PIC32AK1216GC41064 Family Flash Programming Specification

PIC32AK1216GC41064 Family

Programming Overview

This document defines the programming specification for the PIC32AK1216GC41064 family of 32-bit devices:

- PIC32AK1216GC41064
- PIC32AK1216GC41048
- PIC32AK1216GC41036
- PIC32AK6416GC41064
- PIC32AK6416GC41048
- PIC32AK6416GC41036
- PIC32AK3208GC41064
- PIC32AK3208GC41048
- PIC32AK3208GC41036

The programming is implemented via the In-Circuit Serial Programming (ICSP™) interface, which includes the clock and data pins (PGECx and PGEDx).

The PIC32AK1216GC41064 devices have a Nonvolatile Memory (NVM) controller module. This NVM module performs the device Flash memory programming. Its operation is configured and managed by the CPU. The external programmer tool uses the serial programming interface (ICSP) to shift in and execute CPU instructions, to move data in and out of the device and configure the NVM controller for programming operations. The programmer sends `MOV` instructions to the CPU to store (prepare) the Flash data to be programmed in the device RAM or NVM Control registers, depending on the Programming mode. Then, the programmer executes CPU instructions to initiate an erase or write operation using the NVM controller. To read Flash data from the device, a VISI register is implemented. The content of this register can be shifted out to the ICSP interface. For fast Flash content verification, the NVM controller supports a CRC-32 checksum engine. It can reduce the amount of data which should be read over the ICSP communications pins. Figure 1 explains the interactions between the programmer and the device.

Figure 1. Programming Interactions Block Diagram

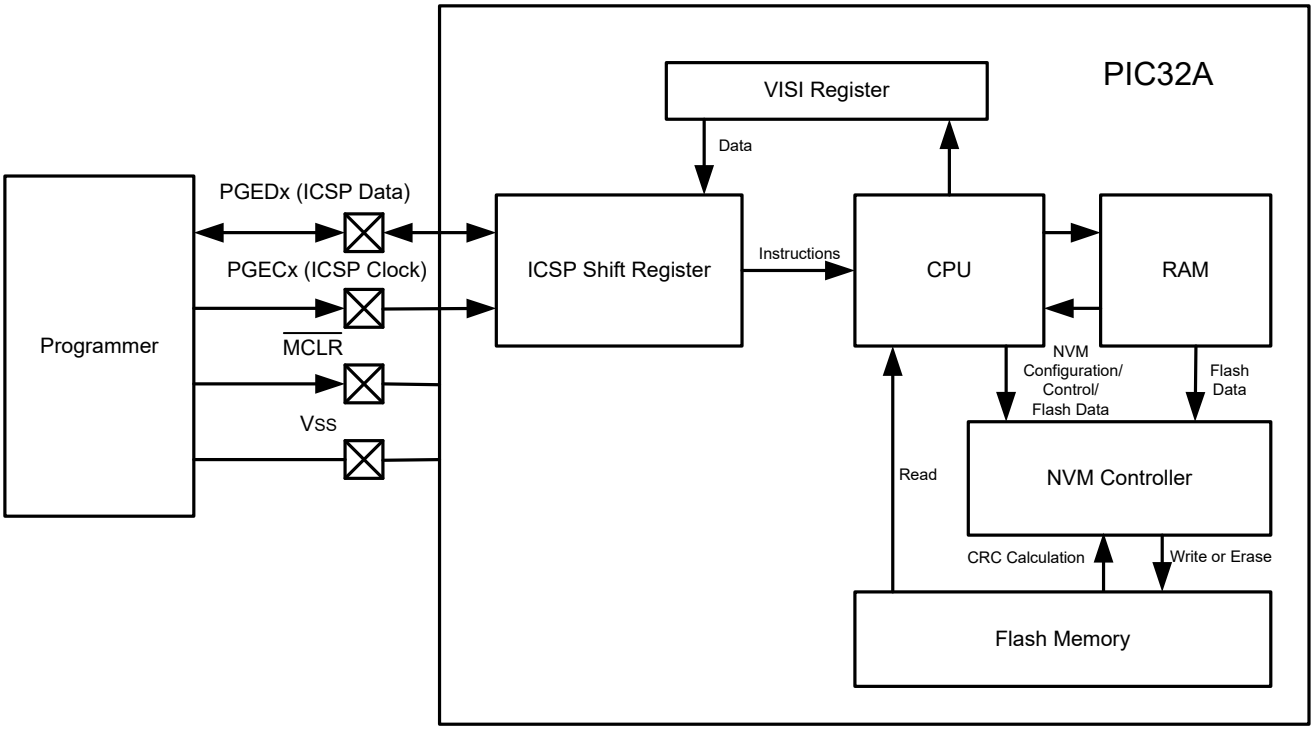


Table of Contents

1. Flash Memory

The Flash memory on PIC32AK1216GC41064 family devices is divided into quadwords (16 bytes each), write rows (512 bytes each) and erase pages (4096 bytes each). The quadword (four 32-bit long words) is the minimum data size that can be written into the Flash. The address of each quadword is aligned by 16 bytes. The Flash memory can be written by rows, and each row consists of 32 quadwords or 512 bytes. The address of each row is aligned by 512-byte boundaries.

The Flash memory must be erased before a write operation can be initiated, and it can be erased by pages. Each erase page consists of eight rows or 4096 bytes. The address of each erase page is aligned by 4096-byte boundaries.

The PIC32AK1216GC41064 family devices' Flash memory regions are shown in Table 1-1.

Table 1-1. Flash Memory Map

Memory Region		Address	Number of Quadwords	Number of Write Rows	Number of Erase Pages	Description
User OTP		0x7F2C00-0x7F3000	64	2	Not applicable	Quadword or row programmed, never erasable
User Configuration A (UCA)		0x7F3000-0x7F4000	256	8	1	Bulk or page erased, quadword programmed
User Configuration B (UCB)		0x7F4000-0x7F5000	256	8	1	
Code Memory	For 32-Kbyte Flash Memory Devices	0x800000-0x808000	2048	64	8	Bulk or page erased, quadword or row programmed
	For 64-Kbyte Flash Memory Devices	0x800000-0x810000	4096	128	16	
	For 128-Kbyte Flash Memory Devices	0x800000-0x820000	8192	256	32	

Some PIC32AK1216GC41064 family device registers are used in the programming procedures. These registers' descriptions and addresses are listed in Table 1-2.

Table 1-2. Registers Used in the Programming Procedures

Register Name	Address	Description
VISI	0x0007C0	Moves (shifts) data out of the device through the ICSP interface.
NVMCON	0x003000	Select a type and initiate an erase or write Flash operation.
NVMADR	0x003004	The destination Flash address for erase or write operations.
NVMDATA0	0x003008	With data to be programmed to Flash when quadword write is used.
NVMDATA1	0x00300C	With data to be programmed to Flash when quadword write is used.
NVMDATA2	0x003010	With data to be programmed to Flash when quadword write is used.
NVMDATA3	0x003014	With data to be programmed to Flash when quadword write is used.

.....continued		
Register Name	**Address**	**Description**
NVMSRCADR	0x003018	Should be set to the address of the RAM buffer loaded with the Flash row data to be programmed.
NVMCRCCON	0x003048	Controls the Cyclic Redundancy Check (CRC) calculation of the Flash region.
NVMCRCST	0x00304C	Contains the start address of the 4-Kbyte Flash memory block for the CRC calculation.
NVMCRCEND	0x003050	Contains the end address of the 4-Kbyte Flash memory block minus one byte for the CRC calculation.
NVMCRCSEED	0x003054	The initial CRC value (seed) should be loaded into this register. If the CRC-32 is calculated for the multiple memory blocks, then the previous CRC result must be written directly to NVMCRCSEED before the next block of CRC calculations.
NVMCRCDATA	0x003058	Contains the CRC calculation result.

1.1 Flash Memory Controller

This Flash (or NVM) controller module performs the erase or write operations of the device Flash memory. The programming configuration and status are provided in the NVMCON register. The NVMOP bits in the NVMCON register select the operation type. The descriptions of the available operations are listed in Table 1-3.

