

JULY 1994

APPLICATION NOTE

LH28F008SA Software Drivers

SHARP -

	LH28F008SA	CONTENTS PAGE
	Software Drivers	1.0 INTRODUCTION 1
	an an the second for a second s	2.0 ASM-86 ASSEMBLY DRIVERS 2
	an a	3.0 "C" DRIVERS
	, ye allar Toloramileo (Collare). A socio terro (Collare) A	an an an State state and state Name and state and sta
·		
		- -

1.0 INTRODUCTION

This application note provides example software code for byte writing, block erasing and otherwise controlling SHARP's LH28F008SA 8 Mbit symmetrically blocked Flash Memory family. Two programming languages are provided; high-level "C" for multi-platform support, and ASM-86 assembly. In many cases, the driver routines can be inserted "as is" into the main body of code being developed by the system software engineer. The text accompanying each routine describes the existing code and suggests areas for possible alteration to fit specific applications. These explanations, along with in-line commenting, minimize driver modification efforts.

The product datasheet for the LH28F008SA is a valuable reference document. This datasheet should be reviewed in conjunction with this application note for a complete understanding of the devices. "LH28F008SA Hardware Interfacing" is the hardware-oriented application note equivalent for these devices and can also be referenced.

The internal automation of the LH28F008SA makes software timing loops unnecessary and results in platformindependent code. This software is designed to be executed in any type of memory and with all processor clock rates. "C" code can be used with many microprocessors and microcontrollers.

1

2.0 ASM-86 ASSEMBLY DRIVERS

;	The follow	ing code controls	s byte write of	data to a single LH28F008SA (x8 writ	te)	
;	DS:(SI) po	ints to the data to	o be written, E	S:[DI] is the location to be written		
;	In protecte	ed mode operatio	n, DS and ES	reference a descriptor		
;	Register A	X is modified by	this procedure			
WRITE_SETU	JP	EQU	40H	-		
READ_ID		EQU	90H			
MANUFACTU	IRER_ID	EQU	89H			
DEVICE_ID		EQU ~	0A2H	•		
DEVICE_ID2		EQU	0A1H			
READY		EQU	80H			¥.
W_ERR_FLA	G	EQU	10H		-	
VPP_FLAG		EQU	08H			
;	Insert cod	e here to ramp V	pp and disable	component /PWD input. If a string of	of bytes is	
		to be written	at one time, V	pp ramp to 12V and ID cheek need o	nly occur o	nce,
;		before the fin	st byte is writte	n.		
	MOV	AX,	"Address 0 1	or target LH28F008SA-segment"	;	Initialize pointer to LH28F008SA address 0
	MOV	ES,	AX	•		
	MOV	DI.	*Address 0 1	or target LH28F008SA-offset"		•
	MOV	BYTE PTR E		READ_ID	;	Write Intelligent Identifier command
	CMP	BYTE PTR E	• •	MANUFACTURER_ID	:	Does manufacturer ID read correctly?
	JNZ	W_BYT_ID_I	• •	·····		•
	MOV	DI.		or target LH28F008SA-offset*	:	Initialize pointer to LH28F008SA address 1
	CMP	BYTE PTR E		DEVICE_ID	:	Does device ID read correctly?
	JZ	W_BYT_ID_I	• •		. '	
	CMP	BYTE PTR E		DEVICE_ID2		
	JNZ	W_BYT_ID_I				•
W_BYT_ID_P	ASS:					
	MOV	AX.	"Byte write o	lestination address-segment"	;	Initialize pointer to byte write dest. address
	MOV	ES.	AX	-	,	······································
	MOV	DI.		lestination address-offset"		
	MOV	BYTE PTR E		WRITE_SETUP -		Write byte write setup command
	MOV	AL,	DS:[SI]		:	Load AL with data to write
	MOV	ES:[DI],	AL			Write to device
W_BYT_LOO		20.(01),	75		,	
011_2000	TEST	BYTE PTR E	S-101	READY		Read LH28F008SA Status Register
	JZ		• •	READT	,	-
	JZ	W_BYT_LOO	JF			Loop until bit 7 = 1
	TEST	BYTE DTO E	S-101		Ň	
	JZ	BYTE PTR E		(W_ERR_FLAG OR VPP FLAG)		Successi
	JZ	W_BYT_CO	N 1		;	Successi
	TEST		0.00			Oback Status Depistes bit 4
		BYTE PTR E	• •	W_ERR_FLAG	;	Check Status Register bit 4
	JNZ	W_BYT_ERF	٦		i	Jump if = 1, Byte Write Error
	TEOT	Ee.mu				Ohadi Status Basistas bit 2
	TEST	ES:[DI],	VPP_FLAG		;	Check Status Register bit 3
	JNZ	W_BYT_VPF	•		;	Jump if = 1, Vpp Low Error
W_BYT_ID_E				and a marked and		
•		e to service impre	•			
W DVT FFF		USSA /PWD Inpu	IT DISADIED? IS	Vcc applied to the LH28F008SA?		
W_BYT_ERR:						-
:		e to service byte	write error her	e		-
W_BYT_VPP:						
;		e to service byte	write Vpp low	error here		
W_BYT_CON						
;	Code cont	inues from this p	oint			

This routine writes a byte of data to a single LH28F008SA. Note the use of BYTE PTR notation to force x8 accesses. If a string of bytes is to be written at one time, the V_{pp} ramp up, \overline{PWD} disable and device ID checks need only be done before the first byte write attempt. Additionally, when writing multiple bytes at once, examination of bits other than bit 7 (WSM Status) need only occur after the last byte write has completed. The Status Register retains any error bits until the Clear Status Register command is written.

			to be written, ES:[DI] is the location to be written on, DS and ES reference a descriptor	
			y this procedure	
VRITE_SETU		EQU	40H	
READ_ID	F	EQU		
ANUFACTU		EQU	90H	
	ien_iu		89H	
		EQU	0A2H	
		EQU	0A1H	
		EQU	80H	
V_ERR_FLAG	i	EQU	- 10H	
PP_FLAG		EQU	08H	· · · ·
	Insert code	-	Vpp and disable component /PWD input. If a string o	
			i at one time, Vpp ramp to 12V and ID cheek need or rst word is written.	nly occur once,
	MOV	AX,	"Address 0 for target LH28F008SA-segment"	; Initialize pointer to LH28F008SA address 0
	MOV	ES,	AX	
	MOV	DI,	"Address 0 for target LH28F008SA-offset"	
	MOV	ES:[DI],	((READ_ID SHL 8) OR READ_ID)	; Write Intelligent Identifier command
	CMP	ES:[DI],		TURER_ID) ; Does manufacturer ID read correctly?
	JNZ	W_WRD_IC		,,,,,, _
	MOV	DL	"Address 1 for target LH28F008SA-offset"	; Initialize pointer to LH28F008SA address 1
	CMP	ES:[DI],	((DEVICE_ID SHL 8) OR DEVICE_ID)	; Does device ID read correctly?
	JZ	W_WRD_ID		, Eves derive is read controlly :
	CMP	ES:[DI],	((DEVICE_ID2 SHL 8) OR DEVICE_ID2)	
	JNZ			
	JINZ	W_WRD_IC	<u></u>	
_WRD_ID_P	ACC			
	MOV	AX.	"Pute write dectination address compat"	; Initialize pointer to byte write dest. address
	MOV	ES,	"Byte write destination address-segment" AX	, Initialize pointer to byte write dest. address
	MOV	DI,	"Byte write destination address-offset"	
	MOV	ES:[DI],	((WRITE_SETUP SHL 8) OR WRITE_STEUP)	; Write byte write setup command
	MOV	AX,	DS:[SI]	; Load AX with data to write
	MOV	ES:[DI],	AX	; Write to devices
/_WRD_LOO			· · · · · · · · ·	
	TEST	ES:[DI],	((READY SHL 8) OR READY)	; Read LH28F008SA Status Registers
	JZ	W_WRD_LO	DOP	; Loop until bit 7 = 1
	TEST	ES:[DI],	(((W_ERR_FLAG_OR_VPP_FLAG) SHL 8) OR (W FRR FLAG OR VPP FLAG))
	JZ	W_WRD_C		; Success!
		_ND_C		
	MOV	AX,	ES:(DI),	; Load Status Register data into AX
	TEST	AL,	W_ERR_FLAG	Check Status Register bit 4 (low byte)
	JNZ	W_WRD_EI		; Jump if = 1 ; Check Status Register bit 4 (high byte)
	TEST	AH,	W_ERR_FLAG	
	JNZ	W_WRD_E		; Jump if = 1
	T 07			Check Status Posister bit 2 (law bits)
	TEST	AL,	VPP_FLAG	; Check Status Register bit 3 (low byte)
	JNŻ	W_WRD_V		; Jump if = 1
	TEST	AH,	VPP_FLAG	; Check Status Register bit 3 (high byte)
	JNZ	W_WRD_V	pp	; Jump if = 1
_WRD_ID_E				
		•	roper device ID read error here.	
	Are LH28F	008SA /PWD ii	nputs disabled? Is Vcc applied to the LH28F008SAs	?
_WRD_ERR				
	Insert code	e to service byte	e write error here	
_WRD_VPP:				
-	Insert code	to service hyte	write Vpp low error here	
	113611 0000		inter the terminer of term	
BYT_CONT				

This routine writes a word of data to a pair of LH28F008SAs. Note that all constants have been "OR'd" for parallel read/write of two devices at once. If a string of words is to be written at one time, the V_{pp} ramp up, \overline{PWD} disable and device ID checks need only be done before the first word write attempt. Additionally, when writing multiple words at once, examination of bits other than bit 7 (WSM Status) need only occur after the last word write has completed. The Status Register retains any error bits until the Clear Status Register command is written.

