Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	HCS12
Core Size	16-Bit
Speed	25MHz
Connectivity	EBI/EMI, SCI, SPI
Peripherals	POR, PWM, WDT
Number of I/O	60
Program Memory Size	96KB (96K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	4K x 8
Voltage - Supply (Vcc/Vdd)	2.35V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 105°C (TA)
Mounting Type	Surface Mount
Package / Case	80-QFP
Supplier Device Package	80-QFP (14x14)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/mc9s12gc96vfue

Chapter 3
Module Mapping Control (MMCV4) Block Description

Chapter 4
Multiplexed External Bus Interface (MEBIV3)

1.4 System Clock Description

The clock and reset generator provides the internal clock signals for the core and all peripheral modules. Figure 1-14 shows the clock connections from the CRG to all modules. Consult the CRG Block User Guide for details on clock generation.

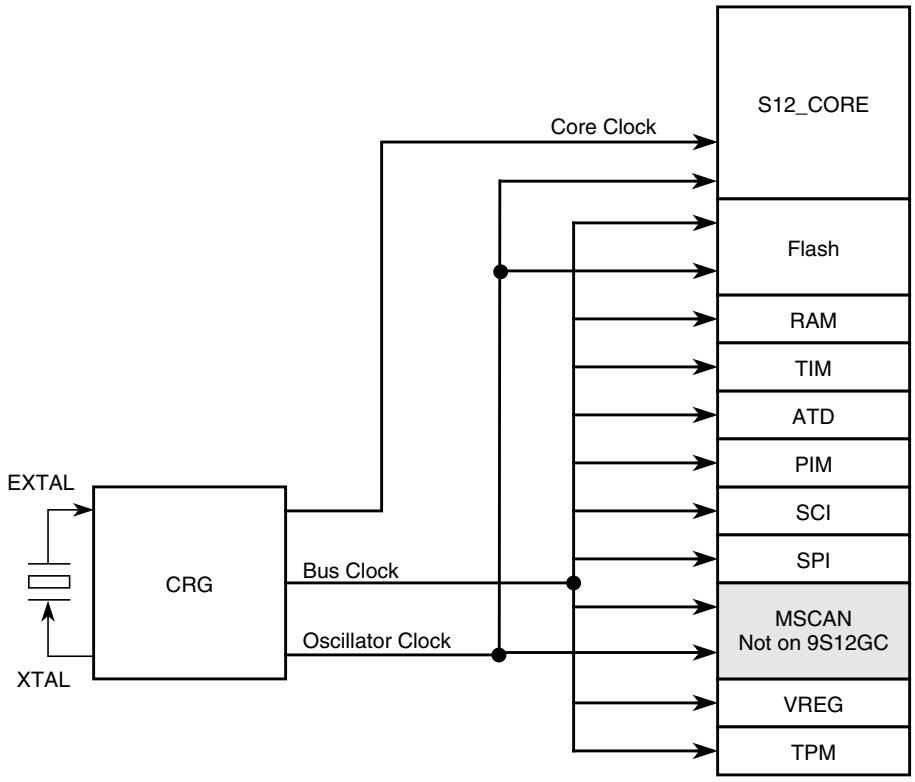


Figure 1-14. Clock Connections

1.5 Modes of Operation

Eight possible modes determine the device operating configuration. Each mode has an associated default memory map and external bus configuration controlled by a further pin.

Three low power modes exist for the device.

1.5.1 Chip Configuration Summary

The operating mode out of reset is determined by the states of the MODC, MODB, and MODA pins during reset. The MODC, MODB, and MODA bits in the MODE register show the current operating mode and provide limited mode switching during operation. The states of the MODC, MODB, and MODA pins are latched into these bits on the rising edge of the reset signal. The ROMCTL signal allows the setting of the ROMON bit in the MISC register thus controlling whether the internal Flash is visible in the memory map. ROMON = 1 mean the Flash is visible in the memory map. The state of the ROMCTL pin is latched into the ROMON bit in the MISC register on the rising edge of the reset signal.

2.1.2 Block Diagram

Figure 2-1 is a block diagram of the PIM.

