



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China





DS4830A

Optical Microcontroller

User's Guide

Rev 0; 12/13

Maxim Integrated cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim Integrated product. No circuit patent licenses are implied. Maxim Integrated reserves the right to change the circuitry and specifications without notice at any time.

Maxim Integrated Products, Inc. 160 Rio Robles, San Jose, CA 95134 USA 1-408-601-1000

© 2013 Maxim Integrated Products

The Maxim logo and Maxim Integrated are registered trademarks of Maxim Integrated Products, Inc.

Contents

SECTION 1 – OVERVIEW	11
SECTION 2 – ARCHITECTURE	13
2.1 – Instruction Decoding	13
2.2 – Register Space	14
2.3 – Memory Types	15
2.3.1 – Flash Memory	15
2.3.2 – SRAM Memory	15
2.3.3 – Utility ROM	15
2.3.4 – Stack Memory	16
2.4 – Program and Data Memory Mapping and Access	16
2.4.1 – Program Memory Access	16
2.4.2 – Program Memory Mapping	17
2.4.3 – Data Memory Access	17
2.4.4 – Data Memory Mapping	18
2.5 – Data Alignment	22
2.6 – Reset Conditions	22
2.6.1 – Power-On/Brownout Reset	22
2.6.2 – Watchdog Timer Reset	23
2.6.3 – External Reset	23
2.6.4 – Internal System Resets	24
2.6.5 – Software Reset	24
2.7 – Clock Generation	24
SECTION 3 – SYSTEM REGISTER DESCRIPTIONS	25
3.1 – Accumulator Pointer Register (AP, 08h[00h])	27
3.2 – Accumulator Pointer Control Register (APC, 08h[01h])	27
3.3 – Processor Status Flags Register (PSF, 08h[04h])	27
3.4 – Interrupt and Control Register (IC, 08h[05h])	28
3.5 – Interrupt Mask Register (IMR, 08h[06h])	28
3.6 – System Control Register (SC, 08h[08h])	28
3.7 – Interrupt Identification Register (IIR, 08h[0Bh])	29
3.8 – Watchdog Control Register (WDCN, 08h[0Fh])	29
3.9 – Accumulator n Register (A[n], 09h[nh])	29
3.10 – Prefix Register (PFX[n], 0Bh[n])	29
3.11 – Instruction Pointer Register (IP, 0Ch[00h])	30
3.12 – Stack Pointer Register (SP, 0Dh[01h])	30
3.13 – Interrupt Vector Register (IV, 0Dh[02h])	30
3.14 – Loop Counter 0 Register (LC[0], 0Dh[06h])	30
3.15 – Loop Counter 1 Register (LC[1], 0Dh[07h])	30
3.16 – Frame Pointer Offset Register (OFFS, 0Eh[03h])	30
3.17 – Data Pointer Control Register (DPC, 0Eh[04h])	31

3.18 – General Register (GR, 0Eh[05h])	31
3.19 – General Register Low Byte (GRL, 0Eh[06h])	31
3.20 – Frame Pointer Base Register (BP, 0Eh[07h])	31
3.21 – General Register Byte-Swapped (GRS, 0Eh[08h])	32
3.22 – General Register High Byte (GRH, 0Eh[09h])	32
3.23 – General Register Sign Extended Low Byte (GRXL, 0Eh[0Ah])	32
3.24 – Frame Pointer Register (FP, 0Eh[0Bh])	32
3.25 – Data Pointer 0 Register (DP[0], 0Fh[03h])	32
3.26 – Data Pointer 1 Register (DP[1], 0Fh[07h])	32
SECTION 4 – PERIPHERAL REGISTER DESCRIPTIONS	33
4.1 – Module 0 Peripheral Registers	34
4.2 – Module 1 Peripheral Registers	35
4.3 – Module 2 Peripheral Registers	36
4.4 – Module 3 Peripheral Registers	37
4.5 – Module 4 Peripheral Registers	38
4.6 – Module 5 Peripheral Registers	39
SECTION 5 – INTERRUPTS	40
5.1 – Servicing Interrupts	41
5.2 – Module Interrupt Identification Registers	42
5.3 – Interrupt System Operation	43
5.3.1 – Synchronous vs. Asynchronous Interrupt Sources	44
5.3.2 – Interrupt Prioritization by Software	44
5.3.3 – Interrupt Exception Window	44
SECTION 6 – DIGITAL-TO-ANALOG CONVERTER (DAC)	45
6.1 – Detailed Description	45
6.1.1 – Reference Selection	46
6.2 – DAC Register Descriptions	46
6.2.1 – DAC Configuration Register (DACCFG)	46
6.2.2 – DAC Data Registers (DACD0-DACD7)	47
6.2.3 – Reference Pin Configuration Register (RPCFG)	47
6.3 – DAC Code Examples	47
SECTION 7 – ANALOG-TO-DIGITAL CONVERTER (ADC)	48
7.1 – Detailed Description	48
7.1.1 – ADC Controller	48
7.1.2 – ADC Conversion Sequencing	49
7.1.3 – Internal Die Temperature Conversion	50
7.1.4 – Sample and Hold Conversion	51
7.1.5 – ADC Frame Sequence	51
7.1.6 – ADC Reference	51
7.1.7 – ADC Conversion Time	52
7.1.8 – Location Override	53
7.1.9 – Averaging	53

7.1.10 – ADC Data Reading	54
7.1.11 – ADC Interrupts	54
7.1.12 – ADC Internal Offset.....	55
7.1.13 – DAC External Reference Pins (REFINA and REFINB) as ADC Channels.....	55
7.1.14 – Fast Conversion Mode (ADST.ENABLE_2X).....	55
7.2 – ADC Register Descriptions.....	56
7.2.1 – ADC Control Register (ADCN).....	56
7.2.2 – ADC Status Register (ADST).....	57
7.2.3 – PIN Select Register (PINSEL)	57
7.2.4 – ADC Status Register (ADST1).....	58
7.2.5 – ADC Address Register (ADADDR)	58
7.2.6 – ADC Data and Configuration Register (ADDATA).....	58
7.2.7 – Reference Pin Configuration Register (RPCFG)	60
7.2.8 – Temperature Control Register (TEMPCN)	61
7.2.9 – Average and Reference Control Register (REFAVG)	61
7.2.10 – ADC Voltage Offset Register (ADVOFF).....	62
7.2.11 – ADC Voltage Scale Trim Registers (ADCG1, ADCG2, ADCG3 and ADCG4)	62
7.3 – ADC Code Examples.....	63
SECTION 8 – SAMPLE AND HOLD	65
8.1 – Detailed Description	65
8.1.1 – Operation	65
8.1.2 – Fast Mode Operation	66
8.1.3 – Sampling Control	67
8.1.4 – Pin Capacitance Discharge	68
8.1.5 – Sample and Hold Data Reading	69
8.1.6 – Sample and Hold Interrupts	69
8.2 – Sample and Hold Register Descriptions.....	70
SECTION 9 – QUICK TRIP (FAST COMPARATOR).....	73
9.1 – Detailed Description	73
9.1.1 – Quick Trip List Sequencing.....	74
9.1.2 – Operation	74
9.1.3 – Setting Quick Trip Thresholds	75
9.1.4 – Quick Trip Interrupts	76
9.2 – Quick Trip Register Descriptions.....	77
SECTION 10 – I ² C-COMPATIBLE MASTER INTERFACE	81
10.1 – Detailed Description	81
10.1.1 – Description of Master I ² C Interface.....	81
10.1.2 – Default Operation.....	81
10.1.3 – I ² C Clock Generation	81
10.1.4 – Timeout.....	82
10.1.5 – Generating a START	83
10.1.6 – Generating a STOP	85

