Project Name:

BTAG – Brazil Tag for SINIAV AVI System

Prototype Definitions Requirements Document

Department: Intel Labs Seattle – BTAG Team

National System for Automatic Vehicle Identification

Prepared By

Document Owner(s)	Projec/Organization Role
Justin Reina	BTAG / Firmware Engineer
Alanson Sample	BTAG / Hardware Engineer
Josh Smith	BTAG / Project Architect
Terrance (Terry) O'Shea	BTAG / Project Manager

Version Control

Version	Date	Author	Change Description
0	3/2/2011	Justin Reina	Document created
1	3/7/2011	Justin Reina	Added Memory Section
2	3/23/2011	Justin Reina	Added Tag Responses
3	4/6/2011	Justin Reina	Added Cmd/Rsp Definitions

Table of Contents

1 Purpose

BTAG is short for Brazil-Tag, and is a battery-assisted RFID Tag for use by the government of Brazil for Automatic Vehicle Identification (AVI). The tag supports a limited set of the EPC C_1G_2 RFID protocol, and the full SINIAV Protocol which is a superset to EPC C_1G_2.

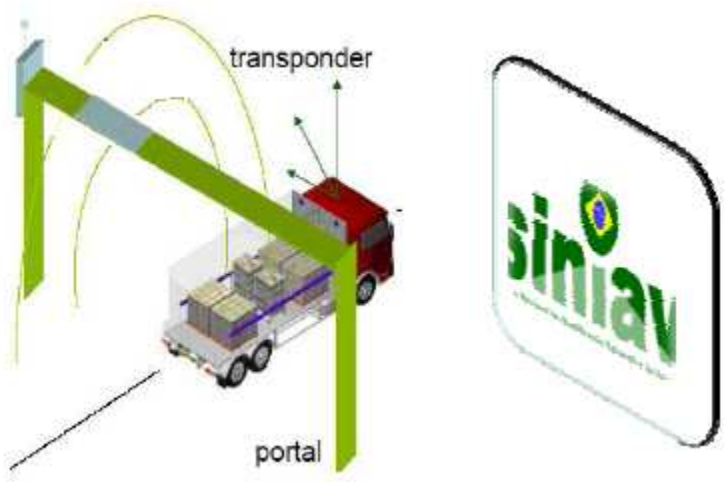


Figure 1: *Depiction of Siniav Tag(OBU) interrogation by Siniav Reader (RSU) along a roadway at uncontrolled speeds.*

The design is optimized for cost. The primary design components discussed in this document are the encasing, electronics, battery and firmware. This document describes and defines the BTAG prototype.

1.1 Future Extensions

A key asset to the BTAG platform is its high level of security and cryptographic protection. This asset will allow extension of the BTAG platform described in this document to other potential "High-Security RF Identification" scenarios.

Figure 2:*Example Extension of BTAG platform: Asset Monitoring chemical drums.*

1.2 BTAG Project-Concept Gathering

The authors also use the Appendix of this document for note-taking during the development of the BTAG project. Such notes include project to-dos, upcoming events (e.g. tests, meetings), any sort of hazard or project risk, etc. The decision to keep these notes here consolidates such information.

2 References

Justin maintains all of the non-highlighted documents listed below. The reference numbering here matches that of siniavDocListing.xlsx, thus the numbering is not sequential. Documents from siniavDocLisiting.xlsx not referenced in this document are not listed below. The numbering is current as of siniavDocListing0_0.

[1] First Report – SINIAV Technology Specifications (SINIAV PROTOCOL BY AUTOFIND)

[2] Equipment Approval Requirements - AVI Physical Layer

[4] Technology Specifications SINIAV (*Especificação da Tecnologia SINIAV: Requisitos da Camada Física)*

[5] Application Report – SLAA221 – CRC Implementation with MSP430. Texas Instruments, November 2004.

[?] Wernher von Braun Advanced Research Center with National Department of Transit collaboration (DENATRAN): SINIAV Technology Specification – Lifecycle of the electronic board, October 2008.

*This document will/should specify battery/lifetime requirements.

3 Definitions

BTAG	Brazil-Tag. Name of Intel's RFID tag for the Siniav Project.
SINIAV	Government program for car tracking under the DENATRAN initiative.
EPC C1G2	EPC Class 1 Generation 2 RFID Protocol. Siniav G1 is a superset of this.
OBU	Onboard Unit, i.e. the Siniav RFID Tag
RSU	Road-Side Unit, i.e. the Siniav RFID Reader
Portal	Combination of RSU/antennas/computers setup up along roadways to read OBUs.
MCU	Micro-controller unit. The BTag uses the TI MSP430F2272
ICT	In-Circuit Test. Performed on BTag's after manufacture using pogo-pins for solderless stimulation/measurement of critical unit verification parameters.
CSP	

Particularly Ambiguous/Loosely Defined Definitions

The protocol [1] tosses around these words seemingly interchangeably, although each should have a very specific meaning. Below is the current state of understanding for these terms.

Manufacture	The act of loading the device firmware and powering up for the first time. Significant amounts of the tag memory have already been loaded onto the device in this stage by means of direct embedment into the binary.
Initialization	An optional step taken at the manufacturer where additional tag memory is written to, over-the-air using the RFID protocol.
Customization	After Initialization, customization occurs at the licensing authority. This is where security keys and vehicle data are recorded.

Personalization	As far as we can tell, personalization is just a different name for customization, or perhaps customization is a sub-state of personalization. The BTag merges customization and personalization as the same mode.
Commisioning	

4 Ownership and Statement of Work

4.1 Project Team Organization Plans

Name	Role	Responsibilities	Phone/E-mail
Marc Alexander			503.329.9628 marc.a.alexander@intel.com
Sean Halton			480.552.0524 sean.m.halton@intel.com
Terrance (Terry) O'Shea			503.456.5034 terrance.j.o'shea@intel.com
Josh Smith			206.214.6177 jrs@cs.washington.edu
Justin Reina	SW/Firmware Engineering		425.760.7291 justinx.m.reina@intel.com
Alanson Sample	HW Engineering		206.465.5426 alanson.p.sample@intel.com
Marco Silva	Internal SINIAV/Brazil Consulant		marcos.silva@intel.com
Max Leite	Internal SINIAV/Brazil Consulant		5511.3365.5514 max.leite@intel.com
Pauline (Polly) Powledge			206.545.2522 pauline.s.powledge@intel.com
Ken Tallo			916.356.3462 ken.tallo@intel.com

4.2 Participating External Members

Company	Role	Relationship	Phone/E-mail
VBC	SINIAV protocol consultant	Develop all SINIAV protocol and initial prototypes. Oversees development of certification/ regulatory testing	
Autofind	Enclosure Design, Contract Manufacturing?	Current Mfg/SysIntegrator for BTAG. Also develops housing for SINIAV ECS	
Sirit / Federal Signal	SINIAV Reader Developer	Only current company to develop a SINIAV G0 firmware for reader	
Denatran	Governing Body of SINIAV		
Brazilian Ministry of Cities	Governing Body of Denatran		
Seagull			

5 General Usage

Feature	Purpose	Component
Reconfigurable Logic and Control	Adapt BTAG to evolving SINIAV standards and future applications	MSP430F2272. 32kB FLASH, 1kB RAM. Ultra-Low Power 16-bit MCU.
Secure, Non-Volatile Memory	Store cryptographically secure data on device	FLASH embedded on MCU.
RFID Radio	EPC C1G2 communication at max data rate	Low-cost, minimum component count front end. Single stage rectifier with VBW comparator for demod. Transmit is through single backscatter FET.
Antenna	Same	Meeting SINIAV requirements for physical layer performance
Low-Power Wakeup Circuitry	Disable RFID radio until RF signal present	Single ultra-low power comparator in parallel with receive comparator.
Enclosure	Device housing	Holds battery, facilitates affixment, provides tamper-proofing mechanisms.
Battery	Long-life, burst operation	Meets the SINIAV power requirements (tba?).
Tamper-Proofing	Resistant to modification or hacking	Single set of metal contacts embedded into casing. Electrical connection is "break-on-contact."
Cryptographic Modules	Low overhead encryption modules based on AES-128 block encryption.	Firmware modules implementing AES in CTR and ECB modes.
Random-Number Generator	Pseudo-random number generation using AES-128 and analog voltage measurement	Firmware module implements NIST 800-38B compliant RNG. Uses analog voltage from radio signal path to seed random number generation.
Side-Channel Attack Mitigation	Prevent the comprimise of secure, stored data	tbd

6 Requirements

Requirement	Governing Body	Description
RFID Downlink	Denatran?	PIE Encoding, Tari = 6.25us Fixed
RFID Uplink	Denatran?	FM0 Modulation, Link Frequency of 640kHz Fixed
SINIAV Protocol Compliance	should it be listed here?	
SINIAV Physical Layer Compliance	tbd	
Lifetime	tbd	
Power-Profile	tbd	e.g. how many reads/sec over lifetime?
form factor	SATO?	fits in SATO existing case?

6.1 Link Timing

The Siniav Link Timing is somewhat scattered and ambiguous. Here is the state of our understanding. As shown below there are two sets of timing definitions given for G1, and none for G0.

- Tari_G0 = 6.25us
- LF_G0 = 320 kHz
- FT_G0 = ±10% (guess)
- RTCAL_G0 = 31.25us (guess)

- Tari_G1 = 6.25us
- LF_G1 = 6.25us
- FT_G1 = ±15%
- RTCAL_G1 = 15.625us

Standard EPC Definitions

The following information is extracted for the EPC C1G2 protocol document for clarification.

- *FT_640 = ±15%*
- *FT_320 = ±10%*

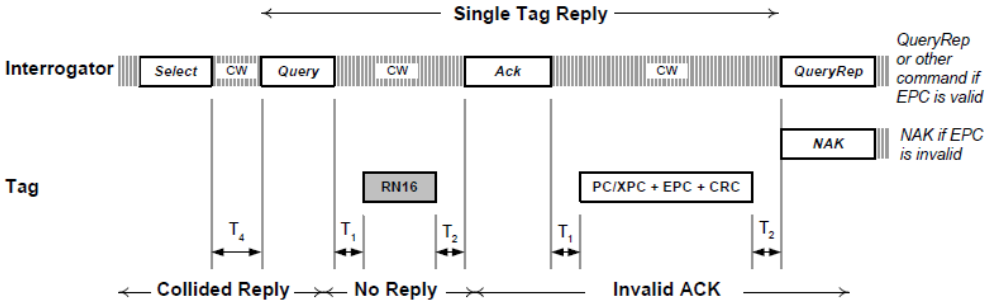


Figure 3: *Illustration from EPC C1G2 spec of the link timing.*

Table 6.13 – Link timing parameters

Parameter	Minimum	Nominal	Maximum	Description
T_1	MAX(RTcal, $10T_{pri}$) × (1 – |FT|) – 2µs	MAX(RTcal, $10T_{pri}$)	MAX(RTcal, $10T_{pri}$) × (1 + |FT|) + 2µs	Time from Interrogator transmission to Tag response (specifically, the time from the last rising edge of the last bit of the Interrogator transmission to the first rising edge of the Tag response), measured at the Tag's antenna terminals
T_2	$3.0T_{pri}$		$20.0T_{pri}$	Interrogator response time required if a Tag is to demodulate the Interrogator signal, measured from the end of the last (dummy) bit of the Tag response to the first falling edge of the Interrogator transmission
T_3	$0.0T_{pri}$			Time an Interrogator waits, after T_1, before it issues another command
T_4	2.0 RTcal			Minimum time between Interrogator commands

Figure 4: *Definitions of the link timing for EPC C1G2*

This results in the following calculations for 640kHz and 320kHz (eqns embedded in xls table):

Table 1: *These are the EPC C1G2 timing values for 640kHz/320kHz with Tari=6.25us*

Parameter	640 Min	640 Nom	640 Max	320 Min	320 Nom	320 Max
T1	11.281	15.625	19.969	26.125	31.250	36.375
T2	4.688		31.250	9.375		62.500
T3	0.000			0.000		
T4	31.250			62.500		

See the following sections for imposed Siniav timing values, which are not comprehensive.

6.1.1 Siniav Technology Specification – Physical Layer Requirements May 2009. Intel ID #14

Table 2: *This is with reference to 640kHz/6.25us, so G1. These values are also reflected in document Intel ID #13, (a presentation).*

Parameter	Minimum	Nominal	Maximum
T1	11.29us	11.29us	15.28us
T2	4.69us		31.25us
T3	24us		1100us
T4	31.25us		

6.1.2 Equipment Approval Requirements – AVI Physical Layer 2008. Intel ID #2

Table 3: *This is with reference to 640kHz/6.25us, so G1.*

Parameter	Minimum	Nominal	Maximum
T2	17.19	17.19	20.02

6.1.3 Selected BTag Timing Values

Because of the Siniav ambiguity and conflicts, the BTag chooses to use the standard EPC values, with the additional restriction that T1 not be greater than it's specified nominal value (e.g. 15.625us for G1).

7 System Diagram

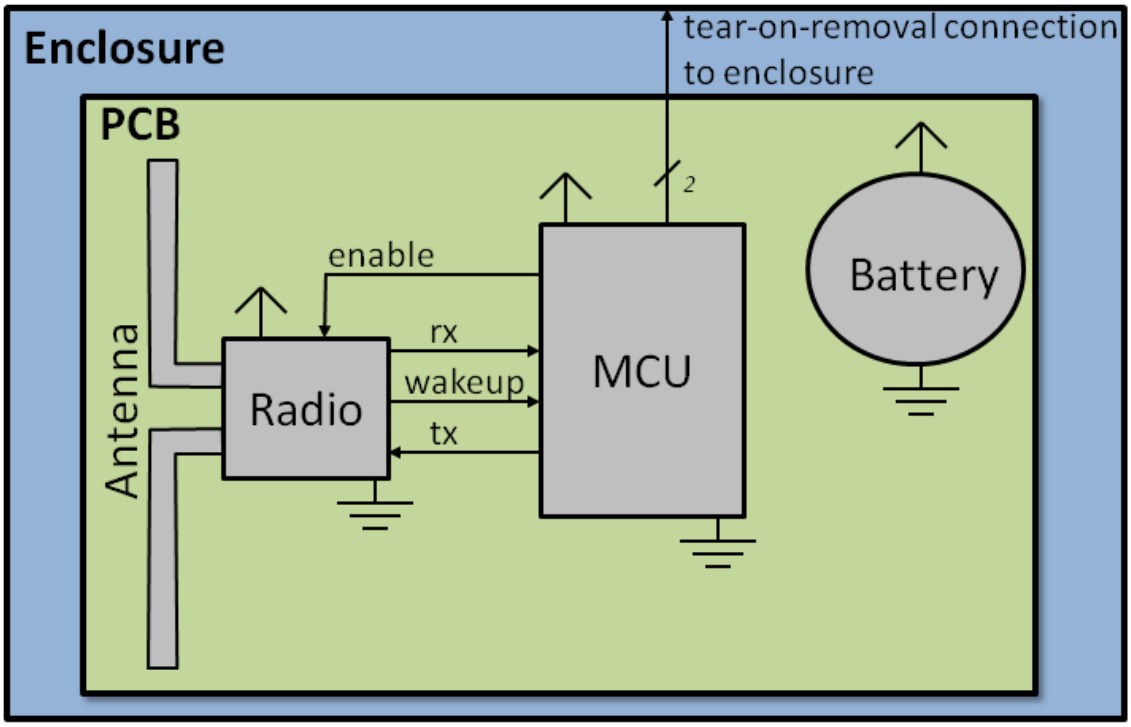


Figure 5: *BTAG System Diagram*

8 Key Design Challenges

- Range
- Power Consumption
- Form Factor
- Operation of Rx/Tx RFID on MCU at 2.7V (speed)
- Complexity firmware
- Insitu Performance
- Specification Compliance
- ?

9 Enclosure Description

9.1 Requirements

The enclosure specification is currently unclear. A rough form factor of 3”x5” is suggested, with as low of a profile as possible (given battery sizing). Design should be optimized for cost.

9.2 Design Selection

There is no current selection for enclosure. A proposed enclosure from AUTOFIND leverages a previous tamper-prevention design that they have already generated molds for. This design is the current reference. It is currently unclear if the BTAG will follow through with this enclosure, as other options are still on the table with regard to enclosure manufacturer.

Figure 6: *Internal View of AUTOFIND Enclosure. 9V Battery is included for sizing reference and is not a part of BTAG system.*

Figure 7: *External view of the AUTOFIND Enclosure. The green area is an adhesive which permanently affixed to the mounting surface. When the enclosure is removed this provides tear-on-removal for tamper proofing.*

9.3 Tamper-Proof Feature

This casing features two electrical contacts connected to the (green) mounting adhesive. The contacts make direct contact (by pressure, no permanent connection) to the PCB. This connection is broken by design if the casing is removed from the mounting surface, breaking the electrical contact as seen by the PCB.

Figure 8: *AUTOFIND Tamper-Proofing Contact.*

This is the method of tamper-proofing provided by this casing. To note, it does not mitigate any other sort of physical tampering other than tag removal from the mounting surface.

9.4 Case Battery Capacity

Although there is not enough information to estimate required battery capacity for the BTAG, this enclosure does not seem to fit sufficient battery capacity, due to the oval removal piece’s obstruction of the main chamber.

10 PCB Description

11 Battery Description

There is no current selected battery design for the BTag. The team estimates a Lithium-Manganese solution may be possible.

11.1 Calculated Requirements

The BTag has two primary modes, waiting for a command (sleep) and acting on a command (active). The current SINIAV requirements dictating the transition and rates of the these modes is unclear. Shown below are the teams current calculations for the BTag power consumption.

Estimated Required Battery Capacity (2.7V, 20℃)
912 mAh (see table below for calculation)

Power Calculation
The following table is the current battery calculation for the BTAG. It was appended as this table out of convenience. If a more presentable format is required, please contact Justin Reina. All calculations are referenced by cell within the table.

Table 4: *BTAG Power Requirements Calculation*

#1. Vals Referenced

EPC T1 Measured	1.80E-05	
EPC T2 Timeout (Std)	1.80E-05	todo get hard num?
EPC T2 Timeout (msrd)		
RSU Preamble Length	6.92E-05	
RSUE Framesync	3.88E-05	
640kHz FM0 Bit Period	1.5625E-06	
Tari=6.25us approx Bit Period	1.09375E-05	
FM0 Preamble Length (no preamble)	0.0000875	

#2. EPC/SINIAV Transaction

Command	#RSU_Bits	T1	#OBU_Bits	T2	Time Ttl	
QUERY	22	1.80E-05	16	1.80E-05	4.58E-04	
QUERYREP	4	1.80E-05	16	1.80E-05	2.31E-04	
QUERYREP	4	1.80E-05	16	1.80E-05	1.94E-04	
QUERYREP	4	1.80E-05	16	1.80E-05	2.03E-04	
QUERYREP	4	1.80E-05	16	1.80E-05	2.80E-04	
QUERYADJ	22	1.80E-05	16	1.80E-05	3.89E-04	
ACK	18	1.80E-05	128	1.80E-05	5.20E-04	128 bits
Req_Handle	41	1.80E-05	34	1.80E-05	6.25E-04	
Mutual_Auth	167	1.80E-05	35	0.02	2.20E-02	20ms for crytpo processing
Finalize	33	1.80E-05	545	1.80E-05	1.34E-03	

26.22 total [ms]

#3. Other On-Time

Self-Testing	30 *100/day?	[ms]
Initiazliation	3000 once?	[ms]

#3. Spurious Activation Time

1440	Spurious On-Times Per Day	
10	Estimated On Time Per Spurious	[us]
14.4	spurious on-time per day	[ms]

#4. Worst Case Battery Life

300	rds/day
5.5E+05	rds/5yrs
2.0E+04	OnTime in 5 years[sec]
26.28	Spurious in 5 years [sec]
4.60	onCurrent[mA]
25.3	mAh
1.6E+08	OffTime in 5years[hr]
0.55	OffCurrent[uA]
0.00055	OffCurrent[mA]
24.1	mAh

#4. Required Capacity

Total Battery Capacity at 20C		
49.43	mAh	

11.2 Measured Results

todo: use monitor board to get a screen shot of power consumption in active, sleep modes.

12 MCU Description

The Texas Instruments MSP430F2272 processor in a 20-QFN package is implemented in the BTag. This MCU provides an excellent balance between low-power / low-cost optimizations. The internal calibrated DCO also helps reduce part count on the BTag. Unfortunately the MSP430F2xxx series do not contain a continuous TX USART, which increases the required CPU speed for operation (see 17.1: CPU Speed).

12.1 Development Environment

Code Composer Studio 4.2.1 was used for the development of this project in conjunction with MSP430 Compiler 3.2.3 (cl430.exe). The project was configured with a 128byte stack, 0 byte heap, --opt_level="", --opt_for_speed="". All other build options were left default, except for the memory map, which was segmented in the lnk_msp430f2272.cmd file, reflecting the map of Table 5.

It may be an interesting exercise to go back and change the optimization levels, but is not necessary.

13 Radio Description

14 Antenna Description

15 Tamper-Proofing Description

15.1 Casing Mechanical Connection

A two-ended metal connection affixed to the PCB enclosure makes contact with the PCB when case is closed. When case is opened the metal connection is removed. MCU can detect removal of connection as an open-circuit at a GPIO pin (P2.4), as one end of metal connection is connected to V_{DD}, and the other to P2.4.

15.2 Glob-Topping

Section to be described.

15.3 JTAG Interface Disable

The JTAG programming interface to the MSP430F2272 is achieved through the TEST interface TEST,~RST,TDI/TDO,TDO.TCK and TMK pins. This interface is disabled by blowing an access fuse, *JTAG_FUSE*.

Fuse Description (From SLAA149, p.14)
Before the MSP430F grants JTAG access, it first checks the state of an internal fuse, *JTAG_FUSE*. The fuse is not located on any of the JTAG lines, rather it is auxiliary to the lines and is maintained by the MSP430F JTAG controller. There is no direct access to the fuse. On JTAG requests to the MSP430F, its JTAG controller always checks the fuse state first before granting JTAG access.

If the fuse has been blown, the JTAG controller will no longer respond to JTAG requests, instead simply passing bits seen on TDI out on TDO with a 1-cycle (TCK) delay. This is called BYPASS mode. This functionality is illustrated in Figure 9 below.

