Infrared Remote Control Techniques on MC9S08RC/RD/RE/RG Family

by: Pavel Lajšner
Freescale Rožnov p. R., Czech Republic, Europe

1 Introduction

Ever wondered how infrared remotes work? What are the different modulations and data formats? This simple, low-cost, yet powerful technology is very often used in today's homes.

This application note provides an insight into several of the most frequently used infrared protocols and especially their implementation using the Freescale 8-bit MC9S08RC/RD/RE/RG Family of microcontrollers.

Freescale's MC9S08RC/RD/RE/RG Family is primarily targeted at those remote control applications equipped with a powerful CMT (Carrier Modulator Timer). The CMT module is a dedicated peripheral that allows the generation of infrared waveforms for **transmitting** infrared signals with minimal software overheads.

See Figure 1.

Table of Contents

© Freescale Semiconductor, Inc., 2005, 2008. All rights reserved.

freescale™ semiconductor

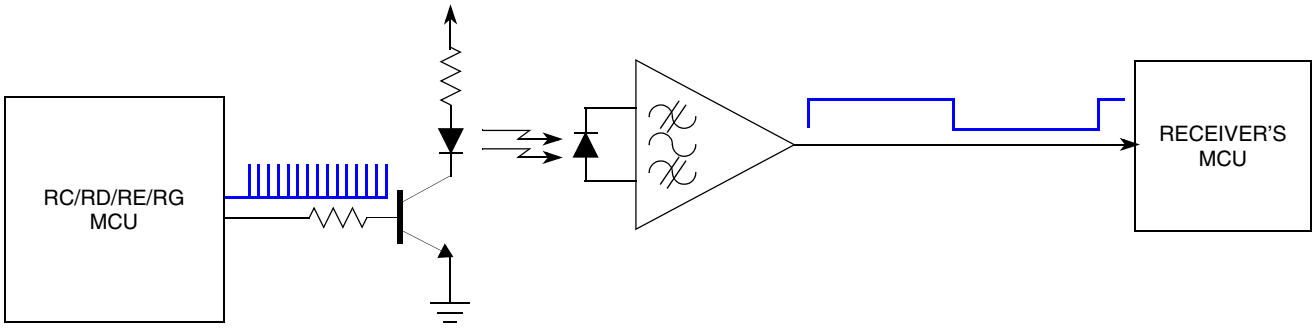

Figure 1. Infrared Modulation and Demodulation

2 Infrared Remote Control Modulation and Encoding Theory

The infrared light used in remote control applications is modulated in order for the receiver to distinguish between wanted signals and all other sources of infrared noise. There are several different modulation and encoding techniques used to distinguish between unwanted noise and useful infrared signals.

Basically, three modulation techniques are used:

1. Amplitude Modulation, On-Off Keying, OOK
2. FSK, Frequency Shift Keying, Frequency Modulation
3. Flash, 'Pulse' Modulation, Base Band

2.1 Amplitude Modulation, On-Off Keying, OOK

Using amplitude modulation is one of the oldest and simplest techniques, where infrared signals form a group of pulses with a certain frequency (typically 30–60 kHz), delimited by space where no signals are generated.

The receiver is tuned to a specific frequency and all other noise won't go through the receiver band pass filter. Fully integrated receivers from various manufacturer are available (for example Infineon, Vishay, Sharp and others). Simple three pin devices provide demodulated signals at logic levels that are very easy to interface with a receiver's microprocessor. They are usually tuned at a specific frequency (like 30, 33, 36, 38, 40, or 56 kHz).

Amplitude modulation systems use several encoding methods as described in the following subsections.

2.1.1 Pulse Distance Encoding

The distance between pulses defines log. ‘1’ or log. ‘0’ respectively, the pulse width is constant

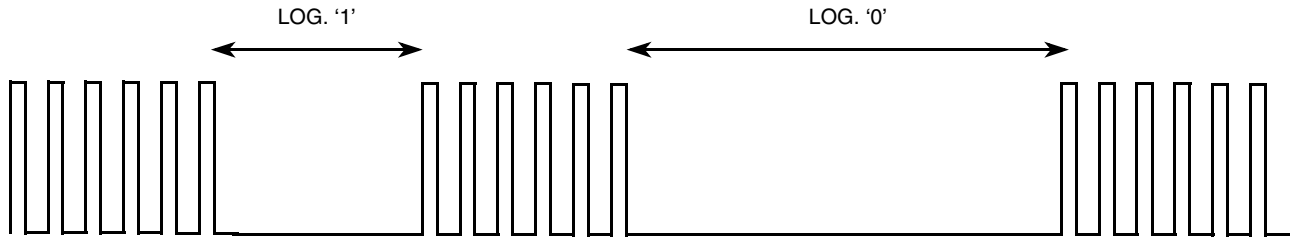

Figure 2. Pulse Distance Encoding

2.1.2 Pulse Width Encoding

The pulse width defines log. ‘1’ or log. ‘0’ respectively, the pulse distance is constant

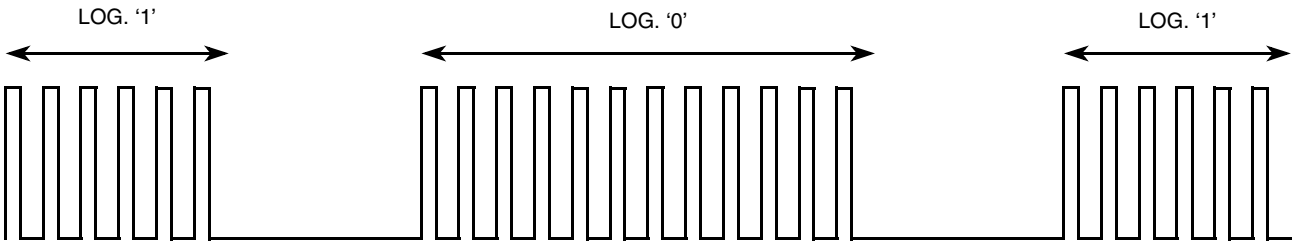

Figure 3. Pulse Width Encoding

2.1.3 Pulse Position Encoding (NRZ)

Here, the timescale is divided into the constant length intervals, the presence of the a pulse denotes log. ‘1’, and the absence denotes log. ‘0’, for example.

