7.2: Timersystem im Input-Capture Modus

Sie verstehen die Betriebsart Input-Capture des Timersystems im HCS08. Sie können das Timersystem zur Auswertung von empfangenen Signalen einsetzen.

1. Timer mit Input Capture

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 ifrRx.h bzw. ifrRxFront_temp.c zu (Drag/Drop/Copy).

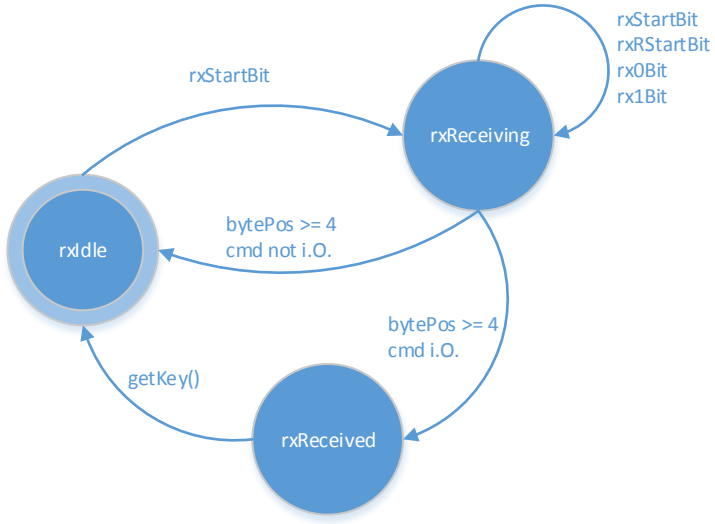
Analysieren Sie den gegebenen Code. Implementieren Sie im File ifrRxFront_temp.c an den 3 gekennzeichneten Stellen (siehe CW Task-View) die fehlende Funktionalität, so dass man mit den Tasten S, W und T der Fernbedienung die Soundwiedergabe wie folgt steuern kann:

S - Start / Pause

W - Skip to next song

T - Skip to previous song

Statemachine:



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 0xFFEC ifrFrontISR       // 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 "ifrRx.h"
#include "sound.h"

const char sound1[] =
    "lastxmas:d=4,o=5,b=112:16d6,e6,8p,e6,8d6,8p,8a,8e6,8e6,8f#6,d.6,8b,8b," \
    "8e6,8e6,f#6,d.6,8b,8c#6,8d6,8c#6,2b.,16e6,f#6,8p,e.6,8p,8b,8f#6,8g6,8f#6," \
    "2e6,8d6,8c#6,8d6,8c#6,c#6,8d6,8p,8c#6,8p,2a,16d6,e6,8p,e6,8d6,8p,8a," \
    "8e6,8e6,8f#6,d.6,8b,8b,8e6,8e6,f#6,d.6,8b,8c#6,8d6,8c#6,2b.,16e6,f#6,8p," \
    "e.6,8p,8b,8f#6,8g6,8f#6,2e6,8d6,8c#6,8d6,8c#6,c#6,8d6,8p,8c#6,8p,a";

const char sound2[] =
    "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";

const char sound4[] = "Chariots of Fire:d=16,o=5,b=85:8c#,f#.,g#.,a#.,4g#,4f,8p,8c#," \
    "f#.,g#.,a#.,2g#,8p,8c#,f#.,g#.,a#.,4g#,4f,8p,8f,f#.,f.,c#.,2c#";