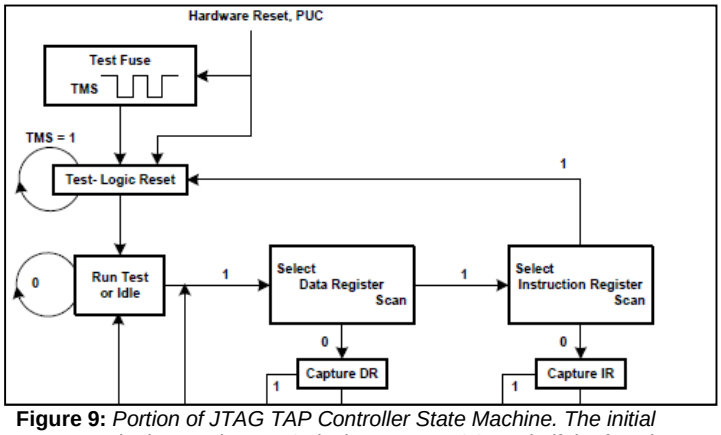


Figure 9: *Portion of JTAG TAP Controller State Machine. The initial 'Test Fuse' logic sets the JTAG device to BYPASS mode if the fuse is blown. Taken From Figure 7 of [5].*

Fuse-Blow Procedure
Fuse Blowing is achieved via a JTAG command, so the entire JTAG port connection is required to blow the fuse. Section 4 of [5] describes the command process. Specifically, VPP (i.e. 6.5V) is applied to TDI during a JTAG routine to blow the fuse. Current drawn on TDI during the fuse blowing procedure can reach 100mA (Section A.5 of [5]). The JTAG procedure is well described in [5].

15.4 Bootstrap-Loader Disable (BSL)

The MSP memory can also be accessed for read or write through its bootstrap loader. Similar to the JTAG checking it's fuse before operation, the BSL checks a byte in FLASH before operating. If the byte's value is equal to 0xAA55, the BSL will not respond to any BSL requests, which disables it.

JTAG/BSL Disable Procedure

1. Write via the JTAG interface '0xAA55h' to memory address 0xFFDEh (also called interrupt vector 15, or *BSL_KEY).*
2. Execute the JTAG function *BlowFuse* as described in [5].
3. Execute the JTAG function *isFuseBlown* as described in [5].
   a. Expected response here is just echoing back the command, as it would be in BYPASS mode.
4. Attempt the BSL command *TransmitBSLVersion.* Device is BSL-Disabled if there is no response.

15.5 PhotoTransistor Detect

One or more phototransistors are implemented in a common-collector transimpedance configuration. When light is incident on the base of the photo-transistor, current flows and Vout increases (Vout is proportional to incident light). If the gain provided by the Resistor is sufficiently high, Vout become a logic-level signal for the MCU and we can merge this tamper-detecting with the mechanical tamper-proofing into the same P2_ISR. When the glob-top is on and incident power is minimum, this circuit draws only the leakage current of the phototransistor, which is <1nA typ.

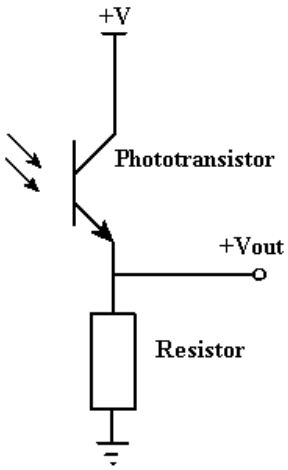


Figure 10: *Photo-detection Option.*

15.6 Unique, Secure Device ID for HW, SW and Casing

It has been identified to incorporate serial numbers to each PCB, Casing and Firmware.

Case ID

A serial number is to be stamped internally on the bottom half of the casing. (note: how do we actually stamp these in a low-cost way?)

Board ID

A serial number to be stamped on the board (note: how do we actually stamp these in a low-cost way?)

MCU ID

The MSP430F2272 has no mechanism for storing a permanent, unique ID. However consider that the TID memory bank contains a Manufacturer Model (e.g Intel BTag 0.5), and a unique 64-bit tag model number (e.g. tag #143563466). These bits would be determined at the time of manufacturing, and loaded with the firmware during initial programming. Each manufactured tag would have a unique value that the Siniav protocol prevents write access to.

15.7 Identified Attacks

The following *attackers* are identified as needing protection against:

1. Customer Misuse of Tag – Customer removes tag from one vehicle and places into another, 'cheating' the system.
2. Siniav System Attacks – Attempts to recover cryptographic keys and unique data. Attempts to generate duplicate devices for production or illegal activities (e.g. hiding).
3. Competitor Attacks – Attempts to recover binary firmware of device. Attempts to reprogram device.
4. …

From these scenarios, the following *attacks* are then identified as needing protection against.

1. Physical Device Removal – The customer removes the device and places into another vehicle.
2. FLASH Memory Access – The attacker gains either read or write access to memory.
3. Side-Channel Attack – The attacker uses direct electrical connections to PCB and/or MCU, in addition to near-field measurements of EMI to statistically brute-force recover cryptographic keys used in the Siniav system.
4. Other forms of Crypto-Analysis for key/data recovery.
5. …

15.8 Analysis of Identified Attacks

Here is the analysis for each type of attack.

1. Physical Device Removal - Device removal is severely mitigated by including the mechanical tamper-proofing. This does not prevent removal, as the case's adhesive mounting can just be razor-bladed off with no compromise to the mechanical tamper-proof connection. This should enough be to *discourage* removal.

   It is easy to envision more robust methods than the current o-ring method of the SATO case, but it appears the o-ring method is sufficiently secure.

2. FLASH Memory Access **-** Only two methods are provided in the MSP hardware to access flash, the JTAG interface and the BSL interface. By first disabling the BSL, and then blowing the JTAG fuse both interfaces are disabled for read/write access.

   To date, we have not found any literature that suggests either interface can be compromised if the BSL is disabled and fuse blown. It has been hypothesized that someone *theoretically*
3. f

15.9 Listing of Fatal Errors

There are several states or actions which will cause the BTag to permanently disable itself and enter a permanent 'Killed Mode'. The most obvious would be activation of a tamper-proofing circuit. Here is a list of defined 'fatal errors' which will transition the tag permanently to killed mode.

Tampering or HW Malfunction

- Mechanical Tampering Activation
- Optical Tampering Activation
- Device Power Reset (only after it has been switched to Siniav Mode)
- *Over-Temperature Event
- *Over-Voltage Event

- Failure of Receive Comparator

Firmware Corruption, Attack or Failure

- Device Programming Interface is Enabled (BSL)
- Failure of Cryptographic Module(s)
- Stack Overflow Detection
- RNG Failure
- Firmware Integrity Failure

16 Power Consumption Derivation

17 Required Software Resources

The following resource identification is based on an MCU platform. FPGA/etc platforms will have - different resources and requirements.

17.1 CPU Speed

There are two timing constraints that set the BTag's required CPU speed:

1. Chip Timing of Transmit (640kHz) 812.5ns per EPC C_1G_2
2. Response Time after RSU Interrogation (20.02us) per Siniav Physical Layer Doc

For the MSP430F2272, #1 is the primary constraint as it does not have a sufficient usart for radio chip output. As described in the transmit section, 9 cycles are required for the 812.5ns chip, setting the min CPU frequency for the BTag at 812.5ns/9 or 11.077 MHz. The BTag is set to 12MHz per the TI factory calibration bytes.

17.2 Power Consumption

17.3 Onboard Peripherals

17.4 Memory Requirements and Mapping

FLASH Required: 22kB estimated
RAM Required: 768B estimated

Todo: List Default Specification

Current Specification

start	end	Size	type	origin	size
65504	65536	33	INT/RESET	0000FFE0	00000021
65502	65503	2	EMPTY	0000FFDE	00000002
35328	65501	30174	OPEN FLASH	00008A00	000075DE
34816	35327	512	Btag Non-Volatile (TBDish)	00008800	00000200
34304	34815	512	memBank_User[]	00008600	00000200
33792	34303	512	memBank_TID[]	00008400	00000200
33280	33791	512	memBank_UII[]	00008200	00000200
32768	33279	512	memBank_Reserved[]	00008000	00000200
4352	32767	28416	EMPTY	00001100	00006F00
4096	4351	256	INFO	00001000	00000100
1536	4095	2560	EMPTY	00000600	00000A00
512	1535	1024	RAM	00000200	00000400
16	511	496	PERIPHERALS(8,16)	00000010	000001F0
0	15	16	SFR	00000000	00000010

Table 5: *MSP430 Memory Map*

Todo: List out my memory estimations here

18 Software Architecture

18.1 Device Modes

The BTag operates in four primary modes: Initialization, Customization, SINIAV and killed. A brief description follows for each.

In-Circuit Test: Tag executes in-circuit test procedure as specified.

Initialization: Tag executes EPC C1G2 subset used to verify units during functional testing at the manufacturer.

Personalization: Tag responds to EPC commands. Licensing Authority loads BTag with CSP data (keys, vehicle identification) and then switches BTag into SINIAV mode.

SINIAV: Responds to a subset of EPC commands, and all of SINIAV commands. Performs authenticated SINIAV authentication, reading and writing. Participates in SINIAV inventory.

Killed: If the tags security is compromised, or some part of initialization or customization fails, the tag zeroizes all CSP and then enters a killed state. In killed the tag will identify itself with a specified KILLED_EPC using the EPC C1G2 interface.

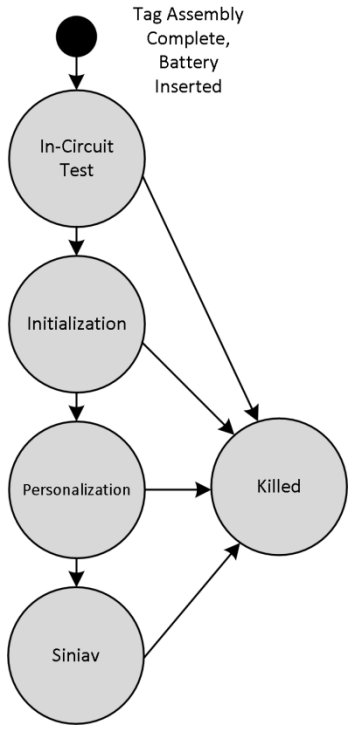


Figure 11: *BTAG Modes of Operation*

18.2 System State Diagram

Each mode contains a somewhat large number of states which adhere to the modes model (e.g. a tag can't go from personalization back to customization). Below is a summary of these states, organized by mode.

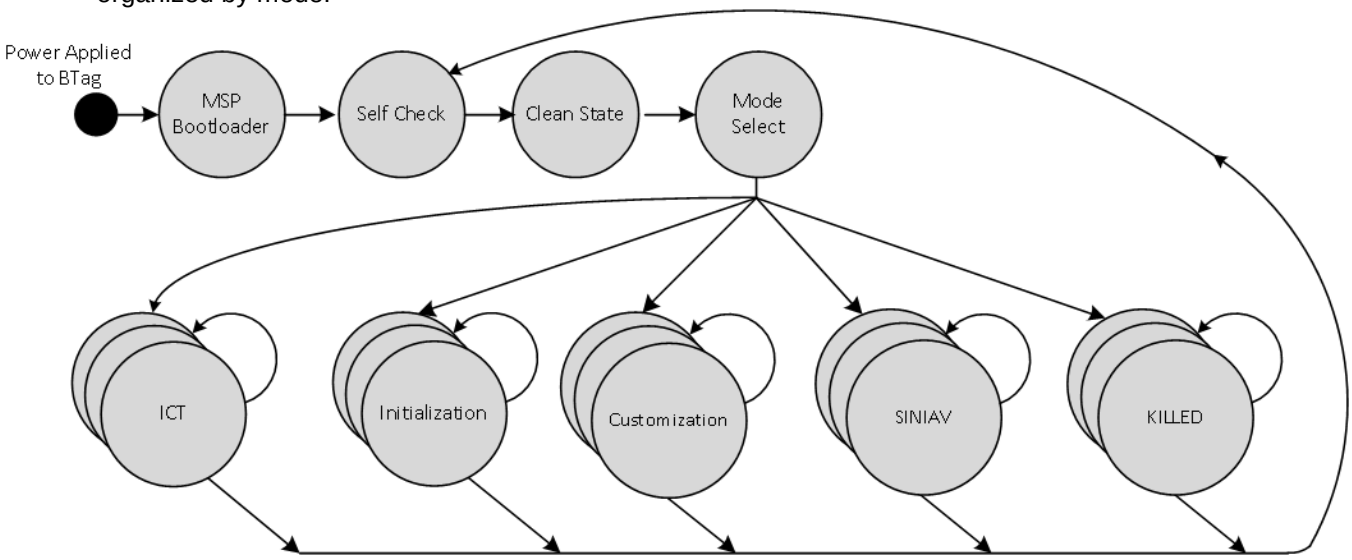


Figure 12: *BTAG State Summary*

18.3 SINIAV States

The following states of operation are defined by the SINIAV protocol, and are included here for reference. They are a superset on top of EPC C1G2.

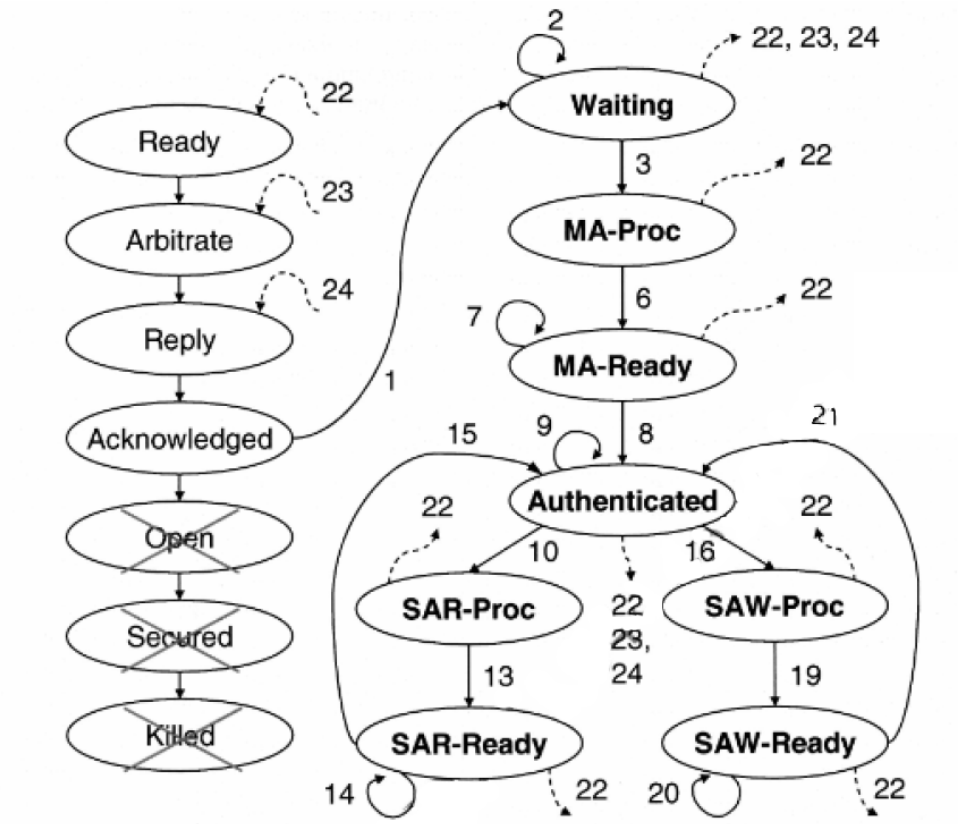
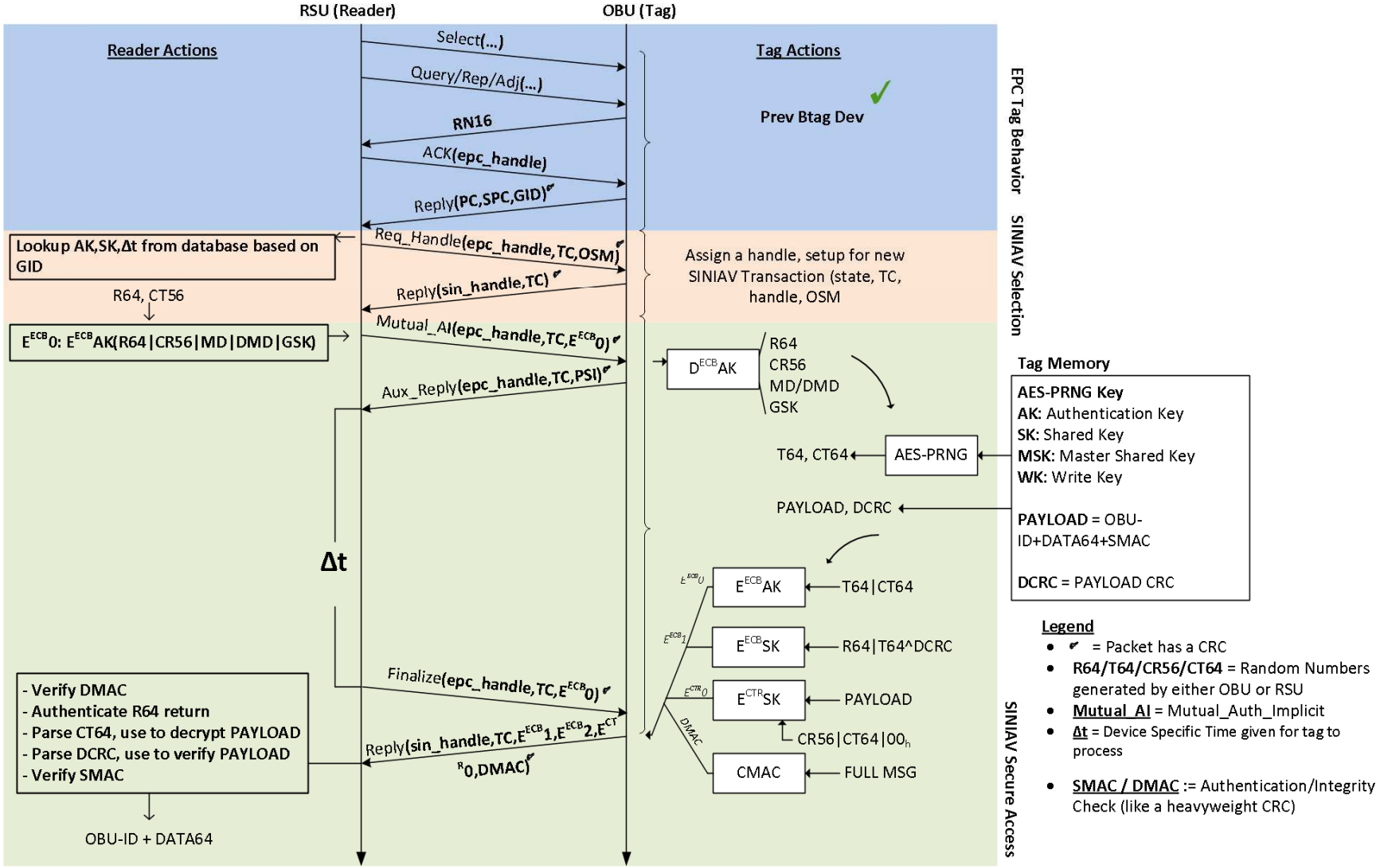


Figure 13: *SINIAV State Diagram as defined in the SINIAV Protocol*

18.4 SINIAV Sequence



18.5 BTag States (Complete)

Below is a listing of all BTag states, organized by mode.

Table 6: *BTAG States, organized by mode.*

Mode	State	Designator	Description
Boot	MSP BootLoader	BOOT_STATE_ON	standard MSP bootloader code
Boot	Self-Check	BOOT_STATE_TEST	Self Test to verify integrity
Boot	Clean-State	BOOT_STATE_CLEAN	Reset RAM/periphs to known state
Boot	Mode Select	BOOT_STATE_SEL	Decide Which Mode to Jump to
Receive	Delay	RX_STATE_DELAY	us.
Receive	DelimSM	RX_STATE_DELIM	wait for EPC delimiter in RX_Chain
Receive	ReceiveSM	RX_STATE_BITS	Process EPC cmd in RX_Chain
ICT	tbd	ICT_STATE_1	tbd
ICT	tbd	ICT_STATE_N	tbd
Initialization	Ready	INIT_STATE_READY	tbd
Initialization	Reply	INIT_STATE_REPLY	tbd
Initialization	Arbitrate	INIT_STATE_ARB	tbd
Initialization	Acknowledged	INIT_STATE_ACKN	tbd
Initialization	Open	INIT_STATE_OPEN	tbd
Initialization	state0	INIT_STATE_0	tbd
Initialization	state1	INIT_STATE_1	tbd
Initialization	…	INIT_STATE_N	tbd
Personalization	Ready	CUST_STATE_READY	tbd
Personalization	Reply	CUST_STATE_REPLY	tbd
Personalization	Arbitrate	CUST_STATE_ARB	tbd
Personalization	Acknowledged	CUST_STATE_ACKN	tbd
Personalization	Open	CUST_STATE_OPEN	tbd
Personalization	state0	CUST_STATE_0	tbd
Personalization	state1	CUST_STATE_1	tbd
Personalization	…	CUST_STATE_N	tbd
SINIAV	Bootup	SIN_STATE_BOOT	configure BTag after power-up
SINIAV	Ready	SIN_STATE_RDY	epc ready state
SINIAV	Arbitrate	SIN_STATE_ARB	epc arb state
SINIAV	Reply	SIN_STATE_REPL	epc reply state
SINIAV	Acknowledged	SIN_STATE_ACKN	epc ackn state
SINIAV	Waiting	SIN_STATE_WAIT	wait for SINIAV access request
SINIAV	MA-Proc	SIN_STATE_MAPROC	siniav processing state
SINIAV	MA-Ready	SIN_STATE_MARDY	siniav processing state
SINIAV	Authenticated	SIN_STATE_AUTH	siniav access state
SINIAV	SAR-Proc	SIN_STATE_SAR_PROC	siniav access state
SINIAV	SAR-Ready	SIN_STATE_SAR_RDY	siniav access state
SINIAV	SAW-Proc	SIN_STATE_SAW_PROC	siniav access state
SINIAV	SAW-Ready	SIN_STATE_SAW_RDY	siniav access state
Killed	Zeroize Tag	KILL_STATE_CLEARTAG	epc ready state using KILLED_EPC
Killed	Ready	KILL_STATE_READY	epc ready state using KILLED_EPC
Killed	Reply	KILL_STATE_REPLY	epc reply state using KILLED_EPC
Killed	Arbitrate	KILL_STATE_ARB	epc arbitrate state using KILLED_EPC

18.6 Firmware Design Rules

The following listing covers most rules imposed on the BTag firmware to increase safety, reliability or robustness.

18.6.1 EPC/SINIAV Tag Memory Access Methods

Memory access shall never use the wordPtr+wordCt index-offset method without checking first if both variables are within bounds to avoid an overflow of the memBank. Bank addressing shall be done indirectly using an enum or equivalent (e.g. bankSel). Before providing access, a separate, well-defined routine shall first be executed to validate the requested range's permission level.

