飞思卡尔半导体
应用笔记

文件编号：AN3266
第 1 版，5/2006

RS08 使用向导

by: Vincent Ko
Systems Engineering
Microcontroller Division

RS08 是飞思卡尔推出的超低成本 8 位 MCU 内核，本文主要介绍 RS08 平台。

第一部分为用户提供使用 RS08 的基本信息，第二部分包括沿用技术和概念的应用示例。

目录

1 RS08 介绍……1
1.1 RS08 架构……2
1.2 RS08 指令集……6
1.3 内存分页结构……15
1.4 MCU 复位……16
1.5 等待模式……17
1.6 停止模式……17
1.7 子程序调用……18
1.8 中断……19
2 模拟 ADC 应用实例……22
2.1 执行……22
2.2 校准……27
2.3 测量结果……28
附录 A
程序实例……30

1 RS08 介绍

本部分通过介绍 RS08 架构、编程模式、指令集等来帮助用户了解 RS08 平台；一般会以 MC9RS08KA2 为基准，通过参考实例的方式来作介绍，有必要的话，也会提供飞思卡尔 HC08 和 S08 平台的交叉参考。

1.1 RS08 架构

RS08 平台是专为超低成本的应用而设计。由于硬件部分设计的优化导致整个系统成本降低；RS08 的管芯可以放在较小的封装中，譬如 6 针双列平面无引脚（Dual Flat No lead ：DFN）封装。RS08 平台保留与流行的 HC08/S08 平台相似的编程模式可以轻易实现平台间程序源代码的移植。

RS08 平台的主要特点有：

- 指令集为 S08 的子指令集；
- 映像程序计数器（Shadow Program Counter ：SPC）的新指令：SHA 和 SLA；
- 为代码长度优化设计的新的微短寻址模式；
- 最大 16K 的可寻址存储空间；
- 取消了中断和复位服务中的向量提取机制；
- 取消了子程序调用中的 RAM 堆栈机制；
- 子程序调用提供单层硬件堆栈；
- 通过执行 STOP 和 WAIT 指令提供低功耗模式的支持；
- 通过内部或外部中断触发实现 Stop 状态的唤醒；
- 提供非法寻址和操作码检测复位；
- 硬件加密实现对非遗失性的存储空间（NVM）非法访问的保护；
- 用单线接口实现调试和非遗失性存储器的编程/擦除

1.1.1 CPU 寄存器

RS08 的 CPU 寄存器包括一个 8 位通用累加器（A），一个 14 位的程序计数器（PC），一个 14 位的映像程序计数器（SPC）和一个 2 位的状态码寄存器（CCR）。CCR 包含 2 个状态标志，提供给像 BCS 和 BEQ 等条件跳转指令来使用。图 1-1 为 RS08 CPU 寄存器。

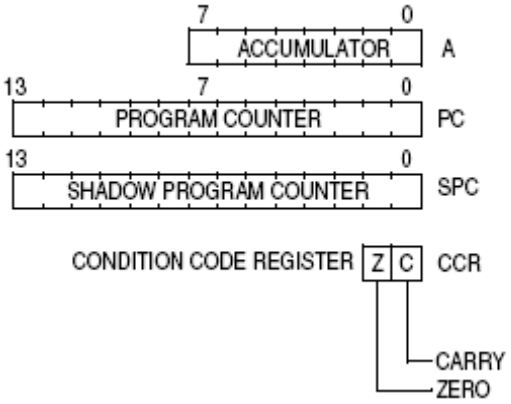


图 1-1. RS08 CPU 寄存器

8 位通用累加器 A 是 RS08 CPU 的基本数据寄存器。利用 LDA 指令可以将数据从存储器中读到 A 中，利用 STA 指令可以将数据从 A 中写入存储器中。新增加的 2 条指令 SHA 和 SLA 可以分别完成累加器 A 与映像程序计数器（SPC）的高 8 位和低 8 位的数据交换。

程序计数器（PC）中存放下一条指令的地址或操作数，由于 RS08 的程序计数器长度为 14 位，意味着 RS08 的最大寻址空间为 16K 字节。

在 HC08/S08 中，在使用 JSR 和 BSR 指令调用子程序时，PC 返回值以堆栈方式放在 RAM 中；在 RS08 中，由于没有 RAM 堆栈机制，所以返回的地址保存在 SPC 寄存器中，在完成子程序调用后，RTS 指令会从 SPC 中恢复 PC 的值。SPC 只能保存单层的地址，所以嵌套的子程序调用必须利用软件堆栈来完成。用户程序可以先利用 SHA 和 SLA 指令来分别完成 A 与 SPC 的高字节和低字节的内容交换，然后堆栈在 RAM 中。

状态寄存器（CCR）的状态位（Z 和 C）反应前一数学运算或其他运算的结果。位的定义与 HC08/S08 中的定义相同，详细定义信息请参考 RS08 内核参考手册。

1.1.2 特殊寄存器

除了 CPU 寄存器，还有两个内存映射寄存器与内核地址产生紧密相关；这就是间接数据寄存器（D[X]）和索引寄存器（X），它们的地址分别位于$000E 和$000F。如图 2 所示。

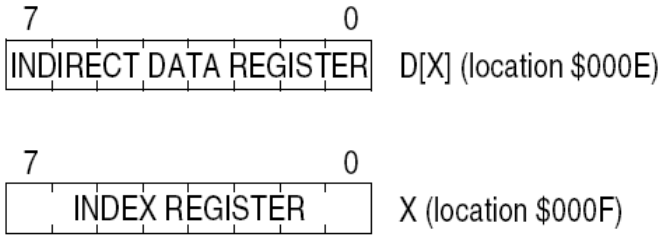


Figure 1-2. RS08 Special Registers

寄存器 D[X]和 X 共同实现间接数据寻址；读取寄存器 D[X]时，X 寄存器内容为对应寄存器的地址。图 1-3 说明了间接寻址方案。X 和 D[X]寄存器不是 CPU 内部寄存器，但是它们与 RS08 通用指令集的无缝集成形成了伪指令集。

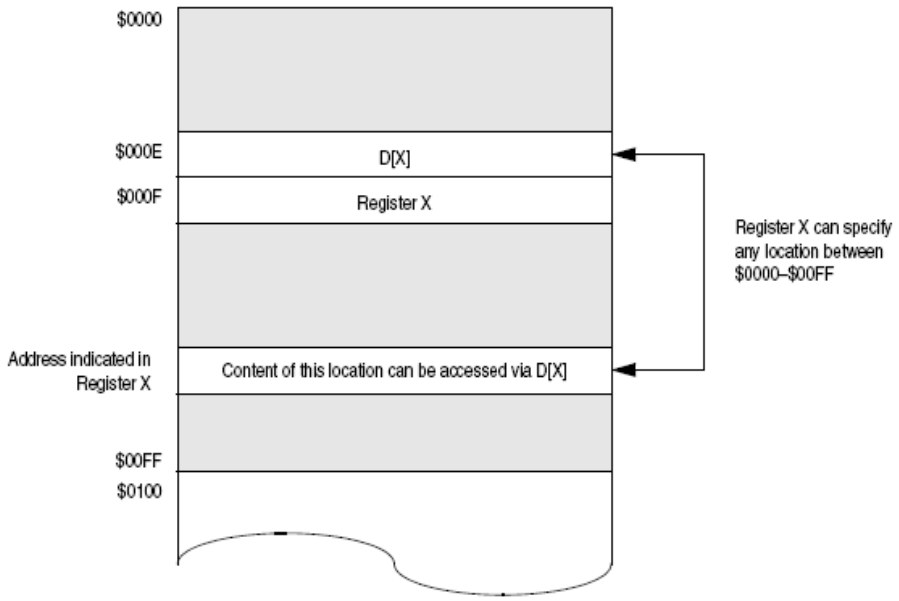


Figure 1-3. Index Addressing Scheme

1.1.3 通用寻址模式

每当 MCU 从存储器读/写数据时，都会使用一种寻址模式来判断数据的实际地址。表 1-1 总结了 RS08 平台支持的通用寻址模式。

Table 1-1. RS08 Addressing Modes

Addressing Mode	Example
Inherent Addressing	CLRA, INCA, SHA, RTS
Direct Addressing	LDA $20, AND $20
Relative Addressing	BRA, BCS, BEQ
Immediate Addressing	LDA #9
Tiny Addressing	INC <$0D
Short Addressing	CLR <$1D
Extended Addressing	JMP, JSR

1.1.3.1 与 HC08/S08 通用的寻址模式

固定寻址、直接寻址、相对寻址、立即数寻址和扩展寻址等寻址模式在 RS08 中的操作与在 HC08/S08 中的操作相同。固定寻址模式用在当 CPU 已经知道要完成指令的所有的信息并且在源代码中不提供任何的寻址信息。相对寻址模式用于跳转指令中明确指定相对于程序计数器的地址偏移量。立即数寻址用于当在一条指令中给出具体数值，这个值位于该指令操作码后面。直接寻址模式用于访问在直接地址空间（从$0000 到$00FF）的操作数。扩展寻址模式用于中含有 2 字节操作数的指令，这种模式仅用于 JMP 和 JSR 指令中，指令中在操作数中给出 14 位的目标地址。

1.1.3.2 微短寻址模式

RS08 平台中引入微短寻址模式，该寻址模式与直接寻址模式具有相似的操作，但是寻址空间受到限制，只有直接寻址的可访问空间（在$0000 - $00FF 范围内）的部分空间能够被微短寻址模式访问到。当然，与这些寻址模式相关的所有指令都是单字节指令，最大限度的使用这些指令可以减少整个的代码长度。

微寻址模式适用于地址影像中前 16 字节空间的寻址，从$0000 到$000F，对应的指令有加 1（INC）指令、减 1（DEC）指令，加（ADD）指令以及减（SUB）指令。直接寻址模式中有同样功能的 2 字节指令，寻址空间从$0000 到$00FF。用户在源程序中在操作数之前加上小于号（<），可以强迫汇编编译器使用微寻址指令。

```
INC  <$0D
DEC <$0D
ADD <$0D
SUB <$0D
```

短寻址模式用于访问前 32 字节的空间，从$0000 到$001F，适用于清除（CLR）指令，加载累加器 A（LDA）指令和存储累加器 A（STA）指令。用户在源程序中在操作数之前加上小于号（<），可以强迫汇编编译器使用短寻址指令。

```
CLR<$1F
LDA <$1F
STA <$1F
```

