Synaptics RMI4 Specification: Internal

Synaptics Confidential - Internal Use Only

PN: 511-000117-01 Rev. A

New in this revision

Function or Section	*Details*	*See Page…*
Global	Reorganized document to place functions in numerical order	All
Table listing RMI functions	Reorganized to place functions in numerical order	13
Packet registers	Added packet register information in the “RMI Structure” chapter	26
Function $01	Updated F01_RMI_Query1	44
Function $11	Design Studio 4 configurable firmware now includes version 1 of F$11. Version 1 does not include the Ctrl12 and Ctrl 13 registers	85
	Updated F11_2D_Query11	93
	Added object segmentation sensitivity	108
	Added RMI clip registers	108
	Added drumming registers	109
	Added anti-bending control registers	109 to 110
	Added contact geometry reporting	92, 114, 122 to124
	Added dual slope Z-scaling registers	105
	Added W tuning registers	106
	Added X/Y pitch and finger width registers	106
	Added Pen Z upper threshold	106
	Added finger width units data	116
Function $25	Updated F$25 Data registers (both writable and read-only)	136
	Added CRTC Block table	141
Function $34	Added configuration tracking registers	162
	Added control registers	167
Function $54	Added numerous new functions	177
	Added registers related to pen curve reporting	200
	Added edge compensation registers	194
	Added frequency sync dithering control registers	196
	Added per-axis compensation	195
Function $91	Added new query register for S1100 F91_2D_Query0; bit 2	249
	Added new control registers for S1100 tuning: Ctrl15 to Ctrl 34	253

Deleted: 26
Deleted: 42
Deleted: 83
Deleted: 91
Deleted: 106
Deleted: 106
Deleted: 107
Deleted: 107
Deleted: 108
Deleted: 90
Deleted: 112
Deleted: 120
Deleted: 122
Deleted: 103
Deleted: 104
Deleted: 104
Deleted: 105
Deleted: 115
Deleted: 135
Deleted: 140
Deleted: 161
Deleted: 166
Deleted: 176
Deleted: 200
Deleted: 193
Deleted: 195
Deleted: 194
Deleted: 249
Deleted: 253

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 11
   1.1. Conventions used in RMI documentation 11
2. RMI structure 12
   2.1. RMI functions 12
      2.1.1. Function numbers 13
      2.1.2. Private Functions 13
   2.2. Registers 14
      2.2.1. Query registers 15
      2.2.2. Control registers 15
      2.2.3. Data registers 16
      2.2.4. Command registers 17
   2.3. Register map organization 17
      2.3.1. Register address pages 18
      2.3.2. Global registers 18
      2.3.3. Register grouping within an RMI function 19
      2.3.4. Function grouping within a page 20
      2.3.5. Page Description table 20
      2.3.6. Replicated registers 23
   2.4. Register map organization summary 24
   2.5. RMI physical layer operations 26
      2.5.1. Writing registers 26
      2.5.2. Reading registers 26
      2.5.3. Packet registers 26
      2.5.4. Signaling attention and interrupts 28
      2.5.5. Robust operation 28
   2.6. Register coherence 28
      2.6.1. Write access into coherent regions 29
      2.6.2. Read access into coherent regions 29
      2.6.3. Coherent regions 29
   2.7. Data reporting 29
      2.7.1. Interrupt requests 30
      2.7.2. Attention signal 30
      2.7.3. Spontaneous resets 32
3. Standard RMI physical layers 33
   3.1. I²C physical interface 33
      3.1.1. I²C transfer protocols 33
      3.1.2. RMI register addressing 34
      3.1.3. Block read operations 34
      3.1.4. Block write operations 34
      3.1.5. I²C protocol compliance 35
   3.2. SMBus physical interface 35
      3.2.1. RMI-on-SMBus addressing 36
      3.2.2. SMBus transfer protocols 36
      3.2.3. Multi-register block read/write operations 38
      3.2.4. Repeated starts 38
      3.2.5. SMBus compliance 38
   3.3. SPI physical interface 40
      3.3.1. SPI signals 40
      3.3.2. SPI clocking 40
      3.3.3. SPI transaction format 41
      3.3.4. SPI attention mechanism 42
4. Function $01: RMI device control 44
   4.1. Function $01: query registers 44

Deleted: 19
Deleted: 20
Deleted: 20
Deleted: 23
Deleted: 24
Deleted: 26
Deleted: 26
Deleted: 26
Deleted: 26
Deleted: 27
Deleted: 27
Deleted: 28
Deleted: 29
Deleted: 29
Deleted: 29
Deleted: 29
Deleted: 29
Deleted: 30
Deleted: 31
Deleted: 32
Deleted: 32
Deleted: 32
Deleted: 33
Deleted: 33
Deleted: 33
Deleted: 34
Deleted: 34
Deleted: 35
Deleted: 35
Deleted: 37
Deleted: 37
Deleted: 37
Deleted: 39
Deleted: 39
Deleted: 39
Deleted: 40
Deleted: 41
Deleted: 42
Deleted: 42

4.1.1. F01_RMI_Query0: manufacturer ID query 44
4.1.2. F01_RMI_Query1: product properties query 44
4.1.3. F01_RMI_Query2 and F01_RMI_Query3: product information queries 45
4.1.4. F01_RMI_Query4 through F01_RMI_Query10: device serialization queries 45
4.1.5. F01_RMI_Query11 through F01_RMI_Query20: product ID queries 46
4.1.6. F01_RMI_Query21: reserved 46
4.1.7. F01_RMI_Query22: sensor ID 46
4.1.8. F01_RMI_Query23 through 31: reserved T1320 manufacturing information 47
4.2. Function $01: control registers 48
4.2.1. F01_RMI_Ctrl0: device control register 48
4.2.2. F01_RMI_Ctrl1.*: interrupt enable register 50
4.2.3. F01_RMI_Ctrl2: doze interval register 50
4.2.4. F01_RMI_Ctrl3: wakeup threshold register 51
4.2.5. F01_RMI_Ctrl4: does not exist 51
4.2.6. F01_RMI_Ctrl5: doze holdoff register 51
4.3. Function $01: data registers 52
4.3.1. F01_RMI_Data0: device status register 52
4.3.2. F01_RMI_Data1.*: interrupt status register 53
4.3.3. Function $01: interrupt source 54
4.4. Function $01: command registers 55
4.4.1. F01_RMI_Cmd0: device command register 55

5. Function $05: Image reporting for T1320 57
5.1. Function $05: query registers 58
5.2. Function $05: control registers 59
5.3. Function $05: data registers 60
5.4. Function $05: interrupt sources 61
5.5. Function $05: command registers 61
5.6. Reading images with Function $05 62

6. Function $07: Image Reporting for LTS 63
6.1. Function $07: query registers 63
6.2. Function $07: control registers 65
6.3. Function $07: data registers 66
6.4. Function $07: interrupt sources 67
6.5. Function $07: command registers 68
6.6. Reading images with Function $07 69

7. Function $08: Built-in self-test for T1021 70
7.1. Typical BIST usage scenario 70
7.2. Function $08: query registers 71
7.3. Function $08: control registers 72
7.4. Function $08: data registers 74
7.5. Function $08: interrupt source 75
7.6. Function $08: command registers 76

8. Function $09: BIST for T1320 77
8.1. Typical BIST usage scenario 77
8.2. Function $09: query registers 78
8.3. Function $09: control registers 79
8.4. Function $09: data registers 82
8.5. Function $09: interrupt source 83
8.6. Function $09: command registers 84

9. Function $11: 2-D sensors 85
9.1. Function version 85
9.2. Number of 2-D sensors 85
9.3. Function $11: query registers 86
9.3.1. F11_2D_Query0: per-device query registers 86
9.3.2. F11_2D_Query1 through F11_2D_Query11: per-sensor query registers 87

Deleted: 42
Deleted: 42
Deleted: 43
Deleted: 43
Deleted: 44
Deleted: 44
Deleted: 44
Deleted: 45
Deleted: 46
Deleted: 46
Deleted: 48
Deleted: 48
Deleted: 49
Deleted: 49
Deleted: 49
Deleted: 50
Deleted: 50
Deleted: 51
Deleted: 52
Deleted: 53
Deleted: 53
Deleted: 55
Deleted: 56
Deleted: 57
Deleted: 58
Deleted: 59
Deleted: 59
Deleted: 60
Deleted: 61
Deleted: 61
Deleted: 63
Deleted: 64
Deleted: 65
Deleted: 66
Deleted: 67
Deleted: 68
Deleted: 68
Deleted: 69
Deleted: 70
Deleted: 72
Deleted: 73
Deleted: 74
Deleted: 75
Deleted: 75
Deleted: 76
Deleted: 77
Deleted: 80
Deleted: 81
Deleted: 82
Deleted: 83
Deleted: 83
Deleted: 83
Deleted: 84
Deleted: 84
Deleted: 85

9.3.3. F11_2D_Query1: general sensor information 87
9.3.4. F11_2D_Query2 and F11_2D_Query3: number of X and Y electrodes 88
9.3.5. F11_2D_Query4: maximum electrodes 88
9.3.6. F11_2D_Query5: absolute data source 89
9.3.7. F11_2D_Query6: relative data source 90
9.3.8. F11_2D_Query7 and F11_2D_Query8: gesture information 90
9.3.9. F11_2D_Query9: advanced sensing features 92
9.3.10. F11_2D_Query10: TouchShape support 92
9.3.11. F11_2D_Query11: advanced capabilities 93
9.4. Function $11: control registers 94
9.4.1. F11_2D_Ctrl0: general control 95
9.4.2. F11_2D_Ctrl1: palm and finger control 97
9.4.3. F11_2D_Ctrl2 and F11_2D_Ctrl3: distance threshold 98
9.4.4. F11_2D_Ctrl4: velocity control 98
9.4.5. F11_2D_Ctrl5: acceleration control 98
9.4.6. F11_2D_Ctrl6 through F11_2D_Ctrl9: maximum X and Y position control 99
9.4.7. F11_2D_Ctrl10 and F11_2D_Ctrl11: gesture control 99
9.4.8. F11_2D_Ctrl12.* does not exist 100
9.4.9. F11_2D_Ctrl13.* does not exist 100
9.4.10. F11_2D_Ctrl14: sensitivity adjustment 101
9.4.11. F11_2D_Ctrl15: maximum tap time 101
9.4.12. F11_2D_Ctrl16: minimum press time 101
9.4.13. F11_2D_Ctrl17: maximum tap distance 101
9.4.14. F11_2D_Ctrl18: minimum flick distance 102
9.4.15. F11_2D_Ctrl19: minimum flick speed 102
9.4.16. F11_2D_Ctrl20: pen control 102
9.4.17. F11_2D_Ctrl21: pen Z lower threshold 102
9.4.18. F11_2D_Ctrl22: proximity control 102
9.4.19. F11_2D_Ctrl23: proximity detection Z threshold 103
9.4.20. F11_2D_Ctrl24: proximity detection X threshold 103
9.4.21. F11_2D_Ctrl25: proximity detection Y threshold 103
9.4.22. F11_2D_Ctrl25: proximity delta Z threshold 103
9.4.23. F11_2D_Ctrl27: palm-detect parameters 103
9.4.24. F11_2D_Ctrl28: multi-finger scroll parameters 104
9.4.25. F11_2D_Ctrl29: z touch threshold 104
9.4.26. F11_2D_Ctrl30: z hysteresis 105
9.4.27. F11_2D_Ctrl31: small z threshold 105
9.4.28. F11_2D_Ctrl32.0/32.1: small z scale factor 105
9.4.29. F11_2D_Ctrl33.0/33.1: large z scale factor 105
9.4.30. F11_2D_Ctrl34: algorithm selection 105
9.4.31. F11_2D_Ctrl35: pen Z upper threshold 106
9.4.32. F11_2D_Ctrl36: Wx scale factor 106
9.4.33. F11_2D_Ctrl37: Wx offset 106
9.4.34. F11_2D_Ctrl38: Wy scale factor 106
9.4.35. F11_2D_Ctrl39: Wy offset 107
9.4.36. F11_2D_Ctrl40.0/40.1: x pitch 107
9.4.37. F11_2D_Ctrl41.0/41.1: y pitch 107
9.4.38. F11_2D_Ctrl42.0/42.1: finger size on X axis 107
9.4.39. F11_2D_Ctrl43.0/43.1: finger size on Y axis 108
9.4.40. F11_2D_Ctrl44: report measured size 108
9.4.41. F11_2D_Ctrl45: segmentation aggressiveness 108
9.4.42. F11_2D_Ctrl46: X clip left 108
9.4.43. F11_2D_Ctrl47: X clip right 108
9.4.44. F11_2D_Ctrl48: Y clip upper 109
9.4.45. F11_2D_Ctrl49: Y clip lower 109
9.4.46. F11_2D_Ctrl50: minimum drumming separation 109
9.4.47. F11_2D_Ctrl51: maximum drumming movement 109
9.4.48. F11_2D_Ctrl52: bending detection threshold 109

Deleted: 85
Deleted: 86
Deleted: 86
Deleted: 87
Deleted: 88
Deleted: 88
Deleted: 90
Deleted: 90
Deleted: 91
Deleted: 92
Deleted: 93
Deleted: 95
Deleted: 96
Deleted: 96
Deleted: 96
Deleted: 97
Deleted: 97
Deleted: 98
Deleted: 98
Deleted: 99
Deleted: 99
Deleted: 99
Deleted: 99
Deleted: 100
Deleted: 100
Deleted: 100
Deleted: 100
Deleted: 100
Deleted: 101
Deleted: 101
Deleted: 101
Deleted: 101
Deleted: 101
Deleted: 102
Deleted: 102
Deleted: 103
Deleted: 103
Deleted: 103
Deleted: 103
Deleted: 103
Deleted: 104
Deleted: 104
Deleted: 104
Deleted: 104
Deleted: 105
Deleted: 105
Deleted: 105
Deleted: 105
Deleted: 106
Deleted: 106
Deleted: 106
Deleted: 106
Deleted: 106
Deleted: 107
Deleted: 107
Deleted: 107
Deleted: 107
Deleted: 107

9.4.49. F11_2D_Ctrl53: bending touch threshold adjustment 110
9.4.50. F11_2D_Ctrl54: flexibility 110
9.4.51. F11_2D_Ctrl55: bend distance / bend interaction distance 110
9.4.52. F11_2D_Ctrl56: bending correction X 110
9.4.53. F11_2D_Ctrl57: bending correction Y 110
9.4.54. F11_2D_Ctrl58: pragma color 111
9.5. Function $11: data registers 112
9.5.1. Data register layout 112
9.5.2. Finger reporting 114
9.5.3. F11_2D_Data0.*: finger status data 115
9.5.4. F11_2D_Data1 and F11_2D_Data 2: X and Y position data (MSB) 116
9.5.5. F11_2D_Data3: X and Y position data (LSB) 116
9.5.6. F11_2D_Data4: finger width (W) data 116
9.5.7. F11_2D_Data5: finger contact (Z) data 116
9.5.8. F11_2D_Data6.* and F11_2D_Data7.*: finger motion deltas 117
9.5.9. F11_2D_Data8 and F11_2D_Data9: gesture data 117
9.5.10. F11_2D_Data10: pinch motion and X flick distance 120
9.5.11. F11_2D_Data11: rotate motion and Y flick distance 120
9.5.12. F11_2D_Data12: finger separation and flick time 120
9.5.13. F11_2D_Data13.*: TouchShape status 120
9.5.14. F11_2D_Data14: x lower scroll motion / MultiFinger horizontal scroll 121
9.5.15. F11_2D_Data15: y right scroll motion / MultiFinger vertical scroll 122
9.5.16. F11_2D_Data16: x upper scroll motion 122
9.5.17. F11_2D_Data17: y left scroll motion 122
9.5.18. F11_2D_Data18 and F11_2D_Data19: contact geometry X and Y target positions (MSB) 122
9.5.19. F11_2D_Data20: contact geometry X and Y target positions (LSB) 123
9.5.20. F11_2D_Data21 and F11_2D_Data22: contact geometry height and width (MSB) 123
9.5.21. F11_2D_Data23: contact geometry height and width (LSB) 123
9.5.22. F11_2D_Data24: contact geometry angle 123
9.5.23. F11_2D_Data25 and F11_2D_Data26: contact geometry X and Y center positions (MSB) 123
9.5.24. F11_2D_Data27: contact geometry X and Y center positions (LSB) 124
9.5.25. F11_2D_Data28: sensor status 124
9.6. Function $11: interrupt source 124
9.7. Function $11: command registers 125

10. Function $19: 0-D Capacitive Button Sensors 126
10.1. Function $19: query registers 126
10.1.1. F19_Btn_Query0: Configurable button query 126
10.1.2. F19_Btn_Query1: ButtonCount query 127
10.2. Function $19: control registers 128
10.2.1. Calculating the number of control registers 128
10.2.2. F19_Btn_Ctrl0: general control 129
10.2.3. F19_Btn_Ctrl1.*: button interrupt enable control 130
10.2.4. F19_Btn_Ctrl2.*: single button participation control 130
10.2.5. F19_Btn_Ctrl3.*: sensor map control 131
10.2.6. F19_Btn_Ctrl4.*: reserved button sensitivity control 132
10.2.7. F19_Btn_Ctrl5: all-button sensitivity adjustment 132
10.2.8. F19_Btn_Ctrl6: all-button hysteresis threshold 132
10.3. Function $19: data registers 133
10.3.1. Calculating the number of data registers 133
10.4. Function $19: interrupt source 133
10.5. Function $19: command registers 134

11. Function $25: Slurper (Private) 135
11.1. Function $25: Signals 135
11.2. Function $25: query registers 135

Deleted: 108
Deleted: 108
Deleted: 108
Deleted: 108
Deleted: 108
Deleted: 109
Deleted: 110
Deleted: 110
Deleted: 112
Deleted: 113
Deleted: 114
Deleted: 114
Deleted: 114
Deleted: 114
Deleted: 115
Deleted: 115
Deleted: 118
Deleted: 118
Deleted: 118
Deleted: 118
Deleted: 119
Deleted: 120
Deleted: 120
Deleted: 120
Deleted: 120
Deleted: 121
Deleted: 121
Deleted: 121
Deleted: 121
Deleted: 121
Deleted: 122
Deleted: 122
Deleted: 122
Deleted: 123
Deleted: 124
Deleted: 124
Deleted: 124
Deleted: 125
Deleted: 126
Deleted: 126
Deleted: 127
Deleted: 128
Deleted: 128
Deleted: 129
Deleted: 130
Deleted: 130
Deleted: 130
Deleted: 131
Deleted: 131
Deleted: 131
Deleted: 132
Deleted: 133
Deleted: 133
Deleted: 133

11.3. Function $25: control registers 135
11.4. Function $25: data registers 136
11.4.1. Image Data Format 138
11.4.2. Frame blocks 0 and 1 139
11.4.3. CRTC blocks 141
11.4.4. Analog Register tables 0 and 1 142
11.5. Function $25: interrupt sources 142
11.6. Function $25: command registers 143
11.7. Function $25 Synchronization 144
11.8. Updating Function $25 Parameters 144
11.8.1. Block Update 144
11.8.2. Stalled Update 144
11.9. Recovering from Errors 145

12. Function $30: GPIO/LED and Mechanical Buttons 146
12.1. Function $30: power management 146
12.2. Function $30: query registers 147
12.3. Function $30: control registers 148
12.3.1. Calculating the number of control registers 148
12.3.2. F30_GPIO_Ctrl0.*: GPIO/LED select 148
12.3.3. F30_GPIO_Ctrl1: GPIO/LED general control 149
12.3.4. F30_GPIO_Ctrl2.* and F30_GPIO_Ctrl3.*: GPIO Input/Output Mode control 149
12.3.5. F30_GPIO_Ctrl4.*: LED active control 150
12.3.6. F30_GPIO_Ctrl5.*: LED ramp period control 151
12.3.7. F30_GPIO_Ctrl6.*: GPIO/LED control 151
12.3.8. F30_GPIO_Ctrl7.*: button-to-GPIO mapping and control 155
12.3.9. F30_GPIO_Ctrl8.*: haptic enable control 156
12.3.10. F30_GPIO_Ctrl9: haptic duration control 156
12.3.11. Future possibilities - private 157
12.4. Function $30: data registers 158
12.5. Function $30: interrupt source 158
12.6. Function $30: command registers 159
12.7. Function $30 example: complete control layout 159
12.8. Function $30 examples: query bits 160
12.8.1. Function $30 example: both GPIOs and LEDs 160
Function $30 example: LEDs only 160
12.8.2. Function $30 example: GPIOs only, with driver control 161
12.8.3. Function $30 example: GPIOs only, without driver control 161

13. Function $34: Flash memory management 162
13.1. Overview 162
13.1.1. Non-volatile memory organization 162
13.1.2. Modality 163
13.2. Function $34: query registers 165
13.2.1. F34_Flash_Query0 and F34_Flash_Query1: bootloader ID query 165
13.2.2. F34_Flash_Query2: flash properties query 165
13.2.3. F34_Flash_Query3 and F34_Flash_Query4: block size query 166
13.2.4. F34_Flash_Query5 and F34_Flash_Query6: firmware block count query 166
13.2.5. F34_Flash_Query7 and F34_Flash_Query8: configuration block count query 166
13.3. Function $34: control registers 167
13.4. Function $34: data registers 168
13.4.1. F34_Flash_Data0 and F34_Flash_Data1: block number registers 168
13.4.2. F34_Flash_Data2.*: block data registers 168
13.4.3. F34_Flash_Data3: flash control/status register 169
13.5. Function $34: interrupt source 172

14. Function $36: Auxiliary analog to digital conversion 173

Deleted: 133
Deleted: 134
Deleted: 136
Deleted: 137
Deleted: 139
Deleted: 140
Deleted: 140
Deleted: 141
Deleted: 142
Deleted: 142
Deleted: 142
Deleted: 142
Deleted: 143
Deleted: 144
Deleted: 144
Deleted: 145
Deleted: 146
Deleted: 146
Deleted: 146
Deleted: 147
Deleted: 147
Deleted: 148
Deleted: 149
Deleted: 149
Deleted: 153
Deleted: 154
Deleted: 154
Deleted: 155
Deleted: 156
Deleted: 156
Deleted: 157
Deleted: 157
Deleted: 158
Deleted: 158
Deleted: 158
Deleted: 159
Deleted: 159
Deleted: 160
Deleted: 160
Deleted: 160
Deleted: 161
Deleted: 163
Deleted: 163
Deleted: 163
Deleted: 164
Deleted: 164
Deleted: 164
Deleted: 165
Deleted: 166
Deleted: 166
Deleted: 166
Deleted: 167
Deleted: 170
Deleted: 171

14.1. Function $36: query register 173
14.2. Function $36: control register 174
14.3. Function $36: data registers 175
14.4. Function $36: command registers 176
14.5. Function $36: interrupt source 176

15. Function $54: Test reporting (T1321 devices) 177
15.1. Function $54: query registers 178
15.1.1. F54_AD_Query0 and F54_AD_Query1: number of electrodes 178
15.1.2. F54_AD_Query2: image reporting modes 178
15.1.3. F54_AD_Query3.0/3.1: clock rate 179
15.1.4. F54_AD_Query4: analog hardware family 179
15.1.5. F54_AD_Query5: analog hardware controls 179
15.1.6. F54_AD_Query6: data acquisition 179
15.1.7. F54_AD_ Query7: per axis compensation 180
15.1.8. F54_AD_Query Query8: data acquisition post-processing controls 180
15.1.9. F54_AD_Query9 through F54_AD_11: reserved 181
15.1.10. F54_AD_Query12: sense frequency control 181
15.1.11. F54_AD_Query13: analog Information 181
15.2. Function $54: control registers 183
15.2.1. F54_AD_Ctrl0: general control 0 184
15.2.2. F54_AD_Ctrl1: general control 1 184
15.2.3. F54_AD_Ctrl2.0/2.1: saturation capacitance 184
15.2.4. F54_AD_Ctrl3: pixel touch threshold 185
15.2.5. F54_AD_Ctrl4: miscellaneous analog control 185
15.2.6. F54_AD_Ctrl5: RefCap RefLo settings 185
15.2.7. F54_AD_Ctrl6: RefCap RefHigh settings 186
15.2.8. F54_AD_Ctrl7: CBC cap settings 187
15.2.9. F54_AD_Ctrl8.0/8.1: integration duration 187
15.2.10. F54_AD_Ctrl9: reset duration 187
15.2.11. F54_AD_Ctrl10: noise sensing bursts per frame 187
15.2.12. F54_AD_Ctrl11.0/11.1: wakeup threshold 188
15.2.13. F54_AD_Ctrl12: slow relaxation rate 188
15.2.14. F54_AD_Ctrl13: fast relaxation rate 188
15.2.15. F54_AD_Ctrl14.*: sensor assignment properties 188
15.2.16. F54_AD_Ctrl15.*: sensor Rx assignment 189
15.2.17. F54_AD_Ctrl16.*: sensor Tx assignment 189
15.2.18. F54_AD_Ctrl17.* through F54_AD_Ctrl19.*: sense frequency overview 189
15.2.19. F54_AD_Ctrl17.*: sense frequency control 190
15.2.20. F54_AD_Ctrl18.* cycles per burst 190
15.2.21. F54_AD_Ctrl19.*: stretch duration 190
15.2.22. F54_AD_Ctrl20 through F54_AD_Ctrl29: noise mitigation 190
15.2.23. F54_AD_Ctrl20: noise mitigation general control 191
15.2.24. F54_AD_ Ctrl21.0/21.1: HNM frequency shift noise threshold format 191
15.2.25. F54_AD_Ctrl22: hardware noise mitigation exit density 191
15.2.26. F54_AD_Ctrl23.0/23.1: medium noise threshold 192
15.2.27. F54_AD_Ctrl24.0/24.1: high noise threshold 192
15.2.28. F54_AD_Ctrl25: FNM frequency shift density 192
15.2.29. F54_AD_Ctrl26: firmware noise mitigation exit threshold 192
15.2.30. F54_AD_Ctrl27: IIR filter coefficient 193
15.2.31. F54_AD_Ctrl28.0/28.1: FNM frequency shift noise threshold 193
15.2.32. F54_AD_Ctrl29: common-mode noise control 193
15.2.33. F54_AD_Ctrl30: CMN cap scale factor 193
15.2.34. F54_AD_Ctrl31: pixel threshold hysteresis 193
15.2.35. F54_AD_Ctrl32 through F54_AD_Ctrl35: edge performance overview 194
15.2.36. F54_AD_Ctrl32.0/32.1: Rx low edge compensation 194
15.2.37. F54_AD_Ctrl33.0/33.1: Rx high edge compensation 194
15.2.38. F54_AD_Ctrl34.0/34.1: Tx low edge compensation 194
15.2.39. F54_AD_Ctrl35.0/35.1: Tx high edge compensation 195

Deleted: 171
Deleted: 172
Deleted: 173
Deleted: 174
Deleted: 174
Deleted: 175
Deleted: 176
Deleted: 176
Deleted: 176
Deleted: 177
Deleted: 177
Deleted: 177
Deleted: 178
Deleted: 178
Deleted: 179
Deleted: 179
Deleted: 179
Deleted: 180
Deleted: 181
Deleted: 182
Deleted: 182
Deleted: 182
Deleted: 183
Deleted: 183
Deleted: 183
Deleted: 184
Deleted: 185
Deleted: 185
Deleted: 185
Deleted: 185
Deleted: 186
Deleted: 186
Deleted: 186
Deleted: 186
Deleted: 187
Deleted: 187
Deleted: 187
Deleted: 188
Deleted: 188
Deleted: 188
Deleted: 188
Deleted: 189
Deleted: 189
Deleted: 189
Deleted: 190
Deleted: 190
Deleted: 190
Deleted: 190
Deleted: 191
Deleted: 191
Deleted: 191
Deleted: 191
Deleted: 192
Deleted: 192
Deleted: 192
Deleted: 192
Deleted: 193
Deleted: 193

15.2.40. F54_AD_Ctrl36.*: axis 1 compensation 195
15.2.41. F54_AD_Ctrl37.*: axis 2 compensation 196
15.2.42. F54_AD_Ctrl38.* through F54_AD_Ctrl40.*: per frequency noise control. 196
15.2.43. F54_AD_Ctrl38.*: noise control 1 196
15.2.44. F54_AD_Ctrl39.*: noise control 2 196
15.2.45. F54_AD_Ctrl40.*: noise control 3 196
15.3. Function $54: data registers 197
15.3.1. F54_AD_Data0: report type 197
15.3.2. F54_AD_Data1 and F54_AD_Data2: FIFO index 199
15.3.3. F54_AD_Data3: FIFO data 199
15.3.4. F54_AD_Data4: current sense frequency selection 199
15.3.5. F54_AD_Data5: reserved 200
15.3.6. F54_AD_Data6.0/6.1: interference metric 200
15.3.7. F54_AD_Data7.0/7.1: current report rate 200
15.3.8. F54_AD_Data8: Cxy LSB 200
15.3.9. F54_AD_Data9: Cxy MSB 201
15.3.10. F54_AD_Data10: pen Z 201
15.4. Function $54: interrupt sources 201
15.5. Function $54: command registers 202
15.6. Reading test reports with Function $54 203

16. Function $81: Device control (Private) 204
16.1. Function $81: query registers 204
16.1.1. F81_RMI_Query0: private properties 204
16.1.2. F81_RMI_Query1 through F81_RMI_Query3: firmware build ID 204
16.2. Function $81: control registers 204
16.3. Function $81: data registers 205
16.3.1. F81_RMI_Data0.*: interrupt enable 205
16.3.2. F81_RMI_Data1.*: interrupt status 206
16.3.3. F81_RMI_Data2 through F81_RMI_Data4: RAM back door 206
16.3.4. F81_RMI_Data5 through 7: FIFO back door 206
16.4. Function $81: interrupt source 207
16.5. Function $81: command registers 207

17. Function $84: Analog Diagnostic for T1021 (Private) 208
17.1. Function $84: query registers 209
17.2. Function $84: control registers 211
17.3. Function $84: data registers 213
17.4. Function $84: interrupt source 216
17.5. Function $84: command registers 217
17.6. Function $84: hardware register window 217
17.7. Reading profiles with Function $84 218

18. Function $85: Analog Diagnostic for T1320 (Private) 220
18.1. Function $85: query registers 221
18.2. Function $85: control registers 223
18.3. Function $85: data registers 226
18.4. Function $85: interrupt sources 229
18.5. Function $85: command registers 230
18.6. Function $85: hardware register window 231
18.7. Reading images with Function $85 231
18.8. Reading baselines with Function $85 232

19. Function $87: Analog Diagnostic for LTS Central Microcontroller (Private) 234
19.1. Function $87: query registers 235
19.2. Function $87: control registers 237
19.3. Function $87: data registers 239
19.4. Function $87: interrupt sources 244
19.5. Function $87: command registers 245
19.6. Reading images with Function $87 246

Deleted: 193
Deleted: 194
Deleted: 194
Deleted: 194
Deleted: 194
Deleted: 194
Deleted: 196
Deleted: 196
Deleted: 198
Deleted: 198
Deleted: 198
Deleted: 199
Deleted: 199
Deleted: 199
Deleted: 199
Deleted: 200
Deleted: 200
Deleted: 200
Deleted: 201
Deleted: 202
Deleted: 203
Deleted: 203
Deleted: 203
Deleted: 203
Deleted: 203
Deleted: 204
Deleted: 204
Deleted: 205
Deleted: 205
Deleted: 205
Deleted: 206
Deleted: 206
Deleted: 207
Deleted: 208
Deleted: 210
Deleted: 212
Deleted: 215
Deleted: 216
Deleted: 216
Deleted: 217
Deleted: 219
Deleted: 220
Deleted: 222
Deleted: 225
Deleted: 228
Deleted: 229
Deleted: 230
Deleted: 230
Deleted: 231
Deleted: 233
Deleted: 234
Deleted: 236
Deleted: 238
Deleted: 243
Deleted: 244
Deleted: 245

19.7. Re-grabbing and reading baselines with Function $87 247

20. Function $91: 2-D private functions 249
20.1. Number of 2-D sensors 249
20.2. Function $91: query registers 250
20.2.1. F91_2D_Query0: Sensor Properties 250
20.2.2. F91_2D_Query1: Position Correction Table Size 250
20.2.3. F91_2D_Query2: Miscellaneous 251
20.2.4. F91_2D_Query3 through F91_2D_Query5: Correction Table Sizes 251
20.3. Function $91: control registers 253
20.4. Function $91: data registers 257
20.4.1. Data register layout 257
20.5. Function $91: interrupt source 257
20.6. Function $91: command registers 257

21. Function $99: 0-D Functions (Private) 258
21.1. Function $99: query registers 258
21.2. Function $99: control registers 258
21.2.1. F99_Btn_Ctrl0: button interference metric control register 258
21.2.2. F99_Btn_Ctrl1: button diagnostic control 258
21.3. Function $99: data registers 258
21.4. Function $99: command registers 258

22. Function $D4: Analog Diagnostic for T1321 Devices (Private) 259
22.1. Function $D4: query registers 260
22.2. Function $D4: control registers 262
22.3. Function $D4: data registers 265
22.4. Function $D4: interrupt sources 268
22.5. Function $D4: command registers 269
22.6. Reading images with Function $D4 270
22.7. Acquiring and reading baselines with Function $D4 271

23. References: Synaptics literature 273

Deleted: 246
Deleted: 248
Deleted: 248
Deleted: 249
Deleted: 249
Deleted: 249
Deleted: 250
Deleted: 250
Deleted: 252
Deleted: 256
Deleted: 256
Deleted: 256
Deleted: 256
Deleted: 257
Deleted: 257
Deleted: 257
Deleted: 257
Deleted: 257
Deleted: 257
Deleted: 257
Deleted: 258
Deleted: 259
Deleted: 261
Deleted: 264
Deleted: 267
Deleted: 268
Deleted: 269
Deleted: 270
Deleted: 272

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 it needs to configure its operation modes,
- The data registers that it uses to report information back to the host,
- The interrupt sources that it uses to signal changes in the contents of its data registers,
- The command registers that it might need to implement function-specific commands, and
- The query registers that would enable a host to learn about its specific capabilities.

RMI functions are not required to implement all of these registers. The specific 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	44
$05	Image Reporting (T1320)	57
$07	Image Reporting (LTS)	63
$08	BIST (T1021)	70
$09	BIST (T1320)	77
$11	2-D TouchPad sensors	85
$19	0-D capacitive button sensors	126
$25	Slurper	135
$30	GPIO/LEDs	
$34	Flash Memory Management	162
$36	Auxiliary ADC	173
$54	Test Reporting (T1321)	177
$81	Private Device Control	204
$84	Analog Diagnostic (T1021)	208
$85	Analog Diagnostic (T1320)	220
$87	Analog Diagnostic (LTS Central Microcontroller)	234
$91	2-D private functions	249
$99	0-D private functions	258
$D4	Analog Diagnostic (T1321)	259

Comment [x1]: These xrefs need to be checked.

Deleted: 43
Deleted: 56
Deleted: 61
Deleted: 68
Deleted: 75
Deleted: 83
Deleted: 124
Deleted: 133
Deleted: 160
Deleted: 171
Deleted: 175
Deleted: 203
Deleted: 207
Deleted: 219
Deleted: 233
Deleted: 248
Deleted: 257
Deleted: 258

2.1.2. Private Functions

The RMI4 specification allows for the definition of ‘private’ functions. Private functions expose registers and controls that would be used for proprietary or diagnostic purposes. The existence and operation of private functions will not be exposed to customers except under exceptional circumstances.

Private functions will not show up in the customer’s product documentation. Private functions will exist in Page Description tables, but they will not be placed on the customer-facing page of the RMI register map.

2.1.2.1. Private function numbers

Many RMI functions contain private registers, and indeed many RMI functions are entirely private. The private registers and functions are for internal use only, and are only documented in this internal version of the RMI4 Specification.

Private functions which are associated with public functions are usually assigned numbers $80 greater than the number of the public function. For example, RMI Function $81 contains private registers related to Function $01, and RMI Function $D4 contains private registers related to Function $54. Not all public functions have private counterparts, and vice-versa.

The RMI4 Specification is organized by function number for ease of use, even though this separates public functions from their private counterparts.

2.1.2.2. Accessing private functions

When an RMI device first powers on, all private functions are hidden and locked, including the PDT registers:

- Attempting to read any of these registers returns a value of $00, which is consistent with an attempt to read any unimplemented RMI register.
- Attempting to write any of these registers is ignored, which is consistent with an attempt to write any unimplemented RMI register.

Private functions are unlocked (made accessible) by writing a special sequence to the Function $01 Device Status Register (as defined in section 4.3.1):

```
$42    $E1
```

After this sequence has been written, the device is in Private Function mode and private functions, including PDT registers, are accessible. Private functions can be locked again by doing another write to the Device Status register with a value of $00.

Another way to exit Private Function mode is by writing a '1' to the configured bit of Function $01's Device Control register. In fact, this is the preferred safe technique for exiting Private Function mode because it also ensures that no further interrupts can occur from private functions.

Some private functions may have certain operations that require an additional unlock, for example, the 'RAM back door' of Function $81. In order to enable this private function, first unlock private functions as described above and then write the following sequence to register F81_RMI_Data2:

```
$4F    $59
```

After private functions are unlocked, but before the RAM is unlocked, registers F81_RMI_Data4 appear to be unimplemented; that is, reading them always returns $00 and writing to them is ignored.

There is no command to exit the ‘RAM back door’, although exiting Private Function mode blocks access to these RAM registers. A device reset will exit both Private Function mode and access to the ‘RAM back door’.

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 need to 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.

2.2.3.1. Private data registers

By changing the specification of data registers in the public portion of the document to define that data registers are *typically* read-only in nature, data registers are permitted to act like control registers that have no selectable configuration default.

The public portions of the spec have not required this ability [yet], but the private functions have shown that there are many private registers that either do not require a settable configuration default, or in fact would not be well served by having a user-settable configuration default. A typical example might be an RMI debug register that provides direct access to a CPU hardware register. For the most part, these registers are there to provide insight into device operation, but under carefully controlled circumstances, they may be written during device operation.

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 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.

Some LTS devices define an additional global register: the Bus Select register.

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.

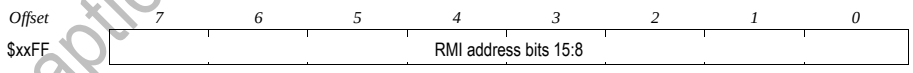

Figure 1. Page Select register

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.

2.3.2.2. Bus Select register

Some large touch screen (LTS) devices define a Bus Select register.

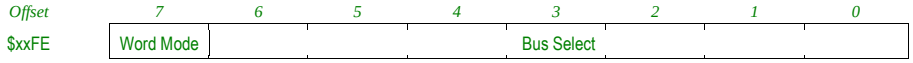

Figure 2. Bus Select register

LTS devices contain multiple processors: one central microcontroller (ARM) and one or more capacitance-sensing touch controllers (T1320s). Each processor may be individually addressed via the *Bus Select* field of the register. The register exists at the same page offset on every RMI address page, on every processor (even, conceptually, on processors which do not exist). This means that it can always be written and read, regardless of its current contents. The default page offset for the Bus Select register is $FE.

Values of the *Bus Select* field:

$00: The central microcontroller (ARM)
$01: Capacitance-sensing touch controller #1 (T1320-1)
$02: Capacitance-sensing touch controller #2 (T1320-2)
$03: Capacitance-sensing touch controller #3 (T1320-3)
$04: Capacitance-sensing touch controller #4 (T1320-4)
$05: Capacitance-sensing touch controller #5 (T1320-5)
$06: Capacitance-sensing touch controller #6 (T1320-6)
$07: Capacitance-sensing touch controller #7 (T1320-7)
$08: Capacitance-sensing touch controller #8 (T1320-8)
….
$7F: Broadcast to all capacitance-sensing touch controllers

While the *Bus Select* field contains $7F, RMI Write transactions are routed to all capacitance-sensing touch controllers simultaneously. The behavior of RMI Read transactions is undefined (except for reads of the Bus Select Register, which behave normally).

Note: After setting the *Bus Select* field to $7F, the Page Select Register should also be written, in order to ensure that all capacitance-sensing touch controllers are on the same page.

The register's MSB, *Word Mode*, selects between 8-bit and 16-bit SPI communications: '0' = 8-bit, '1' = 16-bit.

At reset, the Bus Select register contains $00, which selects the central microcontroller and 8-bit SPI.

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.

2.3.4.1. Private function grouping

The private functions tend to define large numbers of registers. To simplify the placement of these functions in the memory map, private functions will typically be placed one per page.

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.

Offset	7	6	5	4	3	2	1	0
PdtTop-0	—	NonStdPSR	HasBSR	—	—	—	—	—

Figure 3. Page Description table Properties Query register

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.

HasBSR (QueryBase+0, bit 5)

When ‘1’ (LTS only), this bit indicates that the Bus Select register exists and is located at page offset $FE on every page of every device.

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.

Function Descriptors are defined as follows:

Offset	7	6	5	4	3	2	1	0
FuncDescriptor-0	Function Number							
FuncDescriptor-1	—	Function Version		—	—	Interrupt Source Count		
FuncDescriptor-2	DataBase							
FuncDescriptor-3	ControlBase							
FuncDescriptor-4	CommandBase							
FuncDescriptor-5	QueryBase							

Figure 4. Function Descriptor registers

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.

Private interrupt sources defined by private functions in a PDT are allocated in their own interrupt registers. For example, if a hypothetical device defines three public functions that specify two public interrupt sources, and also defines two private functions that define another two private interrupt sources, the public interrupt sources will be assigned bits 0:2 in the public interrupt registers, while the private interrupt sources will be assigned bits 0:1 in the private interrupt registers.

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:

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

Figure 5. A sample Device Control register

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 need to 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 6, 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.

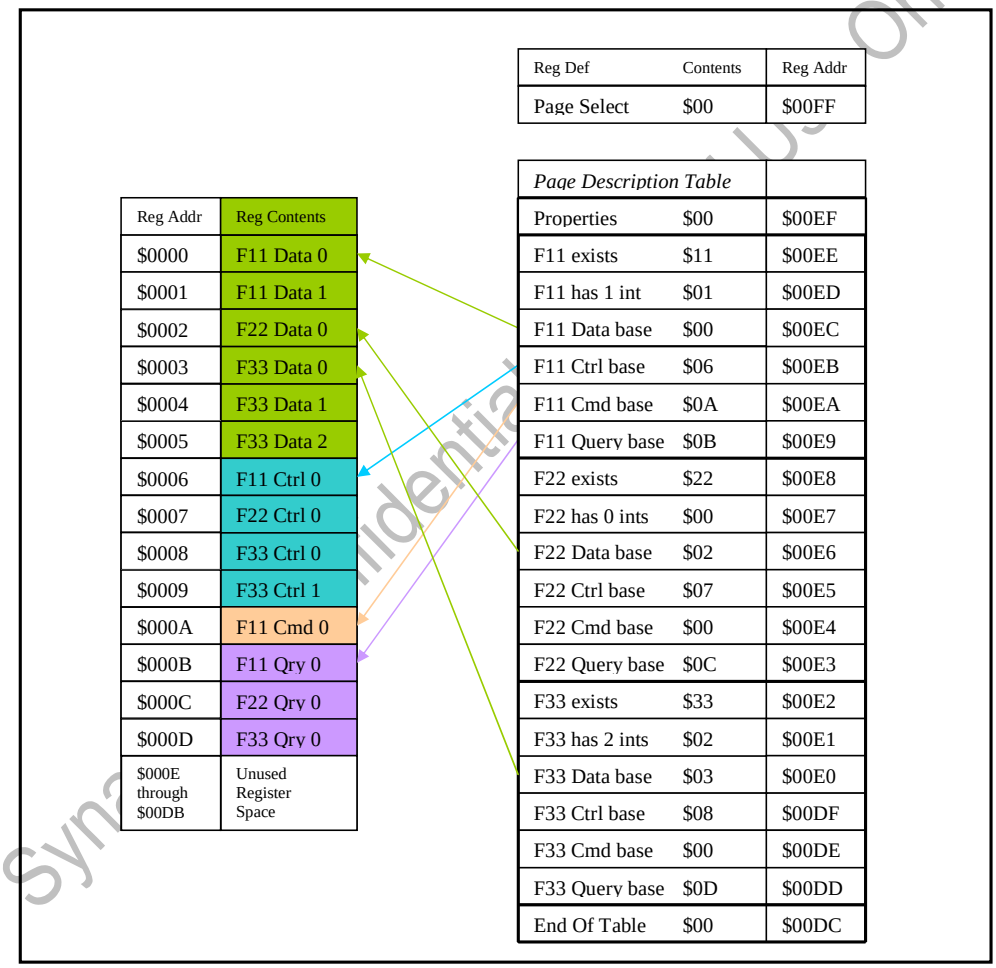

Figure 6. Sample register map

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 7. 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 need to 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 said to be *committed* when the last register in the region is written. Writing the last register in a coherent region triggers an atomic update inside the device firmware for all of the written data belonging to 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-coherency 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.

2.6.3. Coherent regions

Private data registers may be grouped into coherent regions, if desired. The exact semantics of these private coherent regions will be defined by the private function.

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:

Initially, RMI is defined for three physical layers:

- RMI on I²C: See section **Error! Reference source not found.**
- RMI on SMBus: See section 3.2.
- RMI on four-wire SPI: See section 3.3.

Deleted: 3.1

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.

Formatted: Left

Table 2. 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 8 is an example of a Block Read operation in which the host reads one RMI register from address N.

