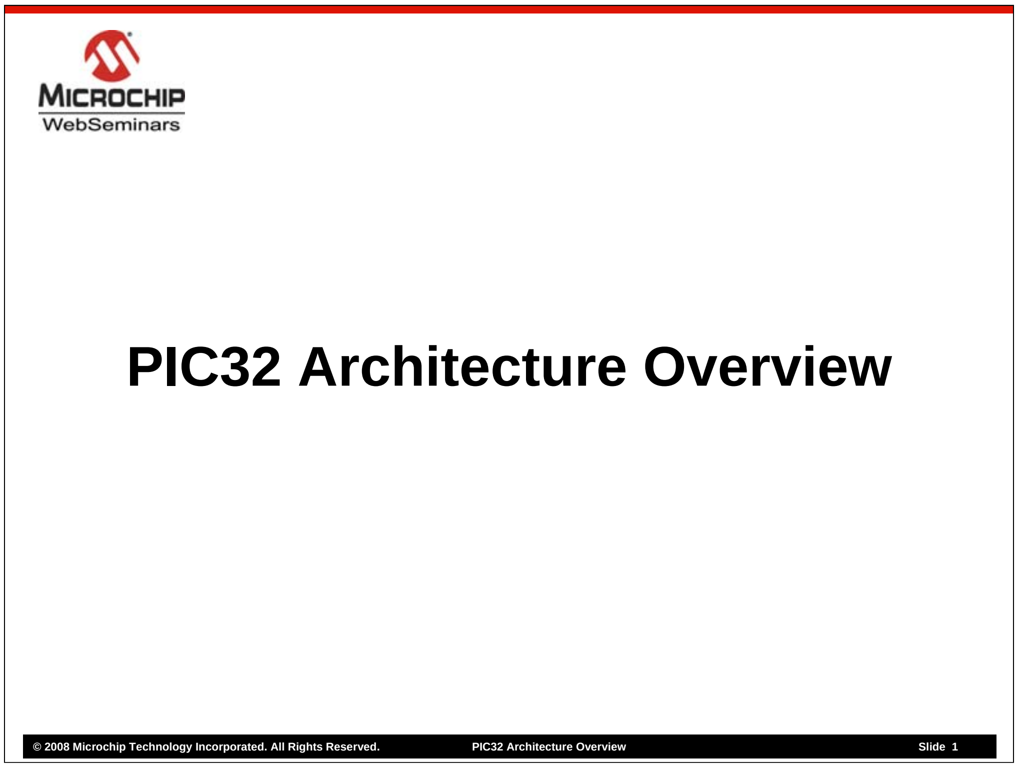


PIC32 Architecture Overview

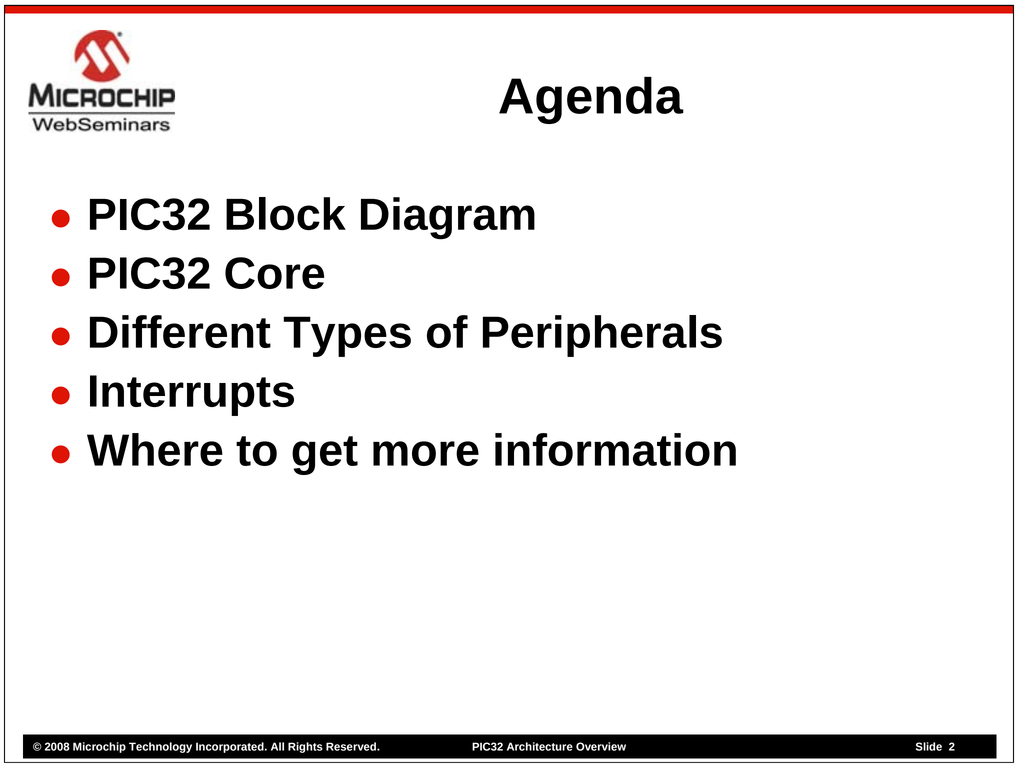
PIC32 Architecture Overview Slide 1

Hello and welcome to the PIC32 Architecture Overview webinar.

My name is Nilesh Rajbharti. I am responsible for managing the 32-bit Microcontroller Applications Engineering department at Microchip Technology. I joined Microchip in the year 2000 and I have had opportunity to work with PIC18, PIC24 and most recently the 32-bit PIC® microcontroller, PIC32.

In about 15 minutes, I will provide a quick overview of the PIC32 architecture and some of its key features. I will not focus on specific PIC32 devices, but instead provide you with architectural details that are common to all PIC32 devices. At the end of the presentation, I will provide you with references from which you can learn more specifics about the device of your choice.

Let’s begin.