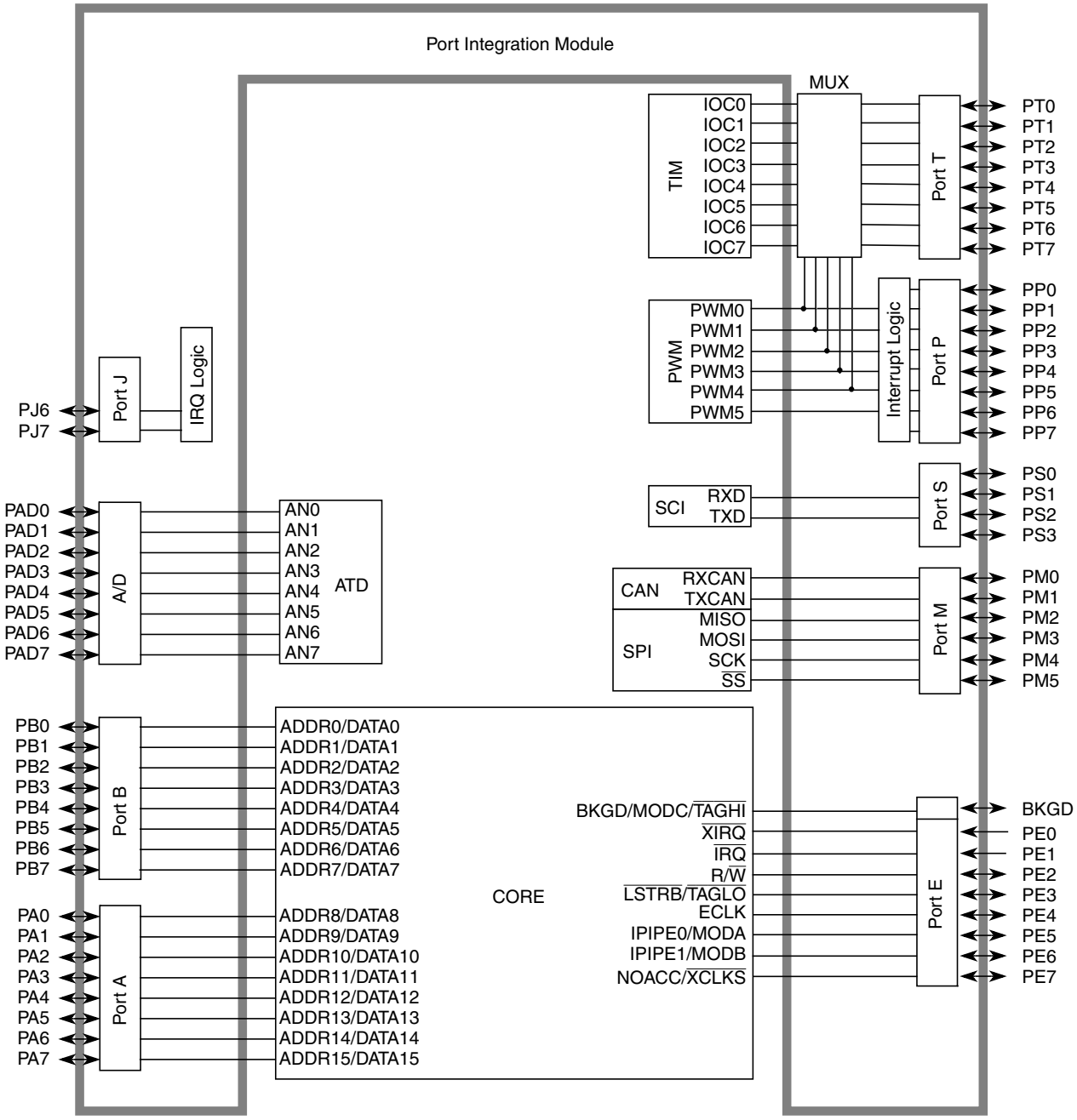


Figure 2-1. PIM Block Diagram

Note: The MODRR register within the PIM allows for mapping of PWM channels to Port T in the absence of Port P pins for the low pin count packages. For the 80QFP package option it is recommended not to use MODRR since this is intended to support PWM channel availability in low pin count packages. Note that

The PPAGE register holds the page select value for the program page window. The value of the PPAGE register can be manipulated by normal read and write (some devices don't allow writes in some modes) instructions as well as the CALL and RTC instructions.

Control registers, vector spaces, and a portion of on-chip memory are located in unpaged portions of the 64K byte physical address space. The stack and I/O addresses should also be in unpaged memory to make them accessible from any page.

The starting address of a service routine must be located in unpaged memory because the 16-bit exception vectors cannot point to addresses in paged memory. However, a service routine can call other routines that are in paged memory. The upper 16K byte block of memory space (0xC000–0xFFFF) is unpaged. It is recommended that all reset and interrupt vectors point to locations in this area.

3.4.3.1 CALL and Return from Call Instructions

CALL and RTC are uninterruptable instructions that automate page switching in the program expansion window. CALL is similar to a JSR instruction, but the subroutine that is called can be located anywhere in the normal 64K byte address space or on any page of program expansion memory. CALL calculates and stacks a return address, stacks the current PPAGE value, and writes a new instruction-supplied value to PPAGE. The PPAGE value controls which of the 64 possible pages is visible through the 16K byte expansion window in the 64K byte memory map. Execution then begins at the address of the called subroutine.

During the execution of a CALL instruction, the CPU:

- Writes the old PPAGE value into an internal temporary register and writes the new instruction-supplied PPAGE value into the PPAGE register.
- Calculates the address of the next instruction after the CALL instruction (the return address), and pushes this 16-bit value onto the stack.
- Pushes the old PPAGE value onto the stack.
- Calculates the effective address of the subroutine, refills the queue, and begins execution at the new address on the selected page of the expansion window.

This sequence is uninterruptable; there is no need to inhibit interrupts during CALL execution. A CALL can be performed from any address in memory to any other address.

The PPAGE value supplied by the instruction is part of the effective address. For all addressing mode variations except indexed-indirect modes, the new page value is provided by an immediate operand in the instruction. In indexed-indirect variations of CALL, a pointer specifies memory locations where the new page value and the address of the called subroutine are stored. Using indirect addressing for both the new page value and the address within the page allows values calculated at run time rather than immediate values that must be known at the time of assembly.

The RTC instruction terminates subroutines invoked by a CALL instruction. RTC unstacks the PPAGE value and the return address and refills the queue. Execution resumes with the next instruction after the CALL.

NOTE

To ensure that you read the value present on the PORTE pins, always wait at least one cycle after writing to the DDRE register before reading from the PORTE register.

4.3.2.7 Data Direction Register E (DDRE)

Module Base + 0x0009
Starting address location affected by INITRG register setting.

	7	6	5	4	3	2	1	0
R	Bit 7	6	5	4	3	Bit 2	0	0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 4-11. Data Direction Register E (DDRE)

Read: Anytime when register is in the map

Write: Anytime when register is in the map

Data direction register E is associated with port E. For bits in port E that are configured as general-purpose I/O lines, DDRE determines the primary direction of each of these pins. A 1 causes the associated bit to be an output and a 0 causes the associated bit to be an input. Port E bit 1 (associated with $\overline{\text{IRQ}}$) and bit 0 (associated with $\overline{\text{XIRQ}}$) cannot be configured as outputs. Port E, bits 1 and 0, can be read regardless of whether the alternate interrupt function is enabled. The value in a DDR bit also affects the source of data for reads of the corresponding PORTE register. If the DDR bit is 0 (input) the buffered pin input state is read. If the DDR bit is 1 (output) the associated port data register bit state is read.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally. Also, it is not in the map in expanded modes while the EME control bit is set.

