Theseus® Titanium 80 Reference Guide v2.0

Notice

At the time of publication, this document reflects the latest features in SST's offering. However, as we are continually enhancing our products, we recommend that you obtain the latest version of this document from your SST representative before finalizing a design.

Limited Warranty

Devices sold by SST are covered by the warranty and patent indemnification provisions appearing in its Terms of Sale only. SST makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. SST makes no warranty of merchantability or fitness for any purpose. SST reserves the right to discontinue production and change specifications and prices at any time and without notice.

SST assumes no responsibility for the use of any circuitry other than circuitry embodied in an Emosyn[1] product. No other circuits, patents, licenses are implied.

Life Related Policy

In situations where semiconductor component failure may endanger life, system designers using this product should design the system with appropriate error detection and correction, redundancy and back-up features to prevent such an occurrence. Emosyn's products are not authorized for use in critical components in life support devices or systems.

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labelling, can be reasonably expected to result in a significant injury to the User.

2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

Edition 2006

Published by SST, Spinners Court, 53-55 West End, Witney, Oxfordshire, OX28 1NH, U.K.

[1] Emosyn is a registered trademark of Silicon Storage Technology, Inc.

Contents

1. Introduction

1.1 Overview

The Theseus®[2]. Titanium 80 is a High Security, High Performance Enhanced 8051 Smart Card device with 80 kbytes of Flash NVM incorporated in the User Configurable Memory (UCM®[3]) blocks and 1280 bytes of SRAM on-chip. Emosyn incorporates Silicon Storage Technology (SST) SuperFlash®[4] technology in its products.

The Theseus Titanium 80 device is a Complimentary Metal-Oxide Semiconductor (CMOS) integrated circuit for use in Smart Cards. Meeting ISO 7816-3 and GSM 11.11 and 11.12 standards, this Integrated Circuit device can communicate with a Smart Card reader.

[2] Theseus® is a registered trademark of Silicon Storage Technology, Inc.

[3] User Configurable Memory® is a registered trademark of Silicon Storage Technology, Inc.

[4] SuperFlash® is a registered trademark of Silicon Storage Technology, Inc.

Features

- Enhanced 8051- compatible core that operates on 4-clock cycles
- ISO 7816-3 Serial interface compliant
- 1280 bytes of RAM (256 bytes internal scratchpad RAM {IRAM} + 1Kbytes external RAM)
- Software controlled Flash NVM
- Interrupts to trap security violations
- Software controlled power-saving options
- User-friendly code loading process
- Physical Memory Security Attributes
- Customer or factory transport code software loading options[5]
- Enhanced high speed NVM operations
- On-chip Random Number Generator
- On-chip 16-bit Timer/Counter

Environment

- Single 3V or 5V supply (± 10%)
- Operating temperature -25^{0}C to +85^{0}C
- Maximum Supply current:
  - <6 mA@20 MHz*
  - <10 mA@45 MHz*
- Low current Idle Mode
- External clock frequency range1-10 MHz
- 60 MHz Internal Oscillator with divide by 1 to 16
- 4kV ESD Protection
- ISO7816-2 compliant 8-contact module

Security

- Unique chip identification number
- Under-Voltage detection
- Over-Voltage detection
- Under-Frequency on ISO clock detection
- Over-Frequency on ISO clock detection
- Out of Temperature detection
- Ultra Violet light detection
- Glitch detection
- Low Voltage alert (Brownout)
- Security violations are detected, enabling
- User to configure interrupt or reset chip

Performance

- Enhanced 8051 offers 2.5x performance improvement over standard 8051
- 60 MHz capable core
- Dual data pointers speed up table handling
- Dual data pointers speed up table handling
- Dynamic clock sources switching
- Fast UCM code loading
- Memory can be written byte-by-byte at very high speeds

*The maximum supply current meets the stated specification with the programmable pull-up resistor OFF, which is the default condition.

[5] For further details of this option please contact your SST representative.

Development Environment

- Supported by Theseus Simulator Rapid Development Environment
- Programmable through SwiftSIM®[6] Programmer
- Code compatible with the Keil® compiler
- Compatible development and implementation software tools

Supported Standards

- ETSI GSM 11.11
- ETSI GSM 11.12
- ISO 7816-3 compliant electrical interface
- ISO 7816-3 compliant reset and response protocols

Support

- Application Notes and API Code

Document References

- Chip and Module delivery specifications (including packaging description)
- Module Qualification report

Flash Non-Volatile Memory

- 80 Kbytes of Flash UCM (excluding 1K bytes of Locked System Memory)
- 80 Kbytes of large page UCM
- UCM erase selectable to 256 or 512 byte pages
- UCM can be configured by User software to operate as:
  - EEPROM (Default)
  - OTPROM
  - Execute Only
- Access to UCM controlled by physical memory security attributes
- 10 year Data retention on all Flash NVM bytes
- Flash NVM Guaranteed for 100K erase/write cycles
- 250K Erase/Write-cycles (typical) endurance on all Flash NVM bytes
- 2 ms page erase time (typical)
- 25 μs byte write time (typical)

[6] SwiftSIM is a registered trademark of Silicon Storage Technology, Inc.

Figure 1 shows the Theseus Titanium 80 architecture in overview.

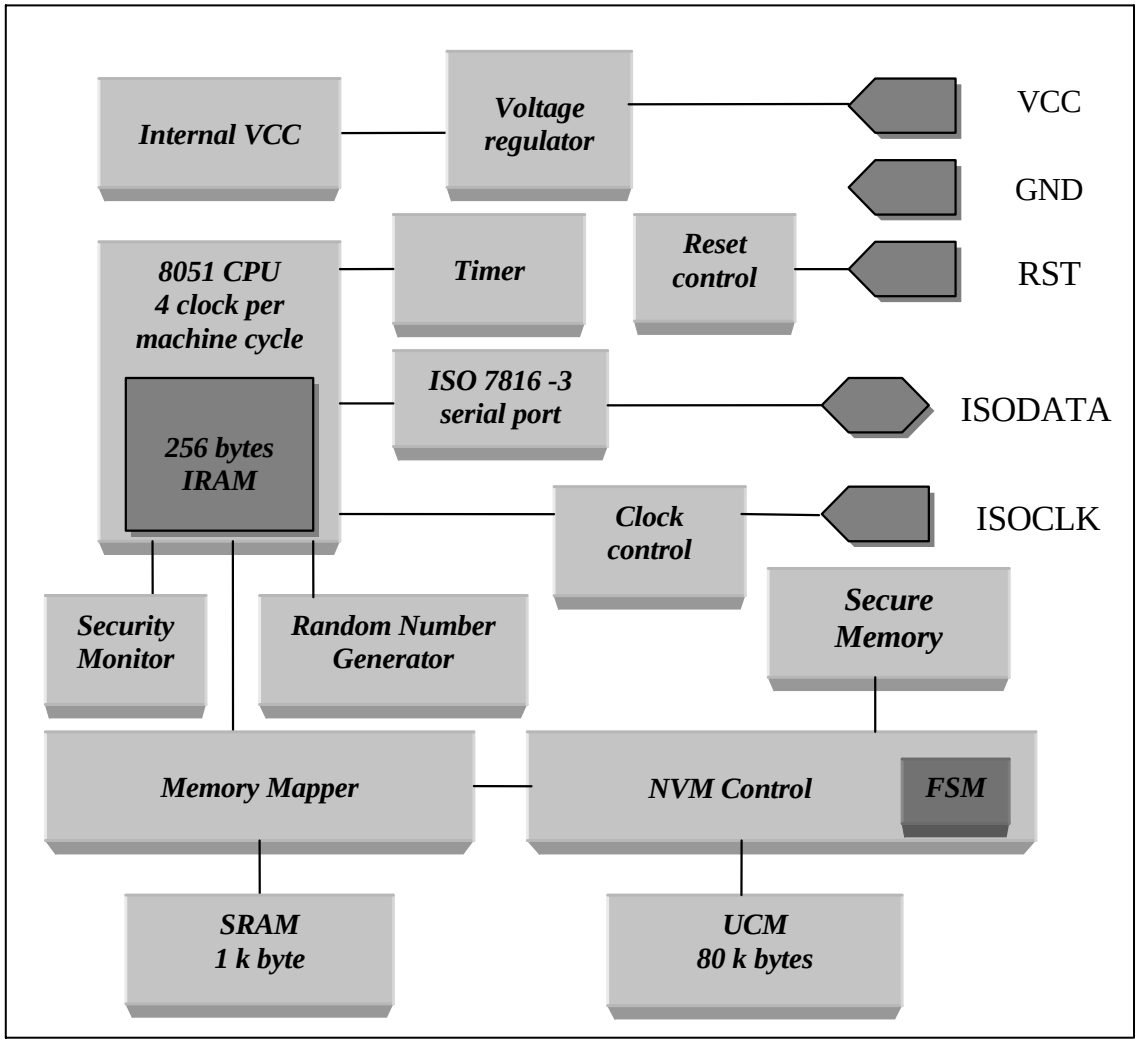


Figure 1: Architecture Overview

2. Technical Data

2.1 Top Level Specifications

The Top Level Specifications for the Theseus Titanium 80 are shown within this section of the document.

2.1.1 Common Specifications

The Common Specifications for the Theseus Titanium 80 are shown in Table 1:

Absolute Maximum Ratings	Min.	Typical	Max.	Unit
External Power Supply (Vcc EXT to GND)	-0.3		+6.0	V
All Pins to GND	-0.3			V
All Pins to Vcc			+0.3	V
Storage Temperature, at R.H.<60%	-15		+30	°C
Recommended Usage Ranges	**Min.**	**Typical**	**Max.**	**Unit**
External Power Supply (Vcc EXT)	2.7	3.0, 5.0	5.5	V
Operating Temperature	-25	+25	+85	°C

Table 1: Common Specifications

Note Operation or storage of the product beyond Absolute Maximum Ratings may damage the product and voids any product warrantees.

2.1.2 ISODATA Driver Characteristics

The ISO_DATA Driver Characteristics are shown in Table 2:

Parameter	Symbol	Conditions*	Min.	Max.	Unit
High Output Voltage	VIH	I/O@3V&5V IIHmax =+ 20uA	0.7 Vcc	Vcc+0.3	V
	VIL	I/O@3V IIL = +1mA	-0.3	0.2 Vcc	
		I/O@5V IIL = + 1mA	-0.3	0.8	
	VOH	Clock @ 3V IOH max = +20 µA	0.7 x Vcc	Vcc	
		Clock @ 5V IOH max = +20 µA	0.7 x Vcc		
		RST@ 3V IOH max = +200 µA	0.8 x Vcc		
		RST@ 5V IOH max = +20 µA	Vcc -0.7		
		I/O@ 3V IOHmax = +1mA	0.7 x Vcc		
		I/O@ 5V IOHmax = +1mA	0.7 x Vcc		
Low Output Voltage	VOL	Clock @ 3V IOL max = -200 µA	0	0.2 Vcc	V
		Clock @ 5V IOL max = -200 µA	0	0.5	
		RST@ 3V IOL max = -200 µA	0	0.2 Vcc	
		RST@ 5V IOL max = -200 µA	0	0.6	
		I/O@ 3V IOLmax = -1mA	0	0.4	
		I/O@ 5V IOLmax = -1mA	0	0.4	
Rise/Fall Time	tR and tF	Clk Cin = Cout = 30pF		50	nS
		Rst Cin = Cout = 30pF		400	uS
		I/O Cin = Cout = 30pF		1	

* The maximum supply current is with the programmable pull-up resistor OFF, the default condition.

Table 2: ISODATA Driver Characteristics

2.1.3 ISODATA, ISOCLOCK and ISORESET

The input buffers ISO_RESET, ISO_CLOCK and ISO_DATA that were shown previously within Figure 1 are further detailed in Figure 2:

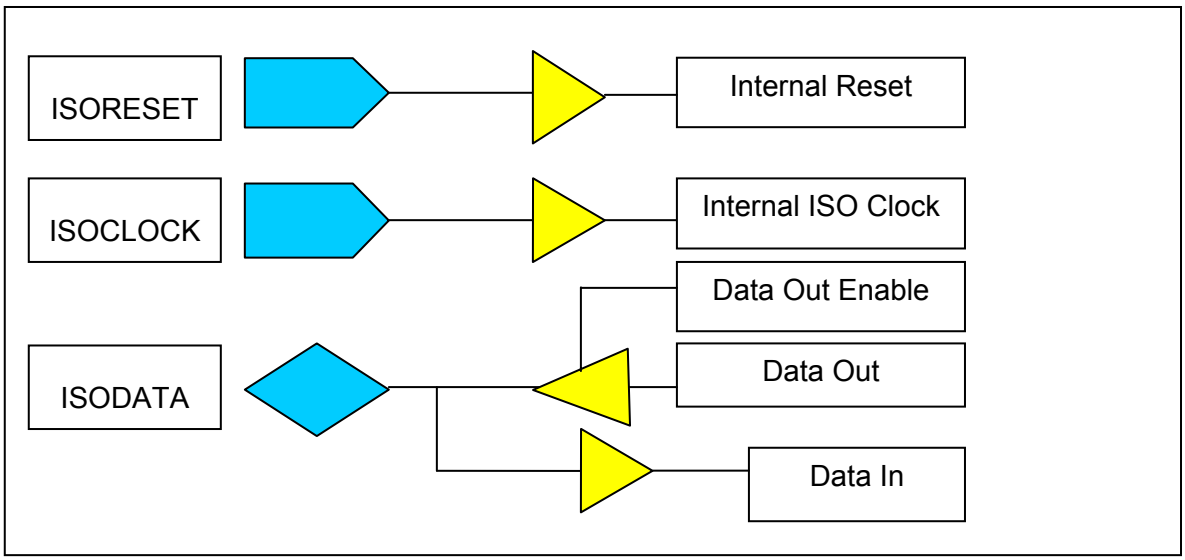


Figure 2: Input buffer

2.1.4 Reset Block

An input buffer detects reset on the reset pin. When reset goes high, the reset signal is held for a further delay by the interface block before being released synchronously with the CPU clock. As a protective measure, if the Vcc goes below a given threshold, a reset will be asserted for the duration of the excursion plus a delay. This protects the chip against attacks at low Vcc.

2.1.5 ISOCLOCK Speed

An internal filter exists to ensure the 10MHz specification (goes up to 20 MHz by design) is not exceeded and limits the ISO_CLK speed.

2.1.6 DC Characteristics

The DC Characteristics are shown in Table 3:

Parameter	Symbol	Conditions*	Min.	Max.	Unit
Supply Current Active	Icc	2.7 to 3.3V, ISO_CLK 4MHz, 20MHz Internal, Vih = VCC +/- 0.3V, Vil = VSS +/- 0.3V		6	mA
Supply Current Active	Icc	4.5 to 5.5V, ISO_CLK 5MHz, 45MHz Internal Vih = VCC +/- 0.3V, Vil = VSS +/- 0.3V		10	mA
Supply Current Idle	Icci	2.7 to 5.5V, ISO_CLK 1MHz, 25°C Vih = VCC +/- 0.3V, Vil = VSS +/- 0.3V		200	uA
Stop Clock Mode	Icci	1MHz, 25°C <100uA at 3V and <200uA at 5V		100 200	uA

*The maximum supply current is with the programmable pull-up resistor OFF, the default condition.

Table 3: DC Characteristics

2.1.7 System-Wide Special Function Register

The System-wide Special Function Register (SFR) map is shown in Figure 3. Their default values after hardware reset are shown in Table 4. These values may be modified after the Firmware Start-up code execution that is also shown in Table 4.

Customer's code can access these SFR's directly. If the User tries to read an invalid SFR address, then 0x00 will be returned. Writing to an invalid SFR will have no effect.

Note SFR's Security Mask, Security Action, Clock Divider and Clock Source Select must be accessed via API's.

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8		Nv FSM Mode							CF
C0	Prog IO								C7
B8	IP								BF
B0	Security Status		ISOCON		ISOTMR	TRNG			B7
A8	IE								AF
A0			MPTR						A7
98	SCON	SBUF							9F
90									97
88	TCON	TMOD	TL0		TH0				8F
80	Nv FSM Con	SP	DPL0	DPH0	DPL1	DPH1	DPS	PCON	87

Bit Addressable

	= Active SFR
	= Protected - Cannot be used

Figure 3: System-wide SFR Map

A summary of the SFR locations are shown in Table 4:

Accessed Directly				
Address	**SFR Name**	**Value at Hardware Reset**	**Value after Firmware Start up Code execution**	**Remarks**
0x81	SP	07h	BFh	
0x82	DPL0	00h	First address of User's code	
0x83	DPH0	00h	First address of User's code	
0x84	DPL1	00h	00h	
0x85	DPH1	00h	00h	
0x86	DPS	00h	00h	
0xD0	PSW	00h	Undefined	
0xE0	ACC	00h	Undefined	
0xF0	B	00h	00h	
0xA2	MPTR	00h	00h	
0x80	NvFSM Con	00h	00h	
0xB0	Security Status	00h	00h	
0xB2	ISOCON	00h	54h	
0xB4	ISOTMR	00h, 00h	01h, 73h	9600 Baud
0xC0	ProgIO	00h	00h	
0x87	PCON	00h	00h	
0x98	SCON	00h	00h	
0x99	SBUF	00h	00h	
0xC9	NvFSM Mode	00h	00h	
0xA8	IE	00h	3Fh	
0xB8	IP	00h	3Eh	
0x88	TCON	00h	00h	
0x89	TMOD	00h	00h	
0x8A	TL0	00h	00h	
0x8C	TH0	00h	00h	
0xB5	TRNG	Undefined	Undefined	
Accessed Indirectly via API's				
--	Security Mask	FFh	00h	
--	Security Action	x1xx xxxxb	x1xx xxxxb	
--	Clock Divider	00h	xxxx 0001b	20MHz
--	Clock Source Select	00h	xxxx xx10b	Internal Oscillator

Table 4: SFR Summary

2.1.8 GSM Specification Conformance

The Theseus Titanium family of devices are capable of operating within and beyond the range from 3V -10% to 5V +10%. These operating conditions conform to GSM specification 11.11 and 11.12 and ISO 7816-3.

- Maximum supply current is less than 6.0mA with the programmable pull-up resistor OFF. Typical values for supply current vs. CPU frequency with the pull-up resistor OFF are shown below.

CPU Frequency	Supplied Current
7.5MHz	3.0mA
15.0MHz	4.0mA
22.5MHz	5.0mA
30.0MHz	6.0mA

Notes These figures are for the Theseus Titanium 80 device based on operation temperature of 23°C, 5.5V Vcc and ISO clock frequency of 4MHz. The Internal Oscillator trimmed to 60MHz.

The CPU clock frequency is obtained using divide ratio.

Supply current is measured as average current during ATR:

- The leakage current at GSM pin C7 is less than 20μA
- The supply current when the device is in idle mode is below 200uA at 1MHz ISO clock frequency at room temperature with the programmable pull-up resistor OFF
- When the device is in clock stop idle mode and the programmable pull-up resistor is OFF, the current consumption is <200uA @5V and <100uA @3V (typical)

3. System Start up

3.1 Power-up

On Power up the processor starts executing firmware code then jumps to 0800h where the User code is loaded.

Notes For security reasons, Emosyn will install Transport code. The device awaits authentication before it erases the transport code and the chip is prepared for loading of customer code.

The Transport code is a small program pre-loaded into the Theseus Titanium 80. When the chip is powered and reset, the chip sends out an Answer-To-Reset (ATR), which identifies the device. If the correct key is sent to the chip, then the Transport code program will erase itself, and prepare the chip for loading. User code may then be loaded into the chip.

It is possible to load other applications at this stage. When not specified, the devices will be loaded with an Emosyn default Transport code.

For further details of this option please contact your local Emosyn sales representative.

Figure 4 shows the User code loading process:

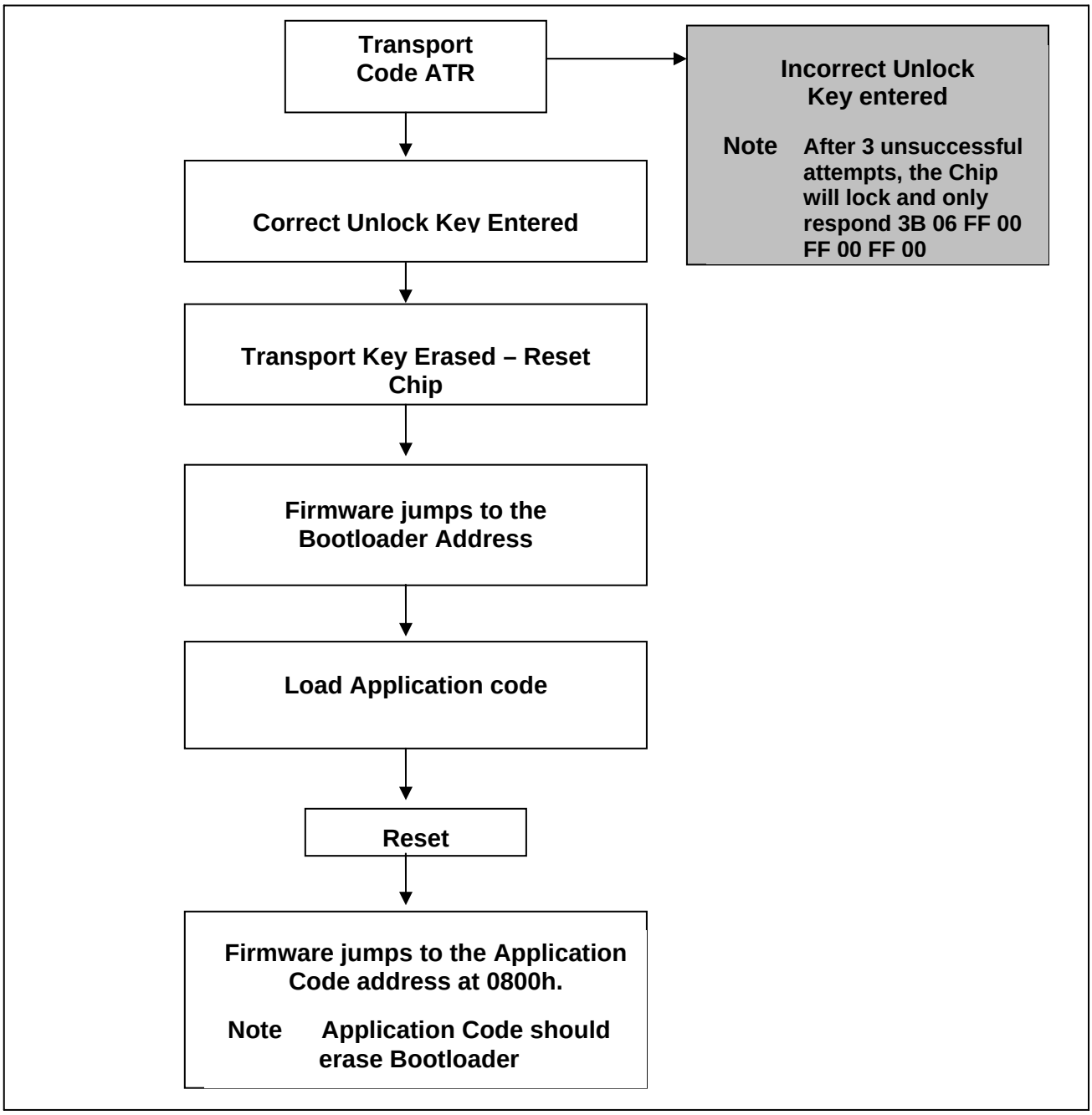


Figure 4: User Code Loading Process

Note The Bootloader is located at the top of the UCM. For more information please see Section 17.

3.2 System software start-up sequence

After hardware reset the Theseus Titanium 80 starts executing firmware code.

The start-up routine performs the following sequence of tasks:

1. Initialises all SFR's as shown in Table 4.
2. Sets Oscillator clock to 20MHz internal OSC.
3. Sets IO communication to 9600 Baud.
4. Decides whether to jump to User code or Bootloader.

This page is intentionally left blank

4. The 8051 Core

4.1 8051 Core Overview

The Theseus Titanium 80 core is fully compatible with the standard 8051 instruction set. The architecture consists of an 8051-processing core with hardware functions controlled by software via memory mapped SFR's. This technique allows wide variations in implementation of peripherals, using the same 8051 software development tools.