MICROCHIP WebSeminars

Agenda

- PIC32 Block Diagram
- PIC32 Core
- Different Types of Peripherals
- Interrupts
- Where to get more information

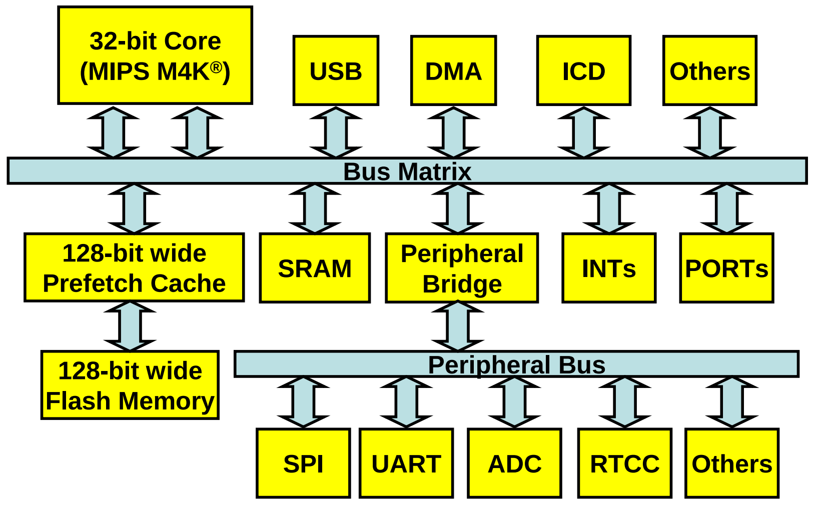
PIC32 Architecture Overview Slide 2

As for the exact agenda, I will show you several block diagram views of the PIC32 architecture, and discuss major features in somewhat more detail. Finally, I will provide you with some pointers to help you get more information.

With that, let's begin with the high level block diagram of the PIC32.

Microchip WebSeminars

PIC32 Block Diagram

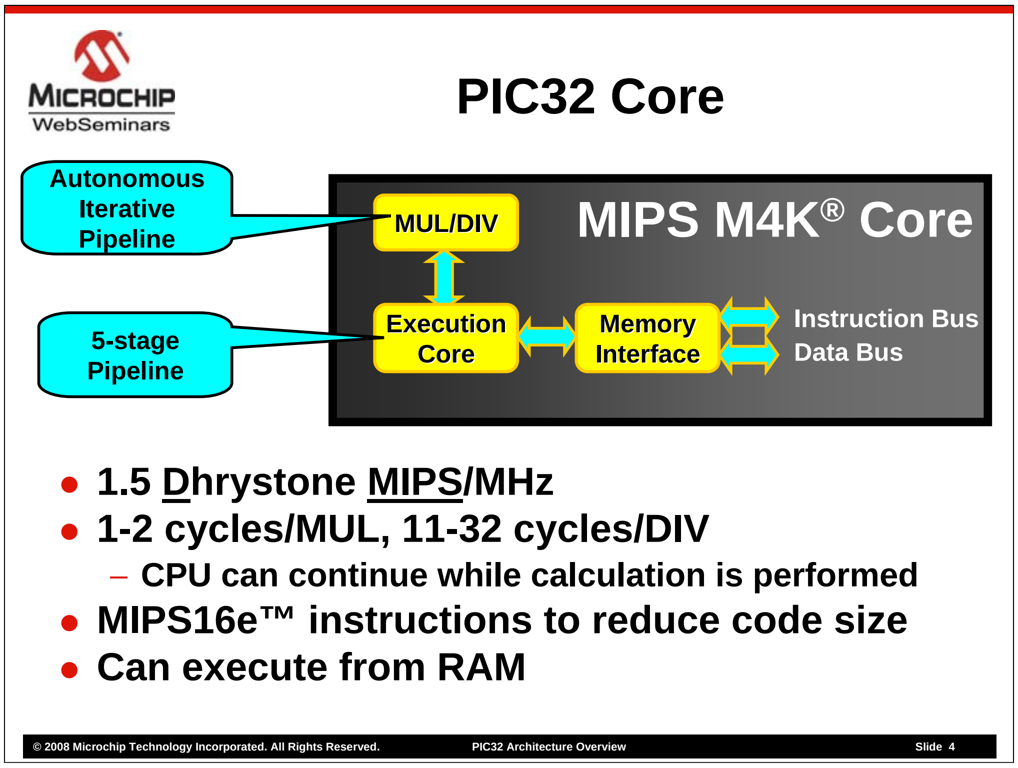


This is a simplified view of the PIC32 chip. The PIC32 employs the M4K® 32-bit core from MIPS Technologies. The M4K is a Harvard architecture based core. It contains separate Instruction and Data busses connected to the Bus Matrix. I will provide you with more information about M4K in next slide.

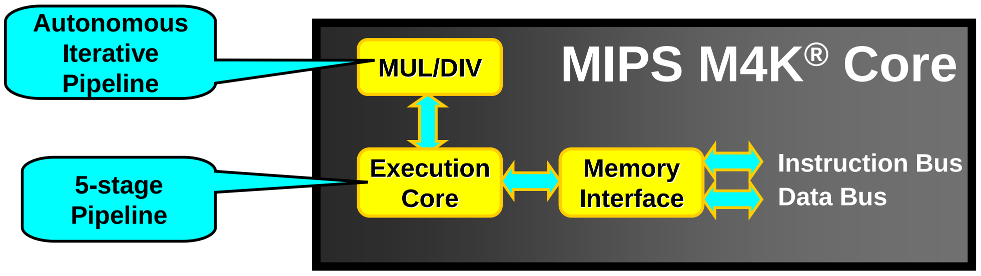
The core connects to the rest of the modules via Bus Matrix. The Bus Matrix is a high-speed switch. It establishes a point to point connection between modules. Modules such as the CPU core, USB and DMA connect to the SRAM, SPI, UART, etc., via the Bus Matrix and Peripheral Bus. The Bus Matrix runs at the same speed as the CPU, while the Peripheral Bus can be programmed to run at a different clock than the CPU. The exact Bus clock is determined by the Peripheral Bridge setting.

