Objectives

- To explore the Enhanced Capture Timer unit (ECT) of the MC9S12DP256B/C
- To program a real-time clock signal with a fixed period and display it using the on-board LEDs (flashing light)
- To produce a timing engine for a digital control application with a fixed sample rate of 1 kHz

Introduction

Timers, counters as well as capture-and-compare mechanisms are fundamental to many microcontroller based mechatronics applications. For example, the drive system of a mobile robot often consists of a pair of DC-motors driven by Pulse Width Modulated (PWM) signals. Many microcontrollers therefore support the generation of these kinds of signals by incorporating specialist PWM units. The MC9S12DP256B/C includes an 8-channel PWM unit (Figure LM4-1).

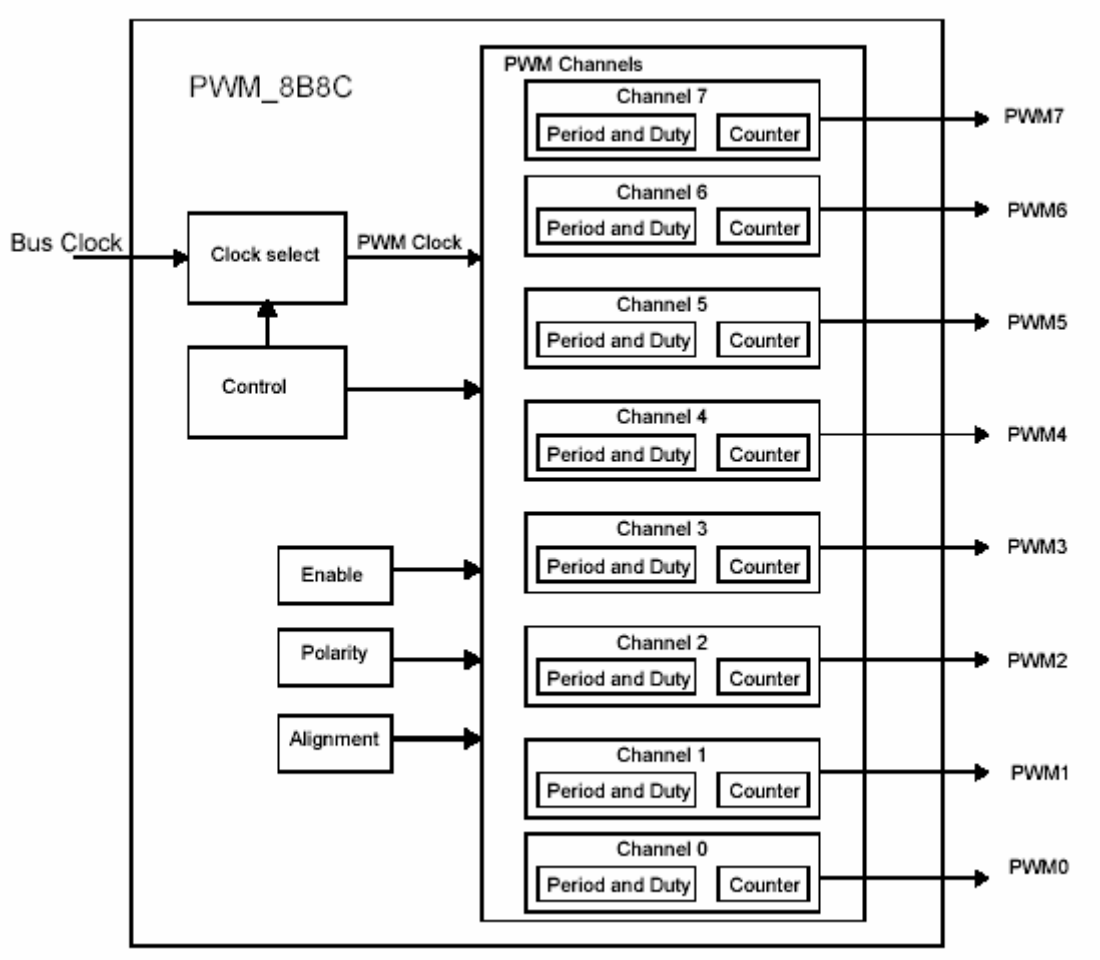

Figure LM4-1 The PWM unit of the MC9S12DP256B/C

The operating principle of such a PWM unit is rather simple: A counter register is incremented with every falling edge (or rising edge or both) of the system bus clock signal. Upon reaching a first threshold an associated output pin is set high (or low)

and/or an interrupt is triggered. This threshold defines the *duty cycle* of the PWM signal (cf. Figure LM4-2). The counter register continues to be incremented until a second threshold is reached. This second threshold defines the period of the PWM signal. Upon reaching the period threshold, the associated pin is toggled again and the counter register is reset to zero. Altogether, this produces a square wave signal of a predefined period and with a programmable on/off phase (i. e. a *PWM signal*).