4.1.1 Performance

The Theseus Titanium 80 processor core has been designed to offer increased performance by executing instructions in a 4-clock bus cycle, as opposed to the 12-clock bus cycle in the standard 8051. The shortened bus timing improves the instruction execution rate for most instructions by a factor of three over standard 8051 architectures.

Another notable improvement is the provision of dual data pointers to ease and speed up table handling.

The Theseus Titanium 80 requires a different number of instruction cycles on some instructions than on a standard 8051. An average speed improvement for the entire instruction set of approximately 2.5X is shown in Table 5, they are calculated as follows:

Number of OPCODES	Speed Improvement
150 of 246	3.0X
51 of 246	1.5X
43 of 246	2.0X
2 of 246	2.4X
Total = 246	Average = 2.5X

Table 5: Performance

Note Comparison is for the Theseus Titanium 80 core and the standard 8051 running at the same clock frequency.

4.1.2 Software Compatibility

The Theseus Titanium 80 core is code compatible with the 8051 (MCS 51 Object Module Format) industry standard micro-controller. That is, object code compiled with an industry standard 8051 compiler or assembler will execute on the Theseus Titanium 80 core and will be functionally equivalent when in 8051 addressing mode. Code with timing loops may require modification because the Theseus Titanium 80 core uses different instruction timing than the standard 8051.

The Instruction set definitions in Section 5.1 shows the number of cycles required and the function performed by the Theseus Titanium 80.

Note For code compatibility Emosyn recommends the Keil® family of compilers.

4.1.3 Instruction Timing

In the standard 8051, all instructions except for MUL and DIV take one or two instruction cycles to complete. In the Theseus Titanium 80 core architecture, instructions can take between one and five instruction cycles to complete, whereas the standard 8051 will take 1-2 instruction cycles.

For example, in the standard 8051, the instructions MOVX A, @DPTR and MOV direct, each take 2 instruction cycles (24 clock cycles) to execute.

In the Theseus Titanium 80 core architecture, MOVX A, @DPTR takes two instruction cycles (8 clock cycles), and MOV direct takes three instruction cycles (12 clock cycles). Both instructions will execute faster by 2 to 3 times on the Theseus Titanium 80 core than they do on the standard 8051, but require different numbers of clock cycles.

For timing of real-time events, use the numbers of instruction clock cycles from the Instruction Set to calculate the timing of software loops. The bytes column indicates the number of memory accesses (bytes) needed to execute the instruction. In most cases, the number of bytes is equal to the number of instruction cycles required to complete the instruction. However, as indicated, there are some instructions (for example, DIV and MUL) that require a greater number of instruction cycles than memory accesses. The number of clock cycles required is 4 times the number of instruction cycles.

See Section 5.1 for the instruction set.

4.2 8051 Core SFR's

The standard 80C5116-bit addressing mode:

Note

Hardware Set:	Indicates an SFR bit that can be set by on-chip hardware (in addition to CPU write operation).
Hardware Clear:	Indicates an SFR bit that can be cleared by on-chip hardware (in addition to CPU write operation).
User Code Read:	Indicates an SFR bit that can be read by software running in Kernel Mode.
User Code Writes:	Indicates and SFR bit that can be written by software running in Kernel mode.

4.2.1 SP SFR- 81h

	SP.7	SP.6	SP.5	SP.4	SP.3	SP.2	SP.1	SP.0
VALUE AT HARDWARE RESET						07h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						BFh		
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES	✓	✓	✓	✓	✓	✓	✓	✓

Symbol	Position	Function
	SP.7	Stack pointer - original 8051 core register
	SP.6	
	SP.5	
	SP.4	
	SP.3	
	SP.2	
	SP.1	
	SP.0	

4.2.2 DPLO SFR- 82h

	DPL0 .7	DPL0 .6	DPL0 .5	DPL0 .4	DPL0 .3	DPL0 .2	DPL0 .1	DPL0 .0
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						First address of User's code		
RESET	0	0	0	0	0	0	0	0
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES	✓	✓	✓	✓	✓	✓	✓	✓

Symbol	Position	Function
	DPL0.7	Low byte of the first DPTR register
	DPL0.6	
	DPL0.5	
	DPL0.4	
	DPL0.3	
	DPL0.2	
	DPL0.1	
	DPL0.0	

4.2.3 DPH0 SFR- 83h

	DPH0 .7	DPH0 .6	DPH0 .5	DPH0 .4	DPH0 .3	DPH0 .2	DPH0 .1	DPH0 .0
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						First address of User's code		
RESET	0	0	0	0	0	0	0	0
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES	✓	✓	✓	✓	✓	✓	✓	✓

Symbol	Position	Function
	DPH0.7	High byte of the first DPTR
	DPH0.6	
	DPH0.5	
	DPH0.4	
	DPH0.3	
	DPH0.2	
	DPH0.1	
	DPH0.0	

4.2.4 DPL1 SFR- 84h

	DPL1 .7	DPL1 .6	DPL1 .5	DPL1 .4	DPL1 .3	DPL1 .2	DPL1 .1	DPL1 .0
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						00h		
RESET	0	0	0	0	0	0	0	0
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES	✓	✓	✓	✓	✓	✓	✓	✓

Symbol	Position	Function
	DPL1.7	Low byte of the second DPTR
	DPL1.6	
	DPL1.5	
	DPL1.4	
	DPL1.3	
	DPL1.2	
	DPL1.1	
	DPL1.0	

4.2.5 DPH1 SFR- 85h

	DPH1 .7	DPH1 .6	DPH1 .5	DPH1 .4	DPH1 .3	DPH1 .2	DPH1 .1	DPH1 .0
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						00h		
RESET	0	0	0	0	0	0	0	0
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES	✓	✓	✓	✓	✓	✓	✓	✓

Symbol	Position	Function
	DPH1.7	High byte of the second data pointer
	DPH1.6	
	DPH1.5	
	DPH1.4	
	DPH1.3	
	DPH1.2	
	DPH1.1	
	DPH1.0	

4.2.6 DPS SFR- 86h

	DPS .7	DPS .6	DPS .5	DPS .4	DPS .3	DPS .2	DPS .1	DPS .0
	-	-	-	-	-	-	-	SEL
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						00h		
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ								✓
USER CODE WRITES								✓

Symbol	Position	Function
-	DPS.7	Reserved for future use
-	DPS.6	
-	DPS.5	
-	DPS.4	
-	DPS.3	
-	DPS.2	
-	DPS.1	
DPS	DPS.0	0 = Select DPL0, DPH0 and DPX0 as the DPTR in @DPTR addressing 1 = Select DPL1, DPH1 and DPX1 as the DPTR in @DPTR addressing

4.2.7 PSW SFR- D0h

	PSW .7	PSW .6	PSW .5	PSW .4	PSW .3	PSW .2	PSW .1	PSW .0
	CY	AC	F0	RS1	RS0	OV	-	P
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						Undefined		
HARDWARE SET	✓	✓				✓		
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓		✓
USER CODE WRITES			✓	✓	✓			✓

Access restrictions: None

Symbol	Position	Function
CY	PSW.7	Carry flag
AC	PSW.6	Auxiliary carry flag (for BCD operations)
F0	PSW.5	User definable flag
RS1	PSW.4	
RS0	PSW.3	
OV	PSW.2	Overflow flag
Reserved	PSW.1	Software always read 0. Write has no effect
P	PSW.0	Parity flag

4.2.8 ACC SFR- E0h

	ACC .7	ACC .6	ACC .5	ACC .4	ACC .3	ACC .2	ACC .1	ACC .0
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						Undefined		
RESET	0	0	0	0	0	0	0	0
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES	✓	✓	✓	✓	✓	✓	✓	✓

Symbol	Position	Function
	ACC.7	Original 8051 register
	ACC.6	
	ACC.5	
	ACC.4	
	ACC.3	
	ACC.2	
	ACC.1	
	ACC.0	

4.2.9 B SFR- F0h

	B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						00h		
RESET	0	0	0	0	0	0	0	0
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES	✓	✓	✓	✓	✓	✓	✓	✓

Symbol	Position	Function
	B.7	Original 8051 register
	B.6	
	B.5	
	B.4	
	B.3	
	B.2	
	B.1	
	B.0	

4.2.10 MPTR SFR- A2h

	MPTR .7	MPTR .6	MPTR .5	MPTR .4	MPTR .3	MPTR .2	MPTR .1	MPTR .0
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						00h		
RESET	0	0	0	0	0	0	0	0
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ	✓	✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES	✓	✓	✓	✓	✓	✓	✓	✓

Symbol	Position	Function

	MPTR.7	This forms the high byte address in the following instructions:
		
		MOVX @R0/1,A
		MOVX A,@R0/1

	MPTR.6	
	MPTR.5	
	MPTR.4	
	MPTR.3	
	MPTR.2	
	MPTR.1	
	MPTR.0	

4.2.11 PCON SFR- 87h

	PCON .7	PCON .6	PCON .5	PCON .4	PCON .3	PCON .2	PCON .1	PCON .0
	-	-	-	-	-	-	-	IDLE
VALUE AT HARDWARE RESET						00h		
VALUE AFTER FIRMWARE START UP CODE EXECUTION						00h		
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ								
USER CODE WRITES								✓
APP RD								
APP WT								

Symbol	Position	Function
-	PCON.7	Reserved for future use
-	PCON.6	
-	PCON.5	
-	PCON.4	
-	PCON.3	
-	PCON.2	
-	PCON.1	
IDLE	PCON.0	Writing a ‘1’ to this bit will cause the CPU to go into Idle mode

This page is intentionally left blank

5. 8051 Instruction Set

5.1 Instruction Set Definitions

The Instruction set definitions are shown in Table 6:

Symbol	Function
A	Accumulator
Rn	Register R7-RO
Direct	Internal register address
@ Ri	Internal register pointed to by RO or R1 (except MOVX) data memory
Rel	Two's complement offset byte
Bit	Direct bit address
# data	8-bit constant immediate
# data 16	16-bit constant
Addr 11	11-bit destination address
Addr 16	16-bit destination address

Table 6: Instruction Set Definitions

The Arithmetic table is shown in Table 7:

Mnemonic	Description	Bytes	Instruction Cycles	Hex Code
Arithmetic				
ADD A, Rn	Add register to A	1	1	28-2F
ADD A, direct	Add direct byte to A	2	2	25
ADD A, @ Ri	Add data memory to A	1	1	26-27
ADD A, #data	Add immediate to A	2	2	24
ADDC A, Rn	Add register to A with carry	1	1	38-3F
ADDC A, direct	Add direct byte to A with carry	2	2	35
ADDC A, @ Ri	Add data memory to A with carry	1	1	36-37
ADDC A, #data	Add immediate to A with carry	2	2	34
SUBB A, Rn	Subtract register from A with borrow	1	1	98-9F
SUBB A. direct	Subtract direct byte from A with borrow	2	2	95
SUBB A, @ Ri	Subtract data memory from A with borrow	1	1	96-97
SUBB A, #data	Subtract immediate from A with borrow	2	2	94
INC A	Increment A	1	1	04
INC Rn	Increment register	1	1	08-0F
INC direct	Increment direct byte	2	2	05
INC @ Ri	Increment data memory	1	1	06-07
DEC A	Decrement A	1	1	14
DEC Rn	Decrement register	1	1	18-1F
DEC direct	Decrement direct byte	2	2	15
DEC @ Ri	Decrement data memory	1	1	16-17
INC DPTR 0 or 1	Increment 2 byte data pointer	1	2	A3
MUL AB	Multiply A by B	1	5	A4
DIV AB	Divide A by B	1	5	84
DA A	Decimal adjust A	1	1	D4

Table 7: Arithmetic Table

The Logical table is shown in Table 8:

Mnemonic	Description	Bytes	Instruction Cycles	Hex Code
Logical				
ANL A, Rn	AND register to A	1	1	58-5F
ANL A, direct	AND direct byte to A	2	2	55
ANL A, @ Ri	AND data memory to A	1	1	56-57
ANL A, #data	AND immediate to A	2	2	54
ANL direct, A	AND A to direct byte	2	2	52
ANL direct, #data	AND immediate data to direct byte	3	3	53
ORL A, Rn	OR register to A	1	1	48-4F
ORL A, direct	OR direct byte to A	2	2	45
ORL A, @ Ri	OR data memory to A	1	1	46-47
ORL A, #data	OR immediate to A	2	2	44
ORL direct, A	OR A to direct byte	2	2	42
ORL direct, #data	OR immediate data to direct byte	3	3	43
XRL A, Rn	Exclusive-OR register to A	1	1	68-6F
XRL A, direct	Exclusive-OR direct byte to A	2	2	65
XRL A, @ Ri	Exclusive-OR data memory to A	1	1	66-67
XRL A, #data	Exclusive-OR immediate to A	2	2	64
XRL direct, A	Exclusive-OR A to direct byte	2	2	62
XRL direct, #data	Exclusive OR immediate to direct byte	3	3	63
CLR A	Clear A	1	1	E4
CPL A	Complement A	1	1	F4
SWAP A	Swap nibbles of A	1	1	C4
RL A	Rotate A left	1	1	23
RLC A	Rotate A left through carry	1	1	33
RR A	Rotate A right	1	1	03
RRC A	Rotate A right through carry	1	1	13

Table 8: Logical Table

5.2 Explanation of Data Transfer commands

The User is able to write and read a byte respectively from the Data space memory with the command movx@RI,a and movx a,@Ri.

Note To use this command, the User needs to load the byte “MPTR” first.

The SFR byte MPTR (see Figure 3) sets up the high byte of the addressed Data space memory location. The MPTR address is 0xA2.

By default this byte is set to 0x00, which means that the range 0x0000 to 0x00FF of Xdata space memory is addressed by this command.

The value of the “Ri” byte, which is the indirect data pointer (R0 or R1), is the Least Significant Byte (LSB) of the Xdata space address to which the desired byte will be written or read from.

The value set in “A”, the accumulator, is the value which will be written or the value read from the address “MPTR:@ri” e.g. 0x0A50.

```
; Write data value 0xCA at address 0x0A50
mov MPTR, #0AH
mov r0, #50H
mov a, #0CAH
movx @r0, a

; Read byte from the address 0x0A50
mov MPTR, #0AH
mov r0, #50H
movx a, @r0
```

The Programme Status Word flags are shown in Table 9, Bit Operations:

Mnemonic	Description	Bytes	Instruction Cycles	Hex Code
Bit Operations				
CLR C	Clear carry	1	1	C3
CLR bit	Clear direct bit	2	2	C2
SETB C	Set carry	1	1	D3
SETB bit	Set direct bit	2	2	D2
CPL C	Complement carry	1	1	B3
CPL bit	Complement direct bit	2	2	B2
ANL C, bit	AND direct bit to carry	2	2	82
ANL C, /bit	AND direct bit inverse to carry	2	2	B0
ORL C, bit	OR direct bit to carry	2	2	72
ORL C, /bit	OR direct bit inverse to carry	2	2	A0
MOV C, bit	Move direct bit to carry	2	2	A2
MOV bit, C	Move carry to direct bit	2	2	92

Table 9: Bit Operations

The Data Transfer instructions are shown in Table 10:

Mnemonic	Description	Bytes	Instruction Cycles	Hex Code
Data Transfer Instructions				
MOV A, Rn	Move register to A	1	1	E8-EF
MOV A, direct	Move direct byte to A	2	2	E5
MOV A, @ Ri	Move data memory to A	1	1	E6-E7
MOV A, #data	Move immediate to A	2	2	74
MOV Rn, A	Move A to register	2	1	F8-FF
MOV Rn, direct	Move direct byte to register	2	2	A8-AF
MOV Rn, #data	Move immediate to register	2	2	78-7F
MOV direct, A	Move A to direct byte	2	2	F5
MOV direct, Rn	Move register to direct byte	2	2	88-8F
MOV direct, direct	Move direct byte to direct byte	3	3	85
MOV direct, @Ri	Move data memory to direct byte	2	2	86-87
MOV direct, #data	Move immediate to direct byte	3	3	75
MOV @RI, A	Move A to data memory	1	1	F6-F7
MOV @ Ri, direct	Move direct byte to data memory	2	2	A6-A7
MOV @ Ri, #data	Move immediate to data memory	2	2	76-77
MOV DPTR, #data	Move 16 bit immediate to data pointer	3	3	90
MOVC A, @ A+DPTR	Move code byte relative DPTR to A	1	2	93
MOVC A, @A+PC	Move code byte relative PC to A	1	2	83
MOVX A, @ Ri	Move external data (A8) to A	1	2	E2-E3
MOVX A, @DPTR	Move external data (A16) to A	1	2	E0
MOVX @ Ri, A	Move A to external data (A8)	1	2	F2-F3
MOVX @DPTR, A	Move A to external data	1	2	F0
PUSH direct	Push direct byte onto stack	2	2	C0
POP direct	Pop direct byte from stack	2	2	D0
XCH A, Rn	Exchange A and register	1	1	C8-CF
XCH A, direct	Exchange A and direct byte	2	2	C5
XCH A, @ Ri	Exchange A and data memory	1	1	C6-C7
XCHD A @RI	Exchange A and data memory nibble	1	1	D6-D7

Table 10: Data Transfer Instructions

The Branching Instructions are shown in Table 11:

Mnemonic	Description	Bytes	Instruction Cycles	Stack Usage	Hex Code
Branching Instructions					
ACALL addr 11	Absolute call to subroutine	2	3	2	11-F1
LCALL addr 16	Long call to subroutine	3	4	2	12
RET	Return from subroutine	1	2	2	22
RETI	Return from Interrupt	1	2	2	32
AJMP addr 11	Absolute jump unconditional	2	3	X	01-E1
LJMP addr 16	Long Jump unconditional	3	4	X	02
SJMP rel	Short jump (relative address)	2	3	X	80
JC rel	Jump on carry =1	2	3	X	40
JNC rel	Jump on carry = 0	2	3	X	50
JB bit, rel	Jump on direct bit = 1	3	4	X	20
JNB bit, rel	Jump on direct bit = 0	3	4	X	30
JBC bit, rel	Jump on direct bit = 1 and clear	3	4	X	10
JMP @A+DPTR	Jump indirect relative DPTR	1	2	X	73
JZ rel	Jump on accumulator = 0	2	3	X	60
JNZ rel	Jump on accumulator / = 0	2	3	X	70
CJNE A, direct, rel	Compare A, direct JNE relative	3	4	X	B5
CJNE A, #d rel	Compare A, immediate JNE relative	3	4	X	B4
CJNE Rn, #d, rel	Compare reg, immediate JNE relative	3	4	X	B8-BF
CJNE @Ri, #d, rel	Compare Ind, immediate JNE relative	3	4	X	B6-B7
DJNZ Rn, rel	Decrement register, JNZ relative	2	3	X	D8-DF
DJNZ direct, rel	Decrement direct byte, JNZ relative	3	4	X	D5

Table 11: Branching Instructions Table

The Miscellaneous table is shown in Table 12:

Mnemonic	Description	Bytes	Instruction Cycles	Hex Code
Miscellaneous				
NOP	No operation	1	1	00
Reserved (NOP)	No operation	1	1	A5

Table 12: Miscellaneous Table

6. Memory Map

6.1 Memory map

The Code and XDATA memory space of Theseus Titanium 80 contain 3 types of memory, shown in Figure 5. These memories are always present at the specified address. Also shown within Figure 5 are the Internal Memory and SFR locations.

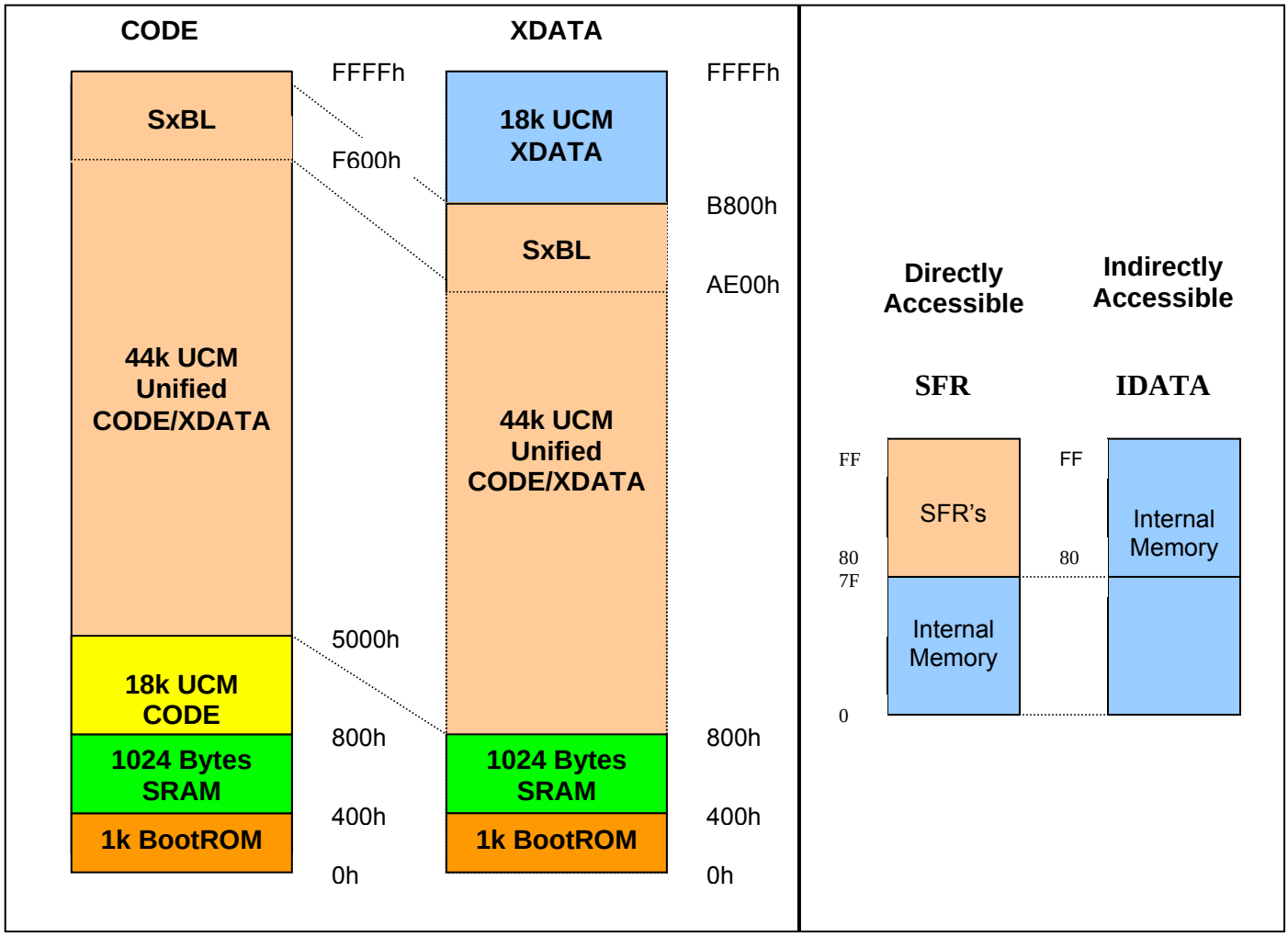


Figure 5: Memory map

Note A "Core bootloader" located between F600h and F700h can be enabled by erasing the normal and ISO bootloader in Code memory from F700 to FFFF.

- The 18k block in CODE space is **ONLY** addressable via CODE space. It does **NOT** appear in XDATA space
- The 18k block at the top of XDATA space is **ONLY** addressable via XDATA space. It does **NOT** appear in CODE space
- The 44k block appears in **BOTH** CODE and XDATA space but at different addresses. The byte at 5000h in CODE space is the **SAME BYTE** as 0800h in XDATA space
- To calculate the XDATA address of a physical byte in CODE space in the 44k UCM block, subtract 4800h
- Conversely, to calculate the CODE address of a physical byte in XDATA space, add 4800h
- The 8051 Core has 256 bytes of Internal RAM (IRAM). The whole of the 256 bytes can be accessed indirectly using the “Mov @R0,XX” command
- The lower 128 bytes of IRAM can also be accessed directly
- The upper 128 bytes of directly accessible memory space is reserved for SFR’s

7. Security

7.1 Principles

The security system on the Theseus Titanium 80 provides a range of protection to assure physical security and reliability of the system. This section shows how the Theseus Titanium 80 provides cover against threats. See Section 2 for the operating limits.