In this block diagram, notice that the PIC32 uses a 128-bit wide Flash memory. Such a wide memory path is specifically designed to increase the instruction throughput and improve overall CPU performance. To further enhance the performance, the PIC32 employs a 128-bit Prefetch Cache module. This module can be programmed to look ahead and prefetch the next 128-bits of instructions and store them in an on-chip cache memory. This module is the reason why the PIC32 can continue to provide high performance even when the CPU is running faster than Flash memory speed.

Now let's review the block diagram one section at a time. We will begin with the core.



PIC32 Core



- 1.5 Dhrystone MIPS/MHz
- 1-2 cycles/MUL, 11-32 cycles/DIV
  - CPU can continue while calculation is performed
- MIPS16e™ instructions to reduce code size
- Can execute from RAM

© 2008 Microchip Technology Incorporated. All Rights Reserved. PIC32 Architecture Overview Slide 4

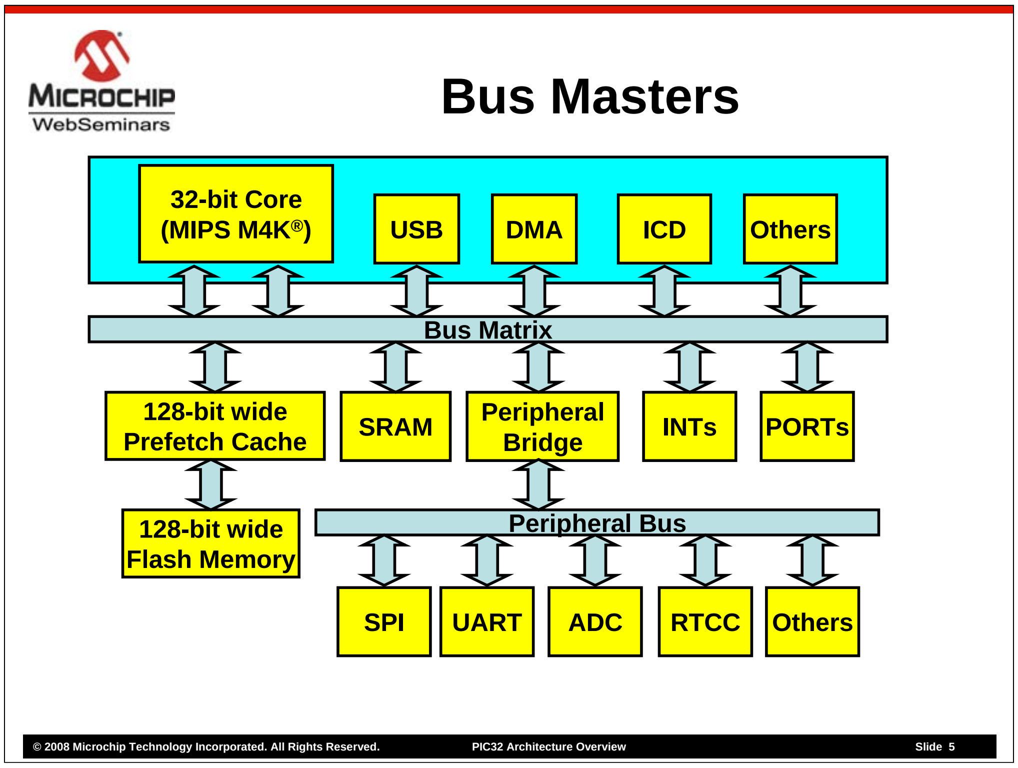
Here is the inside of the M4K® core.

The M4K core uses a 5-stage execution pipeline. This means that each instruction is executed in 5 different stages. Once the pipeline is full, the M4K core executes one instruction per CPU clock. MIPS Technologies rates its M4K core at 1.5 Dhrystone MIPS/MHz. As a microcontroller user, you should be more interested in knowing the Dhrystone performance of the entire chip – i.e., core and the microcontroller memory system put together. We, at Microchip Technology, have conducted our own Dhrystone tests of the PIC32 and confirmed that the PIC32 also offers 1.5 Dhrystone MIPS/MHz at 0 Wait State Flash operation. As with any other microcontrollers with slow Flash memory, when running faster than the Flash speed, the Dhrystone rating would drop. However, in the case of PIC32, the on-chip Prefetch Cache and high-speed SRAM minimize that performance drop.

When it comes to the memory mapping method, the PIC32 uses the unified memory map – meaning that both Instruction and Data space reside in one linear address space, each occupying a unique range of addresses. With this scheme, you as a programmer will use one address pointer to access both Instruction and Data memory areas. Another point to note is that the PIC32 core can execute from RAM. Typical Harvard architectures do not allow execution from RAM, but the PIC32 includes a special bus matrix configuration that allows it to make part of the RAM executable.

The PIC32 uses the high-performance version of the Multiply and Divide hardware module. A very powerful feature of this module is that it contains its own autonomous pipeline. As a result, once the CPU issues a multiply or divide instruction, the CPU may continue to fetch and execute next instructions while the multiply and divide unit performs calculations in parallel. If the CPU tries to access the result before the multiply or divide operation is complete, the CPU will stall until the operation is complete. There are different cycle counts for multiply and divide operations. It takes 1 cycle to perform 16x16 or 32x16 multiply operations, and 2 cycles for other sizes. The divide operation takes from 11 to 32 cycles. Exact cycle count depends on the dividend operand size. The smaller the dividend operand, the shorter the divide operation.

By default, the PIC32 executes 32-bit instructions. The 32-bit instructions are designed to provide higher performance. If the application is code size sensitive, it may use MIPS16e™ instructions. The MIPS16e instructions are 16-bit wide. With the use of MIPS16e instructions, applications can save up to 40% of code size compared to the 32-bit instructions. There will be a reduction in performance when using MIPS16e instructions; however, with the 128-bit wide prefetch cache, some applications see no adverse impact.

