Cika Electrónica

Tutorial: CTU-007
Título: **Introducción a Holtek desde PIC**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	02/06/06	
1	16/02/07	directiva OFFSET

El presente tutorial se orienta a introducir a Holtek a los ingenieros y developers familiarizados con Microchip PIC, de una forma práctica y concisa. Para un análisis de la arquitectura y descripción del uso de los periféricos más comunes, se sugiere el tutorial CTU-005.

Índice de contenido

Diferencias de Arquitectura

La arquitectura de Holtek es muy similar a la de PIC16, sin embargo, en aquellos lugares en los que los usuarios de otros micros se sienten incómodos con PIC, Holtek resulta agradable. La siguiente es una descripción enumerando aquellos factores arquitectónicos que los diferencian:

- El registro W (por Working register) de PIC cumple la misma función que el tradicional acumulador A. En Holtek, se lo llama A; y se ve como un SFR más (ACC), permitiendo una gran cantidad de operaciones adicionales.
- Entre las instrucciones adicionales de que disponemos, figuran la suma y resta con acarreo, lo cual simplifica las operaciones matemáticas, y la operación de ajuste decimal, lo cual simplifica la operación con dígitos BCD.
- La memoria de instrucciones es de 14-bits, 15-bits, o 16-bits de ancho, y hacen falta algunos de ellos para indicar que la instrucción es de control de flujo de programa (call, jump, etc) , empleándose el resto para especificar la dirección. El ancho de palabra de programa es coincidente con la capacidad de memoria de programa, por lo que la totalidad de la misma es direccionable linealmente.

- Para permitir mayor versatilidad en el control del flujo del programa, al igual que PIC, el program counter está accesible en la memoria de datos, es un SFR más. Dado que la memoria de datos es de 8-bits, los bits (14, 15 ó 16) del PC están partidos en PCL (8-bits) y PCH (6, 7 ú 8 bits). Sin embargo, en Holtek el PCH no es accesible directamente, por lo que las instrucciones que modifiquen el flujo mediante cálculo y operación sobre PCL (por oposición al uso de CALL o JMP) están limitadas a un segmento de 256 words de programa.Las operaciones de CALL y JMP operan sobre la totalidad de la memoria de programa sin restricciones ni bank switching.
- La arquitectura permite interrupciones vectorizadas, aunque en realidad se trata de offsets fijos. Una interrupción ocasiona un salto a una posición fija en memoria de instrucciones, dependiente de la causa de la interrupción. El contenido del PC se guarda en el hardware stack para permitir su recuperación al ejecutar una instrucción de retorno, que restablece el estado de las interrupciones. Dado que no existen instrucciones de manejo de stack, tampoco se salva el contexto de la CPU, el programa debe salvar el registro A y el STATUS en alguna posición de memoria. Como sólo las operaciones aritméticas y lógicas afectan el STATUS, es posible prescindir de salvar este registro si el interrupt handler realiza funciones clásicas.
- Al leer un port de salida, Holtek devuelve el estado del latch, como la mayoría de los microcontroladores (excepto PIC).

Comparación

Microchip serie midrange (PIC16F)

Presenta direccionamiento lineal hasta 2K flash y 84 bytes RAM. El resto de la RAM y flash se accede mediante bank switching.

Se trata de un RISC con arquitectura Harvard modificada, stack en hardware de 8 niveles. La CPU acepta un clock de hasta 20 MHz, internamente dividido por cuatro, lo que permite una ejecución a 5 MIPS pico, que no disminuyen demasiado en operación sostenida, dado que la gran mayoría de las operaciones se ejecutan en un ciclo de clock.

Como periféricos, encontramos una vasta variedad de dispositivos con timer de 8-bits, timer de 16-bits, módulos de captura y comparación (CCPs) con capacidad de generación de PWM, conversores AD de 8 a 10-bits de 4 ú 8 canales, comparadores, referencias de tensión, etc. El watchdog timer posee un oscilador independiente.

Holtek series “Cost Effective”, “I/O” y “A/D”

Presenta direccionamiento lineal hasta 8K OTP/MTP y 224 bytes RAM; el modelo que incorpora una cantidad mayor de RAM emplea bank switching.