10.1.7 – Transmitting a Slave Address.....	85
10.1.8 – Transmitting Data.....	85
10.1.9 – Receiving Data.....	87
10.1.10 – I ² C Master Clock Stretching.....	88
10.1.11 – Resetting the I ² C Master Controller.....	88
10.1.12 – Alternate Location.....	89
10.1.13 – Operation as a Slave.....	89
10.1.14 – GPIO.....	89
10.2 – I ² C Master Controller Register Description.....	90
SECTION 11 – I ² C-COMPATIBLE SLAVE INTERFACE.....	94
11.1 – Detailed Description.....	95
11.1.1 – Default Operation.....	95
11.1.2 – Slave Addresses.....	95
11.1.3 – I ² C START Detection.....	95
11.1.4 – I ² C STOP Detection.....	95
11.1.5 – Slave Address Matching.....	95
11.1.6 – Advanced Mode Operation RX FIFO and TX Pages.....	97
11.1.7 – Transmitting Data.....	98
11.1.8 – Receiving Data.....	100
11.1.9 – Clock Stretching.....	100
11.1.10 – SMBus Timeout.....	102
11.1.11 – Resetting the I ² C Slave Controller.....	102
11.2 – I ² C Slave Controller Register Description.....	103
SECTION 12 – SERIAL PERIPHERAL INTERFACE (SPI).....	111
12.1 – Serial Peripheral Interface (SPI) Detailed Description.....	111
12.1.1 – SPI Transfer Formats.....	111
12.1.2 – SPI Character Lengths.....	113
12.2 – SPI System Errors.....	113
12.2.1 – Mode Fault.....	113
12.2.2 – Receive Overrun.....	113
12.2.3 – Write Collision While Busy.....	114
12.3 – SPI Interrupts.....	114
12.4 – SPI Master.....	114
12.4.1 – SPI Transfer Baud Rates.....	114
12.4.2 – SPI Master Operation.....	114
12.4.3 – SPI Master Register Descriptions.....	116
12.5 – SPI Slave.....	118
12.5.1 – SPI Slave Select.....	118
12.5.2 – SPI Transfer Baud Rates.....	118
12.5.3 – SPI Slave Operation.....	118
12.5.4 – SPI Slave Register Descriptions.....	119

SECTION 13 – 3-WIRE.....	121
13.1 – Detailed Description	121
13.1.1 – Operation	121
13.2 – 3-Wire Register Descriptions.....	123
SECTION 14 – PWM	124
14.1 – Detailed Description	124
14.1.1 – PWMCN and PWMDATA SFRs	124
14.1.2 – PWMSYNC SFR.....	125
14.2 – Individual PWM Channel Operation	126
14.2.1 – Duty Cycle Register (DCYCn)	126
14.2.2 – PWM Configuration Register (PWMCFGn)	127
14.2.3 – PWM DELAY Register (PWMDLYn).....	131
14.3 – PWM Output Register Descriptions.....	132
14.4 – PWM Output Code Examples	137
SECTION 15 – GENERAL-PURPOSE INPUT/OUTPUT (GPIO) PINS	138
15.1 – Overview.....	138
15.2 – GPIO Port Register Descriptions.....	141
15.2.1 – GPIO Direction Register Port (PD0, PD1, PD2, and PD6).....	141
15.2.2 – GPIO Output Register Port (PO0, PO1, PO2, and PO6).....	141
15.2.3 – GPIO Input Register for Port (PI0, PI1, PI2, and PI6)	141
15.2.4 – GPIO Port External Interrupt Edge Select Register (EIES0, EIES1, EIES2, and EIES6)	141
15.2.5 – GPIO Port External Interrupt Flag Register (EIF0, EIF1, EIF2, and EIF6).....	142
15.2.6 – GPIO Port External Interrupt Enable Register (EIE0, EIE1, EIE2, and EIE6)	142
15.3 – GPIO Code Example	142
15.3.1 – GPIO Pin as Output	142
15.3.2 – GPIO High-Impedance Input	142
15.3.3 – GPIO Weak Pullup Input.....	142
15.3.4 – GPIO Open-Drain Output	142
SECTION 16 – GENERAL-PURPOSE TIMERS	143
16.1 – Detailed Description	143
16.1.1 – Timer Modes	143
16.1.2 – Clock Selection	144
16.1.3 – Timer Clock Prescaler	144
16.2 – Timer Register Descriptions	145
SECTION 17 – SUPPLY VOLTAGE MONITOR (SVM).....	147
SECTION 18 – HARDWARE MULTIPLIER MODULE	148
18.1 – Hardware Multiplier Organization	148
18.2 – Hardware Multiplier Controls	148
18.3 – Register Output Selection	149
18.3.1 – Signed-Unsigned Operand Selection	149
18.3.2 – Operand Count Selection	149
18.4 – Hardware Multiplier Operations.....	149

18.4.1 – Accessing the Multiplier	149
18.5 – Hardware Multiplier Peripheral Registers.....	151
18.6 – Hardware Multiplier Examples.....	155
SECTION 19 – WATCHDOG TIMER	156
19.1 - Overview	156
19.2 – Watchdog Timer Description	156
19.2.1 – Watchdog Timer Interrupt Operation	157
19.2.2 – Watchdog Timer Reset Operation	157
19.2.3 – Watchdog Timer Applications.....	157
SECTION 20 – TEST ACCESS PORT (TAP).....	159
20.1 – TAP Controller	160
20.2 – TAP State Control.....	161
20.2.1 – Test-Logic-Reset.....	161
20.2.2 – Run-Test-Idle	161
20.2.3 – IR-Scan Sequence.....	161
20.2.4 – DR-Scan Sequence	162
20.3 – Communication via TAP	162
20.3.1 – TAP Communication Examples – IR-Scan and DR-Scan	163
SECTION 21 – IN-CIRCUIT DEBUG MODE	165
21.1 – Background Mode Operation	166
21.1.1 – Breakpoint Registers	167
21.1.2 – Using Breakpoints.....	169
21.2 – Debug Mode	170
21.2.1 – Debug Mode Commands.....	170
21.2.2 – Read Register Map Command Host-ROM Interaction	172
21.2.3 – Single Step Operation (Trace).....	173
21.2.4 – Return	174
21.2.5 – Debug Mode Special Considerations	174
21.3 – In-Circuit Debug Peripheral Registers.....	175
SECTION 22 – IN-SYSTEM PROGRAMMING	179
22.1 – Detailed Description	179
22.1.1 – Password Protection.....	180
22.1.2 – Entering JTAG Bootloader	180
22.1.3 – Entering I ² C Bootloader	181
22.1.4 – I ² C Bootloader Disable.....	181
22.2 – Bootloader Operation	182
22.2.1 – JTAG Bootloader Protocol	182
22.2.2 – I ² C Bootloader Protocol	183
22.3 – Bootloader Commands.....	184
22.3.1 – Command 00h – No Operation.....	184
22.3.2 – Command 01h – Exit Loader	184
22.3.3 – Command 02h – Master Erase.....	184

22.3.4 – Command 03h – Password Match.....	185
22.3.5 – Command 04h – Get Status	185
22.3.6 – Command 05h – Get Supported Commands	186
22.3.7 – Command 06h – Get Code Size.....	186
22.3.8 – Command 07h – Get Data Size.....	186
22.3.9 – Command 08h – Get Loader Version	186
22.3.10 – Command 09h – Get Utility ROM Version	186
22.3.11 – Command 10h – Load Code.....	187
22.3.12 – Command 11h – Load Data.....	187
22.3.13 – Command 20h – Dump Code	187
22.3.14 – Command 21h – Dump Data.....	188
22.3.15 – Command 30h – CRC Code.....	188
22.3.16 – Command 31h – CRC Data.....	188
22.3.17 – Command 40h – Verify Code	189
22.3.18 – Command 41h – Verify Data	189
22.3.19 – Command 50h – Load and Verify Code	189
22.3.20 – Command 51h – Load and Verify Data	189
22.3.21 – Command E0h – Code Page Erase	189
SECTION 23 – PROGRAMMING	190
23.1 – Addressing Modes.....	190
23.2 – Prefixing Operations.....	190
23.3 – Reading and Writing Registers.....	191
23.3.1 – Loading an 8-Bit Register with an Immediate Value.....	191
23.3.2 – Loading a 16-Bit Register with a 16-Bit Immediate Value	191
23.3.3 – Moving Values Between Registers of the Same Size	191
23.3.4 – Moving Values Between Registers of Different Sizes	191
23.4 – Reading and Writing Register Bits	192
23.5 – Using the Arithmetic and Logic Unit	193
23.5.1 – Selecting the Active Accumulator	193
23.5.2 – Enabling Auto-Increment and Auto-Decrement.....	193
23.5.3 – ALU Operations Using the Active Accumulator and a Source	195
23.5.4 – ALU Operations Using Only the Active Accumulator.....	195
23.5.5 – ALU Bit Operations Using Only the Active Accumulator	195
23.5.6 – Example: Adding Two 4-Byte Numbers Using Auto-Increment.....	195
23.6 – Processor Status Flag Operations	195
23.6.1 – Sign Flag.....	195
23.6.2 – Zero Flag.....	196
23.6.3 – Equals Flag	196
23.6.4 – Carry Flag	196
23.6.5 – Overflow Flag.....	197
23.7 – Controlling Program Flow.....	197
23.7.1 – Obtaining the Next Execution Address.....	197

