Migrating to the PIC32MM Microcontroller Family

INTRODUCTION

This document describes many of the architectural features of the PIC32MM devices to ease the migration effort from other devices. This document assumes the reader is knowledgeable about the device they are currently using, and therefore, offers a high-level overview of the PIC32MM family rather than a comprehensive comparison of peripheral differences between various products. This document is intended to complement, not to replace, the Family Reference Manuals and Device Data Sheet.

CPU

The CPU is a 32-bit MIPS® microAptiv™ single-phase, unified Harvard RISC architecture core with a 5-stage pipeline that retires, or completes, one instruction per system clock. Each instruction takes 5 clocks to complete, but because of the pipeline 5 instructions at different stages, they are progressing through the pipeline at the same time. This allows an instruction to complete every clock. The core uses a load and store architecture. This means that data is not operated on while in memory. Instead, data is read from memory into a core register, operated upon and then written back to memory.

The MIPS MCU ASE extension is implemented. This extension reduces interrupt latency and adds atomic read and write instructions. The DSP ASE extension and floating-point coprocessor are not implemented.

CPU Registers and the Zero Register

The CPU does not have a dedicated Accumulator register; instead it contains thirty-two 32-bit General Purpose Registers (GPRs), called Core registers. All of these registers are functionally identical with the exception of the Zero register. Register Number 0, `$0`, is called the Zero register. Reads of the Zero register will always return '`0`'. Writes to the Zero register do not change the Zero register's contents. Some instructions, such as Branch and Link, only operate on a specific non user-selectable register.

By convention, registers are allocated to contain values, such as a function's return value or the Stack Pointer.

Program Counter

MIPS CPUs do not have a directly accessible Program Counter (PC). Instead, the target address, or offset, is loaded into a GPR register and a `JUMP` (J) or `CALL` (JAL) instruction uses the register's contents and the current PC value to generate the target address.

CPU Configuration

The Coprocessor0 (CP0) registers are used to configure the CPU, Interrupt mode and similar operations. The CP0 registers also contain information about the CPU implementation, such as the supported instruction set(s) and the type of hardware multiplier implemented. These registers are accessed through the special instructions, `MFC0` and `MTC0`, that move data between the General Purpose Registers and the CP0 registers.

Data Width

The native data width is 32 bits. Math operations are performed on 32-bit words, and byte and half-word operations are performed by masking the result of a 32-bit operation. Because the maximum instruction length and the data width are the same, the load immediate instruction cannot contain all the possible 2^32-bit values. Therefore, loading immediate values into a register may take multiple instructions to 'build up' the desired value (refer to **"Assembly Language"**). The MPLAB® XC32 compiler supports char (8-bit), short (16-bit), int (32-bit) and long long (64-bit) data types.

Instruction Set, Instruction Width and Orthogonality

Only the microMIPS™ instruction set is supported. This is a mixed dual length, 16-bit and 32-bit instruction set that supports all MIPS32® assembly mnemonics, but is not binary compatible with MIPS32 or MIPS16e® instruction sets. The microMIPS instruction set provides code size reduction by providing 16-bit versions of commonly used instructions. The 16-bit versions do not support all registers, and have reduced offset and value ranges. The compiler uses a 16-bit instruction when possible; otherwise, a 32-bit instruction is used. The instruction length is decoded as part of the fetch operation, therefore, 16-bit and 32-bit instructions can be adjacent, eliminating the need to use calls, as in the MIPS16e implementation, to switch modes. The MPLAB XC32 compiler automatically targets the microMIPS instruction set for this product family. The MIPS DSP ASE is not supported.

The instruction set is orthogonal. With the exception of a few instructions, any instruction can use any CPU register as a source and any register for the result, including the Zero register.

Assembly Language

The microMIPS assembly language contains instructions and multiple pseudo instructions to simplify coding common operations. The pseudo instructions expand into single or multiple instructions, based on the operation and operands (see Example 1).