7.2 Overview

This section shows how the Theseus Titanium 80 provides cover against threats.

Faults are detected and captured through hardware into registers. Except for Vdd faults, a system interrupt is raised upon detection. There is a configurable option for reset.

Faults are not stored permanently and can be cleared through software or RESET. A fault is latched until cleared by software or a reset.

Physical faults are detected by analog circuitry and handled by digital circuitry:

- Under-voltage on Vcc
- Over-voltage on Vcc
- Under-frequency on ISO clock
- Over-frequency on ISO clock
- Out of bounds temperature operation
- Ultra violet light exposure
- Under-voltage on Vdd (wired directly into POR)
- Glitch detection on Vcc
- Brown-out under voltage on Vcc (a low voltage warning detector)

7.3 How Fault Detection works

When one or more of the conditions above become true, a corresponding flag bit will be set in the Security Status SFR. Each of these flags will then be AND'ed with its mask bit from the Security Mask SFR.

If one or more unmasked flag is set, the Physical Fault Register (PFLT) flag bit in SFR Security Status will be set.

Note PFLT is the interrupt flag.

7.4 Physical Faults

1. All detectors are hardware based and are implemented inside the analog block.
2. Detector outputs are latched in security status SFR's. A mask register determines which violations can cause action. The security status register is asynchronously set by faults and cleared synchronously by software or by reset.

3. Unmasked faults are OR'ed together and directly produce interrupts.
4. The Security Action register determines if RESET will occur on unmasked faults.
5. Upon power-up and before firmware initialization, all fault detectors are unmasked and action is set to be INTERRUPT (this is the default condition). Since all interrupts are disabled at reset, fault detectors with the exception of under-voltage on Vdd, will not have any effect on system operation, until they are enabled by the software.

7.5 Maskable detectors

Detector	Maskable
Over Voltage on Vcc Ext	Yes
Under Voltage on Vcc Ext	Yes
Under Voltage on Vdd	No
Over Frequency	Yes
Under Frequency	Yes
Over Temperature	Yes
Under Temperature	Yes
Ultra Violet	Yes
Vcc Glitch detection	Yes
Vcc Brown out detector	Yes

7.5.1 Physical Detectors and Power-Up

The physical (analog) detectors depend on trim settings that are loaded during firmware initialization. Therefore, from Vcc power-up and until firmware initialization is complete the detectors are at their default settings.

7.5.2 Digital implementation of the Physical Faults

The security monitor in the digital block responds to the fault conditions generated by the analog sensors. The security monitor will latch/register the fault conditions in a status register. The output of the status register will be AND'ed with a mask register to generate a physical fault interrupt condition. If that interrupt is enabled, an interrupt will be set.

7.5.3 Over-Voltage Detector on VCC Ext

For security purposes this circuit tests for the Over-Voltage condition on Vcc external. The detection level is in line with the absolute maximum of Vcc external.

7.5.4 Under-Voltage Detector on VCC Ext

For security purposes, an Under-Voltage detector this circuit is provided which tests for that condition on the external supply voltage Vcc. The level of detection is such that proper system operation is guaranteed above this level. Below this voltage level, the Vcc under voltage detector will assert a Flag.

7.5.5 Under-Voltage on VDD

The Vdd is the regulated internal operating voltage of the product. The detector of Under- Voltage on Vdd is directly wired to POR. When this fault occurs, a reset will be triggered.

Note This is not maskable and cannot be changed from a Reset as the default fault condition.

7.5.6 Under-Frequency

The Under-Frequency detector is used to determine if the operating frequency is below 1MHz. Detection is performed on a clock-cycle-by clock-cycle basis.

7.5.7 Over-Frequency

As the ISO_CLOCK input can be any frequency the Over-Frequency detector is used on the chip-level reference clock input to reduce the ISO_CLK speed to 20MHz. This is preformed on a cycle-by-cycle basis.

7.5.8 Temperature Sensor

The Temperature Sensor generates an active-high output whenever the chip operating temperature is outside the allowed temperature range.

7.5.9 Ultra Violet Detectors

The function of the Ultra Violet (UV) Detector is to protect the NVM against attack from UV light. The UV light detector circuit will trigger well before any data in the NVM is compromised. Upon detection, a fault is generated. Depending upon the Users setting, an interrupt or RESET can be generated.

7.5.10 VCC Glitch Detection

The Vcc Glitch Detection will be set when a glitch of approximately 0.01V/ns is detected.

7.5.11 VCC Brownout Detection

The Vcc Brownout Detector sets a warning bit alerting the system that the power is low. This bit is triggered 200mV above the Vcc Under-Voltage detect voltage level.

Note This sensor does not act as an attack detector. Some handsets switch OFF suddenly when the battery is low, possibly preventing completion of a SIM transaction. The Brownout detector gives a warning before battery shutdown occurs. The warning flags can be used for anti-tearing. All SIM transactions should be blocked or ignored when this flag is set.

7.6 Security Mask

Reset Value	FFh (Sensors enabled)
Value after Firmware Start up Code execution	00h (Sensors disabled)
Access	BootRom access only

(MSB) (LSB)

-	-	nMask Brown-out	nMask Over/Under Temperature	nMask Over/Under Frequency	nMask UV Detect	nMask Over/Under Voltage	nMask Glitch Detect

Note This Mask is programmable via the Set_Security_Mask API.

7.7 Security Status SFR-B0h

Reset Value	00h
Value after Firmware Start up Code execution	00h
Access	No restriction

(MSB) (LSB)

PFLT	-	Brown-out	Over/Under Temperature	Over/Under Frequency	UV Detect	Over/Under Voltage	Glitch

Notes Bits are set by HW asynchronously. This is to make sure that fault signals from the analog blocks will always latch the appropriate bit in the SFR, regardless of whether the condition overlaps a rising clock edge.

The Status SFR is clocked by the un-gated version of the CPU clock. It also works in the Idle mode.

The fault conditions: Illegal access of system memory, Over/Under Temperature, Over/Under Frequency, UV Detect and Over/Under Voltage go through their masks and or OR'ed together. The output of the OR gate will not be latched, but instead pass through the Physical Fault Register (PFLT) to an interrupt request.

Certain faults can be set by hardware, but not cleared by hardware. They can be set or cleared by software.

These consist of:

- Illegal access of system memory
- Over/Under Temperature
- Over/Under Frequency
- UV Detect
- Over/Under Voltage

The PFLT condition can be used to trigger a reset. Shown below is the SFR that controls this function.

7.8 Security Action

Reset Value	x1xx xxxxb (by **default**, faults cause an **interrupt**)	
Value after Firmware Start up Code execution	x1xx xxxxb	The value of this SFR remains the same as the hardware default value. Fault sensors, if enabled, will trigger an interrupt on default.
Access	BootRom access only	

(MSB) (LSB)

-	IntOnFault	Enpwrdwn	Cntrlpwrdwn	-	-	-	-

IntOnFault	1 – Physical Fault causes an interrupt 0 – Physical Fault cause reset immediately
Enpwrdwn	Set by manufacturer
Cntrlpwrdwn	Set by manufacturer
All other bits	Unimplemented

Notes This Mask is programmable via the Set_Security_Action API.

If the User wants to set an Interrupt for some sensors and a Reset for some others they should set the Interrupt by default. Then depending on the sensor triggered, Reset the IntOnFault bit and toggle a sensor bit with the Security_Status SFR to trigger a new security sensor fault.

7.8.1 Non-permanent changes to Security Action

Action	Result
Set the chip to interrupt temporarily when the chip is defaulted to interrupt always	- No error returned from the API call - The chip will interrupt on **all** security faults **and will not change its behaviour**
Set the chip to reset temporarily when the chip is defaulted to interrupt always	- No error returned from the API call - The chip will reset on the **first** security fault **and will interrupt on all subsequent security faults**
Set the chip to interrupt temporarily when the chip is defaulted to reset always	- No error returned from the API call - The chip will interrupt on **all** security faults **until a chip reset occurs.** The chip will then **reset on all subsequent security faults**
Set the chip to reset temporarily when the chip is defaulted to reset always	- No error returned from the API call - The chip will reset on **all** security faults **and will not change behaviour**

7.8.2 Permanent changes to the Security Action

Action	Result
Set the chip to interrupt permanently when the chip is defaulted to interrupt always	- No error is returned from the API call - The chip will interrupt on **all** security faults **and will not change behaviour**.
Set the chip to reset permanently when the chip is defaulted to interrupt always	- No error is returned from the API call - The chip will reset on **all** security faults **and will not change behaviour**.
Set the chip to interrupt permanently when the chip is defaulted to reset permanently	- **Fail** > This change is not allowed i.e. once a reset permanent has been set, the chip can **never** be set to interrupt permanently - The chip should continue to reset on **all** security faults **and will not change behaviour**.
Set the chip to reset permanently when the chip is defaulted to reset permanently	- No error is returned from the API call - The chip should reset on **all** security faults **and will not change behaviour**.

8. Interrupts

8.1 Overview

The device has an interrupt controller that is based on that in the standard 80C51 device. It supports 2 levels of interrupt priorities: Low and High.

A list of the interrupts and their vector addresses is shown in Table 13.

Source	Name	Vector address	Priority	Polling order
Timer 0	IT0	080Bh	Low or High	1st
ISO Serial	ISP (RI or TI)	0823h	Low or High	2nd
Fault Interrupt	FLT	082Bh	High	3rd

Table 13: Interrupts

Each of these interrupts can be enabled/disabled by software by setting/clearing the enable bits in the SFR Interrupt Enable (IE) register. There is a bit (EA) in the IE register that can disable all interrupts when cleared.

8.2 Interrupt Enable Register

SFR Address	**A8h**
Value at Hardware Reset	00h
Value after Firmware Start Up Code Execution	3Fh
Access	No access restriction

(MSB) (LSB)

EA	-	EFLT	ES	-	-	ET0	-

EA	EA=0: All interrupts disabled EA=1: Each interrupt is individually enabled/disabled by setting/clearing its enable bit
-	Not implemented
EFLT	Fault interrupt enable. 1 – Enabled, 0 – Disabled
ES	ISO Serial interrupt enable bit. 1 – Enabled, 0 – Disabled
ET0	Timer 0 interrupt enable. 1 – Enabled, 0 – Disabled

An interrupt can have low or high priority. This means:

1. A high priority interrupt can interrupt an on-going low priority interrupt.
2. When a high priority interrupt and low priority interrupt condition are detected simultaneously (meaning they are detected during the same CPU cycle), the high priority interrupt is serviced first.
3. When interrupts of same priority are detected at the same time, the interrupt with higher polling priority will be serviced first. See Table 13 for polling priorities.

8.3 Interrupt Priority Register

SFR Address	B8h
Value at Hardware Reset	00h
Value after Firmware Start Up Code Execution	3Eh
Access	No restriction

MSB LSB

-	-	PFLT	PS	-	-	PT0	-

-	Not implemented
PFLT	Fault Interrupt priority 1 - High priority, 0 - low priority
PS	ISO Serial Interrupt priority. 1 - High priority, 0 - low priority
PT0	Timer 0 interrupt priority 1 - High priority, 0 - low priority

8.4 TCON

When two or more interrupts of the same polling priority are detected, the interrupt with the higher polling priority is serviced. Polling of interrupts is done in a fixed order with the first interrupt in the polling order having the highest priority.

SFR Address	88h
Value at Hardware Reset	00h
Value after Firmware Start Up Code Execution	00h
Access	No restriction

MSB LSB

-	-	TF0*	TR0*	-	-	-	-

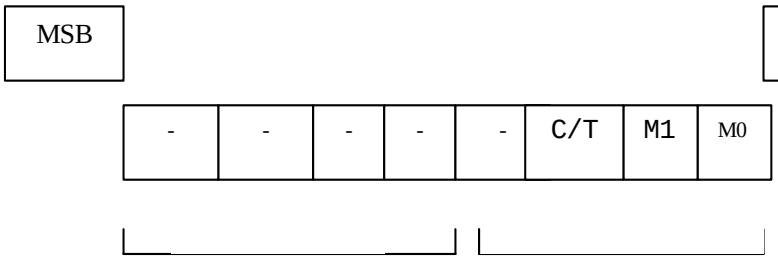
-	Not implemented.
TF0*	Timer0 overflow flag and run bit
TR0*	Timer0 overflow flag and run bit

9. Timer

The Theseus Titanium 80 has one standard 80C51 timer with ETU counting option.

9.1 Timer

The control bits C/T in the SFR TMOD register select the “Timer” or “Counter” function. This Timer/Counter functions has three possible operating modes, which are selected by bit-pairs (M1, M0) in TMOD, see Figure 6:



C/T Timer or Counter Selector cleared for Timer function (input from internal system clock).

- 0 = Timer Source: CPU Clock/4
- 1 = Timer Source: ETU

Set for Counter function.

M1	M0	OPERATING
0	0	13-bit Timer “TLx” serves as 5-bit prescaler.
0	1	16-bit Timer/Counter “THx” and “TLx” are cascaded; there is no prescaler.
1	0	8-bit auto-reload Timer/Counter “THx” holds a value which is to be reloaded into “TLx” each time it overflows.
1	1	Reserved

Figure 6: Timer/Counter Mode Control (TMOD) Register

The Timer/Counter operating Modes are described as follows:

9.1.1 Mode 0

Mode 0 provides a legacy behaviour to the 8048 device behaviour. Putting the Timer into Mode 0 will make it function as an 8-bit Counter with a 5 bit pre-scaler. Figure 7 shows the Mode 0 operation as it applies to Timer 0.

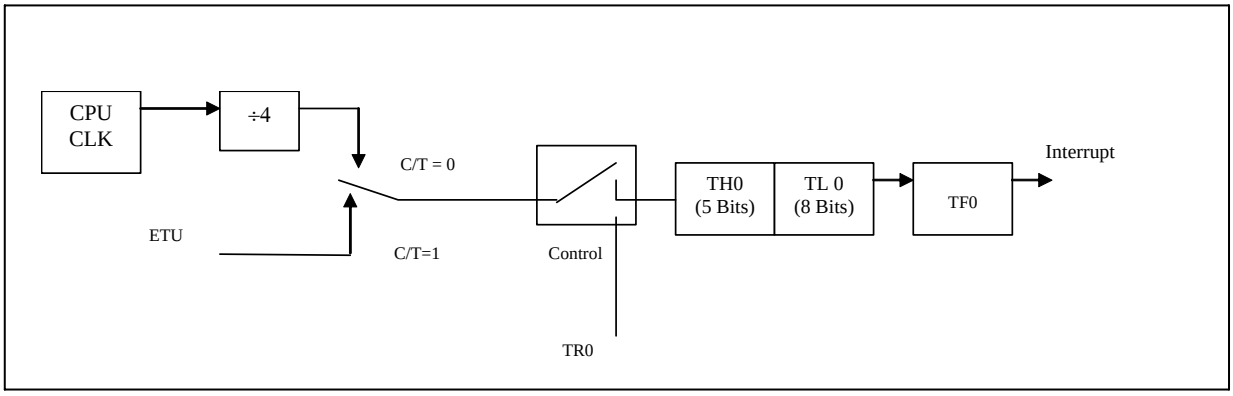
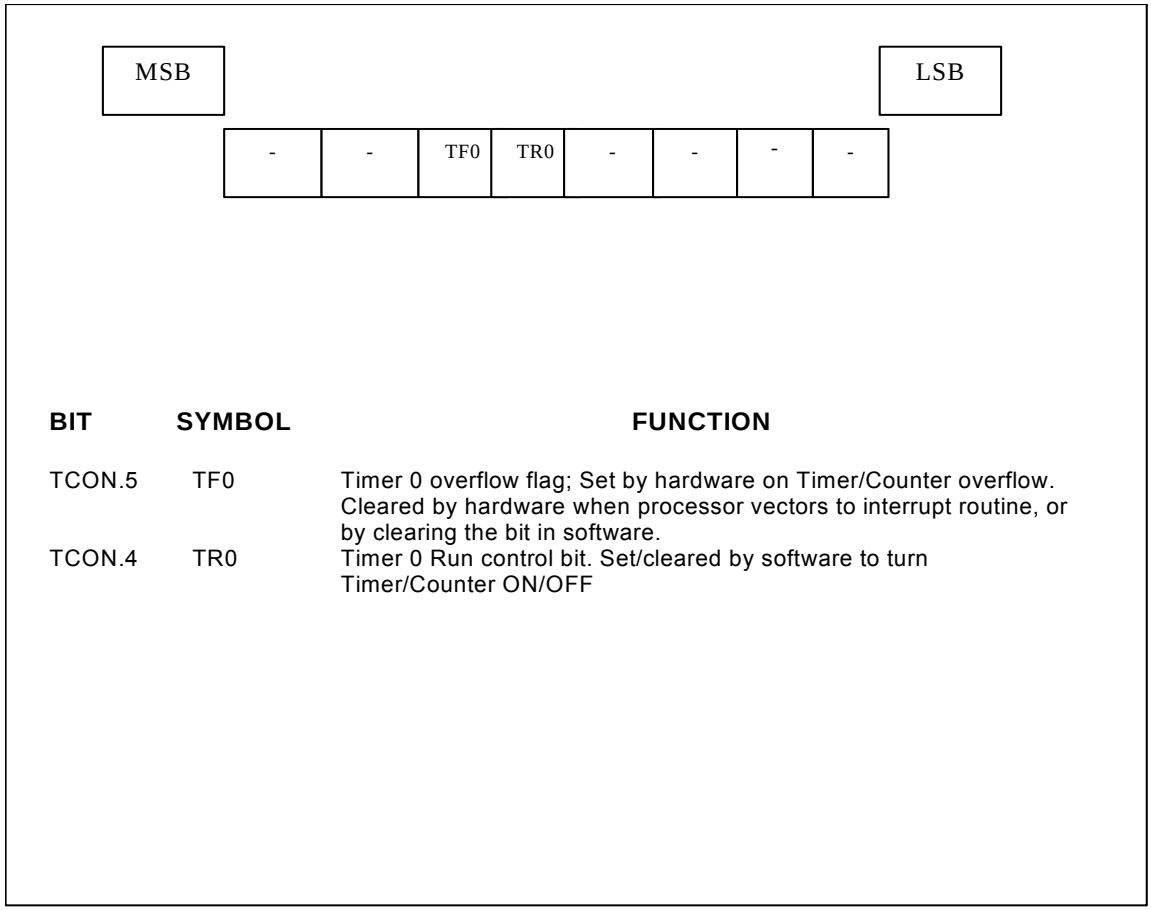


Figure 7: Timer/Counter Mode 0: 13-Bit Counter/timer

In this mode, the Timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the Timer interrupt flag TF0. The counted input is enabled to the Timer when TR0 = 1. TR0 is a control bit in the Special Function Register TCON (see Figure 8). The 13-bit register consists of all 8 bits of TL0 and the lower 5 bits of TH0. The upper 3 bits of TH0 are unused in this mode.



BIT	SYMBOL	FUNCTION
TCON.5	TF0	Timer 0 overflow flag; Set by hardware on Timer/Counter overflow. Cleared by hardware when processor vectors to interrupt routine, or by clearing the bit in software.
TCON.4	TR0	Timer 0 Run control bit. Set/cleared by software to turn Timer/Counter ON/OFF

Figure 8: Timer/Counter Control (TCON) Register

Note The TF0 and TR0 bits are implemented to support Mode 3 operation for the Timer.

9.1.2 Mode 1

In Mode 1, all 8 bits of the TL0 register are used. Utilizing both TH0 with TL0 provides a 16 bit counter. In Mode 1, TH0 is the high byte and TL0 is the low byte.

9.1.3 Mode 2

Mode 2 configures the Timer register as an 8-bit Counter (TL0) with automatic reload, as shown in Figure 9. Overflow from TL0 sets TF0 and reloads TL0 with the contents of TH0, which is preset by software.

The reload leaves TH0 unchanged.

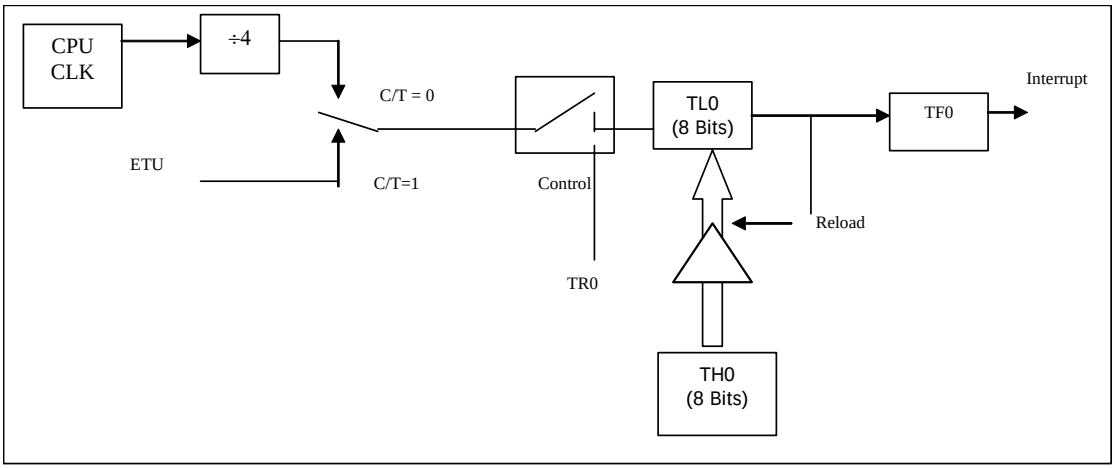


Figure 9: Timer/Counter Mode 2: 8-Bit Auto-Load

9.2 TCON SFR Register

SFR Address	**88h**
Value at Hardware Reset	00h
Value after Firmware Start Up Code Execution	00h
Reset Value	00h
Access	No restriction

(MSB) (LSB)

-	-	TF0	TR0	-	-	-	-

-	Not implemented
TF0	Timer 0 overflow flag and run bit
TR0	Timer 0 overflow flag and run bit

9.3 TMOD SFR Register

SFR Address	**89h**
Value at Hardware Reset	00h
Value after Firmware Start Up Code Execution	00h
Reset Value	00h
Access Restriction	No restriction

(MSB) (LSB)

-	-	-	-	-	C/T	M1	M0

-	Not implemented.
C/T	For Timer 0
M1	
M0	

9.4 TL0 SFR Register

SFR Address	8Ah
Value at Hardware Reset	00h
Value after Firmware Start Up Code Execution	00h
Reset Value	00h
Access Restriction	No restriction

(MSB)							(LSB)
TL0.7	TL0.6	TL0.5	TL0.4	TL0.3	TL0.2	TL0.1	TL0.0

9.5 TH0 SFR Register

SFR Address	8Ch
Value at Hardware Reset	00h
Value after Firmware Start Up Code Execution	00h
Reset Value	00h
Access Restriction	No restriction

(MSB)							(LSB)
TH0.7	TH0.6	TH0.5	TH0.4	TH0.3	TH0.2	TH0.1	TH0.0

This page is intentionally left blank

10. Random Number Generator

10.1 Random Number Generator Overview

The Random Number Generator (RNG) is dedicated hardware implemented using a combination of Analog and Digital circuits that result in a fully compliant FIPS140-2 RNG.

The RNG provides an 8-bit value each time it is read. Values can be combined from several readings of the RNG to generate larger random numbers.

- To ensure backwards compatibility with other members of the Emosyn Platinum family of products, an API is provided to read the RNG value
- To guarantee true randomness and FIPS140-2 compliance, **there must be a minimum of 20us delay** between RNG reads. If the RNG is read at intervals below 20usec true randomness cannot be guaranteed

API	unsigned char Firmware_Get_Random_Number (void)
Jumptable Address	0x01E6
Inputs	None
Outputs	R7 Value
Stack Usage	0
Register Usage	0

This page is intentionally left blank

11. Idle Mode & Deep Idle Mode (Stop Clock Mode)

11.1 Idle Mode & Deep Idle (Stop Clock Mode) Overview

The Theseus Titanium 80 supports the standard 8051 Idle Mode & Deep Idle (Stop Clock Mode) enhanced with system-wide power saving capabilities.