;				a single LH28F008SA (x8 block (erase)	
:		ts to the block t		· · · · · · · · · · · · · · · · · · ·		
;		•	n, ES reference	s a descriptor		
:	Register AX	is modified by	•			
ERASE_SETU	P	EQU	20H			
ERASE_CONF	IRM	EQU	ODOH			
READ_ID		EQU	90H			
MANUFACTU	RER_ID	EQU	89H	-		
DEVICE_ID		EQU	0A2H			
DEVICE_ID2		EQU	0A1H			
READY		EQU -	80H	•		
E_ERR_FLAG		EQU	20H			
E_CMD_FLAG	i	EQU	30H			
VPP_FLAG		EQU	08H			
; _	Insert code	here to ramp V	pp and disable (component /PWD input. If a strin	g of block	(s is
;		to be erased	at one time, Vp	p ramp to 12V and ID check need	d only occ	cur once,
;		before the first	st block is erase	d.		
	MOV	AX.	"Address 0 fo	r target LH28F008SA-segment"	;	Initialize pointer to LH28F008SA address 0
	MOV	ES,	AX			
	MOV	DI,	*Address 0 fo	r target LH28F008SA-offset*		
	MOV	BYTE PTR E		READ_ID	;	Write Intelligent Identifier command
	CMP	BYTE PTR E	• •	MANUFACTURER_ID	;	Does manufacturer ID read correctly?
	JNZ	E_BYT_ID_E		_		
	MOV	DI.		r target LH28F008SA-offset*	:	Initialize pointer to LH28F008SA address 1
	CMP	BYTE PTR E		DEVICE_ID	;	Does device ID read correctly?
	JZ	E_BYT_ID_F		-		2 Bay,
	CMP	BYTE PTR E		DEVICE_ID2		
	JNZ	E_BYT_ID_E	•••	-		
	0.12					
E_BYT_ID_PA	SS:					
	MOV	AX.	"Block erase	destination address-segment"	;	Initialize pointer to block erase dest. address
	MOV	ES.	AX	-		
	MOV	DI,		destination address-offset*		
	MOV	BYTE PTR E		ERASE_STEUP	;	Write block erase setup command
	MOV	BYTE PTR E	• •	ERASE_CONFIRM	;	Write block erase confirm command
E_BYT_LOOP						
	TEST	BYTE PTR E	S:(Dil.	READY	;	Read LH28F008SA Status Register
	JZ	E_BYT_LOC			:	Loop until bit 7 = 1
	9 4	2_011_000				
	TEST	BYTE PTR E	ES:(DI).	(E_CMD_FLAG OR VPP_FLA	AG))	
	JZ	E_BYT_CO		(;	Success!
	02		••			
	TEST	BYTE PTR E	SIDI	E_CMD_FLAG	:	Check Status Register bit 4 and 5
	JNZ	E_BYT_CM				Jump if = 1
	TEST	BYTE PTR 8	-S:(D)	E_ERR_FLAG	:	Check Status Register bit 5
	JNZ	E_BYT_ERF				Jump if = 1
	0.12	2_011_21.4	•			•
	TEST	BYTE PTR E	ES:IDII.	VPP_FLAG	;	Check Status Register bit 3
	JNZ	E_BYT_VPF			:	Jump if = 1
E_BYT_ID_ER		c_ 0				•
		to service imp	roper device ID	read error here.		
				Vcc applied to the LH28F008SA	?	
, E_BYT_CMD						-
E_BTI_OMO	_CRN.			nd sequence error here (setup fo	bliowed by	y a command other than confirm)
;			A BIASE COMINA			,
E_BYT_ERR:		to convine hise	k arasa amar h	210		
	meen cooe		k erase error h			
E_BYT_VPP:	locort code	to convine blac	k erase Vpp lov	error bere		
		TO SELVICE DIOC	w arasa Abb 101			
E_BYT_CON	1.					
	Code conti					

This routine erases a block of a single LH28F008SA. Note the use of BYTE PTR notation to force x8 accesses. If a string of blocks is to be erased at one time, the V_{PP} ramp up, PWD disable and device ID checks need only be done before the first block erase attempt. Additionally, when erasing multiple blocks at once, examination of bits other than bit 7(WSM Status) need only occur after the last block erase has completed. The Status Register retains any error bits until the Clear Status Register command is written.

The following code controls block erase of a pair of LH28F008SAs (x16 block erase) ES:[DI] points to the blocks to be erased In protected mode operation, ES references a descriptor Register AX is modified by this procedure ERASE_SETUP FOU 20H ERASE_CONFIRM EQU ODOH READ_ID EQU 90H MANUFACTURER_ID EOU 89H DEVICE_ID EQU 0A2H DEVICE ID2 EQU 0A1H READY EQU 80H E_ERR_FLAG EQU 20H E_CMD_FLAG FOU 30H VPP_FLAG EQU 08H Insert code here to ramp Vpp and disable component /PWD inputs. If a string of blocks is to be erased at one time. Vpp ramp to 12V and ID check need only occur once, before the first block pair is erased. : "Address 0 for target LH28F008SA-segment" MOV Initialize pointer to LH28F008SA address 0 AX, : MOV ES. AX MOV DI, "Address 0 for target LH28F008SA-offset" ES:[DI], ((READ_ID SHL 8) OR READ_ID) Write Intelligent Identifier command MOV ((MANUFACTURER_ID_SHL 8) OR MANUFACTURER_ID) ; Does manufacturer ID read correctly? CMP ES:[DI], JNZ E_WRD_ID_ERR "Address 1 for target LH28F008SA-offset" Initialize pointer to LH28F008SA address 1 MOV DI. : ES:[DI], ((DEVICE_ID SHL 8) OR DEVICE_ID) Does device ID read correctly? CMP .17 E_WRD_ID_PASS CMP ES:[DI], ((DEVICE_ID2 SHL 8) OR DEVICE_ID2) JNZ W_WRD_ID_ERR E_WRD_ID_PASS: "Block ease destination address-segment" Initialize pointer to block erase dest. address MOV AX. MOV ES. AX MOV DI. "Block erase destination address-offset" MOV ES:[DI], ((ERASE_SETUP SHL 8) OR ERASE_STEUP) ; write block erase setup command ((ERASE_CONFIRM SHL 8) OR ERASE_CONFIRM) ; write block erase confirm command MOV ES:[DI], E_WRD_LOOP: TEST ES:[DI], ((READY SHL 8) OR READY Read LH28F008SA Status Registers Loop until bit 7 = 1 E_WRD_LOOP JZ (((E_CMD_FLAG OR VPP_FLAG) SHL 8) OR (E_CMD_FLAG OR VPP_FLAG)) TEST ES:[DI], E_WRD_CONT Success JZ Load Status Register data into AX MOV AX, ES:[DI], Check Status Register bit 4 and 5 (low byte) TEST AL, E_CMD_FLAG E_WRD_CMD_ERR Jump if = 1 JNZ Check Status Register bit 4 and 5 (high byte) E_CMD_FLAG TEST AH. Jump if = 1 JNZ E_WRD_CMD_ERR Check Status Register bit 5 (low byte) TEST E_ERR_FLAG; AL, Jump if = 1 E_WRD_ERR JNZ Check Status Register bit 5 (high byte) TEST AH, E_ERR_FLAG; Jump if = 1JNZ E_WRD_ERR Check Status Register bit 3 (low byte) VPP_FLAG TEST AL. Jump if = 1 JNZ E_WRD_VPP Check Status Register bit 3 (high byte) VPP_FLAG TEST AH, E_WRD_VPP Jump if = 1 JNZ E_WRD_ID_ERR: Insert code to service improper device ID read error here. Are LH28F008SA /PWD inputs disabled? Is Vcc applied to the LH28F008SAs? E_WRD_CMD_ERR: insert code to service block erase command sequence error here (setup followed by a command other than confirm) E WRD ERR: Insert code to service block erase error here E_WRD_VPP: Insert code to service block erase Vpp low error here E_WRD_CONT: Code continues from this point

This routine erases a block pair of two LH28F008SAs. Note that all constants have been "OR'd" for parallel read/write of two devices at once. If a string of block pairs is to be erased at one time, the V_{pp} ramp up, \overline{PWD} disable and device ID checks need only be done before the first block pair erase attempt. Additionally, when erasing multiple block pairs at once, examination of bits other than bit 7 (WSM Status) need only occur after the last block pair erase has completed. The Status Register retains any error bits until the Clear Status Register command is written.