23.7.2 – Unconditional Jumps	197
23.7.3 – Conditional Jumps	198
23.7.4 – Calling Subroutines.....	198
23.7.5 – Looping Operations.....	198
23.7.6 – Conditional Returns	199
23.8 – Handling Interrupts	199
23.8.1 – Conditional Return from Interrupt	200
23.9 – Accessing the Stack	200
23.10 – Accessing Data Memory	201
SECTION 24 – INSTRUCTION SET	203
SECTION 25 – UTILITY ROM	231
25.1 – Overview.....	231
25.2 – In-Application Programming Functions	232
25.2.1 – UROM_flashWrite	232
25.2.2 – UROM_flashErasePage	232
25.3 – Data Transfer Functions.....	233
25.3.1 – UROM_moveDP0	233
25.3.2 – UROM_moveDP0inc.....	233
25.3.3 – UROM_moveDP0dec	234
25.3.4 – UROM_moveDP1	234
25.3.5 – UROM_moveDP1inc.....	234
25.3.6 – UROM_moveDP1dec	235
25.3.7 – UROM_moveBP	235
25.3.8 – UROM_moveBPinc.....	235
25.3.9 – UROM_moveBPdec.....	236
25.3.10 – UROM_copyBuffer.....	236
25.4 Special Functions	237
25.4.1 – UROM_copyWord.....	237
25.4.2 – Software Reset	237
25.5 – Utility ROM Examples.....	238
25.5.1 – Reading Constant Word Data from Flash.....	238
25.5.2 – Reading Constant Byte Data from Flash (Indirect Function Call).....	238
SECTION 26 – MISCELLANEOUS	239
26.1 – Overview.....	239
26.2 – CRC8.....	239
26.2.1 – CRC Data In (CRC8IN).....	239
26.2.2 – CRC Data Out (CRC8OUT)	239
26.2.3 – Example	239
26.3 – Software Interrupts	239
26.3.1 – User Interrupt Register (USER_INT)	240
26.4 – General-Purpose Registers.....	240
26.4.1 – General-Purpose Register	240

26.5 – Device Number and I²C Bootloader Address Disable..... 240
 26.5.1 – Device Number Register (DEV_NUM)..... 240

SECTION 1 – OVERVIEW

The DS4830A optical microcontroller is a low-power, 16-bit microcontroller with a unique peripheral set supporting a wide variety of optical transceiver controller applications. It provides a complete optical control, calibration, and monitor solution. The DS4830A is based on the high-performance, 16-bit, reduced instruction set computing (RISC) architecture with on-chip flash program memory and SRAM data memory.

The resources and features that the DS4830A provides for monitoring and controlling an optical system include the following:

- ❖ 16-Bit Low-Power Microcontroller
- ❖ 400kHz I²C-Compatible Slave Communication Interface
 - Four User-Programmable Slave Addresses
 - 8-Byte Transmit Page for Each Slave Address
 - 8-Byte Receive Page Shared Between All Slave Addresses
- ❖ 32KWords Flash Program Memory
- ❖ 2KWords Data RAM
- ❖ 32-Level Hardware Stack
- ❖ 13-Bit ADC with a 26 Input Mux
 - 16 Single or 8 Differential Mode ADC Channels
 - Four User-Selectable Gains for Individual Channel
 - V_{DD}, Internal Reference, and DAC External References Measurement
 - ADC Samples Averaging Options
- ❖ 10 PWM Channels
 - Pulse Spreading Using Delta-Sigma Algorithm
 - PWM Output Synchronization
 - User-Selectable 7- to 16-Bit Resolution
 - 1MHz Switching Using 133MHz External Clock
- ❖ 10-Bit Fast Comparator with 16 Input Mux
 - Single and Differential Mode
 - Low and High Threshold Configurations
 - 3.2μs Conversion Time per Channel
- ❖ Two Independent Sample and Hold (S/H)
 - Single, Fast, and Dual Mode Operation
 - Internal and External Trigger Option
 - Pin Discharge
 - S/H Samples Averaging Options
- ❖ Fast Internal Die Temperature Sensors with Averaging Option
- ❖ 12-Bit, 8 Voltage DAC Channels Selectable Internal or External Reference Option
- ❖ Serial Interfaces
 - SPI Master and Slave Interface
 - 400kHz I²C-Compatible Master with Alternate Location Option
 - 3-Wire Master Interface
- ❖ Dual Hardware Multiplier Unit
- ❖ Two 16-Bit Timers with Synchronous and Compare Modes
- ❖ Watchdog Timer
- ❖ Maskable Interrupt Sources
- ❖ Brownout Monitor
- ❖ 31 GPIO pins
- ❖ Supply Voltage Monitoring
- ❖ Internal 20MHz Oscillator, CPU Core Frequency 10MHz
- ❖ Included ROM Routines that allow Bootloading and In-Application Programming of Flash Memory
- ❖ In-System Debugging
- ❖ Four Software Interrupts
- ❖ Fast Hardware CRC-8 for Packet Error Checking (PEC)

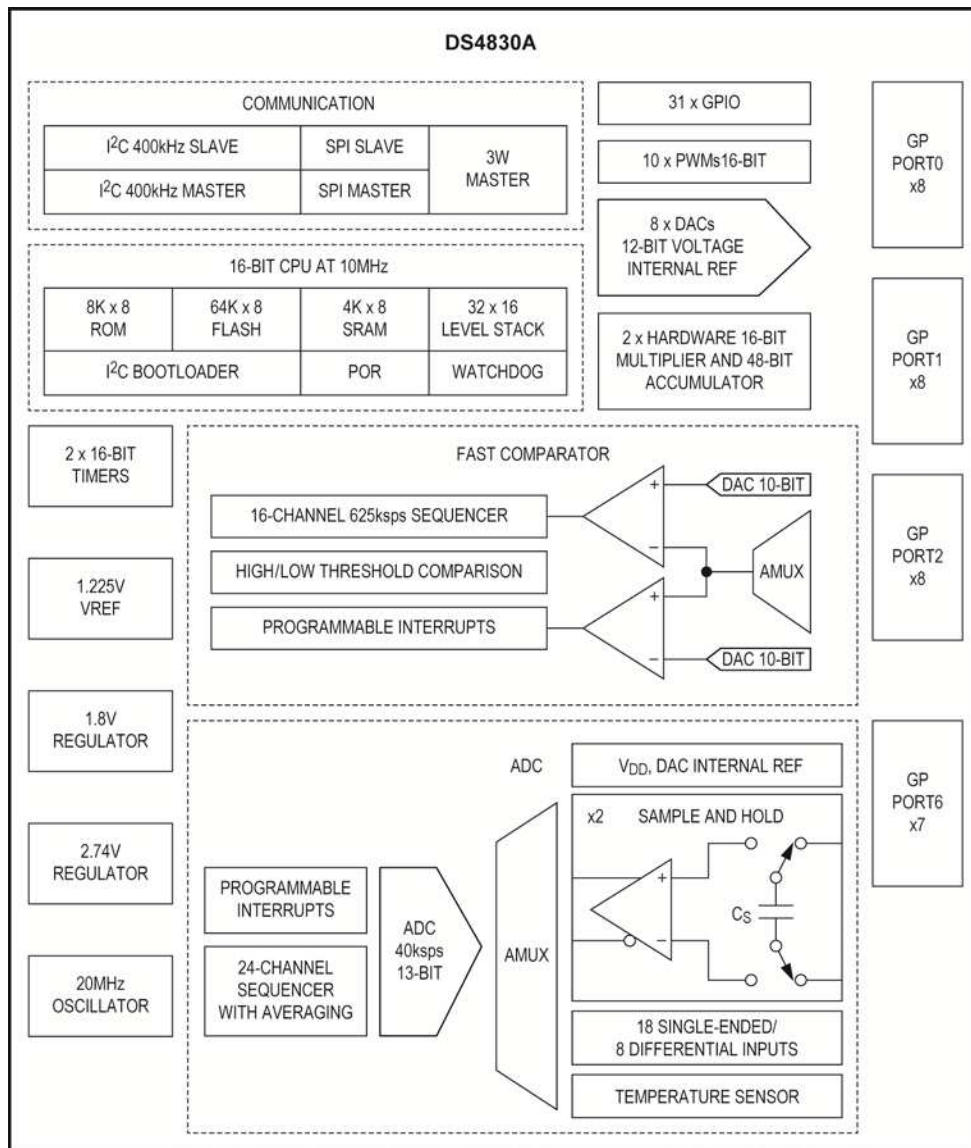


Figure 1-1: DS4830A Block Diagram

This document is provided as a supplement to the DS4830A IC data sheet. This user's guide provides the information necessary to develop applications using the DS4830A. All electrical and timing specifications, pin descriptions, package information, and ordering information can be found in the DS4830A IC data sheet.