Table 4-5. DDRE Field Descriptions

Field	Description
7:2 DDRE	**Data Direction Port E** 0 Configure the corresponding I/O pin as an input 1 Configure the corresponding I/O pin as an output **Note:** It is unwise to write PORTE and DDRE as a word access. If you are changing port E pins from inputs to outputs, the data may have extra transitions during the write. It is best to initialize PORTE before enabling as outputs.

Table 6-2. BDMSTS Field Descriptions (continued)

Field	Description
2 CLKSW	**Clock Switch** — The CLKSW bit controls which clock the BDM operates with. It is only writable from a hardware BDM command. A 150 cycle delay at the clock speed that is active during the data portion of the command will occur before the new clock source is guaranteed to be active. The start of the next BDM command uses the new clock for timing subsequent BDM communications. Table 6-3 shows the resulting BDM clock source based on the CLKSW and the PLLSEL (Pll select from the clock and reset generator) bits. **Note:** The BDM alternate clock source can only be selected when CLKSW = 0 and PLLSEL = 1. The BDM serial interface is now fully synchronized to the alternate clock source, when enabled. This eliminates frequency restriction on the alternate clock which was required on previous versions. Refer to the device overview section to determine which clock connects to the alternate clock source input. **Note:** If the acknowledge function is turned on, changing the CLKSW bit will cause the ACK to be at the new rate for the write command which changes it.
1 UNSEC	**Unsecure** — This bit is only writable in special single-chip mode from the BDM secure firmware and always gets reset to zero. It is in a zero state as secure mode is entered so that the secure BDM firmware lookup table is enabled and put into the memory map along with the standard BDM firmware lookup table. The secure BDM firmware lookup table verifies that the on-chip EEPROM and FLASH EEPROM are erased. This being the case, the UNSEC bit is set and the BDM program jumps to the start of the standard BDM firmware lookup table and the secure BDM firmware lookup table is turned off. If the erase test fails, the UNSEC bit will not be asserted. 0 System is in a secured mode 1 System is in a unsecured mode **Note:** When UNSEC is set, security is off and the user can change the state of the secure bits in the on-chip FLASH EEPROM. Note that if the user does not change the state of the bits to “unsecured” mode, the system will be secured again when it is next taken out of reset.

Table 6-3. BDM Clock Sources

PLLSEL	CLKSW	BDMCLK
0	0	Bus clock
0	1	Bus clock
1	0	Alternate clock (refer to the device overview chapter to determine the alternate clock source)
1	1	Bus clock dependent on the PLL

NOTE

BDM should not be entered from a breakpoint unless the ENABLE bit is set in the BDM. Even if the ENABLE bit in the BDM is cleared, the CPU actually executes the BDM firmware code. It checks the ENABLE and returns if ENABLE is not set. If the BDM is not serviced by the monitor then the breakpoint would be re-asserted when the BDM returns to normal CPU flow.

There is no hardware to enforce restriction of breakpoint operation if the BDM is not enabled.

When program control returns from a tagged breakpoint through an RTI or a BDM GO command, it will return to the instruction whose tag generated the breakpoint. Unless breakpoints are disabled or modified in the service routine or active BDM session, the instruction will be tagged again and the breakpoint will be repeated. In the case of BDM breakpoints, this situation can also be avoided by executing a TRACE1 command before the GO to increment the program flow past the tagged instruction.

7.4.1.4 Using Comparator C in BKP Mode

The original BKP_ST12_A module supports two breakpoints. The DBG_ST12_A module can be used in BKP mode and allow a third breakpoint using comparator C. Four additional bits, BKCEN, TAGC, RWCEN, and RWC in DBGC2 in conjunction with additional comparator C address registers, DBGCCX, DBGCCH, and DBGCCL allow the user to set up a third breakpoint. Using PAGSEL in DBGCCX for expanded memory will work differently than the way paged memory is done using comparator A and B in BKP mode. See Section 7.3.2.5, “Debug Comparator C Extended Register (DBGCCX),” for more information on using comparator C.

7.4.2 DBG Operating in DBG Mode

Enabling the DBG module in DBG mode, allows the arming, triggering, and storing of data in the trace buffer and can be used to cause CPU breakpoints. The DBG module is made up of three main blocks, the comparators, trace buffer control logic, and the trace buffer.

NOTE

In general, there is a latency between the triggering event appearing on the bus and being detected by the DBG circuitry. In general, tagged triggers will be more predictable than forced triggers.

7.4.2.1 Comparators

The DBG contains three comparators, A, B, and C. Comparator A compares the core address bus with the address stored in DBGCAH and DBGCAL. Comparator B compares the core address bus with the address stored in DBGCBH and DBGCBL except in full mode, where it compares the data buses to the data stored in DBGCBH and DBGCBL. Comparator C can be used as a breakpoint generator or as the address comparison unit in the loop1 mode. Matches on comparator A, B, and C are signaled to the trace buffer

the trigger is at the address of a change-of-flow address the trigger event will not be stored in the trace buffer.

7.4.2.9 Reading Data from Trace Buffer

The data stored in the trace buffer can be read using either the background debug module (BDM) module or the CPU provided the DBG module is enabled and not armed. The trace buffer data is read out first-in first-out. By reading CNT in DBGCNT the number of valid words can be determined. CNT will not decrement as data is read from DBGTBH:DBGTBL. The trace buffer data is read by reading DBGTBH:DBGTBL with a 16-bit read. Each time DBGTBH:DBGTBL is read, a pointer in the DBG will be incremented to allow reading of the next word.