18.6.2 Loops Don't Free-Spin

18.6.3 ISRs Sleep Using the Watchdog

With the exception of P1_ISR and P2_ISR, all code that sleeps and waits on an ISR for wakeup must use a timeout. This includes, for example, sleeping the ADC to obtain a cleaner reading. These ISRs shall never use LPM4 in order to maintain the WDT running. In such a situation modify the code in the following way:

Original (example reading a block of ADC values)

```
  ADC10CTL1 = CONSEQ_2;
  ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
  ADC10DTC1 = 16;
  ADC10AE0  = ADCPin;
  ADC10CTL0 &= ~ENC;
  while (ADC10CTL1 & BUSY);
  ADC10SA = (unsigned int)memLoc;
  ADC10CTL0 |= ENC + ADC10SC;
__bis_SR_register(CPUOFF + GIE);      // LPM0, ADC10_ISR will force exit
  ADC10CTL0 = 0;                      // turn off the ADC
  ADC10AE0  = 0;
```

Adding the WDT.(in case ADC10 never triggers the ADC10IFG)

```
  ADC10CTL1 = CONSEQ_2;
  ADC10CTL0 = ADC10SHT_2 + MSC + ADC10ON + ADC10IE;
  ADC10DTC1 = 16;
  ADC10AE0  = ADCPin;
  ADC10CTL0 &= ~ENC;
  while (ADC10CTL1 & BUSY);
  ADC10SA = (unsigned int)memLoc;
  ADC10CTL0 |= ENC + ADC10SC;
  WDTCTL = SOME_TIME_VAL_DELAY;       //set it to intvlMode on some delay
  IE1 |= WDTIE;
  __bis_SR_register(CPUOFF + GIE);    // LPM0, ADC10_ISR will force exit
  WDTCTL = WDTPW + WDTHOLD;;
  IE1 &= ~WDTIE;
  ADC10CTL0 = 0;                      // turn off the ADC
  ADC10AE0  = 0;
```

**note: this loop shows a free-spinning loop for simplicity. Free-spinning is not implemented in firmware.*

ISRs Which Sleep unbounded

P1 and P2 sleep unbounded, to catch RF edges. In the event of sleeping on these with PxIE disabled, the CPU would lock unbounded. However P1IE and P2IE are only accessed in the tightly controlled and well-defined receive chain module. This risk has been carefully evaluated and by design this module should not enter this locked state under normal operating conditions.

Thinking worst-case for a moment, in the event where a tag did lock into faulty P1IE or P2IE, non-functioning units would be returned to sustaining engineering. They would not respond to commands until the case was opened, upon which they would respond with Killed-Mode ACKS.

The only scenario in which a non-responding SINIAV tag begins to respond after mechanical-*compromise is if it had gotten stuck within the P1/P2 state machine, the only place in the firmware which can/does hang.*

18.6.4 All ISRs are Populated

Ensure that any interrupt vector that can be loaded will jump to an actual routine, For all unused interrupts, have them point to some generic function *unregistered_ISR*.

18.7 Firmware Modules

18.7.1 Receive Chain

Dev Note(2.8.11): The Receive Mode states aren't states, but rather a collection of very small states (e.g. delimSM has 7 substates, and Receive has 16!). I can expand out this document to detail each one painfully, but won't do so until there is an AR to do so (e.g. FIPS requirement or SINIAV requirement). Here are pictures of the BTag0_16 states for each:

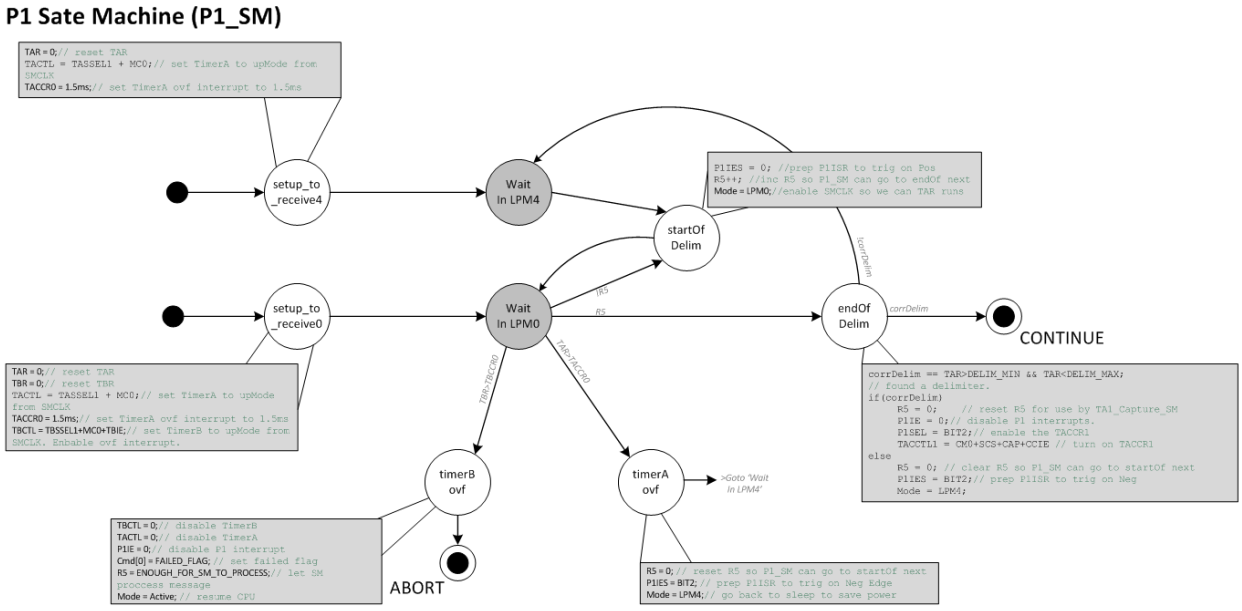


Figure 14: *DelimSM (prev called P1 State Machine)*

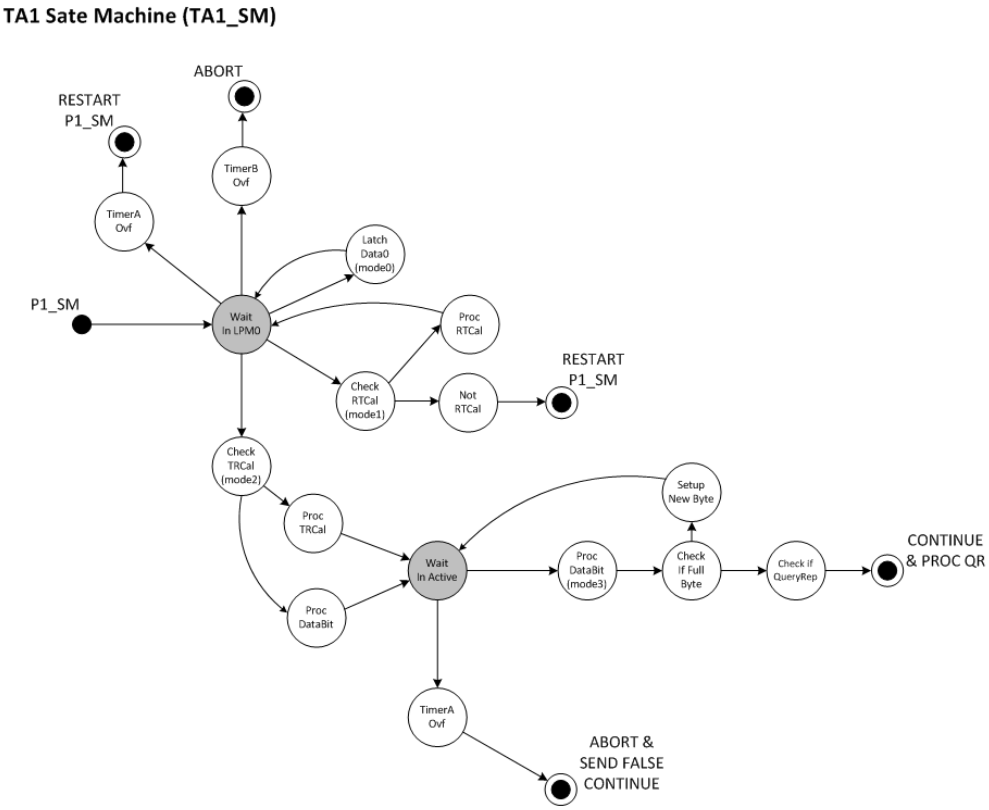


Figure 15: *ReceiveSM (prev called TA1 State Machine)*

For BTag 1_0, I will be reweeding through these eventually.

18.7.2 Transmit Chain

Transmit uses a couple of different buffers and a call to the TxFM0() function to transmit RFID messages back to the reader.

```
void TxFM0( unsigned char    *data,
            unsigned char     numBytes,
            unsigned char     numBits,
            unsigned char     TRext);
```

Data Structures

The transmit chain relies on rfid.TRext to dictate its preamble method, and a few different buffers for storing transmit data. Different buffers for different messages allow for timing optimizations in OBU responses, which allow for the tight timing constraints to be met. There are currently three buffers,

```
unsigned char replyBuf[100];          - For general use
unsigned char readReplyBuf[100];      - For EPC Read only
unsigned char ackReplyBuf[24];        - For EPC Ack only
```

We are rethinking the buffer selection right now, it may be that we consolidate or alter this in coming revisions.

Implementation

Within any rfid/siniav handle function (e.g. sinHandle_MAImplicit_Wait) the calling function is responsible for loading the transmit buffer and then calling the Transmit function. It is also responsible for setting the RSU->OBU interval between RSU command and OBU respsonse, by means if a simple count-down loop.

```
replyBuf[0]= __swap_bytes(rfid.RN16);
replyBuf[1]= rfid.RN16;

timing_i = TX_TIMING_SIN_QUERY_VAL; while(timing_i--);

TxFM0(&replyBuf[0],2,0,rfid.TRext);
```

Transmit Function (TxFM0) Timing and Optimization
The TxFM0 function is a highly unrolled and optimized assembly routine, which is required in order to meet timing requirements. The primary timing constraints is

- Time between chip edges in an FM0 640kHz bit 781.25 ns / 9.375 clock cycles

To dynamically compute FM0 modulation this requires all sorts of timing optimizations all throughout the transmit routine. The critical timing path between chip toggles is set to 9 cycles in many spots throughout the code. This sets the OBU's transmit frequency to 666.67kHz (4.1% deviation which is within ±15% EPC/SINIAV tolerance spec).

Further Optimization

It took us over a week to get the code down from 10 cycles min to 9 cycles min. There are many spots where the 9 cycle constraint is a fundamental limit, e.g. when the bit transmit loop cycles back. All spots are clearly documented in the code.

Here the minimum timing path is highlighted, JNZ+MOV+IN+MOV = 2+2+1+4=9. While not closing the door on future optimizations, it is unclear how the team could optimize further on the MSP430F2xxx architecture.

(Line 248-262, TxFM0.asm)

```
;/(b7)Eighth Bit--------------------
MOV     R_prevState, R_scratch0 ;[1]
INV     R_scratch0              ;[1]
MOV.B   R_scratch0, &P1OUT      ;[4]

XOR     R_currByte, R_prevState ;[1]
;*Timing Optimization Shoved Here   (4 free cycles)*/
DEC     R_byteCt                ;[1] decrement the number of bytes sent
TST.B   R_byteCt                ;[1] test if there are bytes left to send
MOV     R_prevState, R_scratch0 ;[1] load prevStateLogic into scratch0 for calc
                                      (optimized line for b0 of next byte)
NOP                             ;[1] 1 timing cycle
;*End of 4 free cycles*/
MOV.B   R_prevState, &P1OUT     ;[4] *don't worry, MOV doesn't affect Z

JNZ     V1_Load_Data            ;[2] if (byteCt!=0) Cont Sending Bytes

  .
  .
  .
  .
```

(Line 174-181, TxFM0.asm)

```
V1_Load_Data:
    MOV.B     @R_dataPtr+, R_currByte ;[2] load current byte of data

V1_Send_a_Byte:
    ;/(b0)First Bit [FM0 Calculations are only commented for bit0]-----
    ;*optimizedOut:MOV     R_prevState, R_scratch0 ;[1] load prevStateLogic into
                              scratch0 for calc*/;<-Now meets 9 cycles on JMP!
    INV     R_scratch0              ;[1] firstFM0Bit = !prevLogicState (i.e.
                                    STATE0|STATE2 from EPC Spec Table)
    MOV.B   R_scratch0, &P1OUT      ;[4] push bit out on P2.0
```

If the BTAG ever switched to an MCU with a UART peripheral that supported continuous transmit however something even down to 3-4 cycles may be achievable, at which point EPC link timing (e.g. T_1, T_2) becomes our limiting factor. The MSP430F2xxx series does not support continuous transmit, as its USI peripheral has a hardware-appended stop-bit to each byte.

18.7.3 Mode Deployment Module

Mode deployment involves doing a self-test, cleaning state, and then deploying the proper mode as illustrated in Figure 12. A description of this behavior and its implementation follows.

Self-Check: See Section 18.7.11, Self-Check Test Unit for details. Essential modules, hardware and memory are tested for compliance and integrity.

Clean-State: See Section 18.7.12, Clean-State Unit for details. All MCU control and peripherals are reset to a known state, then RAM is reset and the hardware configured. This allows the Btag to reset its HW/SW and memory to a known state in order to mitigate the risk of corrupted state in deployment over the 5 year lifespan.

Mode-Deployment: The Btag mode (`btag.mode`) is stored in the Btag memory bank described in Table 15. This value is read and the proper mode deployed. The mode executes until the appropriate abort flag is thrown, upon which time the mode is aborted and the mode deployment module restarted.

Mode-Abort Flags:
As each mode contains it's own essential state within its own struct (e.g. the `siniav` struct for Siniav Mode), each mode's struct shall contain an `abort` flag, which is reset to FALSE upon every clean-state.

Implementation
The mode deployment module manifests itself as the while(1) in main.c:

```
while(FOREVER) {
      btag_selfCheck();
      btag_cleanState();

      //***Deploy Mode's Thread***//
      switch(btag.mode) {
            case MODE_INIT:
                  initThread();
                  break;
            case MODE_CUST:
                  customThread();
                  break;
            case MODE_SIN:
                  siniavThread();
                  break;
            default:
                  killedThread();
      }
}
```

Each mode, or 'thread', is state-based and obeys the following abort flag architecture. Each mode also includes a boot state which properly configures the state mode for execution.

```
while(!mode.abort) {
      switch(mode.state) {
            case MODE_STATE_BOOT:
            case : …
        }
}
```

18.7.4 EPC Read Handle: Analysis and Design

The EPC Read function presents a significant challenge to our Traditional MCU architecture. Simply put it's a lot to get done in not a lot of clock cycles. By carefully laying out our requirements and timing marks hopefully we can squeeze EPC Read compliance into a sufficient handle using Tari=6.25us and FM0@640kHz.

Reiterating, we assume EPC Read uses 1-byte EBV field.

Maximum Theoretical Time Provided
During command (starting at bit9 received, memBank)

Bits: 125us 2+8+8+16+16=50bits*6.25us*.40ISRLoading=125us
T_1: 15us
Total Cycles: 140us*12 = 1680

Required Components

1. Decoding Fields (unoptimized timing) (86)
Membank(24cyc) *(*`((cmd[1]&0xC0)>>6;`
WordPtr(31cyc) `((cmd[1]&0x1F)<<2) | ((cmd[2]&0xC0)>>6);`
WordCount(31cyc) `((cmd[2]&0x3F)<<2) | ((cmd[3]&0xC0)>>6) );`

2. Loading Read-Bytes into the Buffer
Assuming the max number of bytes to be the length to be 16 words at a time (typical value observed in practice).
Loading 32 bytes in for loop(941!?)
Optimized is 172 (just always loads 32 bytes)

3. Load the RN16 (24 cycles)
Loading RN16 (24cyc)

4. Shift Bytes Right (132cycles)
The fastest way is to always just shift 16 words worth!! My optimized variable routine can't beat this. Shifting bytes with variable routine(549cyc) ← optimize

*Optimized is 4cyc/Byte+ptrLoc preload (4) ->4*33 = 132cycles*

5. CRC calculation over 32 bytes
*Theoretical minimum is 3cycles/bit. (96 cycles*safety factor) ->144*

6. Load CRC
99 cycles

7. Entry into Tx
42 cycles

Conclusion:
***24+31+31+172+24+133+150+99+42 ->958* <- My worst case estimate is 87% (initial guess was 1200)**
Current Iteration:
386+42+150ish+10->588

18.7.5 Tamper-Proof Prevention Module

<Describe the algorithms that disable the MCU here. Section will be described after getting the 0.5 up and running.>

- Light sensor has to be TRUE for 100ms before disabling
- Tamper-Proof has to be TRUE for 50ms before disabling

18.7.6 AES-128 Block Cipher

18.7.7 Block-Mode Ciphers

18.7.8 Optimized CRC16-CCITT Routines

Two ASM versions of the CRC16-CCITT calculation for EPC C1G2 were implemented around the look-up table method (LUT) and bitwise calculates. See the source code for detailed implementation notes. A 256 integer table was generated for the LUT procedure. Both methods are derived from [5] (TI App Note SLAA221). While derived from the app note, considerable amounts of porting and optimization were required in this case!

LUT For Bytes (crc16_ccitt)

The method below is used to calculate the CRC16-ccitt on a sequence of bytes.

1. Load the CRC Register (i.e. if you had a preloaded val, else 0)
   -Bring into working form by inverting
2. CRC[i] = (CRClsb[i-1] | 0x00) ^ asmLUT[CRCmsb[i-1]^data[i]]
3. Repeat for each byte of the message
4. Return the CRC register to output form (invert it again)

Brute Force for Bitwise Calc (crc16Bits_ccitt)

The method below is used to calculate the CRC16-ccitt on a sequence of bits. The BTag crc16Bits_ccitt method uses this after calculating the CRC over numBytes. As no BTag CRC calcs have numBits <16, the crc16Bits_ccitt does not check for the case numBytes = 0 (so don't do it!).

1. Load the CRC Register (i.e. if you had a preloaded val, else 0)
   -Bring into working form by inverting
2. CRC[i] = (CRC[i-1] ^ ((data[i]&0x80) | 0x00)) << 1
3. If a '1' was shifted out, then CRC[i] ^= 0x1021
4. Repeat for each bit of the message
5. Return the CRC register to output form (invert it again)

Implemented Routines for CRC16 Calculation

Two variants for the CRC16-CCITT are designed:

- crc16_ccitt calculate cleanly on a # of bytes
- crc16_ccittBits calculate when not an event # of bytes

Execution Times of Routines

- crc16_ccitt 11+14*numBytes+10
  - (e.g. worst case 4.375cyc/bit, best case 1.75cyc/bit)
- crc16_ccittBits 11+14*numBytes+2+12*numBits+10 (e.g.
  - (e.g. worst case 20.67 cyc/bit, best case 1.75cyc/bit)

Further Optimizations

While increasing the LUT size could decrease cycles/bit a little, with the BTag's low numBytes sizes (16 typ) this will result in <.5cyc/bit perf gain and isn't worth the complexity or memory tradeoffs.

18.7.9 Random-Number Generator

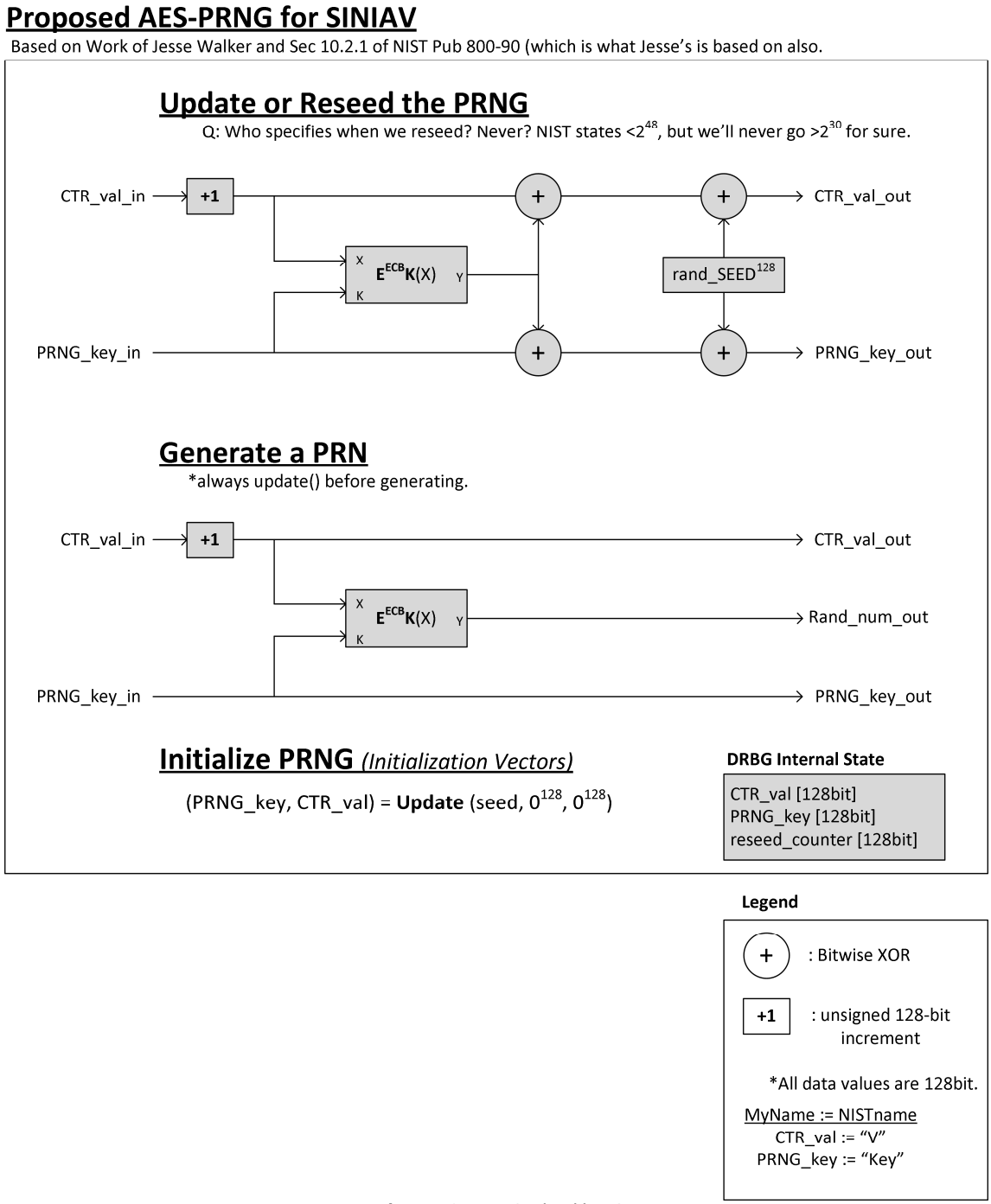


Figure 16: *PRNG Algorithm Summary.*

18.7.10 Tag Memory

18.7.11 Self-Check Test Unit

18.7.12 Clean-State Unit

18.8 MCU Resource Allocation

The 32 interrupt vectors are assigned as follows:
<TODO: list a table here and map it to cut-pasted snipped from source below>

```
*-STATUS--ADDRESS---SOURCE----------INTERRUPT FLAG-------------------------NOTE---------
* (SAFE)  - 0xFFFE    RESER_VECTOR   PWR_UP/EXT_RST/WTD/FLASH_KEY/BAD_PC
* !!!!!!  - 0xFFFC    NMI_VECTOR            NMI/OSCFAULT/FLASH_VIOLATION
* !!!!!!  - 0xFFFA    Timer_B3              TBCCR0 CCIFG                  (TIMERB0_VECTOR)
* (SAFE)  - 0xFFF8    Timer_B3              TBCCR1/2, TBIFG               (TIMERB1_VECTOR
* (SAFE)  - 0xFFF6    (empty)
* (SAFE)  - 0xFFF4    Watchdog_Timer WDTIFG                               (WDT_VECTOR)
* (SAFE)  - 0xFFF2    Timer_A3              TACCR0 CCIFG                  (TIMERA0_VECTOR)
* (SAFE)  - 0xFFF0    Timer_A3              TACCR1/2, CCIFG               (TIMERA1_VECTOR)
* !!!!!!  - 0xFFEE    USCI Rx               UCx0RXIFG                     (USCIAB0RX_VECTOR)
* !!!!!!  - 0xFFEC    USCI Tx               UCx0RXIFG                     (USCIAB0TX_VECTOR)
* (SAFE)  - 0xFFEA    ADC10                 ADC10IFG                      (ADC10_VECTOR)
* (SAFE)  - 0xFFE8    (empty)
* (SAFE)  - 0xFFE6    Port P2 I/O           P2IFG0..7                     (PORT2_VECTOR)
* (SAFE)  - 0xFFE4    Port P1 I/O           P1IFG0..7                     (PORT2_VECTOR)
* (SAFE)  - 0xFFE2    (empty)
* (SAFE)  - 0xFFE0    (empty)
* (SAFE)  - 0xFFDE    BSL                          BSLKEY
* (SAFE)  - 0xFFDC    not used by MSP430F2272
* (SAFE)  - ...               not used by MSP430F2272
* (SAFE)  - 0xFFC0    not used by MSP430F2272
*******************************************************************************
```

18.9 Tag Memory Map

SINIAV requires a specific memory implementation with regards to the over-the-air RFID interface. In this implementation memory “addresses” correspond to how the RFID interface would address the data. There are four memory banks;

- Reserved Memory Bank
- UII Memory Bank
- TID Memory Bank
- User Memory Bank

Each is described below. Also described is the endian-ness and logical partitioning used to implement within the MCUs flash. Each bank is partitioned within it’s own flash segment of 512 bytes.