3.0 "C" DRIVERS

	drive byte write, block erase, Status Register read and clear and array read algorithms. Sa /PWD control blocks are also included, as are example programs combining drivers into ful tions listed below are included: erasbgn(): Begins block erasure erasusp(): Suspends block erase to allow reading data from a block of the LH28F008SA that being erased erasres(): Resumes block erase if suspended end(): Polis the Write State Machine to determine if block erase or byte write have completed eraschk(): Executes full status check after block erase completion writebgn(): Begins byte write writechk(): Executes full status check after byte write completion kdreed(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vpdown(): Disables Vpph pwden(): Disables active low signal /PWD pwddis(): Disables active low signal /PWD	il algorithms other than the control of the control the control of the control the control of the control of the control the control of the c	•1 •1	
	tions listed below are included: erasbgn(): Begins block erasure erassusp(): Suspends block erase to allow reading data from a block of the LH28F008SA that being erased erasres(): Resumes block erase if suspended end(): Polls the Write State Machine to determine if block erase or byte write have completed eraschk(): Executes full status check after block erase completion writebgn(): Begins byte write writechk(): Executes full status check after block erase completion idreed(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vpdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD as are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA		•1	
	erasbgn(): Begins block erasure erassusp(): Suspends block erase to allow reading data from a block of the LH28F008SA that being erased erasres(): Resumes block erase if suspended end(): Polls the Write State Machine to determine if block erase or byte write have completed eraschk(): Executes full status check after block erase completion writebgn(): Begins byte write writechk(): Executes full status check after block erase completion kritechk(): Executes full status check after block erase completion kritechk(): Executes full status check after byte write completion kritechk(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA		•1	
Address	erassus(): Suspends block erase to allow reading data from a block of the LH28F008SA that being erased erasres(): Resumes block erase if suspended end(): Polls the Write State Machine to determine if block erase or byte write have completed eraschk(): Executes full status check after block erase completion writebgn(): Begins byte write writechk(): Executes full status check after block erase completion kritebgn(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statclr(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA		•1	
Address	 that being erased erasres(): Resumes block erase if suspended end(): Polls the Write State Machine to determine if block erase or byte write have completed eraschk(): Executes full status check after block erase completion writechk(): Executes full status check after block erase completion writechk(): Executes full status check after byte write completion kritechk(): Executes full status check after byte write completion kreads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statclr(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (le long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA 		•1	
Address	erasres(): Resumes block erase if suspended end(): Polls the Write State Machine to determine if block erase or byte write have completed eraschk(): Executes full status check after block erase completion writebgn(): Begins byte write writechk(): Executes full status check after byte write completion idreed(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statch(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA		•1	5. 14 15 15
Address	end(): Polls the Write State Machine to determine if block erase or byte write have completed eraschk(): Executes full status check after block erase completion writebgn(): Begins byte write writechk(): Executes full status check after byte write completion idreed(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statch(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA		•1	5. 14 15 15
Address	completed eraschk(): Executes full status check after block erase completion writebgn(): Begins byte write writechk(): Executes full status check after byte write completion idread(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA	SA is then	•1	1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1
Address	eraschk(): Executes full status check after block erase completion writebgn(): Begins byte write writechk(): Executes full status check after byte write completion idreed(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA	SA is then	•1	
Address	writebgn(): Begins byte write writechk(): Executes full status check after byte write completion idreed(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA	SA is then	•1	
Address	writechk(): Executes full status check after byte write completion idreed(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD as are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA	SA is then	•1	•
Address	 idreed(): Reads and returns the manufacturer and device IDs of the target LH28F008SA statrd(): Reads and returns the contents of the Status Register statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA 	SA is then		
Address	 statrd(): Reads and returns the contents of the Status Register statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD as are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA 	SA is then		••••
Address	statcir(): Clears the Status Register rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD as are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA	SA is then		•••
Address	rdmode(): Puts the LH28F008SA in Read Array mode rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD pwddis(): Disables active low signal /PWD as are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA	SA is then		••••
Address	rdbyte(): Reads and returns a specified byte from the target LH28F008SA vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddls(): Disables active low signal /PWD are transferred to functions as pointers to far bytes (ie long integers). An alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008SA	SA is then		× · .
Address	vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddls(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). An alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	SA is then		
Address	vppup(): Enables high voltage Vpph vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddls(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). An alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	3A is then		
Address	vppdown(): Disables Vpph pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). An alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	SA is then		
Address	pwden(): Enables active low signal /PWD pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). An alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	SA is then		
Address	pwddis(): Disables active low signal /PWD es are transferred to functions as pointers to far bytes (ie long integers). An alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	SA is then	•/	
Address	es are transferred to functions as pointers to far bytes (ie long integers). Ah alternate approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	SA is then		
Address	approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	SA is then		
	approach is to create a global array the size of the LH28F008SA and located "over" the LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	SA is then		
	LH28F008SA in the system memory map. Accessing specific locations of the LH28F008S	SA is then		
	specific address. Different microprocessor architectures will require different array			
	definitions; le for the x86 architecture, define it as "byte eightmeg[16][10000]" and pass			
	each function TWO offsets to access a specific location. MCS-96 architectures are			
	limited to "byte eightmeg[10000]"; alternate approaches such as using port pins for paging	line i		
	be required to access the full flash array			
	na radmian in annasa nia inii nasii arrak			
To const	a far pointer, a function such as MK_FP() can be used, given a segment and offset in			
10 0.041	the x86 architecture. I use Turbo-C; see your compiler reference manual for additional			
	information.			
•••••		• • • • • • • • • • • •		• • • •
•••••		•••••		• • • •
Revision	History: Rev 2.0			
Changes	From Revision 1.0 to Revision 2.0:	•		
	Added alternate LH28F008SA device ID to routine idread()			

r	Function: M	ain			•/
r	Description	The following code (c	ommente	d out) shows examples of byte write and block erase algorithms that can be	•/
^				plication and hardware design	•/
		1. A. A. A.			
main()					
(~	
	byte far *ad	dress;			
	byte data si	atus;		,	
		-			
r -				ossible byte write algorithm.	•/
r*				etween byte writes when a string of byte writes occurs. Ramp Vpp to 12V	•/
r		•		until after completion of the last byte write. Doing so minimizes Vpp ramp	•/
• .		lay and maximizes by	e write th	roughput	•/
	vppup();				
-	"INSERT S	OFT WARE DELAY	OR VPP	RAMP IF REQUIRED	•/
	pwddis();				
	address	= 0XxxxxxL;			
	date	= 0Xyy;			
	if (writebgni	data.address) == 1)			•
•		"RECOVERY CO	E-POWE	R NOT APPLIED (ID CHECK FAIL)"	•
	else				
	{				
		while (end(&status)		
		:			- 1 t
		switch (writechk(st	atus))		
		{			
		cas	e 0:	·	
				break;	
		cas	e 1:		
•				"RECOVERY CODE-VPP LOW DETECT ERROR"	•/
				break;	
		cas	ə 2:		
•				"RECOVERY CODE-BYTE WRITE ERROR"	•/
	-			break;	
		}			
		statcir();			
	3				
	vppdown()				

This "C" routine gives an example of combining lower-level functions (found in following pages) to complete a byte write. Routines vppup() and pwddis() enable the LH28F008SA for byte write. Function writebgn() issues a byte write sequence to device, end() detects byte write completion via Status Register bit 7, and writechk() analyzes Status Register bits 3-6 to determine byte write success. If a string of bytes is to be written at one time, V_{pp} ramp up and \overline{PWD} disable need only be done before the first byte write attempt. Additionally, when writing multiple bytes at once, examination of bits other than bit 7 (WSM Ready) need only occur after the last byte write has completed. The Status Register retains any error bits until the Clear Status Register command is written.