1.1.3.3 伪寻址模式

在使用特殊寄存器 D[X]和 X 时，RS08 通用指令集可以被用来模拟 HC08/S08 架构中的累加器 X 的操作。这种模拟由汇编程序/编译器支持，在编译过程中完成。当需要零偏移索引指令或与寄存器 X 相关的指令时，用户可以在 RS08 程序中使用与 HC08/S08 相同的编辑语法。在编译中汇编程序会转换 RS08 伪指令为等同的通用 RS08 指令，这个操作对用户是透明的。

下面总结由 RS08 架构支持的伪寻址模式：

- 伪固定寻址 - 例如：TSTX，DBNZX 等伪固定寻址是利用等同的直接寻址模式模拟，操作数通过 X 寄存器（位置：$000F）加载。有的操作，像 DECX 和 INCX，可利用微短寻址模式指令，则伪指令变为单字节指令。
- 伪直接寻址 - 例如：LDX $20，STX $20 是通过直接 - 直接移动（MOV）操作来模拟的。LDX 操作等同于移动操作数到 X 寄存器（$000F）；STX 操作把 X 寄存器（$000F）中的内容到操作数的目标地址。
- 伪立即数寻址 - 例如：LDX #$09 是通过立即数 - 直接移动（MOV）操作来模拟的。X 寄存器直接加载指定的数据。
- 伪零偏移索引寻址 - 例如：ADD ，X 是通过操作数总是从 D[X]寄存器加载的直接寻址操作来模拟。D[X]寄存器保存的间接数据的地址在 X 寄存器中；从而在 D[X]寄存器上的操作等同于 HC08/S08 中的零偏移索引寻址。RS08 平台中保留相同与 HC08/S08 风格的译码语法，这样可以帮助用户方便的在这些平台之间移植源代码。下面是一些代码例子：

```
LDA   ,X
ADD   ,X
DBNZ ,X,rel
```

注意

伪指令基于模拟完成的，都有等同的 HC08/S08 操作；然而，在 CPU 的时钟周期计数和指令字节数计数上是不同的。在从 HC08/S08 平台向 RS08 平台移植源代码的时候要特别注意定时要求严格的软件。

1.2 RS08 指令集

RS08 的 CPU 内核可以认为是简化版的 S08 内核。为保持源代码的兼容性，在 RS08 架构中保留了大部分的算术指令，但去除了用于复杂运算的一些指令：半位元交换（NSA）、乘（MUL）和除（DIV）。

由于没有堆栈机制，在 RS08 指令去掉了包含堆栈指针操作的指令；同样，状态寄存器（CCR）包括 Z 和 C 两个状态标志位，所以在 RS08 指令中只有包含这两个位的条件转移指令。

表 1-2 总结了 RS08 和 HC08/S08 指令集的差别

Table 1-2. RS08 and S08 Instruction Set Comparison

Description	RS08	S08	Operation
Arithmetic Operations:			
Add with Carry	ADC #*opr8* ADC *opr8* ADC ,X [1] ADC X [1,2]	ADC #*opr8* ADC *opr8* ADC *opr16* ADC *opr8*,X ADC *opr16*,X ADC ,X ADC *opr8*,SP ADC *opr16*,SP	A ← (A) + (M) + (C) A ← (A) + (X) + (C) [2]

Add without Carry	ADD #*opr8*
	ADD *opr8*
	ADD *opr4*
	ADD ,X [1]
	ADD X [1,2]
		ADD #*opr8*
		ADD *opr8*
		ADD *opr16*
		ADD *opr8*,X
		ADD *opr16*,X
		ADD ,X
		ADD *opr8*,SP
		ADD *opr16*,SP
			A ← (A) + (M)
			A ← (A) + (X) [2]

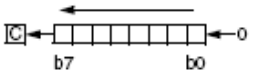
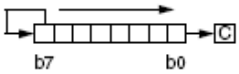
Add Immediate Value (Signed) to Stack Pointer		AIS #*opr8*	SP ← (SP) + (16 « M)
Add Immediate Value (Signed) to Index Register (H:X)		AIX #*opr8*	H:X ← (H:X) + (16 « M)
Arithmetic Shift Left (Same as LSL)	ASLA	ASL *opr8* ASLA ASLX ASL *opr8*,X ASL ,X ASL *opr8*,SP	
Arithmetic Shift Right		ASR *opr8* ASRA ASRX ASR *opr8*,X ASR ,X ASR *opr8*,SP	
Clear	CLR *opr8* CLR *opr5* CLRA CLRX [1] CLR ,X [1]	CLR *opr8* CLRA CLRX CLRH CLR *opr8*,X CLR ,X CLR *opr8*,SP	M ← $00 A ← $00 X ← $00
Decimal Adjust Accumulator		DAA	$(A)_{10}$
Decrement	DEC *opr8* DEC *opr4* DECA DECX [1] DEC ,X [1]	DEC *opr8* DECA DECX DEC *opr8*,X DEC ,X DEC *opr8*,SP	M ← (M) – $01 A ← (A) – $01 X ← (X) – $01
Divide		DIV	A ← (H:A)/(X) H ← Remainder
Increment	INC *opr8* INC *opr4* INCA INCX [1] INC ,X [1]	INC *opr8* INCA INCX INC *opr8*,X INC ,X INC *opr8*,SP	M ← (M) + $01 A ← (A) + $01 X ← (X) + $01

Table 1-2. RS08 and S08 Instruction Set Comparison (continued)

Description	RS08	S08	Operation
Negate (Two's Complement)		NEG *opr8* NEGA NEGX NEG *opr8*,X NEG ,X NEG *opr8*,SP	M ← –(M) = $00 – (M) A ← –(A) = $00 – (A) X ← –(X) = $00 – (X)
Subtract with Carry	SBC #*opr8* SBC *opr8* SBC ,X [1] SBC X [1, 2]	SBC #*opr8* SBC *opr8* SBC *opr16* SBC *opr8*,X SBC *opr16*,X SBC ,X SBC *opr8*,SP SBC *opr16*,SP	A ← (A) – (M) – (C) A ← (A) – (X) – (C) [2]

Subtract	SUB #*opr8*
	SUB *opr8*
	SUB *opr4*
	SUB ,X [1]
	SUB X [1, 2]
		SUB #*opr8*
		SUB *opr8*
		SUB *opr16*
		SUB *opr8*,X
		SUB *opr16*,X
		SUB ,X
		SUB *opr8*,SP
		SUB *opr16*,SP
			A ← (A) – (M)
			A ← (A) – (X) [2]

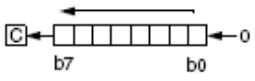
Logical Operations:			
Logical AND	AND #*opr8* AND *opr8* AND ,X [1] AND X [1, 2]	AND #*opr8* AND *opr8* AND *opr16* AND *opr8*,X AND *opr16*,X AND ,X AND *opr8*,SP AND *opr16*,SP	A ← (A) & (M) A ← (A) & (X) [2]
Clear Bit n in Memory	BCLR *n*,*opr8* BCLR *n*,X [1, 2] BCLR *n*,D[X] [1, 2]	BCLR *n*, *opr8*	Mn ← 0 Xn ← 0 [2]
Set Bit *n* in Memory	BSET *n*,*opr8* BSET *n*,X [1, 2] BSET *n*,D[X] [1, 2]	BSET *n*, *opr8*	Mn ← 1 Xn ← 1 [2]
Complement (One's Complement)	COMA	COM *opr8* COMA COMX COM *opr8*,X COM ,X COM *opr8*,SP	M ← ($\overline{M}$)= $FF – (M) A ← ($\overline{A}$) = $FF – (M) X ← ($\overline{X}$) = $FF – (M)
Exclusive OR Memory with Accumulator	EOR #*opr8* EOR *opr8* EOR ,X [1] EOR X [1, 2]	EOR #*opr8* EOR *opr8* EOR *opr16* EOR *opr8*,X EOR *opr16*,X EOR ,X EOR *opr8*,SP EOR *opr16*,SP	A ← (A ⊕ M) A ← (A ⊕ X) [2]
Logical Shift Left (Same as ASL)	LSLA	LSL *opr8* LSLA LSLX LSL *opr8*,X LSL ,X LSL *opr8*,SP	

Table 1-2. RS08 and S08 Instruction Set Comparison (continued)

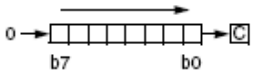
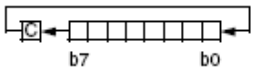
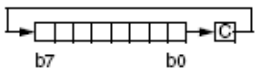
Description	RS08	S08	Operation
Logical Shift Right	LSRA	LSR *opr8* LSRA LSR*X* LSR *opr8*,X LSR ,X LSR *opr8*,SP	
Nibble Swap Accumulator		NSA	A ← (A[3:0]:A[7:4])
Inclusive OR Accumulator and Memory	ORA *#opr8* ORA *opr8* ORA ,X[1] ORA X[1,2]	ORA *#opr8* ORA *opr8* ORA *opr16* ORA *opr8*,X ORA *opr16*,X ORA ,X ORA *opr8*,SP ORA *opr16*,SP	A ← (A) ! (M) A ← (A) ! (X)[2]
Rotate Left through Carry	ROLA	ROL *opr* ROLA ROLX ROL *opr*,X ROL ,X ROL *opr*,SP	
Rotate Right through Carry	RORA	ROR *opr* RORA RORX ROR *opr*,X ROR ,X ROR *opr*,SP	
Branch Operations:			
Branch if Carry Bit Clear	BCC *rel*	BCC *rel*	PC ← (PC) + $0002 + rel ? (C) = 0
Branch if Carry Bit Set (Same as BLO)	BCS *rel*	BCS *rel*	PC ← (PC) + $0002 + *rel* ? (C) = 1
Branch if Equal	BEQ *rel*	BEQ *rel*	PC ← (PC) + $0002 + *rel* ? (Z) = 1
Branch if Greater Than or Equal To (Signed Operands)		BGE *opr*	PC ← (PC) + $0002 + *rel* ? (N ⊕ V) = 0
Branch if Greater Than (Signed Operands)		BGT *opr*	PC ← (PC) + $0002 + *rel* ? (Z) ! (N ⊕ V) = 0
Branch if Half Carry Bit Clear		BHCC *rel*	PC ← (PC) + $0002 + *rel* ? (H) = 0
Branch if Half Carry Bit Set		BHCS *rel*	PC ← (PC) + $0002 + *rel* ? (H) = 1
Branch if Higher		BHI *rel*	PC ← (PC) + $0002 + *rel* ? (C) ! (Z) = 0
Branch if Higher or Same (Same as BCC)	BHS *rel*	BHS *rel*	PC ← (PC) + $0002 + *rel* ? (C) = 0
Branch if $\overline{\text{IRQ}}$ Pin High		BIH *rel*	PC ← (PC) + $0002 + *rel* ? $\overline{\text{IRQ}}$ = 1
Branch if $\overline{\text{IRQ}}$ Pin Low		BIL *rel*	PC ← (PC) + $0002 + *rel* ? $\overline{\text{IRQ}}$ = 0
Branch if Less Than or Equal To (Signed Operands)		BLE *opr*	PC ← (PC) + $0002 + *rel* ? (Z) ! (N ⊕ V) = 1
Branch if Lower (Same as BCS)	BLO *rel*	BLO *rel*	PC ← (PC) + $0002 + *rel* ? (C) = 1
Branch if Lower or Same		BLS *rel*	PC ← (PC) + $0002 + *rel* ? (C) ! (Z) = 1

