This Datasheet Applies to Monza™ Part No. IPJ_W_R_(A and C)

Features

- EPCglobal-certified, assuring robust performance and seamless interoperability across the board.
- Dual antenna input maximizes range and provides for orientation indifference.
- High receptivity yields eight-meter read range, six-meter write range, and excellent tag sensitivity—even when buried deep within a pallet of RF-absorbing material.
- Write rate of >15 tags/second enables rapid programming throughput.
- Extended temperature range (–40° C to +65° C) for reliable performance under harsh conditions.
- Innovative interference rejection affords robust performance in noisy environments.
- Impinj's field-rewritable NVM (optimized for RFID) with 96-bit EPC offers programming flexibility and reliability.
- Available preprogramming of customer EPCs at the wafer level delivers a fast, reliable, and cost-effective turnkey manufacturing solution.
- A key component of the Impinj GrandPrix™ Solution, Monza™ tag silicon is *RFID that just works™*.

Overview

An essential component of Impinj's GrandPrix™ solution, Monza tag silicon is the industry's first to be granted the EPCglobal™ Mark of Certification, guaranteeing total standards conformance, and delivering the many features empowered by UHF Gen 2 conformance, including superior tag throughput and compliance with global spectral regulations. The EPCglobal Gen 2 specification is the ultimate standard for automatic identification requirements ranging from items to cases to pallets—worldwide. For inventory control, unique item tracking, logistics, product integrity, security, and data accuracy, the use of Monza-powered tags yields unprecedented supply chain visibility and confidence.

950110126000000087

Furthermore, Monza tag silicon establishes new benchmarks for range, readability, and high-speed field rewriteability. And in keeping with Impinj's ground-breaking quality standards, Monza chips are 100% factory tested.

Monza tag silicon also benefits from Impinj's innovative Self-Adaptive Silicon® core technology, which enables the creation of a true RFID system-on-a-chip integrating leading-edge analog, digital, and memory functions on a single die no larger than a grain of sand. It's a significant Impinj advantage that yields major performance, sourcing, and cost improvements. More importantly, Monza is the best-performing tag silicon available, exhibiting outstanding receptivity, as well as the ESD protection characteristics that are critical for ensuring inlay manufacturability at the highest assembly speeds.

RFID that just works™. *Everywhere.*

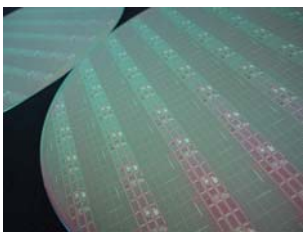
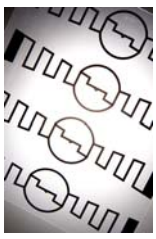




Table of Contents

Table of Contents....2
1 Introduction....7
1.1 Scope....7
1.2 Reference Documents....7
2 Functional Description....8
2.1 Monza™ Block Diagram....8
2.2 Pad Descriptions....8
2.3 Dual Antenna Input....8
2.4 Power Management....9
2.5 ESD Protection....9
2.6 Modulator/Demodulator....9
2.7 Tag Controller....9
2.8 Nonvolatile Memory....10
3 Interface Characteristics....11
3.1 Reader-to-Tag (Forward Link) Signal Characteristics....11
3.1.1 Forward Link Waveform....11
3.1.2 RF Power-up and Power-down Envelope....14
3.1.3 Tag Power-On Initialization....14
3.1.4 Preamble and Frame-Sync....15
3.2 Tag-to-Reader (Reverse Link) Backscatter....16
3.2.1 Reflection Coefficient....16
3.2.2 Modulation as Related to $|\triangle\Gamma|$....17
3.2.3 Radar Cross Section (RCS)....18
3.3 Source Impedance for Maximum Power Transfer....19
3.3.1 Monza™ Frequency Sensitivity....21
3.4 Reverse Link Signal Characteristics....22
3.4.1 Reverse Link Waveform....22
3.4.2 Baseband FM0 Data Modulation....22
3.4.3 Baseband FM0 Preambles....24
3.4.4 Miller-Modulated Subcarrier....25
3.4.5 Subcarrier Preambles....29
3.4.6 Link Frequency....29
3.4.7 Data Rate....29
3.4.8 Read Sensitivity....30
3.4.9 Write Sensitivity....30
3.5 Interface Protocols....31
3.5.1 Link Timing....31
3.5.2 Data Transmission Order....32
3.6 Command Interface....33
3.6.1 Standard Commands....33
3.6.2 Error Codes....34
4 State Machine Operation....35
4.1 Tag States....35
4.1.1 Ready State....35
4.1.2 Arbitrate State....35
4.1.3 Reply state....35
4.1.4 Acknowledged State....35
4.1.5 Open state....35

4.1.6 Secured State 36
4.1.7 Killed State 36
4.2 Monza™ State-Transition Tables 38
4.2.1 Present State: Ready 38
4.2.2 Present State: Arbitrate 39
4.2.3 Present State: Reply 40
4.2.4 Present State: Acknowledged 41
4.2.5 Present State: Open 42
4.2.6 Present State: Secured 43
4.2.7 Present State: Killed 45
4.3 Monza™ Flags 45
4.3.1 Selected Flag 45
4.3.2 Session Flags 46
4.4 Tag Commands 48
4.4.1 Global Commands 48
4.4.1.1 Select Command 48
4.4.2 Inventory Commands 50
4.4.2.1 Query 50
4.4.2.2 QueryAdjust 51
4.4.2.3 QueryRep 52
4.4.2.4 ACK command 52
4.4.2.5 NAK 53
4.4.3 Access Commands 53
4.4.3.1 Req_RN 53
4.4.3.2 Read 54
4.4.3.3 Write 56
4.4.3.4 Kill 57
4.4.3.5 Lock Command 60
4.4.3.6 Access Command 62
4.4.3.7 BlockWrite Command 65
4.5 Tag Memory 67
4.5.1 Memory Map 67
4.5.2 Logical vs. Physical Bit Identification 67
4.5.3 Memory Banks 67
4.5.3.1 Reserved Memory 68
4.5.3.2 Passwords 68
4.5.3.2.1 Access Password 68
4.5.3.2.2 Kill Password 68
4.5.3.3 Tag Identification (TID) Memory 68
4.5.3.4 EPC Memory 68
4.5.3.5 EPC CRC 68
4.5.3.6 CRC Generation and Decoding 68
4.5.3.6.1 CRC-16 68
4.5.3.6.2 CRC-5 69
4.6 ETSI Tag Spectral Mask 70
5 Absolute Maximum Ratings 71
5.1 Temperature 71
5.2 Input Damage Levels 71
5.3 NVM Use Model 71

6 Ordering Information 72

Index of Figures

Figure 2-1 Block Diagram 8
Figure 2-2 Monza™ Die Orientation 9
Figure 3-1 Modulation Characteristics 12
Figure 3-2 RF Envelope Characteristics 12
Figure 3-3 Relationship Between Baseband Signaling and RF Envelope 13
Figure 3-4 Power-up and Power-Down RF Envelope 14
Figure 3-5 Preamble and Frame-Sync Modulation Patterns 15
Figure 3-6 Simulated Tag Reflection Coefficient versus Incident RF Power in Modulator-Off and Modulator-On States 16
Figure 3-7 Measured Tag Delta-Gamma as a Function of Available Input Power 17
Figure 3-8 Tag Radar Cross Section (RCS) as a Function of $|\Delta\Gamma|$ (assumes half-wave dipole tag antenna and carrier frequency of 915 MHz) 18
Figure 3-9 Antenna Source Impedance for Maximum Power Transfer at 915 MHz, Showing –0.5 dB and –1.0 dB Sensitivity Loss Contours 20
Figure 3-10 Monza™ S_{11} versus Frequency 21
Figure 3-11 FM0 Generator State Diagram 23
Figure 3-12 FM0 Symbols and Sequences 24
Figure 3-13 Terminating FM0 transmissions 24
Figure 3-14 FM0 Preambles 25
Figure 3-15 Miller Baseband Generator State Diagram 26
Figure 3-16 Miller Baseband Symbols and Sequences 26
Figure 3-17 Miller-Modulated Subcarrier Frequencies 27
Figure 3-18: Terminating Subcarrier Transmissions 28
Figure 3-19: Miller-modulated subcarrier 29
Figure 3-20: Link Timing 31
Figure 4-1 Monza™ Tag State Diagram 37
Figure 4-2 Successful Write Sequence 57
Figure 4-3 Kill Procedure 59
Figure 4-4 Lock Command Payload Definitions 61
Figure 4-5 Access Procedure 64
Figure 4-6 Physical / Logical Memory Map 67
Figure 4-7: CRC-16 Encoder/Decoder 69
Figure 4-8: CRC-5 Encoder/Decoder 69
Figure 4-9 EN 302 208-1 Backscatter Spectral Mask 70

Index of Tables

Table 2-1 Pad Descriptions 8
Table 3-1 Forward Link Signal Parameters 11
Table 3-2 Power-up and Power-down Parameters 14
Table 3-3 Antenna Source Impedance for Maximum Power Transfer 19
Table 3-4 Reverse Link Signal Parameters 22
Table 3-5 Reverse-Link Tolerances 30
Table 3-6 Reverse Link Data Rates 30
Table 3-7 Link Timing Parameters 32
Table 3-8 Command Table Summary 33
Table 3-9 Tag-Error Reply Format 34

Table 3-10 Monza™ Error Codes 34
Table 4-1 Ready State-Transition 38
Table 4-2 Arbitrate State-Transition 39
Table 4-3 Reply State-Transition 40
Table 4-4 Acknowledged State-Transition 41
Table 4-5 Open State-Transition 42
Table 4-6 Secured State-Transition 43
Table 4-7 Killed State Transition 45
Table 4-8 Tag Flags and their Persistence 47
Table 4-9 Select Command 49
Table 4-10 Tag Response to Action Parameter 49
Table 4-11 Query Command 50
Table 4-12 Tag Reply to Query Command 50
Table 4-13 QueryAdjust Command 51
Table 4-14 Tag Reply to QueryAdjust Command 51
Table 4-15 QueryRep Command 52
Table 4-16 Tag Reply to QueryRep Command 52
Table 4-17 ACK Command 53
Table 4-18 Tag reply to successful ACK command 53
Table 4-19 NAK Command 53
Table 4-20 REQ_RN Command 54
Table 4-21 Tag Reply to REQ_RN Command 54
Table 4-22 Read Command 55
Table 4-23 Tag Reply to Read Command 55
Table 4-24 Tag-error reply format 55
Table 4-25 Write Command 57
Table 4-26 Tag reply to successful write command 57
Table 4-27 Kill Command 58
Table 4-28 – Tag reply to the first Kill command 58
Table 4-29 Tag reply to successful Kill procedure 58
Table 4-30 Lock Command 61
Table 4-31 Tag reply to Lock command 61
Table 4-32 Lock Action Field Functionality 62
Table 4-33 Access Command 63
Table 4-34 Tag reply to Access Command 63
Table 4-35 BlockWrite Command 66
Table 4-36 Tag Reply to BlockWrite Command 66
Table 4-39 CRC-16 Precurser 69
Table 5-1 Temperature parameters 71
Table 5-2 ESD and input limits 71

1 Introduction

1.1 Scope

This datasheet defines the physical and logical specifications for Gen 2-certified Monza™ tag silicon, a reader-talks-first, radio frequency identification (RFID) component operating in the UHF frequency range. Described are the physical interactions (the signaling layer of the communication link) between readers and Monza™-based tags, reader and tag operating procedures and commands, and the arbitration scheme used to identify a specific tag within a tag population.

1.2 Reference Documents

EPCglobal™ Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz (Gen 2 Specification)
Note: This specification includes normative references, terms and definitions, symbols, abbreviated terms, and notation, the conventions of which were adopted in the drafting of this document.

Impinj Wafer/Inlay Assembly Specification.

EPC™ Tag Data Specification

2 Functional Description

Described are the key functional blocks of the Monza™ tag silicon, as well as an overview of its operation within a typical application.

2.1 Monza™ Block Diagram

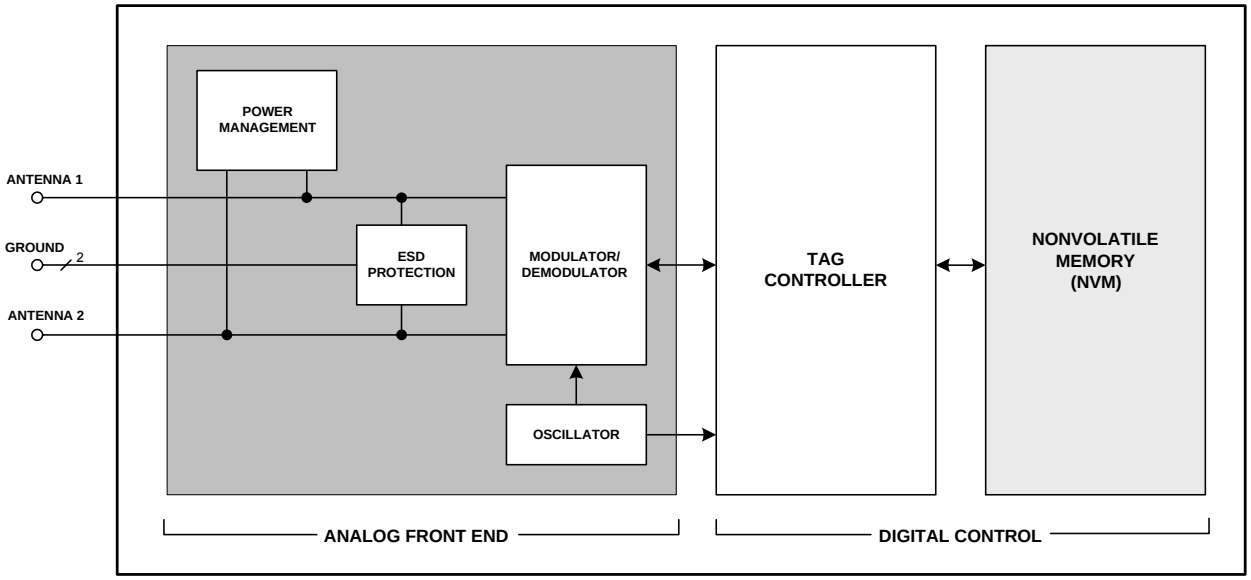


Figure 2-1 Block Diagram

2.2 Pad Descriptions

Monza™ tag silicon has four external pads available to the user: two antenna pads and two ground pads (the ground pads are internally strapped together), as shown in Table 2-1 (see also Figure 2-2).

Table 2-1 Pad Descriptions

External Signals	External Pad	Description
ANT_1	1	RF Input 1
ANT_2	1	RF Input 2
GND	1	Ground
GND	1	Ground

2.3 Dual Antenna Input

All interaction with Monza™ tag silicon, including generation of its internal power, air interface, negotiation sequences, and command execution, occurs via its two antenna ports and associated grounds.

The dual antenna inputs enable antenna diversity, which in turn minimizes a tag's orientation sensitivity, particularly when the two antennae are of different types (e.g., a combination of loop and dipole) or are otherwise oriented on different axis (X-Y). The dual antenna configuration also enables increased read and write ranges.

The two antenna inputs operate quasi-independently. The power management circuitry receives power from the electromagnetic field induced in the pair, and the demodulator exploits the independent antenna connections, combining the two demodulated antenna signals for processing on-chip.

Monza™ tag silicon may also be configured to operate using a single antenna port by simply connecting just one of the two inputs. The unused port may be connected to ground (to either, or both ground pads, as they are identical and connected on-chip) or allowed to float. The two ports should not, however, be connected to each other, as this will reduce power efficiency.

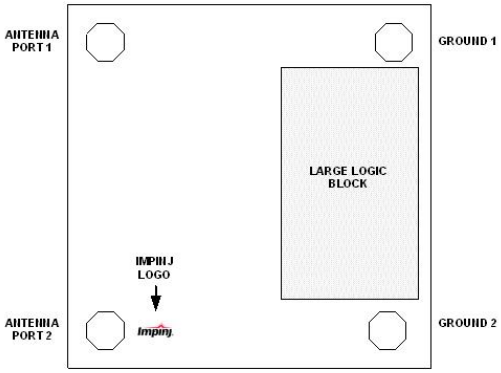


Figure 2-2 Monza™ Die Orientation

2.4 Power Management

The tag is activated by proximity to an active reader. When the tag enters a reader's RF field, the Power Management block converts the induced electromagnetic field to the DC voltage that powers the chip. (For conditions recommended for maximum power transfer, see section 3.3.)

2.5 ESD Protection

To divert ESD energy, the ESD Protection block shunts charge from both positive and negative sources when a high voltage is presented across the inputs, thus protecting the chip from damage.

2.6 Modulator/Demodulator

Monza™ tag silicon demodulates any of a reader's three possible modulation formats, DSB-ASK, SSB-ASK, or PR-ASK. The tag communicates back to a reader via backscatter of the incident RF waveform by switching the reflection coefficient of its antenna pair between reflective and absorptive states. Backscattered data is encoded as either FM0 or Miller subcarrier modulation (with the reader commanding both the encoding choice and the data rate).

2.7 Tag Controller

The preceding sections detail the analog functions of power management and signal acquisition and transmission. In the Tag Controller block, we enter the digital domain. While it also performs a number of overhead duties, the heart of this block is the finite state machine logic that carries out the command sequences.

2.8 Nonvolatile Memory

Monza™'s embedded memory is based on Impinj's multiple-times-programmable (MTP), nonvolatile memory (NVM) cell technology, specifically optimized for exceptionally high performance in RFID applications. All programming overhead circuitry is integrated on-chip. Monza™ NVM provides industry-standard 10-year data retention (see section 5.3).

The NVM block is organized into two segments: (1) EPC Memory (up to 96 bits), and (2) Reserved Memory (which contains the Kill and Access passwords). TID Memory is ROM-based, and contains Impinj's manufacturer ID (000000000001) and the Monza™ model number. Monza™ does not implement the optional User Memory (see section 4.5).

3 Interface Characteristics

Described are the RF interface characteristics of both reader (Forward Link) and tag (Reverse Link).

3.1 Reader-to-Tag (Forward Link) Signal Characteristics

Table 3-1 Forward Link Signal Parameters

Parameter	Minimum	Typical	Maximum	Units	Comments
RF Characteristics					
Carrier Frequency	860		960	MHz	North America: 902–928 MHz Europe: 865–868 MHz
Read Sensitivity Limit		–9		dBm	Input sensitivity is measured on a single RF input. Input sensitivity is specified for a R=>T link using DSB-ASK modulation with 90% modulation depth, Tari=6.25µs, PW=2.1µs, and a T=>R link operating at 160kbps with FM0 encoding.
Write Sensitivity Limit		–7			
Maximum RF Field Strength			+20[1]	dBm	
Fading Rate			0.5 dB (write) 1 dB (other)	dB	1.5 m/sec tag velocity over 14 ms write time 3.6 m/sec tag velocity over 10 ms select cmd
Modulation Characteristics					
Modulation		DSB-ASK, SSB-ASK, or PR-ASK			Double and single sideband amplitude shift keying; phase-reversal amplitude shift keying
Data Encoding		PIE			Pulse-interval encoding
Modulation Depth (A-B)/A	80		100	%	
Ripple, Peak-to-Peak M_h=M_l			5	%	Portion of A-B
Rise Time ($t_{r,10\text{-}90\%}$)	0		0.33Tari	sec	
Fall Time ($t_{f,10\text{-}90\%}$)	0		0.33Tari	sec	
Tari[2]	6.25		25	µs	Data 0 symbol period
PIE Symbol Ratio	1.5:1		2:1		Data 1 symbol duration relative to Data 0
Duty Cycle	48		82.3	%	Ratio of data symbol high time to total symbol time
Pulse Width	MAX(0.265Tari,2)		0.525Tari	µs	Pulse width defined as the low modulation time (50% amplitude)

Note 1. Reader antenna power with tag sitting on antenna. Assumes tag has half-wave dipole antenna. While maximum radiated reader power is +36 dBm for both Read and Write operations, the maximum power the tag should receive is +20 dBm (see section 5.2).

Note 2. Values are nominal; they do not include reader clock frequency error.

3.1.1 Forward Link Waveform

The forward link RF signal is defined by Figure 3-1 through Figure 3-3. Tari is defined in Figure 3-1. The pulse width (*PW*) is the negative modulation duration within a symbol, and is measured as the time between 50% amplitude points on the falling and rising edges of the pulse. The electric field strength *A* is the maximum amplitude

of the RF envelope. Figure 3-3 illustrates the relationship between baseband signaling and the resultant RF envelope for ASK and PR-ASK modulations.

