paragraph

© Copyright 2015– All rights reserved  
Giesecke & Devrient GmbH  
Prinzregentenstr. 159  
Postfach 80 07 29  
81607 München  
Germany

The information or material contained in this document is property of G&D/GAO and any recipient of this document shall not disclose or divulge, directly or indirectly, this document or the information or material contained herein without the prior written consent of G&D/GAO.

All copyrights, trademarks, patents and other rights in connection herewith are expressly reserved to the Giesecke & Devrient group of companies and no license is created hereby.

All brand or product names mentioned are trademarks or registered trademarks of their respective holders

**Contents**

1 Introduction 10

1.1 Purpose and Scope 10

1.2 References 11

1.3 Abbreviations 12

1.4 Further reading 13

1.5 StarSIM architecture quick Overview 14

2 Overview of project tree 16

3 Target platform 20

3.1 Product Versions & Identifier 20

3.1.1 Product family THC20F17BD-XX 20

3.1.2 Product family SCFxxxx 21

3.2 Chip Properties 22

3.2.1 Product family THC20F17BD-XX 22

 Tongfang Microcontroller THC20F17BD/ THC20F17BD-V20/ THC20F17BD-V30 22

 Features of the Tongfang Flash Chips 22

 Difference Between THC20F17BD and THC20F17BD-V20/ THC20F17BD-V30 23

 Hardware Characteristics 23

 The electrical characteristics are as follows: 24

 NVM characteristics 24

 Hardware Security Features 25

 Flash Quality Aspects 26

 Device Notifications from Tongfang 26

3.2.2 Product family SCFxxxx 31

 Starchip Microcontroller SCF136H 31

 Features of the Starchip Flash Chips 31

 Hardware Characteristics 32

 The electrical characteristics are as follows: 33

 NVM characteristics 33

 Hardware Security Features 34

 Flash Quality Aspects 34

 Device Notifications from Starchip 34

3.3 ATR 36

3.3.1 Specific ATR values 36

3.3.2 Boot Loader ATR 37

3.3.3 Explanation of OS ATR values 38

3.3.4 Error ATR values 39

3.4 Boot loader specifications 40

3.4.1 Product family THC20F17BD-XX 40

3.4.2 Product family SCFxxxx 40

3.5 Protocol Type Selection 40

3.6 Footprint 41

4 OTC 42

4.1 Introduction 42

4.2 SCFXXXX (Mizar) family 42

4.2.1 Basic Idea 42

4.2.2 CreDel 43

 CreDel feature – possible combinations 43

4.2.3 MD 45

4.2.4 Possible combinations with MD and CreDel 46

 Error message related to file system size and reclaim area 47

 EFADN <= Reclaimable size 47

 EFADN > Reclaimable size 47

4.2.5 When the reclaimable functionalities gets blocked 48

4.3 THC20F17BD-XX (Zeta) family 49

4.3.1 Basic Idea 49

4.3.2 Memory Layout 50

4.3.3 CreDel 50

 CREDEL Persistent - Creating and Deleting File 50

 CREDEL Volatile - Creating Files 51

 CREDEL Volatile - Removing the Code 53

 CREDEL Volatile – Miscellaneous – 54

4.3.4 MD – 55

 Basic Idea 55

 Memory Layout – 56

 MD/BL - Creating Files 57

 MD/BL Volatile - Removing the Code – 58

 Miscellaneous 59

 MD/BL Space reclaiming – Note for Production Process – 59

4.4 Switches description 60

4.4.1 MASTER\_DEVICE\_ENABLED – 60

4.4.2 ADN\_IN\_MD\_INIT – 60

4.4.3 ADN\_IN\_CREDEL\_INIT – 60

4.4.4 ADN handling – 60

4.5 Profiler flags related to MD – 61

4.6 OTC Commands – 62

4.6.1 SCFXXXX family (Mizar) – 62

 Enable Download 62

 Read Flash 62

4.6.2 THC20F17BD-XX (Zeta) – 64

 MD\_ENABLE\_PERSO 64

 MD\_ACTIVATE 64

 MD\_READDATA 66

5 Installation of required tools 68

5.1 Rational ClearCase UCM 68

5.2 Visual Studio C++ 2005 (Version 8) 68

5.3 Perl 68

5.4 Tools required only for THC20F17BD-XX family 68

5.4.1 Keil Toolchain for THC20F17BD-XX 68

5.5 Tools required only for SCFXXXX family 70

5.5.1 Cortus Toolchain for Starchip 70

6 Build Process 72

6.1 Product family THC20F17BD-XX 72

6.2 Product family SCFxxxx 86

Options implemented with make command – 88

Error messages shown by the build system 89

7 Memory Layout 92

7.1 THC20F17BD-XX family 92

7.1.1 Flash Organization in Detail 92

7.1.2 Memory Map 93

7.1.3 Memory Structure 95

7.1.4 Flash Memory Overview 100

7.2 SCFxxxx family 101

7.2.1 Memory Map 101

7.2.2 Memory Structure 102

7.2.3 Flash Memory Overview 103

8 G&D bootloader 104

9 TearSafe feature – 105

9.1 SCFXXXX (Mizar) family 105

9.2 THC20F17BD-XX (Zeta) family 105

10 Secure write mechanism – 106

11 THC20F17BD-XX family specific 107

11.1 Generation of an Initialisation 107

11.1.1 Preparation of an Initialisation/Variant 107

11.1.2 Linker file & Sections: 109

11.1.3 NVM File System 111

11.1.4 Determination of the File System Size 112

11.1.5 System Files 113

11.1.6 Batch Files and Executables 114

11.2 Porting Assembly Code 118

11.3 Application Notes 118

11.3.1 More Times on Zeta 118

11.3.2 Optimised File System 118

11.4 Initialization of RAM Variables 119

11.5 Linker adjustments 119

11.6 Linking Strategy of the Project 121

11.7 Banking Modifications 122

11.8 Card programming concept 123

11.9 OVERLAY technique 124

11.9.1 Memory Overlay Concept 124

11.9.2 Scope of Memory Overlay Implementation in Zeta 125

11.9.3 Memory Overlay Implementation in Zeta 126

11.9.4 Challenges in using Memory Overlay in Zeta 128

11.9.5 Memory Overlay Syntax for Keil C51 128

11.9.6 Debugging guidelines for memory overlay 129

12 SCFXXXX family specific 130

12.1 Generation of an Initialisation 130

12.1.1 Preparation of an Initialisation/Variant 130

12.1.2 Linker file & Sections: 133

12.1.3 NVM File System 134

12.1.4 Determination of the File System Size 136

12.1.5 System Files 136

12.1.6 Batch Files and Executables 137

12.2 Porting Assembly Code 140

12.3 Application Notes 140

12.3.1 More Times on Mizar 140

12.3.2 Optimised File System 140

12.4 Initialization of RAM 141

12.5 Loading Concept 141

13 New Release Versions and Patches 147

14 High Update Activity Files 148

14.1 Updating a High Update Activity EF 148

14.2 Page Alignment of Files 148

14.2.1 Aligning file Bodies to Page Boundaries 149

14.3 HU File Creation Using Profiler 149

14.4 Transparent Files 149

14.4.1 Updating a High Update Activity Transparent EF 149

14.4.2 Reading a High Update Activity Transparent EF 150

14.5 Cyclic Files 150

14.5.1 Updating a Cyclic file 150

14.5.2 Reading Cyclic Files 152

14.6 Summary 152

15 Debug Commands 155

15.1 Command Check Stack 155

15.2 Command Timing Test 155

15.3 Command Reset Run GSM Flag 156

15.4 Command Check Concat Buffer 156

16 Appendix 158

16.1 How to debug 158

16.1.1 Family THC20F17BD-XX 158

 Firmware update 160

 Target update 161

 Loading Emulator 161

16.1.2 Family SCFXXXX 163

 Using the Emulator 163

 Overcoming StarChip emulator limitation with linker file. 167

16.2 Feature Switches Summary 168

 Switches in Init.h and Variant.h 168

16.2.1 General Defines 168

16.2.2 WIB1.2 Defines 171

16.2.3 WIB1.3 Defines 172

16.2.4 WIB1.3UltraLite Defines 173

16.2.5 RFM13Lite Defines 175

16.2.6 Celltick App Defines 176

16.2.7 WIB1.3-Celltick App Defines 177

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Remark** |
| 0.1 | 24-02-2015 | Yogita R | Started with StarSIM Zeta V913 & Mizar V913 R&D Document as base |
| 0.2 | 27-02-2015 | Vivek J | Incorporated review comments of Abhay L |
| 1.0 | 27-02-2015 | Vivek J | Baseline |
| 1.1 | 4-03-2015 | Bharti B | Updated references, V30 specific details |
| 1.2 | 3-25-2015 | Bharti B | Intermediate |
| 1.3 | 4-10-2015 | Yogita R, Abhay L | Final |

# Introduction

## Purpose and Scope

This document gives detailed technical information about the StarSIM Card Operating System. The target audience of the document are application developers responsible for configuring and extending the card's functionality.

While a document like this may be useful for mask developers as well, the main goal is to get application developers up to speed with a minimum amount of required reading. Thus, this document cannot (and does not even attempt) to be complete. Specifically, topics like these are not covered

* Detailed requirements
* Architectural subjects
  + StarSIM architecture
  + Portability
  + Coding conventions
* Detailed design documentation
* Static analysis (PC-Lint)
* Developer testing

**Samsung S3FC9XX chips** are not in the scope of StarSIM V914. So they are not released as part of this project. Their data is presented here only for completeness of the document.

## References

|  |  |
| --- | --- |
|  | R&D Documentation GSM Phase 2+/CDMA OS CS3EDGSMFATL\_V860 CS3FDGSMFATL\_V860 CS3IDGSMFATL\_V860 CS3JFF  GSMFATL\_V860 80K/96k/136k/176K/Atlas 176K 1.0 (2008-09-25) |
|  | R&D Documentation GSM Phase 2+/CDMA OS CS3EHGSMFATL\_V910 CS3GHGSMFATL\_V910 CS3IHGSMFATL\_V910 Atlas 96K/136K/176K |
|  | R&D Documentation GSM Phase 2+/CDMA OS CF04AFGSMZET\_V912 / V913 (132K) |
|  | R&D Documentation GSM Phase 2+/CDMA OS CC21AGSMFMIZ\_V913-rd |
|  | StarSIM v914 CHL-2.doc |
|  | R&D Documentation for Celltick application on GSM Phase 2+ SIM Cards Version 1.0.0/03.04.2009 |
|  | SCDS2000\_UM\_EN.pdf (Smart Card Development System User Manual Version 1.3) |
|  | tep006\_starbox\_userguide\_rev5.pdf |
|  | tep023\_scf136h\_datasheet\_rev1x.pdf.pdf |
|  | gcc\_4-7-3.pdf |
|  | STARSIM Mizar Production-Concept.doc |
|  | ld.pdf |
|  | StarSIM v914 Perso Guide |
|  | Eclipse\_Quick\_Start.pdf |
|  | SCDS2HID.pdf (HOW TO UPDATE SCDS DRIVER TO HID V1.0) |
|  | UM2F02-THC20F17BD-UM-EN-US.pdf Version UM2F02 Beta |
|  | UM2F04-THC20F17BD-BLUM-EN-US.pdf Version UM2F04 |
|  | cf04AFx0-Production-Concept.doc Version 0.1 |
|  | TMC-CS-AN-20-2012090002-EN.pdf |
|  | UM3102-THC20F17BD-V20-UM-EN-US.pdf Version UM3102 Beta |
|  | DS3402-THC20F17BD-V30-EN-US.PDF |
|  | UM3402-THC20F17BD-V30-UM-EN-US.PDF |
|  | UM3404-THC20F17BD-V30-BLUM-EN-US.PDF |
|  | StarSIM Design.doc |

## Abbreviations

* ATR Answer to reset
* BL Boot Loader
* CHV Card Holder Verification number
* CRC Cyclic redundancy check
* CREDEL Create/Delete in EFADN Feature
* DF Dedicated File
* E²PROM Electrical Erasable Programmable Read Only

Memory

* EF Elementary File
* etu elementary time unit
* GSM Global System for Mobile communications
* MD Master Device
* ME Mobile Equipment (Telephone without SIM)
* MF Master File
* NVM Non Volatile Memory
* OS Operating System
* OTASS Over the Air SIM Services
* RAM Random access memory
* RFU Reserved for future use
* SATlk SIM Application Toolkit
* SATlkA SIM Application Toolkit Application
* SATlkI SIM Application Toolkit Interface
* SIM Subscriber Identity Module
* SM Short Message
* SMS Short Message Service
* SMS-PP Short Message Service Point to Point
* TP Transfer layer Protocol
* TPDU Transfer Protocol Data Unit
* TP-UDL Transfer layer Protocol User Data Length (part

of a SM header)

* ... Hex Notation for hexadecimal expressions
* ... Dec Notation for decimal expressions

## Further reading

Various StarSIM related documents, like

* OTA specifications
* Coding conventions

are available online on the Telco Info Server.

Other documentation as:

* StaRS
* Personalisation Guide
* Release Notes for Application Centre

Can be found under ClearCase, VOB: StarSIM\_Doc

## StarSIM architecture quick Overview

Below is an architectural overview of StarSIM.

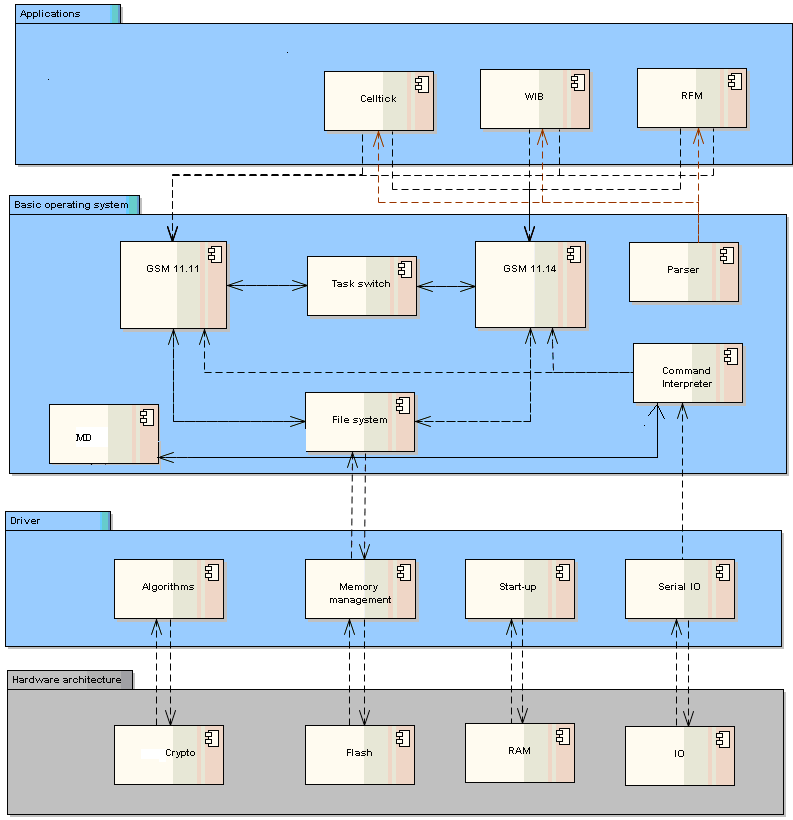


Figure 1‑1 : Architectural Overview

Functionality is grouped in "layers" and "components". Components (e. g. "GSM 11.11") implement a coherent set of functionality and offer this functionality to other components.

Table 1‑1: StarSIM Layers

| Layer | Abbrev. | Description |
| --- | --- | --- |
| Applications | -/- | Native applications include Celltick, WIB, RFM. |
| Basic Operating System | BasicOS | Operating System core components implementing core Telco functionality. |
| Driver | Drv | Platform-independent APIs and logic that gives access to smart card controller devices like UART, NVM. |
| HW Architecture | Arch | Represents platform-specific code (including assembly code), like access to device registers. This layer can be thought of as "vertical"; that is, every other layer/component may implement certain platform-specific aspects in it (e. g. via assembly code for performance reasons). |
|  |  |  |

The structure of StarSIM source code generally follows the concept of layers and components:

For details about component directories see section [2].

# Overview of project tree

src

+---app

| +---AdmCnt

| +---Celltick

| | +---include

| | \---Rb

| | +---include

| | +---StarChip

| | \---TongFang

| +---OTA

| | +---Celltik\_ConCL

| | | \---include

| | +---ConCat

| | | \---include

| | +---Concat\_RingBuffer

| | | \---include

| | \---ConCL

| | \---include

| +---Otass35

| | \---include

| +---Plugins

| | \---include

| +---RFM

| | \---include

| +---RFM13

| | \---include

| +---SAT

| | \---include

| +---WIB12

| | \---include

| \---WIB13

| \---include

+---AutoGen

| \---include

+---Debug

| +---app

| | +---AdmCnt

| | +---Celltick

| | | \---Rb

| | | \---StarChip

| | +---OTA

| | | +---Celltik\_ConCL

| | | +---ConCat

| | | +---Concat\_RingBuffer

| | | \---ConCL

| | +---Otass35

| | \---Plugins

| +---gsm

| | +---11.11

| | +---11.14

| | | \---StarChip

| | \---admin

| +---hal

| +---hware

| | +---StarChip

| | | +---crc

| | | +---Eeprom

| | | +---flash

| | | +---io

| | | +---irq

| | | +---Ram

| | | \---startup

| | \---TongFang

| | +---Eeprom

| | +---flash

| | +---io

| | \---Ram

| +---main

| +---sys

| | +---crypto

| | | +---StarChip

| | | | \---dbg

| | | \---TongFang

| | +---FloatingFunction

| | | \---StarChip

| | +---GdblUtilities

| | | \---StarChip

| | +---MD

| | | +---StarChip

| | | \---TongFang

| | +---MemoryIntegrity

| | +---NVM

| | | +---StarChip

| | | \---TongFang

| | +---RAM

| | | +---StarChip

| | | \---TongFang

| | \---SecWrite

| +---TestOS

| | +---StarChip

| | | \---SCF136H

| | | +---HAL\_Interface

| | | | +---DLib

| | | | +---hw

| | | | | \---chipcfg

| | | | +---IODrv

| | | | +---NVMDrv

| | | | +---RAM

| | | | \---RNG

| | | \---St

| | \---Tongfang

| | \---THC20F17BD

| | +---HAL\_Interface

| | | +---DLib

| | | +---IODrv

| | | +---NVMDrv

| | | \---RNG

| | \---St

| \---UnitTest

+---gsm

| +---11.11

| | \---include

| +---11.14

| | +---include

| | +---StarChip

| | \---TongFang

| \---admin

| \---include

+---hal

| \---include

+---hware

| +---StarChip

| | +---chipcfg

| | | \---SCF136H

| | | \---include

| | +---crc

| | +---Eeprom

| | +---flash

| | +---io

| | | \---include

| | +---irq

| | +---Ram

| | \---startup

| \---TongFang

| +---chipcfg

| | \---THC20F17BD

| | \---include

| +---Eeprom

| +---flash

| | \---include

| +---io

| | \---include

| +---irq

| +---Ram

| \---startup

+---main

| \---include

+---sys

| +---crypto

| | +---StarChip

| | | +---dbg

| | | \---include

| | \---TongFang

| | \---include

| +---FloatingFunction

| | \---StarChip

| +---GdblUtilities

| | \---StarChip

| +---include

| +---IniConfigs

| | +---StarChip

| | | \---SCF136H

| | \---TongFang

| | \---THC20F17BD

| +---MD

| | +---StarChip

| | \---TongFang

| +---MemoryIntegrity

| +---NVM

| | +---include

| | +---StarChip

| | \---TongFang

| +---RAM

| | +---include

| | +---StarChip

| | \---TongFang

| \---SecWrite

+---TestOS

| +---StarChip

| | \---SCF136H

| | +---HAL\_Interface

| | | +---DLib

| | | +---hw

| | | | +---chipcfg

| | | | \---memcfg

| | | +---IODrv

| | | +---NVMDrv

| | | +---RAM

| | | \---RNG

| | \---St

| \---Tongfang

| \---THC20F17BD

| +---HAL\_Interface

| | +---DLib

| | +---hw

| | | \---chipcfg

| | +---IODrv

| | +---NVMDrv

| | +---RAM

| | \---RNG

| \---St

\---UnitTest

\---include

# Target platform

## Product Versions & Identifier

### Product family THC20F17BD-XX

Table 3‑1 : Product Identifier

|  |  |  |  |
| --- | --- | --- | --- |
| **Microcontroller** | **Product Identifier** | **OS Variant** | **Flash (KB)** |
| THC20F17BD | CF04AGSMFZET\_V913 | Zeta 132 | 132 |
| THC20F17BD-V20 | CF04AGSMFZET\_V913 | Zeta 132 | 132 |
| THC20F17BD-V10,V20,V30 | CF04AGSMFZET\_V914 | Zeta 132 | 132 |

Table 3‑2 : Product Identifier Details

|  |  |  |
| --- | --- | --- |
| **Name** | **Code** | **Meaning** |
| SW Type | **C** | Smart card software |
| Chip Manufacturer | **F** | Tongfang |
| Chip | **04A** | THC20F17BD- V10,V20,V30 |
| OS Type | **GSM** | Operating System |
| SW Type | **F** | Flash Mask |
| Mask type | **ZET** | Mask name ZETA |
| Version | **V914** | Version 9.14 |

**Note** : The V10,V20,V30 versions of THC20F17BD are differentiated by unique sequence number for each chip which can be obtained by reading 0th byte of EFIC. Irrespective of differences between these three chip types , GSM operating system OS will have same ATR on these chip’s and same OS can be loaded on all these cards.

### Product family SCFxxxx

Table 3‑3 : Product Identifier

|  |  |  |  |
| --- | --- | --- | --- |
| **Microcontroller** | **Product Identifier** | **OS Variant** | **Flash (KB)** |
| SCF136H | CC21AGSMFMIZ\_V913 | MIZAR 136 | 136 |
| SCF136H | CC21AGSMFMIZ\_V914 | MIZAR 136 | 136 |

Table 3‑4 : Product Identifier Details

|  |  |  |
| --- | --- | --- |
| **Name** | **Code** | **Meaning** |
| SW Type | **C** | Smart card software |
| Chip Manufacturer | **C** | Starchip |
| Chip | **21A** | SCF136H |
| OS Type | **GSM** | Operating System |
| SW Type | **F** | Flash Mask |
| Mask type | **MIZ** | Mask name MIZAR |
| Version | **V914** | Version 9.14 |

## Chip Properties

### Product family THC20F17BD-XX

### Tongfang Microcontroller THC20F17BD/ THC20F17BD-V20/ THC20F17BD-V30

Zeta is the porting of existing StarSIM Atlas product on Tongfang chip THC20F17BD, THC20F17BD-V20 132 KB, THC20F17BD-V30 132 KB (analogous to Atlas 136 KB). Its has both the Test OS and Basic StarSIM OS along with the existing applications

Tool chain used,

- Tool chain: µVision Keil (PK51 Prof. Developers Kit: Version: 9.03)

### Features of the Tongfang Flash Chips

|  |  |
| --- | --- |
| **Feature** | **Zeta132** |
| Chip | THC20F17BD, THC20F17BD-V20,THC20F17BD-V30 |
| FLASH size | 132 KB |
| Flash organisation  (Physical address range) | 0x00000 – 0x20FFF |
| NON ERASABLE DATA (Physical address) | NA |
| Hardware DES | Yes |
| Crypto engine (asymmetric cryptography) | Yes |
| Random Wait States | No |
| Random number generator | Yes |
| Page Size | 512 Bytes |

### Difference Between THC20F17BD and THC20F17BD-V20/ THC20F17BD-V30

The difference between THC20F17BD and THC20F17BD-V20/ THC20F17BD-V30 is the latter supports flash fast erase.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **V10** | **V20** | **V30** |
| **Fast erase** | Not supported | Supported | Supported |
| **Consecutive write** | Not supported | Not supported | Supported |
| **Clock freq** | 30MHz | 30MHz | 35MHz |
| **Divider 8** | Not supported | Not supported | Supported |

On THC20F17BD-V20/THC20F17BD-V30 Mask supports Fast Erase Mode and on THC20F17BD OS support Normal erase mode.No separate variant definations are required for these three chip types and same OS can be loaded and executed on these. The chips can be identified by 0th byte of EFIC ie. Unique Sequence Number as shown in below table.

|  |  |  |
| --- | --- | --- |
| No. | Chip | Unique Sequence Number |
| 1 | THC20F17BD | 0x2F |
| 2 | THC20F17BD-V20 | 0x31 |
| 3 | THC20F17BD-V30 | 0x34 |

### Hardware Characteristics

The THC20F17BD/THC20F17BD-V20/ THC20F17BD-V30 chip is an 8-bit single-chip CMOS microcontroller for smart card with the following characteristics:

1. 8-bit TMCU\_051 CPU core
2. CRC co-processor, True Random Number Generator
3. Timer0,Timer1 and ETU Timer
4. Interrupt Controller
5. UART Interface, supporting ISO 7816-3 T=0 and T=1
6. Power down mode for reduced power consumption

Support Page Erase and Byte Program (512 bytes per Page). Page Erase is always needed before Byte update.

1. The memory dimensions are:

- For 132K

|  |  |  |  |
| --- | --- | --- | --- |
| **ROM (Kbytes)** | **RAM (KBytes) + SRAM(Bytes)** | **Flash (KBytes)** | **SN(Bytes)** |
| 0 | 2 + 256 | 132 | 17 |

### The electrical characteristics are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Conditions** | **Min** | **Typical** | **Max** |
| Supply Voltage [V] |  | 1.62 |  | 5.5 |
| Supply Current[mA] | Vcc = 5.0 V |  |  | 10 |
| Vcc = 3.0 V |  |  | 6 |
| Vcc = 1.8 V |  |  | 4 |
| Standby Current[mA] (Clock Stop) | Vcc = 5.0 V |  |  | 200 |
| Vcc = 3.0 V |  |  | 100 |
| Vcc = 1.8 V |  |  | 100 |
| Internal Clk Frequency [MHz] |  | 7.5 |  | 30 |
| External Clk Frequency [MHz] |  | 1 |  | 5 |
| Ambient Temperature [◦C] |  | -40 |  | 85 |

### NVM characteristics

The NVM is divided into two parts. The first part is of 132KB (for THC20F17BD/ THC20F17BD-V20/THC20F17BD-V30) of flash which is used for OS & Application. The Second part of from 0x0000 to 0x0225(For 132K Chip) is OTP (One Time Programmable) area. This area contains Manufacturing information (Initialised by TMC, User should not modify), serial number & User OTP area from 0x20 to 0xFF which can be configured by users. OTP is not accessible in execution mode, so it is not used to save any OS information.

For 132KB, physically Flash is divided into five sections. For 132KB - Common area of 32KB , Bank 0, Bank 1 & Bank 2 each of 32KB & Bank 3 of 4KB. These sections are physically partitioned into pages of 512 bytes. The smallest physical unit that can be erased separately is also one page size, i.e.512 bytes. Byte write is supported. The erase operations change the selected bytes of the page to FF Hex and the write operations modify only those bits in the byte that must be changed from 1 to 0. Again, as it can not be told often enough, if only one bit has to be changed from 0 to 1 the whole page of 512 bytes has to be erased first! For writing into a page fully or partially, it must be erased before proceeding.

CPU can address four types of logical spaces.

1. 1. CODE: Code space, 64 KB, consists of Flash only.
2. 2. XDATA: External data space, 64 KB, indirect addressing only, consists of RAM and Flash.
3. 3. IDATA: Indirect addressing internal data space, 256 bytes, indirect addressing only, consists of RAM only.
4. 4. DATA: Direct addressing internal data space, 256 bytes, consists of RAM and SFRs.

