晶心科技 技術文章

8051 與 AndesCore™ 的軟體差異與移植

……晶…心…科…技…新…聞…聯…絡…人……

市場及技術服務部 Joyce Chen

電話: 03-6668300 ext.254

E-mail: joycechen@andestech.com

Web: www.andestech.com

8051 與 AndesCore™ 的軟體差異與移植

1. 8051 與 AndesCore™

本文將介紹使用 8051 與 AndesCore™差異事項, 並對兩種 CPU 系統相關事項做說明, 後面再介紹從 8051 移植到 AndesCore™上注意事項, 舉中斷向量表及異常處理函數的例子說明差異及移植, 最後簡要介紹 AndesCore™在 MCU 應用的三款 CPU: N705,N801 和 N968A。

2. 8051 與 AndesCore™ 常見差異事項

2.1. 位寬的差異

位元寬是指處理器一次執行指令的資料頻寬。8051 是 8 位寬的處理器， 而 AndesCore™是 32 位寬的處理器, 支援 32 位與 16 位元的混合指令集, 位元數越寬, 在資料的處理方面就更有效率。

8051 指令例子：

```
MOV A, Rn     ; 暫存器傳送到累加器
INC A         ; 累加器加 1
```

AndesCore™指令例子：

1. 32 位元指令

```
MOVI   Rt, imm20s      ; 將一個立即數 imm20s 賦於暫存器 Rt
ADDI   Rt, Ra, imm15s  ; 將一個立即數 imm15s 與 Ra 相加結果賦於 Rt
```

2. 16 位元指令: 在運算元範圍較小時, 可以被編譯成 16 位元指令

```
MOVI55   Rt5, imm5s    ;
ADDI333  Rt3, Ra3, imm3u ;
```

2.2. 指令差異

8051 組合語言共有 111 條指令集，AndeStar™的 V3m 指令集有 157 條，AndeStar™的 V3 指令集有 200 多條，兩種 CPU 的指令集大概可以分為以下幾類：算數運算，如加，減，乘，除等操作；資料傳送，如數據在暫存器與記憶體間的傳送，賦值等；邏輯跳轉，如函數呼叫，無條件跳轉，條件跳轉，中斷返回等;在 AndesCore™中還有特權模式的指令部分,關於兩種指令集的具體差別，可以分別參考對應的指令集介紹文檔。

2.3. 位址空間映射差異

AndesCore™使用 memory map 方式映射位址空間，主要有兩種，記憶體的空間映射，如其中的 RAM 或 ROM 位址，它們用於存放程式運行時的代碼和資料，在 AndesCore™上代碼在 link 後，程式運行的代碼和資料位址會最終確定，Andes 提供了一個簡便的 link script 工具 SaG，可以很方便的對系統中可用的記憶體空間進行分配設定。

另一個是週邊所對應的位址空間，可以通過查看 SoC 對應的手冊瞭解對應的週邊映射的空間範圍及相應的使用方法。

2.4. 堆疊設置差異

8051 的堆疊的起始位置是固定的（部分衍生 8051 可以做程式設定），它通常固定在晶片內的 RAM 中，8051 記憶體空間有限，非常小，程式中所使用的變數存放於特定的資料空間中，並不會放在堆疊空間，所以在 8051 中所需要的堆疊空間很小。而對於 AndesCore™來說,堆疊可以設置在任意合適的 RAM 上。程式運行時所有的區域變數都存放在堆疊中，只需要確保在設計系統的時候有足夠的堆疊空間。在 AndesCore™中有$sp 暫存器記錄堆疊位置，這需要在系統上電或者是系統 reset 後初始化時進行設置。

2.5. 代碼和資料的存儲差異

在 8051 系列單晶片中，資料存儲區可以分為內部資料存儲區以及外部資料存儲區。

內部資料存儲區有幾個區域：data，bdata，idata。
data： 片內 RAM 直接定址區。bdata: 片內 RAM 位定址區。idata: 片內 RAM 間接定址區。

