FUJITSU

Application Note

IR-Remote-control for 16LX-Series

1. Summary

This Application Note gives some ideas and basic information, how to decode the protocol of standard Infrared-Remote-Control-Units for TV, VCR, etc. with Fujitsu microcontrollers.
By using these basics, also decoding of other than the mentioned codes can be realised.
The software-examples are for the MB90540series, but can be used with minor changes for all other 16bit controllers, too.

History

24.01.2002	HW	V1.0	New version, RC5-code
26.02.2002	JM,HW	V1.1	"PWM"-code

Table of Contents:

Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for **all products delivered free of charge** (eg. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC's etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. **Note, all these products are intended and must only be used in an evaluation laboratory environment**.

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days form the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.

2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH´s entire liability and the customer´s exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH´s sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH in original packing and without further defects resulting from the customer´s use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.

3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.

4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH´s and its suppliers´ liability is restricted to intention and gross negligence.

   NO LIABILITY FOR CONSEQUENTIAL DAMAGES

   To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect.

2. RC5 code

In Europe the RC5 code, developed by Philips, is a very common standard for data transmission by infrared (IR) remote controls for consumer products like TV, VCR, etc. The protocol is illustrated in figure 2.1. One data package consists of totally 14 bits. The first two bits are start-bits and always are '1'-bits. The next bit toggles each time a new key is pressed. This means, for example, when the volume-button is held some seconds, this bit will not toggle.

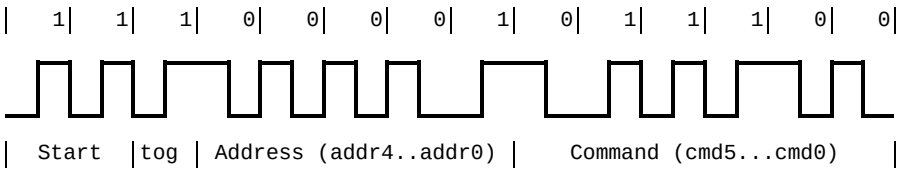

Figure 2.1: RC5 code

The remaining 12 bits carry the real information. In totally 2048 different data-packages can be transmit. In order to address different products the data is split into a 5 bit system-address (like TV1, TV2, VCR1, VCR2, SAT, CD, etc.) and a 6 bit command field (like number-keys 0-9, Volume +, Volume -, etc).
All bits are 'biphase'-coded, that means, each bit consist of two alternated half-bits like shown in figure 2.2. A low-high transition means logical '1', a high-low combination indicates logical '0'. The time for one bit is constant 1.778ms and the same for all 14 bits. The transmission time for a whole package amounts to 24.892ms followed by a pause. The repetition time is 113.778ms.
A dataword always will transmit complete, even if the relevant key is released within the transmission.

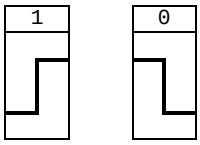

Figure 2.2: 'biphase'-coding

Although a lot of integrated circuits for RC5-encoding (e.g. SAA3006, SAA3010) and decoding (SAA3009, SAA3049) are available, it is near to use a microcontroller. Special receivers-modules for infrared remote control systems, like the TSOP12xx family from Vishay Telefunken, may help for easier application because PIN-diode, preamplifier and IR filter are already included. The demodulated output signal can directly be decoded by the microprocessor. The main benefit is the reliable function even in disturbed ambient and the protection against uncontrolled output pulses. Figure 2.3 shows a typical design how to connect an IR-receiver to the microcontroller.

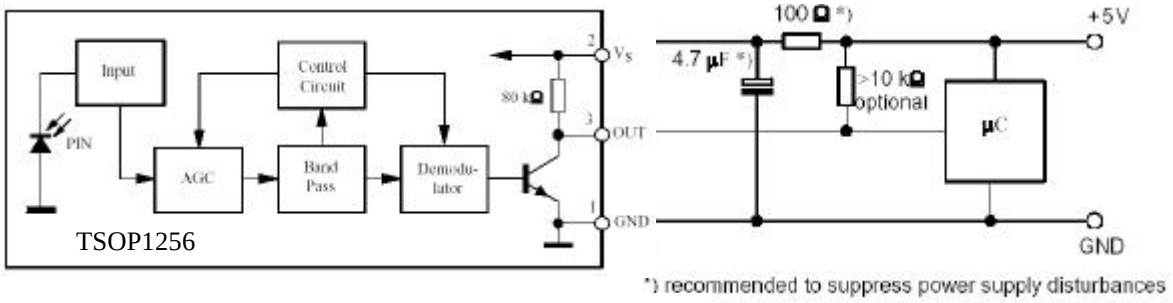

Figure 2.3: Receiver circuit

As easily can be seen the output signal is inverted. This has to be considered when decoding the protocol, and also for the examples within this documentation.
Because of the relatively slow bit timing, no special requirements for the processor are necessary. It is recommended to sample the RC5 input signal interrupt driven, so that the performance of the microcontroller will not get lost.