Se trata de un RISC con arquitectura Harvard modificada, stack en hardware de 2 a 16 niveles, según el modelo. Su operación es bastante similar a PICmicro de Microchip, pero sin el uso de bank switching en flash ni I/O y practicamente nulo en RAM. La CPU acepta un clock de hasta 8 MHz, internamente dividido por cuatro, lo que permite una ejecución a 2 MIPS pico, que no disminuyen demasiado en operación sostenida, dado que la gran mayoría de las operaciones se ejecutan en un ciclo de clock.

Como periféricos, encontramos un timer de 8-bits en casi todos los modelos; y ninguno, uno o dos timers de 16-bits en las series “I/O” y “A/D”. Los timers tienen capacidad de controlar un buzzer mediante dos salidas complementarias con 50% de ciclo de trabajo. Los modelos de la serie “A/D” incorporan un conversor AD de 9 ó 10-bits de 4 ú 8 canales, y los modelos más avanzados agregan interfaz I^2C y generador de PWM. El watchdog timer posee un oscilador independiente.

Paralelo de hardware

La tabla en la hoja siguiente es una comparativa de características similares de hardware

Microchip	*Holtek sin AD*	*Holtek con AD*
12C5xx: 0.5/1K OTP, 25/41 RAM, timer(8), 8 pines, 5 I/O + I, xtal o RC	**48R05/6**: 0.5/1K OTP, 32/64 RAM, timer(8), 16/18 pines, 11/13 I/O, xtal o RC	**46R46:** 1K OTP, 64 RAM, timer(8), 9-bit AD, 18 pines, 13 I/O, xtal o RC
12F629/675:1K flash 64 RAM, timer(8,16), 8 pines, 5 I/O + I, 128 EEPROM, (10-bits AD), osc **16F630/676**:1K flash 64 RAM, timer(8,16), 14 pines, 12 I/O + I, 128 EEPROM, (10-bits AD),osc **16F84A**: 1K flash, 68 RAM, timer (8), 13 I/O, 64 EEPROM, xtal o RC	**48R05/6**: 0.5/1K OTP, 32/64 RAM, timer(8), 16/18 pines, 11/13 I/O, xtal o RC **48E06**: 1K MTP, 64 RAM, timer (8), 18/20 pines, 11/13 I/O, 128 EEPROM, xtal o RC	**46R46(E):** 1K OTP, 64 RAM, timer(8), (128EEPROM), 9-bit AD, 18 pines, 13 I/O, xtal o RC
16F627A: 1K flash, 224 RAM, timer(8,8,16), 18 pines, 15 I/O + I, 128 EEPROM, USART, osc	**48R10**: 1K OTP, 64 RAM, timer(8), 24 pines, 21 I/O, osc **48E10**: 1K OTP, 64 RAM, timer(8), 128 EEPROM, 24 pines, 19 I/O, xtal or RC	**46R46(E):** 1K OTP, 64 RAM, timer(8), (128EEPROM), 9-bit AD, 18 pines, 13 I/O, xtal o RC
16F628A: 2K flash, 224 RAM, timer(8,8,16), 18 pines, 15 I/O + I, 128 EEPROM, USART, osc	**48R30:** 2K OTP, 96 RAM, timer(8), 24/28 pines, 21/25 I/O, osc **48E30:** 2K OTP, 96 RAM, timer(8), 128 EEPROM, 24/28 pines, 19/23 I/O, xtal or RC	**46R47(E)**: 2K OTP, 64 RAM, timer(8), (128EEPROM),9-bit AD, 18 pines, 13 I/O, xtal o RC **46R22**: 2K OTP, 64 RAM, timer(8,16), 9-bit AD, PWM8(1), I^2C, 24 pines, 19 I/O, xtal o RC
16F873A/874A: 4K flash, 192 RAM, timer(8,8,16), 28/40/44 pines, 22/33 I/O, 128 EEPROM, 10-bit AD, USART, I^2C, xtal o RC	**48R50**: 4K OTP, 160 RAM, timers (8,16), 28/48 pines, 15/35 I/O, osc **48E50**: 4K OTP, 160 RAM, timers (8,16), 256 EEPROM, 28/48 pines, 13/33 I/O, osc	**46R23**: 4K OTP, 192 RAM, timer (8,16), 10-bit AD, PWM8(2), I^2C, 24/28 pines, 19/23 I/O, xtal o RC
16F876A/877A: 8K flash, 368 RAM, timer(8,8,16), 28/40/44 pines, 22/33 I/O, 256 EEPROM, 10-bits AD, USART, I^2C, xtal o RC	**48R70**: 8K OTP, 224 RAM, timers (16,16), 48/64 pines, 40/56 I/O, osc **48E70**: 8K OTP, 224 RAM, timers (16,16), 256 EEPROM, 48/64 pines, 40/56 I/O, osc	**46R24**: 8K OTP, 384 RAM, timers (16,16), 10-bit AD, PWM8(4), I^2C, 28/48 pines, 20/40 I/O, xtal o RC