EXAMPLE 1: EXAMPLE LOAD IMMEDIATE PSEUDO INSTRUCTION

```
LI $1, 0x12345678
```

Can be implemented in two 32-bit instructions by the assembler:

```
LUI $2, 0x1234      # 0 | (0x1234<<16) -> $2
ORI $2, $0, 0x5678  #($2) | 0 | 0x5678 -> $2
```

While the same macro with a smaller operand:

```
LI $2, 0x12
```

Can be implemented in a single 16-bit instruction by the assembler:

```
LI16 $2, 0x12       # 0x12 -> $2
```

No Operation Instructions (NOP)

The MIPS CPU does not have a dedicated NOP instruction. Due to the Zero register, there are multiple math operation instructions that execute but produce no visible effect, such as adding a value to the Zero register, thereby providing NOP functionality. By convention, SLL32 $0, $0, 0 is used for the 32-bit NOP, NOP32, and MOVE16 $0, $0 is used for the 16-bit NOP, NOP16. MPLAB XC32 follows this convention and displays a NOP when these instructions are disassembled. Some instructions must be followed by a NOP32, while others must be followed by NOP16.

User and Kernel Modes

Only Kernel mode is supported. In Kernel mode, firmware has direct access to the peripherals.

Bit Operations

Atomic single bit and multi-bit operations are supported by the CPU and hardware.

Atomic single bit operations are supported with the ASET and ACLR instructions. These instructions disable interrupts, perform a multi-clock cycle Read-Modify-Write operation and then re-enable interrupts. These are multi-cycle instructions. Due to the atomic nature, they can be used to implement semaphores, but should only be used to access RAM memory, not peripherals.

Non-atomic single and multi-bit operations, such as T1CONbits.ON, generate an instruction sequence to perform a read, a logical OR operation and a write-back to the register. To provide this functionality as an atomic operation without the Read-Modify-Write sequence, most peripherals support hardware Set/Clear/Invert registers, collectively referred to as SCI registers. The registers are addressed as offsets to the peripherals' register address, +0x4, +0x8, +0xC, and are named, xCLR, xSET, and xINV, accordingly (see Example 2). For example, the Timer1 ON bit can be set via T1CONSET = (1<<15). The SCI registers do not exist as registers, but signal hardware to perform the operation on the desired register using the provided mask pattern. Multiple bits can be operated simultaneously based on the mask value used. The SCI registers are write-only; the values read from them are invalid. Interrupt flags should only be cleared using the SCI registers. This is to prevent inadvertently clearing an interrupt that was asserted during a Read-Modify-Write sequence.

EXAMPLE 2: SCI ADDRESS OFFSETS AND NAMES FOR LATA

```
0xBF802BE0 LATA
0xBF802BE4 LATACLR
0xBF802BE8 LATASET
0xBF802BEC LATAINV
```

Shift and Rotate Operations

Shift and rotate operations operate on 32 bits, can be performed on any CPU register and have a range of 0-31 bits.

Multiply and Divide Unit (MDU)

The CPU has an autonomous Multiply and Divide Unit (MDU) that can execute a multiply or divide in parallel with CPU code execution. The multiply engine performs a 32-bit by 16-bit MAC in 1 clock cycle, or a 32-bit by 32-bit signed or unsigned multiply in two clock cycles. The divide engine can perform a 32-bit by 32-bit divide, signed or unsigned, in 11 to 34 clock cycles based on the data size. No hardware floating point is implemented; these operations are emulated by software libraries.

CPU Math Status Flags

The MIPS CPU does not implement math status flags, such as Borrow, Carry or Zero; instead, compare instructions are used to determine if the result stored in a register was zero, greater than, or less than zero (refer to Example 3).

EXAMPLE 3: TEST FOR RESULT EQUALS ZERO

```
# Sum contents of registers. ($7 + $8) -> $6
ADDU $6, $7, $8
# Test and branch to EqualsZero:
BEQZ $6, EqualsZero
```

Core Timer

The MIPS CPU has a general purpose timer (core timer) that counts at ½ the CPU clock rate. The core timer is accessed through the CP0 registers (refer to Example 4 and Example 5). The timer can be configured to generate an interrupt when a specified value is reached.