Note High speed processing required by a Mobile Interface can provide little time for device response. If no extra guard time is provided by the Mobile Interface, the SIM card may not have enough processing time to switch between the internal clock, ISO clock, and Idle mode. **Due to the time required for the device to process a "Set Clock" API, this API should not be called just before or after entering into Idle Mode & Deep Idle (Stop Clock Mode).**

11.2 External Clock required for exit idle operations

In order to come out of Idle & Deep Idle (Stop Clock Mode), the ISO_CLK pin must continue to be clocked externally:

- The device **Serial port** and **Idle control logic** are clocked by ISO_CLK directly

11.3 Power switching in Idle

- To guarantee GSM 11.12 Idle Current compliance, Emosyn recommends setting the Set Clock API to Direct ISOCLK prior to entering Idle. This will shut OFF power to the Internal Oscillator
- When the SIM is in clock stop Idle mode, the current consumption is <100μA (maximum). When the SIM is in idle mode with the ISO clock @1MHz, the current consumption is <200uA. To guarantee GSM 11.12, 11.11 Idle Current compliance, the inputs CLK and RESET must be driven to CMOS levels. Specifically, these levels are Vih = VCC +/- 0.3V, Vil = VSS +/- 0.3V
- Emosyn recommends that when the software goes into IDLE mode without switching to the Direct ISOCLK, the User **MUST** insert two NOP instructions following the instruction that puts the device into IDLE mode

11.4 Entering in Idle or Deep Idle Mode (Stop Clock Mode)

Idle Mode:

- The device will enter into Idle mode by setting the PCON.0 to "1". This SFR can be accessed directly

Deep Idle Mode or Stop Clock Mode:

- Two bits are controlling this mode in SFR "Security Action": Enpwrdwn and Cntrlpwrdwn bits. This SFR can be accessed via the Set Security Action Firmware API. The High Level is the preferred level. ISO11.11, "File Characteristic" byte returned to the Select MF or DF APDU command refers to this Stop Clock Mode

11.5 The Way to Exit Idle & Deep Idle Mode (Stop Clock Mode)

The 3 ways out of Idle & Deep Idle (Stop Clock Mode) are:

- When the ISO Port receives a start-bit
- An enabled Interrupt occurs
- When the RI flag goes high

Notes Due to some issues that could arise with some handsets, it is recommended to enter in Idle mode immediately after the last byte of the ATR. Entering into Idle Mode only after the reception of the first APDU could freeze the handset because the handset does not detect Low power consumption after the ATR.

Due to the architecture of the silicon, the User should ensure that the instructions executed do not exceed the time of the 2 bits of "guard time" during the following:

- The reception of 2 characters of an APDU
- Sending of the last character of the ATR or an APDU (SW2) and the reception of the first character of the next APDU. Exceeding this time could freeze the device if it enters in Idle mode "too late" or some characters could be "missed" by the device

12. Serial Interface

12.1 ISO 7816-3 Serial Interface Implementation

The Theseus Titanium 80 has a serial interface compliant with the ISO 7816-3 specification. Several modes are implemented that allow standard serial connections from under 9600 to over 115K bits per second.

The Theseus Titanium 80 can also support several divide ratios on the External clock (see Section 11.2). The product supports T=0 asynchronous half-duplex character transmission protocol, T=1 asynchronous half duplex block transmission and a proprietary protocol which the card manufacturer can use for fast loading of code.

12.1.1 ISO 7816- 3 Interface Block Diagram (Clock and Security)

The ISO 7816-3 module is a totally self contained ISO 7816-3 interface that includes the ISO clock divider used to generate the appropriate baud rate, the hardware parity checking and generating logic and Receive/Transmit register. Figure 10 shows this in overview.

The Receive/Transmit register consists of a bi-directional shift register clocked by the output of the baud rate generator. The direction is dependent on whether the transmit buffer is loaded for a transmission or a start bit has been detected starting a reception cycle. Though the programmer sees only an 8-bit bi-directional register Serial Buffer (SBUF), the hardware is realised by two registers, an 8-bit Transmit holding register and an 8-bit Receive register. Writing to SBUF loads the Transmit holding register; reading from SBUF returns the contents of the Receive holding register.

When the serial port is sending the data written to SBUF it is shifted serially to the ISO DATA pin through the parity generator/checker logic. The processing is dependent on the setting of the Parity Error Management Enable (PEME). This bits function is to turn ON and OFF the parity capability of the ISO 7816-3 interface.

The parity generator hardware automatically undertakes the parity requirements of the ISO 7816-3 interface, its function differs dependent on whether the Theseus Titanium 80 is transmitting or receiving data. When the serial port is transmitting data and the PEME is enabled, the parity generator appends the correct parity bit following the 8-bit read from the SBUF register. It then monitors the serial data bit to see if the receiving device asserts the Resend bit.

The detected ‘RESEND’ condition is made available or ‘flagged’ to the programmer via the Resend bit in the ISOCON register (ISOCON.3). In normal operation the application will have to check the TI (SCON and IE references) flag, which indicates the transmission, has completed.

The application needs to reload the SBUF register and repeat the transmission of the last byte. If the Resend bit has been set then the terminal did not receive the byte successfully.

When the Theseus Titanium 80 is receiving a character and the PEME bit is set, the parity is automatically checked:

- If there is no parity error then the Receive Interrupt (RI) bit is set
- If there is a parity error then the parity checker asserts the Resend condition on the serial line, to inform the transmitting device that the byte was not received correctly and it needs to transmit the byte again

The RI bit is not set when Resend is asserted so the Application is only flagged when good bytes have been received.

Not all serial transactions use parity and hence there is no parity bit or Resend bit in the character. If the PEME is disabled, the parity bit is not checked and no Resend bit will be generated. The ISO Timer (ISOTMR) is programmed to get the desired divide ratio from the received ISOCLK to achieve the correct baud rate.

The ISOTMR is an 11-bit counter that is loaded through the 8 bit ISOTMR SFR, using two 'WRITES' to the same register to transfer over the 11 bits needed by the divider:

- The first 'WRITE' transfers the three most significant bits 10,9,8 with the bits in ISOTMR.2, ISOTMR.1, and ISOTMR.0 respectively
- The second 'WRITE' transfers the lower 8 bits

The ISOTMR is a down counter that gives an output and reloads the value loaded into the ISOTMR register when the count reaches zero. Because of this the value loaded into the ISOTMR register is one smaller than the desired Elementary Time Unit (ETU). This allows the 11-bit counter of Theseus Titanium 80 to operate with a maximum division ratio of (0111 1111 1111) b or 07FFh = 2047 = 2048 ISOCLKS per ETU, and a minimum division ratio of (0000 0000 0000) b = 0 = 1 ISOCLKS per ETU.

The real ETU period is not only dependent on the timer but also on the clock frequency applied to the ISOCLK pin. If the frequency monitoring is disabled this can be from 0 to 10 MHz.

12.1.2 Active drive high at the start of the guard time

During ISO communication the device will make the ISODATA I/O pin tri-stated during the Guard Period. Typically a pull-up resistor on the ISODATA line (usually present in the Terminal) will hold the ISODATA line at weak high state. However in some systems, the capacitance on the ISODATA is very large, which causes the 0 - to -1 transition on ISODATA to be very large.

The slow rise-time can be misinterpreted by the Terminal as a Resend signal, causing communication failures between the card and the Terminal.

The Titanium 80 provides a unique feature that causes the card to drive the ISO data line high for a programmable period of time before leaving the line to tri-state. The SFR bits are used to configure the drive high time are GT0, GT1. These two bits are implemented in the SFR ISOCON as bit 5 and 4.

GT0, GT1	Drive high period in ISO clock
0 0	0
0 1	1
1 0	2
1 1	4

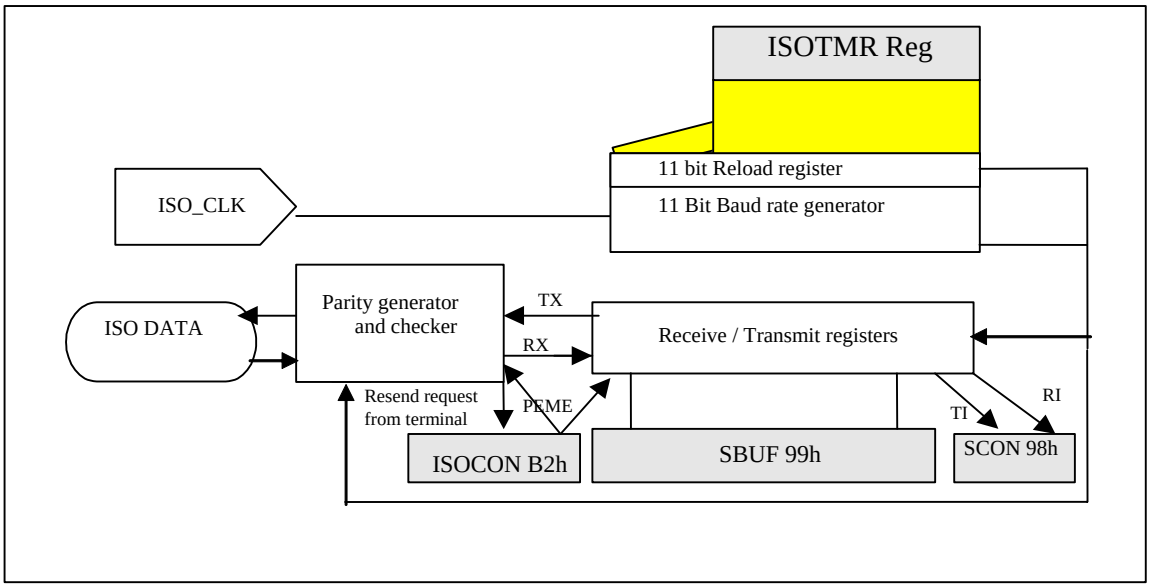


Figure 10: Receive and Transmit Registers

12.1.3 ISO 7816 Control Registers

The ISO 7816 Control Registers are shown in the following tables:

12.1.4 SCON SFR-98h

	SCON .7	SCON .6	SCON .5	SCON .4	SCON .3	SCON .2	SCON .1	SCON .0
	-	-	-	-	-	-	TxDone	RxDone
VALUE AT HARDWARE RESET							00h	
VALUE AFTER FIRMWARE START UP CODE EXECUTION							00h	
HARDWARE SET							✓	✓
HARDWARE CLEAR								
USER CODE READ							✓	✓
USER CODE WRITES							✓	✓

Symbol	Position	Function
-	SCON.7: SCON. 2	RES
TxDone	SCON. 1	Set High when byte written to SFR Serial Data has been fully transmitted
RxDone	SCON. 0	Set High when new byte has been received

12.1.5 SBUF SFR-99h

SFR SBUF								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Function	**Data**							
Read/Write	**RW**	**RW**	**RW**	**RW**	**RW**	**RW**	**RW**	**RW**
Data	Read: Last complete byte received Write: Initiates transmission of byte written							
Value at Hardware Reset	00h							
Value after Firmware Start Up Code Execution	00h							

12.1.6 ISOCON SFR-B2h

SFR ISOCON								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Function	**0**	**RIWK**	**GT1**	**GT0**	**Resend**	**PEME**	**0**	**0**
Read/Write	**R**	**RW**	**RW**	**RW**	**R**	**RW**	**R**	**R**
Resend	'1' when the serial port has detected that a resend of the last transmitted byte has been requested by the remote receiver							
PEME	'1' enables hardware parity checking and generation							
GT0, GT1	Active drive high control at the start of the guard time							
RIWK	When set, RI flag will wake the chip up from idle even when ISO interrupt is not enabled							
Value at Hardware Reset	00h							
Value after Firmware Start Up Code Execution	54h							

12.1.7 ISOTMR SFR-B4h

SFR ISOTMR								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Function (Last write)	b7	b6	b5	b4	b3	b2	b1	b0
(Previous write)						b10	b9	b8
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW
Divisor	Number of ISOCLK periods for each serial bit time -1 (etu). This register is 11 bits wide: write it with b10..b8 first, then write it a second time with b7..b0.							
Value at Hardware Reset	00h, 00h							
Value after Firmware Start Up Code Execution	01h, 73h							

12.1.8 I/O Pull-up Resistor SFR-C0h

Reset Value	00h
Value after Firmware Start up Code execution	00h
Access	No restriction

(MSB) (LSB)

-	-	-	-	Spare	Spare	Spare	EnIOPullup

Bit	Name	Port	Function
0	EnIOPullup	en_iopullup	0 – I/O pull up resistor not enabled 1 – I/O pull up resistor enabled when IO port is not actively driven by ISO serial UART
1-3	Spare(s)	-	-
4-7	-	-	Not implemented

Notes The I/O Pull-up Resistor is only available on version 2.0 and above of the Theseus Titanium 80 devices.

If enabled, the resistor becomes active approximately ~40usec after a device reset

The Theseus Titanium integrated pull-up resistor has a nominal value of 20 kohm. The actual value will depend on external voltage supply (Vcc) and temperature. The integrated resistance will be sufficient to properly raise the I/O line voltage if the equipment lacks a suitable pull-up resistor.

Please contact your local FAE if you have any questions regarding the device performance with the pull-up resistor implemented.

12.1.9 Detailed Timing of ISO7816 Interface

With the PEME bit not set (i.e. Non Parity Mode) the ISO Interface-Timing diagram is shown in Figure 11. The receive Interrupt is set at 9 etu's, and the SBUF can be loaded at this point and the transmission will start at 11 etu's + 1 ISO_CLK. The serial port is able to receive a character after 10 etu's + 1 ISO Clock.

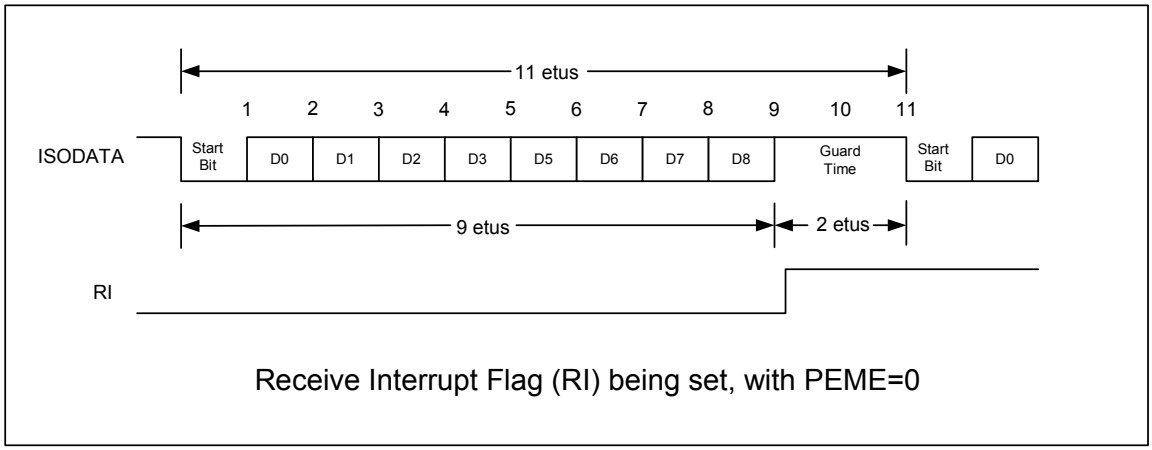


Figure 11: ISO Interface Timing

The timing of the serial port receiving characters when the parity check indicates a valid character has been received. The RI flag (in the SCON register) is set 10 etu's after the start bit detection. At this time it is permissible to load a character to SBUF for transmission. This transmission will be held pending until the period of 12-etu has expired then the transmission will commence on the first ISO_CLK.

The serial port will receive a character after 11 etu's + 1 ISO_CLK. This ensures that the receive period required by the ISO 7816 specification is met, but allows the application programmer the maximum opportunity to utilise the available ISO serial bandwidth available, see Figure 12.

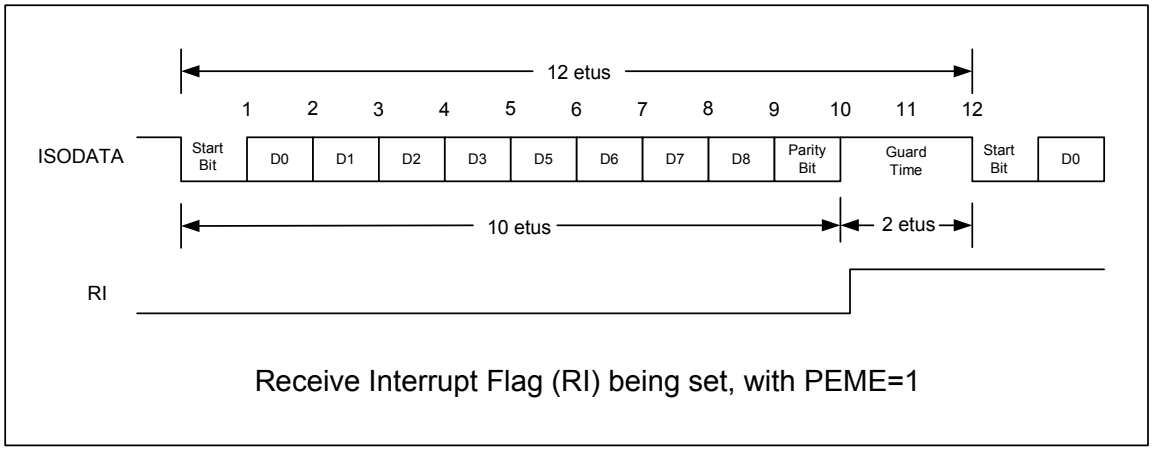


Figure 12: Receive Interrupt Flag, with PEME =1

When the character receive parity fails the Theseus Titanium 80 asserts the Resend bit on the ISO_DATA pin, by holding the line at 0 starting at 10.5 etu's and maintaining this level for 1.5 etu's, the RI flag is not set.

Figure 13 shows the Receive Interrupt (RI) Flag timing, Parity Fail.

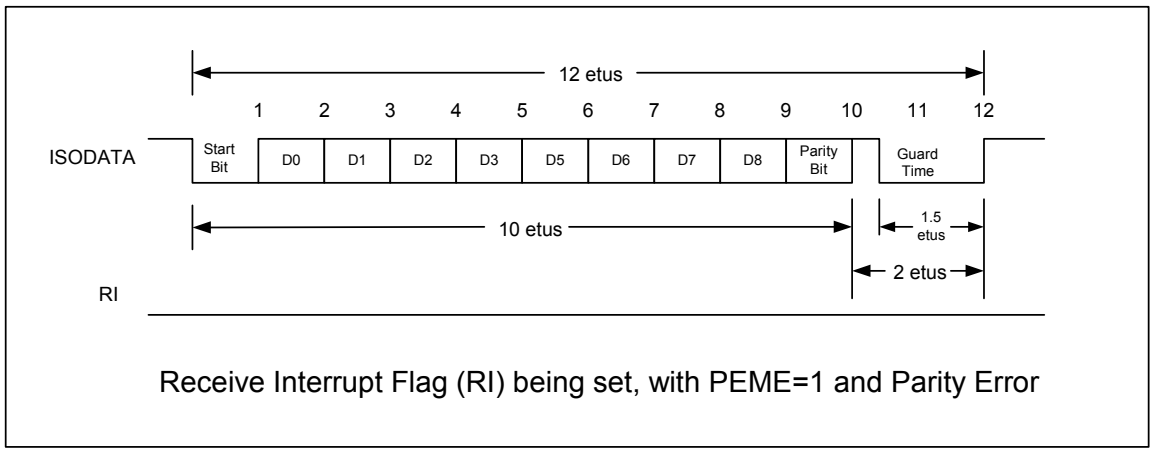


Figure 13: Receive Interrupt Flag timing, Parity Fail

During transmission of characters the behaviour of the serial port is dependent on the setting of the PEME bit. With the PEME bit not set parity is disabled, the timing is shown in Figure 14:

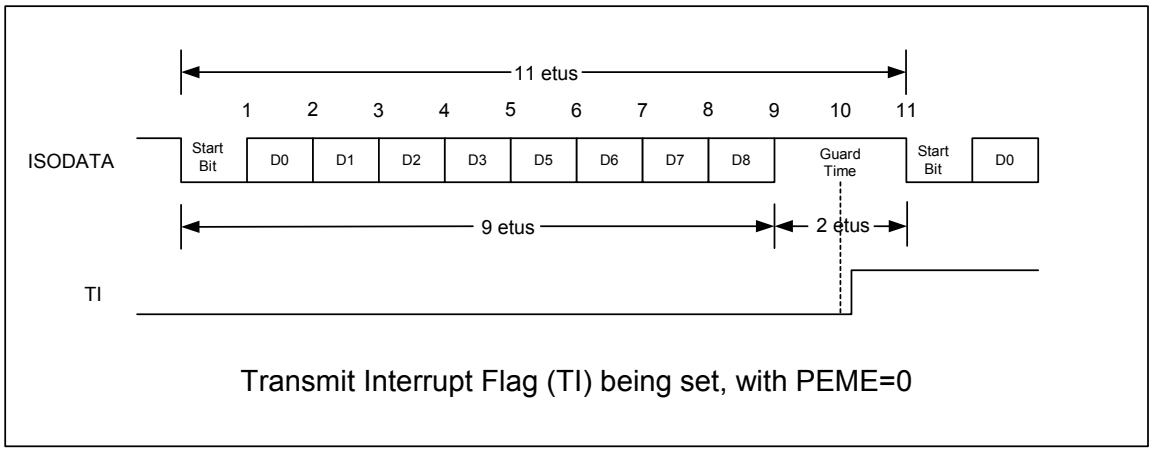
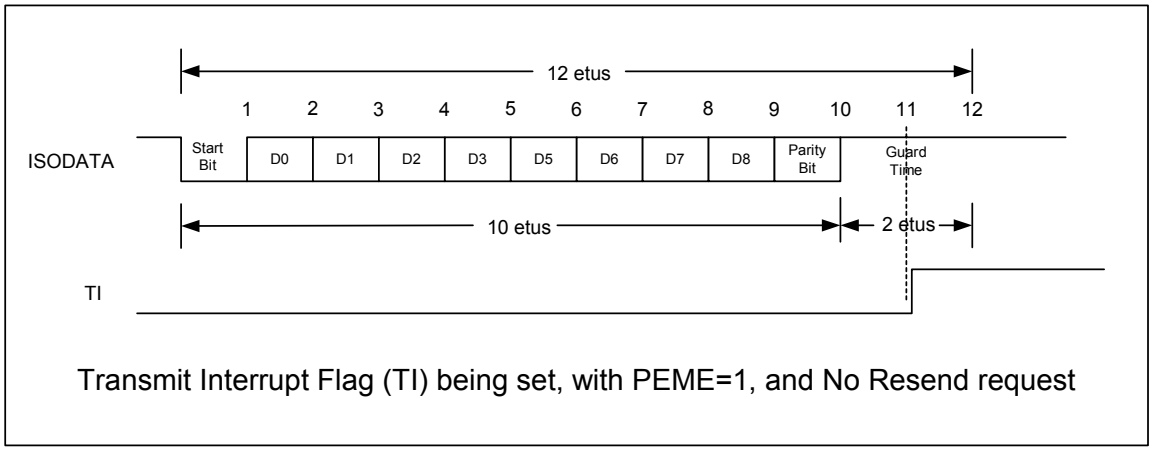


Figure 14: Transmit Timing PEME disabled

The SBUF can be loaded when the TI flag is true; the new character is not transmitted until the 2-etu-guard time has expired.

Note There is a two ISO clock delay from the point of loading the first character into SBUF and it appearing on the ISO DATA pin.