Debido a que esta tabla no se actualiza frecuentemente, es posible que exista mayor variedad, no deje de consultar a su vendedor.

Paralelo de software

A continuación haremos un paralelo de software.

Los listados presentan primero la resolución de una tarea típica en programas simples de control (a lo que estos micros están destinados) para PIC16, y luego el equivalente en Holtek. De este modo, el usuario experimentado en un micro conocido como Microchip PIC puede tener una idea rápida de las capacidades de Holtek. Respecto al assembler de Holtek, el mismo determina mediante las declaraciones qué nombres corresponden a variables y cuáles se trata de constantes, generando automáticamente direccionamiento directo o inmediato según corresponda. Ante una ambigüedad, existe la directiva OFFSET

Por lo general, los mnemónicos y la sintaxis de Holtek resultan más simples de comprender para aquéllos familiarizados con otros micros tradicionales.

Interrupt desde un timer

Evaluamos el tiempo necesario en atender una interrupción de un timer, salvar contexto, y retomar ejecución normal del programa. Dado que en PIC las interrupciones no son vectorizadas, incluimos dicho overhead. En PIC, la variable

para guardar el registro W debe ser definida en un área que exista en todos los bancos, caso contrario se modifica STATUS. El procedimiento es el recomendado por el fabricante para un midrange. En Holtek, no existe este inconveniente, y dado que las instrucciones de *mov*e no afectan los flags y no existe bank switching, la operación es más simple de entender

```
        MOVWF W_TEMP                ;salva W
        SWAPF STATUS,W              ;swap status (salva en W)
        BCF STATUS,RP0              ;bank 0
        MOVWF STATUS_TEMP           ;salva status

        btfss   PIR1,T1IF           ; Chequea si int es Timer1
        goto    intxit              ; exit
        bcf     PIR1,T1IF           ; ack

        ...

intxit  SWAPF STATUS_TEMP,W         ;recupera status en W
        MOVWF STATUS                ;restablece (junto con el banco)
        SWAPF W_TEMP,F              ;recupera W
        SWAPF W_TEMP,W              ;restablece W
        retfie
```

10 ciclos + overhead = 12 ciclos
identificación de interrupción y ACK = 3 ciclos
total: 15 ciclos

```
        mov somewhere,A             ; salva A
        mov A,STATUS
        mov somewhere2,A            ; salva STATUS
        ...
        mov A,somewhere2
        mov STATUS,A
        mov A,somewhere
        reti
```

8 ciclos + overhead = 10 ciclos

Bifurcación (branch) según flag

Este es el típico caso de modificación del flujo de programa debido a circunstancias especificadas en un flag. Dada la carencia de direccionamiento relativo, tanto Holtek como Microchip lo resuelven mediante una instrucción de salteo (skip) y un salto. Generalmente, la bifurcación debe pensarse al revés

```
        btfss var,bit
        goto l1
        ...
l1
```

2/3 ciclos, 2 words

```
        snz var.bit             ; saltea instrucción si el bit 'bit' de la variable 'var' es 1
        jmp l1                  ; caso contrario salta a 'l1'
        ...
l1:
```

