# Timersystem im Output-Compare Modus

Sie verstehen die Betriebsart Output-Compare des Timersystems im HCS08. Sie können das Timersystem zur Generierung von akustischen Signalen einsetzen.

## 1. Timer mit Output Compare

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und nutzen Sie das gegebene File main.c als Hauptdatei. Fügen Sie in der Bibliothek MC_Library zu den Verzeichnissen Lib_Headers bzw. Lib_Sources die gegebenen Files sound.h bzw. sound.c zu.

Analysieren Sie den Code und implementieren Sie im File sound_temp.c die nötigen Konfigurationen für das Timer-Modul TPM1 (siehe CW Task-View), so dass das in main.c gegebene Soundfile abgespielt wird.

**Project.prm**

```
STACKSIZE 0x200    // Stacksize 0x200 => 512 Bytes

VECTOR ADDRESS 0xFFC4 errISR_RTC        // RTC
VECTOR ADDRESS 0xFFC6 errISR_IIC        // IIC
VECTOR ADDRESS 0xFFC8 errISR_ACMP       // ACMP
VECTOR ADDRESS 0xFFCA errISR_ADC        // ADC Conversion
VECTOR ADDRESS 0xFFCC errISR_KBI        // KBI Keyboard
VECTOR ADDRESS 0xFFCE errISR_SCI2T      // SCI2 transmit
VECTOR ADDRESS 0xFFD0 errISR_SCI2R      // SCI2 receive
VECTOR ADDRESS 0xFFD2 errISR_SCI2E      // SCI2 error
VECTOR ADDRESS 0xFFD4 errISR_SCI1T      // SCI1 transmit
VECTOR ADDRESS 0xFFD6 errISR_SCI1R      // SCI1 receive
VECTOR ADDRESS 0xFFD8 errISR_SCI1E      // SCI1 error
VECTOR ADDRESS 0xFFDA errISR_TPM2O      // TPM2 overflow
VECTOR ADDRESS 0xFFDC errISR_TPM2CH1    // TPM2 channel 1
VECTOR ADDRESS 0xFFDE errISR_TPM2CH0    // TPM2 channel 0
VECTOR ADDRESS 0xFFE0 errISR_TPM1O      // TPM1 overflow
      //VECTOR ADDRESS 0xFFE2 errISR_TPM1CH5    // TPM1 channel 5
VECTOR ADDRESS 0xFFE2 soundISRfreq      // TPM1 channel 5
VECTOR ADDRESS 0xFFE4 errISR_TPM1CH4    // TPM1 channel 4
VECTOR ADDRESS 0xFFE6 errISR_TPM1CH3    // TPM1 channel 3
      //VECTOR ADDRESS 0xFFE8 errISR_TPM1CH2    // TPM1 channel 2
VECTOR ADDRESS 0xFFE8 soundISRduration  // TPM1 channel 2
VECTOR ADDRESS 0xFFEA errISR_TPM1CH1    // TPM1 channel 1
VECTOR ADDRESS 0xFFEC errISR_TPM1CH0    // TPM1 channel 0
VECTOR ADDRESS 0xFFF0 errISR_USB        // USB
VECTOR ADDRESS 0xFFF2 errISR_SPI2       // SPI2
VECTOR ADDRESS 0xFFF4 errISR_SPI1       // SPI1
VECTOR ADDRESS 0xFFF6 errISR_MCGLOL     // MCGLOL (Loss of Lock)
VECTOR ADDRESS 0xFFF8 errISR_LowVoltage // Low voltage detect
VECTOR ADDRESS 0xFFFA errISR_IRQ        // IRQ
VECTOR ADDRESS 0xFFFC errISR_SWI        // SWI
VECTOR ADDRESS 0xFFFE _Startup          // Reset vector: this is the default entry point
for an application.
```

## main.c

```
#include "platform.h"      /* include peripheral declarations */
#include "sound.h"

/**
 * Ring Tone Melody to be played in RTTTL format
 */
const char soundFile[] =
"Swiss:d=4,o=5,b=85:8f,16p,16f,f,a#,8a#,16p,16a,a,p,8f,16p,16f,f,c6,8c6,16p,16a#,a#,p,d6,
8p,8d6,8c6,8c6,c6,8p,8a#,8a,2g,e,2f,p,p";
const char soundFile2[] =
"FinalCountdown:d=4,o=5,b=125:p,8p,16b,16a,b,e,p,8p,16c6,16b,8c6,8b,a,p,8p,16c6,16b,c6,e,
p,8p,16a,16g,8a,8g,8f#,8a,g.,16f#,16g,a.,16g,16a,8b,8a,8g,8f#,e,c6,2b.,16b,16c6,16b,16a,1
b,p,p";

/**
 * Switch on Rear LEDs on Port D2
 */
void initPorts(void)
{
    PTDDD = 0x04;
    PTDD = 0x04;
}


/**
 * TPM1: Counter running with frequency 1 MHz
 * - No TOF interrupt
 * - Modulo = default
 * - Prescale = 1
 */
void initTimer(void)
{
  TPM1SC = 0x10;
}


/**
 * main program
 */
void main(void)
{
  int i = 0;

  initPorts();             // Port init

  initTimer();             // Timer init

  soundPlay(soundFile2);    // Play ring tone melody

  EnableInterrupts;        // Interrupts enable

  for(;;) {

    if (!soundIsPlaying()) {
      if (i == 0) {
        soundPlay(soundFile);
        i++;
      } else {
        soundPlay(soundFile2);
        i = 0;
      }
    }

  }
}
```