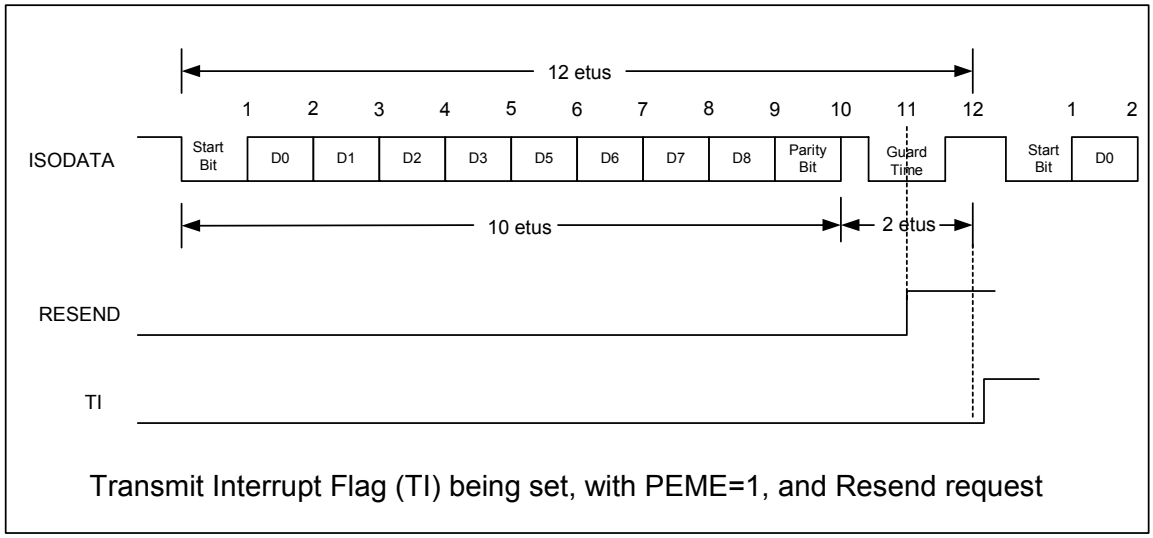
With the PEME bit set the behaviour of the serial port is dependent on whether the Theseus Titanium 80 is receiving a Resend bit or not. The timing of when no Resend bit is asserted is as shown in Figure 15:



Transmit Interrupt Flag (TI) being set, with PEME=1, and No Resend request

Figure 15: PEME bit set No Resend request

The TI flag is set at 11 etu's, at this time SBUF can be loaded with a new character, which will be transmitted after the 12 etu transmit cycle of the previous character has elapsed. If the Resend bit is received from the port terminal then the timing that are shown in Figure 16 apply:



Transmit Interrupt Flag (TI) being set, with PEME=1, and Resend request

Figure 16: PEME Set and Resend requested

An example of the software response to the characters that have been transmitted is shown below:

```
_PutChar:
PutChar:
        mov     a,R7

        push    ACC
        mov     SBUF, a                 ; Write ACC directly to the Serial Buffer

        jnb     TI, $                   ; Only move on when the TI flag goes
                                        high.
        clr     TI

        mov     a, ISOCON
        jnb     ACC.3, PutChar_Done     ; If Serial Control.3 is set after tx
                                        complete
                                        ; A resend is required
        pop     ACC
        anl     ISOCON, #0F7h           ; Reset RESEND bit
        ajmp    PutChar_Value_In_Acc    ; Try again

PutChar_Done:
        pop     ACC
        ret
```

The Theseus Titanium 80 samples the Resend bit at 11 etu's, at this time the Resend bit is set. The TI flag is set one etu later. At this time it is permissible to reload the last character sent, which will be transmitted starting on the 13th etu following the previous transmission.

12.1.10 Standard Baud Rates

The ISOTMR 11 bit divider allows baud rate divisions of the incoming clock from a 1/1 to 1/2048. Standard reader implementations refer to baud rates that assume a fixed ISO clock being applied to the device.

The ATR response is always referred to at 9600 baud, as this is the baud rate seen when the standard clock is applied and the division rate in the device is /372. To obtain this in the Theseus Titanium 80 the following sequence should be followed:

- Mov ISOTMR, #01H
- Mov ISOTMR, #73H

This loads the 11-bit ISO Timer (divider) with 173 Hexadecimal, which is 371 decimal, the desired divide rate 372 minus one as needed by the ISOTMR. The common data baud rates used in Global Systems for Mobile communication (GSM) applications and the values to be loaded are as follows:

ISO_CLK periods per ETU	Required ISOTMR Register Value	First Byte to write to ISOTMR	Second Bytes to write to ISOTMR
372	371d = 0173h	01h	73h
93	92h = 005Ch	00h	5Ch
64	63h = 003Fh	00h	3Fh
62	61h = 003Dh	00h	3Dh
32	31h = 001Fh	00	1Fh
31	30h = 001Eh	00	1Eh

This page is intentionally left blank

13. Flash Non-Volatile Memory

13.1 Principles

The Theseus Titanium 80 utilizes CMOS SuperFlash EEPROM licensed technology from Silicon Storage Technology (SST) Incorporated in the NVM blocks.

This technology provides many advantages over masked ROM and traditional EEPROM when designing embedded memories. Emosyn has built on this technology to provide maximum performance and to allow the same memory cell to be used for both Code and Data storage depending on the architecture of the memory array.

The main User Configurable Memory (UCM) features that are used within the Theseus Titanium 80 Non-Volatile Memory (NVM) are shown in Table 14:

User Configurable Memory	Comments
80K x 8 bits CMOS page mode NV memory	
Page Erase (256/512 bytes)	2ms typical
Mass Erase	5ms
Endurance	250K Erase/Write-cycles (typical)
Data retention	Greater than 10 Years
Low power consumption	Yes
Associated attribute bits	Yes
Fast byte write	Yes
Byte Write	25 μsec typical

Table 14: Configurable Memory main features

13.2 User Configurable Memory (UCM)

The page erase size is configurable between 256 and 512. The memory can also be erased in blocks. The exact block sizes are defined in Table 15.

Associated with each UCM block are two security attributes, these enable the User to define how these blocks behave:

ROM Attribute:

- If this attribute is not set, the associated UCM block will behave as EEPROM with 256/512 byte page size
- If the ROM attribute is set on an individual block, that block cannot be erased by either block erase or mass erase operations
- Execute Only Attribute once set, read access to this memory block is only permitted by the processor code that is running from the block. This effectively stops an attempt by software running from any other area of memory from ‘snooping’ the program contained in the protected block

Note If the User wants to prevent unauthorized access operations, then the Operating System should select **BOTH** the Protect attribute and the Execute-Only feature. It is recommended that the Execute Only attribute is ***always*** used together with the Protect attribute.

Block #	Bytes to Pass into API call		Block size	Logical Address	CODE Space Address	XDATA Space Address
	R6	R7				
0	00h	01h	8k	0000 -1FFF	0800 - 27FF	-
1	00h	02h	6K	2000 - 37FF	27FF - 3FFF	-
2	00h	04h	4K	3800 - 4700	4000 - 4FFF	-
3	00h	08h	2K	4800 - 4FFF	5000 - 57FF	0800 - 0FFF
4	00h	10h	2K	5000 - 57FF	5800 - 5FFF	1000 -17FF
5	00h	20h	2K	5800 - 5FFF	6000 - 67FF	1800 -1FFF
6	00h	40h	2K	6000 - 67FF	6800 - 6FFF	2000 -27FF
7	00h	80h	8K	6800 - 87FF	7000 - 8FFF	2800 - 47FF
8	01h	00h	8K	8800 - A7FF	9000 - AFFF	4800 - 67FF
9	02h	00h	16K	A800 - E7FF	B000 - EFFF	6800 - A7FF
10	04h	00h	8K	E800 -107FF	F000 - FFFF	A800 - C7FF
11	08h	00h	8K	10800 -127FF	-	C800 - E7FF
12	10h	00h	4K	12800 -137FF	-	E800 - F7FF
13	20h	00h	1K	13800 -13BFF	-	F800 - FBFF
14	40h	00h	1K	13C00 -13FFF	-	FC00 - FFFF

Table 15: UCM block sizes

The Erase_UCM_Block API can be used with the R6 and R7 parameters mentioned in Table 15 to erase Flash memory.

13.3 NvFSM Con SFR- 80h

	FSMCon .7	FSMCon .6	FSMCon .5	FSMCon .4	FSMCon .3	FSMCon .2	FSMCon .1	FSMCon .0
	RES	RES	Success	RES	RES	RES	Start	Done
Value at Hardware Reset						00h		
Value after Firmware Start Up Code Execution						00h		
HARDWARE SET			✓					✓
HARDWARE CLEAR			✓					
USER CODE READ			✓				✓	✓
USER CODE WRITES							✓	

Symbol	Position	Function
RES	FSM.7	Reserved for future use
RES	FSM.6	Reserved for future use
Success	FSM.5	1 = NVM completed successfully 0 = NVM operation failed
RES	FSM.4	Reserved for future use
RES	FSM.3	
RES	FSM.2	
Start	FSM.1	1: Start NVM operation When the User sets this bit, it will be read 1 as long as the FSM is busy. Once the FSM completes its operation, this flag will be read as 0.
Done	FSM.0	0: NVM operation in progress 1: NVM operation completed The SFR NvFsmCon.0 bit is a read/write bit that shows the NVM FSM completion. When the NVM FSM completes its operation, the hardware will set this flag. The software is responsible for reading this bit and clearing it. Upon reset, this bit will be read as "0". **Note** Set by hardware, when the NVM operation is finished (regardless of success or failure). Software needs to clear this bit after handling the NVM done event.

13.4 NvFSM Mode SFR-C9h

	NvFSMMode .7	NvFSM .6	NvFSM .5	NvFSM .4	NvFSM .3	NvFSM .2	NvFSM .1	NvFSM .0
	RES	Erase Mode: 2	Erase Mode: 1	Erase Mode: 0	Write Mode: 1	Write Mode: 0	RES	C/D
Value at Hardware Reset						00h		
Value after Firmware Start Up Code Execution						00h		
HARDWARE SET								
HARDWARE CLEAR								
USER CODE READ		✓	✓	✓	✓	✓	✓	✓
USER CODE WRITES		✓	✓	✓	✓	✓	✓	✓

Notes If you need to erase multiple blocks, the R6 and R7 values can be added and passed to the Erase_Block API.

The Mass Erase setting in NvFSM Mode will erase everything, including the bootloader.

Symbol	Position	Function
Erase Mode.2	NvFSM.6	000: No erase 001: 256 Byte Erase in UCM 010: 512 Byte erase in UCM 011: Reserved 100: Reserved 101: Reserved 110: Reserved 111: Mass erase
Erase Mode.1	NvFSM.5	
Erase Mode.0	NvFSM.4	
Write Mode.1	NvFSM.3	00: No byte write selected 01: Byte write selected 10: Reserved 11: Reserved **Note** Memory block target will be determined by last MOVX in NVM.
Write Mode.0	NvFSM.2	
C/D	NVFSM.0	CODE or XDATA Map Switch 0: Set XDATA Space to default memory map 1: Set XDATA Space to match CODE Space memory map This bit decides whether MOVX read/write access should address XDATA space (0) or CODE space (1).

13.5 Finite State Machine example code

13.5.1 Byte Write Example Code

```
; R6 & R7 contain the UCM address to operate on
; Check that the page is within the UCM boundary
; E.g. For the Theseus Titanium 80 UCM: only addresses above 800h are allowed
        clr     c
        mov     a, r6
        subb    a, #08h                 ; check the high address byte is above 08h
        jc      FSM_Error

; Setup the DPTR with the destination address
        mov     dph,r6
        mov     dpl,r7

; Select the NVM operation you want to perform
        mov     NvFsmMode,#04h          ; Byte Write selected in FSM

; Only include this line if you want to write to CODE space instead of XDATA space.
;       orl     NvFsmMode,#1

; Get the byte to write
        mov     a,r5

; Setup the FSM with the desired address.
; This line MUST be included as it sets up hardware in the FSM.
; You MUST have the destination address in the DPTR at this point.
```

```
; For Byte Writes you must have the desired byte in Acc.

; For Page Erases the value of Acc does not matter.
        movx    @dptr,a

; Set the NVM Finite State Machine running
Firmware_API_Run_State_Machine:

        orl     NvFsmCon,#02h           ; Set FSM Start Bit
        jnb     NvFsmCon.0,$            ; Wait for the NvInt flag
; This tells us the FSM has finished its task

        clr     NvFsmCon.0              ; Clear the NvInt flag

; Here we setup a return code for a C function
; Success will be 1 if FSM operation successful
; Will be 0 if FSM operation failed.

; Assume Failure until we check below
        mov     r7,#0

        jb      NvFsmCon.5,Firmware_API_Run_State_Machine__Return
; If Success bit not set then return Fail

FSM_Error:
        mov     r7,#0FFh

Firmware_API_Run_State_Machine__Return:
        Ret
```

13.5.2 Page Erase 256 Bytes Example Code

```
; R6 & R7 contain the UCM address to operate on
; Check that the page is within the UCM boundary
; E.g. For the Theseus Titanium 80 UCM: only addresses above 400h are allowed
        clr     c
        mov     a, r6
        subb    a, #08h                 ; check the high address byte is above 08h
        jc      FSM_Error

; Setup the DPTR with the destination address
        mov     dph,r6
        mov     dpl,r7

; Select the NVM operation you want to perform
        mov     NvFsmMode,#10h          ; Page Erase 256 bytes selected in FSM

; Only include this line if you want to write to CODE space instead of XDATA space.
;       orl     NvFsmMode,#1

; Setup the FSM with the desired address.
; This line MUST be included as it sets up hardware in the FSM.
; You MUST have the destination address in the DPTR at this point.
; For Byte Writes you must have the desired byte in Acc.
; For Page Erases the value of Acc does not matter.
        movx    @dptr,a
```

```
; Set the NVM Finite State Machine running

Firmware_API_Run_State_Machine:

        orl     NvFsmCon,#02h           ; Set FSM Start Bit
        jnb     NvFsmCon.0,$            ; Wait for the NvInt flag
; This tells us the FSM has finished its task

        clr     NvFsmCon.0              ; Clear the NvInt flag

; Here we setup a return code for a C function
; Success will be 1 if FSM operation successful
; Will be 0 if FSM operation failed.

; Assume Failure until we check below
        mov     r7,#0

        jb      NvFsmCon.5,Firmware_API_Run_State_Machine__Return
; If Success bit not set then return Fail

FSM_Error:
        mov     r7,#0FFh

Firmware_API_Run_State_Machine__Return:
        Ret
```

IMPORTANT NOTE: Critical errors may result from calls to the Finite State Machine (FSM) to Write or Erase memory outside the valid NVM memory range. Users should carefully ensure that Addresses to be written or erased are within the valid NVM Address range. See Table 15 in Section 13.2 for the acceptable range for this product.

13.6 Code/Data Map Switch Bit

The code examples within Section 13.4 show that the User must execute a MOVX write instruction in order to setup the address and data before starting the NVM FSM. This means that the physical address must be mapped into the XDATA Space in order for the MOVX instruction to access the desired physical address.

The default memory map shown in Section 6, shows that we have 18K of UCM that is mapped only to CODE space and does not appear in XDATA space. The Code/Data switch allows the User to map this CODE-Only memory into the XDATA space in order to perform Write/Erase operations.

The C/D bit is defaulted to 0, which gives the memory map shown in Section 6. When the C/D bit is set to 1, the Data Space will look exactly like the Code Space. This effectively displaces the 18K Data-Only NVM section from the top of the Data space, but the User can swap that back in by setting C/D back to 0.

14. Clock Generation

14.1 Principles

There is a single external clock input pin defined in the ISO 7816-3 specification. This is the ISO clock pin used to synchronise the ISO serial data. Traditionally this has also been used to derive the master clock used within the Smart card chips to synchronise the operation of the various sub systems of the device. In the Theseus Titanium 80, the input pin goes through a filter at 23MHz, which rejects ringing, impulse transients and illegal spikes.

14.1.1 Clocks

The User may choose through a Firmware call two possible clock sources and the associated division and multiplier ratios for the selected clock source. These selectable opportunities will allow the User to optimise between chip performance and power consumption. A block diagram for the clock generation is detailed in Figure 17.

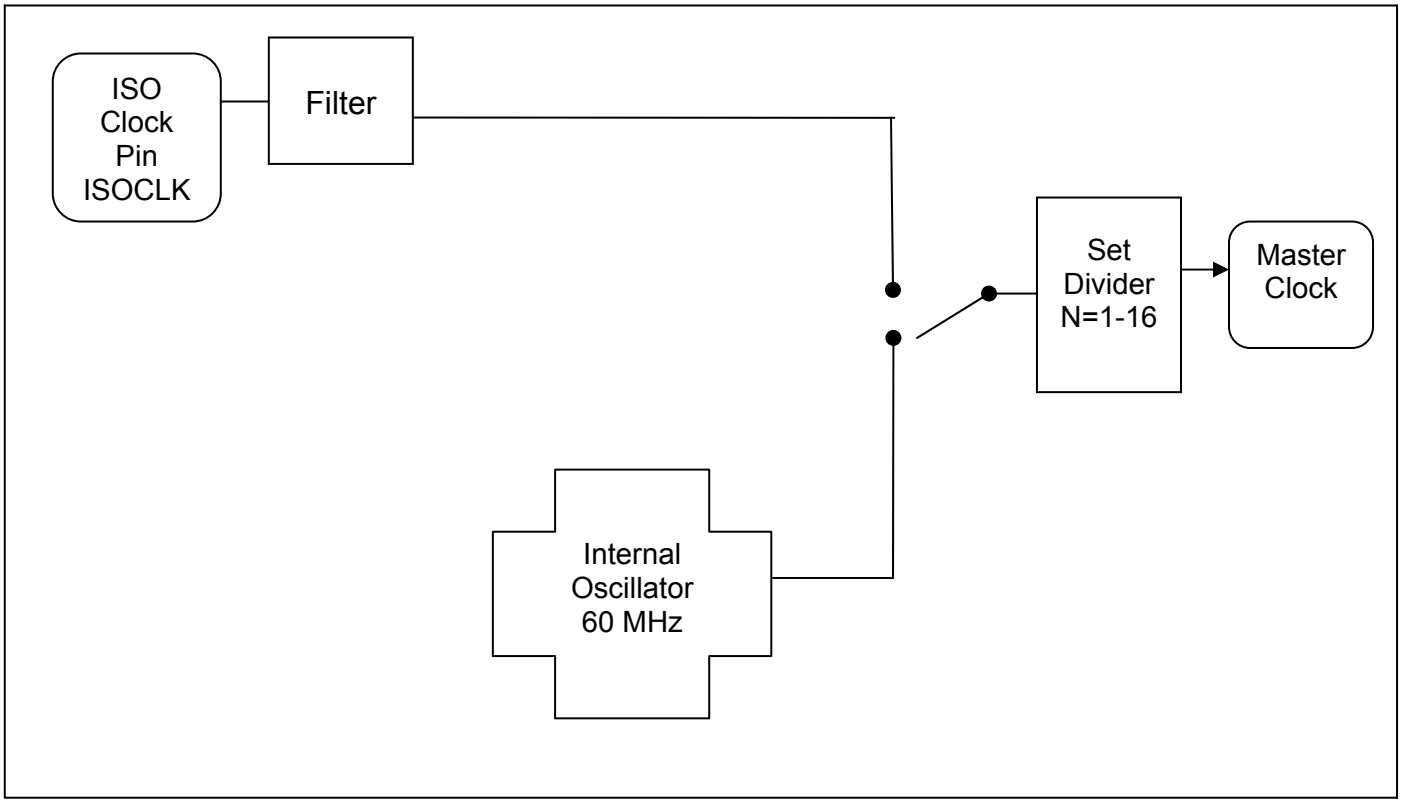


Figure 17: Clock Generation

The Clock Source Frequency limits are shown below:

	Minimum Frequency	Maximum Frequency
ISOCLOCK	1MHz	10MHz
Internal Oscillator	NA	60MHz

You can cleanly change the clock source at any time from one clock frequency to another without having to reset the chip.

The clock generation mechanism allows internal clock option and dynamic switching between clock sources. The two clock modes are: Direct and Internal.

In Direct mode

- The external ISO_CLK input pin is passed through unchanged

In Internal mode

- A free running 60 MHz internal oscillator is used as source. This internal oscillator is asynchronous to any external signal
- The internal oscillator is factory set to 60MHz at a temperature of 25°C. This will be turned OFF to save power when not in use (Clock Source Select bit 4)

Note Clock source and Division ratio are set and controlled via the Set Clock API.

Whichever clock is selected it is fed through an integer divider to the Master clock of the chip. The divider is controlled by the Division ratio. This allows the clock to be divided by up to 16, enabling fine-tuning of the processor power versus electrical current consumption of the device. The Clock Generation mechanism allows dynamic switching between these clock sources.

14.1.2 Clock Generation SFR

Table 16 shows the Clock Control SFR

SFR Clock Control								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Function					**Select Division Ratio**			
Value at Hardware Reset					**00h**			
Value after Firmware Start Up Code Execution					**xxxx 0001b**			

Select Division Ratio

Note The values shown assume a 60MHz clock

b3	b2	b1	b0	
0	0	0	0	30MHz
0	0	0	1	20MHz
0	0	1	0	15MHz
0	0	1	1	12MHz
0	1	0	0	10MHz
0	1	0	1	56.25MHz
0	1	1	0	7.5MHz
0	1	1	1	52.25MHz
1	0	0	0	48.75MHz
1	0	0	1	45MHz
1	0	1	0	5MHz
1	0	1	1	41.25MHz
1	1	0	0	37.5MHz
1	1	0	1	33.75MHz
1	1	1	0	3.75MHz
1	1	1	1	undivided clock (direct)

Table 16: Clock Control SFR

Table 17 shows the Clock Source Select.

SFR Clock Source Select								
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Function		**Res**	**Res**				**Clock Source Selected**	
Value at Hardware Reset							**00h**	
Value after Firmware Start Up Code Execution							**xxx1 xx10b**	

Clock Source Selected

b1	b0	
0	0	External Clock pin after filtering selected
1	0	Internal Oscillator selected

Table 17: Clock Source Select

Note to application programmer, the functions in the above SFR's are accessible only through firmware API. It is not possible for application program to access these SFR's directly.

This page is intentionally left blank

15. Firmware

15.1 Firmware Interface

The Theseus Titanium 80 firmware is jumptable based. The jumptable is located midway through the firmware space starting at address 0x1CE. The Theseus Titanium 80 has a 1k of firmware space.

This jumptable firmware interface is backwards compatible with existing Theseus Platinum products – Theseus Platinum 40 ->Theseus Platinum 130. If an API exists on the Theseus Titanium 80 that existed on prior products, then the interface shall remain consistent with the previous product.

15.2 Firmware Interface details

The User calls the public address of the jumptable, which in turn calls the internal function. If the firmware is updated in the future, then this should not affect the Users code.

E.g.

CSEG AT Jumptable Location

```
Function 1_Outside_Address:
        ajmp    Function1_Internal_Address

Function 2_Outside_Address:
        ajmp    Function2_Internal_Address

Function 3_Outside_Address:
        ajmp    Function3_Internal_Address
```

The firmware calls will use the Keil C complier call convention of passing parameters in registers.

For example, in the C statement
“Firmware Call 1 (unsigned int Test1, unsigned int Test2, unsigned int Test3)” the parameters will be passed in the following registers:

R6 = high (Test 1)
R7 = low (Test 1)

R4 = high (Test 2)
R5 = low (Test 2)

R2 = high (Test 3)
R3 = low (Test 3)

15.3 Firmware API List

The following API calls in the Firmware access protected SFR's.

API Name	Description	Jump Table location
Set_Clock_Trim	Adjust clock trim	0x01CE
Erase_UCM_Block	Erase one or more UCM block. Pass by block number	0x01D0
Set_Security_Action	Set Security action to Reset or to Interrupt	0x01D2
Set_Security_Mask	Mask security options	0x01D4
Set_Execute_Only_Status	Set execute only status for UCM	0x01D6
Set_Protection_Status	Set protection for UCM	0x01D8
Get_Chip_ID	Read 8 bytes Chip ID	0x01DA
Get_Firmware_Version	Read 1 byte of current firmware version	0x01DC
Set_Clock	Select Clock Source and speed	0x01DE
Get_Clock_Source_Select_SFR	Get 1 byte contents of Clock Source Select SFR	0x01E0
Get_Clock_Control_SFR	Get 1 byte contents of Clock Control SFR	0x01E2
Get_Diagnostic_Data	Returns 8 bytes of diagnostic data for Emosyn use	0x01E4
Get_Random_Number	Get 1 byte random number	0x01E6
Verify_Transport_Key	Compares User key with key held in Firmware space	0x01E8
Set_Key	Sets either the Transport or Encryption key	0x01EA
Get_Key_Status	Returns the presence of either the Transport or Encryption key	0x01EC
Init_Key_For_Encryption	Initialises the decryption process using the encryption key	0x01EE
Translate_Byte_In_Acc	Decrypts an encrypted byte using the decryption process	0x01F0
Clear_SBOX_Offset	Clears the SBOX decryption process data	0x01F2
Set_IO_Pullup	Sets the status of the IO pull-up resistor	0x01F4

15.4 Software Library Interface

The following API's are provided to the User as a software library and/or example code for the User to include in their projects.