2/3 ciclos, 2 words

Salteo (skip) según flag

Esta situación se produce generalmente cuando debemos alterar el estado de un pin según nos indica un flag, pero no podemos darnos el lujo de producir glitches (modificarlo primero asumiendo un valor y después corregirlo si el flag indica otra cosa). Consideramos solamente el tiempo empleado en la decisión.

```
        btfsc var,bit
        ; acción si 1
        btfss var,bit
        ; acción si 0
```

3 ciclos, 2 words

```
        sz var.bit
        ; acción si 1
        snz var.bit
        ; acción si 0
```

3 ciclos, 2 words

En un caso más simple, cuando sí nos podemos permitir el lujo de un glitch o simplemente modificamos un flag:

```
        ; acción si 0
        btfsc var,bit
        ; acción si 1
```

2 ciclos, 1 word

```
        ; acción si 0
        sz var.bit
        ; acción si 1
```

2 ciclos, 1 word

Modificar flag

Alteramos el estado (seteamos o reseteamos) de un bit en una variable

```
        bsf var,bit
```

1 ciclo, 1 word

```
        set var.bit
```

1 ciclo, 1 word

Típico loop

Consideramos solamente el overhead que introduce lo necesario para producir el loop, es decir, decremento del contador y salto al inicio del loop.

```
l1:
        ...
        decfsz var,f            ; decrementa 'var' y si ésta es 0, saltea la siguiente instrucción
        goto l1                 ; vuelve al loop
```

3 ciclos, 2 words

```
l1:
        ...
        sdz var                 ; decrementa 'var' y si ésta es 0, saltea la siguiente instrucción
        jmp l1                  ; vuelve al loop
```

3 ciclos, 2 words

Sumar valor a variable 8 bits

Sumamos un determinado valor constante a una variable cualquiera; el resultado debe quedar en la variable en cuestión.

```
        movlw value
        addwf var,f             ; resultado de la suma en var
```

2 ciclos, 2 words

```
        mov A,value
        addm A,var              ; resultado de la suma en var
```

2 ciclos, 2 words

Lo mismo, pero el resultado queda en el acumulador (A o W)

```
        movlw value
        addwf var,w             ; resultado de la suma en W
```

2 ciclos, 2 words

```
        mov A,value
        add A,var               ; resultado de la suma en A
```

2 ciclos, 2 words

Sumar una variable a otra, 8 bits

Sumamos dos variables entre sí; el resultado debe quedar en la última variable direccionada.

```
        movf var1,w
        addwf var2,f            ; resultado de la suma en var2
```

2 ciclos, 2 words

```
        mov A,var1
        addm A,var2             ; resultado de la suma en var2
```

2 ciclos, 2 words

Memory move, inmediato 8 bits

Colocamos un valor constante en una posición de memoria.

```
        movlw value
        movwf var
```
2 ciclos, 2 words

```
        mov A,value
        mov var,A
```
2 ciclos, 2 words

Memory move, directo 8 bits

Colocamos en una variable el valor contenido en otra.

```
        movf var1,w
        movwf var2
```
2 ciclos, 2 words

```
        mov A,var1
        mov var2,A
```
2 ciclos, 2 words

Cálculo en 16 bits: K = next_T - advance – COMP_K

Realizamos un pequeño cálculo no muy complejo, en el cual obtenemos el valor de una variable de proceso que depende de otras dos variables y una constante de calibración. Las variables son todas de 16-bits, y la constante de calibración es de 8 bits, para mostrar un cálculo combinado.

```
        movf advanceh,w
        subwf next_Th                ; next_T – advance (MSB)
        movwf Kh
        movf advancel,w
        subwf next_Tl,w              ; LSB
        movwf Kl
        btfss STATUS,C               ; borrow ?
        decf Kh,f                    ; sí, Kh = Kh – 1
        movf COMP_K,w
        subwf Kl,f                   ; - COMP_K (8 bits)
        btfss STATUS,C               ; borrow ?
        decf Kh,f                    ; sí, Kh = Kh – 1
done:
```
12 ciclos, 12 words

```
        mov A,next_Tl
        sub A,advancel               ; next_T – advance (LSB)
        mov Kl,A
        mov A,next_Th
        sbc A,advanceh               ; MSB
        mov Kh,A
        mov A,Kl
        sub A,COMP_K                 ; - COMP_K (8 bits)
        mov Kl,A
        snz C                        ; borrow ?
        dec Kh                       ; sí, Kh = Kh – 1
done:
```
11 ciclos, 11 words