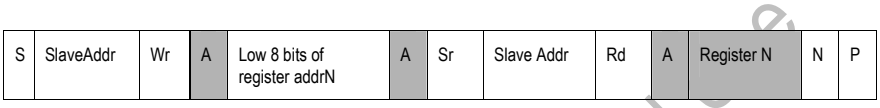

Figure 8. Block Read operation example: host reads one register

Deleted: 9

Figure 9 is an example of a Block Read operation in which the host reads four consecutive RMI registers starting from address N.

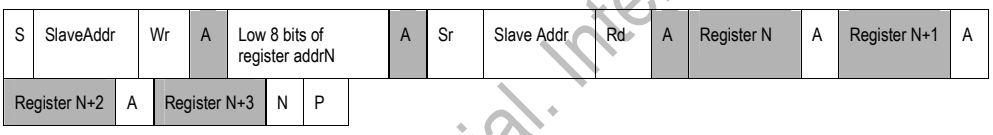

Figure 9. Block Read operation example: host reads four registers

Deleted: 10

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

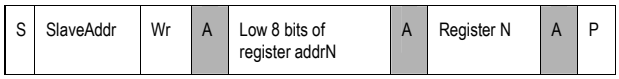

Figure 10, Block Write operation example: host writes to a single register

Deleted: 11

Figure 11 is an example of a Block Write operation in which the host writes three consecutive RMI registers starting with the register at address N.

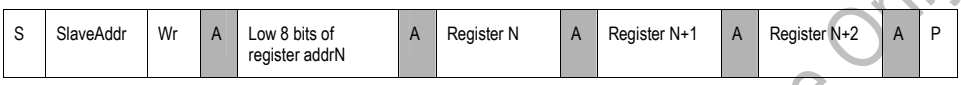

Figure 11, Block Write operation example: host writes to three registe

Deleted: 12

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 3. SMBus transfer protocol terms

Deleted: 34

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:

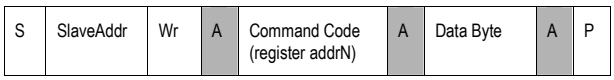

Figure 12. SMBus Byte Write command example

Deleted: 13

- 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:

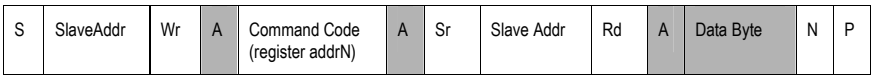

Figure 13, SMBus Byte Read command example

Deleted: 14

- 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:

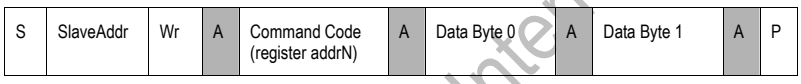

Figure 14, SMBus Word Write command example

Deleted: 15

- 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:

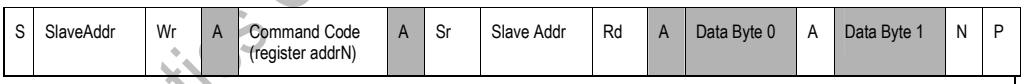

Figure 15, SMBus Word Read command example

Deleted: 16

- 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 16 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'.

Deleted: Figure 17

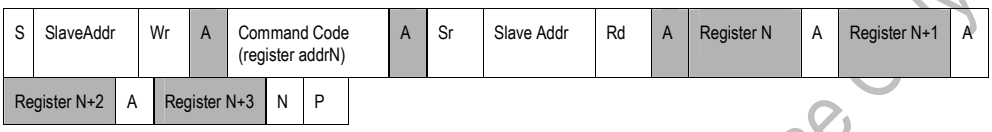

Figure 16. Multi-register Read operation example

Deleted: 17

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

Deleted: Figure 18

Figure 17. Multi-register Write operation example

Deleted: 18

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, $\overline{SS}$, or Chip Select, $\overline{CS}$. 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 18 and

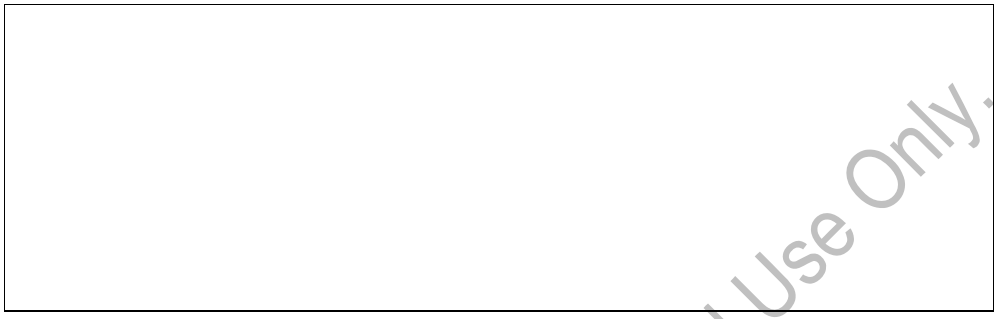

Figure 19). While SSB is high, the RMI device (the SPI slave) floats its MISO pin and ignores the MOSI and SCK pins; this allows multiple devices to share the MISO, MOSI, and SCK pins provided that each device receives a separate SSB signal. While SSB is low, the device drives the MISO pin, and it samples 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.

Deleted: ¶
¶
Figure 19

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).

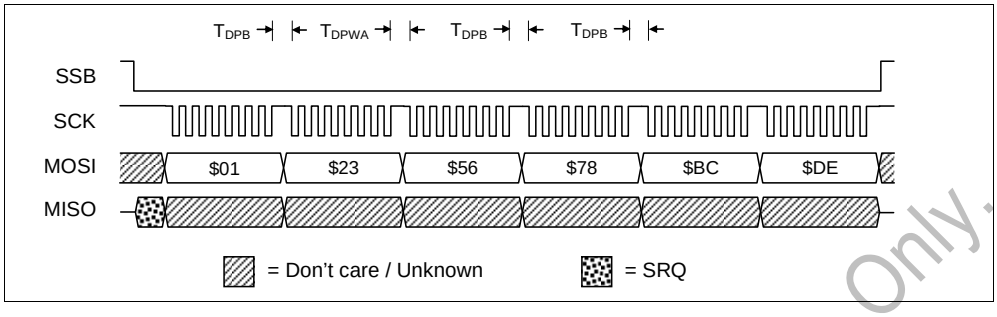

Figure 18, RMI-on-SPI write transaction
(assuming CPOL = 1 and CPHA = 1)

Deleted: 19

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

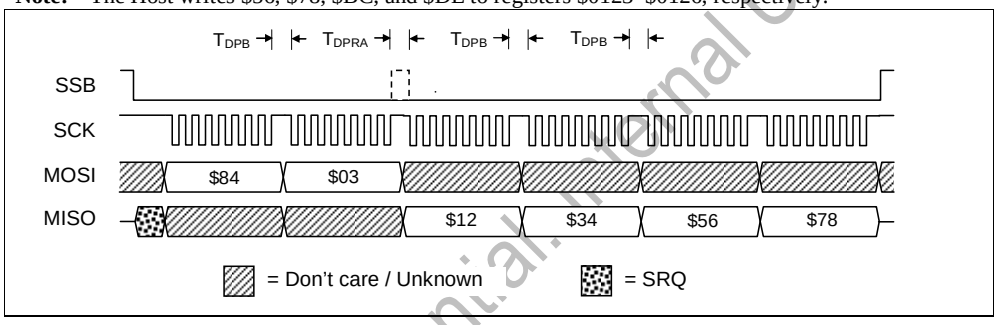

Figure 19, RMI-on-SPI read transaction
(assuming CPOL = 1 and CPHA = 1)

Deleted: 20

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.

In the SRQ option, the RMI device drives the attention signal onto MISO as soon as SSB falls (see Figure 18 and

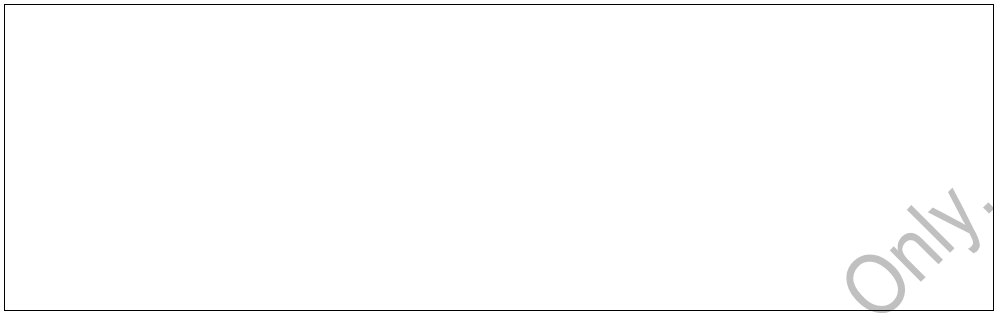

Figure 19).

Deleted: ¶
¶
Figure 19

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

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.

Name	7	6	5	4	3	2	1	0
F01_RMI_Query0	Manufacturer ID							

Figure 20, Function $01 Manufacturer ID Query register

Deleted: 21

4.1.2. F01_RMI_Query1: product properties query

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

Name	7	6	5	4	3	2	1	0
F01_RMI_Query1	—	HasAdjustable DozeHoldoff	HasAdjustable Doze	HasCharger Input	HasSensorID	HasLTS	NonCompliant	CustomMap

Figure 21, Function $01 Product Properties query register

Deleted: 22

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 ‘1’, this bit indicates the presence of at least one custom, non RMI-compatible register in the register address map for this device. RMI defines no other information about the custom register implementation other than to flag its existence.

NonCompliant (F01_RMI_Query1, bit 1)

When ‘1’, this bit indicates that the device implements a register map that is not compliant with the RMI specification.

HasLTS (F01_RMI_Query1, bit 2)

When ‘1’, the *Init Reflash* command (F01_RMI_Cmd0, bit 7) is supported and register F01_RMI_Ctrl2 exists.

HasSensorID (F01_RMI_Query1, bit 3)

When ‘1’, this bit indicates that the *Sensor ID* query register (F01_RMI_Query22) exists.

HasChargerInput (F01_RMI_Query1, bit 4)

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

HasAdjustableDoze (F01_RMI_Query1, bit 5)

When ‘1’, registers F01_RMI_Ctrl3 and F01_RMI_Ctrl4 exist.

HasAdjustableDozeHoldoff (F01_RMI_Query1, bit 6)

When ‘1’, register F01_RMI_Ctrl5 exists.

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 and F01_RMI_Query3: product information queries

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

Name	7	6	5	4	3	2	1	0
F01_RMI_Query2	—	Product Info 0						
F01_RMI_Query3	—	Product Info 1						

Figure 22. Function $01Product Information Query registers

Deleted: 23

4.1.4. 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.

Name	7	6	5	4	3	2	1	0
F01_RMI_Query4	Month (bits 2:0)			Year				
F01_RMI_Query5	CP2	CP1	Day					Month (bit 3)
F01_RMI_Query6	Wafer Lot ID 0 LSB							
F01_RMI_Query7	Wafer Lot ID 0 MSB							
F01_RMI_Query8	Wafer Lot ID 1 LSB							
F01_RMI_Query9	Wafer Lot ID 1 MSB							
F01_RMI_Query10	Wafer Lot ID 2 LSB							

Figure 23. Function $01 Device Serialization Query registers

Deleted: 24

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 as shown in Figure 23. 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.”

Wafer Lot ID (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.

4.1.5. F01_RMI_Query11 through F01_RMI_Query20: product ID queries

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

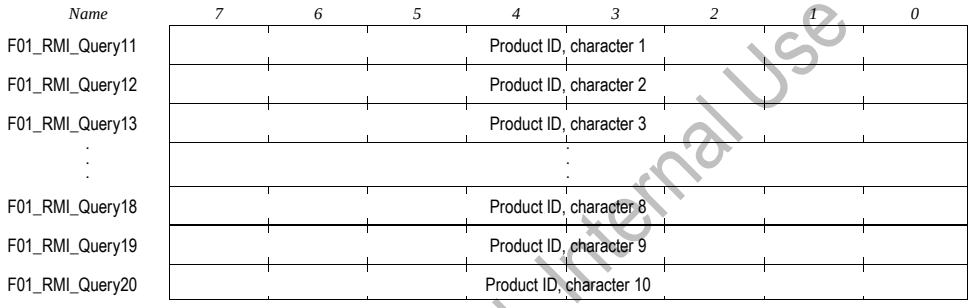

Figure 24, Function $01 Product ID Query registers

Deleted: 25

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.

4.1.6. F01_RMI_Query21: reserved

This register is reserved.

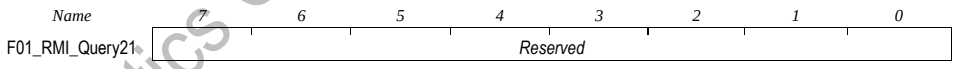

Figure 25, Function $01 Reserved query register

Deleted: 26

4.1.7. 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.

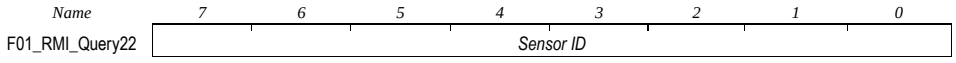

Figure 26, Function $01 Sensor ID query register

Deleted: 27

4.1.8. F01_RMI_Query23 through F01_RMI_Query31: reserved T1320 manufacturing information

These registers are reserved (private). They report manufacturing information for the T1320 touch controller.

4.2. Function $01: control registers

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.

Name	7	6	5	4	3	2	1	0
F01_RMI_Ctrl0	Configured	ReportRate	Charger Input	—	—	NoSleep	Sleep Mode	

Figure 27. Function $01 Device Control register

Deleted: 28

The bits of this register are defined as follows:

Sleep Mode (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.

Sleep Mode = '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.

Sleep Mode = '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.

Sleep Modes = '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.

ChargerInput (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 (see section 4.3.1).

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F01_RMI_Ctrl1.*	Int Enable 7	Int Enable 6	Int Enable 5	Int Enable 4	Int Enable 3	Int Enable 2	Int Enable 1	Int Enable 0

Figure 28. Function $01 Interrupt Enable register

Deleted: 29

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, bits 7:0)

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. A value of 0 indicates that the device will not doze. The maximum doze interval is 2.55 seconds.

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F01_RMI_Ctrl2	Doze Interval							

Figure 29. Function $02 Doze Interval register

Deleted: 30

4.2.4. F01_RMI_Ctrl3: wakeup threshold register

WakeupThreshold (F01_RMI_Ctrl3, bits 7:0)

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

Low values for the doze 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.

Name	7	6	5	4	3	2	1	0
F01_RMI_Ctrl3	Wakeup Threshold							

Figure 30. Function $03 Wakeup Threshold register

Deleted: 31

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, bits 7:0)

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.

0.5-second units: 0x00 = 0.5 seconds, 0xFF = 128 seconds.

Name	7	6	5	4	3	2	1	0
F01_RMI_Ctrl5	Doze Holdoff							

Figure 31. Function $04 Doze Holdoff register

Deleted: 32

4.3. Function $01: data registers

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.

Name	7	6	5	4	3	2	1	0
F01_RMI_Data0	Unconfigured	FlashProg	—	—	Status Code			

Figure 32. Function $01 Device Status register

Deleted: 33

The bits of this register are defined as follows:

Status Code (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' (see section 4.2.1).

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.

Deleted: 12

A ‘0’ indicates the device is operating in UI (User Interface) 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’.

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F01_RMI_Data1.*	Int Request 7	Int Request 6	Int Request 5	Int Request 4	Int Request 3	Int Request 2	Int Request 1	Int Request 0

Figure 33. Function $01 Interrupt Status register

Deleted: *34*

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

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.

Name	7	6	5	4	3	2	1	0
F01_RMI_Cmd0	Init Reflash	–	–	–	–	–	Shutdown	Reset

Figure 34. Function $01 Device Command register

Deleted: 35

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.

Shutdown (F01_RMI_Cmd0, bit 1)

Important: The shutdown feature is only available on products that implement an active-low ATTN signal that is driven in open-drain fashion with a pull-up resistor.

Writing a '1' to this bit causes the device to shut down *all* processing, although it will remain powered. When shut down, the RMI device:

- uses minimum possible power,
- ceases responding to all analog sensing or digital input events, and
- disconnects itself from the physical interface bus, becoming unresponsive to all host interface commands. The host can still use the physical interface bus talk to other devices on the same bus.

To enter the shutdown state, a host must perform the following sequence of events:

1. Host must disable its ATTN interrupt (if any).
2. Host must drive ATTN to '0'.
3. Host must write the *Shutdown* bit to '1' in the Device Command register.

To exit the shutdown state, a host must either power-cycle the device, or drive ATTN to '1' for a short period of time (nominally 100 microseconds). Either of these operations cause the device to perform a full power-on reset.

To exit the shutdown state using the ATTN reset method, the following sequence must occur:

1. The host must drive ATTN to '1' for 100 microseconds (nominally).
2. The host must reconfigure its ATTN signal for normal operation (typically as an interrupt input).
3. As part of the initialization sequence, the device drives ATTN to '1' in an open-drain fashion indicating that it is not ready to perform RMI communications.
4. Host must wait for ATTN to become asserted ('0'), indicating that the device is ready to perform RMI communications.
5. Host performs a standard power-on initialization of the RMI device (if required).

See section 4.2.1 for low-power methods that do not require the host to reset and reinitialize the RMI device upon exit.

Formatted: Font color: Green

Deleted: 4.2.1

Reserved (F01_RMI_Cmd0, bits 6:2)

These command bits are *reserved*.

InitReflash (F01_RMI_Cmd0, bit 7)

This command exists only if *HasLTS* (F01_RMI_Query1, bit 2) reports as '1'.

Prepares for a reflash operation. This command is intended to be used by LTS devices, which must prepare for a reflash operation by configuring all slave devices to run from their internal clocks.

Reserved (F01_RMI_Cmd0, bits 7:1)

These command bits are *reserved*.

5. Function $05: Image reporting for T1320

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.

5.1. Function $05: query registers

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

Name	7	6	5	4	3	2	1	0
F05_AD_Query0	Reserved							
F05_AD_Query1	Reserved							
F05_AD_Query2	Reserved							
F05_AD_Query3	Has16BitDelta							
F05_AD_Query4	SizeOfF05Image Window (bytes)							
F05_AD_Query5	Reserved							

F05_AD_Query0	Reserved				NumberOfReceiverElectrodes			
F05_AD_Query1	Reserved				NumberOfTransmitterElectrodes			

Figure 35. Function $05 query registers

Deleted: 36

The bits of these registers 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.

NumberOfTransmitterElectrodes (F05_AD_Query1, bits 5:0)

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

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.

SizeOfF05ImageWindow (F05_AD_ Query4, bits 7:0)

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).

5.2. Function $05: control registers

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

Figure 36. Function $05 control registers

Deleted: 37

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.

5.3. Function $05: data registers

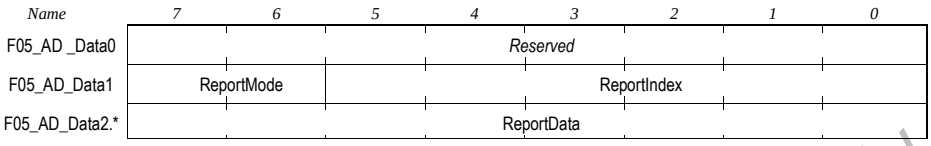

Figure 37. Function $05 data registers

Deleted: 38

The bits of these registers 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.

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*.

5.4. 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).

5.5. 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.

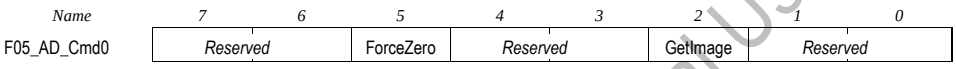

Figure 38, Function $05 command registers

Deleted: 39

The bits of this register are defined as follows:

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'.

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.

5.6. 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.

6. Function $07: Image Reporting for LTS

Function $07 should be documented in the *RMI4 Specification: Internal* (PN 511-000117-01) and the *RMI4 Specification: NDA* (PN: 511-000405-01). This information should not be included in the public-facing *RMI4 Specification* (PN 511-000136-01).

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 F$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 F$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 F$01 is not supported while F$07 is being used. After using $F07, the device must be reset in order to return it to normal operation.

6.1. Function $07: query registers

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

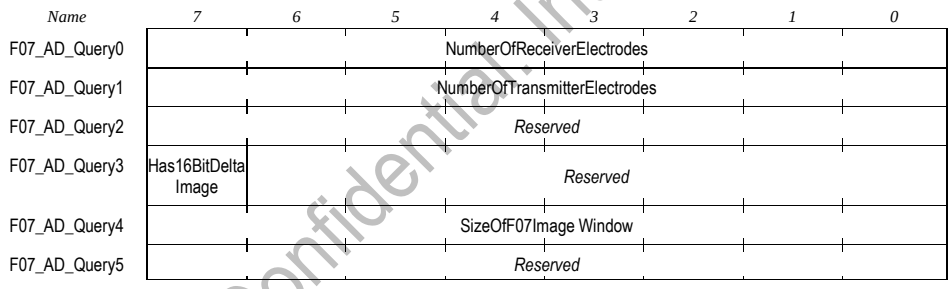

Figure 39, Function $07 query registers

Deleted: 40

The bits of these registers are defined as follows:

NumberOfReceiverElectrodes (F07_AD_Query0)
This field reports the number of sensor electrodes available in the design.

NumberOfTransmitterElectrodes (F07_AD_Query1)
This field reports the number of drive electrodes available in the design.

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)

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).

6.2. Function $07: control registers

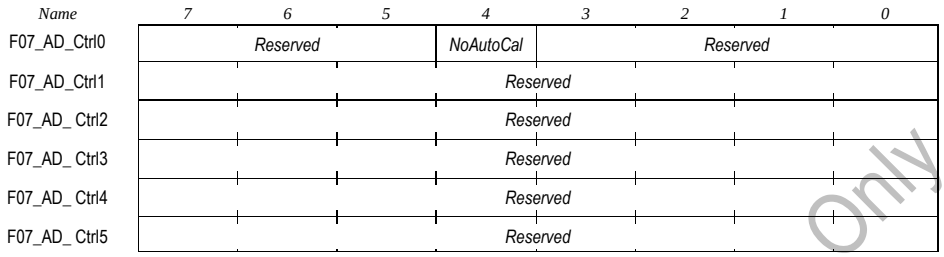

Figure 40. Function $07 control registers

Deleted: 41

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.

6.3. Function $07: data registers

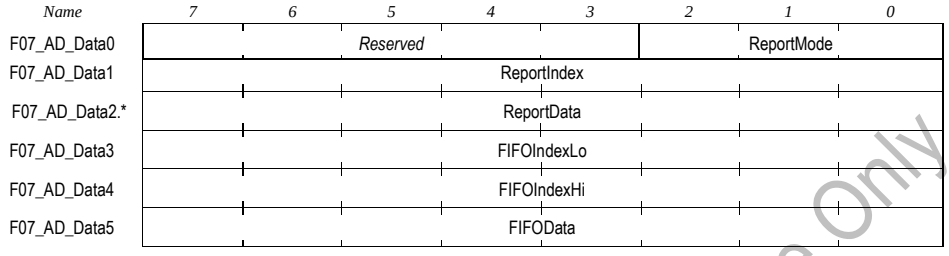

Figure 41. Function $07 data registers

Deleted: 42

The bits of these registers 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.

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.*

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*.

FIFOIndexLo, FIFOIndexHi (F07_AD_Data3, 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.

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.

6.4. 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).

6.5. 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.

Name	7	6	5	4	3	2	1	0
F07_AD_Cmd0	Reserved		ForceZero	Reserved		GetImage	Reserved	

Figure 42. Function $07 command registers

Deleted: 43

The bits of this register are defined as follows:

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.

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.*

6.6. 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.

7. Function $08: Built-in self-test for T1021

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.

Extended BIST capabilities can be backward-compatible with many of the OneTouch registers, and host-driven test mode capabilities can provide new in situ monitoring tools and SAFEPass functionality. Reporting and monitoring is as modeless as possible, but some commands may inhibit sensing until they are complete and may cause a re-zero or re-servo event.

BIST functionality can also be further extended by a host using Function $84, Analog Diagnostic function. Function $84 contains controls for many of the core analog diagnostic registers, which allow capture, configuration, and interpretation of low-level parameters (for example, ADCs).

All registers of the BIST function should be considered private unless they specifically need to be released to a customer. The control limit registers may be Flash-backed or RAM-backed as necessary.

7.1. 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).

Running BIST before POR should only delay the completion of any test command, but it is not characterized. It should be as tolerant as possible, but waiting for POR is recommended.

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 registers can also be written dynamically to allow testing with different values, is necessary. In particular, each TestLimit control register may be set to zero (0) to use internal BIST limits.

The internal BIST limits may be per pixel limits, and may be compressed as necessary for ROM requirements. If no other internal BIST limits are available then the control register FLASH defaults may be used. The presence of internal BIST limits are on a Test by Test basis.

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*). The Test Limit Control Registers will be run with default values, or may be changed if Host set limits are available.

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.

7.2. 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.

Name	7	6	5	4	3	2	1	0
F08_BIST_Query0	Limit Register Count							
F08_BIST_Query1	HostTestEn	—	—	—	—	—	—	—

Figure 43. Function $08 query registers

Deleted: 44

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.

Limit Register Count (F08_BIST_Query0)

Indicates the number of control registers used to set the limits for BIST testing capabilities. This is typically 6, but can be 3 for some T1021 devices.

HostTestEn (F08_BIST_Query1, bit 7)

Indicates that BIST host-level testing is available. Test limit control (such as *F09_BIST_Ctrl0:5*) can be updated by the host before the BIST command is run and those values will be used to determine Failures.

Host-level testing allows an extended test command set with test time configured control limits for those tests. Reserved bits can be used either to extend the *Limit Register Count* or to otherwise indicate changes to the register map.

7.3. 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:

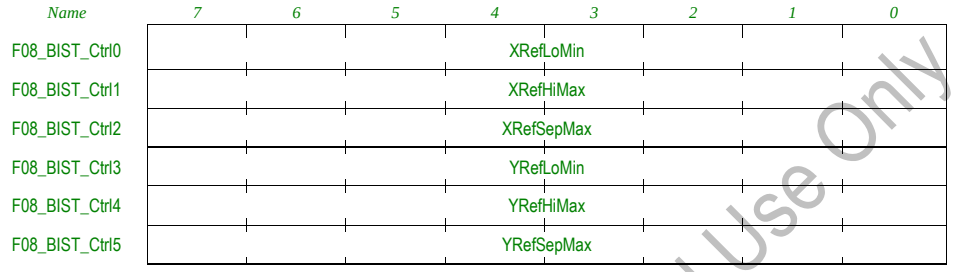

Figure 44. Internal Function $08 Limit control registers

Deleted: 45

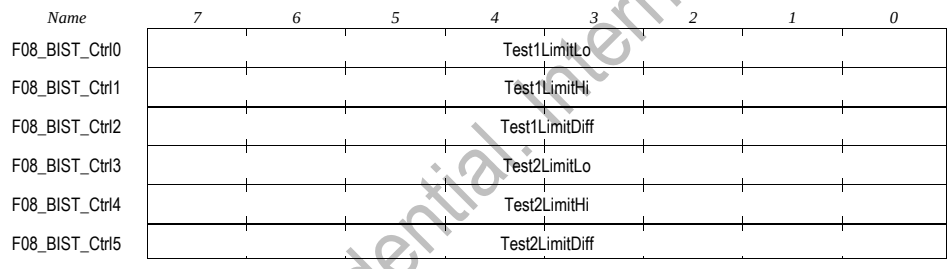

Figure 45. Function $08 Limit control registers

Deleted: 46

These registers are defined as follows:

XRefLoMin (F08_BIST_Ctrl0)
For T1021 devices, controls the acceptance Low BIST Limit for RefLo.
May only be for only XRefLo , if YRefLo limit is separately set.
Values below this fail.

XRefHiMax (F08_BIST_Ctrl1)
For T1021 devices, controls the acceptance High BIST Limit for RefHi.
May only be for only XRefHi , if YRefHi limit is separately set.
Values above this fail.

XRefSepMin (F08_BIST_Ctrl2)
For T1021 devices, controls the acceptance High BIST Limit for RefHi-RefLo.
May only be for only XRefSep , if YRefSep limit is separately set.
Values above this fail.

YRefLoMin (F08_BIST_Ctrl3)
For T1021 devices, controls the acceptance Low BIST Limit for YRefLo, if separately set.
Values below this fail.

YRefHiMax (F08_BIST_Ctrl4)
For T1021 devices, controls the acceptance High BIST Limit for YRefHi, if separately set. Values above this fail.

YRefSepMin (F08_BIST_Ctrl5)
For T1021 devices, controls the acceptance High BIST Limit for YRefHi-YRefLo, if set. Values above this fail.

Test1LimitLo (F08_BIST_Ctrl0)
Controls the acceptance Low BIST Limit for Test1.

Test1LimitHi (F08_BIST_Ctrl1)
Controls the acceptance High BIST Limit for Test1.

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

Test2LimitLo (F08_BIST_Ctrl3)
Controls the acceptance Low BIST Limit for Test2.

Test2LimitHi (F08_BIST_Ctrl4)
Controls the acceptance High BIST Limit for Test2.

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

7.4. 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.

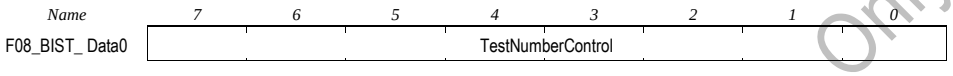

Figure 46. Function $08 Test Number Control data register

Deleted: 47

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.

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

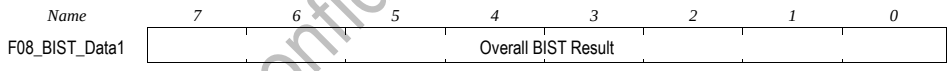

Figure 47. Function $08 Overall BIST Result data register

Deleted: 48

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.

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

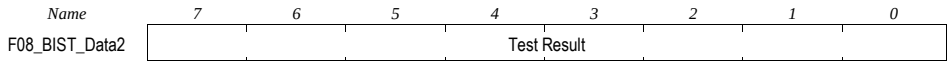

Figure 48. Function $08 Test Result data register

Deleted: 49

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.

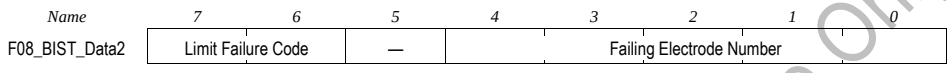

Figure 49. Function $08 Test Result (Test1)

Deleted: 50

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.

Tests the device RefHi, RefLo, and RefSep= RefHi-RefLo against the BIST limits in registers F08_BIST_Ctrl0:2.Reservo (F84_AD_Cmd1, bit 1). Should only require some register read and compares, but does require that *IsServoing (F84_AD_COND, bit 4*) is clear before it can be run. In the case that Test1 RefLo min limit causes the failure, this is the electrode with the lowest ADC. In the case that the Test1 RefHi max limit causes the failure, this is the electrode with the highest ADC. In the case that the Test1 RefSep difference limit is exceeded, then the electrode with the lowest ADCs is reported.

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.

7.5. 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. If multiple commands have been issued, then an interrupt need only be issued when the lowest enabled bit has been cleared.

Future unimplemented extended BIST capabilities may assert an interrupt when data changes without a command bit being set, but they will require a modal control bit to be set.

7.6. 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. Only the lowest bit will execute if multiple bits are set.

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

Figure 50. Function $08 command register

Deleted: 51

RunBIST (F08_BIST_Cmd0, bit 0)

Runs the BIST using the test number and limits set in the data and control registers. If *HostTestEn* (F08_BIST_Query1, bit 7) is enabled, then non-default values written to the control registers should be used.

8. Function $09: BIST for T1320

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.

Extended BIST capabilities can be backward-compatible with many of the OneTouch registers, and host-driven test mode capabilities can provide new in situ monitoring tools and SAFEPass functionality. Reporting and monitoring is as modeless as possible, but some commands may inhibit sensing until they are complete and may cause a re-zero or re-servo event.

BIST functionality can also be further extended by a host using Function $85, Analog Diagnostic functions. They contain controls for many of the core analog diagnostic registers, which allow capture, configuration, and interpretation of low-level parameters (for example, ADCs). All registers of the BIST function should be considered private unless they specifically need to be released to a customer. The control limit registers may be Flash-backed or RAM-backed as necessary.

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).

Running BIST before POR should only delay the completion of any test command, but it is not characterized. It should be as tolerant as possible, but waiting for POR is recommended.

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 registers can also be written dynamically to allow testing with different values, is necessary. In particular, each TestLimit control register may be set to zero (0) to use internal BIST limits.

The internal BIST limits may be per pixel limits, and may be compressed as necessary for ROM requirements. If no other internal BIST limits are available then the control register FLASH defaults may be used. The presence of internal BIST limits are on a Test by Test basis.

All tests will be run with RefLo=-3pF and RefHi=+6pF for compatibility with other Test processes, and to simplify setting Limits.

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). The Test Limit Control Registers will be run with default values, which may be 0 if internal limits are available, or set by the host if enabled.

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.

8.2. 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.

Name	7	6	5	4	3	2	1	0
F09_BIST_Query0	Limit Register Count							
F09_BIST_Query1	HostTestEn	InternalLimits	—	—	—	Result Register Count		

Figure 51. Function $09 query registers

Deleted: 52

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.

Limit Register Count (F09_BIST_Query0)

Indicates the number of control registers used to set the limits for BIST testing capabilities. This is typically 6, but can be 3 for some T1320 devices.

Result Register Count (F09_BIST_Query1, bits 2:0)

Indicates the number of result registers used to report BIST test results.
This is typically 2 for T1320 devices in F$09, but may be increased.
The total number of data registers will typically be 4: 2 for indicating the Test Number and Overall Result, and 2 more to indicate the location (receiver and transmitter numbers) of the failing pixel.

InternalLimits (F09_BIST_Query1, bit 6)

Indicates that BIST internal limits are available. This allows more complete pixel level limits to be stored internal to the firmware. These may be comparable to the limits used for standard module level sensor testing. The use of these limits is controlled by setting or keeping the default corresponding test limit control register to zero (0).

HostTestEn (F09_BIST_Query1, bit 7)

Indicates that BIST host-level testing is available. Test Limit control registers (for example, F09_BIST_Ctrl0 through 5) can be updated by the host before the BIST command is run and those values will be used to determine failures.

Host-level testing allows an extended test command set with test time configured control limits for those tests. Reserved bits can be used either to extend the *Limit Register Count* or to otherwise indicate changes to the register map.

8.3. Function $09: control registers

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

Name	7	6	5	4	3	2	1	0
F09_BIST_Ctrl0	Baseline Min							
F09_BIST_Ctrl1	Baseline Max							
F09_BIST_Ctrl2	Baseline Noise Max							
F09_BIST_Ctrl3	Crtc Typical Max							
F09_BIST_Ctrl4	Crtc Error Max							
F09_BIST_Ctrl5	Floating Transmitter Diff Max							

Figure 52. Internal Function $09 Limit control registers

Deleted: 53

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							

Figure 53. Function $09 Limit control registers

Deleted: 54

These registers are defined as follows:

Baseline Min(F09_BIST_Ctrl0)

Controls the acceptance Low BIST Limit for the Baseline.
The limit should be shifted left 4 bits (x16) and multiplied by *BurstsPerCluster* (F85_AD_Ctrl2, bits 2:0) to keep the range correct, and then compared to each pixel in the Baseline. The absolute minimum Baseline is ~0.5pF (~3185ADCs for two Bursts and -3pF to +6pF = 9pF RefSep).This can be implemented by finding the minimum pixel in a single re-zeroed Baseline. Zero causes internal limits to be used if available in F09_BIST_Query1, bit 6. Values below this fail.

Baseline Max (F09_BIST_Ctrl1)

Controls the acceptance High BIST Limit for the Baseline
The limit should be shifted left 4 bits (x16) and multiplied by *BurstsPerCluster* (F85_AD_Ctrl2, bits 2:0) to keep the range correct, and then compared to each pixel in the Baseline. The absolute maximum Baseline is ~5.5pF (~7735ADCs for two Bursts and -3pF to +6pF = 9pF RefSep). This can be implemented by finding the maximum pixel in a single re-zeroed Baseline. Zero causes internal limits to be used if available in F09_BIST_Query1, bit 6. Values above this fail.

Baseline Noise Max (F09_BIST_Ctrl2)

Controls the acceptance High BIST Limit for Baseline Noise.
Noise is measured as the per-maximum per-pixel peak-to-peak difference between the maximum and minimum image deltas from the baseline over at least 20 frames.

This can be implemented by recording the delta result for each pixel in a delta image that exceeds the current maximum into the maximum image, and recording the delta result for each pixel in a delta image that falls below the current minimum into the minimum image. The delta results should be bleed-over corrected, common-mode compensated, and scaled to signed bytes as they would be for any delta image. The maximum noise should be less than the finger threshold / 4. The tested result value would then be found from the maximum difference at each pixel between the maximum delta image and the minimum delta image. Zero causes internal limits to be used if available in F09_BIST_Query1, bit 6. Values above this fail.

Crtc Typical Max (F09_BIST_Ctrl3) – optional

Controls the acceptance High BIST Limit for the Crtc response error.
The limit should be shifted left 4 bits (x16) and multiplied by *BurstsPerCluster* (F85_AD_Ctrl2, bits 2:0) to keep the range correct for comparison to the expected diagonal response to Crtc. Each sensor response should be above the Typical Min value. This will be a delta near 1pF or dADC=4095*BurstsPerCluster*1pF/RefSep (~910ADCs for two Bursts and -3pF to +6pF = 9pF RefSep) when Crtc is on that channel and the error is the absolute value of the difference. A single transmitter (for example, Row0) should always be excited and the final corrected cluster result without any Crtc selected should be used as the Baseline. Then the Crtc should be selected for each receiver and the dADC response of the receiver corresponding to the Crtc selection should be compared to the scaled minimum limit. A Zero (0) limit causes internal limits to be used if available in F09_BIST_Query1, bit 6. Values below this fail.

Crtc Error Max (F09_BIST_Ctrl4) – optional

Controls the acceptance High BIST Limit for the Crtc Error.
The limit should be shifted left 4 bits (x16) and multiplied by *BurstsPerCluster* (F85_AD_Ctrl2, bits 2:0) to keep the range correct for comparison to the expected off diagonal response to Crtc. Each sensor response should be below the Error Max value. This will be a delta near zero when Crtc is not on that channel, and typically less than +/-10% of typical response when Crtc is on that channel or dADC=820*BurstsPerCluster*1pF/RefSep (~182ADCs for two Bursts and -3pF to +6pF = 9pF RefSep). Baseline Noise can also affect this result. A single transmitter (for example, Row0) should always be excited and the final corrected cluster result without any Crtc selected should be used as the Baseline. Then the Crtc should be selected for each receiver and the maximum dADC response of all the receivers **not** corresponding to the Crtc selection should be compared to the scaled maximum limit. These values may be collected simultaneous with the CrtcTypicalMin. A Zero (0) limit causes internal limits to be used if available in F09_BIST_Query1, bit 6. Values above this fail.

Floating Transmitter Diff Max (F09_BIST_Ctrl5) – optional

Controls the acceptance Difference BIST Limit for Floating Transmitters.
The limit should be shifted left 4 bits (x16) and multiplied by *BurstsPerCluster* (F85_AD_Ctrl2, bits 2:0) to keep the range correct for comparison to the difference between the nominal Baseline with non-active Transmitters grounded and a Baseline with non-active Transmitters floating. When non-actives are floating each Transmitter will be disabled by default with RCVR_EN set to 0 until it is actively transmitting and XMTR_INP can remain 1 rather than switching the input and leaving non-actives enabled as is done for a nominal Baseline. The entire Baseline array should be collected and bleed-over corrected. Typically a change in the array of less than +50% of the nominal Baseline is expected. For a 2pF Baseline dADC=2048*BurstsPerCluster*2pF/RefSep (~910ADCs for two Bursts and -3pF to +6pF = 9pF RefSep).

Each Transmitter should be excited in the same sequence as a normal Baseline including any Button Rows for each receiver in the same order and the maximum dADC between the Grounded and Floating Baseline for each pixel should be compared to the scaled maximum limit. The Floating values may be differenced from the same values as Nominal Baseline collected and used for BaselineMax/Min comparison. The difference should be compared to the scaled maximum Difference Limit for this test. A Zero (0) limit causes internal limits to be used if available in F09_BIST_Query1, bit 6.
Values above this fail.

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.

8.4. 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.

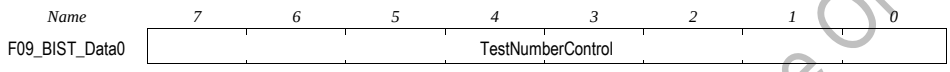

Figure 54. Function $09 Test Number Control data register

Deleted: 55

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.

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

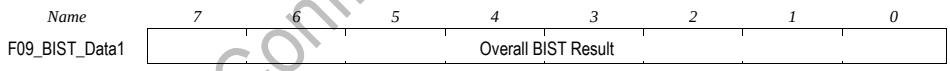

Figure 55. Function $09 Overall BIST Result data register

Deleted: 56

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.

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

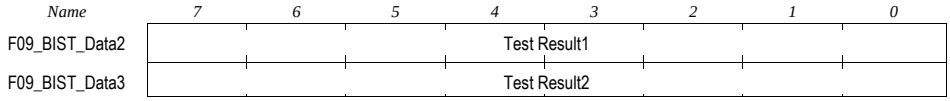

Figure 56. Function $09 Test Results

Deleted: 57

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:

Figure 57. Function $09 Test Results, Test1 example

Deleted: 58

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.

Note: Multiple pixels (transmitter/receiver intersection) may fail and only the first (or largest) deviation from the limit will be reported. In the future, non-zero Receiver Number and Transmitter numbers may be used by the host to indicate which intersection to test with the test limits that are assigned for that BIST run. Normally, however, a device failing BIST will be analyzed using a separate test fixture.

8.5. 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. If multiple commands have been issued, then an interrupt need only be issued when the lowest enabled bit has been cleared.

Future unimplemented extended BIST capabilities may assert an interrupt when data changes without a command bit being set, but they will require a modal control bit to be set.

8.6. 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. Only the lowest bit will execute if multiple bits are set.

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

Figure 58. Function $09 command register

Deleted: 59

RunBIST (F09_BIST_Cmd0)

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

9. 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.

9.1. 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 21.

Deleted: 2

For a description of Function $11 version '0', see PN 511-000136-01 Rev E or PN: 511-000117-01 Rev 34.

9.2. Number of 2-D sensors

The Number of Sensors 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.

9.3. 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.

9.3.1. F11_2D_Query0: per-device query registers

RMI Function $11 defines a single per-device query register.

Name	7	6	5	4	3	2	1	0
F11_2D_Query0	—	—	—	HasQuery11	HasQuery9	NumberOfSensors		

Figure 59. Function $11 Per-device query register

Deleted: 60

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.

9.3.2. F11_2D_Query1 through F11_2D_Query11: per-sensor query registers

Each 2-D sensor has its own set of query registers, used to describe its own particular characteristics.

Name	7	6	5	4	3	2	1	0
F11_2D_Query1	—	HasSensitivity Adjust	Has Gestures	HasAbs	HasRel	NumberOfFingers		
F11_2D_Query2	—	Number of X Electrodes						
F11_2D_Query3	—	Number of Y Electrodes						
F11_2D_Query4	—	Maximum Electrodes						
F11_2D_Query5	—	—	HasBending Correction	HasDribble	HasAdjust Hyst	HasAnchored Finger	Absolute Data Size	
F11_2D_Query6	—	—	—	—	—	—	—	—
F11_2D_Query7	Has Chiral Scroll	Has Pinch	Has Press	Has Flick	Has Early Tap	Has Double Tap	Has Tap and Hold	Has Single Tap
F11_2D_Query8	Has MultiFinger Scroll Inertia	Has MultiFinger Scroll Edge Motion	Has MultiFinger Scroll	Individual ScrollZones	Has Scroll Zones	Has TouchShapes	Has Rotate	Has Palm Detect
F11_2D_Query9	—	—	Has Contact Geometry	Has Two Pen Thresholds	Has Suppress on Palm Detect	Has Palm Detect Sensitivity	Has Proximity	Has Pen
F11_2D_Query10	—	—	—	Number of TouchShapes				
F11_2D_Query11	Has Fast Flick	Has XY Clip	Has Segmentation Aggressiveness	Has Finger Size	Has Pitch Info	Has W Tuning	Has Algorithm Selection	Has Z Tuning

Figure 60. Function $11 Per-sensor query registers

Deleted: 61

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

9.3.3. 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 4

Deleted: 45

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.

The two reserved fields are specifically intended for customer requirements for other finger counts. For example, '110' could be defined as "product-specific finger count: consult your product specification," while '111' could be "the actual finger count is found in query register X" (for example, to support an 8-bit finger count in the future).

HasRel (F11_2D_Query1, bit 3)

If *HasRel* 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.

HasAbs (F11_2D_Query1, bit 4)

If *HasAbs* 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.

Reserved (F11_2D_Query1, bit 7)

Note: The sensor mapping controls are not available starting with Function $11 version 1.

9.3.4. 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 the corresponding axis. The sum of these cannot exceed *MaximumElectrodes*.

9.3.5. 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.

9.3.6. F11_2D_Query5: absolute data source

This query register is present only if *HasAbs* 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.*

Note: All other encodings are temporarily on hold. Synaptics' internal tool chain always needs all of the information. It might make sense to use this field to indicate how the registers pack, instead of defining what is present, not-present, or present in two different fashions. For example, we might always report all five fields, but in one case pack the data as all the XY data for all N fingers, followed by all WZ data for all N fingers. A host that doesn't want the WZ data could skip over reading those registers. This needs to change to 'both small and large'. Synaptics internal tool chain needs the information stored in the large report, even if the customer does not.

