 ("Invalid Block Number")

The response depends on the bootloader's current lockdown state when it receives the command, and the state of the block number and the lockdown data itself.

Flash Command $05: Read Configuration Block.

This command reads a block of data from the configuration space and places it into the Block Data registers to be read by the host. This command is issued after a block number is written to the Block Number register. After the command has completed, the block read data is available to be read from the Block Data registers.

Flash Command $06: Write Configuration Block.

This command is issued after a block number is written to the Block Number register and the data block is written to the Block Data registers. This command actually writes the block into the Configuration space.

The Configuration space must have first been erased with command $07 prior to writing the firmware blocks. An attempt to write to a non-erased configuration block will result in an error 5 – "*Block Not Erased.*"

Flash Command $07: Erase Configuration.

This command clears the Configuration space. It should be used prior to programming (writing) the new configuration, one block at a time, with command $06.

Important: Prior to executing this command, the host must write the Bootloader ID to the Block Data registers as a "key value" to reduce the possibility of an accidental erasure.

Flash Command $08: Read Sensor ID.

This command reads the Sensor ID. The Sensor ID pins are a set of GPIO pins configured to be read as a byte to understand which sensor is connected to the Touch Controller in a particular configuration.

Flash Command $09: Reserved.

Flash Command $0E: Reserved.

Flash Command $0F: Enable Flash Programming.

The Enable Flash Programming command is used to initiate flash programming operations. Prior to the execution of this command the only Function $34 registers that are functional are the query registers and the command register in which only this command is supported.

As a side-effect of this command, the Function $01 Interrupt Enable (*F01_RMI_Ctrl1*) register is forced to disable all interrupts except for the Function $34 interrupt, which is forced enabled. When flash programming is complete, the host must issue an RMI Reset command to put the device back into normal operating mode.

Important: Before executing this command, the host must write the Bootloader ID to the Block Data registers as a "key value" to reduce the possibility of accidentally entering Flash Programming mode.

Attempts to execute this command when Flash Programming mode is already enabled are treated as a no-operation and always return Success.

Flash Status (F34_Flash_Data3, bits 6:4)

This field, at the completion of a flash operation, indicates the success or failure of the operation. Writes to this field are ignored.

Table 60 Function $34 Flash Status values

Value	*Meaning*
0	Success
1	*Reserved*
2	Flash Programming Not Enabled/Bad Command
3	Invalid Block Number
4	Block Not Erased
5	Erase Key Incorrect
6	Unknown Erase/Program Failure
7	Device has been reset

Program Enabled (F34_Flash_Data3, bit 7)

This read-only bit indicates that flash programming has been enabled, either by an Enable Flash Programming command or because a Flash CRC error was detected during boot. If a CRC error is detected, the Function $01 *FlashProg* field *(F01_RMI_Data0, bit 6)* is set and the specific cause of the error can be found in the Function $01 Status Code *(F01_RMI_Data0, bits 3:0)*.

When this bit reports as '0', the device in "UI mode." When this bit reports as '1', the device in "Bootloader mode."

In Bootloader mode, only a very small subset of the RMI interface is available:
the Function $34 registers and the Function $01 registers required to manage the Function $34 interrupt.

This bit's state changes from '1' to '0' only when a Function $01 Reset command is executed and the device successfully enters UI mode.

14.5. Function $34: interrupt source

Function $34 is associated with a single interrupt request source, called the Flash interrupt request. Any product that includes Function $34 allocates a Flash interrupt request bit in the Interrupt Status register (see section 4.3.2), and a Flash interrupt enable bit in the Interrupt Enable register (see section 4.2.2). The position of the Flash interrupt bits in those registers is product-specific; consult the product-specific documentation to determine their location.

15. Function $36: Auxiliary analog to digital conversion

Function $36 implements controls for an analog-to-digital voltage conversion (ADC) function. This can be used for converting external voltages from other types of sensor and reporting these over RMI along with data from the capacitive sensor.

15.1. Function $36: query register

The ADC Function $36 implements a single query register, the General Properties query.

Table 61. Function $36 query register

Name	7	6	5	4	3	2	1	0
F36_ADC_Query0	—	—	—	—	—	—	NumChannels	

The bits in register F36_ADC_Query0 are defined as follows:

NumChannels (F36_ADC_Query0, bits 1:0)

‘00’: One conversion channel available.

‘01’: Two conversion channels available.

15.2. Function $36: control register

The control register determines the mode of operation of the ADC channels.

Table 62. Function $36 control register

Name	7	6	5	4	3	2	1	0
F36_ADC_Ctrl0	—	—	—	—	—	—	Enable1	Enable0

The fields in this control register are defined as follows:

Enable0 (F36_ADC_Ctrl0, bit 0)

Enables continuous conversion by ADC channel 0.

Enable1 (F36_ADC_Ctrl0, bit 1)

Enables continuous conversion by ADC channel 1.

When enabled, continuous conversion takes place at the report rate only while the chip is fully awake. Specifically, the chip is fully awake when the capacitance sensors detect a finger, any RMI register is written to, or the No-Sleep bit is set. When the chip goes into Doze mode, ADC conversions are not performed regardless of the *F36_ADC_Ctrl0* register setting. The Function $36 command register provides a means to force an ADC conversion during Doze mode.

15.3. Function $36: data registers

The data registers contain the results of the conversions. There will be a minimum of one data register but the number available on a given sensor module corresponds to the number of channels available as reported by the query register. The example below is for a sensor module with two channels.

Table 63. Function $36 data registers, two-channel example

Name	7	6	5	4	3	2	1	0
F36_ADC_Data0.0	ConversionResult							
F36_ADC_Data0.1	ConversionResult							

ConversionResult (F36_ADC_Data0.)*

Reports the result of the conversion.

15.4. Function $36: command registers

The command register initiates conversions by the ADC channels.

Table 64. Function $36 command register

Name	7	6	5	4	3	2	1	0
F36_ADC_Cmd0	—	—	—	—	—	—	Conv1	Conv0

The fields in this register are defined as follows:

Conv0 (F36_ADC_Cmd0, bit 0)

Start conversion on ADC channel 0.

Conv1 (F36_ADC_Cmd0, bit 1)

Start conversion on ADC channel 1.

Writing to these bits provides a means to force an ADC conversion during Doze mode. When the ADC conversion completes, the respective command bit is automatically cleared to ‘0’. The clearing of the bit can thus be used as an indicator to gauge when conversions are complete. An interrupt is also available to notify the host when an ADC conversion completes; while the chip is fully awake and continuous ADC conversion is enabled, interrupts will occur at the report rate.

15.5. Function $36: interrupt source

The data registers defined by Function $36 are associated with a single interrupt request source, called the ADC interrupt request. Any product that includes Function $36 allocates an ADC interrupt request bit in the Interrupt Status register (see Function $01), and an ADC interrupt enable bit in the Interrupt Enable register (see Function $01). The position of the ADC interrupt bits in those registers is product-specific; consult the product-specific documentation to determine their location.

16. Function $54: Test reporting

Function $54 provides test reporting and analog diagnostic and control functions that allow direct visibility into image sensing.

Function $54 provides direct access to low-level capacitance data for testing and tuning purposes. Function $54 also provides an interface to the analog data acquisition and noise mitigation mechanisms.

The image data reported by Function $54 is a two-dimensional array of pixels. Each pixel is a signed integer values which corresponds to an intersection in the grid of transmitter and receiver electrodes on the face of the capacitive sensor.

Operation in normal reporting modes after or during the use of Function $54 is not supported. A Reset command must be issued to return the sensor to normal operation.

16.1. Function $54: query registers

The query registers describe the device's capabilities.