Cálculo simple en 32 bits: sumar dos variables

Sumamos dos variables en 32-bits, el resultado lo guardamos en una de ellas. Para Microchip, la operatoria es bastante complicada, dado que no existe operación de suma con acarreo, y debemos mantener registros de acarreos parciales o ir incrementando parcialmente a medida que calculamos. Para Holtek, es igual de simple seguir hasta 48 ó 64-bits

```
        mov A,var1ll
        addm A,var2ll                ; suma LSB
        mov a,var1lh
        adcm A,var2lh                ; acarrea
        mov a,varhl
```

```
        adcm A,var2hl
        mov a,var1hh
        adcm A,var2hh                   ; MSB
```
8 ciclos, 8 words

Máquina de estados

Analizamos a continuación un dispatcher del tipo comunmente utilizado en máquinas de estados, en el cual el flujo de programa es redirigido según el valor de una variable, es decir, el estado de la máquina. En ambos casos, la función y la tabla de saltos deberán estar dentro de un mismo segmento de 256 words. En el caso de Microchip, deberá setearse PCLATH con el valor del segmento en que está la rutina, a menos que siemrpe se lo mantenga en el mismo valor válido.

```
        ORG 0x10
        movf    STATE,w         ; estados posibles: 0,1,2,3,4,etc
        clrf    PCLATH
        addwf   PCL,f           ; resultado de la suma en PCL
        goto    State1
        goto    State2
```
6 ciclos, 1 word por estado

```
        mov A,STATE             ; estados posibles: 0,1,2,3,4,etc
        addm A,PCL              ; resultado de la suma en PCL
        jmp State1
        jmp State2
```
5 ciclos, 1 word por estado

Table read, tabla en RAM

Tomamos un valor de una tabla según nos indica una variable que oficia de índice dentro de la misma. La tabla se encuentra en RAM, y en Microchip su extensión no puede superar el tamaño máximo del banco en donde reside, lo cual depende del modelo y en el mejor de los casos es de 84 bytes. En holtek el límite es de 224 bytes

```
        movlw TABLA             ; tabla
        addwf OFFSET,w          ; + offset
        movwf FSR               ; indirecto
        movf INDF,w
```
4 ciclos, 4 words, resultado en W

```
        mov A,OFFSET TABLA      ; tabla
        add A,OFFST             ; + offset
        mov MP,A                ; indirecto
        mov A,IAR
```
4 ciclos, 4 words, resultado en A

Table read, tabla en memoria de programa

Tomamos un valor de una tabla según nos indica una variable que oficia de índice dentro de la misma. La tabla se encuentra en ROM. Microchip tiene otros modelos que permiten realizar una operación similar a Holtek con instrucciones de table read, sin embargo, comparamos contra la serie midrange que por lo general no tiene esta capacidad. En el caso de Holtek, la tabla puede ocupar un espacio dentro del segmento de 256 bytes en memoria de programa en el que se está ejecutando el código, o en el último del mapa de memoria. También es posible hacerlo de la forma en que se hace con Microchip, aunque sin ganancia. En todos los casos, las tablas no deben pasar un segmento de 256 unidades de memoria de programa.

```
        movlw OFFSET
        call getdata
        ...

getdata:
        addwf PCL,f
        retlw dato1
        retlw dato2
        ...                     ; tabla en forma de instrucciones en memoria de programa
```
7 ciclos, 3 words+tabla, resultado en W

Holtek opción 1:

```
        mov A,OFFST
        add A,TABLA
```

```
        mov TBLP,A
        tabrdl var              ; tabla en última página de 256 bytes en memoria de programa
        ...
tablas  .section at 0700h 'code'
TABLA   dc dato1, dato2,...
```