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, interrupts stop 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'.

HasAntiBending (F11_2D_Query5, bit 5)

If *HasAntiBending* reports as ‘1’, then register F11_2D_Ctrl28 and registers F11_2D_Ctrl52 through F11_2D_Ctrl57 exist.

9.3.7. F11_2D_Query6: relative data source

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

9.3.8. 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)

If *HasChiralScroll* reports as '1', the sensor supports the chiral scrolling gesture.

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.

Size and location of each shape is specified at build time.

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).

Width of each of the four scroll zones is specified at build time.

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.

9.3.9. 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. The register map will contain the F11_2D_Ctrl20 pen control register and the F11_2D_Ctrl21 pen Z threshold register.

HasProximity (F11_2D_Query9, bit 1)

If *HasProximity* reports as '1', detection of fingers nearby – but not touching – the sensor is supported. The register map will contain the F11_2D_Ctrl22 proximity control register, the F11_2D_Ctrl23 proximity detection Z threshold register, and F11_2D_Ctrl24-26, which control the proximity delta X, Y, and Z thresholds.

HasPalmDetectSensitivity (F11_2D_Query9, bit 2)

If *HasPalmDetectSensitivity* reports as '1', the sensor supports the palm detect sensitivity feature. This feature is adjusted using the first four bits of F11_2D_Ctrl27, the *PalmDetectParameters* register.

HasSuppressOnPalmDetect (F11_2D_Query9, bit 3)

If *HasSuppressOnPalmDetect* reports as '1', the sensor supports the palm detect suppression feature. This feature is set using bit 4 of F11_2D_Ctrl27, the *PalmDetectParameters* register; when this bit is set to '1', all F$11 interrupts except PalmDetect are suppressed while a palm is detected.

HasTwoPenThresholds (F11_2D_Query9, bit 4)

If *HasTwoPenThresholds* reports as '1' and *HasPen* reports as '1', register F11_2D_Ctrl35 will contain the Pen Z Upper Threshold.

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. Data registers F11_2D_Data18.* through F11_2D_Data27.* are used to report these positions.

9.3.10. F11_2D_Query10: TouchShape support

This register is used for TouchShape support. It only exists 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. If the *HasTouchShapes* field of F11_2D_Query8 reports as '1', then the register map will contain one or more F11_2D_Data13. The size of this register block can be calculated from the *NumberOfTouchShapes* field:

```
F11_2D_Data13_registerCount = ceil((NumberOfTouchShapes+1)/8)
```

9.3.11. F11_2D_Query11: advanced capabilities

This register identifies the sensor's advanced capabilities. It only exists 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, as found in Design Studio 4. The Z tuning settings are located in registers F11_2D_Ctrl29 through F11_2D_Ctrl33.

HasAlgorithmSelection (F11_2D_Query11, bit 1)

If *HasAlgorithmSelection* reports as '1', the sensor supports configuration of the position-interpolation and post-correction algorithm. The algorithm settings are located in register F11_2D_Ctrl34.

HasWTuning (F11_2D_Query11, bit 2)

If *HasWTuning* reports as '1', the sensor supports Wx and Wy scaling. The settings for Wx and Wy scale factors and offsets are located in registers F11_2D_Ctrl36 through F11_2D_Ctrl39.

HasPitchInfo (F11_2D_Query11, bit 3)

If *HasPitchInfo* reports as '1', the X and Y pitches of the sensor electrodes can be configured. The X and Y pitch settings are located in registers F11_2D_Ctrl40 and F11_2D_Ctrl41.

HasFingerSize (F11_2D_Query11, bit 4)

If *HasFingerSize* reports as '1', the default finger width settings for the sensor can be configured. The settings for the default finger widths on the X and Y axes, and the first finger default width, are located in registers F11_2D_Ctrl42 through F11_2D_Ctrl44.

HasSegmentationAggressiveness (F11_2D_Query11, bit 5)

If *HasSegmentationAggressiveness* reports as '1', the sensor's ability to distinguish multiple objects close together can be configured. This setting is located in register F11_2D_Ctrl45.

HasXYClip(F11_2D_Query11, bit 6)

If *HasXYClip* reports as '1', the inactive outside borders of the sensor can be configured. The four registers, F11_2D_Ctrl46 through F11_2D_Ctrl49, control width of the left, right, upper, and lower inactive borders of the sensor.

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. The settings for minimum and maximum drumming movement, which define the gestures that are **not** a fast flick, are located in registers F11_2D_Ctrl50 and F11_2D_Ctrl51.

9.4. Function $11: control registers

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

Name	7	6	5	4	3	2	1	0
F11_2D_Ctrl0	Report BeyondClip	Dribble	RelBallistics	RelPosFilt	AbsPosFilt	ReportingMode		
F11_2D_Ctrl1	ManTracked Finger	ManTrackEn	MotionSensitivity		PalmDetectThreshold			
F11_2D_Ctrl2	DeltaXPosThreshold							
F11_2D_Ctrl3	DeltaYPosThreshold							
F11_2D_Ctrl4	Velocity							
F11_2D_Ctrl5	Acceleration							
F11_2D_Ctrl6	SensorMaxXPos 7:0							
F11_2D_Ctrl7	—	—	—	—	SensorMaxXPos 11:8			
F11_2D_Ctrl8	SensorMaxYPos 7:0							
F11_2D_Ctrl9	—	—	—	—	SensorMaxYPos 11:8			
F11_2D_Ctrl10	—	Pinch Interrupt Enable	Press Interrupt Enable	Flick Interrupt Enable	Early Tap Interrupt Enable	DoubleTap Interrupt Enable	TapAndHold Interrupt Enable	Single Tap Interrupt Enable
F11_2D_Ctrl11	—	—	—	MultiFinger Scroll Interrupt Enable	ScrollZone Interrupt Enable	TouchShape Interrupt Enable	Rotate Interrupt Enable	PalmDetect Interrupt Enable
	Note: F11_2D_Ctrl12.* and F11_2D_Ctrl13.* do not exist in Function $11 version 1. See section 9.1 for details.							
F11_2D_Ctrl14	Hysteresis Adjustment				Sensitivity Adjustment (per sensor)			
F11_2D_Ctrl15	Maximum Tap Time							
F11_2D_Ctrl16	Minimum Press Time							
F11_2D_Ctrl17	Maximum Tap Distance							
F11_2D_Ctrl18	Minimum Flick Distance							
F11_2D_Ctrl19	Minimum Flick Speed							
F11_2D_Ctrl20	—	—	—	—	—	—	Pen Jitter Filter Enable	PenDetect Interrupt Enable
F11_2D_Ctrl21	Pen Z Lower Threshold							
F11_2D_Ctrl22	—	—	—	—	—	—	Proximity Jitter Filter Enable	Proximity Detect Interrupt Enable
F11_2D_Ctrl23	Proximity Detection Z Threshold							
F11_2D_Ctrl24	Proximity Delta-X Threshold							
F11_2D_Ctrl25	Proximity Delta-Y Threshold							
F11_2D_Ctrl26	Proximity Delta-Z Threshold							

Note: This figure is continued on the next page.

Name	7	6	5	4	3	2	1	0
F11_2D_Ctrl27	—	—	—	SuppressOn PalmDetect	Palm-Detect Sensitivity			
F11_2D_Ctrl28	Multi-Finger Scroll Momentum				—	Edge Motion Enable	Multi-Finger Scroll Mode	
F11_2D_Ctrl29	Z Touch Threshold							
F11_2D_Ctrl30	Z Hysteresis							
F11_2D_Ctrl31	Small Z Threshold							
F11_2D_Ctrl32	Small Z Scale Factor							
F11_2D_Ctrl33	Large Z Scale Factor							
F11_2D_Ctrl34	Algorithm Selection							
F11_2D_Ctrl35	Pen Z Upper Threshold							
F11_2D_Ctrl36	Wx Scale Factor							
F11_2D_Ctrl37	Wx Offset							
F11_2D_Ctrl38	Wy Scale Factor							
F11_2D_Ctrl39	Wy Offset							
F11_2D_Ctrl40.*	X Pitch							
F11_2D_Ctrl41.*	Y Pitch							
F11_2D_Ctrl42.*	Finger Size X Axis							
F11_2D_Ctrl43.*	Finger Size Y Axis							
F11_2D_Ctrl44	Report Measured Size							
F11_2D_Ctrl45	Segmentation Aggressiveness							
F11_2D_Ctrl46	X Clip Left							
F11_2D_Ctrl47	X Clip Right							
F11_2D_Ctrl48	Y Clip Upper							
F11_2D_Ctrl49	Y Clip Lower							
F11_2D_Ctrl50	Minimum Drumming Separation							
F11_2D_Ctrl51	Maximum Drumming Movement							
F11_2D_Ctrl52	Bending Detection Threshold							
F11_2D_Ctrl53	Bending Touch Threshold Adjustment							
F11_2D_Ctrl54	Flexibility							
F11_2D_Ctrl55	Bend Distance / Interaction Distance							
F11_2D_Ctrl56	Bending Correction X							
F11_2D_Ctrl57	Bending Correction Y							
F11_2D_Ctrl58	Pragma Color							

Figure 61. Function $11 control registers

Deleted: 62

9.4.1. 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*.

Notes:

- The contents of the *PalmDetectThreshold* register are used in this mode.
- The contents of the *ZClipThreshold* register are ignored 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.
- The contents of the *ZClipThreshold* register are ignored in this mode.

ReportingMode = ‘011’: *Finger-presence change reporting mode*

In this mode, the absolute data source interrupt is asserted whenever finger presence changes, meaning from the state of having at least one finger to none or the other way around. Increasing or decreasing finger count while there is still a finger on pad does not generate interrupts. Changes in finger position while a finger is down do not generate interrupts either, 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.
- The contents of the *ZClipThreshold* register are ignored in this mode.

ReportingMode = ‘101’: *Z-clip reporting mode*

This mode is identical to mode 001 (Reduced Reporting mode) except that the contents of the *ZClipThreshold* register are not ignored. The absolute data source interrupt is asserted whenever one of the finger Z values is larger than *ZClipThreshold*.

ReportingMode = ‘111’: *Continuous reporting mode (private)*

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*.

AbsPosFilt (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 *AbsPosFilt* bit has an effect only when *ReportingMode* = ‘000’ (continuous when finger present).

RelPosFilt (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.

RelBallistics (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.

9.4.2. 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. Also, the ranges for Low, Medium, and High sensitivity may overlap.

ManualTrackingEn (F11_2D_Ctrl1, bit 6)

This 1-bit field specifies what entity, host or sensor firmware, is in control of determining which finger is the tracked finger:

- If ‘1’, the host selects which finger is the tracked finger.
- If this bit is ‘0’, the firmware will choose which finger is the tracked one.

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

ManuallyTrackedFinger (F11_2D_Ctrl1, bit 7)

This 1-bit field indicates which finger is being tracked:

- If ‘0’, Finger #0 is being tracked.
- If ‘1’, then 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).

This control field is only meaningful when *ManualTrackingEn* (F11_2D_Ctrl1, bit 6) is enabled. This control field is only available when *HasAnchoredFinger* reports as ‘1’.

9.4.3. F11_2D_Ctrl2 and F11_2D_Ctrl3: distance threshold

DeltaXPositionThreshold (F11_2D_Ctrl2) and

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).

9.4.4. 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.

Note: Units are TBD.

9.4.5. 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.

Notes: Units are TBD.

The absolute position is r = sqrt(x^2+y^2). The absolute velocity, v, is the difference in absolute positions from one epoch to the next. The absolute acceleration, a, is the difference in absolute velocities from one epoch to the next. All three of these quantities are vectors.

When ballistics are applied, the position deltas are adjusted according to:

Rc =	Rp + V*v + A*a
DeltaR =	Rc-Rp
Rc =	current value for the position
Rp =	previous computation for Rc
v,a =	absolute velocity and absolute acceleration
V,A =	RMI register values. These are in 4.4 fixed point notation. They range from 0.0625 to 15.9375.

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

SensorMaxXPos (F11_2D_Ctrl6,7) and

SensorMaxYPos (F11_2D_Ctrl8,9)

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 *SensorMaxXpos*, or a reported Y position of 0 or *SensorMaxYPos*, 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 *SensorMaxX/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 *SensorMaxXPos* 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 *SensorMaxYPos* 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).

Implementation Note: The internal representation of the position is expressed in sensor units. The range extends nominally to -0.5 to (N-1)+0.5, where N is the number of electrode in one axis. The position extends to the edges of the diamond pattern (for diamond patterned sensors). There may be some devices where the internal position may not extend across the nominal range.

9.4.7. 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 9.3.3).

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.

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.

9.4.8. F11_2D_Ctrl12.* does not exist

Important: The F11_2D_Ctrl12.* registers no longer exist.

9.4.9. F11_2D_Ctrl13.* does not exist

Important: The F11_2D_Ctrl13.* registers no longer exist.

9.4.10. 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.

9.4.11. 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 9.3.3) and *HasEarlyTap*, *HasSingleTap*, *HasDoubleTap*, or *HasTapAndHold* reports as '1' in the per-sensor query register F11_2D_Query7.

Maximum Tap Time (F11_2D_Ctrl15, bits 7:0)

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.

9.4.12. 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 9.3.3) and *HasPress* reports as '1' in the per-sensor query register F11_2D_Query7.

Minimum Press Time (F11_2D_Ctrl16, bits 7:0)

The minimum duration required for stationary finger(s) to generate a press gesture, in 10-millisecond units.

9.4.13. 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 9.3.3) and *HasEarlyTap*, *HasSingleTap*, *HasDoubleTap*, *HasTapAndHold*, or *HasPress* reports as '1' in the per-sensor query register F11_2D_Query7.

Maximum Tap Distance (F11_2D_Ctrl17, bits 7:0)

Determines the maximum finger movement allowed during a tap, in 0.1-millimeter units.

Maximum allowed inter-tap movement is proportional to these values, as is maximum movement allowed during a press.

9.4.14. 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 9.3.3) and *HasFlick* reports as '1' in the per-sensor query register F11_2D_Query7.

Minimum Flick Distance (F11_2D_Ctrl18, bits 7:0)

Determines the minimum finger movement for a flick gesture, in 1-millimeter units.

9.4.15. 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 9.3.3) and *HasFlick* reports as '1' in the per-sensor query register F11_2D_Query7.

Minimum Flick Speed (F11_2D_Ctrl19, bits 7:0)

Determines the minimum finger speed for a flick gesture, in 10-millimeter/second units. Small values indicate lower speeds, large values indicate higher speeds.

9.4.16. 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.

Pen Detect Interrupt Enable (F11_2D_Ctrl20, bit 0)

Setting this bit to '1' enables an interrupt in response to stylus activity on the sensor.

Pen Jitter Filter Enable (F11_2D_Ctrl20, bit 1)

Setting this bit to '1' enables the stylus anti-jitter filter.

9.4.17. 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.

Pen Z Threshold (F11_2D_Ctrl21, bits 7:0)

This is the stylus-detection lower threshold. Smaller values result in higher sensitivity.

9.4.18. F11_2D_Ctrl22: proximity control

This control register is present only if *HasPen* reports as '1' in the per-sensor query register F11_2D_Query9.

Pen Detect Interrupt Enable (F11_2D_Ctrl20, bit 0)

Setting this bit to '1' enables an interrupt in response to pen activity on the sensor.

Pen Jitter Filter Enable (F11_2D_Ctrl20, bit 1)

Setting this bit to '1' enables the pen anti-jitter filter.

9.4.19. 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.

Z Proximity Threshold (F11_2D_Ctrl23, bits 7:0)

This is the threshold for finger-proximity detection.

9.4.20. 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.

Proximity Delta-X Threshold (F11_2D_Ctrl24, bits 7:0)

In reduced-reporting modes, this is the threshold for proximate-finger movement in the direction parallel to the X-axis.

9.4.21. 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.

Proximity Delta-Y Threshold (F11_2D_Ctrl25, bits 7:0)

In reduced-reporting modes, this is the threshold for proximate-finger movement in the direction parallel to the Y-axis.

9.4.22. F11_2D_Ctrl25: proximity delta Z threshold

This control register is present only if *HasProximity* reports as '1' in the per-sensor query register F11_2D_Query9.

Proximity Delta-Z Threshold (F11_2D_Ctrl26, bits 7:0)

In reduced-reporting modes, this is the threshold for proximate-finger movement in the direction parallel to the Z-axis.

9.4.23. 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.

9.4.24. 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’.

- 0, Off: Scrolling stops immediately when fingers are lifted.
- 1-14, Momentum: Continue scrolling after fingers have been lifted, slowing until scrolling stops or the touchpad is touched again. Higher settings produce longer scroll times.
- 15, Coast: After fingers are lifted, scrolling continues at last recorded rate until the touchpad is touched again.

9.4.25. F11_2D_Ctrl29: z touch threshold

This control register is present only if *HasZTuning* (F11_2D Query11, bit 0) reports as ‘1’.

Z Touch Threshold (F11_2D_Ctrl29, bits 7:0)

Specifies the finger-arrival Z threshold. Large values may cause smaller fingers to be rejected.

9.4.26. F11_2D_Ctrl30: z hysteresis

This control register is present only if *HasZTuning* (F11_2D Query11, bit 0) reports as '1'.

Z Hysteresis (F11_2D_Ctrl30, bits 7:0)

Specifies the difference between the finger-arrival Z threshold (*ZTouchThreshold*) and the finger-departure Z threshold. Range is 0 to 255.

9.4.27. F11_2D_Ctrl31: small z threshold

This control register is present only if *HasZTuning* (F11_2D Query11, bit 0) reports as '1'.

Small Z Threshold (F11_2D_Ctrl31, bits 7:0)

Specifies the raw-Z threshold below which Z is scaled using only a scale factor, but no offset. Range is 0 to 255.

9.4.28. 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'.

Small Z Scale Factor LSB (F11_2D_Ctrl32.0, bits 7:0)

Small Z Scale Factor MSB (F11_2D_Ctrl32.1, bits 7:0)

Specifies the scale factor used when raw Z is below the Small Z Threshold. This is an unsigned fixed-point number in Q1.15 format (range is 0-1.99997).

9.4.29. 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'.

Large Z Scale Factor LSB (F11_2D_Ctrl33.0, bits 7:0)

Large Z Scale Factor MSB (F11_2D_Ctrl33.1, bits 7:0)

Specifies the scale factor used when raw Z is above the Small Z Threshold. This is an unsigned fixed-point number in Q1.15 format (range is 0-1.99997).

9.4.30. F11_2D_Ctrl34: algorithm selection

This control register is present only if *HasAlgorithmSelection* (F11_2D Query11, bit 1) reports as '1'.

Interpolation (F11_2D_Ctrl34, bits 2:0)

Specifies the position-interpolation algorithm to be used:

- 000 – Default (Hat Fit)
- 001 – Algorithm 1 (Gaussian)
- 010 – Algorithm 2 (Centroid)
- 011 – Algorithm 3 (Truncated Centroid)
- 100-111 – Reserved

Post Correction (F11_2D_Ctrl34, bits 5:3)

Specifies the post-correction algorithm to be used:

- 000 – None
- 001 – Default (Hat Fit)
- 010-111 – Reserved

Reserved (F11_2D_Ctrl34, bits 7:6)

These bits are reserved.

9.4.31. 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'.

Pen Z Upper Threshold (F11_2D_Ctrl35, bits 7:0)

When F11_2D_Query9, bit 4 is set to 1, Ctrl35 is the Pen Z Upper Threshold.

9.4.32. F11_2D_Ctrl36: Wx scale factor

This control register is present only if *HasWTuning* (F11_2D Query11, bit 2) reports as '1'.

Wx Scale Factor (F11_2D_Ctrl36, bits 7:0)

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.

9.4.33. F11_2D_Ctrl37: Wx offset

This control register is present only if *HasWTuning* (F11_2D Query11, bit 2) reports as '1'.

Wx Offset (F11_2D_Ctrl37, bits 7:0)

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.

9.4.34. F11_2D_Ctrl38: Wy scale factor

This control register is present only if *HasWTuning* (F11_2D Query11, bit 2) reports as '1'.

Wy Scale Factor (F11_2D_Ctrl38, bits 7:0)

Specifies the scale factor used to calculate *Wy*.

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.

9.4.35. F11_2D_Ctrl39: Wy offset

This control register is present only if *HasWTuning* (F11_2D Query11, bit 2) reports as ‘1’.

Wy Offset (F11_2D_Ctrl39, bits 7:0)

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.

9.4.36. F11_2D_Ctrl40.0/40.1: x pitch

These registers are present only if *HasPitchInfo* (F11_2D Query11, bit 3) reports as ‘1’.

X Pitch LSB (F11_2D_Ctrl40.0, bits 7:0)

X Pitch MSB (F11_2D_Ctrl40.1, bits 7:0)

Specifies the pitch of the X electrodes on the sensor.

This register pair contains the pitch in millimeters for the X-axis in Q4.12 fixed-point format.

9.4.37. F11_2D_Ctrl41.0/41.1: y pitch

These control registers are present only if *HasPitchInfo* (F11_2D Query11, bit 3) reports as ‘1’.

Y Pitch LSB (F11_2D_Ctrl41.0, bits 7:0)

Y Pitch MSB (F11_2D_Ctrl41.1, bits 7:0)

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.

9.4.38. 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’.

Finger Size X LSB (F11_2D_Ctrl42.0, bits 7:0)

Finger Size X MSB (F11_2D_Ctrl42.1, bits 7:0)

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).

9.4.39. 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'.

Finger Size Y LSB (F11_2D_Ctrl43.0, bits 7:0)

Finger Size Y MSB (F11_2D_Ctrl43.1, bits 7:0)

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).

9.4.40. F11_2D_Ctrl44: report measured size

Report Measured Size (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.

9.4.41. F11_2D_Ctrl45: segmentation aggressiveness

This control register is present only if *HasSegmentationAggressiveness* (F11_2D Query11, bit 5) reports as ‘1’.

Segmentation Aggressiveness (F11_2D_Ctrl45, bits 7:0)

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.

9.4.42. F11_2D_Ctrl46: X clip left

This control register is present only if *HasXYClip* (F11_2D Query11, bit 6) reports as ‘1’.

X Clip Left (F11_2D_Ctrl46, bits 7:0)

Specifies the width of the inactive border on the left edge of the sensor, in units of 1/32 mm.

9.4.43. F11_2D_Ctrl47: X clip right

This control register is present only if *HasXYClip* (F11_2D Query11, bit 6) reports as ‘1’.

X Clip Right (F11_2D_Ctrl47, bits 7:0)

Specifies the width of the inactive border on the right edge of the sensor, in units of 1/32 mm.

9.4.44. F11_2D_Ctrl48: Y clip upper

This control register is present only if *HasXYClip* (F11_2D Query11, bit 6) reports as '1'.

Y Clip Upper (F11_2D_Ctrl48, bits 7:0)

Specifies the width of the inactive border on the upper edge of the sensor, in units of 1/32 mm.

9.4.45. F11_2D_Ctrl49: Y clip lower

This control register is present only if *HasXYClip* (F11_2D Query11, bit 6) reports as '1'.

Y Clip Lower (F11_2D_Ctrl49, bits 7:0)

Specifies the width of the inactive border on the upper edge of the sensor, in units of 1/32 mm.

9.4.46. F11_2D_Ctrl50: minimum drumming separation

This control register is present only if *HasFastFlick* (F11_2D Query11, bit 7) reports as '1'.

Minimum Drumming Separation(F11_2D_Ctrl50, bits 7:0)

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.

9.4.47. F11_2D_Ctrl51: maximum drumming movement

This control register is present only if *HasFastFlick* (F11_2D Query11, bit 7) reports as '1'.

Maximum Drumming Movement (F11_2D_Ctrl50, bits 7:0)

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.

[Add a figure or two to illustrate the flick/drumming problem and solution]

Figure 62. Flick versus drumming gestures

Deleted: 63

9.4.48. F11_2D_Ctrl52: bending detection threshold

This control register is present only if *HasAntiBending* (F11_2D_Query5, bit5) reports as '1'.

BendingDetectionThreshold (F11_2D_Ctrl52, bits 7:0)

Sets the threshold for activation of Bending Correction. When set to '0', Bending Correction is always active; at 255 Bending Correction is effectively disabled.

9.4.49. F11_2D_Ctrl53: bending touch threshold adjustment

This control register is present only if *HasAntiBending* (F11_2D_Query5, bit5) reports as '1'.

BendingTouchThresholdAdjustment (F11_2D_Ctrl53, bits 7:0)

While Bending Correction is active, the touch threshold is increased by the value in this register. This is usually set to 0.

9.4.50. F11_2D_Ctrl54: flexibility

This control register is present only if *HasAntiBending* (F11_2D_Query5, bit5) reports as '1'.

Flexibility (F11_2D_Ctrl54, bits 7:0)

Specifies the flexibility of the sensor. 0 represents a rigid, unbendable sensor; 255 represents the most flexible sensor.

Note: This RMI register accepts decimal values from 0 to 255.

9.4.51. F11_2D_Ctrl55: bend distance / bend interaction distance

This control register is present only if *HasAntiBending* (F11_2D_Query5, bit5) reports as '1'.

InteractionDistance (F11_2D_Ctrl55, bits 3:0)

Distance from the peak of the bend within which additional fingers affect the shape of the bend.

BendDistance (F11_2D_Ctrl55, bits 7:4)

Distance from the peak of the bend to the edge of the bend.

Note: These two fields are usually set to equal values.

9.4.52. F11_2D_Ctrl56: bending correction X

This control register is present only if *HasAntiBending* (F11_2D_Query5, bit5) reports as '1'.

BendingCorrectionX (F11_2D_Ctrl56, bits 7:0)

Specifies the magnitude of the Bending Correction in the X direction. Smaller values reduce the amount of correction and larger values increase it.

BendingRxScale (F11_2D_Ctrl56, bits 7:0)

Specifies the magnitude of the Bending Correction in the X direction. Smaller values reduce the amount of correction and larger values increase it.

9.4.53. F11_2D_Ctrl57: bending correction Y

This control register is present only if *HasAntiBending* (F11_2D_Query5, bit5) reports as '1'.

BendingCorrectionY (F11_2D_Ctrl57, bits 7:0)

Specifies the magnitude of the Bending Correction in the Y direction. Smaller values reduce the amount of correction and larger values increase it.

9.4.54. F11_2D_Ctrl58: pragma color

This control register is present only if *HasAntiBending* (F11_2D_Query5, bit5) reports as '1'.

PragmaColor (F11_2D_Ctrl58, bits 7:0)

When the value is zero, Design Studio 4 uses default inking colors. For values other than zero, Design Studio 4 selects an inking color according to the value of *PragmaColor*. *PragmaColor* may be used for a wide range of diagnostic purposes. For example, in switching between ABA and NABA modes (anti- bending) the change can be readily seen in the inking graph.

This is a read-only register. Its value can be set through the configuration file.

9.5. 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.

9.5.1. 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 9.3.3):

   ```
   F11_2D_Data0_registerCount = ceil(decodedNumberOfFingers/4)
   ```

2. If the *HasAbs* 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 9.3.3).
3. If the *HasRel* 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 9.3.3).
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_registerCount = ceil((NumberOfTouchShapes+1)/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 9.3.3).

A two-finger example with individual Scroll Zones and 12 TouchShapes, and contact geometry is shown in

F11_2D_Data28 — Bending

Figure 63

Deleted: ¶
¶
Figure 63

Name	7	6	5	4	3	2	1	0
F11_2D_Data0.0	—		—		FingerState1		FingerState0	
F11_2D_Data1.0	X position (11:4)							
F11_2D_Data2.0	Y position (11:4)							
F11_2D_Data3.0	Y position (3:0)				X position (3:0)			
F11_2D_Data4.0	Wy (3:0)				Wx (3:0)			
F11_2D_Data5.0	Z (7:0)							
F11_2D_Data1.1	X position (11:4)							
F11_2D_Data2.1	Y position (11:4)							
F11_2D_Data3.1	Y position (3:0)				X position (3:0)			
F11_2D_Data4.1	Wy (3:0)				Wx (3:0)			
F11_2D_Data5.1	Z (7:0)							
F11_2D_Data6.0	X Delta							
F11_2D_Data7.0	Y Delta							
F11_2D_Data6.1	X Delta							
F11_2D_Data7.1	Y Delta							
F11_2D_Data8	Chiral Scroll	Pinch	Press	Flick	EarlyTap	DoubleTap	TapAndHold	SingleTap
F11_2D_Data9	Gesture Finger Count			MultiFinger Scroll—	ScrollZone	Shape	Rotate	PalmDetect
F11_2D_Data10	Pinch Motion / X Flick Distance							
F11_2D_Data11	Rotate Motion / Y Flick Distance							
F11_2D_Data12	Finger Separation / Flick Time							
F11_2D_Data13.0	Shape7	Shape6	Shape5	Shape4	Shape3	Shape2	Shape1	Shape0
F11_2D_Data13.1	—	—	—	—	Shape11	Shape10	Shape9	Shape8
F11_2D_Data14	X Lower Scroll Motion / Horizontal MultiFinger Scroll							
F11_2D_Data15	Y Right Scroll Motion / Vertical MultiFinger Scroll							
F11_2D_Data16	X Upper Scroll Motion							
F11_2D_Data17	Y Left Scroll Motion							
F11_2D_Data18.0	Contact Geometry Target X (11:4)							
F11_2D_Data19.0	Contact Geometry Target Y (11:4)							
F11_2D_Data20.0	Contact Geometry Target Y (3:0); Contact Geometry Target X (3:0))							

This figure is continued on the following page.

Name	7	6	5	4	3	2	1	0
F11_2D_Data21.0	Contact Geometry Width (11:4)							
F11_2D_Data22.0	Contact Geometry Height (11:4)							
F11_2D_Data23.0	Contact Geometry Height (3:0); Contact Geometry Width (3:0))							
F11_2D_Data24.0	Contact Geometry Angle							
F11_2D_Data25.0	Contact Geometry Center X (11:4)							
F11_2D_Data26.0	Contact Geometry Center Y (11:4)							
F11_2D_Data27.0	Contact Geometry Center Y (3:0); Contact Geometry Center X (3:0)							
F11_2D_Data18.1	Contact Geometry Target X (11:4)							
F11_2D_Data19.1	Contact Geometry Target Y (11:4)							
F11_2D_Data20.1	Contact Geometry Target Y (3:0); Contact Geometry Target X (3:0))							
F11_2D_Data21.1	Contact Geometry Height (11:4)							
F11_2D_Data22.1	Contact Geometry Width (11:4)							
F11_2D_Data23.1	Contact Geometry Height (3:0); Contact Geometry Width (3:0))							
F11_2D_Data24.1	Contact Geometry Angle							
F11_2D_Data25.1	Contact Geometry Center X (11:4)							
F11_2D_Data26.1	Contact Geometry Center Y (11:4)							
F11_2D_Data27.1	Contact Geometry Center Y (3:0); Contact Geometry Center X (3:0)							
F11_2D_Data28								Bending

Figure 63. Function $11 data register example (two fingers, scroll zones, 12 TouchShapes, and contact geometry

Deleted: 64

9.5.2. 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.

***Important*:** Finger0 is the first set of Data1 through Data5 registers that a user encounters (lowest addresses in the address space for the finger registers). After that first finger is reported, though, the numbering is based not upon which finger touched when, but instead is based upon whatever set of register blocks is available.

At first glance, a finger number would indicate the order that the fingers arrived, but that numbering changes as soon as a finger leaves. Consider that three fingers arrive: 0, then 1, then 2. The finger number indicates the arrival order, which seems like it might be useful to a host. Now assume that finger 1 lifts, but 0 and 2 are still present. If another finger lands, should it be placed in the finger1 location or the finger3 location, or should 2 move to 1, and the new finger shows up in 2? If you put it in finger1, then the finger number has nothing to do with arrival order. To preserve the finger order, we could move finger2 to finger1 when finger1 lifts, but unless the host is paying strict attention, it might interpret finger1 lifting and finger2 shifting to finger1 as finger2 lifting and finger1 jumping to a new position. For this reason, finger numbering and data blocks are assigned first come, first served in RMI. If a host really cares, it should simply read all fingers present and implement its own tracking mechanism to identify the order that the fingers arrived.

9.5.3. F11_2D_Data0.*: finger status data

The size of this register block can be calculated from *NumberOfFingers* field in F11_2D_Query1 (see section 9.3.3). 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 pen or proximity capability (either F11_2D_Query9 *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.

9.5.4. F11_2D_Data1 and F11_2D_Data 2: X and Y position data (MSB)

These data registers report the most-significant bits of the absolute X and 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, F11_2D_Data3 should also be read.

9.5.5. 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 9.3.6).

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, bits3:0)

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, bits7:4)

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.

9.5.6. 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 9.3.6).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.

9.5.7. 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 9.3.6). This register reports a fixed value when a stylus is detected.

Z (F11_2D_Data5, bits 7:0)

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.

If the device has neither stylus nor proximity capability (F11_2D_Query9 does not exist, or both *HasPen* and *HasProximity* are set to '0'), and the *FingerState* field for that finger corresponds to '10', then the Z values for all the fingers on the pad will be the same.

If the device has proximity capability (*HasProximity* is set to '1'), then the Z values reported in this register when a proximate finger is present are on a different scale from the Z values reported when a finger is touching, so proximate-finger Z and touching-finger Z cannot meaningfully be compared.

If the device has stylus capability (*HasPen* is set to '1'), then a Z value of 30 ($1E) is reported in this register when a stylus is present.

9.5.8. F11_2D_Data6.* and F11_2D_Data7.*: finger motion deltas

These registers are only present if the *HasRel* field of F11_2D_Query1 reports as '1' (see section 9.3.3). 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, bits 7:0)

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, bits 7:0)

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).

The device may implement the relative motion accumulators as 'sticky' 8-bit signed integers, which hold at +127 or –128 once reaching either of those values until they are cleared by reading. Or, the accumulators may be implemented as larger accumulators whose values are clipped to +127 or –128 when expressed in the data registers; in this case, they will not appear 'sticky' if an overflowing motion is countered by a reverse motion before the host reads the data registers.

9.5.9. 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 9.3.3).

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 9.4.7), 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 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.

Chiral Scroll (F11_2D_Data8, bit 7)

If this bit is set, then a chiral scroll gesture is either in progress or has completed.
This bit is cleared when it is read, but it will be set again if the chiral scroll gesture continues.

The horizontal or vertical scroll amount is reported in registers F11_2D_Data14 and F11_2D_Data15 respectively.

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 that 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 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 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 in the reported gesture.

Table 5. Number of fingers used in the reported gesture

Deleted: 56

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).

9.5.10. 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.

9.5.11. 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.

9.5.12. 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.

9.5.13. 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.

9.5.14. F11_2D_Data14: x lower scroll motion / MultiFinger horizontal scroll

This register is only present if the *HasScrollZones* field or the *HasMultiFingerScroll* field of F11_2D_Query8 reports as '1'. When the *ScrollZone* bit of F11_2D_ Data9 is set to '1', this register reports the distance of the ScrollZone motion in the 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. When the *MultiFingerScroll* bit of F11_2D_Data9 is set to '1', this register reports the distance of the MultiFinger Scroll motion in the X (horizontal) direction.

Notes:

- This register is cleared to ‘0’ when read.
- If the *IndividualScrollZones* field of F11_2D_Query8 reports as ‘0’, this register reports the combined motion on the “X Lower” and “X Upper” ScrollZones. If the *IndividualScrollZones* field of F11_2D_Query8 reports as ‘1’, this register reports only the motion on the “X Lower” ScrollZone.

9.5.15. F11_2D_Data15: y right scroll motion / MultiFinger vertical scroll

This register is only present if the *HasScrollZones* field or the *HasMultiFingerScroll* field of F11_2D_Query8 reports as ‘1’. When the *ScrollZone* bit of F11_2D_ Data9 is set to 1, this register reports the distance of the ScrollZone motion in the 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 *MultiFingerScroll* bit of F11_2D_Data9 is set to ‘1’, this register reports the distance of the MultiFinger Scroll motion in the Y (vertical) direction.

Notes:

- This register is cleared to ‘0’ when read.
- If the *IndividualScrollZones* field of F11_2D_Query8 reports as ‘0’, this register reports the combined motion on the “Y Right” and “Y Left” ScrollZones. If the *IndividualScrollZones* field of F11_2D_Query8 reports as ‘1’, this register reports only the motion on the “Y Right” ScrollZone.

9.5.16. 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.

9.5.17. 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.

9.5.18. F11_2D_Data18 and F11_2D_Data19: contact geometry X and Y target positions (MSB)

These registers are only present if the *HasContactGeometry* field of F11_2D_Query9 reports as ‘1’.

These data registers report bits 11:4 of the 12-bit 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.

9.5.19. 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.

9.5.20. F11_2D_Data21 and F11_2D_Data22: contact geometry height and width (MSB)

These registers are only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

These data registers report bits 11:4 of the 12-bit contact geometry height and width. If no finger is present on the sensor, the last known valid height and width are reported.

9.5.21. F11_2D_Data23: contact geometry height and width (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 height and width. If no finger is present on the sensor, the last known valid height and width are reported.

The fields in this register are defined as follows:

- WidthX3:0 (F11_2D_Data23, bits3:0)
  Bits 3:0 of the horizontal contact geometry width of the finger on the sensor.
- WidthY3:0 (F11_2D_Data23, bits7:4)
  Bits 3:0 of the vertical contact geometry height of the finger on the sensor.

9.5.22. 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.

9.5.23. F11_2D_Data25 and F11_2D_Data26: contact geometry X and Y center positions (MSB)

These registers are only present if the *HasContactGeometry* field of F11_2D_Query9 reports as '1'.

These data registers report bits 11:4 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.

9.5.24. 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, bits3:0)

Bits 3:0 of the absolute X (horizontal) contact geometry center position of the finger on the sensor.

CenterY3:0 (F11_2D_Data27, bits7:4)

Bits 3:0 of the absolute Y (horizontal) contact geometry center position of the finger on the sensor.

9.5.25. F11_2D_Data28: sensor status

Bending (F11_2D_Data28, bit 0)

When this bit reports as '1', physical bending of the sensor has been detected and is being corrected.

Reserved (F11_2D_Data28, bits 7:1)

Reserved.

9.6. 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 9.4.1). 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.

9.7. Function $11: command registers

Function $11 has the following command register defined:

Name	7	6	5	4	3	2	1	0
F11_2D_Cmd0	—	—	—	—	—	—	—	ReZero

Figure 64, Function $11 command register

Deleted: 65

ReZero (F11_2D_Cmd0, bit0)

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.

10. Function $19: 0-D capacitive button sensors

Deleted: Capacitive
Deleted: Button
Deleted: Sensors

Function $19 implements a 0-D sensing function, typically known as a capacitive button.

10.1. Function $19: query registers

Query registers 0 and 1 contains general query information regarding Function $19, button sensing.

Name	7	6	5	4	3	2	1	0
F19_Btn_Query0	—	—	—	—	—	Has Hysteresis Threshold	Has Sensitivity Adjust	Configurable
F19_Btn_Query1	—	—	—			ButtonCount		

Figure 65. Function $19 query registers

Deleted: 66

10.1.1. F19_Btn_Query0: configurable button query

Deleted: Configurable

The bits in register F19_Btn_Query0 are defined as follows:

Configurable (F19_Btn_Query0, bit 0)

If this field is ‘0’, the button sensitivity control registers and sensor mapping control registers are not configurable at run-time:

- The *ButtonCount* field represents the exact number of buttons that exist in the product.

Note: When Configurable = ‘0’ (as in module-style builds), this count is the sum of real, virtual, and dummy buttons. This allows a tool to calculate the extent of the various replicated register blocks in the F19 register map. There is no run-time method to distinguish dummy buttons from virtual buttons. In practice, the only difference is that dummy buttons will never report, while virtual buttons will, if properly stimulated.

For consistency, unused and virtual buttons will report a sensitivity of zero in their corresponding sensitivity register, indicating that they are not participating in the analog processing code.

- The sensor map control registers are read-only.
- The button sensitivity 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)].

  Configurable builds do not permit ‘holes’ in the button numbering scheme.
- The sensor map control registers are read-write.
- The button sensitivity control registers are read-write.
- Button processing for a given sensor is enabled by assigning a non-zero sensitivity value to the corresponding button sensitivity register. Unused buttons must be assigned a sensitivity of zero.

Configurable builds need this ability so that unused buttons can be excluded from participating in analog data processing. For example, unused button sensors must be excluded from the servoing operation.

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.

10.1.2. F19_Btn_Query1: button count query

Deleted: ButtonCount

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.

10.2. Function $19: control registers

These registers control the operation of the capacitive buttons.

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	SingleBtn0
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_Ctrl4.0	*Reserved* Sensitivity_Btn0							
F19_Btn_Ctrl4.1	*Reserved* Sensitivity_Btn1							
F19_Btn_Ctrl4.2	*Reserved* Sensitivity_Btn2							
F19_Btn_Ctrl4.3	*Reserved* Sensitivity_Btn3							
F19_Btn_Ctrl5	—	—	—	Sensitivity Adjustment				
F19_Btn_Ctrl6	—	—	—	—	Hysteresis Threshold			

Figure 66. Function $19 control registers for an example of ButtonCount=4

Deleted: 67

The example 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.

10.2.1. 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* sensitivity control registers (F19_Btn_Ctrl6 through F19_Btn_Ctrl10 in the example above) is always identical to *ButtonCount*:

```
F19_SensitivityRegisterCount = ButtonCount
```

10.2.2. 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.

10.2.3. 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 10.2.1 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 10.4) 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.

10.2.4. 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.

10.2.5. 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 10.2.1 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 starting at 0, and continuing up 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 button0, the next sensor map control register applies to button1, and so on, up to the final SensorMap 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 with 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 t that does not exist.

If the Configurable field is '1', it indicates that the host can arbitrarily assign any capacitive sensor to a button by writing the sensor number into this field. See the touch controller documentation to find out the mapping of sensor numbers to touch controller pins.

If the *Configurable* field is '0' on a particular product, it means that the sensor mapping to touch controller pins is fixed by the module design, and cannot be changed. As a result, these mapping registers are read-only. In that case, the registers will still report what sensor is assigned to each capacitive button on that product, but the host will be unable to make changes to those mappings.

Regardless of the state of the *Configurable* field, the default values for these sensor mapping registers are always product-specific; consult the product's documentation for more details.

The example register map in Figure 67 assumes a product that has 4 buttons and 28 touch control sensors, S0 through S27.

Name	7	6	5	4	3	2	1	0
F19_Btn_Ctrl0	—	—	—	—	—	—	ButtonUsage='10'	
F19_Btn_Ctrl1.*	—	—	—	—	IntEnBtn3=0	IntEnBtn2=1	IntEnBtn1=1	IntEnBtn0=1
F19_Btn_Ctrl2.*	—	—	—	—	SingleBtn3=0	SingleBtn2=0	SingleBtn1=1	SingleBtn0=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						

Figure 67. Function $19 sensor map control registers for an example of ButtonCount=4

In Figure 67, the first sensor map control register at F19_Btn_Ctrl3 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.

Deleted: Figure 68Figure 67

Deleted: ¶

Deleted: 68

Deleted: Figure 68Figure 67

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.

10.2.6. F19_Btn_Ctrl4.*: reserved button sensitivity control

These registers are reserved for Synaptics use. Do not alter the contents of these registers when updating the device flash configuration.

Each button has a corresponding register that can be used to adjust its sensitivity. Although both registers control sensitivity, the difference between F19_Btn_Ctrl4 and F19_Btn_Ctrl5 is that F19_Btn_Ctrl4 controls each button separately and is RAM-based, while F19_Btn_Ctrl5 is an overall setting for all buttons and is Flash-based.

Note: The format of the data in this register is TBD.

10.2.7. 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.

10.2.8. 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.

10.3. 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.

10.3.1. 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 Figure 68, with the example of a *ButtonCount* of 12.

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

Figure 68. Function $19 capacitive button data registers (for example of 12 buttons)

Deleted: *69*

10.4. 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.

10.5. Function $19: command registers

Function $19 implements a single command register.

Name	7	6	5	4	3	2	1	0
F19_RMI_Cmd0	—	—	—	—	—	—	—	Rezero

Figure 69, Function $19 command register

Deleted: 70

Rezero Command (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.

Deleted: Private

11. Function $25: Slurper (private)

Function $25 implements an interface over which an entire frame image can be read. It is intended for products like the large touch screen (LTS), in which image data from multiple touch controllers is aggregated by a central high-powered microcontroller.

Onebase implementations that include the function also include a way to make one touch controller's CPU clock drive all the others and to synchronize the sensing waveforms of all the touch controllers.

Although in general RMI attempts to maintain independence from the physical interface (SPI, I²C, SMBUS, and so on), Function $25 places bandwidth requirements on the interface that may require special support. For example, some F$25 implementations may require a 16-bit word length. Because of this, the F$25 register set is laid out in a manner that is compatible with 16-bit word lengths.

Deleted: Signals

11.1. Function $25: signals

In addition to the set of signals provided by RMI for host communication, Function $25 requires three additional signals to coordinate the data acquisition process of multiple touch controllers and to communicate that data to the host processor.

HSYNC This signal is used by all the T1320 touch controllers to monitor their synchronization state. If a touch controller determines that it has lost sync with the others, it indicates this in the data packet it sends to the host.

VSYNC This signal is sourced by the host CPU. When this signal is asserted, the touch controllers halt their data acquisition and wait until VSYNC signal deasserts.

DATA_READY This wire-ANDed signal is sourced by the touch controllers to indicate that there is data available to be read by the host CPU. When this signal is asserted, all of the touch controllers have data. A host can use this signal to read image data as it is acquired, with minimal latency. This signal asserts on reset.