Table 65. Function $54 query registers

Name	7	6	5	4	3	2	1	0
F54_AD_Query0	NumberOfReceiverElectrodes							
F54_AD_Query1	NumberOfTransmitterElectrodes							
F54_AD_Query2	—	Has Image16	—	—	Has Image8	Has Baseline	—	—
F54_AD_Query3.0	ClockRateLSB [7:0]							
F54_AD_Query3.1	ClockRateMSB [15:8]							
F54_AD_Query4	TouchControllerFamily							
F54_AD_Query5	—							HasPixel Touch Threshold Adjustment
F54_AD_Query6	Has Relaxation Control	HasOne Byte Report Rate	HasTwo Byte Report Rate	HasLow Power Control	Has Firmware Noise Mitigation	Has Sense Frequency Control	Has Interference Metric	Has Sensor Assignment
F54_AD_Query7	—						Curve CompensationMode	
F54_AD_Query8	—	HasPer-Frequency Noise Control	HasEdge Compensation	HasTouch Hysteresis	HasCmn Maximum	HasCmn Removal	HasIIR Filter	—
F54_AD_Query9	—	HasStatus	Has0D Acquisition Control	Has0D Relaxation Control	HasBurst Span Metric	Has Multi Transmitter Drive	Has MultiMetric State Machine	HasForceFast Relaxation
F54_AD_Query10	—							
F54_AD_Query11	—							
F54_AD_Query12	—				NumberOfSensingFrequencies			
F54_AD_Query13	—				BurstsPerCluster			

16.1.1. F54_AD_Query0 and F54_AD_Query1: number of electrodes

NumberOfReceiverElectrodes (F54_AD_Query0)

Reports the number of available receiver electrodes.

NumberOfTransmitterElectrodes (F54_AD_Query1)

Reports the number of available transmitter electrodes.

16.1.2. F54_AD_Query2: image reporting modes

Reserved (F54_AD_Query2, bits 1:0)

Reserved.

HasBaseline (F54_AD_Query2, bit 2)

If this bit reports as ‘1’, the device is capable of reporting its baseline. The baseline is the measurement of the background capacitance.

HasImage8 (F54_AD_Query2, bit 3)

If this bit reports as ‘1’, the device is capable of reporting 8-bit capacitive image data. These images can be read more quickly than 16-bit images, but it is possible that they might be distorted because of the lower resolution.

Reserved (F54_AD_Query2, bits 5:4)

Reserved.

HasImage16 (F54_AD_Query2, bit 6)

If this bit reports as ‘1’, the device is capable of reporting 16-bit capacitive image data.

Reserved (F54_AD_Query2, bit 7)

Reserved.

16.1.3. F54_AD_Query3.0/3.1: clock rate

This register pair reports the master clock rate of the device.

ClockRateLSB (F54_AD_Query3.0)

ClockRateMSB (F54_AD_Query3.1)

These registers report the master clock rate of the device in units of 10 kHz. For example, a device with a 16 MHz master clock would report the value 1600 (0x0640).

16.1.4. F54_AD_Query4: touch controller family

TouchControllerFamily (F54_AD_Query4)

This register contains a code that defines the touch controller family. When this register reports as ‘00’, registers F54_AD_Ctrl1, F54_AD_Ctrl4, F54_AD_Ctrl5, F54_AD_Ctrl6, F54_AD_Ctrl8, and F54_AD_Ctrl9 exist. When this register reports as ‘01’, registers F54_AD_Ctrl1 and F54_AD_Ctrl4 through F54_AD_Ctrl9 exist.

16.1.5. F54_AD_Query5: analog hardware controls

HasPixelTouchThresholdAdjustment (F54_AD_Query5, bit 0)

When this bit reports as ‘1’, register F54_AD_Ctrl3 exists.

Reserved (F54_AD_Query5, bits 7:1)

Reserved.

16.1.6. F54_AD_Query6: data acquisition

HasSensorAssignment (F54_AD_Query6, bit 0)

This bit indicates the presence of a mechanism for specifying the number and arrangement of transmitter and receiver electrodes used by a sensor.

When this bit reports as ‘1’, registers F54_AD_Ctrl14, F54_AD_Ctrl15.*, and F54_AD_Ctrl16.* exist.

HasInterferenceMetric (F54_AD_Query6, bit 1)

This bit indicates the presence of a mechanism for estimating the amount of electrical noise present in the image data.

When this bit reports as '1', registers F54_AD_Ctrl10 and F54_AD_Data6 exist.

HasSenseFrequencyControl (F54_AD_Query6, bit 2)

This bit indicates the presence of a mechanism for changing the sensing frequency in order to avoid narrow-band noise sources.

When this bit reports as '1', registers F54_AD_Query12, F54_AD_Ctrl17 through F54_AD_Ctrl19, F54_AD_Ctrl21.*, F54_AD_Ctrl28.*, and F54_AD_Data4 exist.

HasFirmwareNoiseMitigation (F54_AD_Query6, bit 3)

This bit indicates the presence of a firmware-based mechanism to mitigate the effects of environmental noise.

When this bit reports as '1', registers F54_AD_Ctrl22 through F54_AD_Ctrl26 exist.

HasLowPowerControl (F54_AD_Query6, bit 4)

HasTwoByteReportRate (F54_AD_Query6, bit 5)

This bit indicates the ability of the device to report its instantaneous report rate.

When this bit reports as '1', registers F54_AD_Data7.0 and F54_AD_Data7.1 exist.

HasOneByteReportRate (F54_AD_Query6, bit 6)

This bit indicates the ability of the device to report its instantaneous report rate.

When this bit reports as '1', register F54_AD_Data7.0 exists.

HasRelaxationControl (F54_AD_Query6, bit 7)

This bit indicates the ability of the device to compensate for thermal effects in the system. When this bit reports as '1', registers F54_AD_Ctrl12 and F54_AD_Ctrl13 exist.

16.1.7. F54_AD_ Query7: curved lens compensation

CurveCompensationMode (F54_AD_Query7, bits 1:0)

This field reports the device's ability to compensate for curved lenses:

'00' = No curve compensation.
'01' = Single-axis curve compensation.
'10' = Dual-axis curve compensation.
'11' = Reserved.

When this field reports as '01', register F54_AD_Ctrl36 exists.

When this field reports as '10', registers F54_AD_Ctrl36 and F54_AD_Ctrl37 exist.

Reserved (F54_AD_Query7, bits 7:2)

Reserved.

16.1.8. F54_AD_Query8: data acquisition post-processing controls

Reserved (F54_AD_Query8, bit 0)

Reserved.

HasIIRFilter (F54_AD_Query8, bit 1)

This bit indicates the presence of an IIR filter that may be applied to every sampled image before it is processed.

When this bit reports as '1', register F54_AD_Ctrl27 exists.

HasCmnRemoval (F54_AD_Query8, bit 2)

This bit indicates the presence of algorithms to remove common-mode noise (CMN).

When this bit reports as '1', register F54_AD_Ctrl29 exists.

HasCmnCapScaleFactor (F54_AD_Query8, bit 3)

This bit indicates whether the Common Mode Noise cap may be adjusted.

When this bit reports as '1', register F54_AD_Ctrl30 exists.

HasPixelThresholdHysteresis (F54_AD_Query8, bit 4)

This bit indicates whether the Pixel Threshold hysteresis may be adjusted.

When this bit reports as '1', register F54_AD_Ctrl31 exists.

HasEdgeCompensation (F54_AD_Query8, bit 5)

This bit indicates the presence of a mechanism for boosting the capacitive signal on the edges of the sensor.

When this bit reports as '1', F54_AD_Ctrl32.*, F54_AD_Ctrl33.*, F54_AD_Ctrl34.*, and F54_AD_Ctrl35.* exist.

HasPerFrequencyNoiseControl (F54_AD_Query8, bit 6)

When this bit reports as '1', F54_AD_Ctrl38.*, F54_AD_Ctrl39.*, and F54_AD_Ctrl40.* exist.

Reserved (F54_AD_Query8, bit 7)

Reserved.

16.1.9. F54_AD_Query9: multi-metric firmware noise mitigation

HasForceFastRelaxation (F54_AD_Query9, bit 0)

When this bit reports as '1', *ForceFastRelaxation* (F54_AD_Ctrl0, bit 2) exists.

HasMultiMetricStateMachine (F54_AD_Query9, bit 1)

When this bit reports as '1', F54_AD_Ctrl43 through F54_AD_Ctrl54, F54_AD_Data9, and F54_AD_Data10 exist.