Table 1-3. Programming Operations

Type of Operation	NVMOP Bits (NVMCON[3:0]) Value	Description
Bulk Erase	1110	Erases the entire code and Configuration bits' memories. OTP area(s) are not erased.
Page Erase	0011	Erases a page of code or Configuration memory. The address of the Flash page must be specified in the NVMADR register. NVMADR[11:0] are ignored (treated as all '0's) for this NVMOP, hardware enforcing 4096-byte page alignment.
Row Write	0010	Writes a row of code or Configuration memory. The address of the Flash row must be specified in the NVMADR register. The row data must be stored (prepared) in RAM. The address of the Flash data RAM buffer must be specified in the NVMSRCADR register. The Flash memory must be erased before the write operation. NVMADR[8:0] are hardware ignored, enforcing 512-byte destination row alignment. NVMSRCADR[1:0] are unimplemented, so source data in RAM must start on a 32-bit long word-aligned boundary.

.....continued

Type of Operation	NVMOP Bits (NVMCON[3:0]) Value	Description
Quadword Write	0001	Writes four 32-bit long words of code, OTP or Configuration memory. The address of the Flash location to be written must be specified in the NVMADR register. The data must be stored (prepared) in the NVMDATA0, NVMDATA1, NVMDATA2 and NVMDATA3 registers. The Flash memory must be erased before the write operation. NVMADR[3:0] are unimplemented, enforcing 16-byte quadword address alignment.

Perform the following steps to erase or write the Flash memory:

1. Set the destination address to be erased or written in the NVMADR register.
2. Load data for the write operation in the NVMDATA0-NVMDATA3 registers or in a RAM buffer located by the address in the NVMSRCADR register.
3. Select the programming operation with the NVMOPx bits (NVMCON[3:0]).
4. Enable the NVM module operation by setting the WREN bit (NVMCON[14]).
5. Initiate the operation selected by setting the WR bit (NVMCON[15]).
6. Read/poll the WR bit (NVMCON[15]). The WR bit remains set while the erase or write operation is still in progress. Hardware clears this bit when the operation has completed.

The full description of the NVM controller and the programming details can be found in the device data sheet.

Also, the NVM controller can calculate a 32-bit Cyclic Redundancy Check (CRC) checksum. This checksum can be used to verify the result of Flash erase or programming. The start and end addresses of the Flash region are 4-Kbyte (page) aligned. The CRC can be calculated even while the Flash region is protected from data read back.

Perform the following steps to get the CRC checksum:

1. Write the start address into the NVMCRCST register.
2. Write the end address into the NVMCRCEND register.
3. Enable the CRC by setting the CRCEN bit (NVMCRCCON[15]).
4. Start the calculation by setting the START bit (NVMCRCCON[14]).
5. Read/poll the START bit (NVMCRCCON[14]). This bit is cleared when the CRC checksum is ready.
6. Read the CRC result from the NVMCRCDATA register.

Though the device hardware implements a parallel CRC-32 algorithm of the calculations, the same result can be achieved in the software by an algorithm using a 32-bit Shift register. The pseudocode in Example 1-1 explains the Shift register algorithm.

Example 1-1. Shift Register CRC Calculation Algorithm

```
// Initialize shift register
SeedValue(31:0) = 0
ShiftRegister(31:0) = INVERT SeedValue(31:0)
REPEAT FOR ALL DATA WORDS
    REPEAT 32 TIMES
        // Exclusive OR of data bit 31 and shift register bit 0
        CRCNext = DataWord(31) EXOR ShiftRegister(0)
        // Shift the register right
        ShiftRegister(30:0) = ShiftRegister(31:1)
        ShiftRegister(31) = 0
        // Shift data left to process the next bit of the word
        DataWord(31:1) = DataWord(30:0)
        IF CRCNext != 0 THEN
            // Exclusive OR with polynomial
            ShiftRegister(31:0) = ShiftRegister(31:0) EXOR 0xEDB88320
        ENDIF
    ENDREPEAT
```

```
ENDREPEAT
CRCResult = INVERT ShiftRegister(31:0)
```

Where:

- SeedValue[31:0] – Initial CRC value written into the NVMCRCSEED register
- DataWord[31:0] – Current data word for the CRC calculation
- CRCResult[31:0] – CRC checksum result in the NVMCRCDATA register

1.2 Device ID (DEVID)

Each device variant with a different memory size and package option can be identified by a device ID located in the DEVID register. The device revision is stored in the REVID register. These register addresses are shown in Table 1-4.

Table 1-4. Device ID and Revision Registers

Register Name	Address	Description
DEVID	0x7C2000	Device ID (device variant)
REVID	0x7C2004	Device revision

The device ID values for each memory and pin count variant are listed in Table 1-5.

Table 1-5. Device ID Values

Device	Device ID Value
PIC32AK1216GC41064	0x09DA3053
PIC32AK1216GC41048	0x09DA2053
PIC32AK1216GC41036	0x09DA1053
PIC32AK6416GC41064	0x09D93053
PIC32AK6416GC41048	0x09D92053
PIC32AK6416GC41036	0x09D91053
PIC32AK3208GC41064	0x09D83053
PIC32AK3208GC41048	0x09D82053
PIC32AK3208GC41036	0x09D81053

1.3 Unique Device ID (UDID)

All PIC32AK1216GC41064 family devices are individually encoded during final manufacturing with a Unique Device Identifier or UDID. The UDID cannot be erased by a Bulk Erase command or any other user-accessible means. This feature allows for manufacturing traceability of Microchip Technology devices in applications where this is a requirement. It may also be used by the application manufacturer for any number of things that may require unique identification, such as:

- Tracking the Device
- Unique Serial Number
- Unique Security Key

The UDID comprises four 32-bit program words. When taken together, these fields form a unique 128-bit identifier. The UDID is stored in four read-only locations in the device configuration space. Table 1-6 lists the UDID words.

Table 1-6. UDID Words

Register Name	Address	Description
UDID1	0x7F2BE0	UDID bits from 0 to 31
UDID2	0x7F2BE4	UDID bits from 32 to 63

.....continued		
Register Name	Address	Description
UDID3	0x7F2BE8	UDID bits from 64 to 95
UDID4	0x7F2BEC	UDID bits from 96 to 127

1.4 Configuration Bits

The Configuration bits are stored in the User Configuration A and B Flash memory areas. These bits can be programmed (= 0) or erased (= 1) to select various device configurations. There are two types of Configuration bits: system operation bits and code-protect bits. The system operation bits determine the power-on settings for system-level components, such as the Watchdog Timer. The code-protect bits prevent program memory from being read and written. Table 1-7 lists the Configuration Words of the PIC32AK1216GC41064 family devices. Refer to the device data sheet for the full Configuration Word register descriptions.

Table 1-7. Configuration Words

Configuration Word Name	Address	Configuration Area	Description
FCP	0x7F3000	UCA	This Configuration Word enables the Flash protection.
FICD	0x7F3010		Debugger Configuration register
FDEVOPT	0x7F3020		This Configuration Word configures some peripherals' features.
WDT	0x7F3030		This Configuration Word controls the Watchdog Timer.