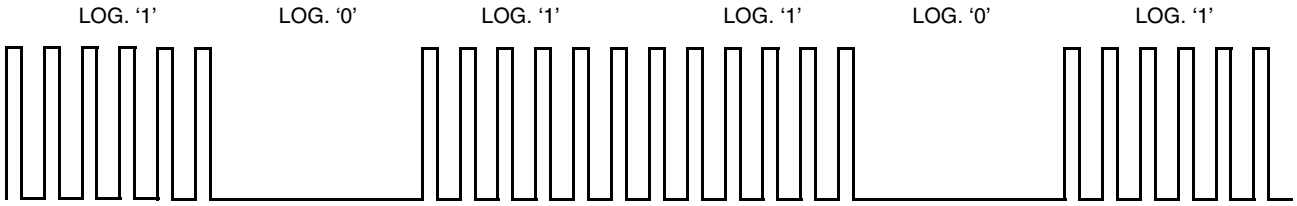

Figure 4. Pulse Position Encoding

2.1.4 Manchester (Biphase) Encoding

Each bit consists of two half-bits that always have a different level, i.e., there’s a transition from mark-to-space or space-to-mark. The polarity of the transition defines the logical level, for example mark-to-space denotes log. ‘1’, space-to-mark denotes log. ‘0’. See Figure 5.

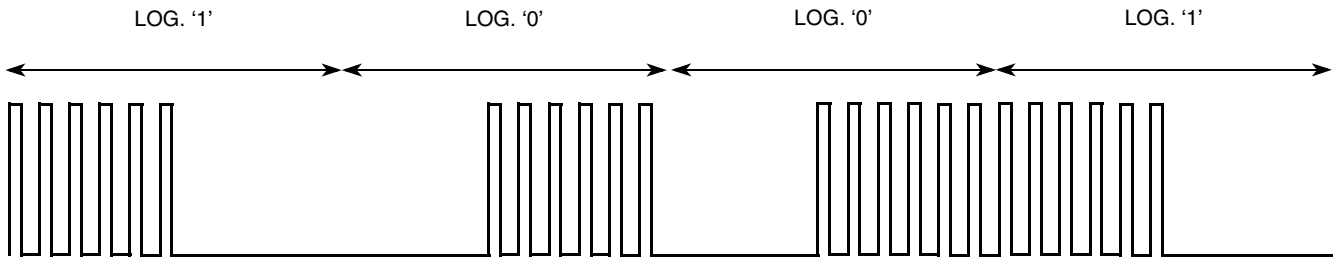

Figure 5. Manchester (Biphase) Encoding

Yet other encodings are possible, with the ones most commonly used mentioned here.

2.2 FSK, Frequency Shift Keying, Frequency Modulation

A frequency modulation uses different modulation frequencies for data logic levels. There’s usually no space between pulses. Frequency modulation is not widely used mainly because of demodulation complexity and a not very high efficiency in terms of power consumption on the transmitter side. Typical waveforms are shown in Figure 6.

Figure 6. Frequency Modulation

2.3 Flash, ‘Pulse’ Modulation, Base Band

In fact, ‘pulse’ modulation doesn’t use any form of modulation but rather short pulses (in CMT terminology, base-band mode). This sort of ‘modulation’ is very effective in terms of power consumption (the pulses are typically in the range of tens of microseconds). A disadvantage could be the complexity of decoding and the fact that the PC-based infrared systems (IrDA) may cause false receiver triggering.

Typically, the distance between pulses defines log. ‘1’ or log. ‘0’ respectively (pulse distance encoding).

Typical waveforms are shown in Figure 7.

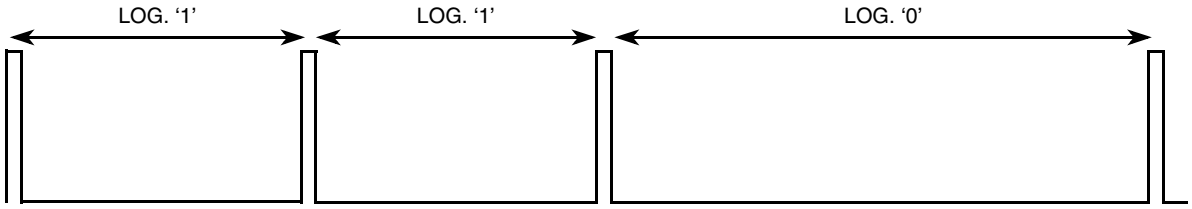

Figure 7. Flash Modulation (‘Pulse’ Modulation)

3 MC9S08RC/RD/RE/RG Family Overview

The MC9S08RC/RD/RE/RG are members of the low-cost, high-performance HCS08 Family of 8-bit microcontroller units (MCUs). All MCUs in this family use the enhanced HCS08 core and are available with a variety of modules, memory sizes, memory types, and package types.