HasMultiTransmitterDrive (F54_AD_Query9, bit 2)

When *HasMultiTransmitterDrive* reports as '1' or *HasBurstSpanMetric* reports as '1', F54_AD_Ctrl41 exists.

HasBurstSpanMetric (F54_AD_Query9, bit 3)

When *HasMultiTransmitterDrive* reports as '1' or *HasBurstSpanMetric* reports as '1', F54_AD_Ctrl41 exists.

When *HasBurstSpanMetric* reports as '1', F54_AD_Ctrl2 and F54_AD_Data8.

Has0-DRelaxationControl (F54_AD_Query9, bit 4)

When this bit reports as '1', registers F54_AD_Ctrl55 and F54_AD_Ctrl56 exist.

Has0-DAcquisitionControl (F54_AD_Query9, bit 5)

When this bit reports as '1', registers F54_AD_Ctrl57 and F54_AD_Ctrl58 exist.

HasStatus (F54_AD_Query9, bit 6)

When this bit reports as '1', register F54_AD_Data11 exists.

Reserved (F54_AD_Query9, bit 7)

Reserved.

16.1.10. F54_AD_10 and F54_AD_11: reserved

Reserved (F54_AD_Query10)

Reserved (F54_AD_Query11)

Reserved.

16.1.11. F54_AD_Query12: sense frequency control

This query register is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as '1'.

NumberOfSensingFrequencies (F54_AD_Query12, bits 3:0)

This field reports the number of available sensing frequencies. The range is 1 to 15.

Reserved (F54_AD_Query12, bits 7:4)

Reserved.

16.1.12. F54_AD_Query13: bursts per cluster

BurstsPerCluster (F54_AD_Query13, bits 3:0)

This field reports the number of bursts per cluster.

Reserved (F54_AD_Query13, bits 7:4)

Reserved.

16.2. Function $54: control registers

Configurations done through Design Studio 4 are written into Function $54 control registers. The following table lists the entire Function $54 control register space.

Table 66. Function $54 control registers

Name	7	6	5	4	3	2	1	0
F54_AD_Ctrl0	—					ForceFast Relaxation	NoScan	NoRelax
F54_AD_Ctrl1	—				BurstsPerCluster			
F54_AD_Ctrl2.0	SaturationCapacitanceLSB							
F54_AD_Ctrl2.1	SaturationCapacitanceMSB							
F54_AD_Ctrl3	PixelTouchThreshold							
F54_AD_Ctrl4	—						ReceiverFeedback Capacitance	
F54_AD_Ctrl5	—			Low Reference Polarity	LowReference Feedback Capacitance		LowReferenceCapacitance	
F54_AD_Ctrl6	—			High Reference Polarity	HighReference Feedback Capacitance		HighReferenceCapacitance	
F54_AD_Ctrl7	—			CBC Transmitter Carrier Selection	CBCPolarity		CBCCapacitance	
F54_AD_Ctrl8	IntegrationDuration [7:0]							
F54_AD_Ctrl8.1	—						IntegrationDuration [bits 9:8]	
F54_AD_Ctrl9	ResetDuration							
F54_AD_Ctrl10	—				NoiseSensingBurstsPerImage			
F54_AD_Ctrl11	—							
F54_AD_Ctrl12	SlowRelaxationRate							
F54_AD_Ctrl13	FastRelaxationRate							
F54_AD_Ctrl14	—						Curve Compensation On Transmitters	Receivers OnX-Axis
F54_AD_Ctrl15.*	SensorReceiverAssignment							
F54_AD_Ctrl16.*	SensorTransmitterAssignment							
F54_AD_Ctrl17.*	FilterBandwidth			—	Disable	BurstCountMSB		
F54_AD_Ctrl18.*	BurstCountLSB							
F54_AD_Ctrl19.*	StretchDuration							
F54_AD_Ctrl20	—							Disable Noise Mitigation
F54_AD_Ctrl21.0	FrequencyShiftNoiseThresholdLSB							
F54_AD_Ctrl21.1	FrequencyShiftNoiseThresholdMSB							
F54_AD_Ctrl22	NoiseDensityThreshold							
F54_AD_Ctrl23.0	MediumNoiseThresholdLSB							
F54_AD_Ctrl23.1	MediumNoiseThresholdMSB							
F54_AD_Ctrl24.0	HighNoiseThresholdLSB							
F54_AD_Ctrl24.1	HighNoiseThresholdMSB							

Name	7	6	5	4	3	2	1	0
F54_AD_Ctrl25	NoiseDensity							
F54_AD_Ctrl26	FrameCount							
F54_AD_Ctrl27	IIRFilterCoefficient							
F54_AD_Ctrl28.0	QuietThresholdLSB							
F54_AD_Ctrl28.1	QuietThresholdMSB							
F54_AD_Ctrl29	CMN Filter Disable	—						
F54_AD_Ctrl30	CMNFilterMaximum							
F54_AD_Ctrl31	TouchHysteresis							
F54_AD_Ctrl32.0	ReceiverLowEdgeCompensationLSB							
F54_AD_Ctrl32.1	ReceiverLowEdgeCompensationMSB							
F54_AD_Ctrl33.0	ReceiverHighEdgeCompensationLSB							
F54_AD_Ctrl33.1	ReceiverHighEdgeCompensationMSB							
F54_AD_Ctrl34.0	TransmitterLowEdgeCompensationLSB							
F54_AD_Ctrl34.1	TransmitterLowEdgeCompensationMSB							
F54_AD_Ctrl35.0	TransmitterHighEdgeCompensationLSB							
F54_AD_Ctrl35.1	TransmitterHighEdgeCompensationMSB							
F54_AD_Ctrl36.*	Axis1Compensation							
F54_AD_Ctrl37.*	Axis2Compensation							
F54_AD_Ctrl38.*	NoiseControl1							
F54_AD_Ctrl39.*	NoiseControl2							
F54_AD_Ctrl40.*	NoiseControl3							
F54_AD_Ctrl41	—						NoBurst SpanMetric	NoMulti TransDrive
F54_AD_Ctrl42.0	BurstSpanMetricThresholdLSB							
F54_AD_Ctrl42.1	BurstSpanMetricThresholdMSB							
F54_AD_Ctrl43	—							
...	...							
F54_AD_Ctrl54	—							
F54_AD_Ctrl55	0-DSlowRelaxationRate							
F54_AD_Ctrl56	0-DFastRelaxationRate							
F54_AD_Ctrl57				CBC-Transmitter Carrier Select	CBC Polarity	CBCCapacitiance		
F54_AD_Ctrl58	—					0-DAcquisitionScheme		

16.2.1. F54_AD_Ctrl0: general control 0

NoRelax (F54_AD_Ctrl0, bit 0)

Setting this bit to ‘1’ disables the thermal compensation mechanisms.

NoScan (F54_AD_Ctrl0, bit 1)

Setting this bit to ‘1’ causes the device to stop driving the transmitter electrodes.

ForceFastRelaxation (F54_AD_Ctrl0, bit 2)

By setting this register, the host forces fast relaxation regardless of the firmware decision. The rate is specified in *FastRelaxationRate* (F54_AD_Ctrl13). Setting it to 0 lets the firmware decide the relaxation rate. It will pick the appropriate rate out of *SlowRelaxationRate* (F54_AD_Ctrl12) and *FastRelaxationRate* (F54_AD_Ctrl13). This applies for both 2-D and 0-D relaxation. Changes in this register do not take effect immediately, but within the next 'force update'.

Reserved (F54_AD_Ctrl0, bits 7:3)

Reserved.

16.2.2. F54_AD_Ctrl1: general control 1

This control register is only present if *TouchControllerFamily* (F54_AD_Query4) reports '00' or '01'.

BurstsPerCluster (F54_AD_Ctrl1, bits 3:0)

Each pixel is sampled multiple times per image report. This field controls the number of samples.

Reserved (F54_AD_Ctrl1, bits 7:4)

Reserved.

16.2.3. F54_AD_Ctrl2.0/2.1: saturation capacitance

F54_AD_Ctrl2 only exists when *HasBurstSpan* (F54_AD_Query9, bit 3) is set to '1'.

The saturation capacitance is the expected difference between a typical pixel's baseline capacitance and its capacitance when completely covered by a conductive object.