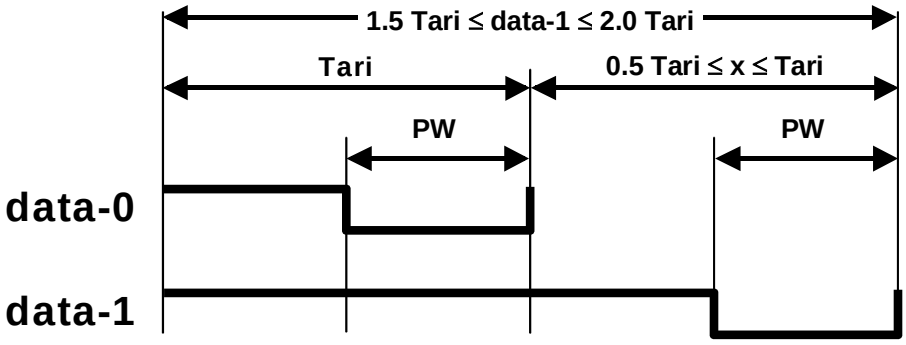


Figure 3-1 Modulation Characteristics

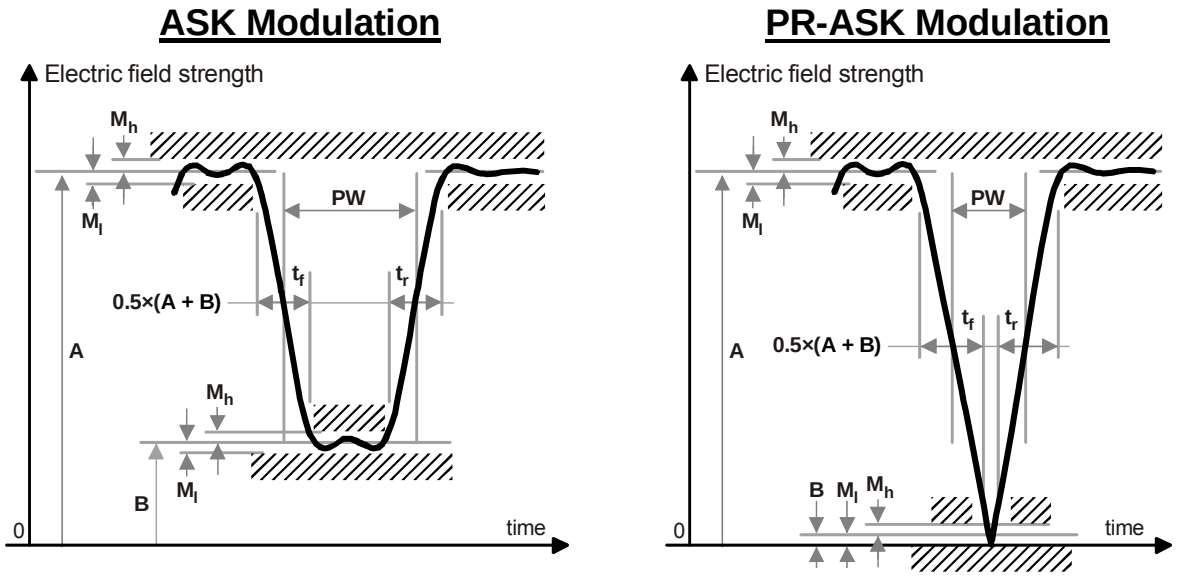


Figure 3-2 RF Envelope Characteristics

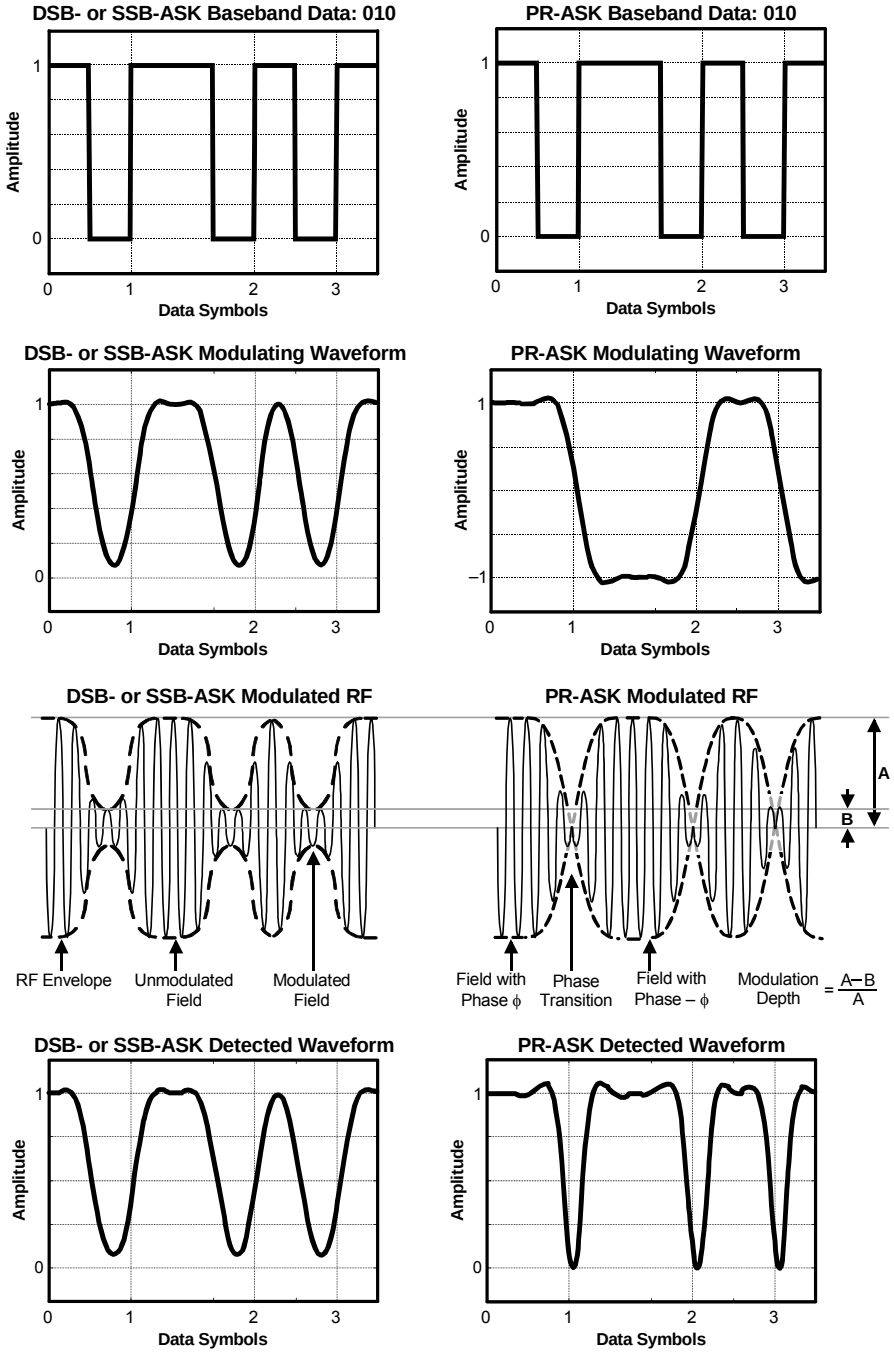


Figure 3-3 Relationship Between Baseband Signaling and RF Envelope

3.1.2 RF Power-up and Power-down Envelope

The allowed power-up and power-down RF envelope is shown in Figure 3-4 and Table 3-2. Once the carrier level has risen above the 10% level, the power-up envelope must rise monotonically until at least the 90% level. The RF envelope must not fall below the 90% point in Figure 3-4 during interval T_s. The carrier envelope must meet the pre-settled and post-settled ripple limits specified in Table 3-2.

In power-down, once the carrier level has fallen below the 90% level, the envelope must continue to fall monotonically until the power-off limit M_s. Once powered off, the carrier must remain off for a least 1 ms before powering up again.

3.1.3 Tag Power-On Initialization

Monza™-based tags conform to the power-up requirements per the Gen 2 specification.

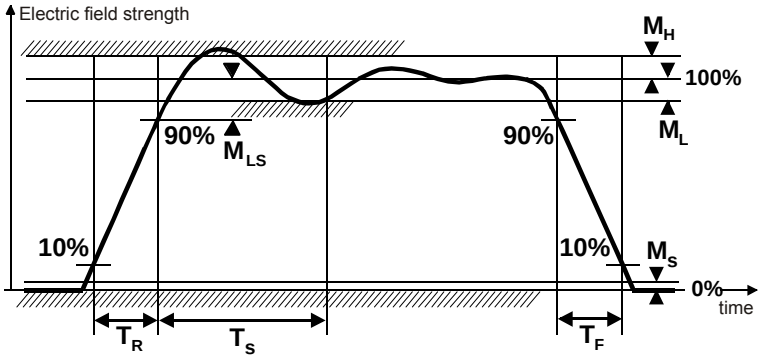


Figure 3-4 Power-up and Power-Down RF Envelope

Table 3-2 Power-up and Power-down Parameters

Parameter	Definition	Minimum	Typical	Maximum	Units
T_F	Fall time	1		500	µs
T_R	Rise time	1		500	µs
M_S	Signal level when OFF			1	% full scale
T_S	Settling time			1.5	ms
M_{LS}	Undershoot			10	% full scale
M_L	Post-Settled Undershoot			5	% full scale
M_H	Post-Settled Overshoot			5	% full scale

3.1.4 Preamble and Frame-Sync

A Gen 2 reader must precede commands with either a preamble or a frame-sync modulation pattern, both of which are shown in Figure 3-5. A preamble precedes a *Query* command, as it denotes the start of an inventory round. All other signaling begins with a frame-sync. Nominal signal durations within the preamble and frame-sync must be consistent with one another and persist throughout an inventory round.

A preamble comprises a fixed-length start delimiter, a Data 0 symbol, an R=>T calibration (RTcal) symbol, and a T=>R calibration (TRcal) symbol.

- **RTcal:** The reader sets RTcal equal to the length of a Data 0 symbol plus the length of a Data 1 symbol ($RTcal = 0_{length} + 1_{length}$). The tag measures the length of RTcal and computes *pivot* = RTcal / 2. The tag then interprets subsequent symbols shorter than *pivot* to represent Data 0, and subsequent symbols longer than *pivot* to represent Data 1. The tag interprets symbols longer than four times the current RTcal to be invalid data.
- **TRcal:** The reader specifies the tag's backscatter link frequency (its FM0 data rate or the frequency of its Miller subcarrier) using the TRcal and divide ratio (*DR*) in the preamble and payload, respectively, of a *Query* command that initiates an inventory round. The equation below defines the relationship between the backscatter link frequency (*LF*), TRcal, and *DR*. The tag measures the length of TRcal, computes *LF*, and adjusts its reverse link rate to be equal to *LF*.

$$LF = \frac{DR}{TRcal}$$

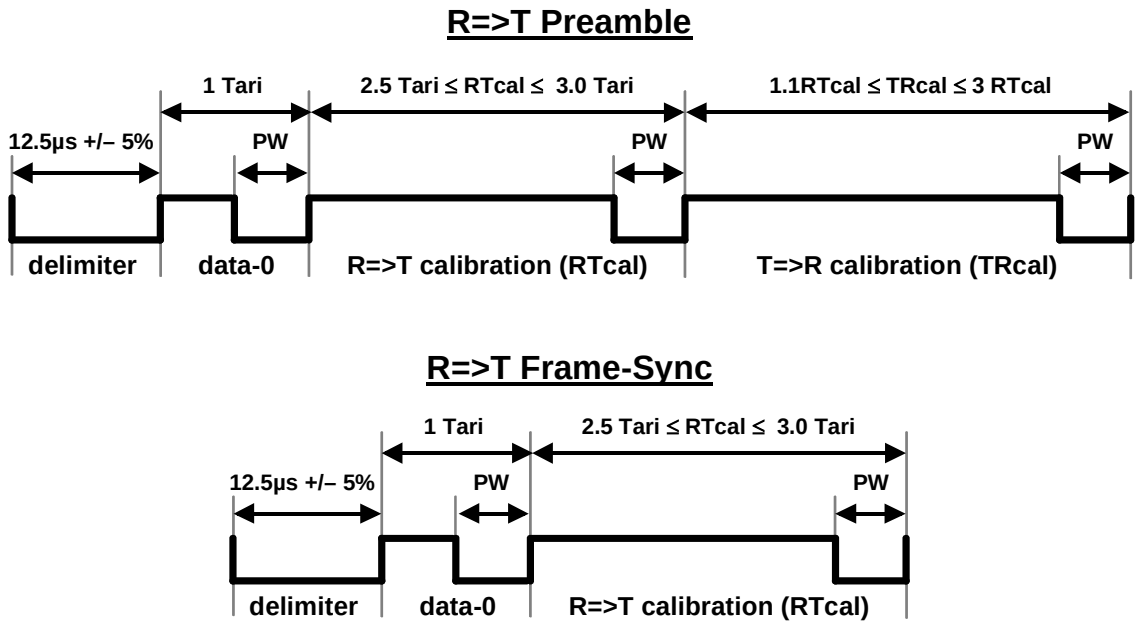


Figure 3-5 Preamble and Frame-Sync Modulation Patterns

3.2 Tag-to-Reader (Reverse Link) Backscatter

The tag transmits information on the tag-to-reader link by reflecting, or backscattering, part of the incident RF energy from the reader. Backscatter modulation is performed by modulating the input impedance of the tag, thereby modulating the reflection coefficient (denoted by Γ, or gamma) at the antenna-to-tag interface. The symbol |△Γ| (delta gamma) represents the magnitude of the change in reflection coefficient from the non-modulating (absorptive) to the modulating (reflective) state.

3.2.1 Reflection Coefficient

Figure 3-6 shows the magnitude of the complex reflection coefficient (|Γ|) for both modulating and non-modulating states, as a function of the power level at the tag (note that these data are simulated design parameters). For the condition where the modulator is off (tag not backscattering), note that the reflection coefficient reaches a minimum at a power level near the tag's lower limit of operation. At a reflection coefficient of 0, the tag absorbs power under the optimal power transfer condition (see section 3.3). As the incident power level increases, the magnitude of the reflection coefficient is intentionally increased, thereby reflecting excess incident power as CW to prevent damage to the tag's analog front end.

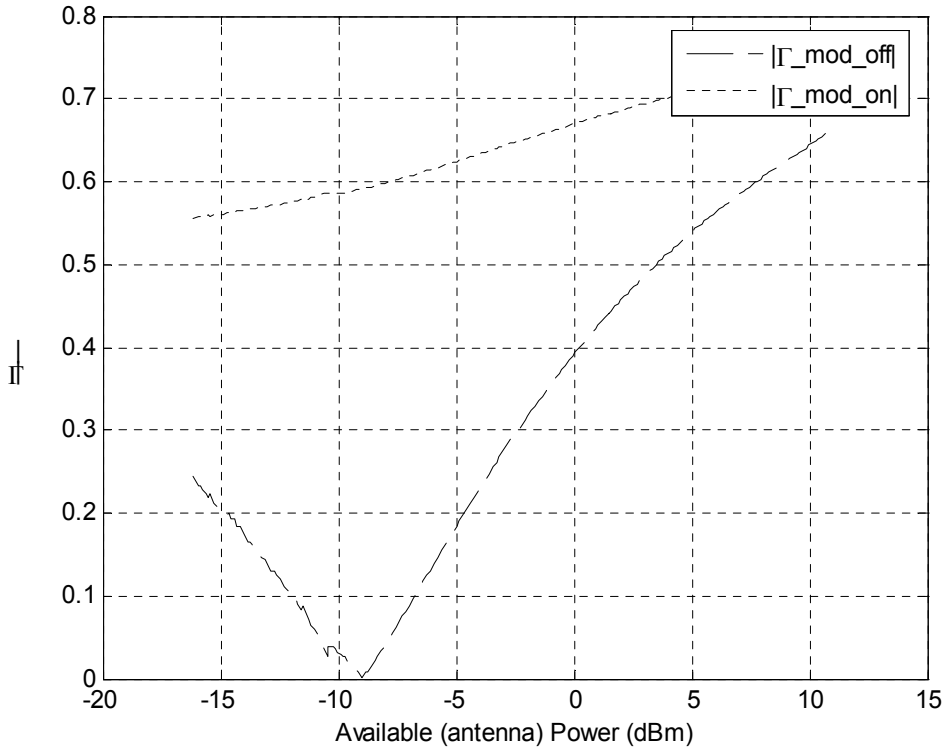


Figure 3-6 Simulated Tag Reflection Coefficient versus Incident RF Power in Modulator-Off and Modulator-On States

3.2.2 Modulation as Related to |△Γ|

Figure 3-7 illustrates the measured magnitude of the difference between the two states of the reflection coefficient:

$$|\Delta\Gamma| = |\Gamma_{mod\_on} - \Gamma_{mod\_off}|$$

When a tag is transmitting information to a reader, the tag modulator switches its reflection coefficient between the two states, producing modulated (information-bearing) sidebands in the reflected signal. The amount of energy in the modulated sidebands is directly proportional to $|\Delta\Gamma|^2$. As such, $|\Delta\Gamma|$ plays a key role in any RF link budget. It should be noted that resistance and other nonlinear parasitic effects in the modulator impose practical limits on the range of $|\Delta\Gamma|$.

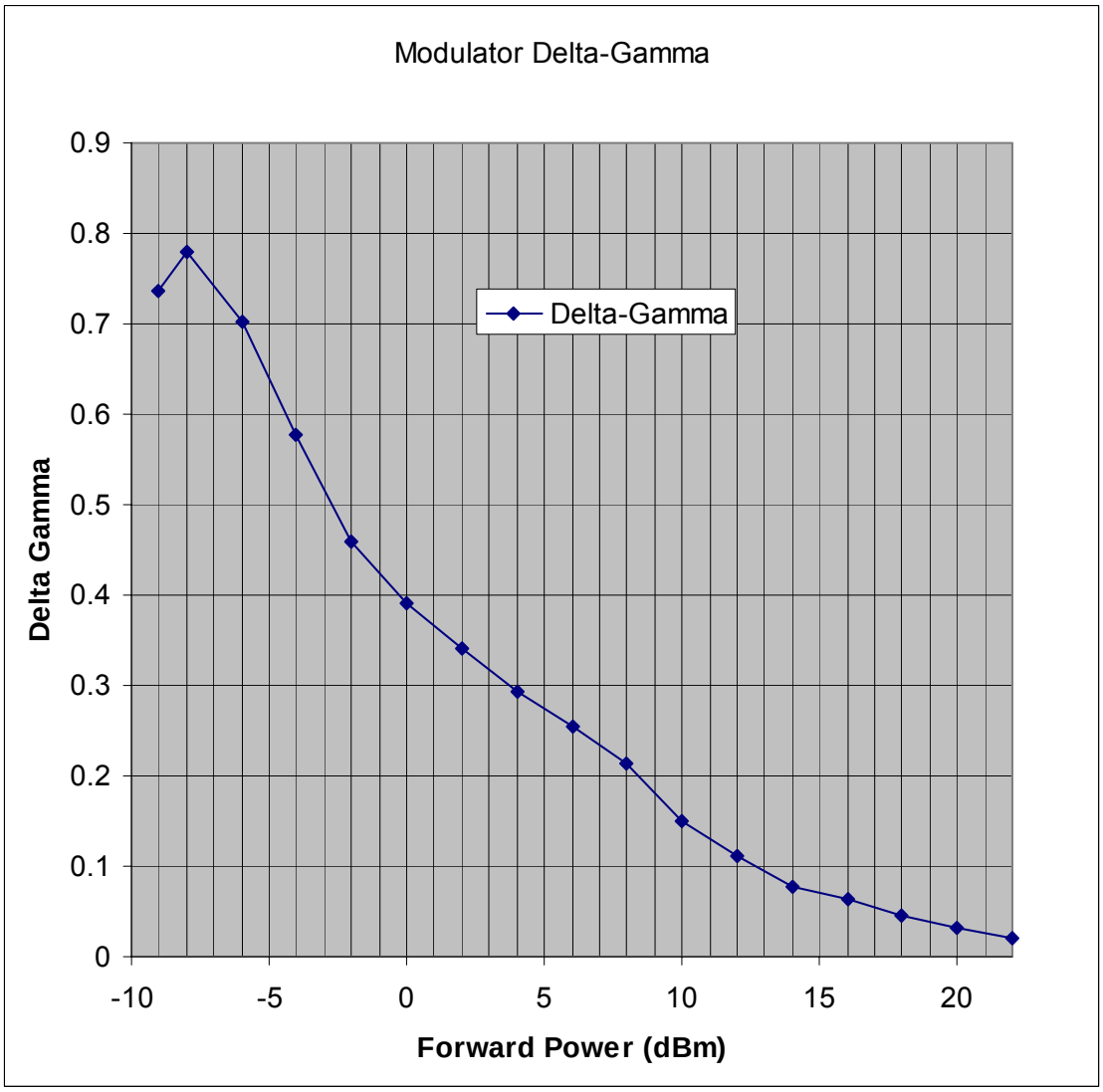


Figure 3-7 Measured Tag Delta-Gamma as a Function of Available Input Power

3.2.3 Radar Cross Section (RCS)

Tag RCS is the measure of the portion of the incident RF energy reflected isotropically back to a reader (a higher $|\triangle\Gamma|$ results in a larger RCS. But $|\triangle\Gamma|$ is not fixed; it changes with power). Figure 3-8 illustrates RCS as a function of $|\triangle\Gamma|$. RCS is given by:

$$\sigma_{bs} = \frac{P_{received}}{P_{incident}} \cdot 4\pi R^2$$

where σ_{bs} is the radar cross section; $P_{incident}$ is the power incident on the tag, and $P_{received}$ is the power at the antenna observing the tag's backscattered signal. Radar-cross section and $|\triangle\Gamma|$are related by:

$$\sigma_{bs} = \frac{\lambda^2}{4\pi} \cdot G_{Tag}^2 \cdot \left|\Delta\Gamma^2\right|$$

where G_{Tag} is the gain of the tag antenna.

