COP888GW Features and Applications

National Semiconductor
Application Note 982
Martin Embacher
April 1995

INTRODUCTION

The COP888GW is a member of National Semiconductor's COP888 family of microcontrollers fabricated in M²CMOS™ technology. The device was established using CCM (Configurable Controller Methodology) design techniques which provide a fast and reliable way to create new derivatives for a fast growing and demanding controller market. In addition to the COP888 feature core this controller includes a math unit to allow fast multiply/divide operations. The controller has special timers for motor control allowing it to be used in applications where enhanced processing power and motor control capabilities are required (as in plotters, robot-arms or information systems.)

This note describes the COP888GW microcontroller emphasizing the powerful new features. The reader will be given easy to use macros to take full advantage of the new CCM blocks.

COP888GW KEY FEATURES

- Multiply/Divide Unit
- Two 16-bit capture modules with 8-bit prescalers
- Four pulse train generators with 16-bit prescalers
- Full duplex UART
- MICROWIRE/PLUS™ serial port with interrupt
- 16 kbytes of on-board ROM
- 512 bytes of on-board RAM
- Multi-Input Wakeup pins (8) optionally usuable hardware interrupts either on falling or rising edge
- Two 16-bit timers, each with two 16-bit autoreload/capture registers supporting:
  — Processor independent PWM mode
  — External Event counter mode
  — Input Capture mode
- Low EMI
- Two power saving modes: HALT and IDLE
- Fourteen multi-source vectored interrupts
- Software selectable I/O options
- Package:
  — 68 PLCC with 56 I/O pins

CAPTURE TIMERS

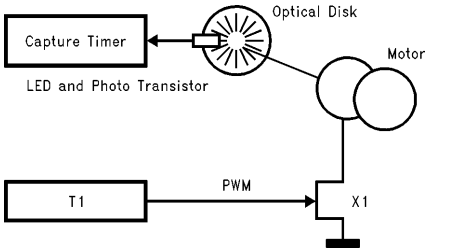
With its two high speed, high resolution capture timers, the COP888GW is ideally suited high resolution frequency measurement. To give a feeling for the capabilities, Table I shows the capture timers resolution versus the measured signal freuency with a processor clock (CKI) of 10 MHz. The sampling rate is CKI divided by a 8-bit prescaler. In the table, the prescaler is set to 1 to allow maximum performance. This results in a maximum resolution of 100 ns.

TABLE I. Capture Timer Resolution vs Signal Frequency

Resolution/ Bits	16	14	12	10	8
Max. Frequency	152 Hz	610 Hz	2.4 kHz	9.6 kHz	39 kHz

The capture timer can be used for a closed loop control in a DC motor. One of the standard COP888 timers (T1 or T2) generates a PWM signal to control the motor. One of the capture timer modules measures the actual RPM of the motor. A simple algorithm is used to adjust the motor's speed to the desired value.

Figure 1 shows a block diagram of a closed loop DC motor control with one capture and one PWM timer. The MOSFET X1 acts as a low-side driver for the motor and controls the current flow. The current through the motor is proportional to the duty cycle of the PWM signal, generated by T1, at the gate of X1. An optical disk, in conjunction with an LED/photo transistor senses the motor's RPM. The pulses generated by the photo transistor are directly fed to the capture timer's input. The capture timer inputs are alternate functions of Port-L. This I/O-port features integrated schmitt trigger inputs eliminating the need for external signal conditioning of the input signals.



TL/DD/12376–1

FIGURE 1. Closed Loop DC Motor Control Block Diagram

SOFTWARE MODULES

The following section provides a guide to general purpose macros and subroutines to set-up the capture timer and the standard timers T1 to T2. The fuctions are written as of macros so the user can easily include them in the application software.

RRD-B30M75/Printed in U. S. A.