SECTION 2 – ARCHITECTURE

The DS4830A contains a low-cost, high-performance microcontroller with flash memory. It is structured on a highly advanced, 16-accumulator-based, 16-bit RISC architecture. Fetch and execution operations are completed in one cycle without pipelining, since the instruction contains both the opcode and data. The highly efficient core is supported by 16 accumulators and a 32-level hardware stack, enabling fast subroutine calling and task switching.

Data can be quickly and efficiently manipulated with three internal data pointers. Two of these data pointers, DP0 and DP1, are stand-alone 16-bit pointers. The third data pointer, Frame Pointer, is composed of a 16-bit base pointer (BP) and an 8-bit offset register (OFFS). All three pointers support post-increment/decrement functionality for read operations and pre-increment/decrement for write operations. For the Frame Pointer (FP=BP[OFFS]), the increment/decrement operation is executed on the OFFS register and does not affect the base pointer. Multiple data pointers allow more than one function to access data memory without having to save and restore data pointers each time.

Stack functionality is provided by dedicated memory with a 16-bit width and a depth of 32. An on-chip memory management unit (MMU) allows logical remapping of the program and data spaces, and thus facilitates in-system programming and fast access to data tables, arrays, and constants located in flash memory.

This section provides details on the following topics.

1. Instruction decoding
2. Register space
3. Memory types
4. Program and data memory mapping and access
5. Data alignment
6. Reset conditions
7. Clock generation

2.1 – Instruction Decoding

The DS4830A uses the standard 16-bit core instruction set, which is described in the Instruction Set section. Every instruction is encoded as a single 16-bit word. The instruction word format is shown in Figure 2-1.

FORMAT	DESTINATION							SOURCE							
f	d	d	d	d	d	d	d	s	s	s	s	s	s	s	s

Figure 2-1: Instruction Word Format

- Bit 15 (f) indicates the format for the source field of the instruction as follows:
 - If f equals 0, the instruction is an immediate source instruction. The source field represents an immediate 8-bit value.
 - If f equals 1, the instruction is a register source instruction. The source field represents the register that the source value will be read from.
- Bits 14 to 8 (ddddddd) represent the destination for the transfer. This value always represents a destination register. The lower four bits contain the module specifier and the upper three bits contain the register index in that module.
- Bits 7 to 0 (sssssss) represent the source for the transfer. Depending on the value of the format field, this can either be an immediate value or a source register. If this field represents a register, the lower four bits contain the module specifier and the upper four bits contain the register index in that module.

This instruction word format presents the following limitations.

1. There are 32 registers per register module, but only four bits are allocated to designate the source register and only three bits are allocated to designate the destination register.
2. The source field only provides 8 bits of data for an immediate value; however a 16-bit immediate value may be required.

The DS4830A uses a prefix register (PFX) to address these limitations. The prefix register provides the additional bits required to access all 32 register within a module. The prefix register also provides the additional 8 bits of data required to make a 16-bit immediate data source. The data that is written to the prefix register survives for only one clock cycle. This means the write to the prefix register must occur immediately prior to the instruction requiring the prefix register. The prefix register is cleared to zero after one cycle so it will not affect any other instructions. The write to the prefix register is

done automatically by the assembler and requires one additional execution cycle. So, while most instructions execute in a single cycle, two cycles are needed for instructions that require the prefix register.

The architecture of the DS4830A is transport-triggered. This means that writing to or reading from certain register locations will also cause side effects to occur. These side effects form the basis of the DS4830A's higher level opcodes, such as ADDC, OR, and JUMP. While these opcodes are actually implemented as MOVE instructions between certain register locations, the encoding is handled by the assembler and need not be a concern to the programmer. The unused "empty" locations in the System Register Modules are used for these higher level opcodes.

The instruction set is designed to be highly orthogonal. All arithmetic and logical operations that use two registers can use any register along with the accumulator. Data can be transferred between any two registers in a single instruction.

2.2 – Register Space

The DS4830A provides a total of 13 register modules broken up into two different groups. These groupings are descriptive only, as there is no difference between accessing the two register groups from a programming perspective.

The two groups are:

1. **System Registers:** These are modules 8h, 9h, and Bh through Fh. The System Registers in the DS4830A are used to implement higher level opcodes as well as the following common system features.
 - 16-bit ALU and associated status flags (zero, equals, carry, sign, overflow)
 - 16 working accumulator registers, each 16-bit, along with associated control registers
 - Instruction pointer
 - Registers for interrupt control, handling, and identification
 - Auto-decrementing Loop Counters for fast, compact looping
 - Two Data Pointer registers and a Frame Pointer for data memory access
2. **Peripheral Registers:** These are the lower six modules (Modules 0h through 5h). The Peripheral Registers in the DS4830A are used for functionalities such as ADC, Fast Comparator, DAC, PWM Outputs, Timers, Sample and Hold, 3-Wire, I²C Master and Slave, SPI Master and Slave, 31-GPIO pins, etc. The Peripheral Registers are not used to implement opcodes.

Each System Register module has 16 registers, while each Peripheral Register module has 32 registers. The number of cycles required to access a particular register depends upon the register's index within the module. The access times based upon the register index are grouped as follows:

- The first eight registers (index 0h to 7h) in each module may be read from or written to in a single cycle
- The second eight registers (index 8h to 0Fh) may be read from in a single cycle and written to in two cycles (by using the prefix register PFX).
- The last sixteen registers (10h to 1Fh) in Peripheral Register modules may be read or written in two cycles (always requiring use of the prefix register PFX).

Registers may be 8 or 16 bits in length. Some registers may contain reserved bits. The user should not write to any reserved bits. Data transfers between registers of different sizes are handled as shown in Table 2-1.

- If the source and destination registers are both 8 bits wide, data is copied bit to bit.
- If the source register is 8 bits wide and the destination register is 16 bits wide, the data from the source register is transferred into the lower 8 bits of the destination register. The upper 8 bits of the destination register are set to the current value of the prefix register; this value is normally zero, but it can be set to a different value by the previous instruction if needed. The prefix register reverts back to zero after one cycle, so this must be done by the instruction immediately before the one that will be using the value.
- If the source register is 16 bits wide and the destination register is 8 bits wide, the lower 8 bits of the source are transferred to the destination register.
- If both registers are 16 bits wide, data is copied bit to bit.

The above rules apply to all data movements between defined registers. Data transfer to/from undefined register locations has the following behavior:

- If the destination is an undefined register, the MOVE is a dummy operation but may trigger an underlying operation according to the source register (e.g., @DPn--).
- If the destination is a defined register and the source is undefined, the source data for the transfer will depend upon the source module width. If the source is from a module containing 8-bit or 8-bit and 16-bit source registers,

the source data will be equal to the prefix data as the upper 8 bits and 00h as the lower 8 bits. If the source is from a module containing only 16-bit source registers, 0000h source data is used for the transfer.