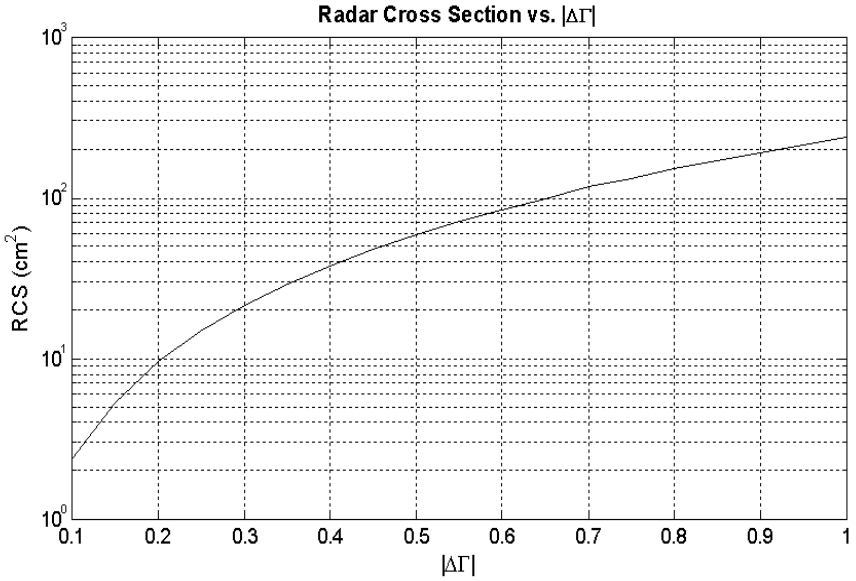


Figure 3-8 Tag Radar Cross Section (RCS) as a Function of |ΔΓ| (assumes half-wave dipole tag antenna and carrier frequency of 915 MHz)

If a tag is in close proximity to a reader, it will reflect a different amount of power than if the tag is at the limit of range. If the power level transmitted by the reader is known, and if the path loss to the tag is known, then one can simply index onto the X axis of Figure 3-7 and determine $|\triangle\Gamma|$. At lower power levels (long range), a large $|\triangle\Gamma|$is desired, as this increases the tag's RCS, and hence its read/write range. But at higher power levels, a lower level of reflection is preferred for the reasons described in section 3.2.1. As can be seen in Figure 3-7, as input power increases, $|\triangle\Gamma|$ decreases.

3.3 Source Impedance for Maximum Power Transfer

Table 3-3 shows the source impedance for an antenna designed to deliver maximum power to a Monza™ tag at minimum input sensitivity (shown are the three center frequencies that correspond to the primary regional frequencies of operation). Figure 3-9 shows the same data graphically, with –0.5 dB and –1 dB sensitivity loss contours. While the Smith chart plots only the case of F_c = 915 MHz, the results of the other frequencies fall within the resolution of the optimal point shown (indicated by the diamond inside the contour boundary; for the other frequencies considered, there is negligible or no change in the size, shape, or radius of the contour, although there is a slight phase shift). Note that due to the nonlinear nature of the tag circuits, this antenna source impedance is *not* the complex conjugate of the tag input impedance. Furthermore, the sensitivity loss due to mismatch is asymmetric (as evidenced by the steep gradients of the contours shown in Figure 3-9), also as a result of nonlinearities in the tag input impedance. The source impedance values that maximize efficiency and read range, as well as those for efficiency loss due to source impedance variation, were determined empirically. The impedance values resulting in maximum efficiency and power transfer are shown in Table 3-3, while the sensitivity to impedance variation is represented in the contours of Figure 3-9. Impinj does not recommend antenna design to the point of maximum power transfer, as this point lies very close to the contour edge; a slight deviation from this point could result in significant loss of performance. A suggested antenna impedance design target, therefore, is in the *center* of the inner contour (which, per Figure 3-9, is 40 +j95 Ω). The resulting mismatch loss from the point of maximum power transfer will be negligible; more importantly, it will result in more robust tag performance overall.

Table 3-3 Antenna Source Impedance for Maximum Power Transfer

Parameter	Europe F_c = 866.5 MHz	North America F_c = 915 MHz	Japan F_c = 953 MHz	Units
Input Power Level @ Minimum Sensitivity				
Impedance	36 +j117	33 +j112	30 +j108	Ω

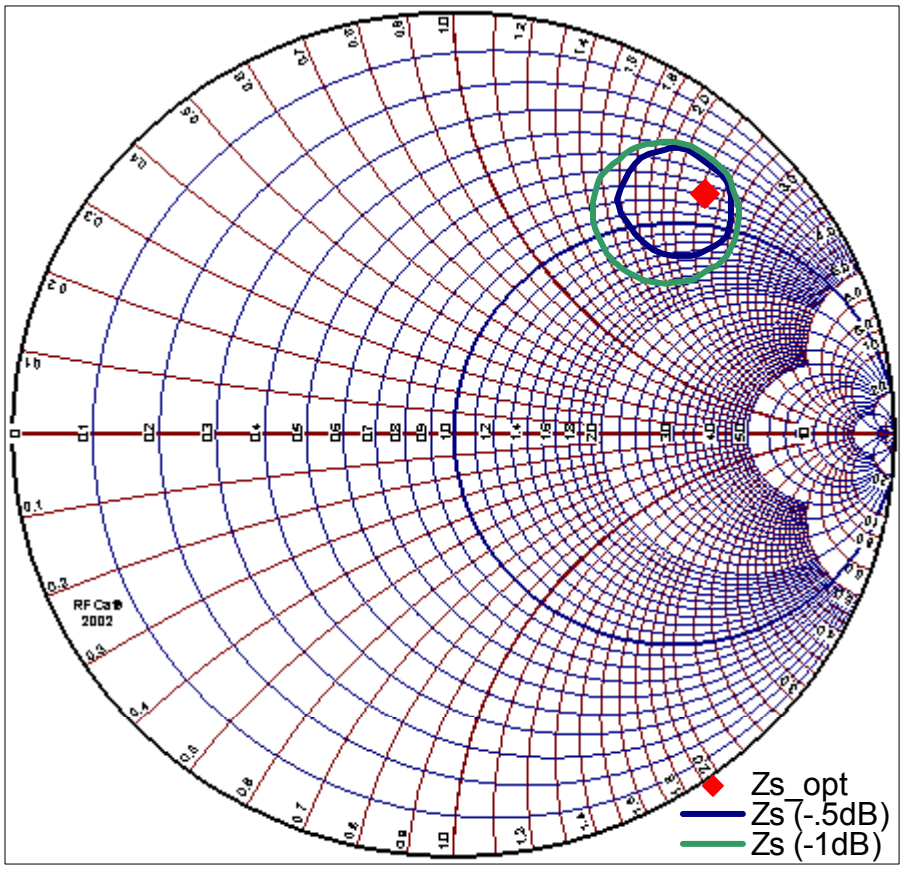


Figure 3-9 Antenna Source Impedance for Maximum Power Transfer at 915 MHz, Showing –0.5 dB and –1.0 dB Sensitivity Loss Contours

3.3.1 Monza™ Frequency Sensitivity

For reference purposes, Figure 3-10 shows measured Monza™ S_{11} versus frequency for a small signal input and a frequency sweep from 800 MHz to 1000 MHz. The data show that Monza™ has minimal change in reflection coefficient over a broad frequency range, resulting in tags that operate worldwide with negligible sensitivity degradation.

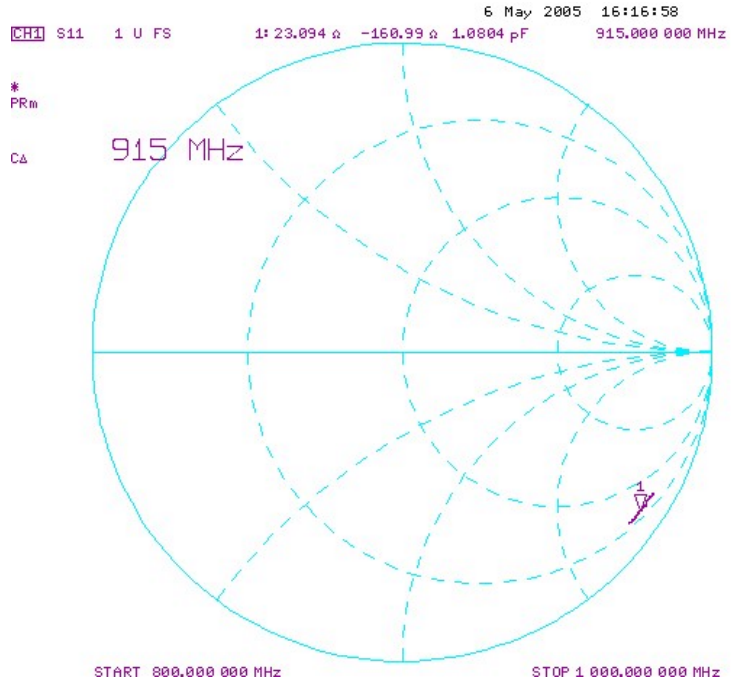


Figure 3-10 Monza™ S_{11} versus Frequency

3.4 Reverse Link Signal Characteristics

Table 3-4 Reverse Link Signal Parameters

Parameter	Minimum	Typical	Maximum	Units	Comments
Modulation Characteristics					
Modulation		ASK			FET Modulator
Data Encoding		Baseband FM0 or Miller Subcarrier			
Change in Modulator Reflection Coefficient $\|\Delta\Gamma\|$ due to Modulation		0.7			$\|\Delta\Gamma\| = \|\Gamma_{reflect} - \Gamma_{absorb}\|$ (per read/write sensitivity, Table 3-1)
ETSI Spectral Mask Compliance					See Section 4.6
Duty Cycle	45	50	55	%	See sections 3.4.1 through 3.4.4
Symbol Period[1]	1.5625		25	µs	Baseband FM0
	3.125		200	µs	Miller-modulated subcarrier
Miller Subcarrier Frequency[1]	40		640	kHz	

Note 1. Values are nominal minimum and nominal maximum, and do not include frequency tolerance. Apply appropriate frequency tolerance to arrive at absolute durations and frequencies (refer to Table 3-5).

3.4.1 Reverse Link Waveform

A tag communicates with a reader by using backscatter modulation, in which the tag switches the reflection coefficient of its antenna between the two states, absorptive and reflective. The tag backscatters using a fixed modulation format, data encoding, and data rate for the duration of an inventory round. The modulation format is ASK; the reader selects the tag's encoding and data rate by means of the *Query* command that initiates the round.

The tag backscatters using ASK modulation, produced by changing the magnitude component of the input reflection coefficient. The low levels in Figure 3-11 through Figure 3-19 correspond to the absorptive tag condition (tag absorbing power, not backscattering), whereas the high levels correspond to the reflective tag condition (tag reflecting power, or backscattering).

The tag encodes the backscattered data as either baseband FM0 or Miller modulation of a subcarrier at the data rate, the choice of which is determined by reader command (via the *M* parameter of the *Query* command. See section 3.4.7).

3.4.2 Baseband FM0 Data Modulation

Figure 3-11 shows a state diagram for generating FM0 (bi-phase space) encoding. FM0 inverts the tag reflectivity condition at every symbol boundary; a Data 0 has an additional mid-symbol phase state change.

The state diagram shown in Figure 3-11 maps a logical data sequence to the FM0 symbols that are transmitted. The state labels, S_1–S_4, correspond to the four possible FM0-encoded symbols shown in Figure 3-12. When a state is entered, the associated FM0 waveform is transmitted. The transition labels indicate the logical values of the next data bit to be encoded. For example, if a Data 1 is to follow the S_1 FM0 symbol, then a S_4 basis waveform is

appended to the sequence. Note that certain symbol transitions are not allowed, such as from state S_2 to S_3 (because the resulting transmission would not have a phase inversion on a symbol boundary).

Figure 3-12 shows generated baseband FM0 symbols and 2-bit sequences. The duty cycle of a 00 or 11 sequence, measured at the modulator output, is as specified in Table 3-4. FM0 encoding has memory; consequently, FM0 sequences depend on prior transmissions. FM0 signaling always starts with a preamble as shown in Figure 3-5. At the end of the preamble the tag will be in state S_1. If the first data bit following the preamble is Data 0, the tag will transition to the S_3 state and backscatter the S_3 basis waveform. If the first data bit following the preamble is Data 1, the tag will transition to the S_4 state and backscatter the S_4 basis waveform. FM0 signaling always ends with a "dummy" Data 1 bit at the end of a transmission, as shown in Figure 3-13. If the dummy Data 1 bit is generated in S_1, then a final negative transition to the absorptive (non-modulating tag) condition will be appended.

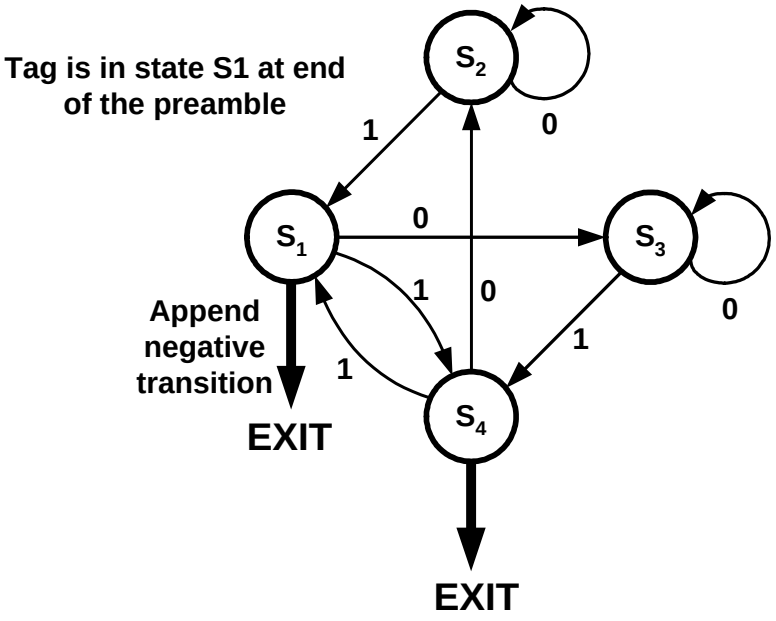


Figure 3-11 FM0 Generator State Diagram

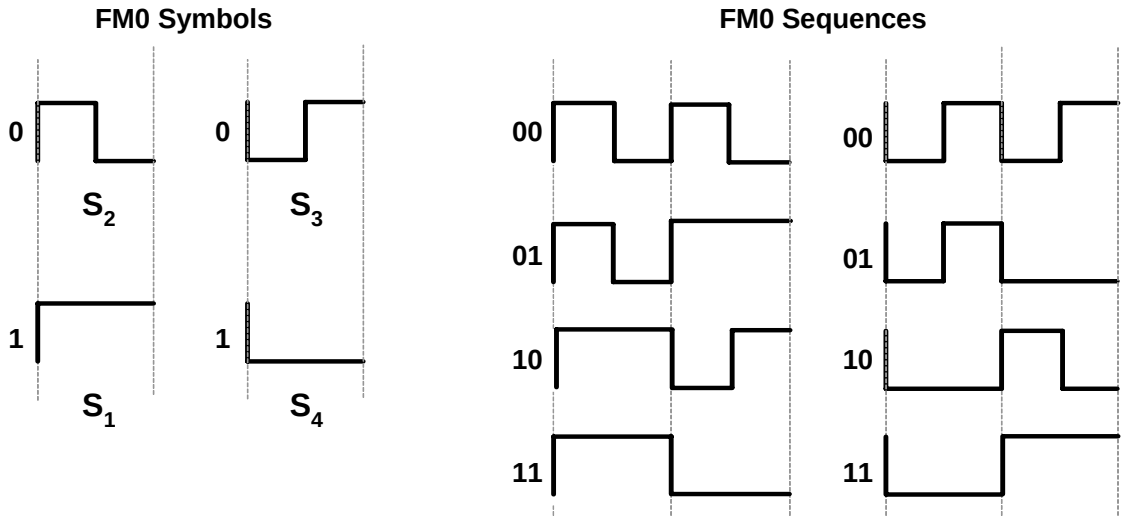


Figure 3-12 FM0 Symbols and Sequences

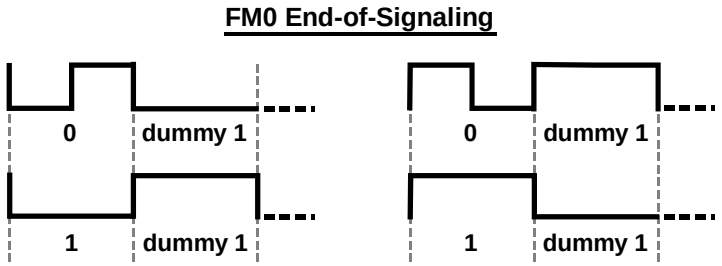


Figure 3-13 Terminating FM0 transmissions

3.4.3 Baseband FM0 Preambles

FM0 signaling begins with one of the two preambles shown in Figure 3-14. The choice depends on the value of the TRext bit specified in the *Query* command that initiated the inventory round. The *V* shown in the bottom waveform of Figure 3-14 indicates an FM0 violation (i.e., a phase inversion should have occurred but did not).

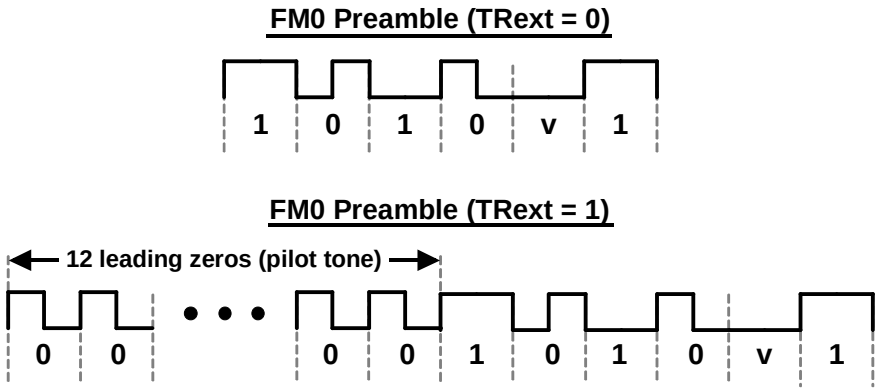


Figure 3-14 FM0 Preambles

3.4.4 Miller-Modulated Subcarrier

Figure 3-15 shows a state diagram for generating Miller encoding. Baseband Miller inverts its phase between two Data 0s in sequence. Baseband Miller also places a phase inversion in the middle of a Data 1 symbol. The state diagram in Figure 3-15 indicates the mapping of a logical data sequence to baseband Miller functions. The state labels, S_1–S_4, correspond to the four possible Miller-encoded symbols shown in Figure 3-16. When a state is entered, the associated Miller baseband waveform is generated. The transition labels indicate the logical values of the next data bit to be encoded. For example, if a Data 1 is to follow the S_1 Miller symbol, then a S_2 symbol is appended to the sequence.

