Microcontroller Programming II | MP6-1

<table>
<thead>
<tr>
<th>week</th>
<th>lecture</th>
<th>topics</th>
</tr>
</thead>
</table>
| 6    | Microcontroller programming II | - General purpose timers  
- Capture and compare unit  
- PWM unit |

Microcontroller Programming II | MP6-2

General Purpose Timers

- One of the most important issues in embedded control is the programming of timers
- Without accurate timing, digital control engineering is not possible – the control signals (controller action) have to happen at the exact right moment in time, e.g. timing control of an engine, etc.
- Large embedded systems are often centred around small real-time kernels which offer services for multi-tasking applications; smaller applications make use of Interrupt Service Routines (ISR) to achieve accurate timing control

Microcontroller Programming II | MP6-3

General Purpose Timers

- The General Purpose Timer unit of the C167CR (GPT1 – there are 2 such units) is controlled through a number of Special Function Registers (SFR)

<table>
<thead>
<tr>
<th>Port 5 Direction Control</th>
<th>Data Registers</th>
<th>Control Registers</th>
<th>Interrupt Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP3</td>
<td>T2</td>
<td>T2CON</td>
<td>T2IC</td>
</tr>
<tr>
<td>PDP5</td>
<td>T3</td>
<td>T3CON</td>
<td>T3IC</td>
</tr>
<tr>
<td>P3</td>
<td>T4</td>
<td>T4CON</td>
<td>T4IC</td>
</tr>
<tr>
<td>P5</td>
<td>PIDD1S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port 5 Data Register</th>
<th>Port 3 Direction Control Register</th>
<th>T2</th>
<th>GPT1 Timer 2 Register</th>
<th>T2IC</th>
<th>GPT1 Timer 2 Interrupt Control Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP5</td>
<td>PDP9</td>
<td>T2</td>
<td>GPT1 Timer 2 Register</td>
<td>T2IC</td>
<td>GPT1 Timer 2 Interrupt Control Register</td>
</tr>
<tr>
<td>PDP3</td>
<td>Port 3 Direction Control Register</td>
<td>T2</td>
<td>GPT1 Timer 2 Register</td>
<td>T2IC</td>
<td>GPT1 Timer 2 Interrupt Control Register</td>
</tr>
<tr>
<td>PDP9</td>
<td>Port 2 Direction Control Register</td>
<td>T2</td>
<td>GPT1 Timer 2 Register</td>
<td>T2IC</td>
<td>GPT1 Timer 2 Interrupt Control Register</td>
</tr>
</tbody>
</table>

Microcontroller Programming II | MP6-4

General Purpose Timers

- GPT1 consists of three 16-bit timers (T2, T3, T4)

Each of these timers can be configured as...

... timer,  
... gated timer,  
... counter  
and  
... incremental interface mode
General Purpose Timers

- In **timer mode**, the timer register is incremented with every tick of the internal clock source.
- In **gated timer mode**, the (internally incremented) timer can be switched on/off via an external signal.
- In **counter mode**, the counter register is incremented with rising/falling/both edges of an external signal.
- The **incremental interface mode** supports interfacing to incremental encoders; both rate and direction are determined and the counter register is incremented / decremented accordingly.

- Timers are probably best explained using the following diagram:

- The timer register is loaded with an initial value; it is decremented at a fixed rate – the timer elapses when the timer register reaches 0.

General Purpose Timers

- Core timer T3 is configured for **timer mode** through its control register T3CON.

<table>
<thead>
<tr>
<th>Timer 3 Control Register</th>
<th>SFR (FF40/A1H)</th>
<th>Reset value: 0000H</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>- - - - - - T3OE T3OE T3OE T3UD T3R T3M T3I</td>
<td>- - - - - - - - - - - -</td>
</tr>
</tbody>
</table>

- Setting T3M (mode) to binary 000 puts T3 into timer mode; field T3I then controls the frequency with which the timer register is updated; T3UD specifies the direction of this update (up/down); T3UDE enables the external control of this direction.

