


```

54      *****
55      *
56      *      The rtd (Return and Deallocate) macro may be used to partially deallocate
57      *      a subroutine stack frame that includes parameters passed on the stack. The
58      *      rtd macro performs the following functions: 1) Deallocates local storage;
59      *      2) Restores the previous stack frame pointer; 3) Returns to the calling
60      *      routine. Rtd DOES NOT remove any parameters from the stack. This function
61      *      must be performed by the calling routine. This macro is useful when
62      *      parameters are passed in registers rather than on the stack.
63      *
64      *      Usage: rtd  <s.f. reg>,<storage bytes>
65      *
66      *      The first parameter passed to link is the index register that is being used
67      *      as the stack frame pointer (either x or y). Although no check is made to
68      *      ensure that a legal index register name is passed to the macro, the assembler
69      *      will produce an "Unrecognized Mnemonic" error message when the macro is
70      *      expanded. The second parameter is the number of bytes of local storage
71      *      used by the subroutine.
72      *
73      *****
74      *
75 M      rtd          macro
76 M          ldab  #:1          ; number of bytes to deallocate.
77 M          ab:0              ; add it to the current stack frame pointer.
78 M          t:0s             ; deallocate storage by updating the stack pointer
79 M          pul:0            ; restore the previous stack frame pointer
80 M          rts              ; return to the calling routine
81 M          endm
82      *
83      *
84      *****
85      *
86      *      The frtd (Function Return and Deallocate) macro may be used to partially
87      *      deallocate a subroutine stack frame that includes parameters passed on
88      *      the stack. The frtd macro performs the following functions: 1) Deallocates
89      *      local storage; 2) Restores the previous stack frame pointer; 3) Returns
90      *      to the calling routine. Frtd DOES NOT remove any parameters from the stack.
91      *      This function must be performed by the calling routine. This macro is
92      *      useful when parameters are passed in registers rather than on the stack and
93      *      a value is being returned in the D-accumulator.
94      *
95      *      Usage: frtd  <s.f. reg>,<storage bytes>
96      *
97      *      The first parameter passed to frtd is the index register that is being used
98      *      as the stack frame pointer (either x or y). Although no check is made to
99      *      ensure that a legal index register name is passed to the macro, the assembler
100     *      will produce an "Unrecognized Mnemonic" error message when the macro is
101     *      expanded. The second parameter is the number of bytes of local storage
102     *      used by the subroutine.
103     *
104     *****
105     *
106 M      frtd        macro
107 M          pshb          ; save the lower byte of the return value.
108 M          ldab  #:1          ; number of bytes to deallocate.
109 M          ab:0              ; add it to the current stack frame pointer.
110 M          pulb          ; restore the lower byte of the return value.
111 M          t:0s             ; deallocate storage by updating the stack pointer
112 M          pul:0            ; restore the previous stack frame pointer.
113 M          rts              ; return to the calling routine.
114 M          endm