7

r

r

r

"

r

"

•/ The following code gives an example of a possible block erase algorithm. ٩ Note that Vpp does not need to be cycled between block erase when a string of block erases occurs. Ramp Vpp to 12V •/ before the first block erase and leave at 12V until after completion of the last block erase. Doing so minimizes Vpp ramp •/ up-down delay and maximizes block erase throughput vppup(); •7 "INSERT SOFTWARE DELAY FOR VPP RAMP IF REQUIRED pwddis(); address = 0XxxxxxL; if (erasbgn(address) == 1) "RECOVERY CODE-POWER NOT APPLIED (ID CHECK FAIL)" •/ else Ł while (end(& status)) switch (erasehk(status)) 1 case 0: break: case 1: •/ "RECOVERY CODE-VPP LOW DETECT ERROR" break: case 2: •/ "RECOVERY CODE-BLOCK ERASE ERROR" break: case 3: "RECOVERY CODE-ERASE SEQUENCE ERROR" •/ break; } statcir(); ļ vppdown();

This "C" routine gives an example of combining lower-level functions (found in following pages) to complete a block erase. Routines vppup() and pwddis() enable the LH28F008SA for block erase. Function erasbgn() issues a block erase sequence to the device, end() detects block erase completion via Status Register bit 7, and eraschk() analyzes Status Register bits 3-6 to determine block erase success. If a string of blocks is to be erased at one time, V_{pp} ramp up and \overline{PWD} disable need only be done before the first block erase attempt. Additionally, when erasing multiple blocks at once, examination of bits other that bit 7 (WSM Ready) need only occur after the last block erase has completed. The Status Register retains any error bits until the Clear Status Register command is written.

Sharp

r	Function: En	980h0	* * * * * * * * * * * * * * * * * * * *	
<i>r</i>	Description	Begins erase of a block.	ו	
<i>r</i>	Inputs;	beyints enaise of a block.		•/
r	Outputs:	Nickador: System addr	ess within the block to be erased	7
r	Returns:	None		• • • • • • • • • • • • • • • • • • • •
<i>r</i>	r tenzi i ta,	0 = Block erase succes	ssfully initiated	. 1
r	Benden Deard	1 = Block erase not init	liated (ID check error)	• • • • • • • • • • • • • • • • • • • •
		Mode on Return: Status	Register (ID if returns 1)	•/
			***************************************	•/
#define	ERASETUP	0X20	d Emer Contra à	
#define	ERASCONF	0XD0 -	/* Erase Setup command	•/
			/* Erase Confirm command	•,
int erasbgr	n(bickaddr)			,
byte far *bi	ckaddr;		/* bickaddr is an address within the block to be erased	
			statutating an address within the block to be erased	•7
{				
	byte mfgrid.de	viceid;		
	if (idread(&mfg	rid,&deviceid) == 1)	/* ID read error; device not powered up?	
		return (1);	is rous error, device not powered up?	•/
	bickaddr	= ERASETUP:	/ Write Erase Setup command to block address	· · ·
	bickaddr	= ERASCONF;	/ Write Fraze Confirm command to block address	•/
	return (0);	•	/" Write Erase Confirm command to block address	•/
}				

Routine erasbgn() issues a block erase command sequence to a LH28F008SA. It is passed the desired system address for the block to be erased. After calling idread(), it writes the erase command sequence at the specified address. It returns "0" if block erase initiation was successful, and "1" if the ID read fails (device not powered up or PWD not disabled).

÷.

~	Function: Era	55050			•/
~		Suspends block erase to rea	d from another	block	·•/
,	inputs:	None			•/
~	Outputs:	None			•/
~	Returns:	0 = Block erase suspende	d		•/
, -	recentio.			block erase suspend not possible)	•/
, •	Device Read	Mode on Return: Status Re	-		•/
, ,		•••••	•••••	******	•••••
#define	RDYMASK	0X80	/" Mask to i	solate the WSM Status bit of the Status Register	•/
#define	WSMRDY	0X80	/* Status R	egister value after masking, signifying that the WSM is	•/
			r	no longer busy	•/
Idefine	SUSPMASK	0X40	/* Mask to	solate the erase suspend status bit of the Status Register	•/
Idefine	ESUSPYES	0X40	/* Status R	egister value after masking, signifying that block erase has	•/
			r	been suspended	•/
Idefine	STATREAD	0X70	/* Read Sta	atus Register command	•/
Idefine	SYSADOR	0	/* This con	stant can be initialized to any address within the memory map	•/
			r	of the target LH28F008SA and is alterable depending on the	•/
			r	system architecture	•/
Idefine	SUSPCMD	0XB0	/* Erase Su	ispend command	•
int erassus	ip()				
	byte far "stati	addr;	/* Pointer v	ariable used to write commands to device	•/
	stataddr	= (byte far *)SYSADDR;			
	stataddr	= STATREAD;	/ Write Re	ed Status Register command to LH28F008SA	•/
	if ((*stataddr	& RDYMASK) == WSMRDY	0		
		return (1);	/" WSM is	not busy; block erase suspend possible. Error code "1"	•/
	stataddr	= SUSPCMD;	/ Write Era	ase Suspend command to the device	•/
	while (("stata	ddr & RDYMASK) ! == WSN	(RDY)		
		:	/* Will rema	ain in while loop until bit 7 of the Status Register goes to 1,	•/
			r	signifying that the WSM is not longer busy	•/
	while ((*stata	ddr & SUSPMASK) ! == ES	USPYES)	•	
		;	/* Will rema	ain in while loop until bit 6 of the Status Register goes to 1,	•/
			<i>r</i>	signifying block erase suspension	•/

return (0);

Routine erassusp() issues the erase suspend command to a LH28F008SA. It first makes sure the WSM is truly busy, then issues the erase suspend command and polls Status Register bits 7 and 6 until they indicate erase suspension. It returns "0" if block erase suspend was successful, and "1" if the WSM was not busy when suspend was attempted.

4	Function: Era	5785		•/
•		Resumes block erase pr	eviously suspended	•/
~	Inputs:	None		•/
~	Outputs:	None		•/
~	Returns:	0 = Block erase resun	ned	•/
•		1 = Error; Block erase	not suspended when function called	•/
r r	Device Read	Mode on Return: Statu:	s Register	•••••
#define	RDYMASK	0X80 -	/* Mask to isolate WSM Status bit of the Status Register	•/
Idefine	WSMRDY	0X80	/* Status Register value after masking, signifying that the WSM is	_ • ⁄
			/* no longer busy	•/
idefine	SUSPMASK	0X40	/* Mask to isolate the erase suspend status bit of the Status Register	•/
Idefine	ESUSPYES	0X40	/* Status Register value after masking, signifying that block erase	•
			/* has been suspended	•/
klefine	STATREAD	0X70	/" Reed Status Register command	•/
Idefine	SYSADDR	0	/* This constant can be initialized to any address within the memory map	•/
			/* of the target LH28F008SA and is alterable depending on the	•/
			/* system architecture	•/
Idefine	RESUMCMD	0XD0	/* Erase Resume command	
int erasres()			
i -	byte far "stati	addr;	/* Pointer variable used to write commands to device	- •/
	stataddr	= (byte far *)SYSADD		
	stataddr	= STATREAD;	/ Write Read Status Register command to LH28F008SA	•/
	if (("stataddr	& SUSPMASK) I = ESU	•	
		return (1);	/* Block erase not suspended. Error code "1"	
	stataddr	= RESUMCMD;	/ Write Erase Resume command to the device	•
	while (("stata	ddr & SUSPMASK) == 1		-
		;	/" Will remain in while loop until bit 6 of the Status Register goes to 0,	•/
			/* signifying block erase resumption	•/

signifying that the WSM is once again busy

•/

•7

while ((*stataddr & RDYMASK) --- WSMRDY) /* Will remain in while loop until bit 7 of the Status Register goes to 0,

/*

:

return (0);

}

Routine erasres() issues the erase resume command to a LH28F008SA. It first makes sure the WSM is truly suspended, then issues the erase resume command and polls Status Register bits 7 and 6 until they indicate WSM resumption. It returns "0" if block erase resume was successful, and "1" if the WSM was not suspended when resumption was attempted.