CAPTURE TIMERS

Capture Timers Setup Routine

This routine sets the capture timer module CCM1 or CCM2 to allow input capture of a signal on the respective port pin. Additionally the I/O pins Port L6 for CCM1 and Port L7 for CCM2 are configured as inputs. This macro should be placed in a separate macro file and included in the source code after the processor specific include file. *Figure 2* shows the source code to include the macros in the applications program.

```
.incld cop888gw.inc
.incld gw_macros.inc
.sect main_prog, rom
..
..
.end
```

FIGURE 2. Including a Macro Into Source Code

The macro is called in the source code as:

__CT__SETUP, __MODULE, __PRESCALER, __EDGE, __INTR

__MODULE is the capture module number

1 = capture module 1
2 = capture module 2

__PRESCALER is the prescaler value the range is 0 to 255.

EDGE selects the input capture edge

0 = rising edge
1 = falling edge

__INTR select if capture interrupt is used

0 = disable capture interrrupt
1 = enable capture interrrupt

The macro will check that all values are in the acceptable range and warn the user if this is not the case.

An example of the generated code is shown below. The macro is called to setup capture timer 1 with a prescaler of 100 (dec) to capture signals on a falling edge without the generation of an interrupt.

```
_CT_SETUP 1 100 1 0

ld      b, #CCMR1 ; point to capture con
ld      [b], #0  ; stop capture, clear
rbit    6, portlc ; cm 1 input on port L
ld      cm1psc, #100 ; load prescaler of c
ld      [b], #ZZ0000 ; start capture
```

TL/DD/12376–2

FIGURE 3. __CT__SETUP Resulting Code

Special attention is needed if the interrupts are enabled in the macro by setting the __INTR directive to one. An appropriate interrupt handler must be placed elsewhere in the code. The macro will allow the capture timer to generate interrupts, however it will not provide the respective interrupt service routine.

Capture Timer Read Macro

__CT__READ __MODULE __VARIABLE

__MODULE is the capture module number

1 = capture module 1
2 = capture module 2

__VARIABLE is a variable to hold the contents of the capture register

Standard Timer Setup for PWM Output

__TIM__SETUP __NUMBER __CYCLE __TON __INIT

__NUMBER is the timer number

1 = timer 1
2 = timer 2
3 = timer 3

__CYCLE is the cycle time of the PWM signal the range is 0 to 65535

__TON is on time of the PWM signal

__INIT is the initialization value

0 = start with low
1 = start with high

Standard Timer Glitch Free PWM Output

This macro is used to provide a glitch free switched PWM output with the standard COP8 PWM timer. In the macro interrupts are disabled to wait until a PWM cycle is completed and then reload the registers are re-written. This produces—in difference to an interrupt driven routine—a glitch free output of the PWM signal.

__TIM__PWM __NUMBER __CYCLE __TON

__NUMBER is the timer number

1 = timer 1
2 = timer 2
3 = timer 3

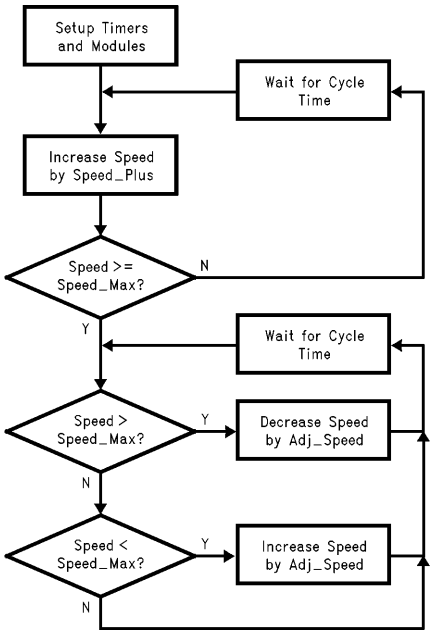
__CYCLE is the cycle time of the PWM signal the range is 0 to 65535

__TON is on the time of the PWM signal

Note: The source of the macros is found in Section Capture Module Macros on page 5.

DC Motor Control Example

An example for the application of the macros is basic DC motor closed loop control. The motor drive consists of the phases start and run. During start, the motor's speed increases up to the maximum speed set by speed__max. During run, the speed is kept around speed __max. The slope of speed increase is set by speed__plus and the maximum allowable speed increase or decrease during run is set by adj__speed. The speed update cycle time may be controlled by the IDLE timer, generating an interrupt every 4096 t_C. *Figure 4* shows the flowchart of a simple closed loop DC-motor control. The macros can be used to setup the times and to read the current speed value.