Table 2-1. Register-to-Register Transfer Operations

SOURCE REGISTER SIZE (BITS)	DESTINATION REGISTER SIZE (BITS)	PREFIX SET?	DESTINATION SET TO VALUE	
			HIGH 8 BITS	LOW 8 BITS
8	8	X	—	Source [7:0]
8	16	No	00h	Source [7:0]
8	16	Yes	PFX [7:0]	Source [7:0]
16	8	X	—	Source [7:0]
16	16	X	Source [15:8]	Source [7:0]

2.3 – Memory Types

In addition to the internal register space, the DS4830A incorporates the following memory types:

- 32KWords of flash memory
- 4KWords of utility ROM contain a debugger and program loader
- 2KWords of SRAM
- 32-level hardware stack for storage of program return addresses

The memory on the DS4830A is organized according to Harvard architecture. This means that there are separate busses for both program and data memory. Stack memory is also separate and is accessed through a dedicated register set.

2.3.1 – Flash Memory

The DS4830A contains 32KWords (32K x 16) of flash memory. The flash memory begins at address 0000h and is contiguous through word address 7FFFh. The flash memory can also be used for storing lookup tables and other non-volatile data.

The incorporation of flash memory allows the contents of the flash memory to be upgraded in the field, either by the application or by one of the bootloaders (JTAG or I²C). Writing to flash memory must be done indirectly by using routines that are provided by the utility ROM. See the Utility ROM and In-System Programming sections for more details.

2.3.2 – SRAM Memory

The DS4830A contains 2KWords (2K x 16) of SRAM memory. The SRAM memory address begins at address 0000h and is contiguous through word address 07FFh. The contents of the SRAM are indeterminate after power-on reset, but are maintained during non-POR resets.

When using the in-circuit debugging features, the highest 19 bytes of the SRAM must be reserved for saved state storage and working space for the debugging routines. If in-circuit debug is not used, the entire 2KWords of SRAM is available for application use.

2.3.3 – Utility ROM

The utility ROM is a 4kWord segment of memory. The utility ROM memory address begins at word address 8000h and is contiguous through word address 8FFFh. The utility ROM is programmed at the factory and cannot be modified. The utility ROM provides the following system utility functions:

- Reset vector (not user code reset vector)
- In-system programming (bootstrap loader) over JTAG or I²C-compatible interfaces
- In-circuit debug routines
- Routines for in-application flash programming

Following any reset, the DS4830A automatically starts execution at the Reset Vector which is address 8000h in the utility ROM. The ROM code determines whether the program execution should immediately jump to the start of application code (flash address 0000h), or to one of the special routines mentioned. Routines within the utility ROM are firmware-accessible and can be called as subroutines by the application software. See the Utility ROM, In-System Programming, and In-Circuit Debug sections for more information on the routines provided by the utility ROM.

2.3.4 – Stack Memory

A 16-bit, 32-level on-chip stack provides storage for program return addresses and temporary storage of system registers. The stack is used automatically by the processor when the CALL, RET, and RETI instructions are executed, and when an interrupt is serviced. The stack can also be used explicitly to store and retrieve data by using the @SP- - source, @++SP destination, or the PUSH, POP, and POPI instructions. The POPI instruction acts identically to the POP instruction except that it additionally clears the INS bit.

The width of the stack is 16 bits to accommodate the instruction pointer size. On reset, the stack pointer SP initializes to the top of the stack (1Fh). The CALL, PUSH, and interrupt vectoring operations first increment SP and then store a value at @SP. The RET, RETI, POP, and POPI operations first retrieve the value at @SP and then decrement SP.

The stack memory is initialized to indeterminate values upon reset or power-up. Stack memory is dedicated for stack operations only and cannot be accessed by the DS4830A program or data busses.

When using the in-circuit debugging features, one word of the stack must be reserved for the debugging routines. If in-circuit debug is not used, the entire stack is available for application use.

2.4 – Program and Data Memory Mapping and Access

The memory on the DS4830A is implemented using Harvard architecture, with separate busses for program and data memory. The Memory Management Unit (MMU) allows the DS4830A to also support a pseudo-Von Neumann memory map. The pseudo Von Neumann memory map allows each of the memory segments (flash, SRAM, and utility ROM) to be logically mapped into a single contiguous memory map. This allows all of the memory segments to be accessed as both program and memory data. The pseudo-Von Neumann memory map provides the following advantages:

- Program execution can occur from the flash, SRAM, or utility ROM memory segments.
- The SRAM and flash memory segments can both be used for data memory.

Using the pseudo-Von Neumann memory map does have one restriction. This restriction is that a particular memory segment cannot be simultaneously accessed as both program and data memory.

2.4.1 – Program Memory Access

The instructions that the DS4830A is executing reside in what is defined as the program memory. The MMU fetches the instructions using the program bus. The Instruction Pointer (IP) register designates the program memory address of the next instruction to fetch. The Instruction Pointer is read/write accessible by the user software. A write to the Instruction Pointer will force program flow to the new address on the next cycle following the write. The content of the Instruction Pointer will be incremented by 1 automatically after each fetch operation. From an implementation perspective, system interrupts and branching instructions simply change the contents of the Instruction Pointer and force the opcode to fetch from a new program location.

2.4.2 – Program Memory Mapping

The DS4830A's mapping of the three memory segments (flash, SRAM, and utility ROM) as program memory is shown in Figure 2-2. The mapping of memory segments into program space is always the same. When referring to memory as program memory, all addresses are given as word addresses. The 32KWord flash memory segment is located at memory location 0000h through 7FFFh and is logically divided into two pages, each containing 16KWords. The utility ROM is located from location 8000h through 8FFFh, followed by the SRAM memory segment at location A000h through A7FFh. The user code reset vector, which is the first instruction of user program code that is executed, is located at flash memory address 0000h. User program code should always begin at this address.

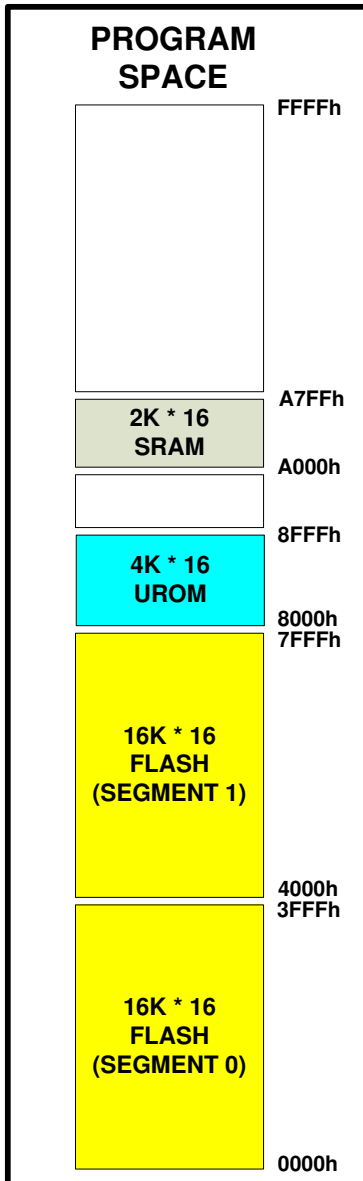


Figure 2-2: Program Memory Mapping

2.4.3 – Data Memory Access

Data memory mapping and access control are handled by the memory management unit (MMU). Read/write access to data memory can be in word or in byte mode. The DS4830A provides three pointers that can be used for indirect accessing of data memory. The DS4830A has two data pointers (@DPn) and one frame pointer (@BP[OFFS]). These pointers are implemented as registers that can be directly accessed by user software. A data memory access requires only one system clock period.