In the following some software examples will be shown, how RC5 protocol can be decoded by using resources of Fujitsu's 16LX-microcontrollers.

2.1. RC5-decoding with the Reload-Timer

All Fujitsu's 16LX-microcontrollers have at least one 16 bit Reload-Timer. It can be used to generate periodical interrupts. Further an external pin (TIN) can be used as a trigger input where the trigger edge is configurable. Both features are helpful for decoding a input signal like the RC5 one.
Four phases are necessary while decoding:

Phase 1 : Initialisation	The input pin has to be watched Waiting for transmission begin: The normal output of the IR-receiver is 'high'-level. Keep in mind, that most IR-receivers, like this one here, will invert the RC5 data-stream. This means, when any data-transmission begins a high-to-low transition has to be detected.
Phase 2 : Format check	The RC5 code begins with two start-bits. These bits can be used to check the right format of the incoming signal in order to prevent from any disturbed light. Therefor the signal-structure of the start-bits should be examined.
Phase 3 : Get the information	Bits containing the information have to be read.
Phase 4 : Final	Preparation for receiving next package

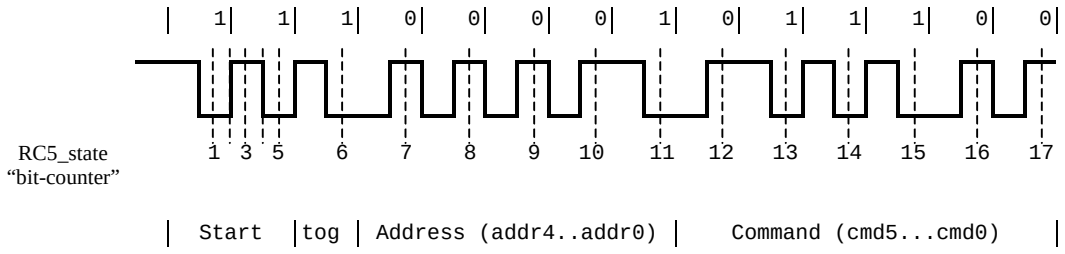

Figure 2.4: inverse RC5 code because of receiver-circuit

2.2. Phase 1: Initialisation

Because the idle level of the RC5-signal is 'high' any activity will be indicated by a high-to-low-edge. Therefore the Reload-Timer will be used first to detect a falling edge. This can be realised by its trigger-input TIN. The operation mode (TMCSR_MOD = "010") prepares the timer in that way that with a high-to-low edge at TIN the contents of the reload-register is loaded to the timer. By this the timer starts counting downwards. The Reload-Time is set in the reload-register TMRLR to the quarter time of one Bit: 1778µs / 4 = 445µs.
The Reload-value is 222 (=445/2) because the clock-source is selected to be 2µs (4 x 222 x 2µs = 1776µs = 1 bit-time).
When the Reload-Timer comes to an underflow (0x0000 → 0xFFFF) an underflow-interrupt will call an interrupt-handler exactly in the middle of the first half-bit. Now the format-check can begin.

```
/* clear „bit-counter“ */

RC5_state = 0;

/* stop the Reload-Timer */

TMCSR0 = 0;

/* set reload-register */

TMRLR0 = 1778/8; /* 1/4 bit-time */

/* configure Reload-Timer: */
   - prescaler 2us at 16 MHz
   - Trigger-Input falling-edge
   - no Reload
   - IRQ enable but no trigger

TMCSR0 = 0x91A;
```