```

```

115      *
116      *
117      *****
118      *
119      *      The rtdx and rtdy (Return and Deallocate using x or y) macros may be used
120      *      to completely deallocate a subroutine stack frame including parameters that
121      *      were passed on the stack. The rtdx macro performs the following functions:
122      *      1) Deallocates the entire stack frame including local storage and passed
123      *      parameters; 2) restores the previous stack frame pointer; and 3) Returns
124      *      to the calling routine.
125      *
126      *      Usage: rtdx <storage bytes>
127      *      Usage: rtdy <storage bytes>
128      *
129      *      The only parameter passed to the routines is the number of bytes of local
130      *      storage that were originally allocated upon entry to the subroutine. These
131      *      macros have the advantage that the a and b accumulators are not used during the
132      *      deallocation process. This allows a value to be loaded into a, b, or d register
133      *      before execution of the rtdx or rtdy macro and returned to calling routine.
134      *
135      *****
136      *
137 M      rtdx      macro
138 M              ldy      :0+2,x      ; Load return address into the y-index register.
139 M              ldx      :0,x        ; restore the previous stack frame pointer
140 M              txs      ; Update stack pointer, removing storage space.
141 M              jmp      0,y        ; Return to the calling routine.
142 M      endm
143      *
144      *
145 M      rtdy      macro
146 M              ldx      :0+2,y      ; Load return address into the x-index register.
147 M              ldy      :0,y        ; restore the previous stack frame pointer.
148 M              tys      ; Update stack pointer, removing storage space.
149 M              jmp      0,x        ; Return to the calling routine.
150 M      endm
151      *
152      *
153      *****
154      *
155      *      The pshd macro pushes the 16-bit d-accumulator onto the stack. The
156      *      b-accumulator is pushed first so that the least significant 8-bits of
157      *      the 16-bit number appear on the stack at the higher address. This is
158      *      consistent with the way all 16-bit numbers are stored in memory.
159      *
160      *      Usage: pshd
161      *
162      *      No parameters are required by the macro.
163      *
164      *****
165      *
166 M      pshd      macro
167 M              pshb
168 M              psha
169 M      endm
170      *
171      *

```



```

233      *          flag is zero, the number is converted as an unsigned number. If the byte
234      *          is non-zero, the number will be converted as a 16-bit signed number.
235      *          Parameters are pushed onto the stack in the following order 1) Signed Flag;
236      *          2) Pointer to ASCII buffer; 3) Number to be converted. A typical
237      *          calling sequence would be:
238      *
239      *          clra                ; Do the conversion as an unsigned number
240      *          psha                ; put the flag on the stack.
241      *          ldd      #Buffer    ; get the address of the ascii buffer.
242      *          pshd                ; put the address on the stack.
243      *          ldd      Num        ; Get the number to convert.
244      *          pshd                ; Put it on the stack
245      *          jsr      Int2Asc    ; Go convert the number.
246      *          .
247      *          .
248      *          .
249      *
250      *          This subroutine has two local variables. The first, zs, is a boolean variable
251      *          used to suppress leading zeros when doing a conversion. It is located at an
252      *          offset of 0 from the stack frame pointer. The second local, Divisor, is a 16-
253      *          bit variable. It is used to divide the number being converted by succeeding
254      *          lower powers of 10. Divisor is located at an offset of 1 from the local stack
255      *          frame pointer.
256      *
257      *          NOTE: This routine was written assuming that the previous stack frame pointer
258      *          is the x-index register. HOWEVER, because the x-index register is required
259      *          by the integer divide instruction, the y-index register is used as the
260      *          stack frame pointer WITHIN the Int2Asc subroutine.
261      *
262      *
263      *          Declare locals
264      *
265 0000      PCSave      set      *          ; save the current PC value
266 0000      org        0          ; set PC to 0 for offsets to locals
267 0000      zs         rmb      1          ; declare zs variable.
268 0001      Divisor    rmb      2          ; declare Divisor variable.
269 0003      LocSize    set      *          ; number of bytes of local storage.
270 0000      org        PCSave
271      *
272      * Offsets to parameters
273      *
274 0007      Num      equ      LocSize+4    ; offset to Num parameter.
275 0009      BuffP    equ      LocSize+6    ; offset to BuffP parameter.
276 000B      Signed   equ      LocSize+8    ; offset to Signed parameter.
277      *
278 0000      Int2Asc   equ      *
279 0000 3C      [ 4]    pshx                ; save the previous stack frame pointer.
280 0001 CC2710 [ 3]    ldd      #10000     ; initialize the divisor to 10000.
281 0004      pshd
282 0004 37      [ 3]    pshb
283 0005 36      [ 3]    psha
284 0006 4F      [ 2]    clra                ; initialize zs to 0.
285 0007 36      [ 3]    psha
286 0008 1830   [ 4]    tsy                ; initialize the new stack frame pointer.
287 000A 18EC07 [ 6]    ldd      num,y      ; get the number to convert. Is it zero?
288 000D 260B   [ 3]    bne      Int2Asc1   ; no go do the conversion.
289 000F CC3000 [ 3]    ldd      #$3000     ; yes.
290 0012 CDEE09 [ 6]    ldx      BuffP,y    ; point to the buffer.
291 0015 18ED00 [ 6]    std      0,y       ; just put an ASCII 0 in the buffer.
292 0018 2050   [ 3]    bra      Int2Asc5   ; then return.
293 001A 186D0B [ 7] Int2Asc1 tst      Signed,y ; do the conversion as a signed number?