.....continued			
Configuration Word Name	Address	Configuration Area	Description
FPRCTRL0	0x7F4000	UCB	These Configuration Words allow definition of up to eight memory regions with limitations on different Flash operations. FPRCTRLx Word enables the region and specifies what kind of Flash access should be restricted (erase, programming, reading, CRC calculation and so on). FPRSTx and FPRENDx should be programmed with the addresses of the first and last 4-Kbyte pages included in the region.
FPRST0	0x7F4004		
FPREND0	0x7F4008		
FPRCTRL1	0x7F4010		
FPRST1	0x7F4014		
FPREND1	0x7F4018		
FPRCTRL2	0x7F4020		
FPRST2	0x7F4024		
FPREND2	0x7F4028		
FPRCTRL3	0x7F4030		
FPRST3	0x7F4034		
FPREND3	0x7F4038		
FPRCTRL4	0x7F4040		
FPRST4	0x7F4044		
FPREND4	0x7F4048		
FPRCTRL5	0x7F4050		
FPRST5	0x7F4054		
FPREND5	0x7F4058		
FPRCTRL6	0x7F4060		
FPRST6	0x7F4064		
FPREND6	0x7F4068		
FPRCTRL7	0x7F4070		
FPRST7	0x7F4074		
FPREND7	0x7F4078		
FIRT	0x7F4080		This Configuration Word enables the Immutable Root of Trust Regions mode.
FSECDBG	0x7F4090		This Configuration Word enables the Secure Debug mode.
FTPED	0x7F40A0		This Configuration Word allows permanently disabling Chip Erase and external programming.
FEPUCB	0x7F40B0		Writing 0x84C1F396 in this Configuration Word disables the erase operation on the UCB area.
FWPUCB	0x7F40C0		Writing 0x5B9B12E4 in this Configuration Word disables the programming operation on the UCB area.

The PIC32AK1216GC41064 family has a backup copy of the Configuration Words in case the primary Configuration Words are corrupted and an ECC error is generated when they are read by the hardware. The backup Configuration Words are listed in Table 1-8 . These Configuration Words replace the primary Configuration Words in the Table 1-7 when the primary words cannot be accessed. In most applications, the backup Configuration Words have the same values as the primary Configuration Words.

Table 1-8. Configuration Words Backup

Configuration Word Name	Address	Configuration Area	Description
FCPBKUP	0x7F3800	UCA	This Configuration Word enables the Flash protection.
FICDBKUP	0x7F3810		Debugger Configuration register
FDEVOPTBKUP	0x7F3820		This Configuration Word configures some peripherals' features.
WDTBKUP	0x7F3830		This Configuration Word controls the Watchdog Timer.

.....continued			
Configuration Word Name	Address	Configuration Area	Description
FPRCTRL0BKUP	0x7F4800	UCB	These Configuration Words allow definition of up to eight memory regions with limitations on different Flash operations. FPRCTRLx Word enables the region and specifies what kind of Flash access should be restricted (erase, programming, reading, CRC calculation and so on). FPRSTx and FPRENDx should be programmed with the addresses of the first and last 4-Kbyte pages included in the region.
FPRST0BKUP	0x7F4804		
FPREND0BKUP	0x7F4808		
FPRCTRL1BKUP	0x7F4810		
FPRST1BKUP	0x7F4814		
FPREND1BKUP	0x7F4818		
FPRCTRL2BKUP	0x7F4820		
FPRST2BKUP	0x7F4824		
FPREND2BKUP	0x7F4828		
FPRCTRL3BKUP	0x7F4830		
FPRST3BKUP	0x7F4834		
FPREND3BKUP	0x7F4838		
FPRCTRL4BKUP	0x7F4840		
FPRST4BKUP	0x7F4844		
FPREND4BKUP	0x7F4848		
FPRCTRL5BKUP	0x7F4850		
FPRST5BKUP	0x7F4854		
FPREND5BKUP	0x7F4858		
FPRCTRL6BKUP	0x7F4860		
FPRST6BKUP	0x7F4864		
FPREND6BKUP	0x7F4868		
FPRCTRL7BKUP	0x7F4870		
FPRST7BKUP	0x7F4874		
FPREND7BKUP	0x7F4878		
FIRTBKUP	0x7F4880		This Configuration Word enables the Immutable Root of Trust Regions mode.
FSECDBGBKUP	0x7F4890		This Configuration Word enables the Secure Debug mode.
FTPEDBKUP	0x7F48A0		This Configuration Word allows permanently disabling Chip Erase and external programming.
FEPUCBBKUP	0x7F48B0		Writing 0x84C1F396 in this Configuration Word disables the erase operation on the UCB area.
FWPUCBBKUP	0x7F48C0		Writing 0x5B9B12E4 in this Configuration Word disables the programming operation on the UCB area.

1.5 Flash Memory Protection

The Configuration Words in User Configuration A (UCA) and User Configuration B (UCB) Flash areas can restrict Flash operations over the entire Flash or defined memory regions. Refer to the device data sheet for a detailed description.

The NVM controller can calculate a CRC-32 checksum, even if the read operation is disabled for the Flash region, unless the CRC operation is also disabled.

1.5.1 Protected Regions

Up to eight protected regions can be created in the Flash memory. The start address, end address and region type are defined by the Configuration Words in UCB memory. The protected region can be configured to restrict read, write, erase (OTP) or CRC calculation by the NVM module.

1.5.2 User Configuration A and B Areas Protection

The FCP Configuration Word, located in UCA, has WPUCA bits which can disable Page Erase and Write operations on the UCA area. The UCA area is always erased on Bulk Erase.

Two Configuration Words located in the UCB area, if programmed with special keys, disable erase or write operations on the UCB area. FEPUCB, when programmed with 0x84C1F396, permanently disables all ability to erase the UCB page via both Bulk Erase and Page Erase NVM commands. FWPUCB, when programmed with 0x5B9B12E4, permanently disables all ability to write further data to the UCB page. FEPUCB Erase Protection and FWPUCB Write Protection apply to both ICSP and Run-Time Self Programming code execution. If both the UCB Program and Erase Protection are activated, it permanently prevents UCB programming and erase operations, and it cannot be deactivated.

1.5.3 Entire Flash Protection

The FCP Configuration Word, located in the UCA area, allows protection of the entire Flash from ICSP read-back (CP bit) and CRC calculations by the NVM module (CRC bit).

The FPED Configuration Word in the UCB area has the PED bit, which disables Bulk Erase, Page Erase, Row and Quadword Write for the entire device Flash. When this feature is enabled (PED bit = 0), it restricts all modification of the user Flash, User OTP and UCA/UCB Configuration pages in ICSP mode. Run-Time Self-Programming code execution is unaffected by PED.

When this Program or Erase Protection feature (PED bit = 0) is used with UCB Erase Protection (FEPUCB word is 0x84C1F396), once activated, it permanently prevents the entire Flash ICSP programming and erase operations, and it cannot be deactivated.

1.6 Flash Memory Operations Timing

The maximum time for the Flash operations can be found in Table 1-9.

Table 1-9. Flash Operations Maximum Time

Flash Operation	Maximum Time
Bulk Erase	20 mS
Page Erase	20 mS
Quad Word Programming	15 µS
Row Programming	500 µS

2. In-Circuit Serial Programming (ICSP™)

The In-Circuit Serial Programming (ICSP) mode allows shifting in and executing CPU instructions, and reading VISI register content through the PGECx/PGEDx pins. The PGECx is a serial clock signal, and PGEDx is a data signal.

2.1 Connections Required for ICSP

For ICSP operations, the device must be powered using all power and ground pins available on the package. In addition to the power pins, the programming interface includes the $\overline{MCLR}$ (Reset) and one ICSP programming pin pair (PGEDx/PGECx). All PIC32AK1216GC41064 devices have three separate pairs of programming pins, labeled as PGEC1/PGED1, PGEC2/PGED2 and PGEC3/PGED3. Table 2-1 explains details about the required connections.

Table 2-1. Pins Used for Programming

Pin Name	Pin Type	Description
V_{DD}/AV_{DD}	Power	Power supply. All power pins must be connected.
V_{SS}/AV_{SS}	Power	Ground. All ground pins must be connected.
$\overline{MCLR}$	Input	Target device Reset/programming enable.
PGECx	Input	ICSP programming clock. Where 'x' is the programming pair number.
PGEDx	Input/Output	ICSP programming data. Where 'x' is the programming pair number.

CMOS logic thresholds apply to $\overline{MCLR}$ and the PGECx/PGEDx signals as specified in Table 2-2.

Table 2-2. Electrical Specification for the Programming Interface Signals

Parameter	Min.	Max.	Description
V_{IL}	V_{SS}	0.2 * V_{DD}	Input low-level voltage
V_{IH}	0.8 * V_{DD}	V_{DD}	Input high-level voltage

For all ICSP operations clocking data into the target device, the PGEDx Output state must be valid and stable before the PGECx clock rising edges. All PGEDx data must be presented Least Significant bit (LSb) first.