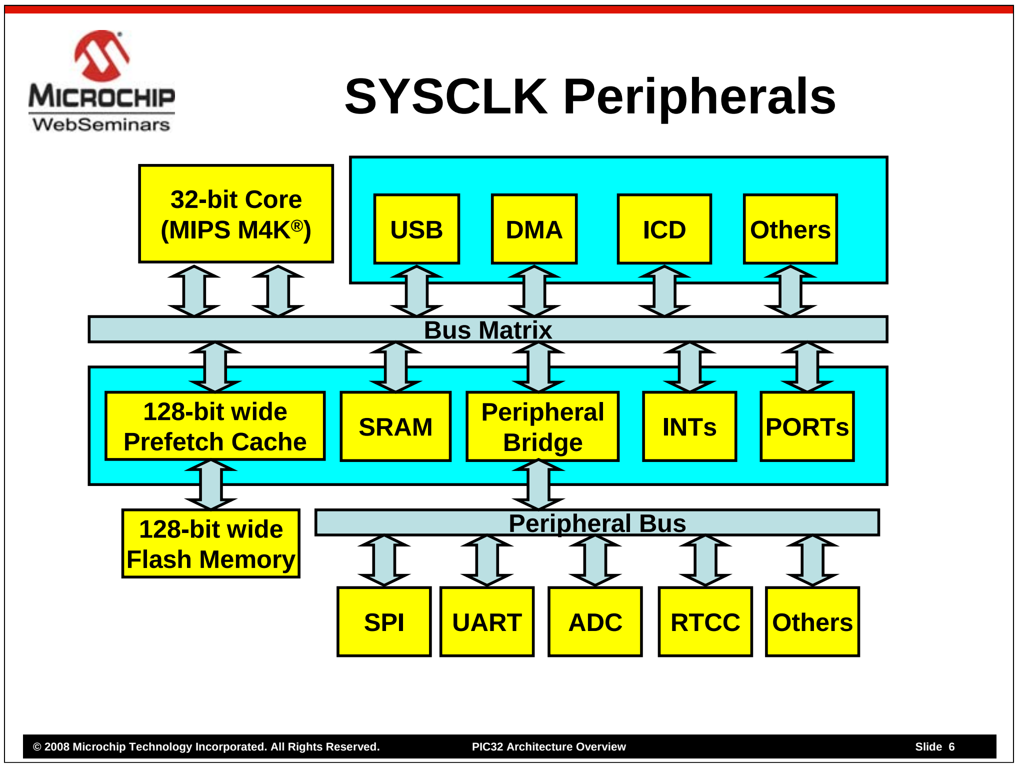


Bus Masters

PIC32 Architecture Overview Slide 5

The PIC32 architecture uses a concept called Bus Master modules. The Bus Masters are a special set of modules that can initiate a read or write transaction of other modules called “Targets”. For example, the CPU can read and write to SRAM or any other peripheral. Similarly, the DMA can read and write to any other peripherals on the bus. At present, CPU, ICD, USB, and DMA are the Bus Masters in PIC32 architecture. Future PIC32 products may add more Bus Master modules.

The Bus Master modules run at the same speed as the CPU. All Bus Masters, except the CPU, essentially have an integrated “DMA” capability to autonomously perform reads and writes of a peripheral. They can transfer data within the microcontroller or outside of the microcontroller without any assistance from the CPU. The Bus Masters may read and write other Bus Masters too. For example, the DMA module may read or write USB registers. However, the Bus Master cannot access core registers in the CPU. Only the CPU can access the core CPU registers.

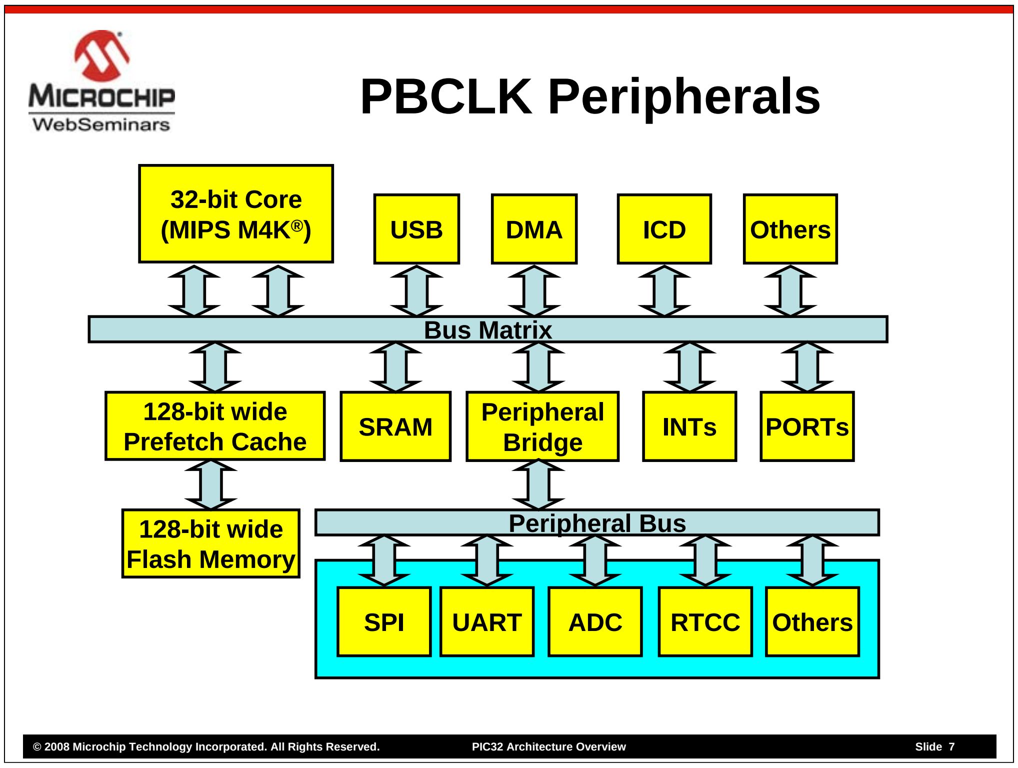


SYSCLK Peripherals

PIC32 Architecture Overview Slide 6

Peripherals such as the Prefetch Cache, USB, DMA, SRAM, Interrupts and I/O PORTs are called SYSCLK Peripherals. These peripherals run at the same speed as the CPU and other Bus Masters. As a result, all accesses to the SYSCLK peripherals complete in one cycle. Typically peripherals with high data throughput are placed on the SYSCLK bus.