- The timer clock frequency is generated by pre-scaling the CPU clock ($f_{CPU}$):

$$f_{T3} = \frac{f_{CPU}}{8 \cdot 2^{IV}}$$

<table>
<thead>
<tr>
<th>$f_{CPU}$ = 20 MHz</th>
<th>Timer Input Selection T2vT3/T4I</th>
</tr>
</thead>
<tbody>
<tr>
<td>000B 001B 010B 011B 100B 101B 110B 111B</td>
<td></td>
</tr>
<tr>
<td>Prescaler Factor</td>
<td>8 16 32 64 128 256 512 1024</td>
</tr>
<tr>
<td>Input Frequency</td>
<td>2.5 MHz 1.25 MHz 625 kHz 312.5 kHz 156.25 kHz 78.125 kHz 39.06 kHz 19.53 kHz</td>
</tr>
<tr>
<td>Resolution</td>
<td>400 ns 800 ns 1.6 µs 3.2 µs 6.4 µs 12.8 µs 25.6 µs 51.2 µs</td>
</tr>
<tr>
<td>Period</td>
<td>25.6 ms 51.2 ms 102.4 ms 204.8 ms 409.6 ms 819.2 ms 1.638 ms 3.276 ms</td>
</tr>
</tbody>
</table>
General Purpose Timers

- The timer is switched on or off using bit T3R in Special Function Register T3CON

- T3OE allows the state of the output toggle latch (T3OTL) to be displayed on pin T3OUT (= P3.3); T3OTL changes its state whenever the timer elapses

- An elapsed timer usually triggers an interrupt; on the C167, every timer comes with its own Interrupt Control register (TxIC)

- The T3IC register allows a priority value to be specified (group level, interrupt level); in the case of several simultaneous interrupts, this value is used to determine which interrupt should be serviced first

- Example:
  Toggle the LED connected to port P3.3 at a fixed rate of 0.25 seconds (ON : 250 ms, OFF : 250 ms)

- The table on slide MP6-8 indicates that the fastest rate, which allows timer T3 to have a period of 250 ms, is 156.25 kHz (T3I = binary 100)

- 250 ms represents the fraction of 250/420 ≈ 59.52 % of the maximum period (420 ms); the initial value of the timer register is therefore 65535 * 0.595 ≈ 39009
# General Purpose Timers

```c
#include <reg167.h>
#define T3_RELOAD 39009      /* T3 reload value : 250 ms */
#define T3_RELOAD 100      /* T3 reload value : 250 ms */

/* Timer T3 ISR */
void T3_elapsed(void)
  interrupt 0x23
{
  T3 = T3_RELOAD;           /* reset T3 */
  P2 ^= 0x0001;             /* toggle P2.1 */
}

/* main program */
void main(void)
{
  DP2 |= 0x0001;             /* P2.1 : output */
  P2  |= 0x0001;             /* initially: LED off */

  (...)  
  T3 = T3_RELOAD;            /* load initial timer value T3 */
  /* T3 in timer mode, counting down, pre-scale factor 128, period: 420 ms */
  /* alternate output function disabled */ /* T3CON = 0000.0000.1000.0100 = 0x0084 */
  T3CON = 0x0084; T3IC  = 0x0044;            /* enable T3 interrupt, ILVL = 1, GLVL = 0 */
  IEN = 1;                   /* allow all interrupts to happen */
  T3R = 1;                   /* start timer (T3CON |= 0x0040) */
  while(1);                  /* forever... */

  (...)  
}
```

Declares function T3_elapsed as Interrupt Service Routine (ISR) with interrupt vector number 0x23; this causes the compiler to install its starting address as entry 0x23 in the interrupt vector jump table.

Each time T3 underflows a T3 interrupt is triggered; the system then saves its current state and diverts program execution to this ISR, which has to reload the timer register to make the process cyclic.

Finally, all interrupts need to be permitted at CPU level (master switch: IEN = 1) and the timer must be started (T3R = 1); the former instruction sets bit ‘IEN’ in the Processor Status Word which, on the C167, is memory mapped in bit-addressable memory – it would have been possible to write this as PSW |= 0x0800.

Both T3CON and T3IC have to be set up to make this timer work; T3CON is configured for timer mode with a pre-scale factor of 128 (maximum period: 420 ms) and counting downwards; T3 interrupts are enabled and an interrupt level (ILVL) of ‘1’ is chosen (needs to be different from ‘0’).
General Purpose Timers

The state of T3 as well as the interrupt controller can be monitored through peripheral windows.

- Setting a conditional breakpoint ($T3 = 1 \rightarrow i.e.$ just before the timer elapses) allows the triggering of the interrupt to be observed.

- The code executes uninterrupted until the value in timer register T3 has been decremented to ‘1’.

- Single stepping through the code reveals that the interrupt request flag $T3IR$ comes on when the timer underruns.