## sound.h

```
#ifndef SOUND_H
#define SOUND_H


bool soundIsPlaying(void);
void soundTooglePlayPause(void);

void soundStart(void);
void soundStop(void);

void soundPlay(const char *soundFile);

#endif /* SOUND_H_ */
```

## sound.c

```
#include "platform.h"
#include "sound.h"

#define FCNT              1000000   // TPM1 counter frequency [Hz]
#define MAX_MELODY_SIZE       800   // max number of notes

typedef struct
{
  uint8 note;     // note frequency (0 = a Okt.4 ... 47 = g# Okt.7, 48 = pause)
  uint8 time;     // note duration (in units of 1/64 notes)
} tNote;


const uint16 noteCounterticks[49]=  // # of counter ticks for half the period duration of
some note
{
    // c            c#              d               d#              e
f               f#            g             g#              a             a#
b/h
    FCNT/(2*261.6),  FCNT/(2*277.2),  FCNT/(2*293.7),  FCNT/(2*311.1),  FCNT/(2*329.6),
FCNT/(2*349.2),  FCNT/(2*370),  FCNT/(2*392),  FCNT/(2*415.3),  FCNT/(2*440),
FCNT/(2*466.2),  FCNT/(2*493.9),  // 4. oktave
    FCNT/(2*523.3),  FCNT/(2*554.4),  FCNT/(2*587.3),  FCNT/(2*622.3),  FCNT/(2*659.3),
FCNT/(2*698.5),  FCNT/(2*740),  FCNT/(2*784),  FCNT/(2*830.6),  FCNT/(2*880),
FCNT/(2*932.3),  FCNT/(2*987.8),  // 5. oktave
    FCNT/(2*1046.5), FCNT/(2*1108.7), FCNT/(2*1174.7), FCNT/(2*1244.5), FCNT/(2*1318.5),
FCNT/(2*1396.9), FCNT/(2*1480), FCNT/(2*1568), FCNT/(2*1661.2), FCNT/(2*1760),
FCNT/(2*1864.7), FCNT/(2*1975.5), // 6. oktave
    FCNT/(2*2093),   FCNT/(2*2217.4), FCNT/(2*2349.3), FCNT/(2*2489),   FCNT/(2*2637),
FCNT/(2*2793.8), FCNT/(2*2960), FCNT/(2*3136), FCNT/(2*3322.4), FCNT/(2*3520),
FCNT/(2*3729.4), FCNT/(2*3951),   // 7. oktave
    65535 //pause
};


static uint16 tick64;    // # of counter ticks for a 1/64 note

static char soundTitle[20];
static tNote melody[MAX_MELODY_SIZE];
static uint16 melodyPos;
static uint16 melodySize;
static uint16 delay;
```

```
/**
 * ISR: TPM1 Channel 2 - Output Compare
 * Realizes tone duration by counting down the
 * number of 1/64 notes of current tone
 */
interrupt void soundISRduration(void)
{
  // @TODO complete ISR TPM1 Ch2
  TPM1C2SC_CH2F = 0;
  TPM1C2V += tick64; // Counter: dauer für 1/64

  if (delay > 0)  // current tone not complete yet
  {
    delay--;
  }
  else            // current tone complete, setup to play next tone
  {
    melodyPos++;
    if (melodyPos < melodySize)
    {
      TPM1C5V = TPM1CNT + noteCounterticks[melody[melodyPos].note]; // next frequency
      delay = melody[melodyPos].time * 4;                           // next duration

      // tone: toggle output, pause (=48): do not toggle output
      TPM1C5SC_ELS5A = (melody[melodyPos].note == 48 ? 0 : 1);
    }
    else
    {
      // end of melody : do not toggle output anymore
      TPM1C5SC = 0;
      TPM1C2SC = 0;
    }
  }
}


/**
 * ISR: TPM1 Channel 5 - Output Compare
 * Realizes tone frequency by toggling output pin
 * every half period duration of current tone
 */
interrupt void soundISRfreq(void)
{
  // @TODO write ISR TPM1 Ch5
  TPM1C5SC_CH5F = 0; // interrupt flag reset
  // CV = current CV + frequency counterTick
  TPM1C5V += noteCounterticks[melody[melodyPos].note]; // CounterTicket curr. frequency
}
```

```
/**
 * Starts a melody from the current position
 *
 * @remarks: Initializes TPM1 as follows:
 *   CH2 : Output Compare without port pin use, Interrupt enable
 *   CH5 : Output Compare, toggle port pin, Interrupt enable
 */
void soundStart(void)
{
  if (melodyPos == melodySize) melodyPos = 0;

  // @TODO Write TPM1 init function
  // configure "freq" timer channel
  // TPM1 channel 5
  TPM1C5V = TPM1CNT + noteCounterticks[melody[melodyPos].note]; // CounterTicket