const char sound3[] = "Toccata:d=16,o=6,b=63:32a7,32g7,4a7,4p,32g7,32f7,32e7," \
    "32d7,8c#7,4d7,4p.,32a,32g,4a,4p,16e,16f,16c#,4d,4p.,32a5,32g5,4a5,4p,32g5,32f5,32e5," \
    "32d5,8c#5,4d5,4p.,4d5,8c#5,8e5,8g5,8a#5,8c#,4e,8g,8e,2f#,4p,16c#,32d,32e,32c#,32d," \
    "32e,32c#,32d,32e,32c#,16d,16e,32f,32g,32e,32f,32g,32e,32f,32g,32e,16f,16g,32a,32a#," \
    "32g,32a,32a#,32g,32a,32a#,32g,8a,4p,16c#7,32d7,32e7,32c#7,32d7,32e7,32c#7,32d7,32e7," \
    "32c#7,16d7,16e7,32f7,32g7,32e7,32f7,32g7,32e7,32f7,32g7,32e7,16f7,16g7,32a7,32a#7,32g7," \
    "32a7,32a#7,32g7,32a7,32a#7,32g7,8a7,4p,16a7.,32g7,32a#7,32e7,32g7,32a#7,32e7,32f7,32a7," \
    "32d7,32f7,32a7,32d7,32e7,32g7,32c7,32e7,32g7,32c7,32d7,32f7,32a#,32d7,32f7,32a#,32c7," \
    "32e7,32a,32c7,32e7,32a,32a#,32d7,32g,32a#,32d7,32g,32a,32c7,32f,32a,32c7,32f,32g,32a#," \
    "32e,32g,32a#,32e,32f,32a,32d,32f,32a,32d,32e,32g,32c#,32e,32g,32c#,2d,4p,4a#,32p.,32a.," \
    "32g.,32f.,32e.,32d.,32c#.,32b5.,32c#.,32a5.,32c#.,32e.,32g.,16g,32f,32g,32f,32g,32f,32g," \
    "32f,32g,32f,32g,32f,32g,32f,32g,32f,16e.,2d,2p,8a,32d7,32a,32e7,32a,32f7,32a,32d7,32a," \
    "32e7,32a,32f7,32a,32g7,32a,32e7,32a,32f7,32a,32g7,32a,32a7,32a,32f7,32a,32g7,32a,32a7,32a," \
    "32a#7,32a,32g7,32a,32a7,32a,32f7,32a,32g7,32a,32e7,32a,32f7,32a,32d7,32a,32e7,32a,32c#7," \
    "32a,32d7,32a,32a,32a,32a#,32a,32g,32a,32a,32a,32f,32a,32g,32a,32e,32a,32f,32a,32d,32a,32g," \
    "32a,32e,32a,32f,32a,32d,32a,32e,32a,32c#,32a,32d,32a,32a5,32a,32a#5,32a,32g5,32a,32a5,32a," \
    "32f5,32a,32g5,32a,32e5,32a,32f5,32a,32d5,32a,32g5,32a,32e5,32a,32f5,32a,32d5,32a,32e5,32a," \
    "32c#5,32a,4d5,4p,32d,32f,32a#,32f,32c,32e,32a,32e,32a#5,32d,32g,32d,32a5,32c#,32e,32a,16d," \
    "16a#,16c,16a,16a#5,16g,8a,32p,32d,32f,32a#,32f,32c,32e,32a,32e,32a#5,32d,32g,32d,32a5,32c#," \
    "32e,32a,16d,16a#,16c,16a,16a#5,16g,4a,16p,32p,32g,32f,32e,32d,32c#,32b5,32c#,32a5,32b5,32c#," \
    "32d,32e,32f,32g,32a,32g,32f,32e,32f,32d,32f,32a,32c#7,32d7,32a,32b,32c#7,32d7,32e7,32f7,32a7," \
    "8a#7,16d,16a#7,16c,16a7,16a#5,16g7,8a7,32p,32d7,32f7,32a#7,32f7,32c7,32e7,32a7,32e7,32a#,32d7," \
    "32g7,32d7,32a,32c#7,32e7,32a7,16d,16a#7,16c,16a7,16a#5,16g7,4a7,8p,4b,8c#7.,16b,16a,16c#7,32e7," \
    "32g7,16a#7,32a7,32g7,32f7,32e7,32f7,32e7,32d7,32c#7,32d7,32c7,32a#,32a,32g,32f,32e,32d,4c#," \
    "32c#7.,32e7,32c#7,32a#,32c#7,32a#,32c#7,32e7,32c#7,32a#,32c#7,32a#,32c#7,32e7,32c#7,32a#,32c#7," \
    "32a#,32c#7,32e7,32c#7,32a#,32c#7,32a#,32g.,32a#,32g,32e,32g,32e,32g,32a#,32g,32e,32g,32e,32g," \
    "32a#,32g,32e,32g,32e,32g,32a#,32g,32e,32g,32e,32c#.,32e,32c#,32a#5,32c#,32a#5,32c#,32e,32c#," \
    "32a#5,32c#,32a#5,32c#,32e,32c#,32a#5,32c#,32a#5,32c#,32e,32c#,32a#5,32c#,32a#5,32c#.,32e,32c#," \
    "32e,32g,32e,32c#,32e,32c#,32e,32g,32e,32c#,32e,32c#,32e,32g,32e,32c#,32e,32c#,32e,32g,32e,32g.," \
    "32a#,32g,32a#,32g,32a#,32g,32a#,32g,32a#,32g,32a#,32c#7.,32a#,32c#7,32e7,32c#7,32e7,32c#7,32e7," \
    "32c#7,32e7,32c#7,32e7,4a7,4a7,4a#7.,16a7,16g7,4a7.,16e,16f,16d,16e,16c#,16d,16b5,16c#,16a5," \
    "16a#5,16g#5,8a5,8g,4f,4d,2a5,1d5";