18.9.1 Memory Access Lifecycle

The tag memory available for external access (i.e. over the air) for both read/write varies dependant on tag mode. This is illustrated below:

Table 7: Over-the-air access permissions to different memory banks across the different tag modes.

Stage	RESERVED		UII		TID_lwr		TID_uppr		USR_lwr		USR_uppr		USR_Killed	
Binary	R	W	R	W	R	W	R	W	R	W	R	W		
ICT	R	W	R	W	R		R	W	R	W	R	W		
Initialization	R	W	R	W	R		R	W	R	W	R	W		
Personalization	R	W	R		R		R	W	R	W	R	W		
Siniav Field Use			R		R		R	W	R		R	W		
Killed													R	

18.9.2 Memory Mapping Procedure

Data is stored differently in the SINIAV map versus the BTag map. This is necessary due to fixed memory architectures on the MCU.

Siniav Map:	Big-Endian stored in 16-bit words.	Uses bit-addresses.
BTag Map:	Big-Endian stored in 8-bit bytes.	Uses byte-addresses.
MSP430:	Little-Endian stored in 8-bit bytes.	Uses bit-addresses.

An explicit example is provided below to alleviate any confusion that may arise from the horrors of memory mapping:

Example: 40-bit RENAVAM Code stored in User Memory
Siniav Start Address: 0x0180 (385th bit in map)
Siniav Layout:

Table 8: *Siniav Memory Map Layout*

Address	0x0180	0x0181	0x0182	…	0x01A5	0x01A6	0x01A7
bit	b39	b38	b37	b36..b3	b2	b1	b0

BTag Start Address: 0x030 (49th byte in map)
BTag Layout:

Table 9: *BTag Memory Map Layout*

Address	0x030	0x031	0x032	0x033	0x034
byte	b39..b32	b31..b24	b23..b16	b16..b8	b7..b0

BTag Byte Mapping *(to be painfully explicit)*:

Table 10: *BTag Byte Mapping of address 0x030*

byte addres	b7	b6	..	b1	b0
data bit	b39	b38	b37..b34	b33	b32

Note
If you're wondering why this painful detail is here, just wait until the day comes when you get it wrong…

18.9.3 Memory Table Definitions

The following tables are derived directly from the Siniav Specification. Listed are both the Siniav and BTag Memory Addresses.

Table 11: *RESERVED Memory Map*

Bank 00: RESERVED Memory				
Siniav Addr	**BTag Addr**	**(MSB) Logical Memory Map (LSB)**	**Siniav LMA**	**Access Control**
0x0000	0x000	Secret Key Authentication (**AK**) [127:120]	0x0007	(L)READ (L)WRITE after Personalized
…	…	…	…	
0x0078	0x00F	Secret Key Authentication (**AK**) [007:000]	0x007F	
0x0080	0x010	Secret Key Session (**DSK**) Standard [127:120]	0x0087	
…	…	…	…	
0x00F8	0x01F	Secret Key Session (**DSK**) Standard [007:000]	0x00FF	
0x0100	0x020	Secret Access Key to Writing (**WK**) [127:120]	0x0107	
…	…	…	…	
0x0178	0x02F	Secret Access Key to Writing (**WK**) [007:000]	0x017F	
0x0180	0x030	Master Key Secret Session (**MSK**) [127:120]	0x0187	
…	…	…	…	
0x01F8	0x03F	Master Key Secret Session (**MSK**) [007:000]	0x01FF	
0x0200	0x040	Secret Key Generator (**DMAK MFK**) [127:120]	0x0207	
…	…	…	…	
0x0278	0x04F	Secret Key Generator (**DMAK MFK**) [007:000]	0x027F	
0x0280	0x050	Secret Key Internal AES-PRNG (**AESK**) [127:120]	0x0287	
…	…	…	…	
0x02F8	0x05F	Secret Key Internal AES-PRNG (**AESK**) [007:000]	0x02FF	

Table 12: *UII Memory Map*

Bank 01: UII Memory				
Siniav Addr	**BTag Addr**	**(MSB) Logical Memory Map (LSB)**	**Siniav LMA**	**Access Control**
0x0000	0x000	CRC-16-CCITT[15:08]	0x0007	(O)READ (L)WRITE after Initialization
0x0008	0x001	CRC-16-CCITT[07:00]	0x000F	
0x0010	0x002	Protocol Control (**PC**) [15:08]	0x0017	
0x0018	0x003	Protocol Control (**PC**) [07:00]	0x001F	
0x0020	0x004	Siniav Protocol Version (**SPV**) [11:04]	0x0027	
0x0028	0x005	Siniav Protocol Version **(SPV)** [03:00] | **SOV** [11:8]	0x002F	
0x0030	0x006	SINIAV OBU Version (**SOV**) [07:00]	0x0037	
0x0038	0x007	OBU Group-ID (**GID**) [23:16]	0x003F	
0x0040	0x008	OBU Group-ID (GID) [15:08]	0x0047	
0x0048	0x009	OBU Group-ID (GID) [07:00]	0x004F	

Table 13: *USER Memory Map*

Bank 11: User Memory				
Siniav Addr	**BTag Addr**	**(MSB) Logical Memory Map (LSB)**	**Siniav LMA**	**Access Control**
0x0000	0x000	Unique Serial Number (**OBU-ID**) [63:56]	0x0007	(O)READ (L)WRITE after Personalization
…	…	…	…	
0x0038	0x007	Unique Serial Number (**OBU-ID**) [07:00]	0x003F	
0x0040	0x008	Plate/Cat/Type/Vehicle (**DATA64**) [63:56]	0x0047	
…	…	…	…	
0x0078	0x00F	Plate/Cat/Type/Vehicle (**DATA64**) [07:00]	0x007F	
0x0080	0x010	SMAC over 0x0000-0x007F (**SMAC**) [127:000]	0x0087	
…	…	…	…	
0x00F8	0x01F	SMAC over 0x0000-0x007F (**SMAC**) [007:000]	0x00FF	
0x0100	0x020	Chassis/VIN [127:120]	0x0107	
…	…	…	…	
0x0178	0x02F	Chassis/VIN [007:000]	0x017F	
0x0180	0x030	RENAVAM Code [39:32]	0x0187	
…	…	…	…	
0x01A0	0x034	RENAVAM Code [07:00]	0x01A7	
0x01A8	0x035	Mfg/Model [23:16]	0x01AF	
…	…	Mfg/Model [15:08]	…	
0x01B8	0x037	Mfg/Model [07:00]	0x01BF	
0x01C0	0x038	License Issuing Identification [63:56]	0x01C7	
…	…	…	…	
0x01F8	0x03F	License Issuing Identification [07:00]	0x01FF	
0x0200	0x040	Agent Registration Number [31:24]	0x0207	
…	…	…	…	
0x0218	0x043	Agent Registration Number [07:00]	0x021F	
0x0220	0x044	Config Date & Time [31:24]	0x0227	
…	…	…	…	
0x0238	0x047	Config Date & Time [07:00]	0x023F	
0x0240	0x048	SINIAV Mfg Ctrl [31:24]	0x0247	
…	…	…	…	
0x0258	0x04B	SINIAV Mfg Ctrl [07:00]	0x025F	
0x0260	0x04C	Authenticity Code over 0x0100-0x025F[127:120]	0x0267	
…	…	…	…	
0x02D8	0x05B	Authenticity Code over 0x0100-0x025F[007:000]	0x02DF	
0x02E0	0x05C	Reserved Gov. Use [31:24]	0x02E7	(O)READ (O)WRITE
…	…	…	…	
0x02F8	0x05F	Reserved Gov. Use [07:00]	0x02FF	
0x0300	0x060	Reserved DENATRAN Use [127:120]	0x0307	
…	…	…	…	
0x0378	0x06F	Reserved DENATRAN Use [007:000]	0x037F	
0x0380	0x070	Reserved Private Enterprise [383:376]	0x0387	
…	…	…	…	
0x04F8	0x09F	Reserved Private Enterprise [007:000]	0x04FF	

Table 14: *TID Memory Map*

Bank 10: TID Memory				
Siniav Addr	**BTag Addr**	**(MSB) Logical Memory Map (LSB)**	**Siniav LMA**	**Access Control**
0x0000	0x000	ISO 15963 ID [07:00]	0x0007	(O)READ (L)WRITE after Manufacture
0x0008	0x001	MDID [11:04]	0x000F	
0x0010	0x002	MDID [03:00] | Tag Model# [11:08]	0x0017	
0x0018	0x003	Tag Model # [07:00]	0x001F	
0x0020	0x004	Tag ID [63:56]	0x0027	
0x0028	0x005	Tag ID [55:48]	0x002F	
0x0030	0x006	Tag ID [47:40]	0x0037	
0x0038	0x007	Tag ID [39:32]	0x003F	
0x0040	0x008	Tag ID [31:24]	0x0047	
0x0048	0x009	Tag ID [23:16]	0x004F	
0x0050	0x00A	Tag ID [15:08]	0x0057	
0x0058	0x00B	Tag ID [07:00]	0x005F	
0x0060	0x00C	Empty (RFFU)	0x0060	(O)READ (O)WRITE
0x0061	0x00C	…	0x04FE	
0x04FF	0x09F	Empty (RFFU)	0x04FF	

Table 15: *BTAG Memory Map*

Siniav Addr	BTag Addr	(MSB) Logical Memory Map (LSB)	Siniav LMA	Access Control
0x0000	0x000	(empty)	0x0007	(L)READ (L)WRITE
0x0008	0x001	DCRC[15:08]	0x000F	
0x0010	0x002	DCRC[07:00]	0x0017	
0x0018	0x003	ACK CRC[15:08]	0x001F	
0x0020	0x004	ACK CRC[07:00]	0x0027	
0x0028	0x005	BTag Mode [07:00]	0x002F	
0x0030	0x006	Debug Data [319:212]	0x0037	
0x0038	0x007	…	0x0167	
0x0168	0x02D	Debug Data [07:00]	0x016F	
0x0170	0x02E	Program FLASH CRC [15:00]	0x0177	
0x0178	0x02F	Program FLASH CRC [07:00]	0x017F	
0x0180	0x030	Killed UserMem CRC_lwr [15:08]	0x0187	
0x0188	0x031	Killed UserMemCRC_lwr[07:00]	0x018F	
0x0190	0x032	Killed UserMem CRC_upr[15:08]	0x0197	
0x0198	0x033	Killed UserMem CRC_upr[07:00]	0x019F	
0x01A0	0x034	Activation Token[63:56]	0x01A7	
0x01A8	0x035	…	0x01D7	
0x01D8	0x03B	Activation Token[07:00]	0x01DF	
0x01E0	0x03C	0^6 [07:02] | TokenDesc [01:00]	0x01E7	

Killed UserMem CRCs are calculated by prefixing with a '0' bit as done in the EPC READ Command. In this way when they are used in killed.READ we have properly pre-calculated the CRC up to the RN16 field.

Table 16: *Killed Mode Memory Results for the USER bank. OBU does not respond to any other read requests.*

Bank 11: USER Memory in Killed Mode			
Siniav Addr	**BTag Addr**	**(MSB) Logical Memory Map (LSB)**	**Siniav LMA**
0x0000	0x000	Max Stack Depth[15:08]	0x0007
0x0008	0x001	Max Stack Depth[07:00]	0x000F
0x0010	0x002	Aborted Mode [07:00]	0x0017
0x0010	0x002	Abort Code [15:08]	0x0017
0x0018	0x003	Abort Code [07:00]	0x001F
0x0020	0x004	Abort Descriptor[15:08]	0x0027
0x0028	0x005	Abort Descriptor[07:00]	0x002F
0x0030	0x006	Ack Count[31:24]	0x0037
0x0038	0x007	Ack Count[23:16]	0x003F
0x0040	0x008	Ack Count[15:08]	0x0047
0x0048	0x009	Ack Count[07:00]	0x004F
0x0050	0x00A	sinMA Count[31:24]	0x0057
0x0058	0x00B	sinMA Count[23:16]	0x005F
0x0060	0x00C	sinMA Count[15:08]	0x0067
0x0068	0x00D	sinMA Count[07:00]	0x006F
0x0070	0x00E	sinSAR Count[31:24]	0x0077
0x0078	0x00F	sinSAR Count[23:16]	0x007F
0x0080	0x010	sinSAR Count[15:08]	0x0087
0x0088	0x011	sinSAR Count[07:00]	0x008F
0x0090	0x012	sinSAW Count[31:24]	0x0097
0x0098	0x013	sinSAW Count[23:16]	0x009F
0x00A0	0x014	sinSAW Count[15:08]	0x00A7
0x00A8	0x015	sinSAW Count[07:00]	0x00AF
0x00B0	0x016	Last Siniav State[07:00]	0x00B7
0x00B8	0x017	No. TC Retries [15:08]	0x00BF
0x00C0	0x018	No. TC Retries [08:00]	0x00C7
0x00C8	0x019	No. Siniav Errors [15:08]	0x00CF
0x00D0	0x01A	No. Siniav Errors [07:00]	0x00D7
0x00D8	0x01B	No. Photo Mistriggers [07:00]	0x00DF
0x00E0	0x01C	No. Mech Mistriggers [07:00]	0x00E7
0x00E8	0x01D	No. Loop Aborts [07:00]	0x00EF
0x00F0	0x01E	No. ISR Mistriggers [07:00]	0x00F7

18.9.4 Memory Access Behavior

RESERVED: Permanently Locked after Personalization Read by crypto secure methods only.
UII: Permanently Locked after Personalization. No restrictions on read.
TID: Tag-ID bits 0x0020 through 0x005F are locked after manufacture, indicated by - the 'Tag-ID Is Active Flag in BTAG Memory 0x00.

BTAG: Write Acess is Disabled.

18.10 Siniav Transmission Counter (TC)

The Siniav transmission counter is used to allow RSU retransmissions of custom commands. It makes things unnecessarily complex. Because of the scattered nature of its definition throughout the protocol (i.e. TC requirements exist in 3 different sections), the explicit BTag interpretation of TC is listed here.

Definition of TC

- The TC is modulo 2, i.e. a binary flag that toggles each time it is incremented (§8.1.4,¶1)
- The TC is *reset* to a state of '0' by the OBU on each successful ACK command sent by a RSU (§8.1.4,¶3)
- The RSU increments the TC before *each* new command. (§8.1.4,¶4). The RSU will then transmit a TC==1 for the first Req_Handle it transmits in a Siniav round (our interpretation).

Management of TC by OBU

- The OBU resets the TC=0 after *every* successful ACK command from RSU
- Upon receiving any Siniav Command, the OBU first compares the received TC with it's (localTC+1).
    - If they match, OBU first increments its TC, then proceeds and treats command as new message
    - If they do not match, the OBU then checks if transmit buffer is ready with a response to this new RSU command.
        - If the buffer has the appropriate response, it is transmitted again without modification
        - If the buffer is not appropriate for the command, the OBU prepares the buffer with the 'other error' error code, and transmits this response instead.

Special Exceptions

- In the special case of Req_Handle, the *last primary response* is retransmitted. This implies having a special transmit buffer for the primary response, which is unfortunately complicated.

Example of TC Flow:

Entry: RSU_TC = D.C., OBU_TC = D.C.

RSU sends ACK Command to OBU. OBU receives and validates ACK, then sets local TC to 0. Upon reception of successful OBU ACK response, RSU also sets local copy of TC to 0.

Exit: RSU_TC=0, OBU_TC=0.

RSU sends Req_Handle command to OBU with TC=1. OBU receives and validates Req_Handle, then compares received TC (1) to (localCopy+1)(1). If they match tag treats as a new command and preps/delivers a new Req_Handle response. Upon successful reception of OBU Req_Handle response, RSU proceeds forward with TC=1. OBU also sets local copy to 1.

Exit: RSU_TC=1, OBU_TC=1.

RSU sends MA_Implicit command to OBU with TC=0. OBU receives and validates MA_Implicit command, then compares received TC(0) to (localCopy+1)(0). If they match the tag treats as new command and preps/delivers new CC_Primary response. Upon successful reception of OBU Req_Handle response, RSU proceeds forward with TC=0. OBU also sets local copy to 0.

Exit: RSU_TC=0, OBU_TC=0.

RSU sends Finalize command to OBU with TC=1. OBU receives and validates Finalize command, then compares received TC(1) to (localCopy+1)(1). If they match the tag treats as new command and

preps/delivers the CC_Primary response. Upon successful reception of OBU response, RSU proceeds forward with TC=1.

Exit: RSU_TC=1, OBU_TC=1

Injected Error
RSU sends SAR command to OBU with TC=0. OBU receives and validates SAR command, then compares received TC(0) to (localCopy+1)(0). If they match the tag treats as new command and preps/delivers new CC_Primary response. However the RSU does not successfully receive the OBU's response, perhaps due to an RF collision. Because the RSU received no response, it does not alter its local TC copy.

Exit: RSU_TC=1, OBU_TC=0.

RSU chooses to resend same SAR command to OBU with RSU_TC=0. OBU receives and validates SAR command, then compares received TC (0) to (localCopy+1)(1). Because they do not match, the OBU marks the command as a retransmission. The OBU buffer is verified to contain correct contents, and then is retransmitted without modification. Upon successful reception, the RSU proceeds forward with TC=0. In this case the tag does not modify it's TC.

Exit: RSU_TC=0, OBU_TC=0.

RSU sends Finalize command to OBU with TC=1. OBU receives and validates Finalize command, then compares received TC(1) to (localCopy+1)(1). If they match the tag treats as new command and preps/delivers the CC_Primary response. Upon successful reception of OBU response, RSU proceeds forward with TC=1.

Exit: RSU_TC=1, OBU_TC=1

…

RSU sends ACK Command to OBU. OBU receives and validates ACK, then sets local TC to 0. Upon reception of successful OBU ACK response, RSU also sets local copy of TC to 0.

Exit: RSU_TC=0, OBU_TC=0.

18.11 Error Codes and Handling

Siniav requires reporting of errors by OBU to RSU during communication. The following is a reproduction of Table 8.3 from [1]. Some errors are handled by only state modification, and some by modification & the reporting of an error code.

Table of Siniav Errors and Naming Conventions (some errors have specific codes, detailed in the next table)

SINERR_FAILEDCRC
SINERR_EARLYFINALIZE
SINERR_CIPHERPWR
SINERR_AUTH
SINERR_INTEG
SINERR_SYN
SINERR_HANDLE
Just means the OBU received a message for someone else.

Table 17: *Table of Siniav required error codes.*

Error Code Support	Error Code	Error Code Name	Notation	Error Description
Specific	0000 0000	Other Error	SINERR_OTHER	Basket for errors not coverred by other codes
	0000 0011	Memory surplus value or unsupported PC value	SINERR_MEMOVF	The address specific memory does not exist or the PC value is not suported by the OBU
	0000 0100	Locked Memory	SINERR_MEMLCK	he address specific memory is locked and / or locked permanently and is or prohibited for OBU writing or reading
	0000 1011	Insufficient Power for Write	N/A	The OBU has insufficient power to perform the memory write operation
	1100 0000	Authentication Failed	SINERR_AUTH	A RSU failed authentication in mutual authentication implicit or as part of the read / write access control
	1100 0011	Integrity Failed	SINERR_INTEG	RSU's integrity checking of custom command parameters failed
	1100 1100	Syntactic Compliance	SINERR_SYN	RSU custom command parameters contained illegitimate values
	1100 1111	Insufficient Power for Cipher Machine	N/A	The OBU has insufficient power to perform cipher machine required operations
	1111 0000	Insufficient Processing Time	N/A	The internal processing of the last custom command received has not yet completed
Unspecified	0000 1111	Unspecified Error	SINERR_UNSP	The OBU does not support an unspecified error.

18.11.1 Unsupported Error Codes

Because the BTag is an active tag using an MCU control unit, a few of the error codes will not occur by design. The BTag will not generate the following error codes:

1. (0Bh) Insufficient Power for Write
2. (CFh) Insufficient Power for Cipher Machine
3. (F0h) Insufficient Processing Time

18.11.2 Handled Errors

The following is a detailed listing of possible error conditions and responses by the OBU. Where applicable the specific reference from the protocol in [1] is provided.