3.1 Features

Features of the MC9S08RC/RD/RE/RG Family of devices are listed here. Please see the data sheet for the features available on the different family members.

3.1.1 HCS08 CPU (Central Processor Unit)

- Object code fully upward-compatible with M68HC05 and M68HC08 Families
- HC08 instruction set with added BGND instruction
- Support for up to 32 interrupt/reset sources
- Power-saving modes: wait plus three stops

3.1.2 On-Chip Memory

- On-chip in-circuit programmable Flash memory with block protection and security option
- On-chip random-access memory (RAM)

3.1.3 Oscillator (OSC)

- Low power oscillator capable of operating from crystal or resonator from 1 to 16 MHz
- 8 MHz internal bus frequency

3.1.4 Analog Comparator (ACMP1)

- On-chip analog comparator with internal reference (ACMP1)
- Full rail-to-rail supply operation
- Option to compare to a fixed internal bandgap reference voltage

3.1.5 Serial Communications Interface Module (SCI1)

- Full-duplex, standard non-return-to-zero (NRZ) format
- Double-buffered transmitter and receiver with separate enables
- Programmable 8-bit or 9-bit character length
- Programmable baud rates (13-bit modulo divider)

3.1.6 Serial Peripheral Interface Module (SPI1)

- Master or slave mode operation
- Full-duplex or single-wire bidirectional option
- Programmable transmit bit rate
- Double-buffered transmit and receive
- Serial clock phase and polarity options
- Slave select output
- Selectable MSB-first or LSB-first shifting

3.1.7 Timer/Pulse-Width Modulator (TPM1)

- 2-channel, 16-bit timer/pulse-width modulator (TPM1) module that can operate as a free-running counter, a modulo counter, or an up-/down-counter when the TPM is configured for center-aligned PWM
- Selectable input capture, output compare, and edge-aligned or center-aligned PWM capability on each channel

3.1.8 Keyboard Interrupt Ports (KBI1, KBI2)

- Providing 12 keyboard interrupts
- Eight with falling-edge/low-level plus four with selectable polarity
- KBI1 inputs can be configured for edge-only sensitivity or edge-and-level sensitivity

3.1.9 Carrier Modulator Timer (CMT)

- Configurable carrier generator module
- Modulator for generation of the waveforms
- Various modes available for generation of different infrared modes
- CMT module can generate interrupts
- Dedicated infrared output (IRO) pin
- Drives IRO pin for remote control communications
- Can be disconnected from IRO pin and used as an output compare timer
- IRO output pin has high-current sink capability

3.1.10 Development Support

- Background debugging system
- Breakpoint capability to allow single breakpoint setting during in-circuit debugging (plus two more breakpoints in on-chip debug module)
- Debug module containing two comparators and nine trigger modes. Eight deep FIFO for storing change-of-flow addresses and event-only data. Debug module supports both tag and force breakpoints.

3.1.11 Port Pins

- Eight high-current pins (limited by maximum package dissipation)
- Software selectable pullups on ports when used as input. Selection is on an individual port bit basis. During output mode, pullups are disengaged.
- 39 general-purpose input/output (I/O) pins, depending on package selection

3.1.12 Package Options

- 28-pin plastic dual in-line package (PDIP)
- 28-pin small outline integrated circuit (SOIC)
- 32-pin low-profile quad flat package (LQFP)
- 44-pin low-profile quad flat package (LQFP)
- 48-pin quad flat package (QFN)

3.1.13 System Protection

- Optional computer operating properly (COP) reset
- Low-voltage detection with reset or interrupt
- Illegal opcode detection with reset
- Illegal address detection with reset (some devices don’t have illegal addresses)

4 MC9S08RC/RD/RE/RG Family Infrared CMT Module

The basic simplified structure of the carrier modulator timer is shown in Figure 8. There are two main blocks of this module:

- Carrier generator
- Modulator

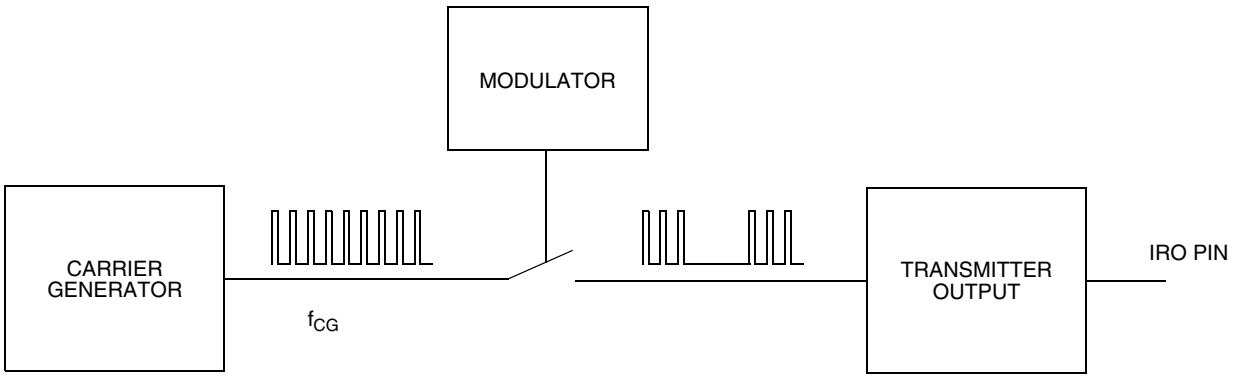

Figure 8. Simple CMT Module Structure

Carrier generator creates the carrier (base) frequency, typically in the range 30-60 kHz with selected duty cycle. The **Modulator** 'gates' this signal, i.e., generates mark and space periods respectively. The **Transmitter output** controls the behavior of the IRO pin, polarity, etc.