EXAMPLE 4: SET/READ CORE TIMER

```
//Set Core Timer
unsigned int counts = 0x12345678;
asm volatile("mtc0   %0, $9": "+r"(counts));

//Read Core Timer
asm volatile("mfc0   %0, $9": "=r"(counts));
```

EXAMPLE 5: SET/READ CORE TIMER WITH MACROS

```
//Set Core Timer
unsigned int counts = 0x12345678;
_CP0_SET_COUNT(counts);

//Read Core Timer
counts = _CP0_GET_COUNT();
```

Performance Counters

The MIPS microAptiv CPU supports two performance counters. These counters can be configured to count CPU instructions completed, CPU cycles, branch instructions executed and other metrics for code profiling. The performance counters are part of the CP0 registers, and are accessed using the `MTC0` and `MFC0` instructions or through the MPLAB XC32 macros (see Example 6).

EXAMPLE 6: CONFIGURE COUNTER

```
/ configure counter
#define  PCNTR0_MODE_NOPS (17<<5)
#define  PCNTR_CNT_KERNEL (1<<1)
_CP0_SET_PERFCNT0_CONTROL(PCNTR0_MODE_NOPS | PCNTR_CNT_KERNEL
);

_CP0_SET_PERFCNT0_COUNT(0);          // reset counts to 0
// user code
counts = _CP0_GET_PERFCNT0_COUNT();  // read counts
```

MEMORY AND BUS MATRIX

Wait States

Both Flash and RAM are zero Wait state access across the device operating frequency range. The Flash and RAM Wait state control bits are not implemented.

Prefetch and Cache

No prefetch or cache is implemented. A line buffer is used to hold a Flash line of data, 64 bits after it has been fetched. If the next instruction to fetch exists in the line buffer, it is read from the line buffer instead of generating another Flash fetch.

Execution from RAM

Because of the unified memory map, RAM memory is mapped in the same manner as Flash memory. Therefore, code can execute from RAM. When executing from RAM, the instruction and data buses must arbitrate for the access to RAM memory, generating core Stalls. To execute code from RAM, the memory must be logically partitioned into instruction and data memory using the EXECADDRx bits (see Example 7). The default partition is all program memory. The compiler will automatically allocate the required memory when a function is placed in RAM using the `__longramfunc__` attribute.

EXAMPLE 7: MANUALLY SETTING THE RAM DATA/CODE PARTITION

```
CFGCONbits.EXECADDR = 1;  // 1k of RAM is
                          // dedicated to
                          // instruction memory
```

Flash Memory and ECC Error Correction

Flash memory is 64 bits wide with additional bits reserved for ECC error correction. All Single Bit Errors (SBE) in Flash can be detected and corrected. All Double-Bit Errors (DBE) can be detected, but not corrected, and generate a bus error exception (data load). Due to the ECC bits, Flash data must be written as a 64-bit aligned, 64-bit double word or an aligned row. Rewriting the data in Flash without a preceding erase operation violates the Flash specification and will, if data is different from the previous data, generate an ECC error when the data is read. ECC errors are only detected during a read operation.

Unified Memory Map

The CPU has a 4-Gbyte linear address range. There are no page or bank select bits for memory. The unified memory maps contains Flash, RAM and peripherals in a common address range. This memory map is divided into multiple sections, called segments. Some of the segments alias other segments to provide compatibility with devices that support caches.

Flash Programming

The self-programming of Flash memory can be performed via double-word writes or row writes. When a Flash write operation is in progress, any Flash accesses by the CPU or DMAs will stall until the Flash write operation completes. The Flash controller has an independent hardware lock mechanism, similar to SYSKEY, to prevent inadvertent operations. If the CPU is executing from RAM, it can continue executing while Flash is being programmed. The CPU will, however, have to arbitrate with the Flash controller for RAM access if a row write is being performed.