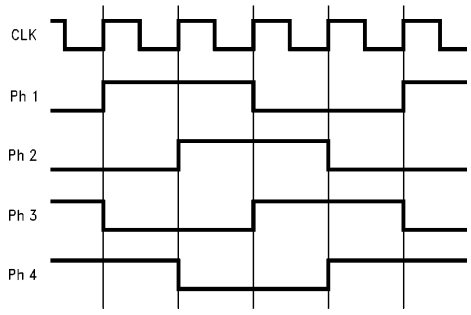
TL/DD/12376–3

FIGURE 4. Basic DC Motor Control Loop Flowchart

Pulse Train Generators

The COP888GW contains four independent pulse train generators designed for stepper motor control. The pulse train generator provides a programmable number of 50% duty cycle pulses on the output pin. Depending on the external sequencer logic these timers can be used for regular step control or for microstep driving.

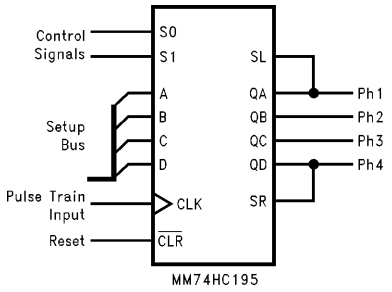
If the user does not require pulse train outputs on all pins the pulse train generators can be used as four independent timers with interrupt capability.



TL/DD/12376–4

FIGURE 5. Standard Stepper Motor Control Waveform (Two Phases On)

Microstepping is used in applications where the step sizes have to be very small to allow smooth transitions from one state to another. Special microstep drivers and motors have been developed. These devices turn the applied pulses into current steps to run the motor. The standard input of these drivers is a pulse train as generated by the COP888GW microcontrollers. To show the basic stepper motor control circuit a standard stepper motor drive circuit is discussed based on a four-phase motor. *Figure 5* shows the waveform required to control a standard motor. CLK is the pulse train generated by the pulse train generator, PH1–PH4 are the outputs to the motor drivers. These signals can be generated by using a simple shift/rotate register.



TL/DD/12376–5

FIGURE 6. Simple Stepper Motor Sequencer

Figure 6 shows an application circuit for a bidirectional stepper motor. The shift register is loaded under software control and then the pulse train is started. More than one motor can be connected in the same manner. Each motor gets its own sequencer the setup bus signals are common for every sequencer. Two control signals, unique for every motor, select the direction and the mode of the sequencer. Mode can be load new pattern, shift left and shift right. The circuit shown above is capable of controlling low current stepper motors used in some applications, if a higher output drive is required appropriate driver stages should be used.

As stepper motors usually have some inertia the pulse train starts up slowly and increases its frequency, under software control, up to full step speed. The same scheme as shown in *Figure 4* can be applied if increase speed is replaced by increase pulse train frequency and decrease speed accordingly to decrease pulse train frequency.

Pulse Train Generator Routines

The first routine sets up the pulse train generators PT1–PT4. Additionally the respective output pins Port E0 for PT1, Port E1 for PT2, Port E2 for PT3 and Port E3 for PT4 can be configured if the user needs to output the data.

Pulse Train Generator Setup

__PT__SETUP __MODULE, __PRESCALER, __PULSES, __OUT, __INTR, __RUN

__MODULE is the pulse train module number

1 = pulse train module 1
2 = pulse train module 2
3 = pulse train module 3
4 = pulse train module 4

__PRESCALER is the prescaler value the range is 0 to 65535.

__PULSES selects the number of pulses: 0 to 65535.

__OUT enables or disables pulse output

0 = output disabled
1 = output enabled

__INTR select if an interrupt is generated on completion of the pulse generation

0 = disable interrupt
1 = enable interrupt

__RUN selects if the pulses are output directly after initialization or at a later time.

0 = don't start pulse train generator

1 = start pulse train generator

The macro will check that all values are in the acceptable range and warn the user if this is not the case.