The time for erasing a page is 4ms for 132KB.

The time for writing a byte is 25 us for 132KB.

(**Note:** If the page is in already in erased state, then a ‘single byte’ write takes 25us.

If the page is not in erased state, then a ‘single byte’ write takes around 4ms + 12.8 ms = 16.8 ms time. Hence the best case ‘single byte’ write time is 25us and worst case single byte write time is 16.8 ms)

### Hardware Security Features

The chip provides the following security features:

1. Write-protection for a configurable FLASH area
2. Scrambling data storage
3. High/low voltage and high/low clock frequency detectors
4. CLK filter
5. Security Certification Targeted: Common Criteria EAL4+

Under abnormal condition or if the access conditions are not fulfilled the card can be configured to reset itself.

### Flash Quality Aspects

Tongfang states:

1. Min. 10 years minimum data retention(25 C)
2. M**i**n **100,000** erase/write cycles per page(25 C)

This means that each page can reliably be written up to 100,000 times with no appreciable degradation in data integrity or device functionality. This number is comparatively low and thus does not allow implementing a proper Roll Back mechanism.

The hardware characteristics (The huge page size of 512 bytes, only byte write supported, page erase supported, page write not supported) cause several severe problems which have to be handled by the OS (see High Update Files and Secure Write mechanism).

### Device Notifications from Tongfang

While implementation of application software, please check notifications received from Tongfang.

I] As per REF [16], Tongfang has issued following notifications related to flash operations.

The notifications list the following concepts to be used for flash operations.

**Flash erase/write procedure for chip THC20F17BD, THC20F17BD-V20 and THC20F17BD-V30**.**:**

For THC20F17BD/THC20F17BD-V20/THC20F17BD-V30, size of each page is 512 bytes. It does not support sector read operation. Only page erase and byte write is supported.

**Page Erase Flow**

1. Write 01h to SFR FL\_CON.PMOD.

Write 00h to SFR FL\_CON.EMOD.

Write 00h to SFR FL\_CON.ETYP.

Write 01h to SFR FL\_CON.EV\_EN.

2. Write 55h to SFR FL\_SDP1 first and then write AAh to SFR FL\_SDP2.

3. Write an FFh to start address in target page.

4. Detect SFR FL\_STS.F\_OVER. When FL\_STS.F\_OVER = 1, Flash

operation finish.

5. Detect SFR FLSTS.OP\_ERR.

6. Detect SFR FLSTS.EV\_ERR.

7. Clear SFR FL\_STS.

**Fast Page Erase Flow (Applicable only for THC20F17BD-V20 and THC20F17BD-V30)\***

1. Write 01h to SFR FL\_CON.PMOD.

Write 01h to SFR FL\_CON.EMOD.

Write 00h to SFR FL\_CON.ETYP.

Write 01h to SFR FL\_CON.EV\_EN.

Write 01h to SFR FL\_CON.EV\_VREAD.

2. Write 55h to SFR FL\_SDP1 first and then write AAh to SFR FL\_SDP2.

3. Write an FFh to start address in target page.

4. Detect SFR FL\_STS.F\_OVER. When FL\_STS.F\_OVER = 1, Flash operation finish.

5.Detect SFR FLSTS.OP\_ERR.

6. Detect SFR FLSTS.EV\_ERR.

7. Clear SFR FL\_STS.

8. If FLSTS.EV\_ERR is set to 1 by hardware, it is suggested to repeat step1 to 7 to retry erasing. The maximum times for retry should be 5. When the 5th times retry, SFR FL\_CON must be configured to 0x13.

\*Current Mask supports Normal Erase mode for THC20F17BD and Fast Erase Mode for THC20F17BD-V20 and THC20F17BD-V30.

**Byte Write Flow**

1. Write 01h to SFR FL\_CON.

2. Write AAh to SFR FL\_SDP1 first and then write 55h to SFR FL\_SDP2

3. Write data to target address.

4. Detect SFR FL\_STS.F\_OVER. When FL\_STS.F\_OVER = 1, Flash

operation finish.

5. Detect SFR FLSTS.OP\_ERR.

6. Clear SFR FL\_STS.F\_OVER.

**Consecutive Byte Program Flow (Applicable only for THC20F17BD-V30)**

1. Configure SFR FL\_RPVAD and SFR FL\_RPVAD1 to set the start address of source data.

2. Write data to RAM area specified by FL\_RPVAD and FL\_RPVAD1.

3. Configure SFR FL\_RPVLEN and SFR FL\_RPVLEN1 to set the length to program.

4. Write 1 to SFR FL\_CON.PTYE to choose the Consecutive Byte Program Mode.

5. Enable interrupt (Optional).

6. Write 0xAA to SFR FL\_SDP1 and write 0x55 to SFR FL\_SDP2.

7. Write the first byte of expected data to target start address in FLASH memory to start the Consecutive Byte Program.

8. Detect SFR FL\_STS.F\_OVER or wait for the interrupt. When OVER = 1, flash operation finish.

9. Detect SFR FLSTS.OP\_ERR.

10. Mask interrupt (Optional).

Note: The consecutive byte program must be used in a half-page.

For Example :

|  |  |  |
| --- | --- | --- |
| Operation | Start Address | Length to Write |
| Correct Operation | 0x8000 | 1~256 Bytes available |
| 0x8080 | 1~128 Bytes available |
| 0x80FF | 1 Byte available |
| 0x8100 | 1~256 Bytes available |
| 0x8180 | 1~128 Bytes available |
| 0x81FF | 1 Byte available |
| Wrong Operation | 0x8000 | 257 Bytes |
| 0x81FF | 2 Bytes |

Note: THC20F17BD & THC20F17BD-V20 use normal byte write whereas THC20F17BD-V30 use Consecutive byte write.

**Loading OS to card**

During loading OS to the card, following steps should be followed.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Status** | **Notes** |
| 0 | Power On | BL is activated now. Any card reader compatible with ISO7816 standard can test the card with BL commands. |
| 1 | Verify BL | The command “Check CRC” should be used to verify BL. （Check CRC） |
| 2 | Erase FLASH  (except for the 1st  page) | Flash must be erased firstly, which will be programmed. |
| 3 | Program FLASH  (except for the 1st  page) | Area to be download should not include:   * First page of FLASH * BL area * Other areas that need to be reserved. (Write Flash) |
| 4 | Verify(except for the 1st page) | The command “Check CRC” should be used to verify  downloaded data (Check CRC） |
| 5 | Write 1st page | Update the vector table in the 1st page.  (Erase & Write Flash) |
| 6 | Reset | After reset, user COS starts to run. |

In step 5, the reset vector should be downloaded at the end, i.e. [C: 0x0000 – C: 0x0080] should be loaded to the card at the end after loading rest of the flash, otherwise there are chances of damaging card.

### Product family SCFxxxx

### Starchip Microcontroller SCF136H

Mizar is the porting of existing StarSIM Zeta product on Starchip chip SCF136H 136KB (analogous to Zeta 132 KB). It has both the Test OS and Basic StarSIM OS along with the existing applications.

- Tool chain :

For this you need to install “cortus-aps-ide-setup-20131213-release.exe”.

It will install following tool on your PC,

1. Eclipse IDE
2. GCC (gcc 4.7.3)
3. GDB Server

- Starbox Drivers / Digilent ADEPT drivers for USB-JTAG probe:

In the STARBOX, a Digilent USB-JTAG probe permits the communication between GDBServer and emulated chip.

This probe needs Digilent Adept tool to be installed to work.

### Features of the Starchip Flash Chips

|  |  |
| --- | --- |
| **Feature** | **Mizar 913** |
| Chip | SCF136H |
| FLASH size | 136 KB including OTP area |
| Flash organisation  (Physical address range) | 0x2000 – 0x23DFF |
| OTP area | 0x7FA00 – 0x7FBFF |
| Hardware Crc | Yes |
| Random number generator | Yes |
| Page Size | 512 Bytes |

### Hardware Characteristics

The SCF136H chip is full flash 32-bit Harvard RISC architecture microcontroller based on SST super flash technology for smart card with the following characteristics:

1. 32-bit CORTUS APS3cd core with Harvard RISC Architecture
2. CRC-16 Engine , True Random Number Generator
3. 16 bits Timer/Counter.
4. ISO7816 dedicated timer for ETU and Cycle counter.
5. Smart card ISO 7816 Controller.
6. Compliant with ISO 7816-3 T=0 and T=1
7. Class A, B, C supported with Class Indicator
8. Advanced Low power modes.

Support Page Erase and Byte Program (512 bytes per Page). Page Erase is always needed before Byte update.

1. The memory dimensions are:

|  |  |  |
| --- | --- | --- |
| **ROM (Kbytes)** | **RAM (KBytes)** | **Flash (KBytes)** |
| 0 | 3.5 | 136 |

### The electrical characteristics are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Conditions** | **Min** | **Typical** | **Max** |
| Supply Voltage [V] |  | 1.62 |  | 5.5 |
| Output Current[mA] | Run : 20MHz |  | 3.9 | 4.2 |
| Run : Clock divided |  |  | 3.8 |
|  |  |  |  |
| Standby Current[uA] (Clock Stop) | Vcc = 5.0 V |  |  | 80 |
| Vcc = 3.0 V |  |  | 80 |
| Vcc = 1.8 V |  |  | 80 |
| Internal Clk Frequency [MHz] |  | 2.5 |  | 20 |
| External Clk Frequency [MHz] |  | 1 |  | 5 |
| Ambient Temperature [◦C] |  | -25 | 25 | 85 |

### NVM characteristics

**The Flash region starts at address 0x00002000. Its size is 136KB (OTP included).**

The NVM is divided into two parts. The first part is of 136KB of flash which is used for OS & Application. The Second part of from 0x0007FA00 to 0x0007FBFF is OTP (One Time Programmable) area., This region (OTP) is divided into 3 sections. The first 32 bytes (0x0007FA00 to 0x0007FA1F)are reserved by StarChip to store factory information and handle Security mechanisms. The next 16 bytes (0x0007FA20 to 0x0007FA2F) are reserved for product configuration settings. The last 464 bytes (0x0007FA30 to 0x0007FBFF) are available for the software usage. Each byte of this area can only be written one time. Furthermore, it is not possible to erase the OTP sector.It is not possible to execute code from OTP region.

136K Bytes of Flash Memory physically partitioned into pages of 512 bytes. The smallest physical unit that can be erased is one page size, i.e.512 bytes and can be programmed by byte, half-word or word. write is supported. The erase operations change the selected bytes of the page to FF Hex and the write operations modify only those bits in the byte that must be changed from 1 to 0. Again, as it can not be told often enough, if only one bit has to be changed from 0 to 1 the whole page of 512 bytes has to be erased first! For writing into a page fully or partially, it must be erased before proceeding.

The time for erasing a page is 4ms.

The time for writing a byte is 32us.

(**Note:** If the page is in already in erased state, then a ‘single byte’ write takes 32us.)

### Hardware Security Features

The chip provides the following security features:

1. Memory Protected Area defined by Software (“Freeze” area) .
2. Address Bus Scrambling, Data Encryption
3. Frequency and Power Supply monitors.
4. Dedicated Secure Personalization mode
5. Software Authentication capability

• Unique Serial Number per chip

Under abnormal condition or if the access conditions are not fulfilled the card can be configured to reset itself.

### Flash Quality Aspects

Starchip states:

1. Min. 10 years minimum data retention(25 C)
2. Sector endurance of M**i**n **100,000** erase/write cycles per page(25 C)

This means that each page can reliably be written up to 100,000 times with no appreciable degradation in data integrity or device functionality. This number is comparatively low and thus does not allow implementing a proper Roll Back mechanism.

The hardware characteristics (The huge page size of 512 bytes, only byte , half-word or word write supported, page erase supported, page write not supported) cause several severe problems which have to be handled by the OS (see High Update Files and Secure Write mechanism).

### Device Notifications from Starchip

**Page Erase Flow**

Below is a detailed step by step description of this sequence:

1. The software wants to erase the sector ‘S’.

2. The software sets (1) the bit FLACTRL.OPMODE.

3. The software executes a store instruction with a dummy byte, half word or word, at address contained in sector ‘S’.

* The Flash Controller will then ask the Power Manager to freeze CPU execution
* The Flash Controller erases the targeted sector
* The Flash Controller clears internal parameters
* The Flash Controller asks the Power Manager to wake the CPU up to resume application execution

Once the operation is over, FLACTRL register is cleared to its default value.

**Byte Write Flow**

The sequence below shows how to program in Flash Memory (see special case about Programming Freeze region in section 8.2.2 of Starchip datasheet document REF [9]):

1. The software wants to write one byte, one half word or one word in flash memory starting at address <@>.

2. Before executing a CPU ‘store’ instruction, the software must check that the flash area (8, 16 or 32bits) to write to is virgin.

* If the flash area is virgin, the software executes a unique CPU ‘store’ instruction to write the byte, half word or word at address <@>.
* If the flash area is not virgin, the software shall copy the whole content of the Flash sector concerned in a RAM buffer and perform an erasing sequence of the sector. Then the software shall perform a programming sequence of the sector with the RAM buffer content, combined with the data to write, by executing as many CPU ‘store’ instructions as necessary.

3. Once the software has stored a byte, half word or word, the flash programming operation starts automatically.

* The Operation Sequencer indicates to Power Manager that it must invoke a specific power down mode. This freezes CPU and stops code execution.
* Hardware reads the target address (es) (between 1 and 4 addresses) in Flash Memory to check if its (their) content(s) is(are) virgin (FF bytes).
* If one of the new bytes to store overwrites a non FF byte (located in OTP sector or not), the Flash Controller raises a critical error and aborts current operation.
* Else the bytes which must be programmed are copied to Flash. Note that if the software attempts to write a 0xFF byte on an already virgin address, this byte will not be copied to Flash.
* Once the byte(s) has(have) been written in Flash target, the sequence is over. Thus, the Flash Controller clears sector indicators.

The Operation Sequencer indicates Power Manager that the sequence is over. Then, it can wake up the CPU and the product. The execution may resume with the instruction right after ‘store’ instruction.

## ATR

Electronic signals and transmission protocols shall be in accordance with ISO/IEC 7816‑3 unless specified otherwise.

The choice of the transmission protocol(s), to be used to communicate between the SIM and the ME, shall at least include that specified and denoted by T=0 in ISO/IEC 7816‑3.

### Specific ATR values

The card has to react with an Answer To Reset (ATR) after reset or power on sequence.

According to ISO/IEC 7816-3 there are

1. 1 initial character (mandatory)
2. 1 format character (mandatory)
3. x inter­face characters (optional)
4. y historical characters (optional)

The meanings of the initial, the for­mat and the interface characters are specified by ISO/IEC 7816-3. The content of the historical characters is free to the card manufacturer.

**The ATR of Zeta 132 is:**

**a. Zeta V913**

3B BE 94 00 40 14 47 47 33 46 30 **34** 41 5A 45 54 39 31 33 00

**b. StarSIM Zeta V914**

3B BE 94 00 40 14 47 47 33 46 30 **34** 41 5A 45 54 39 31 34 00

**The ATR of Mizar 136 is:**

**a. Mizar V913**

3B BE 94 00 40 14 47 47 33 43 32 31 41 4D 49 5A 39 31 33 00

**b. StarSIM Mizar V914**

3B BE 94 00 40 14 47 47 33 43 32 31 41 4D 49 5A 39 31 34 00

### Boot Loader ATR

**The ATR of the boot loader of Zeta 132 is**

1. For THC20F17BD

3B 1F 96 **2F** 05 03 20 06 17 02 04 10 10 00 07 08 90 00

2. For THC20F17BD-V20

3B 1F 96 **31** 05 03 20 06 17 02 04 20 20 00 07 08 90 00

3. For THC20F17BD-V30

3B 1F 96 **34** 05 03 20 06 17 02 04 30 10 00 07 08 90 00

**The ATR of the boot loader of Mizar 136 is**

3B BA 97 00 40 14 47 42 53 43 46 31 33 36 14 00

### Explanation of OS ATR values

**Atlas/Zeta/Mizar**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Byte No.** | **Name** | **Content (Hex)** | **Meaning** |
| **Constant Part** | **Initial Character** | | | |
| 1 | TS | 3B | direct convention |
| **Format Character** | | | |
| 2 | T0 | BE | high nibble: TA1, TB1 and TD1 are transmitted  low nibble: number of historical characters = 14 |
| **Interface Characters** | | | |
| 3 | TA1 | 94 | F = 512, D = 8 ⇨ divider = 64 (TA1 configurable in NVM) |
| 4 | TB1 | 00 | maximum allowed IPP = 25 mA  VPP is generated internally |
| 5 | TD1 | 40 | TC2 is transmitted for T=0 |
| 6 | TC2 | 14 | WI = 20 ⇨ work waiting time WWT = 960\* WI \* F/f  e.g. WWT 2.00 s for F = 372 (default) and 2.75 s for F = 512, both at 3.5712 MHz |
| **Historical Characters** | | | |
| 7 | T1 | 47 | 'G' = Giesecke & Devrient |
| 8 | T2 | 47 | 'G' = GSM |
| **Mask Specific Part** | 9 | T3 | 33 | '3' = Phase 2+ |
| 10 | T4 | 53  46  43 | 'S' = 'Samsung'  'F' = 'Tongfang'  'C' = 'Starchip' |
| 11... 13 | T5...T7 | 33 45 48  or  33 30 30  or  33 47 48  or  33 49 48  or  30 34 41  or  32 31 41 | '3EH' = chiptype S3FC9AC  or  '3DH' = chiptype S3FC9BC  or  '3GH' = chiptype S3FC9AD  or  ‘3IH’= chiptype S3FC9AE  or  '04A' = chiptype THC20F17BD/V20/ V30  or  '21A' = chiptype SCF136H |
| 14... 16 | T8...T10 | 41 54 4C  or  5A 45 54  or  4D 49 5A | 'ATL' = Atlas  or  'ZET' = Zeta  'MIZ' = Mizar |
| 17... 19 | T11...T13 | 39 31 30  39 31 33  39 31 34 | '910' = version 910(For Atlas)  '913' = version 913(For Zeta/Mizar)  '914' = version 913(For StarSIM V914 Zeta/Mizar) |
| **Variable Part** | 20 | T14 | YY | defined in NVM variable uce2LastATR which is also first byte of EFConf  can be set in initialisation  can be set via EFConf |

1. Note 1: The work waiting time is calculated by the following formula: 960 x D x WI etu. D =1; WI = 14 Hex = 20 Dec => Work waiting time = 960 x 1 x 20 etu = 19200 etu (etu means the bit duration which is 104 µs at 3,57 MHz and clock divider 372). The resulting work waiting time is: 2 s.
2. Nevertheless the mask software itself does not rely on this work waiting time and takes worst case conditions (i.e. work waiting time: 1s) into account.

### Error ATR values

During start-up the mask software executes some basic tests. The card sends one of the following ATRs, if it detects an error (the last ATR byte YY is explained in 3.3.3):

|  |  |
| --- | --- |
| 3B FF 01 00 00 YY Hex | Consistence error in system area found. |
| 3B FF 04 00 0x xx xx YY Hex | NVM error at address 00 0x xx xx Hex found. |
| 3B FF 05 00 0x xx xx YY Hex | Inconsistent file header with identifier HI LO found:   1. This ATR is only possible if the SIM is not able to select the MF or EFCHV during the standard start-up routine for cards in initialised mode. 2. If EFSST or even the whole DFGSM900 / DFDCS1800 do not exist on the SIM, this error ATR will not be sent out as it was done by previous SIMs. |

## Boot loader specifications

### Product family THC20F17BD-XX

[Boot loader specifications]

UM2F04-THC20F17BD-BLUM-EN-US.pdf

UM3104-THC20F17BD-V20-BLUM-EN-US.pdf

UM3404-THC20F17BD-V30-BLUM-EN-US.pdf

### Product family SCFxxxx

[Boot loader specifications]

\StarSIM\_Doc\V914\StarSIM\_V914\20000-Dev\21000-ArchDesign\Bootloader\Software Design Document Mizar Bootloader.doc

## Protocol Type Selection

Due to differences in the description of PTS in ISO/IEC 7816-3 and GSM 11.11 concerning the behaviour in case of errors, the mask software behaves as follows:

1. The card accepts as a maximum four bytes of PTS request (PTSS, PTS0, PTS1, and PCK).
2. If the terminal's PTS request is free of syntax errors, the following happens:

|  |  |  |
| --- | --- | --- |
| PTS request (Hex) | PTS response (Hex) | Card switches to speed enhancement? |
| FF 10 96 79 | FF 10 96 79 | YES (clock divider=16)  (except for Atlas 130K) |
| FF 10 95 7A | FF 10 95 7A | YES (clock divider=32) |
| FF 10 94 7B | FF 10 94 7B | YES (clock divider=64) |
| FF 00 FF | FF 00 FF | NO (clock divider=372) |
| In all other cases of PTS request without syntax error | FF 00 FF | NO (clock divider=372) |

1. In case of syntax error, the card does not respond. SIM reset is required!

## Footprint

For footprint details of StarSIM please refer to foot print details available at following location:

StarSIM\_Doc\StarSIMGeneral\Development\ProductionDoc\StarSIM OS Footprint details.xls

# OTC

## Introduction

The OTC feature is designed and developed in order to dispose code from card after code becomes dead or un-useful.

The code which is expected to be dead or removed is collected together during linking time and is put at a flash location where it can be reclaimed by the ADN file.

There are two such types of code. Create-Delete and MD

In order to achieve the expected linking following changes are done in code and linking. (Also there are some post calculations are done on generated map file to obtain space gain)

## SCFXXXX (Mizar) family

### Basic Idea

Some functionalities of the OS which are needed only at the time of production on the card. In Mizar project there are two such functionalities identified. One is MasterDevice which will be referred as MD and the other one is the creating deleting the files which will be referred as CreDel.

Out of these two functionalities the CreDel is required for creating / deleting the files on the cards in the file system area. These files are required to be created and modified as per the customer’s requirements. Once these files are created then generally the CreDel feature is not needed to be on the card anymore. Hence it can be removed and its space can be used by any other functionality or any other file which will be updated later on. For example ADN file is one of such files which can be created but not updated immediately.

The 90% of the files which are created at the time of personalization are identical for every card on a product line. Only 10% of the files are card specific. That means 90% of the personalization process is same for every card. So if we execute the complete personalization process on a single card and read its contents back, then those contents can be used to flash multiple cards. This will save 90% time of the personalization on every single card. The MD functionality is the one which can be used for reading the content of a personalized card. The MD functionality is also a one time utility which is never needed to be on the card after the production process of the card is finished.

Now both the functionalities which we explained above are mutually exclusive. That means MD feature is never needed to be on the card when the CreDel feature is being used and vice versa. This gives us another opportunity of flash memory saving. We can download the MD feature on the card at the same location where the CreDel feature was residing.

### CreDel

Some functionalities of the OS which are needed only at the time of production on the card. In Mizar project there are two such functionalities identified. One is MasterDevice which will be referred as MD and the other one is the creating deleting the files which will be referred as CreDel.

Out of these two functionalities the CreDel is required for creating / deleting the files on the cards in the file system area. These files are required to be created and modified as per the customer’s requirements. Once these files are created then generally the CreDel feature is not needed to be on the card anymore. Hence it can be removed and its space can be used by any other functionality or any other file which will be updated later on. For example ADN file is one of such files which can be created but not updated immediately.

### CreDel feature – possible combinations

The CreDel feature as explained earlier can be a one time code. However in some cases this feature is required to be on the card for the end customer as well. Due to this constraint the CreDel can or can not be a reclaimable feature. Hence in the code there is a switch implemented which decides if the CreDel feature has to be persistent on the card as an integral part of OS or it can be a reclaimable feature whose flash area can be reused for some other purpose (like MD or EFadn). This flag is a precompiler macro switch named CREDEL\_RECLAIMABLE and it is placed in the every individual initialization file of every target.

##### **CREDEL Persistent**

This type of ini can be generated by setting the precompiler switch CREDEL\_RECLAIMABLE to FALSE. Due to this the CreDel functionality will become an integral part of the OS. It wont be reclaimed by the EFadn file neither it can be overwritten by the MD code

##### **CREDEL Volatile**

This type of ini can be generated by setting the precompiler switch CREDEL\_RECLAIMABLE to TRUE. Due to this the CreDel functionality will be located just below the bootloader area. Hence CreDel wont be an integral part of OS, even if the OS can utilize the CreDel feature. Due to this the EFadn file can reclaim the flash area occupied by the CreDel functionality. Also the MD code can be downloaded at the same location where CreDel was present.

##### **CREDEL Volatile – Miscellaneous**

**Free Memory**

The CREDEL area will not be included in calculation of free memory, i.e. the free memory indicated by SELECT etc. will always be the value of free memory in the file system area only.

**RFM**

After removing CREDEL, the commands CREATE FILE, CREATE SPECIAL and DELETE FILE will also be blocked for usage via RFM.

**DELETE FILE**

In initialisations with volatile CREDEL the command DELETE FILE will not be supported, i.e. the card answers with SW = 6D 00. But if EFADN was not created as first EF (i.e. it is NOT created in the CREDEL area but in the normal file system area), the command DELETE FILE is enabled again. The CREDEL feature is lost on such an initialisation, i.e. EFADN can not be created in CREDEL afterwards again.

**READ RECORD**

After EFADN has been created, the READ RECORD command will be blocked (SW 9404) if used for reading this file. After an update to EFADN has occurred or EFConf has been updated, READ RECORD will work again as usual.

**Access Conditions**

Removing CREDEL and initialising the body of EFADN does not care about access conditions for EFADN, i.e. the body will always be initialised.

EFADN **File Type**

In personalisation EFADN always has to be created as linear fixed file, the OS does check current DF (=DFTelecom) and the file ID only, it does not care about the file type in CREATE FILE.

Creating EFADN with another file type might cause erroneous behaviour. Also, the “UpdateLate” property of EFADN in personalisation must be set to true. When false, it can lead to unexpected behaviour.

### MD

The 90% of the files which are created at the time of personalization are identical for every card on a product line. Only 10% of the files are card specific. That means 90% of the personalization process is same for every card. So if we execute the complete personalization process on a single card and read its contents back, then those contents can be used to flash multiple cards. This will save 90% time of the personalization on every single card. The MD functionality is the one which can be used for reading the content of a personalized card. The MD functionality is also a one time utility which is never needed to be on the card after the production process of the card is finished.

Now both the functionalities which we explained above are mutually exclusive. That means MD feature is never needed to be on the card when the CreDel feature is being used and vice versa. This gives us another opportunity of flash memory saving. We can download the MD feature on the card at the same location where the CreDel feature was residing

The MD feature as explained earlier is for reading the contents of the Master card which is personalized. The cards are flashed using a proprietary file format of G&D which is known as BLC format. There are three types of Inis in the form of BLC file. These are

MasterIni – the compiled BLC file which is used to flash the MasterCard

MaxiIni – the BLC file which is generated as an output of reading the contents of the Master card.

MiniIni – The compiled BLC file which does not support the MD functionality

As it was explained earlier if the CreDel is reclaimable (volatile) the MD functionality utilizes the flash area where the CreDel was placed. This is done by downloading the MD patch at the same location where the CreDel was present. This is done by enabling the download feature of the OS. The OS downloads the MD patch by using the functions of the bootloader.

The size of the MD patch is and has to be smaller than the CreDel code. The MD code size is of 0x400 (1024) bytes.

