



SYSTECH J.Schnyder GmbH

Schliefweg 30
CH-4106 Therwil
Telefon 091 827 15 87
www.systech-gmbh.ch

how to generate TIC's (and how better not)

V 0.4

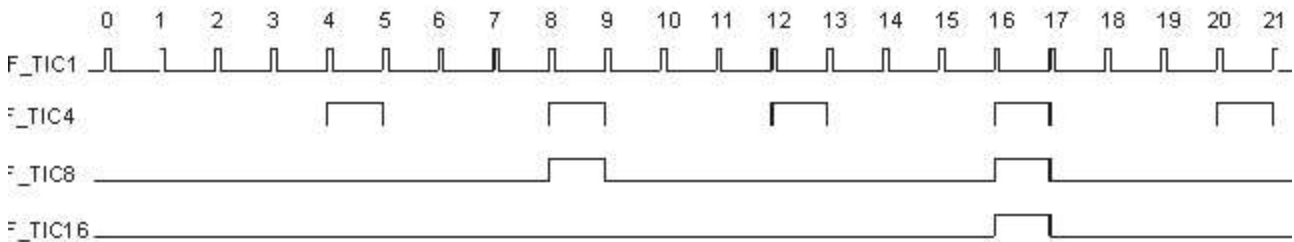
Content

Preface	2
How to calculate the TIC's	3
The Table	3
How it works	6
The Bit Table	6
Code Examples	8
<i>fast solution</i>	8
<i>slow solution</i>	9
<i>compact solution</i>	10
Links	11

Preface

Since the EBS08 ("Einfaches Betriebs-System für HC(S)08") generated the TIC's in a rather simple (as the name "einfach" in German intends) manner. So there were two problems: first all the flags have been set in the same time slice, so there different tasks had to share the time between the two TIC's. The diagram below shows how it worked. The second disadvantage is that the time for calculating the flags was differing from case to case, so that the start of the Task Manager had a jitter of about 0 to 28 CPU cycles (i.e. 0 to 22,8us for a 4,9152MHz clock).

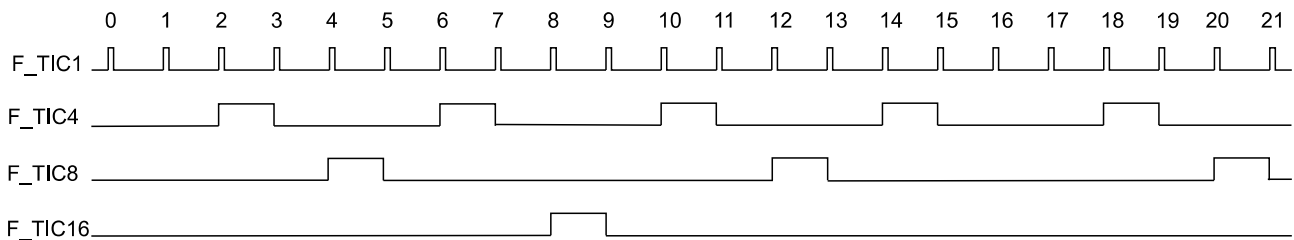
Old Timing



We can see that for F_TIC1 number 8 -> F_TIC4 and F_TIC8 are set at the same time, and at F_TIC1 number 16 -> F_TIC4 and F_TIC8 and F_TIC16 are set simultaneously.

Thanks to BIGMAC (freescall forums) I noticed the problems and changed the timing according to the following diagram.

New Timing



The second disadvantage can either be resolved by a solution with a 128 byte table, called *fast solution* or can remain with two other solutions called *slow solution* or *compact solution*.

The **fast solution** uses, as mentioned above, a 128 Byte table and the code is arranged to remove the jitter. This solution is fast, but needs probably too much ROM space.

The **slow solution** uses a 16 bit table and therefore the ROM usage is much less than the *fast solution*.

The **compact solution** is optimized for ROM usage, since this solution does not use any table, it is really short, but the disadvantage is the different execution time, which is caused by the loops used there. The trick in this solution is to count the zeros on the right side of the core counter to calculate the TIC number.

The user can choose the solution suitable for his application, but by default the compact solution will

be used.