The transmitted subcarrier waveform is derived from the symbol sequence by multiplying by a square-wave, starting at the negative polarity, at *M* times the baseband symbol rate (the *M* value being specified in the *Query* command).

Figure 3-17 shows Miller-modulated subcarrier sequences; the Miller sequence will contain exactly two, four, or eight subcarrier cycles per baseband symbol, depending on the *M* value specified in the *Query* command that initiated the inventory round. Miller subcarrier signaling always starts with one of the preambles shown in Figure 3-19. The tag will be in the S_3 state at the end of the preamble. The duty cycle (percent of high time) of a 0 or 1 modulated symbol is as specified in Table 3-4. Miller signaling always ends with a "dummy" Data 1 bit at the end of a transmission, as shown in Figure 3-18.

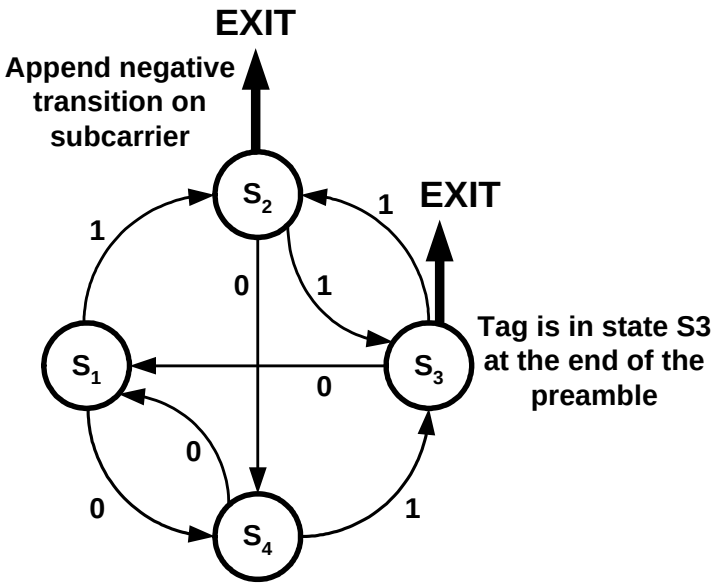


Figure 3-15 Miller Baseband Generator State Diagram

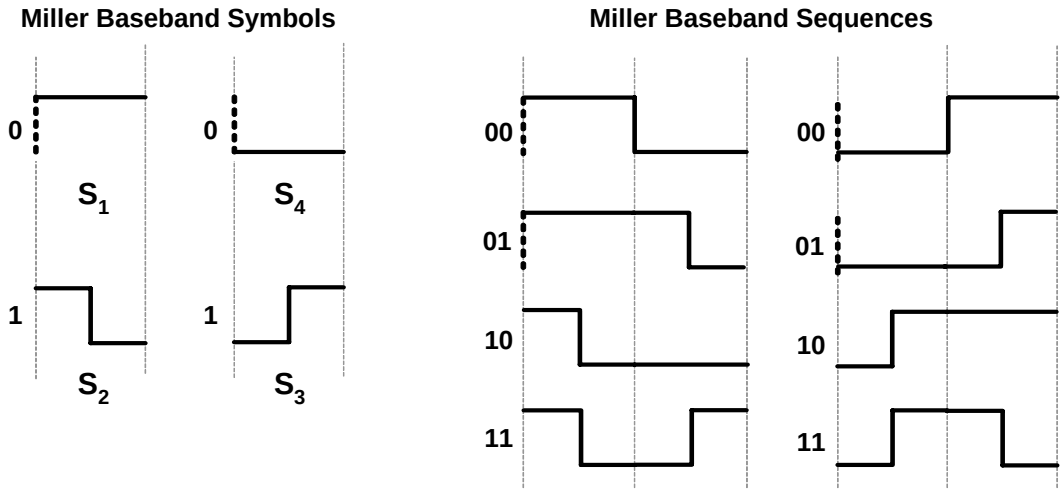


Figure 3-16 Miller Baseband Symbols and Sequences

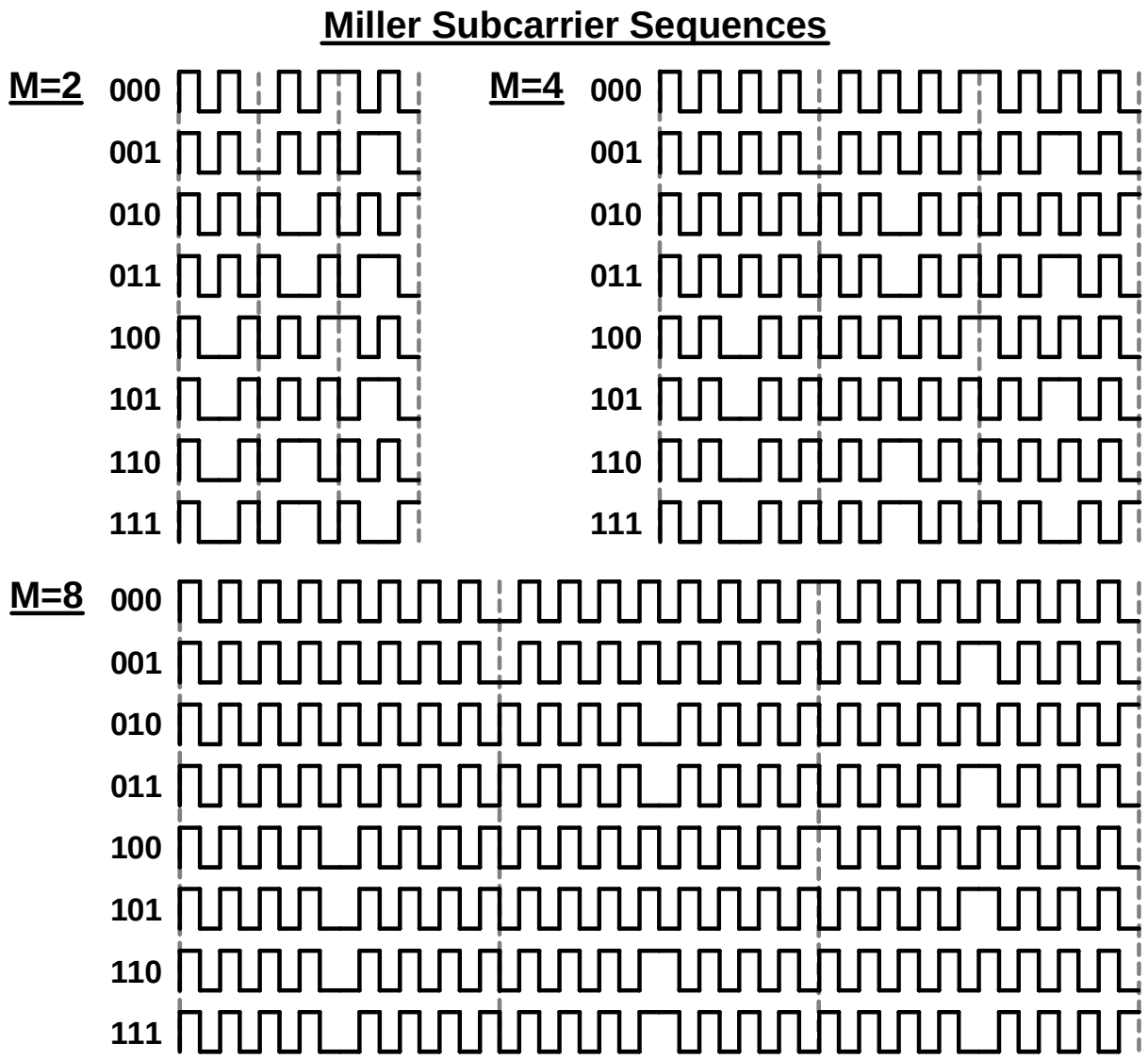


Figure 3-17 Miller-Modulated Subcarrier Frequencies

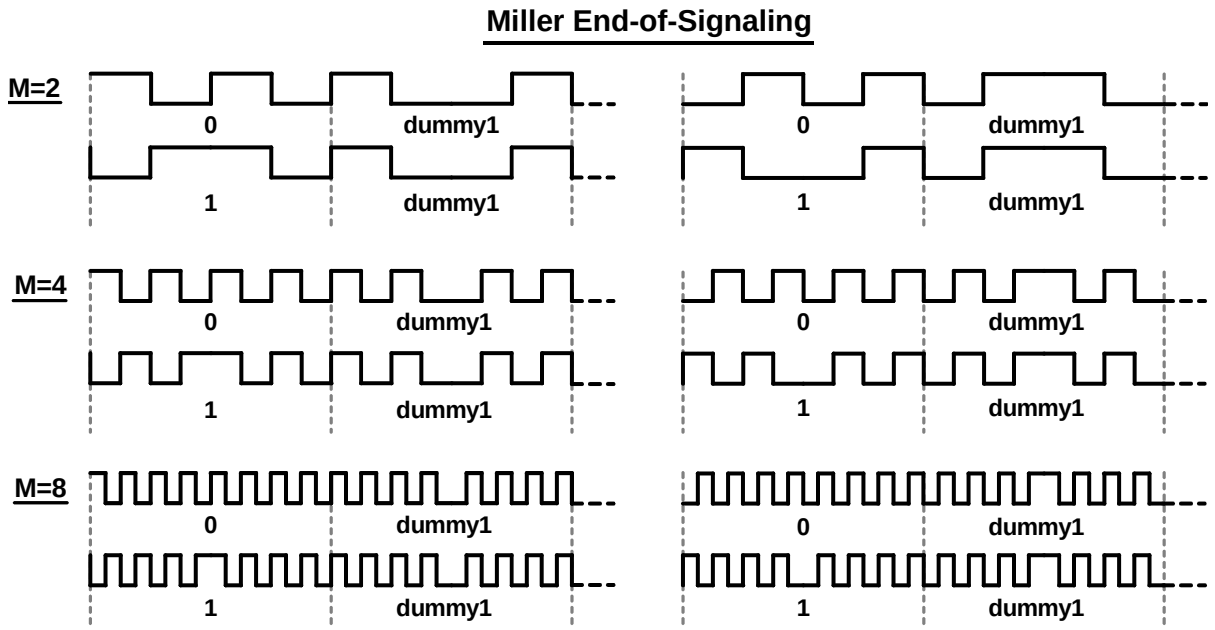


Figure 3-18: Terminating Subcarrier Transmissions

3.4.5 Subcarrier Preambles

Miller-modulated subcarrier signaling begins with one of the two preambles shown in Figure 3-19. The choice depends on the value of the TRext bit specified in the *Query* command that initiated the inventory round.

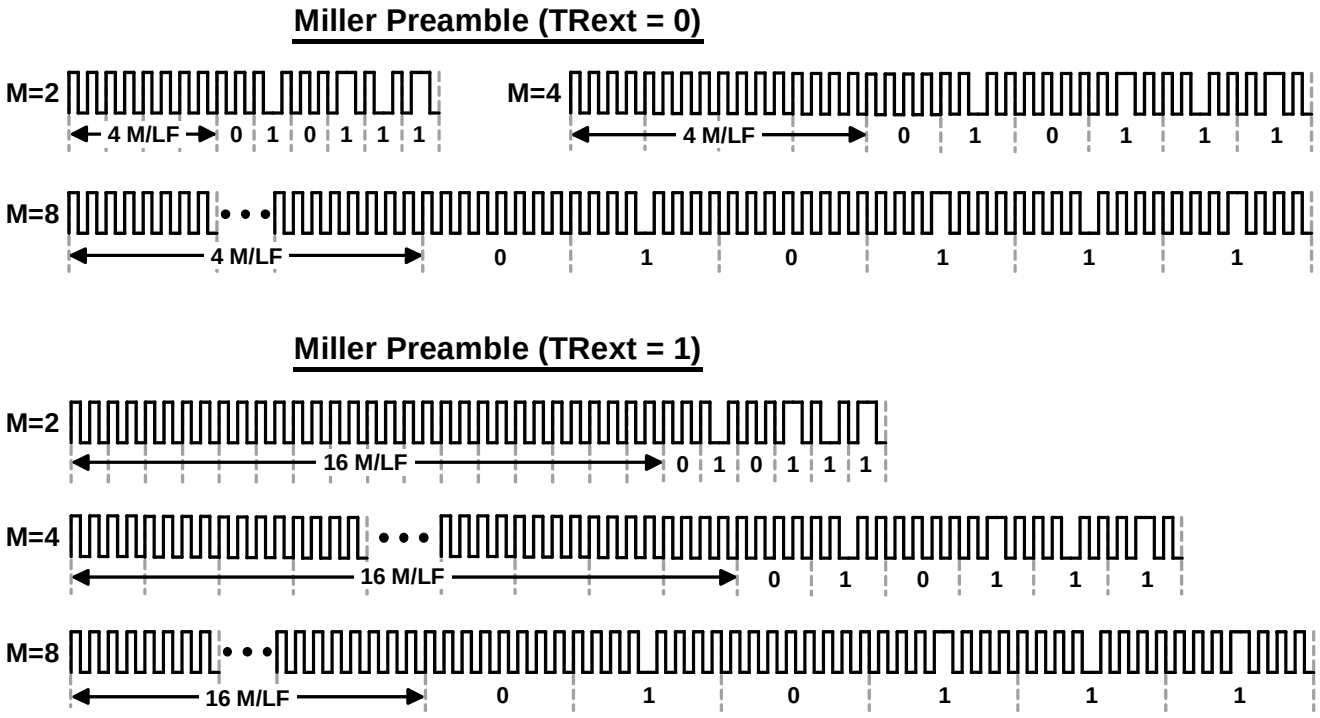


Figure 3-19: Miller-modulated subcarrier

3.4.6 Link Frequency

Tags support the reverse link symbol periods specified in Table 3-4 with the frequency tolerances specified in Table 3-5. The link frequency (*LF*) is calculated in the tag by dividing the Divide Ratio (*DR*) by the measured TRcal. *DR* is a parameter in the *Query* command that initiates an inventory round; TRcal is transmitted by the reader as part of the preamble that precedes the *Query*.

3.4.7 Data Rate

The link frequency (*LF*) and *M*, also a parameter of the *Query* command, together determine the reverse link data rate, as shown in Table 3-6. The tag will use *M* to determine the number of ± 1 square-wave cycles to apply over a baseband symbol when in Miller-modulated subcarrier mode. The larger the *M* value, the lower the data rate for a given *LF*. If the *M* value is 1, then baseband FM0 will be backscattered with no square-wave applied. The data rate is fixed for the duration of an inventory round.

Table 3-5 Reverse-Link Tolerances

DR: Divide Ratio	TRcal (µs +/– 1%)	LF: Link Frequency (kHz)	Frequency Tolerance FT (nominal temp)[1]	Frequency Tolerance FT (extended temp)[1]	Frequency variation during backscatter
64/3	33.3	640	+ / – 15%	+ / – 15%	+ / – 2.5%
	33.3 < TRcal < 66.7	320 < LF < 640	+ / – 22%	+ / – 22%	+ / – 2.5%
	66.7	320	+ / – 10%	+ / – 15%	+ / – 2.5%
	66.7 < TRcal < 83.3	256 < LF < 320	+ / – 12%	+ / – 15%	+ / – 2.5%
	83.3	256	+ / – 10%	+ / – 10%	+ / – 2.5%
	83.3 < TRcal ≤ 133.3	160 ≤ LF < 256	+ / – 10%	+ / – 12%	+ / – 2.5%
	133.3 < TRcal ≤ 200	107 ≤ LF < 160	+ / – 7%	+ / – 7%	+ / – 2.5%
	200 < TRcal ≤ 225	95 ≤ LF < 107	+ / – 5%	+ / – 5%	+ / – 2.5%
8	17.2 ≤ TRcal < 25	320 < LF ≤ 465	+ / – 19%	+ / – 19%	+ / – 2.5%
	25	320	+ / – 10%	+ / – 15%	+ / – 2.5%
	25 < TRcal < 31.25	256 < LF < 320	+ / – 12%	+ / – 15%	+ / – 2.5%
	31.25	256	+ / – 10%	+ / – 10%	+ / – 2.5%
	31.25 < TRcal < 50	160 < LF < 256	+ / – 10%	+ / – 10%	+ / – 2.5%
	50	160	+ / – 7%	+ / – 7%	+ / – 2.5%
	50 < TRcal ≤ 75	107 ≤ LF < 160	+ / – 7%	+ / – 7%	+ / – 2.5%
	75 < TRcal ≤ 200	40 ≤ LF < 107	+ / – 4%	+ / – 4%	+ / – 2.5%

Note 1. Per Gen 2 Specification, nominal temp range is –25 ºC to +40 ºC; extended temp range is –40 ºC to +65 ºC

Table 3-6 Reverse Link Data Rates

M: Number of Subcarrier Cycles per Symbol	Modulation Type	Data Rate (kbps)
1	FM0 baseband	LF
2	Miller subcarrier	LF/2
4	Miller subcarrier	LF/4
8	Miller subcarrier	LF/8

3.4.8 Read Sensitivity

A Monza™ tag will perform a Read when at least one of the antenna ports is receiving power equal to the sensitivity shown in Table 3-1.

3.4.9 Write Sensitivity

A Monza™ tag will perform a Write when at least one of the antenna ports is receiving power equal to the sensitivity shown in Table 3-1.

3.5 Interface Protocols

3.5.1 Link Timing

Figure 3-20 and Table 3-7 define link timing between a reader and tag. The maximum value for T_2 will apply only to tags in the **reply** or **acknowledged** states (see Figure 4-1). For a tag in the **reply** or **acknowledged** states, if T_2 expires (i.e., reaches its maximum value):

- Without the tag receiving a valid command, the tag will transition to the **arbitrate** state,
- During the reception of a valid command, the tag will execute the command,
- During the reception of an invalid command, the tag will transition to **arbitrate** upon determining that the command is invalid.

In all other states the maximum value for T_2 is unrestricted.

A reader may transmit a new command prior to interval T_2 (i.e., during a tag response). In this case the responding tag is not required to demodulate or otherwise act on the new command, and may undergo a power-on reset.
To change the data symbol parameters for a subsequent inventory round, a reader must transmit CW for the duration specified in Table 3-7. This CW period is defined as a link reset.

