71M6511/71M6511H
71M6513/71M6513H
Power Meter IC Family

SOFTWARE USER'S GUIDE

8/11/2006

Revision 2.4

TERIDIAN Semiconductor Corporation

6440 Oak Canyon Rd.

Irvine, CA 92618-5201

Ph: (714) 508-8800 ▪ Fax: (714) 508-8878

meter.support@teridian.com

http://www.teridian.com/

TERIDIAN Semiconductor Corporation makes no warranty for the use of its products, other than expressly contained in the Company's warranty detailed in the TERIDIAN Semiconductor Corporation standard Terms and Conditions. The company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice and does not make any commitment to update the information contained herein.

71M651xH

71M651x

Power Meter IC FAMILY

SOFTWARE USER'S GUIDE

Demo Code Revisions 3.04 and 3.05

Table of Contents

List of Figures

List of Tables

[redacted]

LIMITED USE LICENSE AGREEMENT

Acceptance: By using the Application Programming Interface and / or other software described in this document ("Licensed Software") and provided by TERIDIAN Semiconductor Corporation ("TSC"), the recipient of the software ("Licensee") accepts, and agrees to be bound by the terms and conditions hereof.

Acknowledgment: The Licensed Software has been developed for use specifically and exclusively in conjunction with TSC's meter products: 71M6511, 71M6511H, 71M6513, and 71M6513H. Licensee acknowledges that the Licensed Software was not designed for use with, nor has it been checked for performance with, any other devices.

Title: Title to the Licensed Software and related documentation remains with TSC and its licensors. Nothing contained in this Agreement shall be construed as transferring any right, title, or interest in the Licensed Software to Licensee except as expressly set forth herein. TSC expressly disclaims liability for any patent infringement claims based upon use of the Licensed Software either solely or in conjunction with third party software or hardware.

Licensee shall not make nor to permit the making of copies of the Licensed Software (including its documentation) except as authorized by this License Agreement or otherwise authorized in writing by TSC. Licensee further agrees not to engage in, nor to permit the recompilation, disassembly, or other reverse engineering of the Licensed Software.

License Grant: TSC grants Licensee a limited, non-exclusive, non-sub licensable, non-assignable and non-transferable license to use the software solely in conjunction with the meter devices manufactured and sold by TSC.

Non-disclosure and confidentiality: For the purpose of this Agreement, "Confidential Information" shall mean the Licensed Software and related documentation and information received by Licensee from TSC. All Confidential Information shall be maintained in confidence by Licensee and shall not be disclosed to any third party and shall be protected with the same degree of care as the Licensee normally uses in the protection of its own confidential information, but in no case with any less degree than reasonable care. Licensee further agrees not to use any Confidential Information received from TSC except as contemplated by the license granted herein.

Disclaimer of Warranty: TSC makes no representations or warranties, express or implied, regarding the Licensed Software, including any implied warranty of title, no infringement, merchantability, or fitness for a particular purpose, regardless of whether TSC knows or has reason to know Licensee's particular needs. TSC does not warrant that the functions of the Licensed Software will be free from error or will meet Licensee's requirements. TSC shall have no responsibility or liability for errors or product malfunction resulting from Licensee's use and/or modification of the Licensed Software.

Limitation of Damages/Liability: IN NO EVENT WILL TSC NOR ITS VENDORS OR AGENTS BE LIABLE TO LICENSEE FOR INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH, OR ARISING OUT OF, THIS LICENSE AGREEMENT OR USE OF THE LICENSED SOFTWARE.

Export: Licensee shall adhere to the U.S. Export Administration Laws and Regulations ("EAR") and shall not export or re-export any technical data or products received from TSC or the direct product of such technical data to any proscribed country listed in the EAR unless properly authorized by the U.S. Government.

Termination: TSC shall have the right to terminate the license granted herein in the event Licensee fails to cure any material breach within thirty (30) days from receipt of notice from TSC. Upon termination, Licensee shall return or, at TSC's option certify destruction of, all copies of the Licensed Software in its possession.

Law: This Agreement shall be construed in accordance with the laws of the State of California. The Courts located in Orange County, CA shall have exclusive jurisdiction over any legal action between TSC and Licensee arising out of this License Agreement.

Integration: This License Agreement constitutes the entire agreement of the parties as to the subject matter hereof. No modification of the terms hereof shall be binding unless approved in writing by TSC.

1

1. INTRODUCTION

TERIDIAN Semiconductor Corporation's (TSC) 71M651x and 71M651XH single chip Power Meter Controllers are a family of Systems-on-Chip that supports all functionalities required to build a low-cost power meter. Demo Boards are available for each chip (71M6511/6511H and 71M6513/6513H) to allow development of embedded application, in conjunction with an In-Circuit Emulator. Development of a 71M651x application can be started in either 80515 assembly language, or more favorably in C using the Demo Boards. TSC provides, along with the 71M651x Demo Boards, a development toolkit that includes a demonstration program ("Demo Code") written in ANSI C that controls all features present on the Demo Boards. This Demo Code includes functions to manage the low level 80515 core such as memory, clock, power modes, interrupts; and high level functions such as the LCD, Real Time Clock, Serial interfaces and I/Os. The use of Demo Code portions will help reduce development time dramatically, since they allow the developer to focus on developing the application without dealing with the low-level layer such as hardware control, timing, etc. This document describes the different software layers and how to use them.

The Demo Code should allow customers to evaluate various resources of the 651X ICs but should not be regarded as production code. The Demo Code and all its components, with the exception of the CE code, are only example code and the use of it is as is and without guarantees implied. Customers may use the Demo Code as starting point at any given released revision level but should keep themselves informed about subsequent revisions of the Demo Code. Demo Code revisions may not be directly compatible with previously released revisions and/or embedded software used by customers. Customers need to adapt the Demo Code or other example code supplied by TERIDIAN Application Engineering to their own code base, and in this context TERIDIAN Semiconductor can only provide indirect assistance and support.

This Software User's Guide provides information on the following separate subjects:

- General software architecture and minimum requirements (Design Guide)
- Memory model, programming, test tools (Design Reference)
- Demo code structure, flow-charts, data flow, functions (Demo Code Description)
- Installing and using the EEP, compiler, ICE (Tool Installation Guide)
- Understanding and using the 80515 micro controller (80515 Reference)

1.1. USE OF THIS DOCUMENT

The reader should have a basic familiarity with microprocessors, particularly the 80515 architecture, firmware, software development and power meter applications. Prior experience with, or knowledge of, the applicable ANSI and/or IEC standards will also be helpful.

This document presents the features included in the 71M651x Demo Boards in terms of software and some hardware. To get the most out of this document, the reader should also have available other 71M651x publications such as the 71M651x Demo Board User's Manual, respective data-sheets, errata list and application notes for additional details and recent developments.

1.2. RELATED DOCUMENTATION

Please refer to the following documents for further information:

- 71M6511 Demo Board User's Manual
- 71M6513 Demo Board User's Manual
- 71M6511 Data Sheet
- 71M6513 Data Sheet
- Signum Systems ADM-51 In-Circuit Emulator Manual
- Keil Compiler Manual (Version 7.5 or later)
- µVision2 (Version 2.20a or later) Manual

TERIDIAN's web site (http://www.teridian.com) should be frequently checked for updates, application notes and other helpful information.

Questions to TERIDIAN Applications Engineering can be directed via e-mail to the address:

- meter.support@teridian.com

1.3.COMPATIBILITY STATEMENT

Information presented in this manual applies to the following hardware and software revisions:

- 71M6511 Demo Code Revision 3.04 and 3.05
- 71M6513 Demo Code Revision 3.04 and 3.05
- 71M6511 Demo Board Revision 2.0 and 2.1
- 71M6513 Demo Board Revision 2.0 and 2.1
- Signum Systems Wemu51 Software 3.07.00 (2/14/2005) or later
- Signum Systems ADM51 firmware version 3 (2005/02/08) or later

The revision 3.04 of the Demo Board Code is the basis for all discussed sources, commands, register addresses and so forth. Known issues with revision 3.04 are disclosed within the code description, and workarounds or improvements are shown.

Features unique to Demo Board Code revision 3.05 are highlighted and discussed where necessary.

2

2. DESIGN GUIDE

This section provides designers with some basic guidance in developing power meter applications utilizing the TSC 71M651x devices. There are two types of applications that can be developed:

- Embedded application using the sources provided by TERIDIAN, or
- Embedded application using only customer generated functions.

2.1. HARDWARE REQUIREMENTS

The following are the minimum hardware requirements for developing custom programs:

- TERIDIAN 71M651x Demo Board. This board interfaces with a PC via the RS232 serial interface (COM port).
- AC Adaptor (AC/DC output) or variable power supply.
- PC Pentium with 512MB RAM and 10GB hard drive, 1 COM port and 1 USB port, running either Windows 2000, or Windows ME or Windows XP.
- Signum Systems ADM-51 In-Circuit Emulator (for loading and debugging the embedded application) and its associated cables. Signum references this device as ADM-51.

2.2. SOFTWARE REQUIREMENTS

The following are the minimum software requirements for embedded application programming:

- Keil Compiler version 7.5 or later.
- µVision2 version 3.05c (Note: this version comes with Keil Compiler version 7.5).
- Signum Systems software Wemu51 (comes with Signum Systems ADM-51 ICE hardware).

The following software tools/programs are included in the 71M651x development kit and should be installed on the development PC:

- Demo Code with Command Line Interface (CLI) - Used to interface directly to metering functions and to the chip hardware.
- Source files
- Demo Code object file (hex file).

In order to generate and test software, the Keil compiler and the Signum in-circuit emulator (ICE) must be installed per the instructions in section 4. The include files and header files must also be present on the development PC. Typically, a design session consists of the following steps:

- Editing C source code using µVision2
- Compiling the source code using the Keil compiler
- Modifying the source code and recompiling until all compiler error messages are resolved
- Using the assembler and linker to generate executable code
- Downloading the executable code to the ICE
- Executing the code and watching its effects on the target

Software Architecture:

The 71M651x software architecture is partitioned into three separate layers:

1. The lowest level is the device or hardware layer, i.e. the discrete functional blocks of the chip and the peripheral components, such as RTC, EEPROM, MPU clock management, LCD etc.
2. The second layer consists of the functions, which enable the application to communicate with the device layer.
3. The third layer is the application layer. This layer is partially implemented by the Demo Code for evaluation purposes, but extensions and enhancements can be added by the application software developer to design suitable electronic power meter applications.

Figure 1 shows the partitions of each software component. As illustrated, there are many different designs an application can develop depending on its usage. Section 5 describes in more detail the functions within each component.

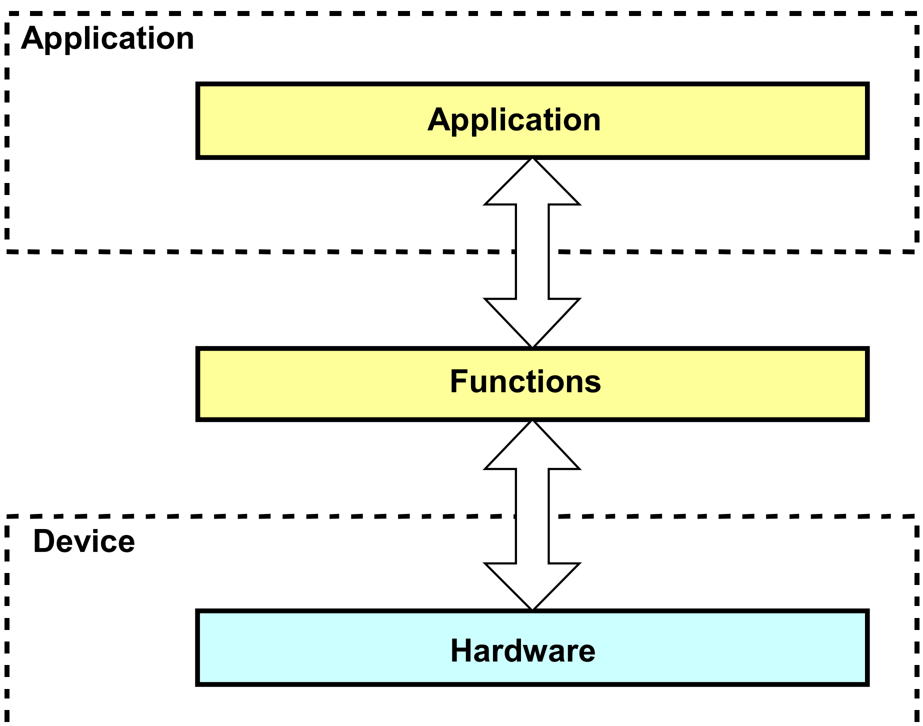


Figure 2-1: Software Structure

2.3.UTILITIES

Two utilities are offered that make it possible to perform certain operations on the object (HEX) files without having to use a compiler:

- IO_MERGE.EXE allows combining the object file with a text script in order to change certain default settings of the program. For example, modified calibration coefficients resulting from an actual calibration can be inserted into the object file.
- CE_MERGE.EXE allows combining the object file with an updated image of the CE code.

Both utilities are executed from a DOS window (DOS command prompt). To invoke the DOS window, the "command prompt" option is selected after selecting Start – All Programs – Accessories.

2.3.1. IO_MERGE

Any changes to I/O RAM (Configuration RAM) can be made permanent by merging them into the object file. The first step for this is to create a maco file (macro.txt) containing the commands adjusting the I/O RAM, such as the following commands affecting calibration:

```
]8=+16381
]9=+16397
]E=+237
```

The io_merge program updates the 6511_demo.hex file with the values contained in the macro file. The io_merge program must be in the same directory as the source files, or a path to the executable must be declared. Executing the io_merge program with no arguments will display the syntax description. To merge the file macro.txt and the object file old_6511_demo.hex into the new object file new_6511_demo.hex, use the command:

```
io_merge old_6511_demo.hex macro.txt new_6511_demo.hex
```

2.3.2. CE_MERGE

The ce_merge program updates the 6511_demo.hex file with the CE program image contained in the CE.CE file and the data image CE.DAT. Both CE.CE and CE.DAT must be in Intel HEX format, i.e. both files are not in the source format but in the compiled format (Verilog HEX). These files will be made available from Teridian in the cases when updates to the CE images are necessary.

To merge the object file old_6511_demo.hex with CE.CE and CE.DAT into the new object file new_6511_demo.hex, use the command:

```
ce_merge old_6513_demo.hex ce.ce ce.dat 6513_demo.hex
```

3

3. DESIGN REFERENCE

As depicted in Figure 1 of section 2, the 71M651x provides a great deal of design flexibility for the application developer. Programming details are described below.

3.1. PROGRAM MEMORY

The embedded 80515 MPU within the 71M651x has separate program (64K bytes) and data memory (2K bytes). In addition, it has 4K bytes of Compute Engine program RAM.

The Flash program memory is addressed as a 64KB block, segmented in 512-byte pages. Selection of these individual blocks is accomplished using the function calls related to flash memory, which are described in more detail below.

When generating code for ROM applications, special precautions have to be taken. Contact TERIDIAN Semiconductor for details.

3.2. DATA MEMORY

The 71M651x has 2K bytes of Data Memory for exclusive use of the embedded 80C515 MPU. In addition, there are 5K byte shared with the Compute Engine. See Table 3-1 for a summary.

Address (hex)	Memory Technology	Memory Type	Typical Usage	Wait States (at 5MHz)	Memory Size (bytes)
0000-FFFF	Flash Memory	Non-volatile	Program and non-volatile data	0	64KB
0000-07FF	Static RAM	Battery-buffered	MPU data XRAM,	0	2KB
1000-13FF	Static RAM	Volatile	CE data	5	1KB
2000-20FF	Static RAM	Volatile	Miscellaneous I/O RAM (configuration RAM)	0	256
3000-3FFF	Static RAM	Volatile	CE Program code	5	4KB

Table 3-1: Memory Map

3.3.PROGRAMMING OF THE 71M651X CHIPS

There are two ways to download a hex file to the 71M651x Flash Memory:

- Using a Signum Systems ADM-51 ICE.
- Using the TERIDIAN Semiconductor Flash Download Board Module (FDBM).

The 71M651x also is available in a ROM version. Testing of the ROM version is supported with the onek_c.asm assembler code.

For both programming and debugging code it is important that the hardware watchdog timer is disabled. See the Demo Board User's Manual for details.

3.4.TEST TOOLS

A command line interface operated via the serial interface of the 71M651X MPU provides a test tool that can be used to exercise the functions provided by the low-level libraries. The command-line interface requires the following environment:

1) Demo Code (651X_demo.hex) must be resident in flash memory
2) The Demo Board is connected via a Debug Board to a PC running Hyperterminal or another type of terminal program.
3) The communication parameters are set at 9600 bps, 8N1, XON/XOFF flow control

3.4.1. Running the 651X_Demo.hex Program

This object file is the 71M651x embedded application developed by TERIDIAN to exercise all low-level function calls using a serial interface. Demo Boards ship pre-installed with this program. To run this program:

- Connect a serial cable between the serial port of the Debug Board RS232 and a COM port of a Windows PC.
- Open a Windows' Hyperterminal session at 9600 bps, 8N1 with XON/XOFF flow control enabled.
- Power on the Demo Board and hit <CR> a few times on the PC keyboard until '>' is displayed on the Hyperterminal screen.
- Type '??' for general usage help. Type '? [Cmd]' for specific command help. For example, ?M will display how to run the Meter Display command.
- All references to 'c' (lower case c) indicate any ASCII character, all other lowercase letters are one-byte numbers
- Numbers can be entered in decimal by preceding them with a plus-sign (e.g. hex 20 = +32)

The 71M6511 and 71M6513 Demo Board User’s Manuals contain instructions on how to connect the serial cable.

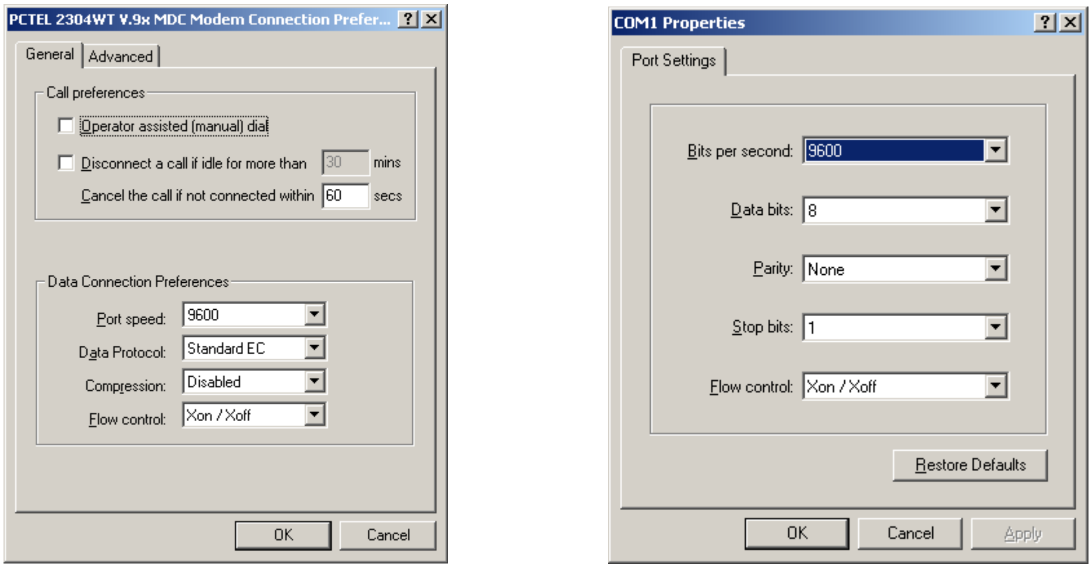


Figure 3-1: Port Speed and Handshake Setup (left) and Port Configuration Setup (right)

HyperTerminal can be found by selecting Programs →Accessories → Communications from the Windows © start menu. The connection parameters are configured by selecting File → Properties and then by pressing the Configure button. Port speed and flow control are configured under the General tab (figure 1-3, left), bit settings are configured by pressing the Configure button (figure 3-1, right), as shown below.

3.4.2. Complete List of CLI Commands

Command Overview

It is best to use the help utility of the demo code to determine how to use the commands of the command line interface. The tables in this section serve only as an overview of the capabilities of the serial command interface.

Letter(s)	Function	Comment
?	Help	
]	CE data access	
)	MPU data access	
,	Repeat command	
/	Ignore rest of line	Comment – for use in macro files
C	Compute engine (CE), RTM, TMUX control	
CL	Calibration (auto-calibration)	
CP	Pulse count control	**Demo Code revision 3.05 only**
EE	EEPROM control	
I	Information message	
M	Meter display	
P	Meter profile	
PS	Power Save Mode	
R	User I/O and SFR access	
RT	Real-Time clock	
T	Display Trim Information	
W	**Wait/reset command**	**Demo Code revision 3.05 only**
Z	Reset	

Detailed Command Descriptions

?	HELP	
Description:	Command help available for each of the options below.	
Usage:	? [option]	
Options:	?	Command line interpreter help menu.
	]	Display help on access CE data RAM
	)	Display help on access MPU RAM
	,	Display help on repeat last command
	/	Display help on ignore rest of line
	C	Display help on compute engine control
	EE	Display help on EEPROM control
	I	Display help on information message
	M	Display help on meter display control
	P	Display help on profile of meter
	R	Display help on SFR control
	RT	Display help on RTC control
	T	Display help on trim control
	Z	Display help on reset
	W	**Wait/reset command - Demo Code revision 3.05 only**
Examples:	??	Display the command line interpreter help menu.
	?C	Displays compute engine control help.

]	CE DATA ACCESS	
Description:	Allows user to read and write to CE data space.	
Usage:	] [Starting CE Data Address] [option]…[option]	
Options:	???	Read consecutive 16-bit words in Decimal
	$$$	Read consecutive 16-bit words in Hex
	=n=n	Write consecutive memory values
	]U	Update default version of CE Data in flash memory
Example:	]40$$$	Reads CE data words 0x40, 0x41 and 0x42.
	]7E=12345678=9876ABCD	Writes two words starting @ 0x7E

CE data space is the address range for the CE DRAM (0x1000 to 0x13FF). All CE data words are in 4-byte (32-bit) format. The offset of 0x1000 does not have to be entered when using the] command, thus typing]A? will access the 32-bit word located at the byte address 0x1000 + 4 * A = 0x1028.

)	MPU DATA ACCESS	
Description:	Allows user to read from and write to MPU data space.	
Usage:	) [Starting MPU Data Address] [option]…[option]	
Options:	???	Read consecutive 32-bit words in Decimal
	$$$	Read consecutive 32-bit words in Hex
	=n=n	Write consecutive memory values
Example:	08$$$$	Reads data words 0x08, 0x0C, 0x10, 0x14
	04=12345678=9876ABCD	Writes two words starting @ 0x04

MPU or XDATA space is the address range for the MPU XRAM (0x0000 to 0x7FFF). All MPU data words are in 4-byte (32-bit) format. Typing]A? will access the 32-bit word located at the byte address 4 * A = 0x28. The energy accumulation registers of the Demo Code can be accessed by typing two Dollar signs ("$$"), typing question marks will display negative decimal values if the most significant bit is set.

RAM access is limited to the lower 1KB address range. Read and write operations will "wrap around" at higher addresses, i.e. `)200?` will yield the same result as `)0?`

, (comma)	REPEAT LAST COMMAND	
Description:	Repeats the last command issued from the command line.	
Usage:	, (comma)	
Options:	NONE	

/	IGNORE REST OF LINE	
Description:	Interpreter ignores anything following this character.	
Usage:	/	Useful to separate comments from commands when sending macro text files via the serial interface.
Options:	NONE	

C	COMPUTE ENGINE, TMUX, and RTM CONTROL	
Description:	Allows the user to enable and configure the compute engine plus other controls	
Usage:	C [option] [argument]	
Options:	En	Compute Engine Enable (1 → Enable, 0 → Disable)
	Tn	Select input n for TMUX output pin
	REn	RTM output control (1 → Enable, 0 → Disable)
	RSa.b.c.d	Selects RTM output
Example:	CE0	Disables CE, followed by "CE OFF" display on LCD. The Demo Code will reset if the WD timer is enabled.
	CT3	Selects VBIAS for TMUX output pin

CL	CALIBRATION CONTROL	
Description:	Allows the user to initiate auto-calibration and to store calibration values.	
Usage:	CL [option]	
Options:	B	Begin auto-calibration. Prior to auto-calibration, the calibration coefficients are automatically restored from flash memory. If the coefficients are not unity gain (0x4000), auto-calibration will yield poor results.
	S	Save calibration coefficients to EEPROM starting at address 0x0004
	R	Restore calibration coefficients from EEPROM
	D	Restore coefficients from flash memory
Example:	CLB	Starts auto-calibration

Before starting the auto-calibration process, target values for voltage and current must be entered in I/O RAM prior to calibration (V at 0x2029, I at 0x202A, duration in accumulation intervals at 0x2028), and the target voltage and current must be applied constantly during calibration. No phase adjustment will be performed. Coefficients can be saved to EEPROM using the CLS command.

CP	PULSE-COUNT CONTROL	Demo Code Revision 3.05 only
Description:	Allows the user to control the pulse count functions.	
Usage:	CP [option]	
Command combinations:	CPA	Start pulse counting for time period defined with the CPD command. Pulse counts will display with commands M15.2, M16.2
	CPC	Clear the absolute pulse count displays (shown with commands M15.1, M16.1)
	CPDn	Set time window for pulse counters to n seconds, n is interpreted as a decimal number.
Example:	CPD60	Set time window to 60 seconds.

Pulse counts accumulated over a time window defined by the CPD command will be displayed by M15.2 or M16.2 **after** the defined time has expired.

Commands M15.1 and M16.1 will display the **absolute** pulse count for the W and VAR outputs. These displays are reset to zero with the CPC command (or the XRAM write)1=2).
Commands M15.2 and M16.2 will display the number of pulses counted during the interval defined by the CPD command. These displays are reset only after a new reading, as initiated by the CPA command.

EE	EEPROM CONTROL	
Description:	Allows user to enable read and write to EEPROM.	
Usage:	EE [option] [arguments]	
Options:	Cn	EEPROM Access (1 → Enable, 0 → Disable)
	Ra.b	Read EEPROM at address 'a' for 'b' bytes.
	Sabc..xyz	Write characters to buffer (sets Write length)
	Ta	Transmit buffer to EEPROM at address 'a'.
	Wa.b...z	Write values to buffer
Example:	EEShello; EET$0210	Writes 'hello' starting at EEPROM address 0x210.

I	INFORMATION MESSAGES	
Description:	Allows user to read and write information messages.	
Usage:	I [option] [argument]	
Options:	0	Displays complete version information
	1	Displays Demo Code version string
	1=abcdef	Change Demo Code version string
	2	Displays Copyright string
	3	CE Version string
	3=abcdef	Change CE Code version string
Example:	I1	Returns Demo Code version

M	**METER DISPLAY CONTROL (LCD)**	
Description:	Allows user to select internal variables to be displayed.	
Usage:	M [option]. [option]	
Options:	None	Displays "HELLO" message
	0	Disables display updates
	1	Temperature (C° delta from nominal)
	2	Frequency (Hz)
	3. [phase]	kWh Total Consumption (display wraps around at 999.999)
	4. [phase]	kWh Total Inverse Consumption (display wraps around at 999.999)
	5. [phase]	kVARh Total Consumption (display wraps around at 999.999)
	6. [phase]	kVAh Total Inverse Consumption (display wraps around at 999.999)
	7. [phase]	VAh Total (display wraps around at 999.999)
	8	Operating Time (in hours)
	9	Real Time Clock
	10	Calendar Date
	11 [phase]	V/I Phase at phase (degrees)
	12. [phase] (6513)	V/V Angle between phases (1= A/B, 2 = A/C))
	12.1 (6511) 13.1 (6513)	Main edge count (accumulated)
	12.2 (6511) 13.2 (6513)	CE main edge count for the last accumulation interval
	13.1 (6511) 14.1 (6513)	Absolute count for W pulses. Reset with CPC command. **Demo Code revision 3.05 only.**
	13.2 (6511) 14.2 (6513)	Count for W pulses in time window defined by the CPD command. **Demo Code revision 3.05 only.**
	14.1 (6511) 15.1 (6513)	Absolute count for VAR pulses. Reset with CPC command. **Demo Code revision 3.05 only.**
	14.2 (6511) 15.2 (6513)	Count for W pulses in time window defined by the CPD command. **Demo Code revision 3.05 only.**
Example:	M3.1	Displays Wh total consumption of phase A.

Displays for total consumption wrap around at 999.999kWh (or kVARh, kVAh) due to the number of available display digits. Internal registers (counters) of the Demo Code are 64 bits wide and do not wrap around.

Values for [phase]: 0 = sum, 1 = A, 2 = B, 3 = C (6513), 0 = sum, 1 = A, 2 = B (6511)

MR	**RMS DISPLAY CONTROL (LCD)**	
Description:	Allows user to select meter RMS display for voltage or current.	
Usage:	MR [option]. [option]	
Options:	1. [phase]	Displays instantaneous RMS current
	2. [phase]	Displays instantaneous RMS voltage
		Values for [phase]: 1 = A, 2 = B, 3 = C
Example:	MR1.1	Displays phase A RMS current.

P	**PROFILE OF METER**	
Description:	Returns current meter configuration profile	
Usage:	P	
Options:	None	

The profile of the meter is a summary of the important settings of the I/O RAM registers.

PS	**POWER SAVE MODE**	
Description:	Enters power save mode	Disables CE, ADC, CKOUT, ECK, RTM, SSI, TMUX VREF, and serial port, sets MPU clock to 38.4KHz.
Usage:	PS	
Options:	None	

R	**USER I/O AND SFR CONTROL**	
Description:	Allows user to read and write to I / O RAM and special function registers.	
Usage:	R [option] [register] … [option]	
Options:	Ix…	Select I/O RAM location x ($2000 offset is automatically added)
	x…	Select internal SFR at address x
	..???..	Read consecutive registers in Decimal
	..$$$..	Read consecutive registers in Hex
	..=n=n..	Set consecutive registers' values
	;	Exit SFR controls
Example:	RI0$$$	Read CE0, CE1 and CE2 registers

RT	REAL TIME CLOCK CONTROL	
Description:	Allows user to read and set the real time clock.	
Usage:	RT [option] [value] … [value]	
Options:	Dy.m.d.w: Day of week	(year, month, day, weekday [1 = Sunday])
	R	Read Real Time Clock.
	Th.m.s	Time of day: (hr, min, sec).
	As.t	Real Time Adjust: (speed, trim)
Example:	RTD05.03.17.5	Programs the RTC to Thursday, 3/17/2005

The "Military Time Format" is used for the RTC, i.e. 15:00 is 3:00 PM.

T	TRIM CONTROL	
Description:	Allows user to read trim and fuse values.	
Usage:	T [option]	
Options:	4	Read fuse 4.
	5	Read fuse 5.
	6	Read fuse 6.
Example:	NONE	

These commands are only accessible for the 6511H (0.1%) parts. When used on a 71M6511 (0.5%) part, the results will be displayed as zero.

W	WATCHDOG RESET
Description:	Halts the Demo Code program, thus suppressing the triggering of the hardware watchdog timer. This will cause a reset, if the watchdog timer is enabled. **Demo Code revision 3.05 only.**
Usage:	W
Options:	NONE

Z	RESET
Description:	Soft Reset.
Usage:	Z
Options:	NONE

3.4.3. Command (Macro) Files

Commands or series of commands may be stored in text (ASCII) files and sent to the 71M651X using the "Transfer – Send Text File" command of Hyperterminal or any other terminal program.

4. TOOL INSTALLATION GUIDE

This section provides detailed installation instructions for the Signum ADM-51 in-circuit emulator and for the Keil compiler.

4.1.INSTALLING THE PROGRAMS FOR THE ADM51 EMULATOR

The AMD51 ICE interfaces with the PC is via the USB serial interface.

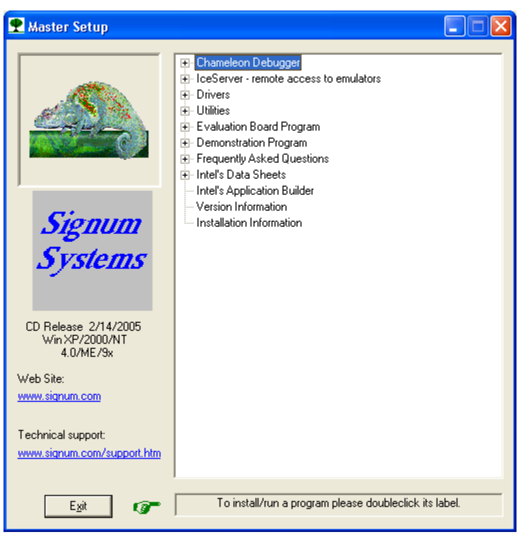
The installation process consists of the following steps:

1. Installing the Chameleon Debugger used with the Signum ICE
2. Installing the ADM51 USB driver
3. Installing updates
4. Creating a project

4.2.INSTALLING THE WEMU PROGRAM (CHAMELEON DEBUGGER)

Insert the CD from Signum Systems and connect the ICE ADM51 to the PC with the provided USB cable.

The following dialog box will appear (this dialog box also shows the release date of the program):



Click on "*Chameleon Debugger*" and then select "*ADM51 Emulator*".

Follow the instructions given by the installation program.

4.3.INSTALLING THE ADM51 USB DRIVER

The Wemu51 program communicates with the emulator ADM51 via the USB interface of the PC. The USB driver for the ADM51 has to be installed prior to using the emulator. After plugging in the USB cable into the PC and the ADM51 ICE the status light of the ADM51 emulator should come on.

A dialog box will appear, asking you to install the ADM51 driver.