Physical and Virtual Memory Addresses

Physical memory addresses are in the address range of: `0x0000_0000` through `0x7FFF_FFFF`. This contains the address of all accessible memory and peripherals on the device. Physical addresses are only used by DMA peripherals, including the Flash controller, stand-alone DMA and the USB DMA.

The CPU accesses memory and peripherals through virtual addressing. In virtual addressing, the contents of the physical memory map are accessed through alias addresses, called segments. These segments can have unique attributes based on the CPU configuration. The FMT translates the virtual addresses to physical addresses.

Memory Segments (ksegx)

The contents of physical memory are accessible via multiple addresses, called segments. Each segment can have unique attributes. The upper 3 bits of the virtual address are used to determine the segment. kuseg addresses are identical to the physical addresses. Only kseg1 and kseg0 are supported by this implementation. kseg1 is non-cacheable memory and kseg0 is cacheable. These regions are functionally identical because a cache is not implemented. To improve code portability, the cacheable/non-cacheable convention should be followed. Memory to be accessed by both a DMA and the CPU should be non-cacheable (kseg1) to prevent coherency issues. Peripherals should be accessed as non-cacheable (kseg1). Memory accessed only by the CPU (variables) should use cacheable memory (kseg0).

Fixed Map Translation Table (FMT)

The FMT is used to translate virtual addresses into physical addresses for CPU reads and writes. The operation of the FMT is transparent to the user. The address supplied is used by the FMT to determine the physical address for the desired operation.

Bus Matrix (BMX)

The bus matrix is a crossbar switch that connects the CPU instruction bus, CPU data bus, peripherals and memory. The CPU, Flash controller, general purpose DMA and USB DMA are referred to as initiators because they initiate reads or writes. Flash memory, RAM and peripherals are called targets because data is written to, or read from them. All peripherals are accessed through a single target. Some peripherals, such as DMA, are both a target and an initiator. The BMX allows multiple initiators to perform concurrent accesses to targets as the long they are not the same target. If the targets are the same, arbitration occurs. For example, a DMA can move data from a peripheral to RAM, while the CPU is fetching instructions, or data from Flash. If both the DMA and CPU attempt to access the same target, arbitration occurs, stalling the CPU or DMA.

INTERRUPTS

Exceptions

Exceptions are a generic term for interrupts. Exceptions can be generated by a peripheral interrupt, instructions, address errors and math overflow. The cause of the exception is contained in the Cause register (CP0.13) and the address of the exception is stored in the Exception Return register (EPC, CP0.14).

Exceptions include:

- Peripheral interrupts
- Address error exception (load or instruction fetch)
- Address error exception (store)
- Bus error exception (instruction fetch)
- Bus error exception (data reference: load or store)
- SYSCALL exception instruction
- Software Breakpoint exception
- Reserved instruction exception (not decoded as a NOP)
- Coprocessor unusable exception (only CP0 is supported)
- Arithmetic overflow exception
- Trap exception (conditionally generated by the result of a TRAP instruction)

EXAMPLE 8: EXCEPTION HANDLER

```
volatile unsigned int epc, cause;

void     __attribute__((naked)) _general_exception_handler(void)
{


    // determine the source of the issue
    epc = _CP0_GET_EPC();
    cause = (((_CP0_GET_CAUSE()) & 0x0000007C) >> 2);

    while(1);
}
```

Interrupts

Peripheral interrupt sources have a unique interrupt vector and Interrupt Request (IRQ) for each interrupt generation source. Each IRQ has a user-selectable priority and sub-priority. Multivector and single vector interrupt processing are supported.

INTERRUPT SERVICE ROUTINE (ISR) FUNCTIONS

The compiler provides attributes to determine implementation of an ISR function (Example 9).