Library Calls	Description
Write UCM	Write multiple UCM bytes from an IDATA buffer
Write Byte UCM	Write 1 byte of data to UCM
Erase Page UCM 256	Erase 256 bytes of UCM
Erase Page UCM 512	Erase 512 bytes of UCM
*Mass Erase UCM	Erase all unprotected UCM blocks

*This mass erase **does not** include the bootloader, which is different from the Mass Erase settings of NvFSM Mode SFR.

15.5 System software start-up sequence

After hardware reset the Theseus Titanium 80 starts executing firmware code.

The start-up routine performs the following sequence of tasks:

1. Initialise all SFR's as shown in Table 4.
2. Set clock to 20MHz internal OSC.
3. Set IO communication to 9600 Baud.
4. Decide whether to jump to User code or Bootloader.

15.6 Set Clock Trim

This function allows the User to trim the Internal Oscillator in order to help with Anti-DPA code.

API	unsigned char Firmware_Set_Clock_Trim (unsigned char Oscillator_Trim_Value)
Jumptable Address	0x01CE
Inputs	R7 Trim Value 0->FE are valid Trim values FF resets the Clock Trim to the factory setting and sets the Main Clock as previously set by the User (30MHz Internal clock for example)
Outputs	R7 Clock Trim Value
Stack Usage	0
Register Usage	DPTR

Notes The "Set_Clock_Trim" API protects the device against DPA/DSA attacks.

Before using this API please contact you local SST representative for additional advice.

15.7 Erase UCM Block

This function allows the application to erase multiple blocks of physical memory from one API call.

API	void Firmware_EraseUCMBlock (unsigned int Erase_UCM_Block_Value)
API Number	0x01D0
Inputs	R6/R7 Erase_UCM_Block_Value
Outputs	None
Stack Usage	0
Register Usage	0

Notes See Table 15 to calculate the desired value of Erase UCM Block Value.

It is possible to add the values of the blocks to be erased into one call. For example, to erase blocks 1 and 3, add the R6 registers to get the result 00h, and add the R7 registers to get the result 0Ah. The integer parameter Erase_UCM_Block_Value passed to this call will be composed of the R6 value as the High byte, and the R7 value as the Low byte.

15.8 Set Security Action

This function allows the application to decide if a security error triggers an interrupt or a reset. See Section 7.8.1 and Section 7.8.2 for a discussion of the Security Action register.

API	char Firmware_Set_SecurityAction (unsigned char Value, unsigned char Permanent_Flag)
Jumptable Address	0x01D2
Inputs	R7 Desired Security Action value R5 Permanent Flag 00 Temporary until next Reset 01 Permanent - cannot revert back
Outputs	R7 00 Success FF Fail
Stack Usage	0 Permanent Flag = 00 2 Permanent Flag = 01
Register Usage	0

Warning - If the permanent flag is set in this function call, the change (to the target SFR bits) that is made by this function call cannot be undone. It becomes permanent for the life of the device.

15.9 Set Security Mask

This function allows the application to Mask/Unmask the desired security options. Once it has been masked (bit set to "1"), the corresponding option will generate a reset or interrupt according to the security action setting. By default the security mask value is set to 0x00.

Note It is possible to set 1 bit only to “permanent” and to set another one to “permanent” later during the execution of the code with another API call.

API	char Firmware_Set_SecurityMask (unsigned char Value, unsigned char Permanent Flag)
Jumptable Address	0x01D4
Inputs	R7 Desired Security Mask value. Bit set (1): sensor enabled Bit reset (0): sensor disabled R5 Permanent Flag 00 Temporary until next Reset 01 Permanent - cannot revert back
Outputs	R7 00 Success FF Fail
Stack Usage	0 Permanent Flag = 00 2 Permanent Flag = 01
Register Usage	0

Warning - If the permanent flag is set in this function call, the change (to the target SFR bits) that is made by this function call cannot be undone. It becomes permanent for the life of the device.

15.10 Set Execute Only Status

This sets the Execute Only protection on physical blocks of memory.

API	char Firmware_Set_Execute_Only_Status (unsigned int Execute_Only_Value, unsigned char Permanent_Flag)
Jumptable Address	0x01D6
Inputs	R6/R7 Execute_Only_Value R5 Permanent_Flag 00 Temporary until next Reset 01 Permanent - cannot revert back
Outputs	R7 00 Success FF Fail
Stack Usage	0 Permanent Flag = 00 2 Permanent Flag = 01
Register Usage	0

Warning - If the permanent flag is set in this function call, the change (to the target SFR bits) that is made by this function call cannot be undone. It becomes permanent for the life of the device.

Note Once the Execute Only protection is set, the memory block cannot be read from using MOVX or MOVC except by code from within that memory block.

The User can "Or" the R6 and R7 register values as shown in Table 15 in order to set the attributes of several blocks with only 1 Set_Execute_Only or Set_Protect_Status API or Erase Block call.

Example:

R6 = 0x01 and R7= 0xFF will address the memory from 0x400 to 0x7BFF".

15.11 Set Protection Status

This sets the ROM protection on physical blocks of memory.

API	char Firmware_Set_Protection_Status (unsigned int Protection_Settings_Value, unsigned char Permanent_Flag)
Jumptable Address	0x01D8
Inputs	R6/R7 Protection_Settings_Value R5 Permanent_Flag 00 Temporary until next Reset 01 Permanent - cannot revert back
Outputs	R7 00 Success FF Fail
Stack Usage	0 Permanent Flag = 00 2 Permanent Flag = 01
Register Usage	0

Warning - If the permanent flag is set in this function call, the change (to the target SFR bits) that is made by this function call cannot be undone. It becomes permanent for the life of the device.

Note Once the protection is set, the memory block becomes read only and the erasing or writing will be inhibited.

See Table 15 to calculate the desired value of Protection Settings Value.

15.12 Get Chip ID

This function allows the application to retrieve the 8-byte Chip ID. This ID is a unique number and is programmed into the chip by Emosyn.

API	void Firmware_Get_Chip_ID (unsigned char IDATA *pDestPtr)
Jumptable Address	0x01DA
Inputs	R7 Pointer to an 8 byte IDATA buffer to copy the Chip ID into
Outputs	None
Stack Usage	2
Register Usage	0

Note The chip ID, programmed into the Theseus product relates to the particular wafer that the die is sawn from. If the chip ID number is made available by the Users software, then Emosyn will be able to identify the type of silicon and the batch it comes from. This will help the User to associate the part with his production process, in case of field issues.

15.13 Get Firmware Version

This function returns the version number of the firmware loaded on the chip

API	unsigned char Firmware_Get_Firmware_Version (void)
Jumptable Address	0x01DC
Inputs	None
Outputs	R7 Firmware version
Stack Usage	0
Register Usage	0

15.14 Set Clock

This function allows the application to set the clock source and its speed, see Table 16.

API	char Firmware_SetClock (unsigned char Clock_Source, unsigned char Multiplier, unsigned char Divider)
Jumptable Address	0x01DE
Inputs	R7 Clock Source 00 Internal Osc 02 ISOCLK R5 00 R3 Divider value
Outputs	R7 00 Success FF Fail
Stack Usage	0
Register Usage	0

15.15 Get Clock Source SFR

This function returns the value of the Clock Source SFR.

API	unsigned char Firmware_Get_ClockSource (void)
Jumptable Address	0x01E0
Inputs	None
Outputs	R7 SFR Value
Stack Usage	0
Register Usage	0

15.16 Get Clock Control SFR

This function returns the value of the Clock Control SFR.

API	unsigned char Firmware_Get_ClockControl (void)
Jumptable Address	0x01E2
Inputs	None
Outputs	R7 SFR Value
Stack Usage	0
Register Usage	0

15.17 Get Diagnostic Data

This function allows the application to retrieve 8 bytes of Diagnostic data.

API	void Firmware_Get_Diagnostic_Data (unsigned char IDATA *pDestPtr)
Jumptable Address	0x01DA
Inputs	R7 Pointer to an 8 byte IDATA buffer to copy the Diagnostic data into
Outputs	None
Stack Usage	2
Register Usage	0

Note The Diagnostic data, programmed into the Theseus product relates to the particular wafer that the die is sawn from. If the diagnostic data number is made available by the Users software, then Emosyn will be able to associate the part with the production process, in case of field issues.

15.18 Verify Transport Key

This function performs a comparison between the passed User key and the key held in Firmware space.

API	char Firmware_Verify_Transport_Key (unsigned char IDATA *Key_Buffer)
Jumptable Address	0x01E8
Inputs	R7 Pointer to an 8 byte IDATA buffer containing User Transport key to attempt unlock
Outputs	R7 00 Success (key comparison OK) FF Fail (Key comparison failed)
Stack Usage	2
Register Usage	0

15.19 Set Key

This function sets the User key into the Firmware space. The key can be either a customer specific Transport or Encryption key.

API	char Firmware_Set_Key (unsigned char Key_Type, unsigned char IDATA *Key_Buffer)
Jumptable Address	0x01EA
Inputs	R7 Key Type 00 Transport Key 01 Encryption Key R5 Pointer to an 8 byte IDATA buffer containing User Transport key to attempt unlock
Outputs	R7 00 Key loaded successfully AA Key already exists in Firmware space FF Key writing error
Stack Usage	3
Register Usage	0

15.20 Get Key Status

This function returns the status of the key presence in the Firmware space. The check can be either the Transport or Encryption key.

API	char Firmware_Get_Key_Status (unsigned char Key_Type)
Jumptable Address	0x01EC
Inputs	R7 Key Type 00 Transport Key 01 Encryption Key
Outputs	R7 00 Key present in Firmware space FF Key not present in Firmware space
Stack Usage	0
Register Usage	0

15.21 Init Key for Encryption

This function initialises the decryption algorithm process to prepare for encrypted OS downloads.

API	void Firmware_Init_Key_for_Encryption (void)
Jumptable Address	0x01EE
Inputs	None
Outputs	None
Stack Usage	0
Register Usage	0

15.22 Translate Byte in Acc

This function decrypts an encrypted byte through the decryption algorithm in Firmware.

API	char Translate_Byte_In_Acc (unsigned char Encrypted_Byte
Jumptable Address	0x01F0
Inputs	Acc Encrypted byte
Outputs	Acc Decrypted byte
Stack Usage	1
Register Usage	0

15.23 Clear SBOX Offset

This function clears the data used for decryption algorithm process.

API	void Firmware_Clear_SBOX_(void)
Jumptable Address	0x01F2
Inputs	None
Outputs	None
Stack Usage	0
Register Usage	0

15.24 Set IO Pull-up

This function sets the IO pull-up resistor state.

API	char Firmware_Set_IO_Pullup (unsigned char Pullup_Value, unsigned char Permanent_Flag)
Jumptable Address	0x01F4
Inputs	R7 IO Resistor Pull-up value R5 Persistence Flag 00 Temporary until next Reset 01 Permanent - cannot revert back
Outputs	R7 00 Success FF Permanent setting unsuccessful
Stack Usage	0 Permanent Flag = 00 2 Permanent Flag = 01
Register Usage	0

Warning - If the permanent flag is set in this function call, the change (to the target SFR bits) that is made by this function call cannot be undone. It becomes permanent for the life of the device.

16. Dimensions

16.1 Pad locations

The bonding pad locations for the Theseus Titanium 80 are shown in Figure 18:

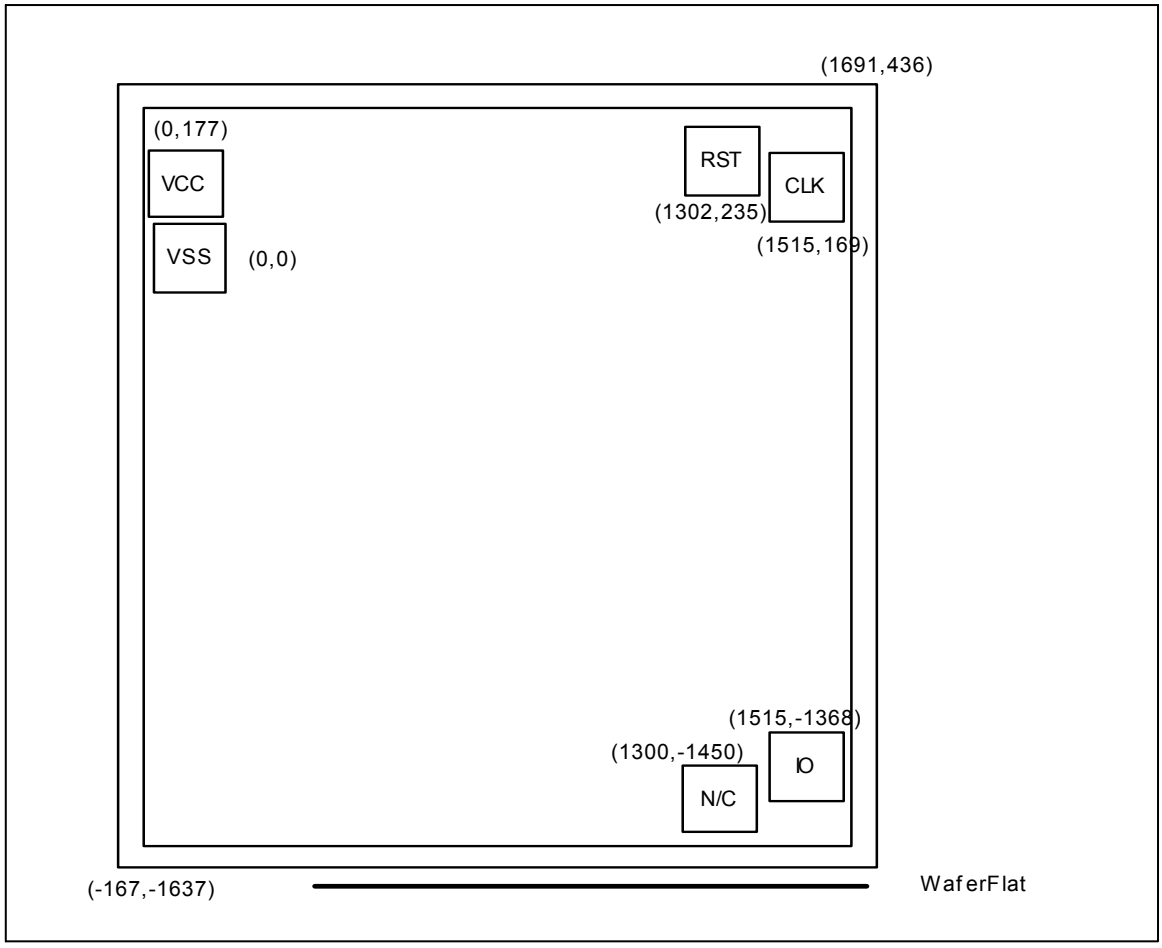


Figure 18: Pad locations

Note: 1) All dimensions are in microns (1 micron = 0.001 mm).
2) Coordinates shown refer to the centre of each contact pad.
3) Coordinates are relative to the centre of the Vss pad.
4) Scribe street is 40 um on each side on wafer. Most of the scribe streets are removed during dicing.

16.2 Scribe and Index sizes

The Theseus Titanium 80 scribe and index sizes are shown in Figure 19:

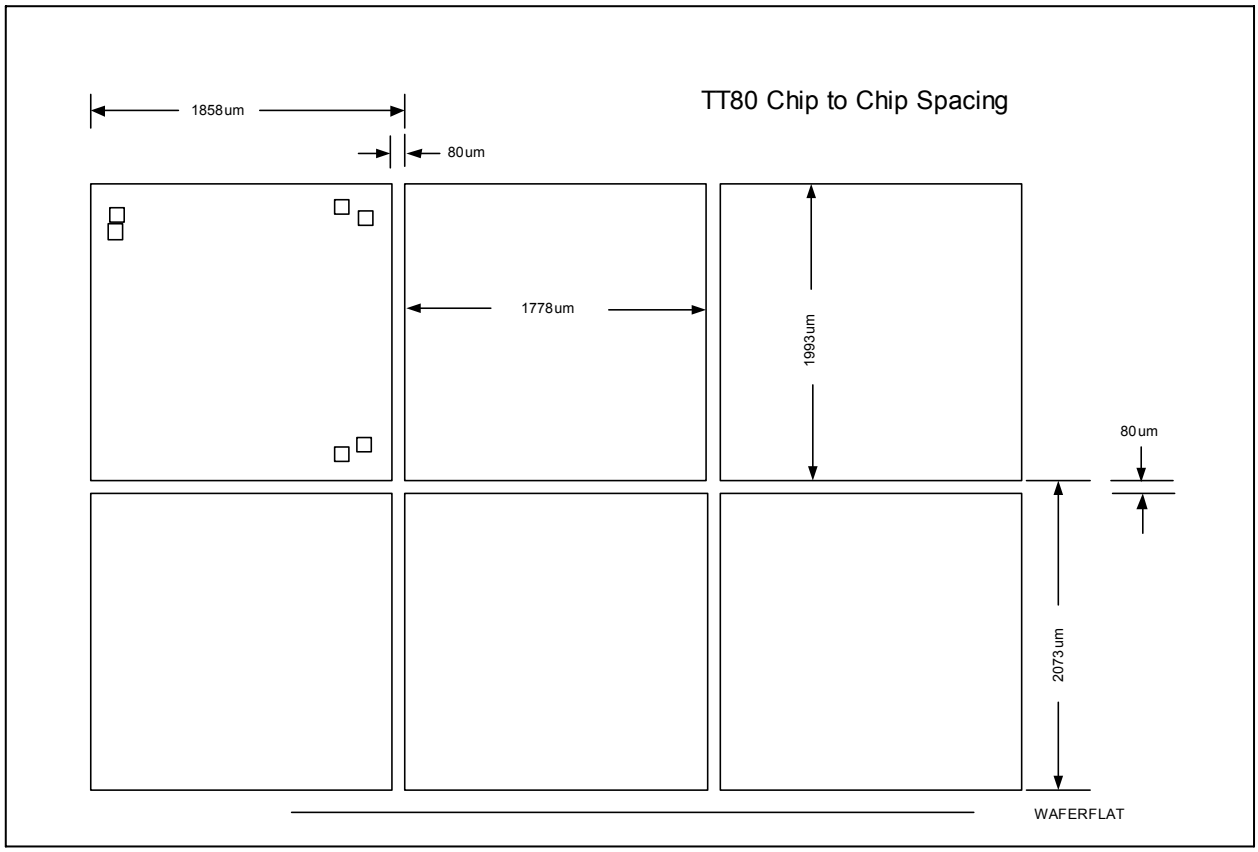


Figure 19: Scribe and Index sizes

- Nominal Wafer size: 8 inches
- Pad size: 90um x 90um
- No need for backside ground contact

A cross-section of the Theseus Titanium 80 Pad is shown in Figure 20:

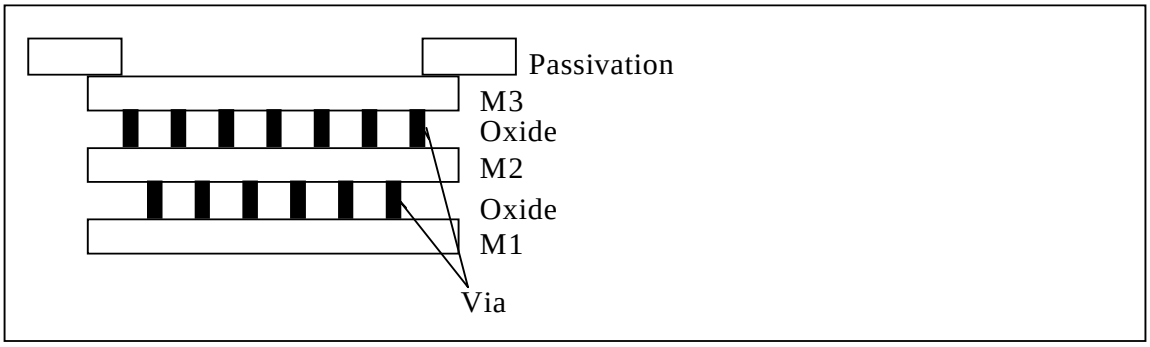


Figure 20: Pad Cross-Section (principle, not at scale)

16.3 Pin Definitions

The mapping of the pins that are used by the Theseus Titanium 80 device and the definition in the ISO specification are shown in Table 18:

Assignment	Symbol	ISO 7816-3 card 8-pin	ISO 7816-3 card 6-pin	Theseus Platinum 80
Supply voltage	VCC	C1	C1	VCC
Reset signal	RST	C2	C2	RST
Clock signal	CLK	C3	C3	CLK
Reserved		C4		-
Ground	GND	C5	C4	GND
Programming Voltage	VPP	C6		-
Reserved			C5	
Data input/output	I/O	C7	C6	I/O
Reserved	-	C8	C5	-

Table 18: Pin Definitions

This page is intentionally left

17. SxBL Bootloader

17.5.1 Overview

The bootloader has to function in three different scenarios:

1. Loading of the Users Software in a high volume automated manufacturing process.
2. Loading of the Users Software in a low volume manual process, using off the shelf smart card readers.
3. Software developers loading erasing and loading development software.

In the Theseus Titanium 80, the bootloader is stored at the top of CODE space UCM. This has the advantage of being able to erase it in the final product that improves security and maximises the available memory to the end user. The drawback is that the software developer who wishes to reprogram the device (referred by some companies as “re-flashing”) needs to preserve the bootloader.

If a method to allow Scenario 2 is provided, then this requires a large increase in code creating a problem for Scenario 3. To make this possible Emosyn created a minimal bootloader referred as the Core Bootloader. This is as small as possible so that the developer can either leave it in situ or back it up it into the minimal number of UCM pages at a **location of their choosing** in order to erase the bootloader at the top of UCM to use this space for testing of the OS.

As shipped to the Customer the device will contain the maximum capability, with loaders capable of handling all the scenarios from the simple core loader to a complex decrypting loader, see Figure 21.

Extensions to the core loader can be added at any time in production at wafer, module or card.

Please contact your local SST representative for any specific or additional requirements.

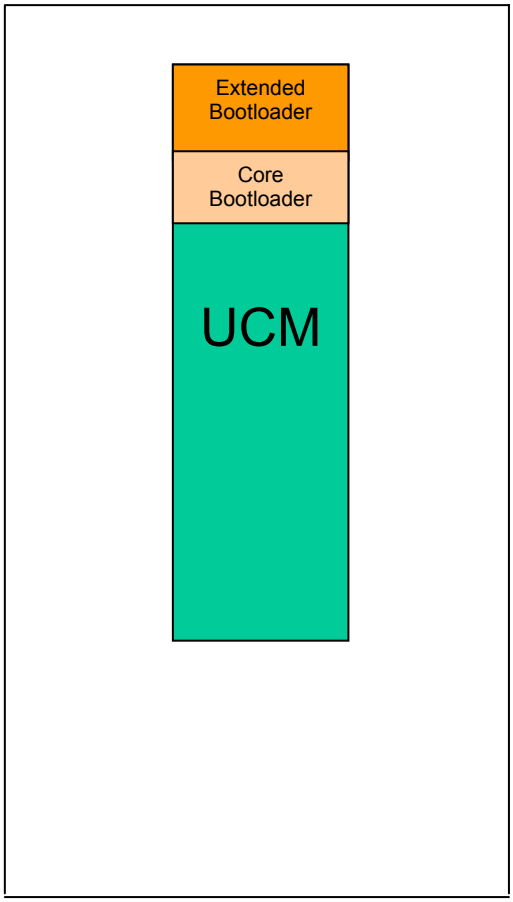


Figure 21: Shipped to customer

While in software development the User can copy the Core loader somewhere into free code space within his OS and put a reflash test APDU so that the bootloader can be copied back into its original position and erase their OS such that the device is ready for loading again, see Figure 22.

Once the final product is released then there is no need for a bootloader meaning the full User OS space is available when the bootloader is removed.

The production equipment will use the Ex factory devices with all of the bootloader capabilities indicated in the following specification, see Figure 22.

