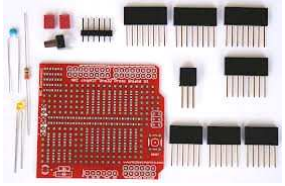


Introducción a la programación PIC32 con ARDUINO

Introducción a la programación de MCUs "PIC32" bajo entorno "ARDUINO"

Por: Andrés Raúl Bruno Saravia - Microchip Certified Trainer
RTC_Argentina – Electrónica Elemon S.A.

Entrega Nº 1

Segunda Parte: Breve Introducción al "PIC32"

Introducción a la Arquitectura PIC32

En el año 2007 Microchip introduce a su portfolio el microcontrolador mas potente que la compañia haya diseñado, el **PIC32.**

Su primera familia fue el **PIC32MX3**, cuyos dispositivos eran en montaje superficial, y en un pin-out de 68 y 100 terminales.

El **PIC32 es el primer microcontrolador de la historia de Microchip no diseñado por la compañía sino que fue encargado a otra empresa denominada MIPS Technologies**, la cual es muy conocida por ser una desarrolladora de nucleos para terceros de 32 y 64 bits.

Esta fue una decisión estratégica para poder ingresar al mundo de los 32 bits con **"los tapones de punta".**

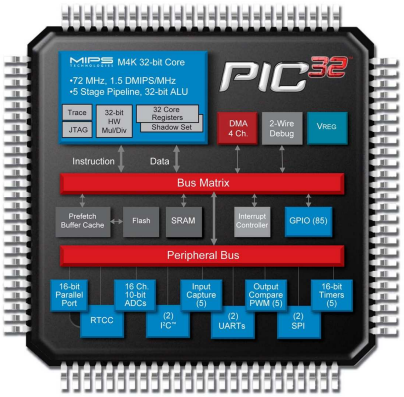
Dicho nucleo se inserto en el interior de un **PIC24H128**, al cual Microchip lo adaptó para poder colocar su nuevo core.

De esta forma el usuario goza de un nucleo de 32 bits de altísimas prestaciones, y del vademecum de perifericos que tiene los PIC24H.

Los PIC32 actualmente se encuentran en pleno apogeo, creciendo a una velocidad logarítmica, actualmente la familia esta integrada por los **PIC32MX1, PIC32MX2, PIC32MX3,PIC32MX4,PIC32MX5, PIC32MX6 y PIC32MX7**. El conjunto de todas estas familias determina que PIC32 actualmente este formada por mas de 70 modelos distíntos!!!.

Nuevamente Microchip demuestra su capacidad a la hora del desarrollo de microcontroladores.

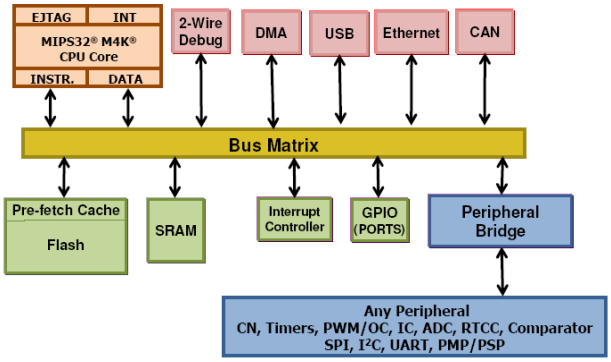
Los PIC32 y sus características.



Para comenzar con nuestra descripción diremos que **un PIC32 tiene internamente una arquitectura mixta,** ya que **dentro de su core, su arquitectura es HARWARE**, es decir que existe un **bus de instrucciones y otro de datos.** Pero si lo vemos desde la memoria de programa y la memoria de datos, **el PIC tiene una arquitectura VON NEWMAN. Esta dualidad termina con el viejo problema de Microchip de los accesos a la memoria de programa para la lectura de tablas de datos,** por existir dos buses diferenciados, y que las arquitecturas anteriores habian salvado el escollo, por medio de la implementación de una herramienta complementaria. En el caso de **PIC18**, se habia hechado mano al mecanismo de **Punteros de Tabla** y un **registro TABLAT**, y luego en los micros de 16 bits, **PIC24 y dsPIC**, se usaba el recurso de **VSP**, mediante el cual se podia visualizar parte de la memoria de programa desde la RAM.

Estos mecanismos aunque eficientes, perdian ciclos de máquina y fue por ello que MIPS decidió que ambas memorias podian enlazarse con un mismo BUS de datos, ya que este era de 32 bits y tenia el espacio suficiente para poder transferir las instrucciones, ya que ahora estas tenían ese ancho.

Pero Microchip, para acelerar el procesamiento de las mismas solicito a MIPS que en el interior del core, las intrucciones y los datos se separasen. De esta manera las instrucciones podian ser procesadas por un nuevo pipeline de 5 niveles. Esto mejoró la perfomance del procesador, permitiendo ejecutar las instrucciones en un pulso de clock, con lo cual, estando el pipeline cargado, **el PIC32 a 80MHZ puede procesar 80 MIPS!!!.**



Este es un diagrama simplificado de la arquitectura de la familia PIC32, mostrando un superset de todas las características posibles.