Table 1-2. RS08 and S08 Instruction Set Comparison (continued)

Description	RS08	S08	Operation
Branch if Less Than (Signed Operands)		BLT *opr*	PC ← (PC) + $0002 + *rel* ? (N ⊕ V) = 1
Branch if Interrupt Mask Clear		BMC *rel*	PC ← (PC) + $0002 + *rel* ? (I) = 0
Branch if Minus		BMI *rel*	PC ← (PC) + $0002 + *rel* ? (N) = 1
Branch if Interrupt Mask Set		BMS *rel*	PC ← (PC) + $0002 + *rel* ? (I) = 1
Branch if Not Equal	BNE *rel*	BNE *rel*	PC ← (PC) + $0002 + *rel* ? (Z) = 0
Branch if Plus		BPL *rel*	PC ← (PC) + $0002 + *rel* ? (N) = 0
Branch Always	BRA *rel*	BRA *rel*	PC ← (PC) + $0002 + *rel*
Branch if Bit *n* in Memory Clear	BRCLR *n,opr8,rel* BRCLR *n*,X,*rel* [1,2] BRCLR *n*,D[X],*rel* [1,2]	BRCLR *n*, *opr8*,*rel*	PC ← (PC) + $0003 + *rel* ? (Mn) = 0 PC ← (PC) + $0003 + *rel* ? (Xn) = 0 [2]
Branch Never		BRN *rel*	PC ← (PC) + $0002
Branch if Bit *n* in Memory Set	BRSET *n,opr8,rel* BRSET *n*,X,*rel* [1,2] BRSET *n*,D[X],*rel* [1,2]	BRSET *n*, *opr8*,*rel*	PC ← (PC) + $0003 + *rel* ? (Mn) = 1 PC ← (PC) + $0003 + *rel* ? (Xn) = 1 [2]
Branch to Subroutine	BSR *rel*	BSR *rel*	**For S08:** PC ← (PC) + $0002; push (PCL) SP ← (SP) – $0001; push (PCH) SP ← (SP) – $0001 PC ← (PC) + *rel* **For RS08:** PC ← (PC) + 2 Push PC to shadow PC PC ← (PC) + *rel*
Compare and Branch if Equal	CBEQ *opr8,rel* CBEQA #*opr8,rel* CBEQ X *rel* [1,2] CBEQ ,X,*rel* [1,2]	CBEQ *opr8,rel* CBEQA #*opr8,rel* CBEQX #*opr8,rel* CBEQ *opr8*, X+,*rel* CBEQ X+,*rel* CBEQ *opr8*,SP,*rel*	**For S08:** PC ← (PC) + $0003 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0003 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0003 + *rel* ? (X) – (M) = $00 PC ← (PC) + $0003 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0002 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0004 + *rel* ? (A) – (M) = $00 **For RS08:** PC ← (PC) + $0003 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0003 + *rel* ? (A) – (X) = $00 [2]
Decrement and Branch if Not Zero	DBNZ *opr8,rel* DBNZA *rel* [1] DBNZX *rel* [1] DBNZ ,X,*rel* [1,2]	DBNZ *opr8,rel* DBNZA *rel* DBNZX *rel* DBNZ *opr8*, X,*rel* DBNZ X,*rel* DBNZ *opr8*, SP,*rel*	A ← (A) – $0001 or M ← (M) – $01 or X ← (X) – $0001 **For S08:** PC ← (PC) + $0003 + *rel* if (result) ≠ 0 for DBNZ direct, IX1 PC ← (PC) + $0002 + *rel* if (result) ≠ 0 for DBNZA, DBNZX, or IX PC ← (PC) + $0004 + *rel* if (result) ≠ 0 for DBNZ SP1 **For RS08:** PC ← (PC) + $0003 + *rel* if (result) ≠ 0 for DBNZ direct, DBNZX, DBNZ ,X PC ← (PC) + $0002 + *rel* if (result) ≠ 0 for DBNZA
Jump	JMP *opr16*	JMP *opr8* JMP *opr16* JMP *opr8*,X JMP *opr*16,X JMP ,X	PC ← Jump Address

Table 1-2. RS08 and S08 Instruction Set Comparison (continued)

Description	RS08	S08	Operation
Jump to Subroutine	JSR *opr16*	JSR *opr8* JSR *opr16* JSR *opr*16,X JSR *opr*8,X JSR ,X	**For S08:** PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) – $0001 Push (PCH); SP ← (SP) – $0001 PC ← Unconditional Address **For RS08:** PC ← (PC) + 3 Push PC to shadow PC PC ← Unconditional Address
Return from Subroutine	RTS	RTS	**For S08:** SP ← SP + $0001; Pull (PCH) SP ← SP + $0001; Pull (PCL) **For RS08:** Pull PC from shadow PC
Data Verification Operations:			
Bit Test		BIT #*opr8* BIT *opr8* BIT *opr116* BIT *opr*8,X BIT *opr*16,X BIT ,X BIT *opr8*,SP BIT *opr*16,SP	(A) & (M)

Compare Accumulator with Memory	CMP #*opr8*
	CMP *opr8*
	CMP ,X [1]
	CMP X [1, 2]
		CMP #*opr8*
		CMP *opr8*
		CMP *opr16*
		CMP *opr8*,X
		CMP *opr*16,X
		CMP ,X
		CMP *opr8*,SP
		CMP *opr*16,SP
			(A) – (M)
			(A) – (X) [2]

Complement (One's Complement)		CPHX #*opr8* CPHX *opr8* CPHX *opr16* CPHX *opr8*,SP	(H:X) – (M:M + $0001)
Compare Index Register (H:X) with Memory		CPX #*opr8* CPX *opr8* CPX *opr16* CPX ,X CPX *opr8*,X CPX *opr*16,X CPX *opr8*,SP CPX *opr*16,SP	(X) – (M)
Test for Negative or Zero	TST *opr8* [1] TSTA [1] TSTX [1]	TST *opr8* TSTA TSTX TST *opr*8,X TST ,X TST *opr*8,SP	(A) – $00 (X) – $00 (M) – $00
Data Movement Operations:			

Load Accumulator from Memory	LDA #*opr8*
	LDA *opr8*
	LDA *opr5*
	LDA ,X [1]
		LDA #*opr8*
		LDA *opr8*
		LDA *opr16*
		LDA *opr8*,X
		LDA *opr*16,X
		LDA ,X
		LDA *opr8*,SP
		LDA *opr*16,SP
			A ← (M)

Table 1-2. RS08 and S08 Instruction Set Comparison (continued)

Description	RS08	S08	Operation

Load Index Register (H:X) from Memory		LDHX #*opr16*
		LDHX *opr8*
		LDHX opr16
		LDHX
		LDHX *opr8*,X
		LDHX *opr*16,X
		LDHX *opr8*,SP
			H:X ← (M:M + $0001)

Load X (Index Register Low) from Memory	LDX #*opr8* [1]
	LDX *opr8* [1]
		LDX #*opr8*
		LDX *opr8*
		LDX *opr16*
		LDX *opr8*,X
		LDX *opr*16,X
		LDX ,X
		LDX *opr8*,SP
		LDX *opr*16,SP
			X ← (M)

Move	MOV *opr8,opr8*
	MOV #*opr8,opr8*
	MOV D[X],*opr8* [1]
	MOV *opr8,D[X]* [1]
	MOV #*opr8,D[X]* [1]
		MOV *opr8,opr8*
		MOV *opr8*,X+
		MOV #*opr8,opr8*
		MOV X+,*opr8*
			For S08/RS08:
			$(M)_{destination} \leftarrow (M)_{source}$
			For S08 only:
			H:X ← (H:X) + $001 in IX+D and DIX+ Modes

Store Accumulator in Memory	STA *opr8* STA *opr5* STA ,X [1]	STA *opr8* STA *opr16* STA *opr8*,X STA *opr*16,X STA ,X STA *opr8*,SP STA *opr*16,SP	M ← (A)
Store H:X (Index Reg.)		STHX *opr* STHX *opr* STHX *opr*,SP	(M:M + $0001) ← (H:X)
Store X (Index Register Low) in Memory	STX *opr8* [1]	STX *opr8* STX *opr16* STX *opr8*,X STX *opr*16,X STX ,X STX *opr8*,SP STX *opr*16,SP	M ← (X)
Transfer Accumulator to CCR		TAP	CCR ← (A)
Transfer Accumulator to X (Index Register Low)	TAX [1]	TAX	X ← (A)
Transfer CCR to Accumulator		TPA	A ← (CCR)
Transfer SP to Index Reg.		TSX	H:X ← (SP) + $0001
Transfer X (Index Reg. Low) to Accumulator	TXA [1]	TXA	A ← (X)
Transfer Index Reg. to SP		TXS	(SP) ← (H:X) – $0001
Other Operations:			
Background	BGND	BGND	Enter Background Debug Mode
Clear Carry Bit	CLC	CLC	C ← 0
Clear Interrupt Mask Bit		CLI	I ← 0
No Operation	NOP	NOP	None
Push Accumulator onto Stack		PSHA	Push (A); SP ← (SP) – $0001
Push H (Index Register High) onto Stack		PSHH	Push (H); SP ← (SP) – $0001

Table 1-2. RS08 and S08 Instruction Set Comparison (continued)

Description	RS08	S08	Operation
Push X (Index Register Low) onto Stack		PSHX	Push (X); SP ← (SP) – $0001
Pull Accumulator from Stack		PULA	SP ← (SP + $0001); Pull (A)
Pull H (Index Register High) from Stack		PULH	SP ← (SP + $0001); Pull (H)
Pull X (Index Register Low) from Stack		PULX	SP ← (SP + $0001); Pull (X)
Reset Stack Pointer		RSP	SP ← $FF
Return from Interrupt		RTI	SP ← (SP) + $0001; Pull (CCR) SP ← (SP) + $0001; Pull (A) SP ← (SP) + $0001; Pull (X) SP ← (SP) + $0001; Pull (PCH) SP ← (SP) + $0001; Pull (PCL)
Swap Shadow PC High with A	SHA		A ⇔ SPCH
Swap Shadow PC Low with A	SLA		A ⇔ SPCL
Set Carry Bit	SEC	SEC	C ← 1
Set Interrupt Mask Bit		SEI	I ← 1
Enable $\overline{IRQ}$ pin; Stop Osc.	STOP	STOP	Stop Oscillator I bit ← 0 for S08 only;
Software Interrupt		SWI	PC ← (PC) + $0001; Push (PCL) SP ← (SP) – $0001; Push (PCH) SP ← (SP) – $0001; Push (X) SP ← (SP) – $0001; Push (A) SP ← (SP) – $0001; Push (CCR) SP ← (SP) – $0001; I ← 1 PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte
Enable Interrupts; Stop Processor	WAIT	WAIT	I bit ← 0 for S08 only;