2.3. Phase 2: Format check

Each RC5 bit-stream begins with two start-bits ‘1’. Because of the ‘bi-phase’ decoding the level of the bit will change from low to high, respectively from high to low because of the inversion by the receiver chip. To check the correct format the half-bits of both start-bits should be sampled. Well, of course, the first half-bit gets lost, because this just passed before the first transition.
The reload-timer automatically loads again the counter with the contents of the reload-register and continues counting downwards. This will call the interrupt-routine periodically each quarter bit-time and the variable `RC5_state` will be increased. The variable functions like a pointer to the position within the bit-stream. On the four positions 1, 3 and 5 the input signal is compared with the expected values: “1”, “0”, “1”. If this inspection fail the receiving is aborted and the Reload-Timer will initialised again. Maybe light disturbing caused noise on the inputs line.
But if all three tests are passed the following bits will interpreted as information bits. It is not more necessary to check both half-bits, but only the second half. Therefore the reload-time is changed to the time of a full bit length (1778µs).

```
__interrupt void IRQ_ReloadTimer0 (void)
{
  unsigned char RC5_in;

 RC5_in   = !PDR5_P56; /* sample RC-stream */
 RC5_state++; /* increment bit-pointer */

 switch (RC5_state) {

 case 1: /* disable Reload-Timer-Trigger
            to avoid retrigger */
         TMCSR0 = 0x81E;

         /* Startbit-control 2. half-bit */

         if (RC5_in != 1)
           RC5init();
         break;

 case 3: /* Startbit-control 3. half-bit */

         if (RC5_in != 0)
           RC5init();
         break;

 case 5: /* Startbit-control 4. half-bit */

         if (RC5_in != 1)
           RC5init();
         else
         { /* stop reload timer */

           TMCSR0 = 0;

          /* new reload-value: 1 Bit-length */
          /* /2 because of 2us clock ! */

           TMRLR0 = 1778/2;
```

2.4. Phase 3: Get the information

```
case 6: /* Toggle-bit */

        RC5_toggle_new = RC5_in;
        break;

case 7: /* Address-bit 4 */
case 8: /* Address-bit 3 */
case 9: /* Address-bit 2 */
case 10:/* Address-bit 1 */
case 11:/* Address-bit 0 */

  RC5_address_new=(RC5_address_new<<1)+ RC5_in;
  break;

case 12:/* Command-bit 5 */
case 13:/* Command-bit 4 */
case 14:/* Command-bit 3 */
case 15:/* Command-bit 2 */
case 16:/* Command-bit 1 */
case 17:/* Command-bit 0 */

  RC5_command_new=(RC5_command_new<<1)+RC5_in;
  break;
```

The first information bit is the toggle-bit. Each time a new key is pressed on the remote-control-unit this bit will change from ‘0’ to 1 or ‘1’ to ‘0’. If the key is pressed for a longer time the toggle-bit will not change anymore. The next five bits define the receiver address and subsequently the six command bits are received. Both, address- and command-bits are sent from MSB to LSB. Errors within the information bits can not be found. Checksum or equivalent error detection does not exist.

For easier data-processing address- and command-bits could be shifted into only one variable. Within the main-program the received information can then be handled with a simple case-instruction.

2.5. Phase 4: Final

When all 12 information-bits are received, the Reload-Timer will be initialised again. This means it wil be stopped and prepeared to be triggered again by the next RC5-package.

```
case 18: /* all bits received
            prepare for next packet */
  RC5init();
  break;

default: break;
} // switch
/* reset underflow interrupt request flag */

  TMCSR0_UF = 0;

} // interrupt Reload_Timer0
```

2.6. Summary of RC5 protocol

The RC5 protocol can be handled very easy. Only one Reload-Time is necessary. We found that the bit length of 1778µs varies between different IR-remote-control-units. So the real bittime has to be measured with an oscilloscope and the defined bit-time has to be adapted within the sourcecode.
Of course, also the input-capture-unit could be used to measure the bit-time. But this would waste resources of the microcontroller, because in reality IR-remote-control and the target application will be close together and therefore the used bit-timing will be known.