If it is required to disable the MD feature completely, then one can do it by setting the ALLOW\_DOWNLOAD flag to FALSE. Due to this actually the download feature of the OS will be disabled. Hence the MD patch can never be downloaded to the OS. Effectively the MD feature will not be available for usage as well as for reclaim.

Apart from this, there is one more precompiler switch named MD\_RECLAIMABLE which controls compilation of the functions under MD code. However this switch is automatically handled by the build system for specific inis.

### Possible combinations with MD and CreDel

Due to two reclaimable features as MD and CreDel, there are multiple combinations of init are possible. The third parameter which adds to these combinations is the type of init viz DelIni or NonDelIni.

Following are the possible combinations of the init.

Please note the sizes of the reclaimable functionalities as

Bootlaoder = 0xA00,

MD = 0x400 (It should be considered as 0 if the CreDel is reclaimable)

CreDel = 0xC00

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ALLOW\_DOWNLOAD | CREDEL\_RECLAIMABLE | INI\_VERSION  (DelIni or NoneDelIni) | File System End | Reclaim area size | Reclaimable functionalities | Download and use MD possible | CreDel volatile | DelIni |
| TRUE | TRUE | 0 | 0x22800 | 0xC00 | MD, CreDel | Yes | Yes | Yes |
| TRUE | TRUE | 1 or 2 | 0x22800 | 0x1600 | MD, CreDel, Bootloader | Yes | Yes | No |
| TRUE | FALSE | 0 | 0x23000 | 0x400 | MD | Yes | No | Yes |
| TRUE | FALSE | 1 or 2 | 0x23000 | 0xE00 | MD, Bootloader | Yes | No | No |
| FALSE | TRUE | 0 | 0x22800 | 0xC00 | CreDel | No | Yes | Yes |
| FALSE | TRUE | 1 or 2 | 0x22800 | 0x1600 | CreDel, Bootloader | No | Yes | No |
| FALSE | FALSE | 0 | 0x23400 | 0x00 | None | No | No | Yes |
| FALSE | FALSE | 1 or 2 | 0x23400 | 0xA00 | Bootloader | No | No | No |

### Error message related to file system size and reclaim area

Apart from the above mentioned pre-compiler macros and switches, there is another macro which defines the size of the file system. This macro is FILESYSTEM\_SIZE. Before compilation one has to enter appropriate size of the file system depending upon the customer requirement and the actual available space. If in case the value of FILESYSTEM\_SIZE is smaller than actual available limit, then there is no problem. The build system will automatically attach the file system to the reclaim area so that the reclaim process will be possible. However if the value of the FILESYSTEM\_SIZE is more than the actual available size, then an error message will be shown by the build system.

### EFADN <= Reclaimable size

If the size of the EFADN body does not exceed the size of the reclaimable area, the CREATE FILE or CREATE SPECIAL command will align the file body to the end of the reclaimable area but the body will not be initialised, i.e. it will still contain the corresponding code, and the corresponding commands are still available.

In this case EFADN can always be stored in the reclaimable area regardless of the order of creation, and thus there's no need to adapt the production process. This also means for EFADN the free file system size will only be diminished by the EF header size because the file body is completely taken from the CREDEL area.

### EFADN > Reclaimable size

If EFADN is larger than the reclaimable area then it is not longer possible to store the complete file body in this area. But the reclaimable area still can be utilised for reclaim, provided EFADN is the first file having a body: The file body of EFADN will again be aligned to the end of the reclaim area by the commands CREATE FILE and CREATE SPECIAL and it will span the complete reclaimable area plus the end of the file system area. Therefore there cannot be another file body created before EFADN, i.e. profiles have to be adapted in that way that EFADN creates the first file body (only DFs without own body can be created before).

In this case also, the body of EF is not initialised during creation, and therefore the commands CREATE FILE and CREATE SPECIAL are still available for personalisation.

Note: If EFADN is larger then reclaimable size but another file body has been created before, the body of EFADN will be allocated in the regular file system area. CREDEL will not be deleted from memory as described earlier.

### When the reclaimable functionalities gets blocked

For below given table, consider that the MD and CreDel both are reclaimable.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Action | Init | MD download feature | CreDel | DelIni  command |
| EFadn created in reclaim area but not updated or deleted | DelIni | Unblocked | Unblocked | Unblocked |
| NonDelIni | Unblocked | Unblocked | Blocked |
| EFadn created in reclaim area and then updated or deleted | DelIni | Blocked | Blocked | Unblocked |
| NonDelIni | Blocked | Blocked | Blocked |
| EFadn created outside the reclaim area and then updated or deleted | DelIni | Unblocked | Unblocked | Unblocked |
| NonDelIni | Unblocked | Unblocked | Blocked |
| EFconf updated with final value (0x30) | DelIni | Unblocked | Blocked | Unblocked |
| NonDelIni | Unblocked | Blocked | Blocked |

## THC20F17BD-XX (Zeta) family

### Basic Idea

The OTC feature is designed and developed in order to dispose code from card after code becomes dead or un-useful.

The code which is expected to be dead or removed is collected together during linking time and is put at a flash location where it can be reclaimed by the ADN file.

There are two such types of code. Create-Delete and MD.

In order to achieve the expected linking following changes are done in code and linking. (Also there are some post calculations are done on generated map file to obtain space gain).

.

### Memory Layout

The following figure depicts the overall memory layout of Zeta:

**OS Code**

**Tables etc**

**.**

**CREDEL Code**

**\_**

**\_**

**Internal Mgmt.**

**Area**

OS

Ini

**Libs**

**Interrupts**

**File System area**

**EF CONF**

**EF CHV**

The CREDEL code is linked immediately behind the file system area of the initialisation, and therefore it lies within 64 KB area that can managed by the file system. Additionally, this layout allows using the CREDEL area very flexibly as will be explained below. The initialisation will contain a flag that defines if CREDEL shall be persistent or volatile

### CreDel

### CREDEL Persistent - Creating and Deleting File

If the initialisation is configured to provide CREDEL persistently, it behaves conventionally, i.e. all files are created in the file system area - file headers at the beginning and file bodies at the end of the area. The CREDEL code is kept in memory and therefore CREATE FILE, CREATE SPECIAL and DELETE FILE can be used after personalisation, directly via APDU as well as via RFM.



The figure shows the memory layout after creation of three files:

* DFTelecom (without own body)
* an arbitrary EF
* EFADN

### CREDEL Volatile - Creating Files

In an initialisation configured for volatile CREDEL EFADN will be managed in a specific way and this will be two ways to reuse the CREDEL memory area:

**Case 1: EFADN <= CREDEL**

If the size of the EFADN body does not exceed the size of the CREDEL memory area, the CREATE FILE or CREATE SPECIAL command will align the file body to the end of the CREDEL area but the body will not be initialised, i.e. it will still contain CREDEL code, and the commands CREATE FILE, CREATE SPECIAL and DELETE FILE are still available.

The following figure depicts the memory layout after creating three files:

* DFTelecom (without own body)
* an arbitrary EF
* EFADN



In this case EFADN can always be stored in the CREDEL area regardless of the order of creation, and thus there's no need to adapt the production process. (Note: The production process has to be adapted, however, if the file shall be written during personalisation, see 4.3.3.3.). This also means for EFADN the free file system size will only be diminished by the EF header size because the file body is completely taken from the CREDEL area.

Note: Until end of development the size of the CREDEL area could vary. All tests must be flexible enough to adapt to changes in CREDEL size quickly.

The valid size of CREDEL will always be published here in this document.

**SIZE OF CREDEL: 0x1200 Bytes**

**Case 2: EFADN > CREDEL**

If EFADN is larger than the memory area occupied by CREDEL, it is not longer possible to store the complete file body in this area. But the CREDEL area still can be utilised, provided EFADN is the first file having a body: The file body of EFADN will again be aligned to the end of the CREDEL area by the commands CREATE FILE and CREATE SPECIAL and it will span the complete CREDEL area plus the end of the file system area. Therefore there cannot be another file body created before EFADN, i.e. profiles have to be adapted in that way that EFADN creates the first file body (only DFs without own body can be created before).



The figure shows the memory layout after creation of

* DFTelecom (without own body)
* EFADN (first file with body)
* an arbitrary EF

As in case 1, the body of EF is not initialised during creation, and therefore the commands CREATE FILE and CREATE SPECIAL are still available for personalisation.

Note: If EFADN is larger then CREDEL but another file body has been created before, the body of EFADN will be allocated in the regular file system area (same as with CREDEL persistent, see [4.3.3.1](#_CREDEL_Persistent_-)). CREDEL will not be deleted from memory as described in 4.3.3.3 in this case!

### CREDEL Volatile - Removing the Code

When personalisation is completed, the CREDEL code will be removed from memory, i.e.

* the body of EFADN will be initialised as empty linear fixed file
* the commands CREATE, CREATE SPECIAL and DELETE will be blocked, the APDUs will return SW = 6D 00

The body will be initialized with the fixed value 0xFF.

There will be two triggers that cause removal of CREDEL:

* In the established production process byte 1 of EFConf is used to indicate the progress of production, and this byte is sent as last byte of the ATR. The byte is set to 3FH at the beginning of personalisation and to 30H at the end. Thus, updating byte 1 in EFConf to 30H indicates reliably that file creation is completed and can be used as trigger for removing CREDEL.
* It might be required to write customer data to EFADN during personalisation (i.e. before updating EFConf), and this will destroy the CREDEL code. Updating EFADN is possible only at the end of personalisation, i.e. after creation of all files but before updating EFConf. Therefore, UPDATE RECORD on EFADN will also trigger removing CREDEL, and the production process has to be adapted accordingly. Of course, an update to EFADN afterwards will disable the trigger by an update to EF Conf.

### CREDEL Volatile – Miscellaneous –

**Free Memory**

The CREDEL area will not be included in calculation of free memory, i.e. the free memory indicated by SELECT etc. will always be the value of free memory in the file system area only.

**RFM**

After removing CREDEL, the commands CREATE FILE, CREATE SPECIAL and DELETE FILE will also be blocked for usage via RFM.

**DELETE FILE**

In initialisations with volatile CREDEL the command DELETE FILE will not be supported, i.e. the card answers with SW = 6D 00. But if EFADN was not created as first EF (i.e. it is NOT created in the CREDEL area but in the normal file system area), the command DELETE FILE is enabled again. The CREDEL feature is lost on such an initialisation, i.e. EFADN can not be created in CREDEL afterwards again.

**READ RECORD**

After EFADN has been created, the READ RECORD command will be blocked (SW 9404) if used for reading this file. After an update to EFADN has occurred or EFConf has been updated, READ RECORD will work again as usual.

**Access Conditions**

Removing CREDEL and initialising the body of EFADN does not care about access conditions for EFADN, i.e. the body will always be initialised.

EFADN **File Type**

In personalisation EFADN always has to be created as linear fixed file, the OS does check current DF (=DFTelecom) and the file ID only, it does not care about the file type in CREATE FILE.

Creating EFADN with another file type might cause erroneous behaviour. Also, the “UpdateLate” property of EFADN in personalisation must be set to true. When false, it can lead to unexpected behaviour.

### MD –

### Basic Idea

The MD feature is introduced in order to reduce card production time.

The init with this feature enabled is called Master Init. The master init has capability of taking the dump of the card image to create a new init called Maxi Init. After creation of a Maxi Init, the Master Init loses the MD functionality. The Maxi Init does not have the MD capability.

If the MD feature is disabled in an init, it is referred as Mini Init.

The master init provides MD commands using which the whole contents of the Flash can be read. For details, please refer Zeta Production Concept Document.

Bootloader of Zeta 132K (THC20F17BD) resides in Bank 3 (4KB area).

In Bootloader area, 1KB space is not used by Bootloader which is used by MD.

Since MD code is required for dumping the maxi init, this area cannot be used for creating files during personalization. But after the maxi init dump is taken this area is usable.

The idea of this concept is making use of this area during personalization which happens before the maxi init is dumped. The concept is similar to the CREDEL feature explained in section above.

In principle there are several approaches: The MD/Bootloader memory could be used for

* file bodies
* data buffers
* program code

Loading program code after personalisation would require introducing an additional initialisation step in production and most data buffers are too small to use the memory space efficiently. Therefore, usage for file bodies is the most promising option.

If the MD code shall be replaced by a file body, then that means, this file body must not be initialized (filled with 0xFF) or written to, or should not be deleted even though it is neither initialized nor updated.. It should be initialized or updated or deleted only after the MD functionality has been used. Otherwise the MD functionality will be erased alongwith the Bootloader area. This could be achieved by compiling an initialisation that contains the body of a large file (e.g. EFSMS or EFADN) mapped to the MD memory area. But this would mean that the size of this particular file is already defined by the initialisation and cannot be set during personalisation. Therefore a more flexible solution will be implemented by Zeta, which allows reusing the MD/BL memory for EFADN during personalisation, i.e. the file can be created during personalisation as usual.

Note 1: On a DELINI init, if MD feature is enabled, then ADN\_IN\_MD\_INIT flag should be disabled. That means on DELINI, MD and Boot Loader space cannot be reclaimed by EFADN file.

Note 2: On a NON-DELINI init, if MD feature is enabled, then ADN\_IN\_MD\_INIT flag can be enabled. MD and Boot Loader space can be reclaimed by EFADN file.

### Memory Layout –

The following figure depicts the overall memory layout of Zeta:

**OS Code**

**Tables etc**

**.**

**\_**

**\_**

**File System** \_THC20F17BD elocate n is not valid for Zeta 112K, please do not change this or do not map any NVM data in this regio

**Area**

OS

Ini

**Libs**

**Interrupts**

MD Area (~1K)

(Bank 3)

BL (~2.2K)

Bank 3

As the MD/BL area is placed in Bank 3, the file system area should be stretched till the start of MD/BL area.

The initialisation will contain a flag that defines if MD/BL shall be persistent or volatile

### MD/BL - Creating Files

Unlike CreDel feature, the MD feature is always volatile. That means it can not be provided for the use of the end user.

In an initialisation configured for volatile MD/BL, EFADN will be managed in a specific way and there will be two ways to reuse the MD + BL memory area.

**Case 1: EFADN <= MD/BL area**

If the size of the EFADN body does not exceed the size of the MD/BL memory area, the CREATE FILE or CREATE SPECIAL command will align the file body to the end of the MD/BL area but the body will not be initialised, i.e. it will still contain MD/BL code.

In this case EFADN can always be stored in the MD/BL area regardless of the order of creation, and thus there's no need to adapt the production process. (Note: The production process has to be adapted, however, if the file shall be written during personalisation). This also means for EFADN the free file system size will only be diminished by the EF header size because the file body is completely taken from the MD/BL area.

The valid size of MD/BL will always be published here in this document

**SIZE OF MD + Boot Loader Area: 0x1000 Bytes**

**Case 2: EFADN > MD/BL Size**

If EFADN is larger than the memory area occupied by MD/BL, it is not longer possible to store the complete file body in this area. But the MD/BL area still can be utilised, provided EFADN is the first file having a body: The file body of EFADN will again be aligned to the end of the MD/BL area by the commands CREATE FILE and CREATE SPECIAL and it will span the complete MD/BL area plus the end of the file system area. Therefore there cannot be another file body created before EFADN, i.e. profiles have to be adapted in that way that EFADN creates the first file body (only DFs without own body can be created before).

As in case 1, the body of EF is not initialised during creation, and therefore the MD commands are still available for taking maxi init dump.

The body will be initialized with the fixed value 0xFF.

Note: If EFADN is larger then MD/BL area but another file body has been created before, the body of EFADN will be allocated in the regular file system area (same as with MD/BL persistent, see [10.3](#_MD/BL_Persistent_-)). Note that for above, it needs to be configured that the EFADN may not be first file created during personalization. By default this option is not present in the mask.

### MD/BL Volatile - Removing the Code –

The MD code will be removed in following scenarios:

* + - * **Master Init:**

1. After the maxi init dump is complete, MD code is erased from master init.
2. Before taking Maxi init dump, if EFADN updated: The MD code + Boot Loader area is erased and record is updated at EFADN file which now occupies the MD Code + Boot Loader area.
3. Before taking Maxi init dump, if EFADN deleted: The MD code + Boot Loader area is erased.

**Warning: It is not expected to update or delete** EFADN **on master init during personalization. MD functionality will be lost and the card will be blocked.**

* + - * **Maxi Init:**

1. Since the MD functionality does not take dump of the MD/BL area, the MD code is not present in the Maxi init.

The production process has to be adapted keeping above things in mind.

### Miscellaneous

**Free Memory**

The MD/BL area will not be included in calculation of free memory, i.e. the free memory indicated by SELECT etc. will always be the value of free memory in the file system area only.

**READ RECORD**

After EFADN has been created, the READ RECORD command will be blocked (SW 9404) if used for reading this file. After an update to EFADN has occurred, READ RECORD will work again as usual.

**Access Conditions**

Removing MD and initialising the body of EFADN does not care about access conditions for EFADN, i.e. the body will always be initialised.

EFADN **File Type**

In personalisation EFADN always has to be created as linear fixed file, the OS does check current DF (=DFTelecom) and the file ID only, it does not care about the file type in CREATE FILE.

Creating EFADN with another file type might cause erroneous behaviour. Also, the “UpdateLate” property of EFADN in personalisation must be set to true. When false, it can lead to unexpected behaviour.

### MD/BL Space reclaiming – Note for Production Process –

The production process needs to ensure below steps.

1. If EFADN has size greater than MD/BL size, then it must be the first file to be created in personalization on Master Init.
2. EFADN should never be updated during personalization on Master Init.
3. EF ADN should never be deleted during personalization on Master Init/Maxi Init
4. After personalization, Maxi Init dump should be taken using MD commands.
5. On maxi init, EFADN must be the first file to be updated.

Note: If total personalization is done on Maxi init, EFADN should be the last file to be updated in personalization.

## Switches description

### MASTER\_DEVICE\_ENABLED –

This switch controls if the MD feature is needed to be in the init or not. Making it true will enable the MD feature and it will become a part of the init.

### ADN\_IN\_MD\_INIT –

This switch controls if the MD should be a reclaimable module or not. In either cases this module is one time usable. Hence it is always recomonded to set this swtich to TRUE if MD functionality is to be used.

The size of the MD feature is 1KB on both the families.

### ADN\_IN\_CREDEL\_INIT –

This switch is used to compile the CreDel feature as a reclaimable module or not. Setting this to TRUE makes the CreDel functionality to act as a reclaimable module. Otherwise the CreDel feature becomes an integral part of the init and it can not be removed.

The size of the CreDel feature is 3K in SCFXXXX family and it is 4K in THC20F17BD-XX family.

### ADN handling –

While handling the reclaim feature, there are certain points related to ADN file which needs to be taken care. These points are as listed below.

* ADN must be created in reclaim area. To achieve this, either ADN file should be created as first EF file during the personalization or the size of the ADN file should be lesser than the reclaim area size.
* The ADN file should not be updated unless the entire reclaim features are fully utilized and are not needed any more. For example, the MD feature is utilized on the Master card and a dump is created using it. Or in case of the CreDel feature, all the required files are either created or deleted as needed.
* The ADN file must be updated on the card before declaring it as ready for the end customer.
* The ADN file should not be deleted if it is created in the reclaim area. If CreDel feature is reclaimable, then one can not delete the ADN file which is created in the reclaim area. On the other hand if only MD feature is reclaimable, then ADN file should not be deleted if it is created in the reclaim area. Otherwise the MD feature will be deleted.
* The EFconf file should not be updated with final value, unless all the reclaim features are utilized and are not needed anymore.

## Profiler flags related to MD –

With introduction of the MD feature, there are certain restrictions on the creation and updating sequence of some specific files. Following are the flags which are introduced in profiler, for the MasterDevice concept.

* CreateLateForMd,
* UpdateMandatoryInMasterDevice
* UpdateLate.

Please refer following for detail and latest explanation of these flags

* [UPPC user manual](http://gdm-pscs/opencms/export/sites/Docu/ChipApplications/GSM/Java/UserManual_UPPC_PPGDTelcoTIP.html#Sect3_ConfigFilesystem)

Path : <http://gdm-pscs/opencms/export/sites/Docu/ChipApplications/GSM/Java/UserManual_UPPC_PPGDTelcoTIP.html#Sect3_ConfigFilesystem>

* http://telcowiki.intern/tw/index.php/Profile\_Attributes\_%28StarSIM%29

Following are the example files which have some restriction on their creation or update sequence.

EFCHV, EFCONF, EFKI, EFPROD, EFADN, EFIMSI, and EFLOCI.

Below table indicates the flag value for each file if the MasterDevice concept is to be used.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Files** | **Parent DF** | **CreateLateForMd** | **UpdateMandatoryInMasterDevice** | **UpdateLate** |
| EFCHV | MF | FALSE | TRUE | TRUE |
| EFCONF | MF | FALSE | TRUE | TRUE |
| EFKI | GSM | FALSE | TRUE | FALSE |
| EFPROD | MF | FALSE | TRUE | FALSE |
| EFADN | TELE | FALSE | FALSE | TRUE |
| EFIMSI | GSM | TRUE | FALSE | FALSE |
| EFLOCI | GSM | TRUE\*1 | FALSE | FALSE |

\*1 – The CreateLateForMd flag for EFLOCIfile should be set to TRUE only when BDN (service 31) **OR** FDN (service 3) allocated and activated in EF\_SST

## OTC Commands –

### SCFXXXX family (Mizar) –

For this family the OTC commands are mainly for the MD feature. These commands are as given below.

### Enable Download

This command is used to access the download functionality of the OS. Successful authentication using this command will enable the Erase sector, Initialize and Verify commands which are explained above.

APDU command format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Command | Class | Ins | P1 | P2 | P3 |
| ENABLE DOWNLOAD | 0x90 | 0xB0 | 0xFE | 0xEE | 0x08 |

Additional required data

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bytes |  | | | | | | | |
| 1…8 | 0x29 | 0x34 | 0x95 | 0x55 | 0x78 | 0x98 | 0x83 | 0x18 |

Possible responses:

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Description |
| 0x90 | 0x00 | No error. The authentication for the erase and initialize command is successful |
| 0x6F | 0x00 | Error in authentication. |

### Read Flash

This will be an optional command which will serve the purpose of the MasterDevice functionality. The MD functionality will act as a pluggable module in case of Mizar. That means the OS may or may not have this command all the time. If user needs to have this command, then he has to flash a small part of the OS which will have this command.

APDU command format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Command | Class | Ins | P1 | P2 | P3 |
| READ FLASH  (to know the length) | 0x90 | 0xB0 | 0xFE | 0x00 | 0x00 |
| READ FLASH  (to know the length) | 0x90 | 0xB0 | 0xFE | 0x00 | XX |

The Read Flash command will be used in two steps. In step one the Read flash command will be sent to the card with the parameter P3 as 0x00. This is to understand the number of data bytes that the card can send. In response to this instruction, the card will send the number of data bytes as 6C XX. Where XX represents the number of data bytes the card can send. Now the card reader should send the Read Flash command again. This time the parameter P3 should be the same what the card responded last time (XX). In response, the card will send the actual data bytes this time.

The data response from the card has following data format. The two most significant bytes represent the ASCII value of “S3”. The byte after that represents the number of bytes left in the record. In other word this length represents the number of bytes left in the record in the MOT file. The four bytes after that represents the address of the flash whose contents are being read back. The bytes one lesser than the remaining, represent the actual flash contents. The last (Least Significant byte) represents the CRC calculated over the record which is as per the MOT file format. It is calculated over the bytes from Length till last byte of flash data.

Note that the card status will change from MasterIni (0xFD) to MaxiIni (0xFF) only after first read command in which data is present. After that if the card is reset by mistake, then neither taking the remaining dump nor downloading the MD patch will be possible

Response Data:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| APDU | | | | | Address | | | | Data | | | |
| 0x00 | 0x58 | 0x00 | 0x00 | 0x08 | 0x00 | 0x00 | 0x20 | 0x00 | 0x00 | 0x01 | 0x02 | 0x03 |

Possible responses:

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Description |
| 0x90 | 0x00 | OK to send next request for the data. (response to actual data read) |
| 0x92 | 0x00 | End of the flash. No more data to fetch. (response to actual data read) |
| 0x6F | 0x00 | Error in reading process (response to actual data read OR response to know the length) |
| 0x6C | XX | Number of bytes available for sending. (response to know the data length) |

### THC20F17BD-XX (Zeta) –

For this family the OTC commands are mainly for the MD feature. One of the command is implemented as an interlock to prevent the personalization of a Master card with old perso tool.

### MD\_ENABLE\_PERSO

This command will execute only on the master card. This will enable the file creation process which is by-default disabled on the master card. This command should be present only in the new personalization tool. During the personalization process, if any file creation command is sent before sending this command, then the file creation will fail. This will ultimately result in to failure of personalization process. Thus file creation is the only command which is not allowed during the MD process unless MD\_ENABLE\_PERSO is given. Otherwise all other GSM commands are allowed even during the MD process. Note that by default the file creation process will be allowed on all other types of INIs except MASTER\_INI. Hence for them there is no need to give the MD\_ENABLED\_PERSO command before the file creation.

**APDU command format**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | P3 |
| 90 | B2 | FE | FC | 00 |

**Possible responses:**

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Description |
| 6E | 00 | Master functionality is absent. |
| 6D | 00 | Master functionality is absent. |
| 6A | 88 | Master functionality is present and personalization is enabled now. |

### MD\_ACTIVATE

The access to MD feature is controlled by this command. Only after providing this command correctly, one can use the MD feature to read the contents of the card.

If the card is reset (both cold as well as warm reset) before the completion of the dump process, then the MasterDevice functionality is disabled from the card. This means, such card will not act as a Master card anymore and hence it won’t recognize any MD related commands. Please note that entering the wrong AV key even for a single time will cause the disabling and erasing of MD functionality.

**APDU command format**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | P3 | Data |
| 90 | B0 | FE | EE | 08 | AV (Authentication Value) for activation Master Device functionality |

**LC = ‘08’**: AV for Activation Master Device functionality

P1/P2: ‘FE’ ‘EE’ dump all, code + program data

**Possible responses**:

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Description |
| 6E | 00 | Master functionality is absent. |
| 6D | 00 | Master functionality is absent |
| 6A | 88 | Master functionality is present and activated now. |

**MD\_ACTIVATE Authentication Value (AV)**

For using the Master Device functionality, it’s necessary to authenticate the user to the card. This is done in the command *MD\_ACTIVATE.*

Currently static value of AV is used. [0x29,0x34,0x95,0x55,0x78,0x98,0x83,0x18]

### MD\_READDATA

The command *MD\_READDATA* is only allowed, if

* it is sent directly after the command *MD\_ACTIVATE* or after a preceding *MD\_READDATA*
* the authentication was successful
* there is still NVM left for dumping (indicated by '90 xx' response to the preceding M*D\_READDATA*)

**Note: Command only available directly after MD\_ACTIVATE or a preceding MD\_READDATA**

The first MD\_READDATA command should have an expected length of 0xF3.

The command MD\_READDATA serves to fetch the NVM MaxiInit initialisation file data from the card.

The positive response to MD\_READDATA will always be either S2 record or S5 record type. The S2 record will represent the MOT file contents

Whereas the S5 record will have the Bank CRC information and the bank erase information

**APDU command format**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLA | INS | P1 | P2 | P3 |
| 90 | B1 | FE | 00 | XX |

**Possible responses**:

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Description |
| 90 | XX | (data + 90 xx): OK, more XX data follows (where XX can be from 0x01 to 0xFC).  Note: The maximum limit is 0xFC. This also includes the three bytes: two ASCII characters “S2” and the CRC byte.  The maximum record length returned in current implementation is 0xF3 (0xF0 Data bytes + 0x03 Bytes as above). |
| 92 | 00 | NVM Dump is completed. |
| 69 | 83 | Master Device is not active. (Reading before activation) |
| 6D | 00 | Master functionality is absent. |
| 6E | 00 | Master functionality is absent. |

**Response Data**:

For StarSIM V913, Response data is not encrypted. For StarSIM V914, Response data will be encrypted.