<i>r</i> ·····			* • • • • • • • • •		•••••	
r	Function: En	id .			•/	
<i>r</i> •	Description:	Checks to see if the WSM is b	usy (is byte w	ite/block erase completed?)	•/	
r	Inputs:	None			•/	
r	Outputs:	statdata; Status Register da	ta read from d	evice	•/	
r	Returns:	0 - Byte Write/Block Erase	completed		•/	
r		1 = Byte Write/Block Erase	still in progres:	B	•/	
r	Device Read	Mode on Return: Status Regi	ister		•/	
/******	•••••	••••••••••	• • • • • • • • • •		•••••	
M-1 - M	001444014	-	e Manak an is	colate WSM Status bit of the Status Register	•/	
#define	RDYMASK	0X80			•	
#define	WSMRDY	0X80		gister value after masking, signifying that the WSM	•	
			/*	is no longer busy	•/	
#define	STATREAD	0X70		tus Register command		
#define	SYSADDR	0	/" This cons	tant can be initialized to any eddress within the memory map	•,	
			~	of the target LH28F008SA and is alterable depending on the	•	
			~	system architecture	,	
int end(stat	data)					
byte "statda	ata:		/* Allows Sta	atus Register data to be passed back to the main program	4	
			r	for further analysis	•/	
{	byte far "stataddr;		/" Pointer variable used to write commands to device			
	stataddr	= (byte far *) SYSADDR;				
	stataddr	= STATREAD:	/ Write Rea	d Status Register command to LH28F008SA	•/	
	if (((*statdata	= "stataddr) & RDYMASK) ! =	WSMRDY	-		
		return (1);		/block erasure still in progresscode "1"	•/	
	return (0);		•	/block erase attempt completedcode "0"	•/	
3				•		

Routine end() detects completion of byte write or block erase operations of a LH28F008SA. It passes back the Status Register data it reads from the device. It also returns "0" if Status Register bit 7 indicates WSM "Ready", and "1" if indication is that the WSM is still "Busy".

<i>r</i> ·····	•••••	• • • • • • • • • • • • • • • • • • • •		•••••
<i>r</i>	Function; Era	schk		•/
r	Description; (Completes full Status Register	r check for block erase (proper command sequence, Vpp low detect,	۲
r		block erase success). This r	outine assumes that block erase completion has already been checked in function	•/
r		end(), and therefore does no	ot check the WSM Status bit of the Status Register	٩
r	inputs;	statdata: Status Register da	ate read in function end	•/
/*	Outputs;	None	_	•/
r	Returns:	0 = Block erase completed s	successfully	•/
r		1 = Error: Vop low detect	•	•/
r		2 = Error: Block erase error		•/
r		3 = Error; Improper commer		•/
r	Device Read	Mode on Return; Same as w	· •	•/
· · · · · · ·	*******			•••••
#define	ESEQMASK	0X30	/" Mask to isolate the erase and byte write status bits of the Status Register	•
#define	ESEQFAIL	0X30	/* Status Register value after masking if block erase command sequence error	· •/
			/* has been detected	•/
#define	EERRMSK	0X20	/* Mask to isolate the erase status bit of the Status Register	•/
#define	ERASERR	0X20	/* Status Register value after masking if block erase error	•/
			/* has been detected	•/
#define	VLOWMASK	0X08	/* Mask to isolate the Vpo status bit of the Status Register	•7
#define	VPPLOW	0X08	/* Status Register value after masking if Vpp low has been detected	•/
int eraschk	(statdata)			
byte statdat	te;		/* Status Register date that has been already read from the 28F008SA	
			/* in function end()	•/
ł				
-	if ((statdate &	VLOWMASK) == VPPLOW)		
		return (1);	/* Vpp low detect error, return code "1"	•/
	if ((statdate &	EERRMSK) == ERASERR)		
		return (2);	/" Block erase error detect, return code "2"	•/
	if ((statdate &	ESEQMASK) == ESEQFAIL)		
	.,	return (3);	/* Block erase command sequence error, return code "3"	•/
	return (0);		/* Block erase success, return code "0"	•/
1				

Routine eraschk() takes the Status Register data read in end() and further analyzes it. It returns "0" if block erase was successful, "1" if V_{pp} low error was detected, "2" if block erase error was reported and "3" if an erase command sequence error was found (erase setup followed by a command other than erase confirm). This is useful after a block or string of blocks has been erased, to check for successful completion.

				• • • • • • • • • • • • • • • • • • • •
r r	Function; Writ	eban		•/
, ,	•	legins byte write sequence		•/
/*	inputs;	wdata; Data to be written inti	o the device	•/
/		waddr; Target address to be		•/
r	Outputs;	None		•/
ŕ	Returns;	0 = Byte write successfully in	hitiated	•/
, ,		1 = Byte write not initiated (il		•/
, ,	Davice Read	Mode on Return; Status Regis	•	•/
, ,		•••••		••••••
M.J	0000	-		•/
#define Int writeban(w	SETUPCMD	0.0.40	/* Byte Write Setup command	'
nu wurandnuw				
byte wdata;			/* Date to be written into the LH28F008SA	•/
byte for *wade	dr;		/" waddr is the destination address for the date to be written	•7
÷				
{				
	byte mfgrid,de	wiceid;		
	if (idrood/8 mt	grid,&deviceid) == 1)	/* Device ID read errorpowered up?	•/
		- · ·		
	Secondala.	return (1);	/* Write Byte Write Setup command and destination address	•/
	*waddr	= SETUPCMD;		'
	waddr	= wdata;	/ Write byte write data and destination address	1
	return (0);			· · · · ·
}				
<i></i>				• • • • • • • • • • • • • • • • • • • •
r	Function; Writ	iechk	•	•/
r			check for byte write (Vpp low detect, byte write success).	
r	Desa (pasi), e		te write completion has already been checked in function end()	•/
, ,			k the WSM Status bit of the Status Register	•/
, ,	Inputs;	statdata; Status Register da	-	•/
, ,	Outputs;	None		•/
,				•
•	Returns;	0 = Byte write completed sur		, ,
/*		1 = Error; Vpp low detect		-/
<i>r</i>		2 = Error; Byte write error		, •/
<i>r</i>	Device Read	Mode on Return; Status Regi	310f	
<i>r</i>				,
#define	WERRMSK	0X10	/* Mask to isolate the byte write error bit of the Status Register	*/
#define	WRITERR	0X10	/* Status Register value after masking if byte write error has been detected	•/
#define	VLOWMASK	0X08	/" Mask to isolate the Vpp status bit of the Status Register	•/
#define	VPPLOW	0X08	/* Status Register value after masking if Vpp low has been detected	*/
int writechk(st	atdata)			
•	•			
byte statdata;			/* Status Register data that has been already read from the LH28F008SA /* in function end()	•/ •/
{				
	if ((statdata &	VLOWMASK) - VPPLOW)		
		return (1);	/* Vpp low detect error, return code "1"	•/
	if ((statdata &	WERRMSK) == WRITERR)		
		return (2);	/* Byte write error detect, return code "2"	•/
	return (0);		/" Byte/string write success, return code "0"	•/
}				

Routine writebgn() issues a byte write command sequence to a LH28F008SA. It is passed the desired system address for the byte to be written, as well as the data to be written there. After calling idread(), it writes the byte write command sequence at the specified address. It returns "0" if byte write initiation was successful, and "1" if the ID read fails (device not powered up or PWD not disabled).

Routine writechk() takes the Status Register data read in end() and further analyzes it. It returns "0" if byte write was successful, "1" if V_{pp} low error was detected, and "2" if byte write error was reported. This is useful after a byte or string of bytes has been written, to check for successful completion.

<i>,</i>				••••••
r	Function; Idread			
r			device IDs from the target LH28F008SA	•/
/•	Inputs;	None		•/
/•	Outputs;	mfgrid; Returned manufact	turer ID	•/
r	•	deviceid; Returned device		
r	Returns;	0 = ID read correct		-7
r		1 = Wrong or no ID		•/
r	Device Read	Mode on Return; intelligent	Identifier	•/
<i>r</i> ·····	********	••••••••••••••••••••••••		•••••
#define	MFGRADDR	0 -	/* Address "0" for the target LH28F008SAalterable depending on the	•/
			/* system architecture	•/
#define	DEVICADD	1	/* Address "1" for the target LH28F008SAalterable depending on the	•/
			/* system architecture	•/
#define	. IDRDCOMM	0X90	/* Intelligent Identifier command	•/
#define	MANUFACTI	JRERID 0X89	/* Manufacturer ID for SHARP devices	•/
#define	DVCID	0X0A2	/ Device ID for LH28F008SA	•/
#define	DVCID2	0X0A1	/* Device ID for LH28F008SA-L	•/
byte *migric	•		 /* The manufacturer ID read by this function, to be transferred back to /* the calling program /* The device ID read by this function, to be transferred back to the /* calling function 	*/ */ */
{	byte far *tem	Daddr;	/* Pointer address variable used to read IDs	•/
	tempaddr	= (byte far *)MFGRADDR;		
	tempaddr	= IDRDCOMM;	/ Write intelligent Identifier command to an address within the LH28F008SA	•/
			/* memory map (in this case 00 hex)	•/
	mtgrid	= "tempaddr;	/ Read mfgr ID, tempaddr still points at address "0"	•/
	tempaddr	= (byte far *)DEVICADD;	/* Point to address "1" for the device specific ID	•/
	deviceid	= 'tempaddr;	/ Read device ID	•/
			eviceid! = DVCID) && (*deviceid! = DVCID2))	
		return (1);	/* ID read error; device powered up?	•/
	return (0);	····	- · · · · · ·	

}

Routine idread() issues the Intelligent Identifier command to a LH28F008SA. It passes back the manufacturer and device IDs it reads. In addition, it returns "0" if the IDs read matched those expected for the LH28F008SA, and "1" if either the manufacturer or device IDs did not match.