EXAMPLE 9: ISR TO CONFIGURE AN ISR FOR TIMER1 ROLLOVER

```
unsigned int val = 0;

// Timer configuration is not shown

// set interrupt priority and enable peripheral interrupt
IPC1bits.T1IP = 7;                                  // set Timer1's interrupt priority to 7
IPC1bits.T1IS = 0;                                  // set Timer1's sub-priority to 0
IEC0bits.T1IE = 1;                                  // enable timer1 int

PRISS = 1<<28;                                      // pair IPL7 with Shadow Set 1

// enable of multi-vector mode is done by start-up code. The code below is not required.
asm      volatile("mfc0 %0,$13":"=r"(val));         // set the CP0 cause IV bit high
val |= 0x00800000;
asm      volatile("mtc0   %0,$13" : "+r"(val));
INTCONSET = _INTCON_MVEC_MASK;

__builtin_enable_interrupts();                      // XC32 macro to enable interrupts

void __ISR(_TIMER_1_VECTOR, IPL7SRS) TimerISR(void)  // ISR
{
IFS0CLR = (1<<17);                                  // clear the timer interrupt flag
// user code
}
```

INTERRUPT PRIORITIES

Each interrupt source, IRQ, has an associated priority, sub-priority and a fixed natural priority. The interrupt priority is used to determine interrupt preemption. An interrupt with a higher priority will preempt an interrupt with a lower priority. The sub-priority is only used when multiple interrupts of the same priority are pending. If multiple interrupts with the same priority and sub-priority are pending, the natural order (the lowest IRQ number) is used to determine the higher priority. Eight levels of priority and 4 levels of sub-priority are available. The IRQ priority and sub-priority are set using bits in the IPCx registers. Interrupt priority has no effect on interrupt latency, assuming no higher priority interrupts that would cause preemption, are pending.

ENABLING INTERRUPTS

Interrupts can be globally and individually enabled and disabled. The individual interrupt enable bits are contained in the IECx registers. Interrupts can be globally enabled and disabled with the 'ei' and 'di' instructions, or by using compiler macros, `__builtin_enable_interrupts()` and `__builtin_disable_interrupts()`. For an interrupt source to generate an interrupt, interrupts must be globally enabled, the interrupt's priority must be greater than zero and the individual interrupt must be enabled.

MULTIVECTOR, SINGLE VECTOR INTERRUPTS AND SPACING

In Multivector mode, each IRQ has a corresponding unique vector address for the interrupt handler. The distance between all the vectors, the maximum Interrupt Service Routine (ISR) length, can be set by the user. The user can specify the ISR code to be located at the vector address, or to use a JUMP instruction at the vector address targeting the actual ISR code, thereby reducing the space used by the vector table. By default, a jump table is created at the vector address to reduce the size of the vector table. The device start-up code enables Multivector mode.

CLEARING INTERRUPTS

Interrupts should only be cleared using the interrupt controller's Interrupt Flag Clear (IFSxCLR) registers. A software Read-Modify-Write operation to the IFSx register would overwrite any interrupt flag that was set between the read and write operation, thereby masking that interrupt. The operation using the Clear register is also faster to execute and uses fewer instructions (Example 10).

EXAMPLE 10:

```
IFS0CLR = 1<< _IFS0_CTIF_POSITION;
// Clear the Core Timer interrupt
```

SHADOW SETS

One hardware shadow set is available for interrupt processing. This is a second set of CPU Core registers, associated with a particular interrupt priority, using parameters in the ISR declaration and the PRISS register. When an interrupt associated with the shadow set occurs, the shadow set replaces the Core registers to eliminate saving Core registers to the stack. For interrupts that do not use the shadow set, the Core registers will be saved on the stack. Typically, the shadow set is used with the highest interrupt priority to minimize that interrupt's latency. The PRISS register is used to pair an interrupt priority to the shadow set.

