SILICON LABORATORIES

AN110

16-BIT PWM USING AN ON-CHIP TIMER

Relevant Devices

This application note applies to the following devices:

C8051F000, C8051F001, C8051F002, C8051F005, C8051F006, C8051F007, C8051F010, C8051F011, C8051F012, C8051F015, C8051F016, C8051F017, C8051F220, C8051F221, C8051F226, C8051F230, C8051F231, and C8051F236.
Note: the C8051F0xx devices have an on-chip PCA which may be more suitable for PWM generation. See AN007 for more information.

Introduction

This document describes how to implement a 16-bit pulse width modulator (PWM) digital-to-analog converter (DAC). The PWM consists of two parts:

1. A timer to produce a PWM waveform of a given period and specified duty cycle.

2. A low-pass filter to convert the PWM wave to an analog voltage level output.

A PWM coupled with a low-pass filter can be used as a simple, low cost digital to analog converter (DAC). This output can be used to drive to a voltage controlled device, or used in a feedback control system where an analog-to-digital convertor (ADC) is used to sample a controlled parameter. PWM's are often used in motor control applications.

Implementation software and hardware is discussed in this application note. An example of a PWM using an on-chip timer and a low-pass filter on the C8051F226-TB target board is provided. The example also configures the target board to sample the PWM output using the on-chip ADC. This DAC implementation may be used to evaluate the C8051F220/1/6's ADC.

Key Points

- The C8051F2xx family SoC's feature three on-board 16-bit timers that can be used for PWM generation. This example uses Timer 0 to produce the PWM wave which is output to a general-purpose port pin.
- The C8051F2xx family of SoC's have an 8-bit ADC that is used in the provided example to sample the output of the PWM DAC.
- The C8051F226-TB target board features a low-pass filter that can readily be used for the PWM DAC and configured to be sampled by the on-chip ADC without soldering or adding extra wiring. Target board use is assumed in the provided example.

Generating the PWM Input Waveform

Pulse-Width Modulation (PWM) is a method of encoding data by varying the width of a pulse or changing the duty cycle of a periodic waveform. Adjusting the duty cycle of this waveform, we control the voltage output from the low-pass filter. This can be thought of as a type of digital-to-analog convertor (DAC). In this example, we use Timer 0 to time the toggling of a general purpose port pin to create the PWM waveform.

Configuring Timer 0

In order to create a PWM wave with a user specified duty cycle, we use Timer 0 in 16-bit counter/timer mode. To do so, we configure the Timer

Copyright © 2003 by Silicon Laboratories

Mode register (TMOD), and the Clock Control register (CKCON), to set Timer 0 to use the system clock (undivided) as follows:

```
;Set TIMER0 in 16-bit counter ;mode

orl   TMOD,#01h


;Set TIMER0 to use system clk/1

orl   CKCON,#08h
```

Timer 0 is used to set the amount of time the PWM wave will be high during one cycle. When the timer overflows, the program vectors to an interrupt service routine (ISR) to take a port pin high or low to produce the PWM wave. We enable the Timer 0 interrupts by setting the ET0 bit to 1 as follows:

```
;Enable Timer 0 interrupts

setb  ET0
```

Additionally, interrupts must be enabled globally:

```
;enable interrupts globally

setb  EA
```

The last step in configuring Timer 0 is to start the timer by setting the TR0 bit:

```
;start Timer0

setb  TR0
```

A variable called *pulse_width* defines the duty cycle of the PWM wave. This determines the amount of time the waveform is high during one period of the wave, and is loaded into Timer 0. The duty cycle can be set with 16-bit resolution. However, due to the number of cycles it takes to execute the Timer 0 interrupt service routine (to be discussed later), the smallest pulse width that can be assigned is 19 clock cycles. Likewise, the interrupt service routine takes 14 cycles to take the PWM wave from high to low. Thus, the maximum value that can be used is 65,522. The variable *pulse_width* is defined as follows:

```
;define variable for user to

;set duty cycle of PWM wave

;input to the low-pass filter


pulse_widthEQU 35000d
```