Figure 2-1, Figure 2-2 and Table 2-3 specify the timing for the ICSP interface signals.

Figure 2-1. ICSP™ Interface Signals Timing Parameters (Programmer Driving PGEDx Pin)

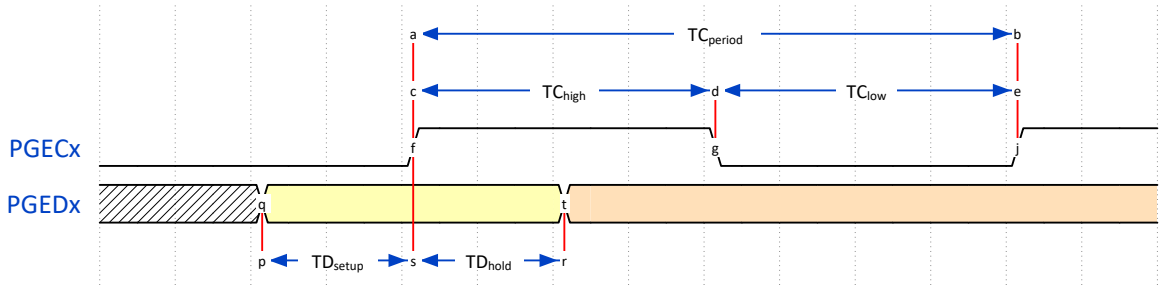


Figure 2-2. ICSP™ Interface Signals Timing Parameters (Target Device Driving PGEDx Pin)

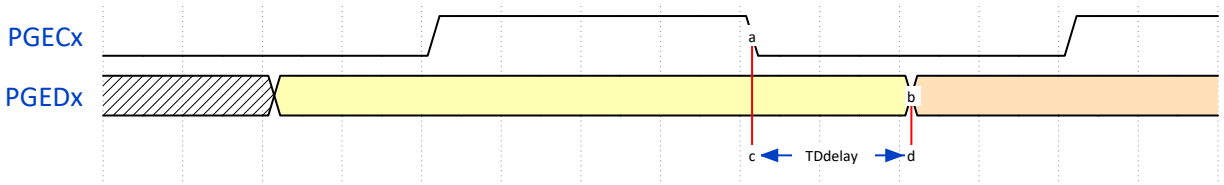


Table 2-3. ICSP™ Interface Signals Timing Parameters

Parameter	Min.	Max.	Description
TD_{setup}	20 nS	—	Minimum PGEDx data setup time before PGECx rising edge
TD_{hold}	1 nS	—	Minimum PGEDx data hold time after PGECx rising edge
TC_{period}	60 nS	—	Minimum PGECx clock period
TC_{low} or TC_{high}	20 nS	—	Minimum PGECx clock low or high pulse width
TD_{delay}	—	20 nS	Maximum data update delay after PGECx falling edge

Figure 2-3 through Figure 2-5 provide the pin diagrams for the different packages of the new for PIC32AK1216GC41064 devices. The detailed pin descriptions can be found in the device data sheet.

Figure 2-3. 36-Pin VQFN

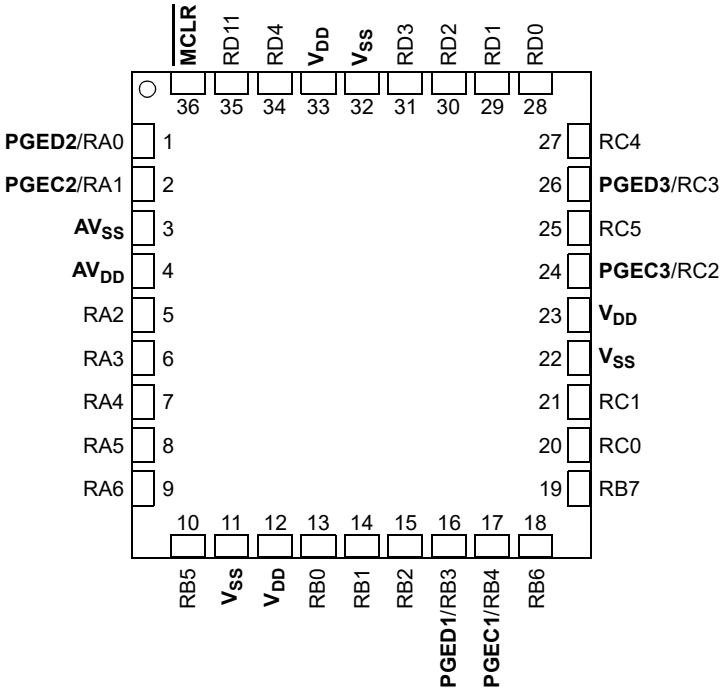


Figure 2-4. 48-Pin VQFN, TQFP

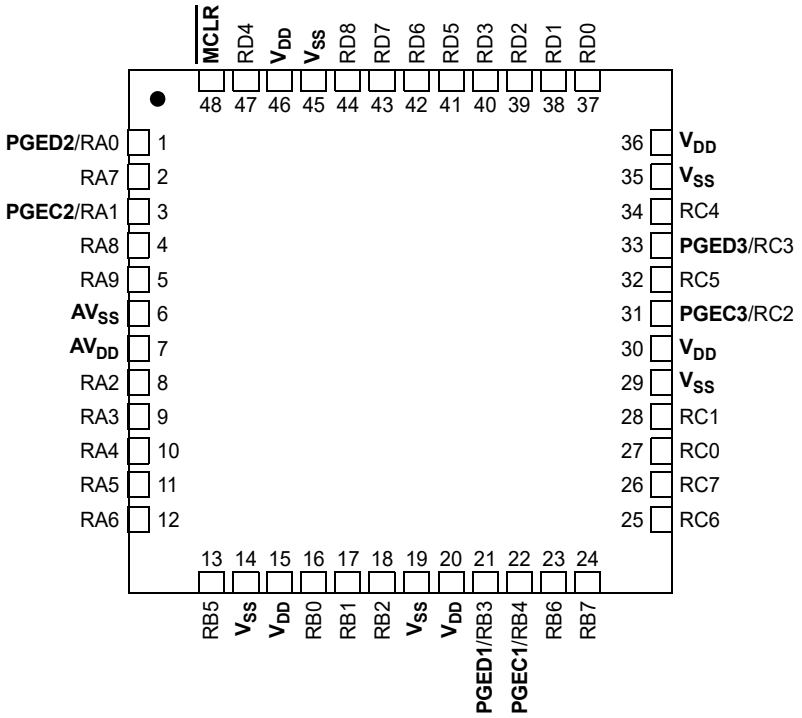
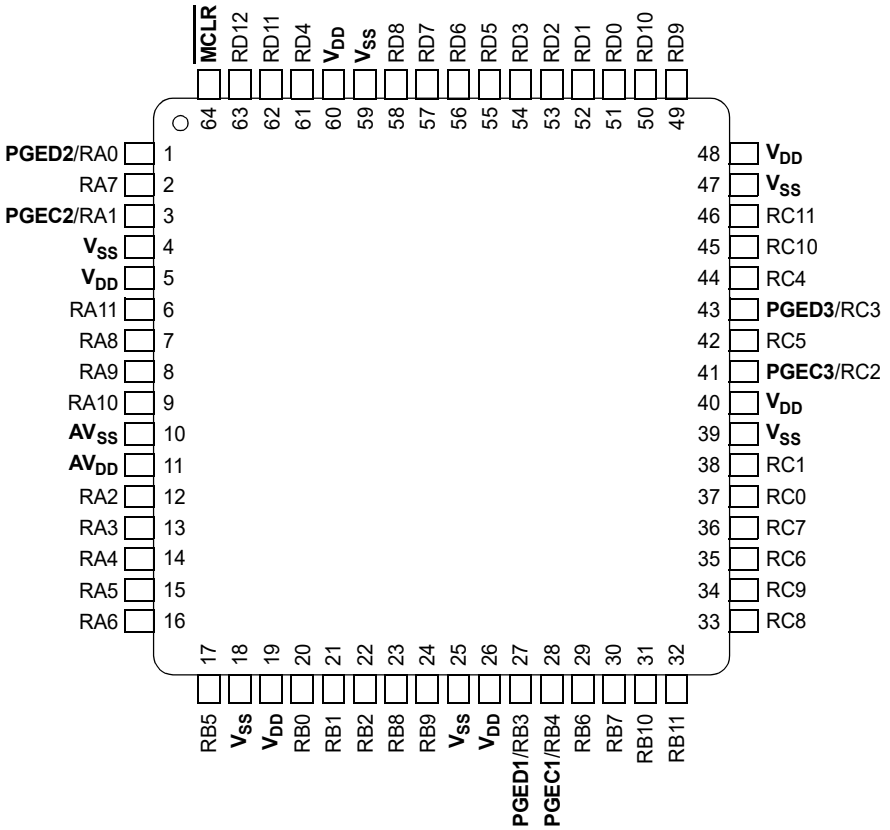