- The timer T3 interrupt service routine is activated – on the C167, this automatically clears the interrupt request flag.

- Toggling a bit in C:
  
  ```
  P2 = P2 ^ 0x00ff;
  ```

  Logical XOR operator                           Mask

  - Example: P2 contains value 0x1234
    
    | P2         | 0001.0010.0111.1100 |
    | P2 ^= 0x00ff | 0000.0000.1111.1111 |
    | P2         | 0001.0010.1100.1011 |
    | P2 ^= 0x00ff | 0000.0000.1111.1111 |
    
    - The above line can be abbreviated as follows:
      
      ```
      P2 ^= 0x00ff;
      ```
Capture and Compare unit

- Capture and Compare units (CAPCOM) are similar to general purpose timers / counters in a sense they monitor internal / external events (e.g. a rising edge on an associated input line, etc.); once the specified number of events has been observed, they trigger an interrupt or directly modify the state of an output pin.

- These units are usually used for high-speed timing operations (e.g. waveform generation, etc.) with a minimum amount of software overhead; other controllers often offer similar mechanisms under slightly different names (e.g. Output Compare timer).

Capture function:
Triggered by an external event on the associated pin; causes the current timer contents to be latched into the associated register → used to measure durations.

Compare function:
Triggered by a match between the current contents of the timer and one of the CAPCOM registers; may cause a signal transition on the associated pin → signal generation.
Capture and Compare unit

- Timers T0/T7 and T1/T8 provide two independent high resolution time bases for the capture / compare registers of each unit.

The timers can operate of either of three clock sources:
- A pre-scaled CPU clock, underflows of GPT2 timer T6 or external events on an associated input.

- CAPCOM timers count upwards; when a timer overflows, it is reloaded with its respective reload value from TxREL.
- The period of the CAPCOM timer depends on this reload value:

\[
P_T = \frac{(2^{16} - TxREL) \cdot 2^{TSD+3}}{f_{CPU}}
\]

- The function of each CAPCOM timer is controlled by a Special Function Register (SFR): the lower half of T01CON controls timer T0, the upper half controls T1; a similar pair exists for T7 and T8.

- The purpose and meaning of each of these bit groups is similar to that of the General Purpose Timers.

Capture and Compare unit

- 32 capture/compare registers CCx are used to store the 16-bit values of a capture or compare operation; the association between a CCx register and any of the CAPCOM timers is detailed in the so-called capture/compare mode control registers (CCMx).

- The mode of each channel is defined by 4 bits.

CCM0 controls channels CC0 – CC3; CCM1 ... CC4 – CC7, etc.
Capture and Compare unit

- **Capture/compare operating modes (CCMODx)**

<table>
<thead>
<tr>
<th>CCMODx</th>
<th>Selected Operating Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Disable Capture and Compare Modes</td>
</tr>
<tr>
<td>0 1</td>
<td>Capture on Positive Transition (Rising Edge) at Pin CCxI0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>Capture on Negative Transition (Falling Edge) at Pin CCxI0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>Capture Mode 0: Interrupt Only</td>
</tr>
<tr>
<td>1 0 1</td>
<td>Compare Mode 1: Toggle Output Pin on each Match</td>
</tr>
<tr>
<td>1 1 0</td>
<td>Compare Mode 2: Interrupt Only</td>
</tr>
<tr>
<td>1 1 1</td>
<td>Compare Mode 3: Set Output Pin on each Match</td>
</tr>
</tbody>
</table>

Capture mode allows the contents of the selected timer to be captured in the associated CCx register when an external signal has a rising/falling edge; this mode is commonly used to measure durations.

- The associated I/O pin must be programmed as input.

Capture and Compare unit

- **Compare mode** allows interrupts to be triggered and/or I/O pins to be toggled when the selected timer matches the value of the associated CCx register; this mode is commonly used for waveform generation.

Example: Compare mode 0

Pulse sequence generated by triggering interrupts at user programmable times (controlled through the compare values cv1 and cv2); upon reaching level cv1, the CCx register is modified to cv2, upon reaching cv2 CCx is reset to cv1.
Capture and Compare unit

```c
#include <reg167.h>
#define PERIOD (1.68 - 1)/1.68*0xFFFF /* 1 second */
define cv1 (0xFFFF - PERIOD/4) /* 1/4 of this period */
define cv2 (0xFFFF - PERIOD/4*2) /* 2/4 of this period */
/* CAPCOM CC0 ISR */
void CC0_event(void) interrupt 0x10 {
  if(CC0 == (int)cv1) CC0 = cv2; else CC0 = cv1; 
}
```

CAPCOM timer count upwards; the period is therefore determined by the duration from 1 second to 1.68 seconds – the relative fraction of this duration (i.e. \((1.68 - 1)/1.68\)) is multiplied by the full scale timer value 0xFFFF to yield the period value. A similar idea leads onto the compare values \(cv1\) and \(cv2\).