El corazón del PIC32 es el nucleo M4K. Una interfaz JTAG mejorada (EJTAG) esta implementada con acceso directo al nucleo para soportar las funciones de debugging.

El núcleo tiene 2 buses: uno usado para buscar las instrucciones, y otro usado para acceder a los datos.

A fin de garantizar el rendimiento mas alto posible de transferencia de datos e instrucciones, se llevo a cabo un "Bus Matriz" en lugar de la típica arquitectura "dato y direcciones", bus común en los CPU de menor porte. El bus matriz puede visualizarse como una autopista con multiples carriles en comparación con el bus de vía única de un microcontrolador típico.

Debido a que hay disponibles muchos carriles para el trafico, los datos y las instrucciones pueden ser transmitidos desde muchas fuentes y hacia muchos destinos en un mismo ciclo de máquina.

Naturalmente hay limitaciones en cuanto a lo que es posible, pero el hardware se asegura de que todo el mundo tenga su tiempo justo en el BUS. De hecho el programador puede seleccionar entre diferentes variantes de prioridad de acceso.

El bus matriz esta funcionando a la misma frecuencia que el core de la CPU.

La memoria FLASH almacena constantes e instrucciones del programa y la RAM almacena los datos y las variables, estan también conectadas a el bus matriz.

En la memoria también se puede observar un elemento nuevo: **un módulo de prebúsqueda.**

La FLASH implementada en los PIC32 no puede entregar la información a la CPU más rápido de lo que la CPU puede consumir dicha información. Con el fin de resolver este desequilibrio el módulo de prebúsqueda ofrece 2 características:

1) **Un mecanismo de CACHE que puede almacenar 64 instrucciones de 32 bits.** La propia CACHE es una memoria RAM de tecnología de alta velocidad que puede entregar datos e instrucciones al nucleo a la misma frecuencia que el micro está ejecutando el código.

2) **Mecanismo de captación previo que usa un algorítmo de "predicción de salto",** mediante el cual se carga la CACHE con instrucciones antes de su tiempo de ejecución. Puede imaginarse dicho mecanismo como una "bola de cristal" que intenta predecir que instrucción se necesita.

Ambas características ayudan a que el código se ejecute con la mayor velocidad posible.

Además vale la pena señalar que cada ciclo de búsqueda lee datos de **128bits,** lo que equivale a **4 instrucciones de 32 bits** (o si se quiere, 8 instrucciones de 16 bits denominadas MIPS16e).

El BUS Matriz puede incluir la conexión a un controlador **ETHERNET,CAN, USB, DMA, Puertos I/O.**

El resto de los periféricos están conectados a un puente de periféricos. Dicho puente puede trabajar a la velocidad del núcleo o a otra alternativa a través de un divisor de frecuencia.

En las Profundidades del Core PIC32 (M4K)

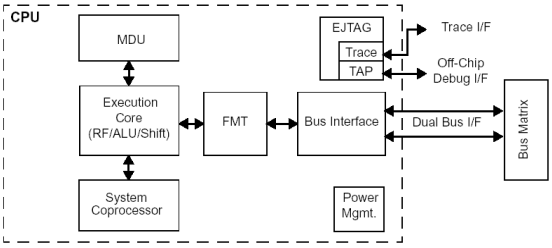
El nucleo es capaz de ejecutar una instrucción por cada pulso de clock.

Esta compuesto por 32 registros de 32 bits disponibles para el almacenamiento de datos locales, y otro conjunto completo de 32 registros de 32 bits que pueden ser utilizados por la rutina de interrupciones, reduciendo así la sobrecarga de entrada y salida de la interrupción.

La ALU es de 32 bits de ancho y soporta instrucciones del tipo MAC (Multiplica y Acumula) en un solo ciclo.

Cuando el espacio del código generado es muy crítico, es posible compilar la totalidad o parte del código siguiendo las instrucciones MIPS16e donde las instrucciones son de 16 bits en lugar de 32. Esto permite ahorrar espacio en memoria de programa, pero aumenta el tiempo de ejecución de forma drástica.

Como en todos los microcontroladores PIC, los datos se almacenan en forma Little Endian, es decir, el byte menos significativo se encuentra en la dirección más baja de memoria.



Núcleo M4K

El corazón del nucleo es la **unidad de ejecución (EC),** la cual se ocupa de la aritmética standar, las operaciones de rotación y las instrucciones de desplazamiento.

El propio nucleo utiliza un pipeline de 5 etapas necesario para cargar, ejecutar y guardar el resultado de cada instrucción, lo que requiere de un total de 5 pulsos de clock, sin embargo cuando el pipeline esta totalmente cargado, la CPU es capaz de ejecutar cada instrucción en un solo pulso de clock.

Las instrucciones de salto implican un retardo adicional, conocido como "retardo de salto", debido a la estructura secuencial del pipeline. Sin embargo la mayoría de los compiladores y ensambladores llenan ese retardo con una instrucción util (siempre que sea posible), eliminando así cualquier pérdida de rendimiento en la mayoría de los casos.