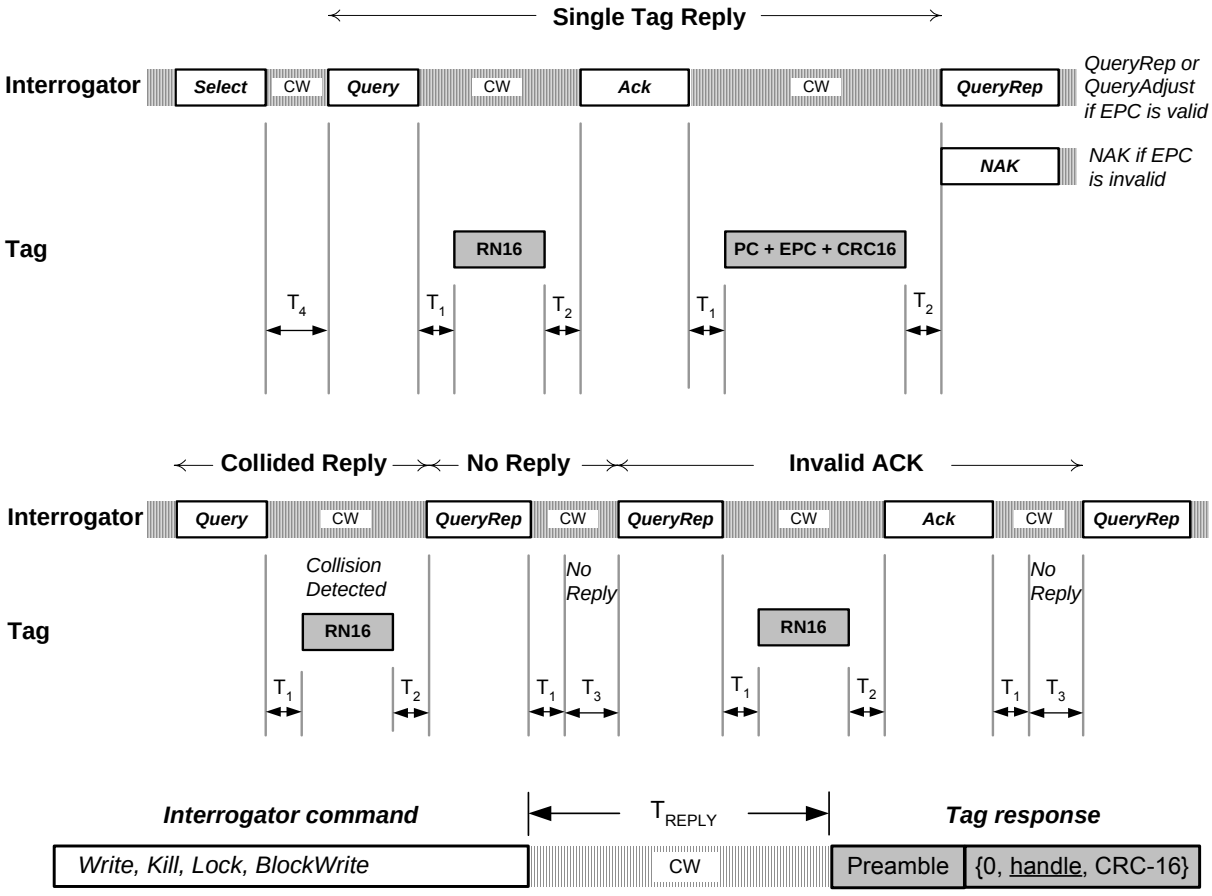


Figure 3-20: Link Timing

Table 3-7 Link Timing Parameters

Parameter	Minimum	Typical	Maximum	Description
Initialization			1.5 ms	Time from power on until the tag is ready to receive commands. Measured from end of T_R period in Figure 3-4.
T_1	MAX(RTcal,10*T_{pri}) × (1 – FT) – 2µs	MAX(RTcal,10*T_{pri})	MAX(RTcal,10*T_{pri}) × (1 + FT) + 2µs	Time from reader transmission to tag response (specifically, the time from the last rising edge of the last bit of the reader transmission to the first rising edge of the tag response), measured at the tag's antenna terminals.[1]
T_2	3*T_{pri}		20*T_{pri}	Time required if a tag is to demodulate the reader signal, measured from the last falling edge of the last bit of the tag response to the first falling edge of the reader transmission.
T_3	0*T_{pri}			Time an Interrogator waits, after T_1, before it issues another command.
T_{REPLY}			20 ms	20 ms is the maximum time a reader waits for a response to a *Write*, *Kill*, *Lock,* or *BlockWrite* command.
T_4	2*RTcal			Minimum time between reader commands.
Link Reset	8*RTcal			Minimum time reader transmits CW before changing RTcal.

Note 1: T_{PRI} is the commanded period of either a FM0 symbol or a single Miller-modulated subcarrier cycle, depending on the reverse-link modulation type, as commanded by the reader. For reference, T_{PRI} = 1/LF = TR_{CAL}/DR (see sections 3.1.4 and 3.4.6 for an explanation of the relationship between these parameters).

3.5.2 Data Transmission Order

The transmission order for all forward and reverse link communications must respect the following conventions:

- Within each message, the most-significant word is transmitted first, and
- Within each word, the most-significant bit (MSB) is transmitted first.

3.6 Command Interface

3.6.1 Standard Commands

Table 3-8 contains a summary of the commands implemented by Monza™. A more complete description of the command codes is contained in section 4.4.

Table 3-8 Command Table Summary

Command	Code[1]	Length (bits)	Protection
QueryRep	**0**0	4	Unique command length
ACK	**0**1	18	Unique command length
Query	**10**00	22	Unique command length and a CRC-5
QueryAdjust	**10**01	9	Unique command length
Select	**10**10	> 44	CRC-16
Reserved for future use	**10**11	–	–
NAK	**110**00000	8	Unique command length
Req_RN	**110**00001	40	CRC-16
Read	**110**00010	> 57	CRC-16
Write	**110**00011	> 58	CRC-16
Kill	**110**00100	59	CRC-16
Lock	**110**00101	60	CRC-16
Access	**110**00110	56	CRC-16
BlockWrite	**110**00111	> 57	CRC-16
Reserved for future use	**110**01001 … **110**11111	–	–
Reserved for custom commands	**1110**0000 00000000 … **1110**0000 11111111	–	Manufacturer specified
Reserved for proprietary commands	**1110**0001 00000000 … **1110**0001 11111111	–	Manufacturer specified
Reserved for future use	**1110**0010 00000000 … **1110**1111 11111111	–	–

Note 1. The bold MSBs indicate the code length (e.g., in the case of **110**, these are recognized by Monza™ as first three bits in an 8-bit code).

3.6.2 Error Codes

If a Monza™ tag encounters an error when executing an access command that reads from or writes to memory, and if the command is a handle-based command (i.e., *Read*, *Write*, *Kill*, *Lock*, or *BlockWrite*), then the tag backscatters an error code in the format shown in Table 3-9 instead of its normal reply.

- The tag uses the error-specific codes shown in Table 3-10.
- Tags backscatter error codes only from the **open** or **secured** states.
- A tag will not backscatter an error code if it receives an invalid access command; instead, it will ignore the command.
- If an error is described by more than one error code, the more specific error code will take precedence and will be the code that the tag backscatters.
- The one-bit header for an error code is a Logic 1, unlike the header for a normal tag response, which is a Logic 0.

Table 3-9 Tag-Error Reply Format

	Header	Error Code	RN	CRC-16
# of bits	1	8	16	16
description	1	Error code	handle	

Table 3-10 Monza™ Error Codes

Error Code (Binary)	Error-Code Name	Error Description
00000000	Other error	Kill command received but tag's kill password is zero; BlockWrite command received with WordCount >1; errors not covered by other codes
00000011	Memory overrun or unsupported PC value	The specified memory location does not exist or the PC value is not supported by the tag
00000100	Memory locked	The specified memory location is locked and/or permalocked and is not writeable
00001011	Insufficient power	The tag has insufficient power to perform the Write, Lock, or Kill operation(s)

4 State Machine Operation

Refer to Figure 4-1.

4.1 Tag States

4.1.1 Ready State

1. **Ready** can be viewed as a "holding state" for energized tags (receiving power) that are neither killed nor currently participating in an inventory round.
2. Upon entering an energizing RF field a tag that is not killed enters the **ready** state. The tag remains in **ready** until it receives a *Query* command whose inventoried parameter (for the session specified in the *Query*) and sel parameter match its current flag values. Matching tags draw a *Q*-bit number (see sections 4.4.2.1 and 4.4.2.2) from their RNG (random number generator), load this number into their slot counter, and transition to the **arbitrate** state if the number is nonzero, or to the **reply** state if the number is zero.
3. If a tag in any state except **killed** loses power it will return to **ready** upon regaining power.

4.1.2 Arbitrate State

1. **Arbitrate** can be viewed as a "holding state" for tags that are participating in the current inventory round, but whose slot counters hold nonzero values.
2. A tag in **arbitrate** will decrement its slot counter every time it receives a *QueryRep* command whose session parameter matches the session for the inventory round currently in progress, and will transition to the **reply** state when its slot counter reaches 0000_h.
3. Tags that return to **arbitrate** (for example, from the **reply** state) with a slot value of 0000_h decrement their slot counters from 0000_h to $7FFF_h$ at the next *QueryRep* (with matching session) and, because their slot value is now nonzero, remain in **arbitrate**.

4.1.3 Reply state

1. Upon entering the **reply** state the tag backscatters an RN16 (16-bit random number).
2. If the tag receives a valid acknowledgement (*ACK*) it transitions to the **acknowledged** state, backscattering its PC, EPC, and CRC-16.
3. If the tag fails to receive an *ACK*, or receives an invalid *ACK*, it returns to **arbitrate.**
4. If the tag receives an invalid command, the tag transitions to **arbitrate**.
5. If the T2 timer expires before receiving a valid command, the tag transitions to **arbitrate**.

4.1.4 Acknowledged State

A tag in **acknowledged** may transition to any state except **killed**, depending on the received command.

1. If the tag receives an invalid command, the tag transitions to **arbitrate**.
2. If the T2 timer expires before receiving a valid command, the tag transitions to **arbitrate**.

4.1.5 Open state

1. A tag in the **acknowledged** state whose Access password is nonzero transitions to **open** upon receiving a *Req_RN* command, backscattering a new RN16 (denoted handle) that the reader will use in subsequent commands and the tag will use in subsequent replies.
2. Tags in the **open** state can execute all access commands except *Lock*.
3. A tag in **open** may transition to any state except **acknowledged**, depending on the received command.

4. Tag and reader must meet all timing requirements specified in 3.5.1, except maximum T_2; in the **open** state the maximum delay between tag response and reader transmission is unrestricted.

4.1.6 Secured State

1. A tag in the **acknowledged** state whose Access password is zero will transition to **secured** upon receiving a *Req_RN* command, backscattering a new RN16 (denoted handle) that the reader will use in subsequent commands and the tag will use in subsequent replies.
2. A tag in the **open** state whose access password is nonzero will transition to **secured** upon receiving a valid *Access* command, maintaining the same handle that it previously backscattered when it transitioned from the **acknowledged** to the **open** state.
3. Tags in the **secured** state can execute all access commands.
4. A tag in **secured** may transition to any state except **open** or **acknowledged**, depending on the received command.
5. Tag and reader must meet all timing requirements specified in 3.5.1, except maximum T_2; in the **secured** state the maximum delay between tag response and reader transmission is unrestricted.

4.1.7 Killed State

1. A tag in either the **open** or **secured** states enters the **killed** state upon receiving a *Kill* command sequence (4.4.3.4) with a valid nonzero kill password and valid handle.
2. *Kill* permanently disables a tag.
3. Upon entering the **killed** state the tag notifies the reader that the kill operation was successful, and will not respond to a reader thereafter.
4. Killed tags remain in the **killed** state under all circumstances, and immediately enter the killed state upon subsequent power-ups.
5. A kill operation is not reversible.

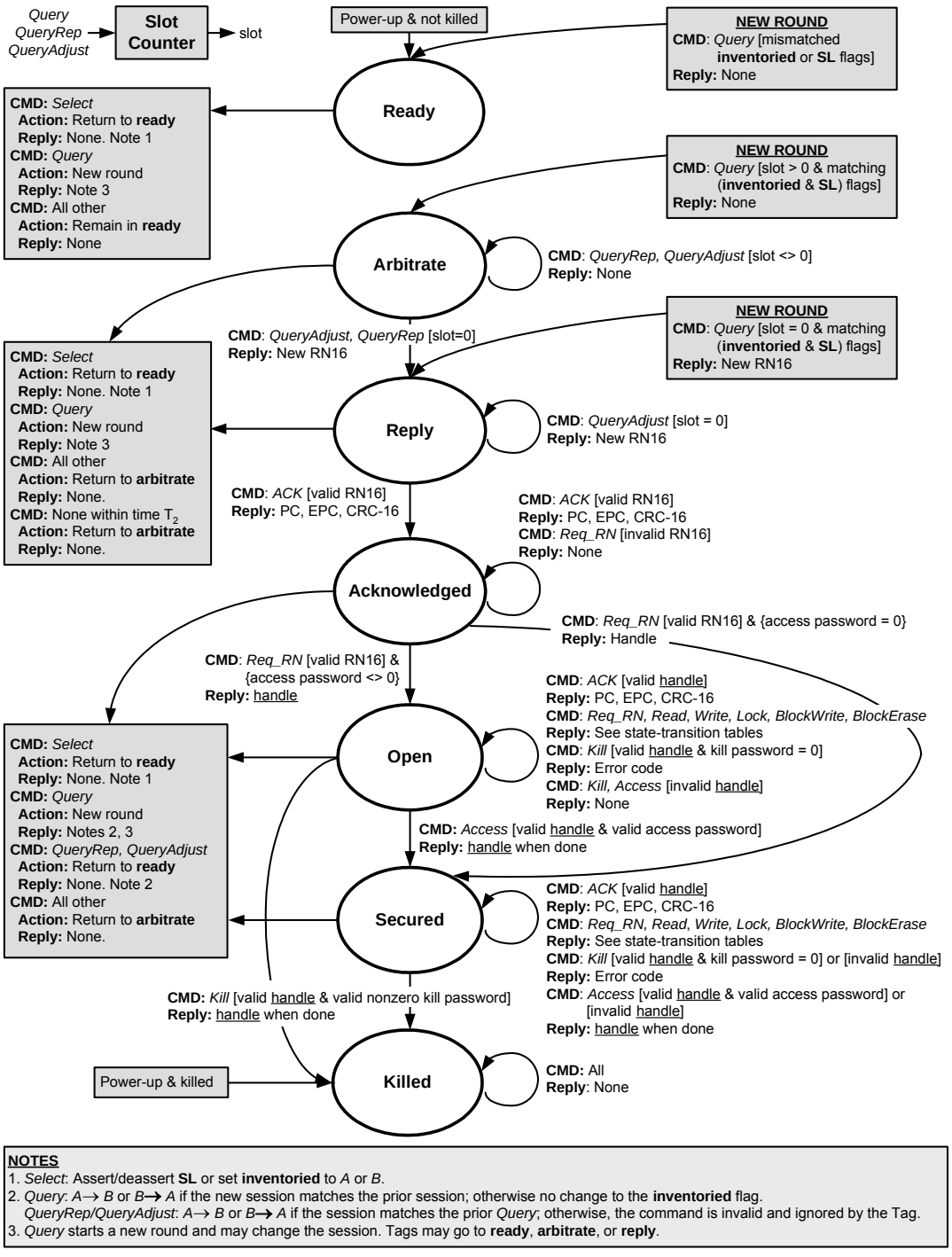


Figure 4-1 Monza™ Tag State Diagram

4.2 Monza™ State-Transition Tables

4.2.1 Present State: Ready

Table 4-1 Ready State-Transition

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	**reply**
	slot<>0; matching **inventoried** & **SL** flags	–	**arbitrate**
	Otherwise	–	**ready**
QueryRep	all	–	**ready**
QueryAdjust	all	–	**ready**
ACK	all	–	**ready**
NAK	all	–	**ready**
Req_RN	all	–	**ready**
Select	all	assert or deassert **SL**, or set **inventoried** to *A* or *B*	**ready**
Read	all	–	**ready**
Write	all	–	**ready**
Kill	all	–	**ready**
Lock	all	–	**ready**
Access	all	–	**ready**
BlockWrite	all	–	**ready**
Invalid [2]	all	–	**ready**

Note 1. *Query* starts a new round and may change the session. *Query* also instructs a tag to load a new random value into its slot counter.

Note 2. "Invalid" means an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than a *Query*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the tag.

4.2.2 Present State: Arbitrate

Table 4-2 Arbitrate State-Transition

Command	Condition	Action	Next State
Query [1,2]	slot=0; matching inventoried & SL flags	backscatter new RN16	**reply**
	slot<>0; matching inventoried & SL flags	–	**arbitrate**
	otherwise	–	**ready**
QueryRep	slot=0 after decrementing slot counter	backscatter new RN16	**reply**
	slot<>0 after decrementing slot counter	–	**arbitrate**
QueryAdjust [2]	slot=0	backscatter new RN16	**reply**
	slot<>0	–	**arbitrate**
ACK	all	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	all	–	**arbitrate**
Select	all	assert or deassert SL, or set inventoried to A or B	**ready**
Read	all	–	**arbitrate**
Write	all	–	**arbitrate**
Kill	all	–	**arbitrate**
Lock	all	–	**arbitrate**
Access	all	–	**arbitrate**
Erase	all	–	**arbitrate**
BlockWrite	all	–	**arbitrate**
Invalid [3]	all	–	**arbitrate**

Note 1. *Query* starts a new round and may change the session.

Note 2. *Query* and *QueryAdjust* instruct a tag to load a new random value into its slot counter.

Note 3. "Invalid" means an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than a *Query*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the tag.

4.2.3 Present State: Reply

Table 4-3 Reply State-Transition

Command	Condition	Action	Next State
Query [1,2]	slot=0; matching **inventoried** & **SL** flags	backscatter new RN16	**reply**
	Slot<>0; matching **inventoried** & **SL** flags	–	**arbitrate**
	otherwise	–	**ready**
QueryRep [3]	all	–	**arbitrate**
QueryAdjust [2, 3]	Slot=0	backscatter new RN16	**reply**
	Slot<>0	–	**arbitrate**
ACK	valid RN16	backscatter {PC, EPC, CRC-16} or {00000_2, truncated EPC, CRC-16}	**acknowledged**
	invalid RN16	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	all	–	**arbitrate**
Select	all	assert or deassert **SL**, or set **inventoried** to *A* or *B*	**ready**
Read	all	–	**arbitrate**
Write	all	–	**arbitrate**
Kill	all	–	**arbitrate**
Lock	all	–	**arbitrate**
Access	all	–	**arbitrate**
BlockWrite	all	–	**arbitrate**
T_2 *timeout*	$T_2 > 20.0T_{pri}$ (see 3.5.1)	–	**arbitrate**
Invalid [4]	all	–	**reply**

Note 1. *Query* starts a new round and may change the session.

Note 2. *Query* and *QueryAdjust* instruct a tag to load a new random value into its slot counter.

Note 3. A *QueryRep* and *QueryAdjust* whose session parameter does not match that of the current inventory round is invalid.

Note 4. "Invalid" means an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than a *Query*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the tag.

4.2.4 Present State: Acknowledged

Table 4-4 Acknowledged State-Transition

Command	Condition	Action	Next State
Query[1]	slot=0; matching **inventoried**[2] & SL flags	backscatter new RN16; transition inventoried from A→B or B→A if and only if new session matches prior session	**reply**
	slot<>0; matching **inventoried** & SL flags	transition inventoried from A→B or B→A if and only if new session matches prior session	**arbitrate**
	otherwise	transition inventoried from A→B or B→A if and only if new session matches prior session	**ready**
QueryRep	all	transition inventoried from A→B or B→A	**ready**
QueryAdjust	all	transition inventoried from A→B or B→A	**ready**
ACK	valid RN16	backscatter {PC, EPC, CRC-16} or {00000_2, truncated EPC, CRC-16}	**acknowledged**
	invalid RN16	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	valid RN16 & access password<>0	backscatter handle	**open**
	valid RN16 & access password=0	backscatter handle	**secured**
	Invalid RN16	–	**acknowledged**
Select	all	assert or deassert SL, or set inventoried to A or B	**ready**
Read	all	–	**arbitrate**
Write	all	–	**arbitrate**
Kill	all	–	**arbitrate**
Lock	all	–	**arbitrate**
Access	all	–	**arbitrate**
BlockWrite	all	–	**arbitrate**
T_2 timeout	$T_2 > 20.0T_{pri}$ (see 3.5.1)	–	**arbitrate**
Invalid[3]	all	–	**acknowledged**

Note 1. *Query* starts a new round and may change the session.

Note 2. A tag transitions its **inventoried** flag prior to evaluating the condition.

Note 3. "Invalid" means an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than a *Query*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the tag.

4.2.5 Present State: Open

Table 4-5 Open State-Transition

Command	Condition	Action	Next State
Query [1]	slot=0; matching **inventoried**[2] & **SL** flags	backscatter new RN16; transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	**reply**
	slot<>0; matching **inventoried**[2] & **SL** flags	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	**arbitrate**
	Otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	**ready**
QueryRep	All	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
QueryAdjust	all	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
ACK	valid handle	backscatter {PC, EPC, CRC-16} or {00000_2, truncated EPC, CRC-16}	**open**
	invalid handle	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	valid handle	backscatter new RN16	**open**
	invalid handle	–	**open**
Select	all	assert or deassert **SL**, or set **inventoried** to *A* or *B*	**ready**
Read	valid handle & valid memory access	backscatter data and handle	**open**
	valid handle & invalid memory access	backscatter error code	**open**
	invalid handle	–	**open**
Write	valid handle & valid memory access	backscatter handle when done	**open**
	valid handle & invalid memory access	backscatter error code	**open**
	invalid handle	–	**open**
Kill	Valid handle & valid nonzero kill password	backscatter handle when done	**killed**
	valid handle & invalid nonzero kill password	–	**arbitrate**
	valid handle & kill password=0	backscatter error code	**open**
	invalid handle	–	**open**
Lock	all	–	**open**
Access (see Figure 4-5)	valid handle & valid access password	backscatter handle	**secured**
	valid handle & invalid access password	–	**arbitrate**
	invalid handle	–	**open**
BlockWrite	valid handle & valid memory access	backscatter handle when done	**open**
	valid handle & invalid memory access	backscatter error code	**open**

Command	Condition	Action	Next State
	invalid handle	–	**open**
Invalid[3]	all, excluding commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Table 4-4 and Table 4-6)	–	**open**
	Commands, except *Req_RN*, interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Table 4-4 and Table 4-6)	–	**arbitrate**

Note 1. *Query* starts a new round and may change the session.