```

Application Note

```

294 001D 2716      [ 3]          beq          Int2Asc2      ; no.
295 001F 4D        [ 2]          tsta                          ; yes. Is the number negative?
296 0020 2A13      [ 3]          bpl          Int2Asc2      ; no. just go do the conversion.
297 0022 43        [ 2]          coma                          ; yes. make it a positive number by negation.
298 0023 53        [ 2]          comb
299 0024 C30001    [ 4]          addd         #$1
300 0027 18ED07    [ 6]          std          Num,y          ; save the result
301 002A 862D      [ 2]          ldaa        #'-'          ; get an ASCII minus sign.
302 002C CDEE09    [ 6]          ldx         BuffP,y        ; point to the buffer.
303 002F A700      [ 4]          staa        0,x           ; put it in the buffer
304 0031 08        [ 3]          inx                          ; point to the next location in the buffer.
305 0032 CDEF09    [ 6]          stx         BuffP,y        ; save the new pointer value.
306 0035 18EC07    [ 6] Int2Asc2  ldd          Num,y          ; get the remainder to convert.
307 0038 CDEE01    [ 6]          ldx         Divisor,y      ;
308 003B 02        [41]          idiv
309 003C 18ED07    [ 6]          std          Num,y          ; save the remainder.
310 003F 8F        [ 3]          xgdx                          ; put the dividend into d.
311 0040 5D        [ 2]          tstb                          ; was the dividend 0?
312 0041 2605      [ 3]          bne          Int2Asc3      ; no. go store the number in the buffer
313 0043 186D00    [ 7]          tst          zs,y           ; are we still suppressing leading zeros?
314 0046 2710      [ 3]          beq          Int2Asc4      ; yes. go setup for the next divide.
315 0048 CB30      [ 2] Int2Asc3  addb        #'0'          ; make the dividend ASCII.
316 004A 8601      [ 2]          ldaa        #1
317 004C 18A700    [ 5]          staa        zs,y           ; don't suppress leading zeros anymore.
318 004F CDEE09    [ 6]          ldx         BuffP,y        ; get a pointer to the buffer.
319 0052 E700      [ 4]          stab        0,x           ; save the digit.
320 0054 08        [ 3]          inx                          ; point to the next location.
321 0055 CDEF09    [ 6]          stx         BuffP,y        ; save the new pointer value.
322 0058 18EC01    [ 6] Int2Asc4  ldd          Divisor,y      ; get the previous divisor.
323 005B CE000A    [ 3]          ldx         #10
324 005E 02        [41]          idiv                          ; divide it by 10.
325 005F CDEF01    [ 6]          stx         Divisor,y      ; save the dividend. Is it zero?:
326 0062 26D1      [ 3]          bne          Int2Asc2      ; no. continue with the conversion.
327 0064 CDEE09    [ 6]          ldx         BuffP,y        ; get a pointer to the buffer.
328 0067 6F00      [ 6]          clr          0,x           ; null terminate the string.
329 0069 30        [ 3]          tsx                          ; this is only needed because we are using y as
                                   ; our sf pointer.

330 006A          Int2Asc5  rtdx         LocSize      ; return & deallocate locals & parameters
331 006A 1AEE05    [ 6]          ldy         LocSize+2,x    ; Load the return address into y-index register.
332 006D EE03      [ 5]          ldx         LocSize,x      ; restore the previous stack frame pointer.
333 006F 35        [ 3]          txs                          ; Update stack pointer, removing storage space.
334 0070 186E00    [ 4]          jmp          0,y           ; Return to the calling routine.
335 *
336 *
337 *
338 *****
339 *
340 *          This subroutine performs a 16 x 16 bit unsigned multiply and produces a 32-bit
341 *          result. Two 16-bit numbers are passed to the subroutine on the stack.
342 *          The 32-bit result is returned on the stack in place of the two 16-bit
343 *          parameters. This allows the calling routine to easily pull the product
344 *          from the stack and store the result. Because multiplication is a
345 *          commutative operation, the order in which the parameters are pushed
346 *          onto the stack is unimportant. A typical calling sequence would be:
347 *
348 *          ldd          Num1
349 *          pshd
350 *          ldd          Num2
351 *          pshd
352 *          jsr          Mull6x16
353 *          puld