SaturationCapacitanceLSB (F54_AD_Ctrl2.0)

SaturationCapacitanceMSB (F54_AD_Ctrl2.1)

Saturation capacitance in femtofarads.

16.2.4. F54_AD_Ctrl3: pixel touch threshold

This register is only present if *HasPixelTouchThresholdAdjustment* (F54_AD_Query5, bit 0) reports as '1'.

PixelTouchThreshold (F54_AD_Ctrl3)

This register adjusts basic touch sensitivity. It is a multiplicative scale factor expressed as a 1.7 fixed-point value (range is 0.00 to 1.99). The default is 0x80 (1.00). Low values lower the sensitivity; high values raise it.

16.2.5. F54_AD_Ctrl4: miscellaneous analog control

This register is only present if *AnalogHardwareFamily* (F54_AD_Query4) reports as $00 or $01.

ReceiverFeedbackCapacitance (F54_AD_Ctrl4, bits 1:0)

This field is set to the optimal value by Design Studio 4 during the tuning process.

Reserved (F54_AD_Ctrl4, bits 7:2)

Reserved.

16.2.6. F54_AD_Ctrl5: low reference

This register is only present if *AnalogHardwareFamily* (F54_AD_Query4) reports as $00 or $01.

These fields are set to the optimal values by Design Studio 4 during the tuning process

LowReferenceCapacitance (F54_AD_Ctrl5, bits 1:0)

LowReferenceFeedbackCapacitance (F54_AD_Ctrl5, bits 3:2)

LowReferencePolarity (F54_AD_Ctrl5, bit 4)

Reserved (F54_AD_Ctrl5, bits 7:5)

Reserved.

16.2.7. F54_AD_Ctrl6: high reference

This register is only present if *Analog Hardware Family* (F54_AD_Query4) reports as $00 or $01.

These registers are set to the optimal values by Design Studio 4 during the tuning process.

HighReferenceCapacitance (F54_AD_Ctrl6, bits 1:0)

HighReferenceFeedbackCapacitance (F54_AD_Ctrl6, bits 3:2)

HighReferencePolarity (F54_AD_Ctrl6, bit 4)

Reserved (F54_AD_Ctrl6, bits 7:5)

Reserved.

16.2.8. F54_AD_Ctrl7: coarse baseline correction

This register is only present if *AnalogHardwareFamily* (F54_AD_Query4) reports as $01.

These registers are set to the optimal values by Design Studio 4 during the tuning process.

CBCCapacitance (F54_AD_Ctrl7, bits 2:0)

CBCPolarity (F54_AD_Ctrl7, bit 3)

CBCTransmitterCarrierSelection (F54_AD_Ctrl7, bit 4)

Reserved (F54_AD_Ctrl7, bits 7:5)

Reserved.

16.2.9. F54_AD_Ctrl8.0/8.1: integration duration

These registers define the integration duration. They are only present if *AnalogHardwareFamily* (F54_AD_Query4) reports as $00 or $01.

These registers are set to the optimal values by Design Studio 4 during the tuning process.

IntegrationDurationLSB (F54_AD_Ctrl8.0)

IntegrationDurationMSB (F54_AD_Ctrl8.1, bits 1:0)

Reserved (F54_AD_Ctrl8.1, bits 7:2)

Reserved.

16.2.10. F54_AD_Ctrl9: reset duration

This register is only present if *Analog Hardware Family* (F54_AD_Query4) reports as $00 or $01. This register is set to the optimal value by Design Studio 4 during the tuning process.

ResetDuration (F54_AD_Ctrl9)

Defines the reset duration.

16.2.11. F54_AD_Ctrl10: noise sensing bursts per image

This register is only present if *HasInterferenceMetric* (F54_AD_Query6, bit 1) reports as '1'.

NoiseSensingBurstsPerImage (F54_AD_ Ctrl10, bits 3:0)

Determines the number of noise sensing bursts per image. Increasing this value from the default may make noise estimation more accurate, at the cost of a reduced time for sensing fingers. Reducing this value from the default is not recommended.

Reserved (F54_AD_Ctrl10, bits 7:4)

Reserved.

16.2.12. F54_AD_Ctrl11: reserved

This register is reserved.

16.2.13. F54_AD_Ctrl12: slow relaxation rate

This register is only present if *HasRelaxationControl* (F54_AD_Query6, bit 7) reports as '1'.

SlowRelaxationRate (F54_AD_Ctrl12)

The slow-relaxation mechanism compensates for slow environmental disturbances like thermal changes.

This register holds a 4.4 fixed-point value representing femtofarads per second. For example, the value 0x18 represents 1.5 femtofarads per second.

16.2.14. F54_AD_Ctrl13: fast relaxation rate

This register is only present if *HasRelaxationControl* (F54_AD_Query6, bit 7) reports as '1'.

FastRelaxationRate (F54_AD_Ctrl13)

The fast-relaxation mechanism compensates for fast environmental disturbances.

This register holds a value representing femtofarads per second. For example, the value 0x18 represents 24 femtofarads per second.

16.2.15. F54_AD_Ctrl14: sensor assignment properties

This register is only present if *HasSensorAssignment* (F54_AD_Query6, bit 0) reports as '1'.

ReceiversOnXAxis (F54_AD_Ctrl14, bit 0)

When this bit is set to '1', the axis composed of receiver electrodes is the X-axis.

When this bit is set to '0', the axis composed of receiver electrodes is the Y-axis.

CurveCompensationOnTransmitters (F54_AD_Ctrl14, bit 1)

This bit has an effect only when *CurveCompensationMode* (F54_AD_Query7, bits 1:0) reports as ‘01’ (single-axis curve compensation).

When this bit is set to ‘1’, the curved-lens compensation is applied to the axis composed of transmitter electrodes. When this bit is set to ‘0’, the curved-lens compensation is applied to the axis composed of receiver electrodes.

Reserved (F54_AD_Ctrl14, bits 7:2)

Reserved.

16.2.16. F54_AD_Ctrl15.*: sensor receiver assignment

This block of replicated registers is only present if *HasSensorAssignment* (F54_AD_Query6, bit 0) reports as ‘1’.

SensorReceiverAssignment (F54_AD_Ctrl15.)*

These registers define the layout of the receiver electrodes on the sensor. The registers list the receiver electrode numbers in order, with the electrode closest to the origin in F54_AD_Ctrl15.0, the next-closest electrode in F54_AD_Ctrl15.1, and so on.

The length of this replicated register block is defined by *NumberOfReceiverElectrodes (*F54_AD_Query0). Unused registers in the block must be set to 0xFF.

Note: Changing the values of these registers at runtime has no effect.

16.2.17. F54_AD_Ctrl16.*: sensor transmitter assignment

This register is only present if *HasSensorAssignment* (F54_AD_Query6, bit 0) reports as ‘1’.

These registers define the layout of the transmitter electrodes on the sensor. The registers list the transmitter electrode numbers in order, with the electrode closest to the origin in F54_AD_Ctrl16.0, the next-closest electrode in F54_AD_Ctrl16.1, and so on.

The length of this replicated register block is defined by *NumberOfTransmitterElectrodes (*F54_AD_Query1). Unused registers in the block must be set to 0xFF.

Note: Changing the values of these registers at runtime has no effect.

16.2.18. F54_AD_Ctrl17.*: sense frequency control 1

This block of replicated registers is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as ‘1’. The length of this replicated register block is equal to *NumberOfSensingFrequencies* (F54_AD_Query12, bits 3:0).

These fields are set to the optimal values by Design Studio 4 during the tuning process.

BurstCountMSB (F54_AD_Ctrl17., bits 2:0)*

Disable (F54_AD_Ctrl17., bit 3)*

Reserved (F54_AD_Ctrl17., bit 4)*

FilterBandwidth (F54_AD_Ctrl17., bits 7:5)*

16.2.19. F54_AD_Ctrl18.*: sense frequency control 2

This block of replicated registers is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as '1'.

The length of this replicated register block is equal to *NumberOfSensingFrequencies* (F54_AD_Query12, bits 3:0).