This is a simplified overview of the CMT module, the detailed structure is shown in Figure 9.

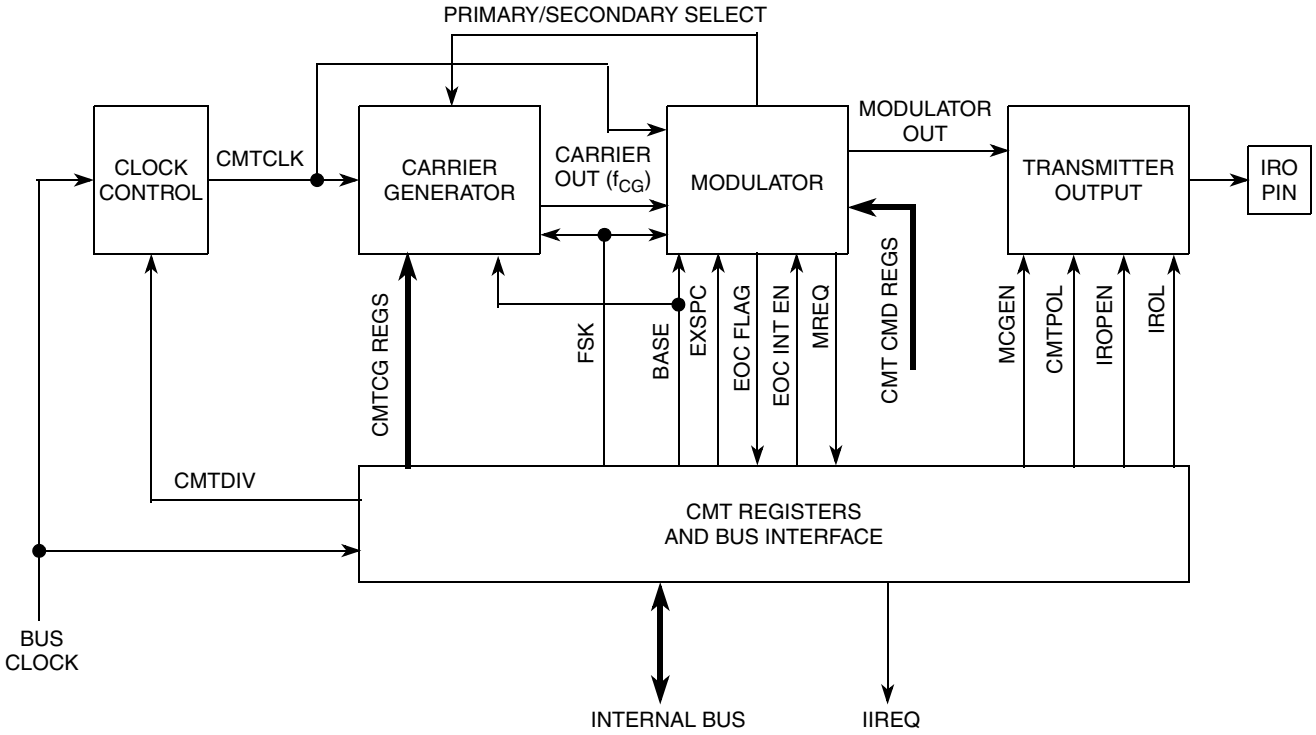

Figure 9. Detailed CMT Module Structure

Further explanation refers to the mostly used Time Mode, other modes are described in the MC9S08RC/RD/RE/RG data sheet. The time mode has been selected because it's the most popular infrared mode used.

4.1 Carrier Generator

Generates the carrier frequency (f_{CG} typically 30–60 kHz). A pair of two 8-bit registers is used to determine the frequency and duty cycle. The f_{CG} frequency is given by the sum of both high (CMTCGH1) and low (CMTCGL1) registers:

$$f_{CG} = \frac{f_{CMTCLK}}{(CMTCGH1 + CMTCGL1)}$$

A duty cycle (typically selected between 30% and 50%) is given as the ratio:

$$DutyCycle = \frac{CMTCGH1}{(CMTCGH1 + CMTCGL1)}$$

To calculate CMTCG registers values, the following equations can be used:

$$CMTCGH1 = \frac{f_{CMTCLK}}{f_{CG}} \cdot DutyCycle$$

$$CMTCGL1 = \frac{f_{CMTCLK}}{f_{CG}} \cdot (1 - DutyCycle)$$

Where:

f_{CMTCLK} = 8,000,000 Hz
Duty Cycle = 0.3 for 30% duty cycle
f_{CG} = 38,000 Hz (required carrier frequency)

Using this example, the CMTCG registers should be loaded with the following values:

CMTCGH1 = 63;
CMTCGL1 = 147;

The real f_{CG} would be 38.095 kHz, the duty cycle is exactly 30%.