11.2. Function $25: query registers

The query registers describe the function capabilities available in this product.

Name	7	6	5	4	3	2	1	0
F25_SLURP_Query0	—	—	—	—	—	—	—	—
F25_SLURP_Query1	—	—	—	—	—	—	—	—

Figure 70. Function $25 query registers

Deleted: 71

The bits of these registers are defined as follows:

Reserved (F25_SLURP_ Query0, bits 7:0)

Reserved (F25_SLURP_ Query1, bits 7:0)

Reserved for future query bit expansion.

11.3. Function $25: control registers

There are no control registers for Function $25; for efficiency, all configuration is performed through writable data registers.

11.4. Function $25: data registers

Registers labeled “packet register” contain multiple bytes of data. Access semantics are described in section 2.5.3.

Name	7	6	5	4	3	2	1	0
F25_SLURP_Data0	—	—	ClockConfig			—	OSC_Nuke	—	—
F25_SLURP_Data1	—	—	—	—	—	—	—	—
F25_SLURP_Data2	Frame Block Table 0 (Packet register)							
F25_SLURP_Data3	Frame Block Table 1 (Packet register)							
F25_SLURP_Data4	Frame Block Table 2 (Short table, Packet register)							
F25_SLURP_Data5	Frame Block Table 3 (Short table, Packet register)							
F25_SLURP_Data6	Analog Register Table 0 (Packet register)							
F25_SLURP_Data7	Analog Register Table 1 (Packet register)							
F25_SLURP_Data8	User-Word Address Low							
F25_SLURP_Data9	User-Word Address High							
F25_SLURP _Data10	Image (Packet register)							
F25_SLURP_Data11	Stalled	—	—	CRT Block		AnalogBlock	FrameBlock	
F25_SLURP_Data12	CRTC Block Table 0							
F25_SLURP_Data13	CRTC Block Table 1							

Figure 71: Function $25 writable data registers

Deleted: 72

The bits of these registers are defined as follows:

OSC_Nuke (F25_SLURP_Data0, bit 2)

When set to ‘1’, this bit disables the OSC modulation that is normally performed on a per-cluster basis.

ClockConfig (F25_SLURP_Data0, bits 5:4)

Controls clock input/output:

00 = Disable clock output and clock input.
01 = Enable clock output on A14.
10 = Enable clock input on A14.
11 = Disable clock output and clock input.

Reserved (F25_SLURP_Data1)

This register is reserved.

Frame Block Table 0 (F25_SLURP_Data2)

This register accepts up to a 258-byte-long description of the transmitter configuration for each cluster in the frame. Unexpected operation may result if this register is written to while Frame Block Table 0 is active. This register is a non-incrementing register that requires the entire Frame Block Table to be read or written in a single burst transaction.

Frame Block Table 1 (F25_SLURP_Data3)

This register accepts up to a 258-byte-long description of the transmitter configuration for each cluster in the frame. Unexpected operation may result if this register is written to while Frame Block Table 1 is active. This register is a non-incrementing register that requires the entire Frame Block Table to be read or written in a single burst transaction.

Frame Block Table 2 (F25_SLURP_Data4)

This configuration is shorter in length than the first two Frame Blocks and supports configurations of up to only 4 clusters. This register is a non-incrementing register that requires the entire Frame Block Table to be read or written in a single burst transaction. Unexpected operation may result if this register is written to while Frame Block Table 2 is active.

Frame Block Table 3 (F25_SLURP_Data5)

This configuration is shorter in length than the first two Frame Blocks and supports configurations of up to only 4 clusters. This register is a non-incrementing register that requires the entire Frame Block Table to be read or written in a burst transaction. Unexpected operation may result if this register is written to while Frame Block Table 3 is active.

Analog Register Table 0 (F25_SLURP_Data6)

This register accepts up to a 26-byte-long description of the analog register configuration. Unexpected operation may result if this register is written to while Analog Register Table 0 is active. This register is a non-incrementing register that requires the entire Frame Block Table to be read or written in a single burst transaction.

Analog Register Table 1 (F25_SLURP_Data7)

This register accepts up to a 26-byte-long description of the analog register configuration. Unexpected operation may result if this register is written to while Analog Register Table 1 is active. This register is a non-incrementing register that requires the entire Frame Block Table to be read or written in a single burst transaction.

User-Word Address Low (F25_SLURP_Data8)

User-Word Address High (F25_SLURP_Data9)

User-Word address. Holds the RAM address whose contents are returned in the image data's "User Word" fields.

Note: The two *User-Word Address* registers (*F25_SLURP_Data8* and *F25_SLURP_Data9*) are a coherent group: To change either register, both registers must be written (see section 2.6).

Formatted: Font color: Green

Deleted: 2.6

Image (F25_SLURP_Data10)

A circular buffer which holds the data to be read by the host. Data can be read continuously from this buffer at high speed (8 MHz), with no inter-byte delays required.

FrameBlock (F25_SLURP_Data11, bits 1:0)

This read-only bit field indicates which frame block is currently in use. This field cannot be used to change the Frame Block number; to change the Frame Block number, use the F25_SLURP_Cmd0 register.

AnalogBlock (F25_SLURP_Data11, bit 2)

This read-only bit field indicates which analog block is currently in use. This field cannot be used to change the Analog Block number; to change the Analog Block number, use the F25_SLURP_Cmd0 register.

CRTC Block (F25_SLURP_Data11, bits 4:3)

This read-only bit field indicates which CRTC block is currently in use. This field cannot be used to change the CRTC Block number; to change the CRTC Block number, use the F25_SLURP_Cmd0 register.

Stalled (F25_SLURP_Data11, bit 7)

This read-only bit field indicates that the touch controller is waiting for VSYNC to deassert. When this bit reports as '1', the data acquisition process is stalled and it is safe to read or write any parameters – analog register blocks, frame tables, and/or CRTC blocks.

CRTC Block Table 0 (F25_SLURP_Data12, bits 7:0)

This register accepts up to a 64-byte-long description of the CRTC configuration for each cluster in the frame. Unexpected operation may result if this register is written to while CRTC Block Table 0 is active. This register is a non-incrementing register that requires the entire CRTC Block Table to be read or written in a single burst transaction.

CRTC Block Table 1 (F25_SLURP_Data13, bits 7:0)

This register accepts up to a 64-byte-long description of the CRTC configuration for each cluster in the frame. Unexpected operation may result if this register is written to while CRTC Block Table 1 is active. This register is a non-incrementing register that requires the entire CRTC Block Table to be read or written in a single burst transaction.

CRTC Block (F25_SLURP_Data11, bits 4:3)

This read-only bit field indicates which CRTC block is currently in use. This field cannot be used to change the CRTC Block number; to change the CRTC Block number, use the F25_SLURP_Cmd0 register.

11.4.1. Image data format

Deleted: Data

Deleted: Format

The format for the image data read from the Image register (F25_SLURP_Data10) is a sequence of packets, one per cluster, with the following format:

Offset	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Byte 0	Accumulated ADC0 LSB							
Byte 1	Accumulated ADC0 MSB							
Byte 2	Accumulated ADC1 LSB							
Byte 3	Accumulated ADC1 MSB							
...	...							
Byte 38	Accumulated ADC19 LSB							
Byte 39	Accumulated ADC19 MSB							
Byte 40	Noise Metric LSB							
Byte 41	Noise Metric MSB							
Byte 42	Common Mode Noise Metric LSB							
Byte 43	Common Mode Noise Metric MSB							

This figure is continued on the following page.

Offset	7	6	5	4	3	2	1	0
Byte 44	User Word LSB							
Byte 45	User Word MSB							
Byte 46	Out Of Sync	Over Flow	Unconfigured	Cluster Number				
Byte 47	Checksum							

Figure 72: One clusters' data

Deleted: 73

The bits of this packet are defined as follows:

Accumulated ADC0-19 (Byte0-Byte39)
Accumulated, bleed-over-filtered values from ADCs 0-19.

Noise Metric (Byte40-Byte41)
Noise metric for this cluster.

Common Mode Noise Metric (Byte42-Byte43)
Common Mode Noise metric for this cluster.

User Word (Byte44-Byte45)
Contents of the RAM location pointed to by the *User Word Address* registers (F25_SLURP_Data8 and F25_SLURP_Data9).

Cluster Number (Byte46, bits 4:0)
This cluster's number (minus 1). First cluster accumulated at the start of a frame is number 0.

Unconfigured (Byte46, bit 5)
Indicates that the device is unconfigured. This is the same as the bit found in Function $01 and indicates that the device has been reset (for example, by a watchdog timeout or a static discharge) since the last time it was configured.

Out Of Sync (Byte46, bit 7)
Indicates an Out-Of-Sync condition when the synchronization check performed by the T1320 touch controller fails.

Overflow (Byte46, bit 6)
Indicates that there was not enough room in the buffer to store an entire cluster of data. This indication can only occur on the last packet of data in the buffer (there is no room to store more packets).

Unconfigured (Byte46, bit 5)
Indicates that the touch controller is unconfigured. This is the same as the bit found in Function $01 and indicates that the touch controller has been reset (by a watchdog timeout or a static discharge) since the last time it was configured.

Checksum (Byte47)
Checksum of the entire packet. This checksum is calculated by exclusive-or'ing bytes 0 – 46 and the value 0xFF.

11.4.2. Frame blocks 0 and 1

Each frame block describes the transmitter configuration for the clusters that compose a frame. Frame blocks 0 and 1 are full length frame blocks intended for full length scans of up to 32 clusters –258 bytes in length. Frame blocks 2 and 3 are intended for short scans of up to 4 clusters each – 34 bytes. These short frame blocks can be useful for proximity sensing or low-power wake checks.

Frame Block Table Offset	7	6	5	4	3	2	1	0
Byte 0	Clusters Per Frame					Bursts Per Cluster		
Byte 1	—	—	—	—	—	Chan Reset Every Frame	Chan Reset Every Cluster	Chan Reset Every Burst
Byte 2	Cluster 0: XMTR_PL0							
Byte 3	Cluster 0: XMTR_PL8							
Byte 4	Cluster 0: XMTR_PL16							
Byte 5	Cluster 0: OSC				Cluster 0: XMTR_PL24			
Byte 6	Cluster 0: XMTR_INP0							
Byte 7	Cluster 0: XMTR_INP8							
Byte 8	Cluster 0: XMTR_INP16							
Byte 9	—	—	—	—	Cluster 0: XMTR_INP24			
Byte 10	Cluster 1: XMTR_PL0							
Byte 11	Cluster 1: XMTR_PL8							
Byte 12	Cluster 1: XMTR_PL16							
Byte 13	Cluster 1: OSC				Cluster 1: XMTR_PL24			
Byte 14	Cluster 1: XMTR_INP0							
Byte 15	Cluster 1: XMTR_INP8							
Byte 16	Cluster 1: XMTR_INP16							
Byte 17	—	—	—	—	Cluster 1: XMTR_INP24			
...								
Byte 250	Cluster 31: XMTR_PL0							
...	...							
Byte 257	—	—	—	—	Cluster 31:XMTR_INP24			

Deleted: ¶ ... [1]

Figure 73. Frame blocks

Deleted: 74

The bits of these registers are defined as follows:

Bursts Per Cluster (Byte0, bits 2:0)
Number of bursts (minus 1) per cluster.

Clusters Per Frame (Byte0, bits 7:3)
Number of clusters (minus 1) per frame. The length of the Frame Block is specified by this field. A Frame Block need not include the full 32 clusters (numbered 0-31) defined.

For example, if a frame block defines this field to be 10 clusters long (this field is 9) then the length of the Frame Block will be (8 bytes per cluster * 10 clusters) + 2 bytes for the Frame Block header = 82 bytes.
Frame Blocks 0 and 1 support 32 clusters (maximum length of 258 bytes) but Frame Blocks 2 and 3 support only 4 clusters (maximum length of 34 bytes).

Chan Reset Every Frame, Cluster, or Burst (Byte1, bits 0-2)
Filter chain reset configuration: Channel Reset is set at the beginning of every Burst, Cluster, or Frame.

XMTR_PLxx and XMTR_INPxx (Byte2-Byte257)
Values to write to the XMTR_PLxx and XMTR_INPxx registers for each cluster packeted into two 32-bit words.

OSC (Byte5-Byte257)
Values to write to bits 5-2 of the OSC_CTL register for each cluster. This field allows a small range of frequency modulation per cluster.

11.4.3. CRTC blocks

To build an automatic regression test environment, we would like to have the ability of introducing a fake finger on sensor so that certain F$87 and F$11 tests could be conducted without human/robot involvement.

The idea is to add one or more CRTC block tables to F$25 so that corresponding CRTC_PL, CRTC_INP, and CRTC_SEL are set while driving the transmitter table. This will add/minus 1pF or remain intact to a specific Rx per cluster.

CRTC Block Table Offset	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Byte 0	—	Cluster 0 CRTC_INP	Cluster 0 CRTC_PL		Cluster 0 CRTC_SEL			
Byte 1	—	—	—		—	—	—	—
Byte 2	—	Cluster 1 CRTC_INP	Cluster 1 CRTC_PL		Cluster 1 CRTC_SEL			
Byte 3	—	—	—		—	—	—	—
…	…							
Byte 62	—	Cluster 31 CRTC_INP	Cluster 31 CRTC_PL		Cluster 31 CRTC_SEL			
Byte 63	—	—	—		—	—	—	—

Figure 74. CRTC blocks

Deleted: 75

The bits of these registers are defined as follows:

Cluster n CRTC_SEL (Byte0, bits 4:0)

Cluster n CRTC_PL (Byte0, bit 5)

Cluster n CRTC_INP(Byte0, bit 6)

If *CrtcBlkSelect* is set to ‘01’ or ‘10’, the values in the current cluster’s entry in the appropriate CRTC Block table are copied to the touch controller’s CRTC_INP, CRTC_PL, and CRTC_SEL registers.

11.4.4. Analog register tables 0 and 1

Deleted: Register

The analog register tables allow control of the analog parameters of the acquisition process. There are two analog tables to allow seamless changing of parameters on a frame boundary.

Frame Block Table Offset	7	6	5	4	3	2	1	0
Byte 0	MISC_CTRL_LSB							
Byte 1	—	—	—	—	MISC_CTRL_MSB			
Byte 2	RCVR_CTRL1_LSB							
Byte 3	—	—	—	—	RCVR_CTRL1_MSB			
Byte 4	RCVR_CTRL2_LSB							
Byte 5	—	—	—	—	RCVR_CTRL2_MSB			
Byte 6	RCVR_CTRL3_LSB							
Byte 7	—	—	—	—	RCVR_CTRL3_MSB			
Byte 8	RCVR_CTRL4_LSB							
Byte 9	—	—	—	—	RCVR_CTRL4_MSB			
Byte 10	REF_HI_CTRL_LSB							
Byte 11	—	—	—	—	REF_HI_CTRL_MSB			
Byte 12	REF_LO_CTRL_LSB							
Byte 13	—	—	—	—	REF_LO_CTRL_MSB			
Byte 14	RCVR_EN0_LSB							
Byte 15	—	—	—	—	RCVR_EN0_MSB			
Byte 16	RCVR_EN12_LSB							
Byte 17	—	—	—	—	RCVR_EN12_MSB			
Byte 18	RCVR_EN24_LSB							
Byte 19	—	—	—	—	RCVR_EN24_MSB			
Byte 20	XMTR_EN0_LSB							
Byte 21	—	—	—	—	XMTR_EN0_MSB			
Byte 22	XMTR_EN12_LSB							
Byte 23	—	—	—	—	XMTR_EN12_MSB			
Byte 24	XMTR_EN24_LSB							
Byte 25	—	—	—	—	XMTR_EN24_MSB			

Figure 75. Analog Register tables

Deleted: 76

These parameters control the static configuration of the analog hardware. These register values will be written to the hardware registers seamlessly on a frame boundary when the AnaBlk0 or AnaBlk1 command bit is written.

Some of the bits in these registers are not allowed to be written to certain values, because doing so would interrupt the normal operation of the code. For this reason, Function $25 might set or clear some of the bits in these registers automatically, overriding the values in the analog register tables.

11.5. Function $25: interrupt sources

Function $25 does not generate RMI interrupts for data transfer, but instead has a dedicated DATA_READY signal to indicate to a host that a cluster of data is available to be read.

Function $25 generates interrupts only when the Stall bit *(F25_SLURP_Data11, bit 7)* is set.

11.6. Function $25: command registers

Name	7	6	5	4	3	2	1	0
F25_SLURP_Cmd0	CRTCBlkSelect		AnaBlk1	AnaBlk0	FrmBlk3	FrmBlk2	FrmBlk1	FrmBlk0
F25_SLURP_Cmd1	Reset Image	—	—	—	—	—	—	—

Figure 76. Function $25 command register

Deleted: 77

As with all command registers, these bits automatically-clear to zero when the requested operation has completed. The bits of this register are defined as follows:

FrmBlk0(F25_SLURP_Cmd0, bit 0)
Set to 1 to change to Frame Block 0. This command can be written at any time, the change will occur at the top of the next frame.

FrmBlk1(F25_SLURP_Cmd0, bit 1)
Set to 1 to change to Frame Block 1. This command can be written at any time, the change will occur at the top of the next frame.

FrmBlk2(F25_SLURP_Cmd0, bit 2)
Set to 1 to change to Frame Block 2. This command can be written at any time, the change will occur at the top of the next frame.

FrmBlk3(F25_SLURP_Cmd0, bit 3)
Set to 1 to change to Frame Block 3. This command can be written at any time, the change will occur at the top of the next frame.

AnaBlk0(F25_SLURP_Cmd0, bit 4)
Set to 1 to change to Analog Register Block 0. This command can be written at any time, the change will occur at the top of the next frame.

AnaBlk1(F25_SLURP_Cmd0, bit 5)
Set to 1 to change to Analog Register Block 1. This command can be written at any time, the change will occur at the top of the next frame.

CRTCBlkSelect(F25_SLURP_Cmd0, bits 7:6)
CRTC block selection.

00 = CRTC disabled
01 = Apply CRTC Table 0
10 = Apply CRTC Table 1
11 = CRTC disabled

ResetImage(F25_SLURP_Cmd1, bit 7)
Set to 1 to reset the image buffer's head and tail pointers and the cluster counter, and to deassert the DATA_READY signal. It is recommended that this be done only while the data-acquisition process is halted (for example, while the device is waiting for VSYNC to deassert).

11.7. Function $25 synchronization

Deleted: Synchronization

All RMI devices built with Function $25 provide a means for synchronizing the image-acquisition process with an external signal and also a mechanism to monitor the state of synchronization. A loss of sync is indicated to the host by setting the Out-Of-Sync status in data packet.

If a host detects an Out-Of-Sync condition, it can halt the scanning process by asserting the VSYNC signal. This will stop all the touch controllers from scanning. When the VSYNC signal is deasserted all the touch controllers will resume scanning, synchronized to the same clock cycle.

11.8. Updating Function $25 parameters

Deleted: Parameters

Function $25 provides two mechanisms for seamlessly changing the acquisition parameters: the block update method and the stalled scan method.

11.8.1. Block update

Deleted: Update

There are two sets of analog register block used to control all the analog parameters. While one set of parameters are in use it is permissible to write a new a new set of parameters into the unused analog register block. When the block has been written with the new parameters the F25_SLURP_Cmd0 register can be written to force a switch to the new analog register set. This mechanism provides a means to update all the analog parameters as a single atomic operation. The data acquisition process will continue with the current analog register parameters until the current frame is completed and will then switch to the new block.

In the same way unused frame block tables can be updated and a single write to F25_SLURP_Cmd0 can be used to cause an atomic update to the new frame table. A single write to this command register can be used to atomically update both the analog registers and the frame block.

11.8.1.1. Multi-Touch Controller block updates

Deleted: Block

Deleted: Updates

On physical interfaces like SPI that support broadcast writes a single write can be issued to multiple touch controllers' F25_SLURP_Cmd0 registers as a single atomic operation to allow seamless frame boundary changes to all the touch controllers in a system. This obviously requires that all the touch controllers in the system are changing to the same analog block number and frame block number – although the contents of those tables can actually be different.

11.8.2. Stalled update

Deleted: Update

Any and all parameters can be updated while the touch controllers are stalled – while the Stalled bit (F25_SLURP_Data11, bit 7) is set. In order to do this, assert VSYNC and wait until the Stall bit is set then update any parameters required. To start a new frame with the new parameters issue a ResetImage command (F25_SLURP_Cmd1, bit 7) to force the start of a new frame. If the current frame block has been changed, the ResetImage command is required prior to releasing VSYNC.

11.9. Recovering from errors

Deleted: Errors

There are several errors that could occur while a host is using Function $25 to acquire data:

- Cluster number in packet is incorrect
- Data packet indicates an Out Of Sync condition
- Data Packet indicates a buffer overflow condition
- ATTN and F$01 indicate an unconfigured state for the touch controller (probably because of a spurious reset)
- Host measured timeout of DATA_READY
- Packet indicates bad checksum
- Error metrics indicate the data is unreliable

Of these possible errors, all but the last two probably indicate a serious condition that require restarting the data acquisition process. A packet checksum or bad error metric might be recoverable by simply discarding the data.

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.

Name	7	6	5	4	3	2	1	0
F30_GPIO_Query0	—	—	HasGpio DriverControl	HasHaptic	HasGpio	HasLed	HasMappable Buttons	Extended Patterns
F30_GPIO_Query1	—	—	—	GpioLedCount				

Figure 77, Function $30 GPIO/LED query registers

Deleted: 78

ExtendedPatterns (F30_GPIO_Query0, bit 0)

This bit indicates whether or not the LED waveforms and ramp registers are the same as those used in RMI3. The Extended LED Patterns bit is '0' to indicate the same patterns and ramps as RMI3. The Extended LED Patterns bit is '1' if there are additional patterns and ramps. See sections 12.3.6 and 12.3.7 for more details.

Formatted: Font color: Green
Deleted: 12.3.6
Formatted: Font color: Green
Deleted: 12.3.7

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.

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. An asterix (‘*’) indicates a variable-sized register block. Every product contains at least one of a 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.

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.

Formatted: Font color: Black

Deleted: 12.3.1

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’.

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F30_GPIO_Ctrl0.*	—	—	—	—	—	—	—	LedSel0

Figure 78. Function $30 pin select register

Deleted: 79

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.

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl1	—	—	Halted	Halt	—	—	—	GpiDebounce

Figure 79, Function $30 General Control register

Deleted: 80

GpiDebounce (F30_GPIO_Ctrl1, bit 0)

If this bit is set to '1', the device applies a debouncing algorithm to all the GPI inputs. The exact debouncing algorithm is device-specific, but in general debouncing strives to eliminate brief glitches in the input signal due, for example, to 'bounce' in the contacts of a mechanical switch. If *GpiDebounce* is not specified, there is still some debounce because transitions are only reported at the report rate.

Halt (F30_GPIO_Ctrl1, bit 4)

Writing this bit to a '1' freezes this function's operation. All LED ramping and animation are stopped. All GPIO input and output operations are stopped. If ramping and animation are frozen, each LED holds at whatever intensity it has at the moment the freeze bit is set, even if the LED is halfway through a ramp operation. When synchronized behavior is required, for example setting several LEDs active, this bit should be used.
When this bit is written to a '0', the ramps will continue where they left off.
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)

This bit is a status indicator that acknowledges that this function is halted. Sometime after the *Halt* bit is set or reset, this bit will be set or reset.

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 below. The reset default of Dir*N* and Data*N* is typically '00' (high-impedance).

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl2.0	—	—	—	—	—	—	—	Dir0
F30_GPIO_Ctrl3.0	—	—	—	—	—	—	—	Data0

Figure 80, Function $30 GPIO input/output control register

Deleted: 81

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 6.

Deleted: Table 7

Table 6. Function $30 GPIO control settings

Deleted: 67

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’.

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl4.*	—	—	—	—	—	—	—	LedAct0

Figure 81. Function $30 LED Active register

Deleted: 82

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:

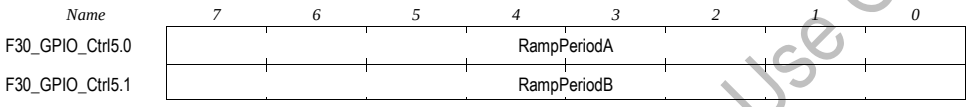

Figure 82. Function $30 LED Ramp Period registers

Deleted: 83

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.

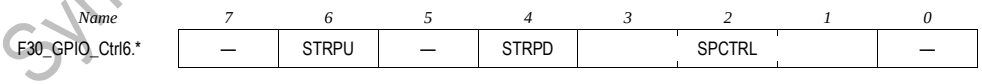

Figure 83. Function $30 per-GPIO Control register

Deleted: 84

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.

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.

SPCTRL (F30_GPIO_Ctrl6., bits 3:0)*

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.

12.3.7.2. F30_GPIO_Ctrl6.*: LED control

This is the register description for controlling an LED.

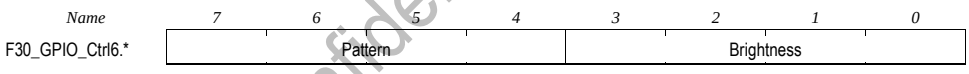

Figure 84. Function $30 per-LED Control register

Deleted: 85

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.

Note: The following patterns are only available if the private field Extended LED Patterns is indicated in Query Register 0.
The exact definition of the extended patterns is still under consideration. "Standard" patterns are being implemented first. However it may be decided that "extended patterns" will remain unimplemented until there is a requirement. The patterns described here are simple variations on existing patterns. A "hold before fall" pattern has been requested.

Pattern = '1000': Rise period A and fall period B.
In this setting, when the LED is activated by setting the LED Active *N*, 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 by clearing the LED Active *N* bit, the LED ramps down to the "off" state over a time determined by *RampPeriodB* and then remains "off."

Pattern = '1001': Rise period C and fall period D.
This pattern is like pattern '1000', except that the *RampPeriodC* and *RampPeriodD* parameter determines the ramp rate instead of *RampPeriodA* and *RampPeriodB*.

Pattern = '1010', '1011': Reserved
Note*: These are left reserved at the Spec Level for Custom Implementation.*

Pattern = '1100': Complex ramping animation using C,D,E,F.
In this setting, when the LED is activated the LED ramps to the intensity indicated by the *Brightness* field over a time determined by *RampPeriodC*, and then holds at that intensity for a time determined by *RampPeriodD*. The LED then ramps down to the "off" state over *RampPeriodE* and remains off for *RampPeriodF*. 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.

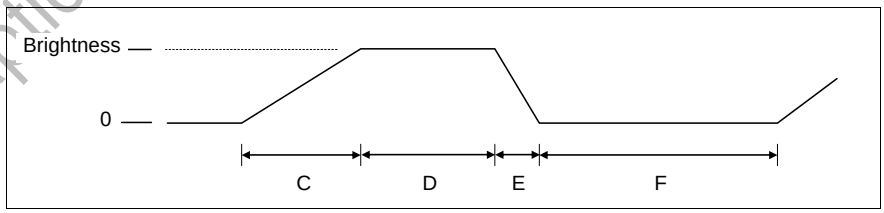

Figure 85

Deleted: *86*

Pattern = '1101': Complex ramping animation using A,B,E,F.
This pattern is like pattern '1100', except that the Ramp Period A,B,E,F registers determine the animation ramp rates.

Pattern = '1110': Ramping animation using C,D.
This pattern is like pattern '0110', except that *RampPeriodC* and *RampPeriodD* determine the ramping animation ramp rate rather than Ramp Period A and B.

Pattern = '1111': Pulsed animation using C,D.
This pattern is like pattern '0111', except that *RampPeriodC* and *RampPeriodD* determine the pulsed animation ramp rates rather than Ramp Period A and B.

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.

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl7.*	OpenDrain	Invert	Valid	CapacitiveButtonNumber				

Figure 86. Function $30 per-GPIO/LEDButton Mapping register

Deleted: 87

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.

Implementation Note: When this bit is '0' the Dir*N* bit is forced to '1' and the button state overrides the mapped GPIO's Data*N* bit. When this bit is a '1' the Data*N* bit is forced to '0' and the button state overrides the mapped GPIO's Dir*N* bit. Regardless of which of the two GPIO control bits is overridden, the resulting Data and Dir bits determine the behavior of the GPIO pin, as described in section 12.3.4.

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.

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F30_GPIO_Ctrl8.0	HapticEn7	HapticEn6	HapticEn5	HapticEn4	HapticEn3	HapticEn2	HapticEn1	HapticEn0
F30_GPIO_Ctrl8.1						HapticEn10	HapticEn9	HapticEn8

Figure 87, Function $30 Haptic Enable registers

Deleted: *88*

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.

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F30_GPIO_Ctrl9	HapticDuration							

Figure 88, Function $30 Haptic Duration register

Deleted: *89*

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. Future possibilities - private

This section points out some things that are not available in this first implementation of RMI. RMI is extensible through specifying capabilities in the query registers. Here are some things to consider for future implementations:

- **More functionality for Button-to-GPIO mapping.**
  The following is a way to give more detailed control over each button that is mapped; this gives all the functionality of the OneTouch spec, with additional flexibility. Each button can individually be specified as toggle on touch, toggle on lift or part of a radio button group. To implement this, a new Query bit is defined, for example *ExtendedButtonMappingControl*. When this bit is on the register, layout of the Button-to-GPIO mapping is different and there is a *ButtonMode* 2-bit field. The *ButtonControl* bits, *ButtonPolarity*, *OpenDrain*, and this new *ButtonMode* might be put into their own registers.

  The ButtonMode is:
  '00' – normal
  '01' – toggle on touch
  '10' – toggle on lift
  '11' – part of radio button group

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.

Name	7	6	5	4	3	2	1	0
F30_GPIO_Data0.*	—	—	—	—	—	—	—	GpiLedData0

Figure 89. Function $30 per-GPIO/LED data register

Deleted: 90

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.

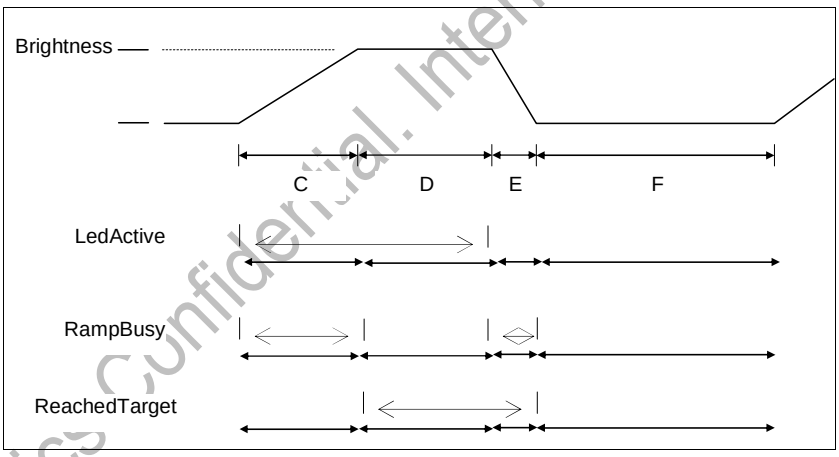

Figure 90. LED ramping status

Deleted: 91

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 of a 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.

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F30_GPIO_Query0	—	—	HasGpio DriverControl	HasHaptic	HasGpio	HasLed	HasMappable Buttons	—
F30_GPIO_Query1	—	—	—	GpioLedCount				
F30_GPIO_Ctrl0.0	LedSel7	LedSel6	LedSel5	LedSel4	LedSel3	LedSel2	LedSel1	LedSel0
F30_GPIO_Ctrl0.1	—	—	—	—	—	LedSel10	LedSel9	LedSel8
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	GpiDebounce
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	—	—	—	—	—	LedAct10	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	OpenDrain	ButtonPolarity	Valid	CapacitiveButtonNumber				
F30_GPIO_Ctrl7.1	OpenDrain	ButtonPolarity	Valid	CapacitiveButtonNumber				

This figure is continued on the following page.

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl7.2 …. F30_GPIO_Ctrl7.8	OpenDrain	ButtonPolarity	Valid	CapacitiveButtonNumber				
F30_GPIO_Ctrl7.9	OpenDrain	ButtonPolarity	Valid	CapacitiveButtonNumber				
F30_GPIO_Ctrl7.10	OpenDrain	ButtonPolarity	Valid	CapacitiveButtonNumber				
F30_GPIO_Ctrl8.0	HapticEn7	HapticEn6	HapticEn5	HapticEn4	HapticEn3	HapticEn2	HapticEn1	HapticEn0
F30_GPIO_Ctrl8.1	—	—	—	—	—	HapticEn10	HapticEn9	HapticEn8
F30_GPIO_Ctrl9	Haptic Duration							
F30_GPIO_Data0.0	GpiLedData7	GpiLedData6	GpiLedData5	GpiLedData4	GpiLedData3	GpiLedData2	GpiLedData1	GpiLedData0
F30_GPIO_Data0.1	—	—	—	—	—	GpiLedData10	GpiLedData9	GpiLedData8

Figure 91

Deleted: *92*

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’.

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	—	—	—	GpiDebounce
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.*	RampPeriodA							
F30_GPIO_Ctrl6.*	Pattern-Brightness-for-LEDs =or= STRPU-STRPD-SPCTRL-for-GPIOs							

Figure 92

Deleted: *93*

Function $30 example: LEDs only

For this example, *HasLed* is ‘1’ and *HasGpio* is ‘0’.

Name	7	6	5	4	3	2	1	0
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	GpiDebounce
F30_GPIO_Ctrl4.*	LedAct7	LedAct6	LedAct5	LedAct4	LedAct3	LedAct2	LedAct1	LedAct0
F30_GPIO_Ctrl5.*	RampPeriod*							
F30_GPIO_Ctrl6.*	Pattern				Brightness			

Figure 93

Deleted: *94*

12.8.2. Function $30 example: GPIOs only, with driver control

For this example, *HasLed* is ‘0’, *HasGpio* is ‘1’, and *HasGpioDriverControl* is ‘1’.

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	GpiDebounce
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			—

Figure 94

Deleted: *95*

12.8.3. Function $30 example: GPIOs only, without driver control

For this example, *HasLed* is ‘0’, *HasGpio* is ‘1’, and *HasGpioDriverControl* is ‘0’.

Name	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
F30_GPIO_Ctrl1	—	—	Halted	LedHalt	—	—	—	GpiDebounce
F30_GPIO_Ctrl2.*	Dir7	Dir6	Dir5	Dir4	Dir3	Dir2	Dir1	Dir0
F30_GPIO_Ctrl3.*	Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0

Figure 95

Deleted: *96*

13. 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.

13.1. Overview

13.1.1. Non-volatile memory organization

Figure 96 shows the basic storage methodology for the device firmware:

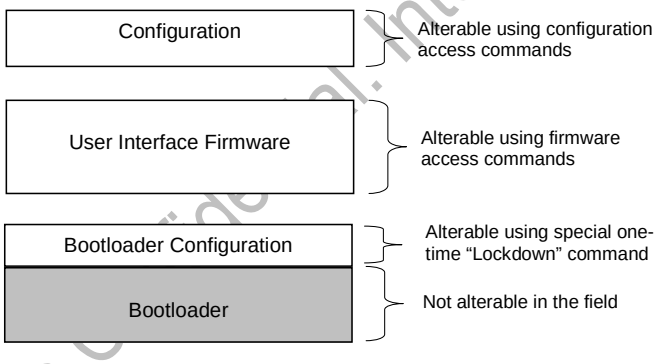

Figure 96. Non-volatile memory organization

Deleted: 97

13.1.1.1. 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.

13.1.1.2. 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.

13.1.1.3. Bootloader

The bootloader checks the integrity of the UI firmware space and provides the ability to re-flash a new UI or configuration area.

Comment [x2]: * the

Comment [x3]: * 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.

13.1.2. 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 *Flash Prog* 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.

13.2. Function $34: query registers

13.2.1. 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'.

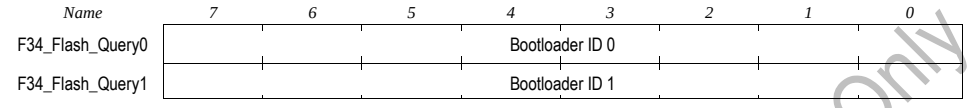

Figure 97. Function $34 Bootloader ID query registers

Deleted: 98

13.2.2. F34_Flash_Query2: flash properties query

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

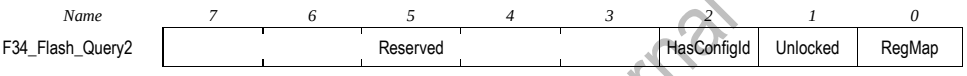

Figure 98. Function $34 Flash Properties query register

Deleted: 99

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.

This query bit 0 allows tools to distinguish between the original Function $34 register map (which had the F34_Flash_Data0 control/status register as the first register) and the new map, which has the control/status data block at the very *end* of the block data registers:

- 0: device supports the original register map (valid **only** for the original RMI implementations: TM-01208, TM-01213-001, TM-01213-002, and TM-01217)
- 1: device supports the optimized register map, used for all other RMI implementations

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.

13.2.3. 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.

Name	7	6	5	4	3	2	1	0
F34_Flash_Query3	Block Size (bits7:0)							
F34_Flash_Query4	Block Size (bits15:8)							

Figure 99, Function $34 Block Size query registers

Deleted: 100

13.2.4. 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.

Name	7	6	5	4	3	2	1	0
F34_Flash_Query5	Firmware Block Count (bits7:0)							
F34_Flash_Query6	Firmware Block Count (bits15:8)							

Figure 100, Function $34 Firmware Block Count query registers

Deleted: 101

Figure 101, Firmware Size

Deleted: 102

13.2.5. 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.

Name	7	6	5	4	3	2	1	0
F34_Flash_Query7	Configuration Block Count (bits: 7:0)							
F34_Flash_Query8	Configuration Block Count (bits 15:8)							

Figure 102, Function $34 Configuration Size query registers

Deleted: 103

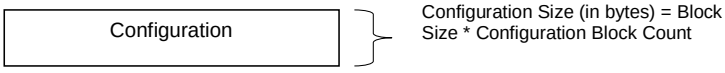

Figure 103, Configuration Size

Deleted: 104

13.3. Function $34: control registers

These control registers are present only if *HasConfigID* (F34_FLASH_Query2, bit 2) reports as '1'.

Name	7	6	5	4	3	2	1	0
F34_Flash_Ctrl0.0	Customer Configuration ID bits 7:0							
F34_Flash_Ctrl0.1	Customer Configuration ID bits 15:8							
F34_Flash_Ctrl0.2	Customer Configuration ID bits 23:16							
F34_Flash_Ctrl0.3	Customer Configuration ID bits 31:24							

Figure 104, Function $34 control registers

Deleted: 105

Customer Configuration ID (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, since 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.

13.4. Function $34: data registers

Function $34's data registers specify block numbers, report block data, and control all aspects of flash update operations.

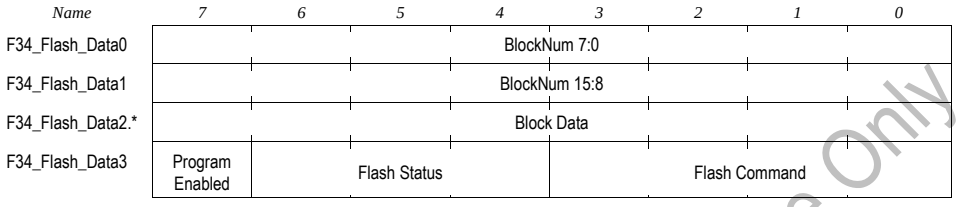

Figure 105. Function $34 data registers

Deleted: 106

13.4.1. 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.

13.4.2. 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.

For V1 bootloaders only, upon completion of a *CRC Firmware Block* command, the first Block Data register (*F34_Flash_Data2.0*) contains a pass/fail indicator – 0 = Valid image, $FF = invalid image.

13.4.3. F34_Flash_Data3: flash control/status register

This register is used to control all aspects of flash update operation.

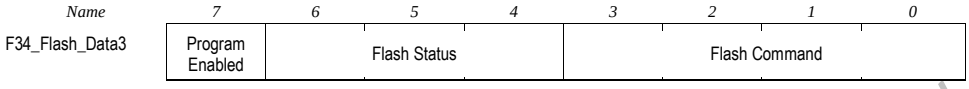

Figure 106. Function $34 control/status register

Deleted: 107

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 7. 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* CRC FW Block (V1 only)	No		No
$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
$8-$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.

In legacy systems (those that use the V1 bootloader), this CRC Firmware Block command could be used to test a firmware image against a device to ensure the firmware image was correct and appropriate for the device without writing that firmware into the device flash. Legacy devices maintained an internal incremental CRC that was initialized when the Block Number was set to 0. Each block of data was written into the Block Data register and this command was issued to decrypt the block and run it through the CRC.

Note: Each block of data in a firmware image must have this command executed **in order**, from beginning to end. At the completion of the last block of data the Block Data register *(F34_Flash_Data2.0)* register is read back and indicates the CRC correctness for the firmware image: 0 = Valid image, $FF = invalid image.

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. For information about the RMI image file, see the *RMI4 Bootloader Procedures* Application Note (PN: 506-000221-01).

See also the *RMI4 Firmware Image File Description* Application Note (PN 506-000215-01).

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 based on the table below.

Table 8

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 $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: 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 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 9 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 *Flash Prog* 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 is said to be in "UI mode". When this bit reports as '1', the device is said to be in "Bootloader mode".

Deleted: .

Deleted: .

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.

13.5. 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.

For details of the flash reprogramming procedures, see the *Flash Reprogramming Using RMI4 Function $34* document (PN 511-000409-01).

14. 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.

14.1. Function $36: query register

The ADC Function $36 implements a single query register, the General Properties query.

Name	7	6	5	4	3	2	1	0
F36_ADC_Query0	—	—	—	—	—	—	NumChannels	

Figure 107, Function $36 query register

Deleted: 108

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.

14.2. Function $36: control register

The control register determines the mode of operation of the ADC channels.

Name	7	6	5	4	3	2	1	0
F36_ADC_Ctrl0	—	—	—	—	—	—	Enable1	Enable0

Figure 108. Function $36 control register

Deleted: 109

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.

14.3. 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.

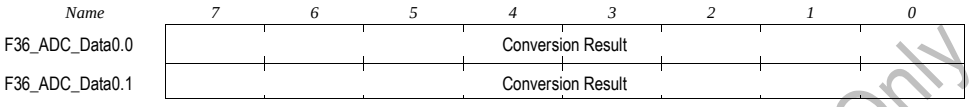

Figure 109. Function $36 data registers, two-channel example

Deleted: 110

ConversionResult (F36_ADC_Data0., bits 7:0)*

Reports the result of the conversion.

14.4. Function $36: command registers

The command register initiates conversions by the ADC channels.

Name	7	6	5	4	3	2	1	0
F36_ADC_Cmd0	—	—	—	—	—	—	Conv1	Conv0

Figure 110. Function $36 command register

Deleted: 111

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.

14.5. 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.

15. Function $54: Test reporting (T1321 devices)

Function $54 provides test reporting and analog diagnostic and control functions that allow direct visibility into image sensing. Only some devices support this feature.

Function $54 provides direct access to low-level capacitance data before processing and interpretation for testing and debugging 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 matrix of signed integer values, corresponding to each intersection in the grid of transmitter and receiver electrodes used for capacitive sensing.

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.

15.1. Function $54: query registers

The query registers describe the capabilities available in a device.

Name	7	6	5	4	3	2	1	0
F54_AD_Query0	Number of Receiver Electrodes							
F54_AD_Query1	Number of Transmitter Electrodes							
F54_AD_Query2	—	Has Image16	—	—	Has Image8	Has Baseline	—	—
F54_AD_Query3.0	Clock Rate LSB							
F54_AD_Query3.1	Clock Rate MSB							
F54_AD_Query4	Touch Controller Family							
F54_AD_Query5	--							Has Pixel Touch Threshold-Adjustment
F54_AD_Query6	Has Relaxation Control	Has One Byte Report Rate	Has Two Byte Report Rate	Has Low Power Control	Has Firmware Noise Mitigation	Has Sense Frequency Control	Has Interference Metric	Has Arbitrary Sensor Assignment
F54_AD_Query7	—						Axis Compensation Mode	
F54_AD_Query8	—	Has Perf Frequency NoiseControl	Has Edge Compensation	Has Pixel Threshold Hysteresis	Has Cmn Cap Scale Factor	Has Cmn Removal	Has IIR Filter	—
F54_AD_Query9	--							
F54_AD_Query10	--							
F54_AD_Query11	--							
F54_AD_Query12	--				Number of Sensing Frequencies			
F54_AD_Query13	--				Bursts per Cluster			

Figure 111. Function $54 query registers

Deleted: 112

15.1.1. 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.

15.1.2. 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 capacitance data to 8 bits. These images can be read more quickly because the image size is smaller, but it is possible that the images may be clipped.

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 a full 16-bit delta image. This image will contain data corresponding to the full dynamic range of the sensor.

Reserved (F54_AD_Query2, bit 7)

Reserved.

15.1.3. 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 as an unsigned 16-bit value, in units of 10 kHz. For example, a device with a 16 MHz master clock would report the value 1600.

15.1.4. 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_Ctrl4, F54_AD_Ctrl5, F54_AD_Ctrl6, F54_AD_Ctrl8, and F54_AD_Ctrl9 exist. When this register reports as '01', registers F54_AD_Ctrl4 through F54_AD_Ctrl9 exist.

Value	Touch Controller Family
$00	T1320
$01	T1321

15.1.5. 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.

15.1.6. F54_AD_Query6: data acquisition

HasArbitrarySensorAssignment (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 through F54_AD_Ctrl16 exist.