BurstCountLSB (F54_AD_Ctrl18.)*

This register is set to the optimal value by Design Studio 4 during the tuning process.

16.2.20. F54_AD_Ctrl19.*: stretch duration

This block of replicated registers is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as '1'.

The length of this replicated register block is equal to *NumberOfSensingFrequencies* (F54_AD_Query12, bits 3:0).

StretchDuration (F54_AD_Ctrl19.)*

This register is set to the optimal value by Design Studio 4 during the tuning process.

16.2.21. F54_AD_Ctrl20: noise mitigation general control

DisableNoiseMitigation (F54_AD_Ctrl20, bit 0)

When this bit is set to '1', the device is prevented from mitigating the effects of environmental noise.

Reserved (F54_AD_Ctrl20 bits 7:1)

Reserved.

16.2.22. F54_AD_ Ctrl21.0/21.1: HNM frequency shift noise threshold

This register pair is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as '1'.

FrequencyShiftNoiseThresholdLSB (F54_AD_Ctrl21.0)

FrequencyShiftNoiseThresholdMSB (F54_AD_Ctrl21.1)

An Interference Metric, representing the amount of noise present, is calculated for every image. These registers define the Interference Metric threshold above which the image data will be discarded and the sense frequency changed while Hardware Noise Mitigation is active.

This is a 16-bit unsigned value. Higher values allow more noise before discarding data and changing the sense frequency.

16.2.23. F54_AD_Ctrl22: hardware noise mitigation exit density

This register is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

NoiseDensityThreshold (F54_AD_Ctrl22)

This register defines the point at which the transition from Hardware Noise Mitigation to Firmware Noise Mitigation occurs.

When the fraction of images that are discarded exceeds *NoiseDensityThreshold*/255, the system switches from Hardware Noise Mitigation to Firmware Noise Mitigation.

The range is 0 to 255. Higher values allow a larger percentage of images to be discarded before the transition occurs.

16.2.24. F54_AD_Ctrl23.0/23.1: medium noise threshold

This register is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

MediumNoiseThresholdLSB (F54_AD_Ctrl23.0)

MediumNoiseThresholdMSB (F54_AD_Ctrl23.1)

This register, in combination with *FrameCount* (F54_AD_Ctrl26), defines the point at which the transition from Firmware Noise Mitigation to Hardware Noise Mitigation occurs.

An Interference Metric, representing the amount of noise present, is calculated for every image. When *FrameCount* consecutive images have Interference Metric values lower than *MediumNoiseThreshold*, the system switches from Firmware Noise Mitigation to Hardware Noise Mitigation.

The range is 0 to 65535. Higher values allow more noise to be present before the transition occurs.

16.2.25. F54_AD_Ctrl24.0/24.1: high noise threshold

This register is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

HighNoiseThresholdLSB (F54_AD_Ctrl24.0)

HighNoiseThresholdMSB (F54_AD_Ctrl24.1)

An Interference Metric, representing the amount of noise present, is calculated for every image. These registers define the Interference Metric threshold above which the image data will be discarded while Firmware Noise Mitigation is active.

This is a 16-bit unsigned value. Higher values allow more noise before discarding data. .

16.2.26. F54_AD_Ctrl25: FNM frequency shift density

This register is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

NoiseDensity (F54_AD_Ctrl25)

This register, in combination with *QuietThreshold* (F54_AD_Ctrl28.*), defines the point at which the sense frequency will be changed while Firmware Noise Mitigation is active.

An Interference Metric, representing the amount of noise present, is calculated for every image. When the fraction of "noisy" images (those whose Interference Metric is greater than *QuietThreshold*) exceeds *NoiseDensity*/255, the sense frequency is changed.

The range is 0 to 255. Higher values allow a larger percentage of noisy images before the sense frequency is changed.

16.2.27. F54_AD_Ctrl26: firmware noise mitigation exit threshold

This register is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

FrameCount (F54_AD_Ctrl26)

This register, in combination with *MediumNoiseThreshold* (F54_AD_Ctrl23.*), defines the point at which the transition from Firmware Noise Mitigation to Hardware Noise Mitigation occurs.

An Interference Metric, representing the amount of noise present, is calculated for every image.

When *FrameCount* consecutive images have Interference Metric values lower than *MediumNoiseThreshold*, the system switches from Firmware Noise Mitigation to Hardware Noise Mitigation.

The range is 0 to 255.

16.2.28. F54_AD_Ctrl27: IIR filter coefficient

This register is only present if *HasIIRFilter* (F54_AD_Query8, bit 1) reports as '1'.

IIRFilterCoefficient (F54_AD_Ctrl27)

This register specifies the coefficient of an IIR filter applied to each image. The range is 0 to 255. A value of 0 represents no IIR filtering at all. A value approaching 255 represents heavy IIR filtering. Higher values will improve performance in the presence of wideband noise, but at the expense of increased touch latency.

16.2.29. F54_AD_Ctrl28.0/28.1: FNM frequency shift noise threshold

This register pair is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

QuietThresholdLSB (F54_AD_Ctrl28.0)

QuietThresholdMSB (F54_AD_Ctrl28.1)

These registers, in combination with *NoiseDensity* (F54_AD_Ctrl25), define the point at which the sense frequency will be changed while Firmware Noise Mitigation is active.

An Interference Metric, representing the amount of noise present, is calculated for every image. When the fraction of "noisy" images (those whose Interference Metric is greater than *QuietThreshold*) exceeds *NoiseDensity*/255, the sense frequency is changed.

The range is 0 to 65535. Higher values allow more noise to be present before an image is considered "noisy."

16.2.30. F54_AD_Ctrl29: common-mode noise filter control

This register is only present if *HasCmnRemoval* (F54_AD_Query8, bit 2) reports as '1'.

Reserved (F54_AD_Ctrl29, bits 6:0)

Reserved.

CmnFilterDisable (F54_AD_Ctrl29, bit 7)

When this bit is set to '1', the common-mode noise filter is disabled.

16.2.31. F54_AD_Ctrl30: common-mode noise filter maximum

This register is present only when *HasCmnMaximum* (F54_AD_Query8, bit 3) reports as '1'.

CmnFilterMaximum (F54_AD_Ctrl30)

This register is set to the optimal value by Design Studio 4 during the tuning process.

16.2.32. F54_AD_Ctrl31: touch hysteresis

This register is present only when *HasTouchHysteresis (*F54_AD_Query8, bit 4) reports as '1'.

TouchHysteresis (F54_AD_Ctrl31)

This register specifies the touch hysteresis. The default is 0x80.

16.2.33. F54_AD_Ctrl32.0/32.1 through F54_AD_Ctrl35.0/35.1: edge compensation

These registers are present only when *HasEdgeCompensation* (F54_AD_Query8, bit 5) reports as '1'.

Sensors designed with shorter traces along the edges may produce lower capacitance readings at the edges of the sensor than elsewhere on the sensor.

The device can compensate for this effect by applying a gain factor to the pixels on each edge.

ReceiverLowEdgeCompensationLSB (F54_AD_Ctrl32.0)

ReceiverLowEdgeCompensationMSB (F54_AD_Ctrl32.1)

These registers contain a 4.12 fixed-point number representing the gain applied to the pixels which share the receiver electrode closest to the origin. The default is 0x1000 (1.0), no gain.

ReceiverHighEdgeCompensation LSB (F54_AD_Ctrl33.0)

ReceiverHighEdgeCompensation MSB (F54_AD_Ctrl33.1)

These registers contain a 4.12 fixed-point number representing the gain applied to the pixels which share the receiver electrode farthest from the origin. The default is 0x1000 (1.0), no gain.

TransmitterLowEdgeCompensationLSB (F54_AD_Ctrl34.0)

TransmitterLowEdgeCompensationMSB (F54_AD_Ctrl34.1)

These registers contain a 4.12 fixed-point number representing the gain applied to the pixels which share the transmitter electrode closest to the origin. The default is 0x1000 (1.0), no gain.

TransmitterHighEdgeCompensationLSB (F54_AD_Ctrl35.0)

TransmitterHighEdgeCompensationMSB (F54_AD_Ctrl35.1)

These registers contain a 4.12 fixed-point number representing the gain applied to the pixels which share the transmitter electrode farthest from the origin. The default is 0x1000 (1.0), no gain.