How to calculate the TIC's

With the following formula you can calculate the number of the Z_CORE for each F_TICx

$$z = n(2^w) + (2^{(w-1)} - 2) + x$$

where

z represents the number of the Z_CORE (value 0 to 255, since it is a 8bit counter)

n is the number of the TICx beginning with 0

w is the index of the flag i.e. for F_TIC4 w is 2 (2²=4)

x is the offset

The Table

Offset x = 2

TIC number	Flag	TIC number	Flag	TIC number	Flag	TIC number	Flag
0		64	F_TIC128	128	F_TIC256	192	F_TIC128
1		65		129		193	
2	F_TIC4	66	F_TIC4	130	F_TIC4	194	F_TIC4
3		67		131		195	
4	F_TIC8	68	F_TIC8	132	F_TIC8	196	F_TIC8
5		69		133		197	
6	F_TIC4	70	F_TIC4	134	F_TIC4	198	F_TIC4
7		71		135		199	
8	F_TIC16	72	F_TIC16	136		200	F_TIC16
9		73		137		201	
10	F_TIC4	74	F_TIC4	138	F_TIC4	202	F_TIC4
11		75		139		203	
12	F_TIC8	76	F_TIC8	140	F_TIC8	204	F_TIC8
13		77		141		205	
14	F_TIC4	78	F_TIC4	142	F_TIC4	206	F_TIC4
15		79		143		207	
16	F_TIC32	80	F_TIC32	144	F_TIC32	208	F_TIC32
17		81		145		209	
18	F_TIC4	82	F_TIC4	146	F_TIC4	210	F_TIC4

19		83		147		211	
20	F_TIC8	84	F_TIC8	148	F_TIC8	212	F_TIC8
21		85		149		213	
22	F_TIC4	86	F_TIC4	150	F_TIC4	214	F_TIC4
23		87		151		215	
24	F_TIC16	88	F_TIC16	152	F_TIC16	216	F_TIC16
25		89		153		217	
26	F_TIC4	90	F_TIC4	154	F_TIC4	218	F_TIC4
27		91		155		219	
28	F_TIC8	92	F_TIC8	156	F_TIC8	220	F_TIC8
29		93		157		221	
30	F_TIC4	94	F_TIC4	158	F_TIC4	222	F_TIC4
31		95		159		223	
32	F_TIC64	96	F_TIC64	160	F_TIC64	224	F_TIC64
33		97		161		225	
34	F_TIC4	98	F_TIC4	162	F_TIC4	226	F_TIC4
35		99		163		227	
36	F_TIC8	100	F_TIC8	164	F_TIC8	228	F_TIC8
37		101		165		229	
38	F_TIC4	102	F_TIC4	166	F_TIC4	230	F_TIC4
39		103		167		231	
40	F_TIC16	104	F_TIC16	168	F_TIC16	232	F_TIC16
41		105		169		233	
42	F_TIC4	106	F_TIC4	170	F_TIC4	234	F_TIC4
43		107		171		235	
44	F_TIC8	108	F_TIC8	172	F_TIC8	236	F_TIC8
45		109		173		237	
46	F_TIC4	110	F_TIC4	174	F_TIC4	238	F_TIC4
47		111		175		239	
48	F_TIC32	112	F_TIC32	176	F_TIC32	240	F_TIC32
49		113		177		241	
50	F_TIC4	114	F_TIC4	178	F_TIC4	242	F_TIC4
51		115		179		243	
52	F_TIC8	116	F_TIC8	180	F_TIC8	244	F_TIC8
53		117		181		245	
54	F_TIC4	118	F_TIC4	182	F_TIC4	246	F_TIC4
55		119		183		447	

56	F_TIC16	120	F_TIC16	184	F_TIC16	248	F_TIC16
57		121		185		249	
58	F_TIC4	122	F_TIC4	186	F_TIC4	250	F_TIC4
59		123		187		251	
60	F_TIC8	124	F_TIC8	188	F_TIC8	252	F_TIC8
61		125		189		253	
62	F_TIC4	126	F_TIC4	190	F_TIC4	254	F_TIC4
63		127		191		255	

As we can see in the table, every odd number has no flag set (apart from F_TIC1), so we could introduce a interleaved set of flags (as BIGMAC proposed) for use if the tasks are requiring more time.

So for compatibility reasons I have chosen two sets of flags to indicate the time slices

This are:

CORE_F0 for the **phase0**

and

CORE_F1 for the **phase1**.

Each register has the following flags:

F_TIC4
F_TIC8
F_TIC16
F_TIC32
F_TIC64
F_TIC128
F_TIC256

F_TIC1 is present in CORE_F0 and CORE_F1.

F_TIC1 in CORE_F0 is used for controlling the Task Manager.