HasInterferenceMetric (F54_AD_Query6, bit 1)

This bit indicates whether the touch controller is capable of estimating the amount of noise present while frame data is acquired. When this bit reports as ‘1’, register F54_AD_Ctrl10 exists.

HasSenseFrequencyControl (F54_AD_Query6, bit 2)

This bit indicates the presence of a mechanism that uses sense-frequency shifting as a means of avoiding narrow-band noise sources.

When this bit reports as ‘1’, registers F54_AD_Query12, F54_AD_Ctrl17 through F54_AD_Ctrl19, F54_AD_Ctrl21, and F54_AD_Data4 exist.

HasFirmwareNoiseMitigation (F54_AD_Query6, bit 3)

This bit indicates the presence of firmware-based mechanisms 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)

This bit indicates the ability of the system to doze in a low power state while no fingers are present on the sensor.

When this bit reports as ‘1’, register F54_AD_Ctrl11 exists.

HasTwoByteReportRate (F54_AD_Query6, bit 5)

This bit indicates the ability of the firmware to calculate an 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 firmware to calculate an 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 firmware to compensate for thermal effects in the system. When this bit reports as ‘1’, registers F54_AD_Ctrl12 and F54_AD_Ctrl13 exist.

15.1.7. F54_AD_Query7: per axis compensation

AxisCompensationMode (F54_AD_Query7, bits 1:0)

A ‘00’ means no axis compensation, ‘01’ means single-axis compensation, ‘10’ means dual-axis compensation and ‘11’ is reserved. This feature is a **build-time** option.

Reserved (F54_AD_Query7, bits 7:2)

Reserved.

15.1.8. 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’, registers F54_AD_Ctrl27 and F54_AD_Ctrl28 exist.

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’, registers F54_AD_Ctrl29 exist.

HasCmnCapScaleFactor (F54_AD_Query8, bit 3)

This bit indicates whether the CMN Cap Scale Factor 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 the Edge Compensation feature, which allows the host to boost the capacitive signal on the four edge electrodes. Each edge can be controlled independently.

When this bit reports as ‘1’, F54_AD_Ctrl32.*, F54_AD_Ctrl33.*, F54_AD_Ctrl34.*, and F54_AD_Ctrl35.* exist.

HasPerfFrequencyNoiseControl (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.

15.1.9. F54_AD_Query9 through F54_AD_11: reserved

15.1.10. F54_AD_Query12: sense frequency control

This query register is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as ‘1’.

NumberOfSensing Frequencies (F54_AD_Query12, bits 3:0)

This field reports the number of sensing frequencies the device supports.

Reserved (F54_AD_Query12, bits 7:4)

Reserved.

15.1.11. F54_AD_Query13: analog Information

BurstsPerCluster (F54_AD_Query13, bits 3:0)

This field reports the number of bursts per cluster. A ‘burst’ is the result of a single ADC conversion across all receivers. A ‘cluster’ is a group of bursts that are summed together to form the aggregate ADC conversion result for each receiver.

Reserved (F54_AD_Query13, bits 7:4)

Reserved.

15.2. Function $54: control registers

Name	7	6	5	4	3	2	1	0
F54_AD_Ctrl0	—						No Scan	No Relax
F54_AD_Ctrl1	—				Bursts per Cluster			
F54_AD_Ctrl2.0	Saturation Capacitance LSB							
F54_AD_Ctrl2.1	Saturation Capacitance MSB							
F54_AD_Ctrl3	Pixel Touch Threshold							
F54_AD_Ctrl4	—						Receiver Feedback Capacitance	
F54_AD_Ctrl5	—			RefLow Polarity	RefLow Feedback Capacitiance		RefLow Capacitance	
F54_AD_Ctrl6	—			RefHigh Polarity	RefHigh Feedback Capacitance		RefHigh Capacitance	
F54_AD_Ctrl7	—			CBC Transmitter Carrier Selection	CBC Polarity		CBC Capacitance	
F54_AD_Ctrl8	Integration Duration bits 7:0							
F54_AD_Ctrl8.1	—						Integration Duration bits 9:8	
F54_AD_Ctrl9	Reset Duration							
F54_AD_Ctrl10	—				Noise Sensing Bursts per Frame			
F54_AD_Ctrl11.0	Wakeup Threshold LSB							
F54_AD_Ctrl11.1	Wakeup Threshold MSB							
F54_AD_Ctrl12	Slow Relaxation Rate							
F54_AD_Ctrl13	Fast Relaxation Rate							
F54_AD_Ctrl14.*	—						Per Axis Compensation	Physical To Logical Axes
F54_AD_Ctrl15.*	Sensor Rx Assignment							
F54_AD_Ctrl16.*	Sensor Tx Assignment							
F54_AD_Ctrl17.*	Filter Bandwidth			—	Disable	Burst Count bits 10:8		
F54_AD_Ctrl18.*	Cycles per Burst bits 7:0							
F54_AD_Ctrl19.*	Stretch Duration							
F54_AD_Ctrl20	—							
F54_AD_Ctrl21.0	HNM Frequency Shift Noise Threshold LSB							
F54_AD_Ctrl21.1	HNM Frequency Shift Noise Threshold MSB							
F54_AD_Ctrl22	Hardware Noise Mitigation Exit Density							
F54_AD_Ctrl23.0	Medium Noise Threshold LSB							
F54_AD_Ctrl23.1	Medium Noise Threshold MSB							
F54_AD_Ctrl24.0	High Noise Threshold LSB							
F54_AD_Ctrl24.1	High Noise Threshold MSB							
F54_AD_Ctrl25	FNM Frequency Shift Density							
F54_AD_Ctrl26	Firmware Noise Mitigation Exit Threshold (Frame Count)							
F54_AD_Ctrl27	IIR Filter Coefficient							
F54_AD_Ctrl28.0	FNM Frequency Shift Noise Threshold LSB							

This figure is continued on the following page.

Name	7	6	5	4	3	2	1	0
F54_AD_Ctrl28.1	FNM Frequency Shift Noise Threshold MSB							
F54_AD_Ctrl29	Cmn Mode Noise Control	—						
F54_AD_Ctrl30	CMN Cap Scale Factor							
F54_AD_Ctrl31	Pixel Threshold Hysteresis							
F54_AD_Ctrl32.0	Rx Low Edge Compensation LSB							
F54_AD_Ctrl32.1	Rx Low Edge Compensation MSB							
F54_AD_Ctrl33.0	Rx High Edge Compensation LSB							
F54_AD_Ctrl33.1	Rx High Edge Compensation MSB							
F54_AD_Ctrl34.0	Tx Low Edge Compensation LSB							
F54_AD_Ctrl34.1	Tx Low Edge Compensation MSB							
F54_AD_Ctrl35.0	Tx High Edge Compensation LSB							
F54_AD_Ctrl35.1	Tx High Edge Compensation MSB							
F54_AD_Ctrl36.*	Axis 1 Compensation							
F54_AD_Ctrl37.*	Axis 2 Compensation							
F54_AD_Ctrl38.*	Noise Control 1							
F54_AD_Ctrl39.*	Noise Control 2							
F54_AD_Ctrl40.*	Noise Control 3							

Figure 112, Function $54 control registers

Deleted: 113

15.2.1. F54_AD_Ctrl0: general control 0

NoRelax (F54_AD_Ctrl0, bit 0)

Setting this bit to ‘1’ disables the thermal relaxation mechanisms.

NoScan (F54_AD_Ctrl0, bit 1)

Setting this bit to ‘1’ allows the host to turn off the Transmitter (Tx) drive pattern that is triggered by the firmware during image frame acquisition. The host is advised not to set this bit run-time, it is strictly for internal purposes.

Reserved (F54_AD_Ctrl0, bits 7:2)

Reserved.

15.2.2. F54_AD_Ctrl1: general control 1

BurstsPerCluster (F54_AD_Ctrl1, bits 3:0)

Reports the number of bursts per cluster.

Reserved (F54_AD_Ctrl1, bits 7:4)

Reserved.

15.2.3. F54_AD_Ctrl2.0/2.1: saturation capacitance

The saturation capacitance represents the expected change in capacitance that results when a nominal pixel on a sensor goes from being completely uncovered to being completely covered by a conductive object.

Note: This represents a new name for what used to be called Delta Ct. There was a nomenclature problem over the years in that Delta Ct was also used to describe things other than a saturation capacitance, so we have decided to rename the term to explicitly define this notion using a name that is meaningful to customers in a fashion that is both effective and easy to understand.

SaturationCapacitanceLSB (F54_AD_Ctrl2.0)

Defines the saturation capacitance as an unsigned 16-bit value specified in femtoFarads. It represents the least significant byte (LSB).

SaturationCapacitanceMSB (F54_AD_Ctrl2.1)

Defines the saturation capacitance as an unsigned 16-bit value specified in femtoFarads. It represents the most significant byte (MSB).

15.2.4. F54_AD_Ctrl3: pixel touch threshold

This register is only present if *HasPixelTouchThreshold* (F54_AD_Query8, bit 4) reports as '1'.

PixelTouchThreshold (F54_AD_Ctrl3)

The pixel touch threshold is used in finger detection. This threshold is set to a specific fraction of the Saturation Capacitance inside the device firmware as a multiplier scale factor.

The value is defined to be a 1.7 fixed-point value. This allows for an adjustment of the internal threshold by increments of approximately 1%.

15.2.5. F54_AD_Ctrl4: miscellaneous analog control

This register is only present if *AnalogHardwareFamily* (F54_AD_Query4) reports as $00 (T1320 family) or $01 (T1321 family).

Receiver Feedback Capacitance (F54_AD_Ctrl4, bits 1:0)

Sets the receiver feedback capacitance.

00 – 4 pF
01 – 8 pF
10 – 12 pF
11 – 16 pF

Reserved (F54_AD_Ctrl4, bits 7:2)

Reserved.

15.2.6. F54_AD_Ctrl5: RefCap RefLo settings

This register is only present if *AnalogHardwareFamily* (F54_AD_Query4) reports as $00 (T1320 family) or $01 (T1321 family).

RefLo Capacitance (F54_AD_Ctrl5, bits 1:0)

Sets the RefLo capacitance:

00 – 0 pF
01 – 1 pF
10 – 2 pF
11 – 3 pF

RefLo Feedback Capacitance (F54_AD_Ctrl5, bits 3:2)

Sets the RefLo feedback capacitance:

00 – 4 pF
01 – 8 pF
10 – 12 pF
11 – 16 pF

RefLo Polarity (F54_AD_Ctrl5, bit 4)

Defines the polarity of the RefLo capacitance setting (0 = negative, 1 = positive).

Reserved (F54_AD_Ctrl5, bits 7:5)

Reserved.

15.2.7. F54_AD_Ctrl6: RefCap RefHigh settings

This register is only present if *Analog Hardware Family* (F54_AD_Query4) reports as $00 (T1320 family) or $01 (T1321 family).

RefHigh Capacitance (F54_AD_Ctrl6, bits 1:0)

Defines the RefHi capacitance:

00 – 1 pF
01 – 2 pF
10 – 4 pF
11 – 6 pF

RefHigh Feedback Capacitance (F54_AD_Ctrl6, bits 3:2)

Sets the RefHigh feedback capacitance:

00 – 4 pF
01 – 8 pF
10 – 12 pF
11 – 16 pF

RefHigh Polarity (F54_AD_Ctrl6, bit 4)

Defines the polarity of the RefHigh capacitance setting (0 = negative, 1 = positive).

Reserved (F54_AD_Ctrl6, bits 7:5)

Reserved.

15.2.8. F54_AD_Ctrl7: CBC cap settings

This register is only present if *AnalogHardwareFamily* (F54_AD_Query4) reports as $01.

CBCCapacitance (F54_AD_Ctrl7, bits 2:0)

The Coarse Baseline Correction (CBC) capacitance setting defines an amount of capacitance to be subtracted from the measured baseline.

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_Ctrl7, bit 3)

Defines the polarity of the CBC capacitance setting (0 = negative, 1 = positive).

CBCTransmitterCarrieSelection (F54_AD_Ctrl7, bit 4)

Defines the CBC carrier select control field.

Reserved (F54_AD_Ctrl7, bits 7:5)

Reserved.

15.2.9. 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 (T1320 family) or $01 (T1321 family).

IntegrationDuration (F54_AD_Ctrl8.0)

This register holds bits 7:0 of the 10-bit Integration Duration hardware setting.

IntegrationDuration (F54_AD_Ctrl8.1, bits 1:0)

This register holds bits 9:8 of the 10-bit Integration Duration hardware setting.

Reserved (F54_AD_Ctrl8.1, bits 7:2)

Reserved.

15.2.10. F54_AD_Ctrl9: reset duration

This register is only present if *Analog Hardware Family* (F54_AD_Query4) reports as $00 (T1320 family) or $01 (T1321 family).

ResetDuration (F54_AD_Ctrl9)

Defines the reset duration.

15.2.11. F54_AD_Ctrl10: noise sensing bursts per frame

This register is only present if *HasInterferenceMetric* (F54_AD_Query6, bit 1) reports as ‘1’.

NoiseSensingBurstsPerFrame (F54_AD_ Ctrl10, bits 3:0)

Determines the number of noise sensing bursts per frame. 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.

15.2.12. F54_AD_Ctrl11.0/11.1: wakeup threshold

This register is only present if *HasLowPowerControl* (F54_AD_Query6, bit 4) reports as '1'.

WakeupThreshold LSB (F54_AD_Ctrl11.0)

Defines the wakeup threshold as an unsigned 16-bit value, specified in femtoFarads. It represents the least significant byte (LSB).

WakeupThreshold MSB (F54_AD_Ctrl11.1)

Defines the wakeup threshold as an unsigned 16-bit value, specified in femtoFarads. It represents the least significant byte (MSB).

15.2.13. 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 is used to compensate for thermal changes to the sensing system (sensor and touch controller).

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.

15.2.14. 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 is used to compensate for events that may have corrupted the baseline. 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.

15.2.15. F54_AD_Ctrl14.*: sensor assignment properties

This register is only present if *HasArbitrarySensorAssignment* (F54_AD_Query6, bit 0) reports as '1'.

Note: This register can be changed at runtime. The change will take effect before the next reporting frame is started.

PhysicalToLogicalAxes (F54_AD_Ctrl14., bit 0)*

When '0', the Rx electrodes are defined to be the Y-axis.

When '1', the Rx electrodes are defined to be the X-axis.

By proper definition of the Physical To Logical Axes field along with careful definition of the Rx and Tx mapping registers, a sensor can be defined to report in either portrait or landscape mode, with the origin of the sensor located in any of the four corners.

PerAxisCompensation (F54_AD_Ctrl14., bit 1)*

When '0', the Rx electrodes are defined to be the Y-axis. Bit 1 is used to select the 'axis' on which scale factors are applied when **only** single-axis compensation is present. A '0' means compensation is applied on physical Rx-axis. A '1' means Tx-axis is compensated.

This register is present **only** when A *HasArbitrarySensorAssignment* (F54_AD_Query6, bit 0) is present. Axis compensation is dependent on having arbitrary mapping in the build.

Reserved (F54_AD_Ctrl14., bits 7:2)*

Reserved.

15.2.16. F54_AD_Ctrl15.*: sensor Rx assignment

This register is only present if *HasArbitrarySensorAssignment* (F54_AD_Query6, bit 0) reports as '1'.

Note: The registers that control the Rx sensor mapping are sampled once as the device boots after a reset. Changing the values of these registers at runtime will have no effect until the next device reset.

SensorRxAssignment (F54_AD_Ctrl15.)*

This section controls the sensor mapping from the physical receiver electrodes to the logical sensor traces. The length of this replicated register block is defined by F54_AD_Query0 *NumberOfReceiverElectrodes*.

The mapping is defined to start from the coordinate 0 side of the sensor. Unused entries should be set to the value $FF.

15.2.17. F54_AD_Ctrl16.*: sensor Tx assignment

This register is only present if *HasArbitrarySensorAssignment* (F54_AD_Query6, bit 0) reports as '1'.

Note: The registers that control the Tx sensor mapping are sampled once as the device boots after a reset. Changing the values of these registers at runtime will have no effect until the next device reset.

SensorTxAssignment (F54_AD_Ctrl16.)*

This section controls the sensor mapping from the physical transmitter electrodes to the logical sensor traces. The length of this replicated register block is defined by F54_AD_Query1 *NumberOfTransmitterElectrodes*. The mapping is defined to start from the coordinate 0 side of the sensor. Unused entries should be set to the value $FF.

15.2.18. F54_AD_Ctrl17.* through F54_AD_Ctrl19.*: sense frequency overview

The F54_AD_Ctrl17 through F54_AD_Ctrl19 registers are used to define the various sense frequencies that are available for use by the device as part of its noise mitigation mechanisms. The values for all of these registers will be calculated by Design Studio 4, as part of the tuning process.

Note: It is anticipated that there will be a maximum of eight sense frequencies available to the system, for a total of up to 24 Control registers. Unused sense frequencies will be indicated by setting the

cycles per burst to $00, or by setting the ‘Disable’ bit in the appropriate F54_AD_Ctrl17.* register.

15.2.19. F54_AD_Ctrl17.*: sense frequency control

This register is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as ‘1’.

Typically, the selection of the active sense frequency should be left to the control of the device so that it can deal most effectively with transient noise events. For testing purposes, the specific frequency being used can be set by writing the F54_AD_Data5 *Sense Frequency* data register.

BurstCount (F54_AD_Ctrl17., bits 2:0)*

Cycles per burst-- bits 10:8.

Disable (F54_AD_Ctrl17., bit 3)*

Reserved (F54_AD_Ctrl17., bit 4)*

Reserved.

FilterBandwidth (F54_AD_Ctrl17., bits 7:5)*

Filter bandwidth.

Note: The contents of this field are CPU-specific. The setting for this register would correspond to the filter bandwidth settings available for a given processor.

15.2.20. F54_AD_Ctrl18.* cycles per burst

This register is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as ‘1’.

CyclesPerBurst (F54_AD_Ctrl18.)*

Cycles per burst.

15.2.21. F54_AD_Ctrl19.*: stretch duration

This register is only present if *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) reports as ‘1’.

StretchDuration (F54_AD_Ctrl19.)*

Stretch duration.

15.2.22. F54_AD_Ctrl20 through F54_AD_Ctrl29: noise mitigation

The noise mitigation mechanism operates in two distinct states: one in which the hardware filtering is sufficient to manage the noise sources (normal circumstances/hardware noise mitigation) and one in which additional firmware filtering is required to deal with the noise sources (high-noise circumstances/firmware noise mitigation). The Control registers that follow define how various controls work within a particular noise state, as well as how the system transitions between the two noise states.

Note: Certain aspects of the noise mitigation control mechanism specified here targets the current FW implementation. It may be that we add new features as the algorithm evolves, or it may be that we trash all these registers and start over for a new noise mitigation algorithm.

15.2.23. F54_AD_Ctrl20: noise mitigation general control

This register is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

Reserved (F54_AD_Ctrl20)

Reserved.

15.2.24. F54_AD_ Ctrl21.0/21.1: HNM frequency shift noise threshold format

This register pair is only present if both *HasSenseFrequencyControl* (F54_AD_Query6, bit 2) and *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) report as '1'.

This hardware noise mitigation (HNM) threshold defines whether the acquired data will be processed by the finger processing firmware (if IM < F54_AD_FrequencyShift_threshold), or if the data should be discarded and the sense frequency changed (IM >= F54_AD_FrequencyShift_threshold).

This is a 16-bit unsigned value, where higher values indicate more noise.

Note: These registers are only active in the hardware noise mitigation state.

FrequencyShiftNoiseThresholdLSB (F54_AD_Ctrl21.0)

This represents the least significant byte (LSB).

FrequencyShiftNoiseThresholdLSB (F54_AD_Ctrl21.1)

This represents the most significant byte (MSB).

15.2.25. 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 controls when the system exits the Hardware Noise Mitigation state and enters the Firmware Noise Mitigation state.

The hardware noise mitigation mechanism discards frames corrupted by noise and attempts to find a quiet sensing frequency. If the density of noise measurements that match or exceed the frequency shift noise threshold (IM >= F54_AD_FrequencyShift_threshold) divided by number of noise measurements exceeds (*Noise Density Threshold*/255) then the noise mitigation system moves into the firmware noise mitigation state.

Higher values make it easier to enter the Firmware Noise Mitigation state. The value in this register is represented as a 0.8 fixed point dimensionless value.

Note: This register is only active in the hardware noise mitigation state.

15.2.26. F54_AD_Ctrl23.0/23.1: medium noise threshold

This register is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

This firmware noise mitigation register defines the threshold that separates what are defined to be low levels of noise from medium levels of noise. This is 16-bit fixed point value, where higher values indicate more noise.

MediumNoiseThresholdLSB (F54_AD_Ctrl23.0)

This represents the least significant byte (LSB).

MediumNoiseThresholdMSB (F54_AD_Ctrl23.1)

This represents the most significant byte (MSB).

Note: These registers are only active in the Firmware Noise Mitigation state.

15.2.27. F54_AD_Ctrl24.0/24.1: high noise threshold

This register is only present if *HasFirmwareNoiseMitigation* (F54_AD_Query6, bit 3) reports as '1'.

This register defines the threshold that separates what are defined to be medium levels of noise from high levels of noise. This is a 16-bit fixed point value, where higher values indicate more noise.

HighNoiseThresholdLSB (F54_AD_Ctrl24.0)

This represents the least significant byte (LSB).

HighNoiseThresholdMSB (F54_AD_Ctrl24.1)

This represents the most significant byte (MSB).

Note: These registers are only active in the Firmware Noise Mitigation state.

15.2.28. 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 determines when the system will change frequency while in the firmware noise mitigation (FNM) state.

If the density of noise measurements that match or exceed the FNM frequency shift noise (IM >= F54_AD_Ctrl28.0/1 FNM frequency shift noise threshold) divided by number of noise measurements exceeds (*Noise Density*/255) then the noise mitigation system will shift the sensing frequency.

The value in this register is represented as a 0.8 fixed point dimensionless value.

Note: This register is only active in the firmware noise mitigation state.

15.2.29. 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 defines the number of consecutive low-noise frames (IM < medium noise threshold) that need to be observed with finger present before the system will exit the Firmware Noise

Mitigation state and reenter the Hardware Noise Mitigation state. Lower values make it easier to exit the Firmware Noise Mitigation state.

Note: This register is only active in the firmware noise mitigation state.

15.2.30. 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)

A value of 0.0 represents no IIR filtering at all. A value approaching 0.999 represents heavy IIR filtering. There is a tradeoff for this value: higher values for the filter coefficient will perform better in the presence of wideband or impulsive noise, but will increase the latency of the system in responding to finger events.

The value in this register is represented as a 0.8 fixed point dimensionless value.

Note: This register is only active in the firmware noise mitigation state.

15.2.31. F54_AD_Ctrl28.0/28.1: FNM frequency shift noise threshold

This register pair is only present if *HasIIRFilter* (F54_AD_Query8, bit 1) reports as '1'.

These registers contain an interference metric (IM) value. If the IM for a frame is above this threshold, the probability of shifting to another sensing frequency is increased. This threshold is a 16-bit fixed point value, where higher values indicate more noise.

IIRQuietThresholdLSB (F54_AD_Ctrl28.0)

This represents the least significant byte (LSB).

IIRQuietThresholdMSB (F54_AD_Ctrl28.1)

This represents the most significant byte (MSB).

15.2.32. F54_AD_Ctrl29: common-mode noise control

This register is only present if *HasCmnRemoval* (F54_AD_Query8, bit 2) reports as '1'.

Reserved (F54_AD_Ctrl29, bits 6:0)

Reserved.

CmnModeNoiseControl (F54_AD_Ctrl29, bit 7)

Writing a '1' to this bit disables all CMN removal algorithms.

15.2.33. F54_AD_Ctrl30: CMN cap scale factor

This register is present only when *HasCMNCapScaleFactor* (F54_AD_Query8, bit 3) reports as '1'.

CmnCapScaleFactor (F54_AD_Ctrl30)

This contains the percentage of pixel threshold used to cap the amount of Common-Mode noise that can be removed per row on an image sensor. It is a Q0.8 fixed-point number with a default value of 0xC0 (75% of pixel touch threshold). It has no units because it is a percentage value.

15.2.34. F54_AD_Ctrl31: pixel threshold hysteresis

This register is present only when *HasPixelThresholdHysteresis (*F54_AD_Query8, bit 4) reports as ‘1’.

PixelThresholdHysteresis (F54_AD_Ctrl31)

This register contains the amount of hysteresis as a percentage of pixel touch threshold used in identifying touches on an image sensor. It helps in minimizing finger drop-outs and achieving better linearity. This is a Q0.8 fixed-point number with a default value of 0x80 (50% of pixel touch threshold). It has no units because it is a percentage value.

15.2.35. F54_AD_Ctrl32 through F54_AD_Ctrl35: edge performance overview

Sensors designed with shorter traces along the edges may provide a lower than average capacitance reading along the edges compared to the interior parts of the sensor. To compensate for this, there are four edge compensation registers for the four edges of a sensor. Each register represents a gain factor that is applied to all the pixels on that particular edge.

The default edge gain factor is 1.0, which indicates that no gain factor is applied.

15.2.36. F54_AD_Ctrl32.0/32.1: Rx low edge compensation

This register is present only when *HasEdgeCompensation* (F54_AD_Query8, bit 5) reports as ‘1’.

This register configures the edge compensation for the receiver low edge. The register value is a gain factor applied to all the pixels on the Rx low edge. Value is an unsigned Q4.12 fixed-point number.

RxLowEdgeCompensationLSB (F54_AD_Ctrl32.0)

This represents the least significant byte (LSB).

RxLowEdgeCompensationMSB (F54_AD_Ctrl32.1)

This represents the most significant byte (MSB).

15.2.37. F54_AD_Ctrl33.0/33.1: Rx high edge compensation

This register is present only when *HasEdgeCompensation* (F54_AD_Query8, bit 5) reports as ‘1’.

This register configures the edge compensation for the receiver high edge. The register value is a gain factor applied to all the pixels on the Rx high edge. Value is an unsigned Q4.12 fixed-point number.

RxHighEdgeCompensation LSB (F54_AD_Ctrl33.0)

This represents the least significant byte (LSB).

RxHighEdgeCompensation MSB (F54_AD_Ctrl33.1)

This represents the most significant byte (MSB).

15.2.38. F54_AD_Ctrl34.0/34.1: Tx low edge compensation

This register is present only when *HasEdgeCompensation* (F54_AD_Query8, bit 5) reports as ‘1’.

This register configures the edge compensation for the transmitter low edge. The register value is a gain factor applied to all the pixels on the Tx low edge. The value is an unsigned Q4.12 fixed-point number.

TxLowEdgeCompensationLSB (F54_AD_Ctrl34.0)

This represents the least significant byte (LSB).

TxLowEdgeCompensationMSB (F54_AD_Ctrl34.1)

This represents the least significant byte (LSB).

15.2.39. F54_AD_Ctrl35.0/35.1: Tx high edge compensation

This register is present only when *HasEdgeCompensation* (F54_AD_Query8, bit 5) reports as '1'.

This register configures the edge compensation knob for the transmitter high edge. The register value is a gain factor applied to all the pixels on the Tx high edge. The value is an unsigned Q4.12 fixed-point number.

TxHighEdgeCompensationLSB (F54_AD_Ctrl35.0)

This represents the least significant byte (LSB).

TxHighEdgeCompensationLSB (F54_AD_Ctrl35.1)

This represents the least significant byte (MSB).

15.2.40. F54_AD_Ctrl36.*: axis 1 compensation

Axis1Compensation (F54_AD_Ctrl36.)*

This set of registers is present only when F54_AD_Query7, bits 1:0 read as '01' or '10'. Each register contains a unit less multiplicative scale factor in unsigned Q1.7 fixed-point format. The first register in this set represents a straight forward scale factor applied to the first pixel on Axis 1. Because each factor has a range 0.0 to 2.0, any extra compensation needed on this first pixel will be handled by the 'Edge Compensation' mechanism. Every other register in this set contains a scale factor relative to its immediate previous pixel on the same axis. The number of registers in this set is defined by F54_AD_Query0 (*RxElectrodeCount*) for dual-axis compensation mode. For single-axis compensation mode, the size is defined by the longer of the two axes (max[TxCount, RxCount]).

Note: A 'write' to this Axis Compensation register takes effect only on a 'ForceUpdate'.

Implementation Note: The assumption is that the curvature of the sensor lens changes gradually rather than abruptly from one end of the axis to the other and a Q1.7 fixed-point precision would suffice to compensate for each pixel. On a single-axis compensation enabled build (F54_AD_Query7, bits 1:0 = '01'), bit1 of F54_AD_Ctrl14 controls the axis on which this compensation is applied. In dual-axis compensation (F54_AD_Query7, bits 1:0 = '10'), Axis 1 is defined to be Rx-axis.

15.2.41. F54_AD_Ctrl37.*: axis 2 compensation

Axis2Compensation (F54_AD_Ctrl37.)*

This set of registers is present only when F54_AD_Query7, bits 1:0 read as '10'. Each register contains a unit less multiplicative scale factor in unsigned Q1.7 fixed-point format. The first register in this set represents a straight forward scale factor applied to the first pixel on Axis 2 (Tx-axis). Since each factor has a range 0.0 to 2.0, any extra compensation needed on this first pixel will be taken care of by the 'Edge Compensation' mechanism. Every other register in this set contains a scale factor relative to its immediate previous pixel on the same axis. The number of registers in this set is defined by F54_AD_Query1 (Tx Electrode count).

Note: A 'write' to this Axis Compensation register takes effect only on a 'ForceUpdate'.

Implementation Note: The assumption s that the curvature of the sensor lens changes gradually rather than abruptly from one end of the axis to the other and a Q1.7 fixed-point precision would suffice to compensate for each pixel.

15.2.42. 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.

Important: There are no user adjustments for these controls: the settings for these control mechanisms are specific to the particular sense frequency. Design Studio 4 will automatically generate the proper settings for these registers based on the contents of the sense frequency table (F54_RMI_Ctrl17*, 18*, 19*]. Users should not modify these settings. The length of each of these registers is identical to the number of entries in the sense frequency table (F54_AD_Query12, bits 3:0).

15.2.43. F54_AD_Ctrl38.*: noise control 1

This register is present if H*asFrequencySyncDithering* (F54_AD_Query8, bit 6) reports as '1'.

NoiseControl1 (F54_AD_Ctrl38.)*

This register is automatically generated by Design Studio 4 and should **not** be modified.

15.2.44. F54_AD_Ctrl39.*: noise control 2

This register is present if H*asFrequencySyncDithering* (F54_AD_Query8, bit 6) reports as '1'.

NoiseControl2 (F54_AD_Ctrl39.)*

This register is automatically generated by Design Studio 4 and should **not** be modified.

15.2.45. F54_AD_Ctrl40.*: noise control 3

This register is present if H*asFrequencySyncDithering* (F54_AD_Query8, bit 6) reports as '1'.

NoiseControl3 (F54_AD_Ctrl40.)*

This register is automatically generated by Design Studio 4 and should **not** be modified.

15.3. Function $54: data registers

Name	7	6	5	4	3	2	1	0
F54_AD_Data0	Report Type							
F54_AD_Data1	FIFO Index LSB							
F54_AD_Data2	FIFO Index MSB							
F54_AD_Data3	FIFO Data							
F54_AD_Data4	Inhibit Frequency Shift	—			Sense Frequency Selection			
F54_AD_Data5	—							
F54_AD_Data6.0	Interference Metric LSB							
F54_AD_Data6.1	Interference Metric MSB							
F54_AD_Data7.0	Current Report Rate LSB							
F54_AD_Data7.1	Current Report Rate MSB							
F54_AD_Data8	Cxy LSB							
F54_AD_Data9	Cxy MSB							
F54_AD_Data10	Pen Z							

Figure 113. Function $54 data registers

Deleted: 114

The bits of these registers are described below.

15.3.1. F54_AD_Data0: report type

ReportType (F54_AD_Data0)

Specifies the type of data that will be collected and reported in the *FIFOData* register when the *GetReport* bit (*F54_AD_Cmd0, bit 0*) is set.

ReportType = 0: Reserved

ReportType = 1: Normalized 8-Bit Image Report

Reports the capacitance variance from the baseline for each pixel location; the variance is represented by an 8-bit signed value clamped at -128 and +127. The number of bytes in the 8-bit image is:

*NumberOfTransmitterElectrodes * NumberOfReceiverElectrodes*

For a sensor with 20 transmitter electrodes and 28 receiver electrodes, the 8-bit image contains 20 rows of 28 bytes each, or 560 bytes.

ReportType = 2: Normalized 16-Bit Image Report

Reports the capacitance variance from the baseline for each pixel location; the variance is represented by a 16-bit signed value. The number of bytes in the 16-bit image is:

*NumberOfTransmitterElectrodes * NumberOfReceiverElectrodes* * 2

For a sensor with 20 transmitter electrodes and 28 receiver electrodes, the 16-bit image contains 20 rows of 56 bytes each, or 1120 bytes.

ReportType = 3: Raw 16-Bit Image Report

Reports the raw capacitance for each pixel location. For devices without attached sensors, this data can be used to find receiver-to-ground shorts.

For touch-sensor modules, it can find open transmitter electrodes, open receiver electrodes, transmitter-to-ground shorts, receiver-to-ground shorts, and transmitter-to-receiver shorts.

Each pixel's raw capacitance is represented by a 16-bit signed value. The number of bytes reported is:

*NumberOfTransmitterElectrodes * NumberOfReceiverElectrodes* * 2

For a sensor with 20 transmitter electrodes and 28 receiver electrodes, the report contains 20 rows of 56 bytes each, or 1120 bytes.

ReportType = 4: Transmitter-to-Transmitter Report

Reports a matrix of resistance between transmitter electrodes. For touch-sensor modules, this data can be used to find transmitter-to-transmitter shorts.

Each cell of the matrix is an 8-bit number: 0x00 if the resistance is high (no short exists), or 0xFF if the resistance is low (a short does exist). The number of bytes reported is:

*NumberOfTransmitterElectrodes * NumberOfTransmitterElectrodes*

For a sensor with 20 transmitter electrodes and 28 receiver electrodes, the report contains 20 rows of 20 bytes each, or 400 bytes.

ReportType = 5: Transmitter-to-Transmitter Short (No Sensor) Report

Reports a matrix of resistance between transmitter electrodes. For devices without attached sensors, this data can be used to find transmitter-to-transmitter shorts.

Each cell of the matrix is an 8-bit number: 0x00 if the resistance is high (no short exists), or 0xFF if the resistance is low (a short does exist). The number of bytes reported is:

*NumberOfTransmitterElectrodes * NumberOfTransmitterElectrodes*

For a device with 20 transmitter electrodes and 28 receiver electrodes, the report contains 20 rows of 20 bytes each, or 400 bytes.

ReportType = 6: Transmitter Open (No Sensor) Report

ReportType = 7: Receiver-to-Receiver Report

Reports a matrix of resistance between receiver electrodes. This data can be used to find receiver-to-receiver shorts.

Each cell of the matrix is an 8-bit number: 0x00 if the resistance is high (no short exists), or 0xFF if the resistance is low (a short does exist). The number of bytes reported is:

*NumberOfReceiverElectrodes * NumberOfReceiverElectrodes*

For a sensor with 20 transmitter electrodes and 28 receiver electrodes, the report contains 28 rows of 28 bytes each, or 784 bytes.

ReportType = 8: Receiver Open (No Sensor) Report

ReportType = 9: High-Resistance Report

ReportType = 10: TX_GND Report

ReportType = 11-255: Reserved

15.3.2. F54_AD_Data1 and F54_AD_Data2: FIFO index

FIFOIndexLSB (F54_AD_Data1)

This is a packet register through which the data report can be read.

FIFOIndexMSB (F54_AD_Data2)

This is a packet register through which the data report can be read.

This pair of registers is a byte-wise pointer to the image data that is being read from the FIFOData register. It is cleared to 0x0000 on reset and upon an Image Complete interrupt. In addition, writing a value into this two-byte register provides an optional way to change the position within the image buffer from which the image data is being read.

While access to the image through this FIFO facility is sequential, the ability to specify the byte index adds a certain level of random access for applications that need it.

Reading this pair of registers will return the current value of the pointer.

15.3.3. F54_AD_Data3: FIFO data

FIFOData (F54_AD_Data3)

Repeated reads of this single register provide consecutive bytes of the image data by auto-incrementing an internal pointer [optionally set through *FIFOIndexLo* and *FIFOIndexHi*]. For 16-bit data, the least significant byte is read first. The content and format of the data present in this FIFO register changes based on the contents of the *ReportMode* field (see the description of modes there).

Note: Setting *FIFOIndex* to a value that is outside the valid range for a given mode, or reading the FIFO past the end of valid data, will result in undefined behavior.

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 packet register. If reached as the result of address auto-increment during an RMI block read, this register will read 0. Subsequent reads within the block will leave the packet register pixel counter unchanged, and move on to the next element in the packet register. For more information, see section.2.5.3.

15.3.4. F54_AD_Data4: current 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)

The report which senses frequency table entry selection is active. The frame rate for that specific combination of control registers will be reported in F54_AD_Data4.

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 sense frequency to the value specified by bits 3:0.

If the new sense frequency does not match the current sense frequency when the register is written, the sense frequency change will take effect at the start of the next frame.

15.3.5. F54_AD_Data5: reserved

Reserved (F54_AD_Data5)

Reserved.

15.3.6. F54_AD_Data6.0/6.1: interference metric

This register is only present if *HasInterferenceMetric* (F54_AD_Query6, bit 1) reports as '1'.

The noise mitigation mechanisms are based on an Interference Metric (IM). The IM represents the current estimate the amount of noise present in the system. The IM is calculated for every frame of data that is acquired. The value reported in this register is the noise estimate for the most recent frame of acquired data, whether or not that frame of acquired data was presented to the finger processing firmware. Higher values indicate more noise.

InterferenceMetricLSB (F54_AD_Data6.0)

This represents the least significant byte (LSB).

InterferenceMetricMSB (F54_AD_Data6.1)

This represents the most significant byte (MSB).

15.3.7. 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'.

This register contains the instantaneous frame rate, in frames per second. This represents the least significant byte (LSB).

CurrentReportRateMSB (F54_AD_Data7.1)

This register is only present if *HasTwoByteReportRate* (F54_AD_Query6, bit 5) reports as '1'.

This register contains the instantaneous frame rate, in frames per second. This represents the most significant byte (MSB).

15.3.8. F54_AD_Data8: Cxy LSB

This register is only present if both *HasPenCurveReporting* (F54_AD_Query6, bit 5) and *HasPen* (F11_2D_Query 9, bit 0) report as '1'.

CxyLSB (F54_AD_Data8)

When a stylus is detected on the sensor, this register reports the least significant bit (LSB) of the x-axis curvature and y-axis curvature. If a stylus is not detected on the sensor, this register reports 0x0000.

15.3.9. F54_AD_Data9: Cxy MSB

This register is only present if both *HasPenCurveReporting* (F54_AD_Query6, bit 5) and *HasPen* (F11_2D_Query 9, bit 0) report as ‘1’.

CxyMSB (F54_AD_Data9)

When a stylus is detected on the sensor, this register reports the most significant bit (MSB) of the x-axis curvature and y-axis curvature. If a stylus is not detected on the sensor, this register reports 0x0000.

15.3.10. F54_AD_Data10: pen Z

This register is only present if both *HasPenCurveReporting* (F54_AD_Query6, bit 5) and *HasPen* (F11_2D_Query 9, bit 0) report as ‘1’.

PenZ (F54_AD_Data10)

When a stylus is detected on the sensor, this register reports its Z value. If a stylus is not detected on the sensor, this register reports 0x00.

15.4. Function $54: interrupt sources

The Test 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 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).

15.5. Function $54: command registers

The bits in this command register automatically clear to ‘0’ when the requested operation has completed.

Name	7	6	5	4	3	2	1	0
F54_AD_Cmd0			—			ForceUpdate	ForceCal	Get Report

Figure 114, Function $54 command register

Deleted: 115

F$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. No interrupt will be generated when the baseline is taken. If a host wants to know when the baseline operation has completed, it should poll the *ForceCal* command bit and wait for it to clear.

ForceUpdate (F54_AD_Cmd0, bit 2)

Under normal circumstances, the analog settings are read from the Function $54 control registers at startup and never changed. Changing these settings while the system is running can be very disruptive to the data being gathered. As a result, updates to the analog control registers will only be processed after the *ForceUpdate* register is set to ‘1’. The *ForceUpdate* command bit will clear once the changes are updated. The next frame of data to be acquired after the command bit clears will have been acquired using the new settings.

Reserved (F54_AD_Cmd0, bits 7:3)

Reserved.

15.6. Reading test reports with Function $54

This section describes the process for reading test reports with Function $54. When reading test reports, the following rules should be kept in mind:

- Do not read any RMI registers outside of Function $54 that are part of a coherent group. If this is done, the current test report will be corrupted and should be discarded.
- Once a test report is ready, no new report will be acquired until *GetReport (F54_AD_Cmd0, bit 0)* is again set to '1'.
- If it is important to maintain the frame rate, the next test report can be requested before the current report is completely read (although at the risk of overwriting data in the current report).

To read a test report:

1. Disable all other interrupts so that only the test reports can cause ATTN to be asserted. To do this, set the F$54 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. Select a report type by setting *ReportType (F54_AD_Data0)* to the appropriate value.
3. Request the test report by writing '1' to *GetReport (F54_AD_Cmd0, bit 0).* 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 test report is available and it may be read through the *FIFOData* register *(F54_AD_Data3).*
5. To request another test report of the same type, go to Step 3. To request a test report of another type, go to Step 2.

Reset the device to restore normal reporting mode.

16. Function $81: Device control (Private)

Function $01 defines a corresponding private Function $81.

16.1. Function $81: query registers

Function $81 defines a number of Query registers, described below:

Name	7	6	5	4	3	2	1	0
F81_RMI_Query0	—	—	—	—	—	—	—	Has FIFO Backdoor
F81_RMI_Query1	Firmware Build ID 7:0							
F81_RMI_Query2	Firmware Build ID 15:8							
F81_RMI_Query3	Firmware Build ID 23:16							

Figure 115, Function $81 query registers

Deleted: 116

16.1.1. F81_RMI_Query0: private properties

The fields in this register are defined as follows:

HasFIFOBackdoor (F81_RMI_Query0, bit 0)

If *HasFIFOBackdoor* reports as '1', registers F81_RMI_Data5 through F81_RMI_Data7 exist.

Reserved (F11_2D_Query0, bits 7:1)

Reserved.

16.1.2. F81_RMI_Query1 through F81_RMI_Query3: firmware build ID

These three registers contain the 24-bit firmware build ID, also known as the 'packrat'.

16.2. Function $81: control registers

Function $81 defines no private control registers.

16.3. Function $81: data registers

Function $81 data registers are defined as shown in Figure 116.

Name	7	6	5	4	3	2	1	0
F81_RMI_Data0.*	Int Enable 7	Int Enable 6	Int Enable 5	Int Enable 4	Int Enable 3	Int Enable 2	Int Enable 1	Int Enable 0
F81_RMI_Data1.*	Int Status 7	Int Status 6	Int Status 5	Int Status 4	Int Status 3	Int Status 2	Int Status 1	Int Status 0
F81_RMI_Data2	RAM Addr 7:6		—	—	—	—	—	XOR mode
F81_RMI_Data3	RAM Addr 15:8							
F81_RMI_Data4.0	RAM Data*N* 7:0							
F81_RMI_Data4.1	RAM Data*N* 15:8							
…	…							
F81_RMI_Data4.126*	RAM Data*N*+63 7:0							
F81_RMI_Data4.127	RAM Data*N*+63 15:8							
F81_RMI_Data5	FIFO Address 7:0							
F81_RMI_Data6	FIFO Address 15:8							
F81_RMI_Data7	FIFO Data							

Figure 116. Function $81 data registers

Deleted: 117

16.3.1. F81_RMI_Data0.*: interrupt enable

Private functions are allowed to define interrupts associated with their private data sources. In essence, there is no difference between public and private interrupts except that private interrupts are constrained to always start out in the disabled state. This ensures that a user cannot accidentally misconfigure the device to generate sources of ATTN that have not been disclosed to users. The private interrupt enable and status registers are kept separate from the corresponding public enable/status registers so that their presence or absence has no effect on the public register map. An interrupt service handler for a diagnostic-style host would have to read both the public and private Interrupt Status registers to make a full determination of what interrupts need to be serviced.

The Function $81 Interrupt Enable private register group a operates in identical fashion to the Function $01 Interrupt Enable registers (see section 4.2.2), except that the private Enable registers are defined as Data registers, not Control registers. The default value for any private Interrupt Enable registers in a device is defined to be $00 (no interrupts enabled). Interrupts from private interrupt sources are enabled by writing a '1' to the appropriate *Int Enable* bit.

Formatted: Font color: Green

Deleted: 4.2.2

16.3.1.1. Interrupt Enable register count

A product will contain zero or more Function $81 private *Interrupt Enable* registers. The number of Interrupt Enable registers implemented by an RMI device can be calculated by counting the total number of private Interrupt Sources in the device:

```
PrivateRegisterCount = trunc((NumberOfPrivateInterruptSources + 7) / 8)
```

Any private function in an RMI device that defines interrupt sources is assigned a bit or bits in a Function $81 private Interrupt Status register. If the RMI device defines more private interrupt sources than will fit in a single Interrupt Enable register, enough additional *Interrupt Status* registers are also defined by

Function $81 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.

If a device defines no private interrupt sources, there will be no *Interrupt Enable* registers present in the register map.