INTERRUPT LATENCY

The hardware interrupt latency is 8 clocks. This is the time from when an interrupt is generated until the CPU retires and completes execution of the first instruction in the ISR prolog. The software latency depends on the ISR type. For hardware shadow sets, there is no additional latency for the Core registers, but some CP0 registers may be saved by the interrupt prolog. For software shadow sets, the latency can be 20 or more clocks because multiple Core registers must be pushed onto the stack. Implementing a jump table at the ISR vector adds an additional 2 clock cycles for the JUMP instruction.

PERSISTENT AND NON-PERSISTENT INTERRUPTS

Peripherals that use a FIFO with a watermark as an interrupt source and comparator have persistent interrupts (UART and SPI). The interrupt is not clearable until the cause of the interrupt is serviced. This type of interrupt data must be read from the FIFO until it is below the watermark. After the interrupt cause is serviced, the interrupt flag can be cleared. The interrupt is reasserted when the watermark threshold is exceeded

Non-watermark interrupt flags can be cleared before reading the appropriate register. Clearing the interrupt flag before reading the registers also reduces the time in which a second interrupt can occur before the flag is cleared and not detected.

NON-MASKABLE INTERRUPT (NMI)

When an NMI occurs, program execution jumps to the NMI handler. If the user does not specify a handler, the weak default handler is used that returns from the NMI. The NMI vector is shared between multiple NMI sources, wake from Sleep, WDT event and Reset. In the case of the WDT event, the NMI timer is started. If the NMI timer expires before the NMI is serviced, a device Reset will occur. The time-out period of the NMI timer in system clocks can be set with the RNMICON register.

SYSTEM

Peripheral Bus (PB)

Only a single Peripheral Bus is implemented. The PB divisor is fixed at 1:1.

System Key (SYSKEY)

To prevent unintended access to certain registers, oscillators and Resets, they are protected by a hardware locking mechanism. To unlock these registers, the key value, 0xAA99665 5, and its inverse must be sequentially written to the SYSKEY register. Any accesses to other system registers between these writes will abort the unlock operation. A write of a non-sequence value will also relock the registers. To prevent this, interrupts and DMA accesses should be disabled/suspended prior to using the unlock sequence. The SYSKEY register can be read to determine the lock status. The system is locked automatically following any Reset.

EXAMPLE 11: UNLOCK SEQUENCE

```
SYSKEY = 0x00000000;
// force lock, reset sequence

SYSKEY = 0xAA996655;   // unlock sequence
SYSKEY = 0x556699AA;
```

Power Save Modes

Two types of power save modes are available; Sleep and Idle.

There are multiple modes for Sleep depending on the desired power consumption and wake-up time requirements. Most clocks are disabled in Sleep. Only select peripherals, including the ADC and Change Notice (CN), can operate in Sleep mode.

Retention Sleep is a lower power mode than Sleep. A dedicated low-voltage regulator is used to maintain state and RAM contents. Due to the lower operating voltage of this mode, the wake-up time is longer than Sleep mode.

There is a single Idle mode, but most peripherals have a software writable bit, SIDL, that determines if the peripheral is active in Idle mode. Clock sources that were enabled in Run mode continue to run in Idle. The CPU stops executing instructions until a wake-up event occurs. The stand-alone DMA peripheral can perform transfers while the device is in Idle.

The device enters power save modes via a `WAIT` instruction. The mode entered depends on the status of the SLPEN bit in the RCON register and the selected Sleep mode. A wake from Sleep or Idle is a NMI event. After waking up, the CPU jumps to the NMI handler. The default NMI handler will return code execution to the instruction following the `WAIT`.

Hardware Trace

Hardware trace is not supported by the PIC32MM device family.

Debug Breakpoints

Four instruction and two data breakpoints are supported. Data breakpoints can be configured to occur on access to a particular location or reading/writing a specific value to the particular location. Software breakpoints are available as a BREAK16 instruction. Complex breakpoints are not supported.

Peripheral Pin Select (PPS)

Full PPS allows the mapping of inputs and/or outputs of select peripherals to any device pin that has an RPn function. PPS peripherals are not grouped as is required by PPS Lite.