16.2.34. F54_AD_Ctrl36.*: axis 1 compensation

Axis1Compensation (F54_AD_Ctrl36.)*

This block of replicated registers is present only when *CurveCompensationMode* (F54_AD_Query7, bits 1:0) reports as '01' (single-axis curve compensation) or '10' (dual-axis curve compensation).

If *CurveCompensationMode* reports as '10' (dual-axis curve compensation), the size of this block is equal to *NumberOfReceiverElectrodes* (F54_AD_Query0) and these registers hold the correction factors for the axis composed of receiver electrodes.

If *CurveCompensationMode* reports as '01' (single-axis curve compensation), the size of this block is equal to the larger of *NumberOfReceiverElectrodes* (F54_AD_Query0) and *NumberOfTransmitterElectrodes* (F54_AD_Query1), and these registers hold the correction factors for the axis specified by *CurveCompensationOnTransmitters* (F54_AD_Ctrl14, bit 1).

These registers are set to the optimal values by Design Studio 4 during the tuning process.

Note: Changes to these registers take effect only when the *ForceUpdate* command (F54_AD_Cmd0, bit 2) is executed.

16.2.35. F54_AD_Ctrl37.*: axis 2 compensation

Axis2Compensation (F54_AD_Ctrl37.)*

This block of replicated registers is present only when *CurveCompensationMode* (F54_AD_Query7, bits 1:0) reports as '10' (dual-axis curve compensation). The size of this block is equal to *NumberOfTransmitterElectrodes* (F54_AD_Query1).

These registers hold the correction factors for the axis composed of transmitter electrodes. They are set to the optimal values by Design Studio 4 during the tuning process.

Note: Changes to these registers take effect only when the *ForceUpdate* command (F54_AD_Cmd0, bit 2) is executed.

16.2.36. F54_AD_Ctrl38.* through F54_AD_Ctrl40.*: per frequency noise control

Registers F54_AD_Ctrl38.* through F54_AD_Ctrl40.* control silicon aspects of the noise rejection mechanisms.

These blocks of replicated registers are only present if *HasPerFrequencyNoiseControl* (F54_AD_Query8, bit 6) reports as '1'. The length of each block is equal to *NumberOfSensing Frequencies* (F54_AD_Query12, bits 3:0).

These registers are set to the optimal values by Design Studio 4 during the tuning process. They should not be modified.

NoiseControl1 (F54_AD_Ctrl38.)*

NoiseControl2 (F54_AD_Ctrl39.)*

NoiseControl3 (F54_AD_Ctrl40.)*

16.2.37. F54_AD_Ctrl41: multi-metric firmware noise mitigation control

This register is only present if *HasMultiTransmitterDrive* (F54_AD_Query9, bit 2) reports as '1' or *HasBurstSpanMetric* (F54_AD_Query9, bit 3) reports as '1'.

NoMultiTransmitterDrive (F54_AD_Ctrl41, bit 0)

Setting this bit to '1' disables the multi-transmitter drive.

NoBurstSpanMetric (F54_AD_Ctrl41, bit 1)

Setting this bit to '1' disables the burst span metric.

Reserved (F54_AD_Ctrl41, bits 7:2)

Reserved.

16.2.38. F54_AD_Ctrl42.0/42.1: burst span metric threshold

These registers are only present if *HasMultiMetricStateMachine* (F54_AD_Query9, bit 1) reports as '1'.

BurstSpanMetricThreholdLSB (F54_AD_Ctrl42.0)

BurstSpanMetricThreholdMSB (F54_AD_Ctrl42.1)

These registers define the point at which the Noise Mitigation State is changed. A Burst Span Metric, representing the amount of noise variation, is calculated for each image. When the "noisy" images appear (those whose Burst Span Metric is greater than the *Burst Span Metric Threshold)*, the Noise Mitigation State is changed.

The range is 0 to 65535. Higher values allow more noise to be present before an image is considered "noisy."

16.2.39. F54_AD_Ctrl43 though F54_AD_Ctrl54: reserved

These registers are reserved.

16.2.40. F54_AD_Ctrl55: 0-D slow relaxation

This register is only present if *Has0DRelaxation* (F54_AD_Query9, bit 4) reports as '1'.

0-DSlowRelaxation (F54_AD_Ctrl55)

The 'slow' relaxation mechanism is used to compensate for thermal changes to 0-D Buttons. This register represents a 4.4 fixed-point value measured in femtofarads per second. For example, the value 0x18 would represent 1.5 femtofarads per second.

16.2.41. F54_AD_Ctrl56: 0-D fast relaxation

This register is only present if *Has0DRelaxation* (F54_AD_Query9, bit 4) reports as '1'.

0-DFastRelaxation (F54_AD_Ctrl56)

The 'fast' relaxation mechanism is used to compensate for events that may have corrupted the baseline for 0-D buttons. The baseline represents the basic measurement of the background capacitance for the sensing system. This register represents a 8.0 fixed-point value measured in femtofarads per second. For example, the value 0x18 would represent 24 femtofarads per second.

16.2.42. F54_AD_Ctrl57: 0-D acquisition scheme

Note: This register is read-only and can only be adjusted by overwriting the configuration block.

When acquiring data using CDM, this register defines how to acquire the 0-D buttons. If CDM is not enabled, this register is ignored.

0DAcquisitionScheme (F54_AD_Ctrl57, bits 2:0)

There are different ways to acquire 0-D data while running CDM on 2-D. The proper scheme must be selected based on the configuration of the 0-D buttons in relation to the 2-D sensor.

000 – Separate Button Frame (scan)
001 – reserved

010 – Normal CDM
Others - reserved

Reserved (F54_AD_Ctrl57, bits 7:3)

Reserved.

16.2.43. F54_AD_Ctrl58: 0-D CBC values

When acquiring data using CDM, this register defines how to set the CBC for 0-D buttons. This only applies when using "Separate Button Frame" in *0-DAcquisitionScheme* (F54_AD_Ctrl57).

CBCCapacitance (F54_AD_Ctrl58, bits 2:0)

The Coarse Baseline Correction (CBC) capacitance setting defines an amount of capacitance to be subtracted from the measured baseline while acquiring 0D capacitive buttons.

000 – 0.0pF
001 – 0.5pF
010 – 1.0pF
011 – 1.5pF
100 – 2.0pF
101 – 2.5pF
110 – 3.0pF
111 – 4.0pF

CBCPolarity (F54_AD_Ctrl58, bit 3)

Defines the polarity of the CBC capacitance setting (0 = negative, 1 = positive).

CBCTransmitterCarrierSelection (F54_AD_Ctrl58, bit 4)

Defines the CBC carrier select control field.

Reserved (F54_AD_Ctrl58, bits 7:5)

Reserved.

16.3. Function $54: data registers

The following sections describe the Function $54 data registers, and also describe how to read test reports generated with Function $54 *ReportType (F54_AD_Data0)*.

16.3.1. Function $54: data registers

Table 67. Function $54 data registers

Name	7	6	5	4	3	2	1	0
F54_AD_Data0	ReportType							
F54_AD_Data1	ReportIndexLSB							
F54_AD_Data2	ReportIndexMSB							
F54_AD_Data3	ReportData							
F54_AD_Data4	Inhibit Frequency Shift	—			SenseFrequencySelection			
F54_AD_Data5								
F54_AD_Data6.0	InterferenceMetricLSB [7:0]							
F54_AD_Data6.1	InterferenceMetricMSB [15:8]							
F54_AD_Data7.0	CurrentReportRateLSB [7:0]							
F54_AD_Data7.1	CurrentReportRateMSB [15:8]							
F54_AD_Data8.0	BurstSpanMetricLSB							
F54_AD_Data8.1	BurstSpanMetricMSB							
F54_AD_Data9	—							
F54_AD_Data10	—							
F54_AD_Data11	—							Transmitter Bank

The bits of these registers are described in the following sections.

16.3.2. F54_AD_Data0: report type

ReportType (F54_AD_Data0)

Specifies the type of data that will be collected and reported in the *ReportData* packet register when the *GetReport* bit (*F54_AD_Cmd0, bit 0)* is set.