Figure 2-5. 64-Pin QFN, VQFN, TQFP



2.2 ICSP Mode Entry

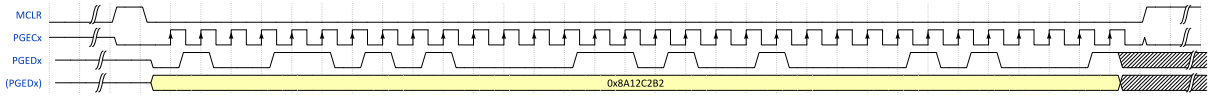
ICSP mode is entered by a specific sequence on the $\overline{MCLR}$, PGECx and PGEDx pins. PGECx/PGEDx may be selected from the three available pairs (PGEC1/PGED1, PGEC2/PGED2 or PGEC3/PGED3).

To enter ICSP mode, the following steps must be taken:

1. Power the device. V_{DD}/AV_{DD} should remain stable throughout all ICSP operations. Cycling power is not required, so the external programming tool and target chip or circuit may be independently powered; however, a shared V_{SS} ground reference is required.
2. Drive $\overline{MCLR}$ low.
3. Begin driving PGECx and PGEDx low. PGEDx is a “don't care” at this stage, but to simplify subsequent operations, it should be made an output and driven low.
4. Wait at least 1 ms (no maximum duration).
5. Pulse $\overline{MCLR}$ high, then low. Pulse duration must be at least 20 ns but should be as short as possible. Long durations allow existing code in the target device’s Flash to begin execution. If this user code has RTSP (Run-Time Self-Programming) procedures, then the user’s code/RTSP procedures may modify the Flash memory content. In this case, some programmer steps, such as a verification of the programmed Flash data, may fail. Therefore, do not exceed a $\overline{MCLR}$ high duration of >2 µs unless it’s unavoidable, and even then, try to minimize the dwell time with $\overline{MCLR}$ high.

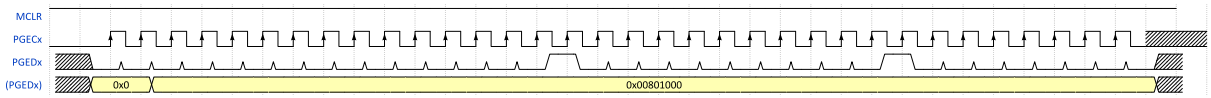
6. Generate 32 PGECx clock pulses and simultaneously shift the value, `0x8A12C2B2`, onto the PGEDx pin, the Least Significant bit first. On the wire, if decoded as big-endian octets, this will generate the byte sequence: 0x4D, 0x43, 0x48, 0x51.
7. After the 32nd falling PGECx clock edge, raise $\overline{\text{MCLR}}$. The ICSP signal waveforms are shown in Figure 2-6.

Figure 2-6. ICSP Entry Sequence Waveforms (PGECx Pulses from 1 to 32)



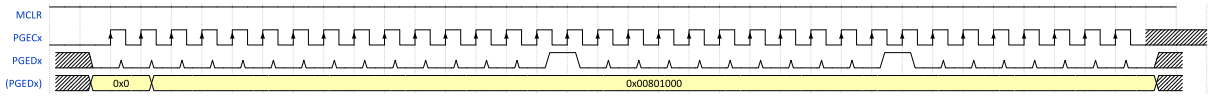
8. Wait at least 500 µs. During this interval, PGECx must remain low, but PGEDx is a “don’t care.”
9. Generate 34 PGECx clock pulses to initialize internal clock selection logic. PGEDx, for these 34 clocks, should be the bit pattern, ‘00’, followed by the 32-bit constant, `0x00801000`. The ICSP signal waveforms for this step are shown in Figure 2-7.

Figure 2-7. ICSP Entry Sequence Waveforms (PGECx Pulses from 33 to 66)



10. Generate 34 PGECx clock pulses to set the (not meaningful) CPU Reset vector target. PGEDx, for these 34 clocks, should be the bit pattern, ‘00’, followed by the 32-bit constant, `0x00801000` (same as in Step 9). The ICSP signal waveforms for this step are shown in Figure 2-8.

Figure 2-8. ICSP Entry Sequence Waveforms (PGECx Pulses from 67 to 100)



The programmer should continue to hold $\overline{\text{MCLR}}$ high for all subsequent ICSP operations. ICSP mode will terminate when $\overline{\text{MCLR}}$ is pulled low for 1 ms or longer, or V_{DD}/AV_{DD} is removed.

2.3 ICSP Mode Exit

To exit ICSP mode and begin internal code execution, drive $\overline{\text{MCLR}}$ low, tri-state PGECx and PGEDx, maintain $\overline{\text{MCLR}}$ low for 1 ms and then raise $\overline{\text{MCLR}}$ (preferably tri-state $\overline{\text{MCLR}}$ instead and let an on-board pull-up to V_{DD} decide if user code execution is wanted at this point or not).

Exiting ICSP mode causes a pseudo-POR event internally. Therefore, all SFRs written during the ICSP session will be cleared back to their Reset default states, generally including the SFRs specifically documented as being reset only on POR or BOR-type Resets. RAM state generated during the ICSP session will be retained until the target is power cycled, overwritten by RAM BIST hardware or overwritten by application code execution.

2.4 ICSP Commands

The ICSP command codes notify the target device of the next operation. The codes are followed by a sequence of clocks to shift data into or out from the target. Before issuing any command, the device must first be placed in ICSP mode (refer to ICSP Mode Entry).

The command codes are a fixed length of two bits, require two PGECx clocks to transfer and are implemented according to Table 2-4.

Table 2-4. ICSP Command Codes

Binary Command	Mnemonic	Operation
00	CMDEXEC	Shifts one 32-bit instruction, two 16-bit instructions or half of a 64-bit instruction into the CPU. The instruction(s) are then executed by the target while receiving the next command and its data.
01	CMDRD	Sets the PGEDx pin as an output, shifts 32 bits of data from the VISI register onto the pin and then restores PGEDx as an input for the next command.
10	CMDSEQWR	Sequentially loads 32 bits of data into memory pointed to by W0 on the target, then post-increments W0 by 4. This command internally decomposes the 32 bits of PGEDx shift data into a 64-bit MOV.L instruction executed by the target processor: MOV.L #PGEDx<31:0>, [W0++]
11	CMDSEQRD	Returns the VISI register and internally executes the following instruction: MOV.L [W0++], [W8] When W0 is initialized to a source memory address and W8 contains the address of the VISI register, this command permits sequential memory addresses to be read from the target with minimal overhead.

The 2-bit command code and 32 bits of data transferred into or out from the target are always presented on the wire, the Least Significant bit first.

All command code and data bits, driven onto the PGEDx pin by the programmer, are latched by the target on the rising edge of the PGECx clock. Therefore, the PGEDx state must be valid and stable for at least one setup time before the PGECx rising edge (TD_{setup}) and held in the same state for at least one hold time after PGECx has risen (TD_{hold}).