外部資料存儲區又有：xdata，pdata。
xdata 和 pdata：是外部存儲區，有些晶片會帶有 XRAM。

在有些開發工具中，如 Keil，可以通過設置存儲模式來處理，存儲模式決定了預設的記憶體類型,此記憶體類型將應用於函數參數,區域變數和定義時未包含記憶體類型的變數。

SMALL 所有的變數存放在片內 RAM（data 區間）
COMPACT 所有的變數存放在外部存儲區(pdata 區間)
LARGE 所有的變數存放在外部存儲區（xdata 區間）

AndesCore™以記憶體映射的方式，記憶體空間不會有特別的限制，就是說不會像 8051 那樣需放在某處區間，這樣的設計更方便靈活，允許程式碼和資料在可用的空間裡自由放置。

有時候需要將某段代碼或者資料存放在指定的位置上，在 8051 中，可以在代碼中使用"at"關鍵字，但該關鍵字是 8051 中所特有的，會造成可攜性和維護的問題，在 AndesCore™上，提供了一種簡便的 link script 工具，如上所提到的 SaG 工具，在 C 代碼中使用 GNU 標準的語法格式，在 link 之後相應的代碼和資料將存放於指定的位置，這樣可以避免在代碼中使用"at"該平臺相關的屬性設置。

2.6. 資料類型及對齊差異

8051 和 AndesCore™是不同類型的 CPU,它們所使用的資料類型所對應的寬度也不同，如下表所示：

Type	AndesCore™	8051	Notes
char	8-bit signed	8-bit signed	
short	16-bit	16-bit	
int	32-bit	16-bit	int is smaller on 8051
long	32-bit	32-bit	
long long	64-bit	N/A	
float	32-bit	32-bit	
double	64-bit	32-bit	8051 has no 64-bit floating point type
long double	64-bit	N/A	

在連結完成後資料通常都會按照本身的屬性對齊，比如 int 類型則會 4 bytes 對齊，short 則會 2 bytes 對齊。這樣的存放方式可以提高 CPU 對資料讀取時的效率。雖然 AndesCore™是 32bit 的 CPU， 在只需要 8bit 和 16bit 的資料時能節省存儲空間，但在處理 16bit 和 32bit 的資料上則有更高效。

在 8051 中有 sbit 關鍵字用於設置對特殊功能暫存器 SFR 的直接 access，8051 的特殊功能暫存器分佈在記憶體位址 0x80 到 0xFF 處，如下表：

F8H									FFH
F0H	B *								F7H
E8H									EFH
E0h	ACC *								E7H
D8H									DFH
D0H	PSW *								D7H
C8H	T2CON *		RCAP2L	RCAP2H	TL2	TH2			CFH
C0h									C7H
B8H	IP *								BFH
B0h	P3 *								B7H
A8h	IE *								AFH

A0H	P2 *								A7H
98H	SCON *	SBUF							9FH
90H	P1 *								97H
88h	TCON *	TMOD	TL0	TL1	TH0	TH1			8FH
80H	P0 *	SP	DPL	DPH				PCON	87H

sbit 是 8051 擴展的變數類型，非標準 C 語法，移植的時候需要將其修改成標準 C 操作語法，另外在 AndesCore™中，所有的暫存器都是單獨存在的，不會佔用記憶體的空間。

2.7. 指標使用差異

8051 中兩種類型的指標，分別是記憶體指標和通用指標，通用指標由 3 個位元組組成，第一個位元組用來指明對應的記憶體類型，所以這種類型的指標類型佔用空間更大也更慢，記憶體指標只能用來訪問指定類型的記憶體空間。

通用指標：
通用指標的聲明和標準 C 語言中一樣。如：

```
char *s;        /* string ptr */
int *numptr; /* int ptr */
long *state; /* long ptr */
```

記憶體指標：

```
char data *str;       /* ptr to string in data */
int xdata *numtab; /* ptr to int(s) in xdata */
```

而在 AndesCore™上指標不會有這方面的限制，它是一個 32bit 的資料，普通的暫存器就可以存放指標內容，可以訪問到系統 4G 範圍內的空間（N705,N801 位址空間只有 16M,N968A 以上的 CPU 位址空間可達 4G）。