Note the example code sets *pulse_width* equal to 35,000. As an example, 35,000 will create a duty cycle of 53.4%. Duty cycle is calculated as follows:

$$\text{dutycycle\%} = \frac{\text{pulsewidth}}{65,536} \times 100$$

Equation 1. Calculating Duty Cycle

The duty cycle also describes the average time that the waveform is high. This time will be converted into a voltage in the low-pass filter. The average output voltage for a given *pulse_width* value is calculated as follows:

$$\text{Voutput} = \text{VDD} \times \frac{\text{pulsewidth}}{65,536}$$

Equation 2. Calculating Average Output Voltage

Hardware Configuration

Port pin P2.7 will be used for the PWM waveform output to the PWM filter. We configure P2.7 as 'push-pull' by setting the Port 2 Configuration Register (PRT2CF):

```
;Set p2.7 as push-pull

orl   PRT2CF, #80h
```

Additionally, if using Silicon Lab's C8051F226-TB target board, a shorting jumper must be placed on the "PWMIN" jumper in order to connect port pin P2.7 to the low-pass filter.

Waiting For Interrupts

The Timer 0 ISR (Timer 0 overflow interrupt service routine) is used to generate the PWM wave by toggling the port pin P2.7. After programming the various peripherals, one may use a simple jump to the current address instruction in a loop to wait for interrupts, which is most common. However, the ISR is being used to generate a PWM waveform, and there will be a small amount undesirable of timing jitter caused by the small variation in delay due to interrupt latency. This variation occurs because the C8051 completes the current instruction before branching to the interrupt service vector. Thus, the time to branch to the ISR will vary depending on where in the 2-cycle jump instruction the MCU is when the interrupt condition occurs. To avoid this, we make use of the C8051 MCU IDLE Mode. The MCU will automatically "wake up" from IDLE Mode when an enabled interrupt occurs. This removes variations in interrupt latency because the core is always in the same state when an interrupt occurs. Note that all peripherals (such as timers) continue to operate when in IDLE Mode.

Setting the Idle Mode Select bit in the Power Control Register (PCON) places the C8051 in IDLE Mode. A jump statement is used to send the program counter back to the instruction to set the IDLE mode upon a return from an interrupt:

```
;Wait for interrupts in IDLE

;mode

IDLE:

      orlPCON,#01h

      sjmpIDLE
```

Upon a return from an ISR (*reti* instruction), the MCU will jump back to the *sjmp* instruction. Here, the program will loop back to set the IDLE Mode bit and wait for the next interrupt condition to occur.

Generating the PWM Wave in Software with Timer 0 ISR

The PWM wave is produced by toggling a port pin in an interrupt service routine (ISR). This ISR is a state machine with two states. In one state, the output pin is high (the high part of the PWM waveform). In this state, Timer 0 is loaded with the value *pulse_width* and the MCU exits the ISR. Next, the port pin is taken 'low' by clearing the bit P2.7. In the low state, the value *-pulse_width* is loaded. This sets the low time of the PWM waveform. At the next overflow, bit P2.7 is tested and then set to go to the high part of the waveform for the next period. In this way, the duty cycle can be varied but the period of the PWM wave will be the same.

The Timer 0 ISR is written as follows:

```
TIMER0_ISR:

;Test to see if low/high in ;wave-
form

jbc   P2.7,LO

setb  P2.7


; Set the low time of the

; PWM waveform

; Stop Timer 0 prior to load

clr   TR0

mov   TH0,#HIGH(-

      pulse_width)
```

```
mov   TL0,#LOW(-pulse_width)

; Restart Timer 0

setb  TR0

;Go to the reti statement

jmp   RETURN


;Set low time of PWM Wave

LO:

; Stop Timer 0

clr   TR0

mov   TH0,#HIGH(pulse_width)

mov   TL0,#LOW(pulse_width)

; Restart Timer 0

setb  TR0


;Return to MAIN and wait for

;interrupt

RETURN:reti
```

The Low-Pass Filter

The PWM wave generated with specified duty cycle is input into a low-pass filter. This filter will remove most of the high frequency components of the PWM wave. In terms of the time domain, the RC circuit will be charged to a voltage level proportional to the percentage of the period that the PWM wave input is positive (duty cycle). In short, the low-pass filter converts the set high *time* of the PWM wave to a *voltage* at the output of the system. Because the system inputs a digital number and outputs a desired voltage, the PWM and low-pass filter may be considered a form of digital-to-analog convertor (DAC).