Reading the trace buffer while the DBG module is armed will return invalid data and no shifting of the RAM pointer will occur.

NOTE

The trace buffer should be read with the DBG module enabled and in the same capture mode that the data was recorded. The contents of the trace buffer counter register (DBGCNT) are resolved differently in detail mode verses the other modes and may lead to incorrect interpretation of the trace buffer data.

7.4.3 Breakpoints

There are two ways of getting a breakpoint in DBG mode. One is based on the trigger condition of the trigger mode using comparator A and/or B, and the other is using comparator C. External breakpoints generated using the $\overline{\text{TAGHI}}$ and $\overline{\text{TAGLO}}$ external pins are disabled in DBG mode.

7.4.3.1 Breakpoint Based on Comparator A and B

A breakpoint request to the CPU can be enabled by setting DBGBRK in DBGC1. The value of BEGIN in DBGC1 determines when the breakpoint request to the CPU will occur. When BEGIN in DBGC1 is set, begin-trigger is selected and the breakpoint request will not occur until the trace buffer is filled with 64 words. When BEGIN in DBGC1 is cleared, end-trigger is selected and the breakpoint request will occur immediately at the trigger cycle.

There are two types of breakpoint requests supported by the DBG module, tagged and forced. Tagged breakpoints are associated with opcode addresses and allow breaking just before a specific instruction executes. Forced breakpoints are not associated with opcode addresses and allow breaking at the next instruction boundary. The type of breakpoint based on comparators A and B is determined by TRGSEL in the DBGC1 register (TRGSEL = 1 for tagged breakpoint, TRGSEL = 0 for forced breakpoint). Table 7-26 illustrates the type of breakpoint that will occur based on the debug run.

9.3.2.4 CRG Flags Register (CRGFLG)

This register provides CRG status bits and flags.

Module Base + 0x0003

	7	6	5	4	3	2	1	0
R	RTIF	PORF	LVRF	LOCKIF	LOCK	TRACK	SCMIF	SCM
W								
Reset	0	Note 1	Note 2	0	0	0	0	0

1. PORF is set to 1 when a power-on reset occurs. Unaffected by system reset.
2. LVRF is set to 1 when a low-voltage reset occurs. Unaffected by system reset.

= Unimplemented or Reserved

Figure 9-7. CRG Flag Register (CRGFLG)

Read: anytime

Write: refer to each bit for individual write conditions

Table 9-2. CRGFLG Field Descriptions

Field	Description
7 RTIF	**Real-Time Interrupt Flag** — RTIF is set to 1 at the end of the RTI period. This flag can only be cleared by writing a 1. Writing a 0 has no effect. If enabled (RTIE = 1), RTIF causes an interrupt request. 0 RTI time-out has not yet occurred. 1 RTI time-out has occurred.
6 PORF	**Power-on Reset Flag** — PORF is set to 1 when a power-on reset occurs. This flag can only be cleared by writing a 1. Writing a 0 has no effect. 0 Power-on reset has not occurred. 1 Power-on reset has occurred.
5 LVRF	**Low-Voltage Reset Flag** — If low voltage reset feature is not available (see the device overview chapter), LVRF always reads 0. LVRF is set to 1 when a low voltage reset occurs. This flag can only be cleared by writing a 1. Writing a 0 has no effect. 0 Low voltage reset has not occurred. 1 Low voltage reset has occurred.
4 LOCKIF	**PLL Lock Interrupt Flag** — LOCKIF is set to 1 when LOCK status bit changes. This flag can only be cleared by writing a 1. Writing a 0 has no effect.If enabled (LOCKIE = 1), LOCKIF causes an interrupt request. 0 No change in LOCK bit. 1 LOCK bit has changed.
3 LOCK	**Lock Status Bit** — LOCK reflects the current state of PLL lock condition. This bit is cleared in self-clock mode. Writes have no effect. 0 PLL VCO is not within the desired tolerance of the target frequency. 1 PLL VCO is within the desired tolerance of the target frequency.
2 TRACK	**Track Status Bit** — TRACK reflects the current state of PLL track condition. This bit is cleared in self-clock mode. Writes have no effect. 0 Acquisition mode status. 1 Tracking mode status.

NOTE

The CANCTL0 register, except WUPE, INITRQ, and SLPRQ, is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable again as soon as the initialization mode is exited (INITRQ = 0 and INITAK = 0).

Read: Anytime

Write: Anytime when out of initialization mode; exceptions are read-only RXACT and SYNCH, RXFRM (which is set by the module only), and INITRQ (which is also writable in initialization mode).