Response data is sent in S2 or S5 Record format.

# Installation of required tools

While most of the tools required for building StarSIM are checked into version control, some installations and configurations must be done manually.

## Rational ClearCase UCM

A typical ClearCase installation is sufficient, as UCM is part of the standard distribution.

Make sure that your local kmdata\ folder is shared and read/modify/full access is given to your login and "clearcase".

## Visual Studio C++ 2005 (Version 8)

This is used for Telib developer tests.

## Perl

Perl is used by the build process and many other utilities needed for StarSIM development. Install version 5.8.x.

Further, it is recommended that you add the perl\bin directory to your global PATH environment variable, e. g.

PATH=%PATH%;c:\perl\bin

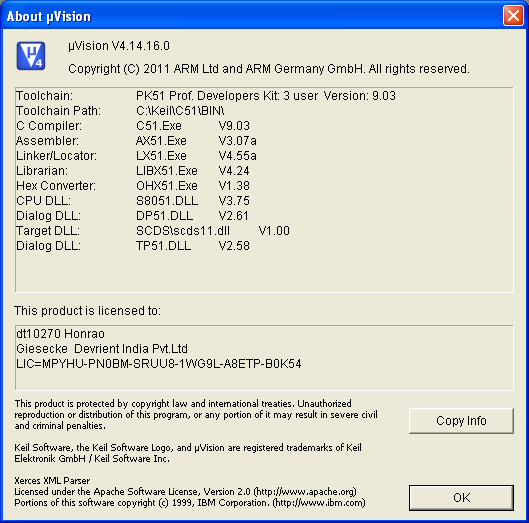
## Tools required only for THC20F17BD-XX family

### Keil Toolchain for THC20F17BD-XX

The following tools were used to generate the FLASH Mask

**µVision: Version V4.14.16.0**

**Copyright (C) 2011 ARM Ltd and ARM Germany GmbH**



The new tool chain is used in the development of the mask. The development work is done on TMC Emulators.

## Tools required only for SCFXXXX family

### Cortus Toolchain for Starchip

Even though the command-line tools for building StarSIM are checked into the VOB, it is necessary to install the complete toolchain for target debugging.

**1. Cortus Toolchain:**

For this you need to install “cortus-aps3-ide-setup-YYYYMMDD.exe”.

Latest is cortus-aps-ide-setup-20130801-release.exe

It will install following tool on your PC,

1. Eclipse IDE.
2. GCC (gcc4.7.3).
3. GDB Server.

**2. Starbox Drivers / Digilent ADEPT drivers for USB-JTAG probe**:

In the STARBOX, a Digilent USB-JTAG probe permits the communication between GDBServer and emulated chip.

This probe needs Digilent Adept tool to be installed to work. You can get it from StarChip FTP server, StarChip Support team or from: http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT

The drivers are available in directory: “tools/starbox\_drivers”.

**4. Starbox programming tool:**

Xilinx public tool (named Impact) which is used to change the bitfile inside the STARBOX and change the product configuration.

This tool is available in directory: “tools\starbox\_programming\_tool”.

Define a global environment variable called STARCHIP\_TOOLS and set its value to the cortus-ide folder path, e. g.

**STARCHIP\_TOOLS** **= C:\cortus-ide**

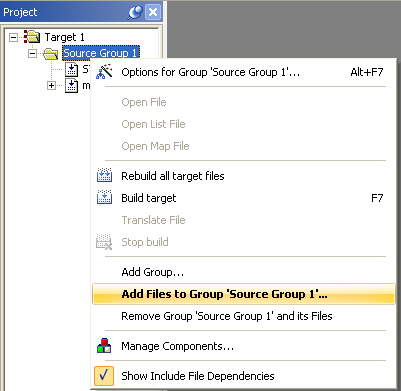
***Note:***

* + 1. ***Get write permissions from System Administrator for the installation folder.***
    2. ***This toolchain requires cygwin.dll, so if there is any previous installation of cygwin, then tool-chain might fail during compilation. “Please remove cygwin path from environmentaion PATH variable OR add “C:\cortus-ide\cygwin\bin” path to environmental PATH variable.***

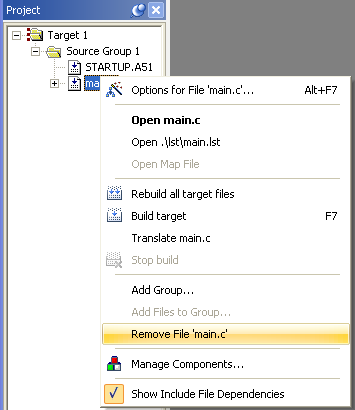
# Build Process

## Product family THC20F17BD-XX

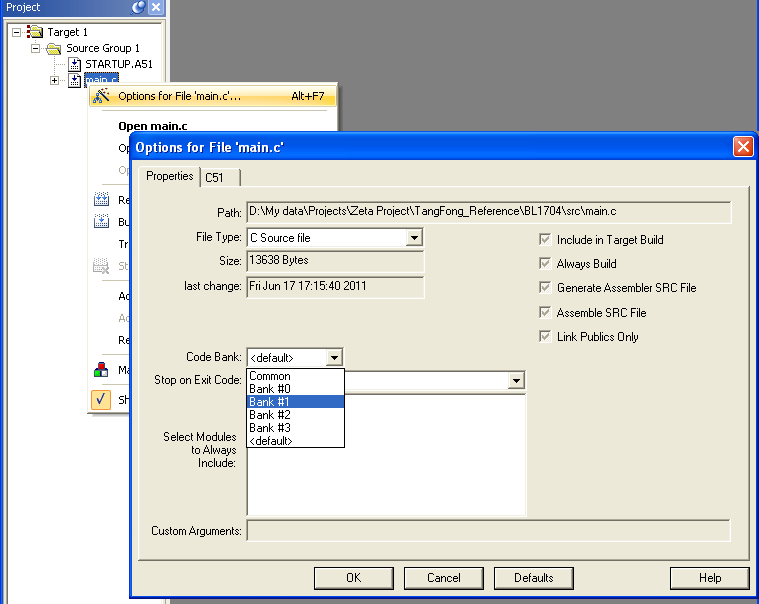
* Adding Source Files to Projects.



* If the files are misplaced, we can remove from one group and add to required group.



* Source files can be placed at required memory locations (Common area, bank0, bank1, bank2, bank3 by setting options for that particular file.



* µVision provides .MAP file as output. This file displays different memory sections as Data Memory, Code Memory (Code Bank 0, Code Bank 1, and Common Area).
* When you click on “Compile” for source file, then output similar to shown below is thrown in “output Window”

Build target 'Target 1'

assembling STARTUP.A51...

compiling main.c...

linking...

Program Size: data=26.0 xdata=389 const=8 code=1523

creating hex file from ".\obj\BLori"...

".\obj\BLori" - 0 Error(s), 0 Warning(s).

* If the source file has error(s) / warning(s) present in it, the Output window shows Output as below ,

Build target 'cf04afx0\_basic'

compiling RamGSM.c...

assembling RamTlkIni.s...

compiling RamTlk.c...

compiling RamBitarea.c...

compiling RamMain.c...

compiling RamIO.c...

assembling IRHandler.s...

assembling IRVectors.s...

assembling cstartup\_THC20F06BDV20.s...

assembling LibReg.s...

compiling LibMem.c...

compiling LibFlash.c...

compiling LibSW.c...

compiling LibSIO.c...

assembling L51\_BANK\_THC20F06BDV20.s...

compiling MainConst.c...

..\basic\main\MainConst.c(106): error C129: missing ';' before ''

compiling Main.c...

compiling LibMain.c...

..\basic\main\LibMain.c(163): warning C322: unknown identifier

compiling LibGSM.c...

compiling NVMTlkTbl.c...

Target not created

* µVision after compilation of the file executes post build scripts. The postscripts in our IDE are used for adding the CRC data to mot files. In this case however due to invocation of the post build batch files by µVision output get generated as below,

Build target ‘ cf05afx0\_basic'

compiling RamIO.c...

compiling RamBitarea.c...

compiling RamGSM.c...

compiling RamTlk.c...

compiling RamMain.c...

assembling RamTlkIni.s...

assembling cstartup\_THC20F17BD-A.s...

assembling L51\_BANK\_THC20F17BD-A.s...

assembling IRVectors.s...

assembling IRHandler.s...

assembling LibReg.s...

compiling LibSW.c...

compiling LibMem.c...

compiling LibFlash.c...

compiling LibSIO.c...

compiling Main.c...

compiling LibMain.c...

compiling MainConst.c...

assembling CryptoBasics.s...

compiling CryptoBasic.c...

assembling NVMIni.s...

assembling NVMFileSystem.s...

assembling NVMSecArea.s...

assembling NVMTlkIni.s...

assembling NVMSecureWrite.s...

compiling NVMClassCmdTbl.c...

compiling NVMTlkTbl.c...

compiling NVMApplTest.c...

compiling NVMCmdDebug.c...

compiling NVMOverlay.c...

compiling NVMOtass35.c...

assembling NVMOTPCopy.s...

assembling NVMCredelLocator.s...

compiling CmdVerify.c...

compiling LibGSM.c...

compiling LibVerify.c...

compiling CmdEnvelope.c...

compiling CmdFetch.c...

compiling CmdGetResp.c...

compiling CmdIncrease.c...

compiling CmdRead.c...

compiling CmdRehabInval.c...

compiling CmdRunGSM.c...

compiling CmdSeek.c...

compiling CmdSelect.c...

compiling CmdSleep.c...

compiling CmdStatus.c...

compiling CmdTermProf.c...

compiling CmdTermResp.c...

compiling CmdUpdate.c...

assembling SwitchTk.s...

compiling LibTlkSys.c...

compiling LibTlkUtil.c...

compiling LibTlkTlv.c...

compiling LibTlkFilter.c...

compiling TlkPro1Refresh.c...

compiling TlkPro2Refresh.c...

compiling TlkProDisText.c...

compiling TlkProGetInkey.c...

compiling TlkProGetInput.c...

compiling TlkProIdleModeText.c...

compiling TlkProLocInf.c...

compiling TlkProMoreTime.c...

compiling TlkProPlayTone.c...

compiling TlkProPollInt.c...

compiling TlkProPollOff.c...

compiling TlkProSendSM\_SS\_USSD.c...

compiling TlkProSetMenuSelectItem.c...

compiling TlkProSetUpCall.c...

compiling TlkProSetupEventList.c...

compiling CmdCreate.c...

compiling CmdDelete.c...

compiling CmdDeleteOS.c...

compiling CmdGetIniData.c...

compiling CmdLock.c...

compiling LibAdm.c...

compiling AdmCnt.c...

compiling Otass35.c...

compiling LibShared.c...

assembling CallOta35.s...

linking...

Program Size: data=211.3 xdata=11575 const=2089 code=45715

creating hex file from “.\Debug\Obj\cf05afx0”...

User command #2: .\Utils\bat\makcrc\_min\_flash.bat

PC-lint for C/C++ (NT) Vers. 8.00u, Copyright Gimpel Software 1985-2006

--- Module: ..\..\..\app\AdmCnt\AdmCnt.c (C)and

PC-lint for C/C++ (NT) Vers. 8.00u, Copyright Gimpel Software 1985-2006

--- Module: ..\..\..\basic\main\LibMain.c (C)

--- Module: ..\..\..\basic\main\Main.c (C)

--- Module: ..\..\..\basic\main\MainConst.c (C)

--- Module: ..\..\..\basic\main\RamBitarea.c (C)

--- Module: ..\..\..\basic\main\RamMain.c (C)

PC-lint for C/C++ (NT) Vers. 8.00u, Copyright Gimpel Software 1985-2006

--- Module: ..\..\..\hware\mem\LibFlash.c (C)

--- Module: ..\..\..\hware\mem\LibMem.c (C)

PC-lint for C/C++ (NT) Vers. 8.00u, Copyright Gimpel Software 1985-2006

--- Module: ..\..\..\hware\io\LibSIO.c (C)

--- Module: ..\..\..\hware\io\LibSW.c (C)

--- Module: ..\..\..\hware\\RamIO.c (C)---------------------------------------------------------

- -

- Lint output -

- ----------------------------------------------------------

--- Module: ..\..\..\app\AdmCnt\AdmCnt.c (C)

--- Module: ..\..\..\basic\main\LibMain.c (C)

--- Module: ..\..\..\basic\main\Main.c (C)

--- Module: ..\..\..\basic\main\MainConst.c (C)

--- Module: ..\..\..\basic\main\RamBitarea.c (C)

--- Module: ..\..\..\basic\main\RamMain.c (C)

--- Module: ..\..\..\hware\mem\LibFlash.c (C)

--- Module: ..\..\..\hware\mem\LibMem.c (C)

--- Module: ..\..\..\hware\io\LibSIO.c (C)

--- Module: ..\..\..\hware\io\LibSW.c (C)

--- Module: ..\..\..\hware\io\RamIO.c (C)---------------------------------------------------------

1 file(s) copied.

1 file(s) copied.

1 file(s) copied.

Extended 8051/251 Object to Hex File Converter V1.38

COPYRIGHT KEIL ELEKTRONIK GmbH 2000 - 2010

GENERATING INTEL H386 FILE: ..\work\cf05afx0\_xdt.hex

Checking if UCPE2SECUREWRITEPAGE is page alligned...OK (2600) 1 file(s) copied.

1 file(s) copied.

label1: SE2EFCFG\_C 23F5

label2: SE2MF\_E 240C

label3: SE2MF\_E+1 240D

label1: SE2CHV\_C 23E3

label2: SE2EFCFG\_E 23F4

label3: SE2EFCFG\_E+1 23F5

label1: SE2EFIC\_C 23D1

label2: SE2CHV\_E 23E2

label3: SE2CHV\_E+1 23E3

label1: UIE2HEACRCSTART 23BF

label2: SE2EFIC\_E 23D0

label3: SE2EFIC\_E+1 23D1

label1: UIE2STORCRC 2232

label2: UCE2STARTSYS\_E 223F

label3: UIE2STORCRCH 2232

Calculations successful!

Calculations successful!

1 file(s) copied.

1 file(s) copied.

1 file(s) copied.

XDATA Range: 0x02000 0x0AFFF

XRAM Range 0x0000 0x07FF

Traversing Map File to check if any Variable falling in codespace...Ok

".\Debug\Obj\cf05afx0" - 0 Error(s), 0 Warning(s).

* If following message is appeared then change

Checking if UCPE2SECUREWRITEPAGE is page aligned...Failed

UCPE2SECUREWRITEPAGE (6c01) needs to be page aligned

-----------------------------------

Please adjust init.h

#define GAP\_FOR\_EF\_CONF\_EF\_CHV\_BODY 0xaa

------------------------------------------------

Then please adjust init.h as mentioned.

* If following message appears then displayed variable is placed outside XDATA area. That variable needs to be placed in XDATA.

Traversing Map File to check if any Variable falling in code space...

ADM\_U\_ANFFREI0 0200F200H

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! !

! \*\*\*ALERT\*\*\* !

! System is prone to tear due to !

! variables placed in code space !

! !

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Generated output is not safe for final deployment

To avoid deployment by mistake, the output files (mot, hxd) are renamed as \*\_void

* If following message appears, then displayed RAM variable is placed inside NVM area by compiler.

Scanning .\AllowedMisplacedRAMVarList.txt to find variables for which warning is supressed.

Traversing Map File to check if any Variable falling in NVM space...

Here is the list:

aucRomRandK 02002A23H

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! !

! \*\*\*ALERT\*\*\* !

! System is prone to tear due to !

! RAM variables allocated in NVM space !

! !

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Generated output is not safe for final deployment

To avoid deployment by mistake, the output files (mot, hxd) are renamed as \*\_void

Note: If the displayed variable is genuinely a NVM variable, but indicated as RAM variable by the post build script, then this variable needs to be added to file AllowedMisplacedRAMVarList.txt. The script treats the variables in the text file as NVM variables. Please be double sure before adding a variable in the text file.

* If following message appears, then that means the FILESYSTEM\_SIZE is not correct depending upon whether the MASTER\_DEVICE\_ENABLED and ADN\_IN\_MD\_INIT are set to 1 or 0.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! !

! \*\*\*ALERT\*\*\* !

! System is prone to tear due to !

! FILESYSTEM\_SIZE defined in init.h !

! FILESYSTEM\_SIZE should be increased by 256 !

! !

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

The same message will also suggest the correction needed in the definition FILESYSTEM\_SIZE. Following table shows all possible conditions when this error can occur.

* If the following message appears, then that means the FILESYSTEM\_SIZE should be reduced by specified amount. Otherwise the bootloader may get corrupted when the filesystem is updated by the OS.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! !

! \*\*\*ALERT\*\*\* !

! System is prone to tear due to !

! FILESYSTEM\_SIZE defined in init.h !

! FILESYSTEM\_SIZE should be reduced by 256 !

! Otherwise Bootloader will be erased. !

! !

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

The same message also suggest the amount by which the FILESYSTEM\_SIZE should be reduced so that the bootloader will remain intact. However this is just a warning message and it will not prevent the creation of the output mot file. Because the bootloader corruption due to the filesystem could be optional.

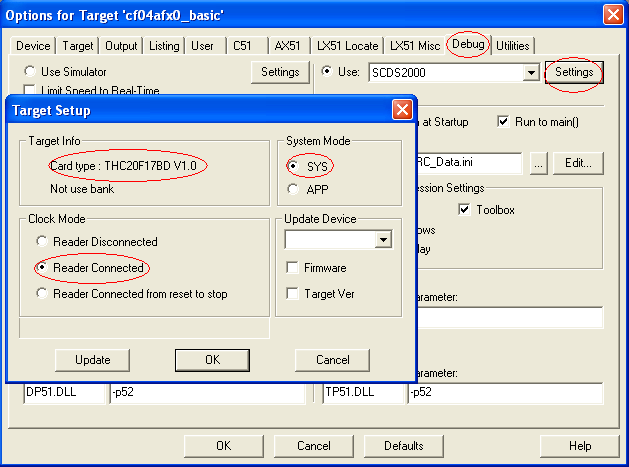
|  |  |  |  |
| --- | --- | --- | --- |
| MASTER\_DEVICE\_ENABLED | ADN\_IN\_MD\_INIT | End of FileSystem and bootloader area comparison | Error message |
| YES | YES | FileSystem end is after the start of Bootloader | FILESYSTEM\_SIZE should be reduced by xxxx |
| YES | YES | FileSystem end is before the start of Bootloader | FILESYSTEM\_SIZE should be increased by xxxx |
| YES | YES | FileSystem end is exactly at the start of Bootloader | No error message for FILESYSTEM\_SIZE |
| YES | NO | FileSystem end is after the start of Bootloader | FILESYSTEM\_SIZE should be reduced by xxxx |
| YES | NO | FileSystem end is before the start of Bootloader | No error message for FILESYSTEM\_SIZE |
| YES | NO | FileSystem end is exactly at the start of Bootloader | No error message for FILESYSTEM\_SIZE |
| NO | XX  (Any condition) | XX  (Any condition) | No error message for FILESYSTEM\_SIZE  Only warning message informing if the bootloader may get corrupted. |

If following message appears, then that means the MD functionality is disabled by setting the MASTER\_DEVICE\_ENABLED to zero. Still there is some code present in the bank 3. This code is prone to tear as the bank 3 will be erased at the time of personalization. For resolving this error, every module (file) other than MD functionality must be checked for placement in bank3. Also linker control file should be checked whether any module other than MD is placed in bank3.\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Error. MD disabled, still code present in bank 3. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Overlay map of Module is generated in .map file. We can find here how linker has overlaid data & xdata automatically. We can set overlay manually also in linker file & in source files. Interbank Call Table is generated in .map file to address the function called through various banks.
* We must check for proper device type, System mode & Clock mode in Options for Target window as below,

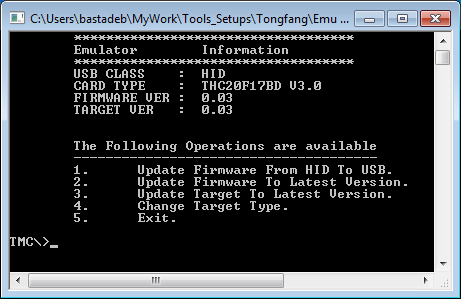


Note: For a required target type, system mode & clock mode has to be as shown above else it may lead to problem with no connection with the device.

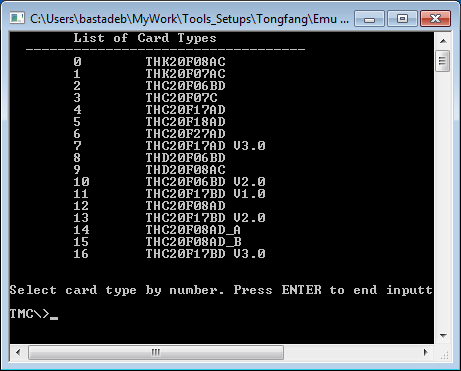
* The device type to be selected must be present in ‘cardtype.ini’ file located at “C:\Keil\C51\scds” folder. If not there is possibility that the emulator/debugger may fail to work.
* Tongfang has provided a utility which is *UpdateEmu.exe* that can be used for changing device type to different chip type on emulator. This is available along with tools required for installing toolchain at \\pun1file06p\Datapune\Departments\Team Telco\projects\Zeta\Tools\Emulator updates\Emulator Update tool\UpdateEmu20140922

You need to have emulator connected in order to use this utility.

Below are screenshots of usecase of this conversion.



Select Option 4 for changing Target Type.



Select Target Type you want emulator to be converted. It will take few minutes for conversion and finally will give message of conversion done successfully.

## Product family SCFxxxx

For Mizar project the compiler and linker are embedded with the IDE – Cortus. Hence it is possible to compile, link and generate the output MOT file using the IDE itself. However as a standard process of building the project, we have implemented the Cygwin based compilation with GCC. The GCC version being used is 4.7.3.

For building the project one has to open the Cygwin prompt, then has to go to the project build folder. In the build folder, there are multiple targets available for the project.

Given below is the list of these builds.

cc21afx0\_Basic\_OTASS35,

cc21afx0\_BasicRFM,

cc21afx0\_BasicTLK,

cc21afx0\_BasicTLK\_MD\_CreDel\_No\_Reclaim,

cc21afx0\_BasicTLK\_MD\_CreDel\_Reclaim,

cc21afx0\_BasicTLK\_UT,

cc21afx0\_Celltick\_14\_2\_6\_L1I0F1,

cc21afx0\_Celltick\_14\_3\_1\_L1I1F1,

cc21afx0\_Celltick\_14\_3\_1\_L1I1F1\_RFM,

cc21afx0\_Celltick\_14\_3\_1\_L1I1F1\_RFM\_MD\_CreDel\_Reclaim,

cc21afx0\_RFM13,

cc21afx0\_RFM13Lite,

cc21afx0\_WIB12,

cc21afx0\_WIB13,

cc21afx0\_WIB13\_Celltick\_14\_2\_6\_L0I0F1,

cc21afx0\_WIB13\_Celltick\_14\_3\_1\_L0I1F1,

cc21afx0\_WIB13\_Celltick\_14\_3\_1\_L0I1F1\_MD\_CreDel\_Reclaim,

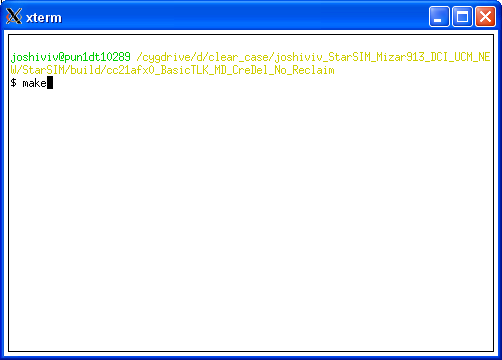
cc21afx0\_WIB13\_MD\_CreDel\_Reclaim,

cc21afx0\_WIB13Lite,

cc21afx0\_WIB13UltraLite

If one wants to add any additional target for some specific configuration, then it should be added similar to the existing targets.

For building one particular init it must be given a make command through bash as given below.



Apart from this, the build system will also generate the log files in the folder log. These files are Buildlog.txt and lint.log. The Buildlog.txt contains the steps executed while building the project and the lint log contains the linting errors in the project.

### Options implemented with make command –

The make command has been modified by updating the make files of the project. These options are as given below.

* make EMU=1 - This option must be given in order to compile a target with debug information so that one can debug the target using a Starchip emulator.
* make OPTI=0 – This option must be given in order to compile a target with optimization level zero. For more details on optimization level, please refer GCC manual.
* make OPTI=2 – This option must be given in order to compile a target with optimization level two. For more details on optimization level, please refer GCC manual.
* If none of the above mentioned options are provided while building the project, then by default the EMU option is not set and the optimization level is set to OS.

### Error messages shown by the build system

* ../../src/cc21afx0\_mizar136/NVM/include/init.h:239:3: error: #error

../../src/cc21afx0\_mizar136/NVM/include/init.h:240:10: note: #pragma message: The value of FILESYSTEM\_SIZE can not be more than (0xFE00 - (0x0400 + 0x400)).

When the above mentioned message is shown by the build system, then the macro definition FILESYSTEM\_SIZE in the corresponding init.h should be reduced by the requested amount.

* Error - Images not matching

This error message is for indicating some internal error in the build system or the error in the generated jtag file. This error is produced by the Ccitt\_Crc16\_8404 tool. While trying to change the record size from 0x80 to 0xF0, if there is some error happened due to objcopy.exe from GCC or by any other means, then this error message will be shown.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! !

! \*\*\*ALERT\*\*\* !

! System is prone to tear due to !

! RAM variables allocated in NVM space !

! !

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

"\nGenerated output is not safe for final deployment\n";

"\nPlease Check the Misplaced\_ram\_var.txt for variables\n";

This error message is for the possible failures if by any reason the RAM variables are placed in the NVM space.

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! !

! \*\*\*ALERT\*\*\* !

! System is prone to tear due to !

! FLASH variables allocated in RAM space !

! !

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Generated output is not safe for final deployment

Please Check the Misplaced\_ram\_var.txt for variables.

This error message is for the possible failures if by any reason the flash variables are placed in the RAM space.

* EFIC Header is not Found in map file

This error message is generated if the EFIC header is not found to the build system during the compilation process.

* EFchv Header is not Found in map file

This error message is generated if the EFchv header is not found to the build system during the compilation process.

* efcfg Header is not Found in map file

This error message is generated if the build system can not find build header

* size\_cin\_u\_chv\_e is not found

This message is generated by the build system if it can not find the size\_cin\_u\_chv

* efdelkey\_address not found

This message is generated by the build system if it can not find the address of efdelkey\_address,

* Section FunctionsUsedWithinDeleteOS

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! !

! \*\*\*ALERT\*\*\* !

! Section .FunctionsUsedWithinDeleteOSshould be Placed !

! Beyond Address :: 0x25ff !

! Current Address of section .FunctionsUsedWithinDeleteOS is:: 0xYYYYYYYY !

! Please refer to R&D document for more details. !

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Please stop the Deployment.

Since within Delete OS command execution; Main OS contents

present at memory addresses 0x2000 to 0x25ff will be erased and part of bootloader will be written in this memory area. Hence such functions which are required after execution of above mentioned operation, for completing Delete OS functionality, should be placed beyond memory address 0x25ff.