2.8. 函式宣告差異

在 8051 中由於堆疊空間有限，如果有函數是可重入的，需要在函式宣告的時候用關鍵字 reentrant 做說明。8051 的中斷處理函數則需要使用關鍵字 interrupt 聲明，中斷處理函數有時也需要用 using 關鍵字指明哪一暫存器組會被使用到。

在 AndesCore™中，都採用標準的 C 語法，在聲明函數時並不需要這些附加的聲明。AndesCore™遵行底層的 ABI 機制，編譯器處理底層的暫存器及堆疊相關機制。對於上層用戶來說是透明的。

3. 系統相關事項說明

3.1. 操作模式

8051 只有一種 mode，AndesCore™有兩種 mode，分別是 superuser mode 和 user mode，當系統上電啟動時是在 superuser mode，或者當系統進入到中斷或者異常時也進入到 superuser mode，當從中斷或者是異常返回後，會返回到 user mode。由於 8051 沒有 mode 切換的問題，所以在移植的時候只需要理解 AndesCore™在 mode 方面的機制就可以。

3.2. 系統的啟動

8051 和 AndesCore™的系統啟動過程類似,通常在 0 位址存放中斷向量表，第一個向量表是 reset，當系統上電或者是 reset 後，經過該向量會跳轉到一個啟動函數中，該啟動函數會完成系統啟動所必要的步驟，比如設置 CPU，初始化 SoC,清理記憶體,初始化 C 運行環境等, 最後完成所有的準備後跳轉到 main 函數。

3.3. 中斷處理

8051 有 5 個中斷源，通常中斷向量表只是一個跳轉，會跳到真正的中斷處理函數，8051 只能設置成兩級的中斷優先順序。

中斷源	中斷向量
上電復位	0000H
外部中斷 0	0003H
計時器 0 溢出	000BH
外部中斷 1	0013H
計時器 1 溢出	001BH
串列口中斷	0023H

AndesCore™包含了 9 個內部異常，中斷向量號對應於從 0 到 8, 9 之後對應於外部中斷，在 Internal Vector Interrupt Controller （IVIC）mode 時可支援 32 個外部中斷，

Entry number	Entry point
0	Reset/NMI
1	TLB fill
2	PTE not present
3	TLB misc
4	TLB VLPT miss
5	Machine Error
6	Debug related
7	General exception
8	Syscall
9	HW0
10	HW1
11	HW2
12	HW3
…	…
40	HW31

當 External Vector Interrupt Controller（EVIC） mode 時由外部中斷控制器決定,最多有 64 個。

Entry number	Entry point
0	Reset/NMI
1	TLB fill

2	PTE not present
3	TLB misc
4	TLB VLPT miss
5	Machine Error
6	Debug related
7	General exception
8	Syscall
9-72	VEP 0-63

中斷的處理由以下幾部分組成：

1. 實現中斷處理函數

可以用組合語言實現 8051 的中斷處理函數，也可以用 C 來實現，在 8051 中 C 實現的中斷處理函數會有一個"interrupt"的關鍵字，如果有暫存器 bank 被使用到，還要加上"using"關鍵字。如果要將中斷處理函數固定在特定位置還需要使用"at"關鍵字，而 AndesCore™使用的是標準的 C 語法，不需要為中斷處理函數做這些設置。

2. 中斷向量表的產生

8051 中斷向量表擺放在 0 開始的位置，在 AndesCore™中硬體可以設定啟動位址，通常設為 0 位址，也可以是非 0 位址，中斷向量表存放在對應系統啟動位址處。在程式編寫過程中可以通過標準的 gnu 語法再加上 link script 的 SaG 工具，以使產生的中斷向量表在連結的時候存放於特定的位置。

3. 中斷配置

在 8051 中，需要做以下設置

1. IE 暫存器中 Individual Interrupt Enable 位設 1
2. IE 暫存器中 EA(Enable All)位設 1
3. 當是外部中斷時，配置相關的 pin 為輸入，並設置對應的觸發屬性為 edge 或 level 觸發。