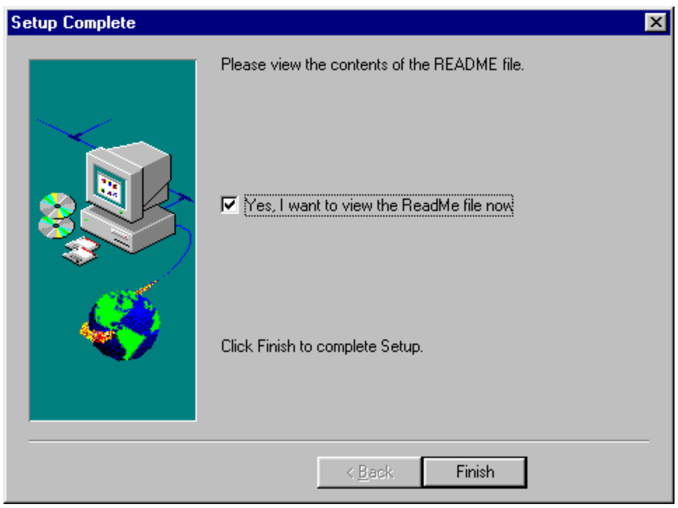
Click *Next*. Another dialog box will appear, asking how to search for the driver. Use the recommended method.

Click *Next*.

Another screen (not shown) will appear asking to locate the driver. Select *Specific Path* and browse to:

C:\Program Files\Signum Systems\Wemu51\Drivers\USB. Click Next.

Click *Finish*.



Click *Finish* again.

USB 1.1 is sufficient for operation of the ADM51. If higher performance is desired and no USB 2.0 port is available on the host PC, a USB 2.0 card can be installed as an option.

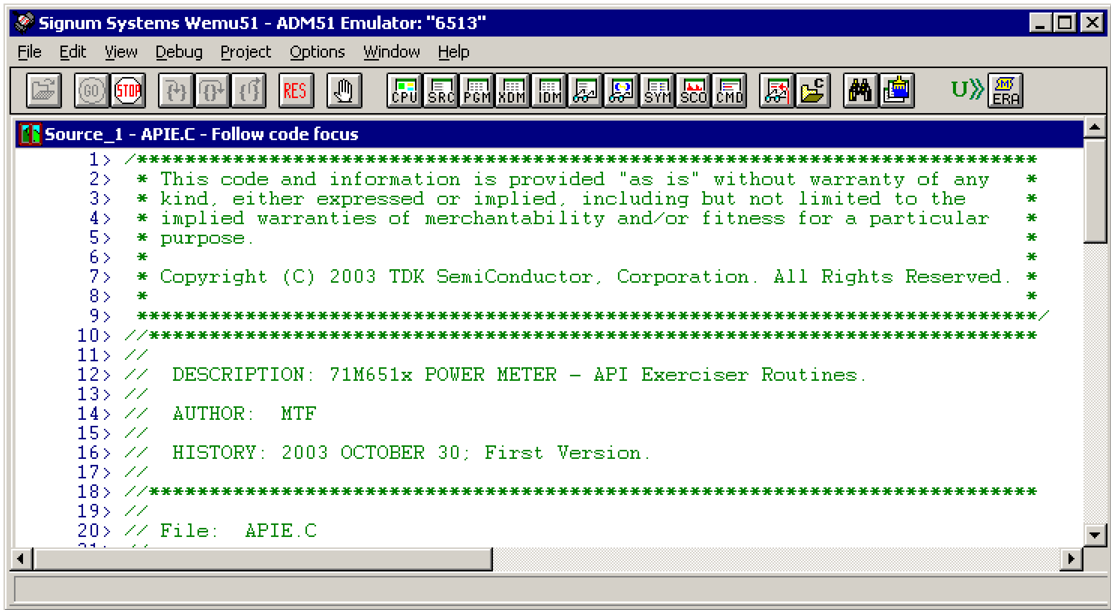
4.4. INSTALLING UPDATES TO THE EMULATOR PROGRAM AND HARDWARE

If the Wemu51 program is revision 3.07 or later, no special precautions have to be taken. Otherwise, the program should be updated using the Signum Systems web site (www.signum.com).

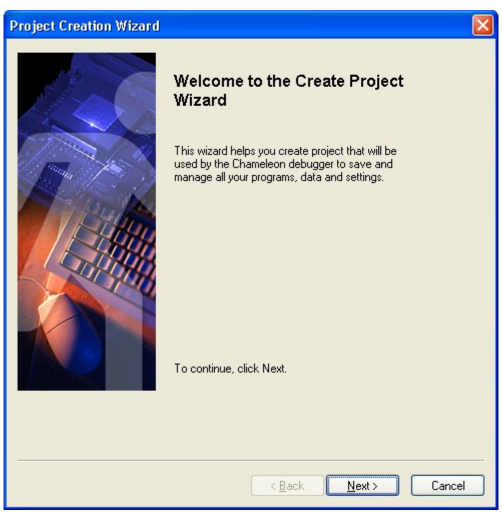
When running the Wemu51 program revision 3.07 or later, the firmware in the ADM51 will be checked automatically. ADM51 emulators with outdated firmware will not function properly. The Wemu51 will offer an automatic update for the ADM51, if necessary. For a successful upgrade it is vital to follow the instructions on screen precisely.

4.5.CREATING A PROJECT

Double click on the WEMU51 icon to start the Chameleon debugger.

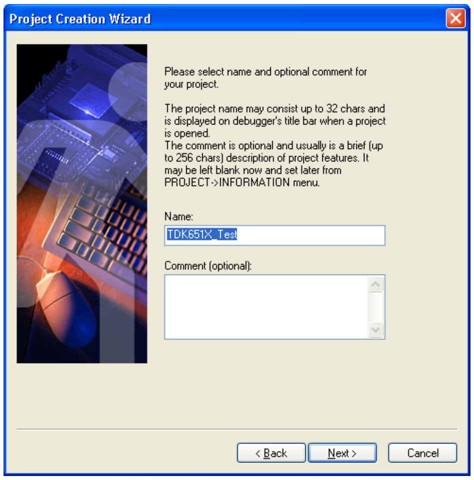


Click *Project/Create New Project*. The following screen will appear:

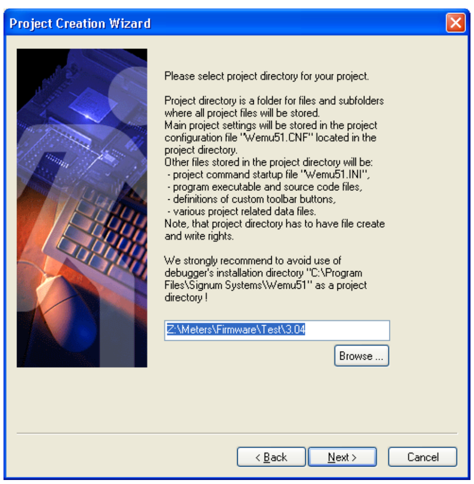


Follow the instructions of the Create Project Wizard by selecting *Next*.

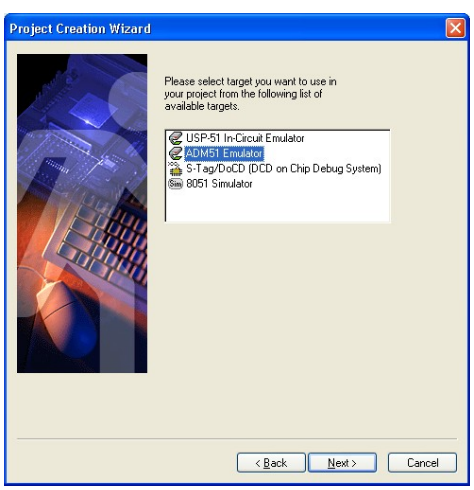
When prompted for the project name to be used, type a convenient project name. Click *Next*.



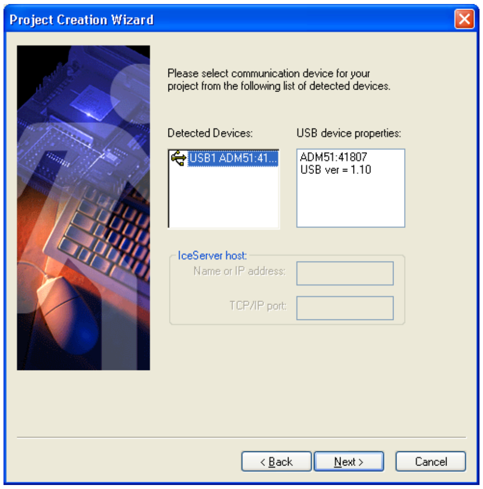
When prompted for the project directory to be used, select an existing folder on the PC.  **Do NOT select any folder in the Wemu51 installation directory**! Click *Next*.



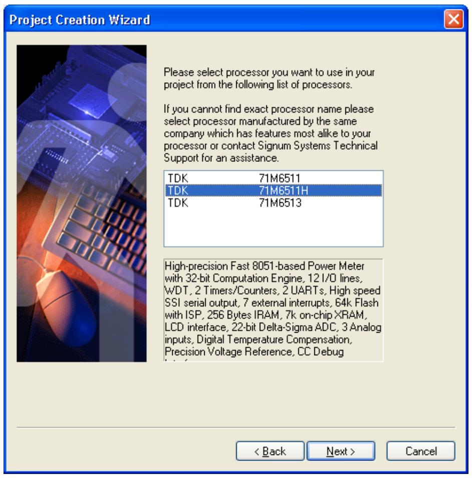
When prompted for the emulator to be used, select *ADM51 Emulator*. Click *Next*.



When prompted for the communication device to be used, select *USB ADM51*. Click *Next*.



When prompted for the processor to be used, select either *71M6511* or *71M6513*. For all firmware purposes, there is no difference between 6511 and 6511H or between 6513 and 6513H. Click *Next*.



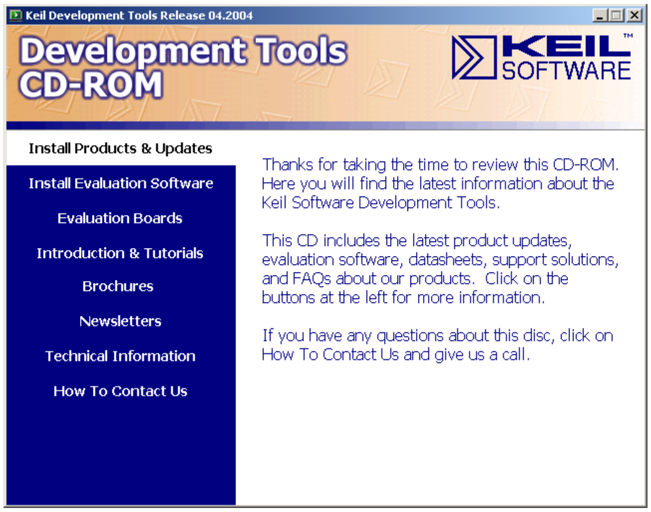
Click *Finish*.

4.6. INSTALLING THE KEIL COMPILER

After inserting the Keil CD-ROM into the CD drive of the PC, the on-screen instructions should be followed to install the Keil compiler.

For PCs that can only use one type of drive at a time (CD-ROM drive, floppy drive, such as certain laptops), it is helpful to copy the contents of the floppy labeled "Add-On Disk" to the hard drive of the PC. That way, drives do not have to be swapped out during the installation.

The installer will display the following screen:



Select Install *Products & Updates*

Select C51 *Compiler and Tools*

Follow the on-screen instructions of the installation program. When prompted for the add-on disk, insert the disk in the floppy drive and click *Next* or browse to the location of the files (if they were previously copied to the hard drive of the PC) by clicking Browse.

4.7. CREATING A PROJECT FOR THE KEIL COMPILER

4.7.1. Directory Structure

The following directory structure is established when the files from the archive 651X_Demo.zip are unpacked while maintaining the structure of subdirectories:

<drive letter>:\…\meter project\

<drive letter>:\…\meter project\CE

<drive letter>:\…\meter project\CLI

<drive letter>:\…\meter project\CLI_651X

<drive letter>:\…\meter project\docs

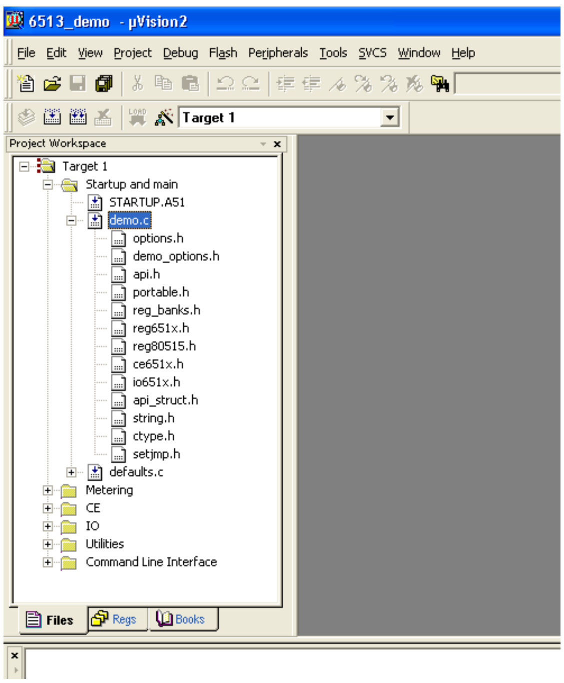
<drive letter>:\…\meter project\IO

<drive letter>:\…\meter project\Main_651X

<drive letter>:\…\meter project\Util

The project control file 651X_demo.uv2 will be in the directory <drive letter>:\…\meter project. The Keil compiler can be configured easily by loading the file 651X_demo.uv2, using the *Project* Menu and selecting the *Open Project* command.

The window shown below should appear when the project control file is opened.



The Project Workspace screen on the left side of the window shows the main components of the source (Startup and main, Metering, CE, IO, Utilities, Command Line Interface) in folders. Folders can be opened by clicking on the plus sign next to them. Opening the folders will display the source files associated with them. Opening the source files will display the header files used by them.

It should be noted that not all header files are physically present in the project directory. The files absacc.h, string.h, ctype.h, and setjmp.h are provided by the compiler manufacturer, and they are located in the Keil\C51\INC directory.

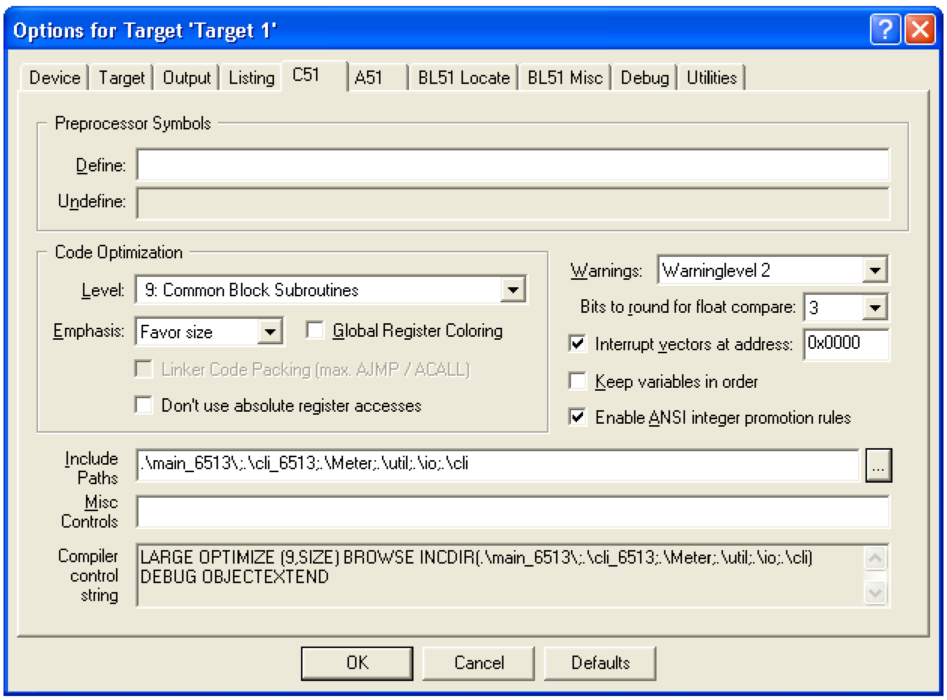
4.7.2. Adjusting the Keil Compiler Settings

Once, the Keil compiler is installed, the most convenient method to start the project is to double-click on the file 651X_demo.UV2. This will start the Keil compiler with the proper settings stored in the 651X_demo.UV2 file.

Directory structures and drive names vary from PC to PC. The settings for the compiler can be adjusted using the following method:

1. Select "target1" in the leftmost window.
2. Select "project" from the top menu and then select "options for target 1".
3. Select the "C51" tab.
4. Click the button right next to the "Include Paths" window. Three paths will be listed, pointing to meter projects, meter projects\demo, and meter projects\demo\header files.
5. If necessary, delete these path entries (X button) and replace them with the corresponding path entries for your PC (button).

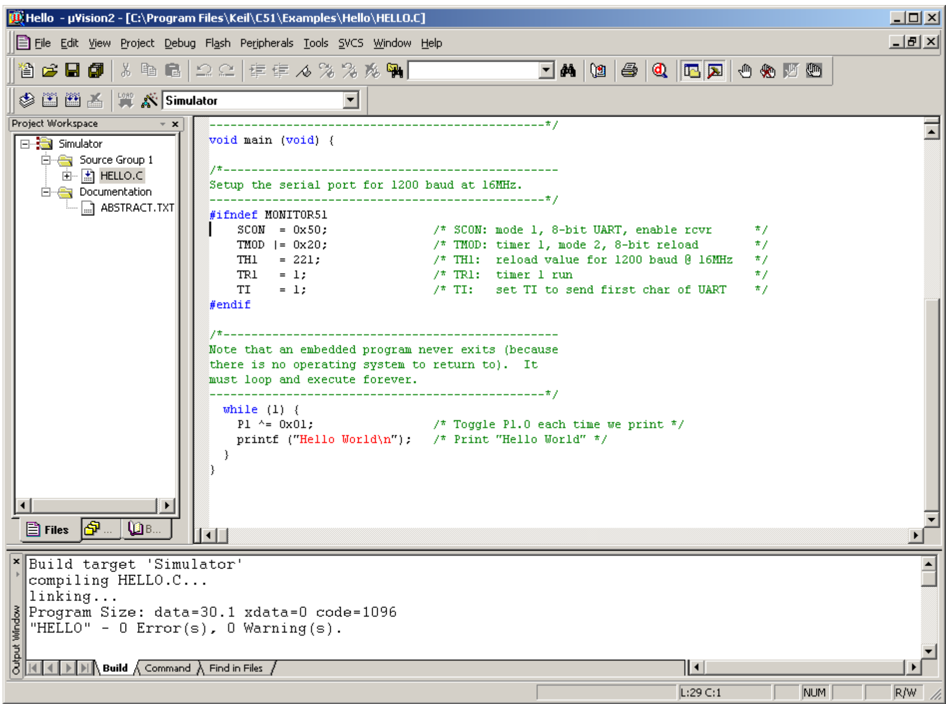
The dialog box should look like shown below. After making the necessary changes, the project file (651X_demo.UV2) should be stored.



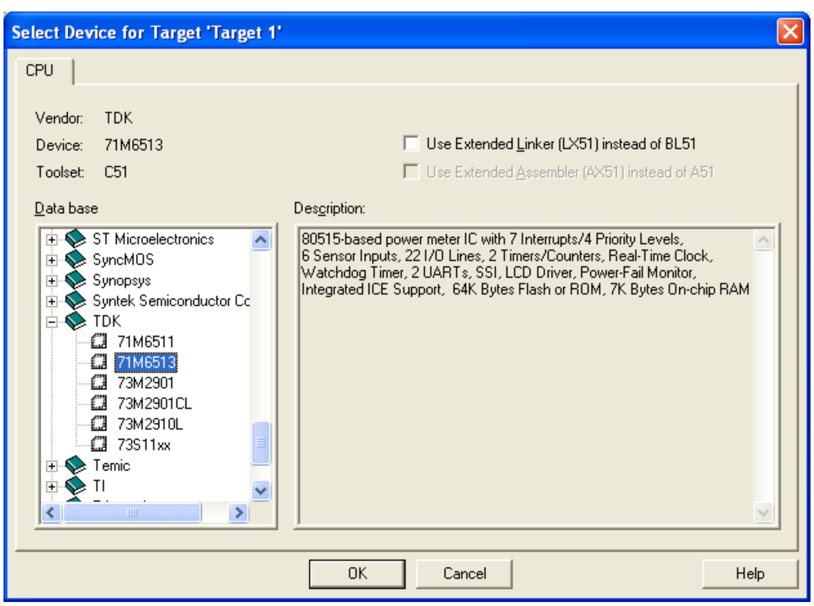
4.7.3. Manually Controlling the Keil Compiler Settings

If the method described in section "Adjusting the Keil Compiler Settings" is not followed, the Keil compiler settings can also be controlled manually.

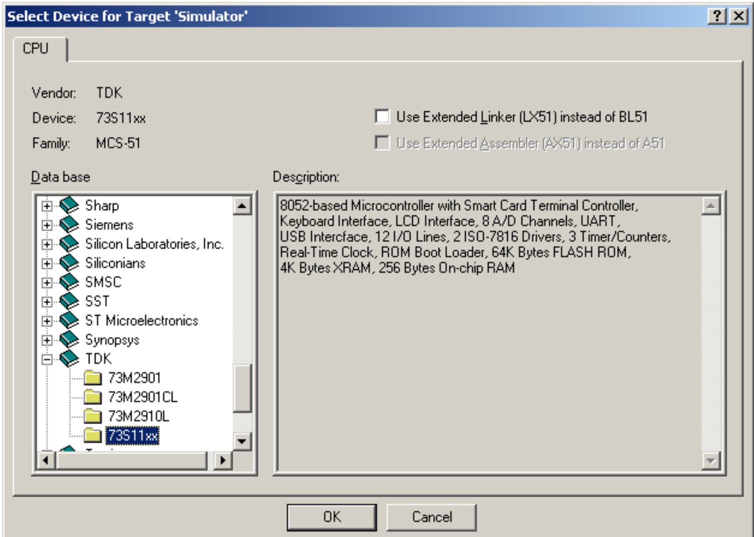
The target options should be selected in order to adapt the compiler controls properly to the target. The uVision compiler environment is started by selecting Programs → Keil → uVision2. uVision should start up and present the following window:



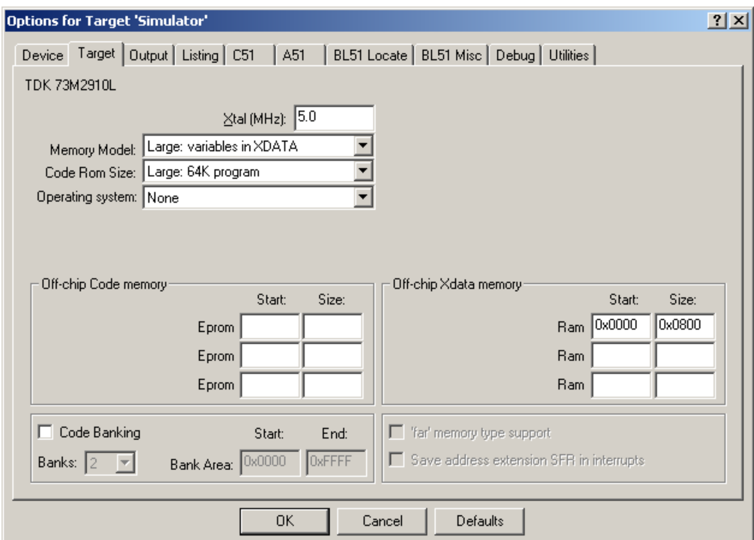
Under *Project → Options for Target1*, select the *Device* tab and check the selected device. Newer versions of the Keil Compiler offer selection of TERIDIAN (labeled "TDK") 71M6511 and 71M6513 devices:



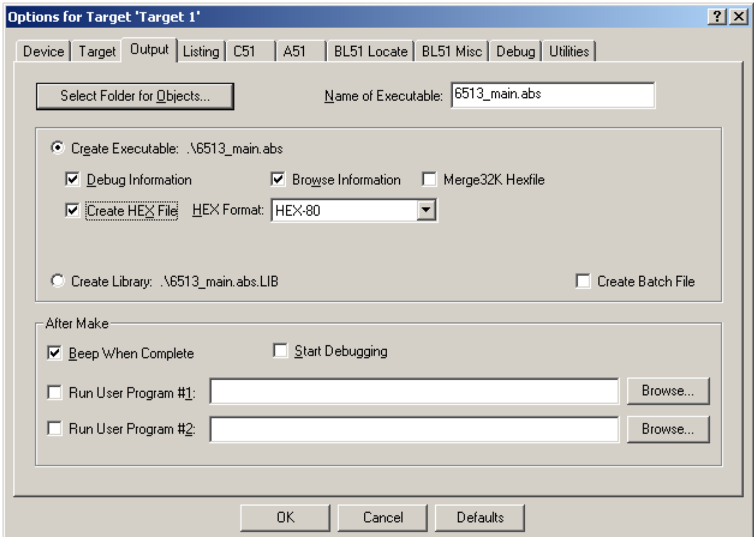
For older versions of the Keil compiler, select the TERIDIAN folder (labeled "TDK"), open it by clicking on the + sign and select *73M2910L* as the target device. Confirm by clicking *OK*.



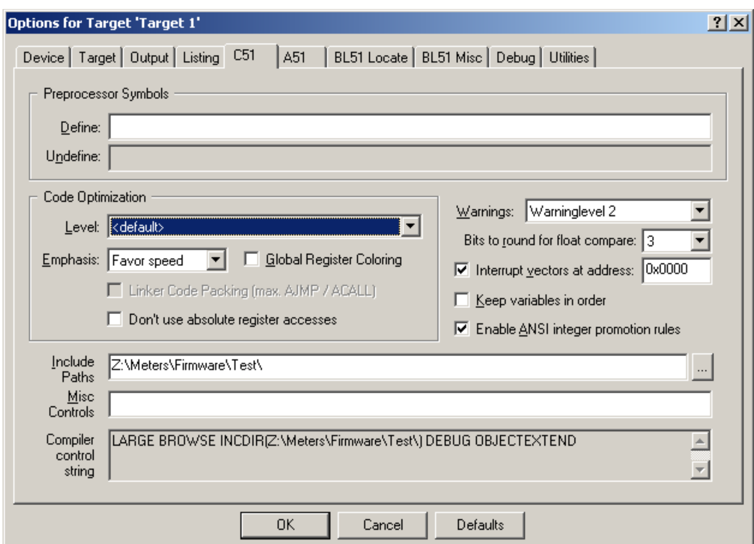
Under *Project → Options for Target1*, select the *Target* tab and enter the values in the fields as shown above. Confirm by clicking *OK*.



Under the Output tab, select a name for the executable (object) file with .abs extension' in the field labeled "Name of the executable" and check the fields by "Debug Information", "Browse Information" and "Create HEX File". This will guarantee that high-level source information will be embedded in the output file. Select *HEX-80* as the output format, as shown below:



Under the C51 tab, provide path names for the source files to be included, as shown below.



Click OK to set all the options selected for project and return to the main menu.

With the source and header files now existing in the newly created project, the files can be compiled using the Build Target option under the Project menu.

4.8.PROJECT MANAGEMENT TOOLS

With large software projects involving a multitude of source, object, list and other files in various revisions, it is very helpful to use a version control tool.

To manage file versions under Windows, Tortoise CVS, a free version control utility, might be useful. This utility can be found at http://www.tortoisecvs.org/ .

4.9.ALTERNATIVE COMPILERS

The Demo Code was written for the Keil compiler. However, alternative compilers may be used if the code is modified to ensure compatibility with the alternative compiler. One example of an alternative compiler is SDCC, a free compiler available from www. Sourceforge.net.

The Keil extensions for the 8051 are not compatible with the 8051 extensions used by the SDCC.

The batch files BUILD6511.BAT and BUILD6513.BAT are provided with the Demo Kit to support building object files using alternative compilers. These batch files use the Keil compiler calls with the applicable compiler options and can therefore serve as examples on how to invoke alternative compilers. The linker control files LINK6511.TXT and LINK6513.TXT called by the batch files can show how to properly invoke linkers.

To compile with DOS-style tools, arrange for a DOS batch file to invoke the tools and set the properties of the batch file to leave the window open, so that errors can be seen. Then, to compile, double click on this batch file in Windows explorer.

4.10.ALTERNATIVE EDITORS

Many modern text editors have a feature called "tag jumping" that helps a programmer to read and understand unfamiliar code. TERIDIAN Semiconductor recommends using such an editor to read, understand and modify demonstration code. Tag jumping is a feature that is not supported by the Keil uVision editor.

This is how tag jumping works:

1. A "tag file generator" program is run on some directories containing .c or .h files. TERIDIAN Semiconductor recommends placing the tag file generator in a DOS batch file in the same directory as the make file of the project. Wattmeter demonstration code includes such a batch file: "T.BAT". To run a batch file, double-click it in windows explorer. A DOS batch file is just an ASCII file (like a .C file) containing DOS commands. DOS commands are described at http://www.computerhope.com/msdos.htm .
2. The tag file should then be copied to convenient places for a text editor. TERIDIAN Semiconductor recommends copying the tag file into each source code directory. In that way, the default tag file location for most editors becomes just ".\tags" for all projects, and multiple projects do not conflict. Copying the tag file can be an automatic part of the DOS batch file that generates the tag file.
3. It is easiest if Windows explorer opens .C files automatically with the editor when they are clicked. To do this, change file associations. (See Windows help.)
4. Inside the editor, select a subroutine name or variable, then use the editor's "tag jump" feature. The editor immediately opens the file at the line where the subroutine or variable is defined. Or, if the same symbol is in several places, it offers a choice of files.

TERIDIAN Semiconductor recommends the "exuberant CTAGs utility" for generating tag files. The code can be found for free at: http://ctags.sourceforge.net/. The choice of a text editor is very personal. Many editors support Exuberant CTAGS. See the list of supporting tools at http://ctags.sourceforge.net/tools.html .

Some editors to be considered are:

- VIM, see http://www.vim.org/ a free VI editor. VIM is available in full-featured versions for Windows. VI is part of the POSIX standard, so using it is a portable skill. VIM wins awards for usability.
- UltraEdit http://www.ultraedit.com/ , an inexpensive (not free), professional Windows programming editor. This editor works like all other Windows applications, with extra features to support programming languages. NEDIT (The Nirvana Editor) is very similar, at http://www.nedit.org/ . NEDIT runs on Unix with Motif, and also supports exuberant CTAGs.
- GNU Emacs, a free editor, also supports exuberant CTAGs. See: http://www.gnu.org/software/emacs/emacs.html

5

5. DEMO CODE DESCRIPTION

5.1. 80515 DATA TYPES AND COMPILER-SPECIFIC INFORMATION

5.1.1. Data Types

The 80515 MPU core is an 8-bit micro controller; thus operations that use 8-bit data types such as "char" or "unsigned char" work more efficiently than operations that use multi-byte types, such as "int" or "long". The Keil C51 compiler supports ANSI C data types as well as data types that are unique to the generic 8051 controller family. Table 5-1 lists available data types. Please refer to the Keil Cx51 Compiler User's Guide for more details.

Various types of address spaces are available for the 80515 MPU core of the 71M651x, and in order to utilize the various memory space types, the Demo Code uses following variable type definitions (typedefs.):

- **typedef unsigned char** data U08d; /* 128 Bytes of direct access RAM,fast, best speed and smallest amount of code for access.
- **typedef unsigned char** idata U08i; /* 128 Bytes of indirect access RAM, second best for speed and amount of code for access.
- **typedef unsigned char** xdata U08x; /* 2K Bytes of indirect access XRAM, slowest speed and largest amount of code for access.
- **typedef unsigned char** code U08r; /* 64K of indirect access FLASH (a good place to put constants).
- **typedef unsigned char** U08; /* Default space selected by 'C' compiler. (User selectable). A similar set has been defined for signed/unsigned char, short and long variables.

Throughout the Demo Code, an attempt has been made to put the most frequently used variables in the fastest memory space.

Data Type	Notation	Bits	Bytes	Comments
Bit	Bbool	1		Unique to 8051
Sbit		1		Unique to 8051
SFR		8	1	Unique to 8051
SFR16		16	2	Unique to 8051
signed/unsigned char	U08	8	1	ANSI C
enum	enum	8 or 16	1 or 2	ANSI C
unsigned short	U16	16	2	ANSI C
signed short	S16	16	2	ANSI C
signed/unsigned int	U16	16	2	ANSI C
signed int	S16	16	2	ANSI C
unsigned long	U32	32	4	ANSI C
Float	F32	32	4	ANSI C

Table 5-1: Data Types

5.1.2. Compiler-Specific Information

The 8051 has 128 bytes of stack, and this motivates Keil C's unusual compiler design. By default, the Keil C compiler does not generate reentrant code. The linker manages local variables of each type of memory as a series of overlays, and uses a call-tree of the subroutines to arrange that the local variables of active subroutines do not overlap.

The overlay scheme can use memory very efficiently. This is useful because the 71M651X chips only have 2k of RAM, and 256 bytes of internal memory.

The compiler treats uncalled subroutines as possible interrupt routines, and starts new hierarchies, which can rapidly fragment each type of memory and interfere with its reuse.

To combat this, the following measures were taken when generating the Demo Code:

- The code is organized as a control loop, keeping most code in a single hierarchy of subroutines,
- The programmers eliminated unused subroutines by commenting them out when the linker complained about them. Also, the Demo Code explicitly defines interrupt code and routines called from interrupt code as "reentrant" so that the compiler keeps their variables on a stack.
- When data has a stable existence, the Demo Code keeps a single copy in a shared static structure.

With these measures applied, the Demo Code uses memory efficiently, and normally no memory issues are encountered. The demo code does not have deep call trees from the interrupts, so "small reentrant" definitions can be used, which keep the stack of reentrant variables in the fast (small) internal RAM.

The register sets are also in internal memory. The C compiler has special interrupt declaration syntax to use them. The "noaregs" pragma around reentrant routines stops the compiler from accessing registers via the shorter absolute memory references. This is because the demo code uses all four sets of registers for different high-speed interrupts.