Interrupt vector number 0x10 has been reserved for CAPCOM channel CC0; here, we use the ISR to swap compare value \(cv1\) for \(cv2\) and vice versa (alternate) – this could for instance be used to program a multi-channel PWM: All channels start with logic level ‘high’ and are switched off at the different compare values.

Capture and Compare unit

```c
(void main(void) { 
  DP2 |= 0x0001; /* P2.0 : output associated with CC0 (CC0IO) */ 
  P2  |= 0x0001; /* set P2.0 high -> LED off */
  T0CON &= 0xFF00; /* reset timer 0 (T0): Timer mode */
  T0CON |= 0x0006; /* set timer frequency: T0I = binary 110 */
  T0REL = PERIOD; /* set RELOAD register (timer 0) */
  T0 = PERIOD; /* reset T0 register */
  CCM0 &= 0xFFF0; /* reset CCMOD0 */
  CCM0 |= 0x0005; /* initialize CCMOD0 : compare mode 1 */ 
  CC0 = cv1; /* initialize CC0 with compare value 1 (cv1) */
  CC0IC = 0x0044; /* enable CC0 interrupt, ILVL = 1, GLVL = 0 */
  TOB = 1; /* start TO */
  IEN = 1; /* allow all interrupts */
  while111; /* forever... */
} /* main */
```

Mode selection for CC0 (compare mode 1); initialize CC0 with \(cv1\).
void main(void) {
    DP2 |= 0x0001;             /* P2.0 : output associated with CC0 (CC0IO) */
    P2  |= 0x0001;             /* set P2.0 high -> LED off */
    T01CON &= 0xFF00;          /* reset timer 0 (T0): Timer mode */
    T01CON |= 0x0006;          /* set timer frequency: T0I = binary 110 */
    T0   = PERIOD;             /* reset T0 register */
    CCM0 &= 0xFFF0;            /* reset CCMOD0 */
    CCM0 |= 0x0005;            /* initialize CCMOD0 : compare mode 1 */
    CC0 = cv1;                 /* initialize CC0 with compare value 1 (cv1) */
    CC0IC = 0x0044;            /* enable CC0 interrupt, ILVL = 1, GLVL = 0 */
    T0R  = 1;                   /* start T0 */
    IEN  = 1;                   /* allow all interrupts */
    while(1);                  /* forever... */
} /* main */

CC0 interrupts are enabled, choosing an interrupt level (ILVL) of ‘1’
Pulse Width Modulation (PWM)

- A *PWM signal* is a pulse train with a fixed period and a variable duty cycle (ON:OFF ratio); the duty cycle can vary from 0% (off) to 100% (always on)

$$\begin{align*}
\text{threshold 2 (period, fixed)} & = 0xFFFF \\
\text{threshold 1 (duty cycle)} & = 0x0000
\end{align*}$$

- On the C167 there are 4 independent PWM units; they are configured through a number SFR:

- Each PWM unit has its own 16-bit counter, a period register ($PPx$) and a pulse width register ($PWx$)

- Both $PPx$ and $PWx$ are shadowed to allow them to be modified while the unit is active

- The PWM channels can be configured to trigger an interrupt to indicate the beginning of a new period; this is specified in *Interrupt Control register PWMIC*

- Four modes of operation can be chosen:
  1. Standard PWM (edge aligned)
  2. Symmetrical PWM (centre aligned)
  3. Burst mode (channel 0 acts as enable for chnl 1)
  4. Single shot mode
Pulse Width Modulation (PWM)

- The operating mode is selected in PWMCON1:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEN</td>
<td>PWM Channel X Output Enable Bit</td>
</tr>
<tr>
<td>PM</td>
<td>PWM Channel X Mode Control Bit</td>
</tr>
<tr>
<td>PB</td>
<td>PWM Channel 0/1 Burst Mode Control Bit</td>
</tr>
<tr>
<td>PS</td>
<td>PWM Channel X Single Shot Mode Control Bit</td>
</tr>
</tbody>
</table>