Note 2. A tag transitions its **inventoried** flag prior to evaluating the condition.

Note 2. "Invalid" means an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than a *Query*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the tag.

4.2.6 Present State: Secured

Table 4-6 Secured State-Transition

Command	Condition	Action	Next State
Query[1]	slot=0; matching **inventoried**[2]& **SL** flags	backscatter new RN16; transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	**reply**
	slot<>0; matching **inventoried**[2]& **SL** flags	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	**arbitrate**
	otherwise	transition **inventoried** from *A*→*B* or *B*→*A* if and only if new session matches prior session	**ready**
QueryRep	all	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
QueryAdjust	all	transition **inventoried** from *A*→*B* or *B*→*A*	**ready**
ACK	valid handle	backscatter {PC, EPC, CRC-16} or {00000_2, truncated EPC, CRC-16}	**secured**
	invalid handle	–	**arbitrate**
NAK	all	–	**arbitrate**
Req_RN	valid handle	backscatter new RN16	**secured**
	invalid handle	–	**secured**
Select	all	assert or deassert **SL**, or set **inventoried** to *A* or *B*	**ready**
Read	valid handle & valid memory access	backscatter data and handle	**secured**
	valid handle & invalid memory access	backscatter error code	**secured**
	invalid handle	–	**secured**
Write	valid handle & valid memory access	backscatter handle when done	**secured**

Command	Condition	Action	Next State
	valid handle & invalid memory access	backscatter error code	**secured**
	invalid handle	–	**secured**
Kill	valid handle & valid nonzero kill password	backscatter handle when done	**killed**
	valid handle & invalid nonzero kill password	–	**arbitrate**
	valid handle & kill password=0	backscatter error code	**secured**
	invalid handle	–	**secured**
Lock	valid handle & valid lock payload	backscatter handle when done	**secured**
	valid handle & invalid lock payload	backscatter error code	**secured**
	invalid handle	–	**secured**
Access (see Figure 4-5)	valid handle & valid access password	backscatter handle	**secured**
	valid handle & invalid access password	–	**arbitrate**
	invalid handle	–	**secured**
BlockWrite	valid handle & valid memory access	backscatter handle when done	**secured**
	valid handle & invalid memory access	backscatter error code	**secured**
	invalid handle	–	**secured**
Invalid[3]	all, excluding commands interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Table 4-4 and Table 4-6)	–	**secured**
	Commands, except *Req_RN*, interspersed between successive *Kill* or *Access* commands in a kill or access sequence, respectively (see Table 4-4 and Table 4-6)	–	**arbitrate**

Note 1. *Query* starts a new round and may change the session. *Query* also instructs a tag to load a new random value into its slot counter.

Note 2. A tag transitions its **inventoried** flag prior to evaluating the condition.

Note 3. "Invalid" means an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, a command (other than a *Query*) with a session parameter not matching that of the inventory round currently in progress, or any other command either not recognized or not executable by the tag.

4.2.7 Present State: Killed

Table 4-7 Killed State Transition

Command	Condition	Action	Next State
Query	all	–	**killed**
QueryRep	all	–	**killed**
QueryAdjust	all	–	**killed**
ACK	all	–	**killed**
NAK	all	–	**killed**
Req_RN	all	–	**killed**
Select	all	–	**killed**
Read	all	–	**killed**
Write	all	–	**killed**
Kill	all	–	**killed**
Lock	all	–	**killed**
Access	all	–	**killed**
BlockWrite	all	–	**killed**
Invalid [1]	all	–	**killed**

Note 1. "Invalid" means an erroneous command, an unsupported command, a command with invalid parameters, a command with a CRC error, or any other command either not recognized or not executable by the tag.

4.3 Monza™ Flags

4.3.1 Selected Flag

Monza™-based tags implement a selected flag, **SL**, which a reader may assert or deassert using a *Select* command. The Sel parameter in the *Query* command allows a reader to inventory tags that have **SL** either asserted or deasserted (i.e., **SL** or ~**SL**), or to ignore the flag and inventory tags regardless of their **SL** value. **SL** is not associated with any particular session; it applies to all tags, regardless of the session.

A Monza™ tag's **SL** flag exhibits the persistence times shown in Table 4-8. The tag powers-up with its **SL** flag either asserted or deasserted, depending on the stored value, unless the tag has lost power for a time greater than the **SL** persistence time, in which case the tag will power-up with its **SL** flag deasserted (set to ~**SL**). The tag is capable of asserting or deasserting its **SL** flag within 2 ms, regardless of the initial flag value. Furthermore, the tag refreshes its **SL** flag when powered, meaning that every time it loses power, its **SL** flag has the persistence times shown in Table 4-8.

4.3.2 Session Flags

Monza™ tags provide four (4) sessions (denoted S0, S1, S2, and S3), and participate in one, and only one, session during an inventory round. Two or more readers can use sessions to independently inventory a common tag population.

Tags maintain an independent **inventoried** flag for each session. Each of the four **inventoried** flags has two values, denoted *A* and *B*. At the beginning of each and every inventory round a reader chooses to inventory either *A* or *B* tags in one of the four sessions. Tags participating in an inventory round in one session will neither use nor modify the **inventoried** flag for a different session. The **inventoried** flags are the only resource a tag provides separately and independently to a given session; all other tag resources are shared among sessions.

Sessions allow tags to associate a separate and independent **inventoried** flag to each of several readers.

Tag **inventoried** flags exhibit the persistence times shown in Table 4-8. The tag powers-up with its **inventoried** flags set as follows:

- The S0 **inventoried** flag is set to *A*.
- The S1 **inventoried** flag is set to either *A* or *B*, depending on its stored value, unless its persistence time has expired, in which case the tag powers-up with its S1 **inventoried** flag set to *A*. Because the S1 **inventoried** flag is not automatically refreshed, it may revert from *B* to *A* even when the tag is powered.
- The S2 **inventoried** flag is set to either *A* or *B*, depending on its stored value, unless the tag has lost power for a time greater than its persistence time, in which case the tag will power-up with the S2 **inventoried** flag set to *A*.
- The S3 **inventoried** flag is set to either *A* or *B*, depending on its stored value, unless the tag has lost power for a time greater than its persistence time, in which case the tag powers-up with its S3 **inventoried** flag set to *A*.

Monza™ tags are capable of setting any of their inventoried flags to either *A* or *B* in 2 ms or less, regardless of the initial flag value. The tag will refresh its S2 and S3 flags while powered, meaning that every time a tag loses power its S2 and S3 **inventoried** flags exhibit the persistence times shown in Table 4-8. A tag will not let its S1 **inventoried** flag lose persistence while the tag is participating in an inventory round. Instead, the tag will retain the flag value until the next *Query* command, at which point the flag may lose its persistence (unless the flag was refreshed during the round, in which case the flag shall assume its new value and new persistence).

Table 4-8 Tag Flags and their Persistence

Flag	Required persistence[2]
S0 **inventoried** flag	Tag energized: Indefinite Tag not energized: None
S1 **inventoried** flag[1]	Tag energized: Nominal temperature range: 500 ms < persistence < 5 s Tag not energized: Nominal temperature range: 500 ms < persistence < 5 s
S2 **inventoried** flag[1]	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2 s < persistence
S3 **inventoried** flag[1]	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2 s < persistence
Selected (**SL**) flag[1]	Tag energized: Indefinite Tag not energized: Nominal temperature range: 2 s < persistence

Note 1: For a randomly chosen and sufficiently large tag population, 95% of the tag persistence times meet the persistence requirement, with a 90% confidence interval.

Note 2. Gen 2 specifies persistence times over nominal temperature range only (–25 ºC to +40 ºC).

4.4 Tag Commands

Tag commands are described in the following sections. If a Monza™ tag detects extra bits within 2 Tari of the end of the command, the tag will interpret the command as an invalid command.

4.4.1 Global Commands

4.4.1.1 Select Command

Select selects a particular tag population based on user-defined criteria, enabling union (U), intersection (∩), and negation (~) based tag partitioning. Readers perform ∩ and U operations by issuing successive *Select* commands. *Select* may assert or deassert a tag's **SL** flag, which applies across all four sessions, or it may set a tag's **inventoried** flag to either *A* or *B* in any one of the four sessions.

Monza™ tags implement the *Select* command shown in Table 4-9. Target indicates whether the *Select* modifies a tag's **SL** or **inventoried** flag, and in the case of the **inventoried** flag, for which session. Action elicits the tag response shown in Table 4-10. The criteria for determining whether a tag is matching or non-matching are specified in the MemBank, Pointer, Length, and Mask fields. Truncate indicates whether a tag's backscattered reply is truncated to include only those EPC and CRC-16 bits following Mask. *Select* passes the following parameters from reader to tags:

- Target indicates whether the *Select* command modifies a tag's **SL** flag or its **inventoried** flag, and in the case of **inventoried** it further specifies one of four sessions. A *Select* command that modifies **SL** shall not modify **inventoried**, and vice versa. The tag will ignore *Select* commands whose Target is 101, 110, or 111 (all values binary notation).
- Action indicates whether matching tags assert or deassert **SL**, or set their **inventoried** flag to *A* or to *B*. Tags conforming to the contents of the MemBank, Pointer, Length, and Mask fields are considered matching. Tags not conforming to the contents of these fields are considered non-matching.
- MemBank specifies whether Mask applies to the EPC or TID. *Select* commands apply to a single memory bank. Successive *Selects* may apply to different banks. MemBank will not specify Reserved memory; if a tag receives a *Select* specifying MemBank = 00_2 it will ignore the *Select*.
- Pointer, Length, and Mask: Pointer and Length describe a memory range. Pointer references a memory bit address (Pointer is not restricted to word boundaries) and uses EBV formatting. Length is 8 bits, allowing Masks from 0 to 255 bits in length. Mask, which is Length bits long, contains a bit string that a tag compares against the memory location that begins at Pointer and ends Length bits later. If Length is zero then all tags are considered matching. If Pointer and Length reference a memory location that does not exist on the tag then the tag considers the *Select* to be non-matching. If Length is zero then all tags will be considered matching, unless Pointer references a memory location that does not exist on the tag, in which case, the tag will consider the *Select* to be non-matching.
- Truncate: If an reader asserts Truncate, and if a subsequent *Query* specifies Sel=10 or Sel=11, then the matching tags truncate their reply to an *ACK* to that portion of the EPC immediately following Mask, followed by the CRC-16 stored in EPC memory 00_h to $0F_h$. Readers shall assert Truncate:
  - in the last (and only in the last) *Select* that the reader issues prior to sending a *Query*,
  - if and only if the *Select* has Target = 100_2, and
  - if and only if Mask ends in the EPC.

  These constraints *do not* preclude a reader from issuing multiple *Select* commands that target the **SL** and/or **inventoried** flags. They *do* require that a reader assert Truncate only in the last *Select*, and only if this last *Select* targets the **SL** flag. Tags power-up with Truncate de-asserted.

Tags decide whether to truncate their backscattered EPC on the basis of the most recently received *Select*. If a tag receives a *Select* with Truncate=1, but Target<>100_2, the tag will ignore the *Select*. If a tag receives a *Select* in which Truncate=1, but Mask ends outside the EPC specified in the PC bits, the tag will ignore the *Select*.

Mask may end at the last bit of the EPC, in which case a selected tag backscatters its CRC-16.

Truncated replies never include PC bits, because Mask must end in the EPC.

The tag prefaces its truncated reply with four leading zeros (0000_2) inserted between the preamble and the truncated reply. The tag will not recalculate the CRC-16 for a truncated reply.

- Readers may use a *Select* command to reset all tags in a session to **inventoried** state *A*, by issuing a *Select* with Action = 000_2 and a Length value of zero.

The CRC-16 is calculated over the first command-code bit to the Truncate bit. Tags do not reply to a *Select*.

Table 4-9 Select Command

	Command	Target	Action	MemBank	Pointer	Length	Mask	Truncate	CRC-16
# of bits	4	3	3	2	EBV	8	Variable	1	16
Description	1010	000: **Inventoried** (S0) 001: **Inventoried** (S1) 010: **Inventoried** (S2) 011: **Inventoried** (S3) 100: **SL** 101: RFU[1] 110: RFU 111: RFU	See Table 4-10	00: RFU 01: EPC 10: TID 11: User[2]	Starting Mask address	Mask length (bits)	Mask value	0: Disable truncation 1: Enable truncation	

Note 1: Reserved for Future Use

Note 2: Monza™ does not implement User memory

Table 4-10 Tag Response to Action Parameter

Action	Matching	Non-Matching
000	assert **SL** or **inventoried** → *A*	deassert **SL** or **inventoried** → *B*
001	assert **SL** or **inventoried** → *A*	do nothing
010	do nothing	deassert **SL** or **inventoried** → *B*
011	negate **SL** or (*A* → *B*, *B* → *A*)	do nothing
100	deassert **SL** or **inventoried** → *B*	assert **SL** or **inventoried** → *A*
101	deassert **SL** or **inventoried** → *B*	do nothing
110	do nothing	assert **SL** or **inventoried** → *A*
111	do nothing	negate **SL** or (*A* → *B*, *B* → *A*)

4.4.2 Inventory Commands

4.4.2.1 Query

Monza™ tags implement the *Query* command shown in Table 4-11. *Query* initiates and specifies an inventory round. *Query* includes the following fields:

- *DR* (TRcal divide ratio) sets the reverse link frequency as described in 3.1.4 and Table 3-5.
- *M* (cycles per symbol) sets the reverse link data rate and modulation format as shown in Table 3-6.
- TRext chooses whether the T=>R preamble is prepended with a pilot tone as described in 3.4.3 and 3.4.5.
- Sel chooses which tags respond to the *Query* (see 4.4.1.1).
- Session chooses a session for the inventory round.
- Target selects whether tags whose **inventoried** flag is *A* or *B* participate in the inventory round. Tags may change their inventoried flag from *A* to *B* (or vice versa) as a result of being singulated.
- *Q* sets the number of slots in the round.

The CRC-5 is calculated over the first command-code bit to the last *Q* bit. If a tag receives a *Query* with a CRC-5 error it will ignore the command.

Upon receiving a *Query*, tags with matching Sel and Target pick a random value in the range $(0, 2^{Q}-1)$, inclusive, and load this value into their slot counter. If a tag, in response to the *Query*, loads its slot counter with zero, then its reply to a *Query* is as shown in Table 4-12; otherwise the tag will remain silent.

A *Query* may initiate an inventory round in a new session, or in the prior session. If a tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose session parameter matches the prior session, it will invert its **inventoried** flag (i.e., $A \rightarrow B$ or $B \rightarrow A$) for the session. If a tag in the **acknowledged**, **open**, or **secured** states receives a *Query* whose session parameter does not match the prior session it will leave its **inventoried** flag for the prior session unchanged when beginning the new round. Tags support all *DR* and *M* values.

Tags in any state other than **killed** will execute a *Query* command; Tags in the **killed** state will ignore a *Query*.

Table 4-11 Query Command

	Command	DR	M	Trext	Sel	Session	Target	Q	CRC-5
# of bits	4	1	2	1	2	2	1	4	5
Description	1000	0: DR=8 1: DR=64/3	00: M=1 01: M=2 10: M=4 11: M=8	0: No pilot tone 1: Use pilot tone	00: All 01: All 10: ~**SL** 11: **SL**	00: S0 01: S1 10: S2 11: S3	0: *A* 1: *B*	0–15	

Table 4-12 Tag Reply to Query Command

	Response
# of bits	16
Description	RN16

4.4.2.2 QueryAdjust

QueryAdjust adjusts *Q* (the number of slots in an inventory round) without changing any other round parameters. *QueryAdjust* includes the following fields:

- Session corroborates the session number for the inventory round (see 4.4.2.1). If a tag receives a *QueryAdjust* whose session number is different from the session number in the *Query* that initiated the round it will ignore the command.
- UpDn determines whether and how the tag adjusts *Q*, as follows:

  110: Increment *Q* (i.e., $Q = Q + 1$).

  000: No change to *Q*.

  011: Decrement *Q* (i.e., $Q = Q - 1$).

  If a tag receives a *QueryAdjust* with an UpDn value different from those specified above it will ignore the command.

Monza™ tags maintain a running count of the current *Q* value. The initial *Q* value is specified in the *Query* command that started the inventory round; one or more subsequent *QueryAdjust* commands may modify *Q*.

A *QueryAdjust* is prepended with a frame-sync.

Upon receiving a *QueryAdjust* tags first update *Q*, then pick a random value in the range $(0, 2^Q - 1)$, inclusive, and load this value into their slot counter. If a tag, in response to the *QueryAdjust*, loads its slot counter with zero, then its reply to a *QueryAdjust* shall be shown in Table 4-14; otherwise, the tag will remain silent. Tags respond to a *QueryAdjust* only if they received a prior *Query*.

Tags in the **acknowledged**, **open**, or **secured** states that receive a *QueryAdjust* invert their **inventoried** flag (i.e., $A \rightarrow B$ or $B \rightarrow A$, as appropriate) for the current session and transition to **ready**.

Table 4-13 QueryAdjust Command

	Command	Session	UpDn
# of bits	4	2	3
Description	1001	00: S0 01: S1 10: S2 11: S3	110: $Q = Q + 1$ 000: No change to Q 011: $Q = Q - 1$

Table 4-14 Tag Reply to QueryAdjust Command

	Response
# of bits	16
Description	RN16

4.4.2.3 QueryRep

QueryRep instructs tags to decrement their slot counters and, if slot = 0 after decrementing, to backscatter an RN16 to the reader.

QueryRep includes the following field:

- Session corroborates the session number for the inventory round (see 4.4.2.1). If a tag receives a *QueryRep* whose session number is different from the session number in the *Query* that initiated the round it will ignore the command.

A *QueryRep* is prepended with a frame-sync.

If a tag, in response to the *QueryRep*, decrements its slot counter and the decremented slot value is zero, then its reply to a *QueryRep* is as shown in Table 4-16; otherwise the tag will remain silent. Tags respond to a *QueryRep* only if they received a prior *Query*.

Tags in the **acknowledged**, **open**, or **secured** states that receive a *QueryRep* invert their **inventoried** flag (i.e., $A \rightarrow B$ or $B \rightarrow A$, as appropriate) for the current session and transition to **ready**.

Table 4-15 QueryRep Command

	Command	Session
# of bits	2	2
Description	00	00: S0 01: S1 10: S2 11: S3

Table 4-16 Tag Reply to QueryRep Command

	Response
# of bits	16
Description	RN16

4.4.2.4 ACK command

A reader sends an *ACK* to acknowledge a single tag. *ACK* echoes the tag's backscattered RN16.

If a reader issues an *ACK* to a Monza™ tag in the **reply** or **acknowledged** states, then the echoed RN16 will be the RN16 that the tag previously backscattered as it transitioned from the **arbitrate** state to the **reply** state. If a reader issues an *ACK* to a tag in the **open** or **secured** states, then the echoed RN16 will be the tag's handle.

An *ACK* is prepended with a frame-sync.

The tag reply to a successful *ACK* is as shown in Table 4-18. The reply may be truncated. A tag that receives an *ACK* with an incorrect RN16 or an incorrect handle (as appropriate) will return to **arbitrate** without responding, unless the tag is in **ready** or **killed**, in which case it shall ignore the *ACK* and remain in its present state.

Table 4-17 ACK Command

	Command	RN
# of bits	2	16
Description	01	Echoed RN16 or handle

Table 4-18 Tag reply to successful ACK command

	Response
# of bits	16 to 288
Description	{PC, EPC, CRC-16} OR {0000_2, truncated EPC, CRC-16}

4.4.2.5 NAK

Monza™ tags implement the *NAK* command as shown in Table 4-19. *NAK* will return all tags to the **arbitrate** state unless they are in **ready** or **killed**, in which case they will ignore the *NAK* and remain in their current state.

A *NAK* is prepended with a frame-sync.

Tags do not reply to a *NAK*.

Table 4-19 NAK Command

	Command
# of bits	8
Description	11000000

4.4.3 Access Commands

The set of access commands comprises *Req_RN, Read, Write*, *Lock*, *Kill*, *Access, and Blockwrite*. When and whether an access command is valid as specified depends on the tag's state, password, and whether the password is locked.