Se suma al nucleo la **unidad de multiplicación y división (MDU),** la cual realiza calculos de 16x16 y 16x32 bits en un solo ciclo de maquina. Las multiplicaciones de 32x32bits usan 2 ciclos de maquina.Las instrucciones de división necesitan de 35 ciclos pero de forma inteligente usa tan solo 13 ciclos si se detecta que la división es menor a 32 bits.

Opcode	Operand Size (mul *rt*) (div *rs*)	Latency	Repeat Rate
MULT/MULTU, MADD/MADDU, MSUB/MSUBU	16 bits	1	1
	32 bits	2	2
MUL	16 bits	2	1
	32 bits	3	2
DIV/DIVU	8 bits	12	11
	16 bits	19	18
	24 bits	26	25
	32 bits	33	32

Tablas de Latencias y velocidad de repeticiones en la MDU del MIPS M4K Core.

El compilador debe reordenar las instrucciones necesarias para garantizar que el resultado esté disponible cuando el pipeline del núcleo lo requiera. Dado que esta operación se ejecuta en paralelo al núcleo y con el reordenamiento inteligente de las instrucciones por parte del compilador, no debería haber perdidas de rendimiento.

El nucleo no es muy inteligente cuando se inicia, no tiene idea por ejemplo de como se conecta la memoria disponible y no tiene ningún registro de estado.

Con el fin de remediar esta situación existe lo que se denomina el coprocesador cero (copro 0). El nombre es un poco engañoso, **el nucleo PIC32 puede soportar coprocesadores, pero no se implementa en el PIC32.**

El coprocesador 0 es para implementar el **registro de estado del nucleo**, **configurar el contador de memoria de cache**, **el manejo de excepciones e interrupciones, configurar la unidad de memoria y programar otras características.**

Antes de que cualquier código se ejecute, los registros del coprocesador 0 deben ser inicializados apropiadamente. Esto es realizado por un código conocido como **"codigo de arranque"** y se vincula automáticamente al código de la aplicación

TABLE 3-2: COPROCESSOR 0 REGISTERS

Register Number	Register Name	Function
0-6	Reserved	Reserved
7	HWREna	Enables access via the RDHWR instruction to selected hardware registers
8	BadVAddr(1)	Reports the address for the most recent address-related exception
9	Count(1)	Processor cycle count
10	Reserved	Reserved
11	Compare(1)	Timer interrupt control
12	Status(1)	Processor status and control
12	IntCtl(1)	Interrupt system status and control
12	SRSCtl(1)	Shadow register set status and control
12	SRSMap(1)	Provides mapping from vectored interrupt to a shadow set
13	Cause(1)	Cause of last general exception
14	EPC(1)	Program counter at last exception
15	PRId	Processor identification and revision
15	EBASE	Exception vector base register
16	Config	Configuration register
16	Config1	Configuration register 1
16	Config2	Configuration register 2
16	Config3	Configuration register 3

TABLE 3-2: COPROCESSOR 0 REGISTERS (CONTINUED)

Register Number	Register Name	Function
17-22	Reserved	Reserved
23	Debug(2)	Debug control and exception status
24	DEPC(2)	Program counter at last debug exception
25-29	Reserved	Reserved
30	ErrorEPC(1)	Program counter at last error
31	DESAVE(2)	Debug handler scratchpad register

Note 1: Registers used in exception processing.
2: Registers used during debug.

TABLE 3-3: PIC32MX3XX/4XX FAMILY CORE EXCEPTION TYPES

Exception	Description
Reset	Assertion $\overline{\text{MCLR}}$ or a Power-on Reset (POR)
DSS	EJTAG Debug Single Step
DINT	EJTAG Debug Interrupt. Caused by the assertion of the external *EJ_DINT* input, or by setting the EjtagBrk bit in the ECR register
NMI	Assertion of NMI signal
Interrupt	Assertion of unmasked hardware or software interrupt signal
DIB	EJTAG debug hardware instruction break matched
AdEL	Fetch address alignment error Fetch reference to protected address
IBE	Instruction fetch bus error
DBp	EJTAG Breakpoint (execution of `SDBBP` instruction)
Sys	Execution of `SYSCALL` instruction
Bp	Execution of `BREAK` instruction
RI	Execution of a Reserved Instruction
CpU	Execution of a coprocessor instruction for a coprocessor that is not enabled
CEU	Execution of a `CorExtend` instruction when `CorExtend` is not enabled
Ov	Execution of an arithmetic instruction that overflowed
Tr	Execution of a trap (when trap condition is true)
DDBL/DDBS	EJTAG Data Address Break (address only) or EJTAG Data Value Break on Store (address + value)
AdEL	Load address alignment error Load reference to protected address
AdES	Store address alignment error Store to protected address
DBE	Load or store bus error
DDBL	EJTAG data hardware breakpoint matched in load data compare

ELECTRONICA ELEMON S.A.
Capdevila 2707, Villa Urquiza
C. A. de Buenos Aires, C1431FKA
Argentina
capacitacion@elemon.com.ar
soporte@elemon.com.ar
ventas@elemon.com.ar

Encontranos en FACEBOOK:
https://www.facebook.com/pages/Electronica-Elemon/119727961396798