2.4.3.1 – Data Pointers

To access data memory, the data pointers are used as one of the operands in a MOVE instruction. If the data pointer is used as a source, the core performs a load operation that reads data from the memory location addressed by the data

pointer. If the data pointer is used as destination, the core performs a store operation that writes data to the memory location addressed by the data pointer. Following are some examples of setting and using a data pointer.

```

move DP[0], #0100h      ; set pointer DP[0] to address 100h
move Acc, @DP[0]        ; read data from location 100h
move @DP[0], Acc        ; write to location 100h

```

The address pointed to by the data pointers can be automatically incremented or decremented. If the data pointer is used as a source, the pointer can be incremented or decremented after the data access. If the data pointer is used as a destination, the increment or decrement can occur prior to the data access. Following are examples of using the data pointers increment/decrement features.

```

move Acc, @DP[0]++      ; increment DP[0] after read
move Acc, @DP[1]--      ; decrement DP[1] after read
move @++DP[0], Acc      ; increment DP[0] before write
move @--DP[1], Acc      ; decrement DP[1] before write

```

2.4.3.2 – Frame Pointer

The frame pointer (BP[OFFS]) is formed by the 16-bit unsigned addition of the 16-bit Frame Pointer Base Register (BP) and the 8-bit Frame Pointer Offset Register (OFFS). The method the DS4830A uses to access data using the frame pointer is similar to the data pointers. When increments or decrements are used, only the value of OFFS is incremented or decremented. The base pointer (BP) will remain unaffected by increments or decrements of the OFFS register, including when the OFFS register rolls over from FFh to 00h or from 00h to FFh. Following are examples of how to use the frame pointer.

```

move BP, #0100h          ; set base pointer to address 100h
move OFFS, #10h          ; set the offset to 10h,
move Acc, @BP[OFFS]      ; read data from location 0110h
move @BP[OFFS], Acc      ; write data to location 0110h
move Acc, @BP[OFFS++]    ; increment OFFS after read
move Acc, @BP[OFFS++]    ; decrement OFFS after read
move @BP[++OFFS], Acc    ; increment OFFS before write
move @BP[--OFFS], Acc    ; decrement OFFS before write

```

2.4.4 – Data Memory Mapping

The DS4830A's pseudo-Von Neumann memory map allows the MMU to read data from each of the three memory segments (flash, SRAM, utility ROM). The MMU can also write data directly to the SRAM memory segment. Data memory can be written to the flash memory segment, but because writing to flash requires the use of the utility ROM routines, this is not a direct access. The logical mapping of the three memory segments as data memory varies depending upon:

- which memory segment instructions are currently being executed from
- if data memory is being accessed in word or byte mode

In all cases, whichever memory segment is currently being used, program memory cannot be accessed as data memory.

When the program is currently executing instructions from either the SRAM or utility ROM memory segments, the flash memory will be mapped to half of the data memory space. If word access mode is selected, both pages (32KWords) can be logically mapped to data memory space. If byte access mode is selected, only one page (32KBytes) can be logically mapped to half of the data memory space. When operating in byte access mode, the selection of which flash page is mapped into data memory space is determined by the Code Data Access bit (CDA0):

CDA0	Selected Page in Byte Mode	Selected Page in Word Mode
0	P0	P0 and P1
1	P1	P0 and P1

The next three sections detail the mapping of the different memory segments as data memory depending upon which memory segment instructions are currently being executed from.

2.4.4.1 – Memory Map When Executing from Flash Memory

When executing from the flash memory:

- Read and write operations of SRAM memory are executed normally.
- The utility ROM can be read as data, starting at 8000h of the data space. The utility ROM cannot be written.

Figure 2-3 illustrates the mapping of the SRAM and utility ROM memory segments into data memory space when code is executing from the flash memory segment.

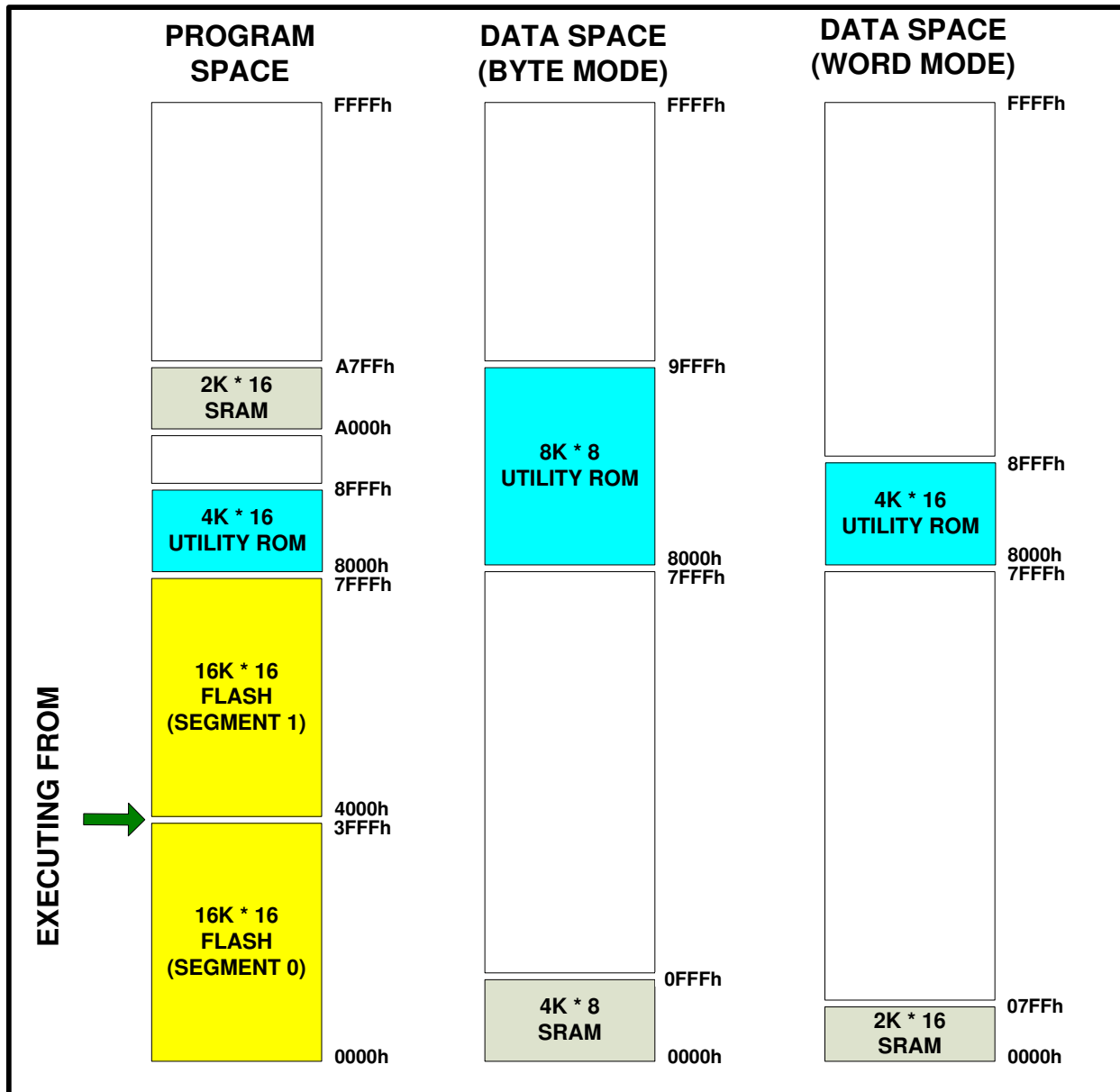


Figure 2-3: Memory Map When Executing from Flash Memory

2.4.4.2 – Memory Map When Executing from Utility ROM

When executing from the utility ROM:

- Read and write operations of SRAM memory are executed normally.
- Reading of flash memory is executed normally. Writing to flash memory requires the use of the utility ROM routines.
- One page (byte access mode) or both pages (word access mode) of the flash memory can be accessed as data with an offset of 8000h as determined by the CDA0 bit.

Figure 2-4 illustrates the mapping of the SRAM and flash memory segments into data memory space when code is executing from the utility ROM memory segment.