注意：

[1] 这是伪指令，CPU 周期计数和指令周期计数可能与 S08 等同的指令不同。

[2] 这个模拟操作在 S08 指令集中没有等同的操作。

1.2.1 微短寻址模式指令

微短寻址模式指令是单字节指令，大量的使用可以有效提高整个代码的密度；所以尽可能的在 16k 空间上使用这些指令，并将常用的变量放在微短寻址模式可寻址空间中。表 1-3 总结了 RS08 支持的微短寻址的指令。

表 1-3. RS08 微短寻址的指令

Description	Tiny/Short Instruction	Addressable Space	Coding Example
Load Accumulator from Memory	LDA *opr5*	$0000 to $001F	LDA *<$1F* LDA <$00
Store Accumulator in Memory	STA *opr5*	$0000 to $001F	STA *<$1F* STA <$00
Clear	CLR *opr5*	$0000 to $001F	CLR *<$1F* CLR <$00
Add without Carry	ADD *opr4*	$0000 to $000F	ADD *<$0F* ADD <$00
Subtract	SUB *opr4*	$0000 to $000F	SUB *<$0F* SUB <$00
Increment	INC *opr4*	$0000 to $000F	INC *<$0F* INC <$00
Decrement	DEC *opr4*	$0000 to $000F	DEC *<$0F* DEC <$00

1.2.2 伪指令

使用位于$000F 的 X 寄存器和位于$000E 的 D[X]寄存器，就可以模拟大部分的 HC08/S08 零偏移索引寻址指令和累加指令。这种索引寻址实际上可以执行所有的直接寻址模式指令。表 1-4 总结了 RS08 支持的所有的伪指令和操作。

注意

指令的转换在汇编程序编译过程中完成，这个过程对用户是透明的。

表 1-4. RS08 的伪指令

Operation	Pseudo Instruction	Emulation	Description	Bytes	Cycles
Add with Carry	ADC ,X ADC X	ADC $0E ADC $0F	A ← (A) + (M) + (C) A ← (A) + (X) + (C)	2 2	3 3
Add without Carry	ADD ,X ADD X	ADD <$0E ADD <$0F	A ← (A) + (M) A ← (A) + (X)	1 1	3 3
Logical AND	AND ,X AND X	AND $0E AND $0F	A ← (A) & (M) A ← (A) & (X)	2 2	3 3
Clear Bit n in Memory	BCLR *n*,D[X] BCLR *n*,X	BCLR *n*, $0E BCLR *n*, $0F	Mn ← 0 Xn ← 0	2 2	5 5
Branch if Bit *n* in Memory Clear	BRCLR *n*,D[X],*rel* BRCLR *n*,X,*rel*	BRCLR *n*, $0E, *rel* BRCLR *n*, $0F, *rel*	PC ← (PC) + $0003 + *rel* ? (Mn) = 0 PC ← (PC) + $0003 + *rel* ? (Xn) = 0	3 3	5 5
Branch if Bit *n* in Memory Set	BRSET *n*,D[X],*rel* BRSET *n*,X,*rel*	BRSET *n*, $0E, *rel* BRSET *n*, $0F, *rel*	PC ← (PC) + $0003 + *rel* ? (Mn) = 1 PC ← (PC) + $0003 + *rel* ? (Xn) = 1	3 3	5 5
Set Bit *n* in Memory	BSET *n*,D[X] BSET *n*,X	BSET *n*, $0E BSET *n*, $0F	Mn ← 1 Xn ← 1	2 2	5 5
Compare and Branch if Equal	CBEQ ,X,*rel* CBEQ X *rel*	CBEQ $0E, *rel* CBEQ $0F, *rel*	PC ← (PC) + $0003 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0003 + *rel* ? (A) – (X) = $00	3 3	5 5

Table 1-4. Pseudo Instructions in RS08 Platform (continued)

Operation	Pseudo Instruction	Emulation	Description	Bytes	Cycles
Clear	CLR ,X CLRX	CLR <$0E CLR <$0F	M ← $00 X ← $00	1 1	2 2
Compare Accumulator with Memory	CMP ,X CMP X	CMP $0E CMP $0F	(A) – (M) (A) – (X)	2 2	3 3
Decrement and Branch if Not Zero	DBNZ ,X,*rel* DBNZX *rel*	DBNZ $0E, *rel* DBNZ $0F, *rel*	M ← (M) – $01 X ← (X) – $01 PC ← (PC) + $0003 + *rel* if (result) ≠ 0	3 3	6 6
Decrement	DEC ,X DECX	DEC <$0E DEC <$0F	M ← (M) – $01 X ← (X) – $01	1 1	4 4
Exclusive OR Memory with Accumulator	EOR ,X EOR X	EOR $0E EOR $0F	A ← (A ⊕ M) A ← (A ⊕ X)	2 2	3 3
Increment	INC ,X INCX	INC <$0E INC <$0F	M ← (M) + $01 X ← (X) + $01	1 1	4 4
Load Accumulator from Memory	LDA ,X	LDA <$0E	A ← (M)	1	3
Load X (Index Register Low) from Memory	LDX #*opr8* LDX *opr8*	MOV #*opr8*, $0F MOV *opr8*, $0F	X ← (M)	3 3	4 5
Inclusive OR Accumulator and Memory	ORA ,X ORA X	ORA $0E ORA $0F	A ← (A) | (M) A ← (A) | (X)	2 2	3 3
Subtract with Carry	SBC ,X SBC X	SBC $0E SBC $0F	A ← (A) – (M) – (C) A ← (A) – (X) – (C)	2 2	3 3
Store Accumulator in Memory	STA ,X	STA <$0E	M ← (A)	1	2
Store X (Index Register Low) in Memory	STX *opr8*	MOV $0F, *opr8*	M ← (X)	3	5
Subtract	SUB ,X SUB X	SUB <$0E SUB <$0F	A ← (A) – (M) A ← (A) – (X)	1	3
Transfer Accumulator to X (Index Register Low)	TAX	STA <$0F	X ← (A)	1	2

Test for Negative or Zero	TST *opr8*
	TSTA
	TSTX
		MOV opr8, opr8
		ORA #$00
		MOV X, X
			(M) – $00
			(A) – $00
			(X) – $00
				3
				2
				3
					5
					2
					5

Transfer X (Index Reg. Low) to Accumulator	TXA	LDA <$0F	A ← (X)	1	3

1.3 存储器分页结构

RS08 指令集中没有扩展寻址能力，在为分页访问保留的直接页中有一个 64 字节的所谓的分页窗口，位于$00C0 到$00FF；通过分页窗口，更高地址的的存储空间可以通过访问直接页访问到。

RS08 整个地址空间为 16K 字节，被分成 256 页，每页 64 字节。通过程序设定 PAGESEL 寄存器（$001F），就可以定义通过分页窗口访问的页。图 1-4 说明了存储空间分页结构。

PAGESEL 寄存器定义了被访问的存储页，X 寄存器指明在分页窗口中指向要访问的高地址位置相应的地址位置。CPU 访问通过 D[X]寄存器和分页窗口，能够索引相应的更高的存储空间的位置。大多伪指令可以利用这个结构执行执行索引寻址来访问更高的存储地址空间。

注意

通过分页窗口访问任何未被合法指定的地址都会产生一个非法寻址复位。

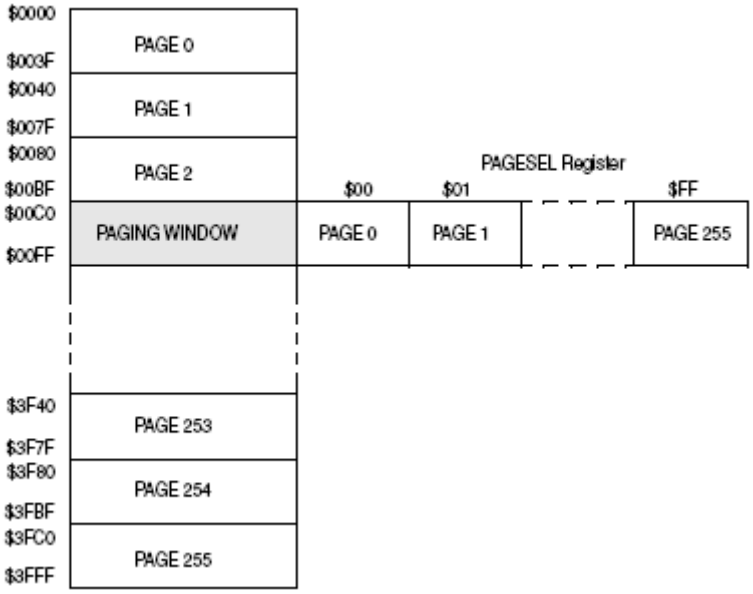


图 1-4.存储空间分页结构

1.4 MCU 复位

MCU 复位将 MCU 恢复到一个已知条件的初始化状态，迫使大多数的控制和状态寄存器恢复初始值，程序计数器（PC）从$3FFD 开始。在 RS08 平台没有中断向量查找机制，一条带 2 字节操作数的 JMP 指令（操作码$BC）必须编程在$3FFD - $3FFF 范围内，操作数表示用户程序开始的位置。

```
;*************************************************************************
; Reset Vector
;*************************************************************************
        org     $3FFC
Security:
        dc.b    $FF             ; SECD=1 is unsecured, SECD=0 is secured
        jmp     main
```

与 HC08/S08 设备相似，RS08 有 7 个复位源：

- 外部管脚复位（PIN）- 由 RSTPE 和 SOPT 使能；
- 上电复位（POR）；
- 低电压检测（LVD）；
- MCU 正常工作（COP）定时器；
- 非法操作码检测（ILOP）；
- 非法地址检测（ILAD）；
- 背景调试模式下，通过 BDC 命令 BDC_RESET 的强迫复位；