Table 10-1. CANCTL0 Register Field Descriptions

Field	Description
7 RXFRM(1)	**Received Frame Flag** — This bit is read and clear only. It is set when a receiver has received a valid message correctly, independently of the filter configuration. After it is set, it remains set until cleared by software or reset. Clearing is done by writing a 1. Writing a 0 is ignored. This bit is not valid in loopback mode. 0 No valid message was received since last clearing this flag 1 A valid message was received since last clearing of this flag
6 RXACT	**Receiver Active Status** — This read-only flag indicates the MSCAN is receiving a message. The flag is controlled by the receiver front end. This bit is not valid in loopback mode. 0 MSCAN is transmitting or idle[2] 1 MSCAN is receiving a message (including when arbitration is lost)(2)
5 CSWAI(3)	**CAN Stops in Wait Mode** — Enabling this bit allows for lower power consumption in wait mode by disabling all the clocks at the CPU bus interface to the MSCAN module. 0 The module is not affected during wait mode 1 The module ceases to be clocked during wait mode
4 SYNCH	**Synchronized Status** — This read-only flag indicates whether the MSCAN is synchronized to the CAN bus and able to participate in the communication process. It is set and cleared by the MSCAN. 0 MSCAN is not synchronized to the CAN bus 1 MSCAN is synchronized to the CAN bus
3 TIME	**Timer Enable** — This bit activates an internal 16-bit wide free running timer which is clocked by the bit clock rate. If the timer is enabled, a 16-bit time stamp will be assigned to each transmitted/received message within the active TX/RX buffer. Right after the EOF of a valid message on the CAN bus, the time stamp is written to the highest bytes (0x000E, 0x000F) in the appropriate buffer (see Section 10.3.3, "Programmer's Model of Message Storage"). The internal timer is reset (all bits set to 0) when disabled. This bit is held low in initialization mode. 0 Disable internal MSCAN timer 1 Enable internal MSCAN timer
2 WUPE(4)	**Wake-Up Enable** — This configuration bit allows the MSCAN to restart from sleep mode when traffic on CAN is detected (see Section 10.4.5.4, "MSCAN Sleep Mode"). 0 Wake-up disabled — The MSCAN ignores traffic on CAN 1 Wake-up enabled — The MSCAN is able to restart

10.3.2.3 MSCAN Bus Timing Register 0 (CANBTR0)

The CANBTR0 register configures various CAN bus timing parameters of the MSCAN module.

Module Base + 0x0002

	7	6	5	4	3	2	1	0
R W	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
Reset:	0	0	0	0	0	0	0	0

Figure 10-6. MSCAN Bus Timing Register 0 (CANBTR0)

Read: Anytime
Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

Table 10-3. CANBTR0 Register Field Descriptions

Field	Description
7:6 SJW[1:0]	**Synchronization Jump Width** — The synchronization jump width defines the maximum number of time quanta (Tq) clock cycles a bit can be shortened or lengthened to achieve resynchronization to data transitions on the CAN bus (see Table 10-4).
5:0 BRP[5:0]	**Baud Rate Prescaler** — These bits determine the time quanta (Tq) clock which is used to build up the bit timing (see Table 10-5).

Table 10-4. Synchronization Jump Width

SJW1	SJW0	Synchronization Jump Width
0	0	1 Tq clock cycle
0	1	2 Tq clock cycles
1	0	3 Tq clock cycles
1	1	4 Tq clock cycles

Table 10-5. Baud Rate Prescaler

BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	Prescaler value (P)
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
:	:	:	:	:	:	:
1	1	1	1	1	1	64

Read: Anytime when TXEx flag is set (see Section 10.3.2.7, "MSCAN Transmitter Flag Register (CANTFLG)") and the corresponding transmit buffer is selected in CANTBSEL (see Section 10.3.2.11, "MSCAN Transmit Buffer Selection Register (CANTBSEL)").

Write: Unimplemented

10.4 Functional Description

10.4.1 General

This section provides a complete functional description of the MSCAN. It describes each of the features and modes listed in the introduction.

The MSCAN then schedules the message for transmission and signals the successful transmission of the buffer by setting the associated TXE flag. A transmit interrupt (see Section 10.4.7.2, "Transmit Interrupt") is generated[1] when TXEx is set and can be used to drive the application software to re-load the buffer.

If more than one buffer is scheduled for transmission when the CAN bus becomes available for arbitration, the MSCAN uses the local priority setting of the three buffers to determine the prioritization. For this purpose, every transmit buffer has an 8-bit local priority field (PRIO). The application software programs this field when the message is set up. The local priority reflects the priority of this particular message relative to the set of messages being transmitted from this node. The lowest binary value of the PRIO field is defined to be the highest priority. The internal scheduling process takes place whenever the MSCAN arbitrates for the CAN bus. This is also the case after the occurrence of a transmission error.

When a high priority message is scheduled by the application software, it may become necessary to abort a lower priority message in one of the three transmit buffers. Because messages that are already in transmission cannot be aborted, the user must request the abort by setting the corresponding abort request bit (ABTRQ) (see Section 10.3.2.9, "MSCAN Transmitter Message Abort Request Register (CANTARQ)".) The MSCAN then grants the request, if possible, by:

1. Setting the corresponding abort acknowledge flag (ABTAK) in the CANTAAK register.
2. Setting the associated TXE flag to release the buffer.
3. Generating a transmit interrupt. The transmit interrupt handler software can determine from the setting of the ABTAK flag whether the message was aborted (ABTAK = 1) or sent (ABTAK = 0).

10.4.2.3 Receive Structures

The received messages are stored in a five stage input FIFO. The five message buffers are alternately mapped into a single memory area (see Figure 10-38). The background receive buffer (RxBG) is exclusively associated with the MSCAN, but the foreground receive buffer (RxFG) is addressable by the CPU (see Figure 10-38). This scheme simplifies the handler software because only one address area is applicable for the receive process.

All receive buffers have a size of 15 bytes to store the CAN control bits, the identifier (standard or extended), the data contents, and a time stamp, if enabled (see Section 10.3.3, "Programmer's Model of Message Storage").

The receiver full flag (RXF) (see Section 10.3.2.5, "MSCAN Receiver Flag Register (CANRFLG)") signals the status of the foreground receive buffer. When the buffer contains a correctly received message with a matching identifier, this flag is set.