Using "noaregs" lets any interrupt routine call any reentrant routine without overwriting a different interrupt's registers.

There is a known defect in version 7.50a of the Keil compiler:

Memory types must be explicitly defined in local variables. Using a predefined type is not explicit enough, i.e. "char xdata c;" is ok. "typedef char int8_t; ... int8_t data c;" is ok, "typedef char data int8d_t; ... int8d_t c;" is not ok.

5.2. PROGRAM FLOW

5.2.1. Startup and Initialization

The top-level functionality of the Demo Board is controlled by the high-level functions. As with every C program, the core of the function is in the main() program. The main() program is contained in the demo.c source file. It performs the following steps (see Figure 5-1, Figure 5-2, and Figure 5-3):

1. Reset watchdog timer
2. Initialization for hardware, pointers, metering variables, UART buffers and pointers, CE, restoration of calibration coefficients, initialization of LCD w/ "HELLO" message), enabling CE and pulse generators.
3. Assign program state (idle state or command mode state) to be entered next.
4. In an endless loop, jump to idle state and command mode state in turns. During the idle state, background tasks are performed, such as process timers, CE updates, CE display. During the command state, the command line interface (CLI) is serviced (Figure 5-4).

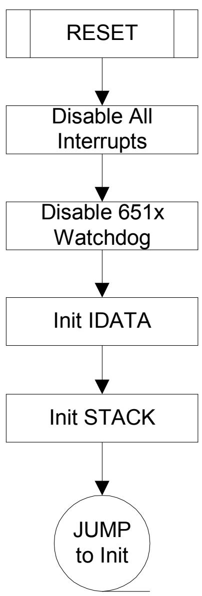


Figure 5-1: STARTUP.A51

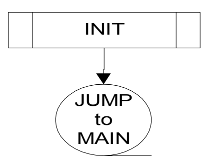


Figure 5-2: INIT.A51

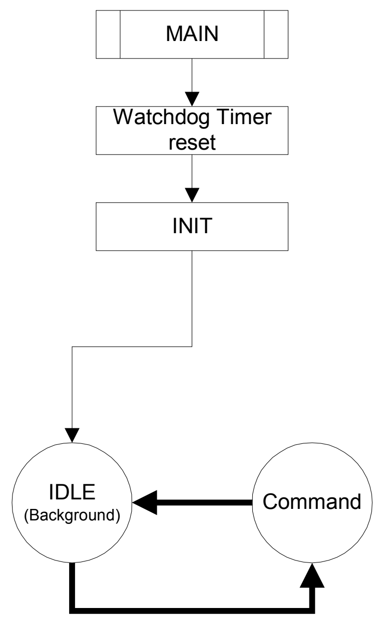


Figure 5-3: DEMO.C

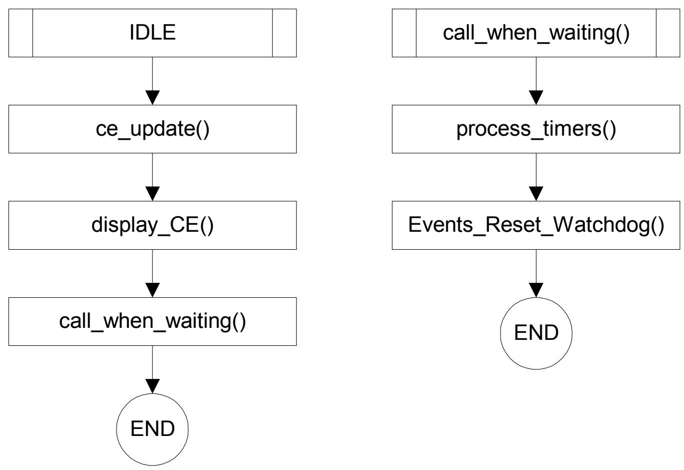


Figure 5-4: MAIN LOOP

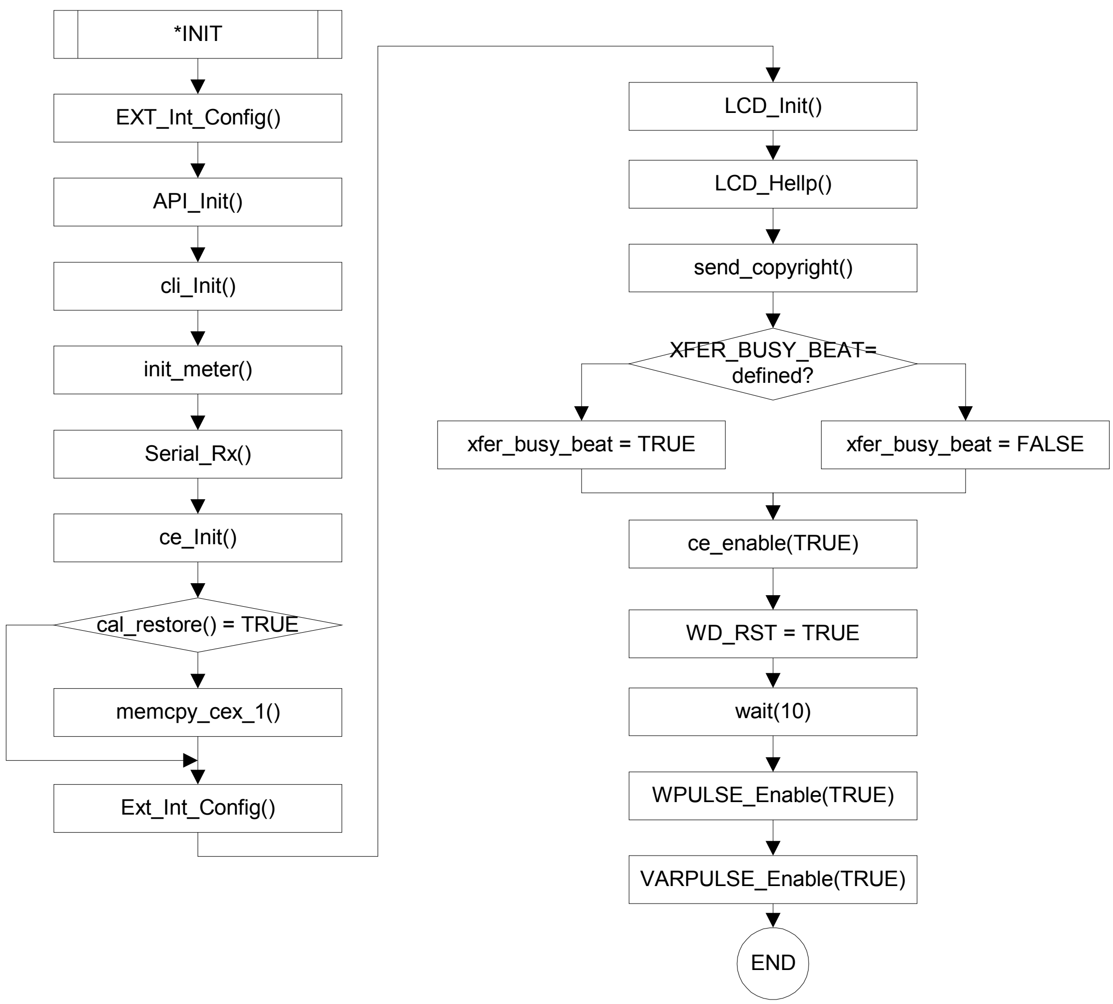


Figure 5-5: INIT Function

Before the MPU gets to execute the main() program, it will execute the startup instructions contained in the STARTUP.A51 assembly program (Figure 5-1). Upon completion, STARTUP.A51 causes a jump to the label

C_START, which is contained in the second startup assembly program named init.A51 (Keil/C51/LIB directory, see Figure 5-3). Init.A51 finally causes the jump to main(). The startup files are described in section 5.7.

The stack is located at 0x80, growing to higher values, while the reentrant stack is located at 0xFF, growing downwards.

Once operating, the main() program expects regular interrupts from the CE. If no interrupts occur, the main() program will cease to trigger the watchdog timer, resulting in a reset condition, if the watchdog timer is enabled.

5.3.BASIC CODE ARCHITECTURE

The TERIDIAN 71M651x firmware can be divided into two code parts. One is the Background task that is executed whenever there are no other higher priority exceptions such as the servicing of interrupts. The second part consists of the interrupt-driven code (Foreground) tasks, such as the CE_BUSY Interrupt, Timer Interrupt, and other Interrupt service routines. The background code takes care of the non time-critical functions starting with the system reset, and this code is executed every time when there are CPU resources available after taking care of all interrupt-driven tasks. The background of the 71M651x firmware is implemented as a very simple state machine. One state is serving the command inputs and the other is idle/Display control.

5.3.1. Initialization

When the power applied for the first time or RESETZ is asserted, the 71M651x device executes the code pointed to by the reset vector.

5.3.2. Foreground

There are total 12 interrupts available for the 80515, and the revision 3.04 demo code uses a total of 11 interrupts. Table 5-2 shows the interrupt service routines (ISRs), the corresponding vectors (Table 6-58 in section 6.3.5.4) and their priority, as assigned by the MPU using the IP0 and IP1 registers (see section 6.3.5.2).

Interrupt Routine	name	in source file	vector	priority (3 = highest)
io_high_priority_isr	EXT0	misc.c	0x03	1
io_low_priority_isr	EXT1	misc.c	0x13	3
compare_falling_isr	EXT2	io651x.c	0x4B	0
ce_busy_isr	EXT3	ce.c	0x53	3
compare_rising_isr	EXT4	io651x.c	0x5B	0
eeprom_isr	EXT5	eeprom.c	0x63	1
xfer_busy_isr	EXT6 (shared w/ RTC)	ce.c	0x6B	2
timer0_isr		timers.c	0x0B	0
timer1_isr		timers.c	0x1B	0
rtc_isr	EXT6 (shared w/ XFER)	rtc.c	0x6B	2
es0_isr		serial.c	0x23	1
es1_isr		serial.c	0x83	1

Table 5-2: Interrupt Service Routines

All interrupt service routines (ISRs) must be declared "small reentrant". Also, all routines called by ISRs must be reentrant as well.

TIMER Interrupt

timer0 of the MPU is used to generate a 10ms timer tick, which is adjusted for MPU clock speed. The timer tick (variable tick_tock) is used to control the software timers. The software timers are updated by the process_timers()

function in the main loop of the background task (see Figure 5-7). Eight software timers can be simultaneously running.

There is an issue with this type of timer processing: The CE_update() routine takes about 400ms, and when it is executing, it is not checking the tick_tock variable, resulting in the software timers running slow.
The process_timers() routine was fixed in firmware revision 3.05, where timer0 increments a count, and the process_timers() routine processes the count, not just the presence of the timer tick.

timer1 is used for delay functions, e.g. for EEPROM or RTC access control. Timer 1 is enabled and starts functioning by calling the "Add_Delay_Func()" function as defined in the timer.c module.

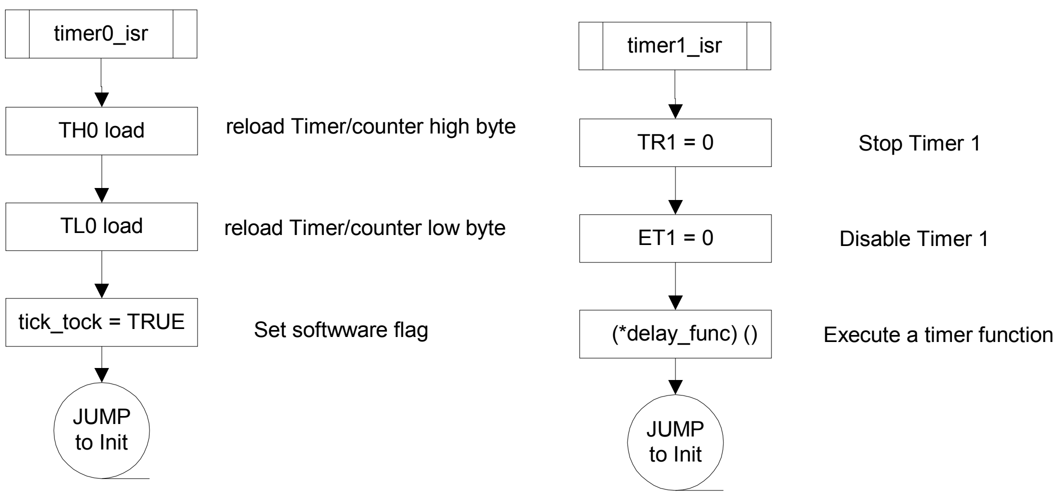


Figure 5-6: Timer ISRs

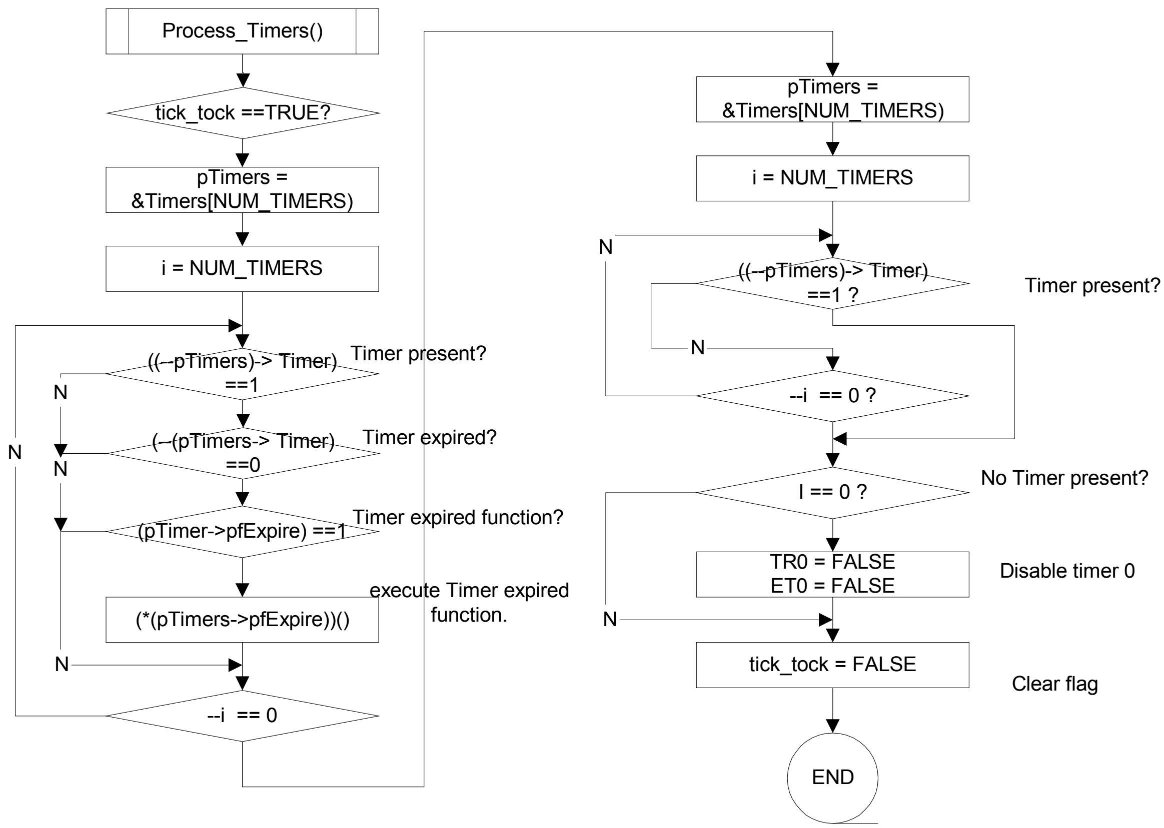


Figure 5-7: Process Timer (non-ISR)

CE_BUSY Interrupt

CE_BUSY interrupt is used for handling the outputs of the CE that are refreshed every 396μs, i.e. CHOP control and SAG detection. When this routine is called the automatic chopping is re-established if it is off.

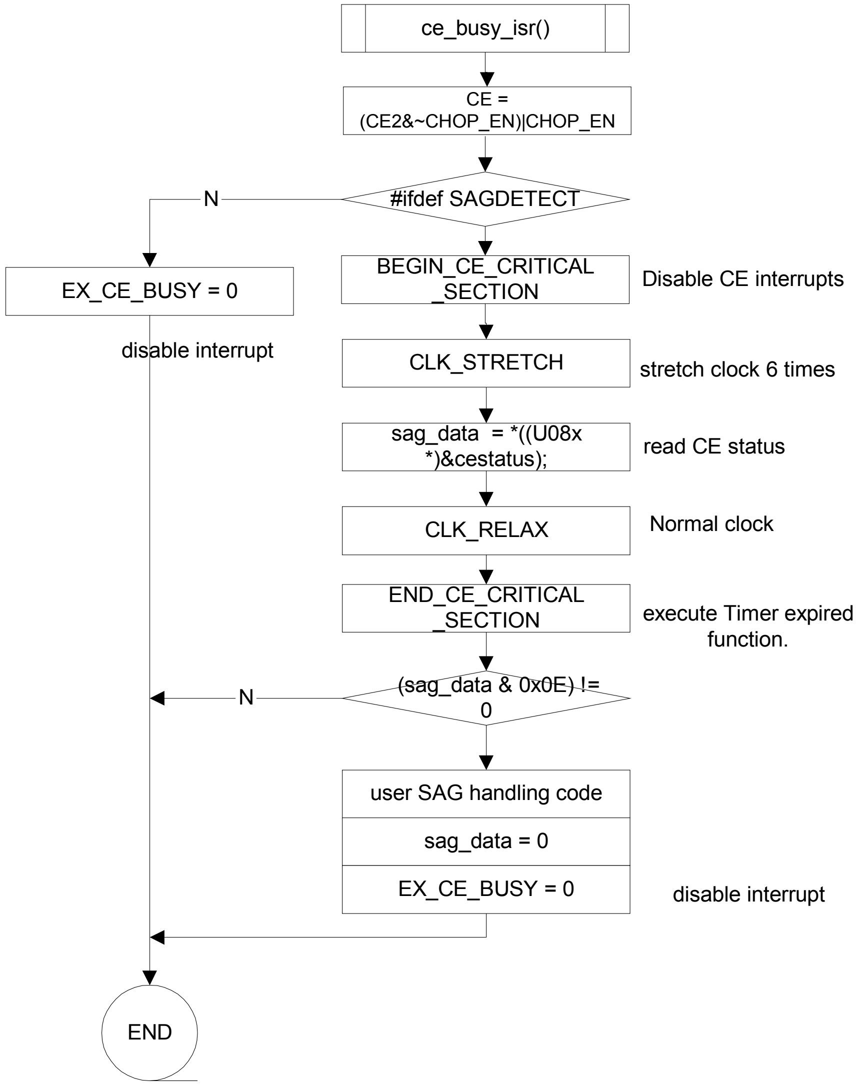


Figure 5-8: CE_BUSY ISR

XFER_BUSY Interrupt

XFER Busy interrupt is requested by the CE at the conclusion of every accumulation cycle. The interrupt service routine copies the CE output data to the MPU data RAM for further processing by the MPU, which is performed by the background task. The handling of data for the generation of pulses is also managed in this ISR.

XFER_Busy waits until the second interrupt after one second has elapsed, since it takes roughly one second for the PLL in the CE to settle and (therefore) for the filtering to be reliable.

The copy operations CE → X and X → CE stated in the flow chart are implemented with the memcpy_cex() and memcpy_xce() routines, which move data between XRAM and CE DRAM or vice versa. Due to the wait states that apply to accesses of CE DRAM, this operation cannot be done directly. Using memcpy_cex(), data is moved from XRAM to CE DRAM, with memcpy_xce(), data is moved from CE DRAM to XRAM.

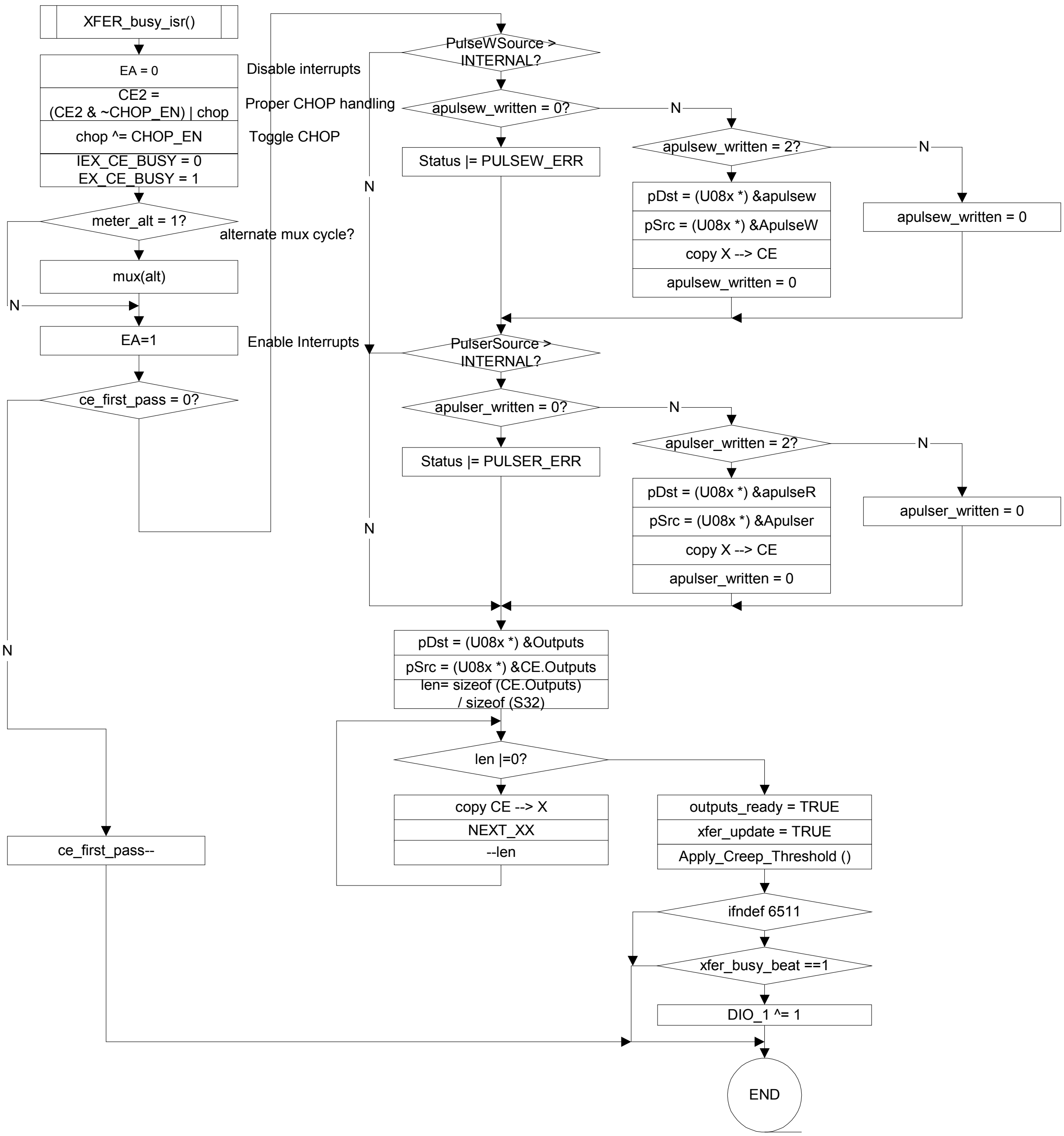


Figure 5-9: XFER_BUSY ISR

SERIAL Interrupt

ES0_isr is the ISR servicing UART 0. In this ISR, the UART data is sent and received along using flow control, if enabled. Parity and other serial controls are managed in this ISR. The alternative serial port, UART 1 uses an ISR with similar code structure.

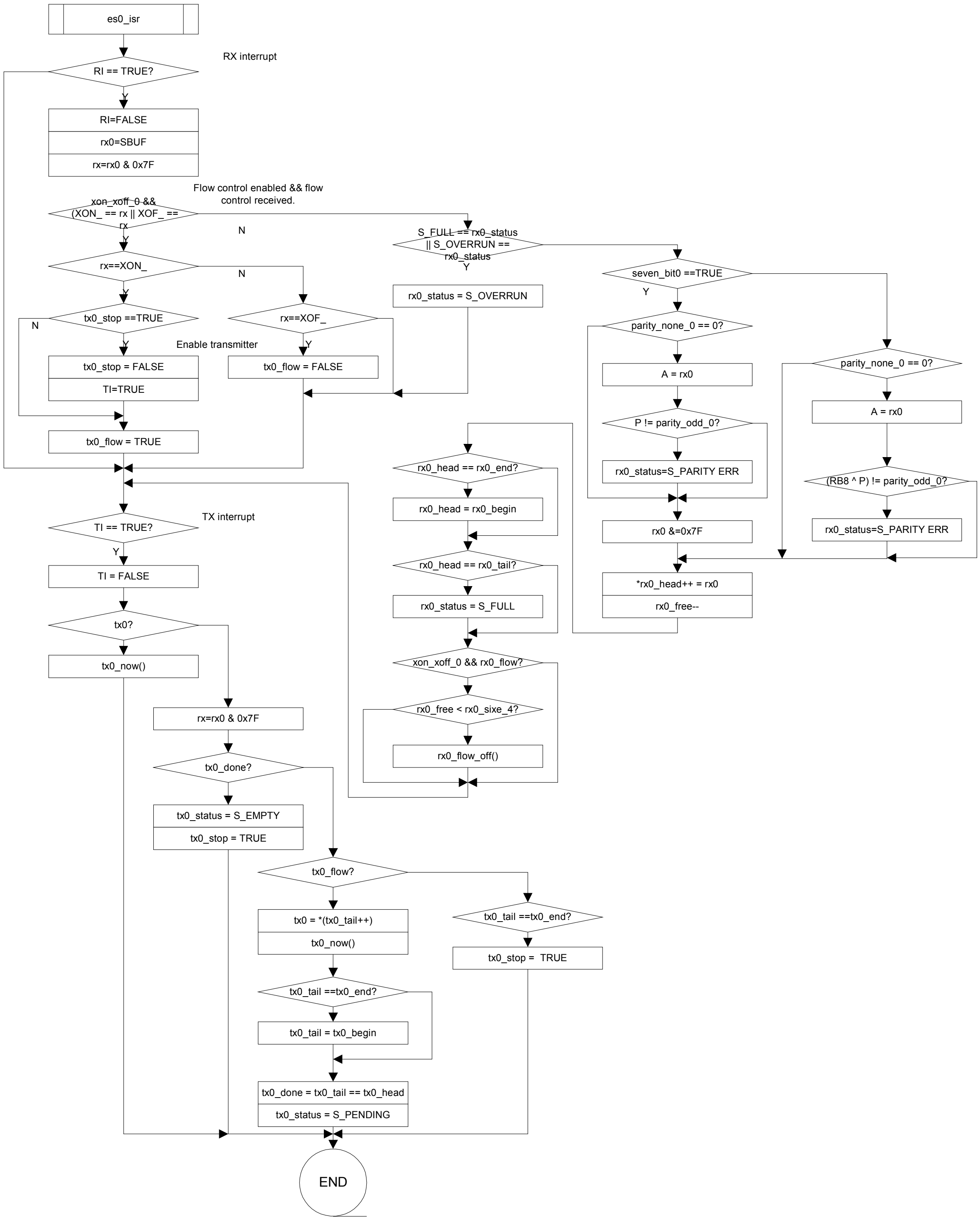


Figure 5-10: Serial 0 isr

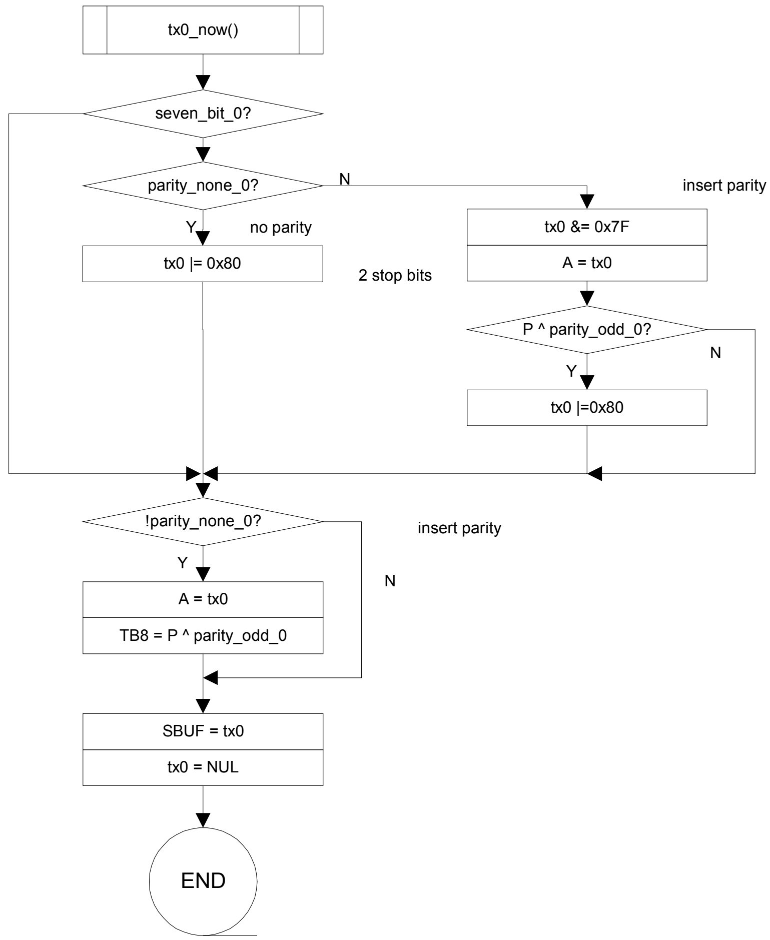


Figure 5-11: tx_now()