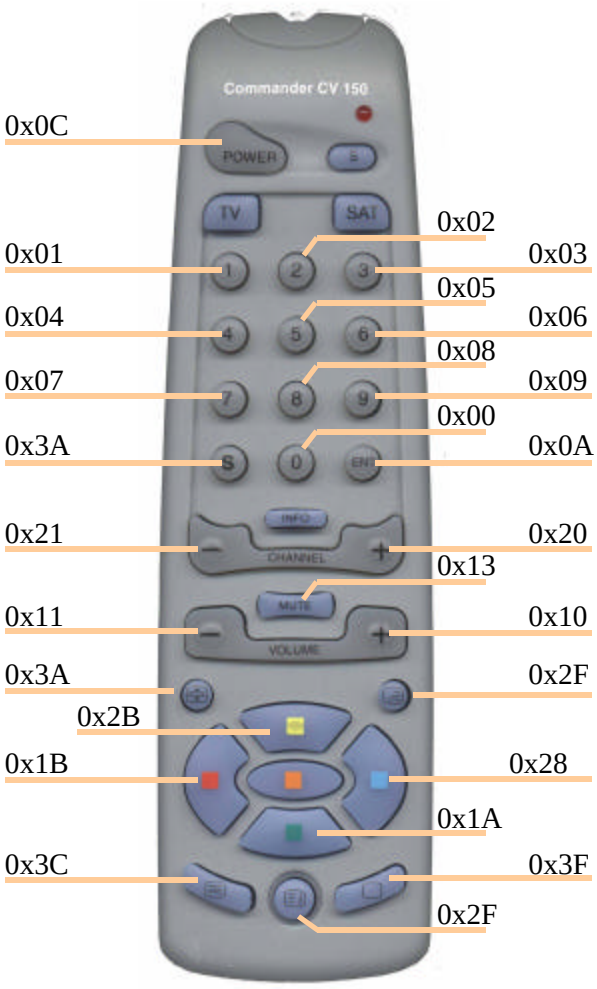

Figure 2.5: Remote-Control CV150 with Code TV 162

3. PWM-based codes

In contrast to the RC5-code other IR-protocols are using different bitlenght to encode ‚1' and ‚0' bits. The receiver-circuit is the same that is already mentioned in chapter 2. An example of the incoming datastream is shown in figure 3.1. Please keep in mind that the receiver-circuit has inverted the signal. The protocol begins with a startbit that is longer than all other bits. By this the receiption can be synchronised.

Figure 2.5: Remote-Control CV150 with Code 089

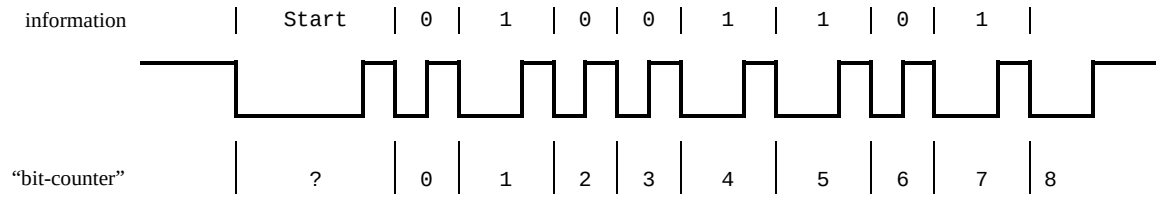

Figure 3.1: inverse PWM-code

The protocol starts with a start-bit, which is always the longest bit within the whole data-package. Further the length of the '1' bit is longer than the '0' bit. This means that three different bit-length are used to distinguish between three informations:

Length ("start"-bit) > Length ("1" bit) > Length ("0" bit)

The length of each bit depends on the IR-protocol and may differ from company to company. Also the number of bits will differ as well as the order (MSB/LSB).
Because this application will not refer to any special IR-remote control, no fixed values are given but can be evaluated easily by an oscilloscope.
For the following software example it is supposed that the protocol will take 12 bits beginning with the start-bit and followed by the LSB.

3.1. Phase 1: Initialisation

In order to evaluate the bit-information the length of one bit has to be measured. This can be done by using the Input-Capture-Unit. Therefore the IR-Receiver has to be connected to Pin INx. In this software example IN0 is used.
With each falling edge of the input signal the ICU will be triggered. Automatically the actual value of the Free-Running-Timer is stored and an interrupt-service-routine is called.

```
/* Initialisation: Input-Capture-Unit */
void InitICU0(void)
{
  TCCS = 0x02;   /*Free-Running-Timer:4us@16MHz */
  DDR7_D70 = 0;    /*IN0 = P70 = IR-input       */
  ICS01_EG01 = 1;  /*TIC0 falling Edge enabled  */
  ICS01_EG00 = 0;  /*TIC0 falling Edge enabled  */
  ICS01_ICP0 = 0;  /*TIC0  Flag cleared         */
  ICS01_ICE0 = 1;  /*IRQ   enabled              */
  IRbitcount = 0;  /*clear bitcounter          */
  IRcomplete = 0;  /*clear receive complete flag*/
  IRcommand  = 0;  /*clear receive comnand      */
}
```