Note that the I/O PORT modules are also on the SYSCLK bus. This means that CPU can access I/O PORTs at max operating frequency.

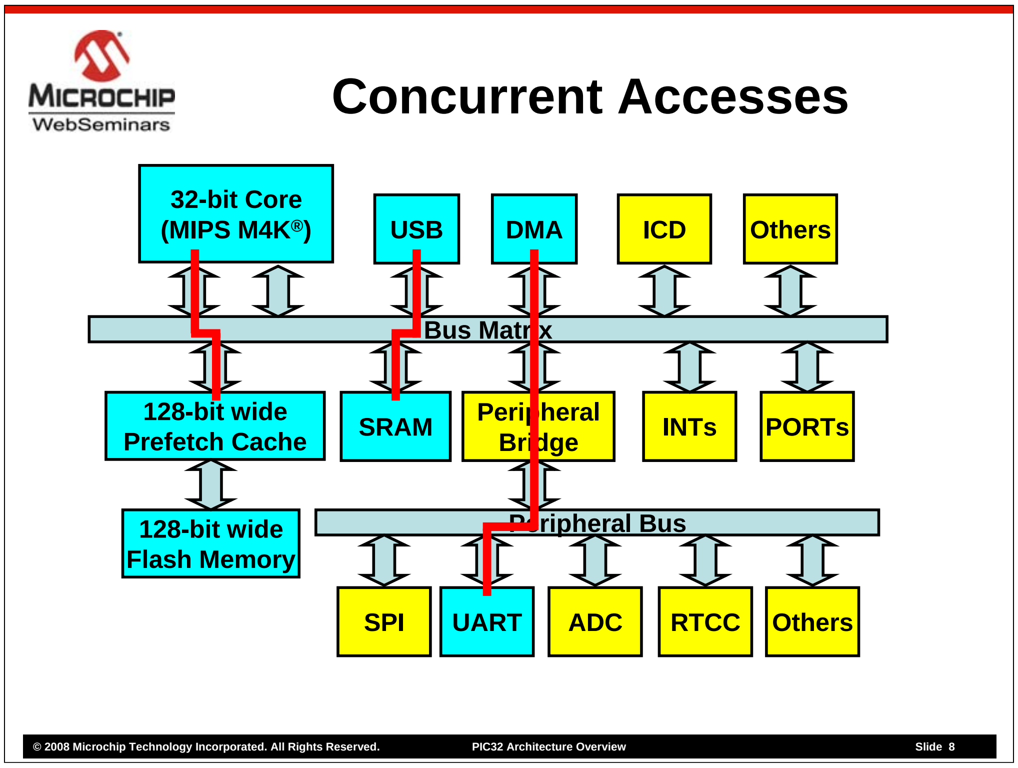


PBCLK Peripherals

PIC32 Architecture Overview Slide 7

The PBCLK Peripherals are one more class of peripherals. These peripherals run from PBCLK. The SPI, UART, ADC, RTCC, I2C, etc., are examples of the PBCLK peripherals. The exact value of the PBCLK is determined by the setting of the Peripheral bridge module. Available options are to run PBCLK at 1:1, 1:2, 1:4 and 1:8 of SYSCLK speed. Normally, PBCLK peripherals are slow in speed and do not require very high data throughput.

When the PBCLK is running at 1:1 with SYSCLK, the CPU and other Bus Masters will be able to access PBCLK peripherals in one cycle. As PBCLK divider gets larger, a read of the PBCLK peripheral will take as many clocks as the divider value. For example, if the PBCLK is running at 1:8 of SYSCLK, it will take 8 SYSCLK to read a PBCLK peripheral. All writes to PBCLK peripherals, on the other hand, are "posted". This means that when the CPU writes to an SFR in a PBCLK peripheral, the Bus Matrix takes over the write operation and allows the CPU to continue with the next operation. As a result, if PBCLK is running at 1:8 of SYSCLK, the CPU will complete its write of PBCLK peripheral in one cycle. However, the actual write will not take effect until after 8 SYSCLKs. In the meantime, if the CPU performs a read of a PBCLK peripheral, the CPU will stall for the exact PBCLK divider value and continue only after corresponding SYSCLK period has expired.