When reading data from the target, each of the 32 data bits is output/changed on PGEDx by the target on the falling edge of the PGECx clock (still generated by the programmer). The data remain unchanged until the next falling edge, so the programmer should sample the data on or after the rising clock edge, but before it generates another PGECx falling edge.

2.4.1 ICSP Instruction Execution (CMDEXEC)

The Instruction Execution Sequence (CMDEXEC) shifts one 32-bit instruction, two 16-bit instructions or half of a 64-bit instruction into the target CPU.

The Instruction Execution Sequence is:

1. The programmer shifts the '00' bits into the device.
2. Then, the programmer sends 32 bits of instruction opcode data to the device. Each PGEDx bit is latched by the device on the rising edge of the PGECx pulse. The CPU executes the instruction(s) over the next 5-10 PGECx clocks while sending the next ICSP command.

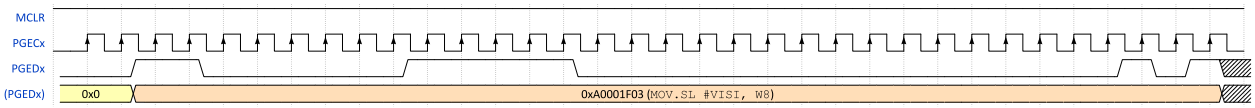
When providing 16-bit instruction(s), the first 16-bit instruction to execute must be aligned to the Least Significant 16 bits and followed by another 16-bit instruction in the upper half of the 32-bit data transfer. If only one 16-bit instruction is available, it must be aligned in the low 16 bits and zero-extended out to 32 bits (upper 16 bits encode a NEOP).

When executing a 64-bit instruction, the first CMDEXEC Sequence will load the first instruction word (bits 0-31 of the opcode). A second CMDEXEC Sequence must be issued to load the second instruction word (bits 32-63 of the opcode) and allow the CPU to begin executing the instruction.

For 64-bit instructions, the second CMDEXEC Sequence must be sent after the first CMDEXEC Sequence without any other intervening ICSP commands. Otherwise, the CPU's instruction flow will be corrupted and an Illegal Opcode Reset may be generated.

Figure 2-9 shows an example where the "MOV.SL #VISI, W8" instruction (load the VISI register address into the W8 register), with opcode 0xA0001F03, is shifted into the CPU.

Figure 2-9. CMDEXEC Sequence Where MOV.SL #VISI, W8 with 0xA0001F03 Opcode is Shifted Into the CPU



2.4.2 ICSP Read of VISI Register (CMDRD)

This command shifts the content of the VISI register onto the PGEDx pin for capture by the programmer. When combined with an Instruction Execution Sequence (CMDEXEC) that writes to VISI, it may be used to read status registers and memory content from the target device.

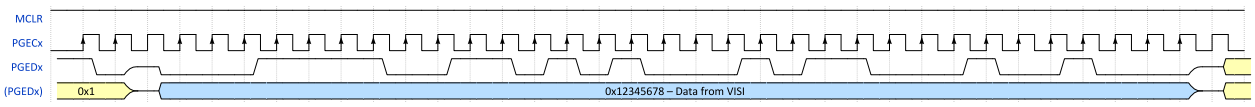
The read of VISI Register Sequence is:

1. The programmer shifts the '01' bits into the device. As all data follow the Least Significant bit first ordering, the '1' bit must be presented on the PGEDx wire before the '0' bit.
2. One Idle PGECx clock is generated by the programmer to accommodate switching the PGEDx pin direction. The programmer should tri-state PGEDx before the falling edge of this clock, as the target device will begin driving bit 0 of VISI at this time.
3. The programmer shifts in the 32-bit VISI register content from the device. The Least Significant bit of the VISI register is presented first. The device updates each data bit on the falling edge of the PGECx pulse. To accommodate propagation delay of the clock into the device and output delay returning the next bit, the programmer should sample the data from the device on or after the PGECx rising edge.
4. One Idle PGECx clock is generated by the programmer to accommodate switching the PGEDx pin direction back to an input. The device tri-states PGEDx on the falling edge that completes the VISI bit 31 transfer, so the programmer can begin driving PGEDx anytime during or at the end of this clock.

All Instruction Execution Sequences (CMDEXEC) that write to the VISI register do not complete execution until several clocks belonging to the next command have been presented. Therefore, any operation that writes to VISI will not have taken effect if immediately followed by a read of the VISI Register Sequence (CMDRD). Placing an operation that writes to VISI immediately before a read of the VISI Register Sequence (CMDRD) will still execute normally, but the data returned will be delayed (will contain old VISI data). To receive the up-to-date VISI content moved by the preceding CMDEXEC Sequence, a second CMDEXEC, with any opcode, should be executed before the CMDRD Sequence.

Figure 2-10 shows an example of the CMDRD Sequence where the 0x12345678 data are shifted out from the VISI register to the programmer.

Figure 2-10. CMDRD Data Sequence with Data Shifted out from VISI Register to Programmer



2.4.3 ICSP Sequential Write (CMDSEQWR)

The ICSP Sequential Write Sequence (CMDSEQWR) permits rapid writes of 32-bit data via an indirect pointer access to RAM or device registers. When the sequence is completed, a 64-bit MOV.L instruction is internally generated, encoding all 32 bits of shifted data and passed to the CPU for

execution. The outcome is equivalent to issuing two CMDEXEC Sequences necessary to execute a 64-bit instruction of:

```
MOV.L #PGEDx<31:0>, [W0++]
```

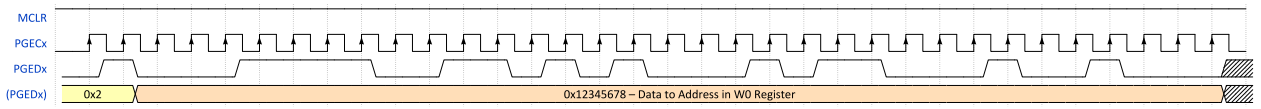
This instruction writes the 32 bits of provided data to the memory pointed to by the 4-byte aligned address in the W0 register, then increments the W0 pointer by four. By pre-initializing W0 to a RAM or register address, incrementing locations may be written to, with minimal overhead, simply by executing a series of back-to-back CMDSEQWR Sequences. Generally, CMDSEQWR should be used for all data writes of four or more bytes from the programmer to 4-byte aligned sequential addresses.

The Sequential Write Sequence is:

1. The programmer shifts the '10' bits into the device. As all data follow the Least Significant bit first ordering, the '0' bit must be presented on the PGEDx wire before the '1' bit.
2. Then, the programmer sends 32 bits of RAM or SFR data to the device. Each PGEDx bit is latched by the device on the rising edge of the PGECx pulse. The CPU executes the generated MOV.L instruction over the next 5-10 PGECx clocks while sending the next ICSP command.

Figure 2-11 shows an example of the CMDSEQWR Sequence, where 0x12345678 from the programmer is written to the address in the W0 register.

Figure 2-11. CMDSEQWR Sequence Where Data from Programmer are Written to Address in W0 Register



2.4.4 ICSP Sequential Read (CMDSEQRD)

This Sequential Read Sequence shifts out the 32-bit VISI register content onto the PGEDx pin, identically to CMDRD, but will internally also execute the following instruction:

```
MOV.L [W0++], [W8]
```

The existing VISI content is buffered before execution of this MOV.L instruction, so all 32 bits of VISI appear on PGEDx as they existed before this instruction is executed. However, this instruction is typically used to reload VISI with more data from SFRs, RAM or program space, and it completes execution by the time all bits have shifted out. Therefore, CMDSEQRD is optimized for use in a loop to transfer multiple sequential memory contents out of the target and into the programmer.