4.2 Modulator

Modulator is another block that generates the waveform envelopes, i.e., it controls (gates) carrier frequency generator output. Two 16-bit registers (CMTCMD12 and CMTCMD34) control the timing of waveform envelopes:

CMTCMD12 (called ‘mark’ period register)
CMTCMD34 (called ‘space’ period register)

The main clock base of the modulator is $f_{CMTCLK}/8$, in the case of f_{CMTCLK} = 8 MHz, the minimum period is 1µs. Mark and space periods are as follows:

$$t_{MARK} = \frac{CMTCMD12 + 1}{f_{CMTCLK} \div 8}$$

$$t_{SPACE} = \frac{CMTCMD34}{f_{CMTCLK} \div 8}$$

To calculate CMTCMD register values, the following equations can be used:

$$CMTCMD12 = t_{MARK} \cdot \frac{f_{CMTCLK}}{8} - 1$$

$$CMTCMD34 = t_{SPACE} \cdot \frac{f_{CMTCLK}}{8}$$

Where:

f_{CMTCLK} = 8,000,000 Hz
t_{MARK} and t_{SPACE} are required mark and space times (in seconds)

For generation of one cycle with a 560 μs mark period followed by a 1690 μs space period, the CMTCMD12 and CMTCMD34 registers should be loaded with the following values:

CMTCMD12 = 559;
CMTCMD34 = 1690;

Using these example values, the following waveform will be generated. This waveform is just one pulse distance encoded bit, as described later in this document:

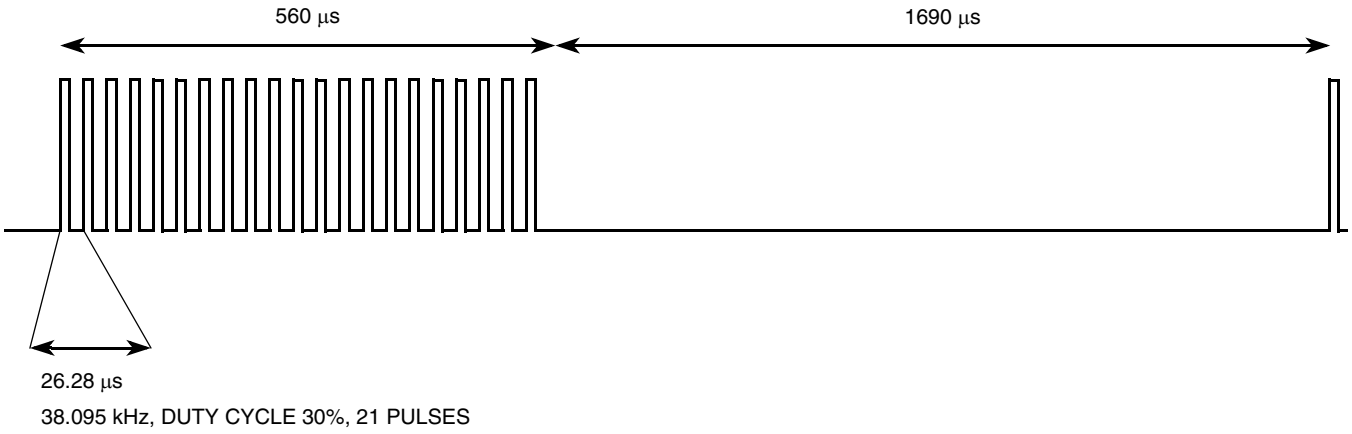

Figure 10. Example Waveform of Pulse Distance Encoded Bit

NOTE

CMTCLK value (8 MHz) used in this example is derived from the widely-used 16-MHz clock (crystal), whose frequency is divided by two to get the bus clock. The bus clock is then divided by 1 in the CMT clock control module. Such a setting allows the most precise timing while longer periods (lower frequencies) are possible using a different divider ratio in clock control module.

5 Example Protocols

There are hundreds of infrared protocols existing world wide, but most of them are just frequency or format variants of a few. Some of the most popular ones were picked up and implemented using the Freescale MC9S08RC/RD/RE/RG microprocessor fully utilizing the CMT module. The example C source codes are also available.

5.1 Pulse Distance Protocol

Pulse distance protocol is often used by Japanese companies (NEC and others). It uses pulse distance encoding and amplitude modulation. The data payload consists of 8 bits address and 8 bits command, both are sent twice for reliability. The second transmission of address and command are complementary, thus the total length of the frame is constant. The data is preceded by a train pulse, 9 ms mark, and a 4.5 ms space in order to settle automatic gain control (AGC) of the receiver. The data is finalized by a 560 μs mark tail pulse, to finish the last bit data gap (distance).

Log. ‘1’ is denoted as a 560 µs mark period followed by a 1690 µs space period

Log. ‘0’ is denoted as a 560 µs mark period followed by a 560 µs space period

Carrier frequency is 38 kHz.

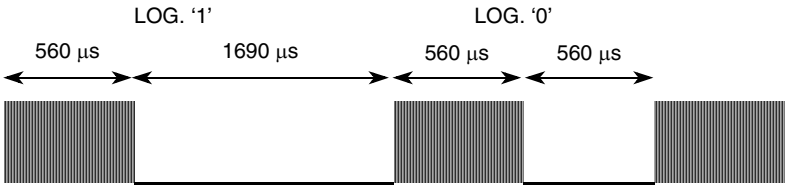

Figure 11. Pulse Distance Protocol, Bit Encoding

Overall data frame structure is shown in Figure 12.

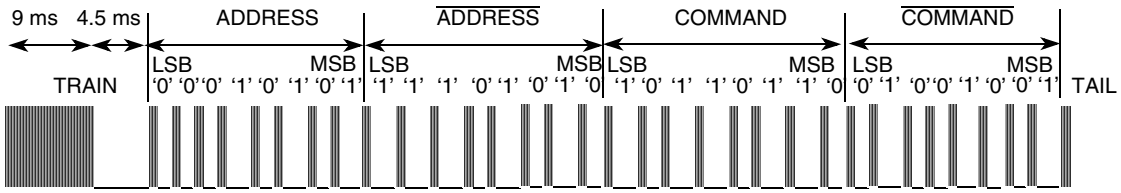

Figure 12. Pulse Distance Protocol, Data Frame Structure

Autorepeat function for this protocol is handled by repeat frames that do not carry any address or command data but are rather train pulses followed by a tail pulse. The repeat frame is repeated every 110 ms till the same key is pressed on the remote control. The format of the repeat frame is shown in Figure 13.