ReportType = 0: Reserved

ReportType = 1: Reserved

ReportType = 2: Delta Capacitance Report

Reports a matrix of Delta Capacitance values for each pixel.

Each pixel's delta capacitance is represented by a 16-bit signed value. The number of bytes reported is:

*NumberOfTransmitterElectrodes * NumberOfReceiverElectrodes * 2*

ReportType = 3: Raw Capacitance Report

Reports a matrix of raw capacitance values.

Each pixel's raw capacitance is represented by a 16-bit signed value. The number of bytes reported is:

*NumberOfTransmitterElectrodes * NumberOfReceiverElectrodes **

ReportType = 4: High Resistance (Sensor) Report

Reports three words representing the receiver maximum, the transmitter maximum, and the minimum pixel value. These values point to high resistance areas in the sensor, including edge cracks.

Read 6-bytes:

- Byte 0 and 1 report the maximum receiver read; low byte is 0 and high byte is 1.
- Byte 2 and 3 report the maximum transmitter read.
- Byte 4 and 5 report the minimum of the Image.

Passing values for Rx data: < 0.45

Passing values for Tx data: < 0.45

Passing values for minimum of the image: > -0.4

ReportType =5: Transmitter-to-Transmitter (Sensor or No Sensor) and Tx Vdd (No Sensor) Report

Reports bits of data signifying transmitter to transmitter shorts. This can be used on discrete modules (In-Cell cannot use) and FPCAs (No Sensor) to catch transmitter-to-transmitter shorts.

Each of the 4-bytes contains 1-bit per transmitter: 0x00 if the resistance is high (no short exists), or 0x01 if the resistance is low (a short exists). The number of bytes reported is shown below.

4-bytes for up to 32 possible transmitter channels:

- Byte-0 houses Tx responses Tx7:Tx0
- Byte-1 houses Tx responses Tx15:Tx8
- Byte-2 houses Tx responses Tx23:Tx16
- Byte-3 houses Tx responses Tx31:Tx24

Read 3-bytes for up to 24 transmitters and 4-bytes for up to 32 transmitters, depending on what the Touch Controller physically supports. The current Touch Controller offering handles up to 20 transmitters (Tx19:Tx0).

ReportType = 6: Reserved

Reserved.

ReportType = 7 and 17: Receiver-to-Receiver Report #1

Reports a matrix of raw delta capacitive responses for each receiver. This data can be used to find receiver-to-receiver shorts. Each cell of the matrix is a 16-bit number corresponding to a capacitance in femtofarads (fF): about 1000fF if the resistance is high (no short exists), or a value less than 900fF if the resistance is low (a short does exist).

The number of bytes reported is:

Report Type 7 max available data buffer:

*2 * NumberOfReceiverElectrodes * NumberOfTransmittersElectrodes*

Report Type 17 bytes to read:

*2 * NumberOfReceiverElectrodes * NumberOfTransmittersElectrodes*

Example:
Product with *NumberOfReceiverElectrodes* = 20 and *NumberOfTransmittersElectrodes* = 12

The firmware buffer size will inherently be equal to the number of Rx and Tx's configured; 2* #ofRx * #ofTx = max bytes in buffer.

Report Type 7 will be able to report data for the first 12 Rx's, because of the limited number of Tx's configured in the firmware.

*Bytes to read = 2 * #ofRx * #ofTx = 2 * 20 * 12 = 480*

Report Type 17 will report the rest of the useful data, though will have some "do not cares," which do not have to be read.

*Bytes to read = 2 * #ofRx * (#ofRx - #ofTx) = 2 * 20 * (20 – 12) =2 * 20 * 8 = 320*

Set ForceCal bit once done, so that the Stored baseline is correct for any normal operations that may be run after this test.

Note: Combine the data from the two report type reads to get the best view of the data.

The table below describes data for a product with 6 receivers on the 2-D area. For 2-D, you will parse the data into a 6x6 matrix. The responses in the diagonal elements of the matrix are the receivers under test and should read close to 1000 fF if there are no shorts.

Table 68. Data for a product with 6 receivers

1000fF	≈0-15fF	≈0-15fF	≈0-15fF	≈0-15fF	≈0-15fF
≈0-15fF	**1000fF**	≈0-15fF	≈0-15fF	≈0-15fF	≈0-15fF
≈0-15fF	≈0-15fF	**1000fF**	≈0-15fF	≈0-15fF	≈0-15fF
≈0-15fF	≈0-15fF	≈0-15fF	**1000fF**	≈0-15fF	≈0-15fF
≈0-15fF	≈0-15fF	≈0-15fF	≈0-15fF	**1000fF**	≈0-15fF
≈0-15fF	≈0-15fF	≈0-15fF	≈0-15fF	≈0-15fF	**1000fF**

Passing values: All channels have between 0.9pF and 1.1pF; (±10% of 1.0pF).

ReportType = 8: Reserved

Reserved.

ReportType = 9: True Baseline Report

Reports a matrix of raw capacitance values which the firmware uses to calculate the delta capacitance.

Each pixel's baseline capacitance is represented by a 16-bit signed value. The number of bytes reported is:

*NumberOfTransmitterElectrodes * NumberOfReceiverElectrodes * 2*

ReportType = 10: Reserved

Reserved.

ReportType = 11: Reserved

Reserved.

ReportType = 12: Reserved

Reserved.

ReportType = 13: Full Raw Capacitance Maximum/Minimum (Sensor) Report

Reports two words representing the maximum and minimum values respectively of the Full Raw Capacitance image. This aides in detecting Rx-Tx shorts on FPCA's and Rx-Tx, Rx-Gnd and Tx-Gnd on Modules.

Read 4-bytes:

Byte 0 and 1 report the maximum pixel value; low byte is 0 and high byte is 1.

Byte 2 and 3 report the minimum pixel value.

ReportType = 14 and 18: Receiver Opens (No Sensor) Report

Reports a matrix of data that will point to receiver opens. This test requires that a grounded conductive plate make contact with all of the receivers. The plate should be on the receiver channels before running the test. Conductive plate can make contact with ground, receiver, and transmitter ACF pads at the same time without issue. Passing receivers will read close to 0fF. Failing receivers will read close to 1000fF, ±10%.

The data reported is represented by a matrix of 16-bit signed numbers, equivalent to capacitance in (fF). The number of bytes to report is:

Report Type 14 max available data buffer:

*2 * NumberOfReceiverElectrodes * NumberOfTransmittersElectrodes*

Report Type 18 bytes to read:

*2 * NumberOfReceiverElectrodes * NumberOfTransmittersElectrodes*

Example:
Product with *NumberOfReceiverElectrodes* = 20 and *NumberOfTransmittersElectrodes* = 12

The firmware buffer size will inherently be equal to the number of receivers and transmitters; 2* #ofRx * #ofTx = max bytes in the buffer.

Report Type 14 will be able to report data for the first 12 receivers, because of the limit due to the number of transmitters configured in the firmware.

Bytes to read = 2 * #ofRx * #ofTx = 2 * 20 * 12 = 480

Report Type 18 will report the rest of the useful data, though will have some 'don't cares' as well, which do not have to be read.

*Bytes to read = 2 * #ofRx * (#ofRx - #ofTx) = 2 * 20 * (20 – 12) =2 * 20 * 8 = 320*

Set ForceCal bit once done, so that the stored baseline is correct for any normal operations that may run after this test.

Combine the data from the two report type reads to get the best view of the data.

The end user will see a 2-D array of data starting at FIFO index 0,0. The user should see a capacitance response lower than 0.90pF, as well on all receivers that are intact on each row of data. The table below represents results for a unit with only three receivers.

The expected response is below 0.90pF for the receiver under test, which appears as a diagonal down the table, and the others should read near the same range.

Table 69.

0.08pF	0.08pF	0.08pF
0.08pF	**0.08pF**	0.09pF
0.06pF	0.07pF	**0.06pF**

Passing values: all channels have less than 0.9pF.

ReportType = 15: Transmitter Open (No Sensor) Report

Reports bits of data signifying transmitter-to-ground shorts.