5 ciclos, 4 words+tabla, resultado en *var* (si desea en A, *var=ACC*), compacta

Holtek opción 2:

```
        mov A,OFFST
        call getdata
        ...
getdata:
        addm A,PCL
        ret A,dato1
        ret A,dato2
        ...                     ; tabla en forma de instrucciones en memoria de programa
```

7 ciclos, 3 words+tabla, resultado en A

Direccionamiento indirecto en RAM

Tomamos un valor de un buffer en RAM e incrementamos el puntero

```
        movlw TABLA             ; tabla
        movwf FSR               ; indirecto
        movf INDF,w
        incf FSR
```

4 ciclos, 4 words

```
        mov A,OFFSET tabla
        mov MP,A
        mov A,IAR
        inc MP                  ; incrementa puntero
```

4 ciclos, 4 words

Direccionamiento indirecto en memoria de programa

Tomamos un valor de una tabla en memoria de programa e incrementamos el puntero. Microchip tiene otros modelos que permiten realizar una operación similar a Holtek con instrucciones de table read, sin embargo, comparamos contra la serie midrange que por lo general no tiene esta capacidad. En el caso de Holtek, la tabla puede ocupar un espacio dentro del segmento de 256 bytes en memoria de programa en el que se está ejecutando el código, o en el último del mapa de memoria. También es posible hacerlo de la forma en que se hace con Microchip, aunque sin ganancia. En todos los casos, las tablas no deben pasar un segmento de 256 unidades de memoria de programa.

```
        movlw PUNTERO           ; offset dentro de la tabla
        call getdata
        incf PUNTERO,f
        ...
        call getdata
        incf PUNTERO,f
        ...

getdata:
        addwf PCL,f
        retlw dato1
        retlw dato2
        ...                     ; tabla en forma de instrucciones en memoria de programa
```

8 ciclos, 7 ciclos por lectura

Holtek opción 1:

```
        mov A,PUNTERO           ; offset dentro del segmento de tablas.
        mov TBLP,A
        tabrdl ACC              ; tabla en última página de 256 bytes en memoria de programa
        inc TBLP
        ...
        tabrdl ACC
        inc TBLP
        ...
tablas  .section at 0700h 'code'
TABLA   dc dato1, dato2,...
```

5 ciclos, 3 ciclos por lectura, compacta

Holtek opción 2:

```
        mov A,PUNTERO
        call getdata
        inc PUNTERO
        ...
        call getdata
        inc PUNTERO
        ...
getdata:
        addm A,PCL
        ret A,dato1
        ret A,dato2
        ...                     ; tabla en forma de instrucciones en memoria de programa
```

8 ciclos, 7 ciclos por lectura

Conversión binario a BCD

Finalmente, evaluaremos una subrutina muy común, como la empleada para presentar resultados en un display de 7-segmentos o LCD alfanumérico, es decir, la tradicional conversión de un byte a BCD. En Microchip, al no disponer de instrucción de ajuste decimal, la conversión se realiza por restas sucesivas. Según la nota de aplicación de Microchip AN526, dicha rutina, con iguales prestaciones que la mostrada para Holtek, demora 81 ciclos como peor caso. Los tiempos se incrementan considerablemente conforme aumenta la cantidad de dígitos, mientras con Holtek el aumento es relativamente lineal.

```
BINBCD: mov A,8
        mov COUNTER,A           ; COUNTER=8
        clr RESULT              ; RESULT=0
L1:     rlc SOURCE              ; rota SOURCE a la izquierda (MSB a Cy)
        mov A,RESULT            ; no afecta Cy
        adc A,RESULT            ; suma A + RESULT + Cy, resultado en A (A=2 x RESULT + Cy)
        daa RESULT              ; ajusta A para que sea BCD válido, resultado en RESULT
        sdz COUNTER             ; decrementa 'COUNTER' y si éste es 0, saltea la siguiente instrucción
        jmp L1                  ; vuelve al loop
        ret                     ; resultado en RESULT
60 ciclos, 10 words
```