而在 AndesCore™中需要做以下設置：

1．設置 CPU IVIC 或者 EVIC mode

2．設置 INT_MASK 位

3．設置中斷的優先順序

4. 關於異常處理差異

   在 8051 中沒有異常處理向量,所以在 8051 中並沒有這部分的處理函數，在 AndesCore™中有一些系統的異常處理向量，比如 Machine Error, General Exception, 建議在 AndesCore™上實現對應的處理函數,當發生這類異常時做一些基本的處理。

3.4. 時序和延遲

在 8051 中可以採用 NOP 指令來延遲，在 AndesCore™中也有 NOP 指令來達到類似目的。

3.5. 電源管理

8051 單晶片中有兩種省電方式，分別是空閒方式和掉電模式,單晶片處於空閒工作方式時，CPU 處於睡眠狀態，它的晶片內其它部件還是會繼續工作，晶片內 RAM 的內容和所有專用暫存器的內容在空閒方式期間都被保存下來了，可以通過中斷或者硬體重定來終止空閒工作方式。單晶片處於掉電工作方式時，晶片內的振盪器停止了工作，因此它的一切都被迫停止了。但晶片內 RAM 的內容和專用暫存器的內容一直保持到掉電方式結束為止。掉電方式的喚醒方式只有一種，就是硬體重定。

在 AndesCore™上,可以通過軟體 standby 指令使 CPU 進入到低功耗模式，通常標準 c 代碼並不能直接控制硬體，Andes 的 compiler 提供了 intrinsic 函數來做到這點。分別是：__nds32_standby_no_wake_grant(), __nds32_standby_wake_grant(), __nds32_standby_wait_done().指定系統進入低功耗模式時被喚醒的方式,分別是外部中斷的中斷喚醒,電源管理模組喚醒,和中斷配合電源管理模組喚醒，可以根據系統需要分別設計。

4. 從 8051 移植到 AndesCore™ 上注意事項

從一個 8051 工程，當移植到 AndesCore™上時有以下注意事項：

1. 記憶體映射，代碼和資料擺放位置相關的設置。
2. 可以不必考慮變數數目，或者是函數的 overlay, 因為在 32bit 的 AndesCore™上開發時記憶體空間通常不會像 8051 那樣小。
3. 如果空間允許，在 AndesCore™上儘量使用 32bit 的資料類型，這樣效率會更高。
4. 在 8051 上用於表示記憶體區域屬性的標誌如(idata, xdata, bdata, pdata 等)在 AndesCore™上可以移除。
5. 在 8051 上不需要設置記憶體區塊模式，比如：small, compact, large 等。
6. 在 8051 上用於表示對象遠近的屬性"near"和"far", 都可以移除，AndesCore™上的指標的訪問可以達到所有位址空間。
7. 在中斷處理函數中不需要像 8051 那樣指定哪塊暫存器塊會被用到的關鍵字"using"。
8. 在 8051 上中斷處理函數就和普通的函數一樣，不需要設置其它的關鍵字，如 interrupt。
9. 如果有 8051 組合語言部分移植到 AndesCore™，需要重新實現，盡可能的用 c 來實現，便於維護和調試。
10. 在 8051 中使用到的#progma 相關部分需要刪除。
11. 在 AndesCore™中函數不需要聲明為"reentrant"屬性。
12. 如果使用了數學運算，在 8051 中默認是使用 32bit 單精確度浮點，如果要保持和 8051 中相同的精度，需要將函數名做一些調整，如將 sin()改成 sinf()。

5. 中斷向量及異常處理函數例子

以中斷向量及中斷處理函數的例子說明差異及移植。

5.1.組合語言實現中斷向量表

[8051]