As the generated MOV.L instruction references W0 and W8, this command must be preceded with CMDEXEC commands to load W8 with the address of the VISI register and W0 with the read address on the target:

```
MOV.SL #VISI, W8

MOV.SL #ReadAddress<23:0>, W0
```

The Sequential Read Sequence is:

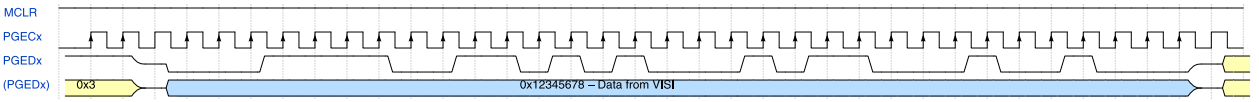
1. The programmer shifts the '11' bits into the device.
2. One Idle PGECx clock is generated by the programmer to accommodate switching the PGEDx pin direction; the programmer should tri-state PGEDx before the falling edge of this clock, as the target device will begin driving bit 0 of VISI at this time.
3. The programmer shifts in the 32-bit VISI register content from the device. The Least Significant bit of the VISI register is presented first. The device updates each data bit on the falling edge of the PGECx pulse. To accommodate propagation delay of the clock into the device and output delay returning the next bit, the programmer should sample the data from the device on or after the PGECx rising edge.

4. One Idle PGECx clock is generated by the programmer to accommodate switching the PGEDx pin direction back to an input; the device tri-states PGEDx on the falling edge that completes the VISI bit 31 transfer, so the programmer can begin driving PGEDx anytime during or at the end of this clock.

Unlike `CMDEXEC` and `CMDSEQWR`, the CPU will finish execution of the internally generated `MOV.L [W0++], [W8]` instruction before the final (buffered) bit 31 of VISI is shifted out. This permits back-to-back reissuing of `CMDSEQRD` to return the next 32 bits of memory without extra clocking following the VISI write.

Figure 2-12 shows an example of `CMDSEQRD`, where the `0x12345678` data are received by a programmer from VISI.

Figure 2-12. Example of `CMDSEQRD` Where the Data are Received by a Programmer from VISI



3. Programming Procedures

The PIC32AK1216GC41064 family devices programming flow consists of the following steps:

1. Enter ICSP mode.
2. Bulk Erase entire Flash.
3. Write code memory.
4. Read back or calculate CRC to verify the code memory.
5. Write User Configuration A memory.
6. Calculate CRC to verify the User Configuration A memory.
7. Write User Configuration B memory.
8. Calculate CRC to verify the User Configuration B memory.
9. Exit ICSP mode.

If the programming or erase operations are disabled in Configuration Words of UCA and UCB areas, then the Flash erase or write procedures will fail with no effect for the protected areas. To erase all Flash memory, the protected regions defined in the UCB area must be disabled by erasing UCB and exiting/re-entering ICSP mode.

The following steps should be done to remove the code protection:

1. Enter ICSP mode.
2. Bulk Erase to erase UCA memory, including the code protection bits of UCB memory.
3. Exit ICSP mode to reload Configuration Words.
4. Enter ICSP mode.
5. Bulk Erase to erase UCB memory, including the code protection bits. This procedure will also erase the entire Code Flash memory.
6. Exit ICSP mode to reload Configuration Words.

When the Code Flash memory is not protected from programming or erase, then the Bulk Erase does a parallel erase operation and completes quickly (refer to the data sheet for the Bulk Erase time). Conversely, if any erase, write or OTP protected regions are defined in the Code Flash, then the Bulk Erase command will still be accepted; however, internally, the NVM controller will perform the Bulk Erase as a self-iterated sequential Page Erase, resulting in a many-times-longer Bulk Erase time.

3.1 Bulk Erase

This algorithm erases the user Flash memory and the UCA and UCB User Configuration pages. User OTP is retained and can never be erased. If hardware write-protect functions have been enabled in the UCA and UCB page contents, the UCA and UCB pages and arbitrary Flash address ranges defined by security region descriptors in UCB may also be permanently retained and will survive Bulk Erase.

Table 3-1. Bulk Erase Algorithm

ICSP Sequence	ICSP Command Code	Data/Opcode	Instruction Executed
Step 1: Initialize pointers. Move VISI address to W8 and NVMCON address to W9.			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C003	MOV.SL #NVMCON, W9
Step 2: Write 0x400E to NVMCON to set up for the Bulk Erase NVM operation.			
CMDEXEC	00	0x8A9004E1	MOVS.W #0x400E, [W9]
Step 3: Set WR bit (NVMCON[15]) to start the NVM operation and move NVMCON to VISI.			

.....continued			
ICSP Sequence	**ICSP Command Code**	**Data/Opcode**	**Instruction Executed**
CMDEXEC	00	0x8E9004E1	MOVS.W #0xC00E, [W9]
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
Step 4: Provide clocks to complete execution of the prior move of NVMCON to VISI by executing a second move of NVMCON to VISI, then shift VISI on PGEDx.			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
Step 5: Repeat Step 4 to poll WR bit (NVMCON[15]) until it is clear, indicating completion.			

3.2 Page Erase

This algorithm (Table 3-2) erases one page of the Code Flash memory. The erase address is page-size aligned, and "don't care" bits are its least significant part. The Page Erase procedure does not have an effect on the User OTP memory. If FPED or Code-Protect have been enabled, Page Erase will have no effect on the User Configuration A and B areas. Like Bulk Erase, the Page Erase command may have no effect when targeting the UCA page, UCB page or an address range with a security descriptor defining the page as hardware write-protected.

Table 3-2. Page Erase Algorithm

ICSP Sequence	ICSP Command Code	Data/Opcode	Instruction Executed
Step 1: Initialize pointers. Move VISI address to W8 and NVMCON address to W9.			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C003	MOV.SL #NVMCON, W9
Step 2: Move NVMCON address to W0.			
CMDEXEC	00	0x00000309	MOV.L W9, W0
Step 3: Write 0x4003 to NVMCON to set up for the Page Erase NVM operation.			
CMDSEQWR	10	0x00004003	MOV.L #PGEDx<31:0>, [W0++]
Step 4: Initialize NVMADR with any address on the target page.			
CMDSEQWR	10	EraseAddress[31:0]	MOV.L #PGEDx<31:0>, [W0++]
Step 5: Set WR bit (NVMCON[15]) to start the NVM operation and move NVMCON to VISI.			
CMDEXEC	00	0x8E900431	MOVS.W #0xC003, [W9]
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
Step 6: Provide clocks to complete execution of the prior move of NVMCON to VISI by executing a second move of NVMCON to VISI, then shift VISI on PGEDx.			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
Step 7: Repeat Step 6 to poll WR bit (NVMCON[15]) until it is clear, indicating completion.			

3.3 Quadword Program

This algorithm (Table 3-3) programs one 16-byte aligned quadword. The Least Significant four bits of the destination address are “don't care” bits when written and will be forced to zeros for the programming operation. The Flash memory on these devices has a built-in Error Correcting Code (ECC) mechanism. The ECC checksum is calculated during write operations and stored, unmapped, alongside each quadword. If the same Flash quadword is programmed more than one time without an intermediate erase cycle, this ECC checksum becomes corrupted. Subsequently, when the location with a corrupted checksum is read, an ECC error interrupt or trap will be generated. After the Flash is erased, the programmer must not write to any Flash quadword location (including OTP areas) more than one time.

Table 3-3. Quadword Programming Algorithm

ICSP Sequence	ICSP Command Code	Data/Opcode	Instruction Executed
Step 1: Move VISI address to W8 and NVMCON address to W9.			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C003	MOV.SL #NVMCON, W9
Step 2: Initialize W0 with an NVMCON address for sequential loading. Load the NVMCON value with WR bit set in W10. Write 0x4001 in NVMCON to set up for Quadword Programming.			
CMDEXEC	00	0x00000309	MOV.L W9, W0
CMDEXEC	00	0xA8030007	MOV.SL #0xC001, W10
CMDSEQWR	10	0x00004001	MOV.L #PGEDx<31:0>, [W0++]
Step 3: Load the destination address in NVMADR.			
CMDSEQWR	10	Destination Address[31:0]	MOV.L #PGEDx<31:0>, [W0++]
Step 4: Load data in the NVM Controller Data registers (NVMDATA0, NVMDATA1, NVMDATA2 and NVMDATA3).			
CMDSEQWR	10	Data[31:0]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	Data[63:32]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	Data[95:64]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	Data[127:96]	MOV.L #PGEDx<31:0>, [W0++]
Step 5: Restore W0 with NVMCON’s address. Set WR bit (NVMCON[15]) to start the NVM operation. Move NVMCON to VISI.			
CMDEXEC	00	0x1F0A0309	MOV.L W9, W0
			MOV.L W10, [W0++]
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
Step 6: Move NVMCON to VISI. Shift VISI on PGEDx.			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
Step 7: Repeat Step 6 to poll WR bit (NVMCON[15]) until it is clear, indicating completion.			
Step 8: Repeat Steps 3-7 until all sequential quadwords have been programmed.			