16.3.2. F81_RMI_Data1.*: interrupt status

The Interrupt Status register reports which of the set of possible interrupt sources are actively requesting interrupt service from the host. This register group operates in identical fashion to the Function $01 Interrupt Status registers.

16.3.2.1. Interrupt Status register count

The number of Interrupt Status registers is always identical to the number of Interrupt Enable registers. See section 16.3.1.1 for information on how to determine the number of Interrupt Enable registers.

Formatted: Font color: Green

Deleted: 16.3.1.1

16.3.3. F81_RMI_Data2 through F81_RMI_Data4: RAM back door

Function $81 provides functionality to enable a diagnostic host to directly access the touch controller register map, more commonly called a ‘RAM back door’. Access occurs in the context of a window into the RAM space. The window is always 64 ASIC memory words wide, and always starts on a 64 word boundary. A RAM operation is initiated by writing the desired window start address 15:6 into the appropriate fields contained in registers F81_RMI_Data2 and F81_RMI_Data3. Since RMI registers are only 8 bits wide, the actual window into the register space is represented by 128 F81_RMI_Data4 registers. The registers are arranged so that the least significant byte of the RAM word is located first in the RMI address space, followed by the most significant byte.

After the start address is written, reading the F81_RMI_Data4 registers will read the contents of touch controller RAM. Writing the F81_RMI_Data4 registers will write data directly to RAM. If the *XOR Mode* bit is set in F81_RMI_Data2, writes will be performed by XOR’ing the data with the current contents of the RAM.

16.3.4. F81_RMI_Data5 through 7: FIFO back door

These registers are present only if *HasFIFOBackdoor* (F81_RMI_Query0, bit 0) reports as ‘1’.

Function $81 provides an alternate, packet-style method for accessing the touch controller RAM.

FIFO Address LSB (F81_RMI_Data5)

FIFO Address MSB (F81_RMI_Data6)

This pair of registers is a bytewise pointer into the touch controller RAM. It is cleared to 0x0000 on reset but is otherwise never automatically changed. These two registers are coherent: When writing to them, both F81_RMI_Data5 and F81_RMI_Data6 must be written, in that order.

Reading this pair of registers will return the current value of the pointer.

FIFO Data (F81_RMI_Data7)

This is a packet register (see section 2.5.3). A block read or write of this single register provides consecutive bytes of the touch controller’s RAM by automatically-incrementing the internal pointer initialized by the two *FIFO Address* registers. The touch controller’s RAM is 16 bits wide; each 16-bit value is accessed LSB first, then MSB.

The *XOR Mode* bit (F81_RMI_Data2, bit 0) affects writes to this register just as it affects writes to the F81_RMI_Data4 registers.

Note: Setting *FIFO Address* to a value that is outside the valid range for the touch controller, or reading past the end of valid data, will result in undefined behavior.

Important: The contents of the packet register can only be accessed by performing a block-read or block-write operation which begins at the packet register's address. If the packet register is not addressed directly, writes to it will have no effect and reads from it will return only a single 0x00 byte.

16.4. Function $81: interrupt source

RMI Function $81 defines no interrupt sources.

16.5. Function $81: command registers

Function $81 defines no command registers.

17. Function $84: Analog Diagnostic for T1021 (Private)

Function $84 implements controls for the core analog diagnostic functions of the hardware and the firmware without requiring direct access to the control registers as RAM. It maintains backward compatibility with many of the OneTouch control registers as well as many of the functions of “nuke bits.” These include such attributes as the XRefHi and XRefLo registers, Guard configuration, and RC-Oscillator frequency, but in a more easily used and safer manner.

The per-electrode Response Correction Factor is controlled within Analog Diagnostic, as an electrode may be shared by more than one function (for example, one electrode for both buttons and strips). Certain core algorithm variables can also be controlled, such as servoing, relaxation, and zeroing. Reporting of ADCs is also available for basic testing.

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

- Controls for algorithms, such as spatial filtering for strips that are only present in functions for certain devices, are implemented in their own function.
- Scaling Sensitivities (not Response Correction Factors) for the electrodes within a function are controlled by that function.
- Any Z reporting (for example, Zx, Zdelta, or Zabs) may be implemented within a function or in Analog Diagnostic.
- Fine-grain control of zeroing or relaxation is controlled within a particular function as the Analog Diagnostic function only implements those features that are expected to appear in every diagnostic function.

Note: See the BIST Function $08 for higher-level and more automated testing functions.

All devices should support basic analog diagnostic controls. More advanced controls may be implemented selectively, or implemented by directly reading, writing, or modifying words in the Hardware Register Window. The query registers and relevant control registers should be read before modifying the control or command registers, and before attempting to read data registers.

All registers of the Analog Diagnostic function are considered private unless they specifically need to be released to a customer. An unlock sequence (or bit) has been provided to prevent any spurious write to the control or command registers of the Analog Diagnostic function, which could result in incorrect data and difficult-to-diagnose problems.

17.1. Function $84: query registers

The query registers describe the available electrodes, modes, and function capabilities available in this product.

Name	7	6	5	4	3	2	1	0
F84_AD_Query0	—	—	—	NumberofElectrodesAvailable				
F84_AD_Query1	HasFullSST	HasIM	—	—	HasZRep	HasXADCRep	HasADCRep	HasADCCtrl
F84_AD_Query2	—	—	—	—	—	—	—	—
F84_AD_Query3	—	Number of Reported Hardware Registers (2 byte words)						
F84_AD_Query4	Reference Capacitor Calibration							

Figure 117. Function $84 query registers

Deleted: 118

The bits of these registers are defined as follows:

NumberOfElectrodes (F84_AD_Query0, bits 4:0)
This 5-bit Field reports the number of electrodes available on the design. For future touch controllers with more than 31 sensors, the field can be expanded. The number and location of data registers within Function $84 is computed from *NumberOfElectrodes*.

HasADCCtrl (F84_AD_Query1, bit 0)
If '1', then Analog Diagnostic Control is available. If '0', then no writable registers are available within Function $84. In this case, query, data, and control registers may still be read.

HasADCReporting (F84_AD_Query1, bit 1)
This field reports the standard available Per-Channel ADC reporting mode in Function $84's Per-Channel Reporting data registers *F84_AD_Data4.**:

- '0': No ADC (or other) reporting is supported.
- '1': Basic ADC mode where all bits of epoch accumulated ADC channels are reported.

HasXtendedADCReporting (F84_AD_Query1, bit 2)
This field reports the available extended ADC reporting mode in Function $84:

- '0': No extended ADC reporting is supported.
- '1': A subset of bits or channels of ADCs are reported in an Extended ADC mode.

HasZReporting (F84_AD_Query1, bit 3)
This field reports the available Z reporting mode in Function $84:

- '0': No Z reporting is supported.
- '1': Electrode Zs are reported each epoch.

HasIM (F84_AD_Query1, bit 6)
This 1-bit field specifies availability of a universal (rather than per function) interference metric, including the metric's presence in the data and its threshold in the control registers.

HasFullSST (F84_AD_Query1, bit 7)

This 1-bit field specifies the format of SST control. If ‘1’, full SST control allows 2-bit control of each channel’s SST caps as on T1021 touch controllers. If ‘0’, only reduced control of a single SST per ADC is allowed, as on T1007 touch controllers.

ExtraQueryBits (F84_AD_Query2)

This register is a placeholder for the extra query bits requested by Firmware.

NumberOfReportedHardwareRegisters (F84_AD_Query3)

This 7-bit field indicates the number of Hardware Register words mapped in *F84_AD_Data3* to byte registers. Typically, for T1021 devices this will be 19, corresponding to $FF87-$FF99, which corresponds to 38 bytes in Function $84.

ReferenceCapacitorCalibration (F84_AD_Query2)

This register is a placeholder for calibration of the reference capacitors at test time.

17.2. Function $84: control registers

The control registers provide a mechanism for configuring the Analog Diagnostic registers and algorithms. They also control the reporting and configuration of the data and command registers.

Name	7	6	5	4	3	2	1	0
F84_AD_Ctrl0	NoAutoServo	NoAutoCal	NoAutoZero	NoRelax	NoAutoGS	NoIFMPunt	NoFreqMod	NoSHLDGND
F84_AD_Ctrl1	—	—	—	—	—	—	—	NukeOnReset
F84_AD_Ctrl2	—							
F84_AD_Ctrl3	—							
F84_AD_Ctrl4	Interference Metric Threshold Lo Byte							
F84_AD_Ctrl5	Interference Metric Threshold Hi Byte							
F84_AD_Ctrl6	—							

Figure 118: Function $84 control registers

Deleted: 119

The bits of these registers are defined as follows:

NoSHLDGND(F84_AD_Ctrl0, bit 0)
This bit sets the A15 IO to a floating input so that when the Guard Mode is set to 000, the EnSHLDPin bit is set. At reset this bit will be ignored unless the *NukeOnReset* bit is set.

NoFrequencyModulaton(F84_AD_Ctrl0, bit 1)
This bit sets the low 4 bits of RC-Oscillator to '0' and sets the low gear bit to '0'. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoIFMetricPunt (F84_AD_Ctrl0, bit 2)
This bit disables the dropping of reported data to other sensing algorithms when an interference metric threshold is exceeded. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoAutoGS (F84_AD_Ctrl0, bit 3)
This bit disables shifting sensing gears when an interference metric threshold is exceeded. This may be overloaded to disable dropping of reported epoch data to other sensing algorithms, if the *NoIFMetricPunt* bit is unimplemented.

This field can also be overloaded to set the low 4 bits of RC-Oscillator and the LSB of the Gear to '0', and disable frequency modulation, if *NoFrequencyModulation* is unimplemented. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoRelax (F84_AD_Ctrl0, bit 4)
This bit disables relaxation algorithms for all channels. This field is ORed with any per-function relaxation control when set, and this field is ANDed when cleared. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoAutoZero (F84_AD_Ctrl0, bit 5)
This bit disables auto-zeroing (lift recovery) algorithms for all channels. This field is ORed with any per-function re-zero control when set, and ANDed in when reset. Clearing the bit should not cause an auto-zero event. This field may be overloaded to disable relaxation algorithms for all channels, if *NoRelax* is unimplemented. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoAutoCalibration (F84_AD _ Ctrl0, bit 6)

This bit disables the auto-sensitivity correction algorithms for all channels, which updates the Sensitivity Correction registers at the time when they would be initiated (for example, at servo time) and should not terminate calibration after it has begun. Clearing this bit should not cause a calibration event. At reset this, bit is ignored unless the *NukeOnReset* bit is set.

NoAutoServo (F84_AD _ Ctrl0, bit 7)

This bit disables auto-servoing (for example, channel clip recovery) algorithms at the time when they would be initiated and should not terminate calibration after it has begun. This bit should be set when changing RefCap registers. Clearing this bit should not cause a servo event. This bit may be overloaded to disable the auto-sensitivity correction algorithms for all channels, which updates the Sensitivity Correction Factor registers, if NoAutoCalibration is unimplemented. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NukeOnReset (F84_AD _Ctrl1, bit 0)

This diagnostic bit enables all other nuke bits on Reset events.

ReservedNukeBits (F84_AD _Ctrl1, bits 1:7)

This byte register is reserved for future Analog Diagnostic control bits to disable or set parameters to diagnostic defaults.

ReservedServoFootRoom (F84_AD_Ctrl2) – Unimplemented

This field controls the device's Foot Room parameter at servoing time. This field is set by Firmware (for example, 100ADC).

ReservedServoHeadRoom (F84_AD_Ctrl3) – Unimplemented

This field controls the device's Head Room parameter at servoing time. This field is set by Firmware (for example, 2pF).

InterferenceMetricThresholdLo (F84_AD _Ctrl4)

This field sets the Lo bits of threshold for the main interference metric. Any other interference metric thresholds will be set in the relevant function. This field is present if *HasIM (F84_AD_Query1, bit 6)* is true. Sensing should be disabled or ignored (for example, before POR) when changing threshold.

InterferenceMetricThresholdHi (F84_AD _Ctrl5)

This field sets the Hi bits of threshold for the main interference metric. Any other interference metric thresholds will be set in the relevant function. This field is present if *HasIM (F84_AD_Query1, bit 6)* is true. Sensing should be disabled or ignored (for example, before POR) when changing threshold.

ReservedEpochLength (F84_AD_Ctrl6) – Unimplemented

This field specifies the length of the epoch length in ADC conversions. This does not directly control timing as Gear, RCOsc Frequency, and selected ADC CHNs can also alter different parameters controlling this. This field is determined by Firmware implementation.

17.3. Function $84: data registers

Function $84's data registers are allocated as available and controlled by *NumberOfElectrodes (F84_AD_Query0, bits 4:0)* as well as *NumberOfReportedHardwareRegisters (F84_AD_ Query3, bits 5:0)*. Hardware Registers must be read and written as coherent pairs of bytes.

Reported values may include the base interference metric register, if enabled, and may include enough reporting bytes for some or all ADCs, if per channel reporting is available. ADC Data should be read coherently. In alternate (extended) ADC reporting modes, only a smaller number of bytes may be required, instead of the standard two bytes per electrode.

Data interrupts do not automatically occur as data changes, but some modes (for example, RepIRQ) may be set such that a data interrupt is set every epoch, or only on epochs when they change (for example, *ServoIRQ*). These require their respective interrupt enable bits to be set in the Private Interrupt Control register of Function $81. The cause of the interrupt is then indicated in the corresponding bit of the first data register.

Name	7	6	5	4	3	2	1	0
F84_AD_Data0	—	—	—	—	—	—	IsServoing	—
F84_AD_Data1	—	—	—	—	EnZRep	EnXADCRep	EnADCRep	EnADCtrl
F84_AD_Data2.*	Enabled Electrode Response to SST or Sensitivity Correction Factors							
F84_AD_Data3.*	Hardware Register Window Lo Byte							
F84_AD_Data3.*	Hardware Register Window Hi Byte							
F84_AD_Data4.*	Per Channel Report Lo Byte							
F84_AD_Data4.*	Per Channel Report Hi Byte							
F84_AD_Data5	Interference Metric Lo Byte							
F84_AD_Data6	Current Gear			—	Interference Metric Hi Byte			

Figure 119. Function $84 data registers

Deleted: 120

The bits of these registers are defined as follows:

ReservedAnalogSensingControlDelayed (F84_AD_Data0, bit 0) – Unimplemented

This field indicates that a write to the Analog Diagnostic control registers has not taken place. '0': Indicates that all writes are fully completed, while '1': Indicates that some write of the control registers has not completed.

IsServoing (F84_AD_Data0, bit 1)

It is useful to know when the device has entered the servoing function and when it has left it. This bit should be set as the device enters and before any change to the RefCaps have been made. This bit should not clear until RefCaps and Electrode Correction Factors have been determined. Because the *ServoIRQ* bit will be set (if enabled) only after servoing has completed, it is important to read *IsServoing* after setting *NoAutoServo*. The *ServoIRQ* bit negates the need for a *HasServoed* bit.

ReservedResetCondition (F84_AD_Data0, bits 5:2) – Unimplemented

This 4-bit field reports the value of the RCOND in the reset condition register. In the case where a reset has occurred, this field indicates the cause of the reset (for example, POR).

ReservedWasSleeping (F84_AD_Data0, bits 7:6) – Unimplemented

This 2-bit field reports the state of the sleep control register just before the register read operation. This field should be the very first register read for this functionality to work.

EnableADCtrl (F84_AD_ Data1, bit 0)

Controls writeability within the Analog Diagnostic control and command registers. If this bit is not set, then no writable registers, including Hardware Register-backed data, are available within Function $84, although register F84_AD_Data1 still can be altered. Query, data, command, and control registers may still be read.

When this bit is set, any automatic refreshing of Analog Control registers is disabled, and those registers may be written to. Setting the bit does not inhibit any automatic updates of the register values (for example, Reflo servoing). Resetting this bit does not cause defaults to be reloaded into the control registers.

EnPerChanReporting (F84_ AD_ Data1, bits 3:1)

This 3-bit field controls the per-channel reporting modes. For future touch controllers with more than 3 modes, the field can be expanded. The low bit 0 corresponds to a basic ADC reporting mode(as described) while bit 1 corresponds to an alternate mode, and the high bit 2 corresponds to Z reporting:

‘000’: No Per Channel reporting is supported.
‘001’: Basic mode where all bits of epoch accumulated ADC channels are reported.
‘010’: Extended ADC mode where fewer bits (or channels) are reported.
‘011’: Reserved (Invalid state).
‘100’: Extended Z reporting mode.
‘101’: Reserved (Invalid state).
‘110’: Reserved (Invalid state).
‘111’: Reserved (Invalid state).

If no bits are set then ADC reporting is disabled in Function $84. Only modes indicated by the related query are valid. No coherent reads outside of Function $84 should be made while any reporting bits are set so that the coherency buffer can be loaded whenever the data is made available. If other coherent registers must be read, they should be read immediately after the Per Channel Reporting registers to reduce the chance of collision, and the last register should be read to ensure collected per-channel values are valid.

The reporting is done through the Per Channel Reporting data registers *F84_AD_Data4.**. The per-channel reports will not be updated until the corresponding interrupt data register in Function $81 is read, and the interrupt cleared.

SensitivityCorrection (F84_AD _Data2.)*

This field contains each of the Response Correction Factors in one byte, based on the response of each channel to SST. This field may also be adjusted by writing to the data register. The length of this sequence of replicated registers is determined by the value of *NumberOfElectrodes*. These replicated registers should be written or read as a coherent block.

HardwareRegisterWindow (F84_AD _Data3.)*

This contains a window onto the hardware registers of the device (in words) *NumberOfReportedHardwareRegisters (F84_AD_ Query3, bits 6:0)* long which will typically be 19 register words from $FF87 to $FF99 in the T1021corresponding to 38 bytes in Function $84.These hardware registers must always be read and written coherently in pairs as a single word or as a group of words, each with a low and high byte.

PerChannelRep (F84_AD_Data4.)*

Typically, reports a full word (low byte first and high byte second) of the ADC epoch accumulated sum for each available electrode. If an alternate reporting mode is set, then an alternate (for example, shortened or compressed) Extended ADC (or Z) reporting structure may be used (for example, only 12 bits per conversion, 1 byte per conversion, a subset of electrodes, etc.). All Per Channel values determined by *NumberOfElectrodes* should be read coherently. Each is present if the corresponding bit of *HasReportingModes (F84_AD_Query1, bits 3:1)* is true.

When Per Channel Reports are turned on, then no other coherent reads should take place outside of Function $84 until the data has been read or after the PerChanIRQ has been cleared by reading. Typically, this means that all other public and private interrupts should be disabled. The Per Channel reports, in turn, will not be updated until the interrupt data register in $81is read, and the interrupt cleared. When the interrupt has been cleared, but no new data is yet available then all registers will report $FFFF, which is invalid.

As shown in the figure below, a set of the available ADC channels are accumulated every epoch into an array of active Sensors (S0:Sn). The availability of the channel is determined by the package, and the number of Sensors determined by the device configuration (in the Hex File).

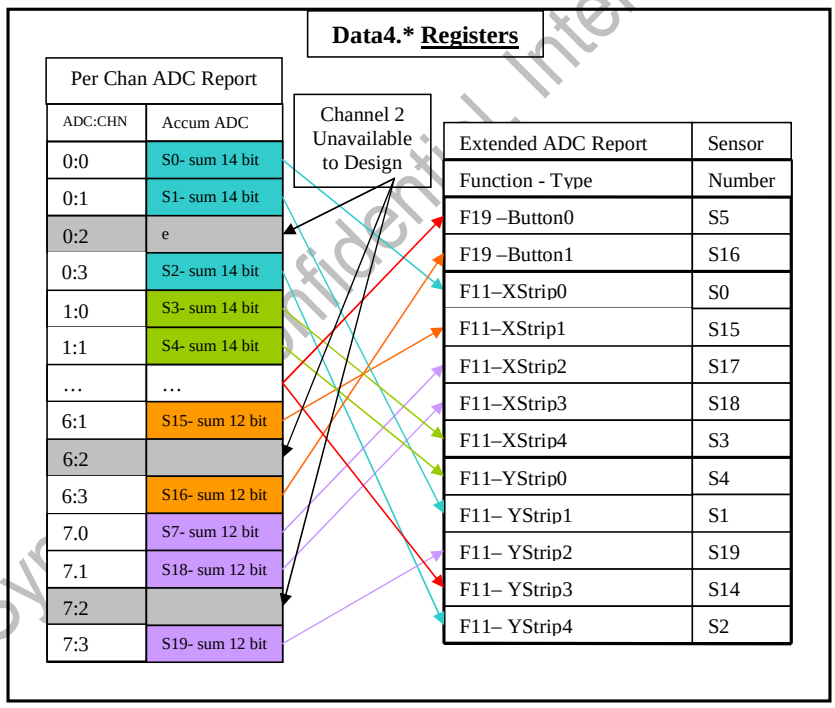

Figure 120 Function $84 Data4. registers*

Deleted: 121

The sensors are in turn mapped by the functions into a sensor order that determines button or strip order. The Per-Channel Reporting order is determined by the functional type and the order assigned within the function.

InterferenceMetricLoByte (F84_AD_Data5)
This byte reports the low byte of the interference metric used for comparison of the InterferenceMetric Threshold for Gear Shifting and Epoch report dropping. This field is present if *HasIM (F84_SA_Query1, bit 6)* is true. Read this register coherent with the following register.

InterferenceMetricHiByte (F84_AD_Data6, bits 0:3)
This 4-bit field reports the high bits of the interference metric used for comparison of the InterferenceMetric Threshold for Gear Shifting and Epoch report dropping. This field is present if *HasIM (F84_SA_Query1, bit 6)* is true. Read this register coherent with the preceding register.

ReservedIMFail (F84_AD_Data6, bit 4) – Unimplemented
This bit reports whether the last epoch Interference Metric was above threshold.

CurrentGear (F84_AD_Data7, bits 7:5)
The top 3 bits of the interference metric high byte are used to indicate the current gear. Where more than one Interference Metric is reported, all will report the same Gear bits.

17.4. Function $84: interrupt source

The Analog Diagnostic Data source can assert an interrupt request at the end of every report period (epoch) for which an interrupt source has been enabled. There are two interrupt sources for Function $84. The interrupts will be enabled and reported in Function $81 with *EnRepIRQ* as the low bit, and *EnServoIRQ* as high.

If the *EnRepIRQ* bit has been set, per channel reporting will interrupt on every epoch when new Per Channel Data is available. It is cleared when the corresponding interrupt data register is clear by reading. Until the interrupt is cleared no new data will be collected. The words of the Per Channel Data registers will report $FFFF until new data is available.

If the *EnServoIRQ* bit is set, a *ServoIRQ* interrupt is set at the end of servoing. It is cleared when the corresponding interrupt data register is read. The IsServoing bit will accurately indicate that servoing has begun, but no interrupt will occur until the next epoch is reported after servoing has completed and the IsServoing bit has cleared.

Even when Function $84 does not assert any interrupts, all data except for the Per Channel Reporting should be available on any polling read and updated every epoch.

17.5. Function $84: command registers

The control registers provide a mechanism for issuing commands for the analog diagnostic system. The commands take varying periods of time and their bits only clear as they finish. The order of the bits has been chosen so that those taking the longest or causing the most severe effect are executed in the lowest bits.

Name	7	6	5	4	3	2	1	0
F84_AD_Cmd0	—	—	—	—	—	ReCalSens	Reservo	—

Figure 121. Function $84 command registers

Deleted: 122

The bits of this register are defined as follows:

ReServo (F84_AD_Cmd0, bit 1)
This bit causes a servoing event in the analog diagnostic system. It is used when the *NoAutoServo* bit is set. It should also cause a re-zero event for each supported function.

ReCalibrateSensor (F84_AD_Cmd0, bit 2)
This bit causes the analog diagnostic system to recalibrate the response of each active channel to the SST capacitors, repopulating the F84_AD_Data1.* registers. This may be used when the *NoAutoCal* bit is set. It should also cause a re-zero event for each supported function.

ReservedGearShift (F84_AD_Cmd0, bit 5) – Unimplemented
This bit forces the gear to shift to the next available gear. It may be used when the *NoAutoGS* bit is set.

ReservedStartADCReporting (F84_AD_Cmd0, bit 7) – Unimplemented
This bit starts ADC reporting if an ADC reporting mode is set. It may cause epoch to stall at the ADC accumulation step. This bit clears when data is first available, but need not set its own interrupt as *EnADCIRQ* can. Optional if ADC data is always (or never) available.

ReservedRestartEpoch (F84_AD_Cmd0, bit 7) – Unimplemented
This bit starts a new epoch of appropriate length after the *AnalogSensingControlDelayed* bit has cleared. The command bit clears at the end of the restarted epoch to indicate all updates to the control registers have completed. It can also be used to guarantee that any information in the data registers is fresh. When the RestartEpoch bit is set, no interrupt should be set until the bit clears and fresh data is available. This function will be implemented if necessary for synchronization.

17.6. Function $84: hardware register window

This section describes the registers available in the Hardware Register Window for T1021 devices. These can be used as a guide, but the latest Reference Manual version takes precedence. The words are reported as low byte and then high byte within the respective RMI register bytes.

***Important*:** Hardware registers must be written in pairs, starting with the RMI register corresponding to the low half of the CPU hardware register, followed by the RMI register corresponding to the high half of the CPU hardware register. The write to the CPU register only occurs after receiving the high half of the CPU hardware register.

This information is an example. For descriptions of the Memory Map contents, access the *T1021 Reference Manual* (PN: 511-000291-01).

Address	Register	7	6	5	4	3	2	1	0	Reset Value
Data3.0	DDACLo	DDAC								$00
Data3.1	DDACHi	DDAC								$00
Data3.2	FCTL Lo	FLIPCTL								$00
Data3.3	FCTL Hi	FLIPCTL								$00
Data3.4	ACTL Lo	ALTCTL								$00
Data3.5	ACTL Hi	ALTCTL								$00
Data3.6	SCTL Lo	ALLSHORT	ALL1AXIS	YREFEN	XREFEN	YSENSE	XSENSE	YADCEN	XADCEN	$00
Data3.7	SCTL Hi	—	—	—	—	AEN	BOFF	FILTLEN[1:0]		$00
Data3.8	GCTLLo	ELMODE[1:0]		GS20	GS10	GS_FINE	NEXTSAMP	NEXTCHN[1:0]		$00
Data3.9	GCTL Hi	—	—	—	—	GUARDPIN	GUARDMODE[2:0]			$00
Data3.10	TCTL Lo	DHOLD	DDRV	PSTR	PSEL[4:0]					$00
Data3.11	TCTL Hi	—	—	—	—	YPERCHAN	XPERCHAN	BIGREF	SHRESET	$00
Data3.12	OCTLLo	OSC[3:0]				OSCMOD[3:0]				$00
Data3.13	OCTLHi	—				CDIV	FASTMODE	OSC[5:4]		$00
Data3.14	XREFLo	XREFHI[1:0]		XREFLO[5:0]						$00
Data3.15	XREFHi	—	—	—	—	XREFHI[5:2]				$00
Data3.16	YREFLo	YREFHI[1:0]		YREFLO[5:0]						$00
Data3.17	YREFHi	—	—	—	—	YREFHI[5:2]				$00
Data3.18	XTRIM0	XTRIM12		XTRIM8		XTRIM4		XTRIM0		$00
Data3.19	**BLANK**	—								$00
Data3.20	YTRIM0	R	R	YTRIM8		YTRIM4		YTRIM0		$00
Data3.21	**BLANK**	—								$00
Data3.22	XTRIM1	XTRIM13		XTRIM9		XTRIM5		XTRIM1		$00
Data3.23	**BLANK**	—								$00
Data3.24	YTRIM1	R	R	YTRIM9		YTRIM5		YTRIM1		$00
Data3.25	**BLANK**	—								$00
Data3.26	XTRIM2	XTRIM14		XTRIM10		XTRIM6		XTRIM2		$00
Data3.27	**BLANK**	—								$00
Data3.28	YTRIM2	STOPMEM	GUARDSTOP	YTRIM10		YTRIM6		YTIRM2		$00
Data3.29	**BLANK**	—								$00
Data3.30	XTRIM3	XTRIM15		XTRIM11		XTRIM7		XTRIM3		$00
Data3.31	**BLANK**	—								$00
Data3.32	YTRIM3	STOPMODE[1:0]		YTRIM11		YTRIM7		YTRIM3		$00
Data3.33	**BLANK**	—								$00
Data3.34	MISCLo	0	CVI	—	—	LFSR_UNLK	WDT_CLR	CLR_CVI	CVI	$30
Data3.35	MISCHi	0	0	0	0	1	1	0	0	$0C
Data3.36	ANSTAT	MASKREV_LO[2:0] = 010 for T1021A3			FLIPPED	STOPPED	DCMP	RCHN[1:0]		$40
Data3.37	**BLANK**	—								$00

Figure 122

Deleted: *123*

17.7. Reading profiles with Function $84

This section describes the process for reading RMI profiles. When reading profiles, keep in mind the following rules:

- When profile reporting is enabled you must not read any RMI registers outside of Function $84 that are part of a coherent group. If you do, the captured profile snapshot will be corrupted and should be discarded.
- After a profile snapshot has been captured no new snapshot will be acquired until you service the Analog interrupt by reading F81_RMI_Data1.n.
- If the reporting mode of F84_AD_Data1 is changed, it is best to ignore the first snapshot following the change, in case it still contains data related to the previous mode.

To read a profile, follow the instructions below.

1. Disable low-power/dozing by setting the *NoSleep* bit of F01_RMI_Ctrl0.
2. Although not required, it is strongly suggested to disable all public interrupts so that only the ADC reports can cause ATTN to be asserted. To do this, write 0x00 to F01_RMI_Ctrl1.*n*.

   Note: *n* indicates that it is possible for there to be more than one Interrupt Enable register, in which case 0x00 must be written to all of them.
3. Unlock private functions via two consecutive writes to Device Status:
   - Write 0x42 to F01_RMI_Data0
   - Write 0xE1 to F01_RMI_Data0
4. If you have not already parsed the private PDTs, do so now in order to calculate the bitmask of Function 0x84's Analog interrupt for use in the next step.
5. Enable the private Analog interrupt by writing its bitmask that was discovered above into the correct private Interrupt Enable. To do this, write "bitmask" to F81_RMI_Data0.*n*:
   - If bitmask is < 0x100, write it to F81_RMI_Data0.0
   - If bitmask is 0x100..0x8000, shift it right by 8 and write it toF81_RMI_Data0.1

   Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), skip this step.
6. Enable ADC reporting by setting both EnADCtrl and ENADCRep in F84_AD_Data1.
7. Assuming the Analog interrupt was enabled above, ATTN will be asserted when a profile snapshot is available. The profile data can then be read from registers F84_AD_Data4.n.

   Note: All of the F84_AD_Data4.n registers will return 0xFF until there is new profile data. If you wish to ignore ATTN and poll (not recommended), poll an odd-numbered F84_AD_Data4.n register (for example, F84_AD_Data4.1 or F84_AD_Data4.3) until it reports a value other than 0xFF. It is important to poll an odd-numbered register because even-numbered registers map to the LSB of the image data and 0xFF is valid for LSB data.
8. When you are ready for the next profile snapshot you must clear the Analog interrupt by reading the private Interrupt Status register(s) in Function $81, F81_RMI_Data1.n.

   Note: This requires setting the Page Select register to Function $81's page, reading F81_RMI_Data1.n, and then setting the Page Select back to Function $84's page.
9. To get back to normal mode:
   - Exit profile mode by writing 0x00 to F84_AD_Data1.
   - Exit private function mode set the Configured bit in F01_RMI_Ctrl0.
   - Re-enable the desired public interrupts by writing to F01_RMI_Ctrl1.n.
   - Or reset the device.

18. Function $85: Analog Diagnostic for T1320 (Private)

Function $85 provides private analog diagnostic functions that allow visibility and control of the base analog functionality used for trans-capacitive image sensing.

In general, it is the intent of the design on the F$85 diagnostics to provide information about and control of the operation of the touch controller and core algorithms without impeding the normal operation of other functions.

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

- Controls for algorithms, such as spatial filtering for strips that are only present in functions for certain devices, are implemented in their own function.
- Scaling Sensitivities (not Response Correction Factors) for the electrodes within a function are controlled by that function.
- Any per-channel Z reporting (for example, Zdelta, or Zabs) may be implemented within Analog Diagnostics while function specific (such as Zx, Zy) will be implemented in that function.
- Fine-grain control of zeroing or relaxation is controlled within a particular function, the Analog Diagnostic function only implements those features that are expected to appear in every diagnostic function, although it may contain global controls that affect all functions, but do not override the fine-grain control.

Note: See the BIST Function $09 for higher-level and more automated testing functions.

All devices should support basic analog diagnostic controls. More advanced controls may be implemented selectively, or implemented by directly reading, writing, or modifying words in the Hardware Register Window. In general, the intent of F$85 is to provide a means to coherently read the ADC data in its various forms in synchronization with the operational firmware.

F$85 provides an interface to the sensing hardware based on trans-capacitive sensing with drive electrodes and sense electrodes. It does not map that to an X, Y coordinate space.

All registers of the Analog Diagnostic function are considered private unless they specifically need to be released to a customer. An unlock sequence (or bit) has been provided to prevent any spurious write to the control or command registers of the Analog Diagnostic function, which could result in incorrect data and difficult-to-diagnose problems.

18.1. Function $85: query registers

The query registers describe the available electrodes, modes, and function capabilities available in this product.

Name	7	6	5	4	3	2	1	0
F85_AD_Query0	—	—	NumberOfSenseElectrodesAvailable					
F85_AD_Query1	—	—	NumberOfDriveElectrodesAvailable					
F85_AD_Query2	—	—	Has AutoServo	HasBitmap	HasImage	HasBaseLine	HasADCRaw	HasADCtrl
F85_AD_Query3	HasEnhanced F85	HasIM2	—	—	—	—	—	—
F85_AD_Query4	SizeOfF85RegisterWindow (bytes)							
F11_2D_Query5	ReferenceCapacitorCalibration							
F85_AD_Query6	—	—	—	—	NumberOfSenseFrequencies			

Figure 123. Function $85 query registers

Deleted: 124

The bits of these registers are defined as follows:

NumberOfSenseElectrodesAvailable (F85_AD_Query0, bits 5:0)
This 6-bit field reports the number of sense electrodes available on the design. For future touch controllers with more than 31 sense electrodes, the field can be expanded.

NumberOfDriveElectrodesAvailable (F85_AD_Query1, bits 5:0)
This 6-bit field reports the number of drive electrodes available on the design. For future touch controllers with more than 31 drive electrodes, the field can be expanded.

HasADCtrl (F85_AD_Query2, bit 0)
If ‘1’, then Analog Diagnostic Control is available. If ‘0’, then no writable registers are available within Function $85. In this case, query, data, and control registers may still be read.

HasADCRaw (F85_AD_Query2, bit 1)
This field indicates the availability of the ADC Raw reporting mode. This report is made through the data registers F85_AD_Data2.*:

- ‘0’: ADC Raw reporting is not supported.
- ‘1’: ADC Raw is supported.

HasBaseline (F85_AD_Query2, bit 2)
This field indicates the availability of the Baseline reporting mode. This report is made through the data registers F85_AD_Data2.*:

- ‘0’: Baseline reporting is not supported.
- ‘1’: Baseline reporting is supported.

HasImage (F85_AD_Query2, bit 3)

This field indicates the availability of the Image Line reporting mode. This report is made through the data registers F85_AD_Data2.*:

- '0': Image Line reporting is not supported.
- '1': Image Line reporting is supported.

HasBitmap (F85_AD_Query2, bit 4)

This field indicates the availability of the Bitmap Image reporting mode. This report is made through the data registers F85_AD_Data2.*:

- '0': Bit Image reporting is not supported.
- '1': Bit Image reporting is supported.

Note: In the current implementation, *HasBitmap* is never set.

HasAutoServo (F85_AD_Query2, bit 5)

This field indicates that the firmware supports auto-servo operations:

- '0': Autoservo is not supported.
- '1': Autoservo is supported.

HasEnhancedF85 (F85_AD_Query3, bit 7)

This field specifies availability of Query6 and the Data3 through Data9 registers:

- '0': The extra registers do not exist.
- '1': The Query6 and the Data3 through Data9 registers are used.

HasIM2 (F85_AD_Query3, bit 6)

This field specifies availability of a secondary Interference Metric and of the Ctrl6, Ctrl7, Data10, and Data11 registers::

- '0': The extra registers do not exist.
- '1': The secondary Interference Metric is available and the Ctrl6, Ctrl7, Data10, and Data11 registers are used.

SizeOfF85RegisterWindow (F85_AD_Query4)

This field indicates the number of register bytes mapped in the F85 register window – F85_AD_Data1. The F85_AD_ Data2 register window is the interface that Function $85 uses to provide access to hardware registers, ADC report data, and image report data in various formats. This window will be the larger of the hardware register window size or the size of a baseline image line.

An example: T1320 devices configured with the maximal 20 sense electrodes and 28 drive electrodes has an image line size of 20 words (40 bytes) and a hardware window size of 52 bytes, (corresponding to addresses $FF20-$FF39), so the value of this query register and the size of the F85_AD_ Data2 register window will be 52_{10}.

ReferenceCapacitorCalibration (F85_AD_Query5)

This register is a placeholder for calibration of the reference capacitors at test time.

NumberOfSenseFrequencies (F85_AD_Query6, bits 3:0)

Specifies the number (-1) of available analog sensing frequencies.

Note: This register only exists if *HasEnhancedF85* (F85_AD_Query3, bit 7) is set to 1.

18.2. Function $85: control registers

The control registers provide a mechanism for configuring the Analog Diagnostic registers and algorithms. They also control the reporting and configuration of the data and command registers.

Name	7	6	5	4	3	2	1	0
F85_AD_Ctrl0	NoAutoServo	NoOscMod	NoAutoZero	NoRelax	NoScan	NoBleedover	NoFreqShift	NoCMNR
F85_AD_Ctrl1	NoPosCorr	—	—	—	—	NoSpatial Filter	Spew	NukeOnReset
F85_AD_Ctrl2	—	—	—	—	—	BurstsPerCluster		
F85_AD_Ctrl3	InterferenceMetricThresholdLo							
F85_AD_Ctrl4	InterferenceMetricThresholdHi							
F85_AD_Ctrl5	Bitmap Threshold							
F85_AD_Ctrl6	InterferenceMetric2ThresholdLo							
F85_AD_Ctrl7	InterferenceMetric2ThresholdHi							

Figure 124: Function $85 control registers

Deleted: 125

The bits of these registers are defined as follows:

NoCMNR (F85_AD_Ctrl0, bit 0)
This bit disables the common-mode noise reduction algorithms for all channels. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoFreqShift (F85_AD_Ctrl0, bit 1)
This bit disables automatic shifting of the analog sensing frequency. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoBleedover (F85_AD_Ctrl0, bit 2)
This bit disables the LPF Bleedover algorithm. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoScan(F85_AD_Ctrl0, bit 3)
This bit causes the image acquisition code to stop writing to the XMTR_INP0, XMTR_INP12 and XMTR_INP24 registers, which allows unfettered access to the diagnostic host. During normal operation, the RCVR_EN, XMTR_EN, and XMTR_POL registers are static and do not over-write the image acquisition, so this bit does not change the behavior of those registers.

NoRelax (F85_AD_Ctrl0, bit 4)
This bit disables relaxation algorithms for all channels. This field is ORed with any per-function relaxation control when set, and this field is ANDed when cleared. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoAutoZero (F85_AD_Ctrl0, bit 5)
This bit disables auto-zeroing (lift recovery) algorithms for all channels. This field is ORed with any per-function re-zero control when set and ANDed in when reset. Clearing the bit should not cause an auto-zero event. This field may be overloaded to disable relaxation algorithms for all channels, if *NoRelax* is unimplemented. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoOscMod (F85_AD_Ctrl0, bit 6)

This bit disables the oscillator-modulation algorithm. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoAutoServo (F85_AD_Ctrl0, bit 7)

This bit disables auto-servoing (for example, channel clip recovery) algorithms at the time when they would be initiated and should not terminate calibration after it has begun. This bit should be set when changing RefCap registers. Clearing this bit should not cause a servo event. This bit may be overloaded to disable the auto-sensitivity correction algorithms for all channels, which updates the Sensitivity Correction Factor registers, if NoAutoCalibration is unimplemented. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NukeOnReset (F85_AD_Ctrl1, bit 0)

This diagnostic bit enables all other nuke bits ("No*Xxxxxxx*" bits) on Reset events.

Spew (F85_AD_Ctrl1, bit 1)

Setting this bit to 1 causes the ATTN signal to be asserted on every frame, regardless of whether there is any new data to report.

NoSpatialFilter (F85_AD_Ctrl1, bit 2)

This bit disables the spatial filter. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoPosCorr (F85_AD_Ctrl1, bit 7)

This bit disables the position-correction tables. At reset this bit is ignored unless the *NukeOnReset* bit is set.

BurstsPerCluster (F85_AD_Ctrl2, bits 2:0)

This field specifies the number of ADC conversions per cluster. One full image frame is captured every *(BurstsPerCluster * NumberOfDriveElectrodesAvailable)* ADC conversions. Most factors of the ADC conversion rate are controlled by analog hardware configuration registers which can be written directly using F85_AD_Data3. This field controls the only factor of image frame rate that is controlled by firmware implementation, as opposed to analog hardware configuration. Depending on firmware implementation, this register may or may not be writable.

InterferenceMetricThresholdLo (F85_AD_Ctrl3)

This field sets the *Lo* bits of the threshold for the main interference metric. Any other interference metric thresholds will be set in the relevant function. Sensing should be disabled or ignored (for example, before POR) when changing the threshold.

InterferenceMetricThresholdHi (F85_AD_Ctrl4)

This field sets the *Hi* bits of the threshold for the main interference metric. Any other interference metric thresholds will be set in the relevant function. Sensing should be disabled or ignored (for example, before POR) when changing the threshold.

Bitmap Threshold (F85_AD_Ctrl5)

This field sets the threshold for the frame image bitmap. Each pixel of the capacitance image map is compared to this value to convert the pixel value into a binary value. The value of this threshold register is usually tuned for a specific module and not intended to be changed in normal usage. This register may or may not be writable in any given firmware implementation.

When changeable, it should be changed primarily to compensate for changes in other diagnostic registers such as *BurstsPerCluster* (F85_AD_ Ctrl2, bits 2:0).

Note: *BitmapThreshold* is not implemented.

InterferenceMetric2ThresholdLo (F85_AD_Ctrl6)

This field sets the *Lo* bits of the threshold for the secondary interference metric. Sensing should be disabled or ignored (for example, before POR) when changing the threshold.

Note: This register exists only if *HasIM2* (F85_AD_Query3, bit 6) is set to '1'.

InterferenceMetric2ThresholdHi (F85_AD_Ctrl7)

This field sets the *Hi* bits of the threshold for the secondary interference metric. Sensing should be disabled or ignored (for example, before POR) when changing the threshold.

Note: This register exists only if *HasIM2* (F85_AD_Query3, bit 6) is set to '1'.

18.3. Function $85: data registers

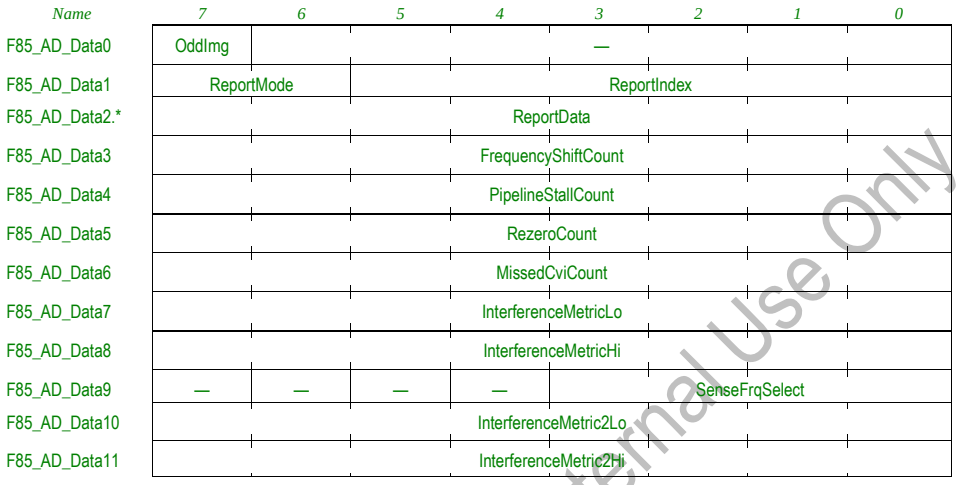

Figure 125. Function $85 data registers

Deleted: 126

The bits of these registers are defined as follows:

OddImg (F85_AD_Data0, bit 7)

Specifies that the current image buffer is the odd (high) image.

ReportIndex (F85_AD_Data1, bits 5:0)

For Baseline Frame Image Line (*ReportMode*=1) and Frame Image Line (*ReportMode*=2) this field specifies the line number of the image data. For Bitmap Frame Image (*ReportMode*=0) this field specifies which block (of size *SizeOfF85RegisterWindow*) of the bitmap is made available in the *ReportData* field. When *ReportMode*=0 and *ReportIndex*= $3F, the F85_AD_ Data1 register window allows read and write access to the hardware analog registers. When *ReportMode*=0 and *ReportIndex*= $3E, Raw ADC data is available for reading in the F85_AD_ Data2 register window.

ReportMode (F85_AD_Data1, bits 7:6)

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

ReportMode = 0, Bitmap Frame Image / Raw ADC / Hardware Register. This mode is used to read the full Bitmap Frame Image. When in this mode, the *ReportIndex* register controls the location of the F85_AD_ Data2 register window within the image. As a special case, when *ReportMode*=0 and *ReportIndex*= $3F, the F85_AD_ Data2 register window allows read and write access to the hardware analog registers. When *ReportMode*=0 and *ReportIndex*= $3E, the F85_AD_ Data2 register window reads the Raw ADC data.