Req_Handle

1. Invalid CRC [SINERR_FAILEDCRC] [§8.5.5, 2-2.b]

NOT HANDLED

2. OSM.CAT !=0/1/7 [SINERR_SYN] [§8.5.5, 2-6.d]

Mutual_Authentication_Implict

1. Invalid CRC [SINERR_FAILEDCRC] [§8.7.4, 2-1.b]
2. Invalid RFFU [SINERR_SYN] [§8.7.4, 2-2.b]
3. Invalid RFFUP [SINERR_SYN] [§8.7.4, 2-2.b]
4. Invalid GSK [SINERR_SYN] [N/A]
5. Invalid SMD [SINERR_SYN] [§8.7.4, 3-12.c]
6. Invalid DMD [SINERR_SYN] [§8.7.4, 3-20.b.ii/.c]

Secure_Access_Read

1. Invalid CRC [SINERR_FAILEDCRC] [§8.8.4, 2-1.b]
2. Invalid RFFU [SINERR_SYN] [§8.8.4, 2-4.b]
3. Invalid T64 [SINERR_AUTH] [§8.8.4, 2-7.b]
4. Invalid WDCRC [SINERR_INTEG] [§8.8.4, 2-8.b]
5. Invalid MLD.Vers [SINERR_SYN] [§8.8.4, 2-9]
6. Invalid MLD range [SINERR_SYN]
7. Invalid WDCRC [SINERR_SYN]

MEMOVF or MEMLCK

8. Invalid MLDDMD [SINERR_SYN] [§8.8.4, 3-8.b/c]

Secure_Access_Write

1. Invalid CRC [SINERR_FAILEDCRC] [§8.9.4, 2-1.b]
2. Invalid DMD [SINERR_SYN] [§8.9.4, 2-4.a.a/.b/.c]
3. Bad DMAC [SINERR_SYN] [§8.9.4, 2-6.b]

NOT HANDLED

1. Invalid RFFU [SINERR_SYN] [§8.9.4, 2-7.b]
4. Invalid MLD.DMD [SINERR_INTEG] [§8.9.4, 2-10.b]
5. Invalid T64 [SINERR_AUTH] [§8.9.4, 2-11.b]
6. Invalid MLDCRC [SINERR_INTEG] [§8.9.4, 2-12.b]
7. Invalid MLD.Vers [SINERR_SYN] [§8.9.4, 2-13]
8. Invalid MLD range [SINERR_SYN] [§8.9.4, 2-14]

MEMOVF or MEMLCK

9. Invalid WDCRC [SINERR_INTEG] [§8.9.4, 3-8.b/c]
10. Invalid DW Pad [SINERR_INTEG] [§8.9.4, 2-18b]

Finalize

(None)

18.11.3 Treatment of Errors

The BTag will handle and respond to only the *last* detected error during processing according to the sequential numbering specified in ‘Handled Errors’ above. This simplifies the processing.

18.12 Error Handling

<todo…>

18.13 Killed Mode

To increase the effectiveness of sustaining engineering, the BTag will have the following behavior and characteristics in the event of a fatal error. Recall that fatal errors are formally defined in Section 15.9.

Killed Mode Behavior
The tag will respond to EPC commands QUERY,QUERYREP,QUERYADJ,REQ_RN,ACK and READ. The tag will only respond in a specific format to Read.

For the EPC in an ACK response, the EPC of 0^{80} concatenated with the 16-bit Abort Code ID will be transmitted. The generic Abort Code ID will state why the tag switched to killed mode in a manner that is generic enough to prevent further security compromise, advancement of future attacks, or the ability to reverse-engineer.

For the READ command, the tag will reveal the killed debug data in two specific scenarios, when an 16-word read from the USER-BANK is requested starting at WordPtr==0xEE12 or WordPtr=0xFF13. The tag will then proceed to return the contents of the Killed Debug Data, as described next.

- WordPtr=0xEE12, WordCt=8 will reveal Killed User Bank 0x0000-0x007F
- WordPtr=0xFF14, WordCt=8 will reveal Killed User Bank 0x0000-0x00FF

Killed Mode Debug Data
The following data is identified as useful in sustaining-engineering defect troubleshooting, and is included as debug data. This information is dedicated partially to its own struct, debugData.

The memory during a killed mode read is mapped as follows, in order:

- Max Stack Depth [INT]
- Abort Mode [CHAR]
- Abort Code [INT]
- Abort ID [INT]
- Ack Count [LONG]
- SinMACount [LONG]
- SinSARCount [LONG]
- SinSAWCount [LONG]
- lastSinState [CHAR]
- numTCRretries [INT]
- numSinErrors [INT]
- numPhotoMistriggers [CHAR]
- numMechMistriggers [CHAR]
- numLoopAborts [CHAR]
- numISRMistriggers [CHAR]

Abort Codes
Please look to the actual source code for the most current set of Abort Codes as #deffed in the Modes/killed.h header file.

19 State-Transition Tables

19.1 Present State: BOOT_STATE_ON

Description:
Default bootloader provided by TI. Responsible for loading the RAM, initializing the peripherals, setting interrupt vectors and launching the main code.

Security:
Assumed secure unless told otherwise. This is default code provided by TI.

Procedure:
After power-up, processor-fault or power-on-reset this state is entered.

Entry States:	Exit States:
• any	• BOOT_STATE_TEST
Pre:	**Post**
• any	• RAM/IV are loaded, peripherals initialized • main code is launched

State Transitions:

Event	Condition	Action	Next State
Configuration Complete	all	launch main code	**BOOT_STATE_TEST**

Data Accessed:
TBD

19.2 Present State: BOOT_STATE_TEST

Description:
BTag executes a set of power-on tests to verify its integrity and security. If any fail it transitions to Killed mode.

Security:
TBD

Procedure:
TBD. These tests are specified by FIPS primarily. May include:

- FLASH Integrity Check
- CSP Data Integrity Check
- Crypto Modules Validation
- Voltage / Temp Check
- Check Num of power-on-resets

Will develop this state when its specifications arrive.

Entry States:	Exit States:
• BOOT_STATE_ON	• BOOT_STATE_SEL
Pre:	**Post**
• FLASH, RAM are freshly loaded by Bootloader	• BTag is validated for integrity

State Transitions:

Event	Condition	Action	Next State
finish	test passed	-	**BOOT_STATE_SEL**
	test failed	mode = killed	**BOOT_STATE_SEL**

Data Accessed:
TBD

19.3 Present State: BOOT_STATE_SEL

Description:
BTag performs a reinitialization routine, performs a self-check, and the selects which mode to enter.

Security:
To be analyzed further.

Procedure:
1. **Reinitalization:** Call btag_cleanState(). Reinitializes peripherals and RAM, then reconfigures IO/CLK.
2. **Self-Check:** TBD. Will do stuff recommended by FIPS
3. **Select Mode:** Deploy thread for proper mode.

Entry States:	Exit States:
• BOOT_STATE_TEST	• KILL_STATE_CLEARTAG • SIN_STATE_BOOT • INIT_STATE_READY **(TBD)** • CUST_STATE_READY **(TBD)**
Pre:	**Post**
• none	• if CRC fails, mode = Killed

State Transitions:

Event	Condition	Action	Next State
valid CRC	mode == init	-	**INIT_STATE_BOOT**
	mode == custom	-	**CUST_STATE_BOOT**
	mode == SINIAV	-	**SIN_STATE_BOOT**
	mode == killed	-	**KILL_STATE_CLEARTAG**
otherwise		mode = killed	**KILL_STATE_CLEARTAG**

Data Accessed:
TBD

19.4 Present State: RX_STATE_DELAY

Description:
If the BTag determines it is being accessed too frequently it may choose to enter this state. This state is simply a timed loop where the tag is inactive, using VLOCLOCK to run at low power.

Security:

Procedure:
Use VLOCLOCK to sleep until a timerA OVF.

Entry States:	Exit States:
• Any	• Return to Caller State
Pre:	**Post**
• Called from within another state.	• TimerB/TimerA are disabled. • Interrupts are disabled, none are pending

State Transitions:

Event	Condition	Action	Next State
Delay Expires	TimerA OVF	-	**Return to Caller State**

Data Accessed:
TBD

19.5 Present State: RX_STATE_DELIM

Description:
This state waits (either forever, or on a timeout) for a valid EPC delimiter reception (e.g. a low pulse [12..17]us). A combination of the Port1_ISR and TimerA1_ISR are used to execute this state (interrupt driven) and the CPU is off for its entire duration.
*TimerA0_ISR and TimerB1_ISR are also used to watch for timeouts.

Most of the BTag's lifespan will be spent in this state, sleeping and waiting for a command.

Security:
This state does not access any CSP data, and only accesses a fixed set of variables.

Procedure:
Call setup_to_receive0() or 4() to enter this state.

- Setup_to_receive0() goes straight to using the receive-comparator to check for delim
- Setup_to_receive4() first goes to using the wakeup-comparator to wait for RF Power. On detection it transitions to watching for a valid delimiter in the same method as _0().

Entry States:	Exit States:
• Any Initialization, Customization, SINIAV or Killed state which is initiating the reception of an EPC command.	• RX_STATE_BITS • Return to Caller State
Pre:	**Post**
• Called from within another state. • D.C. on cmd[] buff, registers or timers. setup_to_receive configures these.	• RX_STATE_BITS: o TimerA/B is prepped for next state o P1_int is disabled o Cmd buffer is prepped for next state • Return to previous State o Abort Signals are thrown o Interrupts are disabled

State Transitions:

Event	Condition	Action	Next State
Delimiter Measured	12 < delim < 17 [us]	Prep for RX_STATE_BITS	**RX_STATE_BITS**
	otherwise	Prep for RX_STATE_DELIM	**RX_STATE_DELIM**
Delimiter Timed Out	TimerA overflowed	Prep for RX_STATE_DELIM	**RX_STATE_DELIM**
EPC T2 Timeout	TimerB overflowed	Abort Signals are thrown	**Return to Caller State**

Data Accessed:
TBD

19.6 Present State: RX_STATE_BITS

Description:
Receive EPC C1G2 bits from the Rx line. Receives bits until:

- Enough bits to decode a reader comman d (e.g. 10 bits)
- A queryRep (4 bits)
- TimerA OVF
- TimerB OVF
- Too many bits

Resumes the CPU clock as soon as any of the above conditions are met.

Security:
This state inputs data into the cmd[] buffer, up to a certain number of bits to prevent buffer overrun. It also accesses CPU registers directly, but in a finitely described way (e.g. just to put bits into the buffer).

Procedure:
In sequence:

1. Discard first data0 bit
2. Measure RTCal. If not right size, restart the RX_STATE_DELIM
    - Increased resistance to false-decodes. See Appendix A: Receive Chain Design
3. Measure TRCal
4. Receive bits until one of the above events happen
5. Return to caller state

Entry States:	Exit States:
• RX_STATE_DELIM	• Return to Caller State
Pre:	**Post**
• TimerA is setup, P1_ISR is disabled, CPU is off • Exclusive access to R4,R5,R6,R7,R8 registers	• Either a received message or an error message is in cmd[]. • R5 indicates total # of bits, R6 indicates # of bits in last cmd byte

State Transitions:

Event	Condition	Action	Next State
Didn't Catch first data0	TimerA OVF	prep for RX_STATE_DELIM	**RX_STATE_DELIM**
A data-bit timed out	TimerA OVF	abort signals are thrown	**Return to Caller State**
RTCal Wrong Size	RTCal ! [x..y]us	prep for RX_STATE_DELIM	**RX_STATE_DELIM**
EPC T2 Timeout	TimerB OVF	abort signals are thrown	**Return to Caller State**
QueryRep Found	first 4 bits are Qrep	signal that it was Qrep	**Return to Caller State**
Too Many Bits	R5>200	abort signals are thrown	**Return to Caller State**

Data Accessed:
TBD

19.7 Present State: INIT_STATE_0

Will develop when initialization specifications arrive.

19.8 Present State: INIT_STATE_1…

Will develop when initialization specifications arrive.

19.9 Present State: CUST_STATE_0

Will develop when customization specifications arrive.

19.10 Present State: CUST_STATE_1…

Description:

template

Security:

Procedure:

Entry States:	Exit States:
•	•
Pre:	**Post**
•	•

State Transitions:

Event	Condition	Action	Next State

Data Accessed:

TBD

19.11 Present State: SIN_STATE_BOOT

Description:
BOOT_STATE_SEL moves tag into this state if in SINIAV mode. This state configures BTag for SINIAV operation.

State reserved for ongoing development of BTag. This state will most likely ensure that SINIAV state variables are configured correctly, and potentially perform some self-testing to ensure integrity, etc.

Security:
TBD

Procedure:
SINIAV state is configured here.

Entry States:	Exit States:
• BOOT_STATE_SEL	• SIN_STATE_READY • BOOT_STATE_SEL?
Pre:	**Post**
• TBD	• Might change mode=killed?

State Transitions:

Event	Condition	Action	Next State
checked ok	-	-	**SIN_STATE_READY**
not ok	-	mode = Killed	**SIN_STATE_SEL**

Data Accessed:
TBD

19.12 Present State: SIN_STATE_READY

Description:
Exact Equivalent to EPC Ready State.

Security:
No access to any CSP. Just does EPC Inventory Stuff.

Procedure:
Respond to EPC select and inventory commands.

Entry States:	Exit States:
• All SINIAV States (READY…SAW_PROC)	• SIN_STATE_READY • SIN_STATE_REPLY • SIN_STATE_ARB • SIN_STATE_BOOT o to do periodic self-testing
Pre:	**Post**
• none	• Changes to EPC state

State Transitions:

Event	Condition	Action	Next State
Query	slot=0; matching inventoried & SL flags	backscatter new RN16	**SIN_STATE_REPLY**
	slot<>0;matching inventoried & SL flags	-	**SIN_STATE_ARB**
	otherwise if lastTest #< MAX_TESTS	-	**SIN_STATE_BOOT**
	otherwise	-	**SIN_STATE_READY**
Select		assert or deassert SL, or set inventoried to A or B	**SIN_STATE_READY**
all other EPC Commands	-	-	**SIN_STATE_REPLY**

Data Accessed:
TBD

19.13 Present State: SIN_STATE_ARB

Description:
Exact Equivalent to EPC Arbitrate State.

Security:
No access to any CSP. Just does EPC Inventory Stuff.

Procedure:
Respond to EPC inventory commands.

Entry States:	Exit States:
• SIN_STATE_READY • SIN_STATE_REPLY • SIN_STATE_ACKN • SIN_STATE_AUTH • SIN_STATE_WAITING	• SIN_STATE_READY • SIN_STATE_REPLY
Pre:	**Post**
• Tag is participating in an inventory round	• Tag is still participating in an inventory round

State Transitions:

Event	Condition	Action	Next State
Query	slot=0; matching inventoried & SL flags	backscatter new RN16	**SIN_STATE_REPLY**
	slot<>0;matching inventoried & SL flags	-	**SIN_STATE_ARB**
	otherwise	-	**SIN_STATE_READY**
QueryRep	slot=0 after decrementing slot counter	backscatter new RN16	**SIN_STATE_REPLY**
	slot<>0 after decrementing slot counter	-	**SIN_STATE_ARB**
QueryAdj	slot=0	backscatter new RN16	**SIN_STATE_REPLY**
	slot<0>	-	**SIN_STATE_ARB**
Select		assert or deassert SL, or set inventoried to A or B	**SIN_STATE_READY**
all other EPC Commands	-	-	**SIN_STATE_ARB**

Data Accessed:
TBD

19.14 Present State: SIN_STATE_REPLY

Description:
Exact Equivalent to EPC Reply State.

Security:
No access to any CSP. Just does EPC Reply Stuff.

Procedure:
Respond to EPC inventory commands.

Entry States:	Exit States:
• SIN_STATE_READY • SIN_STATE_ARB • SIN_STATE_ACKN • SIN_STATE_AUTH • SIN_STATE_WAITING	• SIN_STATE_READY • SIN_STATE_ARB • SIN_STATE_ACKN
Pre:	**Post**
• Tag is participating in an inventory round	• Tag is still participating in an inventory round, or is in EPC Access (i.e. acknowledged). • Tag has revealed its EPC if it backscattered

State Transitions:

Event	Condition	Action	Next State
Query	slot=0; matching inventoried & SL flags	backscatter new RN16	**SIN_STATE_REPLY**
	slot<>0;matching inventoried & SL flags	-	**SIN_STATE_ARB**
	otherwise	-	**SIN_STATE_READY**
QueryRep	all	-	**SIN_STATE_ARB**
QueryAdj	slot=0	backscatter new RN16	**SIN_STATE_REPLY**
	slot<0>	-	**SIN_STATE_ARB**
ACK	valid RN16	backscatter PC+EPC+CRC	**SIN_STATE_ACKN**
	invalid RN16	-	**SIN_STATE_ARB**
Select		assert or deassert SL, or set inventoried to A or B	**SIN_STATE_READY**
all other EPC Commands	-	-	**SIN_STATE_ARB**
EPC T2 Timeout	all	-	**SIN_STATE_ARB**

Data Accessed:
TBD

19.15 Present State: SIN_STATE_ACKN

Description:
Exact Equivalent to EPC Acknowledged State

Security:
No access to any CSP. Just does EPC Acknowledged Stuff.

Procedure:
On ACK, Backscatter: <epcHandle + PC + SPC + GID + CRC>,
then goto SIN_STATE_WAITING.

Entry States:	Exit States:
• SIN_STATE_REPLY	• SIN_STATE_READY • SIN_STATE_REPLY • SIN_STATE_ARB • SIN_STATE_WAIT
Pre:	**Post**
• Tag is participating in an inventory round	• Tag is still participating in an inventory round, or is in EPC Access (i.e. acknowledged). • Tag has revealed its EPC if it backscattered

State Transitions:

Event	Condition	Action	Next State
Query	slot=0; matching inventoried & SL flags	backscatter new RN16	**SIN_STATE_REPLY**
	slot<>0;matching inventoried & SL flags	-	**SIN_STATE_ARB**
	otherwise	-	**SIN_STATE_READY**
QueryRep	all	-	**SIN_STATE_ARB**
QueryAdj	slot=0	backscatter new RN16	**SIN_STATE_REPLY**
	slot<0>	-	**SIN_STATE_ARB**
ACK	valid RN16	backscatter PC+EPC+CRC	**SIN_STATE_ACKN**
	invalid RN16	-	**SIN_STATE_ARB**
Select		assert or deassert SL, or set inventoried to A or B	**SIN_STATE_READY**
Req_Handle	CAT=000	modify invFlag, backscatter	**SIN_STATE_WAIT**
	CAT=001 & !RESET	flag ops, backscatter	**SIN_STATE_WAIT**
	CAT=001 & RESET	flag operations	**SIN_STATE_READY**
	invalid RN16 or invalid TC	-	**SIN_STATE_ARB**
all other EPC Commands	-	-	**SIN_STATE_ARB**
EPC T2 Timeout	all	-	**SIN_STATE_ARB**

Data Accessed: TBD

19.16 Present State: SIN_STATE_WAITING

Description:
SINIAV Waiting is similar to the EPC Acknowledged state where the OBU is just waiting to be accessed. It also marks the tags deviation from the EPC State Diagram into SINIAV.

Security:
No access to any CSP. Just does SINIAV Req_Handle process of getting a new handle.

Procedure:
Respond to EPC/SINIAV Commands. On MA_Implicit provide CC-Prim Response, then switch to SIN_STATE_MAPROC to decode the received command and begin crypto processing.

Save CRC and crypto processing for SIN_STATE_MAPROC (i.e. backscatter response before validating the message).

Entry States:	Exit States:
• SIN_STATE_ACKN	• SIN_STATE_READY • SIN_STATE_REPLY • SIN_STATE_ARB • SIN_STATE_MAPROC
Pre:	**Post**
• RSU used Req_Handle to transition Tag out of SIN_STATE_ACKN	• Received crypto data from RSU, but haven't decoded , verified CRC or prepped it yet

State Transitions:

Event	Condition	Action	Next State
Query	slot=0; matching inventoried & SL flags	backscatter new RN16	**SIN_STATE_REPLY**
	slot<>0;matching inventoried & SL flags	-	**SIN_STATE_ARB**
	otherwise	-	**SIN_STATE_READY**
QueryRep	all	-	**SIN_STATE_ARB**
QueryAdj	slot=0	backscatter new RN16	**SIN_STATE_REPLY**
	slot<0>	-	**SIN_STATE_ARB**
ACK	valid RN16	backscatter PC+EPC+CRC	**SIN_STATE_WAIT**
	invalid RN16	-	**SIN_STATE_ARB**
Select		assert or deassert SL, or set inventoried to A or B	**SIN_STATE_READY**
Req_Handle	CAT=000	modify invFlag, backscatter	**SIN_STATE_WAIT**
	CAT=001 & !RESET	flag ops, backscatter	**SIN_STATE_WAIT**
	CAT=001 & RESET	flag operations	**SIN_STATE_READY**
	invalid RN16 or invalid TC	-	**SIN_STATE_ARB**
Mutual_Auth_Implicit	valid handle	backscatter, Prepare Response	**SIN_STATE_MAPROC**
	invalid handle	not sure. Maybe err?	**???**
all other EPC Commands	-	-	**SIN_STATE_ARB**
No RF Power	-	-	**SIN_STATE_READY**

19.17 Present State: SIN_STATE_MAPROC

Description:
SINIAV kicks a tag into processing after it sends the Mutual_Auth_Implicit Command to a tag in SIN_STATE_WAIT. The tag stays here until it finishes preparing the response for MA_Implicit.

Security:

Procedure:
Oh boy. This is the guy who does all the verification and processing of the MAPROC command.

Entry States:	Exit States:
• SIN_STATE_WAIT	• SIN_STATE_MARDY
Pre:	**Post**
• A MA_IMPLICIT command was received during SIN_STATE_WAIT	• The response to MA_Implicit is prepared and loaded into the transmit buffer. • If RSU authentication failed, an error message was loaded into buffer instead.

State Transitions:

Event	Condition	Action	Next State
Finished Processing	RSU was authenticated	Buffer prepped with response	**SIN_STATE_MARDY**
	otherwise	Buffer prepped with error	**SIN_STATE_ARB**
No RF Power	-	-	**SIN_STATE_READY**

Data Accessed:
TBD

20 SINIAV Commands

As the SINIAV protocol disjointly presented the new SINIAV commands, the following tables are assembled by the Seattle Team representing the SINIAV commands and responses, but formatted in the same was as ISO 18000-6C. Additionally the interpretation of SINIAV handling for EPC C_1G_2 commands is discussed here as well.

20.1 Select

20.2 Query

EPC Command is same as found in EPC C1G2, response is below:

Table 18: *Query Command (22 bits)*

	Command	DR	M	Trext	Sel	Session	Target	Q	CRC-5
# of bits	4	1	2	1	2	2	1	4	5
description	1000	1' for DR=64/3	00' for FM0	0: No pilot tone 1: Use pilot tonge	00: All 01: All 10: ~SL 11: SL	00: S0 01: S1 10: S2 11: S3	0: A 1: B	0-15	ignored

Table 19: *Tag Response to a Query Command (16 Bits)*

	EPC Handle
# of bits	16
description	RN16, new EPC Handle

20.3 QueryRep

20.4 QueryAdj

20.5 ACK

There is a deviation in the Siniav protocol here. EPC ACK response and Siniav ACK response are different. See below.

Table 20: *Ack Command (18 bits)*

	Command	EPC Handle
# of bits	2	16
description	01	Echoed RN16 or handle

Table 21: *Tag Response to an Ack Command (96 Bits)*

	EPC Handle	PC	SPC	GID	CRC16
# of bits	16	16	24	24	16
description	epc Handle	same as received TC	SINIAV Ctrl Prot.	Group ID	CRC16

Table 22: *What a Tag Response would look like to an EPC Ack Command (128 Bits)*

	PC	EPC	CRC-16
# of bits	16	96	16
description	epc Handle	same as received TC	CRC16

20.6 Req_RN

20.7 Read

20.8 Write

20.9 Remaining EPC C_1G_2 Commands

20.10 Siniav MLD Field for SAR and SAW

The MLD field(64bits) describes what memory fields are requested in an SAR and SAW command.