How it works

The Bit Table

Flag	Z-CORE	Bits								
		7	6	5	4	3	2	1	0	
		128	64	32	16	8	4	2	1	Lo nibble value after a AND #\$0E (bit0 is always zero)
F_TIC4	2	0	0	0	0	0	0	1	X	2
	6	0	0	0	0	0	1	1	X	6
	10	0	0	0	0	1	0	1	X	10
	14	0	0	0	0	1	1	1	X	14
	18	0	0	0	1	0	0	1	X	2
	22	0	0	0	1	0	1	1	X	6
	26	0	0	0	1	1	0	1	X	10
	30	0	0	0	1	1	1	1	X	14
									
F_TIC8	4	0	0	0	0	0	1	0	X	4
	12	0	0	0	0	1	1	0	X	12
	20	0	0	0	1	0	1	0	X	4
	28	0	0	0	1	1	1	0	X	12
	36	0	0	1	0	0	1	0	X	4
									
F_TIC16	8	0	0	0	0	1	0	0	X	8
	24	0	0	0	1	1	0	0	X	8
	40	0	0	1	0	1	0	0	X	8
	56	0	0	1	1	1	0	0	X	8
	72	0	1	0	0	1	0	0	X	8
	88	0	1	0	1	1	0	0	X	8
									
										Hi nibble value after swapping the nibbles and AND #\$0F
F_TIC32	16	0	0	0	1	0	0	0	X	1

	48	0	0	1	1	0	0	0	X	3
	80	0	1	0	1	0	0	0	X	5
	112	0	1	1	1	0	0	0	X	7
	144	1	0	0	1	0	0	0	X	9
	176	1	0	1	1	0	0	0	X	11
	208	1	1	0	1	0	0	0	X	13
	240	1	1	1	1	0	0	0	X	15
F_TIC64	32	0	0	1	0	0	0	0	X	2
	96	0	1	1	0	0	0	0	X	6
	160	1	0	1	0	0	0	0	X	10
	224	1	1	1	0	0	0	0	X	14
F_TIC128	64	0	1	0	0	0	0	0	X	4
	192	1	1	0	0	0	0	0	X	12
F_TIC256	128	1	0	0	0	0	0	0	X	8

IF the Flags are arranged like the following:

F_TIC256	F_TIC128	F_TIC64	F_TIC32	F_TIC16	F_TIC8	F_TIC4	F_TIC1
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

for the *slow solution*:

We can see that:

F_TIC4 and F_TIC64,
 F_TIC8 and F_TIC128
 F_TIC16 and F_TIC256

are on the same position in the nibble with the same nibble value.

If the low nibble value (after AND # $\$0E$) is zero, then the flag is in the high nibble or no flag is to be set. This means we can use one table:

```
TIC_TBL:                                ;Nr. definiton of the flags
      DC.B  $00,$01,$02,$01            ;0..3
      DC.B  $04,$01,$02,$01            ;4..7
      DC.B  $08,$01,$02,$01            ;8..11
      DC.B  $04,$01,$02,$01            ;12..15
TIC_TBLE:
```

Where: \$00 means no flag set for the nibble
 \$01 F_TIC32 is active (never possible in the low nibble)

\$02 F_TIC4 resp. F_TIC64 is active
 \$04 F_TIC8 resp. F_TIC128 is active
 \$08 F_TIC16 resp. F_TIC256 is active

So the following code takes the actions:

Code Examples

fast solution

```
(this code is part of the task manager [tsk_f.mmw])

TSK_FLAGS:                                ;calculating the flags
    CLR     CORE_F0                        ;clear flags
    CLR     CORE_F1
    LDHX    #0                             ;clear HX
    LDA     Z_CORE
    LSRA
    TAX
    BCC     TSK_FLG01                     ;phase 0
TSK_FLG10:
    LDA     TIC_TBL,X                      ;load the flags
    STA     CORE_F1                        ;store flags for phase1
TSK_FLG11:
    BRA     TSK_FLAGE

TSK_FLG01:
    LDA     TIC_TBL,X                      ;load the flags
    STA     CORE_F0                        ;store flags for phase0
TSK_FLAGE:
    ;end of calculating the TICs

TIC_TBL:                                   ;Nr.  definiton of the flags
    DC.B   $00,$02,$04,$02                ;0
    DC.B   $08,$02,$04,$02                ;      $02 = F_TIC4
    DC.B   $10,$02,$04,$02                ;8      $04 = F_TIC8
    DC.B   $08,$02,$04,$02                ;      $08 = F_TIC16
    DC.B   $20,$02,$04,$02                ;16     $10 = F_TIC32
    DC.B   $08,$02,$04,$02                ;      $20 = F_TIC64
    DC.B   $10,$02,$04,$02                ;24     $40 = F_TIC128
    DC.B   $08,$02,$04,$02                ;      $80 = F_TIC256
    DC.B   $40,$02,$04,$02                ;32
    DC.B   $08,$02,$04,$02                ;
    DC.B   $10,$02,$04,$02                ;40
    DC.B   $08,$02,$04,$02                ;
    DC.B   $20,$02,$04,$02                ;48
    DC.B   $08,$02,$04,$02                ;
    DC.B   $10,$02,$04,$02                ;56
    DC.B   $08,$02,$04,$02                ;
    DC.B   $80,$02,$04,$02                ;64
    DC.B   $08,$02,$04,$02                ;
    DC.B   $10,$02,$04,$02                ;72
    DC.B   $08,$02,$04,$02                ;
    DC.B   $20,$02,$04,$02                ;80
    DC.B   $08,$02,$04,$02                ;
    DC.B   $10,$02,$04,$02                ;88
    DC.B   $08,$02,$04,$02                ;
    DC.B   $40,$02,$04,$02                ;96
```

```

DC.B $08,$02,$04,$02 ;
DC.B $10,$02,$04,$02 ;104
DC.B $08,$02,$04,$02 ;
DC.B $20,$02,$04,$02 ;112
DC.B $08,$02,$04,$02 ;
DC.B $10,$02,$04,$02 ;120
DC.B $08,$02,$04,$02 ;
TIC_TBLE:

```

slow solution

(this code is part of the task manager [tsk_s.mmw])

```

TSK_FLAGS: ;calculating the flags
CLRH
LDA Z_CORE
PSHA ;for high nibble
AND #$0E ;Mask low nybble
BEQ TSK_FLG1 ;If result is 0 -> high nibble
TAX
PULA ;Adjust stack pointer
LDA TIC_TBL,X
BRA TSK_FLG2 ;Branch always

TSK_FLG1: ;Active flag is in high nybble
PULA
NSA
AND #$0F ;Mask nibble value
TAX
LDA TIC_TBL,X
NSA ;Move active bit to high nybble

TSK_FLG2: ;Phase select
BRSET 0,Z_CORE,TSK_FLG3
STA CORE_F0 ;Phase 0 value
CLR CORE_F1 ;clear F_TIC1
BRA TSK_FLAGE

TSK_FLG3: ;Phase 1
STA CORE_F1 ;Phase 1 value
CLR CORE_F0 ;clear F_TIC1

TSK_FLAGE: ;end of calculation the TICs

.....
.....

TIC_TBL: ;Nr. definiton of the flags
DC.B $00,$01,$02,$01 ;0..3
DC.B $04,$01,$02,$01 ;4..7
DC.B $08,$01,$02,$01 ;8..11
DC.B $04,$01,$02,$01 ;12..15

TIC_TBLE:

```

compact solution

(this code is part of the task manager [tsk_c.mmmw])

```
TSK_FLAGS:                                ;calculating the flags
    LDA    Z_CORE                          ;the core counter is the base
    LDX    #2                              ;the X reg. holds the flag value
    LSRA                               ;first bit does not care
    BEQ    TSK_FLG4                        ;if result is zero
                                              ;-> no flag is set

TSK_FLG1:                                  ;Loop label
    LSRA
    BCS    TSK_FLG2                        ;if a "1" is in the C flag
                                              ;-> the flag is calculated
    LSLX                                  ;move the flag one position to left
    BRA    TSK_FLG1

TSK_FLG2:                                  ;X holds the flag now
    LDA    Z_CORE
    LSRA
    BCC    TSK_FLG3                        ;if the C flag is clear -> set F0
                                              ;else ->
                                              ;set the flag in F1, clear F0
    TXA
    STA    CORE_F1
    CLR    CORE_F0
    BRA    TSK_FLAGE

TSK_FLG3:                                  ;set the flag in F0, clear F1
    TXA
    STA    CORE_F0
    CLR    CORE_F1
    BRA    TSK_FLAGE

TSK_FLG4:
    CLR    CORE_F0
    CLR    CORE_F1                        ;Clear the Core flags

TSK_FLAGE:                                ;end of routine
```

Links

System J.Schnyder GmbH

www.system-gmbh.ch

Development of Hard und Software, Training Systems
Layout Programs, Consultig

freescale

forums.freescale.com

Forum for freescale micros