該例子顯示怎樣用組合語言設置 8051 的中斷向量和中斷處理函數，在 8051 組合語言中 ORG 指定了後面組合語言代碼的位置，後面的中斷向量通常是一個跳躍陳述式。如下例第一個向量跳到主函數 MAIN 函數中，另外一個外部中斷 1,也是一個跳轉指令：LJMP INT 到後面的用組合語言實現的中斷處理函數 INT 中。

```
   ORG     0000H                  /*起始位址*/
   LJMP    MAIN                   /*跳轉到主程序*/
   ORG     0013H                  /*外部中斷 1 的位址*/
   LJMP    INT                   /*跳轉到 INT 執行*/
   ORG     0100H                  /*主程序的起始位址*/
MAIN: MOV    A,#0FEH              /*將 FEH 送給 A*/
   SETB IT1                     /*外部中斷 1 跳變沿觸發方式*/
   SETB EX1                      /*外部中斷 1 開中斷*/
   SETB EA                       /*CPU 開中斷*/
   MOV P0,A                      /*將 A 送給 P0*/
LOP:   LJMP LOP                  /*迴圈等待*/
INT:   RL A                     /*A 迴圈左移*/
   MOV P0,A                      /*將 A 的數值送給 P0*/
   RETI                         /*中斷返回*/
   END                           /*程式結束*/
```

[AndesCore™]

該例子顯示怎樣用組合語言設置 AndesCore™的中斷向量表和中斷處理函數, 該例子中 exception_vector 是中斷向量表的 label, 後面分別表示第 0,1,2,3…個中斷向量，它們只是簡單的跳轉指令，跳到具體的執行實體中去，如 vector 0 跳到_start,做系統相關的初始化操作，_start 是系統啟動代碼，用組合語言來實現。vector 9 後面對應的是外部中斷, 中斷處理函數如 OS_Trap_Interrupt_HW0, OS_Trap_Interrupt_HW1… 它通常用 C 來實現，可以參考後面 5.2 章節的 AndesCore™中斷處理函數範例。

```
! 中斷向量表所在的 section，該 section 在連結後會被存放在第一條指令執行處，通常是 0 位置
      .section .vector, "ax"
!====================================================
! Vector table
!====================================================
      .align 3
exception_vector:                    !    以下是中斷向量表
      j _start                       !  (0) Trap Reset
      j OS_Trap_TLB_Fill             !  (1) Trap TLB fill
      j OS_Trap_PTE_Not_Present      !  (2) Trap PTE not present
      j OS_Trap_TLB_Misc             !  (3) Trap TLB misc
      j OS_Trap_TLB_VLPT_Miss        !  (4) Trap TLB VLPT miss
      j OS_Trap_Machine_Error        !  (5) Trap Machine error
      j OS_Trap_Debug_Related        !  (6) Trap Debug related
      j OS_Trap_General_Exception    !  (7) Trap General exception
      j OS_Trap_Syscall              !  (8) Syscall
      j OS_Trap_Interrupt_HW0        !  (9) Interrupt HW0
      j OS_Trap_Interrupt_HW1        ! (10) Interrupt HW1

ISR_TABLE:
        .long HW0_ISR
```

```
        .long HW1_ISR
      .......
      .......
OS_Trap_Interrupt_HW0:
        SAVE_ALL_HW0
        li    $r0, 0x0
        la    $r1, HW0_ISR
        jral $r1
        RESTORE_ALL_HW0
        iret
```

在上面用組合語言設置 AndesCore™的中斷向量表的例子中，我們需要將中斷向量表最終設定在 0 位址處，可以通過 section 語法配合 SaG 工具實現，例子中我們設定該段的 section 名為.vector, 所以在 SaG 中，我們自訂一個 USER_SECTION 為.vector，並將.vector 放在 0 開始的地方並作為第一個 section。

```
USER_SECTIONS .vector
SDRAM 0x00000000 0x00800000 ; address base 0x00000000,
max_size=8M ;指定 LMA 為從 0 開始
{
  EXEC 0x00000000   ;指定 VMA 為 0
  {
    * (.vector)       ;放在 0 開始的地方並作為第一個 section
    * (+RO,+RW,+ZI)
    STACK = 0x00800000
  }
}
```