5.3.3. Background Tasks

CE Update

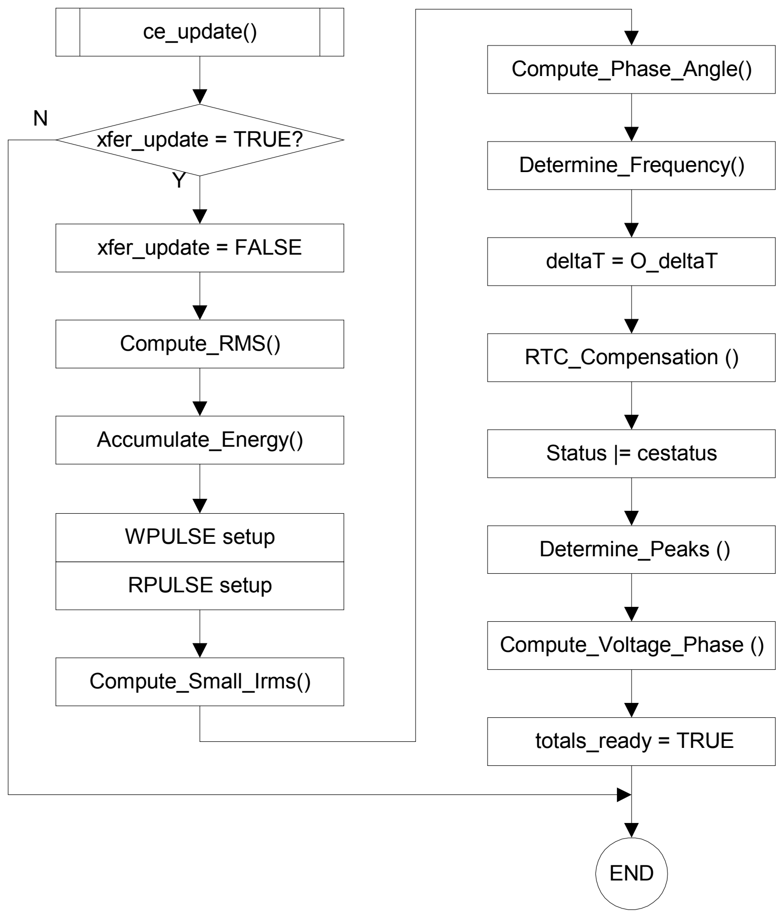


Figure 5-12: ce_update

Display CE

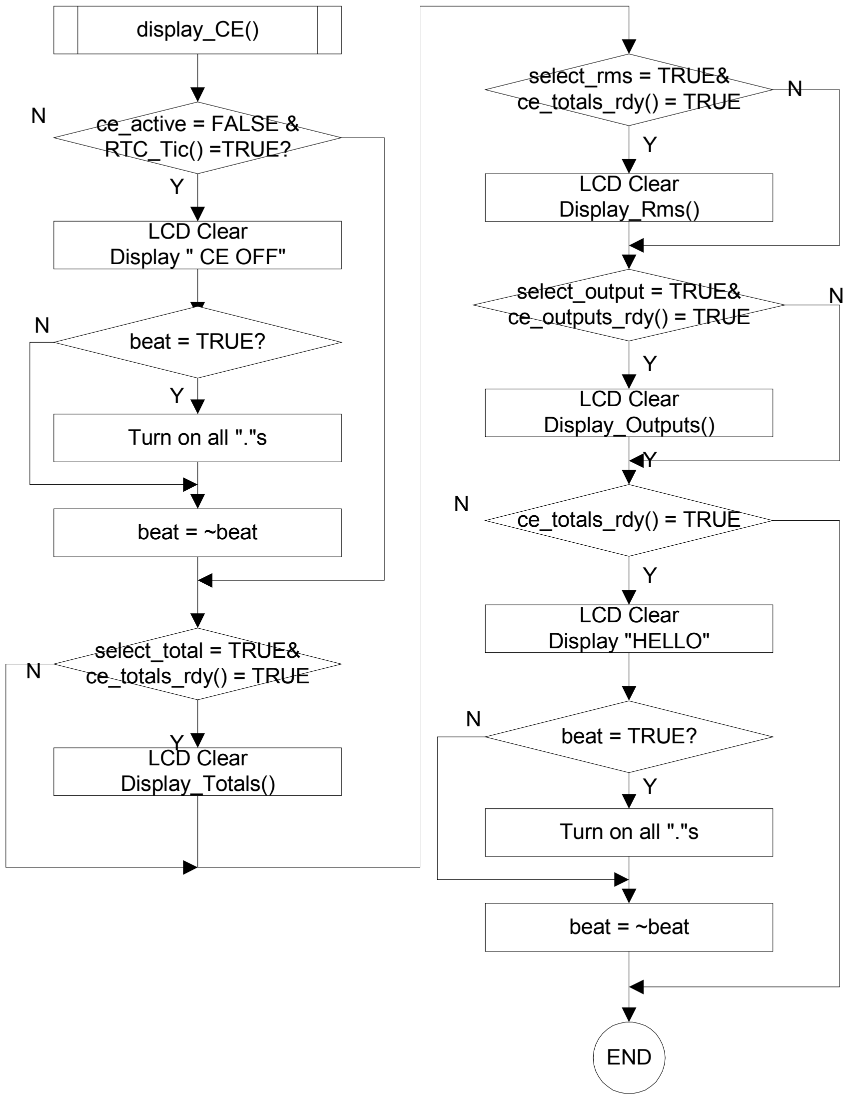


Figure 5-13: Display CE

Command Line Interpreter

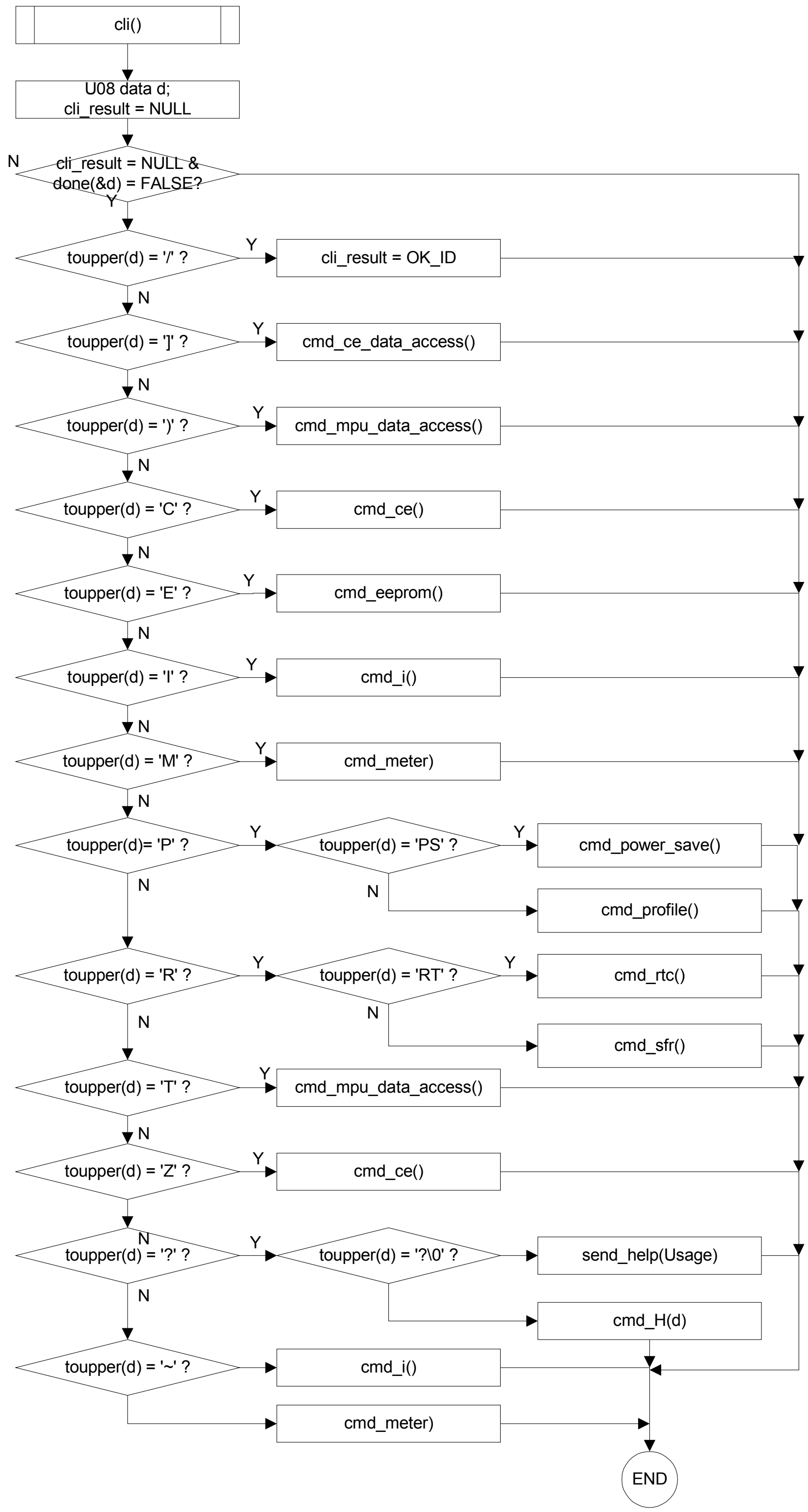


Figure 5-14: Command Line Interpreter

Auto-Calibration

Auto-calibration is a simplified calibration procedure based on voltage and current measurements. A 0° load angle (pure resistive load) is assumed. No phase compensation will be performed. Before starting the auto-calibration process, the user enters the target values for voltage and current that will be applied during the calibration process in the MPU addresses 0x2029 and 0x202A (see description of aut-calibration in the CLI section). The target values should be applied to the meter and held constant during the auto-calibration process.

The routines shown in Figure show how auto-calibration is started. The cal_begin() routine starts a state-machine by setting the flag cal_flag to YES, after setting the calibration factors to default values, recording the calibration temperature, calculating the temperature compensation coefficients and setting the counter cs for calibration cycles.

The actual stabilization delay, measurement and adjustment phases are managed by separate routines that are activated by cal_flag being YES and controlled by the variable cs which counts down accumulation intervals.

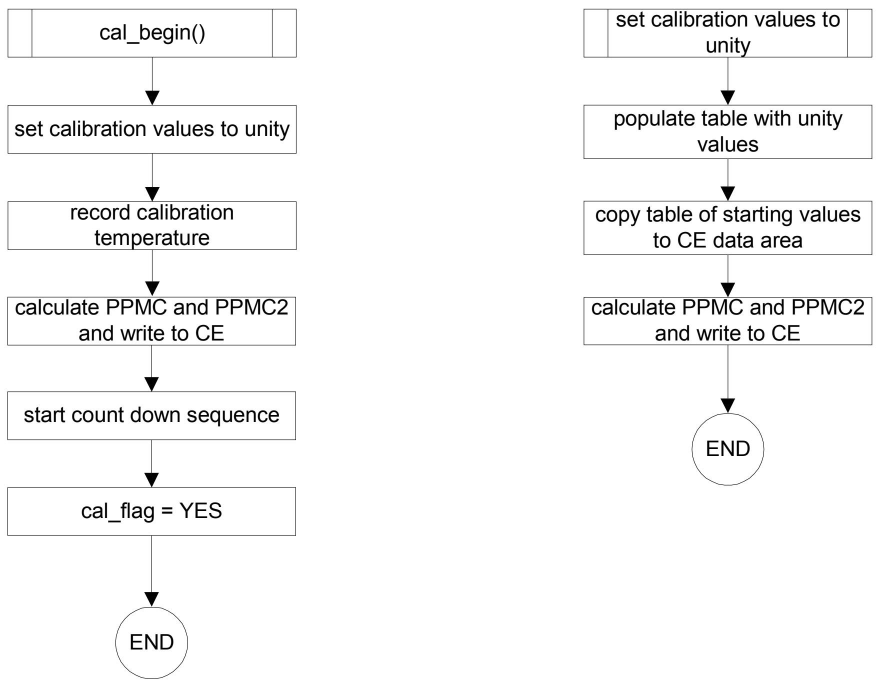


Figure 5-15: Auto-Calibration

The processing of the calibration steps is performed by the routine calibration(), which is called in ce_update() when new data becomes available, i.e. once per accumulation interval. The auto-calibration mechanism functions as a state-machine, sequenced by the variable "cs", which is used to count down accumulation intervals:

1) If cs > Scal: The state machine waits for the CE to settle after the unity gain and temperature compensation data are loaded in the routine cal_begin().
2) If cs = Scal: The variables for each cumulative voltage and current measurement are cleared.
3) If 0 <= cs <= Scal: For two accumulation intervals, prorated measurements of current and voltage are added to the variables. Using two accumulation intervals covers both chop polarities of temperature measurements.
4) If cs = 0: This signals the end of the calibration. Cumulative current and voltage measurements are then used to calculate and set the calibration coefficients for voltage and currents in CE DRAM.

CE Default Calibration

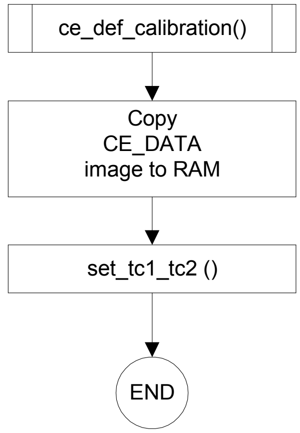


Figure 5-16: ce_default Calibration

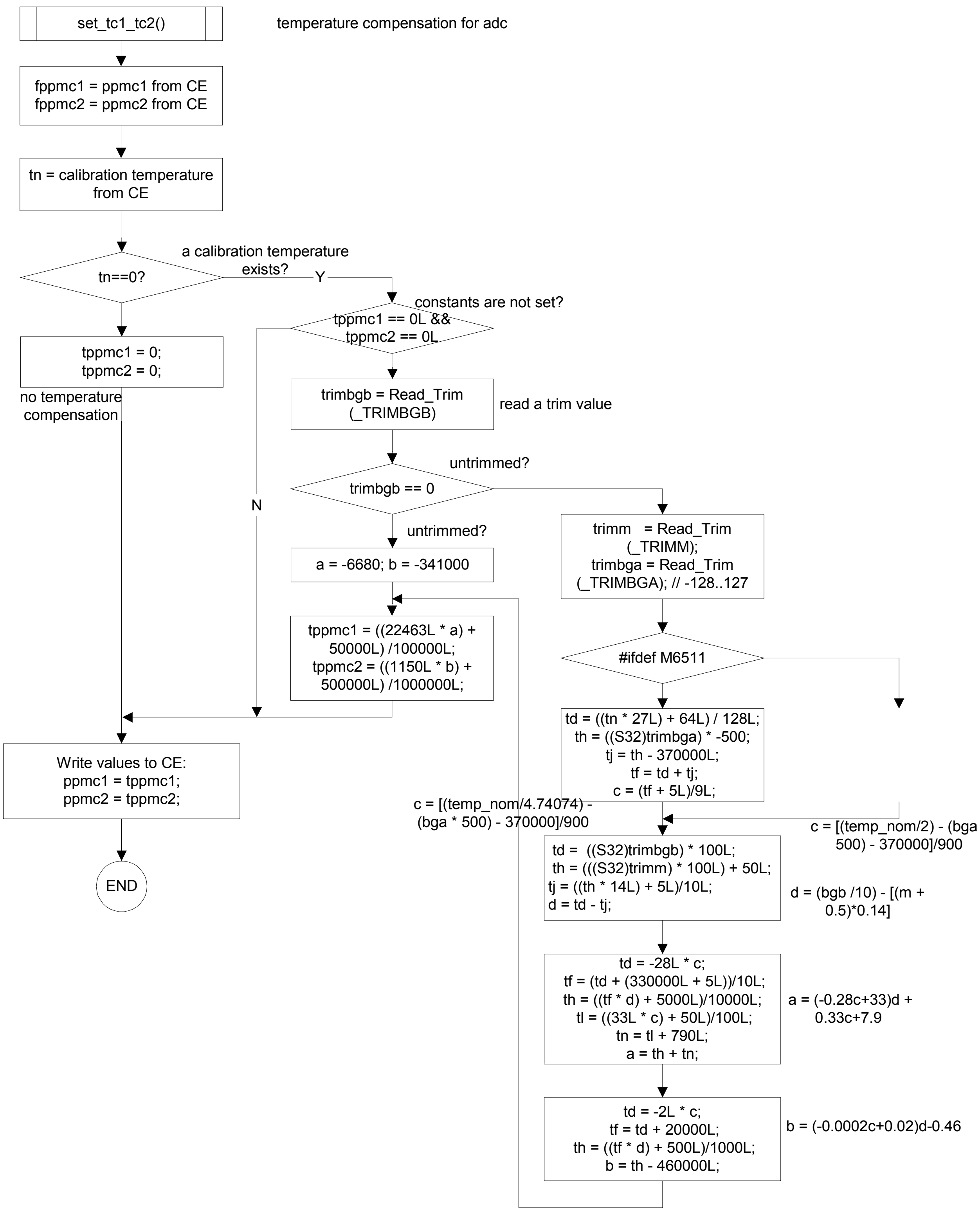


Figure 5-17: Calibration, continued

Command Pending

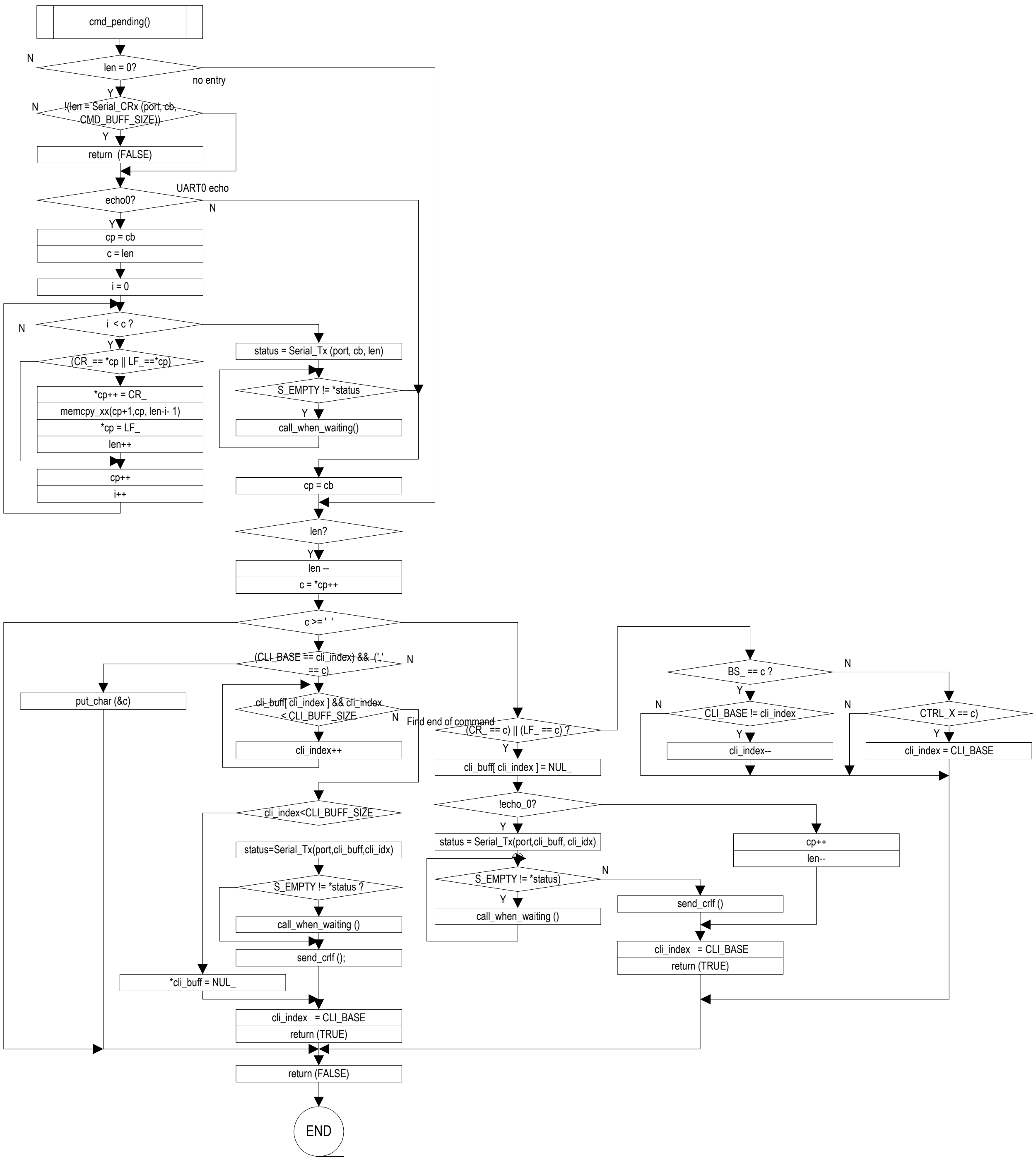


Figure 5-18: cmd_pending()

EEPROM Read/Write

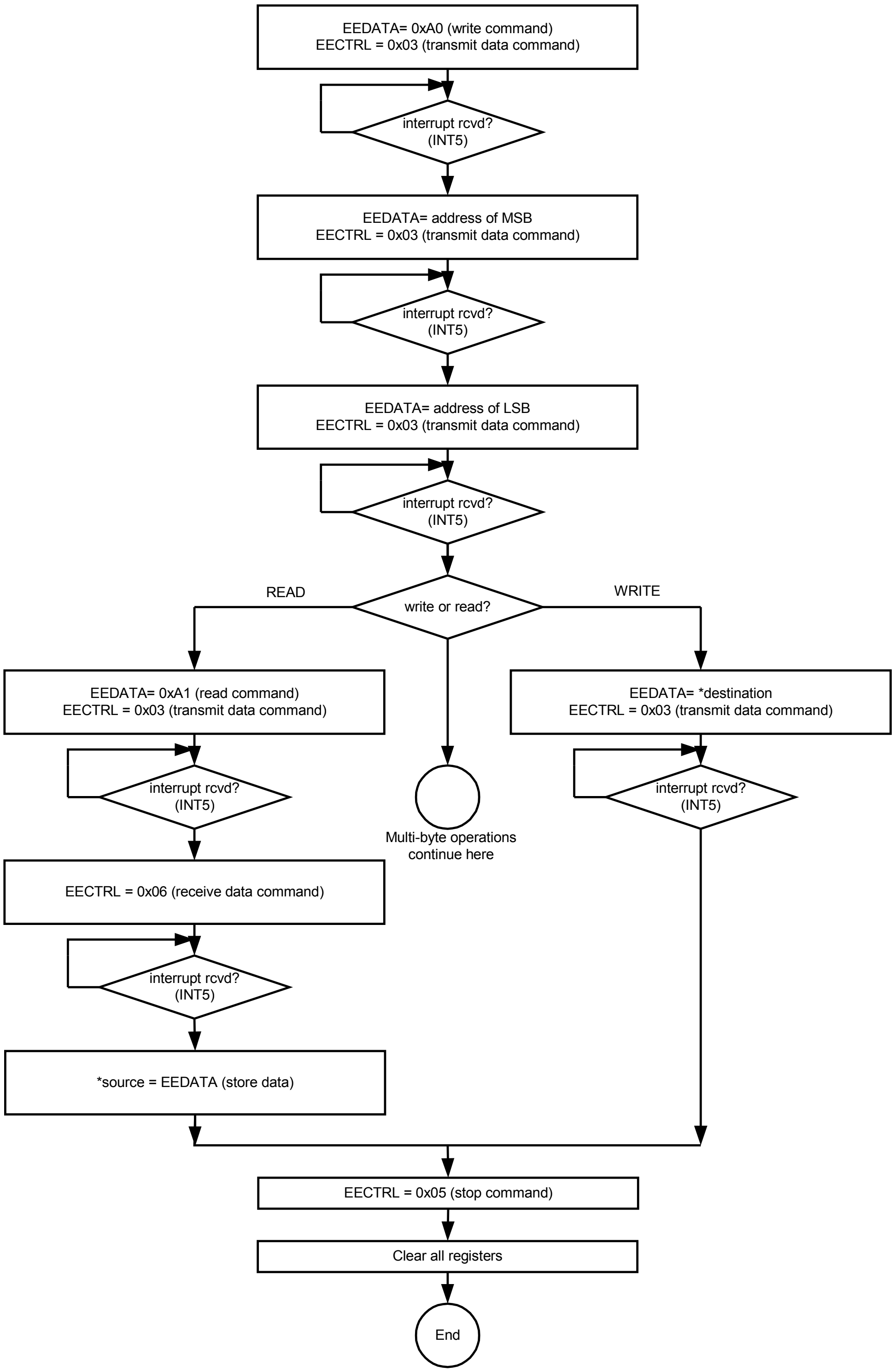


Figure 5-19: Single-Byte Read/Write

Registers and memory locations:

- EEDATA = SFR 0x9E
- EECTRL = SFR 0x9F
- *source = pointer to EEPROM address for read or write
- *destination = pointer to XRAM address
- count = byte count for multiple read/write

If the EEPROM interrupt service routine (INT5) returns the value 0x80 (illegal command), the loop should be exited, all registers should be refreshed and the operation should be restarted.

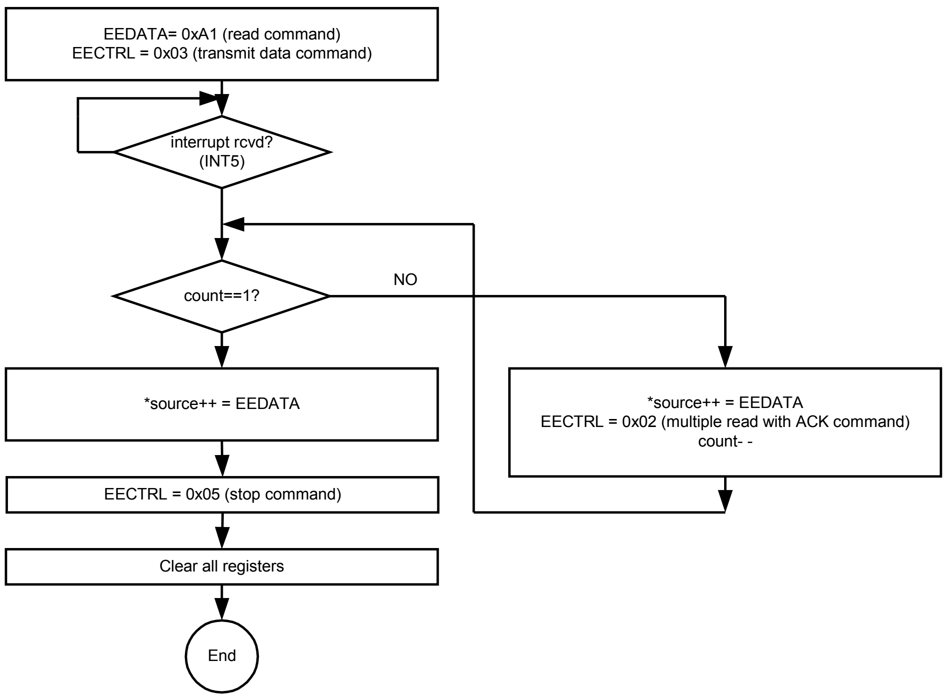


Figure 5-20: Multi-Byte Read

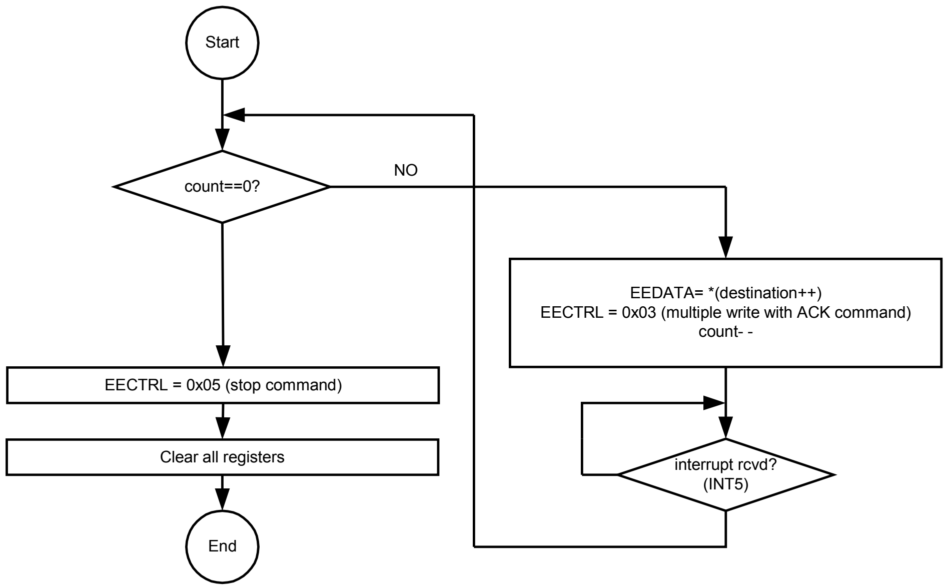


Figure 5-21: Multi-Byte Write

Notes:

- For larger EEPROMS 1010xxR can be the first command (R=1 for read, R = 0 for write operation).
- The START command should be sent to the EEPROM before any read or write operation
- The algorithms cover single and multi-byte operations limited to a single page.
- EEPROMs are organized in pages. In general, ATMEL EEPROMs have 1Kbyte per page (256 x 32 bits). When reading, no special requirements with respect to page boundaries apply.
- Special precautions apply when a page boundary is crossed for write operations: When the end of a page is reached, the write.to the next page has to be preceded by a START command.
- EEPROMs typically respond to START commands with 5ms delay.

Pulse Counters (Revision 3.05)

For the Demo Code Revision 3.05 only, a versatile pulse counter utility is available that is based on the interrupts generated by a high-to-low transition of the W and/or R pulse output pins (DIO pins DIO6 and DIO7).

The CLI command CPD defines a time interval (window) during which pulses are counted that are then displayed with the M14.2, M15.2 or M13.2 commands. We call these pulse counters "accumulated".

The CLI commands M14.1, M15.1, and M13.1 display the "absolute" pulse counts, i.e. the number of pulses detected regardless of the time window. These absolute pulse counts can be reset using the CLI command CPC.

Because pulses can appear at fairly high frequencies, the pulse counter routines had to be designed to be fast and efficient. At the core of the pulse counting mechanism are the two incremental pulse counters `iPulseW_Cnt` and `iPulseR_Cnt`, located in internal (indirect access) RAM for fastest access. As shown in the code sample below (taken from misc.c), these counters are incremented with in the interrupt service routines with each occurrence of the associated pulse.

```
void io_high_priority_isr (void) small reentrant
{
    iPulseW_Cnt++;
}

void io_low_priority_isr (void) small reentrant
{
    iPulseR_Cnt++;
}
```

`iPulseW_Cnt` and `iPulseR_Cnt` hold the pulse counts encountered in one second intervals only, and then are copied to XRAM-based counters.

Each second, when the `RTC_ACTION` macro is called in events.c, the incremental pulse counters `iPulseW_Cnt` and `iPulseR_Cnt` are copied to the XRAM-based pulse counters `xPulseW_Cnt` and `xPulseR_Cnt`. As shown below, the `RTC_ACTION` macro resets the incremental pulse counters to zero, in order to initialize them for the next one-second interval. The RTC interrupt is used to synchronize this 1-second event, since it is more accurate than the accumulation interval of the CE.

```
#define RTC_ACTION()
    IRQ_DISABLE();
    xPulseW_Cnt = iPulseW_Cnt;
    xPulseR_Cnt = iPulseR_Cnt;
    iPulseW_Cnt = iPulseR_Cnt = 0;
    pulse_count_available = 1;
    IRQ_ENABLE();
```

Again, each second, as controlled by the variable `pulse_count_available` (handled by the RTC interrupt in events.c), the routine `ce_update()` calls `count_pulses()`, where the XRAM-based pulse counters `xPulseW_Cnt` and `xPulseR_Cnt` are added to the final pulse counters `PulseW_Cnt` and `PulseR_Cnt`.

The `count_pulses()` routine is shown below.

The absolute pulse counters are updated unconditionally, while the accumulated pulse counters are only updated when the time window, as defined by the CPD command, is still open. This is reflected in the value of the variable `pulse_tmr`. The variable `capture_pulse_cnts` signals the start of pulse counting, i.e. it is only true immediately after the CPA command is issued via the CLI.

The accumulated pulse counters are determined by subtracting the start count from the final count:

```
sub8_8 (pulseW_cnt1.a, pulseW_cnt0.a);
```

The pulse counts to be displayed are stored in the variables `dPulseW_Cnt` and `dPulseR_Cnt`.

```
void count_pulses (void)
{
   static SUM xdata pulseW_cnt0, pulseW_cnt1;
   static SUM xdata pulseR_cnt0, pulseR_cnt1;
   static U16 pulse_tmr;

   add_2 (PulseW_Cnt, xPulseW_Cnt, 8);
   add_2 (PulseR_Cnt, xPulseR_Cnt, 8);

   if (capture_pulse_cnts)
   {
      memcpy_xx (pulseW_cnt0.a, PulseW_Cnt, sizeof (SUM)); // Sample at start.
      memcpy_xx (pulseR_cnt0.a, PulseR_Cnt, sizeof (SUM)); // Sample at start.
      capture_pulse_cnts = FALSE;
      pulse_tmr = Pulse_Duration;
   }
   else if (0 != pulse_tmr)
   {
      if (0 == --pulse_tmr)
      {
       memcpy_xx (pulseW_cnt1.a, PulseW_Cnt, sizeof (SUM)); //Sample at end.
       memcpy_xx (pulseR_cnt1.a, PulseR_Cnt, sizeof (SUM)); //Sample at end.

       sub8_8 (pulseW_cnt1.a, pulseW_cnt0.a);
       sub8_8 (pulseR_cnt1.a, pulseR_cnt0.a);

       memcpy_xx((U08x *) &dPulseW_Cnt, &pulseW_cnt1.a[4], sizeof
(dPulseW_Cnt));
       memcpy_xx((U08x *) &dPulseR_Cnt, &pulseR_cnt1.a[4], sizeof
(dPulseR_Cnt));
      }
   }
}
```

5.3.4. Watchdog Timer

The Demo Code revision 3.04 uses only the hardware watchdog timer provided by the 80515. This fixed-duration timer is controlled with SFR register WDI (0xE8).

The software watchdog timer is described in section 6.3.4, but should not be used. The hardware watchdog timer is more reliable since it cannot be accidentially disabled.

The hardware watchdog timer requires a refresh by the MPU firmware, i.e. bit 7 of WDI set, at least every 1.5 seconds. If this refresh does not occur, the hardware watchdog timer overflows, and the 80515 is reset as if RESETZ were pulled low. When overflow occurs, the bit *WD_OVF* is set in the configuration RAM. Using the *WD_OVF* bit, the MPU can determine whether a reset or a hardware watchdog timer overflow occurred. The *WD_OVF* bit is cleared when RESETZ is pulled low.

The bits of the WDI register (SFR 0xE8) should not be individually set or reset. Instead, byte operations should be used.

The following macro code should be used for resetting (clearing) the watchdog, IE_RTC or IE_XFER bits:

```
#define WD_RST_          0xFF          // WatchDog bit.
#define IE_RTC_          0x02          // RTC ticked.
#define IE_XFER_         0x01          // XFER data available.

#define RESET_WD()       WDI = WD_RST_;
#define CLR_IE_XFER()    WDI = ~IE_XFER_ & 0x7F; // 0x7E
#define CLR_IE_RTC()     WDI = ~IE_RTC_  & 0x7F; // 0x7D
```

5.3.5. Real-Time Clock (RTC)

The RTC is accessible through the I/O RAM (Configuration RAM) registers RTC_SEC through RTC_YR (addresses 0x2015 through 0x201B), as decribed in the data sheets.

Since the RTC runs on a much slower clock than the MPU, only one write operation can be performed per RTC clock cycle. This means that write operations to set the RTC must be separated by at least 396us. The sample code uses hardware timer 1 to perform this delay, so any code modification must make sure that hardware timer 1 is still useable for the RTC functions.

5.4. DATA FLOW

The ADC collects data from the electrical inputs on a cycle that repeats at 2520Hz. On each ADC cycle, the compute engine (CE) code digitally filters and adjusts the data using gain parameters (CAL_Ix, CAL_Vx) and phase adjustment parameters (PHADJ_x). Also, it adjusts for temperature using the linear (PPMC) and squared (PPMC2) temperature gain coefficients and a nominal temperature of calibration (TEMP_NOM).

Normally, a calibration operation during manufacturing finds these adjustments and stores them in flash or EEPROM to be placed into CE memory. The Demo Code includes a basic linear self-calibration function that can typically reach 0.05% accuracy. (ce.c: ce_update(), cal.c: cal_begin(), calibration()).

Better calibration schemes are routinely possible. The calibration save and restore operations (cal_save() and cal_restore()) save and restore all adjustment variables, such as the constants for the real-time clock, not just the ones for electrical measurements.

On each ADC cycle, 2520 times per second, the CE performs the following tasks:

1. It calculates intermediate results for that set of samples.
2. It runs a debounced check for sagging mains, with a configurable debounce.
3. It has three equally-spaced opportunities to pulse each pulse output.

On each ADC cycle, an MPU interrupt, "ce_busy" (see ce.c, ce_busy_isr()) is generated. Normally, the interrupt service routine checks the CE's status word for the sag detection bits, and begins sag logic processing if a sag of the line voltage is detected.

In the event of a sag detection, the cumulative quantities in memory could be written to the EEPROM. The demo code intentionally omits this sag logic, because that varies by application, and might actually interfere with demonstration. It does, however, arrange to keep valid CRC-protected values (see ce.c Accumulate_Energy()) in battery-backed RAM, which permits the demo system to become a simple functional meter with the addition of a suitable battery.