For e.g. According to current source code base following functions should be placed beyond memory address 0x25ff.

\* CmdDeleteOS\_Auth(void)

\* CmdDeleteOS(void)

\* DFA\_error(void)

\* Gdbl\_\_DeleteOs(void)

\* LoopUntilDoomsday(void)

\* LbhSendByte(unsigned char ucByte)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! !

! \*\*\*ALERT\*\*\* !

! Section .FunctionsUsedWithinDeleteOSshould is missing!!!! !

! Section .FunctionsUsedWithinDeleteOSshould should be !

! present and to be Placed !

! Beyond Address :: 0x25ff !

! Check and Update your source code and linker file. !

! Please refer to R&D document for more details.\n !

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Please stop the Deployment.

As explained in above error message all functions which are required after execution of erase operation (over the memory address 0x2000 to 0x25ff) within Delete OS command; should be placed within section FunctionsUsedWithinDeleteOS and this section should be placed beyond memory address 0x25ff.

# Memory Layout

## THC20F17BD-XX family

### Flash Organization in Detail

1. **THC20F17BD ,THC20F17BD-V20 and THC20F17BD-V30**

|  |  |  |
| --- | --- | --- |
| Flash Organisation | Logical Address | Physical Address |
| CODE | Code bank 3 – 0x8000 : 0x8FFF  Code bank 2 – 0x8000 : 0xFFFF  Code bank 1 – 0x8000 : 0xFFFF  Code bank 0 – 0x8000 : 0xFFFF  Common Area – 0x0000 : 0x7FFF | Code bank 3–0x20000: 0x20FFF  Code bank 2–0x18000: 0x1FFFF  Code bank 1–0x10000: 0x17FFF  Code bank 0– 0x8000: 0xFFFF  Common Area–0x0000 : 0x7FFF |
| XDATA | Xdata – 0x2000 : 0xFFFF  Xdata – 0x0800 : 0x1FFF  Xdata – 0x0000 : 0x07FF | Xdata – 0x13000 : 0x20FFF  Blank  XRAM – 0x0000 : 0x07FF |

### Memory Map

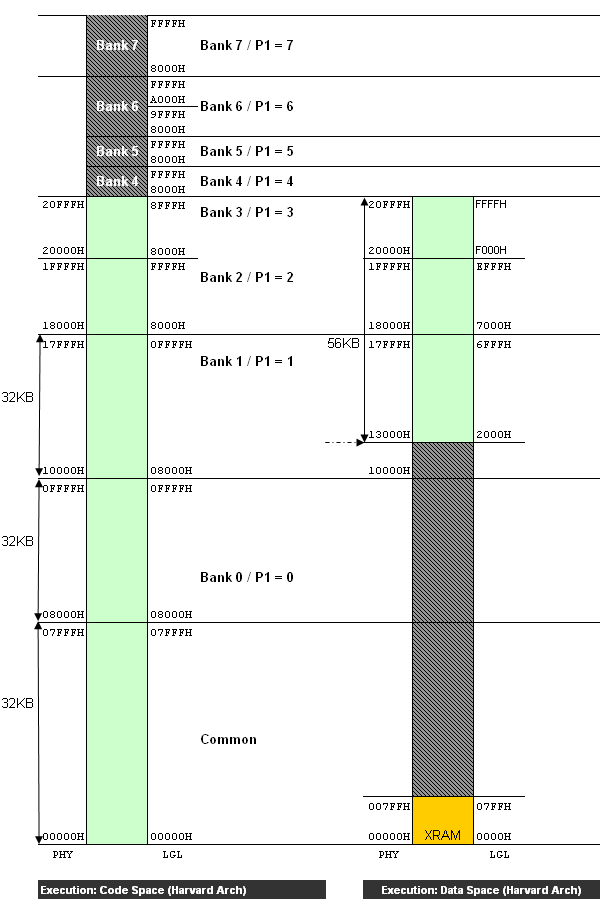
1. THC20F17BD and THC20F17BD-V20/ THC20F17BD-V30

Hardware allows CPU to address additional memory banks to surmount the 64 KB limit of CODE and XDATA space.

The physical FLASH memories are partitioned to 1 static part (namely Common Bank) and 8 dynamic parts (namely bank). There may be some blank areas inside the banks because the total capacity of the banks is much more than the FLASH memories.

The Common Bank is defined as a low-end FLASH area between the address 0 and the starting address of dynamic banks.

Following figure explains about code space and xdata space in normal execution mode.



### Memory Structure

In order to keep the code of the basic sources as constant as possible, all sources are linked deliberately into the project. Several defined sections can be found:

1. As prescribed by the hardware platform the code is started with the interrupt vector table at address 0x0003. There the jump instruction to the cstartup\_THC20F17BD.s can be found:

FD7581BA90168093B4FF0975C30175900102F80012030A021680021680E493A3

Only 56 (hex) bytes are used for the interrupt table.

1. µVision does memory allocations as per the relocate able segments defined in linker file. In the below example, the segments defined would get placed in the region starting from 0x2000.

SEGMENTS

(

?XD?NVMSECUREWRITE(X:0x2000), ?XD?NVMINI,

?XD?NVMTLKINI, ?XD?NVMOTPCOPY,?XD?NVMOTASS35,

?XD?NVMFILESYSTEM

)

1. Start address of NVMSECUREWRITE segment should be calculated as below.

Case 1: Code ends in Common Area

NVMSECUREWRITE Start Address = 0x2000

Case 2: Code ends in Bank0

NVMSECUREWRITE Start Address = 0x2000

Case 3: Code ends in Bank1, Code\_End\_Address < 0xB000

NVMSECUREWRITE Start Address = 0x2000

Case 4: Code ends in Bank1, 0xB000 < Code\_End\_Address < 0xFFFF

NVMSECUREWRITE Start Address = 0x2000 + (Code\_End\_Address - 0xB000) + (0x100 - (Code\_End\_Address % 0x100))

Case 5: Code ends in Bank2,

NVMSECUREWRITE Start Address = 0x7000 + (Code\_End\_Address – 0x8000) + (0x100 - (Code\_End\_Address % 0x100))

The debug flag is located at [C: 0200H]. This is required for switching between Tongfang Bootloader and Main OS. The startup code switches to Tongfang Bootloader if debug flag is 0xFF else it switches to Main OS if it is 0xA5.

During production this flag is changed to A5h after loading complete OS to card. Lock Chip command is used to change debug flag, after OS is loaded to the card and checksum verification is successful.

It makes sense to describe the memory layout from higher to lower address therefore the description starts with the flash end.

1. The linker file contains reserved areas, these areas have significance given below:

Note : Reserve areas mentioned below are wrt BasicOS initialisation. These will change per initialisation.

RESERVE (X:0x0800-X:0x1FFF)

Reserved for Invalid RAM Address.

Tongfang chips THC20F17BD/ THC20F17BD-V20/THC20F17BD-V30 contains only 2K of RAM which is present at logical address [X:0x0000-X:0x07FF]. Below is the table that describes the logical address and its significance for all chip types of THC20F17BD.

THC20F17BD//THC20F17BD-V20/THC20F17BD-V30

|  |  |
| --- | --- |
| XDATA Address Range (logical) | Significance |
| 0x0000 – 0x07FF | External RAM |
| 0x0800 – 0x1FFF | Empty |
| 0x2000 – 0xFFFF | XDATA (56Kb) |

Keil Compiler does not give any error if any code is linked in this region, to overcome this problem this area is reserved so that if any data is mapped in this region compiler will give an error.

RESERVE (X:0x0800-X:0x1FFF) //Reserve for invalid RAM Address

RESERVE (B1:0xB000-B1:0xFFFF)//Reserve XDATA space in code

RESERVE (B2:0x8000-B2:0xFFFF)//Reserve XDATA space in code

RESERVE (B3:0x8000-B3:0x8BFF)//Reserve XDATA space in code

RESERVE (B3:0x9000-B3:0xFFFF)//Reserve XDATA space in code

This is to avoid overlapping of code with XDATA. If an application is added to the OS, it will consume more space than pre calculated space, due to which there are chances that the code extends in XDATA. So if error is generated for code overlapping with this area, change lower address to new code end address after adding application, and relocate the NVMSecArea segment accordingly.

On initialization with MD Credel feature below area is reserved for CRADEL

RESERVE (X:0xE000-X:0xF000) //Reserve for CREDEL code

This is to reserve space for CREDEL code in XDATA so that no NVM variables are mapped to this region.

1. The Flash specific data and G&D internal management values (ATR, OS version, etc.) are constants and written in Code area of Tongfang chip. So for THC20F17BD these values are written as constants. Addresses are defined below:

0x0201 – 0x0202: Last Flash Address  
 0x0203 – 0x0204: Last Ram Address  
 0x0205 – 0x0207: OS Version  
 0x0208 – 0x020A: Chip Type (coded by G&D)  
 0x020B – 0x020D: G&D Mask Name abbreviation (“ZET”)

All these values are already coded in assembly module NVMSecArea.s

1. Now follow the WIB WRITE and SMS Receive/Submit Buffer sections. Again these areas have been separated from the rest of the flash data into special pages so that any write/erase cycle on these data can not destroy code or other important data. Example entries

?XD?NVMWIB13WRITE 0x9B00 0x200 // WIB VarBuf as defined in init.h

Link this section into the project.

It is crucial that both WIB\_WRITEAREA and SMS\_WRITEAREA begin with a page boundary and the size of the sections have to be a multiple of a page size as well. The size of the arrays is defined by the linker file but care has to be taken of that the settings in init.h match with these values. The arrays are defined in module NVMConCatc.c.

1. The page before the WIB\_WRITEAREA and SMS\_WRITEAREA is reserved for the Secure Write buffers (Restricted Secure Write). In total there are three Secure Write buffers which are used alternately. The buffers are coded in NVMSecureWrite.s.

1. The free File System can be extended upto Flash end only in Real card initialization. During development on Delini init’s the file system should spare the Boot loader.
2. The bodies of EF-Conf and EF-CHV are placed above the file system But as the body needs to be page aligned (requirement for secure write), if the flash filesystem does not end at a page boundary, a GAP is left until the page boundary is reached and from there 0x100 bytes (Size of the body) above, the bodies of EFCHV and EFConf would begin. The coding of this can be seen in NVMFileSystem.s
3. The standard file headers for EFIC, EFCHV, EFConf and the MF are linked directly in front of the free file system. As long as no new standard files are added to the project nothing has to be done. The file headers are automatically linked to their correct position in flash.
4. The start address of segment SBOX Constants should have 9 least significant bits equal to zero, so please take care of this requirement while relocating this segment. It can be relocated by changing its address in linker file. For Zeta 132 K Basic OS it is defined in linker file as below,

?CO?SBOXAA55(C:0x7E00)

Start Address: 0x7E00 => 9 Least Significant Bits are zero as required.

0111 111**0 0000 0000**

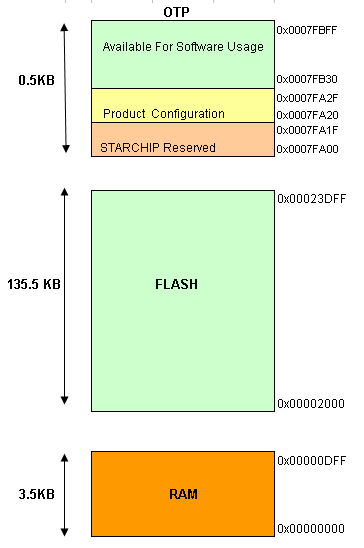
### Flash Memory Overview

|  |  |  |
| --- | --- | --- |
| **Bootloader** |  | 0x020FFF |
|  | 0x020000 |
| **Free File System** |  |  |
| **Bodies of EF\_CHV,EF\_CONFIG, EF\_IC** |  |  |
| **File Headers  (MF,EF\_CHV,EF\_CONFIG, EF\_IC)** |  |  |
| **NVM Data (NVM Tlk Ini data, buffer etc depending on Apllications)** |  |  |
|  |  |
|  |  |
|  |  |
| **IO routines Interrupt Vectore Crypto, GSM 11.11 GSM 11.14 Application Code** |  | 0x00000 |

## SCFxxxx family

### Memory Map

The flash memory of the chip is mapped from the location 0x2000 to 0x23DFF. Apart from this there is a One Time Programmable (OTP) area which is located at the location 0x7FA30 – 0x7FBFF.



### Memory Structure

In order to keep the code of the basic sources as constant as possible, all sources are linked deliberately into the project. Several defined sections can be found:

1. As prescribed by the hardware platform the code is started with the interrupt vector table at address 0x2000. The first location of the vector table points the \_startup which is at the location 0x2080 in case of Mizar. That means the value at the location 0x2000 is 0x2080.

Only 80 bytes are used for the interrupt table.

1. GCC does memory allocations as per the segments defined in the linker file. In the below example, the segment B would be placed after the segment A. Both the segments will be placed in the nvmdata.

.SegmentA :

{

. = ALIGN(\_SECTOR\_SIZE);

KEEP (\*.o (VariableA));

. = ALIGN(\_SECTOR\_SIZE);

} > program\_memory :nvmdata=0x00

. SegmentB :

{

. = ALIGN(\_SECTOR\_SIZE);

KEEP (\*.o (VariableB));

. = ALIGN(\_SECTOR\_SIZE);

} > program\_memory :nvmdata=0x00

1. Refer section Flash Memory Overview for memory overview in Mizar. The Secure Write is reserved for the Secure Write buffers (Restricted Secure Write). The size of secure write buffer is 450 bytes and it is coded in NVMSecureWrite.c
2. The free File System can be extended upto Flash end only in Real card initialization, not during development, it may damage card.
3. The bodies of EF-Cong and EF-CHV are placed above the file system But as the body needs to be page aligned (requirement for secure write), if the flash filesystem does not end at a page boundary, a GAP is left until the page boundary is reached.
4. The standard file headers for EFIC, EFCHV, EFConf, EFDelKey (Optional if delete OS with authentication is enabled) and the MF are linked directly in front of the free file system. As long as no new standard files are added to the project nothing has to be done. The file headers are automatically linked to their correct position in flash.

### Flash Memory Overview

|  |  |  |
| --- | --- | --- |
| **Bootloader** |  | 0x00023DFF |
|  | 0x00023400 |
| **Free File System** |  |  |
| **Bodies of EF\_CHV,EF\_CONFIG, EF\_IC** |  |  |
| **File Headers  (MF,EF\_CHV,EF\_CONFIG, EF\_IC)** |  |  |
| **NVM Data (NVM Tlk Ini data, buffer etc depending on Apllications)** |  |  |
|  |  |
|  |  |
|  |  |
| **IO routines Interrupt Vectore Crypto, GSM 11.11 GSM 11.14 Application Code** |  | 0x00002000 |

# G&D bootloader

For the product family SCFXXXX (StarChip), the bootloader is developed by G&D. The details about this bootloader are available in the document Software Design Document Mizar Bootloader.doc at \StarSIM\_Doc\V914\StarSIM\_V914\20000-Dev\21000-ArchDesign\Bootloader\Software Design Document Mizar Bootloader.doc

This bootloader consumes around 2.5K of space on flash. The bootloader is present on the top of the flash address from 0x23400 to 0x23DFF.

# TearSafe feature –

## SCFXXXX (Mizar) family

This feature is implemented on SCFXXXX (StarChip) family. It is used to ensure that every NVM update operation is executed in atomic way. That means even if the power is interrupted (tear) in between during a NVM update operation, then still the NVM will have either new data or old data. It will never have a garbage data. For more details about the implementation of this feature, please refer the Software Design Document TearSafe.docx

To enable this feature the flag TEAR\_SAFE\_WRITE must be enabled.

The space consumed by this feature on the flash is around 3.5K.

Due to large size of TearSafe feature, sometimes this feature is enabled only for the crucial flash oeprations. For example for the CHV verification and update. Such a feature is referred as partial secure write or only secure write. On the other hand when the full implementation of this feature is considered, it is referred as full secure write or TearSafe feature.

The TearSafe feature on StarSIM Mizar V914 and so called Full Secure Write feature on Atlas serve the same purpose. But the implementation of this feature on Atlas and StarSIM Mizar V914 are different. The details about TearSafe feature on StarSIM Mizar V914 are documented at ,

\StarSIM\_Doc\V914\StarSIM\_V914\20000-Dev\21000-ArchDesign\TearSafe (Full Secure Write)\Software Design Document TearSafe.docx

## THC20F17BD-XX (Zeta) family

Due to complex architecture of this micro family, the TearSafe or the Full secure write feature is not supported on this platform. Instead, this family supports the Secure Write or the partial secure write feature which is implemented only for the crucial data update like CHV update.

# Secure write mechanism –

This limited secure write feature is applicable for both the families in the same way. As mentioned earlier, the TearSafe or the full secure write feature is very flash consuming. Hence it can not be implemented at all on the THC20F17BD-XX (Zeta) family. Where as on SCFXXXX (Mizar) family, this feature is optional. Following explanation is for the Secure write or partial secure write mechanism for both the families..

When the operating system writes sensitive data (file headers, PINs, error counters, etc.) to the FLASH, there is some risk, that data to be written become inconsistent, if the power supply breaks down during page erasing or writing.

In order to anticipate such inconsistencies, data are first written to a buffer located in FLASH. Only if this has been done without errors, the data are copied from the buffer to the actual destination. If this action is interrupted by power loss, the SIM tries to copy from the FLASH buffer to the actual address again directly after the next start-up of the chip. After a successful update the content of the buffer is deleted.

Because of the large page size and restriction of FLASH memory, a secure write mechanism is implemented only for writing PINs, the associated error counters (body of EFCHV) and the CHV1 enable/disable flag (body of EFConf).

This implementation handles buffer of size 69 bytes for Mizar 136K. The bodies of EFConf and EFCHV must, therefore, be located in the same page starting at the page boundary. The rest of the page is erased at each update and, therefore, cannot be used to store other data. A second FLASH page is employed to install three buffers, which are used in an alternating way. Only after three updates this page needs to be erased. Thereby, the stressing of this page is reduced.

# THC20F17BD-XX family specific

## Generation of an Initialisation

In this chapter some basics for the development of initialisations/variants for Zeta are described.

### Preparation of an Initialisation/Variant

The Variant concept remains same as the version 910 of Atlas.

This new Zeta version offers all features like WIB1.2, WIB1.3-Lite, WIB1.3, WIB1.3-UltraLite,BasicRFM,RFM13,RFM13Lite,CELLTICK etc.. Now it is in the responsibility of the application developer to collect the correct sources together to provide an initialisation according to the customer’s requirements. Nevertheless, some preconfigured initialisations are available which serve as a starting point for application development.

Guidelines for application developers:-

To get the source code, fetch the basic source mask tree for Zeta132 v914 from .\StarSIM\_V914\_DCI\StarSIM\. Copy the directory “src” & “Build\_Zeta “ to any drive you like, e.g. “D:\Zeta”. So now one should end with the following path: “D:\zeta\src” & “D:\zeta\ Build\_Zeta”.

To retrieve the source code for development within DCI, open Clear Case Version control tool and join project “StarSIM\_V914\_DCI” from UCM vob for StarSIM which is “StarSIM\_PVOB”

Once the view is created successfully, go to .\STARSIM\src\

Note: Sufficient access permissions of ClearCase are required for development purpose.

development purpose.

This \src folder already contains one folder for preparing initialisations , which is src\sys\IniConfigs\TongFang\THC20F17BD for Zeta 132KB initialisations.

In Zeta there are project files and batch jobs for basic as well as all other applications. We can find project files and batch for specific target in \Build\_Zeta\cf04afx0\_XXXXX. where XXXXX is name of Target/Initialisation to be created.

If you have chosen to create an initialisation starting from one of the default targets you have several already prepared initialisation for a choice. One simply has to execute the batch file of respective target . The correct variant.h, init.h & linker are automatically provided. Then compile project using project files present in respective target .

we need to do only platform dependant, compiler specific changes to files such as init.h, and many more. Create projects files & batch jobs for basic as well as required applications. Make sure that you create correct batch job which creates copy of three files e.g. Basic-tlk-init.h into init.h ,variant-basic.h into variant.h, basic-tlk-THC20F17BD.lin into THC20F17BD.lin. The Above three files of which copies will be created are same for THC20F17BD-V20/ THC20F17BD-V30..

**Check that you have the correct init.h, variant.**

After compiling such a target it is possible to get the compilation output eg. Build log/mot file/lint log etc. under specified target directory itself in respective folders like /images or /buildlog.

Preconfigured initialisations for Zeta132:

* BASIC: Plain initialisation without application
* BASIC RFM: initialisation with basic RFM only
* BASIC OTASS3.5 : Plain initialisation with OTASS3.5 feature enabled. CreDel and MD feature disabled.
* BASIC MD, No CREDEL reclaim: Plain initialisation with MD feature enabled and only MD space reclaimed for ADN File.
* BASIC MD+CREDEL reclaim: Plain initialisation with MD and CREDEL features enabled and CREDEL space reclaimed for ADN file.
* Celltick 14.2.6 : initialisation using Celltick 14.2.6 only
* Celltick 14.3.1 RFM,MD+CREDEL reclaim : initialisation including Celltick 14.3.1 , Basic RFM with MD and CREDEL features enabled and CREDEL space reclaimed for ADN file.
* RFM13: initialisation with RFM13 only
* RFM13Lite: initialisation with RFM13 only
* WIB12: initialisation including Basic RFM and WIB1.2 browser
* WIB13-lite: initialisation including RFM13 and WIB1.3 browser
* WIB13: initialisation including RFM13 and WIB1.3 browser
* WIB13 MD+CREDEL reclaim: initialisation including RFM13 , WIB1.3 browser with MD and CREDEL features enabled and CREDEL space reclaimed for ADN file.
* WIB13-UltraLite: initialisation including RFM13 and WIB1.3 browser

### Linker file & Sections:

Keil uses linker file for defining how the memory is to be utilised. The sections and memory allocation would get modified with the changes carried out in the code, hence always ensure NVM modifiable region is always placed in the last. Map file is useful to view the placements, it would show all the placements as per the region it has got allocated.

While porting the existing code please check and verify the section definitions in the code and the placements shown by Map file are as per expectations.

For example refer linker file structure given below:

The Linker file typically looks as below,



Linker file uses various segments & directives to serve its purpose.

- The *print* directive specifies the name of the map file. If no *print* directive is specified the map file is given the name of the generated object file with a .MAP extension.

- The *reserve* directive reserves specified memory areas (*range*) and prevent the linker from locating anything in them.

- By default, the linker sorts sections by size (largest to smallest) before locating them. This ensures fewer memory gaps and reduces overall memory consumption. The *nosortsize* directive disables segment size sorting.

- By default, the linker generates an inter-bank jump table for code banking programs. The jump table is used for jumps or calls between functions in different code banks. Normally, **AJMP** or **LJMP** instructions are used depending on the size of the table. The *noajmp* directive prevents the use of **AJMP** instructions in the inter-bank jump table. This directive is used in inits wherever Linker Code Packing option is enabled.

- The segment *class* specifies memory space for the segment and is used by linker to access all segments that belong to that class e.g. CODE, XDATA, CONST, DATA, and IDATA

- With the *segments* directive we can specify absolute memory location for a segment, also we can decide segment order in memory.

- The *overlay* directive modifies call tree that is automatically generated by linker. With this we can add new root segments, exclude segments from overlay analysis, and add/remove call references between segments.

- While porting existing code please check and verify section definitions in code and placements shown by linker file as per expectations.

### NVM File System

NVMFileSystem.s has to be page aligned. It is taken care in code as below,

*?XD?NVMFILESYSTEM SEGMENT 'XDATA\_NVMFILESYSTEM' PAGE*

- A special note regarding NVMFileSystem.c

In the file “NVMFileSystem.c” refer to section, MASTER FILE MF (\_se2MF :).

*CRCEND\_SE2MF\_E: ; For Zeta label must be placed here for crc\_cc.exe tool to work*

*DB HIGH adm\_u\_anffrei0 ; pointer to first free memory of directory*

*;CRCEND\_SE2MF\_E: ; For Atlas label must be placed here for crc\_cc.exe tool to work*

*DB LOW adm\_u\_anffrei0 ; pointer to first free memory of directory*

* **File System Header Start Address Alignment**

NVMFileSystem.s consists of the default files created as part of the init. In-case any modifications are carried out in the file system, these would affect the start address of Label “FILESYSTEMSTART”. This label being used for file system management will affect the file system. Hence whenever any change is carried out it is required to ensure the address remain word aligned. This is achieved using following formula wherever word alignment is required.

$IF (($ AND 01H) == 01H)

ORG ($+1)

$ENDIF

* **Secure Write Page Area Start Address**

To ensure that the secure Write Page Area starts on page aligned address, a label “LABEL\_SEC\_WRITE\_PAGE\_START” must be added on top of line DS GAP\_FOR\_EF\_CONF\_EF\_CHV\_BODY in file NVMFileSystem.s. This label is used by the script to check if the secure write page (ucpe2SecureWritePage) is page aligned.

### Determination of the File System Size

To determine the free file system size one has to look into the map file.

To calculate free memory using map file, refer to section XDATA in map file. At the end of the section you will see similar to the following info;

00A300H 00EAFFH 004800H PAGE UNIT XDATA\_NVMFILE ?XD?NVMFILESYSTEM

Now the available free size for Filesystem in Zeta 132 KB is;

[Tongfang Bootloader start address] - [Start address of XDATA\_NVMFILE?XD?NVMFILESYSTEM section] – 0x200 (DF\_EF\_Headers)

[0xF000] - [0x2400] - 0x0200 = 0xD200

Note:

No additional code or any other section should be present in flash memory after NVMFILESYSTEM\_REGION.

In the above example we have taken the free memory available between the two sections, Tongfang Bootloader and Filesystem. It can be extended upto the flash end i.e. FFFFh, in Real ini. But in development environment on card, Tongfang Bootloader space [X: F000h – X: FFFFh] should never be claimed, as it will damage the card, card can never be initialized after that, as Tongfang Bootloader is overwritten.

The free files system size must be an even number. And it could be maximum set to 0xE000 only; to be defined as value of FILESYSTEM\_SIZE in init.h.

**Note: The Filesystem Size must be an even number. And Filesystem cannot be more than 0xE000 for Zeta 132K**

This exercise should be carried out always after adding new application, functions to the base ini.

### System Files

The project files can be found in folder \Build\_Zeta\cf04afx0\_XXXXX. **where XXXXX is name of Target/Initialisation.**

The map file of the project will be created in \Build\_Zeta\cf04afx0\_XXXXX [\images](file:///\\images). The hex file of the project is created in \Build\_Zeta\cf04afx0\_XXXXX \Obj. This hex file is copied to the directory \Build\_Zeta\cf04afx0\_XXXXX [\images](file:///\\images). Another file cf04afx0\_XXXXX \_xdt.hex which is generated by post script contains XDATA. Checksum is added to the file cf04afx0\_XXXXX \_xdt.hex to generate cf04afx0\_XXXXX \_CRC.hex file. This is an intermediate hex file containing code and xdata both, used to generate final mot file cf04afx0\_XXXXX.mot in Motorola format. This has to be delivered to production. The CRC-CCITT checksum information is embedded as S5 record of the mot file.

The final flash initialisations are created in ..\Build\_Zeta\cf04afx0\_XXXXX [\images](file:///\\images).

*Project Files:*  cf04afx0\_XXXXX.uvproj, etc.