In our example, we use a single-pole RC filter installed on the C8051F226-TB target board by placing a shorting jumper on the two pin jumper labeled "PWMIN". The filter used is shown in Figure 1..

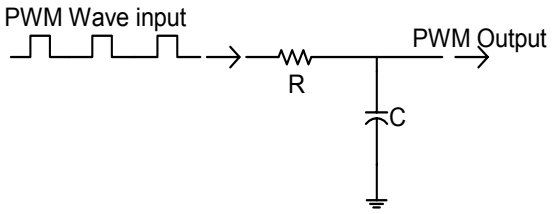

Figure 1. Low-Pass Filter

The filter in Figure 1 is a simple single pole filter. Its transfer function is:

$$\frac{Vout(s)}{Vin(s)} = \frac{\omega_c}{s + \omega_c}, \left(\omega_c = \frac{1}{RC}\right)$$

Equation 3. RC Filter Transfer Function

The RC filter must have a relatively low cutoff frequency in order to remove enough high frequency components of the wave to give a relatively constant DC voltage level. However, if the RC constant is too large, it will take too long for the RC voltage to rise to a constant level (i.e., long settling time.) This trade off can be easily tested in a computer model or a lab to choose good resistor/capacitor values.

This filter has only a single pole and so does not filter out all of the high frequency components of the rectangular PWM waveform. The capacitor is undergoing alternating cycles of charge and discharge, so the output will not be a constant DC voltage. (See Figure 2 below.) The output voltage will have some "ripple" (V*ripple* in Figure 2) associated with the filter's time constant τ=RC. In the frequency domain, the voltage ripple can be thought of as the relationship between the filter's

cutoff frequency (ω=1/RC) and the frequency of the PWM wave.

When designing the low-pass filter, it may be important to predict, or characterize the deviation from the desired constant, DC voltage output. We refer to this as voltage *ripple* (V*ripple*). In order to characterize the V*ripple*, we use the formulae that describes the voltage of a capacitor in an RC circuit.

Figure 2 illustrates the input PWM wave and the resulting low-pass filter output. The output wave is exaggerated to show the alternating charge and discharge of the capacitor in the RC circuit. The ripple for a 50% duty cycle (worst case ripple) for this filter is calculated by using the following expression given R,C, and the period of the PWM wave, *T*:

$$Vripple = VDD\left(1 - \frac{2e^{-\frac{T}{2\tau}}}{1 + e^{-\frac{T}{2\tau}}}\right), \tau = RC$$

Equation 4. Voltage Ripple In Filter Circuit

Equation 4 is derived using the formulae that describe the voltage of a capacitor in an RC circuit and by taking advantage of the symmetry of the PWM waveform as a square wave (i.e., 50% duty cycle). Note that the worst case ripple is determined by both the frequency (f=1/T), and the RC time constant (τ). This makes sense, as the RC combination determines the cutoff frequency of the low-pass filter, and with respect to the PWM wave frequency this will characterize how much of the high frequency components will be filtered from the rectangular PWM waveform.

The RC circuit on the target board uses a 220 kΩ resistor and a 0.47 µF capacitor. These values were chosen to show a relatively constant voltage level with 8-bit ADC sampling and still have a reasonable settling time.

If the ideal output is a constant DC voltage, then the ripple in the output voltage can be considered as the error. To calculate this error when designing the filter (or to evaluate using a simple RC filter), we must know the frequency of the PWM wave, and the time constant (τ). Using the RC values on the target board, τ=RC=0.1034 seconds. If the 16-bit timer is running with system clock speed of 16 MHz, the PWM period in this example is:

$$T = \frac{2^{16}}{sysclk} = \frac{65,536}{16\times10^{6}} \approx 4ms$$

In this example, the predicted V*ripple* is calculated to be 200 mV using Equation 4.