By the end of each accumulation interval, each second on the demo code, the CE performs the following tasks:

1. It calculates deviation from nominal calibration temperature (TEMP_X).
2. It calculates the frequency on a particular phase (FREQ_X).
3. It calculates watt hours (Wh) for each conductor, and the meter (WxSUM_X).
4. It calculates var hours (VARh) for each phase and the meter (VARxSUM_X).
5. It calculates summed squares of currents for each phase (IxSQSUM_X).
6. It calculates summed squares of voltages for each phase (VxSQSUM_X).
7. It calculates lags between different phases (on 3-phase meters) (PH_Atox).
8. It counts zero crossings on the same phase as the frequency (MAINEDGE_X).

The CE code (see ce1x.c) digitally filters out the line frequency component of the signals, eliminating any long-term inaccuracy caused by heterodyning between the line frequency and the sampling or calculation rates. This also permits a meter to be used at 50 or 60Hz, or with inaccurate line frequencies.

The CE has several equations of calculation, so that it can calculate according to the most common methods.

Once per accumulation interval, the MPU requests the CE code to fetch an alternative measurement (alternate multi-plexer cycle).

At the end of each accumulation interval, an MPU interrupt, the "xfer_interrupt" occurs (see ce.c, xfer_busy_isr()) occurs. This is the signal for the MPU to copy the above data to stable storage for further use.

At this time, the MPU performs creep detection (ce.c Apply_Creep()). If the current or the accumulated energy (watt hours) are below the minimum, no current or watts are reported. If volts are below the threshold, no frequency or edge counts are reported. The MPU's creep thresholds are configurable (CREEP_THR, VThrshld, IThrshld).

The MPU calculates human-readable values, and accumulates cumulative quantities (see ce.c, ce_update()). The MPU scales these values to the PCB's voltage and current sensors (see VMAX and IMAX).

Watt hours and Var hours are signed, permitting the MPU to perform net metering by assigning negative values to "export" and positive values to "import" (see ce.c. WSum_Isrc(), WSum_ESrc(), VarSum_Isrc() and VarSum_Esrc()).

At very low currents, the watt hours have a lower noise-floor than the direct current measurements. The Demo Code includes a mode to detect this case, and use the alternate low-noise calculations. In this mode, with disabled (zero) creep thresholds, it measures current down to a few milliamps (ce.c, ce_updated(), Compute_Small_Irms()).

The multiple-precision calculations needed for a meter require more precision than standard C floating point provides. The Demo Code has many reusable extended-precision calculations (classics.c, library.c), including a square-root that takes a 64-bit number for accurate calculation of VA (library.c).

The MPU also places a scaled value into the CE RAM for each pulse output (ce.c, see xfer_busy_isr(), Pulse_Src_Func[]). This adjusts the pulse output frequency in such a way as to reflect that accumulation's contribution to the total pulse interval. Pulse intervals are cumulative, and cumulatively accurate, even though the frequency is updated only periodically.

Placing the pulse value selection logic into the MPU software means that any quantity from any phase or combination of phases can control either pulse output (see Pulse_Src_Func[] for a list of transfer functions).

The MPU also performs temperature adjustments of the real-time clock (rtc.c, RTC_Trim(), RTC_Adjust()). The Demo Code can adjust the clock speed to a resolution of 1 part per billion, roughly one second per thirty years. The adjustments include offset (Y_CAL), temperature-linear (Y_CALC) and temperature-squared (Y_CALC2) parameters.

Once a human-readable quantity is available, it can be translated into a set of segments (display.c, lcd.c) to display on the liquid crystal display, or read from a register in memory by means of the command-line interface (cli.c), or possibly some other serial protocol such as Flag or NEMA.

5.5. CE/MPU INTERFACE

The interface between the CE and the MPU is described completely in the 71M6511 and 71M6513 Data Sheets.

5.6. SOURCE FILES

The functionality of the Demo Code is implemented in the following files and directories:

1. **CLI:** **Command Line Interface – General Commands**
   access.c SFR, I/O RAM, MPU and CE data access
   cli.c parser for command line interface
   cmd_ce.c sub-parser for CE commands
   cmd_misc.c sub-parser for RTC, EEPROM, trim and PS commands
   io.c number conversion functions and auxiliary routines for CLI
   load.c upload and download
   sfrs.c access to SFRs
   When compiled, CLI.C takes about 20Kbytes of program space. When designing a real meter, CLI.C can easily be removed without major changes to the software.

2. **CLI_6513** **Command Line Interface, 6513-Specific**
   help.c display of help text
   profile.c data collection for support of profile command

3. **CLI_6511** **Command Line Interface, 6511-Specific**
   help.c display of help text
   profile.c data collection for support of profile command

4. **IO:** **Input/Output**
   eeprom.c interrupt-driven serial EEPROM routines
   eepromp.c high-speed polling EEPROM routines
   iicdio.c I2C bus interface using direct control of DIO4 and DIO5
   iiceep.c I2C bus interface using the chip's I2C hardware
   lcd.c initialization, configuration, read and write routines for LCDs
   lcd_VIM808.c routines for driving Varitronix VIM-808 LCS
   rtc.c RTC read, write, reset, and trim routines
   serial.c initialization, configuration, flow-control, interrupt service routines for serial ports

5. **Main_6513:** **Main top-level tasks, 6513-specific**
   defaults.c meter initialization with default values
   demo.c main() with startup sequence and main task switch
   display.c top-level display routines

6. **Main_6511:** **Main top-level tasks, 6511-specific**
   defaults.c meter initialization with default values
   demo.c main() with startup sequence and main task switch
   display.c top-level display routines

7. **Meter:** **Metering Functions**
   cal.c auto-calibration
   ce.c compute metering values from data provided by CE
   ce651X.c data exchange between CE data RAM and XRAM
   io651X.c control of analog front end, multiplexer, RTM, I/O pins
   misc.c interrupts, reset, port configuration, MPU power management

8. **Util:** **Utilities**
   classics.c math routines (add, subtract, multiply, divide, square
   events.c event management
   flash.c flash memory read, write, erase, compare and checksum calculation
   library.c memory copy, compare, CRC calculation, string length, square root and shift
   onek_c.asm assembly test program exercising I/O pins and testing RAM, ROM, CE DRAM, CE PRAM
   timers.c timer configuration, delay

5.7. AUXILIARY FILES

A variety of startup files is provided with the Demo Kits. The function of these files is as follows:

1. STARTUP.A51:
   This file provides memory and stack initialization. It is part of the Keil compiler package.

2. STARTUP_SECURE.A51:
   This file is almost identical to STARTUP.A51. The only difference is that this variation sets the *SECURE* bit. This bit enables security provisions that prevent external reading of flash memory and CE program memory. The code segment below sets the security bit located at SFR register address 0xB2:

```
STARTUP1:
        CLR     0xA8^7          ; Disable interrupts
        MOV     0B2h,#40h      ; Set security bit.
        MOV     0E8h,#0FFh     ; Refresh nonmaskable watchdog
```

3. INIT.A51:
   A secondary startup file. It is part of the Keil compiler package. This code is executed, if the application program contains initialized variables at file level.

5.8. INCLUDE/HEADER FILES

- API.H - Common basic library function definitions used for the API, such as soft_reset(), API_init(), ce_config(), etc. The library routines can be called without prior knowledge of the hardware.
- API_struct.H - API structures, enumerates, and defines.
- ce651x.H - CE data and structure declarations
- cli.H - Result code and Common ASCII code definition used for CLI
- demo_options.H - _DEMO_OPTIONS declaration
- help.H - HELP messages prototype declarations
- iic.H - IIC API declaration for IIC control
- io651x.H - 651x register definitions.
- options.H - Options selected for compile time.
- portable.H - System and data format definition.
- reg_banks.H - Compile time register bank usage
- reg651x.H - Register definition of 651x SFRs and IOs
- reg80515.H - Register definition of 80515 SFRs and Internal memory.

5.9.CE IMAGE FILES

The CE code uses pre-designed, pre-validated algorithms and calculations that are accurate to the noise floor of the integrated circuit, saving substantial engineering and development time.

The source code for the CE is proprietary. Only the code and data images (binary images) are available to the user. The code image must be merged with the MPU code residing in flash memory. Before enabling the CE program, the MPU has to upload the code and data images to the CE PRAM and DRAM, as described in the 71M6511 and 71M6513 Data Sheets.

Images of the CE data and program code are provided with the Demo Kits. They are to be linked into the object code. CE images are provided by the following files:

1. CE11C_CE.C:
   This file provides the image of the 6511 CE program in C notation.
2. CE11C_DAT.C:
   This file provides the image of the 6511 CE default data in C notation.
3. CE13B_CE.C:
   This file provides the image of the 6513 CE program in C notation.
4. CE13B_DAT.C:
   This file provides the image of the 6513 CE default data in C notation.
5. CE13B_ROG_CE.C:
   This file provides the image of the 6513 CE program for Rogowski coil application in C notation.
6. CE13B_ROG_DAT.C:
   This file provides the image of the 6513 CE default data for Rogowski coil application in C notation.

5.10.COMMON MPU ADDRESSES

In the Demo Code, certain MPU XRAM parameters have been given fixed addresses in order to permit easy external access. These variables can be read via the serial interface, with the **)n$** command and written with the **)n=xx** command where n is the word address. Note that accumulation variables are 64 bits long and are accessed with **)n$$** (read) and **)n=hh=ll** (write) in the case of accumulation variables.

MPU INPUT PARAMETERS

The parameters listed in Table 5-3, Table 5-4, and Table 5-5 are loaded by the MPU at startup and should not need adjustment during meter calibration.

XRAM Word Address	Default Value	Name	Description
0x00	8311 (6511) 1536 (6513)	*CREEP_THR*	For each element, if WSUM_X or VARSUM_X of that element exceeds CREEP_THR, the sample values for that element are not zeroed. Otherwise, the accumulators for Wh, VARh, and VAh are not updated and the instantaneous value of IRMS for that element is zeroed. 6511: LSB = $6.6952 \cdot 10^{-13}$ VMAX IMAX Wh 6513: LSB = $9.4045 \cdot 10^{-13}$ VMAX IMAX Wh **Demo Code revision 3.05 offers a separate set of creep-related variables corresponding to phase B.**
0x01	0	*CONFIG*	Bit 0: Sets VA calculation mode. 0: $V_{RMS} * A_{RMS}$ 1: $\sqrt{W^2 + VAR^2}$ Bit 1: Clears accumulators for Wh, VARh, and VAh. This bit need not be reset.

XRAM Word Address	Default Value	Name	Description
0x02	191181742 (6511) 136105056 (6513)	*PK_VTHR*	Demo Code revision 3.04: Not implemented. **Demo Code revision 3.05:** When the voltage exceeds this value, bit 5 in the MPU status word is set, and the MPU might choose to log a warning. Event logs are not implemented in Demo Code. 6511: LSB = $6.6952 \cdot 10^{-13} \cdot VMAX^2\ V^2h_{RMS}$ 6513: LSB = $9.4045 \cdot 10^{-13} \cdot VMAX^2\ V^2h_{RMS}$ The default value is equivalent to $407.3V_{RMS}$ if VMAX = 600V and a 1-second accumulation interval is used.
0x03	24856631 (6511) 17695797 (6513)	*PK_ITHR*	Demo Code revision 3.04: Not implemented. **Demo Code revision 3.05:** When the current exceeds this value, bit 6 in the MPU status word is set, and the MPU might choose to log a warning. Event logs are not implemented in Demo Code. 6511: LSB = $6.6952 \cdot 10^{-13} \cdot IMAX^2\ V^2h_{RMS}$ 6513: LSB = $9.4045 \cdot 10^{-13} \cdot VMAX^2\ V^2h_{RMS}$ The default value is equivalent to $50.9A_{RMS}$ if IMAX = 208A and a 1-second accumulation interval is used.
0x04	0	*Y_CAL*	Implement RTC trim. $CORRECTION(ppm) = \frac{Y\_CAL}{10} + T \cdot \frac{Y\_CALC}{100} + T^2 \cdot \frac{Y\_CALC2}{1000}$
0x05	0	*Y_CALC*	
0x06	0	*Y_CALC2*	
0x09	6000	*VMAX*	The nominal external RMS voltage that corresponds to 250mV pk at the ADC input. The meter uses this value to convert internal quantities to external. LSB=0.1V
0x0A	2080	*IMAX*	The nominal external RMS current that corresponds to 250mv pk at the ADC input. The meter uses this value to convert internal quantities to external. LSB=0.1A
0x0B	0	*PPMC*	PPM/C*26.84. Linear temperature compensation. A positive value will cause the meter to run faster when hot. This is applied to both V and I and will therefore have a double effect on products.
0x0C	0	*PPMC2*	$PPM/C^2 \cdot 1374$. Square-law compensation. A positive value will cause the meter to run faster when hot. This is applied to both V and I and will therefore have a double effect on products.
0x0D	9585 (6511) 22721 (6513)	*DEGSCALE*	Scale factor for TEMP_X. TEMP_X=*DEGSCALE*$\cdot 2^{-22}\cdot$(TEMP_RAW_X - TEMP_NOM)
0x13	61 (6511) 16 (6513)	*ICREEP*	For each element, if the current of that element is below *ICREEP*, the sample values for that element are zeroed. In that case, the accumulators for Wh, VARh, and VAh are not updated and the instantaneous value of IRMS for that element is zeroed. The default values correspond to 80mA, if the default setting for IMAX is used. 6511: LSB = $6.6952 \cdot 10^{-13}\ IMAX^2$ Wh 6513: LSB = $9.4045 \cdot 10^{-13}\ IMAX^2$ Wh

Table 5-3: MPU Input Parameters

0x07 0x08	1 5	*PULSEW_SRC* *PULSER_SRC*	PULSEW source and PULSER source. _I refers to imported by the consumer. _E refers to power exported by the consumer. Values are: 0 – reserved **1 - *W0SUM*** 2 - *W1SUM* 3 - reserved 4 – reserved **5 - *VAR0SUM*** 6 - *VAR1SUM* 7 - reserved 8 - *I0SQSUM* 9 - *I1SQSUM* 10 - reserved 11 – reserved 12 - *V0SQSUM* 13 – 15 reserved 16 - *VA0SUM* 17 - *VA1SUM* 18 - reserved 19 – reserved 20 – *W0SUM_I* 21 – *W1SUM_I* 22 – reserved 23 – reserved 24 – *VAR0SUM_I* 25 – *VAR1SUM_I* 26 – reserved 27 – reserved 28 – *W0SUM_E* 29 – *W1SUM_E* 30 – reserved 31 – reserved 32 – *VAR0SUM_E* 33 – *VAR1SUM_E* 34 – reserved

Table 5-4: Pulse Source Parameters (6511)

Number	Pulse Source	Description
0	**WSUM**	***Default for PULSEW_SRC***
1	W0SUM	
2	W1SUM	
3	W2SUM	
4	**VARSUM**	***Default for PULSER_SRC***
5	VAR0SUM	
6	VAR1SUM	
7	VAR2SUM	
8	I0SQSUM	
9	I1SQSUM	
10	I2SQSUM	
11	INSQSUM	
12	V0SQSUM	
13	V1SQSUM	
14	V2SQSUM	
15	VASUM	
16	V0ASUM	
17	V1BSUM	

Number	Pulse Source	Description
18	V2SUM	
19	WSUM_I	Sum of imported real energy
20	*W0SUM_I*	Imported real energy on element A
21	*W1SUM_I*	Imported real energy on element B
22	*W2SUM_I*	Imported real energy on element C
23	VARSUM_I	Sum of imported reactive energy
24	*VAR0SUM_I*	Imported reactive energy on element A
25	*VAR1SUM_I*	Imported reactive energy on element B
26	*VAR1SUM_I*	Imported reactive energy on element C
27	WSUM_E	Sum of exported real energy
28	*W0SUM_E*	Exported real energy on element A
29	*W1SUM_E*	Exported real energy on element B
30	*W2SUM_E*	Exported real energy on element C
31	VARSUM_E	Sum of exported reactive energy
32	*VAR0SUM_E*	Exported reactive energy on element A
33	*VAR1SUM_E*	Exported reactive energy on element B
34	*VAR2SUM_E*	Exported reactive energy on element C

Table 5-5: Pulse Source Parameters (6513)

MPU INSTANTANEOUS OUTPUT VARIABLES

The Demo Code processes CE outputs after each accumulation interval. It calculates instantaneous values such as VRMS, IRMS, W and VA as well as accumulated values such as Wh, VARh, and VAh. Table 5-7 lists the calculated instantaneous values for single-phase ICs.

Accumulated values are calculated by summing the CE XFER outputs into 64-bit variables. Thus, the accumulators will hold at least 136 years of data when XFER rate is 1Hz. The values calculated by the MPU are in XRAM according to Table 5-10. Table 5-8 lists the calculated instantaneous values for poly-phase ICs.

XRAM Word Address	Name	Description
0x14 0x15 0x16	Vrms_A reserved reserved	Vrms: $LSB = \frac{3.7610 \cdot 10^{-8} VMAX}{\sqrt{Nacc}}$
0x17 0x18 0x19	Irms_A Irms_B reserved	Irms from element 0, 1, 2. $LSB = \frac{3.7610 \cdot 10^{-8} IMAX\ In8}{\sqrt{Nacc}}$
0x1A 0x1B 0x1C	IPhase_A reserved reserved	Phase between voltage and current. The number of degrees I lags V. LSB=0.001°, range: 0…+360
0x1D	Frequency	Frequency of the voltage signal selected by the CE input. If the selected voltage is below the sag threshold, Frequency = 0. LSB $\equiv \frac{F_S}{2^{32}} \approx 0.587 \cdot 10^{-6}$ Hz
0x1E	Delta_T	Deviation from Calibration temperature. LSB = 0.1 ^{0}C.
0x1F 0x20	reserved reserved	
0x21	Status	MPU Status Word
0x22	OperatingTime	Total operating time. LSB = 0.01h = 36s
0x23	Reserved	

Table 5-6: MPU Instantaneous Output Variables (6511)

XRAM Word Address	Name	Description
0x14 0x15 0x16	Vrms_A Vrms_B* Vrms_C*	V_{rms} from element 0, 1, 2. $LSB = \frac{4.4575 \cdot 10^{-8} VMAX}{\sqrt{Nacc}}$
0x17 0x18 0x19	Irms_A Irms_B Irms_C*	I_{rms} from element 0, 1, 2. $LSB = \frac{4.4575 \cdot 10^{-8} IMAX\ In8}{\sqrt{Nacc}}$
0x1A 0x1B 0x1C	IPhase_A IPhase_B* IPhase_C*	In-Vn phase from element n. The number of degrees In lags Vn. LSB=0.001°. Range = 0 to +360.
0x1D	Frequency	Frequency of voltage selected by CE input. If the selected voltage is below the sag threshold, Frequency=0. LSB $\equiv \frac{F_S}{2^{32}} \approx 0.587 \cdot 10^{-6}$ Hz
0x1E	Delta_T	Deviation from Calibration temperature. LSB = 0.1 ^{0}C.
0x1F 0x20	VPhase_AB* VPhase_BC*	Amount phase B lags phase A and amount phase B lags phase C. LSB=1° (0,360). If V_{rms}_A<5% of VMAX, VPhase_AB and VPhase BC are cleared to 0.
0x21	Status	MPU Status Word

Table 5-7: MPU Instantaneous Output Variables (6513)

MPU STATUS WORD

The MPU maintains the status of certain meter and I/O related variables in the Status Word. The Status Word is located at address 0x21. The bit assignments are listed in Table 5-8.

Status Word Bit	Name	Description
0	reserved	
1	SAGA	Reserved for sag flag phase A – not implemented[1]
2	SAG B	Reserved for sag flag phase B (6513 only) – not implemented[1]
3	SAG C	Reserved for sag flag phase C (6513 only) – not implemented[1]
4	F0	Reserved for reconstructed line signal flag – not implemented[1]
5	MAXV	Overvoltage. 1 when overvoltage is detected.
6	MAXI	Overcurrent. 1 when overcurrent is detected.
7	ONE_SEC	Reserved for one-per-second flag – not implemented[1]
8	VXEDGE	Reserved for DIO port transition – not implemented[1]
9	reserved	
10	reserved	
11	XFER	Reserved – not implemented[1]
12	CREEP	Creep bit. 1 when creep is detected. The creep bit ios only set if creep is detected in all channels (elements).
13-31	reserved	

Note: [1]: Not implemented in Demo Code revision 3.04

Table 5-8: MPU Status Word Bit Assignment

MPU ACCUMULATION OUTPUT VARIABLES

Accumulation values are accumulated from XFER cycle to XFER cycle (see Table 5-9 and Table 5-10). They are all in 64-bit format. The 6511 has an LSB of $6.6925 \times 10^{-13} \times VMAX \times IMAX \times In\_8$ Wh.

Accumulated values are calculated by summing the CE XFER outputs into 64 bit variables. Thus, the accumulators will hold at least 136 years of data when XFER rate is 1Hz. The values calculated by the MPU are in XRAM according to the following table.

XRAM Word Address	Name	Description
0x2F	reserved	
0x31	reserved	
0x33	reserved	
0x35	reserved	
0x37	reserved	
0x39	*Wh_A*	Total Watt hours consumed through phase A[1)]
0x3B	*Whe_A*	Total Watt hours generated (inverse consumed) through phase A[1)]
0x3D	*VARh_A*	Total VAR hours consumed through phase A[2)]
0x3F	*VARhe_A*	Total VAR hours generated (inverse consumed) through phase A[2)]
0x41	*VAh_A*	Total VA hours in phase A[3)]
0x43	*Wh_B*	Total Watt hours consumed through phase B[1)]
0x45	*Whe_B*	Total Watt hours generated (inverse consumed) through phase B[1)]
0x47	*VARh_B*	Total VAR hours consumed through phase B[2)]
0x49	*VARhe_B*	Total VAR hours generated (inverse consumed) through phase B[2)]
0x4B	*VAh_B*	Total VA hours in phase B[3)]
0x4D	reserved	
0x4F	reserved	
0x51	reserved	
0x53	reserved	
0x55	reserved	

[1)]: If *EQU* = 0, Wh_A = real component of IA*VA and Wh_B = real component of IB*VA.
If *EQU* = 1, Wh_A = real component of VA * (IA – IB)/2 and Wh_B = real component of VA*IB.

[2)]: If *EQU* = 0, VARh_A = reactive component of IA*VA and VARh_B = reactive component of IB*VA.
If *EQU* = 1, VARh_A = VA * (IA – IB)/2 and Wh_B = VA*IB.

[3)]: If *EQU* = 0, VAh_A = apparent power based on IA*VA and VAh_B = apparent power based on IB*VA.
If *EQU* = 1, VAh_A = apparent power based on VA * (IA – IB)/2 and VAh_B = apparent power based on VA*IB.

Table 5-9: MPU Accumulation Output Variables (6511)

XRAM Address	Name	Description
0x2F	Wh	Total Watt hours consumed (imported)
0x31	Whe	Total Watt hours generated (exported)
0x33	VARh	Total VAR hours consumed
0x35	VARhe	Total VAR hours generated (inverse consumed)
0x37	VAh	Total VA hours
0x39	Wh_A	Total Watt hours consumed through element 0
0x3B	Whe_A	Total Watt hours generated (inverse consumed) through element 0
0x3D	VARh_A	Total VAR hours consumed through element 0
0x3F	VARhe_A	Total VAR hours generated (inverse consumed) through element 0
0x41	VAh_A	Total VA hours in element 0
0x43	Wh_B	Total Watt hours consumed through element 1
0x45	Whe_B	Total Watt hours generated (inverse consumed) through element 1
0x47	VARh_B	Total VAR hours consumed through element 1
0x49	VARhe_B	Total VAR hours generated (inverse consumed) through element 1
0x4B	VAh_B	Total VA hours in element 1
0x4D	Wh_C	Total Watt hours consumed through element 2
0x4F	Whe_C	Total Watt hours generated (inverse consumed) through element 2
0x51	VARh_C	Total VAR hours consumed through element 2
0x53	VARhe_C	Total VAR hours generated (inverse consumed) through element 2
0x55	VAh_C	Total VA hours in element 2

Table 5-10: MPU Accumulation Output Variables (6513)

MPU VARIABLES INDEPENDENTLY CONTROLLING THE SECOND CURRENT CHANNEL

The 6511 Demo Code revision 3.05 offers independent scaling and creep suppression parameters for the second current channel (phase B). This is useful for meter configurations where different sensors are used for the primary and secondary channel. An example for this configuration would be a meter using a CT for phase A and an additional resistive shunt for phase B to counter tampering. In most cases, the input voltage at the IA and IB inputs generated by the current flowing through both sensors will not be identical due to the differing characteristics of the sensors. In order to enable comparative measurements and true application of tariffs, Demo Code 6511 revision 3.05 offers a set of variables used to scale the IB channel (phase B).

The variables added in 6511 Demo Code revision 3.05 are shown in Table 5-11.

Note that the values for *IMAX2*, *ICREEP2*, and *WCREEP2* are closely coupled: When entering a different value for *IMAX2*, *ICREEP2* and *WCREEP2* should be changed accordingly, if the accuracy of the creep detection is of interest.

XRAM Word Address	Default Value	Name	Description
0x2B	2080	*IMAX2*	The nominal external RMS current that corresponds to 250mV peak at the IB input. The meter uses this value to convert internal quantities to external. LSB=0.1A
0x2C	61	*ICREEP2*	If the squared current (ISQSUM from the CE) of phase B is below *ICREEP2*, the sample values for phase B are zeroed. In that case, the accumulators for Wh, VARh, and VAh are not updated and the instantaneous value of IRMS for phase B is zeroed. The default value corresponds to $0.00636A^2$ (80mA), if the default setting for *IMAX2* is used. LSB = $6.6952*10^{-13}$ $IMAX2^2$ Wh
0x2D	8311	*WCREEP2*	If *WSUM_2* or *VARSUM_2* are below *WCREEP2*, the sample values for phase B are zeroed. In that case, the accumulators for Wh, VARh, and VAh are not updated and the instantaneous value of IRMS for phase B is zeroed. The default value corresponds to 2.5W, if the default settings for *VMAX* and *IMAX2* are used. LSB = $6.6952*10^{-13}$ *VMAX IMAX2* Wh

Table 5-11: MPU Variables Related to Phase B (6511, Revision 3.05 only)

5.11.FIRMWARE APPLICATION INFORMATION

5.11.1.Sag Detection

A sag is defined as an undervoltage condition that persists for more than one period. A shorter undervoltage condition is called a dip (see Figure). The occurrence of sags can announce an impending loss of power. Since accumulated energy values etc. in the meter will have to be saved to non-volatile memory in the case of loss of power, a sag can be used to initiate data saving operations. Some applications may instead save or count the sag event for the purpose of recording power quality data.

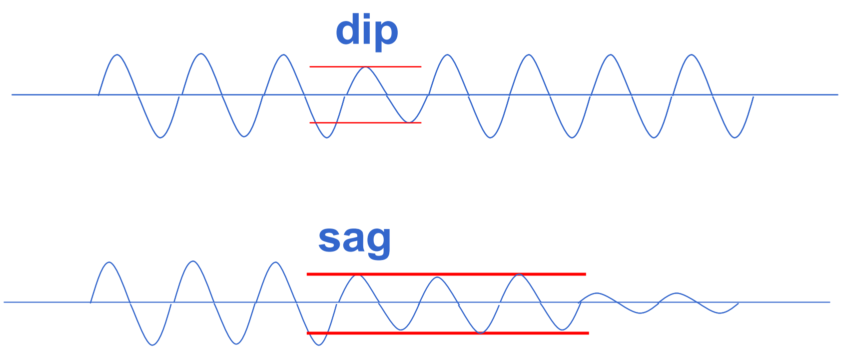


Figure 5-22: Sag and Dip Conditions

Sag detection is performed by the CE, based on the CE DRAM registers SAG_THR and SAG_CNT. SAG_THR defines the threshold which the input voltage has to be continuously below, and SAG_CNT defines the number of samples required to trigger the sag bit (see Figure 5-23).

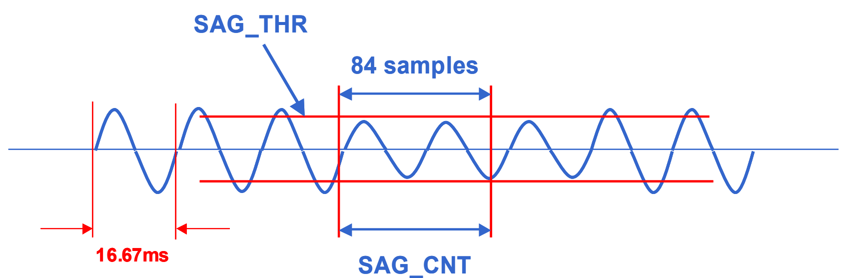


Figure 5-23: Sag Event

When the CE detects a sag that meets the sag conditions specified in SAG_THR and SAG_CNT on one of the input voltage channels, it will reflect this in the corresponding bit (SAG for single-phase, or SAG_A, SAG_B, SAG_C for poly-phase) of the CE STATUS Word. See the CE Interface section in the 651X Data Sheet for details.

It is up to the MPU firmware to decide what is to be done in case a sag is detected. The Demo Code does not have any provisions for actions due to sags detected by the CE.

5.11.2.Temperature Measurement

The temperature output of the on-chip temperature sensor (*TEMP_RAW*) is stored by the CE in CE DRAM location 0x54. The relative chip temperature (*TEMP_X*) is derived by subtracting the raw temperature from the nominal temperature (*TEMP_NOM*) stored in CE DRAM location 0x11 and multiplying it with the constant factor *DEGSCALE* (decimal 22721 for the 6513 and 9585 for the 6511). Thus, once the raw temperature obtained at a known environmental temperature is stored in *TEMP_NOM*, TEMP_X will always reflect the deviation from nominal temperature. The scaling is in tenths of Centigrades, i.e. a reading of 75 means that the measured temperature is 7.5°C higher than the reference temperature.

5.11.3.Temperature Compensation for Measurements

Internal Compensation: The internal voltage reference of the 651X ICs is calibrated during device manufacture. Trim data is stored in on-chip fuses.

For the 71M651X, the temperature coefficients TC1 and TC2 are given as constants that represent typical component behavior.

For the 71M651XH, the temperature characteristics of the chip are measured during production and then stored in the fuse registers *TRIMBGA, TRIMBGB* and *TRIMM[2:0].* TC1 and TC2 can be derived from the fuses by using the relations given in the Electrical Specifications section. TC1 and TC2 can be further processed to generate the coefficients *PPMC* and *PPMC2*.

TRIMM[2:0], TRIMBGA and *TRIMBGB* are read by first writing either 4, 5 or 6 to *TRIMSEL* (0x20FD) and then reading the value of *TRIM* (0x20FF).

When the *EXT_TEMP* register in CE DRAM (address 0x38) is set to 0, the CE automatically compensates for temperature errors by controlling the *GAIN_ADJ* register (CE DRAM address 0x2E) based on the *PPMC*, *PPMC2*, and *TEMP_X* register values. In the case of internal compensation, *GAIN_ADJ* is an output of the CE.

External Compensation: Rather than internally compensating for the temperature variation, the bandgap temperature is provided to the embedded MPU, which then may digitally compensate the power outputs. This permits a system-wide temperature correction over the entire system rather than local to the chip. The incorporated thermal coefficients may include the current sensors, the voltage sensors, and other influences. Since the band gap is chopper stabilized via the *CHOP_EN* bits, the most significant long-term drift mechanism in the voltage reference is removed.

When the *EXT_TEMP* register in CE DRAM is set to 15, the CE ignores the *PPMC*, *PPMC2*, and *TEMP_X* register values and applies the gain supplied by the MPU in *GAIN_ADJ*. External compensation enables the MPU to control the CE gain based on any variable, and when *EXT_TEMP* = 15, GAIN_ADJ is an input to the CE.

5.11.4.Temperature Compensation for the RTC

The flexibility provided by the MPU allows for compensation of the RTC using the substrate temperature. To achieve this, the crystal has to be characterized over temperature and the three coefficients *Y_CAL*, *Y_CALC*, and *Y_CAL_C2* have to be calculated. Provided the IC substrate temperatures tracks the crystal temperature the coefficients can be used in the MPU firmware to trigger occasional corrections of the RTC seconds count, using the *RTC_DEC_SEC* or *RTC_INC_SEC* registers in I/O RAM.

Example: Let us assume a crystal characterized by the measurements shown in Table 5-12:

Deviation from Nominal Temperature [°C]	Measured Frequency [Hz]	Deviation from Nominal Frequency [PPM]
+50	32767.98	-0.61
+25	32768.28	8.545
0	32768.38	11.597
-25	32768.08	2.441
-50	32767.58	-12.817

Table 5-12: Frequency over Temperature

The values show that even at nominal temperature (the temperature at which the chip was calibrated for energy), the deviation from the ideal crystal frequency is 11.6 PPM, resulting in about one second inaccuracy per day, i.e. more than some standards allow. As Figure 5-24 shows, even a constant compensation would not bring much improvement, since the temperature characteristics of the crystal are a mix of constant, linear, and quadratic effects.

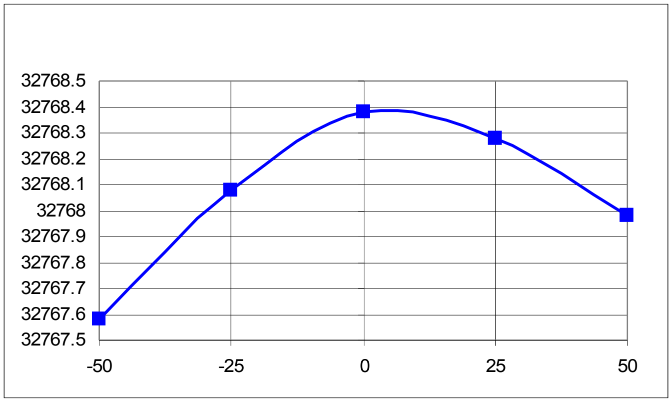


Figure 5-24: Crystal Frequency over Temperature

One method to correct the temperature characteristics of the crystal is to obtain coefficients from the curve in Figure 31 by curve-fitting the PPM deviations A fairly close curve fit is achieved with the coefficients a = 10.89, b = 0.122, and c = –0.00714 (see Figure 32).

$$f = f_{nom} * (1 + a/10^6 + T * b/10^6 + T2 * c/10^6)$$

When applying the inverted coefficients, a curve (see Figure 5-25) will result that effectively neutralizes the original crystal characteristics.