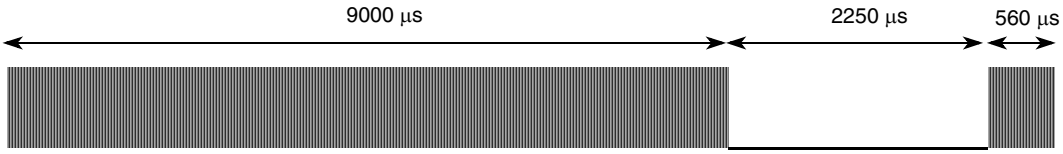

Figure 13. Pulse Distance Protocol, Repeat Frame Structure

The full sequence is shown in Figure 14.

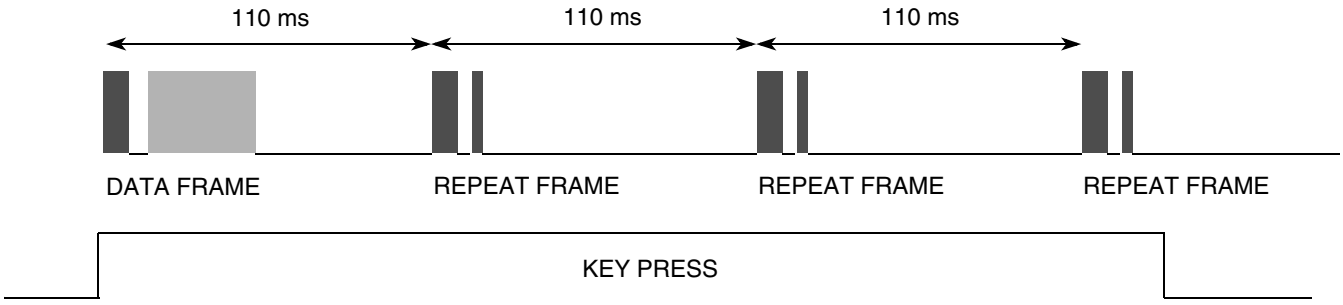

Figure 14. Pulse Distance Protocol, Full Sequence Structure

5.1.1 Pulse Distance Driver Flowchart

The overall structure the pulse distance driver is shown in Figure 15 (other protocols are not shown here but are very similar).