  TPM1C5SC_MS5x = 1;   // Channel Mode: Output Compare
  TPM1C5SC_ELS5x = 1;  // toggle port pin
  TPM1C5SC_CH5F = 0;   // clear interrupt flag
  TPM1C5SC_CH5IE = 1;  // Enable Interrupt
  //TPM1C5SC = 0x54;

  // configure "delay" timer channel
  // TPM1 channel 2
  delay = melody[melodyPos].time; // die anzahl 64tel in Delay speichern
  TPM1C2V = TPM1CNT + tick64; // Counter: dauer für 1/64
  TPM1C2SC_MS2x = 1;   // Channel Mode: Output Compare
  TPM1C2SC_ELS2x = 0;  // without port pin use (software compare)
  TPM1C2SC_CH2F = 0;   // TOF-Flag reset
  TPM1C2SC_CH2IE = 1;  // Enable Interrupt
}


/**
 * Stops the current melody
 */
void soundStop(void)
{
  TPM1C5SC = 0;
  TPM1C2SC = 0;
}


/**
 * Returns true, if the player is playing a meldoy
 *
 * @returns
 *    true, if playing, false else
 */
bool soundIsPlaying(void)
{
  return (TPM1C5SC != 0);
}


/**
 * Toggles between play and pause.
 */
void soundTooglePlayPause(void)
{
  if (soundIsPlaying()) soundStop();
  else soundStart();
}
```

```
/*
 * Parses the ring tone string and stores frequency and duration
 * for every tone in an array of type tNote.
 *
 *  @param [in] *soundFile   - RTTTL string
 */
void soundPlay(const char *p)
{
  uint8 i, value, param, note;
  uint8 duration, defDuration;
  uint8 octave, defOctave;

  // Read Title
  for (i=0; i<sizeof(soundTitle); i++)
  {
    if (*p == ':') break;
    soundTitle[i] = *p;
    p++;
  }

  while (*p && *p != ':') p++;
  if (!*p) return;
  p++;


  // parse default values
  while (*p) {
    while (*p == ' ') p++;     // Skip Spaces
    if (!*p) return;           // abort at the end of the string
    if (*p == ':') break;

    param = *p++;
    if (*p != '=') return;

    p++;
    value = 0;
    while (*p >= '0' && *p <= '9') value = value * 10 + (*p++ - '0');

    switch (param) {
      case 'd': defDuration = 64 / value; break;
      case 'o': defOctave = ((uint8)value - 4) * 12; break;
      case 'b': tick64 = (uint16)(((60 * 10000000l) / (value * 64)) - 1); break; // bpm
    }

    while (*p == ' ') p++;
    if (*p == ',') p++;
  }
  p++;

  melodySize = 0;
  while (*p)
  {
    duration = defDuration;
    octave = defOctave;

    // Skip whitespace
    while (*p == ' ') p++;
    if (!*p) return;

    // Parse duration
    if (*p >= '0' && *p <= '9')
    {
     value = 0;
     while (*p >= '0' && *p <= '9') value = value * 10 + (*p++ - '0');
     duration = (uint8) (64 / value);
    }
```

```
// Parse note
    switch (*p){
      case 0: return;
      case 'C': case 'c': note = 0; break;
      case 'D': case 'd': note = 2; break;
      case 'E': case 'e': note = 4; break;
      case 'F': case 'f': note = 5; break;
      case 'G': case 'g': note = 7; break;
      case 'A': case 'a': note = 9; break;
      case 'H': case 'h': note = 11; break;
      case 'B': case 'b': note = 11; break;
      case 'P': case 'p': note = 48; break;
    }
    p++;

    if (*p == '#'){
       note++;
       p++;
    }

    if (*p == 'b'){
       note--;
       p++;
    }

    // Parse special duration
    if (*p == '.'){
      duration += duration / 2;
      p++;
    }

    // Parse octave 4..7
    if (*p >= '0' && *p <= '9'){
      octave = ((*p++ - '0') - 4) * 12;
    }

    // Parse special duration (again...)
    if (*p == '.') {
      duration += duration / 2;
      p++;
    }

    // Skip delimiter
    while (*p == ' ') p++;
    if (*p == ',') p++;

    note += octave;
    if (note > 48) note = 48;

    melody[melodySize].note = note;
    melody[melodySize].time = duration;
    melodySize++;
    if (melodySize >= MAX_MELODY_SIZE) break;
  }

  melodyPos = 0;
  soundStart();
}
```