On reception, each message is checked to see whether it passes the filter (see Section 10.4.3, "Identifier Acceptance Filter") and simultaneously is written into the active RxBG. After successful reception of a valid message, the MSCAN shifts the content of RxBG into the receiver FIFO[2], sets the RXF flag, and generates a receive interrupt (see Section 10.4.7.3, "Receive Interrupt") to the CPU[3]. The user's receive handler must read the received message from the RxFG and then reset the RXF flag to acknowledge the interrupt and to release the foreground buffer. A new message, which can follow immediately after the IFS field of the CAN frame, is received into the next available RxBG. If the MSCAN receives an invalid

1. The transmit interrupt occurs only if not masked. A polling scheme can be applied on TXEx also.
2. Only if the RXF flag is not set.
3. The receive interrupt occurs only if not masked. A polling scheme can be applied on RXF also.

The direction of each serial I/O pin depends on the BIDIROE bit. If the pin is configured as an output, serial data from the shift register is driven out on the pin. The same pin is also the serial input to the shift register.

The SCK is output for the master mode and input for the slave mode.

The $\overline{SS}$ is the input or output for the master mode, and it is always the input for the slave mode.

The bidirectional mode does not affect SCK and $\overline{SS}$ functions.

NOTE

In bidirectional master mode, with mode fault enabled, both data pins MISO and MOSI can be occupied by the SPI, though MOSI is normally used for transmissions in bidirectional mode and MISO is not used by the SPI. If a mode fault occurs, the SPI is automatically switched to slave mode, in this case MISO becomes occupied by the SPI and MOSI is not used. This has to be considered, if the MISO pin is used for other purpose.

14.4.6 Error Conditions

The SPI has one error condition:

- Mode fault error

14.4.6.1 Mode Fault Error

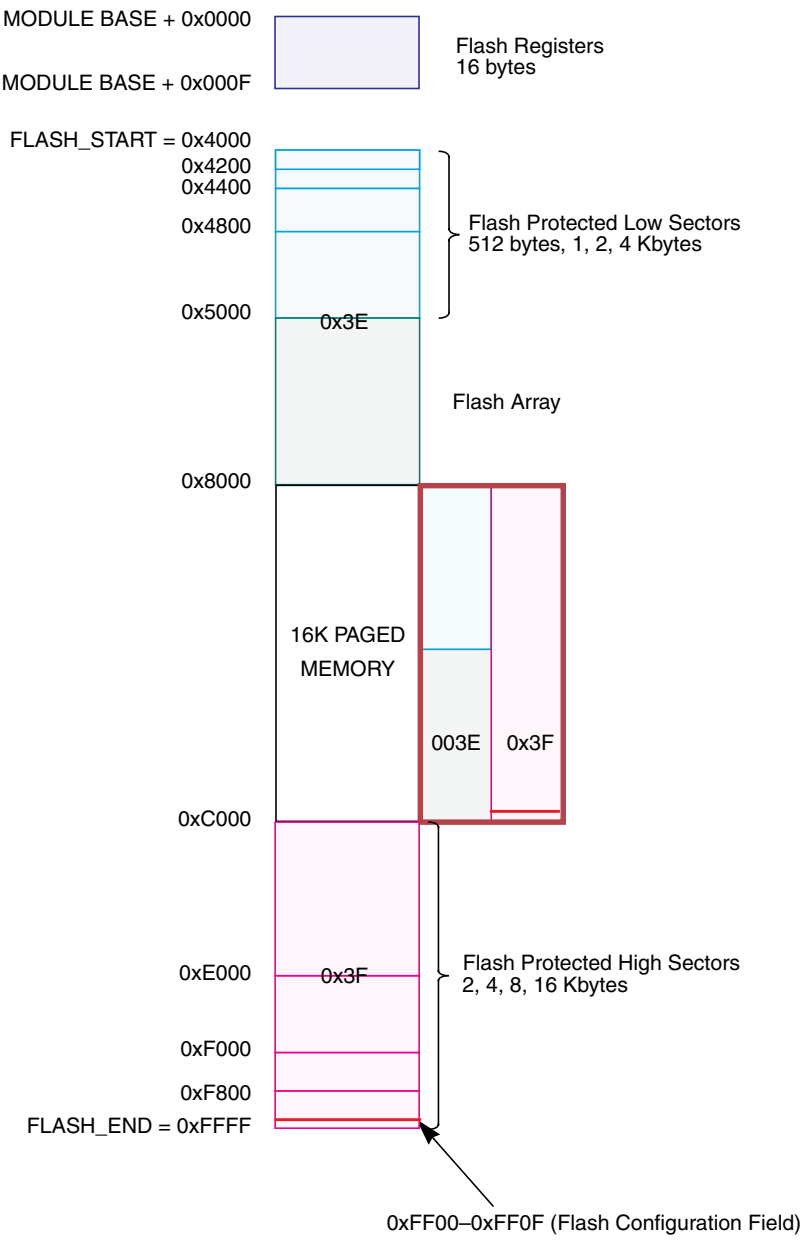
If the $\overline{SS}$ input becomes low while the SPI is configured as a master, it indicates a system error where more than one master may be trying to drive the MOSI and SCK lines simultaneously. This condition is not permitted in normal operation, the MODF bit in the SPI Status Register is set automatically provided the MODFEN bit is set.

In the special case where the SPI is in master mode and MODFEN bit is cleared, the $\overline{SS}$ pin is not used by the SPI. In this special case, the mode fault error function is inhibited and MODF remains cleared. In case the SPI system is configured as a slave, the $\overline{SS}$ pin is a dedicated input pin. Mode fault error doesn't occur in slave mode.

If a mode fault error occurs the SPI is switched to slave mode, with the exception that the slave output buffer is disabled. So SCK, MISO and MOSI pins are forced to be high impedance inputs to avoid any possibility of conflict with another output driver. A transmission in progress is aborted and the SPI is forced into idle state.

If the mode fault error occurs in the bidirectional mode for a SPI system configured in master mode, output enable of the MOMI (MOSI in bidirectional mode) is cleared if it was set. No mode fault error occurs in the bidirectional mode for SPI system configured in slave mode.