- PWMCON0 controls the PWM timers and interrupts:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTR</td>
<td>PWM Timer x Run Control Bit</td>
</tr>
<tr>
<td>PTI</td>
<td>PWM Timer x Input Clock Selection</td>
</tr>
<tr>
<td>PEI</td>
<td>PWM Channel x Interrupt Enable Flag</td>
</tr>
<tr>
<td>PRI</td>
<td>PWM Channel x Interrupt Request Flag</td>
</tr>
</tbody>
</table>

- The clock source of the PWM unit can be either the CPU clock (\(f_{CPU}\)) or a pre-scaled version thereof (\(f_{CPU}/64\))

- With \(f_{CPU} = 20\) MHz this allows for the following maximum / minimum PWM frequencies:

<table>
<thead>
<tr>
<th>Input Clock ((f_{CPU}/64))</th>
<th>8-bit PWM Resolution</th>
<th>10-bit PWM Resolution</th>
<th>12-bit PWM Resolution</th>
<th>14-bit PWM Resolution</th>
<th>16-bit PWM Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x 20 MHz (50μs)</td>
<td>1: 6.10 kHz</td>
<td>1: 23.6 kHz</td>
<td>1: 35.45 kHz</td>
<td>1: 45.7 kHz</td>
<td>1: 54.8 kHz</td>
</tr>
<tr>
<td>2 x 10 MHz (25μs)</td>
<td></td>
<td>1: 33.8 kHz</td>
<td>1: 47.3 kHz</td>
<td>1: 57.3 kHz</td>
<td>1: 68.6 kHz</td>
</tr>
<tr>
<td>5 x 4 MHz (2 μs)</td>
<td></td>
<td>1: 4.8 kHz</td>
<td>1: 6.2 kHz</td>
<td>1: 7.3 kHz</td>
<td>1: 8.2 kHz</td>
</tr>
</tbody>
</table>

- Note that the centred PWM only runs at half the rate of the edge aligned PWM
Pulse Width Modulation (PWM)

- **Standard PWM** is selected by clearing bit PMx in register PWMCON1; this causes the corresponding timer to count up until it matches the value of the **period shadow register**

- Upon reaching the period, the timer count register is automatically reset to 0 and the process starts over

- The associated output (if enabled) is kept ‘low’ until the timer has reached the value of the **pulse width shadow register**; upon reaching this value, the output latch is set ‘high’

---

**Example:**
A small DC-motor is to be driven using an edge aligned PWM signal on P7.3 with a period of 100 ms and a pre-programmed duty cycle

![Diagram](image)

- A period of 100 ms means a PWM rate of 10 Hz; the table on slide MP6-48 indicates that this can be achieved with a pre-scaled clock (\(f_{CPU}/64\)) and using between 14 and 16 bit of the counter register

- The **PWMCON0** register is thus set to 0x0088 (binary 0000.0000.1000.1000 – pre-scaled clock on PWM channel 3, timer is running)

- The mode control register **PWMCON1** is set to 0x0008 (binary 0000.0000.0000.1000 – edge aligned PWM, output pin enabled)
Pulse Width Modulation (PWM)

```
#include <reg167.h>
#define PERIOD     (20e6/64)*100e-3    /* 100 ms */
#define OFF_PHASE  PERIOD/10*9         /* duty cycle */

void main(void) {
    DP7 |= 0x08;              /* P7.3 : output associated with CC7 (CC7IO) */
    P7  &= ~0x08;              /* set P7.3 low -> motor off */
    PP3 = PERIOD;              /* initialize channel 3 period register */
    PW3 = OFF_PHASE;           /* initialize channel 3 pulse width register */
    PWMCON1 = 0x0008;          /* channel 3: edge aligned PWM, o/p enabled */
    PWMCON0 = 0x0088;          /* channel 3 operates of fCPU/64, T. running */
    while(1);                  /* forever... */
} /* main */
```

PERIOD is defined in relation to the pre-scaled CPU clock (fCPU/64); one tick of this clock lasts for \(\frac{1}{(20\cdot10^6/64)} \approx 3.2 \mu s\); the PERIOD value is \(100 \text{ ms} / 3.2 \mu s = (20\cdot10^6/64) \cdot 100 \cdot 10^{-3}\); the PWM runs in the background — the program does not have to do anything anymore!

Capture and Compare unit

The operation of the PWM unit can be monitored through a peripheral window.