系统复位状态寄存器（SRS）位于$0200，包含反应最新的复位源的只读状态标志。

1.5 等待模式

等待模式是通过执行 WAIT 指令开始的，在执行 WAIT 指令后，CPU 进入没有时钟的低功耗状态，程序计数器（PC）停止在跟随 WAIT 指令后面的位置；借助于任何使能的中断源和复位都可以退出等待模式。当一个中断请求发生时：

1. MCU 退出等待模式，恢复进程。
2. 取得接下来需要执行的指令，继续程序流程。

用户程序应该自行检查唤醒 MCU 的中断源。

1.6 停止模式

当在系统选项寄存器中的 STOPE 位设为 1 时，停止模式通过执行 STOP 指令进入；在 STOP 模式，所有与 CPU 相关的内部时钟和模块都被挂起。通过复位、任何使能的异步中断（像 KBI）或者实时中断可从 STOP 模式中退出。当请求发生时：

1. MCU 时钟模块使能。
2. MCU 退出停止模式继续进行进程处理。
3. 取得接下来需要执行的指令，继续程序流程。

用户程序应该自行检查唤醒 MCU 的中断源。

在 STOP 模式下，有选项可以使能不同的模块，例如内部时钟源（ICS），模拟比较器（ACMP）等。如果需要详细信息，请参考详细的器件手册。

注意

如果 STOPE 没有设为 1，当 CPU 执行 STOP 指令时，MCU 不能进入停止模式，并且会强制产生非法操作码复位。

1.7 子程序调用

RS08 平台仅提供单级的硬件堆栈。当 JSR 或 BSR 指令执行时，当前的程序计数器（PC）的值在被修改前保存到映像程序计数器（SPC）寄存器中；遇到 RTS 指令时，被保存的 PC 值从 SPC 寄存器中恢复到程序计数器中，程序从 SPC 寄存器中恢复的地址继续执行。

单级子程序调用对某一些应用来说可能不够，利用 SHA/SLA 指令可以模拟多级的软件堆栈；这些指令分别用来交换 SPC 寄存器的高字节和低字节与累加器 A。软件堆栈可以用来将完成将每一级子程序调用时 SPC 的值放置在 RAM 中。

下面的源代码说明了软件堆栈在宏格式下是如何实现的。在这个例子中，$00 是任意为堆栈指针（STACKPTR）变量选择的存储位置，堆栈内容从$4F 开始往下放置。代码中没有提供堆栈溢出检测。

```
SPInit          equ     $4F             ; Stack block allocation
FLASHSTART      equ     $3800           ; For MC9RS08KA2

RESETSP: MACRO
         mov    #SPInit, STACKPTR       ; Init Stack pointer
         ENDM

PSH_SPC: MACRO                          ; 20 CPU cycles, 14 bytes code
         ; NOTE: Destructive to X content
         ldx    STACKPTR                ; Load Stack pointer
         sha                            ; Swap SPC high byte
         sta    ,X                      ; Push high byte to stack
         sha                            ; Resume A content
         decx                           ; update stack pointer
         sla                            ; Swap SPC low byte
         sta    ,X                      ; Push low byte to stack
         sla                            ; Resume A content
         decx                           ; update stack pointer
         stx    STACKPTR                ; Save stack pointer
         ENDM

PUL_SPC: MACRO                          ; 22 CPU cycles, 14 byte code
         ; NOTE: Destructive to X content
         ldx    STACKPTR                ; Load Stack pointer
         incx                           ; Update stack pointer
         sla                            ; Swap SPC low byte
         lda    ,X                      ; Pull low byte
         sla                            ; Resume A and SPCL content
         incx                           ; Update stack pointer
         sha                            ; Swap SPC high byte
         lda    ,X                      ; Pull high byte
         sha                            ; Resume A and SPCH content
         stx    STACKPTR                ; Save stack pointer
         ENDM

         org    TINY_RAM
STACKPTR         ds.b    1              ; Stack pointer location
         org    FLASHSTART
```

```
;*************************************************************************************
; Subroutine A
;*************************************************************************************
SubA:
        PSH_SPC                         ; Stack SPC
        ;... <Subroutine Content> ...
        bsr SubB                        ; Multi-level subroutine call
        ;... <Subroutine Content> ...
        PUL_SPC                         ; Unstack SPC
        rts
;*************************************************************************************
; Subroutine B
;*************************************************************************************
SubB:
        PSH_SPC                         ; Stack SPC
        ;... <Subroutine Content> ...
        PUL_SPC                         ; Unstack SPC
        rts
;*************************************************************************************
; Main
;*************************************************************************************
Main:
        RESETSP
        ;... <Software Content> ...
        jsr SubA
        ;... <Software Content> ...
        jsr SubB
        ;... <Software Content> ...
```

这儿定义了 3 个宏。RESETSP 用来将堆栈指针初始化；PSH_SPC 用来将 SPC 的内容压入堆栈，减小 STACKPTR 变量；与此相似，PUL_SPC 用来从堆栈中弹出 SPC 内容并且增加 STACKPTR 变量。在每个子程序的开始调用 PSH_SPC，在执行 RTS 指令之前调用 PUL_SPC，这样就会找回每一级调用的子程序的返回地址。

注意

PSH_SPC 和 PUL_SPC 宏指令对 X 寄存器都是破坏性的。如果 X 寄存器的内容需要在子程序调用中传送，需要改进这些宏指令。

1.8 中断

RS08 平台目标市场通常是那些不需要精确中断的小应用。RS08 中需要的中断功能就是将 MCU 从等待模式或停止模式中唤醒，同时相应的中断标志位会被置位以表示该中断事件已经发生。如果同时有多个事件发生，这就需要软件来决定服务的优先级。当 MCU 在运行模式或在背景调试模式（BDM），中断事件不会影响到软件的正常运行。用户可以通过定期轮询中断标志位来检测中断事件，并且判断是否需要中断服务。

与 HC08/S08 平台相似，在 RS08 中，每一个中断源都与对应于相应的中断标志位和中断使能位。只有一个中断源的中断使能位置位后，才能唤醒等待/停止模式中的 MCU。当 MCU 从等待

/停止模式唤醒时，程序继续从停止的地方继续执行。从这一点上，软件可以通过检测中断标志位来决定哪个中断发生了，并且跳转到相应的服务子程序。

单独模块的中断标志位被分散在几个寄存器中，因此，由软件在几个寄存器中检测相应的标志位效率并不高。RS08 平台提供一个系统中断未决（SIP1）寄存器，来集中反映中断源的状态；如果硬件中断使能，当中断事件发生时，在 SIP1 寄存器中相应的标志位会被置位。例如，如果需要键盘中断，通过设定 KBISC 寄存器的 KBIE 位来让该中断使能。当 KBI 中断事件发生时，KBISC 寄存器中 KBF 标志位和 SIP1 寄存器中的 KBI 标志位都会被置位；用户可以选择检测这两个标志位中的任何一个来决定中断事件是否存在。把 KBISC 寄存器的 KBACK 位置 1 可以同时清除 KBISC 寄存器中的 KBF 位和 SIP1 寄存器中的 KBI 位。

1.8.1 中断处理代码示例

与 MC9RS08KA2 相关的中断源如下所示：

- 低电压检测（LVD）
- 实时中断（RTI）
- 模数定时器溢出（MTIM）
- 模拟比较器（ACMP）
- 键盘中断（KBI）

首先，中断服务的优先级应该根据应用需求进行定义，需要最短延时的中断请求优先级最高。为了说明中断服务优先级的理念，做如下定义：

Table 1-5. Interrupt Servicing Priority Example

Highest →				Lowest
MTIM	KBI	ACMP	RTI	LVD

在许多中断驱动应用中，中断事件周期对应用来说是不可知的，MCU 的大部分时间是在闲置状态等待事件的触发。一旦中断事件发生，MCU 就会被唤醒执行预先定义好的任务，然后再返回到闲置状态。根据表 1-5 中的定义，中断服务循环可以写成如下形式：

```
InfLoop:        sta     SRS                             ;Bump COP
                wait
Priority1:      brset   SIP1_MTIM, SIP1, MTIM_ISR       ;5 bus cycles
Priority2:      brset   SIP1_ACMP, SIP1, ACMP_ISR       ;5 bus cycles
Priority3:      brset   SIP1_KBI, SIP1, KBI_ISR         ;5 bus cycles
Priority4:      brset   SIP1_RTI, SIP1, RTI_ISR         ;5 bus cycles
Priority5:      brset   SIP1_LVD, SIP1, LVD_ISR         ;5 bus cycles
                bra     InfLoop
MTIM_ISR:
                ;... <ISR coding> ...
```

```
                bra     InfLoop
ACMP_ISR:
                ;... <ISR coding> ...
                bra     InfLoop
KBI_ISR:
                ;... <ISR coding> ...
                bra     InfLoop
RTI_ISR:
                ;... <ISR coding> ...
                bra     InfLoop
LVD_ISR:
                ;... <ISR coding> ...
                bra     InfLoop
```

上面的例子说明了优先级的软件处理机制。例子中，MCU 在应用闲置时进入等待模式。RS08 CPU 从等待模式唤醒一般需要 3 个总线时钟周期，中断延迟时间主要就是软件的执行时间。如果总线频率为 10MHz（总线周期为 100ns），相应的中断延迟时间如表 1-6 所示。用户可以自由的调整软件循环时间，根据应用需求使中断延迟时间最小。

Table 1-6. Interrupt Latency based on 10MHz Bus Clock

Interrupt	Latency (μs)
MTIM	0.8
ACMP	1.3 [1]
KBI	1.8 [1]
RTI	2.3 [1]
LVD	2.8 [1]

注意：
[1] 可能存在额外延迟(以 2 个总线时钟周期为代表),来实现异步中断源与总线时钟的同步。

注意

在上面的例子中，COP 在进入等待模式前刷新一次。为了防止产生 COP 复位，在一个 COP 溢出时间内要求至少一个中断事件发生。

在许多应用中中断周期比较长，这时候让 MCU 进入停止模式以让功耗降到最低是很明智的，特别是用电池供电。由于 RS08 CPU 从停止模式唤醒只能由像 KBI、ACMP 等异步中断源完成，像 MTIM 那样的所有同步中断事件检测可以从中断服务循环中除去。对 MC9RS08KA2 来说，除了 MTIM 所有的中断源都有停止状态唤醒能力（更多详情请参考 MC9RS08KA2 技术手册）。除了软件执行时，从 MCU 停止模式唤醒的中断延时必须有包含 MCU 恢复时间，MCU 恢复时间包括系统时钟和内部调整器从备用模式唤醒所用时间。停止恢复时间由于产品系列不同而异，主要是取决于使用的时钟模块和内部调整器的技术。