/**
 * Configures the port F and G as follows:
 * - Port F as output
 * - Port G as input with pull-up enabled
 */
void initPorts(void)
{
    PTFDD = 0x07;           // Port F = Output
    PTFD = 0x07;

    PTGDD = 0x00;           // Port G = Input
    PTGPE = 0xff;           // Port G = Pull-Up enable
}

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

/**
 * main program
 */
void main(void)
{
  const char * soundFiles[] = {sound1, sound2, sound3, sound4};
  uint8 count = sizeof(soundFiles) / sizeof(soundFiles[0]);

  uint8 playIndex = 0;

  initPorts();            // Ports konfigurieren

  ifrRxFrontInit();       // init IR receiver

  initTimer();            // Timer init

  EnableInterrupts;       // Interrupts aktivieren
```

```
  for(;;)
  {
    switch (ifrRxFrontGetKey())
    {
      case 'S' : // play/pause
        soundTooglePlayPause();
        break;

      case 'W' : // play next melody
        playIndex++;
        if (playIndex >= count) playIndex = 0;
        soundPlay(soundFiles[playIndex]);
        break;

      case 'T' : // play previous melody
        if (playIndex == 0) playIndex = count;
        playIndex--;
        soundPlay(soundFiles[playIndex]);
        break;

      case '+' :  break;
      case '-' :  break;
    }
  }
}
```

ifrRx.h

```
#ifndef IFRRX_H_
#define IFRRX_H_

#include "platform.h"

#define TIMER_FREQ               1000000.0

#define START_BIT_PULSE_TIME        9000.0e-6          // 9000 us pulse
#define START_BIT_PAUSE_TIME        4500.0e-6          // 4500 us pause
#define RSTART_BIT_PAUSE_TIME       2250.0e-6          // 2250 us pause
#define PULSE_TIME                   560.0e-6          //  560 us pulse
#define PAUSE_1_TIME                1690.0e-6          // 1690 us pause
#define PAUSE_0_TIME                 560.0e-6          //  560 us pause

#define MIN_TOLERANCE                  0.7
#define MAX_TOLERANCE                  1.3