4.4.3.1 Req_RN

Req_RN instructs a tag to backscatter a new RN16. Both the reader's command and the tag's response depend on the tag's state:

- **Acknowledged** state: When issuing a *Req_RN* command to a tag in the **acknowledged** state, a reader shall include the tag's last backscattered RN16 as a parameter in the *Req_RN*. The *Req_RN* command is protected by a CRC-16 calculated over the command bits and the RN16. If the tag receives the *Req_RN* with a valid CRC-16 and a valid RN16 it will generate and store a new RN16 (denoted

handle), backscatter this handle, and transition to the **open** or **secured** state. The choice of ending state depends on the tag's access password, as follows:

- Access password <> 0: Tag transitions to **open** state.
- Access password = 0: Tag transitions to **secured** state.

If the tag receives the *Req_RN* command with a valid CRC-16 but an invalid RN16 it will ignore the *Req_RN* and remain in the **acknowledged** state.

- **Open** or **secured** states: When issuing a *Req_RN* command to a tag in the **open** or **secured** states, a reader shall include the tag's handle as a parameter in the *Req_RN*. The *Req_RN* command is protected by a CRC-16 calculated over the command bits and the handle. If the tag receives the *Req_RN* with a valid CRC-16 and a valid handle it will generate and backscatter a new RN16. If the tag receives the *Req_RN* with a valid CRC-16 but an invalid handle it will ignore the *Req_RN*. In either case the tag will remain in its current state (**open** or **secured**, as appropriate).

The first bit of the backscattered RN16 is denoted the MSB; the last bit is denoted the LSB.

A *Req_RN* is pre-appended with a frame-sync.

The tag reply to a *Req_RN* is as shown in Table 4-21. The RN16 and handle are protected by a CRC-16.

Table 4-20 REQ_RN Command

	Command	RN	CRC-16
# of bits	8	16	16
Description	11000001	Prior RN16 or handle	

Table 4-21 Tag Reply to REQ_RN Command

	RN	CRC-16
# of bits	16	16
Description	New RN16 or handle	

4.4.3.2 Read

Read allows a reader to read part or all of a tag's Reserved, EPC, or TID memory. *Read* has the following fields:

- MemBank specifies whether the *Read* accesses Reserved, EPC, or TID memory. *Read* commands apply to a single memory bank. Successive *Reads* may apply to different banks.
- WordPtr specifies the starting word address for the memory read, where words are 16 bits in length. WordPtr uses EBV formatting.
- WordCount specifies the number of 16-bit words to be read. If WordCount = 00_h the tag will backscatter the contents of the chosen memory bank starting at WordPtr and ending at the end of the bank, unless MemBank = 01, in which case the tag will backscatter the EPC memory contents starting at WordPtr and ending at the length of the EPC specified by the first 5 bits of the PC if WordPtr lies within the EPC, and will backscatter the EPC memory contents starting at WordPtr and ending at the end of EPC memory if WordPtr lies outside the EPC.

The *Read* command also includes the tag's handle and a CRC-16. The CRC-16 is calculated over the first command-code bit to the last handle bit.

If a tag receives a *Read* with a valid CRC-16 but an invalid handle, it will ignore the *Read* and remain in its current state (**open** or **secured**, as appropriate).

A *Read* is prepended with a frame-sync.

If all of the memory words specified in a *Read* exist and none are read-locked, the tag reply to the *Read* is as shown in Table 4-23. The tag responds by backscattering a header (a 0-bit), the requested memory words, and its handle. The reply includes a CRC-16 calculated over the 0-bit, memory words, and handle.

If a one or more of the memory words specified in the *Read* command either do not exist or are read-locked, the tag will backscatter an error code, within time T_1 in Table 3-7, rather than the reply shown in Table 4-23.

Table 4-22 Read Command

	Command	MemBank	WordPtr	WordCount	RN	CRC-16
# of bits	8	2	EBV	8	16	16
Description	11000010	00: Reserved 01: EPC 10: TID 11: User[1]	Starting address pointer	Number of words to read	handle	

Note 1: Monza™ does not implement User memory.

Table 4-23 Tag Reply to Read Command

	Header	Memory Words	RN	CRC-16
# of bits	1	Variable	16	16
Description	0	Data	handle	

Table 4-24 Tag-error reply format

	Header	Error Code	RN	CRC-16
# of bits	1	8	16	16
Description	1	Error code	handle	

4.4.3.3 Write

Write allows a reader to write a word in a tag's Reserved or EPC memory (Monza™ does not implement User memory, and Monza™'s TID is implemented in ROM). *Write* has the following fields:

- MemBank specifies whether the *Write* occurs in Reserved, EPC, or TID memory.
- WordPtr specifies the word address for the memory write, where words are 16 bits in length. WordPtr uses EBV formatting.
- Data contains a 16-bit word to be written. Before each and every *Write* the reader shall first issue a *Req_RN* command; the tag responds by backscattering a new RN16. The reader shall cover-code the data by EXORing it with this new RN16 prior to transmission.

The *Write* command also includes the tag's handle and a CRC-16. The CRC-16 is calculated over the first command-code bit to the last handle bit.

If a tag in the **open** or **secured** states receives a *Write* with a valid CRC-16 but an invalid handle, or it receives a *Write* before which the immediately preceding command was not a *Req_RN*, it will ignore the *Write* and remain in its current state.

A *Write* is prepended with a frame-sync.

After issuing a *Write* a reader transmits CW for T_{REPLY} or the maximum T_{REPLY}, where T_{REPLY} is the time between the reader's *Write* command and the tag's backscattered reply (see T_{REPLY} in Table 3-7). A reader may observe several possible outcomes from a *Write*, depending on the success or failure of the tag's memory-write operation:

- **The *Write* succeeds:** After completing the *Write* a tag shall backscatter the reply shown in Table 4-26 comprising a header (a 0-bit), the tag's handle, and a CRC-16 calculated over the 0-bit and handle. If the reader observes this reply within the maximum T_{REPLY} (see T_{REPLY} in Table 3-7) then the *Write* completed successfully.
- **The tag encounters an error:** The tag will backscatter an error code during the CW period rather than the reply shown in Table 4-26.
- **The *Write* does not succeed:** If the reader does not observe a reply within the maximum T_{REPLY} as specified in Table 3-7, then the *Write* did not complete successfully. The reader may issue a *Req_RN* command (containing the tag's handle) to verify that the tag is still in the reader's field, and may reissue the *Write* command.

Table 4-25 Write Command

	Command	MemBank	WordPtr	Data	RN	CRC-16
# of bits	8	2	EBV	16	16	16
Description	11000011	00: Reserved 01: EPC 10: TID[1] 11: User[2]	Address pointer	RN16 ⊗ word to be written	handle	

Note 1: Monza™ TID is implemented in ROM and cannot be user-written.
Note 2: Monza™ does not implement User memory.

Table 4-26 Tag reply to successful write command

	Header	RN	CRC-16
# of bits	1	16	16
Description	0	handle	

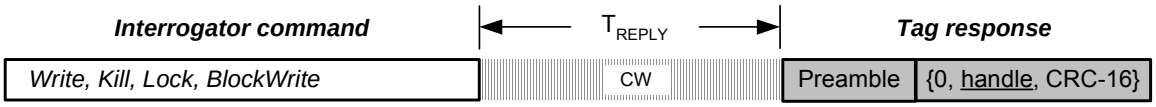


Figure 4-2 Successful Write Sequence

4.4.3.4 Kill

Kill allows a reader to permanently disable a tag.

To kill a tag, a reader must follow the multi-step kill procedure outlined in Figure 4-3. Briefly, a reader issues two *Kill* commands, the first containing the 16 MSBs of the tag's kill password EXORed with an RN16, and the second containing the 16 LSBs of the tag's kill password EXORed with a different RN16. Each EXOR operation is performed MSB first (i.e., the MSB of each half-password shall be EXORed with the MSB of its respective RN16). Prior to issuing each *Kill* the reader first issues a *Req_RN* to obtain a new RN16.

Readers must not intersperse commands other than *Req_RN* between the two successive *Kill* commands. If a tag, after receiving a first *Kill*, receives any command other than *Req_RN* before the second *Kill*, it will return to **arbitrate**, unless the intervening command is a *Query*, in which case the tag will execute the *Query* (inverting its **inventoried** flag if the session parameter in the *Query* matches the prior session).

Tags whose kill password is zero will not execute the kill operation; if such a tag receives a *Kill* it will ignore the command and backscatter an error code (see 3.6.2).

After issuing the second *Kill* a reader shall transmit CW for the lesser of T_{REPLY} or 20ms, where T_{REPLY} is the time between the reader's second *Kill* command and the tag's backscattered reply. A reader may observe several possible outcomes from a *Kill*, depending on the success or failure of the tag's kill operation:

- **The *Kill* succeeds:** After completing the *Kill* the tag will backscatter the reply shown in Table 4-29 and Figure 4-2 comprising a header (a 0-bit), the tag's handle, and a CRC-16 calculated over the 0-bit and handle. Immediately after this reply the tag will render itself silent and will not respond to a reader thereafter. If the reader observes this reply within the maximum T_{REPLY} (see T_{REPLY} in Table 3-7) then the *Kill* completed successfully.
- **The tag encounters an error:** The tag will backscatter an error code during the CW period rather than the reply shown in Table 4-29 (see 3.6.2 for error-code definitions and for the reply format). Tags whose kill password is zero will not implement a kill function. If such a protected tag receives a *Kill* it will ignore the command and backscatter an error code.
- **The *Kill* does not succeed:** If the reader does not observe a reply within the maximum T_{REPLY} as specified in Table 3-7, then the *Kill* did not complete successfully. The reader may issue a *Req_RN* command (containing the tag's handle) to verify that the tag is still in the reader's field, and may reinitiate the multi-step kill procedure outlined in Figure 4-3.

A *Kill* is prepended with a frame-sync.

Upon receiving a valid *Kill* command sequence a tag will render itself killed. The tag's reply to the second *Kill* command will use the extended preamble shown in Figure 3-14 or Figure 3-19, as appropriate (i.e., a tag will reply as if TRext=1 regardless of the TRext value in the *Query* that initiated the round).

Table 4-27 Kill Command

	Command	Password	RFU	RN	CRC-16
# of bits	8	16	3	16	16
Description	11000100	(½ kill password) ⊗ RN16	000_2	handle	

Table 4-28 – Tag reply to the first Kill command

	RN	CRC-16
# of bits	16	16
Description	handle	

Table 4-29 Tag reply to successful Kill procedure

	Header	RN	CRC-16
# of bits	1	16	16
Description	0	handle	

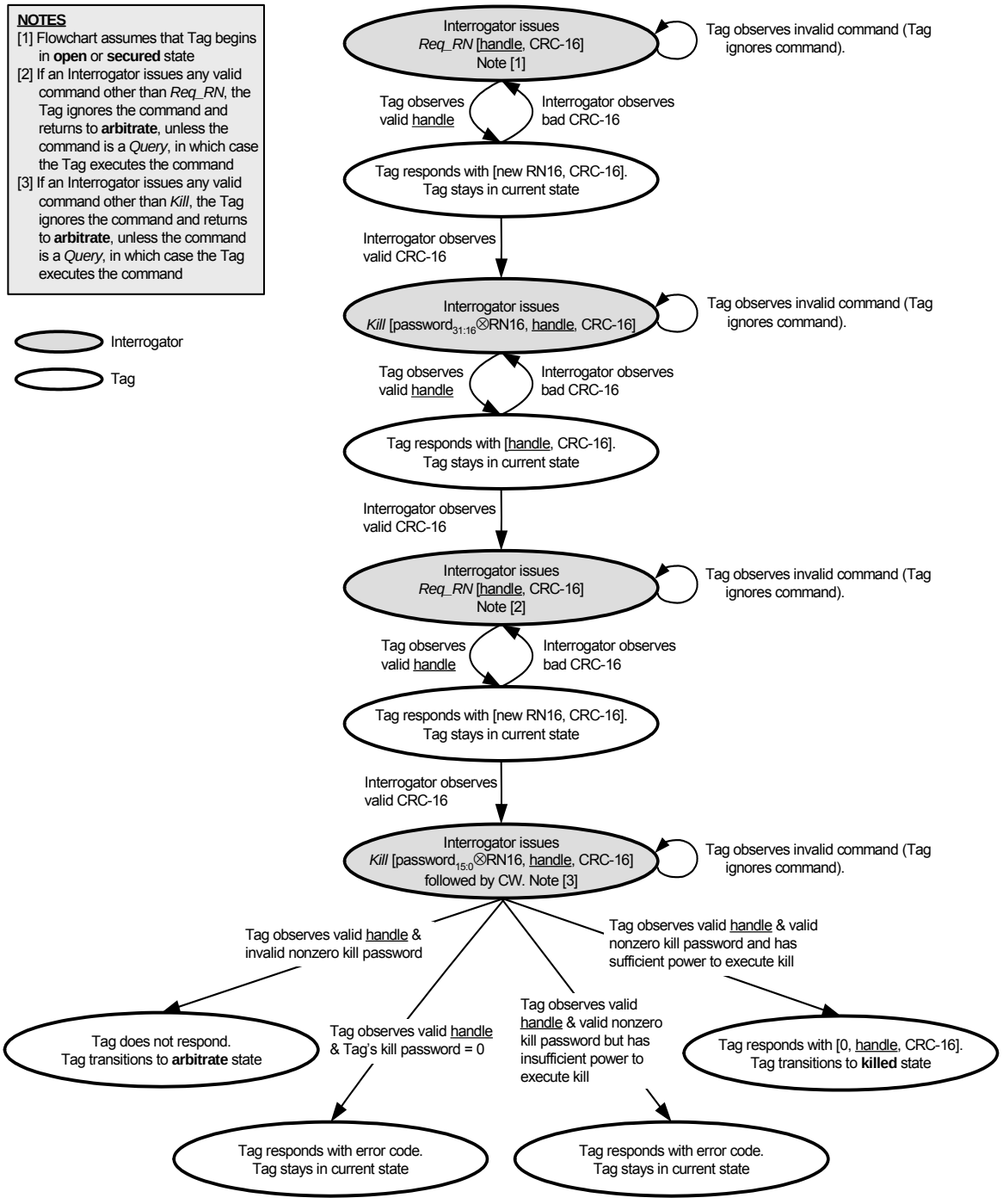


Figure 4-3 Kill Procedure

4.4.3.5 Lock Command

Only tags in the **secured** state may execute a *Lock* command. *Lock* allows a reader to:

- Lock individual passwords
- Lock individual memory banks
- Permalock (make permanently unchangeable) the lock status for a password or memory bank.

Lock contains a 20-bit payload defined as follows:

- The first 10 payload bits are Mask bits. A tag will interpret these bit values as follows:
  - Mask = 0: Ignore the associated Action field and retain the current lock setting.
  - Mask = 1: Implement the associated Action field and overwrite the current lock setting.
- The last 10 payload bits are Action bits. A tag will interpret these bit values as follows:
  - Action = 0: Deassert lock for the associated memory location.
  - Action = 1: Assert lock or permalock for the associated memory location.

The functionality of the various Action fields is described in Table 4-32.

Permalock bits, once asserted, cannot be deasserted. If a tag receives a *Lock* whose payload attempts to deassert a previously asserted permalock bit, the tag will ignore the *Lock* and backscatter an error code. If a tag receives a *Lock* whose payload attempts to reassert a previously asserted permalock bit, the tag will simply ignore this particular Action field and implement the remainder of the *Lock* payload.

While all tags implement memory locking and implement the *Lock* command, tags need not support all the Action fields shown in Figure 4-4 (depending on whether the password location or memory bank associated with an Action field exists and is lockable and/or unlockable). Specifically, if a tag receives a *Lock* it cannot execute because one or more of the passwords or memory banks do not exist, or one or more of the Action fields attempt to change a previously permalocked value, or one or more of the passwords or memory banks are either not lockable or not unlockable, the tag will ignore the entire *Lock* and instead backscatter an error code.

A *Lock* is prepended with a frame-sync.

After issuing a *Lock,* a reader shall transmit CW for the lesser of T_{REPLY} or the maximum T_{REPLY}, where T_{REPLY} is the time between the reader's *Lock* command and the tag's backscattered reply (see T_{REPLY} in Table 3-7). A reader may observe several possible outcomes from a *Lock*, depending on the success or failure of the tag's memory-write operation:

- **The *Lock* succeeds:** After completing the *Lock* the tag will backscatter the reply shown in Table 4-31, comprising a header (a 0-bit), the tag's handle, and a CRC-16 calculated over the 0-bit and handle.
- **The tag encounters an error:** The tag will backscatter an error code during the CW period rather than the reply shown in Table 4-31 (see 3.6.2 for error-code definitions and for the reply format).
- **The *Lock* does not succeed:** If the reader does not observe a reply within the maximum T_{REPLY} as specified in Table 3-7, then the *Lock* did not complete successfully. The reader may issue a *Req_RN* command (containing the tag's handle) to verify that the tag is still in the Reader's field, and may reissue the *Lock*.

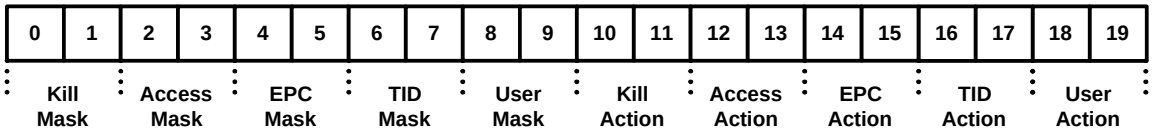
Table 4-30 Lock Command

	Command	Payload	RN	CRC-16
# of bits	8	20	16	16
Description	11000101	Mask and Action Fields	handle	

Table 4-31 Tag reply to Lock command

	Header	RN	CRC-16
# of bits	1	16	16
Description	0	handle	

Lock-Command Payload



Masks and Associated Action Fields[1,2]

	Kill pwd		Access pwd		EPC memory		TID memory[3]		User memory[4]	
	0	1	2	3	4	5	6	7	8	9
Mask	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write
	10	11	12	13	14	15	16	17	18	19
Action	pwd read/ write	perma lock	pwd read/ write	perma lock	pwd write	perma lock	pwd write	perma lock	pwd write	perma lock

Note 1: Lock commands may be applied to two general classes of Monza™ memory: passwords and EPC

Note 2: Default values for Lock values are all 0s.

Note 3: Monza™ TID memory is ROM-based, and is unaffected by the Lock command

Note 4: Monza™ does not incorporate User memory; values written to those bit locations will have no effect.

Figure 4-4 Lock Command Payload Definitions

Table 4-32 Lock Action Field Functionality

pwd-write	Permalock	Description
0	0	Associated memory bank is writeable from either the **open** or **secured** states.
0	1	Associated memory bank is permanently writeable from either the **open** or **secured** states and may never be locked.
1	0	Associated memory bank is writeable from the **secured** state but not from the **open** state.
1	1	Associated memory bank is not writeable from any state.
pwd-read/write	**Permalock**	**Description**
0	0	Associated password location is readable and writeable from either the **open** or **secured** states.
0	1	Associated password location is permanently readable and writeable from either the **open** or **secured** states and may never be locked.
1	0	Associated password location is readable and writeable from the **secured** state but not from the **open** state.
1	1	Associated password location is not readable or writeable from any state.

4.4.3.6 Access Command

Access causes a tag with a nonzero-valued *Access* password to transition from the **open** state to the **secured** state (a tag with a zero-valued *Access* password is never in the **open** state. See Figure 4-1), or if the tag is already in the **secured** state, to remain in **secured**.

To access a tag, a reader must follow the multi-step procedure outlined in Figure 4-5. Briefly, a reader issues two *Access* commands, the first containing the 16 MSBs of the tag's access password EXORed with an RN16, and the second containing the 16 LSBs of the tag's access password EXORed with a different RN16. Each EXOR operation is performed MSB first (i.e., the MSB of each half-password shall be EXORed with the MSB of its respective RN16). Prior to issuing each *Access* command the reader first issues a *Req_RN* to obtain a new RN16.

Readers must not intersperse commands other than *Req_RN* between the two successive *Access* commands; if a tag observes non-successive *Access* commands it will return to **arbitrate**, unless the intervening command is a *Query*, in which case the tag will execute the *Query*.

An *Access* is prepended with a frame-sync.