Dual Watchdog Timer (WDT) and Determinism

The WDT has two timers; one for operation in Run mode and the other for operation in Sleep mode. The Run mode WDT increments at the rate of the user-selected clock. The Sleep mode WDT operates using the LPRC clock source. The DMA peripherals arbitrating for memory and peripheral access may reduce the number of CPU instructions executed within a given time period. This can make the CPU execution appear non-deterministic, and therefore, cause unexpected WDT time-outs when code is expected to clear the WDT near the end of the period or window.

Oscillators and Clocks (FOSC vs. FCY)

Due to the single phase nature of the CPU FCY, the system clock is 1:1 with FOSC. No divisors are used.

Clock Out

The signal on the clock out pin is derived from the PB clock. It is the same frequency as the system clock.

Reference Clock (REFO)

The REFO peripheral divides a user-specified input clock with a user-selected integer divider to create a wide range of integer divided output. The output of this peripheral is available on a GPIO pin and as the clock input to select peripherals. This can be used to generate frequencies from common device clocks, such as the 48 MHz USB clock. Fractional output divide is not supported.

Software Reset

The CPU does not have a RESET instruction. A software Reset is performed by writing a value to, and reading back from, a Reset Control register, RSWRST (see Example 12). The Reset Control register is hardware locked by SYSKEY. The Reset sequence should be followed by four NOPs or a while(1) instruction to prevent any write operations during the Reset sequence.

EXAMPLE 12:

```
unsigned    int temp;
SYSKEY = 0x00000000;
// force lock, reset sequence

SYSKEY = 0xAA996655;    // unlock sequence
SYSKEY = 0x556699AA;
RSWRST = 1;             // write the reset bit
temp = RSWRST;
// read operation will generate a reset

while(1);               // wait for the reset
```

Direct Memory Access (DMA) and Peripherals with DMA

DMA engines are used to transfer data in either direction between Flash, RAM and peripherals. The only operation not supported by the stand-alone DMA is a write to Flash. All DMA operations go through the system bus matrix, and therefore, arbitrate with the CPU and other DMA(s) for bus access. DMA transfers are configured using physical memory addresses only.

The general purpose DMA peripheral is a stand-alone DMA used to transfer data without CPU intervention. Multiple DMA channels are implemented in the single DMA controller. Data transfers occur based on the DMA channel trigger event and the channel priority. Only one DMA channel can actively transfer data at a given time. All devices in the family do not have the DMA peripheral.

The general purpose DMA also has an integrated programmable CRC engine that can be linked to a DMA channel.

The USB peripheral contains a dedicated linked list-based DMA engine. The descriptors and data transfer configurations are stored in system RAM. All devices in the family do not have the USB peripheral.

The Flash controller contains a dedicated non-programmable DMA engine used to transfer data from system RAM to Flash during row write operations. This is the only DMA that can perform Flash write operations.

Cyclic Redundancy Check (CRC) Engine

The CRC engine is a programmable CRC calculator used to offload calculations from the CPU. The CRC is part of the general purpose DMA or a similar stand-alone CRC module for devices without DMA.

REFERENCES

MIPS® Architecture for Programmers Volume I-A: Introduction to the MIPS32® Architecture, Revision 5

MIPS® Architecture for Programmers Volume II-B: The microMIPS32™ Instruction Set

MIPS32® M14K™ Processor Core Data Sheet

PIC32MM0064GPL036 Family Data Sheet (DS60001324)

PIC32 Family Reference Manual, **"Section 50. CPU for Devices with MIPS32® microAptiv™ and M-Class Cores"** (DS60001192)

MPLAB® XC32 C/C++ Compiler User's Guide, **"Chapter 14. Interrupts"** (DS50001686)

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

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Trademarks

ISBN: 978-1-5224-0711-9

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Dusseldorf
Tel: 49-2129-3766400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Venice
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Poland - Warsaw
Tel: 48-22-3325737

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

07/14/15