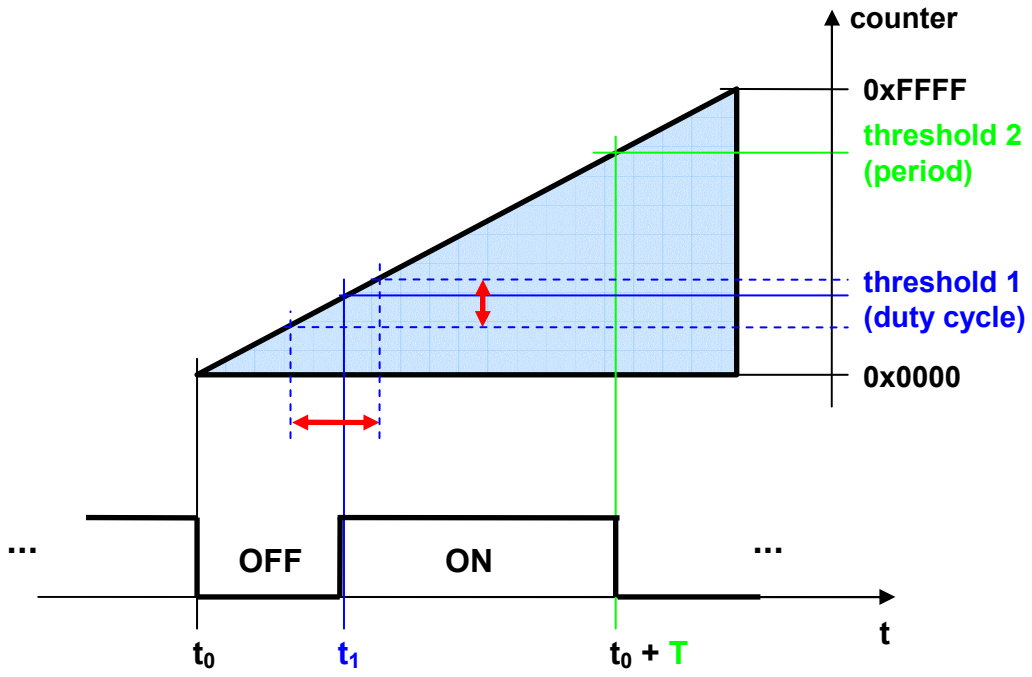

Figure LM4-2 Generating PWM signals

In this exercise we are going to program an internal timing signal with a fixed period. This is very similar to setting up a PWM unit. Most of the above described operating principle remains valid with the exception that we won't bother with setting up a duty cycle – all we are interested in at this stage is the period of our timer signal. We will thus have to configure the Enhanced Capture Timer unit (ECT) to run at a fixed rate and to trigger an interrupt every time a pre-programmed counter value (period) is reached.

To test our timer, we will install a simple *interrupt service routine* in which we will toggle one of the digital I/O pins. This produces a square wave signal with a period of two timer periods (one timer period for the on-phase, one timer period for the off-phase → two timer periods). The correct shape and period of this square wave signal will be verified using an oscilloscope.

The Enhanced Capture Timer (ECT) unit

Open the user guide of the Enhanced Capture Timer unit *(ECT_18B8C.pdf)* and familiarize yourself with the block diagram of this unit (Figure LM4-3). The basic timer consists of a 16-bit software-programmable up-counter, driven by a pre-scaled clock signal. In *normal mode* the timer keeps running until it is switched off explicitly. There also exists a *modulus down counter*. This type of counter decrements an associated counter register until it reaches 0x0000 (underflow), whereupon the initial starting value

is reloaded. The modulus down counter is therefore well-suited to the generation of fixed-period time base signals.