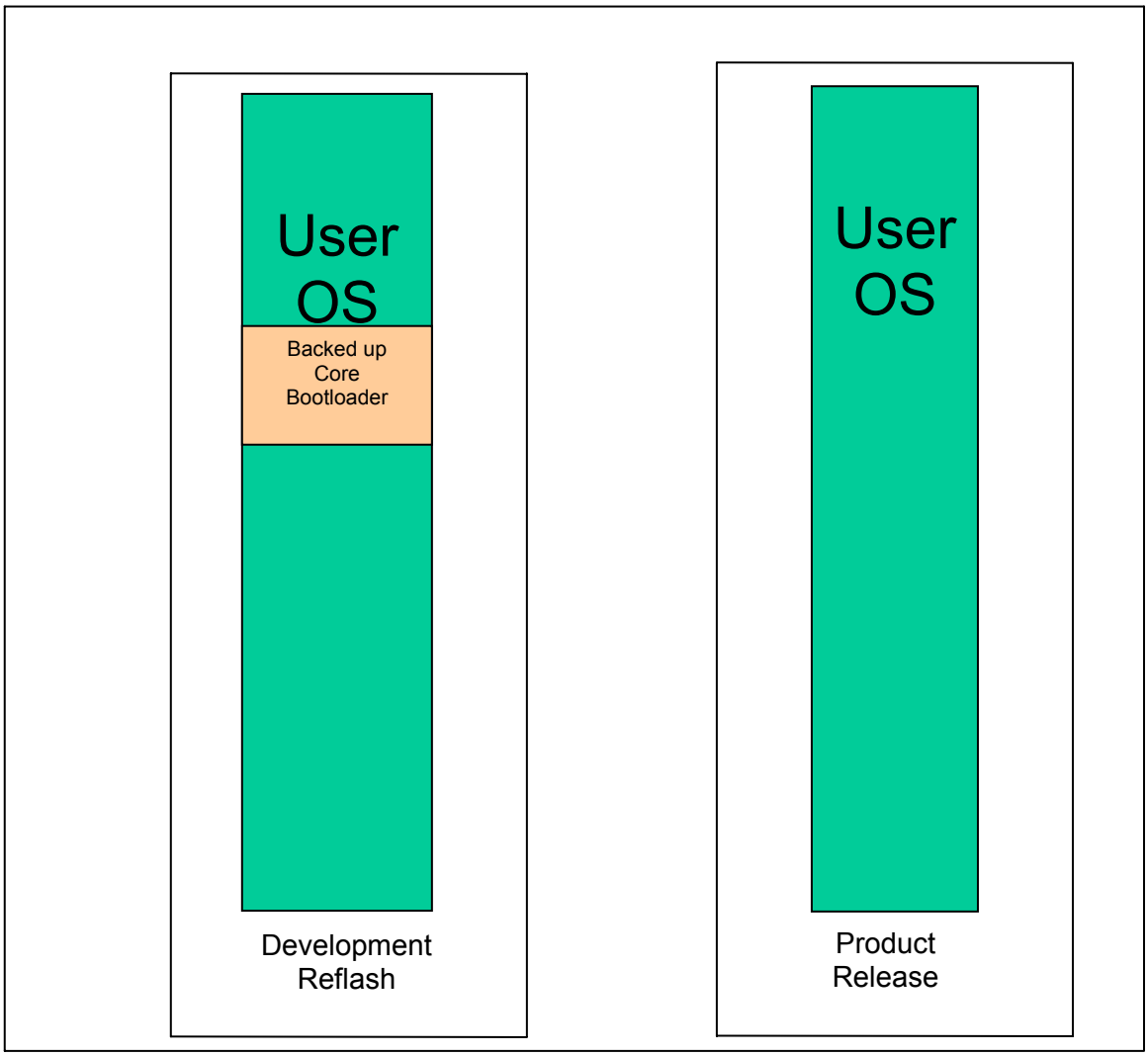


Figure 22: Core Bootloader Backed up and OS Released

17.5.2 Introduction

There are 3 different “bootloader's” programmed into the Theseus Titanium 80:

- Core Bootloader
- Normal Bootloader
- ISO Bootloader

The Core bootloader contains the basic S-Algorithm code to provide high speed loading for customers that can download on a byte-by-byte basis. It does not have the functionality of the Normal or APDU bootloaders. It is 246 bytes in size and provides a way of loading the chip during development. This allows the developer to only backup these 246 bytes instead of the entire bootloader.

The Normal Bootloader gives the full functionality of high speed S-Algorithm loading commands including decryption of data and erasing of UCM.

The APDU compatible interface is provided to allow developers/production who cannot use the S-Algorithm to program cards on any PC/SC compatible smart card reader.

Decrypting versions of both the S-Algorithm and the APDU algorithm are provided to allow the software developer to hand over an encrypted file to the production company instead of plaintext data that could be disassembled.

Due to the syntax of the S-Algorithm (described later), with respect to the production equipment, the S-algorithm is 99% backwards compatible with the K-Algorithm (Theseus Platinum 100, 100-2 and 130). If the production equipment can parse a GNG file and download Theseus Platinum 100-2/130 GNG's, then it can download S-algorithm GNG's as well with no software change.

If only the core bootloader is present when the firmware jumps to the bootloader, the ATR outputted from the card will be 3B 00. If the bootloader detects that the full SxBL loader is present then the ATR outputted from the card will be 3B 99 96….

SST has provided backward compatibility for its existing bootloaders, with the exception of the APDU bootloader, which has been replaced by this new combination loader. To keep a high level of security in the downloading process, it will not be possible to download any OS with any bootloader in plaintext after a customer specific encrypting key is downloaded into the firmware.

Note All values in this section are in Hex.

17.5.3 SxBL location

The Bootloader starting address is F600h (2.5KB is allocated for it) and ends at FFFFh in CODE space.

17.5.4 Backward compatibility with Core and Normal Bootloader

The SxBL must be backward compatible with these loaders. This means that these loaders commands must be compatible with former version of bootloaders. However, the ATR of the files compatible with the Core and Normal bootloaders have to be updated with the ATR of the SxBL.

The First byte of the commands of these bootloader is different from the Class byte of the ISO bootloader as defined in this document. For the reasons mentioned above, when the first byte of an APDU will be the first byte of a Core or Normal Bootloader command, the command of these bootloader will be executed. As a consequence, the ISO bootloader as defined in this document will not return any "Bad Class", 0x6E00 status words.

If the User wants to use the core bootloader, they have to erase all the code memory from address F700h to FFFFh (Normal and ISO bootloader location). To do so, pages F900h to FF00h have to be erased first, followed by page F700h and then page F800h with the page erase command.

17.5.4.1 Core Bootloader commands table

Core Bootloader Commands	Description
50	Write a data block of X bytes to UCM
12	Change ISOTMR value
30	Mass Erase UCM – not including the bootloader

Table 19: Core Bootloader Commands

17.5.4.2 Normal Bootloader commands table

Normal Bootloader Commands	Description
50	Write a data block of X bytes to UCM
12	Change ISOTMR value
30	Mass Erase UCM - not including the bootloader
55	Change to 60 MHz clock
51	Change data block size
60	Write an encrypted data block of X bytes to NVM
65	Receive 16 byte decryption key
59	Set / Clear C/D bit in NvFsmMode SFR

Table 20: Normal Bootloader Commands

17.5.5 ISO bootloader instruction list

The ISO bootloader Class is E5h.

ISO Bootloader INSTRUCTIONS	Description
12h	Direct Division Factor Setting
56h	Set Chip Clock Internal Frequency
A0h	Write Plaintext Data to UCM.
A6h	Write Encrypted data to UCM
58h	Set/Clear the C/D bit in the NvFsmMode SFR
A8h	Bulk Memory Checksum command
AAh	Unlock Device
A2h	Load Key
30h	Mass erase bootloader safe
3Ch	Page erase in XDATA space
36h	Page Erase in Code space
34h	Get Chip ID
32h	Get Firmware version

Table 21: ISO bootloader instruction list

17.6 ISO Bootloader: Functional specifications

17.6.1 ISO 7816-3 compliancy

- An APDU bootloader interface has been implemented in accordance with the latest version of the ISO 7816-3 T=0 specification. The T=1 protocol is not supported by this bootloader

17.6.2 Downloadable file generation

- An Output File Generator (OFG) will be provided to the customer. The entry files of this utility tool will be the Hex files generated with the Keil compiler
- Depending on the type of bootloader selected, the OFG will generate either:
  - An Emosyn proprietary file format “gng” (core and Normal Bootloader) or,
  - An ISO compliant commands format (ISO Bootloader) containing the OS to download into the card
- The OFG will also optionally generate files containing the transport key and the encryption key that may be provided to Emosyn approved factories for downloading at CP2 stage
- A separate OFG tutorial is provided with the OFG

17.6.3 Answer-To-Reset

The bootloader Answer-To-Reset (ATR) will be used to reflect error and diagnostic information to the User.

The Transport code ATR is defined as:

3B
XX: # Bytes Following apart from TCK Checksum
YY: Transport Code Version
ZZ: 01 = Platinum
02 = Titanium
HH: Product ID High
LL: Product ID Low
VV: Firmware version
TCK: Checksum

If the product becomes locked, due to the fourth incorrect unlock attempt, the device ATR will show:

3B 06 FF 00 FF 00 FF 00

Bootloader ATR Value:

3B: TS

Direct byte convention

99: T0

TA1, TD1 & 8 historical bytes

96: TA1 (06h forTT80 V2.0 Bootloader Beta 0.9 version).

Value for Reference baud rate
Reference Fi: 512 at 5MHz
Reference Di = 32

00:TD1, no further interface byte

Historical characters:

E5h: T0 (values 00, 10, 8x not allowed)

00h 80h: T1 – T2: Device type

Device Status T3:
00h: Virgin
01h: Partially loaded

Encryption key Status T4:

00h: No OS encryption key downloaded
01h: OS encryption key downloaded

Error code T5:

FFh: No error occurred
FFh: Last error recovered (page erase or mass erase)
01h: Serial checksum error
02h: Bulk memory checksum error
03h: Memory error

Additional information concerning error code T6 – T8:

FFh FFh FFh: No error occurred, or Last error recovered, or page erased (only first address checked)
00h 00h 00h: Bulk Memory Checksum Error Occurred
XXhXXhXXh: Address (3 bytes, including the DPX byte): Address where serial checksum error occurred (error code 01h and 03)

17.6.4 The Bootloader OS

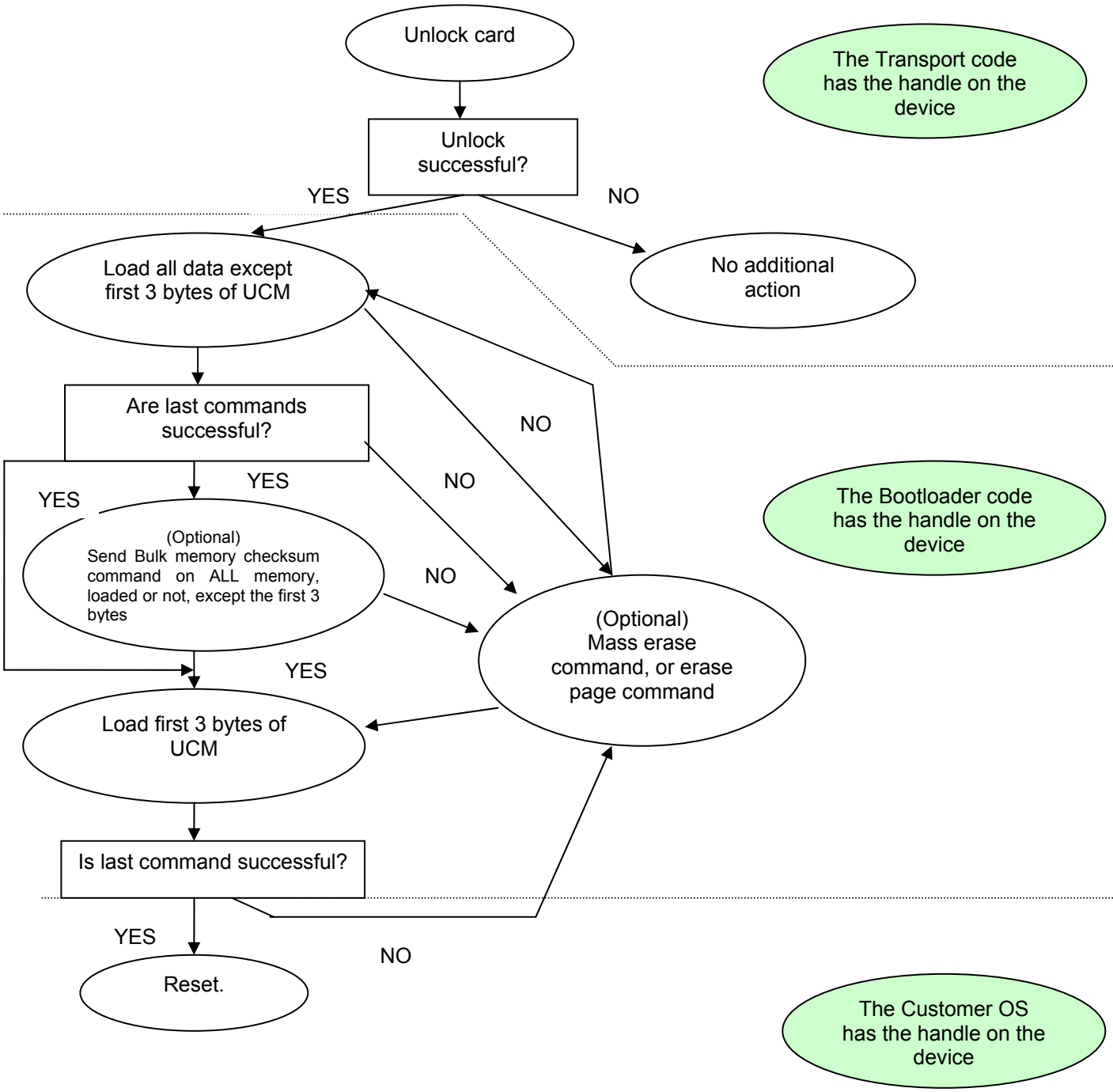
The bootloader is a program located at the top of the Code space. This program ensures the successful downloading of the device.

The bootloader interface is fully compliant with the ISO7816-3 specification. As defined in this specification, all the commands provided by this bootloader will have the ISO APDU format and the parity bit will be checked for according to the T=0 protocol.

17.6.5 Card Loading Process Flow

- Unlock card
- Get ATR
- Set device frequency or Baud Rate (optional)
- Load code via ISO commands
  - Parity checking
  - Serial checksum per APDU at the time of memory writing
  - On error; stop and no action
- Load completed except last 3 bytes
- Memory checksum command sent (optional)
- Load last 3 bytes
- Do not erase bootloader
- On reset
  - Check that the first three bytes of the customer OS are written
  - Jump to OS start if the above condition is correct

Recommended procedure



Note: Executing next command does not depend on previous result (checking optional).

17.7 Protocol and Parameters Selection (PPS)

The customer is expected to use the most optimal PPS setting based on the card, reader and ISO clock frequency:

- The PPS speed negotiation has been implemented in accordance with ISO 7816-3 chapter 7.2.2, see Table 22
- The PPS setting is dependent on ISO clock frequency, UCM writing times and internal clock frequency
- All Fi and Di values are supported by the card at 3.57MHz
- Our testing shows that all Fi and Di possible combinations are supported by the card until 10MHz for both encrypted and plaintext OS

7.2.2 Transmission factors F, D and G

This clause specifies the values of the transmission factors for fixing the etu (see 6.1). Not-synchronized communication makes use of factors F and D. Synchronized communication makes use of factor G.

According to Tables 8 and 9, FI and DI indicate values Fi and Di of factors supported by the card. FI also specifies the range of values of the frequency *f* supported by the card. The minimum value shall be 1 MHz. Table 8 gives the maximum value as a function of FI. The default maximum value is 5 MHz.

Table 8 — Fi, indicated values of the clock rate conversion factor

FI	0000	0001	0010	0011	0100	0101	0110	0111
Fi	372	372	558	744	1116	1488	1860	RFU
f (max.) MHz	4	5	6	8	12	16	20	—

FI	1000	1001	1010	1011	1100	1101	1110	1111
Fi	RFU	512	768	1024	1536	2048	RFU	RFU
f (max.) MHz	—	5	7,5	10	15	20	—	—

Table 9 — Di, indicated values of the baud rate adjustment factor

DI	0000	0001	0010	0011	0100	0101	0110	0111
Di	RFU	1	2	4	8	16	32	64

DI	1000	1001	1010	1011	1100	1101	1110	1111
Di	12	20	RFU	RFU	RFU	RFU	RFU	RFU

Table 22: PPS speed negotiation

Important Notes

The SxBL supports all the PPS values at 3.57MHz. However, our internal initial testing shows that the maximum baud rate is 450KB for encrypted data loading and 500KB for plaintext data loading.

It is the responsibility of the User to check the maximum frequency applied to fit within the baud rate mentioned.

The bootloader will not send back any error message if the communication applied is not suitable.

17.7.1 Direct Division Factor Setting

- The command to set the division factor directly is shown in the following table The Successful divisor setting is dependent on ISO clock frequency and internal frequency
- The direct division factor will set the ISOTMR SFR. The ISOTMR is a decrementing counter that gives an output and reloads the value loaded into the ISOTMR register when the count reaches zero. Because of this the value loaded into the ISOTMR register is one smaller than the desired Elementary Time Unit (ETU). This allows the 11-bit counter of the Theseus Titanium 80 to operate with a maximum division ratio of (0111 1111 1111) b or 07FFh = 2047 = 2048 ISOCLKS per ETU, and a minimum division ratio of (0000 0000 0000) b = 0 = 1 ISOCLKS per ETU

Direct Division Factor command

APDU command	
CLA	E5h
INS	12h
P1	ISOTMR Value Hi = clock cycles(Hi)/ETU desired
P2	ISOTMR Value Hi = clock cycles(Lo) +1 /ETU desired
Lc	00h
Data Field	--
Le	--

P1 P2: Number of ISO clock per ETU

Status	Meaning
6B00h	Incorrect **INS** value
9000h	Correct execution

Example: for 31 clocks per etu, P1 and P2 should be 00 1E.

17.7.2 Set Chip Clock Frequency

- The command to set the internal clock frequency is shown in Section 17.7.2.1. Refer to Table 23 for possible frequencies
- The maximum clock frequency that can be set is 60MHz
- The default clock frequency for the bootloader is 60 MHz
- Average current consumption should remain within the 10 – 15 mA range during loading; the clock frequency may need adjusting in order to meet this current consumption

Note Although SST may provide recommendations for frequency settings to ensure the best downloading yield, it is the responsibility of the customer to characterize their own process. Under no condition does SST provide any contingent warranty on yields.

Select Division Ratio	b3 b2 b1 b0	
	0 0 0 0	30MHz
	0 0 0 1	20MHz
	0 0 1 0	15MHz
	0 0 1 1	12MHz
	0 1 0 0	10MHz
	0 1 0 1	56.25MHz
	0 1 1 0	7.5MHz
	0 1 1 1	52.25MHz
	1 0 0 0	48.75MHz
	1 0 0 1	45MHz
	1 0 1 0	5MHz
	1 0 1 1	41.25MHz
	1 1 0 0	37.5MHz
	1 1 0 1	33.75MHz
	1 1 1 0	3.75MHz
	1 1 1 1	undivided clock: 60MHz

Table 23: Set Chip Clock Frequency

Note: When the Finite State Machine (FSM) performs memory operations it is clocked at 10 MHz on the Theseus Titanium, this is not variable.

17.7.2.1 Set Frequency command

APDU command	
CLA	E5h
INS	56h
P1	Clock Control value according to Table 23
P2	00h
Lc	00h
Data Field	--
Le	--

Status	Meaning
6B00h	Incorrect **INS** value
9000h	Correct execution

17.7.3 Writing data to UCM

17.7.3.1 Memory writing flow

- The default memory space selected is the code space
- The address location is checked before writing data. The address must be located between 0800h and F600h in code space and between 0800h and AE00h or B800h and FFFFh in XDATA space
- Each byte is written directly to UCM after the byte reception from the ISO port
- Parity checking is mandatory. See section 17.7.3.2 for information

- Serial checksum will be run on each APDU. See section 17.7.3.3 for information
- A memory checksum can optionally be performed once the loading is complete. See section 17.7.3.9 for information

17.7.3.2 Parity Check

- Parity bit checking is mandatory for the reception of each byte

17.7.3.3 Serial Checksum

- 16 bit serial checksum functionality to be added on every APDU data block
- Serial checksum calculation to be performed via “add with carry”
- Serial checksum calculation will be performed on the data once received from the ISO port
- Return code to indicate success or failure
- An error status word is returned in case of error
- The SxBL will not lose the handle on the device in cases of checksum errors. It is the responsibility of the customer to handle the checksum errors properly as the bootloader will not take any special action after such errors
- A hardware memory check is also performed as each byte is written to memory
- The checksum is calculated with an addition of P1, P2 and the data field (expect the 2 bytes of checksum)

17.7.7.4 Memory Check

- Each byte written in the UCM is checked against the value received from the IO port
- If an error occurs, the byte will not be erased. It is the customer responsibility to erase it with an erase page command or mass erase command

17.7.3.5 Encryption

- P1, P2 and the complete data field are encrypted when instruction is A6
- The encryption method is based on the SST private algorithm implemented into the devices
- Bytes P1, P2 and the data field are encrypted

Write Data Command:

APDU command	
CLA	E5h
INS	A6h: Encrypted data, A0: Plaintext data
P1	AddrHi
P2	AddrLo
Lc	Length of data field. [4:255]
Data Field	DPX byte (0 for Theseus Titanium 80 device)
	Data to write into UCM [1:252]
	Checksum value Hi of Data write in UCM
	Checksum value Lo of Data write in UCM
Le	--

P1 P2: Is the lower address of the data block to write into memory.

Data field:

- Variable data block-size sent to the bootloader
- Maximum size of 255 bytes including in 2 bytes of serial checksum and DPX byte
- Minimum size 4 bytes with 2 bytes of serial checksum and 1 DPX byte

Status	Meaning
6A80h	Serial checksum comparison failure
6A81h	Memory Write failure
6A82h	Address out of range.
6A84h	Encryption Key not loaded
6B00h	Incorrect INS value
9000h	Correct execution

17.7.3.7 Set/Clear the C/D bit in NvFsmMode

Set/Clear the C/D bit in the NvFsmMode SFR to select CODE or XDATA space for memory loading.

Set/Clear the C/D bit in NvFsmMode command:

APDU command	
CLA	E5h
INS	58h
P1	C/D bit in the NvFsmMode SFR 00 = XDATA space 01 = CODE space
P2	00
Lc	00
Data Field	--
Le	--

Status	Meaning
6B00h	Incorrect **INS** value
9000h	Correct execution

17.7.3.9 Bulk Memory Checksum command

- The command to initiate and perform a CRC16 memory checksum is shown in section 17.11
- Memory checksum will be performed on all memory up to the last byte before the bootloader start address expects the first 3 bytes of UCM
- If an incorrect checksum comparison occurs, an incorrect checksum status word will be sent
- The code checksum will be calculated from the top byte of the code space location (F5FFh) down to first location of the code space (0803h), apart from the last 3 bytes that have to be downloaded after the checksum command
- The XDATA checksum will perform a CRC16 calculation from address 0xB800 to 0xFFFF of XDATA space
- The source code of the checksum is located in section 17.11

Bulk Memory checksum command:

APDU command	
CLA	E5h
INS	A8h: Code Space. A4: XDATA space
P1	Checksum Hi
P2	Checksum Lo
Lc	00
Data Field	--
Le	--

Status	Meaning
6A86h	Bulk memory checksum comparison failure
6B00h	Incorrect **INS** value
9000h	Correct execution

17.7.4 Transport Key and Code

The transport code is an OS that secures the transport of the device. It is located at the bottom part of the UCM. Once the correct key is inserted, this OS auto-erases. Only the “Unlock Transport code” command will be available for unlocking the device.