The mode fault flag is cleared automatically by a read of the SPI Status Register (with MODF set) followed by a write to SPI Control Register 1. If the mode fault flag is cleared, the SPI becomes a normal master or slave again.



Note: 0x3E–0x3F correspond to the PPAGE register content

Figure 18-2. Flash Memory Map

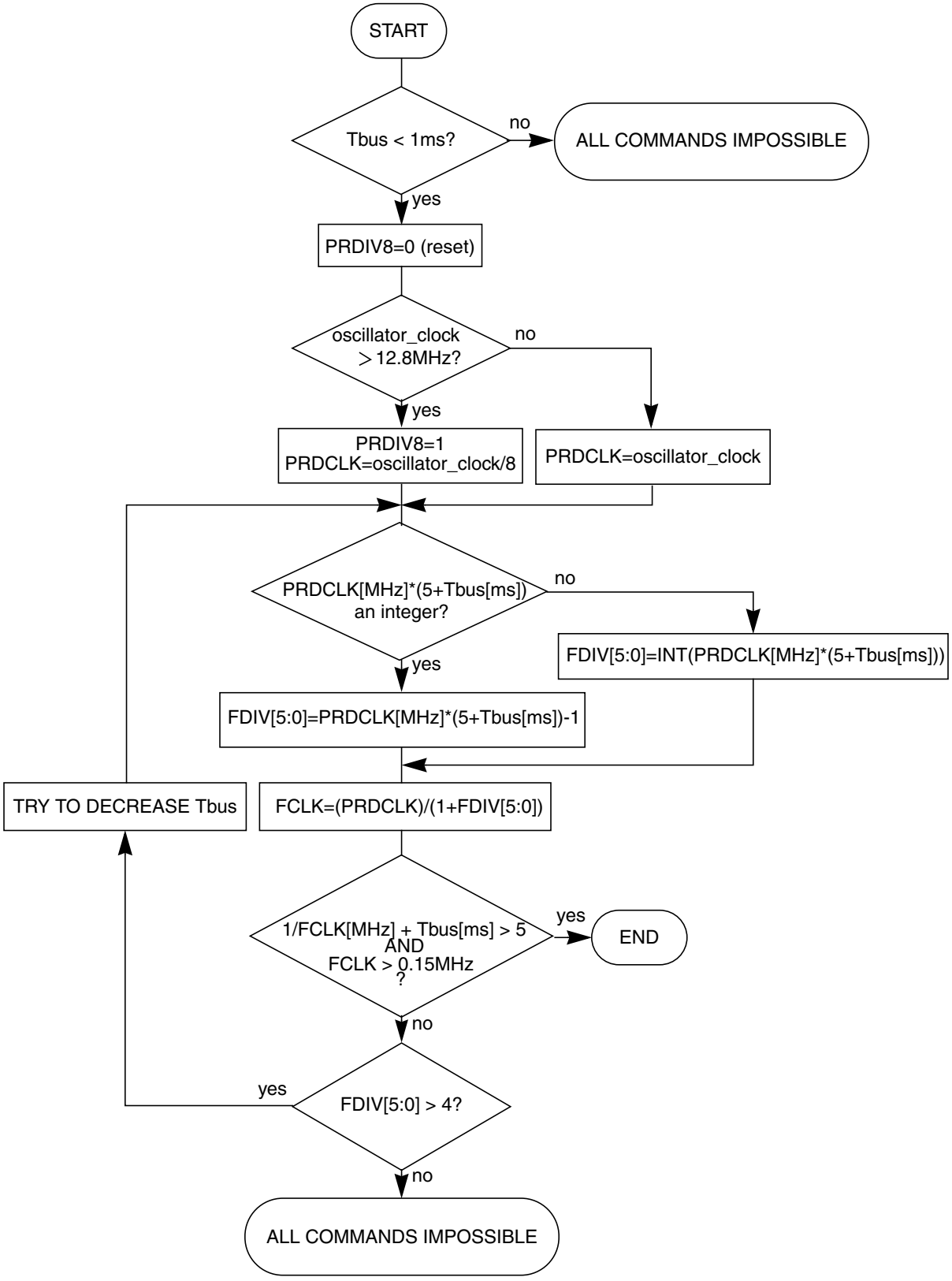


Figure 18-21. PRDIV8 and FDIV Bits Determination Procedure

18.4.1.2 Command Write Sequence

The Flash command controller is used to supervise the command write sequence to execute program, erase, and erase verify algorithms.

Before starting a command write sequence, the ACCERR and PVIOL flags in the FSTAT register must be clear and the CBEIF flag should be tested to determine the state of the address, data, and command buffers. If the CBEIF flag is set, indicating the buffers are empty, a new command write sequence can be started. If the CBEIF flag is clear, indicating the buffers are not available, a new command write sequence will overwrite the contents of the address, data, and command buffers.

A command write sequence consists of three steps which must be strictly adhered to with writes to the Flash module not permitted between the steps. However, Flash register and array reads are allowed during a command write sequence. The basic command write sequence is as follows:

1. Write to a valid address in the Flash array memory.
2. Write a valid command to the FCMD register.
3. Clear the CBEIF flag in the FSTAT register by writing a 1 to CBEIF to launch the command.

The address written in step 1 will be stored in the FADDR registers and the data will be stored in the FDATA registers. When the CBEIF flag is cleared in step 3, the CCIF flag is cleared by the Flash command controller indicating that the command was successfully launched. For all command write sequences, the CBEIF flag will set after the CCIF flag is cleared indicating that the address, data, and command buffers are ready for a new command write sequence to begin. A buffered command will wait for the active operation to be completed before being launched. Once a command is launched, the completion of the command operation is indicated by the setting of the CCIF flag in the FSTAT register. The CCIF flag will set upon completion of all active and buffered commands.