3.2. Phase 2: Get the information

Each time when the interrupt-service-routine of the Input-Capture-Unit (ICU) is called the bit-counter is increased in order to find out the last bit. Also the captured value of the Free-Running-Timer is saved. In comparison with the last (old) value the bit-length is calculated. In case that the bit-time is longer than the given time for a start-bit, the bit-counter and the command-receive-buffer (IRcommandx) are cleared.
If no start-bit but a “1”-bit is detected than “1” is shifted into the receive-buffer from the left, because of LSB-first.
If all bits are received, than the receive-buffer is saved and a complete flag is set. So no overrun will occure.
At last the timer-value is copied in order to save it for the next calculation when the next falling-edge is detected. Of course, the interrupt-flag has to be cleared, too.

```
__interrupt void IRQ_ICU0(void)
{
  IRbitcount++;          /* Bit-counter              */
  IRtime_new = IPCP0;    /* capture time             */

  if (IRtime_new > IRtime_old)
  {
    IRtime =(IRtime_new - IRtime_old);
  }
  else
  {
    IRtime =(0xffff-IRtime_old + IRtime_new + 1);
  }

  if (IRtime > IR_TSYNC)
  {
    IRbitcount  = 0;             /* Sync found         */
    IRcommandx  = 0;             /* clear receive word */
  }
  else
  {
    IRcommandx = (IRcommandx >> 1); /*shift receive data */
    if (IRtime>(IR_THIGH))
      {
        IRcommandx = (IRcommandx | 0x400);   /* high bit */
      }
  }

  if (IRbitcount > 10)
  {
    IRcommand  = IRcommandx;
    IRcomplete = 1;
  }

  IRtime_old = IRtime_new;
  ICS01_ICP0 = 0;           /* TIC0  Flag cleared       */
}
```

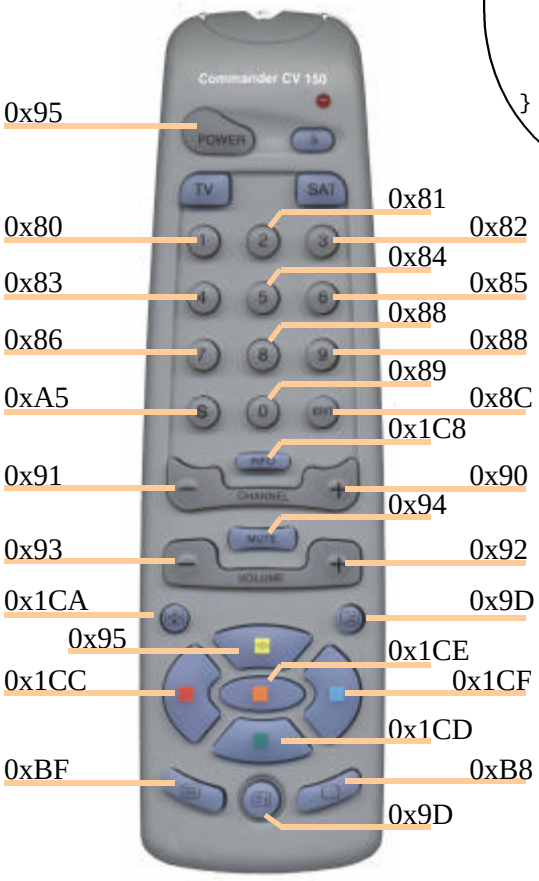

Figure 3.2: Remote-Control CV150 with Code TV 089

3.3. Summary of PWM protocol

The PWM protocol is another very simple protocol that can be handled easily. Unfortunately no format check is possible. When once started the software expects all data-bits. But a new start-bit within the data-transmission will reject the actual receive-buffer. So receive-error will be very rare, too.
Because only one Free-Running-Timer is available within Fujitsu´s 16LX microconroller the user has to take care that all Input-Capture-Units and also the Output-Compare-Units will use the same Free-Running-Timer. This means, that the timer should be set to the maximum needed frequency.