Sampling the PWM Output With the On-Chip ADC

The C8051F226-TB target board includes a C8051F226 SoC that features an 8-bit analog-to-

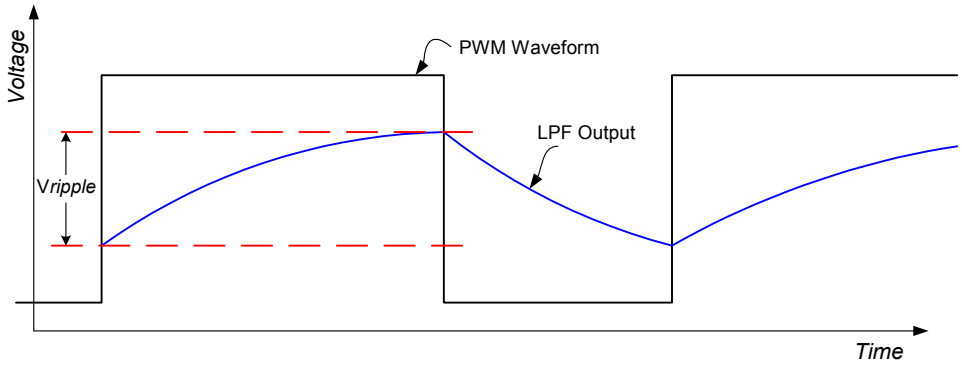

Figure 2. PWM Waveform and Filter Output

digital convertor (ADC). In this example, we wish to sample the output voltage with the ADC. Alternatively, the output can also be measured using a voltmeter at the test point labeled "PWM" on the target board. To use the ADC we must configure a port for ADC input and program the ADC to sample at a desired rate to measure the PWM output.

Configuring the ADC

The C8051F2xx family of devices can use any general purpose port pin as an input for analog signals. The AMX0SL register configures the ADC's multiplexer (AMUX) to select which port pin will be the input to the ADC. The target board used in this example provides a circuit for easily placing the PWM output to port pin P3.0, which is configured as the ADC input as follows:

```
;enable AMUX and configure for
;P3.0 as an input port pin

mov   AMX0SL,#38h
```

The ADC0CF configuration register sets the SAR conversion clock based on the system clock, and sets the programmable gain amplifier (PGA) gain. The maximum frequency the SAR clock should be set to is 2 MHz. The system clock is operating at 16 MHz, thus, the SAR conversion clock is set to 1/8 of the system clock frequency (i.e., SAR conversion clock = sysclk/8). We also program the PGA for a gain of one as follows:

```
;set conv clk at one sys clk and
;PGA at gain = 1

mov   ADC0CF, #60h
```

ADC0CN is the ADC control register. This register is set to configure the ADC to start conversions upon a Timer 2 overflow and set the ADC to low power tracking mode (tracking starts with Timer 2 overflow):

```
; SAR clock = SYSCLK/8
; PGA gain = 1

;Timer 2 overflow

mov   ADC0CN, #01001100b
```

Finally, we enable the ADC. This bit is located in the ADC0CN register which is bit addressable, and so we use *setb*:

```
;enable ADC

setb  ADCEN
```

In this example, we use the VDD voltage supply as the ADC voltage reference. This is set in the REF0CN register:

```
;set ADC to use VDD as Vref

mov   REF0CN, #03h
```

Before we can use Timer 2 overflows to initiate ADC conversions, we must configure and start Timer 2. We place a value called *ADCsampl* in Timer 2 to initialize its operation, and place the same value into the Timer 2 Capture registers, RCAP2H:RCAP2L, so that it will overflow at the desired sampling frequency. Timer 2 has an auto-reload feature making this convenient. A sampling frequency that is independent of PWM wave frequency is desirable because the output of the filter will have a periodic variation in the DC level because the filter is not ideal (charging and discharging of our capacitor causing V*ripple*.) Sampling at a different frequency will allow us to observe the voltage ripple with the ADC. In this example, we use a sampling frequency of 1.6 kHz.