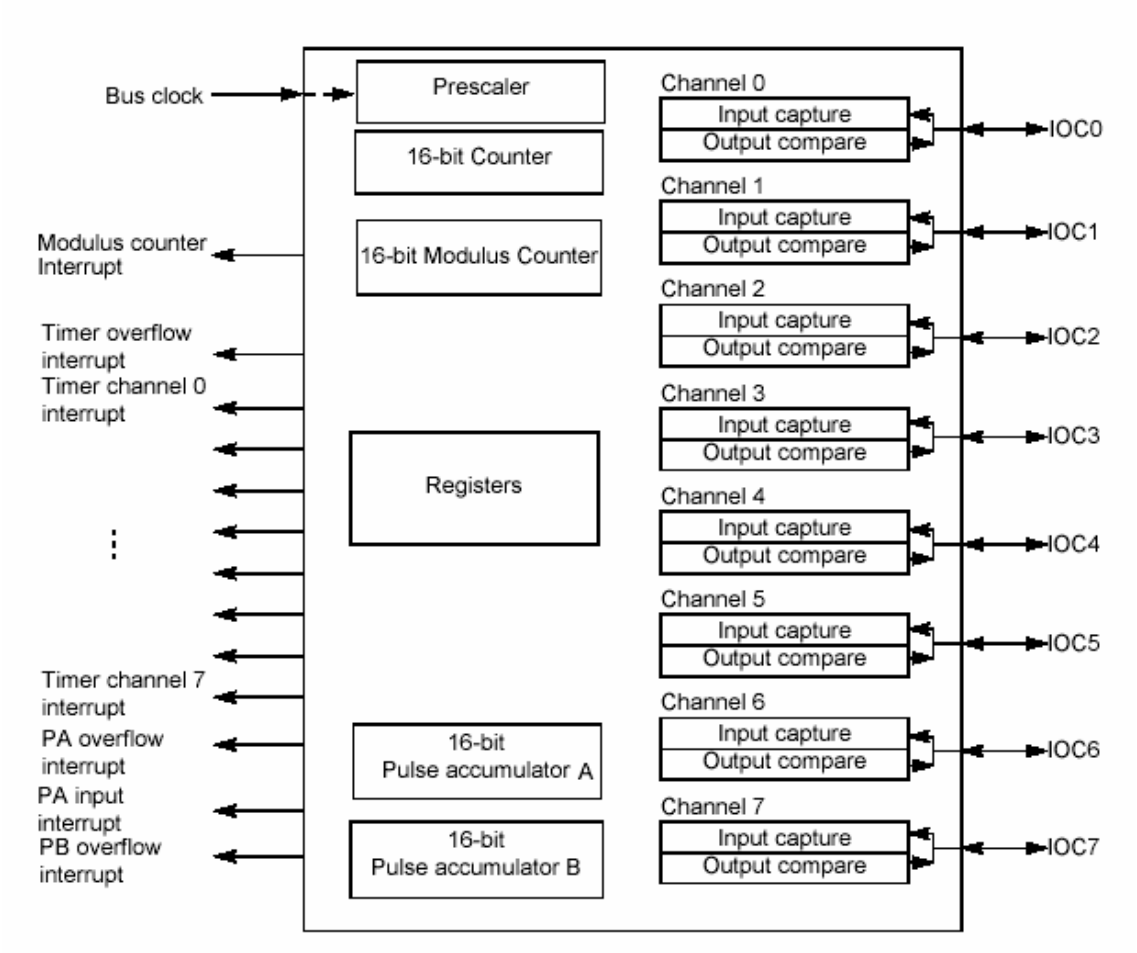

Figure LM4-3 Block diagram of the ECT unit of the MC9S12DP256B/C

One constraint of any timer application is the maximum period which can be achieved. This depends on two factors, namely the size of the counter register (here: 16 bit) as well as the maximum pre-scale divider. The latter is used to slow down the commonly very high bus clock frequency (here: 24 MHz). The slower the effective timer clock frequency, the longer periods we can realise. Figure LM4-4 shows the prescaler section of the ECT unit. It seems that the main timer can be run at a minimum timer clock frequency of $f_{bus}/128 = (24 \cdot 10^6)/128$ Hz = 187.5 kHz. The frequency of the modulus down counter, on the other side, can only be slowed down to $f_{bus}/16 = (24 \cdot 10^6)/128$ Hz = 1.5 MHz. This means that the main timer can be used for periods of up to 0xFFFF x $1/(187.5 \cdot 10^6)$ = 65535 x $1/(187.5 \cdot 10^6) \approx 0.35$ seconds, whereas the modulus down counter only reaches 65535 x $1/(1.5 \cdot 10^6) \approx 43.7$ ms.

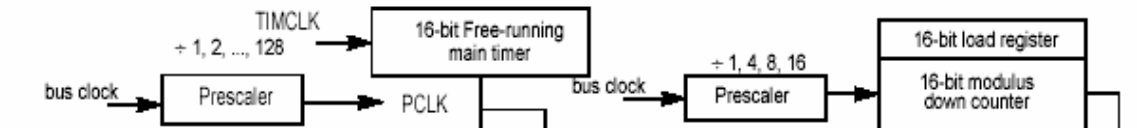

Figure LM4-4 Prescaler section of the ECT

Table LM4-1 lists all achievable periods for both the main timer as well as the 16-bit modulus down counter.