```

```

354          *          std          Result32
355          *          puld
356          *          std          Result32+2
357          *          .
358          *          .
359          *          .
360          *
361          *          This subroutine has four local variables. Each variable occupies 1 byte
362          *          on the stack. These four bytes are used to hold the partial product as
363          *          the final answer is being computed. These four byte variables are
364          *          treated as 16-bit variables during the calculation.
365          *
366          *          NOTE: This routine was written assuming that the stack frame pointer
367          *          is the x-index register.
368          *
369          *          Declare locals
370          *
371 0073 PCSave      set          *          ; save the current PC value
372 0000          *          org          0          ; set PC to 0 for offsets to locals
373 0000 Prd0        rmb          1          ; declare ms byte of partial product variable.
374 0001 Prd1        rmb          1          ; declare next ms byte of partial product variable
375 0002 Prd2        rmb          1          ; declare next ls byte of partial product variable
376 0003 Prd3        rmb          1          ; declare ls byte of partial product variable.
377 0004 LocSize     set          *          ; number of bytes of local storage.
378 0073          *          org          PCSave
379          *
380          *          Offsets to parameters
381          *
382 0008 Fact1        equ          LocSize+4      ; offset to factor 1 parameter.
383 000A Fact2        equ          LocSize+6      ; offset to factor 2 parameter.
384          *
385          *          cycles clear
386          *
387 0073 Mull6x16     equ          *
388 0073 3C          [ 4]          pshx          ; save the previous stack frame pointer.
389 0074          [ 4]          clrd          ; clear the d-accumulator.
390 0074 4F          [ 2]          clra
391 0075 5F          [ 2]          clrb
392 0076          [ 2]          pshd          ; allocate & initialize the locals prd0 - prd3
393 0076 37          [ 3]          pshb
394 0077 36          [ 3]          psha
395 0078          [ 3]          pshd
396 0078 37          [ 3]          pshb
397 0079 36          [ 3]          psha
398 007A 30          [ 3]          tsx          ; initialize the new stack frame pointer.
399 007B A609        [ 4]          ldaa         Fact1+1,x      ; get the ls byte of factor 1.
400 007D E60B        [ 4]          ldab         Fact2+1,x      ; get the ls byte of factor 2.
401 007F 3D          [10]         mul          ; multiply them.
402 0080 ED02        [ 5]          std          Prd2,x      ; save the first term of the partial product.
403 0082 A608        [ 4]          ldaa         Fact1,x      ; get the ms byte of factor 1.
404 0084 E60B        [ 4]          ldab         Fact2+1,x      ; get the ls byte of factor 2.
405 0086 3D          [10]         mul          ; multiply them.
406 0087 E301        [ 6]          addd         Prd1,x      ; add the result into the partial product.
407 0089 ED01        [ 5]          std          Prd1,x      ; save the result.
408 008B A609        [ 4]          ldaa         Fact1+1,x      ; get the ls byte of factor 1.
409 008D E60A        [ 4]          ldab         Fact2,x      ; get the ms byte of factor 2.
410 008F 3D          [10]         mul          ; multiply them.
411 0090 E301        [ 6]          addd         Prd1,x      ; add the result into the partial product.
412 0092 ED01        [ 5]          std          Prd1,x      ; save the result.
413 0094 2402        [ 3]          bcc          Mull6          ; Was there a carry into Prd0?
414 0096 6C00        [ 6]          inc          Prd0,x      ; yes. 'add' it in.