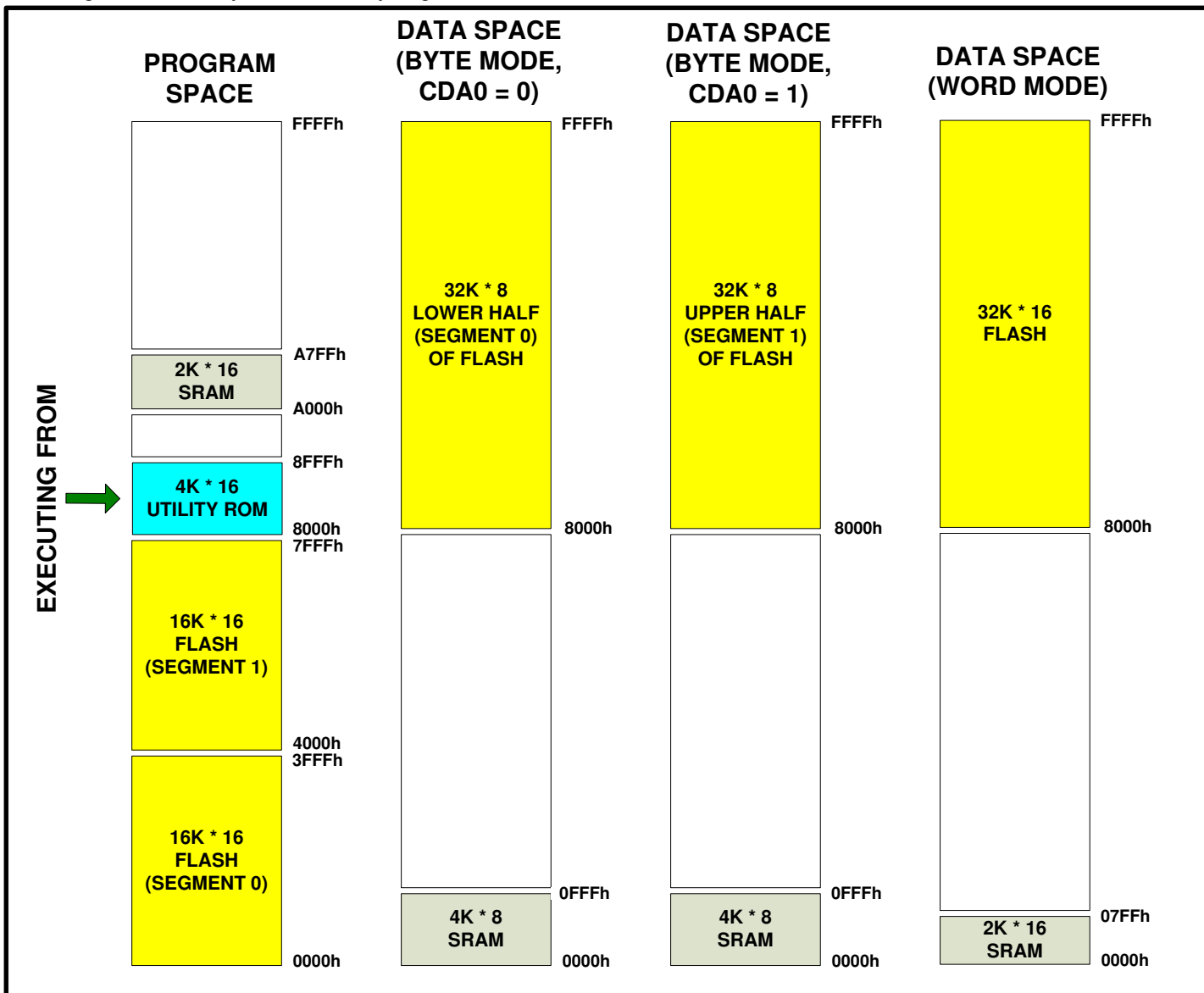


Figure 2-4: Memory Map When Executing from Utility ROM

2.4.4.3 – Memory Map When Executing from SRAM

When executing from the SRAM:

- The utility ROM can be read as data, starting at 8000h of the data space. The utility ROM cannot be written.
- Reading of flash memory is executed normally. Writing to flash memory requires the use of the utility ROM routines.
- One page (byte access mode) or both pages (word access mode) of the flash memory can be accessed as data with an offset of 0000h. For byte access mode, the page of flash accessed is determined by the CDA0 bit.

Figure 2-5 illustrates the mapping of the flash and utility ROM memory segments into data memory space when code is executing from the SRAM memory segment.

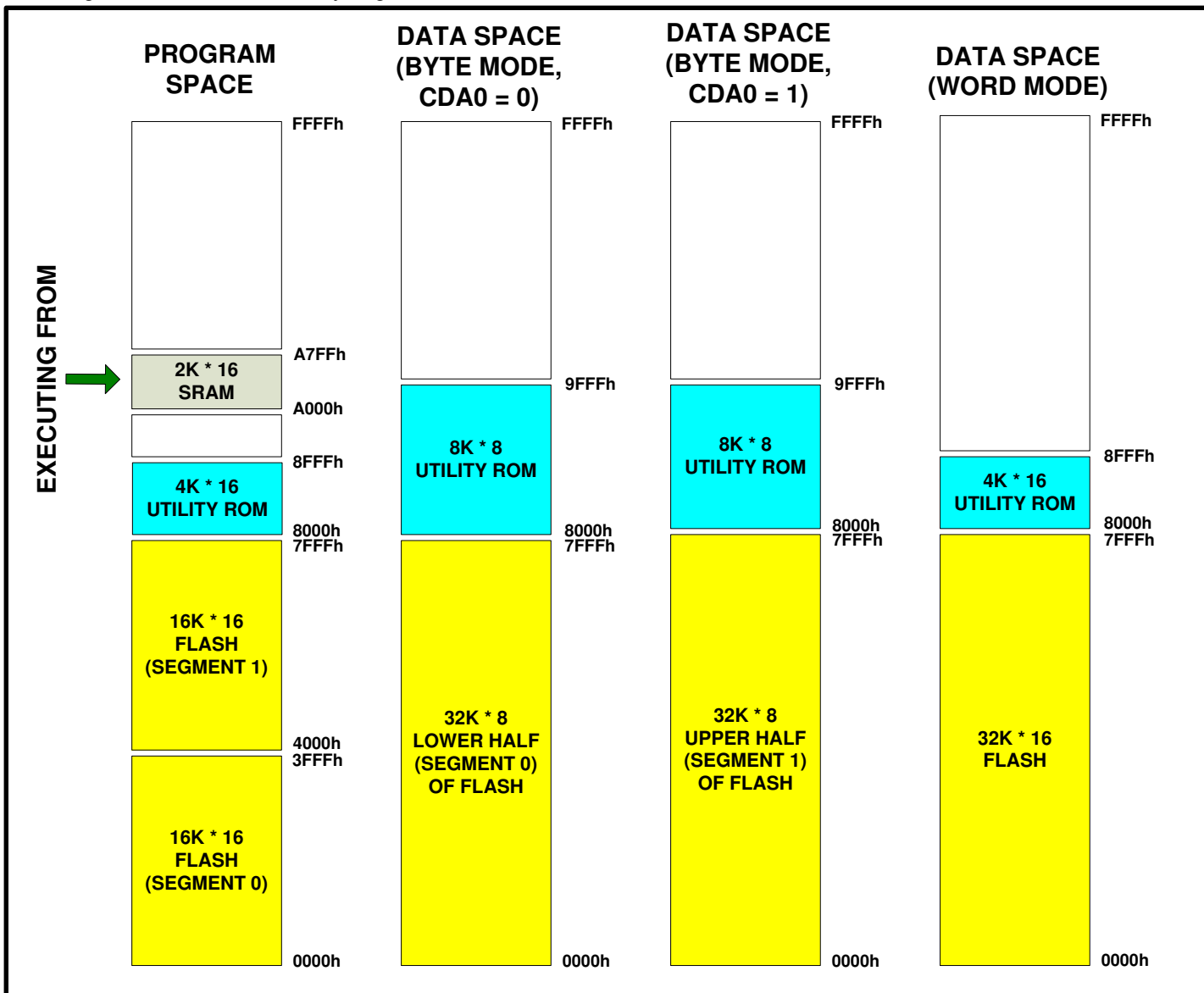


Figure 2-5: Memory Map When Executing from SRAM

2.5 – Data Alignment

To support merged program and data memory operation while maintaining efficient memory space usage, the data memory must be able to support both byte and word mode accessing. Data is aligned in data memory as words, but the effective data address is resolved to bytes. This data alignment allows program instruction fetching in words while maintaining data accessibility at the byte level. It is important to realize that this accessibility requires strict word alignment. All executable or data words must align to an even address in byte mode. Care must be taken when updating a code segment as misalignment of words will likely result in loss of program execution control.