*Map File:*  cf04afx0\_XXXXX.map

*Target File:*  cf04afx0\_XXXXX \_ChecksumDelta.hex, cf04afx0\_XXXXX.mot, cf04afx0\_XXXXX.hex

**Note: The file to be delivered for production** cf04afx0\_XXXXX**.mot!**

### Batch Files and Executables

The following batch files and executables are used during Flash generation. To run the batch jobs the following files must exist:

* \StarSIM\Build\_Zeta\makcrc\_min\_flash.bat , XDATA\_HEX\_FileCorrection.pl , ZetaPostProcessing.pl , ZetaVariableInCode.pl, FindMisplacedRamVariables.pl , MergeSolInMot.pl
* \StarSIM\Build\_Zeta\Tools\CRC\_cc\_win.exe , Ohx51.exe , CS\_MIN.lda , checksum.exe(with hexio.dll)

**Note: There is no compareROM.bat and Init.s\_Create.bat on Zeta!**

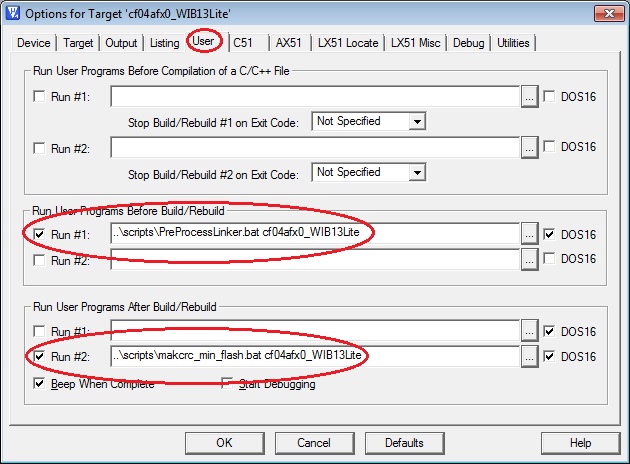
**On Atlas**

**makcrc\_min\_flash.bat**

This batch recalculates the checksums in the NVMini.s file (system area) and the file header checksums and enters them at the correct addresses in the initialisation. Further it creates the cf04afx0\_XXXXX \_CRC.hex, where all FF blocks are deleted.

Same batch file also checks if the MD functionality is enabled or disabled in the file Init.h. If it is disabled then expectation is there should be no code residing in the bank3. If in case there is code in bank 3 even after disabling the MD functionality then it will be a problem. This is because the same location is used by the ADN file in final product use. Hence the code in the bank 3 will be deleted then which will result in to a code crash and final product failure. To avoid this mishap, the batch file ensures the usage of bank 3 only by the MD functionality which is not a part of final product.

The batch jobs runs subsequent to linking the project. Failure to run this file will result in an error ATR. The batch runs automatically each time the project is compiled. The automatic execution is evoked by setting the options for target as below



Furthermore this batch job also is responsible for calculation the CRC-CCITT checksum , which is needed for the verification of an initialized Flash card.

**XDATA\_HEX\_FileCorrection.pl**

This Perl script is introduced in Zeta. It is used for concatenation subsequent parts of the final \*.hex file, code & xdata. Its usage is

perl XDATA\_HEX\_FileCorrection.pl ../%CURRENT\_TARGET%/TempFiles/%OUTPUT\_FILE\_NAME%\_xdt.hex

Where,

CURRENT\_TARGET = initialisations to be created (cf04afx0\_XXXXX)

OUTPUT\_FILE\_NAME= initialisations to be created (cf04afx0\_XXXXX)

Note: Please ensure that the Perl path has been set in the system path so that the batch files can automatically pick up the perl.exe for executing the scripts.

**MergeSolInMot.pl**

This perl script is used to merge the SOL file contents in the S5 record of the MOT file. Its usage is

perl MergeSolInMot.pl ..\output\%OUTPUT\_FILE\_NAME%.Mot ..\output\%OUTPUT\_FILE\_NAME%.sol

After this command the SOL file is deleted using the batch file command

del ..\output\%OUTPUT\_FILE\_NAME%.sol /Q/F/AH

**CRC\_cc\_win.exe**

This executable is to calculate CRC for the labels listed in file CS\_MIN.tpl, and replace the values of these labels by calculated CRCs. For Zeta 132K, list of CRC labels and files containing them is as below.

The labels are placed in sequence, as per their location in NVM, addresses are mentioned in below table are for basic initialization on Zeta 132K.

|  |  |  |
| --- | --- | --- |
| CRC Label | File Name | Address in NVM  (For Zeta 132K Basic Ini) |
| UIE2STORCRC | NVMIni.s | 0x2232 |
| UCE2STARTSYS\_E | NVMIni.s | 0x223F |
| UIE2STORCRCH | NVMIni.s | 0x2232 |
| UIE2HEACRCSTART | NVMFileSystem.s | 0x23BF |
| SE2EFIC\_E | NVMFileSystem.s | 0x23D0 |
| SE2EFIC\_C | NVMFileSystem.s | 0x23D1 |
| SE2CHV\_E | NVMFileSystem.s | 0x23E2 |
| SE2CHV\_C | NVMFileSystem.s | 0x23E3 |
| SE2EFCFG\_E | NVMFileSystem.s | 0x23F4 |
| SE2EFCFG\_C | NVMFileSystem.s | 0x23F5 |
| SE2MF\_E | NVMFileSystem.s | 0x240C |

Note: If any new label for CRC is added to NVM files, corresponding entry should be made to the file CS\_MIN.tpl, if not done, an error ATR is received.

## Porting Assembly Code

For porting assembly code to this new compiler based inits we need to take care of how data would be exchanged between the C and assembly code.

For Zeta, the microcontroller type is 8-bit extended 8051 core. It supports instruction set for 8051. For porting assembly code from Atlas to Zeta, reference is taken from Electra code base.

## Application Notes

### More Times on Zeta

The DES calculates fast enough that even for 7 encrypted, concatenated messages the calculation time is below 2s. Therefore More Time is not needed for this purpose.

ETU Timer is is set to send null byte and increment more time counter, every time NULL byte is sent.

More times are sent at equal intervals, after receiving command.

### Optimised File System

Zeta is optimised for a maximum file system size. It is sufficient on this platform to support a file system below 56 KB (Zeta132K). Therefore the type definitions for EFs and DFs have been adapted in comparison with the latest ROM products on the Atlas product line. The size of the DF header was reduced from 38 to 24 bytes and the size of the EF header was reduced from 22 to 18 bytes.

Note: For Atlas v910 DF Header size is 32 bytes and for Zeta it is reduced to 24. For Atlas the pointer size is 4 bytes and for Zeta it is 2 bytes. DF Header contains such 4 pointers, so its size is reduced by 8.

#if #if VARIANT\_FILESYSTEM\_LARGE == 1

#define EF\_BODYPOINTER spCurEF->cpEFBody

#else

#define EF\_BODYPOINTER LibFilePhysicalAdd (spCurEF->uiEFBodyOff)

#endif

The base/reference point of the file system is set by the label FILESYSTEMSTART. The function LibFilePhysicalAdd calculates the offset to this address and returns the offset as an integer. The macro EF\_BODYPOINTER can be used in applications to replace the former spCurEF→cpEFBody.

**Every application has to be reviewed if spCurEF->uiEFBodyOff is used somewhere!**

## Initialization of RAM Variables

XDATA:

A routine to clear XRAM is added, ClearXRAM().This function is called in main after transmitting the first ATR byte, as it takes longer to clear 2K of XRAM, and within this time if first ATR byte is not sent, communication of card with reader fails.

**Note: Do not initialize any XRAM variable before clearing it with function ClearXRAM(). They should be initialized only in function LbhInitGlobalVariables().**

## Linker adjustments

While modifying the switches in init.h/variant.h, to enable/disable a feature, project may not compile at first place.

For e.g. In Basic-Tlk ini switch modified, to disable test applications as below,

#define TEST\_APPL 0

When project is compiled, it will give errors as follows,

\*\*\* ERROR L107: ADDRESS SPACE OVERFLOW

SPACE: DATA

SEGMENT: \_DATA\_GROUP\_

LENGTH: 000053H

\*\*\* ERROR L110: CANNOT FIND SEGMENT

SEGMENT: ?XD?NVMOTASS35

\*\*\* ERROR L110: CANNOT FIND SEGMENT

SEGMENT: ?XD?NVMOTASS35

\*\*\* ERROR L110: CANNOT FIND SEGMENT

SEGMENT: ?XD?NVMOTASS35

ERROR L110: CANNOT FIND SEGMENT: To remove this error remove following line from the linker file that is generated,

?XD?NVMOTASS35,

ERROR L107: ADDRESS SPACE OVERFLOW: This error is generated, as few functions from test applications are uncalled, and occupy global space in Data RAM.

Ignore this error and add application, this error will vanish once a call to all the functions that are in Data segment is made by application.

**While adding new function or an application to the OS please make sure that the code space and xdata space do not overlap. This can be verified using following table.**

**Code: Xdata mapping for THC20F17BD/THC20F17BD-V20/THC20F17BD-V30**

|  |  |
| --- | --- |
| Logical Address in Code Space | Equivalent Logical Address in Xdata Space |
| [B1:B000H – B0:FFFFH] | [X:2000H – X:6FFFH] |
| [B2:8000H – B2:FFFFH] | [X:7000H – X:EFFFH] |
| [B3:8000H – B3:8FFFH] | [X:F000H – X:FFFFH] |

In linker file, code space overlapping with XDATA space is reserved as follows,

RESERVE (B1:0xB000-B1:0xFFFF) //Reserve XDATA space in code

If new application code is added and it overflows in the XDATA region, then this reserve should also be extended accordingly.

For e.g., if new code added extends in Code Bank 1 till B1:0xB548, then above reserve space should be modified as follows,

RESERVE (B1:0xC000-B1:0xFFFF) //Reserve XDATA space in code

If not modified, compiler will generate overflow error in respective bank area as follows.

\*\*\* ERROR L134: ADDRESS SPACE OVERFLOW IN BANKAREA

SPACE: CODE/B1

SEGMENT: ?L?COM00B8

LENGTH: 000011H

\*\*\* ERROR L134: ADDRESS SPACE OVERFLOW IN BANKAREA

SPACE: CODE/B1

SEGMENT: ?L?COM00DA

LENGTH: 00000AH

\*\*\* ERROR L134: ADDRESS SPACE OVERFLOW IN BANKAREA

SPACE: CODE/B1

SEGMENT: ?L?COM00E3

LENGTH: 00000CH

\*\*\* ERROR L134: ADDRESS SPACE OVERFLOW IN BANKAREA

SPACE: CODE/B1

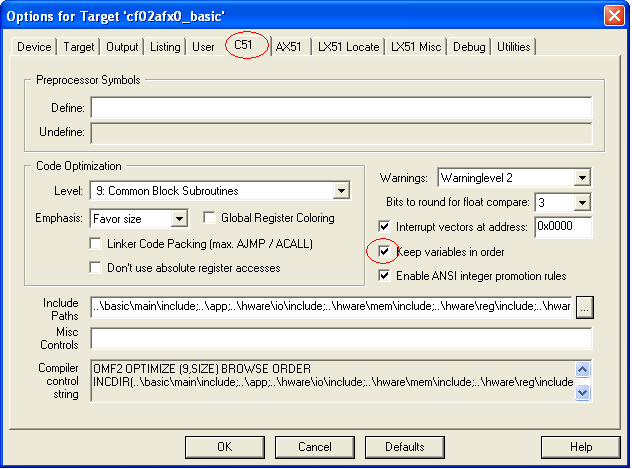
SEGMENT: ?L?COM00ED

LENGTH: 00000CH

If this error is observed, check all RESERVE settings in linker file.

## Linking Strategy of the Project

All source files are linked into the project according to their sequence given in the µVision project window. The variable order in a file is maintained by marking option as shown below,



For Zeta special pages have to be reserved due to the huge page size. This is due to a special precaution to protect important data. The strategy is to link such data into separate pages. Any erase/write action on the data can not corrupt other data in flash if a page/section is exclusively used for special data.

## Banking Modifications

Tongfang chips support code banks. Refer to above point 3.2, to know how to change banks. This setting helps us to locate the entire object file into a bank. So while adding care should be taken to relocate the object files in other bank.

Whenever a new application is added, this code should be placed in the last used bank, looking at the map file. While adding, if the bank overflows, place it in the next available bank. All this analysis can be done only with the help of map file.

Empty regions in the bank are reserved, if they are overlapping with the XDATA region. So do not forget to change this reserve area while adding new application code, if extended beyond this reserved area.

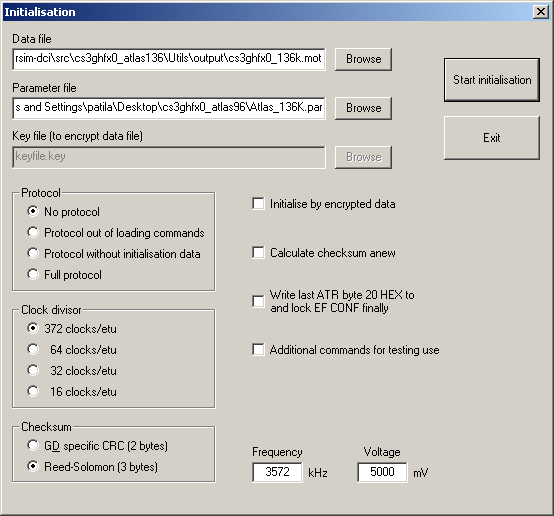
While switching off features of base ini, the code area occupied in banks is always modified, so do not forget to rearrange the source files, to get optimum possible free code memory.

Please do not modify code banks selected for object files of the base ini, until and unless it is really required.

## Card programming concept

The Initialize suit is updated to version 2.8 from 2.7. **Initialize2.8** supports the Tongfang boot loader for initialisation of Zeta 132. You can get the IFDSIM project from .\Tools\ Initialise\_Suit\TongFang\ Initialise.IFD.

Regarding clock divider only dividers 372 and 16 are supported (though is not a real divider 16 but a divider working in combination with CCR5 and IFDSIM).



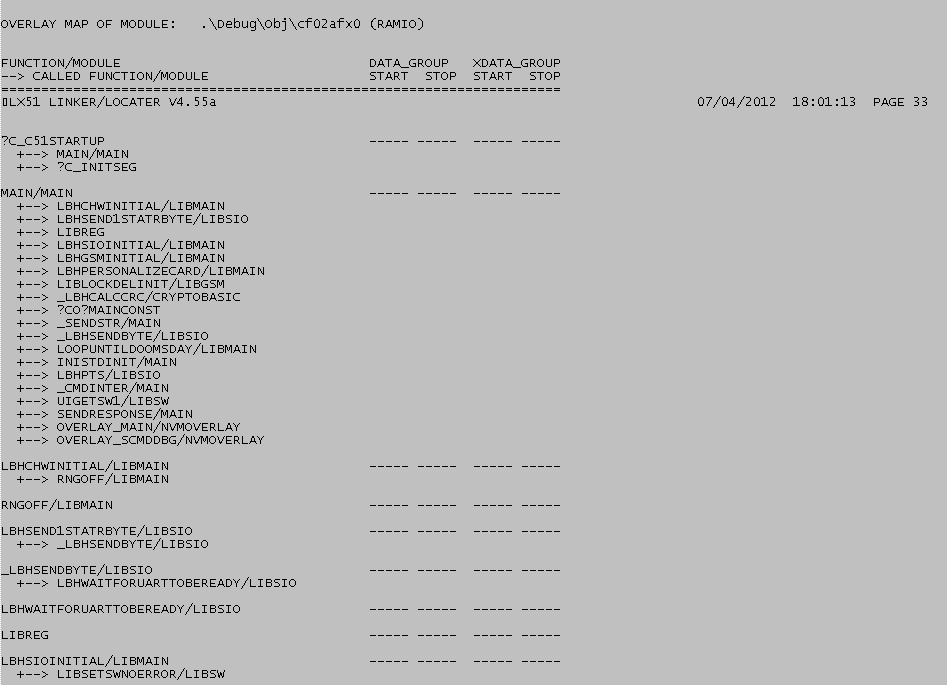
Loading is only terminated successfully, if the external Reed Solomon checksum (from S5 record of \*.mot file) is equal to the internally calculated checksum.

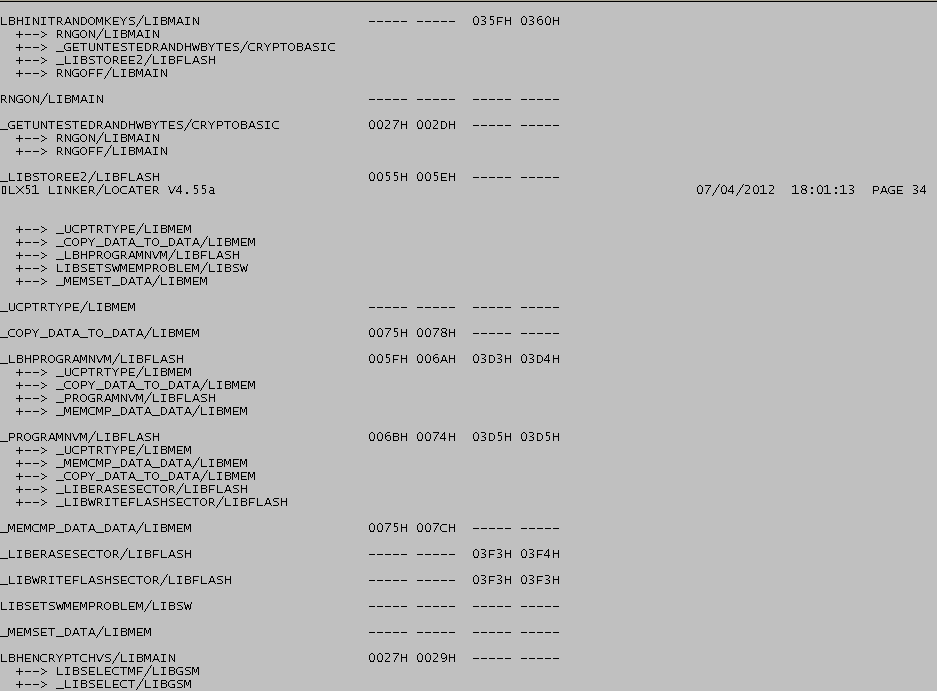
## OVERLAY technique

Tongfang Chips THC20F17BD/ THC20F17BD-V20/ THC20F17BD-V30 supports only 2K of XRAM, which is insufficient for porting larger applications like RFM-13, Celltick, WIB12, and WIB13-Lite. Zeta uses Memory Overlay to overcome RAM limitations. Memory Overlay helps to use RAM optimally.

### Memory Overlay Concept

Memory Overlay uses same block of data RAM for multiple functions which are completely independent for their execution. In order to preserve as much data space as possible, the linker performs call tree analysis to determine if some memory areas may be safely overlaid. This common data space is known as XDATA\_GROUP. We can check the correctness of this assignment by observing tree and data RAM assignments in map file. Linker generated overlay map looks as follows,





### Scope of Memory Overlay Implementation in Zeta

Function pointers are one of the many difficult features of the C programming language. Due to the unique requirements of a C compiler for the 8051 architecture, function pointers and reentrant functions have even greater challenges to surmount. This is primarily due to the way function arguments are passed.

Typically, (for most every chip other than the 8051) function arguments are passed on the stack using the push and pop assembly instructions. Since the 8051 has a size limited stack (only 128 bytes for Zeta), function arguments must be passed using a different technique.

When Intel introduced the PL/M-51 compiler for the 8051, they introduced the technique of storing arguments in fixed memory locations. When the linker was invoked, it built a call tree of the program, figured out which function arguments were mutually exclusive, and overlaid them.

### Memory Overlay Implementation in Zeta

**bit** boverlay;

A bit variable defined in file RamBitArea.c, which is never set. This variable is used to give dummy call to indirectly called functions.

#define OVERLAY\_SET 1

Switch defined in Sys.h to enable or disable overlay. This is applicable to the Targets which are built with µVision.

NVMOverlay.c

This file contains dummy call for the functions which are indirectly called using function pointers. The problem with function pointers occurs when the call tree cannot be correctly constructed by compiler. The reason is that the linker cannot determine to which function a function pointer references. There is no automatic way around this problem. However, there is a manual one, albeit a bit cumbersome.

The functions which are called using function pointers can be given dummy call, to correct the tree generated by linker.

E.g. Function CmdStatus () is called in Function main (), using function pointer as follows,

LbhCallFunc ((unsigned char code \*)(&(spCurCmd->fpCommand)));

Dummy call to the function CmdStatus () can be given in function main () as below,

#if OVERLAY\_SET

if (boverlay)

{

CmdStatus ();

}

Such all the functions in function table sCmdGSM are given dummy call in function Overlay\_main(), and this function is called in function main() as follows.

void Main()

{

while(TRUE)

{

#if OVERLAY\_SET

if(boverlay)

{

Overlay\_main();

}

#endif

}

}

Below is the table which defines dummy called functions, their respective tables and their callers.

| Called Function | Function Table | Caller |
| --- | --- | --- |
| Overlay\_sCmdDBG() | const struct CMD\_TABLE code sCmdDBG[4] | Main() |
| Overlay\_se2TlkApplCall() | struct TLK\_COMMAND\_TABLE code se2TlkApplCall[] | TlkSysApplParserMain() |
| Overlay\_main() | const struct CMD\_TABLE code sCmdGSM[22] | Main() |
| Overlay\_CmdTblMicroBrsRFM() | const struct CMD\_TABLE code CmdTblMicroBrsRFM[] | ucRFMCmdHdl() |
| Overlay\_sae2MBCmdTbl() | struct MB\_COMMAND\_TABLE code sae2MBCmdTbl[] | MBScriptParser() |
| overlay\_MB\_RFMtbl() | const struct CMD\_TABLE code CmdTblMicroBrsRFM[] | RFMCmdHdl() |
| Overlay\_PlugInTable() | struct strPlugInTable code PlugInTable[] | ucScrCmdPlugIn() |
| overlay\_MBCmdTable() | struct MB\_COMMAND\_TABLE code sae2MBCmdTbl[] | MBScriptParser() |
| Overlay\_PlugIns() | struct strPlugInTable code PlugInTable[] | CmdPlugIn |

**Note for Application:**  If you add any new function to the function table in column ‘Function Table’ please do not forget to add dummy call inside respective function in column ‘Called Function’.

### Challenges in using Memory Overlay in Zeta

Native OS switches between two sessions GSM and TLK. The problem of data corruption can occur only if a function in GSM session is overlaid with a function in TLK session and they have successive but partial execution. This can be eliminated by listing all the functions which have two instances of execution and assigning global RAM to their local variables.

Also if in runtime it is observed that a function in GSM session and TLK session are overlaid and they are corrupting each other’s data, then dummy call should be given to one function inside another function. Decision should be taken only if the corruption is confirmed by debugging.

Also to recursive functions and their sub functions, global RAM should be assigned or it can corrupt its own data and other functions data due to overlay. This can be achieved by excluding recursive function from overlay.

### Memory Overlay Syntax for Keil C51

Another way to include or exclude certain functions from call tree is adding them in linker file using below syntax.

Overlay syntax to include function in overlay tree:

?PR?MAIN?MAIN ! ?PR?\_COPY\_CODE\_TO\_DATA?LIBMEM

Overlay syntax to exclude function from overlay tree:

?PR?MAIN?MAIN ~ ?PR?\_COPY\_CODE\_TO\_DATA?LIBMEM

?PR?MAIN?MAIN:

Caller function

?PR?\_COPY\_CODE\_TO\_DATA?LIBMEM:

Function to be excluded from call tree of function main ()

### Debugging guidelines for memory overlay

Some of the most difficult types of problems to debug are what are generally called "memory overlaps". Following are some guidelines to debug overlay issues.

**Step 1**: Confirm whether it is overlay issue or not.

- Symptoms of overlay failure:

Failures are observed due to overlay during proactive session. If GSM command is fired during execution of some proactive TLK session, and if the two functions are overlaid, there are possible chances of corruption of RAM data.

For e.g. Envelope triggers a proactive session, and session is waiting for next Terminal Response to complete, between these if a CmdUpdate() is fired by ME, then if CmdUpdate() and waiting Tlk session commands are overlaid, then data of proactive session may be corrupted. This results in incorrect Terminal Response operation, and failure of Terminal Response.

**Step 2**: Finding function which when overlaid with current function causing corruption

-To debug this issue, put a breakpoint in code inside function CmdEnvelope(). Trace the complete flow, make list of ranges of all the variables used in the flow.

- Then trace the flow of interrupting CmdUpdate(), and check if any of the function in this flow uses same variable as in earlier flow.

- This will give you 2 functions which are overlapping and corrupting each other’s data.

**Step 3**: Once you know the functions which are overlapping, check their tree from root in corresponding OVERLAY MAP in map file. Check if the trees are complete and contains both the functions. If everything is ok, and still the functions should not be overlaid is precondition, then give dummy call to parent function of one of the two functions in parent function of the other.

# SCFXXXX family specific

## Generation of an Initialisation

In this chapter some basics for the development of initialisations/variants for Mizar are described.

### Preparation of an Initialisation/Variant

The Variant concept remains same as the version 913 of Mizar.

This new Mizar version offers all features like WIB1.2, WIB1.3-Lite, WIB1.3, WIB1.3-UltraLite,BasicRFM,RFM13,RFM13Lite,CELLTICK.,WIB13+Celltick. Now it is in the responsibility of the application developer to collect the correct sources together to provide an initialisation according to the customer’s requirements. Nevertheless, some preconfigured initialisations are available which serve as a starting point for application development.

Guidelines for application developers:-

To get the source code, fetch the basic source mask tree for Mizar 136 v914 from \StarSIM\_V914\_DCI\StarSIM\. Copy the directory “src” & Build\_Mizar to any drive you like, e.g. “D:\Mizar”. So now one should end with the following path: “D:\Mizar\src”.

To retrieve the source code for development within DCI, open Clear Case Version control tool and join project “StarSIM\_V914\_DCI” from UCM vob for StarSIM which is “StarSIM\_PVOB”

Once the view is created successfully, go to .\STARSIM\src\

Note: Sufficient access permissions of ClearCase are required for development purpose.

This \src folder already contains one folder for preparing initialisations , which is src\sys\IniConfigs\StarChip for Mizar 136KB initialisations.

StarSIM Mizar v913 is porting of existing Zeta 132KB OS on Starchip SCF136H platform. StarSIM Mizar V914 mainly aim’s for footprint reduction & feature additions to Mizar.

Folder structure remains same as Zeta we need to do only platform dependant, compiler specific changes to files such as init.h, chwcon.h and many more.

In Zeta there are project files and batch jobs for basic as well as all other applications. In Mizar the procedure is somewhat different , there are no by default project file available for compilation. Compilation is based on make files. We need to create c project in Eclipse IDE for source browsing and debugging purpose. We need to create make file for specific target in \build\makefiles. Global rules for making targets are included in \build\makefiles\includes.mak. Target specific rules can be included in \Build\_Mizar\makefiles\cc21afx0\_XXXXX.mak where XXXXX is name of Target/Initialisation to be created. There is a “Makefile” present in each target directory which will create copy of two files e.g. Basic basic-init.h into init.h,.h, variant-basic.h into variant.h.. There is a common linker file present at \Build\_Mizar\linkerfile\ which is “linkscript\_common.ld” used to define linker and MAP sections. This linker file can updated for any new target/feature under the respective feature switch.

If you have chosen to create an initialisation starting from one of the default targets you have several already prepared initialisation for a choice. One simply has to execute the make file of respective target using cygwin. We can go to target folder on cygwin prompt and say ‘make’ which will compile the source for that target. The correct variant.h, chwcon.h and init.h are automatically provided if the appropriate make file is used.

**Check that you have the correct init.h, variant.h**

After compiling such a target it is possible to get the compilation output eg. Build log/mot file/lint log etc. under specified target directory itself in respective folders like /images or /buildlog.