```

Application Note

```

415 0098 A608      [ 4] Mul16      ldaa      Fact1,x      ; get the ms byte of factor 1.
416 009A E60A      [ 4]          ldab      Fact2,x      ; get the ms byte of factor 2.
417 009C 3D        [10]         mul                ; multiply them.
418 009D E300      [ 6]         addd     Prd0,x      ; add it to the partial product.
419 009F ED08      [ 5]         std       Fact1,x      ; overwrite the two parameters with the result.
420 00A1 EC02      [ 5]         ldd      Prd2,x
421 00A3 ED0A      [ 5]         std      Fact2,x
422 00A5           rtd                ; return and deallocate the locals.
423 00A5 C604      [ 2]         ldab     #LocSize    ; number of bytes to deallocate.
424 00A7 3A        [ 3]         abx                ; add it to the current stack frame pointer.
425 00A8 35        [ 3]         txs                ; deallocate storage by updating stack pointer.
426 00A9 38        [ 5]         pulx               ; restore the previous stack frame pointer.
427 00AA 39        [ 5]         rts                ; return to the calling routine.
429                cycles total=170      ; Total number of E cycles for a 16 x 16 multiply.
430                *
431                *
432                *****
433                *
434                *           This subroutine performs a 32 by 16 bit unsigned divide and produces a 32-bit
435                *           quotient and a 16-bit remainder. Both the divisor and dividend are passed to
436                *           the subroutine on the stack. The 32-bit quotient and 16-bit remainder are
437                *           returned on the stack in place of the divisor and dividend. This allows the
438                *           calling routine to easily pull the answer from the stack and store the result.
439                *           The divisor is pushed onto the stack first, followed by the lower 16-bits of
440                *           the dividend and finally the upper 16-bits of the dividend. A typical calling
441                *           sequence would be:
442                *
443                *           ldd      Divisor
444                *           pshd
445                *           ldd      Dividend+2
446                *           pshd
447                *           ldd      Dividend
448                *           pshd
449                *           jsr      Div32x16
450                *           puld
451                *           std      Quotient
452                *           puld
453                *           std      Quotient+2
454                *           puld
455                *           std      Remainder
456                *           .
457                *           .
458                *           .
459                *
460                *
461                *           This subroutine has two local variables. A 32-bit variable for partial
462                *           quotient results that is treated as two 16-bit variables and a 16-bit
463                *           variable for intermediate remainder results.
464                *
465                *           NOTE: This routine was written assuming that the previous stack frame pointer
466                *           is the x-index register. HOWEVER, because the x-index register is required
467                *           by the integer and fractional divide instructions, the y-index register is
468                *           used as the stack frame pointer WITHIN the Div32x16 subroutine.
469                *
470                *           Declare locals
471                *
472 00AB           PCSave      set      *           ; save the current PC value.
473 0000           org         0           ; set PC to 0 for offsets to locals.
474 0000           Quo0       rmb       2           ; declare upper 16-bits of quotient.
475 0002           Quo2       rmb       2           ; declare lower 16-bits of quotient.
476 0004           Rem        rmb       2           ; declare remainder.