2 模拟 ADC 应用例子

在这一部分中，MC9RS08KA2 中模拟比较模块主要用来实现一个 8 位的模数转换器（ADC）。

在许多应用中，并不需要精确的 ADC。通过 MCU 中内建的一个定时器模块和一个低成本高性能的模拟比较器可以模拟实现一个 ADC 的功能，模拟 ADC 的分辨率取决于定时器的分辨率。MC9RS08KA2 中包含一个 8 位的模数定时器（MTIM）时，可以方便的模拟一个 8 位的 ADC 操作。与专用的 ADC 模块相比，主要的差别在于采样时间和动态范围，模拟 ADC 通常是较长的采样时间、较窄的动态范围，而且并不支持轨至轨操作。

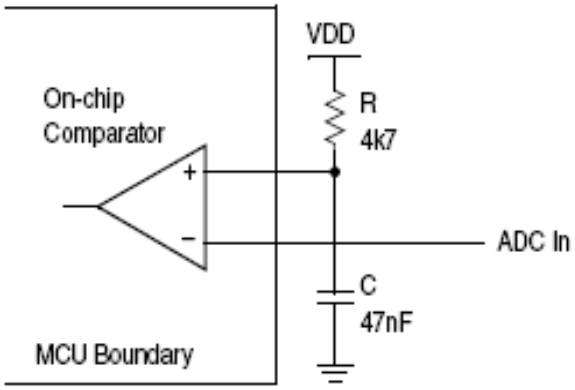


Figure 2-5. Emulated ADC Schematic

图 2-5 所示为一个简单的模拟 ADC 的示意图。比较器的正电压接线端接 RC 网络，负电压接线端作为 ADC 的输入端。在使用比较器的功能之前，两个接线端为通用的输入/输出口。比较器的正电压接线端初始化设置为输出低电平来使 RC 放电。当需要 ADC 功能时，就让比较器使能，通过 ADC 输入和电容器 C 上的电压的比较来模拟实现 ADC 功能。定时器用来监视 RC 电路给 ADC 输入电压充电的时间。由于 RC 电路的充电的过程不是线性的，如果 ADC 的动态范围很小，定时器读数可以认为是线性的。通常通过一个简单的查表来转换定时器读数为一个线性的数值会更加方便。

2.1 操作

下面的介绍使用 MC9RS08KA2 来完成模拟 ADC 功能。完整的程序在附录 A 中。

1. **定义采样时间和定时器分辨率。**采样样时间就是 RC 充电到 ADC 最大电压（动态范围内）的时间，在这个例子中，随意选择 1ms 为采样时间。当使用 8 位定时器（n=8）时，定时器分辨率是 3.9 微秒（数学计算式为式 1），取近似值 4 微秒。最大的定时器溢出周期为 255 乘以 4 微秒，也就是 1.02ms。

$$TimerResolution = \frac{ChargeUpTime}{2^n - 1} \qquad \textbf{式 1}$$

2. 定义 RC 常数。RC 电路充电电压-时间关系为数学计算式 2。

$$V = V_{DD}\left(1 - e^{-\frac{t}{RC}}\right) \qquad \textit{式 2}$$

时间 *t* 为 RC 常数的 4.6 倍时，电容器充电达到 99%。为了达到最大的测量范围，定时器溢出周期希望不小于这个值。在这个例子中，在 1.02ms 的定时器溢出周期内，RC 常数为 2.21E-4。

$$RC = \frac{TimerOverflowPeriod}{4.61} \qquad \textit{式 3}$$

电阻器 R 的阻值是根据该管脚的吸收电流能力定义的。参考 MC9RS08KA2 的数据手册可以知道，该管脚的电流吸收能力为 1mA 左右，所以初始化的充电电压可以接近 0V。如果 V_{DD} 是 5V，那么就需要选择一个 4700Ω 的电阻。根据给出的 2.21E-4 的时间常数，电容器的容量为 47nF。注意到该管脚的吸收电流增加整个系统的 I_{DD} 消耗；如果 ADC 功能未启用，或者在 MCU 进入停止模式之前，将该管脚设为输入或高阻以避免电流流失。

3. 构建查询表。以给出点定时器分辨率 4 微秒为例，可以构造一个查询表来补偿基于等式 2 的充电关系非线性。对一个线性的 8 位 ADC，给出如下的步长数学计算式：

$$Step = \frac{V_{DD}}{255} \qquad \textit{式 4}$$

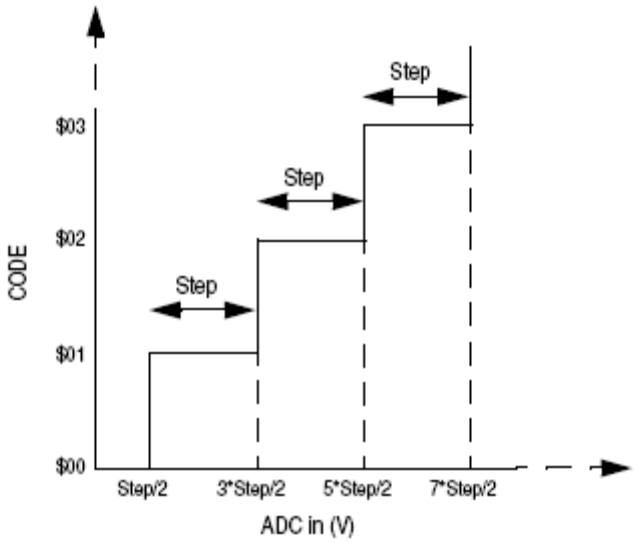


Figure 2-6. ADC Quantization Diagram

一个线性的 ADC 希望在步边界如图 2-6 所示的输入电压的步/2 开始量化输入电压，转换函数为：

$$Code = \begin{cases} \left(\frac{ADCin - \frac{Step}{2}}{Step}\right) + 1 & ;\left(ADCin \geq \frac{Step}{2}\right) \\ 0 & ;\left(ADCin < \frac{Step}{2}\right) \end{cases} \qquad \textit{式 5}$$

定时器计数到线性 ADC 代码转换的查询表如表 2-7 所示。

Table 2-7. Non-Linearity Compensation Lookup Table

Time (μs)	ADC Input (V) (Equation 2)	Timer Count	Linear ADC Code (Equation 5)
0	0	0	0
4	0.09	1	5
8	0.18	2	10
12	0.26	3	14
16	0.35	4	18
20	0.43	5	23
and so on...			
1012	4.95	253	253
1016	4.95	254	253
1020	4.95	255	253

4. **定义总线频率**。在开始测量之前，由软件使能定时器和比较器。为了避免软件延迟错误，推荐选择一个至少 5 倍于定时器时钟频率的总线频率。在本例中，初始化时选择一个 2MHz 的总线频率，并且定时器分频数设置为 8 分频，这样就给定时器一个 250kHz 的时钟频率，也就是 4 微秒的分辨率。应用中时钟频率的选择不是随意的，可以借助查询表来补偿软件造成的时间延迟。

5. **RS08 代码**。软件代码可以被分为 4 部分： 声明部分、初始化部分、ADC 读数部分和表查询部分。

   a) 开始声明所需要的变量和查询表的位置。使用频率最高的变量应该放在低端可寻址 RAM 区域内，也就是在$0000 到$000D 范围内，这样单字节的微短指令可以用来进行数据处理。查询表应该放在高端存内存中，在本例中，存放地址$3E00 是随意选择的，所有的高端内存访问都是通过放在第一页 64 字节的分页窗口完成的。

```
;*******************************************************************
; Application Definition
;*******************************************************************
RC              equ     PTAD_PTAD0
mRC             equ     mPTAD_PTAD0
TableStart      equ     $3E00
        org     Tiny_RAMStart
; variable/data section
SensorReading   ds.b 1
ADCOut          ds.b 1
```

b) 比较器正电压接线端必须初始化为输出低，这样 RC 电路才可以启动时为完全的放电状态。原程序代码如下：

```
;------------------------------------------------------
; Init RAM
;------------------------------------------------------
        clr     SensorReading                           ; Single byte instruction
        clr     ADCOut                                  ; Single byte instruction
;------------------------------------------------------
; Config GPIO
; RC - init L
;------------------------------------------------------
        mov     #(mDATAOUT), PTAD                       ; RC Initial low
        mov     #(mRC|mDATAOUT), PTADD                  ; Set Output pins
```

c) 在 ADCRead 子程序，在使能比较器前进行定时器初始化并且开始运行，一旦比较器使能后，它的两个端口都作为模拟输入，RC 电路开始充电。这时 MCU 进入等待模式，等待中断事件的触发。由于定时器（MTIM）溢出中断和比较器中断都可以将 MCU 从等待模式唤醒，所以这两个中断都是使能的。当一个中断触发时，程序继续执行下一条指令，定时器计数器的值立即被读出并且保存到 SensorReading 变量中，接着就检测比较器的标志位。如果这个标志位为零，这说明没有比较器事件发生，ADC 输入可能超出范围，那样的话保存在 SensorReading 中的值被废弃；否则关闭比较器，正电压接线端返回到输出为低并且使 RC 电路放电。

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
; Read Sensor (ADC) Value
; Timer prescalar=8 -> Timer clk~250kHz
; Bus = 2MHz
; Max OF period = 1.02ms
; Timer resolution = 4us
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ADCRead:
        mov     #(MTIM_BUS_CLK|MTIM_DIV_8), MTIM1CLK        ;Change Timer resolution
        mov     #255, MTIM1MOD                              ;OF period
        mov     #(mMTIM1SC_TRST|mMTIM1SC_TOIE), MTIM1SC     ;Reset and Start Timer
        mov     #(mACMP1SC_ACME|mACMP1SC_ACIE|ACMP_OUTPUT_RAISING), ACMP1SC
                                                            ; Enable ACMP, start RC rise
        bset    ACMP1SC_ACF, ACMP1SC                        ;Clear ACMP Flag
        wait
        mov     MTIM1CNT, SensorReading
        brclr   ACMP1SC_ACF, ACMP1SC, NoReading
```

```
        bset    ACMP1SC_ACF, ACMP1SC                        ;Clear ACMP Flag
        clr     ACMP1SC                                     ;disable ACMP
        mov     #(mMTIM1SC_TSTP|mMTIM1SC_TRST), MTIM1SC     ;mask int and clear flag
        rts
NoReading:
        mov     #$FF, SensorReading                         ;Biggest Number
        clr     ACMP1SC                                     ;disable ACMP
        mov     #(mMTIM1SC_TSTP|mMTIM1SC_TRST), MTIM1SC     ;mask int and clear flag
        rts
```

d) 在表查询子程序，变量 SensorReading 的高两位（MSB）被扩展且加到保存查询表的页码上。相应的查询表的内容放在 64 字节的页面窗口中，地址范围从$00C0 到$00FF。接着变量 SensorReading 的低 6 位（LSB）用作从分页窗口直接读高地址存储器内容的一个索引。