通過上面的 SaG 語法，並使用 Andes 提供的 SaG 轉 ld 的工具，可以產生類

似以下的 ld , 在工程進行連結的時候選擇該 ld 時就能確保 .vector 連結的位址位於 0 處。

關於詳細的 SaG 使用，可以參考我們的另一篇文章：《Andes 的分散聚合（SaG）機制》
http://www.andestech.com/cn/news-events/technical-article/2014/Andes20141008.pdf

5.2. 中斷處理函數的 C 實現

[8051]

怎樣用 C 寫 8051 的中斷處理函數範例

```
/* com interrupt handler */
void com_int(void) interrupt 4        // 有指定 interrupt 號
{
   /* com interrupt handler here */
}
```

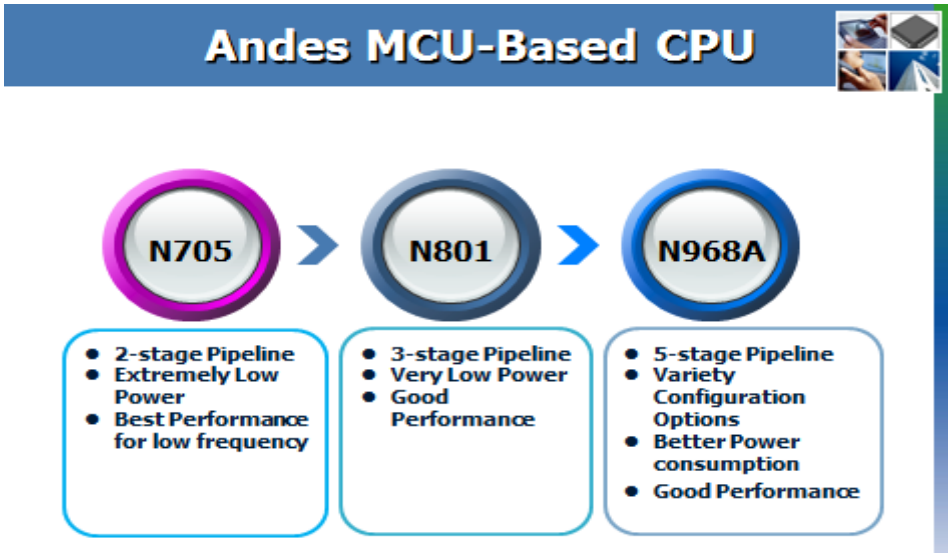
[AndesCore™]

怎樣用 C 寫 AndesCore™的中斷處理函數範例

```
void HW0_ISR ()                  // 和普通函數的寫法相同
{
      puts("* Enter HW0 ISR with GPIO *\n");
}
```

6. 適用於 MCU 的 Andes CPUs

Andes 有三款非常適用于 MCU 應用的 CPU,分別是：N705,N801,N968A,如下圖所列：



N705 和 N801 分別採用了兩級和三級流水線,都具有很低的功耗和很好的性能，當應用需要的頻率較低時，使用兩級流水線的 N705 能發揮出更好的性能和更低功耗的特性，相比於 8051，兩級流水線的 N705 在頻率方面高出許多，比如在 TSMC 40nm LP 工藝下能跑到超過 240MHz，所以完全能勝任 8051 的應用需求。N968A 使用了五級的流水線，同樣有低功耗的特性和很好的性能，同時該款 CPU 具有很強的可配置性，如支援多種匯流排界面，還支持了專門為 audio 的加速指令，N968A 是一個性能好，功耗低，又具備強大的可配置特性，適合於多種應用。

7. 總結

AndesCore™使用標準的 C 語法開發,方便快捷,同時作為 32 位元 RISC(精簡指令集) 架構的 CPU , AndesCore™有多款適用于 MCU 應用的 CPU,相對於 8051 具有功耗，性能方面優勢。想瞭解更多 AndesCore™細節，可以登錄 www.andestech.com。

……晶…心…科…技…新…聞…聯…絡…人……

市場及技術服務部 Joyce Chen

電話: 03-6668300 ext.254

E-mail: joycechen@andestech.com

Web: www.andestech.com