- Transport key size will be:
  - 8 bytes when unlock device command will be used
  - 5 bytes when the key will be sent directly to the card (for backward compatibility)

- The 8 bytes transport key will be stored in the firmware space
- If the 8 bytes transport key has been loaded into the firmware space with the “load key” command, it will not be possible to send the 5 bytes key directly. The unlock device command will have to be applied
- If the 8 bytes has not been loaded into the firmware space with the “load key” command, it is not possible to send the key. Then the 5 bytes default Emosyn keys are applicable. A firmware API will compare the transport key received from the corresponding APDU command and key stored in the firmware
- The 8 byte key can be downloaded during CP2 stage in Emosyn-SST approved partner’s factories
- If a specific customer key is downloaded at CP2, the product will be a CSPEC part
- If the customer does not require any specific transport key, the default Emosyn transport key will be downloaded
- The key can be loaded only once. An additional checksum will follow the key data in the key downloading data to ensure a successful key downloading
- The Value E5h shall not be used for the first byte of the key
- The value 90 shall not be used for the second byte of the key

17.7.4.1 Unlock Device Command

APDU command	
CLA	E5h
INS	AAh
P1	00
P2	00
Lc	0x0A
Data Field	8 Bytes: Transport Key value
	Addition checksum [2 bytes]
Le	--

Status	Meaning
6A80h	Incorrect checksum comparison
6A81h	Unlock unsuccessful and out of retries. Card permanently locked
6A83h	Unlock unsuccessful but not definitively locked
9000h	Correct execution

17.7.4.2 Load key command

APDU command	
CLA	E5h
INS	A2h
P1	Key type 00 = Transport Key 01= Encryption Key
P2	00
Lc	0x0A: Transport code, 0x12 Encryption key
Data Field	Transport Key: 0x08 bytes plus 2 bytes checksum Encryption Key: 0x10 bytes plus 2 bytes checksum
Le	--

Data Field:

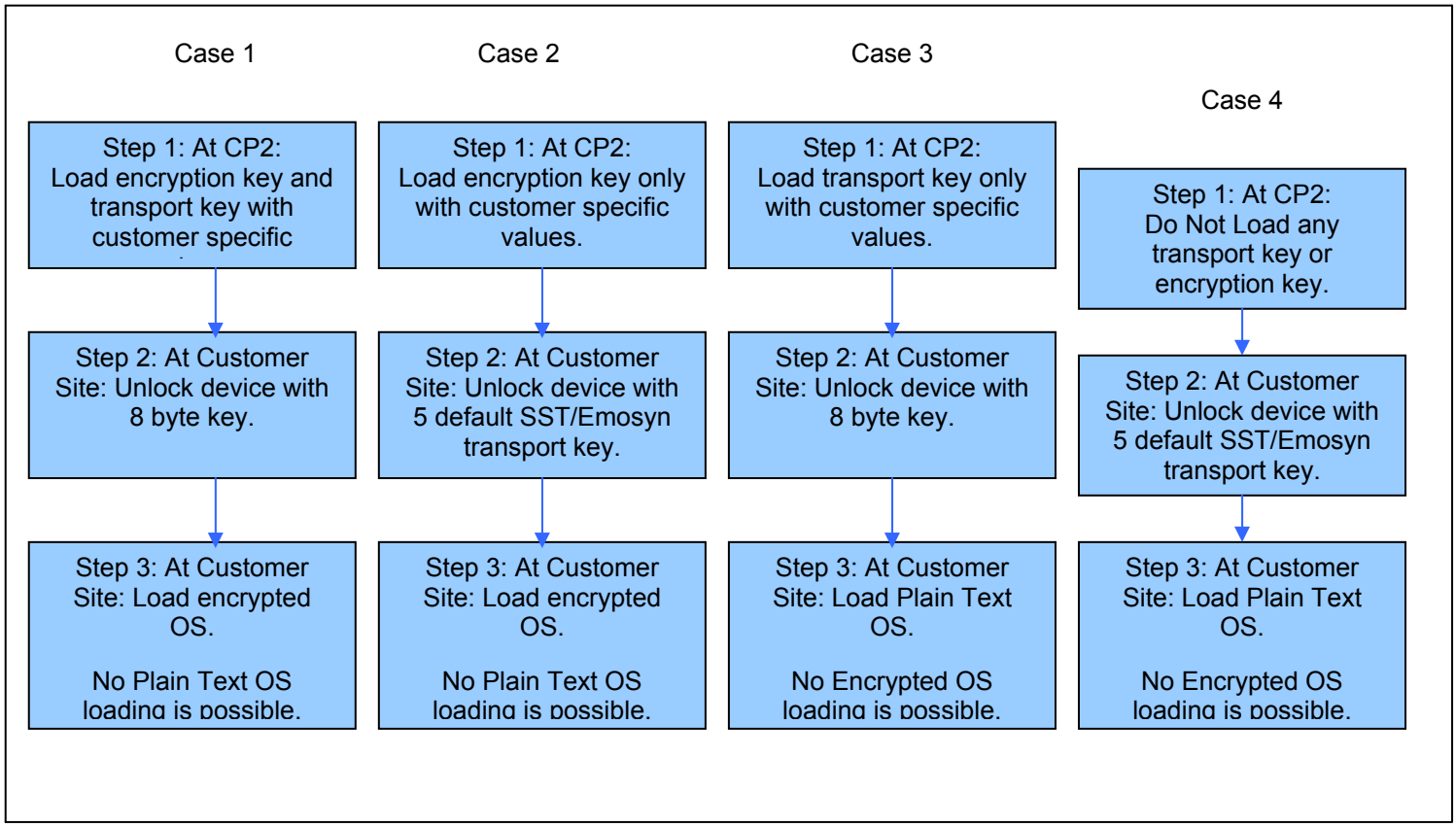
Transport Key: 8 Bytes key. 2 Bytes checksum, Key sent unencrypted in a plaintext format.

Encryption Key: 16 Bytes key. 2 Bytes checksum, Key sent unencrypted in a plaintext format.

Status	Meaning
6A80h	Incorrect checksum comparison
6A82h	Command not available
6A84h	Key already downloaded
6B00h	Incorrect **INS** value
9000h	Correct execution

17.7.5 Key loading flow charts

- Loading the key is optional
- If any key is loaded at CP2 stage of SST approved factories, the devices will be subject to a CSPEC
- The customer specific encryption key can be downloaded either at CP2 or after unlocking the device at customer's site



17.7.6 OS Encryption

- The encryption algorithm is based on a private symmetric SST algorithm
- If no key is downloaded, the OS will be downloaded in plaintext
- If a key is downloaded, the OS will be decrypted before loaded in memory
- Encryption key is fixed to a 16 byte length
- The key may be optionally stored during CP2 in SST approved partner's factories. The key may also be loaded through a Bootloader APDU command
- The key can only be loaded once. An "addition checksum" shall be performed to validate that the key is correct
- Once a key is downloaded, no OS can be downloaded unencrypted. If a key is downloaded at CP2, the product will be a CSPEC part

- The sequence for encrypting data and generating the checksum from the OSOFG in the write command as follows:
  - The data field including the DPX byte is encrypted from Data [0] to data [0xFF] (if total length is 0xFF)
  - P2 (Add LSB) is encrypted
  - P1 (Add MSB) is encrypted
  - The serial checksum is calculated with P1 (address MS byte), P2 (address LS byte), the complete data field, including the DPX byte. It is calculated on encrypted data
  - The serial checksum is NOT encrypted and appended to the command data
  - There is no security issue with it as the CKS is calculated on encrypted data

17.7.7 Diagnostics and error handling commands

- SST will provides the following mechanisms to recover from an error:
  - ATR status updated to indicate the nature of the problem
  - Mass erase UCM preserving the bootloader memory area
  - Page erase command 256

17.7.7.1 Mass Erase UCM bootloader safe

- Mass erase UCM command supplied
- Bootloader space will not be erased
- The 4 Error code bytes will be erased to 0xFF after a mass erase.
- The Status byte will be reset.

Mass erase command

APDU command	
CLA	E5h
INS	30h
P1	00
P2	00
Lc	00
Le	--

Status	Meaning
6B00h	Incorrect **INS** value
9000h	Correct execution

17.7.7.2 Page Erase 256 Bytes

- A page erase function will be provided via an APDU
- The erase page command erases 256 bytes
- The four Error code bytes will set to 0xFF if the erase page command erases the same page as the one indicated with the Error code address. This feature is not available on Theseus Titanium 80 v2.0 Bootloader Beta 0.9
- The Status byte will be reset only if 1 write data command has been sent. This feature is not available on Theseus Titanium 80 v2.0 Bootloader Beta 0.9

17.7.7.3 Page Erase Command

APDU command	
CLA	E5h
INS	3Ch: XDATA - 36h: CODE
P1	Address Hi
P2	Address Low
Lc	01
Le	--
Data Field	DPX byte

Status	Meaning
6B00h	Incorrect **INS** value
9000h	Correct execution

17.7.8 Device information related commands

17.7.8.1 Get Chip ID

- This command provides the 8 bytes chip ID

Get Chip ID Command

APDU command	
CLA	E5h
INS	34h
P1	00h
P2	00h
Lc	--
Le	08h
Data Field	Chip ID [0:8]

Status	Meaning
6B00h	Incorrect **INS** value
9000h	Correct execution

17.7.8.2 Get Firmware Version

- This command provides the Firmware version coded on 1 byte

Get Firmware command

CLA	E5h
INS	32h
P1	00h
P2	00h
Lc	--
Le	01h
Data Field	Firmware version

Status	Meaning
6B00h	Incorrect **INS** value
9000h	Correct execution

17.8 Non-Functional Specifications

17.8.1 Chip Setup

- Chip will be setup with minimal restrictions during programming:
  - No security faults
  - No interrupts
  - No resets

17.8.2 Bootloader Location

- Bootloader is located at the top of the code space
- Bootloader will start at the lower page boundary

17.8.3 Reader Frequency Range

- SST silicon can support a maximum of 10MHz on the ISO clock

17.8.4 PPM Failures during Production

- As the PPM failure rates highly depends on the production environment, SST cannot guarantee the PPM failure rate, but provides the ability to detect and notify errors via the ATR and status word error code

17.9 Core Bootloader: Functional Specification

The Core bootloader uses the S-Algorithm for downloading. It lives in a single 256 byte UCM page. If the developer wishes to use the Core Bootloader during development they will need to erase the pages from C: 0 x F700h to C: 0 x FFFF (this space contains the Normal and APDU Bootloader).

This Core Bootloader lives in the space F600->F6FF.

Core Bootloader Commands	Description
50h	Write a data block of X bytes to UCM
12h	Change ISOTMR value
30h	Mass Erase UCM – not including the bootloader

17.9.1 Command 50 - Write UCM

Write a data block of X bytes to UCM. X is fixed at 40h bytes.

Logical Address Extended is 00 in the case of the Theseus Titanium 80. This byte has been added into the algorithm to make it future proof as well as being backwards compatible with the Theseus Platinum 100-2/130's K-Algorithm.

The value of the Memory checksum is calculated by the bootloader reading back the data that has just been programmed and summing this.

The value of the Serial checksum is calculated by the bootloader summing the serial traffic it received - not including 50 command byte:

- If the Serial Checksum is correct, but the Memory checksum is not correct then there is a problem with the Memory, as the card received the correct data
- If the Serial Checksum is not correct then there is a problem with the connection to the card

Incoming data	
Write UCM command	50h
Address	0xXX: Logical address extended 0xXX: Logical address High 0xXX: Logical address Low
Data bytes:	Data byte 0 Data byte 1… Data byte 0x3E Data byte 0x3F

Outgoing data	
Memory checksum	0xXX: Memory Checksum High 0xXX: Memory Checksum Low
Serial checksum	0xXX: Serial Checksum High 0xXX: Serial Checksum Low

17.9.2 Command 12 - Change ISOTMR

Change ISOTMR value. Use this command to change from the default value of /372 to faster values.

E.g. 12 00 1E will tell the bootloader to change to /31 and expect data at this new divisor rate.

Incoming data	
Change ISOTMR	12h
ISOTMR value	0xXX: ISOTMR Value High 0xXX: ISOTMR Value Low

Outgoing data
No data sent back for backward compatibility

17.9.3 Command 30 - Mass Erase UCM - Bootloader Safe

Mass Erase all of the UCM – not including the Bootloader.

Incoming data	
Mass erase command	30h

Outgoing data	
Success Status word	0x90 0x00

17.10 Normal Bootloader: Functional specification

This is the full functionality S-Algorithm bootloader. The available commands are listed in Table 24.

Normal Bootloader Commands	Description
50	Write a data block of X bytes to UCM
12	Change ISOTMR value
30	Mass Erase UCM - not including the bootloader
55	Change to 60 MHz clock
51	Change data block size
60	Write an encrypted data block of X bytes to NVM
65	Receive 16 byte decryption key
59	Set / Clear C/D bit in NvFsmMode SFR

Table 24: Availability Commands

17.10.1 Command 50 - Write UCM

Write a data block of X bytes to UCM. X is 40 bytes by default and can be changed using the 51 command.

Logical Address Extended is 00 in the case of the Theseus Titanium 80. This byte has been added into the algorithm to make it future proof as well as being backwards compatible with the Theseus Platinum 100-2/130's K-Algorithm.

The value of the Memory checksum is calculated by the bootloader reading back the data that has just been programmed and summing this.

The value of the Serial checksum is calculated by the bootloader summing the serial traffic it received - not including 50 command byte:

- If the Serial Checksum is correct, but the Memory checksum is not correct then there is a problem with the Memory, as the card received the correct data
- If the Serial Checksum is not correct then there is a problem with the connection to the card

Incoming data	
Write UCM command	50h
Address	0xXX: logical address extended 0xXX: Logical address High 0xXX: Logical address Low
Data bytes:	Data byte 0 Data byte 1… Data byte 0x3E Data byte 0x3F

Outgoing data	
Memory checksum	0xXX: Memory Checksum High 0xXX: Memory Checksum Low
Serial checksum	0xXX: Serial Checksum High 0xXX: Serial Checksum Low

17.10.2 Command 12 - Change ISOTMR

Change ISOTMR value. Use this command to change from the default value of /372 to faster values.

E.g. 12 00 1E will tell the bootloader to change to /31 and expect data at this new divisor rate.

Incoming data	
Change ISOTMR	12h
ISOTMR value	0xXX: ISOTMR Value High 0xXX: ISOTMR Value Low

Outgoing data
No data sent back for backward compatibility.

17.10.3 Command 30 - Mass Erase UCM - Bootloader Safe

Mass Erase all of the UCM – not including the Bootloader.

Incoming data	
Mass erase command	30h

Outgoing data	
Success Status word	0x90 0x00

17.10.4 Command 60 - Write Encrypted UCM

Write a data block of X encrypted bytes to UCM. X is 40 bytes by default and can be changed using the 51 command.

The algorithm that this command uses is a propriety Emosyn algorithm.

Logical Address Extended is 00 in the case of the Theseus Titanium 80. This byte has been added into the algorithm to make it future proof as well as being backwards compatible with the Theseus Platinum 100-2/130's K-Algorithm.

The value of the Memory checksum is calculated by the bootloader reading back the plaintext data that has just been programmed and summing this.

The value of the Serial checksum is calculated by the bootloader summing the serial traffic it received - not including 50 command byte:

- If the Serial Checksum is correct, but the Memory checksum is not correct then there is a problem with the Memory, as the card received the correct data
- If the Serial Checksum is not correct then there is a problem with the connection to the card

In the syntax diagram below, the White boxes indicate plaintext data to and from the card, while Yellow boxes indicate encrypted data to and from the card.

Incoming data	
Write encrypted UCM command	60h
Address	0xXX: logical address extended. Encrypted byte 0xXX: Logical address High. Encrypted byte 0xXX: Logical address Low. Encrypted byte
Data bytes:	Encrypted data byte 0 Encrypted data byte 1… Encrypted data byte 0x3E Encrypted data byte 0x3F

Outgoing data	
Memory checksum	0xXX: Memory Checksum High 0xXX: Memory Checksum Low
Serial checksum	0xXX: Serial Checksum High 0xXX: Serial Checksum Low

17.10.5 Command 55 - Change to 60 MHz Clock

This command changes the internal CPU clock to its maximum speed of 60 MHz. This allows downloading on readers that can source the extra current (<10mA) that this speed requires. This results in a faster downloading and therefore programming speed.

Incoming data	
Change to 60MHz Clock	55h
ISOTMR value	0xXX: ISOTMR Value High 0xXX: ISOTMR Value Low

Outgoing data
No data sent back for backward compatibility

17.10.6 Command 51 - Change S-Algorithm Block Size

This command lets the downloading equipment change the bootloader block size from 01 to 00 (256 decimal) bytes. Changing to 00 (256 decimal) will result in a faster downloading and programming speed as the percentage of protocol bytes decreases with larger block size.

Incoming data	
Change to 60MHz Clock	51h
ISOTMR value	0xXX: Data block size

Outgoing data
No data sent back for backward compatibility

17.10.7 Command 65 - Load Decryption Key

This command loads in the 16 byte decryption key. Further processing is done on this key by the bootloader so there is no harm in it being sent in plaintext.

Incoming data	
Load Decryption Key	65h
Key data	0xXX: Key byte 0 0xXX: Key byte 1… 0xXX: Key byte E 0xXX: Key byte F

Outgoing data	
Success Status word	0x90 0x00

17.10.8 Command 59 - Set C/D bit in NvFsmMode

This command sets/clear the C/D bit in NvFsmMode to select CODE space or XDATA space for loading.

C/D Bit = 00 = Select XDATA space
C/D Bit = 01 = Select CODE space

Incoming data	
Set C/D bit in NvFsmMode command	59h
Data field	0xXX: C/D bit

Outgoing data	
Success Status word	0x90 0x00

17.11 CRC16 Checksum

The source code that is shown below is for reference only; it is shown in example in case the User wants to calculate any checksum in the top XDATA area (above B800h).

Note The OFG does not currently use this area to store any code.

```
/*****************************************************************/
/*                                                               */
/* Function Name:      CRC16                                     */
/* Function Inputs: Pointer on Result_CRC16 which is passed as initial value
for CRC16 calculation                                             */
/*
* data_buf: pointer on buffer containing data to calculate        */
/*unsigned int Data_length: length of data -1: length 0 = CRC16 on 256
bytes   */
/* Function Outputs:   void                                      */
/* Description:
/* Calculation of CRC according to x^15+x^12+x^5+1               */
/* Input:                                                        */
/*       data_buf - pointer to data buffer                       */
/*       data_len - length of data -1. Max length 256 bytes.     */
/* Result output:
/*      CRC16 value.                                             */
/*****************************************************************/

void CRC16 (unsigned int data * Result_CRC16, unsigned char xdata *
data_buf, unsigned char Data_Length)
{
  /* Current value of CRC */
  unsigned short CRC16_m;
  unsigned short A;
  /*Cycle loop variable */
```

```
            /* initial value is 0 */
            CRC16_m = *Result_CRC16;
            /* Cycle loop */

            do{

               CRC16_m = CRC16_m  ^ data_buf[--Data_Length];
               A = (CRC16_m ^ (CRC16_m <<4)) & 0x00FF;
               CRC16_m = (CRC16_m >> 8) ^ (A <<8) ^ (A<<3) ^ (A>>4);
             }
            while(Data_Length != 0);
                 *Result_CRC16 = CRC16_m;

          }
/*****************************************************************************
*
* Name:        BulkChecksum_TT80.c
* Description:
*
* Copyright:    Emosyn Inc. 2005
*
* The Example Code herein below is provided "AS IS" and at your sole risk.
* As to this Code, EMOSYN expressly disclaims all warranties, including, but
* not limited to the implied warranties of merchantability and fitness for a
* particular purpose. EMOSYN does not warrant that the functions contained in
* this code will meet your requirements, or that the operation of the code will
* be accurate, error-free, or non-infringing. No oral or written information or
* advice given by EMOSYN or an authorized representative shall create or affect
* any Warranty.
*****************************************************************************
* $Revision: 1.3 $
* $Author: xquillien $
* $Date: 2006/01/06 00:44:55 $
*****************************************************************************/
/*****************************************************************************
* Header file inclusions
*****************************************************************************/
#include "Theseus_Titanium_Family.h"
/*****************************************************************************
* external functions declaration
*****************************************************************************/
void CRC16 (unsigned int data * Result_CRC16, unsigned char xdata * data_buf,
unsigned char Data_Length);
/*****************************************************************************************/
/*
 */ Function Name:      Calculate_Bulk_Memory_Checksum_TT80                    */
/* Function Inputs: These functions calculate a complete CRC16 memory
checksum                                                                      */
/* from address F3FBh down to 903h in code space and FFFFh to B800h in xdata
/* memory space                                                               */
```

```
/* Function Outputs:   void                                                    */

/* Description:                                                               */
/* Calculation of CRC according to x^15+x^12+x^5+1                            */

/* Input:                                                                     */
/*Void                                                                        */
/*    result put into global variable:Actual_Bulk_CKS                        */
/*****************************************************************************/
            unsigned int data * Actual_Bulk_CKS;
            unsigned char data Actual_Bulk_CKS_Hi;
            unsigned char data Actual_Bulk_CKS_Lo;
            void Calculate_Bulk_Memory_Checksum_TT80(unsigned char
            xdata_0_code_1)
            {
             *(Actual_Bulk_CKS) = 0;
             if (xdata_0_code_1 == 1)
             {
                 // read in code space
               NvFsmMode =0x01;
               // checksum from F5FF to 09FB (included)
               CRC16 (Actual_Bulk_CKS, 0x803, 0xEDFD);
             }
             else
             {
                // this code is let for future reference.
               // calculate all top, 4800h bytes, of the xdata space.
               NvFsmMode&= 0xFE; // reset bit 0
             NvFsmMode|=0x04;
             CRC16 (Actual_Bulk_CKS, 0xB800, 0x4800);
             Actual_Bulk_CKS_Lo = (unsigned char) (*Actual_Bulk_CKS)
             Actual_Bulk_CKS_Hi = (unsigned char) ((*Actual_Bulk_CKS)>>8);

            }
            }
```

This page is intentionally left blank

18. Glossary of Terms

This section contains a list of the Acronyms that have been used within this document.

A-C

API	Application Program Interface
ATR	Answer To Reset
CLK	Clock
CPU	Central Processing Unit
CMOS	Complimentary Metal-Oxide Semiconductor

D-E

DLL	Dynamic Link Library
DPTR	Data Pointer
EEPROM	Electronic Erasable Programmable Read Only Memory
ESOC	Emosyn System On Chip
ETU	Elementary Time Unit

F-H

FPGA	Field Programmable Gate Array
FS	Function Stack
GSM	Global System for Mobile communication
HV	High Voltage

I-L

IDE	Integrated Development Environment
ISO	International Standards organisation
ISR	Interrupt Service Routine
LSM	Locked Secure Memory
LSB	Least Significant Byte

M-P

MMU	Memory Management Unit
MFCON	Memory Fault Control
NVM	Non Volatile Memory
NVDR	Non Volatile Data offset Register
OPTROM	One Time Programmable Read Only Memory
PLL	Phase Locked Loop
PEME	Parity Error Management Enable
PFLT	Physical Fault

Q-R

RAM	Random Access Memory
RI	Receive Interrupt
RFU	Reserved for Future Use
ROM	Read Only Memory
RST	Reset

S-V

SFR	Special Function Register
SINT	Software Interrupt
SBUF	Serial Buffer
SOC	System On Chip
SP	Stack Pointer
TI	Transmit Interrupt

W-Z

This page is intentionally left blank

19. Document Control

This section details the current document control/version number:

Date	Name	Comment	Version Number
26-01-2006	Glenn Marsh	Firmware API list revised Document content revised	6.1
07-02-2006	Glenn Marsh	APDU Bootloader section added	6.1A
13-02-2006	Glenn Marsh	APDU Bootloader section revised	7.0
22-03-2006	Glenn Marsh	Table 2 revised Table 3 revised Entering Deep Idle Mode (Stop Clock Mode) added The Way to Exit Idle & Deep Idle Mode (Stop Clock Mode) added Set Clock Trim revised Set Security Mask revised Set Execute Only Status revised	8.0

This page is intentionally left blank

20. Summary of Recent Document Changes

This section gives details of the three most recent changes that have been made within the document and those area(s) that have been affected by the change(s):

Date	Revised Document Version #	Change to:	The information that has been changed is shown within
07-02-2006	6.1A	APDU Bootloader section added	Section 17.5
13-02-2006	7.0	APDU Bootloader section revised	Section 17.5
22-03-2006	8.0	Table 2 revised Table 3 revised Entering Idle & Deep Idle Mode (Stop Clock Mode) added The Way to Exit Idle & Deep Idle Mode (Stop Clock Mode) added Set Clock Trim Set Security Mask Set Execute Only Status	Section 2.1.2 Section 2.1.6 Section 11.4 Section 11.5 Section 15.6 Section 15.9 Section 15.10