#define START_BIT_PULSE_LEN_MIN     ((uint16)(TIMER_FREQ * START_BIT_PULSE_TIME * MIN_TOLERANCE + 0.5)-1)
#define START_BIT_PULSE_LEN_MAX     ((uint16)(TIMER_FREQ * START_BIT_PULSE_TIME * MAX_TOLERANCE + 0.5)-1)
#define START_BIT_PAUSE_LEN_MIN     ((uint16)(TIMER_FREQ * START_BIT_PAUSE_TIME * MIN_TOLERANCE + 0.5)-1)
#define START_BIT_PAUSE_LEN_MAX     ((uint16)(TIMER_FREQ * START_BIT_PAUSE_TIME * MAX_TOLERANCE + 0.5)-1)
#define RSTART_BIT_PAUSE_LEN_MIN    ((uint16)(TIMER_FREQ * RSTART_BIT_PAUSE_TIME * MIN_TOLERANCE + 0.5)-1)
#define RSTART_BIT_PAUSE_LEN_MAX    ((uint16)(TIMER_FREQ * RSTART_BIT_PAUSE_TIME * MAX_TOLERANCE + 0.5)-1)
#define PULSE_LEN_MIN               ((uint16)(TIMER_FREQ * PULSE_TIME * MIN_TOLERANCE + 0.5)-1)
#define PULSE_LEN_MAX               ((uint16)(TIMER_FREQ * PULSE_TIME * MAX_TOLERANCE + 0.5)-1)
#define PAUSE_1_LEN_MIN             ((uint16)(TIMER_FREQ * PAUSE_1_TIME * MIN_TOLERANCE + 0.5)-1)
#define PAUSE_1_LEN_MAX             ((uint16)(TIMER_FREQ * PAUSE_1_TIME * MAX_TOLERANCE + 0.5)-1)
#define PAUSE_0_LEN_MIN             ((uint16)(TIMER_FREQ * PAUSE_0_TIME * MIN_TOLERANCE + 0.5)-1)
#define PAUSE_0_LEN_MAX             ((uint16)(TIMER_FREQ * PAUSE_0_TIME * MAX_TOLERANCE + 0.5)-1)

char ifrRxFrontGetKey(void);
void ifrRxFrontInit(void);

#endif /* IFRRX_H_ */
```

ifrRxFront.c

```
#include "platform.h"
#include "ifrRx.h"

/**
 * Sets a bit of the @ref rxBuf to '1'
 *
 * @param[in] buf
 *      the desired buffer to set the bit
 * @param[in] bytePos
 *      the desired byte [0..(rxBufSize - 1)]
 * @param[in] bitPos
 *      the desired bit [0...7]
 */
#define SetRxBit(buf, bytePos, bitPos)      (buf[bytePos] |= (1 << bitPos))

/**
 * Sets a bit of the @ref rxBuf to '0'
 *
 * @param[in] buf
 *      the desired buffer to clear the bit
 * @param[in] bytePos
 *      the desired byte [0..(rxBufSize - 1)]
 * @param[in] bitPos
 *      the desired bit [0...7]
 */
#define ClearRxBit(buf, bytePos, bitPos)    (buf[bytePos] &= ~(1 << bitPos))

typedef enum
{
  rxIdle,
  rxReceiving,
  rxReceived
} rxStates;

typedef enum
{
  rxStartBit,
  rxRStartBit,
  rx1Bit,
  rx0Bit
} rxBits;

typedef union
{
  uint8 buf[4];
  struct
  {
    uint16 adr;
    uint8 cmd;
    uint8 cmdN;
  } cmd;
} tCommand;
```

```
#define rxBuf   (rxCommand.buf)
static tCommand rxCommand;
static rxStates rxState;



/**
 * Front infrared receiver interrupt service routine
 */
interrupt void ifrFrontISR(void) // TPM1CH0
{
  static uint16 oldTicks = 0;
  static uint16 pulse, pause;
  static uint8 bitPos = 0;
  static uint8 bytePos = 0;
  rxBits rxBit;

  PTFD_PTFD1 = 0;               // switch on led for debug purposes
  TPM1C0SC_CH0F = 0;            // clear interrupt flag

  if (PTED_PTED2) {             // Port E Channel 2 == 1
    // rising edge -> calculate pulse time
    pulse = TPM1C0V - oldTicks; // calculate pulse time
    oldTicks = TPM1C0V;         // set start ticks
    PTFD_PTFD1 = 1;             // switch off led for debug purposes
    return;                     // exit ISR (we need pulse and pause time)
  } else {
    // falling edge -> calculate pause time
    pause = TPM1C0V - oldTicks; // calculate pause time
    oldTicks = TPM1C0V;         // set start ticks
  }