4. Perspective

The protocol of standard Infrared-Remote-Control-Units for TV, VCR, etc. is very simple to handle in general. Most protocols are relatively slow. In most cases the data-format is fixed. The kind of resource that will be the best in use depends on the protocol.
It is very important to check the used protocol of an Infrared-Remote-Control-Unit by an oscilloscope. In the meantime there are such a lot of derivatives with small changes that the receiver has to be adapted especially to the transmitter.
The examples given within this application-note are only basics in order to give an idea how to handle Infrared-protocols, but it can be used also for similar serial data protocols.
Side-effect is that the used microcontroller-resources are explained very detailed together with a practical application.

5. Appendix A: “RC5”-code - the program

MAIN.C:

```
/*---------------------------------------------------------------------
  MAIN.C (RC5-Decoding)
/*---------------------------------------------------------------------*/

#include "mb90540.h"

#define RC5_bit_time 1778   /* bit-length: normaly one bit in RC5-code takes 1778 us */
                            /* but we found changes: within different remote controls*/
//#define RC5_bit_time 1720 /* e.g.: "Commander CV150" (Conrad-Electronic) code=162  */

unsigned char RC5_state;
unsigned char RC5_command, RC5_command_new;
unsigned char RC5_address, RC5_address_new;
unsigned char RC5_toggle,  RC5_toggle_new;


void RC5init(void)
{
  RC5_state       = 0;      /* set RC5 state-machine to the beginning */
  RC5_command_new = 0;
  RC5_address_new = 0;
  RC5_toggle_new  = 0;
  TMCSR0 = 0;               /* stop reload timer                      */
  TMRLR0 = RC5_bit_time/8;  /* set reload value 1/4 Bit-length
                               (/8 because of 2us clock)              */
  TMCSR0 = 0x91A;           /* prescaler 2us at 16 MHz
                               Trigger-Input falling-edge
                               no Reload, IRQ enable but no trigger   */
}

// Description: RC5-receiver
// IR-Receiver at TIN0

void main(void)
{
  InitIrqLevels();
  __set_il(7);    /* allow all levels */

  DDR5_D56 = 0;   /* Input port (P56=TIN0) for RC5 signal */
  PDR4_P46 = 0;   /* for measurement purposes */
  DDR4_D46 = 1;

  RC5_command = 0;
  RC5_address = 0;
  RC5init();

  __EI();         /* globaly enable interrupts */


  while (1)
  {
    if (RC5_command != 0)
    {
      /* here you have to put your RC5-handler that reacts depending on
         RC5_address, RC5_command and RC5_toggle */

    RC5_command = 0;
    }
  }
} //main
```

```
__interrupt void IRQ_ReloadTimer0 (void)
{
  unsigned char RC5_in;

  RC5_in   = !PDR5_P56; /* RC5-signal at TIN0
                           (inverted because of Receiver-circuit)          */

  PDR4_P46 = !PDR4_P46; /* for measurement purposes:
                           toggles each time the RC5-signal is scaned      */

  RC5_state++;          /* bit-counter */

  switch (RC5_state)
  {
    case 1: TMCSR0 = 0x81E;   /* disable Trigger to avoid retrigger         */
            if (RC5_in != 1)  /* Startbit-control                           */
              RC5init();      /* if wrong format, then break cycle          */
            break;

    case 3: if (RC5_in != 0)  /* Startbit-control                           */
              RC5init();      /* if wrong format, then break cycle          */
            break;

    case 5: if (RC5_in != 1)  /* Startbit-control                           */
              RC5init();      /* if wrong format, then break cycle          */
            else
            {
              TMCSR0 = 0;                /* stop reload timer */
              TMRLR0 = RC5_bit_time/2;   /* set reload value 1 Bit-length
                                            (/2 because of 2us clock)       */
              TMCSR0 = 0x81B;            /* prescaler 2us at 16 MHz Reload,
                                            IRQ enable, Trigger             */
            }
            break;

    case 6: RC5_toggle_new = RC5_in;     /* Toggle-bit */
            break;

    case 7:  /* Address-bit 4 */
    case 8:  /* Address-bit 3 */
    case 9:  /* Address-bit 2 */
    case 10: /* Address-bit 1 */
    case 11: /* Address-bit 0 */
            RC5_address_new = (RC5_address_new << 1) + RC5_in;
            break;

    case 12: /* Command-bit 5 */
    case 13: /* Command-bit 4 */
    case 14: /* Command-bit 3 */
    case 15: /* Command-bit 2 */
    case 16: /* Command-bit 1 */
    case 17: /* Command-bit 0 */
            RC5_command_new = (RC5_command_new << 1) + RC5_in;
            break;

    case 18: /* all bits received, stop RC5-recieve, wait for next packet  */
            RC5_command  = RC5_command_new;
            RC5_address  = RC5_address_new;
            RC5_toggle   = RC5_toggle_new;
            RC5init();
            break;

    default: break;
  }

  TMCSR0_UF = 0;       /* reset underflow interrupt request flag */
}
```