```
;*************************************************************************************
; 8bit Table Lookup
;*************************************************************************************
TableLookup:
        lda     SensorReading                               ;
        rola                                                ;Extract 2 MSB
        rola                                                ;
        rola                                                ;
        and     #$03                                        ;Mask all other bits
        add     #(TableStart>>6)                            ;Add to Lookup table page
        sta     PAGESEL                                     ;High page
        lda     SensorReading                               ;
        and     #$3F                                        ;Extract 6 LSB
        add     #$c0                                        ;Index to paging window
        tax                                                 ;
        lda     ,x                                          ;Read upper memory
        sta     ADCOut                                      ;Store lookup table content
        mov     #(HREG), PAGESEL                            ;Return to register page
        rts                                                 ;

;*************************************************************************************
; ADC Lookup Table - RC charging profile
;*************************************************************************************
        org  TableStart
        dc.b    0,  5, 10, 14, 18, 23, 27, 31, 35, 39, 43, 47, 50, 54, 58, 61
        dc.b   65, 68, 71, 75, 78, 81, 84, 87, 90, 93, 96, 99,102,105,107,110
        dc.b  113,115,118,120,123,125,127,130,132,134,136,138,141,143,145,147
        dc.b  149,150,152,154,156,158,160,161,163,165,166,168,169,171,173,174
        dc.b  175,177,178,180,181,182,184,185,186,188,189,190,191,192,193,195
        dc.b  196,197,198,199,200,201,202,203,204,205,206,206,207,208,209,210
        dc.b  211,211,212,213,214,215,215,216,217,217,218,219,219,220,221,221
        dc.b  222,223,223,224,224,225,225,226,226,227,228,228,228,229,229,230
        dc.b  230,231,231,232,232,233,233,233,234,234,235,235,235,236,236,236
        dc.b  237,237,237,238,238,238,239,239,239,240,240,240,240,241,241,241
        dc.b  241,242,242,242,242,243,243,243,243,244,244,244,244,244,245,245
        dc.b  245,245,245,246,246,246,246,246,246,247,247,247,247,247,247,247
        dc.b  248,248,248,248,248,248,248,249,249,249,249,249,249,249,249,249
        dc.b  250,250,250,250,250,250,250,250,250,250,251,251,251,251,251,251
        dc.b  251,251,251,251,251,251,252,252,252,252,252,252,252,252,252,252
        dc.b  252,252,252,252,252,252,253,253,253,253,253,253,253,253,253,253
```

2.2 校准

模拟 ADC 的性能很大程度上依赖于 RC 电路时间常数的精确性，如果实际 RC 元件的值偏离标称值，那么 RC 充电曲线可能会偏移并且定时器捕捉不准确。实际上，PCB 电路板的寄生电容也会导致 RC 时间常数的错误。执行简单的校准可以补偿由 RC 常数变化引起的误差。

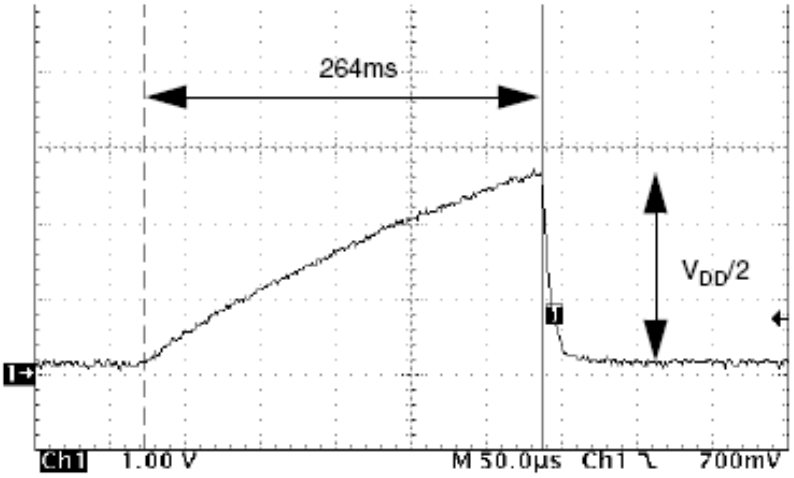


Figure 2-7. 2.5V Input R-C Charging Profile

为了测量真实的 RC 常数，必须记录充电曲线，可以通过在 ADC 输入端加一个 $V_{DD}/2$ 电压，那么充电曲线就如图 2-7 记录的那样。在这个例子中 RC 电路中达到 $V_{DD}/2$ 电压水平所用的时间为 264 微秒。根据表 2-7 的计算结果，期望的上升时间是 152 微秒，也就是 38 个定时器计数周期。这儿有以下几种方法进行校准：

- 从式 2 可以推算出真实的 RC 常数和重建查询表。.
- 使用可变电阻/电容代替固定的 R 和 C。通过调整 R 和 C 的值直到上升时间降到期望值（在这个例子中为 152 微秒）。
- 可以通过调整定时器精度进行补偿。MC9RS08KA2 和许多飞思卡尔的 MCU 都有一个软件可编程的时钟源（ICS），总线频率可以通过简单的重新编程 TRIM 寄存器来调整。在这个例子中，定时器的计数期望达到 $V_{DD}/2$ 电压水平，所以，根据测得的 264 微秒上升时间，新的定时器分辨率应该是 264 除以 38，也就是 6.94 微秒。根据从定时器时钟源所选的 8 分频的选项，补偿总线周期应该是 6.94 微秒除以 8，是 868ns，也就是 1.15MHz 的总线频率。因此，如果使用 1.15MHz 的总线频率，就既不需要硬件调整也不需要修正查询表。对 MC9RS08KA2 来说，总线频率可以通过重编程的 TRIM 寄存器和 ICSC2 寄存器的除数位来调整。（更多信息请参考 MC9RS08KA2 数据手册）。

2.3 测量结果

当 ADCRead 子程序执行时，RC 电路开始充电过程。一旦 ADC 输入电压达到 RC 电压，定时器计数器的值被读出并且比较器停止工作，RC 电路转入放电状态。图 2-8 所示为在不同的 ADC 输入电压时的充电和放电过程。

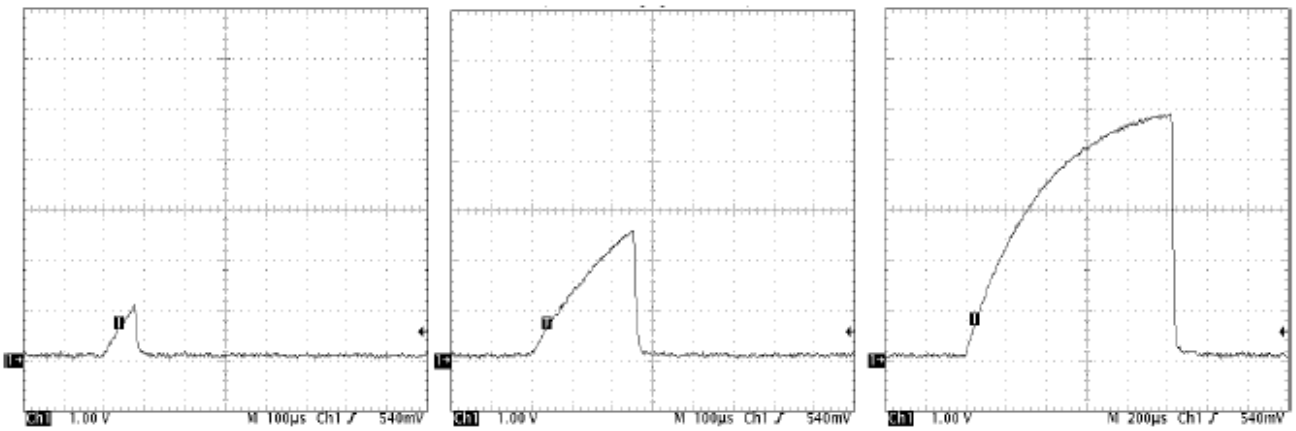


Figure 2-8. RC Charging Profile Against Different ADC Input Voltages

在总线频率调整为 1.15MHz 后，在 V_{DD}=5V 下模拟 ADC 的的性能表 2-8 和图 2-9 所示。

Table 2-8. Emulated ADC Performance

ADC Input Voltage (V)	Expected ADC code (Decimal)	Measured ADC code (Decimal)
1	51	50
1.5	76	75
2	102	99
2.5	127	123
3	153	150
3.5	178	175
4	204	202
4.5	229	234

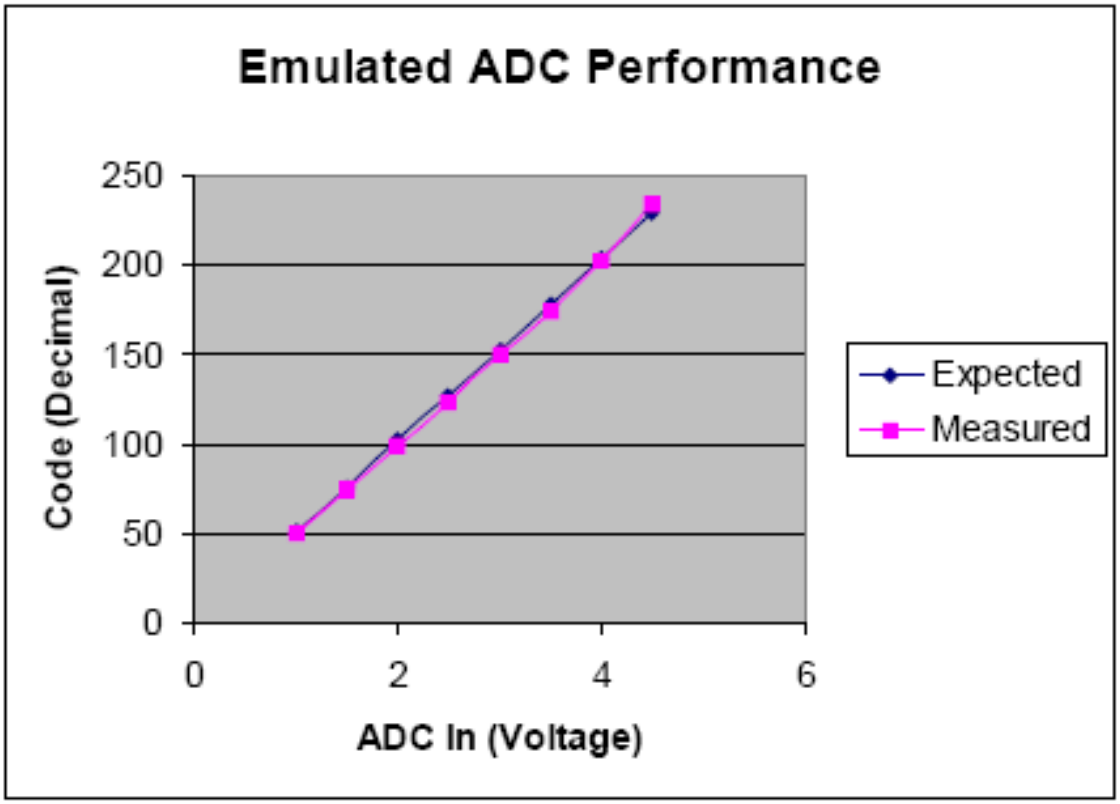


Figure 2-9. Emulated ADC Performance

附录 A 程序实例