3.4 Row Program

This algorithm (Table 3-4) programs one or more Flash rows. The row data will be preloaded into the device’s RAM. While the NVM controller programs/transfers the data from RAM to Flash, an additional row of data can be moved into device RAM for a subsequent Row Programming operation. By overlapping the transfer of data from the programmer to the device RAM, and from the device RAM to Flash via the NVM controller, this programming algorithm attains maximal programming throughput approaching the raw clocking rate of the PGECx/PGEDx pins. This

operation should always be used when the source data are suitably aligned and big enough to define at least one row.

The Flash memory on these devices has a built-in Error Correcting Code (ECC) mechanism. The ECC checksum is calculated during a write operation and stored with data in Flash. If the Flash memory is programmed more than one time, this ECC checksum becomes corrupted. Next time, when the location with a corrupted checksum will be accessed, a single-bit ECC error interrupt or double-bit ECC error trap may be generated. After the Flash is erased, the programmer must not write to any Flash location (including OTP areas) more then one time.

The Row Programming is disabled for User Configuration A and B areas and will not have an effect on them.

Table 3-4. Row Programming Algorithm

ICSP Sequence	ICSP Command Code	Data/Opcode	Instruction Executed
Step 1: Initialize pointers. Move VISI address to W8 and NVMCON address to W9.			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C003	MOV.SL #NVMCON, W9
Step 2: Set W1 to a starting address in RAM (0x4000), then duplicate this pointer in W0 for CMDSEQWR use. Initialize NVMCON with 0x4002 to set up for Row Programming.			
CMDEXEC	00	0x84010003	MOV.SL #0x4000, W1
CMDEXEC	00	0x00000301	MOV.L W1, W0
CMDEXEC	00	0x8A900421	MOVS.W #0x4002, [W9]
Step 3: Load data into the RAM buffer.			
CMDSEQWR	10	Data[31:0]	MOV.L #PGEDx<31:0>, [W0++]
Step 4: Repeat Step 3 until all row data are moved to the RAM buffer. 128 total CMDSEQWR commands must be issued, each generating 32 bits of data for ascending RAM locations.			
Step 5: To prepare for WR bit polling, move NVMCON to VISI.			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
Step 6: Move NVMCON to VISI. Shift VISI on PGEDx.			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
Step 7: Repeat Step 6 to poll WR bit (NVMCON[15]) until it is clear, indicating completion of the previous Row Programming operation.			
Step 8: Write the target Flash row address to NVMADR, the RAM source address to NVMSRCADR and then set WR bit (NVMCON[15]) to begin hardware Row Programming.			
CMDEXEC	00	0x94030195	MOV.L W1, NVMSRCADR
CMDEXEC	00	0x8000C013	MOV.SL #NVMADR, W0
CMDSEQWR	10	Destination Address[31:0]	MOV.L #PGEDx<31:0>, [W0++]
CMDEXEC	00	0x8E900421	MOVS.W #0xC002, [W9]
Step 9: Change the RAM buffer address to continue data loading while NVM controller programs the previous buffer. The address will be switched between 0x4000 and 0x4200. Load the new starting address of the RAM buffer to W0.			
CMDEXEC	00	0x03014491	BTG.L W1, #9
			MOV.L W1, W0
Step 10: Repeat Steps 3-9 for any additional rows of data available for programming.			
Step 11: To prepare for WR bit polling for the last Row Programming, move NVMCON to VISI.			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
Step 12: Move NVMCON to VISI. Shift VISI on PGEDx.			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
Step 13: Repeat Step 12 to poll WR bit (NVMCON[15]) until it is clear, indicating programming completion for the last row.			

3.5 Read Memory

This algorithm (Table 3-5) may be used to read from any implemented memory space, including DEVID, DEVREV, device registers, RAM, User Flash, User OTP, UCA and UCB Configuration Words and SFRs. A 32-bit aligned read address is required.

Attempts to read from unimplemented memory, reserved addresses, regions protected by code-protect or persistent security restrictions defined by UCB security region descriptors will return zero.

Table 3-5. Read Memory Algorithm

ICSP Sequence	ICSP Command Code	Data/Opcode	Instruction Executed
Step 1: Initialize Transfers Destination Pointer (W8) to the VISI register.			
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
Step 2: Initialize pointer to read from.			
CMDEXEC	00	0x80000003 | (Read Address [23:0] << 2)	MOV.SL #ReadAddress<23:0>, W0
Step 3: Issue a dummy CMDSEQRD to copy the first read data into VISI. Throw the data returned by this dummy operation away as they are the existing contents of VISI, not the data read from Read Address[23:0].			
CMDSEQRD	11	VISI	MOV.L [W0++], [W8]
Step 4: Read the VISI register using CMDSEQRD.			
CMDSEQRD	11	VISI	MOV.L [W0++], [W8]
Step 5: Repeat Step 4 until all sequential data have been read from the target device. To update the address for a random access, return to Step 2.			

3.6 CRC Checksum Calculation

This algorithm (Table 3-6) calculates the 32-bit Cyclic Redundancy Check (CRC) checksum of the Flash memory region. This checksum can be used to verify the written data in the previous programming steps or verify if the existing Flash contents match a known pattern, but which cannot be directly read due to code-protect or security restrictions defined in UCA or UCB. For details about the CRC calculation algorithm, refer to Flash Memory Controller.

The NVM controller CRC calculation can be disabled by Configuration Words in the User Configuration A and B areas.

Table 3-6. CRC Checksum Calculation Algorithm

ICSP Sequence	ICSP Command Code	Data/Opcode	Instruction Executed
Step 1: Move VISI address to W8 and NVMCRCCON address to W9. Set the CRCEN bit (NVMCRCCON[15]) to enable CRC hardware.			
CMDEXEC	00	0x9C00C163	MOV.SL #NVMCRCDATA, W7
CMDEXEC	00	0xA0001F03	MOV.SL #VISI, W8
CMDEXEC	00	0xA400C123	MOV.SL #NVMCRCCON, W9
CMDEXEC	00	0xC2F92008	BSET.L [W9], #15
Step 2: Load the start address in the NVMCRCST register, the end address in the NVMCRCEND register and the initial value in the NVMCRCSEED register.			
CMDEXEC	00	0x8000C133	MOV.SL #NVMCRCST, W0
CMDSEQWR	10	Start Address[31:0]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	End Address[31:0]	MOV.L #PGEDx<31:0>, [W0++]
CMDSEQWR	10	Initial Value[31:0]	MOV.L #PGEDx<31:0>, [W0++]
Step 3: Set the START bit (NVMCRCCON[14]) to start the CRC calculation. Prepare for polling by moving NVMCRCCON to VISI.			
CMDEXEC	00	0xC2E92008	BSET.L [W9], #14
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
Step 4: Move NVMCRCCON to VISI. Shift VISI on PGEDx.			
CMDEXEC	00	0x83892400	MOV.L [W9], [W8]
CMDRD	01	VISI	
Step 5: Repeat Step 4 to poll the START bit (NVMCRCCON[14]) until it is clear, indicating completion.			
Step 6: Move NVMCRCOUT to VISI. Shift VISI on PGEDx.			
CMDEXEC	00	0x83872400	MOV.L [W7], [W8]
CMDEXEC	00	0x00000000	NOP
CMDRD	01	VISI	

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 978-1-6683-0433-4

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.