Configuring Timer 2:

```
;initialize T2 for ADC sampling
;rate of 1.6 kHz with 16 MHz
;sysclk

mov   TL2,#LOW(ADCsampl)
```

```
mov   TH2,#HIGH(ADCsampl)

;Load autoreload values for ;sam-
pling rate of ADC

mov   RCAP2L,#HIGH(ADCsampl)

mov   RCAP2H,#HIGH(ADCsampl)


;Set Timer 2 to use sysclk/1

orl   CKCON, #20h


;start Timer 2

setb  TR2
```

We must enable ADC end of conversion interrupts so we can process ADC samples. To enable ADC interrupts, we configure the Extended Interrupt Enable 2 register (EIE2):

```
;enable ADC interrupts

orl   EIE2,#00000010b
```

The ADC is now configured for sampling an input from P3.0 using Timer 2 to set the sampling frequency. All that is required now is to configure the port pin for analog use described in the following section, and connect it to the low-pass filter output.

Configuring the Port For the ADC

The ADC has been configured to input analog from P3.0. We now must configure the port for analog input use.

The port pins default to *digital input* mode upon reset. We place port pin P3.0 in *analog input* mode by configuring the Port 3 Digital/Analog Port Mode register, P3MODE:

```
;Set p3.0 in analog input mode

orl   P3MODE, #01h
```

Note that we must physically connect the PWM output to the ADC input. One could solder a wire or design a PCB to provide this connection. The target board in this example conveniently provides headers that allow easy configuration using shorting jumpers to connect the provided PWM low-pass filter to port pin P3.0. No soldering or external wiring is necessary for this demonstration.

To configure external circuitry to input the PWM output to port pin P3.0 (set for ADC input), place a shorting jumper onto header J6, connecting "PWM" pin to "P3.0AIN". P3.0AIN is connected to the P3.0 port pin on the device.

The ADC Interrupt Service Routine

The ADC interrupt service routine's only function in our example is to clear the ADC interrupt flag, the ADCINT bit. This flag must be cleared in software, and we do so as follows:

```
ADC_ISR:

clr ADCINT

reti  ;return from interrupt
```

The ADC ISR is a convenient place to read the sampled data from the ADC data registers and process the data. This example leaves the data in the word register (ADC0H) and will be overwritten with each new sample. This data may be observed by using Silicon Lab's *Integrated Development Environment* (IDE) tool to view the special function register, ADC0H which holds the ADC conversion results.

Interpreting the Results

The PWM outputs a voltage level corresponding to the *pulse_width* variable which determines the

PWM wave duty cycle. As aforementioned, the voltage level output can be calculated using Equation 2 on page 3.

VDD refers to the supply voltage of the device. The number 65,536 is the highest number that can be represented in 16 bits (as our PWM timer is a 16 bit counter/timer). *Voutput* is the value one would measure at the output of the PWM's low-pass filter. Note that due to the number of cycles is takes to execute the Timer 0 ISR, the minimum number that can be effectively used as the *pulse_width* is 19. Thus, the lowest *Voutput* that can be generated is 0.028% of *VDD*. Any number used for *pulse_width* less than 19 will yield the same result as entering 19. Similarly, it takes 14 cycles for the Timer 0 ISR to process the falling edge of the PWM waveform. Thus, the maximum effective *pulse_width* is 65,522 (65,536-14). Therefore, the resulting output will be 99.98% of *VDD*. There are no other limitations due to software inside of the 0.028%-99.98% range other than the quantization imposed by 16-bit timer resolution. If, for example, *VDD*=3.0V, then the voltage resolution will be 46 μV with code and the range of the output voltage values is 0.87 mV to 2.9994 V.

In our example, we measure the PWM output with the on-chip ADC. The result in the ADC register (ADC0H) will be a number between 0 and 255 (8-bit ADC). This example uses VDD as the reference for the ADC conversion. The ADC output number can be interpreted as follows:

$$Vresult = VDD \times \frac{ADC0H}{256}$$

Note that *Vresult* may not match the ideal *Voutput* calculated as output from the PWM. This is due to the aforementioned V*ripple* (see section, "*The Low-Pass Filter"*).