	Main timer		Modulus down timer	
Prescaler	**Resolution**	**Maximum period**	**Resolution**	**Maximum period**
1	42 ns	2.73 ms	42 ns	2.73 ms
2	84 ns	5.46 ms	84 ns	5.46 ms
4	167 ns	10.9 ms	167 ns	10.9 ms
8	333 ns	21.8 ms	333 ns	21.8 ms
16	667 ns	43.7 ms	667 ns	43.7 ms
32	1.33 μs	87.4 ms	-	-
64	2.67 μs	174.8 ms	-	-
128	5.33 μs	349.5 ms	-	-

Table LM4-1 Programmable periods of the ECT

As we would like to test our timer by flashing the on-board LED (PB0) we should try to make it run as slowly as possible. The maximum achievable period is 349.5 ms (circa 1/3 of a second). This should be sufficiently slow to be able to discern a distinct on-phase followed by an equally long off-phase. We therefore choose to program the *main timer* to a period of 349.5 ms (counter value 0xFFFF). For a general function description of the ECT unit see chapter 4 of the corresponding user guide.

Note: Even slower signals can be produced when reducing the bus clock frequency (default: 24 MHz). This can be done by reprogramming the phase locked loop (PLL) circuit within the *Clock and Reset Generator (CRG)* unit of the microcontroller. However, a slower running system will also need more time to evaluate an expression such as a transfer function or a filter equation.

Implementation

To implement a fixed period time base on the MC9S12DP256B/C we can make use of the automatic reload mechanism of timer *channel 7 in output compare* mode. To configure the ECT for this mode of operation we will have to set-up the following registers:

(1) *TIOS – Timer Input Capture / Output Compare Setup register*. This register determines whether a timer channel is used in input capture or output compare mode. Set-up channel 7 for output compare mode.

(2) As we don't want to affect the output pins associated with timer channel 7 we will have to ensure that the output logic is switched off. Have a look at the *Timer Control* registers *(TCTL1, TCTL2)* as well as at the *Output Compare 7 Mask* register *(OC7M)* to find out how this can be done. Note that, depending on the reset value of a register, you may or may not have to modify anything.

(3) The prescaler value will have to be programmed *(Timer System Control Register 2, TSCR2)*. This register also allows you to enable the automatic resetting of the main timer register when a match occurs between the current value of the main timer register and the value programmed the timer register of in channel 7. Look for a bit called *TCRE*.

(4) The *Timer Input Capture / Output Compare register of channel 7 (TC7)* will have to be initialised to its maximum value of 0xFFFF.

(5) The *Timer Interrupt Enable Register (TIE)* will certainly have to be initialised so that our interrupt service routine can be called.

(6) Finally, the timer needs to be switched on by setting the *Timer Enable Bit (TEN)* in the *Timer System Control Register 1 (TSCR1).*

Create an empty project based on stationery *Dragon12_flat*. Create the new source file *timer.c* and add it to project group *Sources*. Create a corresponding header file *timer.h* and add this to the same group. To save you some time, I've uploaded a skeleton project onto myUni:

Mechatronics IIIM → Course Documents → Tutorials → 9S12 → timer_interrupt

Write a function *TOC7_Init* which sets up the ECT timer channel 7 for output compare mode.

Write an interrupt service routine for channel 7 timer interrupt *C7I*. This ISR should toggle bit 0 of port B (connected to the on-board LED of the Dragon12). Don't forget to acknowledge the interrupt by clearing the appropriate flag in the *Main Timer Interrupt Flag* register *(TFLG1)*.

Install your interrupt service routine in the interrupt vector table *(isr_vectors.c)*.

Your *main* program should set-up port B as output (configure DDRB), call upon the initialisation routine TOC7_Init allow all interrupts to happen *(`asm cli`)* and finally enter an infinite loop in which it does nothing in particular (`for(;;);` or `while(1);`).

Download and debug your program using the Hi-Wave debugger.

Extension

Modify your program to produce a square wave with a period of 1 kHz and a 50% duty cycle (0.5 ms on, 0.5 ms off). The signal should be made available on pin 0 of port A. You will have to choose a suitable prescaler value (see Table LM4-1) as well as the corresponding threshold value to be written into the *Timer Input Capture / Output Compare register of channel 7 (TC7)*.

Verify your 1 kHz timing engine using an oscilloscope.

Can you see how the techniques introduced in this laboratory session can be used to implement a digital control system?