Name	Bit Index	Length	Default Val	Descrip	Semantics
Version	0	2	0b00	vers of MLD	-
OBUMemBank	2	2	-	memBank	RES/TID/UII/MEMBANK
MBWordPtr	4	16	-	wordPtr	where to grab words
MBWordCount	20	8	-	wordCt	how many to grab
MLDDMD	28	2	0b00	DMAC?	should OBU do DMAC?
MLDRFFU	30	2	0b00	RFFU	-
MLDCRC	32	16	-	CRC16	crc over bits 0-31
WDCRC	48	16	0x0000	CRC16	crc over SAW payload

20.11 ACK (mandatory)

EPC Command is same as found in EPC C1G2, response is below.

On reception of the RSU ACK command, the OBU will set the Siniav Handle equal to the previous EPC Handle. E.g. on entry to this command if the OBU's epc handle (RN16) was 0x1234, on exit the OBU's siniav handle is also now 0x1234.

Table 23: *ACK Command (18 bits)*

	Command	RN16
# of bits	2	16
description	01	Echoed RN16 or handle

Table 24: *Tag Response to an ACK Command (96 Bits)*

	EPC Handle	PC	SPC	GID	CRC16
# of bits	16	16	24	24	16
description	epc Handle	same as received TC	SINIAV Ctrl Prot.	Group ID	CRC16

20.12 Req_Handle (mandatory)

Description to be completed.

Table 25: *Req_Handle Command (57 bits)*

	Command	Handle	TC	OSM	CRC
# of bits	16	16	1	8	16
description	**1110**0000 00000000	Siniav Handle	Tx Counter	See Table X Below	

Table 26: *OSM Field Description*

Field	OSM			
Subfield	RFFU	CAT	SS	IFS
# of bits	1	3	2	2

Table 27: *Tag Action to a Req_Handle Command*

CAT	SS	IFS	Action
000	00: S0	00	do nothing
	01: S1	01	invFlag -> A
	10: S2	10	invFlag -> B
	11: S3	11	invFlag toggle
001	00: S0		do nothing
	01: S1	01: invFlag == A	EPC Reset
	10: S2	10: invFlag == B	EPC Reset
	11: S3		do nothing
111	-	-	EPC Reset

Table 28: *Tag Response to a Req_Handle Command (35bits)*

	Header	Siniav Handle	TC	RFFU	CRC
# of bits	1	16	1	1	16
description	0'	new SINIAV Handle	same as received TC		

20.13 Mutual_Auth_Implicit (mandatory)

Description to be completed.

Table 29: *Mutual_Auth_Implicit Command (180 bits)*

	Command	Siniav Handle	TC	RFFU	DECBAK()	CRC
# of bits	16	16	1	3	128	16
description	11100000 00000010	Siniav Handle	Tx Counter		See Table X Below	CRC16

Table 30: *$D^{ECB}AK()$ Field Description*

Field	DECBAK()					
Subfield	**R64**	**CR56**	**SMD**	**DMD**	**GSK**	**RFFUP**
# of bits	64	56	2	2	1	3

Table 31: *Tag Response to a Mutual_Auth_Implicit Command (546 bits)*

	Header	Siniav Handle	TC	EECBAK()	EECBSK()	ECTRSK()	DMAC	CRC
# of bits	1	16	1	128	128	256	0	16
description	0'	Siniav Handle	same as received TC				curr siniav doesn't use	

Table 32: *$E^{ECB}AK()$ Field Description*

Field	EECBAK()	
Subfield	**T64**	**CT64**
# of bits	64	64

Table 33: *$E^{ECB}SK()$ Field Description*

Field	EECBSK()	
Subfield	**R64**	**T64^DCRC**
# of bits	64	64

Table 34: *$E^{CCTR}SK()$ Field Description*

Field	ECTRSK()		
Subfield	**OBU-ID**	**DATA64**	**SMAC**
# of bits	64	64	128

20.14 Secure_Auth_Read (mandatory)

Table 35: *Secure_Auth_Read Command (180 bits)*

	Command	Siniav Handle	TC	RFFU	DECBSK()	CRC
# of bits	16	16	1	3	128	16
description	11100000 00000011	Siniav Handle	Tx Counter		See Table X Below	CRC16

	Command	Siniav Handle	TC	RFFU	DECBSK()	CRC
# of bits	16	16	1	3	128	16
description	11100000 00000011	Siniav Handle	Tx Counter		See Table X Below	CRC16

Table 36: *$D^{ECB}SK()$ Field Description*

Field	DECBSK()	
Subfield	**T64**	**R64^MLD**
# of bits	64	56

Table 37: *Tag Response to a Secure_Auth_Read command (variable bits)*

	Header	Siniav Handle	TC	EECBSK()	ECTRSK()	DMAC	CRC
# of bits	1	16	1	128	128*N	0	16
description	0	Siniav Handle	same as received TC		N is number of blocks requested	curr siniav doesn't use	

Table 38: *$E^{ECB}SK()$ Field Description*

Field	EECBSK()	
Subfield	**R64**	**T64^RCRC**
# of bits	64	64

Table 39: *$E^{CTR}SK()$ Field Description*

Field	ECTRSK()		
Subfield	**DW1**	**…**	**DWn**
# of bits	128	128	128

20.15 Secure_Auth_Write (mandatory)

**Emulated Reader only supports writes for 128 bits.

Table 40: *Secure_Auth_Write Command (308/436/564 bits, 3 blocks max)*

	Command	Siniav Handle	TC	DMD	RFFU	DECBWK()	ECTRWK()	DMAC	CRC16
# of bits	16	16	1	2	1	128	128*N	0	16
description	11100000 00000010	Siniav Handle	Tx Counter	00	0	See Table X Below			CRC16

Table 41: *Tag Response to a Secure_Auth_Write command (162 bits)*

	Header	Siniav Handle	TC	EECBWK()	CRC
# of bits	1	16	1	128	16
description	0	Siniav Handle	same as received TC	See Table X Below	

Table 42: *$D^{ECB}WK()$ Field Description*

Field	DECBWK()	
Subfield	T64	R64^MLD
# of bits	64	64

Table 43: *$E^{ECB}WK()$ Field Description*

Field	EECBWK()	
Subfield	R64	T64^WCRC
# of bits	64	64

Table 44: *$E^{CTR}WK()$ Field Description*

Field	ECTRWK()		
Subfield	DW1	...	DWn
# of bits	128	128	128

20.16 Finalize (mandatory)

Description to be completed.

Table 45: *Finalize Command (49 bits)*

	Command	Siniav Handle	TC	CRC
# of bits	16	16	1	16
description	**1110**0000 00000001	Siniav Handle	Tx Counter	CRC16

Table 46: *Tag Response to a Finalize (i.e. the Auxiliary confirmation)(36 bits)*

	Header	Siniav Handle	TC	PSI	CRC
# of bits	1	16	1	2	16
description	0	Siniav Handle	same as received TC	00: in progress 01: complete 10: RFFU 11: RFFU	

20.17 CC-Primary / Response Pair

21 SINIAV Protocol Compliance

Here we illustrate all major deviations of the BTag operation from the Siniav Protocol v1. All Siniav Protocol Requirements not listed here are assumed to be in-compliance by the development team.

This section also provides analysis of the protocol deviations, and states their impact on device operation and behavior.

21.1 (-) PHYS Integrity Verification with CRC

All communication links use integrity checks to verify the contents of delivered data. EPC C1G2 and Siniav use CRC16. Fundamentally, the RISC Von-Nuemann Architecture of our system does not have time to verify the CRC sent on each RSU command within the 15.6us (or 190 clock cycles @ 12MHz). A brief illustration of this limitation:

- Fastest Documented CRC16 algorithm on MSP430 CPU: 2.1cycles/bit
- Cycles to CRC a Siniav MA Implicit Command(546 bits) 1145 cycles

This does not even consider all the program execution, decision making, state changes and data movement the CPU must also do in this short window (190cycles) before the tag has to begin talking back to the reader. To give some more perspective, currently on the Query command *the only way we can respond in time* is by breaking early on the Query command's CRC5.

Impact: MINIMAL TO NONE

Receiving and using a corrupted message at the PHYS level can have the following impacts on a system. It will be shown that within the Siniav System, none of these impacts can cause any permanent damage or compromise the security of the tag:

1. Incorrect State Modification

By design EPC (except tag memory) state only provides the ability to singulate a tag, and is reset to a specific, known state by the reader during inventorying. Also, all NV state is periodically reset to a static, known state in btg_cleanState().

a. During EPC Inventorying (Query/Req_Handle/Finalize)

Only the inventorying variables RN16, Q, slotCount, SL are modified. These are only used for tag inventorying, and are reset by the reader with each inventory round.

Considering an erroneous TC in Req_Handle or Finalize, there is an incredibly small chance that a retransmission may occur, or the reader/tag lose synchronization.

b. During Siniav Secure Commands (MA_Implicit, SAR, SAW)

As MA_Implicit/SAR or MA_Implict/SAW are heavily coupled in encryption and internal CRC checks, and reference to previously transmitted session keys (R64,T64) it is not seen how a successful Siniav secure command handle could occurred in the event of an erroneous bit.

c. During Customization Commands (Read, Block_Write)

There is no secondary level of protection to Reading and Writing during customization. Thus not observing CRC's has the potential for incorrect Read/Write during customization. This will be mitigated sufficiently by the customization protocol in which tag memories are read after they are written to.

2. Incorrect State Transition

By design there are very few state transitions the tag can take other than through the steps of the EPC/Siniav Protocol, to the 'scrub state' or permanently to killed mode. Incorrect

commands as identified in #1 can only cause the OBU to move on to an incorrect EPC/Siniav State, but this would be equivalent to normal operation. An incorrect command cannot trigger killed mode (only self-tests can), except in the event of a false-positive WRITE command to tag memory, which is assumed to be rare or 0%.

3. Incorrect Tag Response
   Incorrect responses to inventorying commands would result in nearly equivalent RSU OBU behavior as if the OBU didn't respond at all; the inventory round would just reset. Outside of #1 and #2 above, there are no incorrect tag responses that can occur from a corrupted message.

Solutions: If the event comes where CRC verification is required, two ideas are proposed

1. Modify Receive Chain of Btag to calculate CRC 'on-the fly'. There is very little time already within the receive chain's critical path. The team would need to in detail revisit the Receive Chain and its timing and very thoroughly it's timing capacity and compliance. A minimalist and heavily optimized CRC routine would also have to be derived.

   However it seems possible to modify the receive chain to calculate CRC on the fly. This will be a significant amount of work, perhaps 7-8 days of dedicated work to design, implement and verify.

2. Don't respond to the first reader command after reception, and directly begin processing/verifying CRC. Hope that the reader retransmits that message to you, on which event you will be ready to transmit back your prepared response.

   This method assumes a whole lot about the EPC and SINIAV reader inventorying procedure. What if the reader doesn't always retransmit? What affect does this have in multi-tag environments? How does this impact the Siniav Interleaved-Inventory scheme? This solution would have a significantly modify the functionality of a Siniav Tag, and its behavior significantly compounded in multi-tag environments

3. Use a HW CRC co-processor. Because CRC is in plaintext this presents no security or hacking risk. A PLD solution is estimated to cost $.70/ea in Qty ~1000. This would not compromise the power consumption as it can be enabled (powered) by a GPIO pin on the MSP430.

21.2 (-) Sessioning of Tags ($S_{0/1/2/3}$)

We only respond to S0. This does limit the ability of a reader to use sessioning, but in Siniav scenario of 122ms window for reading, sessioning outside of S0 probably won't be used anyways.

Impact: MINOR

The only risk here is that the Siniav certification body may raise alarm and require sessioning, or special provisions be made to the firmware.

Solution:

The team estimates they should be able to implement sessioning, albeit with significant added complexity into the timing-sensitive receive chain modules. This will take approximately 7 full days of work to fix.

21.3 (-) SMAC Compliance for MA_Implicit

The OBU only responds to MA_Implicit commands which request data that includes the SMAC. As the BTag uses a fixed-message length procedure for assembling the MA_Implicit command, this would require restructuring the command to be dynamically sized.

Impact:
If reader's requests no SMAC (SMD==0x00), perhaps in high-throughput regions, the tag will respond with an error code instead of a MA_Implicit response. This would decrease the compliance of the BTag.

Solution:
Change the MA_Implicit Response to dynamic length format. While at it, restructure SAR and SAW also using same framework with similar naming/style conventions.

21.4 (-) EPC Reads assume fixed EBV Length (WordPtr)

We assume the EBV field to be one-byte in length to relax command parsing requirements.

Impact:
If a reader requests a read using a EBV of differing length, then we will not be able to correctly handle the response. However multi-byte EBV's are only for tag's with large memory maps, a category in which SINIAV tags do not come close to. Also, the Siniav G0 specification explicitly states that communication use 1-byte EBV fields.

22 Supplements to SINIAV Protocol

- Periodic self-checks and power-up checks that may result in killing a tag

- A killed mode which responds with empty EPC

- siniav.TC is set upon the reception of each matching Req_Handle command (ref. 8.1.2)
  - the spec is ambiguous about what to do if reader and tag are out of sync on TC

- The OBU's Siniav Handle is assigned the value of the OBU's previous EPC Handle (RN16) on each successful ACK Command. Each time the Siniav handle is updated, the RFID handle gets updated too.
  - This was a design decision to maintain two separate states for RFID and Siniav. This makes sense because the tag has different modes (e.g. init and customization) which do not have siniav parameters. It is nice for encapsulation and scalability reasons to isolate the two handles
  - Also, siniav.handle only ever gets modified in ACK and Req_Handle, so this doesn't have complicated design ramifications.

- The BTag does not process any RSU message CRCs for validation
  - Our receive architecture needs to break early from a message to process just to get a message out the door in time. And then you want me to process it too? Don't think so, nice try…

- Fixed Read Lengths for the EPC READ Command. They still allow for accessing the entire memory contents, but just in predefined block sizes.

- We use SK for CMAC calculations. The implemented CMAC is selected as NIST SP800-38B (see sec 7.4 of Siniav Protocol for discussion).

- The firmware is architected to only handle SPV = 1. It will ignore any command not matching SPV = 1.
- For SAW, in the case where new keys are written, all Rx/Tx computations are performed with previous key values. The next handled command will begin using the newly written key values.
- Section 18.10, Siniav Transmission Counter (TC) on page 45 describes the BTag's explicit interpretation of TC handling. This should be in compliance with what they are requesting, but the periphery details may have been misinterpreted.

23 Future Optimizations

23.1 Drop Clock Speeds

- Set the receive chain to a lower clock frequency to save power. Maybe could decrease to 8MHz?
- Set the crypto-processing to a lower clock frequency, up to the max crypto-processing time the RSU will allow. This will save power too.
- Switch to an MCU with a continuous TX usart. This will solve the transmit module's 9 cycle bottleneck.

23.2 Add Optional Features

- Support GSK generation

Todos / Dev Notes

- Just go back and thoroughly make sure no command is handled if handle doesn't match.
- Need to make sure all command handling options goto the correct state while in Siniav mode. For example, in wait if I get a Req_RN, should I go back to arb? Should I ignore it? Should I check to see if it matches my RN16. For now I just go back to arb, but this seems limiting. Justin 3.3.11
- What was the max number of bits set to in the code on Rx? Need to recheck that
- EPC T2 Timeout is still set really high. Bring it down when I get a chance.
- Remove Handling of ReqRN command from all Siniav states. It is supposed to ignore them. Also updated state transition tables in doc.
- Make sure error codes are handled for TC errors
- Make sure retransmission attempts are handled
- EPC ACK Response and Siniav ACK Response aren't compatible. Right now we do EPC ACK Response. Wait until Siniav Reader comes todo the Siniav ACK Response (4/4/11, jmr)
- EPC T2 timeout has been disabled during Emulated Reader Development. Make sure it is re-enabled once real reader comes
- I think we may not be handling QueryRep/QueryAdj right; are these supposed to be ignored if we aren't in an inventoried round? I think this is to be the case, but it isn't implemented that way. Just realized recently.
    - EPC Spec doesn't explicitly say this, but it is inferred by vocabulary.
- SAW is only verified for writing of one block. This is due to the lack of a Siniav reader, and that the emulated reader only supports 1 block writes. Also consider the verification of write data padding!
- Tag has not been verified for multi-block reads or writes. If no reader still comes in, consider making emulated reader support multi-block and non-standard length blocks. This would be a significant amount of work (3-4 days).
- Make sure port IO is set as outputs, in correct direction in code before final release happens. Not doing it right now for dev purposes.
- Go back and update state transition tables, in depth. It's been awhile.
- Retransmissions don't have timing fine-tuned. Also probably a good idea to go back and hand-tune/doc response timing for all commands.
- USE WATCHDOG TO IMPLEMENT SESSIONING. SHEESH.
    - Add third way to wakeup, (break-up setup_0)
- Mention to Terry/Josh/Alanson that we should verify operating CPU frequency and temperature reporting when assembled.
    - CPU for obvious timing reasons

  - Temp because tag kills itself in (gross) over-under scenarios
- Ensure all ISRs within the Receive State Machine have an exit ability.
- Recount Critical Path Timing for receiving a dataBit. Document. Do this during cleaning of Receive Chain.
- Noticed that sometimes the receive chain sleeps with the receive comparator on. Check this.
- Discuss flip-flop of queryAdj handling.
- Mention 'in inventory' discarding.
- Analyze uniform entry timing of the handles
- For EPC Write / Blockwrite, what is the timeout? Am I supposed to write, then transmit response back?
  - Once EPC Write is actually insitu (gosh…) I can test this and resolve it.

Open Issues

- OBSERVATION: At the limitations of input power (i.e. ~26dBm), the BTag's performance has hysteresis with regards to whether it had come into increasing RF field, or if it was leaving.
  - Read-threshold from 0W: --21dBm
  - Read-threshold from 30dBm: -26dBm
  - Is this a reader thing, or a tag thing? Need to investigate
- OBSERVATION: during SATO multi-week battery testing, RSSI had long-periods where the RSSI reports very low (see pic below).
  - Is this a radio issue? I don't want to reset the tag because I want the firmware to not reset during the SATO test

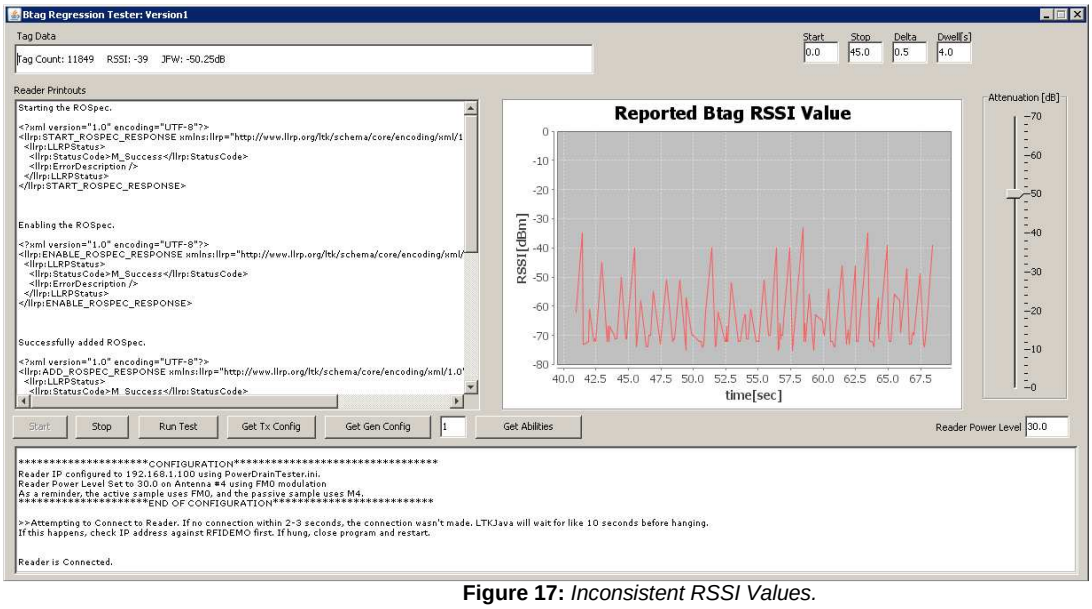


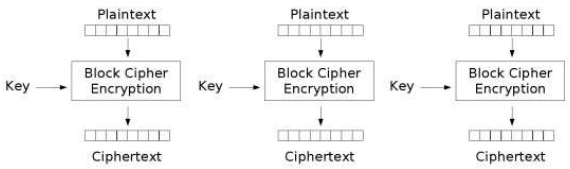
Figure 17: *Inconsistent RSSI Values.*

- There may be an issue with receive chain where if start too early (i.e. within current tx) it grabs wrong first bit. Noticed in read function. This shouldn't be this way though, I specifically designed against it?
- Orientation is sensitive?

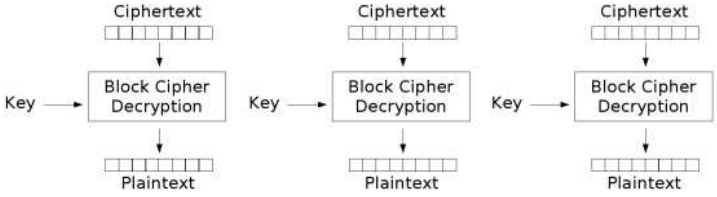
Crypto Module Definitions

Electronic Code Book (ECB): Standard use of block encryption, where:

$$cipherText = E_{ECB}^{KEY}(plaintext)$$



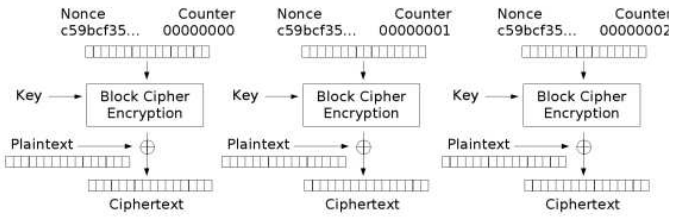
Electronic Codebook (ECB) mode encryption



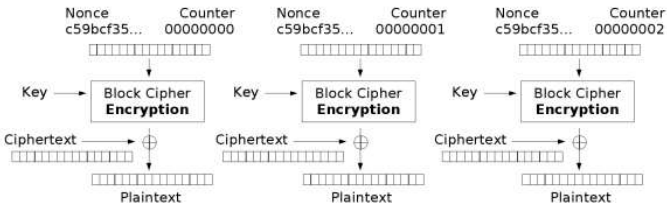
Electronic Codebook (ECB) mode decryption

Counter Mode (CTR): Encrypting the cover and XORing into datastream

$$cipherText = plaintext \wedge E_{CTR}^{KEY}(CTR{+}{+})$$



Counter (CTR) mode encryption



Counter (CTR) mode decryption

CMAC(for even blocks): NIST SP800-38B

The following is a specification of the subkey generation process of CMAC:

1. Let $L = CIPH_K(0^b)$.
2. If $MSB_1(L) = 0$, then $K1 = L << 1$;
   Else $K1 = (L << 1) \oplus R_b$; see Sec. 5.3 for the definition of R_b.
3. If $MSB_1(K1) = 0$, then $K2 = K1 << 1$;
   Else $K2 = (K1 << 1) \oplus R_b$.