VECTOR.C: only the changes are listed (please refer to template.prj)

```
/*---------------------------------------------------------------------------
  VECTOR.C (rot_enc_extIRQ)
/*---------------------------------------------------------------------------*/

#include "mb90540.h"

void InitIrqLevels(void)
{
/*  ICRxx               shared IRQs for ICR */

ICR02 = 2;      /*  IRQ15 IRQ16 */

__interrupt void IRQ_RotaryEncoder (void);

#pragma intvect IRQ_RotaryEncoder 15    /* external interrupt INT0/INT1 */
```

6. Appendix B: “PWM”code - the program

MAIN.C:

```
/*---------------------------------------------------------------------------
  MAIN.C (PWM-Decoding)
/*-------------------------------------------------------------------------*/

#include "mb90540.h"

#define IR_TSYNC  0.85*3000/4     /* min time for synchronisation */
#define IR_THIGH  0.85*1800/4     /* min time for high bit */

unsigned int  IRtime, IRtime_new, IRtime_old;
unsigned int  IRbitcount;
unsigned int  IRcommand, IRcommandx, IRcommand_last;
unsigned char IRcomplete;

/*-----------------------------------------------------------------------*/

void InitICU0(void)   /* Inilisation: Input-Capture-Unit */
{
  TCCS = 0x02;     /* Free-Running-Timer: 4us @ 16MHz */
  DDR7_D70 = 0;    /* IN0 = P70 = IR-input            */
  ICS01_EG01 = 1;  /* TIC0 falling Edge enabled       */
  ICS01_EG00 = 0;  /* TIC0 falling Edge enabled       */
  ICS01_ICP0 = 0;  /* TIC0  Flag cleared              */
  ICS01_ICE0 = 1;  /* IRQ   enabled                   */
  IRbitcount = 0;  /* clear bitcounter                */
  IRcomplete = 0;  /* clear receive complete flag     */
  IRcommand  = 0;  /* clear receive comnand           */
}



void main (void)
{
  InitICU0();
  InitIrqLevels();
  __set_il(7);    /* allow all levels */
  __EI();         /* globaly enable interrupts */

  while (1)
  {
    if (IRcomplete)
    {
      switch (IRcommand)
      { /*  Buttons :      */
        case 0x80: /* 1 */
                   break;

        default  : /* unknown         */
                   break;
      }
      IRcomplete = 0;
    }
  }
}
```

```
__interrupt void IRQ_ICU0(void)
{
  IRbitcount++;             /* Bit-counter              */
  IRtime_new = IPCP0;       /* capture time             */

  if (IRtime_new > IRtime_old)
  {
    IRtime =(IRtime_new - IRtime_old);
  }
  else
  {
    IRtime =(0xffff-IRtime_old + IRtime_new + 1);
  }

  if (IRtime > IR_TSYNC)
  {
    IRbitcount  = 0;               /* Sync found         */
    IRcommandx  = 0;               /* clear receive word */
  }
  else
  {
    IRcommandx = (IRcommandx >> 1); /* shift receive data */
    if (IRtime>(IR_THIGH))
      {
        IRcommandx = (IRcommandx | 0x400);  /* high bit found */
      }
  }

  if (IRbitcount > 10)
  {
    IRcommand  = IRcommandx;
    IRcomplete = 1;
  }

  IRtime_old = IRtime_new;
  ICS01_ICP0 = 0;         /* TIC0  Flag cleared       */
}
```

VECTOR.C: only the changes are listed (please refer to template.prj)

```
/*-----------------------------------------------------------------------
  VECTOR.C (IR_PWM)
/*-----------------------------------------------------------------------*/

#include "mb90540.h"

void InitIrqLevels(void)
{
/*  ICRxx               shared IRQs for ICR */

ICR06 = 6;      /*  IRQ23  IRQ24 */

__interrupt void IRQ_ICU0 (void);

#pragma intvect IRQ_ICU0          23    /* input capture CH.0           */
```