}

	Function; Sta	itrd		•/
-	Description; i	Returns contents of the targe	t LH28F008SA Status Register	•
•	Inputs;	None	· · · · ·	•
•	Outputs;	statdata; Returned Status I	Register data	•
-	Returns;	Nothing	-	•
•	Device Read	Mode on Return; Status Reg	pister	•/
• • • • • • • •	•••••			
Klefine	STATREAD	0X70	/* Read Status Register command	•
Idefine	SYSADDR	0 -	/" This constant can be initialized to any address within the memory map	•
		-	/* of the target LH28F008SA and is alterable depending on the	•
			/* system architecture	: •
nt stated(si	tatdata)			
•	n statrd(statdata) yte *statdata;		/* Allows Status Register data to be passed back to the calling program	•,
.,	•		/* for further analysis	•,
l				
	byte far*stata	ıddr;	/* Pointer variable used to write commands to device	•
	stataddr	= (byte far *)SYSADDR;		· · · ·
	stataddr	= STATREAD;	/ Write Read Status Register command to LH28F008SA	•
	"statdata = "s	stataddr;		
	return;			
}				1111
• • • • • • •				
•••••	Function; Sta	ıtcir		••••••
• • • • • • • • •	•	ntcir Clears the LH28F008SA Stat	us Register	•
• • • • • • • • •	•		us Register	
• • • • • • • • • • • • • • • • • • •	Description; (Clears the LH28F008SA Stat	us Register	•
• • • • • • • • • • • • • • • • • • •	Description; (Inputs;	Clears the LH28F008SA Stat None	us Register	•
• • • • • • • • • • • • • •	Description; (Inputs; Outputs; Relums;	Clears the LH28F008SA Stat None None	us Register	•
•	Description; (Inputs; Outputs; Relums;	Clears the LH28F008SA Stat None None Nothing	us Register	•
	Description; (inputs; Outputs; Relums; Device Read	Clears the LH28F008SA Stat None None Nothing		•
r r r r r Rdefine Klafine	Description; (Inputs; Outputs; Relurns; Device Read STATCLER	Clears the LH28F008SA Stat None None Nothing Mode on Return; Array	/* Clear Status Register command	•
- - - - - - - - - - - - - - - - - - -	Description; (inputs; Outputs; Relums; Device Read	Clears the LH28F008SA Stat None None Nothing Mode on Return; Array 0X50	/* Clear Status Register command /* This constant can be initialized to any address within the memory map	•
	Description; (Inputs; Outputs; Relurns; Device Read STATCLER	Clears the LH28F008SA Stat None None Nothing Mode on Return; Array 0X50	/* Clear Status Register command	•
	Description; (Inputs; Outputs; Relurns; Device Read STATCLER SYSADDR	Clears the LH28F008SA Stat None None Nothing Mode on Return; Array 0X50	/* Clear Status Register command /* This constant can be initialized to any address within the memory map /* of the target LH28F008SA and is alterable depending on the	-
Keine	Description; (Inputs; Outputs; Relurns; Device Read STATCLER SYSADDR	Clears the LH28F008SA Stat None None Nothing Mode on Return; Array 0X50 0	/° Clear Status Register command /° This constant can be initialized to any address within the memory map /° of the target LH28F008SA and is alterable depending on the /° system architecture	- - - - - - - - - - - - - - - - - - -
Idefine	Description; (Inputs; Outputs; Relurns; Device Read STATCLER SYSADDR	Clears the LH28F008SA Stat None None Nothing Mode on Return; Array 0X50 0	/* Clear Status Register command /* This constant can be initialized to any address within the memory map /* of the target LH28F008SA and is alterable depending on the	-

Routine statrd() reads a LH28F008SA Status Register. It issues the Read Status Register command and passes back the data it obtains.

Routine statclr() issues the Clear Status Register command to a LH28F008SA. This routine is required after analyzing Status Register contents in routines like eraschk() and writechk(). The LH28F008SA Status Register retains state of bits 3-6 until they are cleared by the Clear Status Register command.