ReportMode = 1, Baseline Frame Line. This mode is used to read the baseline frame image. When in this mode, the *ReportIndex* field controls the line number of the baseline image visible within the F85_AD_ Data2 register window. Each pixel of the baseline is two bytes, low order byte first (lower addressed register).

ReportMode = 2, Image Frame Line. This mode is used to read the frame image. When in this mode, the *ReportIndex* field controls the line number of the image visible within the F85_AD_ Data2 register window. Each pixel of the image is a single byte.

ReportMode = 3 is Reserved.

ReportData (F85_AD_Data2.)*

The content and format of the data present in the F85_AD_ Data2 register window changes based on the contents of the *ReportMode* and *ReportIndex* fields:

ReportMode = 0, ReportIndex = 0 - $3D: Bitmap Frame Image.

This report mode makes available a single *SizeOfF85RegisterWindow* byte block of the 2D capacitance binary image. Bitmap Frame Images larger than *SizeOfF85RegisterWindow* bytes are broken up into *SizeOfF85RegisterWindow* byte blocks with *ReportIndex* specifying which block to make visible in the F85_AD_ Data2 register window.

The bitmap image is composed of *NumberOfDriveElectrodesAvailable* lines each of which is *NumberOfSenseElectrodesAvailable* bits rounded up to the nearest 16-bit word. The bitmap image is a 2-D binary image that is thresholded version of the Frame Image. It is possible to read more than one line or block of data per interrupt. Given a fast enough communications channel it is possible to read an entire image each interrupt. The Bitmap Frame Image mode can also be used as a sort of compression mechanism – a smart host can use a bitmap to determine which lines and which pixels within those lines contain interesting information, and read only those pixels.

Note*:* In the current implementation, Bitmap Frame Image mode is not implemented.

ReportMode = 0 and *ReportIndex = $3F: HardwareRegisterWindow*

This contains a window onto the hardware registers of the device. These hardware registers must always be read and written coherently in pairs as a single word or as a group of words, each with a low first followed by a high byte.

ReportMode = 0 and *ReportIndex = $3E: Raw ADC.*

This report mode makes raw unprocessed ADC data available in *the F85_AD_ Data2* register window. This data is a sequence of 16 bit words, each stored low-then-high byte followed by status information for the ADC conversion sample set. This mode allows inspection of data that is not intended for image sensing applications.
The additional data consists of one byte that indicates the burst number within the cluster, followed by a bit-mask of the drivers.

The total size of data stored in the *ReportData* field is
*NumberOfSenseElectrodesAvailable*2 +*
FLOOR((NumberOfDriveElectrodesAvailable + 7)/8) + 1 bytes.

For a T1320 configured with the maximal 20 sense electrodes and 28 drive electrodes, the size of RAW ADC Data will be (20 * 2) + 4 + 1 = 45 bytes.

ReportMode = 1: Baseline Frame Line.

This report mode makes available a single line of the baseline image. The line stored in the *ReportLine* field is the line reported.

The baseline image is composed of *NumberOfDriveElectrodesAvailable* lines each of which is *NumberOfSenseElectrodesAvailable* 16-bit words. The baseline is a 2-D image that represents the base capacitance for each pixel location. The lines of this image can be read one at a time.

For a T1320 configured with the maximal 20 sense electrodes and 28 drive electrodes the size of a Baseline frame Image line is *NumberOfSenseElectrodesAvailable * 2* bytes.

ReportMode = 2: Image Frame Line.
This report mode makes available a single line of the 2-D capacitance image. The line stored in the *ReportLine* field is the line reported.

The image is composed of *NumberOfDriveElectrodesAvailable* lines each of which is *NumberOfSenseElectrodesAvailable* bytes. It represents the capacitance variance from the baseline for each pixel location. The lines of this image can be read one at a time.

For a T1320 configured with the maximal 20 sense electrodes and 28 drive electrodes, the size of an Image line is *NumberOfSenseElectrodesAvailable* bytes.

FrequencyShiftCount (F85_AD_Data3)
Read-only counter that increments whenever the noise-avoidance algorithm tries to shift the analog sensing frequency, regardless of whether the *NoFreqShift* bit prevents the shift from actually occurring.

PipelineStallCount (F85_AD_Data4)
Read-only counter that increments whenever the analog-sensing pipelines transitions from the unstalled to the stalled state.

RezeroCount (F85_AD_Data5)
Read-only counter that increments whenever the auto-rezero algorithm attempts to rezero the analog baseline, regardless of whether the *NoRezero* bit prevents the rezero from actually occurring.

MissedCviCount (F85_AD_Data6)
Read-only counter that increments whenever a Conversion Interrupt is missed.

InterferenceMetricLo (F85_AD_Data7)
Read-only register that holds the LSB of the current frame's Interference Metric.

InterferenceMetricHi (F85_AD_Data8)
Read-only register that holds the MSB of the current frame's Interference Metric.

SenseFreqSelect (F85_AD_Data9, bits 3:0)
Read/write field that indicates/selects the current analog sensing frequency. Range is 0 to *NumberOfSenseFrequencies* (F85_AD_Query6, bits 3:0); values outside that range will cause undefined behavior.

InterferenceMetric2Lo (F85_AD_Data10)
Read-only register that holds the LSB of the current frame's secondary Interference Metric.

Note: This register exists only if *HasIM2* (F85_AD_Query3, bit 6) is set to '1'.

InterferenceMetric2Hi (F85_AD_Data11)

Read-only register that holds the MSB of the current frame's secondary Interference Metric.

Note: This register exists only if *HasIM2* (F85_AD_Query3, bit 6) is set to '1'.

18.4. Function $85: interrupt sources

The Analog Diagnostic Data source can assert an interrupt request at the end of every report period (epoch) for which an interrupt source has been enabled. There are three interrupt sources for Function $85 – ADC Conversion, Image completion, and Servo completion. The interrupts will be enabled and reported in Function $81.

When one of these three interrupts is enabled by setting the corresponding bit in the F85_AD_Cmd0 register, the bit will remain set until the interrupt has occurred and then will be cleared automatically when the interrupt is asserted. When the interrupt has occurred, further analog conversions will be stalled until either one of two things occur: one of the three interrupt command bits is set, or the interrupt is cleared by reading the interrupt status register in Function $81.

18.5. Function $85: command registers

The control registers provide a mechanism for issuing commands for the analog diagnostic system. These bits automatically-clear to zero when the requested operation is complete.

Name	7	6	5	4	3	2	1	0
F85_AD_Cmd0	—	—	FrcZero	EnZeroIRQ	EnBitmapIRQ	EnBaseline IRQ	EnImageIRQ	—

Figure 126. Function $85 command registers

Deleted: 127

The bits of this register are defined as follows:

EnImageIRQ (F85_AD_Cmd0, bit 1)

Setting this bit causes an interrupt to be generated on the completion of the image acquisition. Prior to reading the image this bit should be set and the host should wait for the resulting interrupt. In order to ensure coherency of the image during the reading process new image acquisition is stalled until the next write to F85_AD_Cmd0.

EnBaselineIRQ (F85_AD_Cmd0, bit 2)

Setting this bit generates an interrupt the next time it is safe to read a baseline image. This bit must be set before reading a baseline image and the host should wait for the resulting interrupt. In order to ensure coherency of the baseline image during the reading process, new baseline acquisition is stalled until the next write to F85_AD_Cmd0.

EnBitmapIRQ (F85_AD_Cmd0, bit 3)

Setting this bit causes an interrupt to be generated on the completion of the bitmap image acquisition. This bit must be set before reading a bitmap image and the host should wait for the resulting interrupt. In order to ensure coherency of the bitmap image during the reading process, new bitmap image acquisition is stalled until the next write to F85_AD_Cmd0.

Note: *EnBitmapIRQ* is not implemented.

EnZeroIRQ (F85_AD_Cmd0, bit 4)

This bit causes an interrupt to occur the next time a rezeroing occurs (a new baseline is taken and stored). This can be used in combination with *FrcZero* to cause a rezero and then generate an interrupt when it occurs. This bit stays set until the rezero operation is completed, and then the bit is automatically cleared.

FrcZero (F85_AD_Cmd0, bit 5)

This bit causes a new baseline to be taken and stored. This can be used in combination with *EnZeroIRQ* to cause a rezero and then generate an interrupt when it occurs. This bit stays set until the rezero operation is completed, and then the bit is automatically cleared.

18.6. Function $85: hardware register window

This section describes the registers available in the Hardware Register Window for T1320 devices. These can be used as a guide, but the latest reference manual version takes precedence. The words are reported as low byte and then high byte within the respective RMI register bytes.

Important: Hardware registers must be written in pairs, starting with the RMI register corresponding to the low half of the CPU hardware register, followed by the RMI register corresponding to the high half of the CPU hardware register. The write to the CPU register only occurs after receiving the high half of the CPU hardware register.

For descriptions of the Memory Map contents, access the T1320 Reference Manual:

http://infoweb.synaptics.com/System_Silicon/www/projects/t1320/doc/docs.html

http://infoweb.synaptics.com/System_Silicon/www/projects/t1320/doc/t1320_requirements.doc

18.7. Reading images with Function $85

This section describes the process for reading RMI images with Function $85. When reading images, keep in mind the following rules:

- When image reporting is enabled, you must not read any RMI registers outside of Function $85 that are part of a coherent group. If you do, the captured snapshot will be corrupted and should be discarded.
- After a snapshot has been captured, no new snapshot will be acquired until you enable the next Analog Diagnostic interrupt by writing a '1' to *EnImageIRQ* (F85_AD_Cmd0, bit 1).
- If maintaining the frame rate is more important than the coherency of the report data, the next interrupt can be enabled before moving all the data for the current frame.

To read an image:

1. Disable low-power/dozing by setting the *NoSleep* bit of F01_RMI_Ctrl0.
2. Although not required, it is strongly suggested to disable all public interrupts so that only the ADC reports can cause ATTN to be asserted. To do this, write 0x00 to F01_RMI_Ctrl1.*n*.

   Note: *n* indicates that it is possible for there to be more than one Interrupt Enable register, in which case 0x00 must be written to all of them.
3. Unlock private functions via two consecutive writes to Device Status:
   - Write 0x42 to F01_RMI_Data0
   - Write 0xE1 to F01_RMI_Data0
4. If you have not already parsed the private PDTs, do so now in order to calculate the bitmask of Function 0x85's Analog interrupt for use in the next step.

5. Enable the private Analog interrupt by writing its bitmask (that was discovered above) into the correct private Interrupt Enable. To do this, write "bitmask" to F81_RMI_Data0.*n*:
    - If bitmask is < 0x100, write it to F81_RMI_Data0.0
    - If bitmask is 0x100..0x8000, shift it right by 8 and write it toF81_RMI_Data0.1

    Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), skip this step.
6. Set Reporting Mode in *ReportMode* (F85_AD_Data1, bits 7:5).
7. Request an Analog Diagnostic interrupt by writing a '1' to *EnImageIRQ* (F85_AD_Cmd0, bit 1). When this interrupt occurs its source can be determined by reading the interrupt status in F81_RMI_Data1. This will clear the interrupt.
8. At this point the entire image is available, and some or all of the image data can be read by writing *ReportIndex* (F85_AD_Data1, bits 4:0) and reading from *ReportData* (F85_AD_Data2.*).
9. When you are ready for the next image snapshot you must request the next Analog Diagnostic interrupt by writing a '1' to *EnImageIRQ* (F85_AD_Cmd0, bit 1).
10. To exit image-reading mode:
    - Exit image mode by writing a 0x00 to the F85_AD_Cmd0 register.
    - Although it is not necessary, you may exit the private function mode by setting the *Configured* bit in F01_RMI_Ctrl0.
    - Re-enable the public interrupts by writing to F01_RMI_Ctrl1.n.

    Alternatively, you can reset the device.

18.8. Reading baselines with Function $85

This section describes the process for reading the image baseline with Function $85. To read the baseline:

1. Disable low-power/dozing by setting the *NoSleep* bit of F01_RMI_Ctrl0.
2. Although not required, it is strongly suggested to disable all public interrupts so that only the ADC reports can cause ATTN to be asserted. To do this, write 0x00 to F01_RMI_Ctrl1.*n*.

Note: *n* indicates that it is possible for there to be more than one Interrupt Enable register, in which case 0x00 must be written to all of them.

3. Unlock private functions via two consecutive writes to Device Status:
    - Write 0x42 to F01_RMI_Data0
    - Write 0xE1 to F01_RMI_Data0
4. If you have not already parsed the private PDTs, do so now in order to calculate the bitmask of Function 0x85's Analog interrupt for use in the next step.
5. Enable the private Analog interrupt by writing its bitmask (that was discovered above) into the correct private Interrupt Enable. To do this, write "bitmask" to F81_RMI_Data0.*n*:
    - If bitmask is < 0x100, write it to F81_RMI_Data0.0

- If bitmask is 0x100..0x8000, shift it right by 8 and write it to F81_RMI_Data0.1

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

6. Set Reporting Mode in *ReportMode* (F85_AD_Data1, bits 7:5).
7. Request an Analog Diagnostic interrupt by simultaneously writing ones to F85_AD_Cmd0, bit 4 (*EnZeroIRQ*) and F85_AD_Cmd0, bit 5 (*FrcZero*). When this interrupt occurs its source can be determined by reading the interrupt status in F81_RMI_Data1. This will clear the interrupt.
8. At this point the entire baseline frame is available, and some or all of its data can be read by writing *ReportIndex* (F85_AD_Data1, bits 4:0) and reading from *ReportData* (F85_AD_Data2.*).
9. Repeat steps 7 and 8 to read another captured baseline frame.
10. To exit baseline-reading mode:
    - Exit baseline mode by writing a 0x00 to F85_AD_Cmd0 register.
    - Although it is not necessary, you may exit the private function mode by setting the *Configured* bit in F01_RMI_Ctrl0
    - Re-enable the public interrupts by writing to F01_RMI_Ctrl1.*n*.

    Alternatively, you can reset the device.

19. Function $87: Analog Diagnostic for LTS Central Microcontroller (Private)

Function $87 provides private analog diagnostic functions that allow visibility and control of the base analog functionality used for large-touchscreen transcapacitive image sensing. It is present only in the central microcontroller that collects and processes analog data from one or more capacitance-sensing touch controllers in an LTS device.

In general, it is the intent of the F$87 diagnostics to provide information about, and control of, the central microcontroller and its core algorithms, by offering a means to coherently read the aggregated capacitance data in its various forms in synchronization with the operational firmware, without impeding the normal operation of other functions. All devices should support basic analog diagnostic controls. More advanced controls may be implemented selectively.

Controls for the individual touch controllers' analog-sensing functions are not contained within this function; those controls are implemented in each touch controller's own Analog Diagnostics function.

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

- Controls for algorithms, such as spatial filtering for strips that are only present in functions for certain devices, are implemented in their own function.
- Scaling Sensitivities (not Response Correction Factors) for the electrodes within a function are controlled by that function.
- Any per-channel Z reporting (for example, Zdelta or Zabs) may be implemented within Analog Diagnostics while function specific (for example, Zx and Zy) will be implemented in that function.
- Fine-grain control of zeroing or relaxation is controlled within a particular function, the Analog Diagnostic function only implements those features that are expected to appear in every diagnostic function, although it may contain global controls that affect all functions, but do not override the fine-grain control.

Note: See the BIST Functions for higher-level and more automated testing functions.

All registers of the Analog Diagnostic function are considered private, unless they specifically need to be released to a customer. An unlock sequence has been provided to prevent any spurious write to the control or command registers of the Analog Diagnostic function, which could result in incorrect data and difficult-to-diagnose problems.

19.1. Function $87: query registers

The query registers describe the available electrodes, modes, and function capabilities available in this product.

Name	7	6	5	4	3	2	1	0
F87_AD_Query0	NumberOfSenseElectrodesAvailable							
F87_AD_Query1	NumberOfDriveElectrodesAvailable							
F87_AD_Query2	—	HasImage16	—	HasBitmap	HasImage8	HasBaseLine	HasADCRaw	HasADCtrl
F87_AD_Query3	—	—	—	—	—	—	—	—
F87_AD_Query4	SizeOfF87RegisterWindow (bytes)							
F87_AD_Query5	—	—	—	—	—	—	—	—
F87_AD_Query6	—				NumberOfSenseFrequencies			

Figure 127. Function $87 query registers

Deleted: 128

The bits of these registers are defined as follows:

NumberOfSenseElectrodesAvailable (F87_AD_Query0)

Reports the number of sense electrodes available on the design.

NumberOfDriveElectrodesAvailable (F87_AD_Query1)

Reports the number of drive electrodes available on the design.

HasADCtrl (F87_AD_Query2, bit 0)

If ‘1’, then Analog Diagnostic Control is available. If ‘0’, then no writable registers are available within Function $87. In this case, query, data, and control registers may still be read.

HasADCRaw (F87_AD_Query2, bit 1)

This field indicates the availability of the ADC Raw reporting mode. This report is made through the data registers F87_AD_Data2.*:

- ‘0’: ADC Raw reporting is not supported.
- ‘1’: ADC Raw is supported.

HasBaseline (F87_AD_Query2, bit 2)

This field indicates the availability of the Baseline reporting mode. This report is made through the data registers F87_AD_Data2.*:

- ‘0’: Baseline reporting is not supported.
- ‘1’: Baseline reporting is supported.

HasImage8 (F87_AD_Query2, bit 3)

This field indicates the availability of the Image Line reporting mode for 8-bit images. This report is made through the data registers F87_AD_Data2.*:

- ‘0’: 8-bit Image Line reporting is not supported.
- ‘1’: 8-bit Image Line reporting is supported.

HasBitmap (F87_AD_Query2, bit 4)

This field indicates the availability of the Bitmap Image reporting mode. This report is made through the data registers F87_AD_Data2.*:

- '0': Bitmap Image reporting is not supported.
- '1': Bitmap Image reporting is supported.

Reserved (F87_AD_Query2, bit 5)

Reserved.

HasImage16 (F87_AD_Query2, bit 6)

This field indicates the availability of the Image Line reporting mode for 16-bit images. This report is made through the data registers F87_AD_Data2.*:

- '0': 16-bit Image Line reporting is not supported.
- '1': 16-bit Image Line reporting is supported.

Reserved (F87_AD_Query3)

Reserved.

SizeOfF87RegisterWindow (F87_AD_Query4)

This field indicates the number of register bytes mapped in the F87 register window – F87_AD_ Data1. The F87_AD_ Data2 register window is the interface through which Function $87 provides access to ADC, image, and baseline report data.

Reserved (F87_AD_Query5)

Reserved.

NumberOfSenseFrequencies (F87_AD_Query6, bits 3:0)

Specifies the number (-1) of available analog sensing frequencies.

19.2. Function $87: control registers

The control registers provide a mechanism for configuring the Analog Diagnostic registers and algorithms. They also control the reporting and configuration of the data and command registers.

Name	7	6	5	4	3	2	1	0
F87_AD_Ctrl0	—	—	NoAutoZero	NoRelax	—	NoBleedover	NoFreqShift	NoCMNR
F87_AD_Ctrl1	NoPosCorr	—	—	—	—	NoSpatial Filter	Spew	NukeOnReset
F87_AD_Ctrl2								
F87_AD_Ctrl3	InterferenceMetricThresholdLo							
F87_AD_Ctrl4	InterferenceMetricThresholdHi							
F87_AD_Ctrl5	Bitmap Threshold							

Figure 128: Function $87 control registers

Deleted: 129

The bits of these registers are defined as follows:

NoCMNR (F87_AD_Ctrl0, bit 0)

This bit disables the common-mode noise reduction algorithms for all channels. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoFreqShift (F87_AD_Ctrl0, bit 1)

This bit disables automatic shifting of the analog sensing frequency on all touch controllers. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoBleedover (F87_AD_Ctrl0, bit 2)

This bit disables the LPF Bleedover algorithm on all touch controllers. At reset this bit is ignored unless the *NukeOnReset* bit is set.

Reserved (F87_AD_Ctrl0, bit 3)

Reserved.

NoRelax (F87_AD_Ctrl0, bit 4)

This bit disables relaxation algorithms for all channels. When set to 1, it overrides any per-function relaxation control. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoAutoZero (F87_AD_Ctrl0, bit 5)

This bit disables auto-zeroing (lift recovery) algorithms for all channels. When set to 1, it overrides any per-function rezero control. Clearing the bit should not cause an auto-zero event. This field may be overloaded to disable relaxation algorithms for all channels, if *NoRelax* is unimplemented. At reset this bit is ignored unless the *NukeOnReset* bit is set.

Reserved (F87_AD_Ctrl0, bit 6)

Reserved.

Reserved (F87_AD_Ctrl0, bit 7)

Reserved.

NukeOnReset (F87_AD_Ctrl1, bit 0)

This diagnostic bit enables all other nuke bits (“No*Xxxxxxx*” bits) on Reset events.

Spew (F87_AD_Ctrl1, bit 1)

Setting this bit to 1 causes the ATTN signal to be asserted on every frame, regardless of whether there is any new data to report.

NoSpatialFilter (F87_AD_Ctrl1, bit 2)

This bit disables the spatial filter. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoPosCorr (F87_AD_Ctrl1, bit 7)

This bit disables the position-correction tables. At reset this bit is ignored unless the *NukeOnReset* bit is set.

Reserved (F87_AD_Ctrl2)

Reserved.

InterferenceMetricThresholdLo (F87_AD_Ctrl3)

InterferenceMetricThresholdHi (F87_AD_Ctrl4)

These registers set the threshold for the main interference metric. Any other interference metric thresholds will be set in the relevant function. Sensing should be disabled or ignored (for example, before POR) when changing threshold.

Note: These two registers are a coherent group: To change either register, both registers must be written (see section 2.6).

Formatted: Font color: Green

Deleted: 2.6

Bitmap Threshold (F87_AD_Ctrl5)

This field sets the threshold for the frame image bitmap. Each pixel of the capacitance image map is compared to this value to convert the pixel value into a binary value. The value of this threshold register is usually tuned for a specific module and not intended to be changed in normal usage.

This register may or may not be writable in any given firmware implementation. When changeable, it should be changed primarily to compensate for changes in the touch controllers' internal registers (such as *BurstsPerCluster)*.

Note: The analog parameters of the touch controllers can be configured through the frame block tables of the individual touch controllers, by accessing their Function $25 interfaces directly.

19.3. Function $87: data registers

Name	7	6	5	4	3	2	1	0
F87_AD_Data0	—	—	—	—	—	ReportMode		
F87_AD_Data1	ReportIndex							
F87_AD_Data2.*	ReportData							
F87_AD_Data3	FrequencyShiftCount							
F87_AD_Data4	—	—	—	—	—	—	—	—
F87_AD_Data5	RezeroCount							
F87_AD_Data6	—	—	—	—	—	—	—	—
F87_AD_Data7	InterferenceMetricLo							
F87_AD_Data8	InterferenceMetricHi							
F87_AD_Data9	SenseFreqSelect							
F87_AD_Data10	FIFOIndexLo							
F87_AD_Data11	FIFOIndexHi							
F87_AD_Data12	FIFOData							
F87_AD_Data13	SyncLostCount							
F87_AD_Data14	ConfigLostCount							
F87_AD_Data15	DroppedFrameCount							
F87_AD_Data16	IncorrectClusterCount							
F87_AD_Data17	QueueOverflowCount							
F87_AD_Data18	ChecksumErrorCount							

Figure 129. Function $87 data registers

Deleted: 130

The bits of these registers are defined as described below. The roles of the *ReportMode* field, the *ReportIndex* and *ReportData* registers, and the *FIFOIndex** and *FIFOData* registers are quite intertwined, Pay special attention when studying the description of these registers.

ReportMode (F87_AD_Data0, bits 2:0)

Specifies the report mode for the data in the *ReportData* and *FIFOData* registers. The primary use of these two facilities is to read numerical data pertaining to image frames.

- Through the ReportData interface, simultaneous image data for all sensor channels is available in the registers of the *ReportData* register bundle with the size of *SizeOfF87RegisterWindow* bytes. (See the calculation of the *SizeOfF87RegisterWindow* value under *ReportMode*=4.) Sensor data corresponding to the consecutive transmitter drive patterns (referred to as “lines”) can be accessed by setting the *ReportIndex* register to the desired line number. The complete image is made up by the collection of lines corresponding to each individual transmitter drive pattern. See the descriptions of the individual modes for further details.

- Through the FIFOData interface, the image data described above is read from an individual RMI register address in a sequential manner. (See the descriptions of the *FIFOIndex** and *FIFOData* registers for further details on the FIFOData interface.)

In modes that are exceptions to the primary use mentioned above, the differences are highlighted below.

ReportMode = 0: Bitmap Frame Image

This special report mode is used to read the full Bitmap Frame Image, which is the thresholded binary version of the Frame Image. The bitmap image is composed of *NumberOfDriveElectrodesAvailable* lines, each of which is *NumberOfSenseElectrodesAvailable* bits rounded up to the nearest 16-bit word (representing the bitmap of each line). Formulaically, the number of bytes in the bitmap image in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes is:

$$n_{Bitmap\ Bytes} = NumberOfDriveElectrodesAvailable * FLOOR((NumberOfSenseElectrodesAvailable+15)/16) * 2$$

$$= 56 * FLOOR((40+15)/16) * 2 = 56 * 3 * 2 = 336$$

When using the ReportData interface to read bitmap frame images, bitmaps larger than *SizeOfF87RegisterWindow* bytes are broken up into *SizeOfF87RegisterWindow* byte blocks, with *ReportIndex* specifying which block to make visible in the *ReportData* register window. Formulaically, the byte-length and number of blocks in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes are:

$$n_{Bitmap\ Blocklen\ Bytes} = SizeOfF87RegisterWindow = 88$$

$$n_{Bitmap\ Blocklen\ Last\ Bytes} = n_{Bitmap\ Bytes}\ MOD\ SizeOfF87RegisterWindow = 72$$

$$n_{Bitmap\ Blocks} = FLOOR((n_{Bitmap\ Bytes}+SizeOfF87RegisterWindow-1)/SizeOfF87RegisterWindow)$$

$$= FLOOR((336+88-1)/88) = 4$$

It is possible to read more than one line or block of data per interrupt. Given a fast enough communications channel, it is possible to read an entire image each interrupt. The Bitmap Frame Image mode can also be used as a sort of compression mechanism – a smart host can use a bitmap to determine which lines and which pixels within those lines contain interesting information, and read only those pixels.

ReportMode = 1: Frame Baseline

This report mode is used to read the frame baseline, representing the base capacitance for each pixel location. The baseline is composed of *NumberOfDriveElectrodesAvailable* lines, each of which is *NumberOfSenseElectrodesAvailable* 16-bit words, presented low-order byte first.

Formulaically, the number of bytes in the baseline frame image in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes is:

$$n_{Baseline\ Bytes} = NumberOfDriveElectrodesAvailable * NumberOfSenseElectrodesAvailable * 2$$

$$= 56 * 40 * 2 = 4480$$

When using the ReportData interface to read the baseline, the byte- length of the blocks, and number of blocks to be indexed by *ReportIndex* in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes are:

$$n_{Baseline\ Blocklen\ Bytes} = NumberOfSenseElectrodesAvailable * 2 = 80$$

$$n_{Baseline\ Blocks} = NumberOfDriveElectrodesAvailable = 56$$

ReportMode = 2: 8-bit Image Frame Line

This report mode is used to read the frame image as 8-bit pixel data, representing the capacitance variance from the baseline for each pixel location. If a given build implements 16-bit image data, the results are saturated to the maximum value of 255. The image is composed of *NumberOfDriveElectrodesAvailable* lines, each of which is *NumberOfSenseElectrodesAvailable* bytes. Formulaically, the number of bytes in the 8-bit frame image in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes is:

$$N_{8\text{-}bit\ Image\ Bytes} = NumberOfDriveElectrodesAvailable * NumberOfSenseElectrodesAvailable$$

$$= 56 * 40 = 2240$$

When using the ReportData interface to read 8-bit frame images, the byte- length of the blocks, and number of blocks to be indexed by *ReportIndex* in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes are:

$$n_{8\text{-}bit\ Image\ Blocklen\ Bytes} = NumberOfSenseElectrodesAvailable = 40$$

$$n_{8\text{-}bit\ Image\ Blocks} = NumberOfDriveElectrodesAvailable = 56$$

ReportMode = 3: 16-bit Image Frame Line

This report mode is used to read the frame image as 16-bit pixel data, representing the capacitance variance from the baseline for each pixel location. If a given build implements 8-bit image data, the results are undetermined. The image is composed of *NumberOfDriveElectrodesAvailable* lines, each of which is *NumberOfSenseElectrodesAvailable* 16-bit words, presented low-order byte first. Formulaically, the number of bytes in the 16-bit frame image in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes are:

$$N_{16\text{-}bit\ Image\ Bytes} = NumberOfDriveElectrodesAvailable * NumberOfSenseElectrodesAvailable * 2$$

$$= 56 * 40 * 2 = 4480$$

When using the ReportData interface to read 16-bit frame images, the byte- length of the blocks, and number of blocks to be indexed by *ReportIndex* in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes are:

$$n_{16\text{-}bit\ Image\ Blocklen\ Bytes} = NumberOfSenseElectrodesAvailable * 2 = 80$$

$$n_{16\text{-}bit\ Image\ Blocks} = NumberOfDriveElectrodesAvailable = 56$$

ReportMode = 4: Raw ADC Data

This report mode is used to read raw unprocessed ADC image frames. The image is composed of *NumberOfDriveElectrodesAvailable* lines, each of which is *NumberOfSenseElectrodesAvailable* 16-bit words, presented low-order byte first.

Formulaically, the number of bytes in the 16-bit frame image in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes are:

$$N_{\text{Raw ADC Image Bytes}} = \textit{NumberOfDriveElectrodesAvailable} * \textit{NumberOfSenseElectrodesAvailable} * 2$$

$$= 56 * 40 * 2 = 4480$$

When using the ReportData interface to read 16-bit frame images, the byte- length of the blocks, and number of blocks to be indexed by *ReportIndex* in general, and for an LTS configured with 56 drive electrodes and 40 sense electrodes are:

$$N_{\text{Raw ADC Image Blocklen Bytes}} = \textit{NumberOfSenseElectrodesAvailable} * 2 = 80$$

$$N_{\text{Raw ADC Image Blocks}} = \textit{NumberOfDriveElectrodesAvailable} = 56$$

ReportIndex (F87_AD_Data1)

When using the ReportData interface, this register is used to index the queried data into the *ReportData* register window. Depending on the mode selected by the *ReportMode* field, the use of this register is as follows:

- Bitmap Frame Image (*ReportMode*=0): Specifies which block (of size *SizeOfF87RegisterWindow*) of the bitmap is made available in the *ReportData* field.
- Baseline Frame Image Line (*ReportMode*=1) and Frame Image Line (*ReportMode*=2 and *ReportMode*=3): Specifies the line number of the image data.
- Raw ADC Data (*ReportMode*=4): Unused. (The *SizeOfF87RegisterWindow* value was chosen such that the entire raw ADC data block fits into the *ReportData* register block.

Note: Setting *ReportIndex* to a value that is outside the valid range for a given mode will result in undefined behavior.

ReportData (F87_AD_Data2.)*

When using the ReportData facility, this register is used to as the register window to present the queried data. The contents of this block for the different report modes are explained under the description of *ReportMode*.

FrequencyShiftCount (F87_AD_Data3)

Read-only counter that increments whenever the noise-avoidance algorithm tries to shift the analog sensing frequency, regardless of whether the *NoFreqShift* bit prevents the shift from actually occurring.

Reserved (F87_AD_Data4)

Reserved.

RezeroCount (F87_AD_Data5)

Read-only counter that increments whenever the auto-rezero algorithm attempts to rezero the analog baseline, regardless of whether the *NoRezero* bit prevents the rezero from actually occurring.

Reserved (F87_AD_Data6)

Reserved.

InterferenceMetricLo (F87_AD_Data7)

InterferenceMetricHi (F87_AD_Data8)

Read-only registers that hold the current frame's Interference Metric.

Note: These two registers are a coherent group (see section 2.6).

SenseFreqSelect (F87_AD_Data9, bits 3:0)

Read/write field that indicates/selects the current analog sensing frequency. Range is 0 to *NumberOfSenseFrequencies* (F87_AD_Query6, bits 3:0); values outside that range will cause undefined behavior.

FIFOIndexLo and FIFOIndexHi (F87_AD_Data10 and F87_AD_Data11)

As an added alternative to the F85-style image access through the *ReportIndex* and *ReportData* registers, Function $87 also provides a FIFO-style interface for reading the image data. With the byte index automatically incremented upon each read of one byte data [see below], high-speed reading of the image with no inter-byte latency can be achieved. When reading an image in 16-bit resolution, each pixel takes 2 bytes in figuring this index.

This pair of registers stores the bytewise index of the image data that is being read through the FIFO interface. It is reset to zero after boot and upon an Image Complete interrupt, and set to the beginning of the most recent line upon a Line Complete interrupt (see Section 19.5). In addition, writing a byte index explicitly into this 2-byte register provides an optional way to change the position within the image buffer, from which the image data are read. While access to the image through this FIFO facility is sequential, the ability to specify the byte index within the FIFO adds a certain level of random access for applications that need to seek within the entire image.

Reading this pair of registers will return the current value of the byte index counter.

Note: These two registers are a coherent group: To change either register, both registers must be written (see section 2.6).

FIFOData (F87_AD_Data12)

Repeated reads of this single register provides consecutive bytes of the image data, by auto-incrementing the internal pixel index counter (optionally set through *FIFOIndexLo* and *FIFOIndexHi*) . For multi-byte pixel data, the least significant byte is served first. The content and format of the data present in this FIFO register changes based on the contents of the *ReportMode* field (see the description of modes there).

Note: Setting *FIFOIndex* to a value that is outside the valid range for a given mode, or reading the FIFO past valid data, will result in undefined behavior.

Important: This register must be read by specifying its exact address, in which case its address will not be auto-incremented upon repeated reads. Instead, as stated above, repeated reads auto-increment the internal pixel counter of the FIFO, so that subsequent pixel data can be accessed.

On the other hand, this register will read 0 if reached through a "run-by", that is, as the result of address auto-increment during an RMI block read. Subsequent reads within the block will leave the FIFO pixel counter unchanged, and move on to the data register past *FIFOData*.

SyncLostCount (F87_AD_Data13)

Counter that increments whenever any touch controller signals that it has detected a loss-of-sync condition. This counter saturates at 255; it can be reset by writing a 0 into the register.

Deleted: 2.6

Formatted: Font color: Green

Deleted: 19.5

Formatted: Font color: Green

Formatted: Font color: Green

Deleted: 2.6

Comment [MSOffice4]: In the next revision (Rev B), rewrite this section to be similar to the FIFO Address and FIFO Data registers in Function $81 (section 16.3.4)

ConfigLostCount (F87_AD_Data14)

Counter that increments whenever any touch controller signals that it has detected a loss-of-configuration condition caused by a spontaneous chip reset. This counter saturates at 255; it can be reset by writing a 0 into the register.

DroppedFrameCount (F87_AD_Data15)

Counter that increments whenever the host does not read a frame in time so that the frame buffer has to be recycled for the next frame. This counter saturates at 255; it can be reset by writing a 0 into the register.

IncorrectClusterCount (F87_AD_Data16)

Counter that increments whenever the host receives a Function $25 cluster image with an unexpected cluster index. This counter saturates at 255; it can be reset by writing a 0 into the register.

QueueOverflowCount (F87_AD_Data17)

Counter that increments whenever the host receives a Function $25 cluster image with the status header indicating an image queue overflow has occurred at the slave after acquiring this cluster. This counter saturates at 255; it can be reset by writing a 0 into the register.

ChecksumErrorCount (F87_AD_Data18)

Counter that increments whenever the host receives a Function $25 cluster image whose calculated checksum does not match the value given in the header. This counter saturates at 255; it can be reset by writing a 0 into the register.

19.4. Function $87: interrupt sources

The Analog Diagnostic Data source can assert an interrupt request at the end of every report period (epoch), for which an interrupt source has been enabled. There are three interrupt sources for Function $87 – Image ready, Baseline ready, and Servo complete. The interrupts will be enabled and reported in Function $81.

When one of these three interrupts is enabled by setting the corresponding bit in the F87_AD_Cmd0 register, the bit will remain set until the interrupt has occurred and then will be cleared automatically when the interrupt is asserted. When the interrupt has occurred, further analog conversions will be stalled until either one of two things occur: one of the three interrupt command bits is set, or the interrupt is cleared by reading the interrupt status register in Function $81.

19.5. Function $87: command registers

The control registers provide a mechanism for issuing commands for the analog diagnostic system. These bits auto-clear to zero when the requested operation is complete.

Name	7	6	5	4	3	2	1	0
F87_AD_Cmd0	QuitCmds	EnRawADCIRQ	FrcZero	EnZeroIRQ	EnBitmapIRQ	EnBaseline IRQ	EnImageIRQ	EnLineIRQ

Figure 130: Function $87 command register

Deleted: 131

The bits of this register are defined as follows:

EnLineIRQ (F87_AD_Cmd0, bit 0)

Setting this bit causes an interrupt to be generated on the completion of the acquisition of one line within the image. Prior to reading the line, this bit should be set and the host should wait for the resulting interrupt. In order to ensure coherency of the image during the reading process new image acquisition is stalled until the next write to F87_AD_Cmd0 if the current image buffer is about to be overwritten.

Note: This command works for both the 8-bit and the 16-bit image reporting modes, whichever is applicable for a given implementation.

EnImageIRQ (F87_AD_Cmd0, bit 1)

Setting this bit causes an interrupt to be generated on the completion of the image acquisition. Prior to reading the image this bit should be set and the host should wait for the resulting interrupt. In order to ensure coherency of the image during the reading process new image acquisition is stalled until the next write to F87_AD_Cmd0 if the current image buffer is about to be overwritten.

Note: This command works for both the 8-bit and the 16-bit image reporting modes, whichever is applicable for a given implementation.

EnBaselineIRQ (F87_AD_Cmd0, bit 2)

Setting this bit generates an interrupt the next time it is safe to read a baseline image. This bit must be set before reading a baseline image and the host should wait for the resulting interrupt. In order to ensure coherency of the baseline image during the reading process, any baseline updating is stalled until the next write to F87_AD_Cmd0 if the current image buffer is about to be overwritten.

EnBitmapIRQ (F87_AD_Cmd0, bit 3)

Setting this bit causes an interrupt to be generated on the completion of the bitmap image acquisition. This bit must be set before reading a bitmap image and the host should wait for the resulting interrupt. In order to ensure coherency of the bitmap image during the reading process, new bitmap image acquisition is stalled until the next write to F87_AD_Cmd0.

EnZeroIRQ (F87_AD_Cmd0, bit 4)

This bit causes an interrupt to occur the next time a rezeroing occurs (a new baseline is taken and stored). This can be used in combination with *FrcZero* to cause a rezero and then generate an interrupt when it occurs. This bit stays set until the rezero operation is completed, and then the bit is automatically cleared.

FrcZero (F87_AD_Cmd0, bit 5)
This bit causes a new baseline to be taken and stored. This can be used in combination with *EnZeroIRQ* to cause a rezero and then generate an interrupt when it occurs. This bit stays set until the rezero operation is completed, and then the bit is automatically cleared.

EnRawADCIRQ (F87_AD_Cmd0, bit 6)
Setting this bit causes an interrupt to be generated on the completion of image acquisition without further processing like baseline removal. This bit must be set before reading a raw image and the host should wait for the resulting interrupt. To ensure coherency of the raw image during the reading process, new image acquisition is stalled until the next write to F87_AD_Cmd0.

QuitCmds (F87_AD_Cmd0, bit 7)
This bit causes all pending commands to be canceled, and their respective command bit to be reset. When no command is pending, writing 1 to this bit means that no new command will be issued, and the data acquisition stall requested by the completion of the previous command can be canceled.

19.6. Reading images with Function $87

This section describes the process for reading RMI images with Function $87. An image could be the current baseline used by firmware, the next delta ADC frame in 8-bit or 16-bit format, or the next raw ADC frame, depending on the report mode. When reading images, keep in mind the following rules:

- After a snapshot has been captured, no new snapshot will be acquired until you enable the next Analog Diagnostic interrupt by writing into the F87_AD_Cmd0 register.
- If maintaining the frame rate is more important than the coherency of the report data, the next interrupt can be enabled before moving all the data for the current frame.

To read an image:

1. Disable low-power/dozing by setting the *NoSleep* bit of F01_RMI_Ctrl0.
2. Although not required, it is strongly suggested to disable all public interrupts so that only the ADC reports can cause ATTN to be asserted. To do this, write 0x00 to F01_RMI_Ctrl1.*n*.

   Note: *n* indicates that it is possible for there to be more than one Interrupt Enable register, in which case 0x00 must be written to all of them.
3. Disable "Slave" Bus Attention Enable by writing 0x01 to F01_RMI_Ctrl2.00 and 0x00 to F01_RMI_Ctrl2.*n* (if exist any *n*>0).
4. Clear public interrupt status by reading F01_RMI_Data1.00.
5. Unlock private functions via two consecutive writes to Device Status:
   - Write 0x42 to F01_RMI_Data0
   - Write 0xE1 to F01_RMI_Data0
6. If you have not already parsed the private PDTs, do so now in order to calculate the bitmask of Function $87's Analog interrupt for use in the next step.

7. Enable the private Analog interrupt by writing its bitmask (that was discovered above) into the correct private Interrupt Enable. To do this, write "bitmask" to F81_RMI_Data0.*n*:

- If bitmask is < 0x100, write it to F81_RMI_Data0.0
- If bitmask is 0x100..0x8000, shift it right by 8 and write it toF81_RMI_Data0.1

Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), skip steps 2, 3, 4, and 7.

8. Set Reporting Mode (1 for baseline, 2 for 8-bit image, 3 for 16-bit image, 4 for raw ADC) in *ReportMode* (F87_AD_Data0, bits 2:0).
9. Request an Analog Diagnostic interrupt by writing a '1' to *EnImageIRQ* (F87_AD_Cmd0, bit 1, if reading an 8-bit or 16-bit image) or *EnBaselineIRQ* (F87_AD_Cmd0, bit 2, if reading a baseline image) or *EnRawADCIRQ* (F87_AD_Cmd0, bit 6, if reading a raw ADC frame). When this interrupt occurs its source can be determined by reading the interrupt status in F81_RMI_Data1. This will clear the interrupt.
10. At this point the entire image is available, and some or all of the image data can be read by:
    - Writing *ReportIndex* (F87_AD_Data1) and reading from *ReportData* (F87_AD_Data2.*) **or**
    - Reading repeatedly from *FIFOData* (F87_AD_Data12).
11. When you are ready for the next image snapshot you must request the next Analog Diagnostic interrupt by repeating step 9.
12. To exit image-reading mode:
    - Exit image mode by writing a '1' into the *QuitCmds* bit (bit 7) of the F87_AD_Cmd0 register.
    - Disable private Analog interrupt.
    - Clear private interrupt status by reading F81_RMI_Data1.00.
    - Although it is not necessary, you may exit the private function mode by setting the *Configured* bit in F01_RMI_Ctrl0.
    - Re-enable Bus Attention Enable by restoring F01_RMI_Ctrl2.*n*.
    - Re-enable the public interrupts by restoring F01_RMI_Ctrl1.n.

Alternatively, you can reset the device.

19.7. Re-grabbing and reading baselines with Function $87

This section describes the process for re-grabbing and reading the image baseline with Function $87. To re-grab and read the baseline:

1. Disable low-power/dozing by setting the *NoSleep* bit of F01_RMI_Ctrl0.
2. Although not required, it is strongly suggested to disable all public interrupts so that only the ADC reports can cause ATTN to be asserted. To do this, write 0x00 to F01_RMI_Ctrl1.*n*.

Note: *n* indicates that it is possible for there to be more than one Interrupt Enable register, in which case 0x00 must be written to all of them.

3. Disable "Slave" Bus Attention Enable by writing 0x01 to F01_RMI_Ctrl2.00 and 0x00 to F01_RMI_Ctrl2.*n* (if exist any *n*>0).
4. Clear public interrupt status by reading F01_RMI_Data1.00.
5. Unlock private functions via two consecutive writes to Device Status:
   - Write 0x42 to F01_RMI_Data0
   - Write 0xE1 to F01_RMI_Data0
6. If you have not already parsed the private PDTs, do so now in order to calculate the bitmask of Function $87's Analog interrupt for use in the next step.
7. Enable the private Analog interrupt by writing its bitmask (that was discovered above) into the correct private Interrupt Enable. To do this, write "bitmask" to F81_RMI_Data0.*n*:
   - If bitmask is < 0x100, write it to F81_RMI_Data0.0
   - If bitmask is 0x100..0x8000, shift it right by 8 and write it to F81_RMI_Data0.1

   Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), skip steps 2, 3, 4, and 7.
8. Set Reporting Mode to baseline by writing '1' to *Report Mode* (F87_AD_Data0, bits 2:0).
9. Request an Analog Diagnostic interrupt by simultaneously writing ones to F87_AD_Cmd0, bit 4 (*EnZeroIRQ*) and F87_AD_Cmd0, bit 5 (*FrcZero*). When this interrupt occurs its source can be determined by reading the interrupt status in F81_RMI_Data1. This will clear the interrupt.
10. At this point a new baseline has been grabbed (another saying is a new rezero is done) and the entire baseline frame is available, and some or all of its data can be read by:
    - Writing *ReportIndex* (F87_AD_Data1) and reading from *ReportData* (F87_AD_Data2.*) **or**
    - Reading repeatedly from *FIFOData* (F87_AD_Data12).
11. Repeat steps 9 and 10 to read another captured baseline frame.
12. To exit baseline-reading mode:
    - Exit baseline mode by writing a 1 into the *QuitCmds* bit (bit 7) of the F87_AD_Cmd0 register.
    - Disable private Analog interrupt.
    - Clear private interrupt status by reading F81_RMI_Data1.00.
    - Although it is not necessary, you may exit the private function mode by setting the *Configured* bit in F01_RMI_Ctrl0
    - Re-enable Bus Attention Enable by restoring F01_RMI_Ctrl2.*n*.
    - Re-enable the public interrupts by restoring F01_RMI_Ctrl1.*n*.