Memory will always be read as a complete word, whether for program fetch or data access. The program decoder always uses a full 16-bit word. The data access can utilize a word or an individual byte. Data memory is organized as two byte-wide memory banks with common word address decode but two 8-bit data buses. In byte mode, data pointer hardware reads out the full word containing the selected byte using the effective data word address pointer (the least significant bit of the byte data pointer is not initially used). Then, the least significant data pointer bit functions as the byte select that is used to place the correct byte on the data bus. For write access, data pointer hardware addresses a particular word using the effective data word address while the least significant bit selects the corresponding data bank for write. The contents of the other byte are left unaffected.

2.6 – Reset Conditions

The DS4830A has several possible sources of reset.

- Power-On/Brownout Reset
- Watchdog Timer Reset
- External Reset
- Internal System Reset
- Soft Reset

Once a reset condition has completed or been removed, code execution begins at the beginning of utility ROM, which is address 8000h. The utility ROM code interrogates the I2C_SPE, JTAG_SPE, and PWL bits to determine if bootloading is necessary. If bootloading is not required, execution will jump to the user code reset vector, which is at flash memory address 0000h.

The $\overline{\text{RST}}$ pin is an input only.

2.6.1 – Power-On/Brownout Reset

The DS4830A provides a power-on reset (POR) circuit to ensure proper initialization of internal device states and analog circuits. The POR voltage threshold range is between approximately 1.1V and 1.7V. When V_{DD} is below the POR level, the state of all the DS4830A pins (except DAC port pins), including $\overline{\text{RST}}$, is weak pullup. The port pins having DAC function are high impedance on POR.

The DS4830A also includes brownout detection capability. This is an on-chip precision reference and comparator that monitors the supply voltage, V_{DD} , to ensure that it is within acceptable limits. If V_{DD} is below the brownout level (V_{BO}), the power monitor generates a reset. This can occur when:

- The DS4830A is being powered up and V_{DD} is above the POR level but still less than V_{BO} .
- V_{DD} drops from an acceptable level to less than V_{BO} .

Once V_{DD} exceeds V_{BO} , the DS4830A exits the reset condition and the internal oscillator starts up. After approximately 1ms the DS4830A performs the following tasks.

- All registers and circuits enter their reset state
- The POR flag in the Watchdog Control Register is set to indicate the source of the reset
- The DS4830A begins normal operation (CPU State)
- Code execution begins at utility ROM location 8000h

The transition between POR, Brownout, and normal operation is detailed in Figure 2-6: DS4830A State Diagram.

Note: If V_{DD} is below V_{BO} , there is a chance that the SRAM gets corrupted. If the POR flag in WDCN is set, all data in SRAM should be re-initialized.

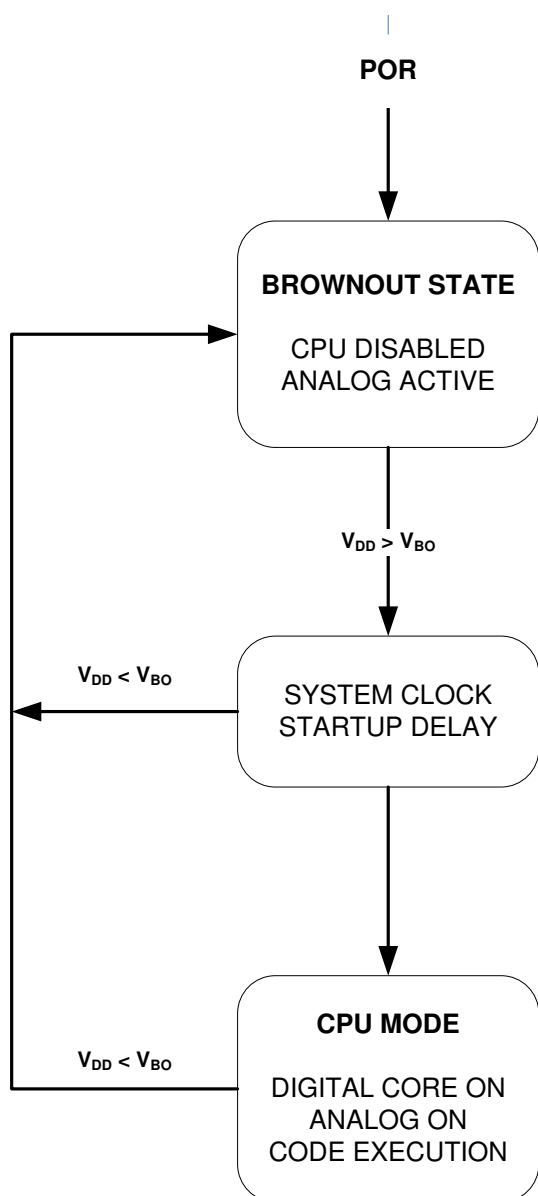


Figure 2-6: DS4830A State Diagram

2.6.2 – Watchdog Timer Reset

The watchdog timer is a programmable hardware timer that can be used to reset the processor in case a software lockup or other unrecoverable error occurs. Once the watchdog is enabled, software must reset the watchdog timer periodically. If the processor does not reset the watchdog timer before it elapses, the watchdog can initiate a reset.

If the watchdog resets the processor, the DS4830A will remain in reset for 12 clock cycles. When a reset occurs due to a watchdog timeout, the Watchdog Timer Reset Flag (WTRF) in the WDCN register is set to indicate the source of the reset.

2.6.3 – External Reset

During normal operation, the DS4830A is placed into external reset when the $\overline{\text{RST}}$ pin is held at logic 0 for at least four clock cycles. Once the DS4830A enters reset mode, it remains in reset as long as the $\overline{\text{RST}}$ pin is held at logic 0. After the $\overline{\text{RST}}$ pin returns to logic 1, the processor exits reset within 12 clock cycles.

An external reset pulse on the $\overline{\text{RST}}$ pin will reset the DS4830A and return to normal CPU mode operation within 10 clock cycles.

2.6.4 – Internal System Resets

There are two possible sources of internal system resets. An internal reset will hold the DS4830A in reset mode for 12 clock cycles.

1. When data BBh is written to the special I²C slave address 34h.
2. When in-system programming is complete and the ROD bit is set to 1.

2.6.5 – Software Reset

The device UROM provides option to soft reset through the application program. The application program jumps to UROM code which generates the internal system reset. UROM location 8854h has code when executed generates internal reset. Application program can jump to this location to generate software reset.

asm (“LJUMP #8854h”)

2.7 – Clock Generation

The DS4830A generates its 20MHz peripheral clock using an internal oscillator and generates 10MHz instruction clock using divide by 2 circuit. This oscillator starts up when V_{DD} exceeds the brownout voltage level, V_{BO} . There is a delay of approximately 1ms in the oscillator start up and beginning of clock. This delay ensures that the clock is stable prior to beginning normal operation.

SECTION 3 – SYSTEM REGISTER DESCRIPTIONS

Most functions of the DS4830A are controlled by sets of registers. These registers provide a working space for memory operations as well as configuring and addressing peripheral registers on the device. Registers are divided into two major types: system registers and peripheral registers. The common register set, also known as the system registers, includes ALU access and control registers, accumulator registers, data pointers, interrupt vectors and control, and stack pointer. The peripheral registers define additional functionality and the functionality is broken up into discrete modules.

This section describes the DS4830A's system registers. Table 3-1 shows the DS4830A system register map. Table 3-2 explains system register bit functions. This is followed by a detailed bit description.

Table 3-1: System Register Map

REGISTER INDEX	REGISTER MODULE						
	AP (08h)	A (09h)	PFX (0Bh)	IP (0Ch)	SP (0Dh)	DPC (0Eh)	DP (0Fh)
00h	AP	A[0]	PFX[0]	IP			
01h	APC	A[1]	PFX[1]		SP		
02h		A[2]	PFX[2]		IV		
03h		A[3]	PFX[3]			OFFS	DP[0]
04h	PSF	A[4]	PFX[4]			DPC	
05h	IC	A[5]	PFX[5]			GR	
06h	IMR	A[6]	PFX[6]		LC[0]	GRL	
07h		A[7]	PFX[7]		LC[1]	BP	DP[1]
08h	SC	A[8]				GRS	
09h		A[9]				GRH	
0Ah		A[10]				GRXL	
0Bh	IIR	A[11]				FP	
0Ch		A[12]					
0Dh		A[13]					
0Eh		A[14]					
0Fh	WDCN	A[15]					