Description: Puts the target LH28F008SA in Read Array Mode. This function might be used, for example, to prepare the system for return to code execution out of the Reah memory after byte write or block ense algorithms have been executed off-chip linputs: None Ouputs: None Returns: Nothing Device Read Mode on Return; Array Refine BDARRAY OXFF / Read Array command Krefine SYSADDR O / This constant can be initialized to any address within the memory map / of the target LH28F008SA and is alterable depending on the / eystem architecture nt rdmode() byte far "tempaddr; / Pointer variable used to write commands to the device tempaddr = (byte far')SYSADDR; "tempaddr = (byte far')SYSADDR; "tempaddr = RDARRAY; / Write Reed Array command to LH28F008SA return; Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program linputs; raddr; Target address to be read from Ouputs; rate; Data the specified address Returns; Nothing Device Read Mode on Return; Array Keifne RDARRAY 0XFF / Read array command th rdbyte(rdata, raddr) yre "rdata; ; / Returns data need from the device at specified address return; / Returns; Nothing Device Read Mode on Return; Array Keifne RDARRAY 0XFF / Read array command the device at specified address to be read from "raddr = RDARRAY; / Write read array command the target address to be read from "raddr = RDARRAY; / Write read array command to an address within the LH28F008SA (in this / Case the target address and area	~	Function; R	dmode		
Inputs: None Outputs: None Outputs: None Outputs: None Outputs: None Returns: Nothing Device Read Mode on Return: Array Kiefine RDARRAY SYSADDR 0 /* Press Constant can be initialized to any address within the memory map /* of the target LASPROSSA and is alterable depending on the /* of the target LASPROSSA and is alterable depending on the /* of the target LASPROSSA and is alterable depending on the /* of the target LASPROSSA and is alterable depending on the /* of the target LASPROSSA and is alterable depending on the /* eystem architecture int rdmode() /* byte far "tempeddr: /* '* unpeddr: - '* unpeddr: - '* Read Array command to LH28F008SA Description: Reads: Description: Read and the specified address Returns: Nothing Device Read Mode on Return: Array tdefine RDARRAY OUTFF <th>•</th> <th>Description;</th> <th>Puts the target LH28F008S</th> <th>A In Read Array Mode. This function might be used, for example, to prepare</th> <th rowspan="2">•</th>	•	Description;	Puts the target LH28F008S	A In Read Array Mode. This function might be used, for example, to prepare	•
Inputs: None Outputs: None Returns: Notifing Device Read Mode on Return; Array Riefine RDARRAY SYSADDR 0 /* This constant can be initialized to any address within the memory map /* of the target LH28F008SA and is alterable depending on the /* of the target LH28F008SA and is alterable depending on the /* eystem architecture nt rdmode() byte far "tempeddr; /* 'tempeddr = (byte far")SYSADDR; 'tempeddr = RDARRAY; 'return; 'Pointer variable used to write commands to the device tempeddr = RDARRAY; /* Write Read Array command to LH28F008SA return; - Punction; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; return; Outputs; return; array Device Read Mode on Return; Array Celline RDARRAY Outputs; /* Reed array command trictyte(rdata, raddr) /* Read array command	•	-	the system for return to co	de execution out of the Flash memory after byte write or block erase algorithms	
Inputs: None Outputs: None Resums: Notifing Device Read Mode on Return; Array Riefine RDARRAY SYSADDR 0 /* of the target LH28F008SA and is alterable depending on the /* eystem architecture nt rdmode() /* byte far "tempeddr; /* return; - byte far "tempeddr; /* return; - Pointer variable used to write commands to the device tempeddr = (byte far")SYSADDR; "tempeddr = RDARRAY; /* Write Read Array command to LH28F008SA return; - Pointer variable used to write commands to the device tempeddr = RDARRAY; /* Write Read Array command to LH28F008SA return; - Punction; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; radid: rate address to the read from Outputs; radid: rate address and returns it to the calling program Inputs; radid: rated address </th <th>*</th> <th></th> <th>have been executed off-ci</th> <th>hip</th> <th></th>	*		have been executed off-ci	hip	
Resums: Nothing Device Read Mode on Return; Array Klefine RDARRAY OXFF / Read Array command Klefine SYSADDR O / This constant can be initialized to any address within the memory map // of the target LH22F008SA and is alterable depending on the / of the target LH22F008SA and is alterable depending on the // of the target LH22F008SA is alterable depending on the // system architecture / Pointer variable used to write commands to the device tempador = (byte far 'Isempaddir; / Pointer variable used to write commands to the device tempador = RDARRAY; // Write Reed Array command to LH28F008SA return; Pointer variable used to write commands to the device Function; Rdbyte Description; Reeds a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be reed from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array Cellne RDARRAY OxFF /* Reed array command tribyte(rdata; readir; /* Returns data read from the device at specified address yie far 'raddr; -* ReA	•	Inputs;	None		•
Device Read Mode on Return: Array Modifine RDARRAY OXFF /* Read Array command /* This constant can be initialized to any address within the memory map /* of the target LH28F008SA and is attenable depending on the /* system architecture Int rdmode() byte far "tempaddr: /* Pointer variable used to write commands to the device tempaddr = (byte far')SYSADDR; "tempaddr = RDARRAY; /* Write Read Array command to LH28F008SA return; Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; radat; Date at the specified address Return; Anthing define RDARRAY 0XFF /* Reed array command returns data read from the device at specified address readr; ** Reddr; ** /* Reed array command define RDARRAY 0XFF /* Reed array command return; ** Reddr is the target address readr; ** /* Reed array command tradit; /* Reed array command ** Readress to be read from ** Raddr; ** /* Reddr is the target address to be read from	•	Outputs;	None		
Rdefine RDARRAY OXFF /* Read Array command Modifine SYSADDR 0 /* This constant can be initialized to any address within the memory map /* of the target LH28F008SA and is alterable depending on the /* /* of the target LH28F008SA and is alterable depending on the /* eystem architecture nt rdmode() /* byte far "tempaddr; /* Pointer variable used to write commands to the device tempaddr = (byte far')SYSADDR; *tempaddr = RDARRAY; /* Write Read Array command to LH28F008SA Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddir sthe specified address Returns; Nothing Device Read Mode on Return; Array /* Read array command tridpte(rdata; raddr) /* Returns data read from the device at specified address yre far 'raddr; */* */* Redurins data read from the device at specified address yre far 'raddr; */* */* Returns data read from the device at specified address yre far 'raddr; */* */* Returns data read from the device at specified address yre far 'raddr; */* */* Returns data read from the device at sp	•	Returns;	Nothing	~	
Kieffine SYSADDR O /* This constant can be initialized to any address within the memory map /* of the target LH28F008SA and is alterable depending on the /* of the target LH28F008SA and is alterable depending on the /* system architecture byte far "tempeddr: /* Pointer variable used to write commands to the device tempeddr = (byte far')SYSADDR; * tempeddr - RDARRAY; /* Write Read Array command to LH28F008SA Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; raddr; Target address Returns; Nothing Device Read Mode on Return; Array define RDARRAY yre "rdata; /* Returns data read from the device at specified address yre "rdata; /* Returns data read from the device at specified address yre "rdata; /* Returns data read from the device at specified address /* Raddr is the target address to be read from 'raddr; - RDARRAY; /* Returns data read from the device at specified address /* Raddr is the target address to be read from </th <th>•</th> <th>Device Read</th> <th>Mode on Return; Array</th> <th></th> <th></th>	•	Device Read	Mode on Return; Array		
define SYSADDR O /* This constant can be initialized to any address within the memory map /* of the target LH28F008SA and is alterable depending on the /* eystem architecture tir rdmode() byte far "tempeddr: /* Pointer variable used to write commands to the device tempeddr = (byte far')SYSADDR; * *tempeddr = (byte far')SYSADDR; * *tempeddr = RDARRAY; /* Write Read Array command to LH28F008SA Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program inputs; raddr; Target address to be read from Outputs; r raddr; Target address to be read from Outputs; returns; define RDARRAY 0XFF /* Reed array command trdbyte(rdata, raddr) /* Reed array command trdbyte(rdata, raddr) r/set "raddr; /* Returns data read from the device at specified address ** raddr; ** Returns data read from the device at specified address ** raddr; /* Returns data read from the device at specified address ** raddr; /* Returns data read from the device at specified address ** raddr; ** Readdr is the target address to be read from <t< td=""><td></td><td>•••••</td><td>~</td><td>•••••••••••••••••••••••••••••••••••••••</td><td>*****</td></t<>		•••••	~	•••••••••••••••••••••••••••••••••••••••	*****
f " of the target LH29F0085A and is alterable depending on the f " system architecture transpaddr = (byte far')SYSADDR; '' Pointer variable used to write commands to the device tempaddr = (byte far')SYSADDR; ''tempaddr = RDARRAY; f " Write Read Array command to LH29F008SA return; Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; raddr; Target address to be read from Outputs; raddr; Target address to be read from Outputs; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command tr rdbyte(rdata, raddr) yte *rdata; '' Returns data read from the device at specified address return; '' Returns data read from the device at specified address return; '' Returns data read from the device at specified address return; '' Returns data read from the device at specified address return;	Idefine	RDARRAY	OXFF	/* Read Array command	
/* system architecture ht rdimode() byte far "tempeddir; /* Pointer variable used to write commands to the device tempeddr tempeddr = (byte far")SYSADDR; "tempeddr * Write Read Array command to LH28F008SA return; Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; Punction; Rdbyte Description; Reads a byte of data from a specified address Description; Reads a byte of data from a specified address Returns it to the calling program Inputs; raddr; Target address to be read from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY VXFF /* Read array command tt rdbyte(rdata; readdr; /* Returns data read from the device at specified address returns; yis far "raddr; /* Returns data read from the device at specified address returns; * Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this r' case the target address)	define	SYSADDR	0	/* This constant can be initialized to any address within the memory map	•
ti rdimode() byte far "tempeddir; /* Pointer veriable used to write commands to the device tempeddr = (byte far')SYSADDR; "tempeddr = RDARRAY; /* Write Read Array command to LH28F008SA returm; Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; raddr; Target address to be read from Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command tr rdbyte(rdata, raddr) yte *rdata; /* Returns data read from the device at specified address yte far 'raddr; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)				/* of the target LH28F008SA and is alterable depending on the	1
byte far "tempeddi; /* Pointer variable used to write commands to the device tempeddr itempeddr = (byte far")SYSADDR; "tempeddr = RDARRAY; /* Write Reed Array command to LH28F008SA return; Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program inputs; raddress to be reed from Outputs; Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array vdefine RDARRAY Not FF /* Reed array command vit aftria; /* Returns data read from the device at specified address return; vite array; /* Returns data read from the device at specified address return; vite far "raddr; /* Returns data read from the device at specified address return;				/* system architecture	1
tempeddr = (byte far")SYSADDR; "tempeddr = RDARRAY; "tempeddr = RDARRAY; "return; * Write Read Array command to LH28F008SA Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command tr rdbyte(rdata, raddr) /* Returns data read from the device at specified address yte *rdata; /* Returns data read from the device at specified address tr rdbyte(rdata, raddr) /* Returns data read from the device at specified address yte far "raddr; /* Returns data read from the device at specified address 'readr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)	nť rdmode	0			
tempeddr = (byte far")SYSADDR; "tempeddr = RDARRAY; "tempeddr = RDARRAY; "return; * Write Read Array command to LH28F008SA Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command tr rdbyte(rdata, raddr) /* Returns data read from the device at specified address yte *rdata; /* Returns data read from the device at specified address tr rdbyte(rdata, raddr) /* Returns data read from the device at specified address yte far "raddr; /* Returns data read from the device at specified address 'readr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)					
"tempeddr = RDARRAY; /* Write Read Array command to LH28F008SA return; Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; rddta; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Reed array command tr rdbyte(rdata, raddr) yite *rdata; /* Returns data read from the device at specified address yite arraddr; /* Returns data read from the device at specified address *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this *raddr = RDARRAY; /* Write read array command to an address be read from		•	• •	/* Pointer variable used to write commands to the device	
return; Function; Rdbyte Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command t rdbyte(rdata, raddr) yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Returns data read from the device at specified address *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)		•	•••		
Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command t rdbyte(rdata, raddr) /te *rdata; /* Returns data read from the device at specified address /te far *raddr; /* Returns data read from the device at specified address *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)			= RDARRAY;	/* Write Read Array command to LH28F008SA	
Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command tt rdbyte(rdata, raddr) yte *rdata; /* Returns data read from the device at specified address yte ar *raddr; /* Returns data read from the device at specified address *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)					
Description; Reads a byte of data from a specified address and returns it to the calling program Inputs; raddr; Target address to be read from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command tt rdbyte(rdata, raddr) yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Returns data read from the device at specified address *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)	• • • • • • • •	· · · · · · · · · · · · · · · · ·	*****************	•••••••••••••••••••••••••••••••••••••••	
Inputs; raddr; Target address to be read from Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command tt rdbyte(rdata, raddr) yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Returns data read from the device at specified address yte far *raddr; /* Returns data read from the device at specified address '* Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)			•		•
Outputs; rdata; Date at the specified address Returns; Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command tr dbyte(rdata, raddr) yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)					
Returns: Nothing Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command it rdbyte(rdata, raddr) yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Returns data read from the device at specified address /* Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)			· •		
Device Read Mode on Return; Array define RDARRAY 0XFF /* Read array command nt rdbyte(rdata, raddr) /* Returns data read from the device at specified address yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)		• •		d address	
define RDARRAY 0XFF /* Read array command tr dbyte(rdata, raddr) /* Returns data read from the device at specified address yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Returns data read from the device at specified address *raddr = RDARRAY; /* Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)			•		
vite robyte(rdata, raddr) /* Returns data read from the device at specified address yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)	•••••		Mode on Heturn; Array		• • • • •
yte *rdata; /* Returns data read from the device at specified address yte far *raddr; /* Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)	define	RDARRAY	0XFF	/* Read array command	•
yte far *raddr; /* Raddr is the target address to be read from *raddr = RDARRAY; /* Write read array command to an address within the LH28F008SA (in this /* case the target address)	nt rdbyte(rd	data, raddr)		·	
raddr = RDARRAY; / Write read array command to an address within the LH28F008SA (in this /* case the target address)	.			•	•
/* case the target address)	yte far "ra	aar;		/" Haddr is the target address to be read from	•
/* case the target address)		- ··		· · · · · · · · · · · · · · · · · · ·	
- · ·		"raddr	= RDARRAY;		
rdata = 'raddr: /' Read from the specified address and store					
· · · · · · · · · · · · · · · · · · ·			= "raddr;	/* Read from the specified address and store	•
retum;			= "raddr;	/* Read from the specified address and store	