Software

```
;Copyright 2003 Cygnal, Inc.
;Implementing an 16-bit PWM on SA_TB4PCB-002 target board and sampling to test
;  the 8-bit analog-to-digital convertor (ADC). The following program will
;  configure on-chip peripherals and use a low-pass filter on the target board.
;
;FILE:    PWM_200.asm
;DEVICE:  C8051F2xx
;TOOL:    Cygnal IDE, 8051 assembler (Metalink)
;AUTHOR:  LS
;-----------------------------------------------------------------------
$MOD8F200
;-----------------------------------------------------------------------
;
;Reset Vector
;
   org    00h
   jmp    MAIN
;
;-----------------------------------------------------------------------
;
;ISR Vectors
   org    0Bh
   jmp    TIMER0_ISR


   org    7Bh
   jmp    ADC_ISR

;-----------------------------------------------------------------------
;CONSTANTS
pulse_width   EQU          35000d             ; Value to load into TIMER0 which
                                              ; adjusts
                                              ; pulse width (duty cycle)
                                              ; in PWM and thus sets the
                                              ; DC bias level output from the
                                              ; low-pass
                                              ; filter. Set from 19-65522d.
                                              ; 32768 = VDD/2
ADCsampl      EQU          55536d             ; Load into TIMER2 for ADC sampling rate



;-Start of MAIN code-------------------------------------------------


       org 0B3h

MAIN:
       mov    OSCICN,#07h                     ; Configure internal OSC for 15MHz
       mov    WDTCN,#0DEh
       mov    WDTCN,#0ADh
       mov    P3MODE,#0FEh                    ; Configure P3.0 for analog input
       orl    PRT2CF,#80h                     ; Configure P2.7 as push-pull input to   ;
low-pass filter
       orl    CKCON,#28h                      ; Set TIMER0 and TIMER2 to use SYSCLK/1
```

```
       mov    TMOD,#01h                        ; Set TIMER0 in 16-bit counter mode
       mov    RCAP2L,#LOW(ADCsampl)            ; Load autoreload values for sampling
                                               ; rate of ADC
       mov    RCAP2H,#HIGH(ADCsampl)           ; using TIMER2 overflow for ADC
                                               ; conversion start
       mov    TL2,#LOW(ADCsampl)               ; initialize T2 for ADC sampling
                                               ; rate=1.6KHz
       mov    TH2,#HIGH(ADCsampl)
       mov    AMX0SL,#38h                      ; Set AMUX for P3.0 input/Enable AMUX
       mov    ADC0CF,#60h                      ; SAR clock = SYSCLK/8, and GAIN = 1
       mov    ADC0CN,#00001100b                ; Set the ADC to start a conversion on
                                               ; Timer2 overflow
       orl    REF0CN,#03h                      ; Set to the internal reference
       orl    EIE2,#00000010b                  ; Enable ADC end of conv. interrupts
       setb   ET0                              ; Enable timer0 interrupts
       setb   EA                               ; Global interrupt enable
       setb   TR0                              ; Start TIMER0
       setb   TR2                              ; Start TIMER2
       setb   ADCEN                            ; Enable the ADC
IDLE:
       orl    PCON,#01h                        ; BWCLD
       sjmp   IDLE


;------TIMER0 ISR----------------------------------------------------------
TIMER0_ISR:
       jbc    P2.7,LO                          ; Test to see if low/high in waveform
       setb   P2.7                             ; Transition low to high
       clr    TR0                              ; Stop Timer 0 during reload
       mov    TL0,#LOW(-pulse_width)           ; Set length of pulse for DC bias level
       mov    TH0,#HIGH(-pulse_width)          ;
       setb   TR0                              ; Restart Timer 0
       jmp    RETURN
LO:    clr    TR0                              ; Stop Timer 0 for reload
       mov    TL0,#LOW(pulse_width)            ; Set low time of duty cycle
       mov    TH0,#HIGH(pulse_width)
       setb   TR0                              ; Restart Timer 0
RETURN:reti

;------ADC ISR-------------------------------------------------------------
ADC_ISR:
       clr    ADCINT                           ; flag must be cleared in software
       reti


;--------------------------------------------------------------------------


;End of program
;All your base are belong to us.
END
```


SILICON LABORATORIES

Notes:

Contact Information

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032
Email: productinfo@silabs.com
Internet: www.silabs.com

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories 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. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.

Silicon Laboratories and Silicon Labs are trademarks of Silicon Laboratories Inc.

Other products or brandnames mentioned herein are trademarks or registered trademarks of their respective holders.