The tag reply to an *Access* command is as shown in Table 4-34. If the *Access* is the first in the sequence, then the tag backscatters its handle to acknowledge that it received the command. If the *Access* is the second in the sequence and the entire received 32-bit access password is correct, then the tag backscatters its handle to acknowledge that it has executed the command successfully and has transitioned to the **secured** state; otherwise the tag does not reply. The reply includes a CRC-16 calculated over the handle.

Table 4-33 Access Command

	Command	Password	RN	CRC-16
# of bits	8	16	16	16
Description	11000110	(½ access password) ⊗ RN16	handle	

Table 4-34 Tag reply to Access Command

	RN	CRC-16
# of bits	16	16
Description	handle	

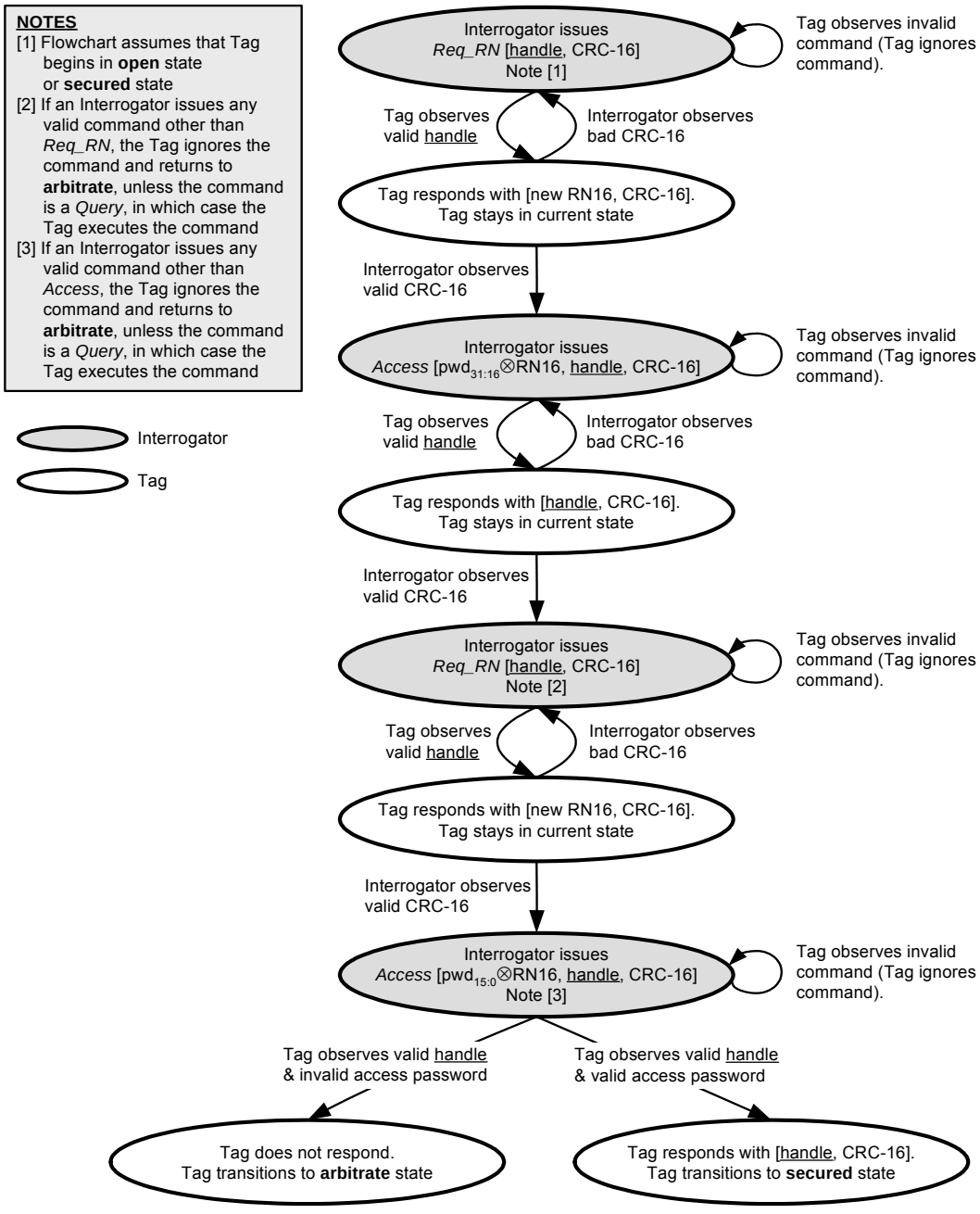


Figure 4-5 Access Procedure

4.4.3.7 BlockWrite Command

BlockWrite is an optional Gen 2 command. Monza™'s specific implementation allows a reader to write a 16-bit word in a tag's Reserved or EPC memory using a single command, and without the cover-coding overhead. *BlockWrite* has the following fields:

- MemBank specifies whether the *BlockWrite* occurs in Reserved or EPC memory. *BlockWrite* commands apply to a single memory row. Successive *BlockWrites* may apply to different rows.
- WordPtr specifies the starting word address for the memory write, where words are 16 bits in length. WordPtr uses EBV formatting.
- WordCount specifies the number of 16-bit words to be written. If WordCount = 00_h the tag will ignore the *BlockWrite*. If WordCount = 01_h the tag will write a single data word. If the word count is greater than 01_h, an error code will be returned.
- Data contains the 16-bit word to be written. Unlike a *Write*, the data in a *BlockWrite* are not cover-coded, and a reader need not issue a *Req_RN* before issuing a *BlockWrite*.

The *BlockWrite* command also includes the tag's handle and a CRC-16. The CRC-16 is calculated over the first command-code bit to the last handle bit.

If a tag receives a *BlockWrite* with a valid CRC-16 but an invalid handle it will ignore the *BlockWrite* and remain in its current state (**open** or **secured**, as appropriate).

A *BlockWrite* shall be prepended with a frame-sync.

After issuing a *BlockWrite* a reader shall transmit CW for the lesser of T_{REPLY} or 20ms, where T_{REPLY} is the time between the reader's *BlockWrite* command and the tag's backscattered reply. A reader may observe several possible outcomes from a *BlockWrite*, depending on the success or failure of the tag's memory-write operation:

- **The *BlockWrite* succeeds:** After completing the *BlockWrite* a tag will backscatter the reply shown in Table 4-36 comprising a header (a 0-bit), the tag's handle, and a CRC-16 calculated over the 0-bit and handle. If the reader observes this reply within the maximum T_{REPLY} (see T_{REPLY} in Table 3-7) then the *BlockWrite* completed successfully.
- **The tag encounters an error:** The tag will backscatter an error code during the CW period rather than the reply shown in Table 4-36 (see 3.6.2 for error-code definitions and for the reply format).
- **The *BlockWrite* does not succeed:** If the reader does not observe a reply within the maximum T_{REPLY} as specified in Table 3-7, then the *BlockWrite* did not complete successfully.

Upon receiving a valid *BlockWrite* command a tag will write the commanded Data into memory. The tag's reply to a *BlockWrite* uses the extended preamble shown in Figure 3-14 or Figure 3-19, as appropriate (i.e., a tag will reply as if TRext=1 regardless of the TRext value in the *Query* that initiated the round).

Table 4-35 BlockWrite Command

	Command	MemBank	WordPtr	WordCount	Data	RN	CRC-16
# of bits	8	2	EBV	8	Variable	16	16
Description	11000111	00: Reserved 01: EPC 10: TID 11: User	Starting address pointer	Number of words to write	Data to be written	handle	

Table 4-36 Tag Reply to BlockWrite Command

	Header	RN	CRC-16
# of bits	1	16	16
Description	0	handle	

4.5 Tag Memory

4.5.1 Memory Map

Figure 4-6 depicts both a physical and logical chip memory map. The memory comprises Reserved, EPC, and TID (which is ROM-based, and not user-writable) memory banks. Monza™ tags do not implement User memory.

MEM BANK #	MEM BANK NAME	MEM BANK BIT ADDRESS	BIT NUMBER															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10_2	TID (ROM)	10_h-$1F_h$	0	0	0	1	MODEL NUMBER											
		00_h-$0F_h$	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
01_2	EPC (NVM)	70_h-$7F_h$	EPC[15:0]															
		60_h-$6F_h$	EPC[31:16]															
		50_h-$5F_h$	EPC[47:32]															
		40_h-$4F_h$	EPC[63:48]															
		30_h-$3F_h$	EPC[79:64]															
		20_h-$2F_h$	EPC[95:80]															
		10_h-$1F_h$	PROTOCOL-CONTROL BITS (PC)															
		00_h-$0F_h$	CRC-16															
00_2	RESERVED (NVM)	30_h-$3F_h$	ACCESS PASSWORD[15:0]															
		20_h-$2F_h$	ACCESS PASSWORD[31:16]															
		10_h-$1F_h$	KILL PASSWORD[15:0]															
		00_h-$0F_h$	KILL PASSWORD[31:16]															

Figure 4-6 Physical / Logical Memory Map

4.5.2 Logical vs. Physical Bit Identification

For purposes of distinguishing most significant from least significant bits, a logical representation is used in this datasheet where MSBs correspond to large bit numbers and LSBs to small bit numbers. For example, Bit 15 is the logical MSB of a memory row in the memory map. Bit 0 is the LSB. A multi-bit word represented by WORD[N:0] is interpreted as MSB first when read from left to right. This convention should not be confused with the physical bit address indicated by the rows and column addresses in the memory map; the physical bit address describes the addressing used to access the memory.

4.5.3 Memory Banks

Described in the following sections are the contents of the NVM and ROM memory, and the parameters for their associated bit settings.

4.5.3.1 Reserved Memory

Reserved Memory contains the *Access* and *Kill* passwords.

4.5.3.2 Passwords

1. Monza™ tags have a 32-bit Access Password and 32-bit Kill Password.
2. The default password for both Kill and Access is $00000000_{h.}$

4.5.3.2.1 Access Password

The Access Password is a 32-bit value stored in Reserved Memory 20_h to $3F_h$ MSB first. The default value is all zeroes. Tags with a non-zero Access Password will require a reader to issue this password before transitioning to the **secured** state. A tag that does not implement an Access Password acts as though it had a zero-valued Access Password that is permanently read/write locked.

4.5.3.2.2 Kill Password

The Kill Password is a 32-bit value stored in Reserve Memory 00_h to $1F_{h,}$ MSB first. The default value is all zeroes. A reader shall use a tag's kill password once to kill the tag and render it silent thereafter. A tag will not execute a kill operation if its Kill Password is all zeroes. A tag that does not implement a Kill Password acts as though it had a zero-valued Kill Password that is permanently read/write locked.

4.5.3.3 Tag Identification (TID) Memory

The ROM-based Tag Identification memory contains Impinj-specific data. The Impinj MDID (Manufacturer Identifier) is 000000000001 (shown in Figure 4-6 as the lighter grey-shaded bits across both TID memory map rows). The Monza™ model number is shown in Figure 4-6 as the darker grey-shaded bits in TID memory row 10_h-$1F_h$. The non-shaded bit locations in TID row 00_h-$0F_h$ store the EPCglobal™ Class ID (0xE2).

4.5.3.4 EPC Memory

EPC memory contains the 16 protocol-control bits (PC) at memory addresses 10_h to $1F_h$, a CRC16 at memory addresses 00_h to $0F_h$, and an EPC value beginning at address 20_h. A reader accesses EPC memory by setting MemBank = 01_2 in the appropriate command, and providing a memory address using the extensible-bit-vector (EBV) format. The PC, CRC16, and EPC are stored MSB first (i.e., the EPC's MSB is stored in location 20_h).

The EPC written at time of manufacture is as follows:

Impinj Part Number	96-Bit EPC Value Preprogrammed at Manufacture (hex)
IPJ_W_R_(A or C)	300833b2ddd9014000000000

4.5.3.5 EPC CRC

The calculation of the EPC CRC is performed internally by the tag, based on the contents of the EPC memory, thus removing this burden from the user. The CRC generation and decoding information provided in the following sections are for reference purposes.

4.5.3.6 CRC Generation and Decoding

4.5.3.6.1 CRC-16

A CRC-16 is a cyclic-redundancy check that a reader uses when protecting certain forward link commands, and a tag uses when protecting certain backscattered sequences. To generate a CRC-16 the reader or tag first generates the

CRC-16 precursor shown in Table 4-37, then takes the ones-complement of the generated precursor to form the CRC-16.

To encode a CRC-16, first preload the entire CRC register (i.e., C[15:0]) with FFFFh, then clock the data bits to be encoded into the input labeled DATA, MSB first. After clocking in all the data bits, C[15:0] holds the ones-complement of the CRC-16 value.

To decode a CRC-16, first preload the entire CRC register (C[15:0]) with FFFFh, then clock the received data and CRC-16 {data, CRC-16} bits into the input labeled DATA, MSB first. The CRC-16 check passes if C[15:0] = D0Fh.

Table 4-37 CRC-16 Precurser

CRC Type	Length	Polynomial	Preset	Residue
ISO/IEC 13239	16 bits	$x^{16} + x^{12} + x^5 + 1$	$FFFF_h$	$1D0F_h$

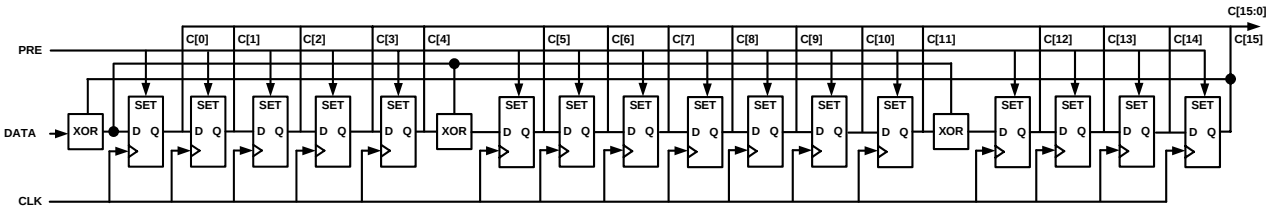


Figure 4-7: CRC-16 Encoder/Decoder

4.5.3.6.2 CRC-5

CRC-5 is a cyclic redundancy check with shift register length 5 as shown in Figure 4-8. To encode a CRC-5, first preload the entire CRC register (i.e., C[4:0]) with 01001_2, then clock the data bits to be encoded into the input labeled DATA, MSB first. After clocking in all the data bits, C[4:0] holds the CRC-5 value.

To decode a CRC-5, first preload the entire CRC register (C[4:0]) with 01001_2, then clock the received data and CRC-5 {data, CRC-5} bits into the input labeled DATA, MSB first. The CRC-5 check passes if C[4:0] = 00000_2.

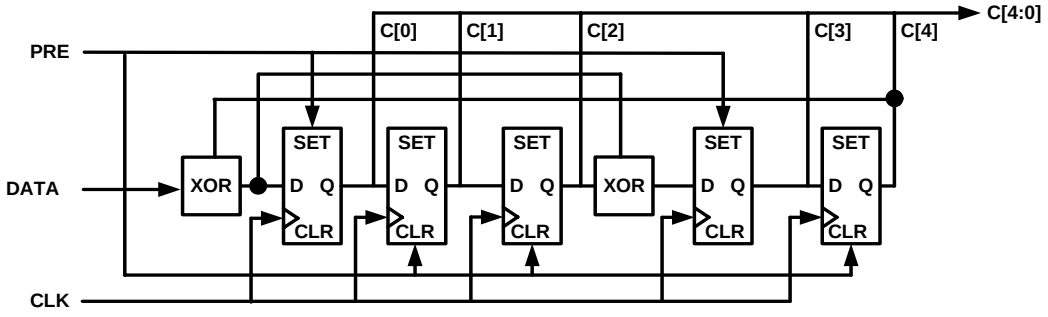


Figure 4-8: CRC-5 Encoder/Decoder

4.6 ETSI Tag Spectral Mask

Figure 4-9 shows the ETSI spectral mask required of a backscattering tag (per EN 302 208-1). For complete regulatory requirements, and specific methods of measurement, refer to ETSI EN 302 208.

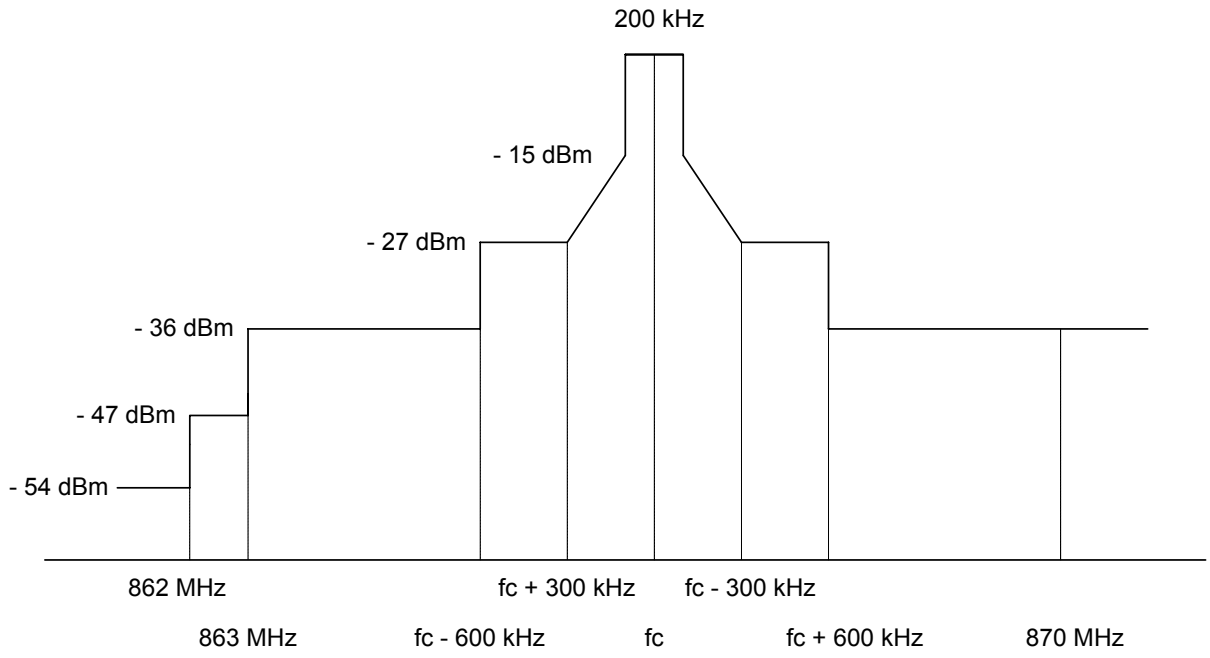


Figure 4-9 EN 302 208-1 Backscatter Spectral Mask

5 Absolute Maximum Ratings

Stresses beyond those listed in this section may cause permanent damage to the tag. These are stress ratings only. Functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this datasheet is not guaranteed or implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

5.1 Temperature

Several different temperature ranges will apply over unique operating and survival conditions. Table 5-1 lists the ranges that will be referred to in this specification. Tag functional and performance requirements are met over the operating range, unless otherwise specified.

Table 5-1 Temperature parameters

Parameter	Minimum	Typical	Maximum	Units	Comments
Extended Operating Temperature	-40		+65	ºC	Default range for all functional and performance requirements
Storage Temperature	-40		+85	ºC	
Assembly Survival Temperature			+150	ºC	Applied for one minute, max
Temperature Rate of Change			4	ºC / sec	During operation

5.2 Input Damage Levels

The tag is guaranteed to survive the inputs specified in Table 5-2.

Table 5-2 ESD and input limits

Parameter	Minimum	Typical	Maximum	Units	Comments
ESD			1,000	V	HBM (Human Body Model)
Reader antenna power with tag sitting on antenna			36[1]	dBm	Tag has 10 cm half-wave dipole antenna
DC input voltage			± 3.5	volts	Applied across two pads
DC input current			± 0.5	mA	Into any input pad

Note 1. Assumes tag has half-wave dipole antenna. While maximum radiated reader power is +36 dBm for both Read and Write operations, the maximum power the tag should receive is +20 dBm (see Table 3-1).

5.3 NVM Use Model

Tag memory will retain data for 10 years if <= 1k cycles; 1 year if 1k cycles to 100k cycles. The cycling limit is reached when any bit in the memory has been written the indicated number of times.

6 Ordering Information

Part Number	Form	Product	Processing Flow
IPJ_W_R_A	Wafer	Monza™	Raw: non-bumped, non-thinned
IPJ_W_R_C	Wafer	Monza™	Bumped, thinned (to 6 mils, or ~150 µm), and sawn

Notices:

www.impinj.com