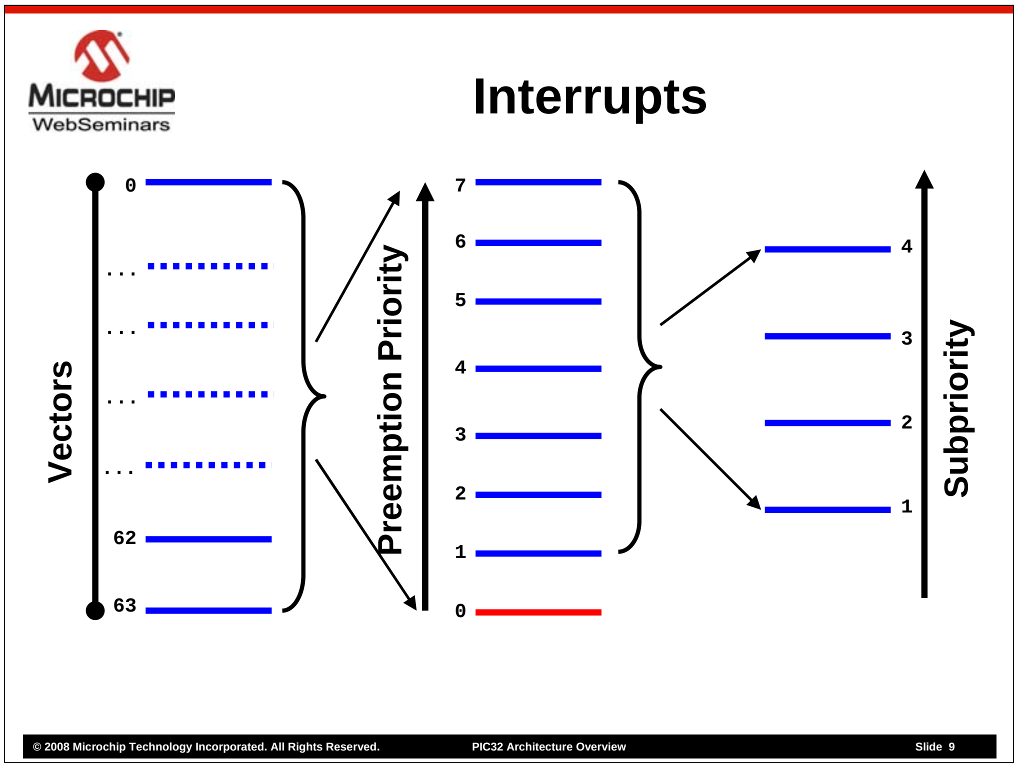
Concurrent Accesses

© 2008 Microchip Technology Incorporated. All Rights Reserved. PIC32 Architecture Overview Slide 8

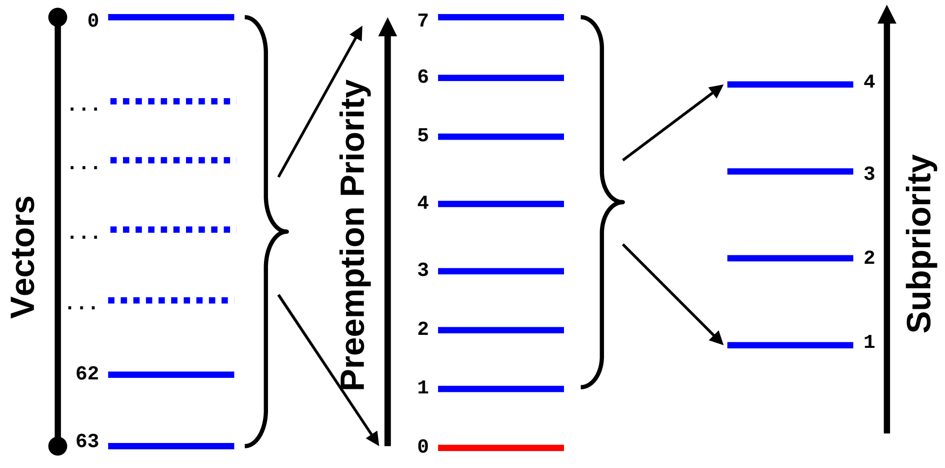
Now that you know about Bus Masters and targets, let me explain an important feature of the Bus Matrix.

As I mentioned earlier, the Bus Matrix is essentially a high-speed switch. Once a Bus Master initiates a transaction, the Bus Matrix establishes a point-to-point path from the Bus Master to the target module. While this first transaction is in progress, another Bus Master may initiate a second transaction to yet another target. Depending on the target, the Bus Matrix may establish a parallel path. This slide shows an example of three concurrent data paths. While the CPU is fetching instructions from Flash via the Prefetch Cache module, the USB may read or write SRAM and, at the same time, the DMA may read data from the UART module. In this example, all three paths are separate and there will not be any conflict or delay.

If the CPU were to access SRAM while the USB is in middle of accessing SRAM, there will be conflict and the Bus Matrix will arbitrate and allow one to complete before the other can continue. The exact priority is determined by the programming of the Bus Matrix registers. The software may give highest priority to the CPU, DMA or USB. In addition, the software may select one of three different arbitration schemes – fixed priority, fixed priority with CPU at lowest priority, and rotating priority scheme. Depending on the system requirements, an application would select an appropriate arbitration scheme to achieve the required data throughput and timings.



Interrupts

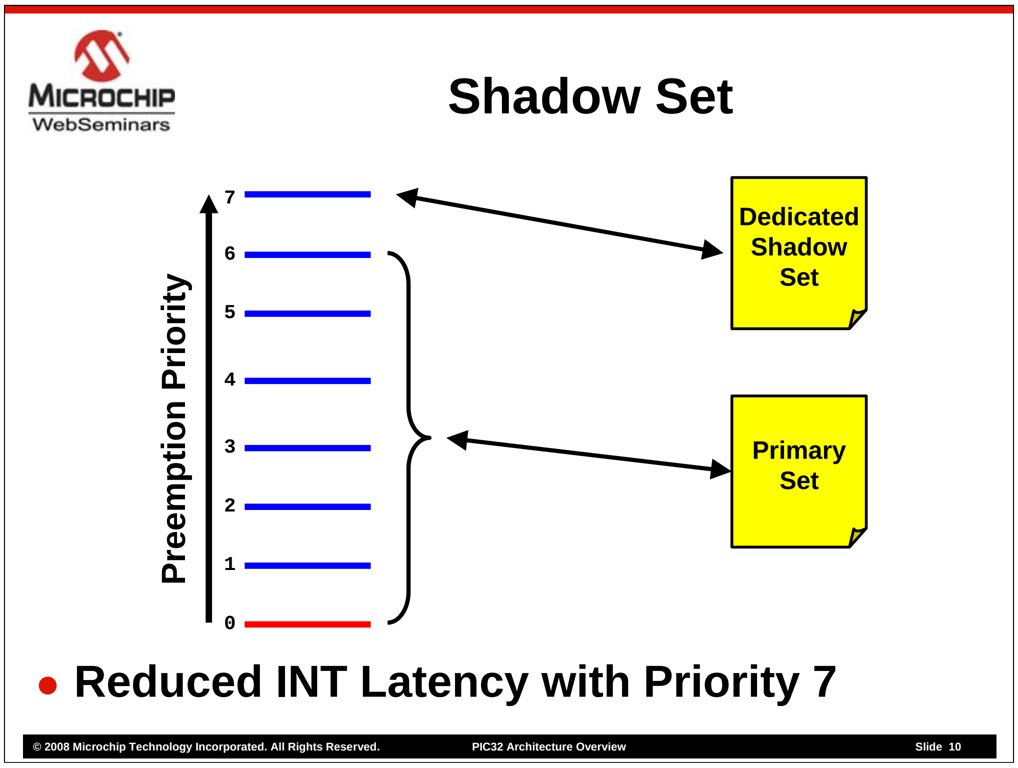


© 2008 Microchip Technology Incorporated. All Rights Reserved. PIC32 Architecture Overview Slide 9