Pulse train output:

The second routine loads a new number of pulses into the pulse train registers and starts the generation. Setting regarding interrupts and output are not affected. The macro is called in the source code as:

__PT__CHANGE__MODULE,__PULSES

__MODULE is the pulse train module number

1 = pulse train module 1
2 = pulse train module 2
3 = pulse train module 3
4 = pulse train module 4

__PULSES selects the number of pulses: 0 to 65535.

MULTIPLE/DIVIDE

The device constains a multiply/divide block to speed up the multiply/divide operations. To give an idea about the speed increase, Table II shows a comparison of the standard multiply/divide operations done in software with the same operation performed using the MUL/DIV unit.

TABLE II. Multiply/Divide Operation Speed

Type of Operation	MUL/DIV Operation in Software	MUL/DIV Block
Multiply 8 x 8	129 t_C	11 t_C
Multiply 8 x 16	243 t_C	11 t_C
Divide 8 x 8	208 t_C	14 t_C
Divide 24 x 8	827 t_C	14 t_C

Note: Comparisons include loading the registers for the MUL/DIV block and accordingly the RAM cells used for software multiply/divide on standard COP8 microcontrollers. The actual 16 x 8 multiply operation takes one t_C and a 24 x 16 divide takes 2 t_C. However for accurate comparison the register setup times have to taken into account.

The following section provides software modules used to set up the multiply/divide unit and to load the respective registers. Although the MUL/DIV block does not allow signed arithmetic it can be used with some additional software to perform signed operations. The signed operations check for a sign in every number to be divided or multiplied—performs a sign conversion if negative—does the operation and adjusts the output accordingly.

Note: These software modules use constants to show the basic operation example of loading the MUL/DIV registers. These need to be customized in the application software.

Macro for unsigned 8 x 16 multiply

__MUL816U __MUL1 __MUL2
__MUL1 is the 8-bit unsigned multiplier
__MUL2 is the 16-bit unsigned multiplicand

Macro for unsigned 16 x 16 divide

__DIV1616U __DIV1 __DIV2
__DIV1 is the 16-bit unsigned dividend
__DIV2 is the 16-bit unsigned divisor

Macro for signed 6 x 16 multiply

__MUL816S __MUL1 __MUL2
__MUL1 is the 8-bit signed multiplier
__MU21 is the 16-bit signed multiplicand

Macro for signed 16 x 16 divide

__DIV1616S __DIV1 __DIV2
__DIV1 is the 16-bit unsigned dividend
__DIV2 is the 16-bit unsigned divisor

FURTHER APPLICATION HINTS

It was shown that the COP888GW with its new and enhanced functions can operate as a system contoller in various applications. Due to the large number of I/O lines additional features can be included in the system with minimal external hardware effort. For example it was shown in application note AN-673 how a simple two way multiplexed LCD control can be done with a COP8 microcontroller. The feature can be easily adopted to the COP888GW allowing system e.g. a simple plotter device to be controlled. Another application example is a climate control system, where COP888GW can be used to control the heating process. Control algorithms can be generated with National's NeuFuz™ software package. User interfaces can be done as described above with LCD drive. Diagnostics is possible with the on-chip UART.

CAPTURE MODULE MACROS

```
; This macro sets up the cm on the COP888xW including
; prescaler, edge select, interrupt and I/O port configuration
;
.macro  _CT_SETUP, _MODULE, _PRESCALER, _EDGE, _INTR
.mloc   _CMSETUP        ; local variable


.ifstr _MODULE NE 1     ; check for correct use of _MODULE
.ifstr _MODULE NE 2
.error capture module must be selected 1 or 2
.endif
.endif

.if _PRESCALER > 255    ; check for correct use of _PRESCALER
.error capture prescaler out of range -  must be 0 to 255
.endif

.if _EDGE > 1           ; check for correct use of _EDGE
.error select rising (0) or falling (1) edge
.endif

.if _INTR > 1           ; check for correct use of _INTR
.error select interrupt disabled (0) or enabled (1)
.endif

.ifstr _MODULE EQ 1
        ld      b, #CCMR1       ; point to capture control 1
.else
        ld      b, #CCMR2       ; point to capture control 2
.endif
        ld      [b], #0         ; stop capture, clear pending

.ifstr _MODULE EQ 1
        rbit    6, portlc       ; cm 1 input on port L6
        ld      cm1psc, #_PRESCALER     ; load prescaler of cm 1
.else
        rbit    7, portlc       ; cm 2 input on port L7
        ld      cm1psc, #_PRESCALER     ; load prescaler of cm 2
.endif


        _CMSETUP = ((1) OR (_EDGE*010) OR (_INTR*002))
        ld      [b], #_CMSETUP  ; start capture
.endm
```