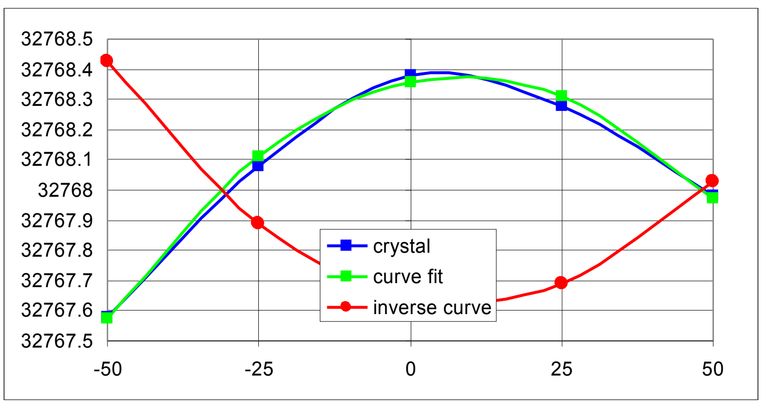


Figure 5-25: Crystal Compensation

The MPU Demo Code supplied with the TERIDIAN Demo Kits has a direct interface for these coefficients and it directly controls the *RTC_DEC_SEC* or *RTC_INC_SEC* registers. This interface is implemented by the MPU variables *Y_CAL*, *Y_CALC*, and *Y_CALC2* (MPU addresses 0x04, 0x05, 0x06). For the Demo Code, the coefficients have to be entered in the form:

$$CORRECTION(ppm) = \frac{Y\_CAL}{10} + T \cdot \frac{Y\_CALC}{100} + T^2 \cdot \frac{Y\_CALC2}{1000}$$

Note that the coefficients are scaled by 10, 100, and 1000 to provide more resolution. For our example case, the coefficients would then become (after rounding):

Y_CAL = 109, *Y_CALC* = 12, *Y_CALC2* = 7

Alternatively, the mains frequency may be used to stabilize or check the function of the RTC. For this purpose, the CE provides a count of the zero crossings detected for the selected line voltage in the *MAIN_EDGE_X* address. This count is equivalent to twice the line frequency, and can be used to synchronize and/or correct the RTC.

5.12.VALIDATING THE BATTERY

For applications that utilize the RTC it is very important to validate the battery. A brief loss of battery power when the 651X IC is powered down may result in corrupted RTC data.

After battery power is lost, the RTC will read the year 2001, the month January, and the day 1 (2001/01/01). The time information will be 01:01:01. If the MPU firmware program detects the date 01/01/2001 upon power-up or reset, it is safe to conclude that the RTC is corrupted, most likely due to a missing or low-voltage battery.

5.13.ALPHABETICAL FUNCTION REFERENCE

Function/Routine Name	Description	Input	Output	File Name
abs_x()	x = abs(x0); Take absolute value of n-byte 'x0'.	U08x *x, U08x *x0, n	none	classics.c
add()	*x += y; where 'x' & 'y' are 'n' bytes wide.	U08x *x, U08x *y, n	U08	classics.c
add_1()	*x += y; where 'x' is 'n' bytes wide, 'y' is single byte.	U08x *x, y, n	U08	classics.c
add8_4()	(U64) x += (U32) y;	U08x *x, U08x *y	none	classics.c
add8_8 ()	(U64) x += (U64) y;	U08x *x, U08x *y	none	classics.c
add8_8()	adds two unsigned 8-byte numbers	U08x *x, U08x *y	none	ce.c
API_Init()	Initializes most I/O functions that control the hardware	none	none	misc.c
Assign_Resource()	assigns the selected DIO to the selected internal resource	enum USER_PIN pin, enum IRESOURCE resource	none	misc.c
background()	executes background processing	none	none	demo.c
cal_begin()	starts auto-calibration process	none	Bbool	cal.c
cal_restore()	Restores calibration from EEPROM	none	Bbool	cal.c
cal_save()	saves calibration data to EEPROM	none	none	cal.c
calibrate()	processes measurements during auto-calibration	none	none	cal.c
ce_active()	returns CE status	none	Bbool	io651x.c
ce_config()	configures the compute engine with power equation, PRE_SAMPS and SUM_CYCLES	enum PWR_EQUATION equation, enum PRE_SAMP samples, U08 sum_cycles	none	io651x.c
ce_enable()	Enables or disables the CE	Bbool enable	none	io651x.c
ce_init()	copies CE code and data from flash to CE DRAM and CE PRAM	none	Bbool	ce651x.c
ce_outputs_rdy()	gets a snapshot of the CE status	none	Bbool	ce.c
ce_reset()	resets the CE	none	none	io651x.c
ce_totals_rdy()	gets a snapshot of the CE totals	none	Bbool	ce.c
ce_update()	performs the calculations for data just imported from the CE to the MPU	none	Bbool	ce.c
cli ()	command Line Interpreter	none	none	cli.c
cmax()	returns maximum of unsigned char 'a' and 'b'.	U08 a, U08 b	U08	library.c
cmd_ce ()	processes C commands	none	none	cmd_ce.c
cmd_ce_data_access()	Processes context for CE DATA	none	none	access.c
cmd_download()	downloads/uploads code/data between various sources and serial port	none	none	load.c
cmd_eeprom()	processes EEPROM commands	none	none	cmd_misc.c
cmd_error()	assigns generic command mode error result code	none	none	cli.c
cmd_lcd()	processes "D" commands	none	none	display.c

cmd_load()	implements user dialog for data/code download/upload	none	none	load.c
cmd_meter()	processes "M" commands	none	none	display.c
cmd_mpu_data_access()	processes context for MPU DATA	none	none	access.c
cmd_power_save()	processes power save command	none	none	cmd_misc.c
cmd_rtc()	processes RTC commands	none	none	cmd_misc.c
cmd_trim()	processes trim commands	none	none	cmd_misc.c
cmin()	returns minimum of unsigned char 'a' and 'b'.	U08 a, U08 b	U08	library.c
compare_falling_isr()	discovers falling edges on port bits	none	none	io651x.c
compare_int_enable()	enables interrupts from the analog comparators	enum COMPARE comp_int	none	io651x.c
compare_rising_isr()	discovers rising edges on port bits	none	none	io651x.c
compare_status()	gets the status of the comparator inputs	none	U08	io651x.c
complement_x()	x = x0 ^ 1s; takes ones-complement of n-byte 'x0'.	U08x *x, U08x *x0, n	none	classics.c
count_pulses()	adds the incremental pulse counts taken by the DIO ISRs to the absolute pulse counters (revision 3.05 only)	none	none	ce.c
CRC_Calc()	calculates standard 16-bit CRC polynomial per ISO/IEC 3309 on flash memory ($x^{16}+x^{12}+x^5+1$)	U08r *ptr, U16 len, U01 set	Bbool	flash.c
CRC_Calc_NVR()	calculates the 16-bit CRC polynomial per ISO/IEC 3309 on NVRAM	U08x *ptr, U16 len, U01 set	Bbool	library.c
ctoh()	converts ascii hex character to hexadecimal digit	U08 c	U08	load.c
divide()	*u /= *v;	U08x *u, U08x *v, m, n, U08x *v0	U08	classics.c
divide_ ()	*u /= *v;	U08x *u, U08x *v, m, n, U08x *v0	none	classics.c
divide_1()	*x /= y;	U08x *x, y, n	none	classics.c
done()	exits control	U08d *c	*c	cli.c
EEProm_Config()	connects/disconnects DIO4/5 for I2C interface to serial EEPROM	Bbool access, U16 page_size, U08 tWr	none	eeprom.c
EEProm_Config()	connects/disconnects DIO4/5 for I2C interface to serial EEPROM	Bbool access, U16 pg_size, U08 tWr	none	eepromp.c
es0_isr ()	serial port 0 service routine	none	none	serial.c
es1_isr()	serial port 1 service routine	none	none	serial.c
Events_Clear()	clears the specified event	U16 events	none	events.c
Events_Init()	initializes events	none	none	events.c
Ext_Int_Config()	configures the 8 external interrupts. Selects trigger type and polarity.	U08 Select, U08 Enable, U08 EdgeRising	none	misc.c
Ext_Int_Priority()	sets priority of all 7 external interrupts to one of four values	U16 Select, U16 Priority	none	misc.c
Ext_Int_Read()	returns the value and enable status of the 7 external interrupts	U08x *pExtIntValue, U08x *pExtInt	none	misc.c

free ()	returns number of available buffer bytes	U08x *head, U08x *tail, U16 size	U16	serial.c
get_char()	gets next character from CLI buffer	none	U08	io.c
get_char_d()	gets next character from CLI buffer	U08 idata *d	U08	io.c
get_digit()	gets next decimal (or hex) digit from CLI buffer	U08 idata *d	U08	io.c
Get_Event_Vector()	returns vector for specified event	enum EVENT_ID event	EventVectors [event]	events.c
get_long()	converts ascii decimal (or hex) long to binary number	none	S32	io.c
get_long_decimal()	converts ascii decimal long to binary number.	U08 c	S32	io.c
get_long_hex()	converts ASCII hexadecimal number to binary number	none	U32	io.c
get_num()	converts ascii decimal (or hex) number to binary number	none	S08	io.c
get_num_decimal()	converts ascii decimal number to binary number	none	S08	io.c
get_num_hex()	converts ascii hexdecimal byte to binary number	none	U08	io.c
get_short()	converts ascii decimal (or hex) short to binary number	none	S16	io.c
get_short_decimal()	converts ascii decimal short to binary number	none	S16	io.c
get_short_hex()	converts ascii hexdecimal short to binary number	none	U16	io.c
htoc()	converts hexadecimal digit to ascii hex character	U08 c	U08	load.c
IICGetBit()	gets a bit, used to reset some parts	none	bit	iicdio.c
IICGetBit()	gets a bit, used to reset some parts	none	bit	iiceep.c
IICInit()	initializes DIO4/5 as EEPROM interface	none	none	iicdio.c
IICInit()	initializes DIO4/5 as EEPROM interface	none	none	iiceep.c
IICRead()	receive a counted string of bytes from an IIC bus	unsigned char xdata *pchIn, unsigned short cnt	none	iicdio.c
IICStart()	IIC bus's start condition	none	none	iicdio.c
IICStart()	IIC bus's start condition	none	none	iiceep.c
IICStop()	IIC bus's stop condition	none	none	iicdio.c
IICStop()	IIC bus's stop condition	none	none	iiceep.c
IICWrite()	transmits a counted string of bytes to an IIC bus	unsigned char xdata *pchOut, unsigned short cnt	bit	iicdio.c
init_meter()	Initializes meter to default values	none	none	defaults.c
labs()	returns the absolute value	S32 x	S332	library.c
latan2()	returns the arcTangent	S32 sy, S32 sx	U32	library.c

LCD_CE_Off()	displays "CE OFF" on LCD	none	none	demo.c
LCD_Command()	turns LCD on or off, clears display	U08 LcdCmd	none	lcd.c
LCD_Config()	configures LCD parameters	U01 boost, U08 num, enum LCD_BIAS bias, enum LCD_CLK clock, U08 voltage	none	lcd.c
LCD_Data_Read()	reads from selected icon of LCD	U08 Icon	U16	lcd.c
LCD_Data_Write()	writes to selected icon of LCD	U08 icon, U16 Mask	none	lcd.c
LCD_Hello()	displays "HELLO" on LCD	none	none	demo.c
LCD_Init()	clears LCD, enables LCD segment drivers	none	none	lcd.c
lmax()	returns maximum of unsigned long 'a' and 'b'.	U32 a, U32 b	U32	library.c
lmin()	returns minimum of unsigned long 'a' and 'b'.	U32 a, U32 b	U32	library.c
log2()	returns binary logarithm	U16 k	U08	library.c
macn()	*w -= *u * *v;	U08x *w, U08x *u, v, n, U08x *u0	U08	classics.c
macp()	*w += *u * v;	U08x *w, U08x *u, v, n, U08x *u0	U08	classics.c
macp4()	(U32) w += (U32) u * (U08) y;	U08x *w, U08x *u, v, U08x *u0	U08	classics.c
macp8()	(U64) w += (U64) u * (U08) y;	U08x *w, U08x *u, v, U08x *u0	U08	classics.c
max()	returns maximum of unsigned int 'a' and 'b'.	U16 a, U16 b	U16	library.c
memcmp_rx()	compares xdata to flash code	U08r *rsrc, U08x *xsrc, U16 len	S08	library.c
memcmp_xx()	compares xdata to xdata	U08x *xsrc1, U08x *xsrc2, U16 len	S08	library.c
memcpy_ix()	copies xdata to idata	U08i *dst, U08x *src, U08 len	none	library.c
memcpy_px()	Copies data to serial EEPROM	U32 Dst, U08x *pSrc, U16 len	enum	eeprom.c
memcpy_rce()	reads from or writes to flash	S32r *dst, S32x *src, U08 len	none	flash.c
memcpy_rx()	Copies xdata to code (flash)	U08r *dst, U08x *src, U16 len	Bbool	flash.c
memcpy_xi()	Copies idata to xdata	U08x *dst, U08i *src, U08 len	none	library.c
memcpy_xp()	copies data from serial EEPROM	U08x *pDst, U32 Src, U16 len	enum	eeprom.c
memcpy_xr()	copies xdata from code (flash)	U08x *dst, U08r *src, U16 len	none	library.c
memcpy_xx()	copies xdata to xdata	U08x *dst, U08x *src, U16 len	none	library.c
memset_x()	sets xdata to specified value	U08x *dst, U08 s, U16 len	none	library.c
min()	returns minimum of unsigned int 'a' and 'b'.	U16 a, U16 b	U16	library.c

`MPU_Clk_Select()`	selects MPU clock speed	enum MPU_SPD speed	Bbool	serial.c
`multiply()`	*w = *x * *y;	U08x *w, U08x *x, U08x *y, m, n, U08x *x0	none	classics.c
`multiply_1()`	W = x * y;	U08x *w, U08x *x, y, n	U08	classics.c
`multiply_4_1()`	(U32) w = (U32) x * (U08) y;	U08x *w, U08x *x, y	U08	classics.c
`multiply_4_4()`	(U64) w = (U32) x * (U32) y;	U08x *w, U08x *x, U08x *y	none	classics.c
`multiply_8_1()`	(U64) w = (U64) x * (U08) y;	U08x *w, U08x *x, y	U08	classics.c
`multiply_8_4()`	(U96) w = (U64) x * (U32) y;	U08x *w, U08x *x, U08x *y	none	classics.c
`mux_alt()`	forces an alternate MUX sequence	none	none	io651x.c
`mux_config()`	configures the multiplexer and ADC	enum MUX_STATE states, enum ADC_SIZE size, enum CHOP enable, Bbool alt	none	io651x.c
`normalize ()`	Normalizes dividend and divisor.	U08x *u, U08x *v, m, n, U08x *v0	U08	classics.c
`Pin_Config()`	configures the data direction for port pins	enum USER_PIN pin, enum DIRECTION dir	none	misc.c
`Pin_Read()`	reads individual port pin	enum USER_PIN pin	Bbool	misc.c
`Pin_to_opto()`	enables/disables optical TX pin	Bbool	none	misc.c
`Pin_Write()`	writes to individual port pin	enum USER_PIN pin, U01 value	none	misc.c
`Port_Config()`	sets direction (input or output) of pins on selected port	enum USER_PORT port, U08 Dir	none	misc.c
`Port_Read()`	reads data from selected port	enum USER_PORT port, U08x *UserIO	none	misc.c
`Port_Write()`	writes data to selected port	enum USER_PORT port, U08 UserIO, U08 Select	none	misc.c
`PowerOFF()`	MPU Power Management (idle or halt)	enum MPU_POWER off	none	misc.c
`put_char()`	puts character into CLI buffer	U08 idata *c	none	io.c
`Read_Trim()`	reads the trim value for selected trim word	enum eTRIM select	S08	io651x.c
`RTClk_Read()`	reads current values of RTC	none	none	rtc.c
`RTClk_Reset()`	resets the RTC	none	none	rtc.c
`RTClk_Write()`	writes/sets to RTC	none	none	rtc.c
`rtm_enable()`	enables/disables real-time data monitoring	Bbool enable	none	io651x.c
`send_a_result()`	sends passed result code to UART	U08 c	none	cli.c
`send_byte()`	sends a [0, 255] byte to DTE.	S08 n	none	io.c
`send_char()`	sends single character	U08 c	none	io.c
`send_crlf()`	sends <CR><LF> out to UART.	none	none	io.c
`send_digit()`	sends single ASCII hex or decimal digit out to SERIAL0	U08 c	none	io.c
`send_help()`	sends text in code at specified location to serial port	U08r * code *s	none	cli.c

`send_hex()`	sends byte out SERIAL0 in HEX	U08 n	none	io.c
`send_long()`	sends a [0, 9,999,999,999] value to DTE.	S32 n	none	io.c
`send_long_hex()`	sends a [0, FFFFFFFF] value to DTE	U32 n	none	io.c
`send_num()`	sends a [0, 9,999,999,999] value to DTE	S32 n, U08 size	none	io.c
`send_result()`	looks up result code, primes pump for result codes	none	none	cli.c
`send_rtc()`	displays RTC data	none	none	cmd_misc.c
`send_short()`	sends a [0, 99,999] value to DTE.	S16 n	none	io.c
`send_short_hex()`	sends a [0, FFFF] value to DTE	U16 n	none	io.c
`Serial_CRx()`	gets additional bytes from the receive buffer	enum SERIAL_PORT port, U08x *buffer, U16 len	U16	serial.c
`Serial_CTx ()`	puts additional bytes into the transmit buffer	enum SERIAL_PORT port, U08x *buffer, U16 len	U16	serial.c
`Serial_Init()`	configures selected serial port	enum SERIAL_PORT port, enum SERIAL_SPD speed, Bbool parity_enb, Bbool parity, Bbool two_stop_bits, Bbool xon_xoff, Bbool seven_bits	Bbool	serial.c
`Serial_Rx ()`	sets up receive buffer and starts receiving	enum SERIAL_PORT port, U08x *buffer, U16 len	enum SERIAL_RC data	serial.c
`Serial_RxLen()`	returns the number of bytes received	enum SERIAL_PORT port	U16	serial.c
`Serial_Tx()`	sets up transmission buffer and starts transmission	enum SERIAL_PORT port, U08x *buffer, U16 len	enum SERIAL_RC data	serial.c
`Serial_TxLen()`	returns the number of bytes left to transmit	enum SERIAL_PORT port	U16	serial.c
`Set_Event_Vector()`	sets vector for specified event	enum EVENT_ID event, (code *pEventVector)	none	events.c
`SFR_Read()`	reads from SFR	U08 s, S08d *pc	enum SFR_RC	sfrs.c
`SFR_Write()`	writes to SFR	U08 s, U08 c_set, U08 c_clr	enum SFR_RC	sfrs.c
`shift_right()`	shifts (in-place) n-byte 'x' right 's' bits.	U08x *x, s, n	none	classics.c
`shift_right8_1()`	right-shifts 8-byte number by 1-bit	U08x *x	none	library.c
`shift_right8_2()`	right-shifts 8-byte number by 2-bits	U08x *x	none	library.c
`Soft_Reset()`	initiates soft reset	none	none	misc.c
`sqrt4_2()`	returns the 16-bit square root of the 32-bit argument	U08x *z, U08x *x	none	library.c
`sqrt4_4 ()`	returns the 32-bit square root of the 32-bit argument. The 16 LSBs are the fractional part of the square root.	U08x *z, S08x *x	none	library.c
`sqrt8_4()`	returns the square root of the argument	U08x *z, U08x *x	none	library.c

`square()`	*w = *x ** 2; Square n-byte 'x'.	U08x *w, U08x *x, n	none	classics.c
`SSI_Config()`	configures the SSI	U08 flags, U08 addr, U08 count	none	io651x.c
`start_tx_ram()`	sends RAM string out PC UART	U08x *c	none	io.c
`start_tx_rslt()`	sends ROM string out PC UART	U08r *c	none	io.c
`strlen_r ()`	returns length of string in flash code	U08r *src	U16	library.c
`strlen_x()`	returns length of string in xdata	U08x *src	U16	library.c
`sub8_4()`	(U64) x -= (U32) y;	U08x *x, U08x *y	none	classics.c
`sub8_8()`	(U64) x -= (U64) y;	U08x *x, U08x *y	none	classics.c
`subtract()`	*x -= *y; where 'x' & 'y' are 'n' bytes wide.	U08x *x, U08x *y, n	U08	classics.c
`subtract_1()`	x -= y; where 'x' is 'n' bytes wide, 'y' is single byte.	U08x *x, y, n	U08	classics.c
`temperature_enable()`	enables/disables temperature measurements	Bbool temperature	none	io651x.c
`tmux_config()`	configures the source for the test multiplexer	enum TMUX_INPUT tmux	none	io651x.c
`used ()`	returns number of buffer bytes in use	U08x *head, U08x *tail, U16 size	U16	serial.c
`VARPULSE_Enable()`	configures VARPULSE to be output on DIO4	Bbool connect	none	eeprom.c
`version()`	returns the chip version number	none	U08	io651x.c
`WPULSE_Enable()`	configures WPULSE to be output on DIO4	Bbool connect	none	eeprom.c

5.14.ERRATA FOR DEMO CODE REVISION 3.04

The up-to-date list of known issues with revision 3.04 of the Demo Code can be found in the readme.txt file contained in the 6511_demo or 6513_demo ZIP files shipped with the Demo Kits.

All issues were fixed in Demo Code revisison 3.05. The factory should be contacted for updates to the Demo Code.

Known Firmware Errata for version 3.04 are listed in the table below.

Number	Product	Issue	Comment
93	6511	DEGSCALE should be 9585, not 9879, causing a 3% error in the temperature display	A fix is straight forward by entering the proper value for DEGSCALE.
94	651X	Trim registers 2 and 7 are read incorrect-ly.	
96	651X	If external (outside of the chip) pulse sources ever occur, then the pulse R source will not work	
98	651X	Software timers lose time because tic-toc was set while ce_update() is running in the background	A fix is possible by having the interrupt accumulate counts.
99	651X	A pulse counter is not implemented.	A fix is possible by having the interrupt count pulses for a fixed number of seconds as measured by the RTC interrupt
100	651X	The watchdog is being reset improperly. On rare instances, a CE or RTC interrupt might be lost.	
101	651X	When the CE is turned off by the command line, a watchdog timer reset occurs.	A fix is possible by disabling the CE interrupt in the command line, which removes the CE from the set of interrupts that are required in order to reset the watchdog.
102	651X	DEG_SCALE is stored both as an MPU variable ")0D" and CE variable "]30". The two version could have different values and then saved and restored in various ways leading to inconsistencies.	
124	651X	Priority levels are reversed in Ext_Int_Priority() in misc.c	
134	651X	Creep logic in pulse sources often uses current from i0, instead of the correct element's current	
135	651X	Creep has observed when the actual creep value was zero.	
137	651X	CE's data is invalid for the first second, and if the unit is set up for an accumulation interval not equal to one second, the start-up can get bad data	
136	651X	Vcal, Ical and Scal are not initialized when the EEPROM is empty.	
141	651X	Sag was not being reported to the status register.	

144	651X	In rare circumstances, the flag for RTC-1-sec is not cleared by the interrupt, causing the code to hang up. A restart may also fail to clear both flags, causing a hang up.	
146	651X	The divide_1() function in classics.c calculates an incorrect residue.	
147	651X	The default temperature compensation factor A, used to calculate PPMC in set_tc1_tc2 () in ce651x.c, is -6680, but it should be –668.	A fix is straight forward by entering the proper value for A.
148	651X	The demo code can have priority inversion	Fixes involve redesigning the code's critical sections

5.15.TEST MODULES

Various Test Modules are available from TERIDIAN. These Test Modules are small Keil projects that can be used to test various functions of the 71M6521 IC. The available Test Modules are described in this section.

5.15.1.6513 CE Example

This Test Module builds a simple test code that starts and runs the compute engine, collects meter data in RAM, and generates pulses for one accumulation interval.

The Keil project file is `6513_ce_example.uv2`.

5.15.2.Serial Port Tests

These Test Modules build simple tests of the serial ports. The tests start by sending the ASCII character "E" in a loop, e.g. for testing with an oscilloscope. As soon as a character is received, the test code begins echoing typed characters, using polling IO. Sending the period character (".") switches the I/O to interrupting I/O.

Note that ser0test.c and ser1test.c use identical text, except for the include file. This is a very convenient technique for moving serial I/O to a different port when requirements change.

The Keil project files are `ser0test.uv2` and `ser1test.uv2`.

5.15.3.Timer Tests

These Test Modules build simple routines for testing of the interrupting timers, run both once, and periodically. The routines include an extended 30-second test that can be used with a stop-watch timer to measure accuracy.

Note that tmr0test.c and tmr1test.c use identical text except for the include file. This is a very convenient technique for moving a timer IO to a different port when requirements change.

The Keil project files are `tmr0test.uv2` and `tmr1test.uv2`.

5.15.4.EEPROM Tests

This routine demonstrates the use and test of the eeprom interface.

The Keil project files is `eepromtest.uv2`.

5.15.5.Generating DIO Pulses on Reset

This Test Module is written in 8051 assembler and is executed after processor reset. It pulses DIO7 on a meter chip. This function is useful as a scope loop to discover if the chip resets when expected.

The Keil project file is `RESET_PULSES_DIO7.UV2`.

5.15.6.Testing the Security Bit

This Test Module is written in 8051 assembler and is executed after processor reset. It sets the security bit and then displays the security bit on DIO_7. It is useful to test the behavior of the security bit under various system conditions.

The Keil project file is `RESET_READ_SE.UV2`.

5.15.7.Software Timer Test

This project, consisting of several files, demonstrates the use and test of the software timer using a hardware timer that is multiplexed into many slower timers.

The Keil project file is `stmtest.uv2`.

5.15.8.Interrupt Test

This Test Module is written in 8051 assembler and can be used for testing the function of the INT0, INT1, TMR0, and TMR1 control using DIO_Rx. When the code is run it configures all possible DIO pins as DIO input pins. When testing, a breakpoint should be set on the vector for one of INT0, INT1, TMR0, TMR1. Also, one of the I/O RAM registers DIO_R0...DIO_R11 should be set to the resource code for that vector. When the DIO pin under test is probed with GND or V3P3, the programmed interrupt is generated.

The code sets up DIO 4, 5, 6 and 7 for one each of four interrupts.

The Keil project file is `inttest.uv2.`

5.15.9.RTC Test

This test module, consisting of `rtctest.uv2` and `rtctest.c` implement a digital clock/calendar on a Demo Board, using the real-time-clock. To set the clock, just press the button on the Debug Board. A decimal point will then appear next to the leftmost digit. Release the button, and the decimal point will move to the right until it gets to the digit that needs to be set. Hold the button to cause the selected digit to be incremented until the digit is right. To set the seconds, set the minute one minute ahead, then hold the button on the seconds until the next minute starts.

For 6511 and 6513 Demo Boards, the button is SW2 on the Debug Board. For 6521 Demo Boards the WAKE pushbutton is used.

5.15.10.LCD Test

This test module, consisting of `lcdtest.uv2` and `lcdtest.c` builds a simple test of the LCD driver. The test module sends the texts or numbers "HELLO", "CE OFF", "-8888888", "-   00.001", "-   10.000", "00000000", "11111111"... "99999999" in a loop.

6

6. 80515 REFERENCE

An 80515 core is implemented on the TERIDIAN 71M651X chips. This section is intended for software engineers who plan to use the 80515.

6.1.80515 OVERVIEW

The 80515 is a fast single-chip 8-bit micro controller core. It is a fully functional 8-bit embedded controller that executes all ASM51 instructions and has the same instruction set as the 80C31. The 80515 provides software and hardware interrupts, an interface for serial communications, a timer system and a watchdog timer.

6.1.1. 80515 Performance

The architecture eliminates redundant bus states and implements parallel execution of fetch and execution phases. Normally a cycle is aligned with a memory fetch, therefore, most of the 1-byte instructions are performed in a single cycle. The 80515 uses 1 clock per cycle leading to an 8x performance improvement (in terms of MIPS) over the Intel 8051 device running at the same clock frequency.

The original 8051 had a 12-clock architecture. A machine cycle needed 12 clocks and most instructions were either one or two machine cycles. Thus, except for the MUL and DIV instructions, the 8051 used either 12 or 24 clocks for each instruction. Furthermore, each cycle in the 8051 used two memory fetches. In many cases the second fetch was a dummy, and extra clocks were wasted.

Table 6-1 shows the speed advantage of the 80515 over the standard 8051. A speed advantage of 12 means that the 80515 performs the same instruction twelve times faster that the 8051.

Speed advantage	Number of instructions	Number of opcodes
24	1	1
12	27	83
9.6	2	2
8	16	38
6	44	89
4.8	1	2
4	18	31
3	2	9
Average: 8.0	Sum: 111	Sum: 255

Table 6-1: Speed Advantage Summary

The average speed advantage is 8x, however, the actual speed improvement observed in a system will depend on the instruction mix.

6.1.2. 80515 Features

Below is a summary of the 80515 features:

- Control Unit
  - 8-bit instruction decoder
  - Reduced instruction cycle time up to 12 times
- Arithmetic-Logic Unit
  - 8 bit arithmetic and logical operations
  - Boolean manipulations
  - 8 x 8 bit multiplication and 8 / 8 bit division
- 32-bit Input/Output ports
  - Four 8-bit I/O ports
  - Alternate port functions such as external interrupts and serial interfaces are separated, providing extra port pins when compared with the standard 8051
- Two 16-bit Timer/Counters
- Two Serial Peripheral Interfaces in full duplex mode, capable of parity generation
  - Synchronous mode, fixed baud rate, Serial 0 only
  - 8-bit UART mode, variable baud rate
  - 9-bit UART mode, fixed baud rate, Serial 0 only
  - 9-bit UART mode, variable baud rate
  - 7E1, 7O1, 7N2, 7E2, 7O2, 8N1, 8E1, 8O1, 8N2, 9N1 data formats supported
  - Two Internal baud rate generators independent from timers
- Interrupt Controller
  - Four priority levels with 11 interrupt sources
- 15-bit Programmable Watchdog Timer
- Internal Data Memory interface
  - Can address up to 256B of internal data memory space
- External Memory Interface (External = external to the 80515 core, but on-chip)
  - Can address up to 64kB of external program memory
  - Can address up to 2kB of external data memory plus 1kB CE DRAM and 4kB CE PRAM
  - Separate address/data bus to allow easy connection to memories
  - Variable length code fetch and MOVC to access fast/slow program memory
  - Variable length MOVX to access fast/slow RAM or peripherals
  - Dual data pointer for fast data block transfer
- Special Function Registers interface
  - Services up to 74 special function registers (SFRs)

6.2.80515 ARCHITECTURAL OVERVIEW

6.2.1. Memory organization

The 80515 Micro controller core incorporates the Harvard architecture with separate code and data spaces.

Memory organization in the 80515 is similar to that of the industry standard 8051. There are three memory areas: program memory (External Flash or ROM), external data memory (External RAM), and internal data memory (Internal RAM).

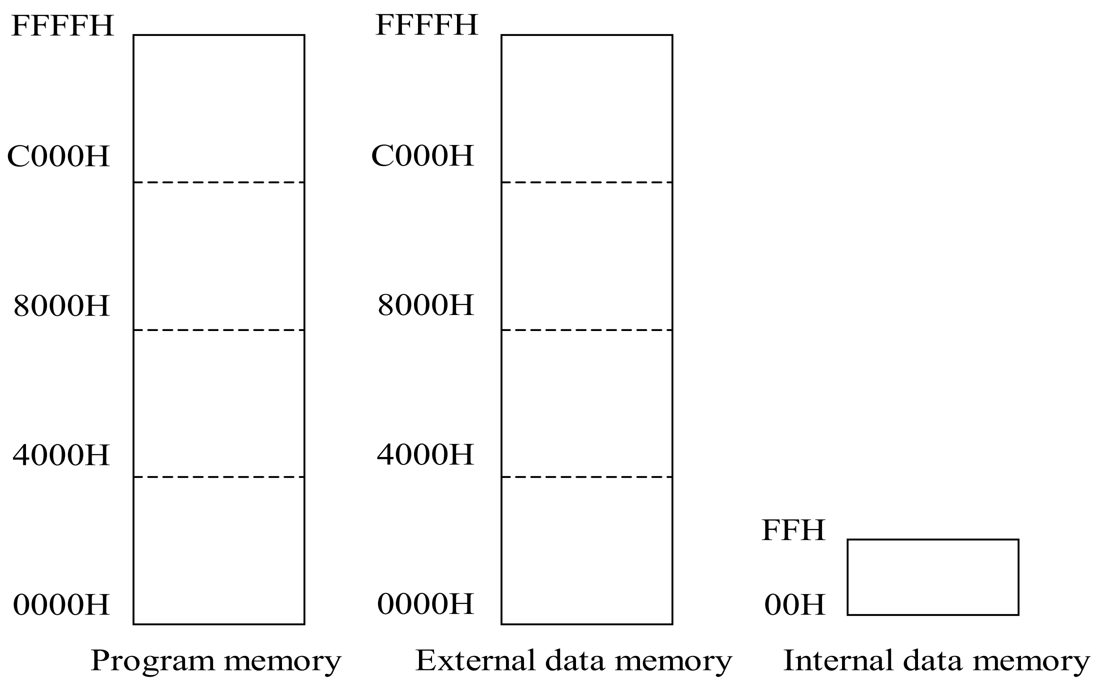


Figure 6-1: Memory Map

Program Memory

The 80515 can address up to 64kB of program memory space from 0000H to FFFFH. Program memory is read when the MPU fetches instructions or performs a MOVC.