  // state machine
  // Check if Pulse is Start / Repeat Start or Data (0/1) Bit
  if (pulse > PULSE_LEN_MIN && pulse < PULSE_LEN_MAX)
  {
    // Get Value of the Data Bit: 0 / 1
    if (pause > PAUSE_0_LEN_MIN && pause < PAUSE_0_LEN_MAX) rxBit = rx0Bit;
    else if (pause > PAUSE_1_LEN_MIN && pause < PAUSE_1_LEN_MAX)rxBit = rx1Bit;
  }
  else if (pulse > START_BIT_PULSE_LEN_MIN && pulse < START_BIT_PULSE_LEN_MAX)
  {
    // Get Value of Start / Repeat Start
    if (pause > START_BIT_PAUSE_LEN_MIN && pause < START_BIT_PAUSE_LEN_MAX) rxBit = rxStartBit;
    else if (pause > RSTART_BIT_PAUSE_LEN_MIN && pause < RSTART_BIT_PAUSE_LEN_MAX) rxBit = rxRStartBit;
  }

  // Evaluate current state
  switch (rxState)
  {
    // State: Idle
    case rxIdle:
      // check for start bit
      if (rxBit == rxStartBit)
      {
        bitPos = bytePos = 0;   // reset bit/byte position
        rxState = rxReceiving;  // set state to receiving
      }
      else if (rxBit == rxRStartBit) { /* Repeated Start */ }
      break;

    // State: Receiving
    case rxReceiving:
      // check for start / repeated start bit
      if (rxBit == rxStartBit || rxBit == rxRStartBit)
      {
        bitPos = bytePos = 0;   // reset bit/byte position
      }
      else
      {
        // get Bit -> command
        if (rxBit == rx1Bit) SetRxBit(rxBuf, bytePos, bitPos); // set current Bit = 1
        else ClearRxBit(rxBuf, bytePos, bitPos);               // set current Bit = 0

        // check if command read completed
        if (++bitPos > 7)
        {
          bitPos = 0;           // set current Pos = 0
          bytePos++;            // increment byte Pos
          // check if all bytes read successfully
          if (bytePos >= 4)
          {
            // check if the command negated the same as the received negated command -> set State Received
            // verification of the received message
            if ((rxCommand.cmd.cmd ^ rxCommand.cmd.cmdN) == 0xff) rxState = rxReceived;
            else rxState = rxIdle;  // Received failed -> set State Idle
          }
        }
      }
      break;

    // State: Received
    case rxReceived: break;
```

```
    // Default state
    default: rxState = rxIdle; break; // set Idle state
  }

  PTFD_PTFD1 = 1;               // switch off led for debug purposes
}

/**
 * Returns a key or 0 if no key was pressed.
 */
uint8 ifrRxFrontGetKey(void)
{
  volatile uint16 adr; // not used
  uint8 cmd = 0;

  //check if command received
  if (rxState == rxReceived)
  {
    adr = rxCommand.cmd.adr;      // adr is atm not used

    // translate cmd into readable char-cmd (Key-Desc)
    switch (rxCommand.cmd.cmd)
    {
      case 1 : cmd = 'S'; break;  // 1 --> "S-Key"
      case 2 : cmd = 'W'; break;  // 2 --> "W-Key"
      case 3 : cmd = 'T'; break;  // 3 --> "T-Key"
      case 5 : cmd = '+'; break;  // 5 --> "Plus-Key"
      case 4 : cmd = '-'; break;  // 4 --> "Minus-Key"
    }
    rxState = rxIdle;             // set State back to idle
  }
  return cmd;
}

/**
 * This function configures the TPM1 channel 0 as follows:
 * - input capture on both edges
 * - with interrupt enabled
 */
void ifrRxFrontInit(void)
{
  TPM1C0SC_MS0x  = 0;           // Input capture
  TPM1C0SC_ELS0x = 3;           // Capture on rising or falling edge
  TPM1C0SC_CH0F  = 0;           // clear interrupt flag
  TPM1C0SC_CH0IE = 1;           // Enable Interrupt
}
```