TL/DD/12376–6

```
;
; macro for capture module (cm) read
;
; This macro reads the contents of the cm on the COP888xW
; to the RAM location of the specified variable
;
.macro  _CT_READ, _MODULE, _VARIABLE
.mloc   _CTREAD

.ifstr _MODULE NE 1     ; check for correct use of _MODULE
.ifstr _MODULE NE 2
.error capture module must be selected 1 or 2
.endif
.endif

.if _VARIABLE > 127    ; check for correct use of _VARIABLE
.error capture variable out of range -  must be 0 to 127
.endif

_CTREAD = _MODULE

        ld      b, #_VARIABLE
        ld      x, #_CTREAD
        ld      a, [x+]
        x       a, [b+]
        ld      a, [x]
        x       a, [b]
.endm
```

TL/DD/12376–7

STANDARD TIMER PWM SETUP MACRO

```
; macro for standard timer pwm setup
;
; This macro configures a standard timer of the COP888xW
; including PWM cycle on-time and init value

.macro  _TIM_SETUP, _NUMBER, _CYCLE, _TON, _INIT

.ifstr _NUMBER NE 1     ; check for correct use of _NUMBER
.ifstr _NUMBER NE 2
.error timer must be selected 1 or 2
.endif
.endif

.if _CYCLE < 48         ; check min cycle time
.warning timer _NUMBER cycle time to low for glitch free PWM output
.endif
.if _CYCLE > 65535      ; check max cycle time
.error timer _NUMBER cycle time too high must be below 2^16
.endif

.if _TON > _CYCLE       ; check on time
.error on time can`t be longer than cycle
.endif

.if _INIT > 1           ; check init value
.error init value can be only 0 or 1
.endif

.if _NUMBER = 1         ; timer 1
        ld      b, #CNTRL
        ld      a, [b]
        and     a, #00f         ; stop all timer activity
        x       a, [b]
        ld      b, #TMR1LO
        ld      [b+], #low(_TON)                ; on-time low
        ld      [b+], #high(_TON)               ; on-time high
        ld      [b+], #low(_CYCLE - _TON)       ; off-time low
        ld      [b], #high(_CYCLE - _TON)       ; off-time high
        ld      b, #T1RBLO
        ld      [b+], #low(_TON)                ; on-time low
        ld      [b], #high(_TON)                ; on-time high

        ld      b, #portgd
.if     _INIT = 0
        rbit    3, [b]
```

TL/DD/12376–8

```
.else
        sbit    3, [b]
.endif

        ld      a, [b-]                         ; dummy read to point
                                                ; to portgc
        sbit    3, [b]                          ; port as output

        ld      b, #CNTRL
        ld      a, [b]
        or      a, #b`10110000                  ; PWM and start
        x       a, [b]
.endif          ; end timer 1 setup

.if _NUMBER = 2         ; timer 2
        ld      b, #T2CNTRL
        ld      [b], #0                 ; stop all activity

        ld      b, #TMR2LO
        ld      [b+], #low(_TON)                ; on-time low
        ld      [b+], #high(_TON)               ; on-time high
        ld      [b+], #low(_CYCLE - _TON)       ; off-time low
        ld      [b+], #high(_CYCLE - _TON)      ; off-time high
        ld      [b+], #low(_TON)                ; on-time low
        ld      [b], #high(_TON)                ; on-time high

        ld      b, #portld
.if     _INIT = 0
        rbit    4, [b]
.else
        sbit    4, [b]
.endif

        ld      a, [b-]                         ; dummy read to point
                                                ; to portlc
        sbit    4, [b]                          ; port as output

        ld      b, #T2CNTRL
        ld      [b], #b`10110000                ; PWM and start
.endif          ; end timer 2 setup
.endm
```