After reset, the MPU starts program execution from location 0000H. The lower part of the program memory includes a reset and interrupt vectors. The interrupt vectors are spaced at 8-byte intervals, starting from 0003H.

External Data Memory

The 80515 can address up to 64kB of external data memory in the space from 0000H to FFFFH. The 80515 writes into external data memory when the MPU executes a MOVX @Ri,A or MOVX @DPTR,A instruction. The external data memory is read when the MPU executes a MOVX A,@Ri or MOVX A,@DPTR instruction.

There is an improved variable length access for the MOVX instructions to access fast or slow external RAM and external peripherals. The three low ordered bits of the CKCON register define the stretch memory cycles. Setting all the CKCON stretch bits to one allows access to very slow external RAM or external peripherals.

Table 6-2 shows how the signals of the External Memory Interface change when stretch values are set from 0 to 7. The widths of the signals are counted in MPU clock cycles. The post-reset state of the CKCON register, which is in bold in the table, performs the MOVX instructions with a stretch value equal to 1.

CKCON register			Stretch Value	Read signals width		Write signal width	
CKCON.2	CKCON.1	CKCON.0		memaddr	memrd	memaddr	memwr
0	0	0	0	1	1	2	1
0	0	1	1	2	2	3	1
0	1	0	2	3	3	4	2
0	1	1	3	4	4	5	3
1	0	0	4	5	5	6	4
1	0	1	5	6	6	7	5
1	1	0	6	7	7	8	6
1	1	1	7	8	8	9	7

Table 6-2: Stretch Memory Cycle Width

There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type (MOVX@Ri), the contents of R0 or R1, in the current register bank, provide the eight lower-ordered bits of address. The eight high-ordered bits of address are specified with the USR2 SFR. This method allows the user paged access (256 pages of 256 bytes each) to the full 64KB of external data RAM. In the second type of MOVX instruction (MOVX@DPTR), the data pointer generates a sixteen-bit address. This form is faster and more efficient when accessing very large data arrays (up to 64 Kbytes), since no additional instructions are needed to set up the eight high ordered bits of address.

It is possible to mix the two MOVX types. This provides the user with four separate data pointers, two with direct access and two with paged access to the entire 64KB of external memory range.

Dual Data Pointer

The Dual Data Pointer accelerates the block moves of data. The standard DPTR is a 16-bit register that is used to address external memory or peripherals. In the 80515 core the standard data pointer is called DPTR, the second data pointer is called DPTR1. The data pointer select bit chooses the active pointer. The data pointer select bit is located at the LSB of the DPS register (DPS.0). DPTR is selected when DPS.0 = 0 and DPTR1 is selected when DPS.0 = 1.

The user switches between pointers by toggling the LSB of the DPS register. All DPTR-related instructions use the currently selected DPTR for any activity.

The second data pointer may or may not be supported by certain compilers.

Internal Data Memory

The Internal data memory interface services up to 256 bytes of off-core data memory. The internal data memory address is always 1 byte wide. The memory space is 256 bytes (00H to FFH), and can be accessed by either direct or indirect addressing. The Special Function Registers occupy the upper 128 bytes. This SFR area is available only by direct addressing. Indirect addressing accesses the upper 128 bytes of Internal RAM.

The lower 128 bytes contain working registers and bit-addressable memory. The lower 32 bytes form four banks of eight registers (R0-R7). Two bits on the program memory status word (PSW) select which bank is in use. The next 16 bytes form a block of bit-addressable memory space at bit addressees 00H-7FH. All of the bytes in the lower 128 bytes are accessible through direct or indirect addressing.

Table 6-3 shows the internal data memory map.

address	direct addressing	Indirect addressing
0xFF	**Special Function Registers (SFRs)**	RAM
0x80		
0x7F		
0x30		
0x2F	**bit-addressable area**	
0x20		
0x1F	register banks R0…R7	
0x00		

Table 6-3: Internal Data Memory Map

Special Function Registers Location

A map of the Special Function Registers is shown in Table 6-4. Only a few addresses are occupied, the others are not implemented. SFRs specific to the 651X are shown in **bold** print (see 71M651X data sheet for descriptions of these registers). Any read access to unimplemented addresses will return undefined data, while any write access will have no effect. The registers at 0x80, 0x88, 0x90, etc., are bit-addressable, all others are byte-addressable.

Hex/Bin	X000	X001	X010	X011	X100	X101	X110	X111	Bin/Hex
	Bit-addressable	Byte-addressable							
F8	**INTBITS**								FF
F0	B								F7
E8	**WDI**								EF
E0	A								E7
D8	WDCON								DF
D0	PSW								D7
C8									CF
C0	IRCON								C7
B8	IEN	IP1	S0RELH	S1RELH				USR2	BF
B0			**FLSHCTL**					**PGADR**	B7
A8	IEN0	IP0	S0RELL						AF
A0	**DIO11 (P2)**	**DIO12 (P2)**	**DIO8 (P0)**						A7
98	S0CON	S0BUF	IEN2	S1CON	S1BUF	S1RELL	**EEDATA**	**EECTRL**	9F
90	**DIO9 (P1)**	**DIO10 (P1)**	DPS		**ERASE**				97
88	TCON	TMOD	TL0	TL1	TH0	TH1	CKCON		8F
80	**DIO7 (P0)**	SP	DPL	DPH	DPL1	DPH1	WDTREL	PCON	87

Table 6-4: Special Function Registers Locations

Generic Special Function Register Overview

All generic SFRs are explained in detail in section 6.3.2.

Register	Symbol	Description
Program status word	PSW	The PSW contains program status information
Accumulator	ACC	The accumulator register. Mmemonics for instructions involving the accumulator refer to the accumulator as A.
B register	B	This register is used for multiply and divide operations. It may also be used as a scratchpad (temporary) register.
Stack pointer	SP	The stack pointer is 8 bits wide and is incremented before data is stored with PUSH and CALL operations. SP is initialized to 0x07 after reset.
Data pointer	DPL, DPH	Since the DP consists of two bytes (DPH and DPL), it can hold a 16-bit address. It may be manipulated as a 16-bit register or as two separate 8-bit registers.
Secondary data pointer	DPL1, DPH1 CAUTION	This register is a second 16-bit data pointer. Check with the documentation on the compiler used for generating MPU code whether this pointer is utilized or not.
Data pointer select register	DPS	This register selects which data pointer is to be used for the current operation.
Port registers	P0, P1, P2	These registers hold bit patters that are written to or read from the DIO ports
Serial data buffer	S0BUF, S1BUF	These registers hold data received from the serial interfaces 0 and 1. Data to be transmitted via the serial interfaces is written to S0BUF or S1BUF.
Serial port reload registers	S0RELL, S0RELH, S1RELL, S1RELH	These register pairs can be used to control the baud rate for the serial ports 0 and 1.
Timer registers	TL0, TL1, TH0, TH1	These register pairs (TH0/TL0 and TH1/TL1) are the 16-bit counting registers for timers 0, 1, and 2.
Interrupt control registers	IP0, IP1, IEN, IEN0, TMOD, TCON, T2CON, SCON, PCON, IRCON	These registers contain control and status bits pertaining to the interrupt system, the timers/counters, and the serial port.
Clock control register	CKCON	The clock control/configuration register. It is used to implement stretch memory cycles for memory access.
Watchdog timer reload register	WDTREL	This register holds the reload count for the software watchdog timer
Baud rate generator selector	WDCON	This register determines whether UART0 is controlled by timer 1 or by the internal baud rate generator.

Generic Special Function Registers Location and Reset Values

Table 6-5 shows the location of the SFRs and the value they assume at reset on power-up.

Register	Location	Reset value	Description
P0	80H	FFH	Port 0
SP	81H	07H	Stack Pointer
DPL	82H	00H	Data Pointer Low 0
DPH	83H	00H	Data Pointer High 0
DPL1	84H	00H	Data Pointer Low 1
DPH1	85H	00H	Data Pointer High 1
WDTREL	86H	00H	Watchdog Timer Reload register
PCON	87H	00H	UART Speed Control
TCON	88H	00H	Timer/Counter Control
TMOD	89H	00H	Timer Mode Control
TL0	8AH	00H	Timer 0, low byte
TL1	8BH	00H	Timer 1, high byte
TH0	8CH	00H	Timer 0, low byte
TH1	8DH	00H	Timer 1, high byte
CKCON	8EH	01H	Clock Control (Stretch=1)
P1	90H	FFH	Port 1
DPS	92H	00H	Data Pointer select Register
S0CON	98H	00H	Serial Port 0, Control Register
S0BUF	99H	00H	Serial Port 0, Data Buffer
IEN2	9AH	00H	Interrupt Enable Register 2
S1CON	9BH	00H	Serial Port 1, Control Register
S1BUF	9CH	00H	Serial Port 1, Data Buffer
S1RELL	9DH	00H	Serial Port 1, Reload Register, low byte
P2	A0H	00H	Port 2
IEN0	A8H	00H	Interrupt Enable Register 0
IP0	A9H	00H	Interrupt Priority Register 0
S0RELL	AAH	D9H	Serial Port 0, Reload Register, low byte
P3	B0H	FFH	Port 3
IEN1	B8H	00H	Interrupt Enable Register 1
IP1	B9H	00H	Interrupt Priority Register 1
S0RELH	BAH	03H	Serial Port 0, Reload Register, high byte
S1RELH	BBH	03H	Serial Port 1, Reload Register, high byte
USR2	BFH	00H	User 2 Port, high address byte for MOVX@Ri
IRCON	C0H	00H	Interrupt Request Control Register
PSW	D0H	00H	Program Status Word
WDCON	D8H	00H	Baud Rate Control Register (only WDCON.7 bit used)
A	E0H	00H	Accumulator
B	F0H	00H	B Register

Table 6-5: Special Function Registers Reset Values

Special Function Registers Specific to the 651X

Register	Alternative Name	SFR Address	R/W	Description
DIO0	DIO_0	0x80	R/W	Register for port 0 read and write operations (pins DIO0…DIO7)
DIO8	DIO_DIR0	0xA2	R/W	Data direction register for port 0. Setting a bit to 1 means that the corresponding pin is an output.
DIO9	DIO_1	0x90	R/W	Register for port 1 read and write operations (pins DIO8…DIO15)
DIO10	DIO_DIR1	0x91	R/W	Data direction register for port 1. Setting a bit to 1 means that the corresponding pin is an output.
DIO11	DIO_2	0xA0	R/W	Register for port 2 read and write operations (pins DIO16…DIO21)
DIO12	DIO_DIR2	0xA1	R/W	Data direction register for port 2. Setting a bit to 1 means that the corresponding pin is an output.
ERASE	FLSH_ERASE	0x94	W	This register is used to initiate either the Flash Mass Erase cycle or the Flash Page Erase cycle. Specific patterns are expected for FLSH_ERASE in order to initiate the appropriate Erase cycle (default = 0x00). 0x55 – Initiate Flash Page Erase cycle. Must be proceeded by a write to FLSH_PGADR @ SFR 0xB7. 0xAA – Initiate Flash Mass Erase cycle. Must be proceeded by a write to FLSH_MEEN @ sfr 0xB2 and the debug (CC) port must be enabled. Any other pattern written to FLSH_ERASE will have no effect.
PGADDR	FLSH_PGADR	0xB7	R/W	Flash Page Erase Address register containing the flash memory page address (page 0 thru 127) that will be erased during the Page Erase cycle. (default = 0x00). Must be re-written for each new Page Erase cycle.
EEDATA		0x9E	R/W	I2C EEPROM interface data register
EECTRL		0x9F	R/W	I2C EEPROM interface control register. If the MPU wishes to write a byte of data to EEPROM, it places the data in EEDATA and then writes the 'Transmit' code to EECTRL. The write to EECTRL initiates the transmit.
FLSHCRL		0xB2	 R/W W R/W R	This multi-purpose register contains the following bits: <u>Bit 0 (FLSH_PWE): Program Write Enable:</u> 0 – MOVX commands refer to XRAM Space, normal operation (default). 1 – MOVX @DPTR,A moves A to Program Space (Flash) @ DPTR. This bit is automatically reset after each byte written to flash. Writes to this bit are inhibited when interrupts are enabled. <u>Bit 1 (FLSH_MEEN): Mass Erase Enable:</u> 0 – Mass Erase disabled (default). 1 – Mass Erase enabled. Must be re-written for each new Mass Erase cycle. <u>Bit 6 (SECURE):</u> Enables security provisions that prevent external reading of flash memory and CE program RAM. This bit is reset on chip reset and may only be set. Attempts to write zero are ignored. <u>Bit 7 (PREBOOT):</u> Indicates that the preboot sequence is active.

WDI		0xE8	 R/W R/W W	Only byte operations on the whole WDI register should be used when writing. This multi-purpose register contains the following bits: Bit 0 (IE_XFER): XFER Interrupt Flag: This flag monitors the XFER_BUSY interrupt. It is set by hardware and must be cleared by the interrupt handler Bit 1 (IE_RTC): RTC Interrupt Flag: This flag monitors the RTC_1SEC interrupt. It is set by hardware and must be cleared by the interrupt handler Bit 7 (WD_RST): WD Timer Reset: The WDT is reset when a 1 is written to this bit.
INTBITS	INT0…INT6	0xF8	R	Interrupt inputs. The MPU may read these bits to see the input to external interrupts INT0, INT1, up to INT6. These bits do not have any memory and are primarily intended for debug use

6.2.2. The 80515 Instruction Set

All 80515 instructions are binary code compatible and perform the same functions as they do with the industry standard 8051. The following tables give a summary of the instruction set cycles of the 80515 Micro controller core.

Table 6-6 and Table 6-7 contain notes for mnemonics used in instruction set tables.

Tables 6-8 through 6-12 show the instruction hexadecimal codes, the number of bytes, and the number of machine cycles required for each instruction to execute.

Rn	Working register R0-R7
direct	256 internal RAM locations, any Special Function Registers
@Ri	Indirect internal or external RAM location addressed by register R0 or R1
#data	8-bit constant included in instruction
#data 16	16-bit constant included as bytes 2 and 3 of instruction
bit	256 software flags, any bit-addressable I/O pin, control or status bit
A	Accumulator

Table 6-6: Notes on Data Addressing Modes

addr16	Destination address for LCALL and LJMP may be anywhere within the 64-kB of program memory address space.
addr11	Destination address for ACALL and AJMP will be within the same 2-kB page of program memory as the first byte of the following instruction.
rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127/-128 bytes relative to the first byte of the following instruction

Table 6-7: Notes on Program Addressing Modes

Instructions Ordered by Function

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

Table 6-8. Arithmetic Operations

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

Table 6-9. Logic Operations

Mnemonic	Description	Code	Bytes	Cycles
MOV A,Rn	Move register to accumulator	E8-EF	1	1
MOV A,direct	Move direct byte to accumulator	E5	2	2
MOV A,@Ri	Move indirect RAM to accumulator	E6-E7	1	2
MOV A,#data	Move immediate data to accumulator	74	2	2
MOV Rn,A	Move accumulator to register	F8-FF	1	2
MOV Rn,direct	Move direct byte to register	A8-AF	2	4
MOV Rn,#data	Move immediate data to register	78-7F	2	2
MOV direct,A	Move accumulator to direct byte	F5	2	3
MOV direct,Rn	Move register to direct byte	88-8F	2	3
MOV direct1,direct2	Move direct byte to direct byte	85	3	4
MOV direct,@Ri	Move indirect RAM to direct byte	86-87	2	4
MOV direct,#data	Move immediate data to direct byte	75	3	3
MOV @Ri,A	Move accumulator to indirect RAM	F6-F7	1	3
MOV @Ri,direct	Move direct byte to indirect RAM	A6-A7	2	5
MOV @Ri,#data	Move immediate data to indirect RAM	76-77	2	3
MOV DPTR,#data16	Load data pointer with a 16-bit constant	90	3	3
MOVC A,@A+DPTR	Move code byte relative to DPTR to accumulator	93	1	3
MOVC A,@A+PC	Move code byte relative to PC to accumulator	83	1	3
MOVX A,@Ri	Move external RAM (8-bit addr.) to A	E2-E3	1	3-10
MOVX A,@DPTR	Move external RAM (16-bit addr.) to A	E0	1	3-10
MOVX @Ri,A	Move A to external RAM (8-bit addr.)	F2-F3	1	4-11
MOVX @DPTR,A	Move A to external RAM (16-bit addr.)	F0	1	4-11
PUSH direct	Push direct byte onto stack	C0	2	4
POP direct	Pop direct byte from stack	D0	2	3
XCH A,Rn	Exchange register with accumulator	C8-CF	1	2
XCH A,direct	Exchange direct byte with accumulator	C5	2	3
XCH A,@Ri	Exchange indirect RAM with accumulator	C6-C7	1	3
XCHD A,@Ri	Exchange low-order nibble indirect RAM with A	D6-D7	1	3

Table 6-10: Data Transfer Operations

Mnemonic	Description	Code	Bytes	Cycles
ACALL addr11	Absolute subroutine call	xxx11	2	6
LCALL addr16	Long subroutine call	12	3	6
RET	Return from subroutine	22	1	4
RETI	Return from interrupt	32	1	4
AJMP addr11	Absolute jump	xxx01	2	3
LJMP addr16	Long jump	02	3	4
SJMP rel	Short jump (relative addr.)	80	2	3
JMP @A+DPTR	Jump indirect relative to the DPTR	73	1	2
JZ rel	Jump if accumulator is zero	60	2	3
JNZ rel	Jump if accumulator is not zero	70	2	3
JC rel	Jump if carry flag is set	40	2	3
JNC	Jump if carry flag is not set	50	2	3
JB bit,rel	Jump if direct bit is set	20	3	4
JNB bit,rel	Jump if direct bit is not set	30	3	4
JBC bit,direct rel	Jump if direct bit is set and clear bit	10	3	4
CJNE A,direct rel	Compare direct byte to A and jump if not equal	B5	3	4
CJNE A,#data rel	Compare immediate to A and jump if not equal	B4	3	4
CJNE Rn,#data rel	Compare immed. to reg. and jump if not equal	B8-BF	3	4
CJNE @Ri,#data rel	Compare immed. to ind. and jump if not equal	B6-B7	3	4
DJNZ Rn,rel	Decrement register and jump if not zero	D8-DF	2	3
DJNZ direct,rel	Decrement direct byte and jump if not zero	D5	3	4
NOP	No operation	00	1	1

Table 6-11: Program Branches

Mnemonic	Description	Code	Bytes	Cycles
CLR C	Clear carry flag	C3	1	1
CLR bit	Clear direct bit	C2	2	3
SETB C	Set carry flag	D3	1	1
SETB bit	Set direct bit	D2	2	3
CPL C	Complement carry flag	B3	1	1
CPL bit	Complement direct bit	B2	2	3
ANL C,bit	AND direct bit to carry flag	82	2	2
ANL C,/bit	AND complement of direct bit to carry	B0	2	2
ORL C,bit	OR direct bit to carry flag	72	2	2
ORL C,/bit	OR complement of direct bit to carry	A0	2	2
MOV C,bit	Move direct bit to carry flag	A2	2	2
MOV bit,C	Move carry flag to direct bit	92	2	3

Table 6-12: Boolean Manipulations

Instructions Ordered by Opcode (Hexadecimal)

Opcode	Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic
00 H	NOP	20 H	JB bit.rel	40 H	JC rel
01 H	AJMP addr11	21 H	AJMP addr11	41 H	AJMP addr11
02 H	LJMP addr16	22 H	RET	42 H	ORL direct,A
03 H	RR A	23 H	RL A	43 H	ORL direct,#data
04 H	INC A	24 H	ADD A,#data	44 H	ORL A,#data
05 H	INC direct	25 H	ADD A,direct	45 H	ORL A,direct
06 H	INC @R0	26 H	ADD A,@R0	46 H	ORL A,@R0
07 H	INC @R1	27 H	ADD A,@R1	47 H	ORL A,@R1
08 H	INC R0	28 H	ADD A,R0	48 H	ORL A,R0
09 H	INC R1	29 H	ADD A,R1	49 H	ORL A,R1
0A H	INC R2	2A H	ADD A,R2	4A H	ORL A,R2
0B H	INC R3	2B H	ADD A,R3	4B H	ORL A,R3
0C H	INC R4	2C H	ADD A,R4	4C H	ORL A,R4
0D H	INC R5	2D H	ADD A,R5	4D H	ORL A,R5
0E H	INC R6	2E H	ADD A,R6	4E H	ORL A,R6
0F H	INC R7	2F H	ADD A,R7	4F H	ORL A,R7
10 H	JBC bit,rel	30 H	JNB bit.rel	50 H	JNC rel
11 H	ACALL addr11	31 H	ACALL addr11	51 H	ACALL addr11
12 H	LCALL addr16	32 H	RETI	52 H	ANL direct,A
13 H	RRC A	33 H	RLC A	53 H	ANL direct,#data
14 H	DEC A	34 H	ADDC A,#data	54 H	ANL A,#data
15 H	DEC direct	35 H	ADDC A,direct	55 H	ANL A,direct
16 H	DEC @R0	36 H	ADDC A,@R0	56 H	ANL A,@R0
17 H	DEC @R1	37 H	ADDC A,@R1	57 H	ANL A,@R1
18 H	DEC R0	38 H	ADDC A,R0	58 H	ANL A,R0
19 H	DEC R1	39 H	ADDC A,R1	59 H	ANL A,R1
1A H	DEC R2	3A H	ADDC A,R2	5A H	ANL A,R2
1B H	DEC R3	3B H	ADDC A,R3	5B H	ANL A,R3
1C H	DEC R4	3C H	ADDC A,R4	5C H	ANL A,R4
1D H	DEC R5	3D H	ADDC A,R5	5D H	ANL A,R5
1E H	DEC R6	3E H	ADDC A,R6	5E H	ANL A,R6
1F H	DEC R7	3F H	ADDC A,R7	5F H	ANL A,R7

Table 6-13: Instruction Set in Hexadecimal Order (1/3)

Opcode	Mnemonic	Opcode	Mnemonic	Opcode	Mnemonic
60 H	JZ rel	80 H	SJMP rel	A0 H	ORL C,bit
61 H	AJMP addr11	81 H	AJMP addr11	A1 H	AJMP addr11
62 H	XRL direct,A	82 H	ANL C,bit	A2 H	MOV C,bit
63 H	XRL direct,#data	83 H	MOVC A,@A+PC	A3 H	INC DPTR
64 H	XRL A,#data	84 H	DIV AB	A4 H	MUL AB
65 H	XRL A,direct	85 H	MOV direct,direct	A5 H	Reserved
66 H	XRL A,@R0	86 H	MOV direct,@R0	A6 H	MOV @R0,direct
67 H	XRL A,@R1	87 H	MOV direct,@R1	A7 H	MOV @R1,direct
68 H	XRL A,R0	88 H	MOV direct,R0	A8 H	MOV R0,direct
69 H	XRL A,R1	89 H	MOV direct,R1	A9 H	MOV R1,direct
6A H	XRL A,R2	8A H	MOV direct,R2	AA H	MOV R2,direct
6B H	XRL A,R3	8B H	MOV direct,R3	AB H	MOV R3,direct
6C H	XRL A,R4	8C H	MOV direct,R4	AC H	MOV R4,direct
6D H	XRL A,R5	8D H	MOV direct,R5	AD H	MOV R5,direct
6E H	XRL A,R6	8E H	MOV direct,R6	AE H	MOV R6,direct
6F H	XRL A,R7	8F H	MOV direct,R7	AF H	MOV R7,direct
70 H	JNZ rel	90 H	MOV DPTR,#data16	B0 H	ANL C,bit
71 H	ACALL addr11	91 H	ACALL addr11	B1 H	ACALL addr11
72 H	ORL C,direct	92 H	MOV bit,C	B2 H	CPL bit
73 H	JMP @A+DPTR	93 H	MOVC A,@A+DPTR	B3 H	CPL C
74 H	MOV A,#data	94 H	SUBB A,#data	B4 H	CJNE A,#data,rel
75 H	MOV direct,#data	95 H	SUBB A,direct	B5 H	CJNE A,direct,rel
76 H	MOV @R0,#data	96 H	SUBB A,@R0	B6 H	CJNE @R0,#data,rel
77 H	MOV @R1,#data	97 H	SUBB A,@R1	B7 H	CJNE @R1,#data,rel
78 H	MOV R0.#data	98 H	SUBB A,R0	B8 H	CJNE R0,#data,rel
79 H	MOV R1.#data	99 H	SUBB A,R1	B9 H	CJNE R1,#data,rel
7A H	MOV R2.#data	9A H	SUBB A,R2	BA H	CJNE R2,#data,rel
7B H	MOV R3.#data	9B H	SUBB A,R3	BB H	CJNE R3,#data,rel
7C H	MOV R4.#data	9C H	SUBB A,R4	BC H	CJNE R4,#data,rel
7D H	MOV R5.#data	9D H	SUBB A,R5	BD H	CJNE R5,#data,rel
7E H	MOV R6.#data	9E H	SUBB A,R6	BE H	CJNE R6,#data,rel
7F H	MOV R7.#data	9F H	SUBB A,R7	BF H	CJNE R7,#data,rel

Table 6-14: Instruction Set in Hexadecimal Order (2/3)

Opcode	Mnemonic	Opcode	Mnemonic
C0 H	PUSH direct	D0 H	POP direct
C1 H	AJMP addr11	D1 H	ACALL addr11
C2 H	CLR bit	D2 H	SETB bit
C3 H	CLR C	D3 H	SETB C
C4 H	SWAP A	D4 H	DA A
C5 H	XCH A,direct	D5 H	DJNZ direct,rel
C6 H	XCH A,@R0	D6 H	XCHD A,@R0
C7 H	XCH A,@R1	D7 H	XCHD A,@R1
C8 H	XCH A,R0	D8 H	DJNZ R0,rel
C9 H	XCH A,R1	D9 H	DJNZ R1,rel
CA H	XCH A,R2	DA H	DJNZ R2,rel
CB H	XCH A,R3	DB H	DJNZ R3,rel
CC H	XCH A,R4	DC H	DJNZ R4,rel
CD H	XCH A,R5	DD H	DJNZ R5,rel
CE H	XCH A,R6	DE H	DJNZ R6,rel
CF H	XCH A,R7	DF H	DJNZ R7,rel
E0 H	MOVX A,@DPTR	F0 H	MOVX @DPTR,A
E1 H	AJMP addr11	F1 H	ACALL addr11
E2 H	MOVX A,@R0	F2 H	MOVX @R0,A
E3 H	MOVX A,@R1	F3 H	MOVX @R1,A
E4 H	CLR A	F4 H	CPL A
E5 H	MOV A,direct	F5 H	MOV direct,A
E6 H	MOV A,@R0	F6 H	MOV @R0,A
E7 H	MOV A,@R1	F7 H	MOV @R1,A
E8 H	MOV A,R0	F8 H	MOV R0,A
E9 H	MOV A,R1	F9 H	MOV R1,A
EA H	MOV A,R2	FA H	MOV R2,A
EB H	MOV A,R3	FB H	MOV R3,A
EC H	MOV A,R4	FC H	MOV R4,A
ED H	MOV A,R5	FD H	MOV R5,A
EE H	MOV A,R6	FE H	MOV R6,A
EF H	MOV A,R7	FF H	MOV R7,A

Table 6-15: Instruction Set in Hexadecimal Order (3/3)

Instructions that Affect Flags

Instruction	Affected Flag			Instruction	Affected Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C, bit	X		
MUL	0	X		ANL C, /bit	X		
DIV	0	X		ORL C, bit	X		
DA	X			ORL C, /bit	X		
RRC	X			MOV C, bit	X		
RLC	X			CJNE	X		
SETB C	1						

Table 6-16: Instructions Affecting Flags

CAUTION

Operations affecting the PSW or bits in the PSW will also affect flag settings

6.3. 80515 HARDWARE DESCRIPTION

The 80515 core implemented in the 71M651X chips consists of:

1. Control processor unit (CPU), also referred to as MPU throughout this document
2. Arithmetic-logic unit
3. Clock control unit
4. Memory control unit
5. RAM and SFR control unit
6. Ports registers unit
7. Timer 0, 1 unit
8. Serial 0, 1 interfaces
9. Watchdog timer
10. Interrupt service routine unit

6.3.1. Block Diagram

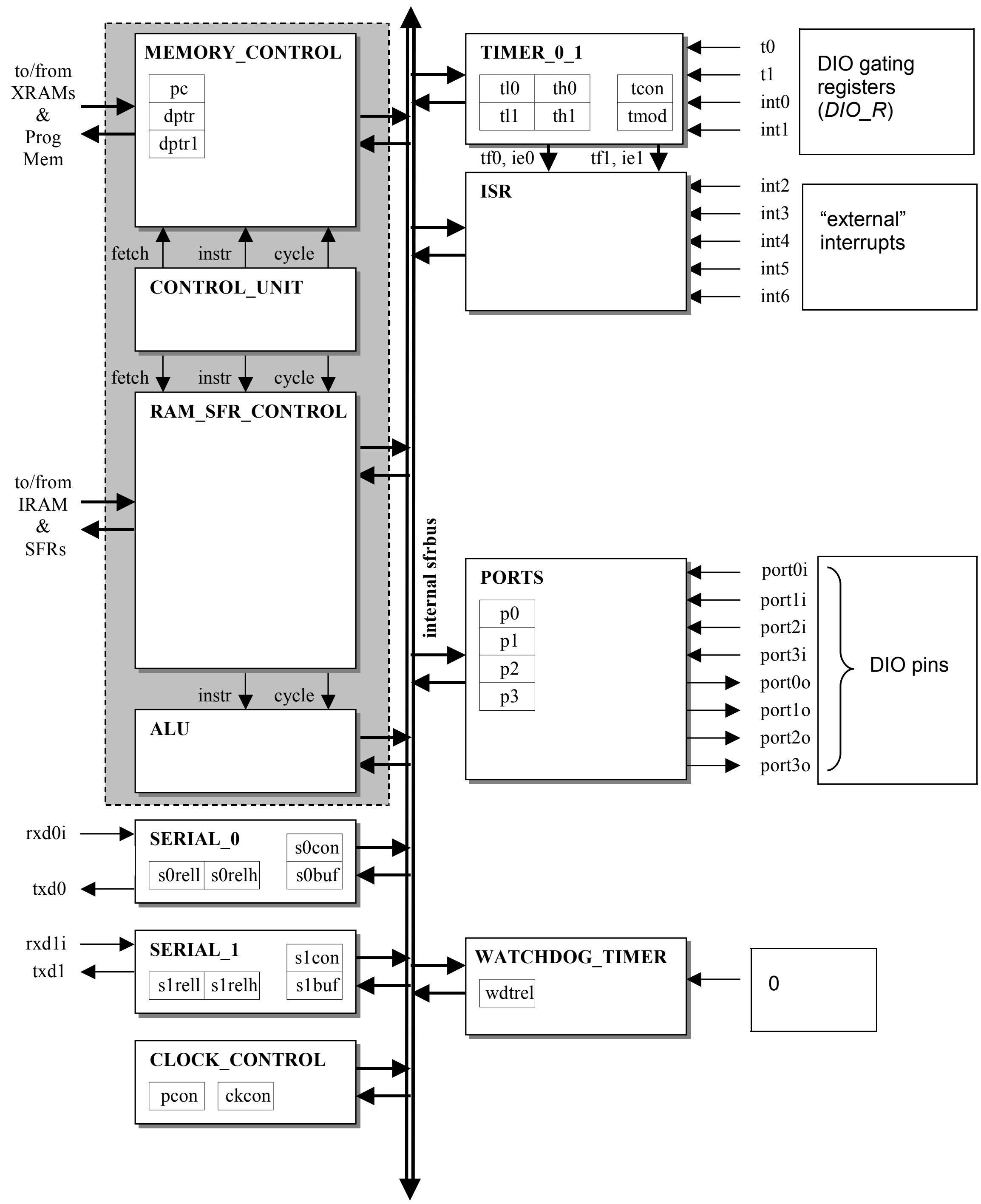


Figure 6-2: 80515 MPU Block Diagram

6.3.2. 80515 MPU

The 80515 MPU is composed of four components:

1. Control unit
2. Arithmetic-logic unit
3. Memory control unit
4. RAM and SFR control unit

The 80515 MPU allows instruction fetch from program memory and instruction execution using RAM or SFR. The following chapter describes the main MPU registers.

Accumulator

ACC is the accumulator register. Most instructions use the accumulator to hold the operand. The mnemonics for accumulator-specific instructions refer to accumulator as "A", not ACC.

The B Register

The B register is used during multiply and divide instructions. It can also be used as a scratch-pad register to hold temporary data.

Program Status Word (PSW)

MSB LSB

CV	AC	F0	RS1	RS	OV	-	P

Table 6-17: PSW Register Flags

Bit	Symbol	Function
PSW.7	CV	Carry flag
Psw.6	AC	Auxiliary Carry flag for BCD operations
PSW.5	F0	General purpose Flag 0 available for user
PSW.4	RS1	Register bank select control bits. The contents of rs1 and rs0 select the working register bank as follows: (0, 0): Bank 0 (0x00-0x07) (0,1): Bank 1 (0x08-0x0F) (1, 0): Bank 2 (0x10-0x17) (1, 1): Bank 3 (0x18-0x1F)
PSW.3	RS0	
PSW.2	OV	Overflow flag
PSW.1	-	User defined flag
PSW.0	P	Parity flag, affected by hardware to indicate odd / even number of "one" bits in the Accumulator, i.e. even parity.

Table 6-18: PSW Bit Functions

The state of bits RS1 and RS0 select the working registers bank as follows:

rs1/rs0	Bank selected	Location
00	Bank 0	(00H – 07H)
01	Bank 1	(08H – 0FH)
10	Bank 2	(10H – 17H)
11	Bank 3	(18H – 1FH)

Table 6-19: Register Bank Location

Stack Pointer

The stack pointer is a 1-byte register initialized to 07H after reset. This register is incremented before PUSH and CALL instructions, causing the stack to begin at location 08H.