The following is a specification of the MAC generation process of CMAC:

1. Apply the subkey generation process in Sec. 6.1 to K to produce $K1$ and $K2$.
2. If $Mlen = 0$, let $n = 1$; else, let $n = \lceil Mlen/b \rceil$.
3. Let $M_1, M_2, \ldots, M_{n-1}, M_n$
   * denote the unique sequence of bit strings such that $M = M_1 \| M_2 \| \ldots \| M_{n-1} \| M_n$
   *, where $M_1, M_2, \ldots, M_{n-1}$ are complete blocks.2
4. If M_n
   * is a complete block, let $M_n = K1 \oplus M_n$

$_*$; else, let $M_n = K2 \oplus (M_n$
$_*\|10^j)$,
where $j = nb\text{-}Mlen\text{-}1$.
5. Let $C_0 = 0^b$.
6. For $i = 1$ to n, let $C_i = \text{CIPH}_K(C_{i-1} \oplus M_i)$.
7. Let $T = \text{MSB}_{Tlen}(C_n)$.
8. Return T.

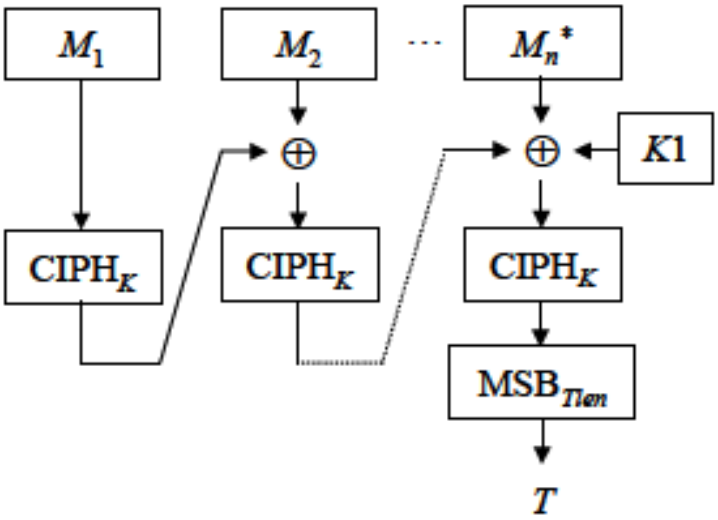


Figure 18: *CMAC Generation for more than one block.*

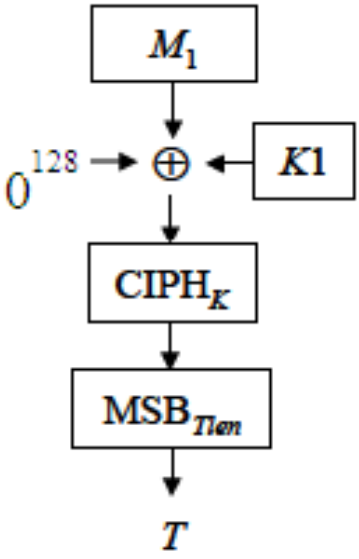


Figure 19: *Single Block CMAC Generation.*

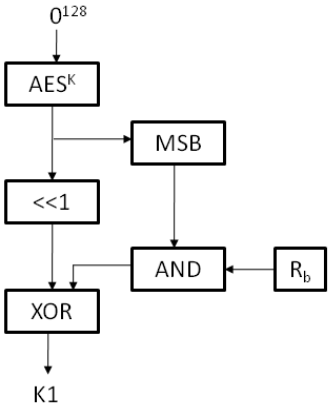


Figure 20:i CMAC Subkey Generation for a single block (e.g. SMAC)

Example CRC16 Calculations
Example CRC5 Calculations
Example CMAC Calculations
Example AES-128 Calculations
Example CTR Mode Calculations

Emulated SINIAV Reader Development

**Emulated Reader only supports 128 bit writes to save design complexity.

Tag Data Memory (U64)

Name	Length	Start Index	Stop Index	Descrip
Seq ID	1	0	0	Tag Seq Value
Status	1	1	1	Status of Response
Timestamp	1	2	2	microsends
SAMode	1	3	3	(0/1/2)==(MA/SAR/SAW)
EPCHandle	1	4	4	epc id
SinHandle	1	5	5	siniav id
EPC	2	6	7	ACK response w/o CRC
MA Implicit Rsp	12	8	19	full response to MA Impl
SA Rsp	22	20	41	full response to SAR/W

Tag Data Message sent in TagData FIFO (FPGA to Host)

Name	Length	Start Index	Stop Index	Descrip
1^32|(Seq ID)	1	0	0	(1^64)|Tag Seq Value
Status	1	1	1	Status of Response
Timestamp	1	2	2	microsends
SAMode	1	3	3	(0/1/2)==(MA/SAR/SAW)
EPCHandle	1	4	4	epc id
SinHandle	1	5	5	siniav id
EPC	2	6	7	ACK response w/o CRC
MA Implicit Rsp	12	8	19	full response to MA Impl
SA Rsp	22	20	41	full response to SAR/W

*1^32 == 0xFFFFFFFF and is the synchronization bits.

- The emulated reader sends 1^64 as the value of T64. It would require significant modifications otherwise.
- The emulated reader reinitializes the CTR value to 00/01/../0F for the SAW command.

Remaining Unknowns

This set is current as of 3.23.11.

- Required Activity Profile
- Security Certification Procedure
- To be filled out.
- Crypto Processing Time Delta-T
- Which key is used for CMAC generation?
- For the CTRmode CTRVal, is it reset upon every command, or does it keep a running coutner?
    - I think this is buried in the spec somewhere…
    - We keep a running counter value.
- Todo: consolidate all previous lists here.
- **Question for Code Review**: How should I treat the following concepts for a product like the BTag
    - FLASH corruption
        - For lookup tables in AES and CRC, if 1 bit gets thrown off then the whole thing is ruined! But I guess the same goes for program flash too…
    - RAM corruption
    - Loops that don't timeout

Verification Data for a SAR Transaction

This section represents (3) different test vectors for verifying the protocol compliance of either a Siniav RSU (Reader) or OBU (tag).

This data set represents Intel's understanding of the Authentication process and algorithms required by the Siniav Protocol.

1 Notation and Definitions

All data is described in hex notation unless it is not a multiple of 4 bits. This is not normally done, but this document describes large quantities of binary data, and this will increase reliability.

Example 1: 01010001_2 is denoted : 51_h
Example 2: 0101000111_2 is denoted: 51_h11_2

Data is presented MSB first. In 'Example 1' above, $b_0 = 0$, $b_1 = 1$.

OBU: On-board Unit. Name for the Siniav RFID Tag.
RSU: Road-Side Unit. Name for the Siniav RFID Reader.
MA_Impl: Mutual Authentication Implicit command in Siniav Protocol.
SAR: Secure-Access Read command in Siniav Protocol
SAW: Secure-Access Write command in Siniav Protocol

2 Suggested Response by Reader Company to This Document

Minimum

- Verify bit field interpretations and locations in each table of this document
- Verify the SMAC calculation performed in MA_Impl
- Verify the link timing is acceptable in Section 3

Target

- Verify use of counter mode, and when the counter-mode's counter value is incremented
- Verify Transmission Counter (TC) management

Ideal

- Verify that the link-layer data bits displayed in Section 3 will produce a successful Siniav Transaction for MA_Impl, SAR and SAW, and that an RFID tag meeting these link-layer requirements will be treated as a successful transaction.
    - For example, if there were a MATLAB script, or validation test bench with which to equip with this data.
    - Or, For Example, if it were possible to directly inject this test vector into the live system, and verify its correctness.

3 Summary of Siniav Transaction

The following presents a single SINIAV transaction between RSU and OBU for either MA_Impl, SAR or SAW. This section includes link-layer timing and raw message bits. For an explanation of each set of message bits, read the remaining sections of the document.

Valid sequences in the context of this verification document are:

(Mutual Authentication Only)	Stage 1, Stage 2, Stage 3	Δt = 15.5ms
(Secure Access Read)	Stage 1, Stage 2, Stage 3, Stage 4	Δt = 10.5ms
(Secure Access Write)	Stage 1, Stage 2, Stage 3, Stage 5	Δt = 31.0ms

**Timing values are for single block read or write.*

3.1 EPC Singulation

Pre: Idle RF

Stage 1: RSU Enables RF Signal for >30us as CW (i.e. no modulation)

Stage 2i: RSU sends a Query command as:
[MSB] $886A2_h01_2$ **[LSB]**

Stage2ii: 20.0us later the OBU responds to Query with:
[MSB] 1234_h **[LSB]**

Stage2iii: 20.0us later the RSU sends an ACK command as:
[MSB] $886A2_h01_2$ **[LSB]**

Stage2iv: 20.0us later the OBU responds to ACK with:
[MSB] $12343400ABCDEF0123458528_h$ **[LSB]**

Stage2v: 20.0us later the RSU sends a Req_Handle command as:
[MSB] $E0001234047939_h0_2$ **[LSB]**

Stage2vi: 20.0us later the OBU responds to Req_Handle with:
[MSB] $09AB9EED_h110_2$ **[LSB]**

3.2 Mutual Authentication

Stage3i: 20.0us later the RSU sends an MA_Impl command as:
[MSB] $E00213578F8E038106410328DFD5E2C93C61FDF2ED0B8_h$ **[LSB]**

Stage3ii: 20.0us later the OBU responds to MA_Impl with:
[MSB] "Standard Reply-Auxillary Command(Sec.14)"**[LSB]**

Stage3iv: 22.0ms later the RSU sends a Finalize command as:
[MSB] "Standard CC-Primary Command(Sec.14)"**[LSB]**

Stage3v: 20.0us later the OBU responds to Finalize_MAImpl with:
[MSB] 09ABDEB283F66F35BB1F27E5D19985B9
A8A08A0104CC53E9D422B2BFAD56E622
24BD6CCDEE46216B5D0417426562B465
F79890841FB52AF23758BD0F9FAD4973
$1E0A8711_h10_2$ **[LSB]**

3.3 Secure Access ([4]Read or [5]Write)

Stage4i: 20.0us later the RSU sends a SAR command as:
[MSB] E003135708891536B395AB94814ACCA5587D997F13122$_h$ **[LSB]**

Stage4ii: 20.0us later the OBU responds to SAR with:
[MSB] "Standard Reply-Auxillary Command(Sec.14)"**[LSB]**

Stage4iv: 40.0ms later the RSU sends a Finalize command as:
[MSB] "Standard CC-Primary Command(Sec.14)"**[LSB]**

Stage4v: 20.0us later the OBU responds to Finalize_SAR with:
[MSB] 09AB939C703FEE7CEAD748ABB2D311C0
95A494C9FD75BA2F2F3DC9929571C865AAC46607$_h$ [LSB]

Stage5i: 20.0us later the RSU sends a SAW command as:
[MSB] E0041357081314EE74693D2A78CD418C
CF6C159368BF42A0E0E213571D2A5A63
3DB674EEBC5E1$_h$ **[LSB]**

Stage5ii: 20.0us later the OBU responds to SAW with:
[MSB] "Standard Reply-Auxillary Command(Sec.14)"**[LSB]**

Stage5iv: 40.0ms later the RSU sends a Finalize command as:
[MSB] "Standard CC-Primary Command(Sec.14)"**[LSB]**

Stage5v: 20.0us later the OBU responds to Finalize_SAW with:
[MSB] 09ABB4DE198110DE3BE2A6B95D63C2A6FB1CBFB0$_h$10$_2$ **[LSB]**

**The timing values in Section 3 do not have to be exact, but do represent a condition in which the Intel tag will successfully operate.*

**22/40ms for Finalize response is a generalized upper bounds. Actual performance is less than 22/40ms.*

4 Pre-Conditions

Preliminary to a SINIAV Transaction, the data below represents the OBU and RSU's initial state. The description of each variable below is brief.

It is recommended to first visit the remainder of this document to first to gain an understanding of what each variable is. Come back to this section for confirmation of its value.

R64(RSU): **[MSB]** ABCDEFABCDEF0123$_h$ **[LSB]**
CR56(RSU): **[MSB]** 01230123012301$_h$ **[LSB]**

T64(OBU): **[MSB]** 0001020304050607$_h$ **[LSB]**
CT64(OBU): **[MSB]** 08090A0B0C0D0E0F$_h$ **[LSB]**

OBU-ID(OBU): **[MSB]** 272C31363B40454A$_h$ **[LSB]**

DATA64(OBU): **[MSB]** $4F54595E63686D72_h$ **[LSB]**
SMAC(OBU): **[MSB]** E604757761A6BED47B1D89BDC8AF9362 **[LSB]**
DCRC(OBU): **[MSB]** $8FF4_h$ **[LSB]**

SK(Both): **[MSB]** $101112131415161718191A1B1C1D1E1F_h$ **[LSB]**
AK(Both): **[MSB]** $000102030405060708090A0B0C0D0E0F_h$ **[LSB]**
WK(Both): **[MSB]** $F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF_h$ **[LSB]**

Siniav Handle: **[MSB]** 1357_h **[LSB]**

5 Query – Reader to Tag & Tag to Reader

Query command sent by RSU to OBU, and response by OBU.

Table 47: *Query Command (22 bits)*

	Command	DR	M	Trext	Sel	Session	Target	Q	CRC-5
# of bits	4	1	2	1	2	2	1	4	5
description	1000	1' for DR=64/3	00' for FM0	0: No pilot tone 1: Use pilot tonge	00: All 01: All 10: ~SL 11: SL	00: S0 01: S1 10: S2 11: S3	0: A 1: B	0-15	ignored

Table 48: *Tag Response to a Query Command (16 Bits)*

	EPC Handle
# of bits	16
description	RN16, new EPC Handle

Bit Definitions for the Query Command Fields

Command: **[MSB]** 8_h **[LSB]**
DR: **[MSB]** 1_2 **[LSB]**
M: **[MSB]** 00_2 **[LSB]**
Trext: **[MSB]** 0_2 **[LSB]**
Sel: **[MSB]** 01_2 **[LSB]**
Session: **[MSB]** 10_2 **[LSB]**
Target: **[MSB]** 1_2 **[LSB]**
Q: **[MSB]** 4_h **[LSB]**
CRC-5: **[MSB]** $\mathbf{4_h1_2}$ **[LSB]**

Bit Definitions for the Query Response Fields

RN16: **[MSB]** 1234_h **[LSB]**

6 Final Composed Message from Reader to Tag (22 bits)

[MSB] $886A2_h01_2$ **[LSB]**

7 Final Composed Message from Tag to Reader (16 bits)

[MSB] 1234_h **[LSB]**

ACK - Reader to Tag & Tag to Reader

ACK command sent by RSU to OBU, and response from OBU.

Table 49: *Ack Command (18 bits)*

	Command	EPC Handle
# of bits	2	16
description	01	Echoed RN16 or handle

Table 50: *Tag Response to an Ack Command (96 Bits)*

	EPC Handle	PC	SPC	GID	CRC16
# of bits	16	16	24	24	16
description	epc Handle	same as received TC	SINIAV Ctrl Prot.	Group ID	CRC16

Table 51: *What a Tag Response would look like to an EPC Ack Command (128 Bits)*

	PC	EPC	CRC-16
# of bits	16	96	16
description	epc Handle		CRC16

Bit Definitions for the ACK Command Fields

Command: **[MSB]** 01_2 **[LSB]**
EPC Handle: **[MSB]** 1234_h **[LSB]**

Bit Definitions for the ACK Response Fields

EPC Handle: **[MSB]** 1234_h **[LSB]**
PC: **[MSB]** 3400_h **[LSB]**
SPC: **[MSB]** $ABCDEF_h$ **[LSB]**
GID: **[MSB]** 012345_h **[LSB]**
CRC16: **[MSB]** 8528_h **[LSB]**

8 Final Composed Message from Reader to Tag (22 bits)

[MSB] $886A2_h01_2$ **[LSB]**

9 Final Composed Message from Tag to Reader (24 bits)

[MSB] $12343400ABCDEF0123458528_h$ **[LSB]**

10 Req Handle – Reader to Tag & Tag to Reader

Req Handle command sent by RSU to OBU, and response from OBU.

Table 52: *Req_Handle Command (57 bits)*

	Command	Handle	TC	OSM	CRC
# of bits	16	16	1	8	16
description	**1110**0000 00000000	previous EPC Handle OR current Siniav Handle	Tx Counter	See Table 7 Below	

Table 53: *OSM Field Description*

Field	OSM			
Subfield	RFFU	CAT	SS	IFS
# of bits	1	3	2	2

Table 54: *Tag Response to a Req_Handle Command (35bits)*

	Header	Siniav Handle	TC	RFFU	CRC
# of bits	1	16	1	1	16
description	0'	new SINIAV Handle	same as received TC		

Bit Definitions for the Req Handle Command Fields

Command:	**[MSB]**	$E000_h$	**[LSB]**
EPC Handle:	**[MSB]**	1234_h	**[LSB]**
TC:	**[MSB]**	0_2	**[LSB]**
OSM:	**[MSB]**	08_h	**[LSB]**
RFFU:	**[MSB]**	0_2	**[LSB]**
CAT:	**[MSB]**	000_2	**[LSB]**
SS:	**[MSB]**	10_2	**[LSB]**
IFS:	**[MSB]**	00_2	**[LSB]**
CRC:	**[MSB]**	$F272_h$	**[LSB]**

Bit Definitions for the Req Handle Response Fields

Header Bit:	**[MSB]**	0_2	**[LSB]**
Siniav Handle:	**[MSB]**	1357_h	**[LSB]**
TC:	**[MSB]**	0_2	**[LSB]**
RFFU:	**[MSB]**	0_2	**[LSB]**
CRC16:	**[MSB]**	$F76E_h$	**[LSB]**

11 Final Composed Message from Reader to Tag (57 bits)

[MSB] $E0001234047939_h0_2$ **[LSB]**

12 Final Composed Message from Tag to Reader (35 bits)

[MSB] $09AB9EED_h110_2$ **[LSB]**

13 MA Implicit – Reader to Tag

Mutual Authentication Implicit Command from Reader to Tag, Decomposed by Field:

Table 55: *Mutual_Auth_Implicit Command (180 bits)*

	Command	Siniav Handle	TC	RFFU	DECBAK()	CRC
# of bits	16	16	1	3	128	16
description	11100000 00000010	Siniav Handle	Tx Counter		See Table 10 Below	CRC16

Table 56: *$D^{ECB}AK()$ Field Description (128 bits)*

Field	DECBAK()					
Subfield	**R64**	**CR56**	**SMD**	**DMD**	**GSK**	**RFFUP**
# of bits	64	56	2	2	1	3

Final Bit Definitions for the MA_Impl Command Fields

Command: **[MSB]** $E002_h$ **[LSB]**
Siniav Handle: **[MSB]** 1357_h **[LSB]**
TC: **[MSB]** 1_2 **[LSB]**
RFFU: **[MSB]** 000_2 **[LSB]**
DECBAK(): **[MSB]** $F8E038106410328DFD5E2C93C61FDF2E_h$ **[LSB]**
  R64: **[MSB]** $ABCDEFABCDEF0123_h$ **[LSB]**
  CR56: **[MSB]** 01230123012301_h **[LSB]**
  SMD: **[MSB]** 01_2 **[LSB]**
  DMD: **[MSB]** 00_2 **[LSB]**
  GSK: **[MSB]** 0_2 **[LSB]**
  RFFUP: **[MSB]** 000_2 **[LSB]**
CRC: **[MSB]** D0B8 **[LSB]**

14 Intermediate Calculation Values Used to Get Final Bit Definitions

DECBAK(), initial value (R64|CR56|SMD|DMD|GSK|RFFUP):
[MSB] $ABCDEFABCDEF01230123012301230140_h$ **[LSB]**

DECBAK(), after decryption call using AK:
[MSB] $F8E038106410328DFD5E2C93C61FDF2E_h$ **[LSB]**

15 Final Composed Message From Reader to Tag (180 bits)

[MSB] $E00213578F8E038106410328DFD5E2C93C61FDF2ED0B8_h$ **[LSB]**

16 MA Implicit – Tag to Reader

Tag response to a Mutual Authentication Implicit Command. This would be the response sent to a Reader's Finalize Command, in accordance with the 'Two-Phase Custom Command Execution Scheme' of Section 9.1 of the Siniav Protocol.

Table 57: *Tag Response to a Mutual_Auth_Implicit Command (546 bits)*

	Header	Siniav Handle	TC	EECBAK()	EECBSK()	ECTRSK()	DMAC	CRC
# of bits	1	16	1	128	128	256	0	16
description	0'	Siniav Handle	same as received TC				curr siniav doesn't use	

Table 58: *$E^{ECB}AK()$ Field Description*

Field	EECBAK()	
Subfield	T64	CT64
# of bits	64	64

Table 59: *$E^{ECB}SK()$ Field Description*

Field	EECBSK()	
Subfield	R64	T64^DCRC
# of bits	64	64

Table 60: *$E^{CCTR}SK()$ Field Description*

Field	ECTRSK()		
Subfield	OBU-ID	DATA64	SMAC
# of bits	64	64	128

Final Bit Definitions for the MA_Impl Response Fields

```
Header Bit:        [MSB] 0₂                                  [LSB]
Siniav Handle:     [MSB] 1357ₕ                               [LSB]
TC:                [MSB] 1₂                                  [LSB]
EECBAK():          [MSB] 7ACA0FD9BCD6EC7C9F97466616E6A282ₕ   [LSB]
  T64:             [MSB] 0001020304050607ₕ                   [LSB]
  CT64:            [MSB] 08090A0B0C0D0E0Fₕ                   [LSB]
EECBSK():          [MSB] 280413314FA7508ACAFEB55B988892F5ₕ   [LSB]
  T64:             [MSB] 0001020304050607ₕ                   [LSB]
  R64:             [MSB] ABCDEFABCDEF0123ₕ                   [LSB]
  DCRC:            [MSB] 8FF4ₕ                               [LSB]
  R64^DCRC:        [MSB] ABCDEFABCDEF8ED7ₕ                   [LSB]
ECTRSK₁():         [MSB] B337B91885AD74105D09958AD197DE62ₕ   [LSB]
  OBU-ID:          [MSB] 272C31363B40454Aₕ                   [LSB]
  DATA64:          [MSB] 4F54595E63686D72ₕ                   [LSB]
ECTRSK₂():         [MSB] 42107ED4ABC8DD62F43E7EB525CC782Aₕ   [LSB]
  SMAC:            [MSB] E604757761A6BED47B1D89BDC8AF9362ₕ   [LSB]
DMAC:              [MSB] (empty)                             [LSB]
CRC:               [MSB] 1C46ₕ                               [LSB]
```

17 Intermediate Calculation Values Used to Get Final Bit Definitions

EECBAK(), initial value (T64|CT64):
[MSB] $000102030405060708090A0B0C0D0E0F_h$ **[LSB]**

EECBAK(), after encryption call using AK:
[MSB] $7ACA0FD9BCD6EC7C9F97466616E6A282_h$ **[LSB]**

EECBSK(),initial value (R64|T64^DCRC):
[MSB] $ABCDEFABCDEF0123ABCDEFABCDEF8ED7_h$ **[LSB]**

EECBSK(),after encryption call using SK:
[MSB] $280413314FA7508ACAFEB55B988892F5_h$ **[LSB]**

$ECTRSK_1$(),initial value (Counter Value [n=0])
[MSB] $08090A0B0C0D0E0F0123012301230100_h$ **[LSB]**

$ECTRSK_1$(),after encryption call using SK:
[MSB] $941B882EBEED315A125DCCD4B2FFB310_h$ **[LSB]**

$ECTRSK_1$(),after XORing in data (OBU-ID|DATA64):
[MSB] $B337B91885AD74105D09958AD197DE62_h$ **[LSB]**

$ECTRSK_2$(),initial value (Counter Value [n=1]):
[MSB] $08090A0B0C0D0E0F0123012301230101_h$ **[LSB]**

$ECTRSK_2$(),after encryption call using SK:
[MSB] $A4140BA3CA6E63B68F23F708ED63EB48_h$ **[LSB]**

$ECTRSK_2$(),after XORing in data (SMAC):
[MSB] $42107ED4ABC8DD62F43E7EB525CC782A_h$ **[LSB]**

Counter-Mode Block Encryption State:

These values represent the interpretation of Siniav Protocol Sections 6.1, 6.2 with regards to how to initialize and use the CTR-Mode counter value.