TL/DD/12376–9

STANDARD TIMER PWM OUTPUT MACRO

```
.macro  _TIM_PWM, _NUMBER, _CYCLE, _TON
.mloc   _WAIT0, _WAIT1, WAIT2, WAIT3
.mloc   _MORE0, _MORE1, MORE2, MORE3

.ifstr _NUMBER NE 1     ; check for correct use of _NUMBER
.ifstr _NUMBER NE 2
.error timer must be selected 1 or 2
.endif
.endif

.if _CYCLE < 48         ; check min cycle time
.warning timer _NUMBER cycle time to low for glitch free PWM output
.endif
.if _CYCLE > 65535      ; check max cycle time
.error timer _NUMBER cycle time too high must be below 2^16
.endif

.if _TON > _CYCLE       ; check on time
.error on time can`t be longer than cycle
.endif

        rbit    GIE, PSW        ; disable interrupts !
.warning interupts are disabled for at least _CYCLE tC

.if     _ON > (_CYCLE / 2)

.if     _NUMBER = 1
        ld      [b], ICNTRL
        rbit    T1PNDB, [b]
_WAIT0:
        ifbit   T1PNDB, [b]
        jp      _MORE0
        jp      _WAIT0
_MORE0:
        ld      b, #T1RALO
        ld      [b+], #low(_CYCLE - _TON)       ; off-time low
        ld      [b], #high(_CYCLE - _TON)       ; off-time high
        ld      b, #T1RBLO
        ld      [b+], #low(_TON)                ; on-time low
        ld      [b], #high(_TON)                ; on-time high
.else
        ld      [b], T2CNTRL
        rbit    T2PNDB, [b]
_WAIT1:
        ifbit   T2PNDB, [b]
```

TL/DD/12376–10

```
        jp      _MORE1
        jp      _WAIT1
_MORE1:
        ld      b, #T2RALO
        ld      [b+], #low(_CYCLE - _TON)       ; off-time low
        ld      [b+], #high(_CYCLE - _TON)      ; off-time high
        ld      [b+], #low(_TON)                ; on-time low
        ld      [b], #high(_TON)                ; on-time high
.endif

.else
        ld      [b], PSW
        rbit    T1PNDA, [b]
_WAIT2:
        ifbit   T1PNDA, [b]
        jp      _MORE2
        jp      _WAIT2
_MORE2:
        ld      b, #T1RBLO
        ld      [b+], #low(_TON)                ; on-time low
        ld      [b], #high(_TON)                ; on-time high
        ld      b, #T1RALO
        ld      [b+], #low(_CYCLE - _TON)       ; off-time low
        ld      [b], #high(_CYCLE - _TON)       ; off-time high
.else
        ld      [b], T2CNTRL
        rbit    T2PNDA, [b]
_WAIT3:
        ifbit   T2PNDA, [b]
        jp      _MORE3
        jp      _WAIT3
_MORE3:
        ld      b, #T2RBHI
        ld      [b-], #high(_TON)               ; on-time high
        ld      [b-], #low(_TON)                ; on-time low
        ld      [b-], #high(_CYCLE - _TON)      ; off-time high
        ld      [b],  #low(_CYCLE - _TON)       ; off-time low
.endif

        sbit    GIE, PSW        ; enable interrupts
.endm
```

TL/DD/12376–11

PULSE TRAIN GENERATOR SETUP MACRO

```
; macro for pulse train generator (ptg) setup
;
; This macro sets up the ptg on the COP888xW including
; prescaler, # pulses, interrupt and I/O port configuration
;
.macro  _PT_SETUP, _MODULE, _PRESCALER, _PULSES, _OUT, _INTR, _RUN
.mloc _PTSETUP

.if _MODULE < 1         ; check for correct use of _MODULE
.error pulse train generator must be selected 1 to 4
.endif
.if _MODULE > 5
.error pulse train generator must be selected 1 to 4
.endif

.if _PRESCALER > 65535  ; check for correct use of _PRESCALER
.error pulse train prescaler out of range -  must be 0 to 2^16 - 1
.endif

.if _PULSES > 65535  ; check for correct use of _PULSES
.error pulse train: number of pulses out of range -  must be 0 to 2^16 - 1
.endif

.if _INTR > 1           ; check for correct use of _INTR
.error select interrupt disabled (0) or enabled (1)
.endif

.if _RUN > 1           ; check for correct use of _RUN
.error select run later (0) or immedately (1)
.endif

.if _MODULE = 1
        _PTSETUP = 1
        ld      b, #C1PRL       ; point to counter 1 prescaler low
.endif

.if _MODULE = 2
        _PTSETUP = 5
        ld      b, #C2PRL       ; point to counter 1 prescaler low
.endif

.if _MODULE = 3
        _PTSETUP = 1
        ld      b, #C3PRL       ; point to counter 1 prescaler low
.endif
```