```

```

477 0006          LocSize      set      *          ; number of bytes of local storage.
478 00AB          org          PCSave
479              *
480              *          Offsets to parameters
481              *
482 000A          Num0          equ       LocSize+4    ; upper 16-bits of Dividend.
483 000C          Num2          equ       LocSize+6    ; lower 16-bits of Dividend.
484 000E          Denm          equ       LocSize+8    ; 16-bit divisor.
485              *
486              cycles clear
487              *
488 00AB          Div32x16      equ       *
489 00AB 3C      [ 4]          pshx
490 00AC          clrld
491 00AC 4F      [ 2]          clra
492 00AD 5F      [ 2]          clrb
493 00AE          pshd
494 00AE 37      [ 3]          pshb
495 00AF 36      [ 3]          psha
496 00B0          pshd
497 00B0 37      [ 3]          pshb
498 00B1 36      [ 3]          psha
499 00B2          pshd
500 00B2 37      [ 3]          pshb
501 00B3 36      [ 3]          psha
502 00B4 1830    [ 4]          tsy
503 00B6 18EC0A [ 6]          ldd      Num0,y
504 00B9 CDA30E [ 7]          cpd      Denm,y
505 00BC 2507    [ 3]          blo      Div32x16a
506 00BE CDEE0E [ 6]          ldx      Denm,y
507 00C1 02     [41]         idiv
508 00C2 CDEF00 [ 6]          stx      Quo0,y
509 00C5 CDEE0E [ 6]Div32x16a ldx      Denm,y
510 00C8 03     [41]         fdiv
511 00C9 CDEF02 [ 6]          stx      Quo2,y
512 00CC 18ED04 [ 6]          std      Rem,y
513 00CF 18EC0C [ 6]          ldd      Num2,y
514 00D2 CDEE0E [ 6]          ldx      Denm,y
515 00D5 02     [41]         idiv
516 00D6 18E304 [ 7]          addd     Rem,y
517 00D9 18ED04 [ 6]          std      Rem,y
518 00DC 8F     [ 3]          xgdx
519
520 00DD 18E302 [ 7]          addd     Quo2,y
521 00E0 18ED0C [ 6]          std      Num2,y
522 00E3 18EC00 [ 6]          ldd      Quo0,y
523 00E6 C900    [ 2]          adcb    #0
524 00E8 8900    [ 2]          adca    #0
525 00EA 18ED0A [ 6]          std      Num0,y
526 00ED 8F     [ 3]          xgdx
527 00EE CDA30E [ 7]          cmpd     Denm,y
528 00F1 2519    [ 3]          blo      Div32x16b
529 00F3 18A30E [ 7]          subd     Denm,y
530 00F6 18ED04 [ 6]          std      Rem,y
531 00F9 18EC0C [ 6]          ldd      Num2,y
532 00FC C30001 [ 4]          addd     #1
533 00FF 18ED0C [ 6]          std      Num2,y
534 0102 18EC0A [ 6]          ldd      Num0,y
535 0105 C900    [ 2]          adcb    #0

```

Application Note

```

536 0107 8900      [ 2]      adca      #0      ; add the possible carry to the upper 8-bits.
537 0109 18ED0A    [ 6]      std      Num0,y  ; save the result.
538 010C 18EC04    [ 6]Div32x16b ldd      Rem,y  ; get the final remainder.
539 010F 18ED0E    [ 6]      std      Denm,y  ; overwrite the divisor.
540 0112 30        [ 3]      tsx      ; need to do this for rtd to work correctly. See
                    ; NOTE.
541 0113          rtd      x,LocSize ; deallocate locals & return.
542 0113 C606      [ 2]      ldab     #LocSize ; number of bytes to deallocate.
543 0115 3A        [ 3]      abx      ; add it to the current stack frame pointer.
544 0116 35        [ 3]      txs      ; deallocate storage by updating stack pointer.
545 0117 38        [ 5]      pulx     ; restore the previous stack frame pointer.
546 0118 39        [ 5]      rts      ; return to the calling routine.
547                *
548                cycles total=347 ; Total number of E cycles for a 32 x 16 divide.
549                *

Errors:  None
Labels: 30
Last Program Address: $0118
Last Storage Address: $FFFF
Program Bytes: $0119 281
Storage Bytes: $000D 13

```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor
 Technical 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

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