The PIC32 offers a flexible interrupt controller. It can be programmed to operate in Single Vector mode or Multi Vector mode. In Single Vector mode, all interrupts use a common vector. While in Multi Vector mode, there are a total of 64 vectors. Each vector can have up to 8 different preemption priority levels. A value of 7 indicates the highest priority, while a value of 1 indicates the lowest priority and a value of 0 indicates that the vector is disabled. In summary, the higher the value, the higher the priority and it can preempt lower priority interrupts. You may assign one priority level to more than one interrupt vector.

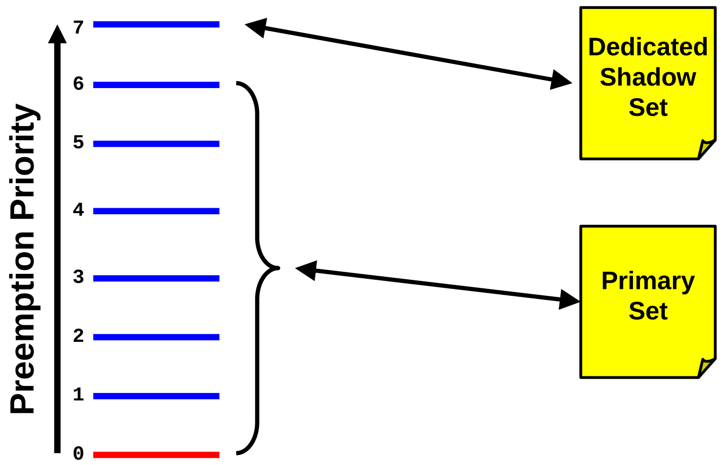
In addition to preemption priority, each vector can also have up to 4 levels of subpriorities. A subpriority defines the order in which interrupts will be taken if there is more than one interrupt of the same priority pending.

This module also offers programmable base addresses for vectors. It means that you may place your interrupt handlers at any address – in Flash or RAM – by simply changing the base address of the vector table.