Data Pointer

The data pointer (DPTR) is 2 bytes wide. The lower part is DPL, and the highest is DPH. It can be loaded as a 2-byte register (MOV DPTR,#data16) or as two registers (e.g. MOV DPL,#data8). It is generally used to access external code or data space (e.g. MOVC A,@A+DPTR or MOVX A,@DPTR respectively).

Program Counter

The program counter (PC) is 2 bytes wide initialized to 0000H after reset. This register is incremented during the fetching operation code or when operating on data from program memory.

Ports

Ports 'P0', 'P1', 'P2' and 'P3' are Special Function Registers. The contents of the SFR can be observed on corresponding pins on the chip. Writing a '1' to any of the ports causes the corresponding pin to be at high level (VCC), and writing a '0' causes the corresponding pin to be held at low level (GND).

All four ports on the chip are bi-directional. Each of them consists of a Latch (SFR 'P0' to 'P3'), an output driver, and an input buffer, therefore the MPU can output or read data through any of these ports if they are not used for alternate purposes.

Timers 0 and 1

The 80515 has two 16-bit timer/counter registers: Timer 0 and Timer 1. These registers can be configured for counter or timer operations.

In timer mode, the register is incremented every machine cycle meaning that it counts up after every 12 periods of the MPU clock signal.

In counter mode, the register is incremented when the falling edge is observed at the corresponding input pin t0 or t1. Since it takes 2 machine cycles to recognize a 1-to-0 event, the maximum input count rate is 1/2 of the oscillator frequency. There are no restrictions on the duty cycle, however to ensure proper recognition of 0 or 1 state, an input should be stable for at least 1 machine cycle.

Four operating modes can be selected for Timer 0 and Timer 1. Two Special Function Registers (TMOD and TCON) are used to select the appropriate mode.

Bits TR1 (TCON.6) and TR0 (TCON.4) in the TCON register start their associated timers when set.

Timer/Counter Mode Control Register (TMOD)

MSB ... LSB

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				**Timer 0**			

Table 6-20: The TMOD Register

Bit	Symbol	Function
TMOD.7 TMOD.3	Gate	If set, enables external gate control (pin int0 or int1 for Counter 0 or 1, respectively). When int0 or int1 is high, and trx bit is set (see TCON register), a counter is incremented every falling edge on t0 or t1 input pin
TMOD.6 TMOD.2	C/T	Selects Timer or Counter operation. When set to 1, a Counter operation is performed. When cleared to 0, the corresponding register will function as a Timer.
TMOD.5 TMOD.1	M1	Selects the mode for Timer/Counter 0 or Timer/Counter 1, as shown in TMOD description.
TMOD.4 TMOD.0	M0	Selects the mode for Timer/Counter 0 or Timer/Counter 1, as shown in TMOD description.

Table 6-21: The TMOD Register Bits Description

M1	M0	Mode	Function
0	0	Mode 0	13-bit Counter/Timer with 5 lower bits in the TL0 or TL1 register and the remaining 8 bits in the TH0 or TH1 register (for Timer 0 and Timer 1, respectively). The 3 high order bits of TL0 and TL1 are held at zero.
0	1	Mode 1	16-bit Counter/Timer.
1	0	Mode2	8-bit auto-reload Counter/Timer. The reload value is kept in TH0 or TH1, while TL0 or TL1 is incremented every machine cycle. When tl(x) overflows, a value from th(x) is copied to tl(x).
1	1	Mode3	If Timer 1 m1 and m0 bits are set to '1', Timer 1 stops. If Timer 0 m1 and m0 bits are set to '1', Timer 0 acts as two independent 8 bit Timer/Counters.

Table 6-22: Timers/Counters Mode Description

TL0 is affected by tr0 and gate control bits, and sets TF0 flag on overflow.

TH0 is affected by tr1 bit, and sets TF1 flag on overflow.

Timer/Counter Control Register (TCON)

MSB ... LSB

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Table 6-23: The TCON Register

Bit	Symbol	Function
TCON.7	TF1	The Timer 1 overflow flag is set by hardware when Timer 1 overflows. This flag can be cleared by software and is automatically cleared when an interrupt is processed.
TCON.6	TR1	Timer 1 Run control bit. If cleared, Timer 1 stops.
TCON.5	TF0	Timer 0 overflow flag set by hardware when Timer 0 overflows. This flag can be cleared by software and is automatically cleared when an interrupt is processed.
TCON.4	TR0	Timer 0 Run control bit. If cleared, Timer 0 stops.
TCON.3	IE1	Interrupt 1 edge flag is set by hardware when the falling edge on external pin int1 is observed. Cleared when an interrupt is processed.
TCON.2	IT1	Interrupt 1 type control bit. Selects either the falling edge or low level on input pin to cause an interrupt.
TCON.1	IE0	Interrupt 0 edge flag is set by hardware when the falling edge on external pin int0 is observed. Cleared when an interrupt is processed.
TCON.0	IT0	Interrupt 0 type control bit. Selects either the falling edge or low level on input pin to cause interrupt.

Table 6-24: The TCON Register Bit Functions

Allowed Combinations of Operation Modes

Table 6-25 specifies the combinations of operation modes allowed for timer 0 and timer 1.

	Timer 1		
	Mode 0	**Mode 1**	**Mode 2**
Timer 0 - mode 0	YES	YES	YES
Timer 0 - mode 1	YES	YES	YES
Timer 0 - mode 2	Not allowed	Not allowed	YES

Table 6-25: Timer Modes

Timer/Counter Mode Control register (PCON):

MSB							LSB
SMOD							

Table 6-26: The PCON Register

The SMOD bit in the PCON register doubles the baud rate when set.

6.3.3. Serial Interface 0 and 1

The serial buffer consists of two separate registers, a transmit buffer and a receive buffer.

Writing data to the Special Function Register S0BUF or S1BUF sets this data in the serial output buffer and starts transmission. Reading from the S0BUF or S1BUF reads data from the serial receive buffer. The serial port can simultaneously transmit and receive data. It can also buffer 1 byte at receive, preventing the receive data from being lost if the MPU reads the first byte before transmission of the second byte is completed.

Serial Interface 0 Modes

The Serial Interface 0 can operate in 4 modes:

Mode 0

Pin rxd0 serves as an input and an output. Txd0 outputs the shift clock. 8 bits are transmitted starting with the LSB. The baud rate is fixed at 1/12 of the MPU frequency. Reception is initialized in Mode 0 by setting the flags in S0CON as follows: RI0=0 and REN0=1. In other modes, when REN0 = 1, a start bit initiates receiving serial data.

Mode 1

Pin rxd0 serves as an input, and txd0 serves as a serial output. No external shift clock is used. 10 bits are transmitted: a start bit (always 0), 8 data bits (LSB first), and a stop bit (always 1). On receive, a start bit synchronizes the transmission. 8 data bits are available by reading S0BUF, and the stop bit sets the flag RB80 in the Special Function Register S0CON. In mode 1 either the internal baud rate generator or timer 1 can be use to specify the baud rate.

Mode 2

This mode is similar to Mode 1, with two differences. The baud rate is fixed at 1/32 or 1/64 of the oscillator frequency and 11 bits are transmitted or received: a start bit (0), 8 data bits (LSB first), a programmable 9^{th} bit, and a stop bit (1). The 9^{th} bit can be used to control the parity of the serial interface: at transmission, bit TB80 in S0CON is output as the 9^{th} bit, and at receive, the 9^{th} bit affects RB80 in the Special Function Register S0CON.

Mode 3

The only difference between Mode 2 and Mode 3 is that in Mode 3, either the internal baud rate generator or timer 1 can be use to specify the baud rate.

The common FLAG protocol requires the data format to be 7E1. This can be implemented using one of the 8-bit modes, where the MSB (bit 0) is the parity bit. In this mode, the MPU calculates parity.

Serial Interface 0 Control Register (S0CON).

The function of the serial port 0 depends on the setting of the Serial Port Control Register S0CON.

MSB							LSB
SM0	SM1	SM20	REN0	TB80	RB80	TI0	RI0

Table 6-27: The S0CON Register

Bit	Symbol	Function
S0CON.7	SM0	Sets baud rate
S0CON.6	SM1	Sets baud rate
S0CON.5	SM20	reserved
S0CON.4	REN0	If set, enables serial reception. Cleared by software to disable reception.
S0CON.3	TB80	The 9th transmitted data bit in Modes 2 and 3. Set or cleared by the MPU, depending on the function it performs (parity check, multiprocessor communication etc.)
S0CON.2	RB80	In Modes 2 and 3 it is the 9th data bit received. In Mode 1, if SM20 is 0, RB80 is the stop bit. In Mode 0 this bit is not used. Must be cleared by software
S0CON.1	TI0	Transmit interrupt flag, set by hardware after completion of a serial transfer. Must be cleared by software.
S0CON.0	RI0	Receive interrupt flag, set by hardware after completion of a serial reception. Must be cleared by software

Table 6-28: The S0CON Bit Functions

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	Fclk/12
0	1	1	8-bit UART	Variable
1	0	2	9-bit UART	Fclk/32 or /64
1	1	3	9-bit UART	Variable

Table 6-29: Serial Port 0 Modes

The speed in Mode 2 depends on the smod bit in the Special Function Register PCON when smod = 1, Fclk/32. See the PCON register description.

Serial Interface 1 Modes

The Serial Interface 1 can operate in 2 modes:

sm	Mode	Description	Baud Rate
0	A	9-bit UART	variable
1	B	8-bit UART	variable

Table 6-30: Serial 1 Modes

Mode A

This mode is similar to Mode 2 and 3 of serial interface 0. 11 bits are transmitted or received: a start bit (0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (1). The 9th bit can be used to control the parity of the serial interface: at transmission, bit tb81 in S1CON is output as the 9th bit, and at receive, the 9th bit affects rb81 in the Special Function Register S1CON. The only difference between Mode 3 and A is that in Mode A, only the internal baud rate generator can be used to specify the baud rate.

Mode B

This mode is similar to Mode 1 of serial interface 0. Pin rxd1 serves as an input, and txd1 serves as a serial output. No external shift clock is used. 10 bits are transmitted: a start bit (always 0), 8 data bits (LSB first), and a stop bit (always 1). On receive, a start bit synchronizes the transmission. 8 data bits are available by reading S1BUF, and the stop bit sets the flag rb81 in the Special Function Register S1CON. In mode B, the internal baud rate generator specifies the baud rate.

Serial Interface 1 Control Register (S1CON).

The function of the serial port depends on the setting of the Serial Port Control Register S1CON.

MSB LSB

sm	-	sm21	REN1	tb81	rb81	ti1	ri1

Table 6-31: The S1CON Register

Bit	Symbol	Function
S1CON.7	sm	Sets baud rate
S1CON.5	sm21	Enables the multiprocessor communication feature (see description above).
S1CON.4	REN1	If set, enables serial reception. Cleared by software to disable reception.
S1CON.3	Tb81	The 9^{th} transmitted data bit in Mode A. Set or cleared by the MPU, depending on the function it performs (parity check, multiprocessor communication etc.)
S1CON.2	Rb81	In Modes 2 and 3, it is the 9^{th} data bit received. In Mode B, if sm21 is 0, rb81 is the stop bit. Must be cleared by software
S1CON.1	ti1	Transmit interrupt flag, set by hardware after completion of a serial transfer. Must be cleared by software.
S1CON.0	ri1	Receive interrupt flag, set by hardware after completion of a serial reception. Must be cleared by software

Table 6-32: The S1CON Bit Functions

Baud Rate Generator

Serial 0 modes 1 and 3 only (Fclk = MPU clock rate):

Timer1 baud rate generator (WDCON.7 = 0)

$$baudrate = \frac{F_{clk} \cdot 2^{s\bmod}}{384 \cdot (256 - th1)}$$

Internal baud rate generator (WDCON.7 = 1)

$$baudrate = \frac{F_{clk} \cdot 2^{s\bmod}}{64 \cdot (2^{10} - s0rel)}$$

s0rel is a 10 bit value formed by concatenating S0RELH and S0RELL as follows:

s0rel = {S0RELH.[1:0], S0RELL.[7:0]}

Serial 1 all modes: (Fclk = MPU clock rate):

Internal baud rate generator only

$$baudrate = \frac{F_{clk}}{32 \cdot (2^{10} - s1rel)}$$

s1rel is a 10 bit value formed by concatenating S1RELH and S1RELL as follows:

s1rel = {S1RELH.[1:0], S1RELL.[7:0]}

6.3.4. Software Watchdog Timer

The watchdog timer is a 16-bit counter that is incremented once every 24 or 384 clock cycles. After an external reset, the watchdog timer is disabled and all registers are set to zero.

Software Watchdog Timer structure

The watchdog consists of a 16-bit counter (wdt), a reload register (WDTREL), prescalers (by 2 and by 16), and control logic.

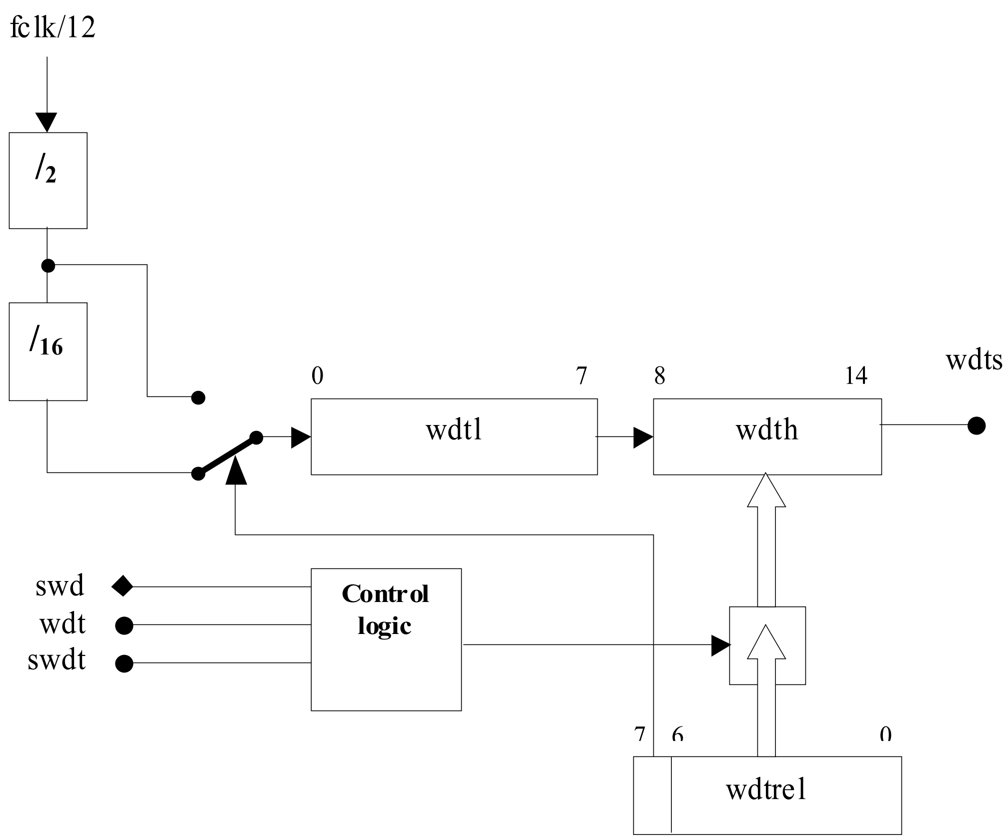


Figure 6-3: Watchdog Block Diagram

WD Timer Start Procedure

During an active internal rst signal, the programmer can start the watchdog later. It will occur when the swd signal becomes active. Once the watchdog is started, it cannot be stopped unless the internal rst signal becomes active.

When the wdt registers enters the state 0x7CFF , an asynchronous wdts signal will become active. The signal wdts sets bit 6 in the IP0 register and requests a reset state. Wdts is cleared either by the rst signal or changing the state of the wdt timer.

Refreshing the WD Timer

The watchdog timer must be refreshed regularly to prevent the reset request signal from becoming active. This requirement imposes an obligation on the programmer to issue two instructions. The first instruction sets wdt and the second instruction sets swdt. The maximum delay allowed between setting wdt and swdt is 12 clock cycles. If this period has expired and swdt has not been set, the WDT is automatically reset, otherwise the watchdog timer is reloaded with the content of the WDTREL register and wdt is automatically reset.

Since the WDT requires exact timing, firmware needs to be designed with special care in order to avoid unwanted WDT resets. TERIDIAN strongly discourages the use of the software WDT.

Special Function Registers for the WD Timer

Interrupt Enable 0 Register (IEN0):

MSB							LSB
eal	wdt	et2	es0	et1	ex1	et0	ex0

Table 6-33: The IEN0 Register

Bit	Symbol	Function
IEN0.6	wdt	Watchdog timer refresh flag. Set to initiate a refresh of the watchdog timer. Must be set directly before swdt is set to prevent an unintentional refresh of the watchdog timer. Wdt is reset by hardware 12 clock cycles after it has been set.

Table 6-34: The IEN0 Bit Functions

The remaining bits in the IEN0 register are not used for watchdog control

Interrupt Enable 1 Register (IEN1):

MSB							LSB
exen2	swdt	ex6	ex5	ex4	ex3	ex2	

Table 6-35: The IEN1 Register

Bit	Symbol	Function
IEN1.6	swdt	Watchdog timer start/refresh flag. Set to activate/refresh the watchdog timer. When directly set after setting wdt, a watchdog timer refresh is performed. Bit swdt is reset by the hardware 12 clock cycles after it has been set.

Table 6-36: The IEN1 Bit Functions

The remaining bits in the IEN1 register are not used for watchdog control

Interrupt Priority 0 Register (IP0):

MSB LSB

owds	wdts	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0

Table 6-37: The IP0 Register

Bit	Symbol	Function
IP0.6	wdts	Watchdog timer status flag. Set by hardware when the watchdog timer was started. Can be read by software.

Table 6-38: The IP0 Bit Functions

The remaining bits in the IP0 register are not used for watchdog control.

Watchdog Timer Reload Register (WDTREL):

MSB LSB

7	6	5	4	3	2	1	0

Table 6-39: The WDTREL Register

Bit	Symbol	Function
Wdtrel.7	7	Prescaler select bit. When set, the watchdog is clocked through an additional divide-by-16 prescaler
Wdtrel.6 to WDTREL.0	6-0	Seven bit reload value for the high-byte of the watchdog timer. This value is loaded to the wdt when a refresh is triggered by a consecutive setting of bits wdt and swdt.

Table 6-40: The WDTREL Bit Functions

The WDTREL register can be loaded and read at any time.

6.3.5. The Interrupt Service Routine Unit

The 80515 provides 11 interrupt sources with four priority levels. Each source has its own request flag(s) located in a special function register (TCON, IRCON, SCON). Each interrupt requested by the corresponding flag can be individually enabled or disabled by the enable bits in SFRs IEN0, IEN1, and IEN2.

6.3.5.1. Interrupt Overview

When an interrupt occurs, the MPU will vector to the predetermined address as shown in Table 6-58. Once interrupt service has begun, it can be interrupted only by a higher priority interrupt. The interrupt service is terminated by a return from instruction, "RETI". When a RETI instruction is performed, the processor will return to the instruction that would have been next when the interrupt occurred.

When the interrupt condition occurs, the processor will also indicate this by setting a flag bit. This bit is set regardless of whether the interrupt is enabled or disabled. Each interrupt flag is sampled once per machine cycle, then samples are polled by the hardware. If the sample indicates a pending interrupt when the interrupt is enabled, then the interrupt request flag is set. On the next instruction cycle, the interrupt will be acknowledged by hardware forcing an LCALL to the appropriate vector address, if the following conditions are met:

- No interrupt of equal or higher priority is already in progress.
- An instruction is currently being executed and is not completed.
- The instruction in progress is not RETI or any write access to the registers IEN0, IEN1, IEN2, IP0 or IP1.

Interrupt response will require a varying amount of time depending on the state of the microcontroller when the interrupt occurs. If the microcontroller is performing an interrupt service with equal or greater priority, the new interrupt will not be invoked. In other cases, the response time depends on the current instruction. The fastest possible response to an interrupt is 7 machine cycles. This includes one machine cycle for detecting the interrupt and six cycles to perform the LCALL.

6.3.5.2. Special Function Registers for Interrupts

Interrupt Enable 0 Register (ie0)

MSB							LSB
eal	wdt		es0	et1	ex1	et0	ex0

Table 6-41: The IEN0 Register

Bit	Symbol	Function
IEN0.7	eal	eal=0 – disable all interrupts
IEN0.6	wdt	Not used for interrupt control
IEN0.5	-	
IEN0.4	es0	es0=0 – disable serial channel 0 interrupt
IEN0.3	et1	et1=0 – disable timer 1 overflow interrupt
IEN0.2	ex1	ex1=0 – disable external interrupt 1
IEN0.1	et0	et0=0 – disable timer 0 overflow interrupt
IEN0.0	ex0	ex0=0 – disable external interrupt 0

Table 6-42: The IEN0 Bit Functions

Interrupt Enable 1 Register (IE1)

MSB							LSB
	swdt	ex6	ex5	ex4	ex3	ex2	

Table 6-43: The IEN1 Register

Bit	Symbol	Function
IEN1.7	-	
IEN1.6	swdt	Not used for interrupt control
IEN1.5	ex6	ex6=0 – disable external interrupt 6
IEN1.4	ex5	ex5=0 – disable external interrupt 5
IEN1.3	ex4	ex4=0 – disable external interrupt 4
IEN1.2	ex3	ex3=0 – disable external interrupt 3
IEN1.1	ex2	ex2=0 – disable external interrupt 2
IEN1.0	-	

Table 6-44: The IEN1 Bit Functions

Interrupt Enable 2 Register (ie2)

MSB							LSB
-	-	-	-	-	-	-	es1

Table 6-45: The IEN2 Register

Bit	Symbol	Function
IEN2.0	es1	es1=0 – disable serial channel 1 interrupt

Table 6-46: The IEN2 Bit Functions

Timer/Counter Control Register (TCON)

MSB | | | | | | | LSB

tf1	tr1	tf0	tr0	ie1	it1	ie0	it0

Table 6-47: The TCON Register

Bit	Symbol	Function
TCON.7	tf1	Timer 1 overflow flag
TCON.6	tr1	Not used for interrupt control
TCON.5	tf0	Timer 0 overflow flag
TCON.4	tr0	Not used for interrupt control
TCON.3	ie1	External interrupt 1 flag
TCON.2	it1	External interrupt 1 type control bit
TCON.1	ie0	External interrupt 0 flag
TCON.0	it0	External interrupt 0 type control bit

Table 6-48: The TCON Bit Functions

Interrupt Request Register (IRCON)

MSB | | | | | | | LSB

		iex6	iex5	iex4	iex3	iex2	

Table 6-49: The IRCON Register

Bit	Symbol	Function
IRCON.7	-	
IRCON.6	-	
IRCON.5	iex6	External interrupt 6 edge flag
IRCON.4	iex5	External interrupt 5 edge flag
IRCON.3	iex4	External interrupt 4 edge flag
IRCON.2	iex3	External interrupt 3 edge flag
IRCON.1	iex2	External interrupt 2 edge flag
IRCON.0	-	

Table 6-50: The IRCON Bit Functions

Only tf0 and tf1 (timer 0 and timer 1 overflow flag) will be automatically cleared by hardware when the service routine is called (Signals t0ack and t1ack – port ISR – active high when the service routine is called).

6.3.5.3. Interrupt Priority Level Structure

All interrupt sources are combined in groups:

group			
0	External interrupt 0	Serial channel 1 interrupt	
1	Timer 0 interrupt	-	External interrupt 2
2	External interrupt 1	-	External interrupt 3
3	Timer 1 interrupt	-	External interrupt 4
4	Serial channel 0 interrupt	-	External interrupt 5
5	-	-	External interrupt 6

Table 6-51: Priority Level Groups

Each group of interrupt sources can be programmed individually to one of four priority levels by setting or clearing one bit in the special function register IP0 and one in IP1. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced first.

The functionality of the external interrupts is described in Table 6-51.

External Interrupt	Connection	Polarity	Flag Reset
0	Digital I/O High Priority	see *DIO_Rx*	automatic
1	Digital I/O Low Priority	see *DIO_Rx*	automatic
2	Comparator	falling	automatic
3	CE_BUSY	falling	automatic
4	Comparator	rising	automatic
5	EEPROM busy	falling	automatic
6	XFER_BUSY OR RTC_1SEC	falling	manual

Table 6-52: External MPU Interrupts

Enable Bit	Description
EX0	Enable external interrupt 0
EX1	Enable external interrupt 1
EX2	Enable external interrupt 2
EX3	Enable external interrupt 3
EX4	Enable external interrupt 4
EX5	Enable external interrupt 5
EX6	Enable external interrupt 6
EX_XFER	Enable XFER_BUSY interrupt
EX_RTC	Enable RTC_1SEC interrupt

Flag Bit	Description
IE0	External interrupt 0 flag
IE1	External interrupt 1 flag
IEX2	External interrupt 2 flag
IEX3	External interrupt 3 flag
IEX4	External interrupt 4 flag
IEX5	External interrupt 5 flag
IEX6	External interrupt 6 flag
IE_XFER	XFER_BUSY interrupt flag
IE_RTC	RTC_1SEC interrupt flag

Table 6-53: Control Bits for External Interrupts

SFR (special function register) enable bits must be set to permit any of these interrupts to occur. Likewise, each interrupt has its own flag bit which is set by the interrupt hardware and is reset automatically by the MPU interrupt handler (0 through 5). XFER_BUSY and RTC_1SEC, which are OR-ed together, have their own enable and flag bits in addition to the interrupt 6 enable and flag bits (see Table 6-52), and these interrupts must be cleared by the MPU software.

Interrupt Priority 0 Register (IP0)

MSB LSB

owds	Wdts	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0

Table 6-54: The IP0 Register:

owds, wdts are not used for interrupt controls

Interrupt Priority 1 Register (IP1)

MSB LSB

-	-	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0

Table 6-55: The IP1 Register:

IP1.x	IP0.x	Priority Level
0	0	Level0 (lowest)
0	1	Level1
1	0	Level2
1	1	Level3 (highest)

Table 6-56: Priority Levels

Bit	Group		
Ip1.0, IP0.0	External interrupt 0	Serial channel 1 interrupt	
Ip1.1, IP0.1	Timer 0 interrupt	-	External interrupt 2
Ip1.2, IP0.2	External interrupt 1	-	External interrupt 3
Ip1.3, IP0.3	Timer 1 interrupt	-	External interrupt 4
Ip1.4, IP0.4	Serial channel 0 interrupt	-	External interrupt 5
		-	External interrupt 6

Table 6-57: Groups of Priority

External interrupt 0	Polling sequence
Serial channel 1 interrupt	
Timer 0 interrupt	
External interrupt 2	
External interrupt 1	
External interrupt 3	
Timer 1 interrupt	
External interrupt 4	
Serial channel 0 interrupt	
External interrupt 5	
External interrupt 6	

Table 6-58: Polling Sequence:

6.3.5.4. Interrupt Sources and Vectors

Interrupt Request Flags	Interrupt Vector Address
ie0 – External interrupt 0	0003H
tf0 – Timer 0 interrupt	000BH
ie1 – External interrupt 1	0013H
tf1 – Timer 1 interrupt	001BH
RI0/TI0 – Serial channel 0 interrupt	0023H
ri1/ti1 – Serial channel 1 interrupt	0083H
iex2 – External interrupt 2	004BH
iex3 – External interrupt 3	0053H
iex4 – External interrupt 4	005BH
iex5 – External interrupt 5	0063H
iex6 – External interrupt 6	006BH

Table 6-59: Interrupt Vectors

External Interrupt Edge Detect

The external interrupts 4, 5 and 6 are activated by a positive transition. The external source must hold the request pin low (high for int2 and int3, if it is programmed to be negative transition-active) for at least one MPU clk period. Afterwards, it must be held high (low) for at least one MPU clk period to ensure the transition is recognized and the corresponding interrupt request flag is set.

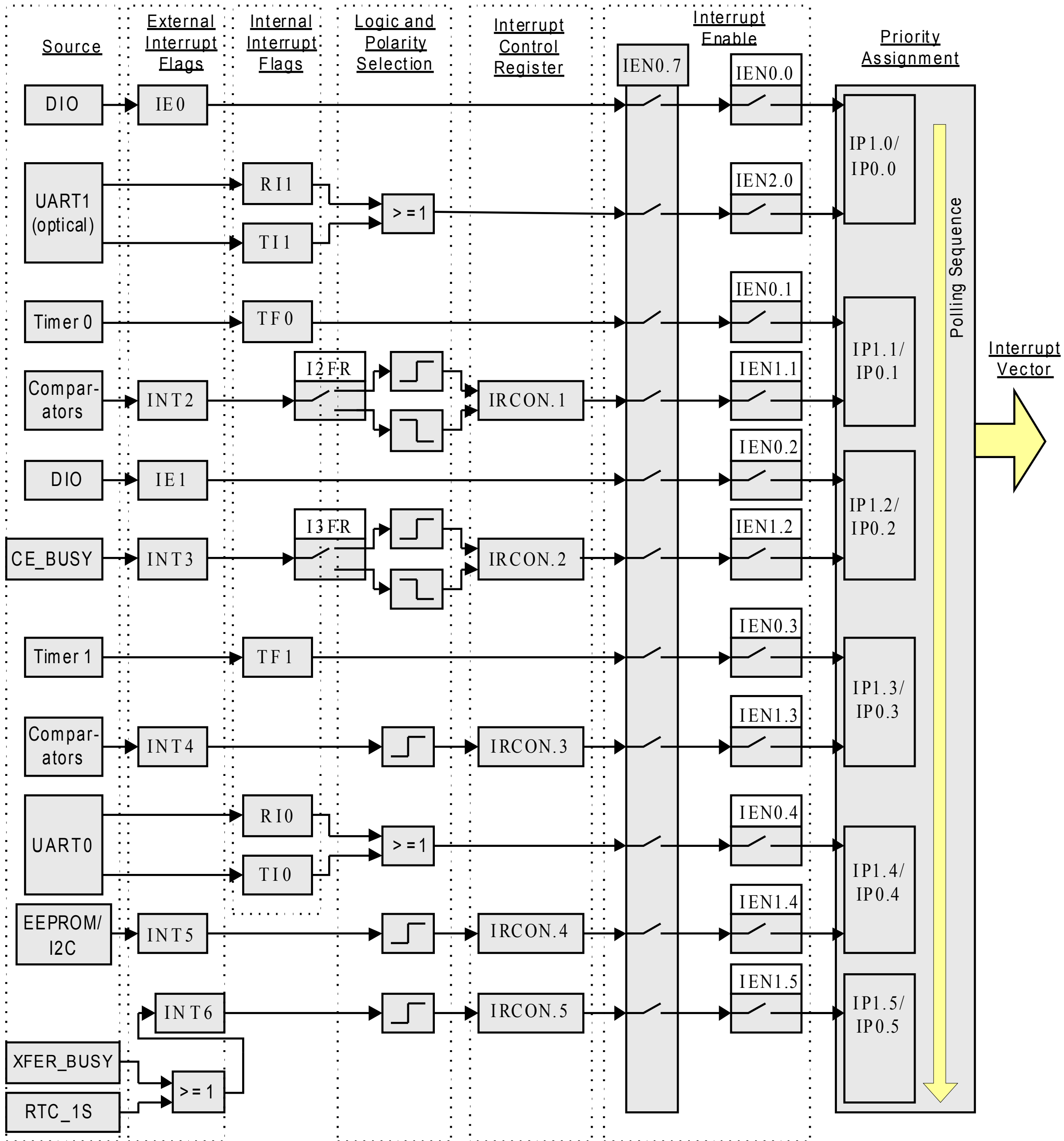


Figure 6-4: Interrupt Sources Diagram

7

7. ACRONYMS

AC	Alternating Current – current with changing polarity
ANSI	American National Standardization Institution, part of ISO
ANSI C	C Programming Language, standardized by ANSI in 1983. Keil C, used throughout this User's Guide is not strictly ANSI compliant.
API	Application Programming Interface
C	The C Programming Language, as defined by Kernighan and Ritchie
CE	Computation Engine
<CR>	Carriage Return or Enter Key on PC Keyboard
COM	Communication Port
CPU	Control Processor Unit (MPU)
DC	Direct Current
EEP	Engineering Evaluation Platform (Demo Board)
EEPROM	Electrically Eraseable PROM
FLAG	Ferranti-Landis&Gyr. An international protocol for reading of meters using an optical port.
GB	Gigabyte(s)
ICE	In-Circuit Emulator
IDE	Integrated Development Environment – usually a combination of editor, compiler, assembler, linker, debugger, ICE
IEC	International Electrotechnical Commission (Geneva, Switzerland)
INT	Interrupt
ISO	International Standards Organization
ISR	Interrupt Service Routine
KB	Kilobyte(s) – 1,024 bytes
LCD	Liquid Crystal Display
<LF>	Line-feed character
LSB	Least Significant Bit
MB	Megabyte(s) – 1,024 kilobytes
MPU	Microprocessor Unit
MSB	Most Siginificant Bit
NV	Non-Volatile

PC	Personal Computer, Program Counter
PROM	Programmable ROM
PSU	Power Supply Unit
PSW	Program Status Word
RAM	Random Access Memory
ROM	Read Only Memory
SFR	Special Function Register
TSC	TERIDIAN Semiconductor Corporation
USB	Universal Serial Bus
VA	Volt-Amperes (apparent power unit)
VAR	Reactive Power
VARh	Reactive energy unit
W	Watt (power unit)
WD	Watchdog
WDT	Watchdog timer
WEMU51	The emulator control program by Signum Systems
Wh	Watt-Hour (energy unit)

TERIDIAN Semiconductor Corp., 6440 Oak Canyon Rd., Irvine, CA 92618-5201
TEL (714) 508-8800, FAX (714) 508-8877, http://www.teridian.com