```
;*****************************************************************
;
;  (c) copyright Freescale Semiconductor, Inc. 2006.
;  ALL RIGHTS RESERVED
;
;*****************************************************************
;*****************************************************************
;* Emulated ADC Coding for MC9RS08KA2
;*
;* Author:       Vincent Ko
;* Date:         Jan 2006
;*
;* PTA0/KBI0/ACMP+                          RC network
;* PTA1/KBI1/ACMP-                          ADCIN
;* PTA5/KBI5                                DATAOUT
;*
;*****************************************************************
; include derivative specific macros
         XDEF    Entry

         include "MC9RS08KA2.inc"
;===============================================================================
; ICS Definition
;===============================================================================
ICS_DIV_1                   equ      $00
ICS_DIV_2                   equ      $40
ICS_DIV_4                   equ      $80
ICS_DIV_8                   equ      $c0
;===============================================================================
; MTIM Definition
;===============================================================================
MTIM_DIV_1                  equ      $00
MTIM_DIV_2                  equ      $01
MTIM_DIV_4                  equ      $02
MTIM_DIV_8                  equ      $03
MTIM_DIV_16                 equ      $04
MTIM_DIV_32                 equ      $05
MTIM_DIV_64                 equ      $06
MTIM_DIV_128                equ      $07
MTIM_DIV_256                equ      $08
MTIM_BUS_CLK                equ      $00
MTIM_XCLK                   equ      $10
MTIM_TCLK_FALLING           equ      $20
MTIM_TCLK_RISING            equ      $30
;===============================================================================
; ACMP Definition
;===============================================================================
ACMP_OUTPUT_FALLING                 equ      $00
ACMP_OUTPUT_RAISING                 equ      $01
ACMP_OUTPUT_BOTH                    equ      $03
;===============================================================================
; RTI Definition
;===============================================================================
RTI_DISABLE                 equ      $00
RTI_8MS                     equ      $01
```

```
RTI_32MS                    equ     $02
RTI_64MS                    equ     $03
RTI_128MS                   equ     $04
RTI_256MS                   equ     $05
RTI_512MS                   equ     $06
RTI_1024MS                  equ     $07

;===============================================================================
; Application Definition
;===============================================================================
RC                          equ     PTAD_PTAD0
mRC                         equ     mPTAD_PTAD0
DATAOUT                     equ     PTAD_PTAD5
mDATAOUT                    equ     mPTAD_PTAD5

TableStart                  equ     $3E00

        org     Tiny_RAMStart
; variable/data section
SensorReading               ds.b 1
ADCOut                      ds.b 1
BitCount                    ds.b 1

        org     Z_RAMStart
; variable/data section

        org     ROMStart
; code section
main:
Entry:
;-----------------------------------------------------
; Config ICS
; Device is pre-trim to 18.4MHz ICLK frequency
; TRIM value are stored in $3FFA:$3FFB
;-----------------------------------------------------
        mov   #$FF, PAGESEL
        mov   $FB,  ICSSC                                   ; $3FFB
        mov   $FA,  ICSTRIM                                 ; $3FFA
        mov   #ICS_DIV_8, ICSC2                             ; Use 1.15MHz bus
;-----------------------------------------------------
;Config System
;-----------------------------------------------------
        mov     #HREG, PAGESEL                              ; Init Page register
        mov     #(mSOPT_COPE|mSOPT_COPT|mSOPT_STOPE), SOPT  ; SOPT, COP enabled
        mov     #(mSPMSC1_LVDE|mSPMSC1_LVDRE), SPMSC1       ; LVI enable
        mov     #(RTI_128MS|mSRTISC_RTIE), SRTISC           ; 128ms RTI
;-----------------------------------------------------
; Init RAM
;-----------------------------------------------------
        clr     SensorReading                               ; Single byte instruction
        clr     ADCOut                                      ; Single byte instruction
;-----------------------------------------------------
; Config GPIO
; RC - init L
;-----------------------------------------------------
        mov     #(mDATAOUT), PTAD                           ; RC Initial low
        mov     #(mRC|mDATAOUT), PTADD                      ; Set Output pins
```

```
;-------------------------------------------------------
; Application Loop
; 1) Wakeup every 128ms
; 2) Read ADC input
; 3) Dump code to serially output port (DATAOUT)
;-------------------------------------------------------
InfLoop:
        wait
        bset    SRTISC_RTIACK, SRTISC
        bsr     ReadSensor                              ; Read Charge up time data
        bsr     TableLookup                             ; Decode 8bit level
        bsr     DataDump                                ; Dump ADC code
        sta     SRS                                     ; Bump COP
        bra     InfLoop

;*************************************************************************************
; Read Sensor (ADC) Value
; Timer prescalar=8 -> Timer clk~250kHz
; Bus = 2MHz
; Max OF period = 1.02ms
; Timer resolution = 4us
;*************************************************************************************
ADCRead:
        mov     #(MTIM_BUS_CLK|MTIM_DIV_8), MTIM1CLK        ;Change Timer resolution
        mov     #255, MTIM1MOD                              ;OF period
        mov     #(mMTIM1SC_TRST|mMTIM1SC_TOIE), MTIM1SC     ;Reset and Start Timer
        mov     #(mACMP1SC_ACME|mACMP1SC_ACIE|ACMP_OUTPUT_RAISING), ACMP1SC
                                                            ; Enable ACMP, start RC rise
        bset    ACMP1SC_ACF, ACMP1SC                        ;Clear ACMP Flag
        wait
        mov     MTIM1CNT, SensorReading
        brclr   ACMP1SC_ACF, ACMP1SC, NoReading
        bset    ACMP1SC_ACF, ACMP1SC                        ;Clear ACMP Flag
        clr     ACMP1SC                                     ;disable ACMP
        mov     #(mMTIM1SC_TSTP|mMTIM1SC_TRST), MTIM1SC     ;mask int and clear flag
        rts
NoReading:
        mov     #$FF, SensorReading                         ;Biggest Number
        clr     ACMP1SC                                     ;disable ACMP
        mov     #(mMTIM1SC_TSTP|mMTIM1SC_TRST), MTIM1SC     ;mask int and clear flag
        rts
;*************************************************************************************
; 8bit Table Lookup
;*************************************************************************************
TableLookup:
        lda     SensorReading                               ;
        rola                                                ;Extract 2 MSB
        rola                                                ;
        rola                                                ;
        and     #$03                                        ;Mask all other bits
        add     #(TableStart>>6)                            ;Add to Lookup table page
        sta     PAGESEL                                     ;High page
        lda     SensorReading                               ;
        and     #$3F                                        ;Extract 6 LSB
        add     #$c0                                        ;Index to paging window
        tax                                                 ;
        lda     ,x                                          ;Read upper memory
```

```
        sta     ADCOut                                  ;Store lookup table content
        mov     #(HREG), PAGESEL                        ;Return to register page
        rts                                             ;
;*******************************************************************************
; Serial Data dump
; <LOW><b7><b6><b5><b4><b3><b2><b1><b0><HIGH>
;*******************************************************************************
DataDump:
        mov     #8,     BitCount
        lda     ADCOut

        bclr    DATAOUT, PTAD                           ;5 Start bit
        bclr    DATAOUT, PTAD                           ;5 dummy
        cmp     0                                       ;3 dummy
        nop                                             ;1 dummy
NextBit:
        lsla                                            ;1
        bcc     ClrPort                                 ;3
        bset    DATAOUT, PTAD                           ;5
        bra     BitEnd                                  ;3
ClrPort:
        bclr    DATAOUT, PTAD                           ;5
        bra     BitEnd                                  ;3
BitEnd:
        dbnz    BitCount, NextBit                       ;6
ByteEnd:
        bset    DATAOUT, PTAD                           ;5 End bit
        rts

;*******************************************************************************
; ADC Lookup Table - RC charging profile
;*******************************************************************************
        org     TableStart
        dc.b    0,  5, 10, 14, 18, 23, 27, 31, 35, 39, 43, 47, 50, 54, 58, 61
        dc.b   65, 68, 71, 75, 78, 81, 84, 87, 90, 93, 96, 99,102,105,107,110
        dc.b  113,115,118,120,123,125,127,130,132,134,136,138,141,143,145,147
        dc.b  149,150,152,154,156,158,160,161,163,165,166,168,169,171,173,174
        dc.b  175,177,178,180,181,182,184,185,186,188,189,190,191,192,193,195
        dc.b  196,197,198,199,200,201,202,203,204,205,206,206,207,208,209,210
        dc.b  211,211,212,213,214,215,215,216,217,217,218,219,219,220,221,221
        dc.b  222,223,223,224,224,225,225,226,226,227,228,228,228,229,229,230
        dc.b  230,231,231,232,232,233,233,233,234,234,235,235,235,236,236,236
        dc.b  237,237,237,238,238,238,239,239,239,240,240,240,240,241,241,241
        dc.b  241,242,242,242,242,243,243,243,243,244,244,244,244,244,245,245
        dc.b  245,245,245,246,246,246,246,246,246,247,247,247,247,247,247,247
        dc.b  248,248,248,248,248,248,248,249,249,249,249,249,249,249,249,249
        dc.b  250,250,250,250,250,250,250,250,250,250,251,251,251,251,251,251
        dc.b  251,251,251,251,251,251,252,252,252,252,252,252,252,252,252,252
        dc.b  252,252,252,252,252,252,253,253,253,253,253,253,253,253,253,253
;*******************************************************************************
; Reset Vector
;*******************************************************************************
        org     $3ffc
Security:
        dc.b    $FF
        jmp     main
```

本页有意留空

本页有意留空

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale SemiconductorTechnical
Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax:303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.
For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

freescale™
飞思卡尔半导体

文件编号：AN3266
第 1 版
5/2006

General Business Information