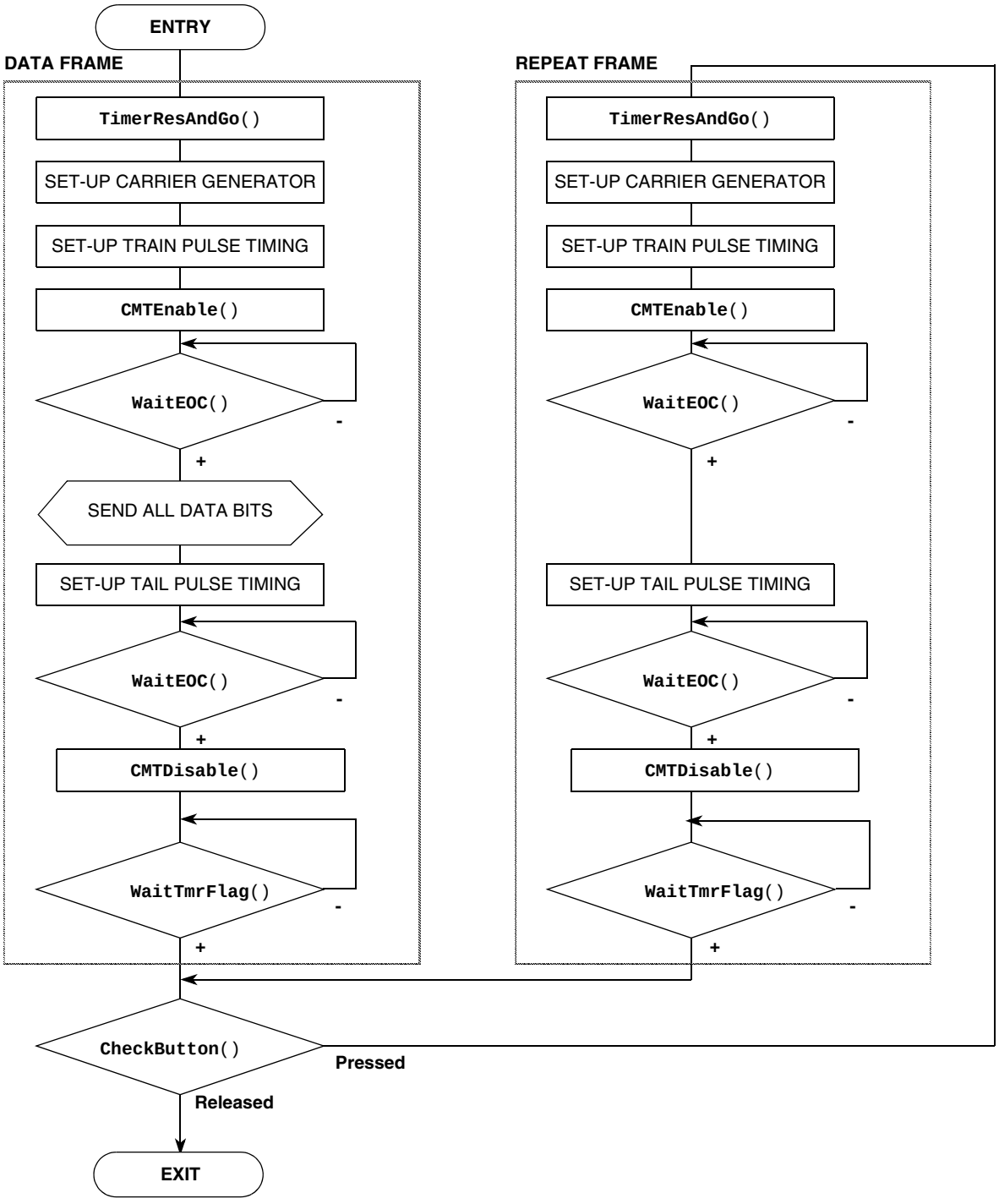

Figure 15. Pulse Distance Protocol, Driver Flowchart

5.1.2 Pulse Distance Protocol Source Code

```
#define PDP_REPEAT_PERIOD          TPMCLK_MS(110.0)
#define PDP_CARRIER_FREQUENCY      38000             // [Hz]
#define PDP_IR_RATIO               30                // [%]
#define PDP_PULSETRAIN_ON          9000.0            // usec
#define PDP_PULSETRAIN_OFF         4500.0            // usec

#define PDP_DATA_ON                560.0             // usec
#define PDP_DATA_1_OFF             2250.0-560.0      // usec
#define PDP_DATA_0_OFF             1120.0-560.0      // usec

void PDP_byte(unsigned char data)
{
unsigned char bits = 8;

    do {
        if (data & 0x01)
            ModClockOff(PDP_DATA_1_OFF);         // log. '1' space
        else
            ModClockOff(PDP_DATA_0_OFF);         // log. '0' space

        WaitEOC();                               // wait for CMT to load values
        data >>= 1;

    } while (--bits);
}

void PDP_protocol(unsigned char address, unsigned char data)
{
unsigned int cmdCnt;

    cmdCnt = 0;     /* command counter */

    TimerResAndGo(PDP_REPEAT_PERIOD);

    // set up Carrier generator
    CGClockOn(PDP_CARRIER_FREQUENCY, PDP_IR_RATIO);
    CGClockOff(PDP_CARRIER_FREQUENCY, PDP_IR_RATIO);

    // train pulse
    ModClockOn(PDP_PULSETRAIN_ON);           // on-time train pulse
    ModClockOff(PDP_PULSETRAIN_OFF);         // off-time train pulse

    CMTEnable(CMTMSC_MCGEN_MASK);            // enable & start CMT
    WaitEOC();                               // wait for CMT to load values

    ModClockOn(PDP_DATA_ON);                 // on-time always the same

    // send adress and data
    PDP_byte(address);
    PDP_byte(~address);
    PDP_byte(data);
    PDP_byte(~data);

    // tail pulse
```

```
    ModClockOff(0);                         // tail pulse is zero
    WaitEOC();                              // wait for CMT to load values

    CMTDisable();                           // disable CMT entirely

    WaitTmrFlag();

    while (CheckButton(cmdCnt++))           // see whether the key is still pressed
                                               (to send repeats)
    {
        TimerResAndGo(PDP_REPEAT_PERIOD);

        // set up Carrier generator
        CGClockOn(PDP_CARRIER_FREQUENCY, PDP_IR_RATIO);
        CGClockOff(PDP_CARRIER_FREQUENCY, PDP_IR_RATIO);

        // train pulse
        ModClockOn(PDP_PULSETRAIN_ON);      // on-time train pulse
        ModClockOff(PDP_PULSETRAIN_OFF);    // off-time train pulse

        CMTEnable(CMTMSC_MCGEN_MASK);       // enable & start CMT
        WaitEOC();                          // wait for CMT to load values

        // tail pulse
        ModClockOn(PDP_DATA_ON);            // on-time always the same
        ModClockOff(0);                     // tail pulse is zero
        WaitEOC();                          // wait for CMT to load values

        CMTDisable();                       // disable CMT entirely

        WaitTmrFlag();
    };
}
```

5.2 Macros and Common Functions Description

In the source code examples several C macros and common functions are used in order to improve readability and simplicity of the code. The macros are found in **ircommon.h**:

```
#define ModClockOn(usec)  CMTCMD12 = ((usec)*CMTCLOCK/(8*1000000.0)-1)
// useconds on-time macro for carrier generator

#define ModClockOff(usec) CMTCMD34 = ((usec)*CMTCLOCK/(8*1000000.0))
// useconds off-time macro for carrier generator

#define CGClockOn(freq,ratio)  CMTCGH1 = ((CMTCLOCK)/(freq)*(ratio)/100.0)
// mark time macro

#define CGClockOff(freq,ratio) CMTCGL1 = ((CMTCLOCK)/(freq)*(100.0-ratio)/100.0)
// space time macro
```

where **CMTCLOCK** defines the CMT module clock in Hz (here 8,000,000). Using these macros, the proper values are calculated at compile time, just as shown in the example source code.