Preconfigured initialisations for Mizar 136:

* BASIC: Plain initialisation without application
* BASIC RFM: initialisation with basic RFM only
* BASIC OTASS3.5 : Plain initialisation with OTASS3.5 feature enabled. CreDel and MD feature disabled.
* BASIC MD, No CREDEL reclaim: Plain initialisation with MD feature enabled and only MD space reclaimed for ADN File.
* BASIC MD+CREDEL reclaim: Plain initialisation with MD and CREDEL features enabled and CREDEL space reclaimed for ADN file.
* Celltick 14.2.6 : initialisation using Celltick14.2.6 only
* Celltick 14.3.1 : initialisation using Celltick 14.3.1 only
* Celltick 14.3.1 RFM,MD+CREDEL reclaim : initialisation including Celltick 14.3.1 , Basic RFM with MD and CREDEL features enabled and CREDEL space reclaimed for ADN file.
* RFM13: initialisation with RFM13 only
* RFM13Lite: initialisation with RFM13 only
* WIB12: initialisation including Basic RFM and WIB1.2 browser
* WIB13-lite: initialisation including RFM13 and WIB1.3 browser
* WIB13: initialisation including RFM13 and WIB1.3 browser
* WIB13 MD+CREDEL reclaim: initialisation including RFM13 , WIB1.3 browser with MD and CREDEL features enabled and CREDEL space reclaimed for ADN file.
* WIB13-UltraLite: initialisation including RFM13 and WIB1.3 browser
* WIB13- Celltick 14.2.6 : initialisation including RFM13 , WIB1.3 browser and 1 Celltick 4.2.6
* WIB13- Celltick 14.3.1 : initialisation including RFM13 , WIB1.3 browser and Celltick 14.3.1
* WIB13- Celltick 14.3.1 MD+CREDEL reclaim - : initialisation including RFM13 , WIB1.3 browser and Celltick 14.3.1 with MD and CREDEL features enabled and CREDEL space reclaimed for ADN file.

Note that MD/CREDEL feature is enabled for the initialization mentioned as above.

### Linker file & Sections:

GCC uses linker file (.ld file) for defining how the memory is to be utilised. The sections and memory allocation would get modified with the changes carried out in the code, hence always ensure NVM modifiable region is always placed in the last. Map file is useful to view the placements, it would show all the placements as per the region it has got allocated.

While porting the existing code please check and verify the section definitions in the code and the placements shown by Map file are as per expectations.

Linker file can be referred from directory /Build\_Mizar/linkerfile. There is a common linker file used for all the available targets and sections are defined as per feature available. Eg. For a particular target like WIB12 required NVM sections are made available under variant switch of WIB12 in common linker file.

Linker file uses various commands to serve its purpose.

- MEMORY : The linker's default configuration permits allocation of all available memory. You can override this by using the MEMORY command.

The MEMORY command describes the location and size of blocks of memory in the target.You can use it to describe which memory regions may be used by the linker, and which memory regions it must avoid.

- OUTPUT\_FORMAT : This command names the BFD format to use for the output. Eg OUTPUT\_FORMAT("aps3").

- PHRDS : The linker will create reasonable segments by default. PHRDS command can be used to specify the segments more precisely. When the linker sees the PHDRS command in the linker script, it will not create any segment other than the ones specified.

- SECTIONS : This command is used to describe the memory layout of the output file.

These are few examples of linker commands.

#if ( VARIANT\_RFM == 1 || VARIANT\_RFM13 == 1 )

.NvmRFM :

{

KEEP(\*(.NvmRFM .NvmRFM\*));

. = ALIGN(4);

} > program\_memory :nvmdata=0x00

#endifAbove example shows the placement of variables under NvmRFM section.

By specifying = ALIGN(4); we can allow linker to align particular section to 4 byte address.

The variables that shall go under particular section through linker file shall be specified with the attribute in their definition/prototype.

Eg. The variable uce2RFM\_MoreTime is provided with section attribute as below and shall get placed in “.NvmRFM” section.

extern volatile unsigned char eeprom uce2RFM\_MoreTime \_\_attribute\_\_((section (".NvmRFM")));

For more details on Different linker commands and their purpose refer [12].

### NVM File System

In Mizar all .s files has been ported to .c files. The default file system is present at NVMFileSystem.c file. This file has to be aligned. It is taken care in linker file as below,

.defaultfilesystem FILE\_SYSTEM\_START:

{

PROVIDE(\_defaultfilesystem\_start = .);

. = ALIGN(\_SECTOR\_SIZE);

KEEP (\*(.efic));

KEEP (\*(.efchv));

KEEP (\*(.efcfg)); ………

……… }

**File System Header Start Address Alignment**

The NVMFileSystem.c contains default file system as part of the init. The start of the file system is ensured to be page aligned. Also after the default file system ends , file system headers will starts which also has to be page aligned.

This alignment is taken care in linker file by adding padding section at end of default file system as shown below.

.defaultfilesystem FILE\_SYSTEM\_START:

{

. = ALIGN(\_SECTOR\_SIZE);

KEEP (\*(.efic));

KEEP (\*(.efchv));

KEEP (\*(.efcfg));

……………

……………

KEEP (\*(.bodyCHV1));

KEEP (\*(.bodyCHV2));

KEEP (\*(.bdyubchv2));

KEEP (\*(.bodyCHV4));

KEEP (\*(.bodyCHV5));

KEEP (\*(.padding));

} > program\_memory :nvmdata=0x00

**Secure Write Page Area Start Address**

Secure write buffers used for protected PIN/counters writing are placed in separate page. The Secure Write Page Area is ensured to be page aligned through linker setting as below.

.secwrbuf :

{

. = ALIGN(\_SECTOR\_SIZE);

KEEP(\*(.secwrbuf .secwrbuf\*));

. = ALIGN(\_SECTOR\_SIZE);

} > program\_memory :nvmdata=0x00

### Determination of the File System Size

To determine the feasible file system size one has to look into the map file.

To calculate the feasible file system size, find the difference between the macro RECLAIEM\_AREA\_START\_ADR and .defaultfilesystem. The RECLAIEM\_AREA\_START\_ADR macro is available in the init.h of the corresponding target. Whereas the .defaultfilesystem is available in the map file. The final value which can be assigned to the macro FILESYSTEM\_SIZE can be calculated as

FILESYSTEM\_SIZE =

( RECLAIEM\_AREA\_START\_ADR – defaultfilesystem –

MANDATORY\_FILE\_SYSTEM\_SIZE )

There is a limit on the maximum filesystem size that can be handled by the OS which is 0xFE00 at the moment. Hence the following sum should not exceed the limit of 0xFE00.

( FILESYSTEM\_SIZE + MANDATORY\_FILE\_SYSTEM\_SIZE +

RECLAIM\_AREA\_SIZE ).

Otherwise the build system will give the error. For more details please refer the section [Error messages shown by the build system](#_Error_messages_shown)

### System Files

The map file of the project will be created in ..\Build\_Mizar\ TARGET\_NAME\images. The jtag file of the project is created in ..\Build\_Mizar\ TARGET\_NAME\images. Post build scripts are executed on jtag file to generate final mot file TARGET\_NAME.mot in Motorola format and TARGET\_NAME.blc in BLC format. The CRC-CCITT checksum information is embedded as S5 record of the mot file. The TARGET\_NAME.blc has to be delivered to production.

The final flash initialisations are created in ..\build\TARGET\_NAME\images folder.

For e.g. for BASIC Initialization,

*Project Files:*  NA.

*Map File:*  cc21afx0\_BasicTLK.map

*Target File:*  cc21afx0\_BasicTLK.jtag, cc21afx0\_BasicTLK.BLC, cc21afx0\_BasicTLK.mot

**Note: The file to be delivered for production cc21afx0\_BasicTLK.BLC!**

### 

### Batch Files and Executables

The following batch files and executables are used during Flash generation. To run the batch jobs the following files must exist:

* \StarSIM\Build\_Mizar\Tools\CCitt\_Crc16\_8404.exe, GenerateMotFile.bat, MotToBlcCOnvertor.pl, objcopy.exe
* \StarSIM\Build\_Mizar\scripts: MizarVariableInCode.pl, Mizar\_Header\_CRC.pl

**CCitt\_Crc16\_8404.exe:**

This executable is used mainly for calculating the CRC on the mot file. Same executable also removes the blank records from the output file. It accepts the jtag file as an input argument and then it generates the mot file at the same location. This executable has a dependency over the location from where it is operating. According to that location it access another executable named objcopy.exe.

Following are the steps for generating the output mot file which contains the S5 record at the end which has the erase information and the crc verification for the mot file.

* Read the argument which represents the jtag file
* Using objcopy.exe, convert the input jtag file into Motorola S3 record with record length 0x80
* Generate the binary image of the S3 file.
* Remove the blank records from the file and save it.
* Using objcopy.exe increase the record size from 0x80 to 0xF0
* Compute the S5 record having erase information and the Crc information for the chunks in the flash.
* Save the output file as mot file.

**GenerateMotFile.bat**

This batch file is used for calling the Ccitt\_Crc16\_8404.exe for the execution. It passes the target name jtag as an argument to the CCitt\_Crc16\_8404.exe. It also removes the unused files which are generated at the build time.

**MotToBlcCOnvertor.pl**

This perl script file is used for converting the mot file into a blc file. The blc file contains the every APDU of the bootloader followed by respective data from the mot file. It also decodes the S5 record of the Mot file and generates the information for the erase APDU and the verify APDU of the bootloader. At the end both the Mot and Blc files are maintained in the output folder just to maintain the backward compatibility.

**objcopy.exe**

This executable is the part of standard GCC compiler. It is used for converting the compiled output files in to various formats. For example it can be used for converting the jtag file in to a mot file. It also provides the flexibility of changing the record size while converting the file.

**MizarVariableInCode.pl**

This script finds if any RAM Variable is falling into Flash area and vice versa. In case the script finds such instances, it throws warning.

**Mizar\_Header\_CRC.pl**

This generates and patches information like System area CRC, File Header CRC and information like link to next element, offset to file body.

Since some of the fields like offset to body of file, link to next element are generated only after compilation, there is 2 stage compilation carried in

below steps.

1. The init\_crc.h initially contains dummy CRC and other field values.
2. The first round of compilation takes place with dummy values. Intermediate binary output files are generated which does not contain the CRC, Offset or link values.
3. The script operates on the binary (mot file) and generates CRC and other information and all the dummy values in init\_crc.h header file are replaced with this actual new values.
4. Since the 2 files NVMFileSystem.c and NVMIni.c needs the above changed values, only these are recompiled in second stage and linked to get the final binary output in BLC and mot format.

Note: It was possible to generate the final output MOT file with only one phase compilation. But the CRC would fails while debugging on emulator as the JTAG would not have this information.

System Area CRC:

The System area CRC in the linker file should be specified like below:

.NVMSystemArea :

{

KEEP (\*(.NvmSysAreaCRC));

KEEP (\*(.NvmIniSecConData));

KEEP (\*(.NvmIniATR\_TA1Data));

KEEP (\*(.NvmIniConfCreaDelData));

KEEP (\*(.NvmIniDummyFillByteData));

} > program\_memory :nvmdata=0x00

where

.NvmSysAreaCRC is the location where the Start CRC value is mentioned.

.NvmIniSecConData is the start byter of the System Area CRC

.NvmIniDummyFillByteData is the last byte of the System Area.

Note that the order of the variables must be strictly in above order for the script to correctly calculate the System area CRC.

CRC and other information patching is performed to headers of following files:

1> EFIC

2> EFCHV

3> EFCFG

4> EFDELKEY (Optional: Present in Init with Delete OS with Authentication)

5> EF MF

Note that the layout of the concerned files in the NVMFileSystem.c must be preserved for the script to correctly generate CRC and other information.

In case of CRC failure, error ATR is given.

**ModifyRecordS0.pl**

This perl script is used for modifying the S0 record as per the production requirement. The S0 record reflects the status of the OTC code wiz MD and CreDel. Using the S0 record the UPPC tool at the production is able to understand the capability of the given Ini. For example, using S0 record it is possible to understand if the ini support the MD feature or not. The same script is also used for inserting the MD separator flag named [MasterDevice patch]. This script also removes the erase sector commands of at the beginning of the BLC file. Then it replaces a single full chip erase co

## Porting Assembly Code

For porting assembly code to new compiler based inits we need to take care of how data would be exchanged between the C and assembly code

For Mizar, the microcontroller type is 32-bit Cortus core. For porting assembly code from Zeta to Mizar reference is taken from Zeta code base.

## Application Notes

### More Times on Mizar

ETU Timer is set to send null byte.

Incremented the more time counter, when each NULL byte is sent.

More times are sent at equal intervals, after receiving command.

More time for DES calculations is handled through the OS.

### Optimised File System

Mizar is optimised for a maximum file system size. It is sufficient on this platform to support a file system more than 64 KB (Mizar 136K) but it is limited in OS to 64 KB only, after claiming MD and CREDEL.

The size of the DF header 32 bytes and the size of the EF header is 18 bytes.

Note: For Zeta DF Header size is 24 bytes and for Mizar it is 32. For Zeta the pointer size is 2 bytes and for Mizar it is 4 bytes. DF Header contains such 4 pointers, so its size is increased by 8.

#

#if VARIANT\_FILESYSTEM\_LARGE == 1

#define EF\_BODYPOINTER spCurEF->cpEFBody

#else

#define EF\_BODYPOINTER LibFilePhysicalAdd (spCurEF->uiEFBodyOff)

#endif

The base/reference point of the file system is set by the linker setting. The function LibFilePhysicalAdd calculates the offset to this address and returns the offset as an integer. The macro EF\_BODYPOINTER can be used in applications to replace the former spCurEF→cpEFBody.

**Every application has to be reviewed if spCurEF->uiEFBodyOff is used somewhere!**

## Initialization of RAM

The complete RAM of size 3.5K is cleared before calling the main function in startup routine. This is done by writing a loop in the function \_startup. This is a for loop which operates on the entire RAM from highest address till the lowest one. The last address is made zero at the end of the for loop.

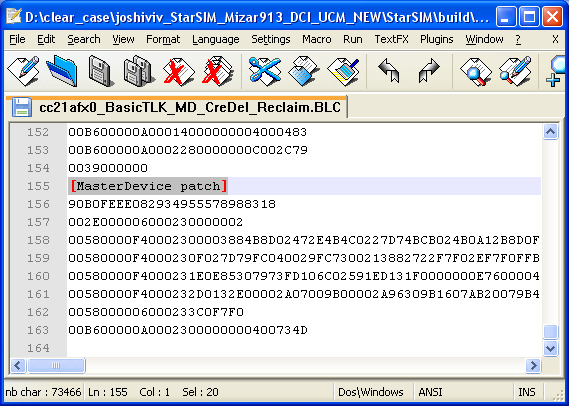
## Loading Concept

The loading concept for Mizar is same as Zeta.

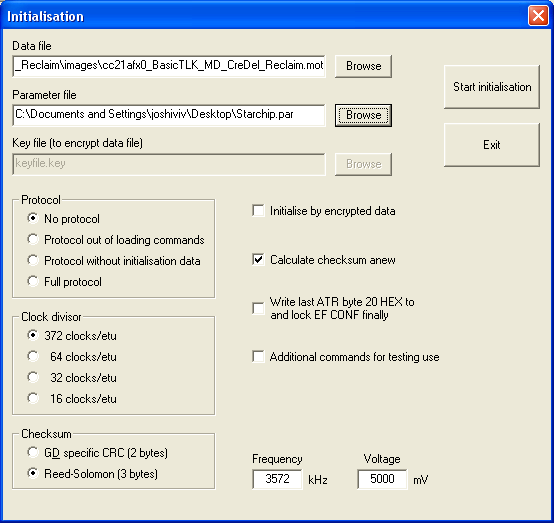
There is additional support for loading the project using the BLC file. Also it is possible to load the MD patch present in the same BLC file using the same initialization tool.

The bootloader supports the dividers 372, 64, 32, 16 and 8.

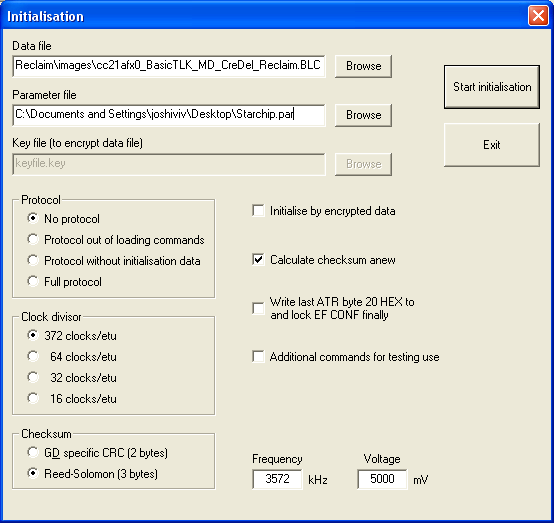
The MD patch with the BLC format is same BLC init file appended with the MD APDU and a key separator as “[MasterDevice patch]”. The patch does not have its own S0 record.



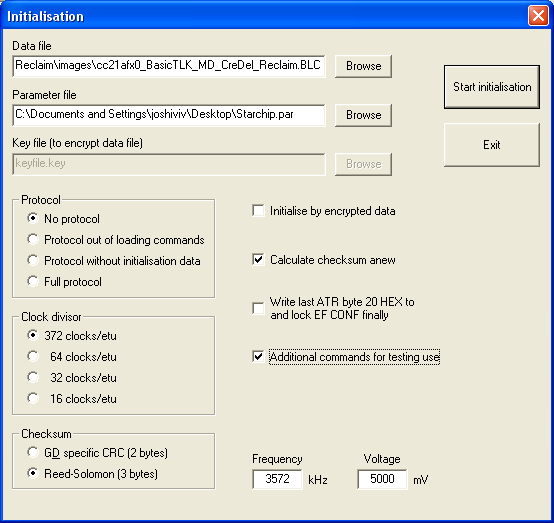
Following screenshot shows the way to flash the init using mot file.



Following screenshot shows the way to flash the init using mot file.



Following screenshot shows how to load the MD patch on the card having OS loaded on it.



* **Loading OS to card**

All the Starchip cards are loaded with the G&D proprietary bootloader which is specifically designed for Mizar. Hence the steps involved for loading the OS on the card are as per this bootloader.

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Status** | **Notes** |
| 0 | Power On | BL is activated now. Any card reader compatible with ISO7816 standard can test the card with BL commands. |
| 1 | Erase the entire chip by providing the full chip erase command | Send the command APDU  002E000006 00002000010D |
| 2 | Load segments by sending write commands | Send the command APDU  00580000F0 00002000XXXXXXXXXX |
| 3 | Verify the written data by sending the verify command | Send the command APDU  00B600000A0000200000007E00F95E |
| 4 | Finalize the chip for switching the control from bootloader to OS | Send the command APDU  0039000000 |
| 5 | Reset the card | The OS ATR should be visible now. |

# New Release Versions and Patches

The StarSIM V914 version number is as follows: MAJOR.MINOR.REVISION

The mask version will be incremented by one MAJOR every time a new major release is made. The mask version will be incremented by one MINOR every time a new minor release is made.

All StarSIM V914 masks will be released in major release steps.

The current release version is v914. The ATR indicated by version v914 on all platforms.

Please note that the ATR will NOT be adapted for new release revisions, it will always indicate MAJOR & MINOR version only. E.g. v914

The patches will be provided in a PGP file. The current directory structure of the source code will be maintained. The patch will contain only the patched source files. The list of patch files and relevant information is shared in an excel sheet. The patch is uploaded to the ePlanet/ClearCase. The link is updated on StarSIM WIKI page.

# High Update Activity Files

The exceptionally large page size of 512 bytes and the limited number of write cycles poses a major problem regarding the reliability and security of data on the SIM. Especially the high update files are affected by these limitations. The high update handling for Mizar is same as Zeta. As like Zeta, Mizar also supports page alignment introduced in Atlas V910. For page alignment feature of Atlas V910 please refer to document REF [2].

## Updating a High Update Activity EF

To update an HU file, follow a procedure similar to following code snippet. The following snippet updates file through using UpdateData() function call, which takes care of writing the status bytes as required by the HU files.

Set the P1, P2 & P3 parameters and make a call to UpdateData(). Before calling UpdateData() CheckEF() call is mandatory. The argument for CheckEF() may vary as per your use case. In this case Help buffer is being used to send the data.

*LibSelect(ID\_File);*

*if ( uiGetSW1()== SW1\_NO\_ERROR )*

*{*

*memcpy(ucaHlpBuf, LocalBuffer, NoOfBytes);*

*ucP1 = 0x00;*

*ucP2 = 0x00;*

*ucP3 = 0x03;*

*CheckEF(TRANSPARENT + MOD\_UPDATE);*

*UpdateData(MOD\_FROM\_HLPBUF, 3);*

*}*

## Page Alignment of Files

Certain file data which are vital for the functioning of the SIM (e.g. EFKC, EFIMSI) must be kept in separate pages onMizar. ForMizar, even if the physical page size is 512, logical page size is kept same as Zeta, i.e. 256 bytes, to achieve minimum changes in the OS design. Just imagine a file is updated, which is located next to one of the vital files, and the current is interrupted, although the page has already been erased. In such a case all data of this page are lost and the SIM would be destroyed.

Therefore such data must be kept in separate pages or must be stored in a page where only files are stored without any data update.

In order to achieve this, a new feature was introduced. It is now possible to create files page aligned. This means that the new file created will be placed into the next free page.

### Aligning file Bodies to Page Boundaries

The CREATE FILE command supports the alignment of file bodies to page boundaries. This feature is controlled by Bit 6 in Byte 12 of the command parameters:

* 0 = the file body will be created adjacent to the previous file body
* 1 = the end of the file body will be aligned to the end of the next free page

This alignment can be applied to all file types and high-update as well as low-update files.

## HU File Creation Using Profiler

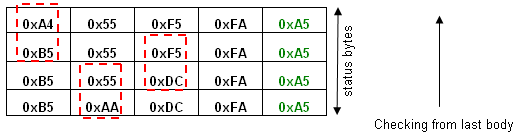
When using the profiler for creating HU files, we request you to ensure the file created is always of even address. This does not affect the functionality but this is required for matching the size of profiler free memory and card free memory.

## Transparent Files

Transparent EFs with high update activity are created with four file bodies.

### Updating a High Update Activity Transparent EF

1. Copy the whole body to an intermediate RAM buffer.
2. Update the RAM buffer with new file body contents.
3. To identify the present active body, a check to find 0xA5 from the last body towards the first body is made. If the next body’s status byte is 0xFF and the current body’s status byte is 0xA5, the current and next bodies are identified.
4. Point to next empty body.
5. Set last byte of file body flag to 0xA5 and make it active
6. If no active body found or its last body, erase page, select first body, set flag to 0xA5.
7. Copy the contents of the RAM buffer into the (new) active body.



Here is an example of EF File body of size 4 bytes. The status byte occupies 1 byte.

### Reading a High Update Activity Transparent EF

1. Find active body starting from last body and read data
2. If no active body found, return 0xFF.

## Cyclic Files

The implementation of cyclic files is same as Zeta v913/Mizar v913.

### Updating a Cyclic file

As per existing implementation, here also we have M=2N records. Update is done starting from higher address(rec N) to lower address(rec 1).The updating of the EF is principally done in the following steps:

1. Get present N records index set. i.e. lower half or upper half index set of file body
2. Find the current active record.
3. If active record found point to next empty record.

Copy the whole body to an intermediate RAM buffer.

Update the RAM buffer with new file record contents

Set flag of newly updated record to 0xA5 and make it active.

1. If it is (N-1)th Record is found to be active then erase the other half of N records.

Copy the whole body to an intermediate RAM buffer.

Update the RAM buffer with new file record contents

Set last byte of record flag of newly updated record to 0xA5 and make it active.

1. If Nth record is found to be active then point to next empty record in next M-N space.

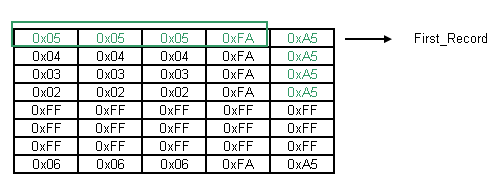
Copy the whole body to an intermediate RAM buffer.

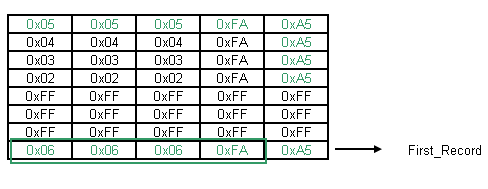
Update the RAM buffer with new file record contents

Set last byte of record flag of newly updated record to 0xA5 and make it active.

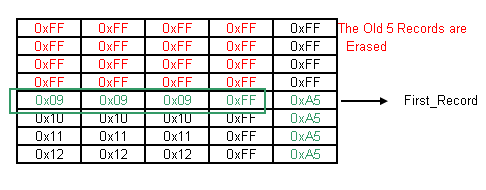
1. Copy the contents of the RAM buffer into the (new) active body.

Illustration of Cyclic files update



Here is an example of a Cycle file of which records of the file is 4 and created records are 8. Each Record is of size 4. When a record is updated it is written as follows:

When the record to be updated is the 8th Physical record then the oldest 4 records are erased as there are no more valid.



### Reading Cyclic Files

Reading cyclic files are done by identifying the latest updated record and from that record taken as reference to read all the other records.

Illustration of Cyclic files update buffer

First logical record

First physical record

Active records

Active records

Erased records

xx…..00

FF…..00

FF…..00

FF…..00

FF…..00

FF…..00

FF…..00

FF…..00

FF…..00

xx…..00

xx…..00

xx…..00

xx…..00

xx…..00

xx…..00

xx…..00

Not active any more

xx…..00

xx…..00

xx…..00

xx…..00

**Example: Cyclic EF with N = 10, M = 20**

## Summary

High Update File Changes:

The below differences wrt Atlas v910 are kept only for reference purpose.

|  |  |
| --- | --- |
| **(StarSIM v914)**  **Tongfang**  **Starchip** | **(Atlas v910)**  **Samsung 90nm** |
| The maximum file size of transparent high update file is 62 bytes. When creating a high update transparent file, a 1 byte status is added to each body making the size to 63 bytes.  Four such bodies are created for high update operation and to reduce the erase/write cycles. Therefore the total size is 63\*4=252 bytes. This fits in one NVM page of 256 bytes | The maximum file size of transparent high update file is 62 bytes. When creating a high update transparent file, a 4 byte status word is added to body if file size is in multiples of 4,  If file size is not in multiples of 4 then file size is aligned in multiples of 4 and last byte is decided to be as a status byte. This allows us the flexibility for allocating the maximum file size up to 64 bytes.  Four such bodies are created for high update operation and to reduce the erase/write cycles. Therefore the total size is 64\*4=256 bytes. This fits in one NVM page of 256 bytes |
| The maximum record length of a cyclic file is 254 bytes. The number of records are calculated as  N = abs (File size/Record length)  A 1 byte status is added to each record.  When creating a cyclic file, a file body of double the number of records is created for cyclic operations and to reduce the erase/write cycles. | The maximum record length of a cyclic file is 254 bytes. The number of records are calculated as  N = abs (File size/Record size)  A 4 byte status word is added to record if record size is in multiples of 4. If record size is not in multiples of 4 then record size is aligned in multiples of 4 and last byte is decided to be as a status byte.  When creating a cyclic file, a file body of double the number of records is created for cyclic operations and to reduce the erase/write cycles. |
| Memory calculation of file bodies:  For high update transparent files:  Size = (M+1)\*4 where M=1 to 62  For cyclic files:  M = Record Length = 1 to 254  Size of 1 record (L) = M + 1  N = File Size/L  Size = 2N \* L | The body size must be 4 byte aligned to avoid erase and write operation.  Memory calculation of file bodies:  For high update transparent files:  K = M%4 where M = 1 to 62 (file size)  If K > 0  Size = ((4-K)+M) \* 4  Else  Size = (M+4)\*4  For cyclic files:  K = M%4 where M = 1 to 254(record Length)  If K > 0  Size of 1 record(L) = ((4-K)+M)  Else  Size of 1 record(L) = (M+4)  N = (File size/L)  Size = 2N \* L |
| No additional memory is reserved internally | The write operations must start at a 4 byte aligned address to avoid erase and write operation. Therefore, when creating a high update or cyclic file, if the start of the file body is not at an aligned address, it will reserve up to 3 bytes internally to align the file body. These reserved bytes will be recovered when the file is deleted. |

# Debug Commands

These are special commands introduced for debugging the card behaviour.