TL/DD/12376–12

```
.if _MODULE = 4
        _PTSETUP = 5
        ld      b, #C4PRL       ; point to counter 1 prescaler low
.endif

        ld      [b+], #low(_PRESCALER)  ; load prescaler low
        ld      [b+], #high(_PRESCALER) ; load prescaler high
        ld      [b+], #low(_PULSES)     ; load # pulses low
        ld      [b],  #high(_PULSES)    ; load # pulses high

.if _OUT = 1
        sbit    (_MODULE-1), portec     ; counter _MODULE output on port E3
.endif

.if _MODULE < 3
        ld      b, #CCR1        ; point to counter control 1
.else
        ld      b, #CCR2        ; point to counter control 2
.endif

        rbit    (_PTSETUP+2), [b] ; make sure test bit = 0
        rbit    (_PTSETUP+1), [b] ; clear intr. pending bit

.if _INTR = 1
        sbit    (_PTSETUP+0), [b] ; enable interrupt
.else
        rbit    (_PTSETUP+0), [b] ; disable interrupt
.endif

.if _RUN = 1
        sbit    (_PTSETUP-1), [b] ; start pulse train output
.else
        rbit    (_PTSETUP-1), [b] ; don't start pulse train output
.endif

.endm
```

TL/DD/12376–13

PULSE TRAIN GENERATOR CHANGE MACRO

```
; macro for pulse train generator (ptg) change
;
; This macro changes the ptg on the COP888xW including to
; output a number of pulses and starts the ptg. It does not
; change the output configuration or the interrupt configuration.
;
; Note:
; this routine should only be used after the CxIPND bit was
; set or a pulse train interrupt occured
;

.macro  _PT_CHANGE, _MODULE, _PULSES

.if _MODULE < 1         ; check for correct use of _MODULE
.error pulse train generator must be selected 1 to 4
.endif
.if _MODULE > 5
.error pulse train generator must be selected 1 to 4
.endif

.if _PULSES > 65535  ; check for correct use of _PULSES
.error pulse train: number of pulses out of range -  must be 0 to 2^16 - 1
.endif

.if _MODULE = 1
        ld      b, #C1CTL               ; point to counter 1 register low
        ld      [b+], #low(_PULSES)     ; load # pulses low
        ld      [b],  #high(_PULSES)    ; load # pulses high
        ld      b, #CCR1                ; point to counter control 1
        rbit    2, [b]                  ; reset interrupt pending bit
        sbit    0, [b]                  ; start counter
.endif

.if _MODULE = 2
        ld      b, #C2CTL               ; point to counter 1 register low
        ld      [b+], #low(_PULSES)     ; load # pulses low
        ld      [b],  #high(_PULSES)    ; load # pulses high
        ld      b, #CCR1                ; point to counter control 1
        rbit    6, [b]                  ; reset interrupt pending bit
        sbit    4, [b]                  ; start counter
.endif
```

TL/DD/12376–14

```
.if _MODULE = 3
        ld      b, #C3CTL               ; point to counter 1 register low
        ld      [b+], #low(_PULSES)     ; load # pulses low
        ld      [b],  #high(_PULSES)    ; load # pulses high
        ld      b, #CCR2                ; point to counter control 2
        rbit    2, [b]                  ; reset interrupt pending bit
        sbit    0, [b]                  ; start counter
.endif

.if _MODULE = 4
        ld      b, #C4CTL               ; point to counter 1 register low
        ld      [b+], #low(_PULSES)     ; load # pulses low
        ld      [b],  #high(_PULSES)    ; load # pulses high
        ld      b, #CCR2                ; point to counter control 2
        rbit    6, [b]                  ; reset interrupt pending bit
        sbit    4, [b]                  ; start counter
.endif
.endm
```