Routine rdmode() simply puts a LH28F008SA in Read Array mode. This is useful after byte write and block erase operations, to return the LH28F008SA to its "normal" mode of operation. After block erase or byte write, the LH28F008SA will continue to output Status Register data until the Read Array command is issued to it, for example.

Routine rdbyte() not only puts the LH28F008SA in Read Array mode, it also reads a byte of data. It is passed the desired system byte address, and passes back the data at that address.

}

•	Function; Vpp	un		*/
	Description: P	home the Vee supply to the t	arget LH28F008SA to enable byte write or block erase. This routine can be	•/
	Description, n	toiloand to the individual syst	tem architecture. For purposes of this example, I assumed that a system	•/
			system address 20000 hex, with the following definitions;	•/
				•/
		Bit 7; Vpph C	0 = Disabled	•/
				•
		Bit 6; PWD C	0 = PowerDown Disabled	•
		-		•
		Bits 5-0; Und		
				•
	inputs;	None		•
,	Outputs;	None		-
	Returns;	Nothing		
••••	Device Read	Mode on Return; As existed t	before entering the function. Part is now ready for program or erase	• • • • • •
define	VPPHIGH	0X80	/* Bit 7 = 1, Vpp elevated to Vpph	•
define	SYSCADDR	0X20000	Assumed system Control Register Address	•
it vppup()				
	byte far *cont	addr;	/* Pointer variable used to write data to the system Control Register	
	contaddr	= (byte far *)SYSCADDR;		
	"contaddr return;	= "contaddr 1 VPPHIGH;	/* Read current Control Register data, "OR" with constant to ramp Vpp	
•••••				
	Function: Vpr	down		•
,	Description: [Ramps down the Voo supply i	to the target LH28F008SA to disable byte write/block erase.	•
,	Description, i		of the assumed system Control Register.	•
	inputs;	None		
	Outputs;	None		
,		Nothing		•
-	Returns;		before entering the function. Part now has high Vpp disabled. If byte write or	
-	Device Kead	NUGE ON RELIN, AS EXISTED	s when this function was called, it will complete unsuccessfully with Vpp low error	
-			אוסו עוש ועויטערו אפי כמוסט, ון אוו כטווקופנט פווטנטטטטטער אין ווער דיך וכו כווטי	•
	• • • • • • • • • • • •	in the Status Register.	• • • • • • • • • • • • • • • • • • • •	
klefine	VPPDWN	0X7F	/* Bit 7 = 0, Vpp lowered to Vppl	•
define	SYSCADDR	0X20000	/* Assumed system Control Register Address	
nt vppdown	()			
	h. da fa a t	te data i	/* Pointer variable used to write data to the system Control Register	
	byte far *coni contaddr	= (byte far *)SYSCADDR;	/ Funter variable used to write data to are system senser register	
	"contaddr	= (byte fair) SYSCADDR, = "contaddr & VPPDWN;	/* Read current Control Register data, "AND" with constant to lower Vpp	

Functions vppup() and vppdown() give examples of how to control via software the hardware that enables or disables $12V V_{pp}$ to a LH28F008SA. The actual hardware implementation chosen will drive any modification of these routines.

	Function; Pwd	ien		•/	
,	Description; Toggles the LH28F008SA/PWD pin low to put the device in Deep PowerDown mode.				
•			of the assumed system Control Register.	•	
,	inputs;	None	······································	•	
	Outputs;	None		•	
	Returns;	Nothing	-	•	
•			powered down. If byte write or block erase was in progress when this function	•,	
,		was called it will short with	resulting partially written or erased data. Recovery in the form of repeat of	•	
			I be required once the part transitions out of Powerdown,	•	
		to initialize data to a known		•	
•••••			31210. • • • • • • • • • • • • • • • • • • •		
	DWD	0X40	/* Bit 6 = 1, /PWD enabled	•	
ideline ideline	PWD SYSCADDR	0X20000	/* Assumed system Control Register Address	•	
nt pwden()	31304001				
	byte far*conta	kddr;	/* Pointer variable used to write data to the system Control Register	•	
	contaddr	= (byte far *)SYSCADDR;			
	*contaddr	= "contaddr 1 PWD;	/" Reed current Control Register data, "OR" with constant to enable /" Deep PoweDown		
	retum;				
)				·····	

•	Function; Pw	ddis		•	
•	Description: 1	Locales the LH28F008SA/PW	/D pin high to transition the part out of Deep Powerdown.	•	
•	÷,		of the assumed system Control Register.	1	
	Inputs;	None	· · ·		
	Outputs;	None		•	
	Returns;	Nothing		•	
	newna,	Mode on Return: Read Array	mode. Low voltage is removed from the /PWD pin. LH28F008SA output		
			me tPHQV after the /PWD pin transitions high (reference the datasheet AC Read	1	
•	Device Head	size will output voild data th	and a rider ditor bio is the pirt bandhadine ingri (interested and		
	Device Head	pins will output valid data ti	plid states on all other control and power supply Dins.		
•		pins will output valid data ti	valid states on all other control and power supply pins.		
• • •		pins will output valid data ti Characteristics) assuming v	ralid states on all other control and power supply pins.		
	PDOFF	pins will output valid data ti Characteristics) assuming v 0XBF	/* Bit 6 = 0, /PWD disabled	••••	
Idefine		pins will output valid data ti Characteristics) assuming v 0XBF	ralid states on all other control and power supply pins.	•••••	
idefine nt pwddis()	PDOFF SYSCADDR	pins will output valid data ti Characteristics) assuming v 0XBF 0X20000	/alid states on all other control and power supply pins. /* Bit 6 = 0, /PWD disabled /* Assumed system Control Register Address	•••••	
idefine Idefine Idefine nt pwddis()	PDOFF SYSCADDR byte far *cont	pins will output valid data ti Characteristics) assuming v 0XBF 0X20000	/* Bit 6 = 0, /PWD disabled		
idefine nt pwddis()	PDOFF SYSCADDR byte far *cont contaddr	pins will output valid data ti Characteristics) assuming v 0XBF 0X20000 taddr; = (byte far *)SYSCADDR;	/* Bit 6 = 0, /PWD disabled /* Assumed system Control Register Address /* pointer variable used to write data to the system Control register	• • • • • •	
idefine nt pwddis()	PDOFF SYSCADDR byte far *cont	pins will output valid data ti Characteristics) assuming v 0XBF 0X20000	/alid states on all other control and power supply pins. /* Bit 6 = 0, /PWD disabled /* Assumed system Control Register Address		

Functions pwden() and pwddis() give examples of how to control via software the hardware that enables or disables a LH28F008SA PWD input. The actual hardware implementation chosen will drive any modification of these routines.

8 Mb, Flash, Memory Software Driver, LH28F008SA