## Command Check Stack

This function checks the C-Stack and the application stack usage. If insufficient stack memory in either of these is detected an error status word is returned.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Command | Class | Ins | P1 | P2 | P3 |
| Check Status | B0 Hex | AE Hex | 00 Hex | 00 Hex | XX |

**P3:**

0x06 – Return size of free stack

0xFF – Return dump of C stack

Possible responses:

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Error description |
| 90 | 00 | Normal ending: The stack size is returned |
| 67 | 00 | Incorrect P3 |
| 92 | 10 | Not Enough Memory |

## Command Timing Test

This function is used to measure timings of Crypto Algorithms (Single and Triple DES) and Sending of NULL bytes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Command | Class | Ins | P1 | P2 | P3 |
| Timing Test | B0 Hex | BA Hex | XX | XX | 00 |

**P1:**

0x01 – Measure timings for Single DES

0x02 – Measure timings for Triple DES

0x03 – Measure timings for LION Bytes

Possible responses:

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Error description |
| 90 | 00 | Normal ending: The measured timings are returned in response |

## Command Reset Run GSM Flag

This function resets the flag which indicates that the ADM counter has been used before.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Command | Class | Ins | P1 | P2 | P3 |
| Timing Test | B0 Hex | AC Hex | 00 | 00 | 00 |

Possible responses:

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Error description |
| 90 | 00 | Normal ending: RUNGSM Flag is reset |

## Command Check Concat Buffer

This function gives free space available in Multiapplication concat buffer.

This command is applicable for WIB-Celltick only when RB\_G\_RB\_USED flag is ON as WIB-Celltick uses multiapplication concat buffer.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Command | Class | Ins | P1 | P2 | P3 |
| Check Concat Buffer | B0 Hex | DD Hex | 00 | 00 | 02 |

Possible responses:

|  |  |  |
| --- | --- | --- |
| SW1 Hex | SW2 Hex | Error description |
| 90 | 00 | Normal ending with 02 bytes of free size are sent. |

# Appendix

## How to debug

### Family THC20F17BD-XX

#### Emulator Setup

Emulator SCDS200 received from Tongfang is USB version. USB version has connection problems with different brands of computers. It connects only with some computers, last observed with Dell Computer. To overcome this compatibility issue of USB version, SCDS emulator version should be changed to HID.

After first installation, emulator should be updated to HID version. Procedure to install the emulator setup is described below.

Download updated Toolchain setup from below path.

<http://eplanet.gi-de.com/portal/page/portal/MarketsAndSolutions/Technologien/Card%20related%20Technologies/Tools1/Tools/TongFangTools>

Refer to Installation manual above Ref [7].

USB Version:

\Toolchain\USB\setup\Setup\ Setup.exe

HID Version:

\Toolchain\HID\setup\Setup\ Setup.exe

For USB setup, drivers for USB should also be installed after emulator setup is successfully installed.

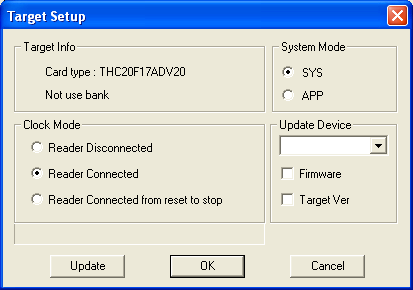
Path for USB drivers for SCDS2000 is as below,

C:\Keil\C51\SCDS\driver\lpc214x.inf

Once the setup is completed and the emulator is connected the LED labelled as LINK on the emulator blinks. This LED confirms the emulator connection with the PC. Then open existing project, select Debug label. Select SCDS2000.



Push “Settings” button, will begin to setup Target



Target Info: Shows target information

Clock Mode: Select Reader Connected mode.

System Mode: Select SYS mode of operation

**Note: Confirm these setting, after opening the project every time, as they do change, this may lead to no response from the card.**

### Firmware update

For Probe board settings please refer to the document “Zeta CHL-2.doc”.

Emulator SCDS200 received from Tongfang is USB version. USB version has connection problems with different brands of computers. It connects only with some computers, last observed with Dell Computer. To overcome this compatibility issue of USB version, SCDS emulator version should be changed to HID. Following Steps should be carried out for firmware change from USB to HID (Ref [15]).

Step1: Install Emulator setup, follow procedure in above 4.6.1.

Step2: Copy files under “setup/binFirmware/” to SCDS directory on your PC at following path.

C:\Keil\C51\SCDS

Open an existing keil project, select “project / Options for Target / Debug”, and then select “SCDS2000”. Press button “Settings”, the following “Target Setup” window will appear. Please select the same card type with your current card (Figure 1.1).

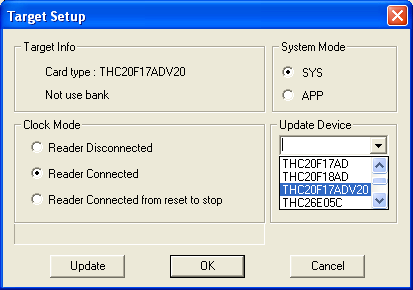


Figure 1.1: Target setup

Select update Firmware

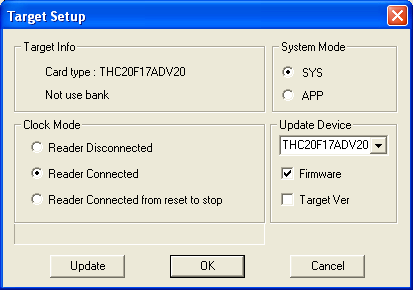


Figure 1.2: Only update firmware

Press “Update” to start updating. As to avoid unintended operation, the following warning dialog will appear. Please press “Ok” to confirm updating (Figure 1.3).

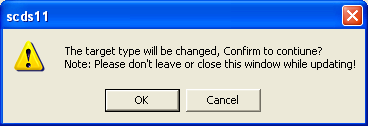


Figure 1.2: warning dialog

Step 3: This emulator is now ready to be used as HID. Please install the HID setup on the computer with which this emulator will be connected. If USB setup is already installed, then uninstall it and then install HID setup.

### Target update

This option is provided to change the Target device from any device to THC20F17BD/ THC20F17BD-V20/THC20F17BD-V30 and so on.

Please refer to section [6.1](#_Product_family_THC20F17BD-XX) for details about updating Target Type on Emulator.

### Loading Emulator

Before one starts debugging it is mandatory to check the settings for Target Info, Clock Mode and System mode as mentioned in section 4.6.1. If any of the options are not correct, card may not respond.

**It is strongly recommended to delete all H/W breakpoints before you stop debug**

**ging!**

Note:

1. It sometimes has been observed that a hex file was not downloaded correctly into the emulator. In this case the paddle was probably inserted into the CCR5 and the IFDSIM was switched on. Under such circumstances the paddle draws current from the CCR5 and thus the downloaded hex file does not overwrite the contents in the emulator which have been downloaded before. Please release the paddle from the CCR5, reset probe board and emulator, insert the paddle again and switch the IFDSIM on. Then download the hex file.
2. When debugging is started on emulator, it takes longer to load the hex file to emulator; reason being flash is cleaned before loading OS to emulator, Tongfang Bootloader image is loaded to the emulator. This is done with batch file Load\_CRC\_Data.ini.

### Family SCFXXXX

### Using the Emulator

For using the StarChip emulator for debugging the code, one has to get the Cortus IDE installed on the machine. The debugging of the code is done using the same IDE. For debugging the code one has to build the project through bash with the EMU parameter passed as one. For more details please refer the section [Options implemented with make command](#_Options_implemented_with).

After getting the code compiled, one has to set the debug environment for the emulator. This has to be done in the IDE itself. Please refer the following screenshots for more details.

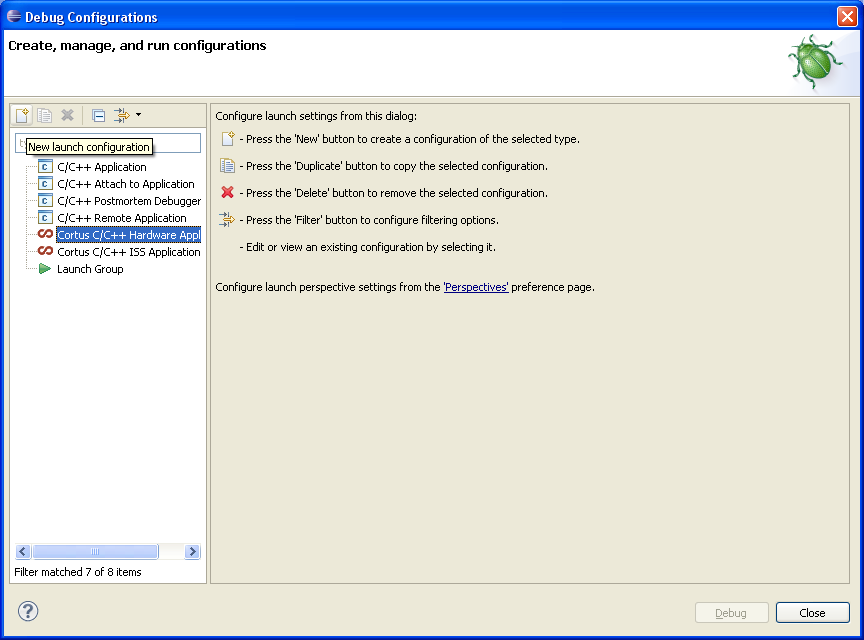
**1. GDB server setup:**

GDBServer is designed to translate Client requests into jtag commands sent and executed by emulated chip.

Before running GDBServer, it must be configured, by updating default .jtag\_ini file. This file contains all the parameters for GDBServer communication with emulated chip.

**2. Go to Run 🡪 Debug Configurations…**

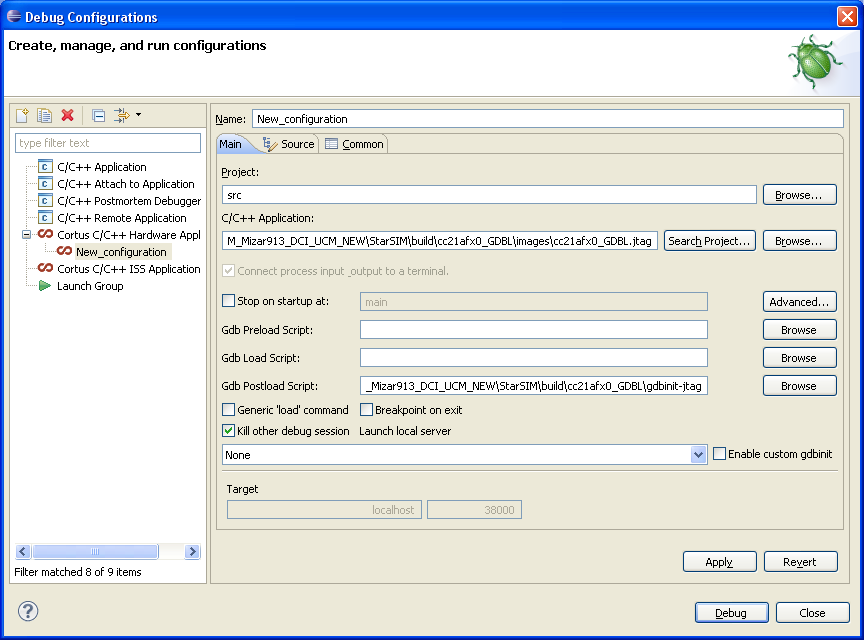
Following window will popup.



**3 Setting Hardware debug configuration.**

a. Select Cortus C/C++ Hardware Application

b. Click on New launch configuration.



c. Browse the project for your appropriate project name.

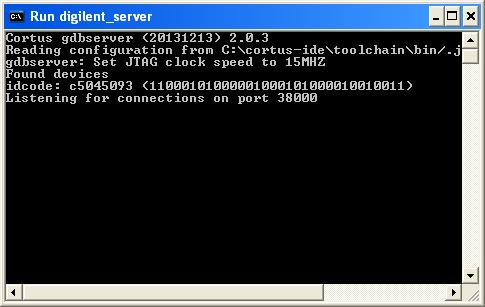
d. Browse for the C/C++ Application: where in the recently compiled output .jtag file must be selected.

e. Click on Apply and then on Debug button.

4. **Start digilent server**

Execute the Digilent\_server from Start 🡪 Programs 🡪 Cortus IDE 🡪 Run Digilent\_server.

Following window should popup



NOTE:

* If Eclipse runs time out when programming your application, it may be because aps3-gdb doesn’t not indicate to Eclipse it is still alive. Go in Eclipse Menu:

“Cortus/Run Debug Server/Configure Timeouts ...”.

You can increase these Values to avoid this programming error:



For more details on emulator section, please refer the [08] .

### Overcoming StarChip emulator limitation with linker file.

The emulator driver for initializing/writing flash can only program 32 bits words. Hence each linker section should be managed to have total variable sizes in multiple of 4 bytes, to make debugger find only ’32 bits’. This can be achieved using ALIGN keyword .

For example:

.NvmWIB13WriteArea :

{

KEEP(\*(.NvmWIB13Wr .NvmWIB13Wr\*));

. = ALIGN(4);

} > program\_memory :nvmdata=0x00

.NvmWIB13Variables :

{

KEEP(\*(.NvmWIB13 .NvmWIB13\*));

. = ALIGN(4);

} > program\_memory :nvmdata=0x00

Observation while debugging on StarChip emulator.

In this example section NvmWIB13Variables contained total variables size not in multiple of 4 bytes. Hence if we do not align it to 4 bytes using . = ALIGN(4); NVM variables do not get initialize properly on StarChip emulator. Also above method of writing linker script [putting entire section] is preferred instead of writing each variable separately.

### 

## Feature Switches Summary

## Switches in Init.h and Variant.h

The contents in Flash are determined by the settings of switches in two files, namely variant.h and init.h. The correct variant.h is provided by executing the appropriate prepare-xxx.bat. It is recommended that no changes are made to variant.h (unless you know exactly what you want to do).

Also the settings in init.h are already preconfigured, but some changes can/must be made for an initialisation.

### General Defines

#define INI\_LOCATION

#define INI\_NUMBER\_HIGH

#define INI\_NUMBER\_LOW

As usual these defines are used to identify the initialisation number according to specification “Get Initialisation Data” by Uli Huber.

#define INI\_VERSION

There are three different INI\_VERSION:

- Test Ini is used during development. Test Inis can always be deleted by

the DeleteIni command.

- Real Ini: on Real Inis the DeleteIni command is blocked after a

successful CREATE or UPDATE BINARY command.

- AU Ini: This initialisation is locked with the first ATR. This means that

such an initialisation can never be deleted!

**All customer inis must be at least “Real Inis”!**

#define DEBUG\_FLAG

All customer inis must be delivered with the DEBUG\_FLAG set to 0. During development one can set the DEBUG\_FLAG to 1 to already load an activated initialisation into the emulator.

While debugging on Emulator, Please do following setting in LbhCHWInitial

ISOTCON = 0x00; This sets ETU Timer to MODE 0 in which NULL Byte is sent automatically but NULL Byte timing not as SPT specs

In Release mode Set ISOTCON = 0x01; This sets ETU Timer to MODE 1 in which NULL Byte is sent through UART\_ISR and NULL Byte timing is as per SPT specs.

The chip is set to use internal clock of frequency 30 MHz. But emulator does not support 30 MHz internal clock. To test more time functionality on Zeta 132K emulator, debug flag must be set to 1.

**Note:** define ‘DEBUG\_FLAG’ is applicable to Product family THC20F17BD-XX only

#define MASTER\_DEVICE\_ENABLED

This compiler switch controls the enabling and disabling the MD functionality. If set to 1, then MD functionality is enabled. Otherwise it is disabled. The same file is also checked during the compile time by makcrc\_min\_flash.bat to ensure that only the MD code utilizes the bank 3. When this switch is set to 1, then the compiler output will be a MasterIni. Whereas if it is set to 0 then compiler output will be a MiniIni.

#define ADN\_IN\_MD\_INIT

This compiler switch controls the enabling and disabling of reclaiming of MD/BL area. If set to 1, then reclaim of MD/BL area is enabled. The EF-ADN file is created in the MD/BL area. If set to 0, then the EF-ADN file will be created in the File System Area. The MD/BL area will remain un utilised.

If MASTER\_DEVICE\_ENABLED is not enabled, then this flag should be disabled. In this case, the behaviour of the init will be same as that when MASTER\_DEVICE\_ENABLED is disabled as explained in above paragraph.

#define COMP128\_1, COMP128\_2, COMP128\_3, COMP128-4

The COMP algorithms can be activated or deactivated. These switches only include/exclude code in module CmdRunGSM.c. Please note that the necessary crypto functions are not automatically included /excluded. For Comp128-1 you need Comp1fast\_V10.s and for Comp128-2, 3 you need Comp128\_23\_V10fast.s. These modules can be found in the \src\sys\crypto folder.

#define DELETE\_INIT\_AUTH

This switch controls the DeleteOs with authorization feature. If this is enabled, then it allows the Os deletion only with correct authorization. By default this is enabled in Basic OTASS init. For other inis can be enabled depending upon request from customer by enabling DELETE\_INIT\_AUTH switch.

#define FILESYSTEM\_SIZE

This macro defines the file system size available and visible to the user while doing the personalization. This does not include the reclaimable area size and the mandatory file system size.

#define ADM\_COUNTER

This switch defines the behaviour of the ADM counter. It can be one of the following.

* + // 0 - disable ADM Counter,
  + // 1 - enable ADM Counter and blocking
  + // 2 - enable ADM Counter and non-blocking

#define MD\_RECLAIMABLE

This precompiler switch controls compilation of the functions under MD code. However this switch is automatically handled by the build system based on the ALLOW\_DOWNLOAD switch. Hence the programmer should not modify this switch.

#define ALLOW\_DOWNLOAD

This flag controls the feature of the OS which allows the user to download some code using the InitTool. In Mizar, since the MD is in the form of a downloadable patch, this flag must be set to TRUE if one wish to use the MD patch for generating the MaxiInit.

#define CREDEL\_RECLAIMABLE

This flag controls the CreDel feature being reclaimable or not. If one wish to use the CreDel only at the time of personalization and want to reclaim the flash space occupied by the CreDel code, then this switch must be set to TRUE

#define TEST\_APPL

This flag activate or deactivates the Toolkit Testapplication. If it is one Toolkit Test Application activated. If it is zero, Toolkit Test Application deactivated.

#define STACK\_VERIFICATION – This is a debug flag. When enabled, the stack verification utility of the debug commands will be enabled. Using this utility one can verify if there is a stack overflow possibility. One has to enable the WITH\_DEBUG\_COMMANDS flag before enabling this flag. Otherwise a compile time error message will be displayed by the build system.

**Note1** - define ‘STACK\_VERIFICATION’ is applicable to Product family SCFxxxx only

**Note2** – Both the STACK\_VERIFICATION and WITH\_DEBUG\_COMMANDS flags are for the internal verification and debug purpose only. They should not be left as TRUE or 1 for the final customer release.

#define OTASS35\_INIT

OTASS3.5 can be included. The following files must be added to the project: Otass35.c and LibShared.c.

### WIB1.2 Defines

#define MAX\_RECEIVE  
#define MAX\_SUBMIT

Maximum number of receive and submit buffers for concatenated SMS. If these defines are changed the SMS\_WRITEAREA must be adapted accordingly. The size of SMS\_WRITEAREA must be a multiple of the page size and this memory section must start with a Flash page.

#define MAX\_VAR\_VALUES

Maximum size of the variable buffer. If the size is changed then WIB\_WRITEAREA must be adapted accordingly. The size of WIB\_WRITEAREA must be a multiple of the page size and this memory section must start with a Flash page.

### WIB1.3 Defines

#define MAX\_SUBMIT 5 (default)

Maximum size for sent concatenated SMS.

#define MAX\_VAR\_VALUES 1060 (default)

Maximum size of the variable buffer. To minimize page stress the variable buffer is currently located in RAM by default. If it shall be moved to Flash, the size and location of WIB\_WRITEAREA must be adapted accordingly.

#define WIG\_VAR\_RAM 0x00

This define determines if the location of the variable buffer is either in RAM or Flash. For Zeta it is recommended to have the variable buffer in Flash due to limited RAM of 2K. If the buffer shall be located in RAM the WIB\_WRITEAREA has to be adapted accordingly.

#define WIB13\_WFRS\_TEXTS\_IN\_RAM 0x00 (default)

This define determines if the location of the WFRS Text Buffer is either in RAM or in Flash. Per default the WRFS buffer is located in Flash (module NVMWib13Write.c). If the buffer shall be moved to RAM appropriate changes/entries must be made in linker file.

**After the mask development, in order to prevent deletion of locked file,** \_**uce2ConfCreaDel variable should be set to 0x20.**

### WIB1.3UltraLite Defines

**Note**: Following defines enables or disables the functionality . When set to ‘1’ enables functionality & disables when set to ‘0’.

#define BOOKMARKING\_PRESENT

When this feature is OFF, the bookmarking support will be disabled.

#define VARIANT\_WIB\_ICONS

When this feature is OFF, the icon support SIM will be disabled. All the icon related parameters in the WIB commands will not be processed.

#define VARIANT\_WIB\_EVENTS

When this feature is OFF, no events will be registered or processed by WIB 1.3.

#define TIMER\_MANAGEMENT

When this feature is OFF, no timers will be supported.

#define SUPPORT\_RELATIV\_ADDRESSING

When this switch is OFF, relative addressing mode will be disabled. Only absolute addressing mode will be present. Note that absolute addressing mode is mandatory and always present irrespective of whether relative mode is enabled or not.

#define WIB12\_MENU\_MODE

When this feature is OFF, WIB12 compatibility mode will not be supported.

#define VARIANT\_RFM\_WITH\_POR

When feature is off, PoR support will be disabled

#define VARIANT\_REFRESH\_WITHIN\_OTASS\_3\_5\_ENVELOPE

This switch enables Otass 3.5 Refresh command support when set to 1. This feature is now part of WIB 1.3 Initialization.

#define VARIANT\_REFRESH\_WITHIN\_OTASS\_4\_1\_ENVELOPE

This switch enables Otass 4.1 Refresh command support when set. This switch is now part of WIB 1.3 Initialization.

#define OTASS\_REFRESH\_CMD\_WITHIN\_RFM\_ENVELOPE\_POR\_FIX

This switch enables the POR support in case of OTASS refresh command for both OTASS3.5 & OTASS4.1.

This switch will be ON only if either VARIANT\_REFRESH\_WITHIN\_OTASS\_3\_5\_ENVELOPE or VARIANT\_REFRESH\_WITHIN\_OTASS\_4\_1\_ENVELOPE is supported.

#define WIB\_CMD\_INSTALLREMOVE\_PLUGIN

When this feature is OFF, install or remove plugin admin commands support will be disabled.

#define VARIANT\_WIB\_WAITLOOP

When this feature is ON, WIB 1.3 displays progress information in WFRS state to the user till the response arrives. When this feature is OFF, WFRS is OFF and hence display of all progress information is disabled.

#define WFRS\_PROGRESS\_INFO

When this feature is OFF, the receiving information is not displayed. Even in receiving state, only the intermediate info is displayed. **Since Progress Information is a part of Wait Loop, it will be OFF when Wait Loop is OFF. This switch has effect only when Wait Loop switch is ON.**

#define RFM\_CMD\_CREADEL

Create/Delete File functionality via RFM can be configurable and hence can be enabled or disabled via a switch

For more details refer [25]

### RFM13Lite Defines

#define SPI\_FIRST\_BYTE\_CONFIG

RFM13Lite initialization can be configured for single SPI setting.

Default Initialization of RFM13Lite shall be configured with

#define SPI\_FIRST\_BYTE\_CONFIG 0x16 .

SPI FIRST BYTE configuration:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| B8 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | Funtionality Supported. |
| 0 | 0 | 0 | x | x | x | 0 | 1 | Redundancy Check |
| 0 | 0 | 0 | x | x | x | 1 | 0 | Cryptographic Checksum |
| 0 | 0 | 0 | x | x | 1 | x | x | Ciphering |
| 0 | 0 | 0 | 1 | 0 | x | x | x | Process if and only if counter value is higher than the value in the RE |
| 0 | 0 | 0 | 1 | 1 | x | x | x | Process if and only if counter value is one higher than the value in the RE |

**Note:** Counter bits (B5:B4) configuration 00 and 01 are always supported irrespective of Counter bits (B5:B4) configuration.

e.g. If an initialization is configured with

#define SPI\_FIRST\_BYTE\_CONFIG 0x16

Then messages with counter bit settings (B5:B4), 00,01 and 10 will get processed.

POR will always be OFF for RFM13Lite

For more details refer [25]

### Celltick App Defines

#define CLTK\_CAROUSELMODE 1

#define CLTK\_TEST\_APPL 1

#define CLTK\_MULTILANG 1

#define CLTK\_WAP\_SUPPORT 1

#define CLTK\_AT\_SUPPORT 1

#define CLTK\_ICON\_SUPPORT 0

#define CLTK\_JAVAFS 1

#define CLTK\_3\_GD\_14\_3\_1 1/0 Depending on ini

#if CLTK\_CAROUSELMODE == 1

#define CLTK\_NVMCAROUSEL 1

#endif

//Ring buffer configurations

#define CLTK\_MAX\_RB\_PAGES 18

#define CLTK\_MAX\_MSGS\_IN\_CAROUSEL 10

* Memory Calculation for Ring Buffer

The buffer size allocated for Ring Buffer determines the life span of the card.

Size of Ring Buffer is decided by two factors:

1. Number of messages need to be supported - **NUMBER\_MSG\_SUPPORT**
2. Life span of the card

SIZE\_RING\_BUFFER = (NUMBER\_MSG\_SUPPORT\*NVM\_RB\_MAX\_MESSAGE\_LENGTH\*4) + (SIZE\_ADMIN\_PAGE\*2\*NVM\_PAGE\_SIZE)

NVM\_NUM\_RINGBUFFER\_PAGES = (SIZE\_RING\_BUFFER/NVM\_PAGE\_SIZE) + 1

1. Zeta132K

Required

SIZE\_RING\_BUFFER = (11\*140\*4) + (2\*2\*512) = 6160 + 2048

= 8208

Actual:

SIZE\_RING\_BUFFER = CLTK\_MAX\_RB\_PAGES \* NVM\_PAGE\_SIZE

= 18 \* 512 = 9216 > Required

**Note: Please refer document R&D Documentation for Celltick application** **[6].**

### WIB1.3-Celltick App Defines

#define RB\_G\_RB\_USED 1

Multiapplication buffer implementation switch

#define MTI\_MMS\_CHECK 1

MMS check switch for incoming message