TL/DD/12376–15

MULTIPLY/DIVIDE MACROS

```
; macro for 8 x 16 multiplication

.macro  _MUL816U, _MUL1, _MUL2

.if _MUL1 > 255
;.error multiplier out of range
.endif

.if _MUL2 > 65535
.error multiplicant out of range
.endif

        ld      b, #mdr2
        ld      [b+], #_MUL1
        ld      a, [b+]                 ; dummy read to inc b
        ld      [b+], #low(_MUL2)
        ld      [b+], #high(_MUL2)
        ld      [b], #001               ; multiply
.endm

; macro for 16 / 16 unsigned division

.macro  _DIV1616U, _DIV1, _DIV2

.if _DIV1 > (256*256-1)
.error divident out of range
.endif

.if _DIV2 > 65535
.error divisor out of range
.endif

        ld      b, #mdr1
        ld      [b+], #low(_DIV1)
        ld      [b+], #high(_DIV1)
        ld      [b+], #0
        ld      [b+], #low(_DIV2)
        ld      [b+], #high(_DIV2)
        ld      [b], #002               ; divide
        nop                             ; wait one cycle
.endm
```

TL/DD/12376–16

```
; macro for 8 x 16 signed multiplication

.macro  _MUL816S, _MUL1, _MUL2
#SIG
.if     (_MUL1 > 0) & (_MUL2 > 0)       ; positive multiply
#NOSIG
        _MUL816U _MUL1 _MUL2
.endif

#SIG
.if     (_MUL1 > 0) & (_MUL2 < 0)       ; negative multiply
#NOSIG
        _MUL816U _MUL1 -(_MUL2)

        ld      b, #mdr1                ; build one`s complemet
        ld      a, [b]
        xor     a, #0ff
        rc
        adc     a, #1
        x       a, [b+]
        ld      a, [b]
        xor     a, #0ff
        adc     a, #0
        x       a, [b]
.endif

#SIG
.if     (_MUL1 < 0) & (_MUL2 > 0)       ; negative multiply
#NOSIG
        _MUL816U -(_MUL1) _MUL2

        ld      b, #mdr1                ; build one`s complemet
        ld      a, [b]
        xor     a, #0ff
        rc
        adc     a, #1
        x       a, [b+]
        ld      a, [b]
        xor     a, #0ff
        adc     a, #0
        x       a, [b]
.endif
```

TL/DD/12376–17

```
#SIG
.if     (_MUL1 < 0) & (_MUL2 < 0)       ; positive multiply
#NOSIG
        _MUL816U -(_MUL1) -(_MUL2)
.endif
.endm


; macro for 16 / 16 signed division

.macro  _DIV1616S, _DIV1, _DIV2

#SIG
.if     (_DIV1 > 0) & (_DIV2 > 0)
#NOSIG
        _DIV1616U _DIV1 _DIV2
.endif

#SIG
.if     (_DIV1 > 0) & (_DIV2 < 0)
#NOSIG
        _DIV1616U _DIV1 -_DIV2
        ld      b, #mdr1                ; build one`s complemet
        ld      a, [b]
        xor     a, #0ff
        rc
        adc     a, #1
        x       a, [b+]
        ld      a, [b]
        xor     a, #0ff
        adc     a, #0
        x       a, [b]
.endif

#SIG
.if     (_DIV1 < 0) & (_DIV2 > 0)
#NOSIG
        _DIV1616U -_DIV1 _DIV2
        ld      b, #mdr1                ; build one`s complemet
        ld      a, [b]
        xor     a, #0ff
        rc
        adc     a, #1
        x       a, [b+]
```

TL/DD/12376–18

```
        ld      a, [b]
        xor     a, #0ff
        adc     a, #0
        x       a, [b]
.endif

#SIG
.if     (_DIV1 < 0) & (_DIV2 < 0)
#NOSIG
        _DIV1616U -_DIV1 -_DIV2
.endif
.endm
```

TL/DD/12376–19

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998