A grounded conductive plate is required to make contact with all of the transmitters on the ACF pads. This should be on the pads before running the test. Conductive plate can make contact with ground, receiver, and transmitter ACF pads at the same time without issue.

Each of the 4-bytes houses 1-bit per transmitter: a value of '0' on all active transmitters means low resistance (no opens), or a value of '1' on any of the transmitters means there is high resistance (an open exists). The number of bytes reported is:

4-bytes for up to 32 possible transmitter channels:

- Byte-0 houses Tx responses Tx7:Tx0
- Byte-1 houses Tx responses Tx15:Tx8
- Byte-2 houses Tx responses Tx23:Tx16
- Byte-3 houses Tx responses Tx31:Tx24

Read 3-bytes for up to 24 transmitters and 4-bytes for up to 32 transmitters, depends on what the ASIC physically supports. Current ASIC offering handles up to 20 transmitters (Tx19:Tx0).

ReportType = 16: Transmitter-to-Ground (No Sensor)Report

Reports bits of data signifying transmitter-to-ground shorts.

Each of the 4-bytes contains 1-bit per transmitter: a value of '1' on all active transmitters means high resistance (no shorts), and a value of '0' on any of the transmitters means there is low resistance (a short exists). The number of bytes reported is:

4-bytes for up to 32 possible transmitter channels:

- Byte-0 houses Tx responses Tx7:Tx0
- Byte-1 houses Tx responses Tx15:Tx8

- Byte-2 houses Tx responses Tx23:Tx16
- Byte-3 houses Tx responses Tx31:Tx24

Read 3-bytes for up to 24 transmitters and 4-bytes for up to 32 transmitters, depending on what the Touch Controller physically supports. The current Touch Controller offering handles up to 20 transmitters (Tx19:Tx0).

ReportType = 19: Full Raw Capacitance Report

Reports a matrix of raw capacitance values with the reference capacitance values set wide open (-2 to +6pF).

Each pixel's full raw capacitance is represented by a 16-bit signed value. The number of bytes reported is:

*NumberOfTransmitterElectrodes * NumberOfReceiverElectrodes * 2*

ReportType = 20: Full Raw Capacitance with Receiver Offset Removed

Reports a matrix of raw capacitance values with the reference capacitance values set wide open (-2 to +6pF). Rx-offsets removed means removing the inherent coupling of the traces due to the design. This provides a clear view of the capacitance of the module alone.

Each pixel's full raw capacitance is represented by a 16-bit signed value. The number of bytes reported is:

*NumberOfTransmitterElectrodes * NumberOfReceiverElectrodes * 2*

ReportType = 21 through 255: Reserved

16.3.3. F54_AD_Data1 and F54_AD_Data2: report index

ReportIndexLSB (F54_AD_Data1)

ReportIndexMSB (F54_AD_Data2)

This pair of registers is a byte-wise pointer to the report data that is being read from the *PacketData* register. It is cleared to 0x0000 on reset and upon an Image Complete interrupt. In addition, writing a value into this two-byte register changes the position within the image buffer from which data will be read.

Reading this pair of registers will return the current value of the pointer.

Note: Setting *ReportIndex* to a value that is outside the valid range for a given mode, or reading the packet register beyond the end of valid data, will result in undefined behavior.

16.3.4. F54_AD_Data3: report data

ReportData (F54_AD_Data3)

This is a packet register through which the reports selected by *ReportType* (F54_AD_Data0) are read.

The content and format of the data contained in this packet register depends on the contents of the *ReportType* field. For 16-bit data, the least significant byte is read first, followed by the most significant byte.

See section 2.5.3 for a detailed description of packet registers.

16.3.5. F54_AD_Data4: sense frequency selection

This register is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as '1'.

SenseFrequencySelection (F54_AD_Data4, bits 3:0)

This field contains the number of the currently active sensing frequency.

Reserved (F54_AD_Data4, bits 6:4)

Reserved.

InhibitFrequencyShift (F54_AD_Data4, bit 7)

Setting this bit to '1' inhibits the firmware from changing the sensing frequency, and forces the sensing frequency to the value specified by the *SenseFrequencySelection* field.

Selecting a sensing-frequency number greater than *NumberOfSensing Frequencies* (F54_AD_Query12, bits 3:0) will cause undefined behavior.

16.3.6. F54_AD_Data5: reserved

Reserved (F54_AD_Data5)

Reserved.

16.3.7. F54_AD_Data6.0/6.1: interference metric

These registers are only present if *HasInterferenceMetric* (F54_AD_Query6, bit 1) reports as '1'.

InterferenceMetricLSB (F54_AD_Data6.0)

InterferenceMetricMSB (F54_AD_Data6.1)

These registers report an estimate of the noise currently present in the system. The range is 0 to 65535. Larger values indicate more noise.

16.3.8. F54_AD_Data7.0/7.1: current report rate

CurrentReportRateLSB (F54_AD_Data7.0)

This register is only present if *HasOneByteReportRate* (F54_AD_Query6, bit 6) reports as '1' or *HasTwoByteReportRate* (F54_AD_Query6, bit 5) reports as '1'.

This register reports the least significant byte of the current report rate, in reports per second.

CurrentReportRateMSB (F54_AD_Data7.1)

This register is only present if *HasTwoByteReportRate* (F54_AD_Query6, bit 5) reports as '1'.

This register reports the most significant byte of the current report rate, in reports per second.

16.3.9. F54_AD_Data8.0/8.1: burst span metric

These registers are only present if *HasBurstSpanMetric* (F54_AD_Query9, bit 3) is set to '1'.

BurstSpanMetricLSB (F54_AD_Data8.0)

BurstSpanMetricMSB (F54_AD_Data8.1)

These registers report an estimate of the noise variations currently present in the system. The range is 0 to 65535. Larger values indicate more noise.

16.3.10. F54_AD_Data9 and F54_AD_Data10: reserved

These registers are reserved.

16.3.11. F54_AD_Data11: status

This register is only present if *HasStatus* (F54_AD_Query9, bit 6) is set to '1'.

TransmitterBank (F54_AD_Data11, bit 0)

For Touch Controllers with multiple transmitter banks, this bit reports the bank in use.

Reserved (F54_AD_Data11, bits 7:1)

Reserved.

16.4. Function $54: interrupt sources

The Test Reporting data source can assert an interrupt request at the end of every report period. Interrupts are enabled and reported in Function $01.

When a report is requested by setting the *GetReport* bit in the F54_AD_Cmd0 register, an interrupt is generated once the data is available to be read. The *GetReport* bit will remain set until the interrupt has occurred and then will be cleared automatically.

No interrupts will be generated from F$54 unless the *GetReport* bit has been set *and* the F$54 interrupt is enabled in the F01_RMI_Ctrl1 register(s).

16.5. Function $54: command registers

The bits in this command register automatically clear to ‘0’ when the requested operation has completed.

Table 70. Function $54 command register

Name	7	6	5	4	3	2	1	0
F54_AD_Cmd0	—					Force Update	ForceCal	Get Report

Function $54 implements a single command register.

GetReport (F54_AD_Cmd0, bit 0)

Setting this bit to ‘1’ requests the report selected by the *ReportType* register (F54_AD_Data0). When the report is ready, this command bit will automatically clear to ‘0’ and an interrupt will be generated.

ForceCal (F54_AD_Cmd0, bit 1)

Setting this bit to ‘1’ requests that a new baseline be taken. When the new baseline has been taken, this command bit will automatically clear to ‘0’ but no interrupt will be generated.

ForceUpdate (F54_AD_Cmd0, bit 2)

Changes to registers [TBD] have no effect until this bit is set to ‘1’. Once the changes have taken effect, this command bit will automatically clear to ‘0’ but no interrupt will be generated.

Reserved (F54_AD_Cmd0, bits 7:3)

Reserved.

17. References: Synaptics literature

For more information, refer to the following documents:

- *System Management Bus (SMBus) Specification*, Version 2, August 3, 2000 (http://www.smbus.org/)
- *I²C Bus Specification,* Version 2.1 by Philips

Contact Us

To locate the Synaptics office nearest you, visit our website at www.synaptics.com.