Counter Seed: **[MSB]** $CT64|CR56|00_h$ **[LSB]**

Counter Value[n=0]: **[MSB]** $08090A0B0C0D0E0F0123012301230100_h$ **[LSB]**
Counter Value[n=1]: **[MSB]** $08090A0B0C0D0E0F0123012301230101_h$ **[LSB]**
Counter Value[n=2]: **[MSB]** $08090A0B0C0D0E0F0123012301230102_h$ **[LSB]**
Counter Value[n=3]: **[MSB]** $08090A0B0C0D0E0F0123012301230103_h$ **[LSB]**
…
Counter Value[n=N]: Counter Value[n=0] + N

18 Final Composed Message from Tag to Reader (546 bits)

[MSB] 09ABDEB283F66F35BB1F27E5D19985B9
A8A08A0104CC53E9D422B2BFAD56E622
24BD6CCDEE46216B5D0417426562B465
F79890841FB52AF23758BD0F9FAD4973
$1E0A8711_h10_2$ **[LSB]**

19 SAR – Reader to Tag

In the single atomic sequence of SAR, the Secure Access Read Message sent from Reader to Tag.

Table 61: *Secure_Auth_Read Command (180 bits)*

	Command	Siniav Handle	TC	RFFU	DECBSK()	CRC
# of bits	16	16	1	3	128	16
description	11100000 00000011	Siniav Handle	Tx Counter		See Table 16 Below	CRC16

Table 62: *$D^{ECB}SK()$ Field Description (128 bits)*

Field	DECBSK()	
Subfield	T64	R64^MLD
# of bits	64	56

Table 63: *MLD Definitions for SAR and SAW (64 bits)*

Name	Bit Index	Length	Default Val	Descrip	Semantics
Version	0	2	0b00	vers of MLD	-
OBUMemBank	2	2	-	memBank	RES/TID/UII/MEMBANK
MBWordPtr	4	16	-	wordPtr	where to grab words
MBWordCount	20	8	-	wordCt	how many to grab
MLDDMD	28	2	0b00	DMAC?	should OBU do DMAC?
MLDRFFU	30	2	0b00	RFFU	-
MLDCRC	32	16	-	CRC16	crc over bits 0-31
WDCRC	48	16	0x0000	CRC16	crc over SAW payload

Final Bit Definitions for the SAR Command Fields

```
Command:          [MSB] E003h                                  [LSB]
Siniav Handle:    [MSB] 1357h                                  [LSB]
TC:               [MSB] 02                                     [LSB]
RFFU:             [MSB] 0002                                   [LSB]
DECBSK():         [MSB] 8891536B395AB94814ACCA5587D997F1h      [LSB]
  T64:            [MSB] 0001020304050608h                      [LSB]
  R64:            [MSB] ABCDEFABCDEF0124h                      [LSB]
  MLD:            [MSB] 30000080C65E0000h                      [LSB]
    Version:      [MSB] 002                                    [LSB]
    OBUMemBank:   [MSB] 112                                    [LSB]
    MBWordPtr:    [MSB] 0000h                                  [LSB]
    MBWordCount:  [MSB] 08h                                    [LSB]
    MLDDMD:       [MSB] 002                                    [LSB]
    MLDRFFU:      [MSB] 002                                    [LSB]
    MLDCRC:       [MSB] C65Eh                                  [LSB]
    WDCRC:        [MSB] 0000h                                  [LSB]
  R64^MLD:        [MSB] 9BCDEF2B0BB10124h                      [LSB]
CRC:              [MSB] 3122h                                  [LSB]
```

***T64,R64 are incremented before use, as illustrated in the listing above.*

20 Intermediate Calculation Values Used to Get Final Bit Definitions

DECBSK (), initial value (T64|R64^MLD):

[MSB] $00010203040506089BCDEF2B0BB10124_h$ **[LSB]**

DECBSK (), after decryption call using SK:

[MSB] $8891536B395AB94814ACCA5587D997F1_h$ **[LSB]**

21 Final Composed Message from Reader to Tag (180 bits)

[MSB] $E003135708891536B395AB94814ACCA5587D997F13122_h$ **[LSB]**

22 SAR – Tag to Reader

In the single atomic sequence of SAR, the Secure Access Read Response sent from Tag to Reader.

Table 64: *Tag Response to a Secure_Auth_Read command (variable bits)*

	Header	Siniav Handle	TC	EECBSK()	ECTRSK()	DMAC	CRC
# of bits	1	16	1	128	128*N	0	16
description	0	Siniav Handle	same as received TC		N is number of blocks requested	curr siniav doesn't use	

Table 65: *$E^{ECB}SK()$ Field Description*

Field	EECBSK()	
Subfield	R64	T64^RCRC
# of bits	64	64

Table 66: *$E^{CTR}SK()$ Field Description*

Field	ECTRSK()		
Subfield	DW1	…	DWn
# of bits	128	128	128

Final Bit Definitions for the SAR Response Fields

```
Header Bit:       [MSB] 0₂                                  [LSB]
Siniav Handle:    [MSB] 1357ₕ                               [LSB]
TC:               [MSB] 0₂                                  [LSB]
EECBSK():         [MSB] 4E71C0FFB9F3AB5D22AECB4C47025692ₕ   [LSB]
  R64:            [MSB] ABCDEFABCDEF0124ₕ                   [LSB]
  T64:            [MSB] 0001020304050608ₕ                   [LSB]
  RCRC:           [MSB] A430ₕ                               [LSB]
  T64^RCRC:       [MSB] 000102030405A238ₕ                   [LSB]
ECTRSK():         [MSB] 5327F5D6E8BCBCF7264A55C72196AB11ₕ   [LSB]
  DW1:            [MSB] 272C31363B40454A4F54595E63686D72ₕ   [LSB]
DMAC:             [MSB] (empty)                             [LSB]
CRC:              [MSB] 981Cₕ                               [LSB]
```

***T64,R64 are incremented before use, as illustrated in the listing above.*

23 Intermediate Calculation Values Used to Get Final Bit Definitions

```
EECBSK (), initial value (R64|T64^RCRC):
      [MSB] ABCDEFABCDEF01240001020304050608ₕ [LSB]

EECBSK (), after encryption call using SK:
      [MSB] 4E71C0FFB9F3AB5D22AECB4C47025692ₕ [LSB]

ECTRSK (), initial value (Counter Value[n=2]):
      [MSB] 08090A0B0C0D0E0F0123012301230102ₕ [LSB]

ECTRSK (), after encryption call using SK:
      [MSB] 740BC4E0D3FCF9BD691E0C9942FEC663ₕ [LSB]
```

ECTRSK (), after XORing the data in (DW_1):
[MSB] 5327F5D6E8BCBCF7264A55C72196AB11$_h$ **[LSB]**

24 Final Composed Message from Reader to Tag (290 bits)

[MSB] 09AB939C703FEE7CEAD748ABB2D311C0
95A494C9FD75BA2F2F3DC9929571C865AAC46607$_h$ [LSB]

25 SAW – Reader to Tag

In the single atomic sequence of SAW the Secure Access Write Message sent from Reader to Tag.

Table 67: *Secure_Auth_Write Command (308/436/564 bits, 3 blocks max)*

	Command	Siniav Handle	TC	DMD	RFFU	DECBWK()	ECTRWK()	DMAC	CRC16
# of bits	16	16	1	2	1	128	128*N	0	16
description	11100000 00000100	Siniav Handle	Tx Counter	00	0	See Table 22 Below			CRC16

Table 68: *$D^{ECB}WK()$ Field Description*

Field	DECBWK()	
Subfield	T64	R64^MLD
# of bits	64	64

Table 69: *$E^{CTR}WK()$ Field Description*

Field	ECTRWK()		
Subfield	DW1	…	DWn
# of bits	128	128	128

Final Bit Definitions for the SAW Command Fields

Command: **[MSB] E004$_h$ [LSB]**
Siniav Handle: **[MSB] 1357$_h$ [LSB]**
TC: **[MSB] 0$_2$ [LSB]**
DMD: **[MSB] 00$_2$ [LSB]**
RFFU: **[MSB] 0$_2$ [LSB]**
DECBWK(): **[MSB]** 81314EE74693D2A78CD418CCF6C15936$_h$ **[LSB]**
  T64: **[MSB]** 0001020304050608$_h$ **[LSB]**
  R64: **[MSB]** ABCDEFABCDEF0124$_h$ **[LSB]**
  MLD: **[MSB]** 30008080DDC6F43D$_h$ **[LSB]**
    Version: **[MSB]** 00$_2$ **[LSB]**
    OBUMemBank: **[MSB]** 11$_2$ **[LSB]**
    MBWordPtr: **[MSB]** 0008$_h$ **[LSB]**
    MBWordCount: **[MSB]** 08$_h$ **[LSB]**
    MLDDMD: **[MSB]** 00$_2$ **[LSB]**
    MLDRFFU: **[MSB]** 00$_2$ **[LSB]**
    MLDCRC: **[MSB]** DDC6$_h$ **[LSB]**
    WDCRC: **[MSB]** F43D$_h$ **[LSB]**
  R64^MLD: **[MSB]** 9BCD6F2B1029F519$_h$ **[LSB]**
ECTRWK(): **[MSB]** 8BF42A0E0E213571D2A5A633DB674EEB$_h$ **[LSB]**
  DW1: **[MSB]** FFFFEEEEDDDDCCCCBBBBAAAA99998888$_h$ **[LSB]**

CRC: **[MSB] C5E1 [LSB]**

***T64,R64 are incremented before use, as illustrated in the listing above.*

26 Intermediate Calculation Values Used to Get Final Bit Definitions

DECBWK (), initial value (T64|R64^MLD):
[MSB] $00010203040506089BCD6F2B1029F519_h$ **[LSB]**

DECBWK (), after decryption call using WK:
[MSB] $81314EE74693D2A78CD418CCF6C15936_h$ **[LSB]**

ECTRWK (), initial value (Counter Value[n=2]):
[MSB] $08090A0B0C0D0E0F0123012301230102_h$ **[LSB]**

ECTRWK (), after encryption call using WK:
[MSB] $740BC4E0D3FCF9BD691E0C9942FEC663_h$ **[LSB]**

ECTRWK (), after XORing the data in (DW_1):
[MSB] $8BF42A0E0E213571D2A5A633DB674EEB_h$ **[LSB]**

27 Final Composed Message from Reader to Tag (308 bits)

[MSB] E0041357081314EE74693D2A78CD418C
CF6C159368BF42A0E0E213571D2A5A63
$3DB674EEBC5E1_h$ **[LSB]**

28 SAW - Tag to Reader

In the single atomic sequence of SAW, the Secure Access Write Response sent from Tag to Reader.

Table 70: *Tag Response to a Secure_Auth_Write command (162 bits)*

	Header	Siniav Handle	TC	EECBWK()	CRC
# of bits	1	16	1	128	16
description	0	Siniav Handle	same as received TC	See Table 25 Below	

Table 71: *$E^{ECB}WK()$ Field Description*

Field	EECBWK()	
Subfield	**R64**	**T64^WCRC**
# of bits	64	64

Final Bit Definitions for the SAR Response Fields

```
Header Bit:       [MSB] 0₂                                                    [LSB]
Siniav Handle:    [MSB] 1357ₕ                                                 [LSB]
TC:               [MSB] 0₂                                                    [LSB]
EECBWK():         [MSB] D37866044378EF8A9AE5758F0A9BEC72ₕ                     [LSB]
  R64:            [MSB] ABCDEFABCDEF0124ₕ                                     [LSB]
  T64:            [MSB] 0001020304050608ₕ                                     [LSB]
  WCRC:           [MSB] 75F1ₕ                                                 [LSB]
  T64^WCRC:       [MSB] 00010203040573F9ₕ                                     [LSB]
CRC:              [MSB] FEC2ₕ                                                 [LSB]
```

***T64,R64 are incremented before use, as illustrated in the listing above.*

29 Intermediate Calculation Values Used to Get Final Bit Definitions

```
EECBWK (), initial value (R64|T64^WCRC):
      [MSB] ABCDEFABCDEF012400010203040573F9ₕ [LSB]

EECBWK (), after encryption call using WK:
      [MSB] D37866044378EF8A9AE5758F0A9BEC72ₕ [LSB]
```

30 Final Composed Message from Reader to Tag (162 bits)

```
      [MSB] 09ABB4DE198110DE3BE2A6B95D63C2A6FB1CBFB0ₕ10₂ [LSB]
```

31 Standard Two-Phase Custom Command Definitions

In reference to the " `Standard Reply-Auxillary Command`" and "`Standard CC-Primary Command` ", the following interpretation applies. This is in reference to Section 9 of the Siniav Protocol, and left out of the previous sections for clarity.

Table 72: *Standard CC-Primary Command from Reader to Tag(49 bits)*

	Command	Siniav Handle	TC	CRC
# of bits	16	16	1	16
description	**1110**0000 00000001	Siniav Handle	Tx Counter	CRC16

Table 73: *Standard Reply-Auxillary Command from Tag to Reader (36 bits)*

	Header	Siniav Handle	TC	PSI	CRC
# of bits	1	16	1	2	16
description	0	Siniav Handle	same as received TC	00: in progress 01: complete 10: RFFU 11: RFFU	

Test Plan with a Reader Company

Rule: We give them as little as possible. When we hand over a tag it is gooped over with a photodetector to kill it.

Procedure: In 4 rough phases

- (Sirit->Intel)
- This will get filled out more as we go along I'd imagine.

Verification Plan

At some point soon this will become very important. How do we verify the tag works? How do we thoroughly test the firmware and hardware so that we catch the bugs early? A meeting on this should probably occur.

Firmware Version History

***Goal:BTG_V3?:**
… Firmware version deployed on first run of manufactured units.

***Goal: BTG_V2:** Siniav Compliant Prototype Tag
This stage of development represents finally testing against a siniav reader from Sirit. I

- BTGV2_3: *This version will rigorously test TC and error codes. Also state changes perhaps.
- BTGV2_2: *This version will complete SAR/SAW
- BTGV2_1: *This version will complete MA_IMPL
- BTGV2_0: *This version will complete ACKs against the siniav reader.

BTG_V2: Tag for 11/06/12 RIO Testing
After receiving notification that the RIO testing would use G0, a new firmware had to be developed. V2 is meant to work for both customization and G0 protocol, but is not meant as the foundation for a final product.

- BTGV2_0: Developed without G0 reader. Supports the siniav G0 commands and basic state machine.
- *BTGV2_1: Also supports personalization.

BTG_V1: Siniav Dev Tag
After getting the protocol specification for the Siniav Tag, [1], protocol development began. This included establishing firmware architecture, implementing memory maps, supporting cryptography, and supporting the Siniav states and commands. Development is accomplished against an emulated reader built in-house ontop of an FPGA platform (NI R7841R).

- BTGV1_3: *This version will implement the personalization mode that we are anticipating.
- BTGV1_2: *This version will have a thoroughly robust and finely combined receive chain. Sessioning may also be implemented with the watchdog timer here (maybe).
- BTGV1_1: *This version will implement tamper-proofing on the 0.5 board. Also will implement self-checking and zeroization.
- BTGV1_0a: Branched from BTGV1_0-r50. Also called /SATOTemp in project structure. This firmware is for SATO to prove we can report temperature. Not necessarily a BTag dev branch, but a proof-of-concept for SATO.
- BTGV1_0: Version supports tag memory, siniav states, and siniav commands. Most of siniav state transitions are executed properly. TC and error codes supported. Killed mode is implemented. Framework for remaining modes is erected. Clean state is implemented. Self-Check is identified and template, but not implemented.

BTG_V0: Proof of Concept Tag – EPC Dev
(description to be completed)

- BTG_V0_17: Version implemented on the 0.4. Changed the P1ISR to its new awesomeness. receive chain is humming smoothly.
- BTG_V0_16a: - Version used to execute BER test. This version uses the RX_SM to listen only to backscatter). It receives each QUERY and validates its CRC5. This is used as an indicator of BER.
- BTG_V0_16: Fixed to implement Session0. First version implemented on BTag0.4 HW
- BTG_V0_15a: Implemented several "documentation" type changes that were on the backburner:
  - Changed over to doxygen style commenting.
  - Changed the BTAG_VERSIONING_README.txt into doxygen style (i.e. this very file)
  - Started the doxygen Glossary
  - Started the doxygen Todo List
  - Started the @link mainPageAnch BTag SW Documentation Files/Pages @endlink
  - Tons of other Doxy Stuff
- BTG_V0_15: Sleeps w/VLOCLK after 1000 reads for 1 second (by using prep for new). Now uses btgHaltRxSM() to stop machine, instead of just a simple _BIC_SR(GIE). This will lower power consumption by turning of unused peripherals TimerA and TimerB.

- BTG_V0_14a: Version deployed to Sean on 12-22-10. Snapshot of V0_14 while in progress. Modified though to only stay in reply and respond to every Query&ACK. Also increments EPC.
- BTG_V0_14: Supports WRITE. Also uses radio pin to generate RN16. Power measurements are taken for this module (Peak current was 4.4mA!).
- BTG_V0_13: Supports a much larger suite of debug statements. Fixed the ACK Hang(kinda). Supports the READ implementation.
- BTG_V0_12: Supports READY, REPLY, ARBITRATE and ACKNOWLEDGED and OPEN states. No support of sessioning. Support of inventory. Each Response has been tweaked and measured to hit 21.42us. Works smoothly via the RFIDEMO. Still has a hanging issue, which I'm pretty sure is the TimerA OVF. Will fix soon. Also includes the framework for the debug printing.
- BTG_V0_11b: Also the new board demods differently. 11.68us (134 cycles). This is the version that got sent to Sean on 12-23-10. Also was closest to version sent to Marc the month or so previous. Modified to Tx on P1.7.
- BTG_V0_11a: Now increments the EPC after every 500 ACKs. This should allow HW testing of the antenna.Modified to Tx on P1.7 using 2272. This is for Alansons new BTG0.3 board. Built on top of a.
- BTG_V0_11: Code back up to doing ACKS in CCS. Just stays in ready state though, and responds to every Query/Rep/Adjust :). Final zipped vers actually only responds to query (just commented out the tx line from the responses)
- BTG_V0_10: Ported to CCS. Oh man. Long story short changed Everything. This included:
    - Receive Chain
    - State Machine
    - Processing of inventoring will still be the same.
    - WISP Code is officially gone.
- BTG_V0_9: Changed to that bits are now shifted out on P3.7... this requires a wire soldered to board too!
    - Added from what I can tell is all of inventoring.
    - Starting to identify quirky potential bugs in state machine architecture.
    - Fixing lots of the access responses.
    - Need to switch to CCS now because code is too big :(.
-
- BTG_V0_8a: Trying to get to work with BTG board 0.3. didn't finish porting it (timing was weird?) this code is disgusting. Justin 12-23-10
- BTG_V0_8X: The version where Query/Rep/Adjust all respond. This version hits huge read rates of >1000/sec. just for fun. (this version is missing)
- BTG_V0_8: Uses a correct RN16. Built on LFSR. A table of R16's is used, and they are recalculated each time. Also put timing adjustment back into handle_reply_ack.
- BTG_V0_7: Modified Rx to Stop when a queryRep command is received. this way we can deterministically parse queryRep, which wasn't happening before. Modified to support:
    - ready_query (minus slots)
    - ready_queryRep
    - Also rewrote P1ISR.
    - Delay after ACK sets RFIDDemo read rate to 42 rds/sec.
-
- BTG_V0_6: Supports the read of a single byte using the READ command.
- BTG_V0_5: Modified V0_3a to use the unbuffered FM0 version for 640(new). Also removes old sleep functionality on P2.4.
- BTG_V0_4: Changed the State Machine to wait for enough bits inside the while loop. improves response time (T2 from spec)? Concl: Not really. *This is a dead version, didn't improve performance. 0.5 is built off of 0.3.
- BTG_V0_3: Now uses the HT Buffer FM0 code @ 640kHz (I would try a middle of the road option, but our reader only does 160/640 (sigh)
- BTG_V0_2a: Inverted Trig Edges on Rx. Tx is 2.0. Uses 2232. All sleepOnLowVoltage are commented out.

- BTG_V0_2: Now uses the HT Buffer FM0 code @160kHz, where the buffer is pre-computed. This is a lead in to the high-throughput method of transmit required by 640kHz FM0. Tx is on P3.0.
- BTG_V0_1: Modified to support 160kHz FM0 Transmit (verified against demoApp). At this point it could switch between M4 and FM0, but omitting M4. Tx on P3.7
- BTG_V0_0a: Inverted Trig Edges on Rx. Rx:1.2. Tx:1.1. 2.0 is left as input so you can jump a wire on it from 1.1. Also removed sleep functionality. note: a is all configured currently as the simple read app.
- BTG_V0_0: Original WISP41 Code implemented on the 2272 processor. removed a lot of the extraneous features (#ifs, etc).

WISP_41: Reference design and starting point for BTG_V0.
Original BTag development called explicitly for development of the WISP 4.1 code, so the code base started here. Eventually specifications were identified that forced development to migrate away from the WISP architecture completely. This migration happens in the BTGV_0 stage.

- WISP_BTG1_X.1 :1/1a are a pair that implement the WISP using a separate MCU board. '1' is deployed on the MCU board and '1a' on the WISP. This allows for a separate MSP dev board to use the WISP's AFE/antenna.
- WISP_BTG1a X.1a.: '1a' shuts down the WISP's onboard MCU so that the separate MCU board can use the WISP's AFE.