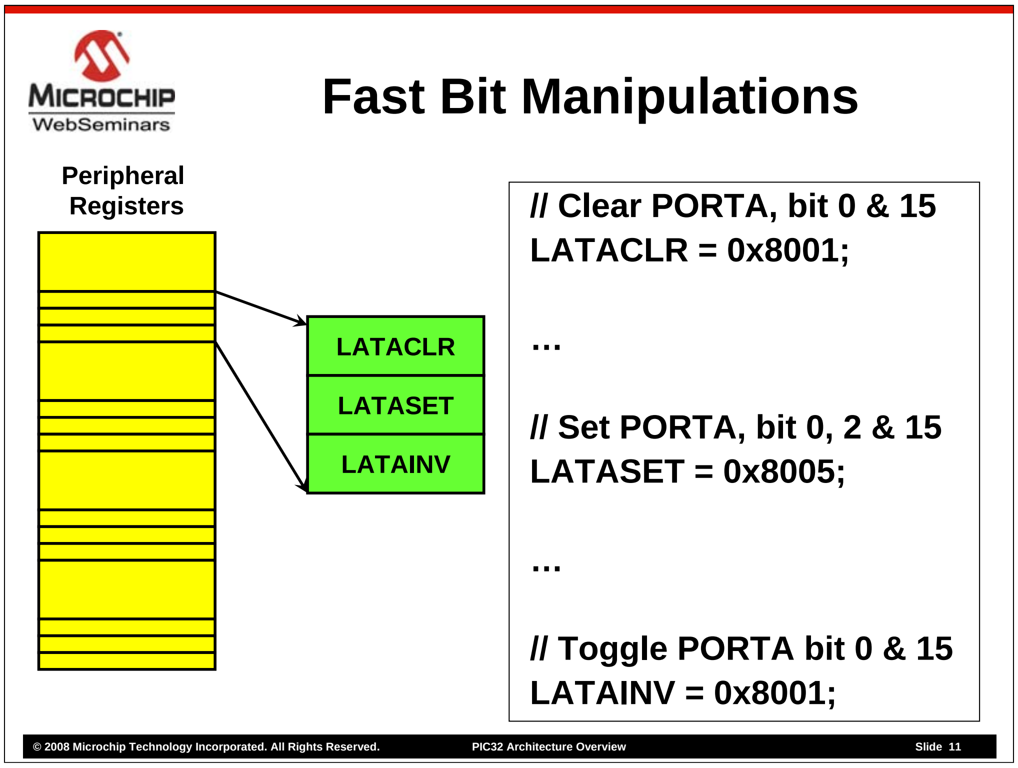
MICROCHIP WebSeminars

Shadow Set

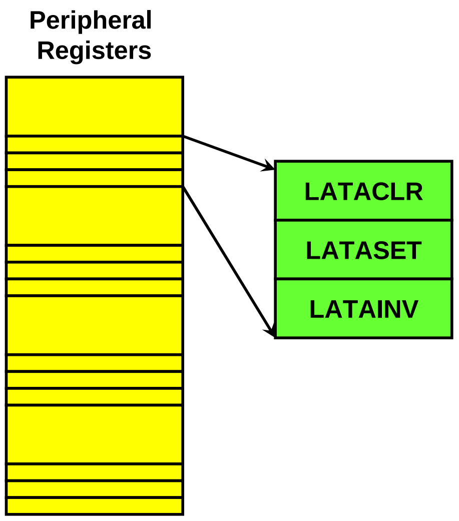


- Reduced INT Latency with Priority 7

In the previous slide, we learned that priority level 7 vectors get the highest priority. In addition to that, the priority 7 vectors also get a dedicated shadow register set. In normal operation when the CPU is executing at priority 6 or lower, the CPU operates on a primary register set. But when a priority 7 interrupt occurs, the interrupt controller automatically switches to the shadow set and jumps to the appropriate vector. With the dedicated shadow set, the priority 7 interrupt offers faster interrupt response as compared to other priority interrupts. The reason for this is that when the priority 7 interrupt occurs, the application does not have to save the entire register set context. Instead, it only needs to save a few critical registers and start executing the user interrupt handler. Similarly, when the priority 7 interrupt handler finishes its task, the application does not have to restore the full context either. It only needs to perform few steps and immediately return to the previous execution state.



Fast Bit Manipulations



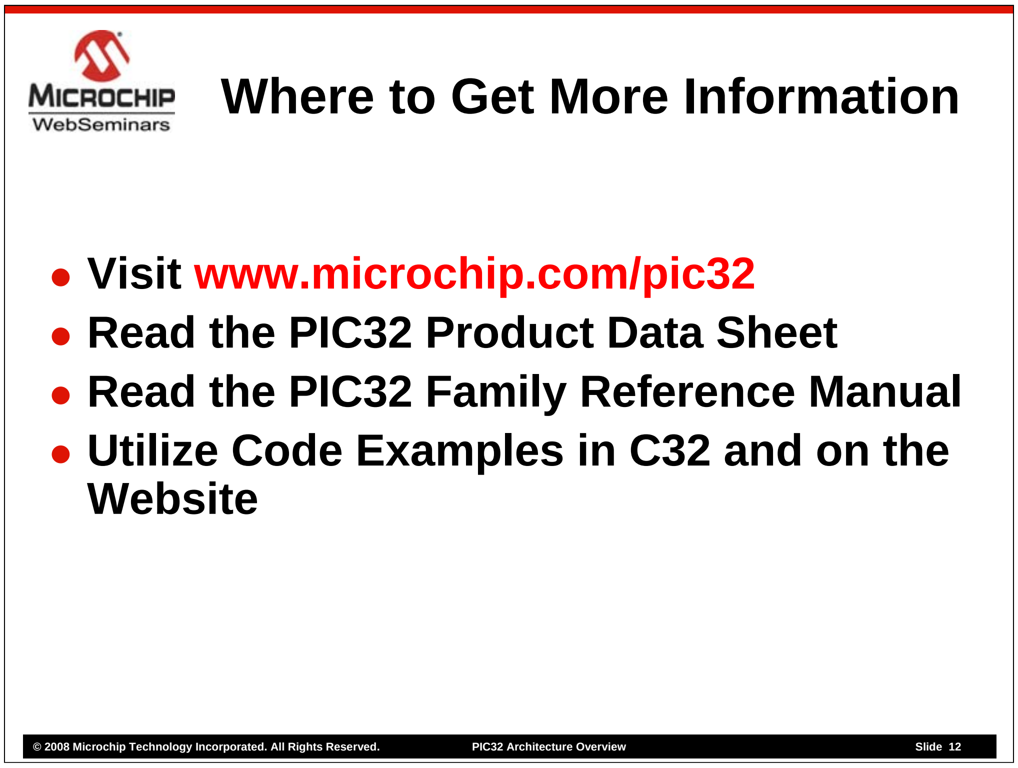
```
// Clear PORTA, bit 0 & 15
LATACLR = 0x8001;

…

// Set PORTA, bit 0, 2 & 15
LATASET = 0x8005;

…

// Toggle PORTA bit 0 & 15
LATAINV = 0x8001;
```

PIC32 Architecture Overview Slide 11

In an embedded system, the ability to quickly manipulate I/O ports and bits is highly desired. Most 32-bit microcontrollers are generally not very good at this requirement. The PIC32 architecture resolves this limitation by providing a set of registers called SET, CLEAR, and INVERT. Essentially, for a majority of Special Function Registers (or SFRs), there are three additional registers. For example, the LATA SFR is followed by LATACLR, LATASET, and LATAINV.

To clear a group of bits in the LATA register, you would write the corresponding mask values into the LATACLR register. For example, a write of 0x8001 to the LATACLR register would clear bits 0 and 15. Similarly, a write to the SET register would set the corresponding bits and a write to INV register would toggle the bits. When you write to any of the SET, CLR, or INV registers, the underlying hardware performs the read-modify-write operation in a single clock. This hardware assistance not only accelerates the bit manipulation, but it also provides atomicity. This means that the SET, CLR, and INV operations cannot be interrupted. This atomic bit manipulation capability simplifies the programming logic – now you don’t have to guard your I/O bit manipulation logic with interrupt disable and enable sequences, or worry about read-modify-write problems of I/O ports.

In the beginning, we noted that the I/O PORT peripherals are connected to the SYSCLK bus. This means that with the help of the INV registers, you can toggle any general purpose I/O pin at the SYSCLK speed!



Where to Get More Information

- Visit www.microchip.com/pic32
- Read the PIC32 Product Data Sheet
- Read the PIC32 Family Reference Manual
- Utilize Code Examples in C32 and on the Website

PIC32 Architecture Overview Slide 12

With that, we are now at the end of the presentation. You now have a high level overview of PIC32 architecture and some of its key features. To learn more about the PIC32 and start evaluation or application development, you should visit www.microchip.com/pic32. This site contains the PIC32 Data Sheet, Family Reference Manual and various Application Notes. It also includes links to Microchip and third party hardware and software tools. This site also provides C and assembly language code examples for the PIC32. These same code examples are also distributed in the C32 compiler for PIC32.

If you have any question, visit support.microchip.com.

Thanks for your time.