    Alternatively, you can reset the device.

20. Function $91: 2-D private functions

Function $91 implements the private registers for two-dimensional touch position sensors, such as the Synaptics TouchPad™ or ClearPad™ products. The entire function is private and only those registers that need to be revealed to a customer should be. Some registers may be used by customers and set by Synaptics tools, but the customer should have no need to change them outside of this tool set.

Important: For functionality that exists in both Function $11 and Function $91, please use Function $11. Function $91 is being deprecated.

20.1. Number of 2-D sensors

The Number of Sensors 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.

20.2. Function $91: query registers

RMI Function $91 defines 6 per-device query register, F91_2D_Query0 through F91_2D_Query5.

20.2.1. F91_2D_Query0: Sensor Properties

Name	7	6	5	4	3	2	1	0
F91_2D_Query0	—	—	—	—	—	IsS1100	Has SPR	Has SPR Limit

Figure 131, Function $91 Per-sensor query register

Deleted: 132

Has Servo-Palm-Reject Limits (F91_2D_Query0, bit 0)

1 = If HasSPR is set, limits are in Function $91's control registers.
0 = If HasSPR is set, limits are in Function $08's control registers.
This bit should be set for all new builds.

Has Servo Palm Reject (F91_2D_Query0, bit 1)

1 = Firmware does not servo while a palm is on the sensor.
0 = It does.
This bit should be set for all new builds.

IsS1100 (F91_2D_Query0, bit 2)

1 = Registers F91_2D_Ctrl15 through F91_2D_Ctrl34 exist
0 = These registers do not exist.

20.2.2. F91_2D_Query1: Position Correction Table Size

Name	7	6	5	4	3	2	1	0
F91_2D_Query1	—	Has Per-Axis PosCorr	Position Correction Table Size					

Figure 132, Function $91 Per-sensor query register

Deleted: 133

PositionCorrectionTableSize (F91_2D_Query1, bits 5:0)

The position correction corrects for non-linear behavior in the position interpolation algorithm. The table's size is fixed at compile time, and its contents are product-specific.

HasPerAxisPosCorr (F91_2D_Query1, bit 6)

This bit indicates the number of correction tables used:

- '0' means that the correction table is shared by both axes.
- '1' means that each axis has its own correction table. The correction table size is identical for each axis; the CorrectionTableSize field in bits 5:0 reports the size of one axis's correction table. The two tables are placed contiguously in memory: the first table is for the X axis and the second is for the Y axis.

20.2.3. F91_2D_Query2: Miscellaneous

Name	7	6	5	4	3	2	1	0
F91_2D_Query2	—	—	—	HasZW	Has WTF	Has W CorrTable	Has Z CorrTable	Has LCCS CorrTable

Figure 133. Function $91 Per-sensor query register

Deleted: 134

Has LCCS Correction Table (F91_2D_Query2, bit 0)

If this bit is '1', the sensor supports a tri-linear correction mechanism, and a tri-linear correction table exists at F91_2D_Ctrl11.*.

Has Z Correction Table (F91_2D_Query2, bit 1)

If this bit is '1', the sensor supports a Z correction mechanism, and a Z correction table exists at F91_2D_Ctrl12.*.

Has W Correction Table (F91_2D_Query2, bit 2)

If this bit is '1', the sensor supports a W correction mechanism, and a W correction table exists at F91_2D_Ctrl13.*.

HasWTF (F91_2D_Query2, bit 3)

Reports the availability of the W Threshold Factor control register at F91_2D_Ctrl14.

HasZW (F91_2D_Query2, bit 4)

Reports the availability of the ZW Threshold data register at F91_2D_Data2.

20.2.4. F91_2D_Query3 through F91_2D_Query5: Correction Table Sizes

Name	7	6	5	4	3	2	1	0
F91_2D_Query3	—	—	—	LCCS Correction Table Size				
F91_2D_Query4	—	—	Has Per-Axis ZCorr	Z Correction Table Size				
F91_2D_Query5	—	—	Has Per-Axis WCorr	W Correction Table Size				

Figure 134. Function $91 Per-sensor query register

Deleted: 135

LCCS Correction Table Size (F91_2D_Query3, bits 4:0)

Describes the length of the LCCS correction table. A length of 0 indicates that no table is present.

Note: Unlike the Z, W, and position correction tables, it is not possible for each axis to have its own LCCS correction table.

Z Correction Table Size (F91_2D_Query4, bits 4:0)

Describes the length of the Z correction table. A length of 0 indicates that no table is present. The size field refers to the size of each table.

HasPerAxisZCorr (F91_2D_Query4, bit 5)

This bit indicates the number of correction tables used:

- '0' means that the correction table is for a single axis, or is shared by both axes.
- '1' means that each axis has its own correction table. The correction table size is identical for each axis; the ZCorrectionTableSize field in bits 4:0 reports the size of one axis's correction table. The two tables are placed contiguously in memory: the first table is for the X axis and the second is for the Y axis.

W Correction Table Size (F91_2D_Query5, bits 4:0)

Describes the length of the W correction table. A length of 0 indicates that no table is present. The size field refers to the size of each table.

HasPerAxisWCorr (F91_2D_Query5, bit 5)

This bit indicates the number of correction tables used:

- '0' means that the correction table is for a single axis, or is shared by both axes.
- '1' means that each axis has its own correction table. The correction table size is identical for each axis; the WCorrectionTableSize field in bits 4:0 reports the size of one axis's correction table. The two tables are placed contiguously in memory: the first table is for the X axis and the second is for the Y axis.

20.3. Function $91: control registers

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

Name	7	6	5	4	3	2	1	0
F91_2D_Ctrl0	WxScale							
F91_2D_Ctrl1	WyScale							
F91_2D_Ctrl2	—	—	—	—	—	—	noRelax	noRezero
F91_2D_Ctrl3	ZTouchThreshold							
F91_2D_Ctrl4	ZReleaseThreshold							
F91_2D_Ctrl5	ZClipThreshold							
F91_2D_Ctrl6.*	Position Correction Table							
F91_2D_Ctrl7	WidthOffsetX							
F91_2D_Ctrl8	WidthOffsetY							
F91_2D_Ctrl9	PalmRejectXrefHi							
F91_2D_Ctrl10	PalmRejectYrefHi							
F91_2D_Ctrl11.*	LCCS Correction Table							
F91_2D_Ctrl12.*	Z Correction Table							
F91_2D_Ctrl13.*	W Correction Table							
F91_2D_Ctrl14	—	—	—	—	W Threshold Factor			
F91_2D_Ctrl15.*	Gains							
F91_2D_Ctrl16	Global Stretch Center Shift LSB							
F91_2D_Ctrl17	Global Stretch Center Shift MSB							
F91_2D_Ctrl18	Global Stretch Stretch Factor							
F91_2D_Ctrl19	Stretch Left Bound LSB							
F91_2D_Ctrl20	Stretch Left Bound MSB							
F91_2D_Ctrl21	Stretch Right Bound LSB							
F91_2D_Ctrl22	Stretch Right Bound MSB							
F91_2D_Ctrl23	Stretch Left Linear Factor							
F91_2D_Ctrl24	Stretch Left Quadratic Factor							
F91_2D_Ctrl25	Stretch Right Linear Factor							
F91_2D_Ctrl26	Stretch Right Quadratic Factor							
F91_2D_Ctrl27	Wings (Tails) Correction Left Bound LSB							
F91_2D_Ctrl28	Wings (Tails) Correction Left Bound MSB							
F91_2D_Ctrl29	Wings (Tails) Correction Right Bound LSB							
F91_2D_Ctrl30	Wings (Tails) Correction Right Bound MSB							
F91_2D_Ctrl31	Wings (Tails) Correction Left Linear Factor							
F91_2D_Ctrl32	Wings (Tails) Correction Left Quadratic Factor							
F91_2D_Ctrl33	Wings (Tails) Correction Right Linear Factor							
F91_2D_Ctrl34	Wings (Tails) Correction Right Quadratic Factor							

Figure 135. Function $91 control registers

Deleted: 136

WxScale (F91_2D_Ctrl0)
WyScale (F91_2D_Ctrl1)

W is a measure of the width of the finger profile. The W scaling factors convert finger profiles widths into normalized widths.

noRezero (F91_2D_Ctrl2, bit 0)

If set to '1', the internal automatic re-zeroing algorithm is inhibited.

noRelax (F91_2D_Ctrl2, bit 1)

If set to '1', the internal automatic relaxation algorithm is inhibited.

ZTouchThreshold (F91_2D_Ctrl3)

Specifies the threshold the finger Z must exceed before the finger is considered to be touching. A large Z threshold means that smaller fingers are more likely rejected.

ZReleaseThreshold (F91_2D_Ctrl4)

Specifies the threshold below which the finger Z must fall before the finger is considered to be not touching. The difference between *ZTouchThreshold* and *ZReleaseThreshold* is called the Z hysteresis. A large hysteresis reduces the chances of inadvertent drop outs in noisy environments.

Note: *ZReleaseThreshold* must be smaller than *ZTouchThreshold.*

ZClipThreshold (F91_2D_Ctrl5)

Specifies the threshold below which the finger Z must fall before data reporting is halted. This is only used if the *ReportingMode = '101'* (Z-clip reporting mode).

PositionCorrectionTable (F91_2D_Ctrl6.)*

The size of this area is defined by query register F91_2D_Query1. If the Correction Table size is reported as 0, then no F91_2D_Ctrl6.* registers will be present. This block of control registers contains the lookup table that is used to correct for device-dependent position non-linearities.

WidthOffsetX (F91_2D_Ctrl7)

WidthOffsetY (F91_2D_Ctrl8)

WidthOffsetX and *WidthOffsetY* are the signed, 8-bit parameters for the linear scaling algorithm used to convert the raw, per-axis internal Width value to the Width reported to the host in F11. *WidthOffsetX* and *WidthOffsetY* are the additive part of the linear scaling for the X and Y axis respectively.

PalmRejectXrefHi (F91_2D_Ctrl9)

PalmRejectYrefHi (F91_2D_Ctrl10)

Important: These 8-bit parameters are used by the checkServo routine to prevent an infinite loop when a palm is present during servoing. If the servo routine ever tries to servo to a value larger than the reject threshold, it will indicate that servoing was NOT successful, and should be tried again (presumably after the palm is removed). If servo type is 'single-axis', only the X limit register is defined. If servo type is 'dual-axis' (the default), both X and Y limit registers are defined.

Note: All current firmware is dual-axis. There is no query bit to indicate that a firmware is single or dual axis.

LCCS Correction Table (F91_2D_Ctrl11.)*

This is the tri-linear correction table replicated register. This register is only present if the F91_2D_Query2, bit 0 is '1'. The replicated register length is defined by F91_2D_Query3. A length of 0 indicates that no F91_2D_Ctrl11.* registers are present.

Z Correction Table (F91_2D_Ctrl12.)*

This is the Z parameter correction table replicated register. This register is only present if the F91_2D_Query2, bit 1 is '1'. The replicated register length is defined by F91_2D_Query4. A length of 0 indicates that no F91_2D_Ctrl12.* registers are present.

W Correction Table (F91_2D_Ctrl13.)*

This is the W parameter correction table replicated register. This register is only present if the F91_2D_Query2, bit 2 is '1'. The replicated register length is defined by F91_2D_Query5. A length of 0 indicates that no F91_2D_Ctrl13.* registers are present.

WThresholdFactor (F91_2D_Ctrl14, bits 3:0)

This sets the threshold used in the 2D W calculation. Range is 0x1-0xF, which corresponds to a threshold between 1/16 span and 15/16 span; 0x0 is undefined. Default value is 0x8.

Gains (F91_2D_Ctrl15.)*

Gains (unsigned 1.7). Two registers per receiver electrode.

Global Stretch Center Shift LSB (F91_2D_Ctrl16)

Global Stretch Center Shift MSB (F91_2D_Ctrl17)

Global Stretch Center Shift (signed 5.11)

Global Stretch Stretch Factor (F91_2D_Ctrl18)

Global Stretch Stretch Factor (unsigned 1.7)

Stretch Left Bound LSB (F91_2D_Ctrl19)

Stretch Left Bound MSB (F91_2D_Ctrl20)

Stretch Left Bound (unsigned 5.11)

Stretch Right Bound LSB (F91_2D_Ctrl21)

Stretch Right Bound MSB (F91_2D_Ctrl22)

Stretch Right Bound (unsigned 5.11)

Stretch Left Linear Factor (F91_2D_Ctrl23)

Stretch Left Linear Factor (signed 2.6)

Stretch Left Quadratic Factor (F91_2D_Ctrl24)

Stretch Left Quadratic Factor (signed 2.6)

Stretch Right Linear Factor (F91_2D_Ctrl25)

Stretch Right Linear Factor (signed 2.6)

Stretch Right Quadratic Factor (F91_2D_Ctrl26)

Stretch Right Quadratic Factor (signed 2.6)

Wings (Tails) Correction Left Bound LSB (F91_2D_Ctrl27)

Wings (Tails) Correction Left Bound MSB (F91_2D_Ctrl28)

Wings (Tails) Correction Left Bound (unsigned 8.8)

Wings (Tails) Correction Right Bound LSB (F91_2D_Ctrl29)

Wings (Tails) Correction Right Bound MSB (F91_2D_Ctrl30)

Wings (Tails) Correction Right Bound (unsigned 8.8)

Wings (Tails) Correction Left Linear Factor (F91_2D_Ctrl31)

Wings (Tails) Correction Left Linear Factor (signed 2.6)

Wings (Tails) Correction Left Quadratic Factor (F91_2D_Ctrl32)

Wings (Tails) Correction Left Quadratic Factor (signed 2.6)

Wings (Tails) Correction Right Linear Factor (F91_2D_Ctrl33)

Wings (Tails) Correction Right Linear Factor (signed 2.6)

Wings (Tails) Correction Right Quadratic Factor (F91_2D_Ctrl34)

Wings (Tails) Correction Right Quadratic Factor (signed 2.6)

20.4. Function $91: data registers

Function $91's data registers report unfiltered Zx and Zy in addition to filtered Z for 2-D tuning.

20.4.1. Data register layout

For each sensor in a device, there will be a set of Zx/Zy/ZW registers. For example, if a device contains three 2-D sensors, there will be three sets of Zx/Zy/ZW registers. But the register layout allows only a single (the first) finger for each sensor to be reported for testing. The register map construction rules are applied as follows for each sensor defined:

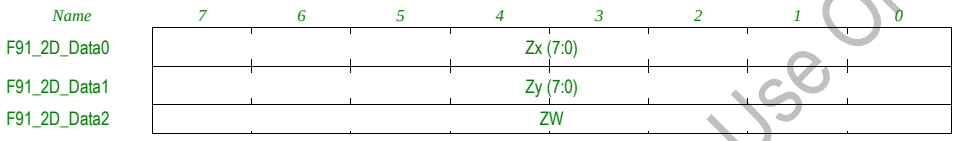

Figure 136, Function $91 data registers

Deleted: 137

Zx (F91_2D_Data0, bits 7:0)

This field reports the unfiltered amount of finger contact or finger signal strength as calculated using the X-axis sensor elements.

Zy (F91_2D_Data1, bits 7:0)

This field reports the unfiltered amount of finger contact or finger signal strength as calculated using the Y-axis sensor elements.

ZW (F91_2D_Data2, bits 7:0)

This register exists only if *HasZW* (F91_2D_Query2, bit 4) reports as '1'. This field is a signed, fixed-point (4.4) representation of the palm signal.

20.5. Function $91: interrupt source

Function $91 does not have an interrupt source. Use the public interrupts for Function $11 if necessary.

20.6. Function $91: command registers

Function $91 does not define any private commands.

21. Function $99: 0-D Functions (Private)

Function $99 implements the private registers for 0-D sensing buttons. The entire function is private and only those registers that need to be revealed to a customer should be. Some registers may be used by customers and set by Synaptics tools, but the customer should have no need to change them outside of this tool set.

21.1. Function $99: query registers

Function $99 contains one query register, which is completely reserved.

21.2. Function $99: control registers

These registers control the operation of the capacitive buttons.

Name	7	6	5	4	3	2	1	0
F99_Btn_Ctrl0	InterferenceThreshold							
F99_Btn_Ctrl1	—	—	—	—	—	—	NoRelax	NoRezero

Figure 137, Function $99 control registers

Deleted: *138*

There is always exactly one InterferenceThreshold control register. There is always exactly one diagnostic control register (F99_Btn_Ctrl1 in the example above).

21.2.1. F99_Btn_Ctrl0: button interference metric control register

The noise processing mechanisms implemented by F$19 capacitive buttons require the specification of an interference metric. This value will be generated by tuning tools, and will typically not be customer visible, or adjustable. The format of the data in this register is TBD.

21.2.2. F99_Btn_Ctrl1: button diagnostic control

This register allows access to certain algorithmic controls as they relate to button processing.

NoRezero (F99_Btn_Ctrl1, bit 0)

This field disables the automatic rezeroing mechanism. The reset default is ‘0’, meaning that automatic rezeroing is enabled.

NoRelax (F99_Btn_Ctrl1, bit 1)

This field disables the relaxation processing algorithm. The reset default is ‘0’, meaning that relaxation is enabled.

21.3. Function $99: data registers

Function $99 does not implement any data registers.

21.4. Function $99: command registers

Function $99 does not implement any command registers.

22. Function $D4: Analog Diagnostic for T1321 Devices (Private)

Function $D4 provides private analog diagnostic functions that allow visibility and control of the base analog functionality used for trans-capacitive image sensing.

Important: For functionality that exists in both Function $D4 and Function $54, please use the Function $54 iteration. Function $D4 is being deprecated.

In general, Function $D4 diagnostics are intended to provide information about and control of the operation of the touch controller and core algorithms without impeding the normal operation of other functions.

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

- Controls for algorithms, such as spatial filtering for strips that are only present in functions for certain devices, are implemented in their own function.
- Scaling Sensitivities (not Response Correction Factors) for the electrodes within a function are controlled by that function.
- Any per-channel Z reporting (for example, Zdelta or Zabs) may be implemented within Analog Diagnostics while function specific Z reporting (such as Zx, Zy) will be implemented in that function.
- Fine-grain control of zeroing or relaxation is controlled within a particular function; the Analog Diagnostic function only implements those features that are expected to appear in every diagnostic function, although it may contain global controls that affect all functions, but do not override the fine-grain control.

Note: See the BIST Function $09 for higher-level and more automated testing functions.

All devices should support basic analog diagnostic controls. More advanced controls may be implemented selectively, or implemented by directly reading, writing, or modifying hardware registers. In general, the intent of F$D4 is to provide a means to coherently read the ADC data in its various forms in synchronization with the operational firmware.

F$D4 provides an interface to the sensing hardware based on trans-capacitive sensing with drive electrodes and sense electrodes. It does not map that to an X, Y coordinate space.

All registers of the Analog Diagnostic function are considered private unless they specifically need to be released to a customer. An unlock sequence (or bit) has been provided to prevent any spurious write to the control or command registers of the Analog Diagnostic function, which could result in incorrect data and difficult-to-diagnose problems.

22.1. Function $D4: query registers

The query registers describe the available electrodes, modes, and function capabilities available in this product.

Name	7	6	5	4	3	2	1	0
FD4_AD_Query0	NumberOfSenseElectrodesAvailable							
FD4_AD_Query1	NumberOfDriveElectrodesAvailable							
FD4_AD_Query2	—	HasImage16	Has AutoServo	HasBitmap	HasImage8	HasBaseLine	—	HasADCtrl
FD4_AD_Query3	—	HasIM2	—	—	—	—	—	—
FD4_AD_Query4	—							
F11_2D_Query5	Reference Capacitor Calibration							—
FD4_AD_Query6	—				NumberOfSenseFrequencies			

Figure 138. Function $D4 query registers

Deleted: 139

The bits of these registers are defined as follows:

NumberOfSenseElectrodesAvailable (FD4_AD_Query0, bits 7:0)

Reports the number of sense electrodes available on the design.

NumberOfDriveElectrodesAvailable (FD4_AD_Query1, bits 7:0)

Reports the number of drive electrodes available on the design.

HasADCtrl (FD4_AD_Query2, bit 0)

If ‘1’, then Analog Diagnostic Control is available. If ‘0’, then no writable registers are available within Function $D4. In this case, query, data, and control registers may still be read.

HasBaseline (FD4_AD_Query2, bit 2)

This field indicates the availability of the Baseline reporting mode. This report is made through the data registers FD4_AD_Data2.*:

- ‘0’: Baseline reporting is not supported.
- ‘1’: Baseline reporting is supported.

HasImage8 (FD4_AD_Query2, bit 3)

This field indicates the availability of the Image Line reporting mode for 8-bit images. This report is made through the data registers FD4_AD_Data2.*:

- ‘0’: 8-bit Image Line reporting is not supported.
- ‘1’: 8-bit Image Line reporting is supported.

HasBitmap (FD4_AD_Query2, bit 4)

This field indicates the availability of the Bitmap Image reporting mode. This report is made through the data registers FD4_AD_Data2.*:

- ‘0’: Bitmap Image reporting is not supported.
- ‘1’: Bitmap Image reporting is supported.

HasAutoServo (FD4_AD_Query2, bit 5)

This field indicates that the firmware supports auto-servo operations:

- '0': Autoservo is not supported.
- '1': Autoservo is supported.

HasImage16 (FD4_AD_Query2, bit 6)

This field indicates the availability of the Image Line reporting mode for 16-bit images. This report is made through the data registers FD4_AD_Data2.*:

- '0': 16-bit Image Line reporting is not supported.
- '1': 16-bit Image Line reporting is supported.

HasIM2 (FD4_AD_Query3, bit 6)

This field specifies availability of a secondary Interference Metric and of the Ctrl6, Ctrl7, Data10, and Data11 registers::

- '0': The extra registers do not exist.
- '1': The secondary Interference Metric is available and the Ctrl6, Ctrl7, Data10, and Data11 registers are used.

Reserved (FD4_AD_ Query4)

Reserved.

ReferenceCapacitorCalibration (FD4_AD_ Query5, bits 7:0)

This register is a placeholder for calibration of the reference capacitors at test time.

NumberOfSenseFrequencies (FD4_AD_ Query6, bits 3:0)

Specifies the number (-1) of available analog sensing frequencies.

22.2. Function $D4: control registers

The control registers provide a mechanism for configuring the Analog Diagnostic registers and algorithms. They also control the reporting and configuration of the data and command registers.

Name	7	6	5	4	3	2	1	0
FD4_AD_Ctrl0	NoAutoServo	NoOscMod	NoAutoZero	NoRelax	NoScan	NoBleedover	NoFreqShift	NoCMNR
FD4_AD_Ctrl1	NoPosCorr	—	—	—	—	NoSpatial Filter	Spew	NukeOnReset
FD4_AD_Ctrl2	—	—	—	—	—	BurstsPerCluster		
FD4_AD_Ctrl3	InterferenceMetricThresholdLo							
FD4_AD_Ctrl4	InterferenceMetricThresholdHi							
FD4_AD_Ctrl5	Bitmap Threshold							
FD4_AD_Ctrl6	InterferenceMetric2ThresholdLo							
FD4_AD_Ctrl7	InterferenceMetric2ThresholdHi							

Figure 139. Function $D4 control registers

Deleted: 140

The bits of these registers are defined as follows:

NoCMNR (FD4_AD_Ctrl0, bit 0)

This bit disables the common-mode noise reduction algorithms for all channels. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoFreqShift (FD4_AD_Ctrl0, bit 1)

This bit disables automatic shifting of the analog sensing frequency. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoBleedover (FD4_AD_Ctrl0, bit 2)

This bit disables the LPF Bleedover algorithm. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoScan (FD4_AD_Ctrl0, bit 3)

This bit causes the image acquisition code to stop writing to the XMTR_INP0, XMTR_INP12 and XMTR_INP24 registers, which allows unfettered access to the diagnostic host. During normal operation, the RCVR_EN, XMTR_EN, and XMTR_POL registers are static and do not over-write the image acquisition, so this bit does not change the behavior of those registers.

NoRelax (FD4_AD_Ctrl0, bit 4)

This bit disables relaxation algorithms for all channels. This field is ORed with any per-function relaxation control when set, and this field is ANDed when cleared. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoAutoZero (FD4_AD_Ctrl0, bit 5)

This bit disables auto-zeroing (lift recovery) algorithms for all channels. This field is ORed with any per-function re-zero control when set and ANDed in when reset. Clearing the bit should not cause an auto-zero event. This field may be overloaded to disable relaxation algorithms for all channels, if *NoRelax* is unimplemented. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoOscMod (FD4_AD_Ctrl0, bit 6)

This bit disables the oscillator-modulation algorithm. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoAutoServo (FD4_AD_Ctrl0, bit 7)

This bit disables auto-servo-ing (for example, channel clip recovery) algorithms at the time when they would be initiated and should not terminate calibration after it has begun. This bit should be set when changing RefCap registers. Clearing this bit should not cause a servo event. This bit may be overloaded to disable the auto-sensitivity correction algorithms for all channels, which updates the Sensitivity Correction Factor registers, if NoAutoCalibration is unimplemented. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NukeOnReset (FD4_AD_Ctrl1, bit 0)

This diagnostic bit enables all other nuke bits ("No*Xxxxxxx*" bits) on Reset events.

Spew (FD4_AD_Ctrl1, bit 1)

Setting this bit to 1 causes the ATTN signal to be asserted on every frame, regardless of whether there is any new data to report.

NoSpatialFilter (FD4_AD_Ctrl1, bit 2)

This bit disables the spatial filter. At reset this bit is ignored unless the *NukeOnReset* bit is set.

NoPosCorr (FD4_AD_Ctrl1, bit 7)

This bit disables the position-correction tables. At reset this bit is ignored unless the *NukeOnReset* bit is set.

BurstsPerCluster (FD4_AD_Ctrl2, bits 2:0)

This field specifies the number of ADC conversions per cluster. One full image frame is captured each *(BurstsPerCluster * NumberOfDriveElectrodesAvailable)* ADC conversions. Most factors of the ADC conversion rate are controlled by analog hardware configuration registers which can be written directly using FD4_AD_ Data3. This field controls the only factor of image frame rate that is controlled by firmware implementation, as opposed to analog hardware configuration. Depending on firmware implementation, this register may or may not be writable.

InterferenceMetricThresholdLo (FD4_AD_Ctrl3)

This field sets the *Lo* bits of the threshold for the main interference metric. Any other interference metric thresholds will be set in the relevant function. Sensing should be disabled or ignored (for example, before POR) when changing the threshold.

InterferenceMetricThresholdHi (FD4_AD_Ctrl4)

This field sets the *Hi* bits of the threshold for the main interference metric. Any other interference metric thresholds will be set in the relevant function. Sensing should be disabled or ignored (for example, before POR) when changing the threshold.

Bitmap Threshold (FD4_AD_Ctrl5)

This field sets the threshold for the frame image bitmap. Each pixel of the capacitance image map is compared to this value to convert the pixel value into a binary value. The value of this threshold register is usually tuned for a specific module and not intended to be changed in normal usage. This register may or may not be writable in any given firmware implementation. When changeable, it should be changed primarily to compensate for changes in other diagnostic registers such as *BurstsPerCluster* (FD4_AD_ Ctrl2, bits 2:0).

Note: *BitmapThreshold* is not implemented.

InterferenceMetric2ThresholdLo (FD4_AD_Ctrl6)

This field sets the *Lo* bits of the threshold for the secondary interference metric. Sensing should be disabled or ignored (for example, before POR) when changing the threshold.

Note: This register exists only if *HasIM2* (FD4_AD_Query3, bit 6) is set to '1'.

InterferenceMetric2ThresholdHi (FD4_AD_Ctrl7)

This field sets the *Hi* bits of the threshold for the secondary interference metric. Sensing should be disabled or ignored (for example, before POR) when changing the threshold.

Note: This register exists only if *HasIM2* (FD4_AD_Query3, bit 6) is set to '1'.

22.3. Function $D4: data registers

Name	7	6	5	4	3	2	1	0
FD4_AD_Data0	—	—	—	—	—	ReportMode		
FD4_AD_Data1	FIFOIndexLo							
FD4_AD_Data2	FIFOIndexHi							
FD4_AD_Data3	FIFOData							
FD4_AD_Data4	FrequencyShiftCount							
FD4_AD_Data5	PipelineStallCount							
FD4_AD_Data6	RezeroCount							
FD4_AD_Data7	MissedCviCount							
FD4_AD_Data8	InterferenceMetricLo							
FD4_AD_Data9	InterferenceMetricHi							
FD4_AD_Data10	InterferenceMetric2Lo							
FD4_AD_Data11	InterferenceMetric2Hi							
FD4_AD_Data12	—				SenseFreqSelect			

Figure 140. Function $D4 data registers

Deleted: 141

The bits of these registers are defined as described below.

ReportMode (FD4_AD_Data0, bits 2:0)

Specifies the report mode for the data in the *FIFOData* register.

ReportMode = 0: Image Bitmap

This report mode is used to read the full Image Bitmap, which is the thresholded binary version of the Image. The bitmap is composed of *NumberOfDriveElectrodesAvailable* rows, each of which is *NumberOfSenseElectrodesAvailable* bits wide (rounded up to the nearest multiple of 16). The number of bytes in the bitmap is:

$$n_{Bitmap\ Bytes} = NumberOfDriveElectrodesAvailable * FLOOR((NumberOfSenseElectrodesAvailable+15)/16) * 2$$

For a sensor with 20 drive electrodes and 28 sense electrodes, the bitmap contains 20 rows of 32 bits each, or 80 bytes.

ReportMode = 1: 16-bit Baseline

This report mode is used to read the Baseline, representing the base capacitance for each pixel location. The baseline is composed of *NumberOfDriveElectrodesAvailable* rows, each of which is *NumberOfSenseElectrodesAvailable* 16-bit words wide (low-order byte first). The number of bytes in the baseline is:

$$n_{Baseline\ Bytes} = NumberOfDriveElectrodesAvailable * NumberOfSenseElectrodesAvailable * 2$$

For a sensor with 20 drive electrodes and 28 sense electrodes, the baseline contains 20 rows of 56 bytes each, or 1120 bytes.

ReportMode = 2: 8-bit Image

This report mode is used to read the Image as 8-bit pixel data; each byte represents the capacitance variance from the baseline for a pixel location. The image is composed of *NumberOfDriveElectrodesAvailable* rows, each of which is *NumberOfSenseElectrodesAvailable* bytes wide. The number of bytes in the 8-bit image is:

$$N_{\text{8-bit Image Bytes}} = \textit{NumberOfDriveElectrodesAvailable} * \textit{NumberOfSenseElectrodesAvailable}$$

For a sensor with 20 drive electrodes and 28 sense electrodes, the 8-bit image contains 20 rows of 28 bytes each, or 560 bytes.

ReportMode = 3: 16-bit Image

This report mode is used to read the Image as 16-bit pixel data; each 16-bit word represents the capacitance variance from the baseline for a pixel location. The image is composed of *NumberOfDriveElectrodesAvailable* rows, each of which is *NumberOfSenseElectrodesAvailable* 16-bit words wide (low-order byte first). The number of bytes in the 16-bit image is:

$$N_{\text{16-bit Image Bytes}} = \textit{NumberOfDriveElectrodesAvailable} * \textit{NumberOfSenseElectrodesAvailable} * 2$$

For a sensor with 20 drive electrodes and 28 sense electrodes, the 16-bit image contains 20 rows of 56 bytes each, or 1120 bytes.

ReportMode = 4: Hardware Register Window

This report mode is used to read the T1321 Analog registers at T1321 addresses 0xFF40 through 0xFF7F. Each of the 64 16-bit registers is represented by a coherent pair of bytes (low-order byte first), so the number of bytes in the Hardware Register Window is:

$$N_{\text{Hardware Register Window Bytes}} = 64 * 2 = 128$$

Note: Because each pair of bytes is coherent, each T1321 register is modified only after the host has written to both its LSB and MSB.

ReportMode = 5 through 7: Reserved

FIFOIndexLo and FIFOIndexHi (FD4_AD_Data1 and FD4_AD_Data2)

This pair of registers is a bytewise pointer to the image data that is being read from the FIFOData register. It is cleared to 0x0000 on reset and upon an Image Complete interrupt. In addition, writing a value into this 2-byte register provides an optional way to change the position within the image buffer from which the image data is being read. While access to the image through this FIFO facility is sequential, the ability to specify the byte index adds a certain level of random access for applications that need it.

Reading this pair of registers will return the current value of the pointer.

FIFOData (FD4_AD_Data3)

Repeated reads of this single register provide consecutive bytes of the image data by auto-incrementing an internal pointer [optionally set through *FIFOIndexLo* and *FIFOIndexHi*]. For 16-bit data, the least significant byte is read first. The content and format of the data present in this FIFO register changes based on the contents of the *ReportMode* field (see the description of modes there).

Note: Setting *FIFOIndex* to a value that is outside the valid range for a given mode, or reading the FIFO past the end of valid data, will result in undefined behavior.

Important: This register must be read by specifying its exact address, in which case its address will not be auto-incremented upon repeated reads: Repeated reads will auto-increment the internal pointer so that subsequent image data can be accessed.

On the other hand, this register will read 0 if reached as the result of address auto-increment during an RMI block read. Subsequent reads within the block will leave the FIFO pixel counter unchanged, and move on to the data register past *FIFOData*.

Comment [MSOffice5]: In the next revision (Rev B), rewrite this section to be similar to the FIFO Address and FIFO Data registers in Function $81 (section 16.3.4)

FrequencyShiftCount (FD4_AD_Data4)

Read-only counter that increments whenever the noise-avoidance algorithm tries to shift the analog sensing frequency, regardless of whether the *NoFreqShift* bit prevents the shift from actually occurring.

PipelineStallCount (FD4_AD_Data5)

Read-only counter that increments whenever the analog-sensing pipelines transitions from the unstalled to the stalled state.

RezeroCount (FD4_AD_Data6)

Read-only counter that increments whenever the auto-rezero algorithm attempts to rezero the analog baseline, regardless of whether the *NoRezero* bit prevents the rezero from actually occurring.

MissedCviCount (FD4_AD_Data7)

Read-only counter that increments whenever a Conversion Interrupt is missed.

InterferenceMetricLo (FD4_AD_Data8)

Read-only register that holds the LSB of the current frame's Interference Metric.

InterferenceMetricHi (FD4_AD_Data9)

Read-only register that holds the MSB of the current frame's Interference Metric.

InterferenceMetric2Lo (FD4_AD_Data10)

Read-only register that holds the LSB of the current frame's secondary Interference Metric.

Note: This register exists only if *HasIM2* (FD4_AD_Query3, bit 6) is set to '1'.

InterferenceMetric2Hi (FD4_AD_Data11)

Read-only register that holds the MSB of the current frame's secondary Interference Metric.

Note: This register exists only if *HasIM2* (FD4_AD_Query3, bit 6) is set to '1'.

SenseFreqSelect (FD4_AD_Data12, bits 3:0)

Read/write field that indicates/selects the current analog sensing frequency. Range is 0 to *NumberOfSenseFrequencies* (FD4_AD_Query6, bits 3:0); values outside that range will cause undefined behavior.

22.4. Function $D4: interrupt sources

The Analog Diagnostic Data source can assert an interrupt request at the end of every report period (epoch) for which an interrupt source has been enabled. There are three interrupt sources for Function $D4 – ADC Conversion, Image completion, and Servo completion. The interrupts will be enabled and reported in Function $81.

When one of these three interrupts is enabled by setting the corresponding bit in the FD4_AD_Cmd0 register, the bit will remain set until the interrupt has occurred and then will be cleared automatically when the interrupt is asserted. When the interrupt has occurred, further analog conversions will be stalled until either one of two things occur: one of the three interrupt command bits is set, or the interrupt is cleared by reading the interrupt status register in Function $81.

No interrupts occur in the Analog Diagnostic Function $D4 by default; interrupts only occur when their corresponding enable bits are set and private-function interrupts are enabled in Function $81. When an interrupt occurs, the data source is indicated in the Function $81 Interrupt Status register.

22.5. Function $D4: command registers

The control registers provide a mechanism for issuing commands for the analog diagnostic system. These bits automatically-clear to zero when the requested operation is complete.

Name	7	6	5	4	3	2	1	0
FD4_AD_Cmd0	QuitCmds	—	FrcZero	EnZero IRQ	EnBitmap IRQ	EnBaseline IRQ	EnImage16 IRQ	EnImage8 IRQ

Figure 141: Function $D4 command register

Deleted: 142

The bits of this register are defined as follows:

EnImage8IRQ (FD4_AD_Cmd0, bit 0)

Setting this bit causes an interrupt to be generated on the completion of an 8-bit image acquisition. Prior to reading the image, this bit should be set and the host should wait for the resulting interrupt. In order to ensure coherency of the image during the reading process, new image acquisition is stalled until the next write to FD4_AD_Cmd0 if the current image buffer would otherwise be overwritten.

EnImage16IRQ (FD4_AD_Cmd0, bit 1)

Setting this bit causes an interrupt to be generated on the completion of a 16-bit image acquisition. Prior to reading the image, this bit should be set and the host should wait for the resulting interrupt. In order to ensure coherency of the image during the reading process, new image acquisition is stalled until the next write to FD4_AD_Cmd0 if the current image buffer would otherwise be overwritten.

EnBaselineIRQ (FD4_AD_Cmd0, bit 2)

Setting this bit generates an interrupt the next time it is safe to read a baseline image. Before reading a baseline image, this bit should be set and the host should wait for the resulting interrupt. In order to ensure coherency of the baseline image during the reading process, baseline updating is stalled until the next write to FD4_AD_Cmd0 if the current image buffer would otherwise be overwritten.

EnBitmapIRQ (FD4_AD_Cmd0, bit 3)

Setting this bit causes an interrupt to be generated on the completion of a bitmap image acquisition. Before reading a bitmap image, this bit should be set and the host should wait for the resulting interrupt. In order to ensure coherency of the bitmap image during the reading process, new bitmap image acquisition is stalled until the next write to FD4_AD_Cmd0.

EnZeroIRQ (FD4_AD_Cmd0, bit 4)

This bit causes an interrupt to occur the next time a rezeroing occurs (the next time a new baseline is taken and stored). This can be used in combination with *FrcZero* to cause a rezero and then generate an interrupt when it occurs. This bit will remain set to '1' until the rezero operation is completed, at which point the bit will be automatically cleared to '0'.

FrcZero (FD4_AD_Cmd0, bit 5)

This bit causes a new baseline to be taken and stored. This can be used in combination with *EnZeroIRQ* to cause a rezero and then generate an interrupt when it occurs. This bit will remain set to '1' until the rezero operation is completed, at which point the bit will be automatically cleared to '0'.

QuitCmds (FD4_AD_Cmd0, bit 7)

This bit causes all pending commands to be canceled and their respective command bits to be cleared to '0'. When no command is pending, the host may set this bit to '1' to signal that no more commands will be issued, so any data-acquisition stall forced by the previous command can be canceled.

22.6. Reading images with Function $D4

This section describes the process for reading capacitance images with Function $D4. When reading images, keep in mind the following rules:

- When image reporting is enabled, you must not read any RMI registers outside of Function $D4 that are part of a coherent group. If you do, the captured image will be corrupted and should be discarded.
- Once an image has been captured, no new image will be acquired until you enable the next Analog Diagnostic interrupt by writing into the FD4_AD_Cmd0 register.
- If maintaining the frame rate is more important than the coherency of the report data, the next interrupt can be enabled before moving all the data for the current frame.

To read an image:

1. Disable low-power/dozing by setting the *NoSleep* bit of F01_RMI_Ctrl0.
2. Although not required, it is strongly suggested to disable all public interrupts so that only the ADC reports can cause ATTN to be asserted. To do this, write 0x00 to each of the F01_RMI_Ctrl1.* registers.

   Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), this step may be skipped.
3. Clear public interrupt status by reading F01_RMI_Data1.00.

   Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), this step may be skipped.
4. Unlock private functions via two consecutive writes to Device Status:
   - Write 0x42 to F01_RMI_Data0
   - Write 0xE1 to F01_RMI_Data0
5. If you have not already parsed the private PDTs, do so now in order to calculate the bitmask of Function $D4's Analog interrupt for use in the next step.

   Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), this step may be skipped.
6. Enable the private Analog interrupt by writing its bitmask into the correct private Interrupt Enable register. To do this, write the bitmask to the appropriate F81_RMI_Data0.* register:
   - If bitmask is in the range 0x0000-0x00FF, write it to F81_RMI_Data0.0
   - If bitmask is in the range 0x0100-0x8000, shift it right by 8 and write it to F81_RMI_Data0.1

   Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), this step may be skipped.
7. Set the Reporting Mode by writing to *ReportMode* (FD4_AD_Data0, bits 2:0):

- Bitmap: write 0x00
- 8-bit image: write 0x02
- 16-bit image: write 0x03

8. Request an Analog Diagnostic interrupt:
   - If reading an 8-bit image, set EnImage8IRQ (FD4_AD_Cmd0, bit 0) to '1'
   - If reading a 16-bit image, set EnImage16IRQ (FD4_AD_Cmd0, bit 1) to '1'
   - If reading an image bitmap, set EnBitmapIRQ (FD4_AD_Cmd0, bit 3) to '1'
9. When the interrupt occurs, determine its source by reading the interrupt status in F81_RMI_Data1. This will automatically clear the interrupt.
10. At this point the entire image is available, and some or all of the image data can be read by reading repeatedly from *FIFOData* (FD4_AD_Data3).
11. To read another image, repeat steps 8-10.
12. To exit image-reading mode, either reset the device or perform these steps:
    - Set the QuitCmds bit (FD4_AD_Cmd0, bit 7) to '1'
    - Disable the private Analog interrupt
    - Clear the private interrupt status by reading F81_RMI_Data1.00
    - (Optional) Exit the private function mode by setting the Configured bit in F01_RMI_Ctrl0
    - Re-enable the public interrupts by restoring the F01_RMI_Ctrl1.* register

22.7. Acquiring and reading baselines with Function $D4

This section describes the process for acquiring and reading the baseline with Function $D4. To acquire and read the baseline:

1. Disable low-power/dozing by setting the *NoSleep* bit of F01_RMI_Ctrl0.
2. Although not required, it is strongly suggested to disable all public interrupts so that only the ADC reports can cause ATTN to be asserted. To do this, write 0x00 to each of the F01_RMI_Ctrl1.* registers.

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

3. Clear public interrupt status by reading F01_RMI_Data1.00.

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

4. Unlock private functions via two consecutive writes to Device Status:
   - Write 0x42 to F01_RMI_Data0
   - Write 0xE1 to F01_RMI_Data0
5. If you have not already parsed the private PDTs, do so now in order to calculate the bitmask of Function $D4's Analog interrupt for use in the next step.

6. Enable the private Analog interrupt by writing its bitmask into the correct private Interrupt Enable register. To do this, write the bitmask to the appropriate F81_RMI_Data0.* register:
   - If bitmask is in the range 0x0000-0x00FF, write it to F81_RMI_Data0.0
   - If bitmask is in the range 0x0100-0x8000, shift it right by 8 and write it to F81_RMI_Data0.1

   Note: If the device does not have an ATTN line or if you prefer to ignore it and poll (not recommended), this step may be skipped.
7. Set the Reporting Mode by writing 0x01 to *ReportMode* (FD4_AD_Data0, bits 2:0).
8. Request an Analog Diagnostic interrupt:
   - To rezero and then read the new baseline: simultaneously write '1' to *EnZeroIRQ* (FD4_AD_Cmd0, bit 4) and *FrcZero* (FD4_AD_Cmd0, bit 5)
   - To read the current baseline without rezeroing first: write '1' to *EnZeroIRQ* (FD4_AD_Cmd0, bit 4)
9. When the interrupt occurs, determine its source by reading the interrupt status in F81_RMI_Data1. This will automatically clear the interrupt.
10. At this point a new baseline has been acquired and is available; its data can be read by reading repeatedly from *FIFOData* (FD4_AD_Data3).
11. To read another captured baseline frame, repeat steps 8-10.
12. To exit baseline-reading mode, either reset the device or perform these steps:
    - Set the QuitCmds bit (FD4_AD_Cmd0, bit 7) to '1'
    - Disable the private Analog interrupt
    - Clear the private interrupt status by reading F81_RMI_Data1.00
    - (Optional) Exit the private function mode by setting the Configured bit in F01_RMI_Ctrl0
    - Re-enable the public interrupts by restoring the F01_RMI_Ctrl1.* register

23. References: Synaptics literature

For more information, refer to the following Synaptics documents:

RMI4 Bootloader Procedures Application Note (PN 506-000221-01)

Flash Reprogramming Using RMI4 Function $34 (PN 511-000409-01)

ClearPad 3000 Platform Datasheet (PN 505-000167-01)

RMI4 Firmware Image File Description Application Note (PN 506-000215-01)

T1021 Reference Manual (PN 511-000291-01)

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.

Page 140: [1] Deleted	Donna Weatherall	8/30/2011 5:26:00 PM