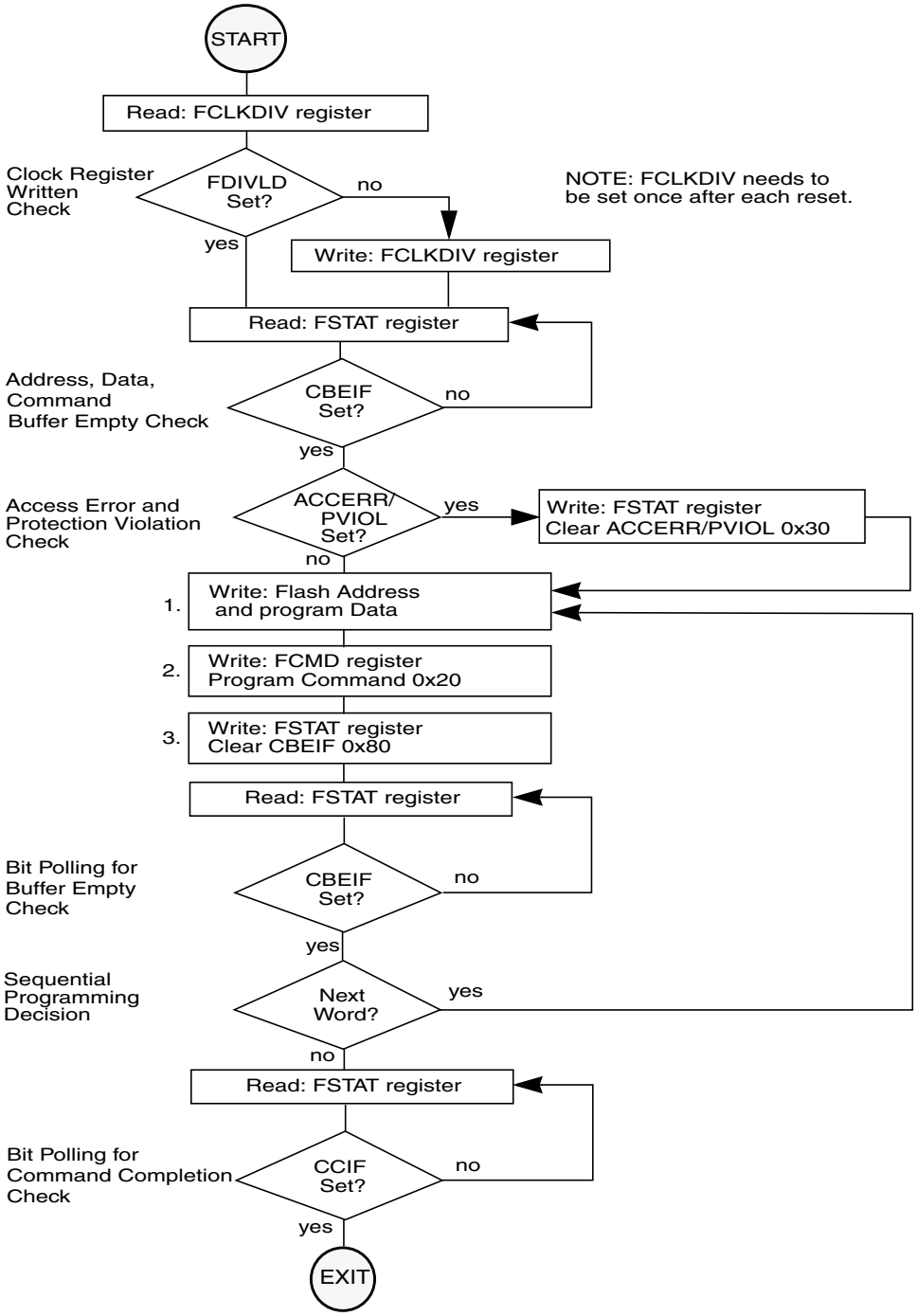


Figure 20-25. Example Program Command Flow

A.5.1.1 Single Word Programming

The programming time for single word programming is dependant on the bus frequency as a well as on the frequency f˜$_{NVMOP}$ and can be calculated according to the following formula.

$$t_{swpgm} = 9 \cdot \frac{1}{f_{NVMOP}} + 25 \cdot \frac{1}{f_{bus}}$$

A.5.1.2 Row Programming

Generally the time to program a consecutive word can be calculated as:

$$t_{bwpgm} = 4 \cdot \frac{1}{f_{NVMOP}} + 9 \cdot \frac{1}{f_{bus}}$$

For the C16, GC16, C32 and GC32 device flash arrays, where up to 32 words in a row can be programmed consecutively by keeping the command pipeline filled, the time to program a whole row is:

$$t_{brpgm} = t_{swpgm} + 31 \cdot t_{bwpgm}$$

For the C64, GC64, C96, C128 and GC128 device flash arrays, where up to 64 words in a row can be programmed consecutively by keeping the command pipeline filled, the time to program a whole row is:

$$t_{brpgm} = t_{swpgm} + 63 \cdot t_{bwpgm}$$

Row programming is more than 2 times faster than single word programming.

A.5.1.3 Sector Erase

Erasing either a 512 byte or 1024 byte Flash sector takes:

$$t_{era} \approx 4000 \cdot \frac{1}{f_{NVMOP}}$$

The setup times can be ignored for this operation.

A.5.1.4 Mass Erase

Erasing a NVM block takes:

$$t_{mass} \approx 20000 \cdot \frac{1}{f_{NVMOP}}$$

This is independent of sector size.

The setup times can be ignored for this operation.