Similarly **CMTEnable()** and **CMTDisable()** macros are defined:

```
#define CMTEnable(config) { CMTOC = CMTOC_IROPEN_MASK; CMTMSC = (config);}
#define CMTDisable()      { CMTMSC = 0; }
```

In addition, several common functions are defined in **ircommon.c**:

TimerResAndGo() and **WaitEOC()** functions are used to handle the inter-frame timing (i.e., the timing between data frames and repeat frames (where applicable)).

```
void TimerResAndGo(unsigned int repeattime)
{
    TPM1SC = (0x07&TPM1SC_PS_MASK) | TPM1SC_CLKSA_MASK; // for clock/128, BUS clock as source
    TPM1C0SC = TPM1C0SC_MS0A_MASK;// software timer, no port ctrl, clear CH0F flag
    TPM1C0V = TPM1CNT + repeattime;// set up timer period for repeat frame
    TPM1C0SC_CH0F = 0;// clear the flag
}
```

TimerResAndGo() function sets up the 16-bit TPM timer that will generate a software output compare event at specified interval from the current time. For example, calling **TimerResAndGo**(TPMCLK_MS(110.0)); will ensure that after 110 ms the software output compare occurs and can be detected using the next function:

```
void WaitTmrFlag(void)
{
    while(!TPM1C0SC_CH0F)// wait for Timer Flag
        __RESET_WATCHDOG(); /* kicks the dog */

    TPM1C0SC_CH0F = 0;// clear the flag
}
```

WaitTmrFlag() will wait till the software output compare (set up using **TimerResAndGo()**) event occurs. Then the function is terminated and the software control is returned back to the main flow.

The next function is related to synchronization of writes to the modulator block registers. The mark and space timing registers can only by updated after the **EOC** flag is set in the **CMTMSC** register.

```
void WaitEOC(void)
{
    while (!CMTMSC_EOCF)
        __RESET_WATCHDOG(); /* kicks the dog */
}
```

WaitEOC() function is used to wait till the modulator block takes over the values for the next cycle (as shown in Figure 10) so that the next values can be written.

Finally, the last function **CheckButton()** is used by the infrared driver to check the status of the remote control keyboard. This callback function should return a non-zero value if the same key is pressed, zero if released.

```
unsigned char CheckButton(unsigned int cmdCnt)
{
   __RESET_WATCHDOG(); /* kicks the dog */

//  scan buttons here and return non-zero if the same key is still pressed

  return (cmdCnt < 2);    // here just two presses **DEMO ONLY**
}
```

5.3 Pulse Width Protocol

Pulse width protocol, known also as SIRC, this protocol was developed by Sony. It uses pulse width encoding and amplitude modulation. The data payload consists of 7 bits command and 5 bits address. The data is preceded by a train pulse, 2.4 ms mark, and a 0.6 ms space in order to settle automatic gain control (AGC) of the receiver.

Log. ‘1’ is denoted as a 1200 μs mark period followed by a 600 μs space period

Log. ‘0’ is denoted as a 600 μs mark period followed by a 600 μs space period

Carrier frequency is 40 kHz.

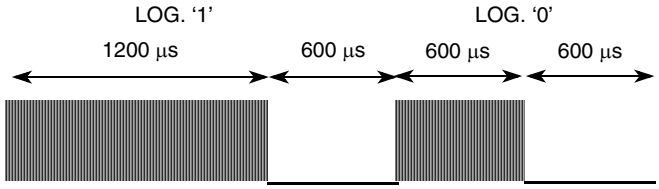

Figure 16. Pulse Width Protocol, Bit Encoding

Overall data frame structure is shown in Figure 17.

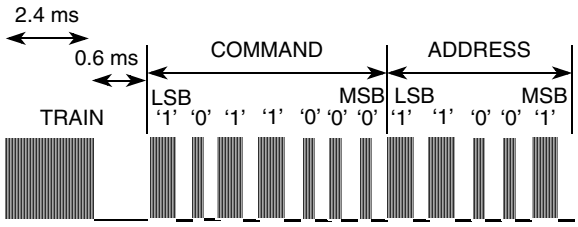

Figure 17. Pulse Width Protocol, Data Frame Structure

Autorepeat function for this protocol is handled by repeating the data frames The frames are repeated every 45 ms until the same key is pressed on the remote control. The full sequence is shown in Figure 18.

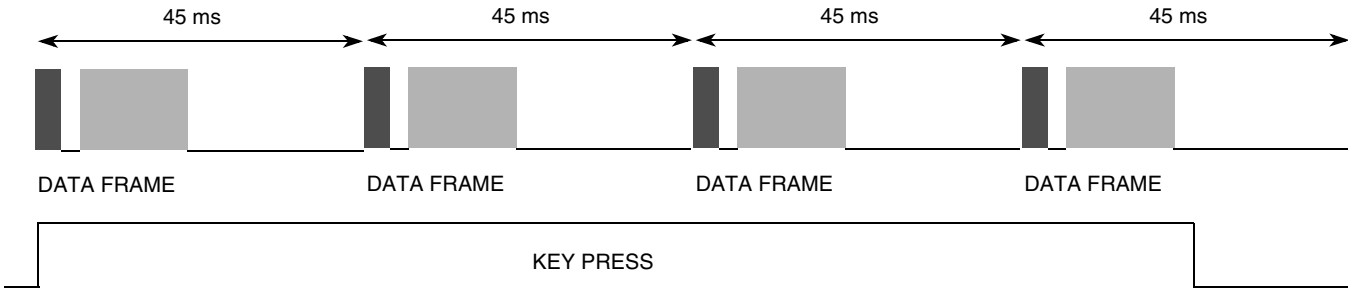

Figure 18. Pulse Width Protocol, Full Sequence Structure

5.3.1 Pulse Width Protocol Source Code

```
#define   SIRC_REPEAT_PERIOD       TPMCLK_MS(45.0)
#define   SIRC_CARRIER_FREQUENCY   40000              // [Hz]
#define   SIRC_IR_RATIO            33                 // [%]
#define   SIRC_PULSETRAIN_ON       2400.0             // usec
#define   SIRC_PULSETRAIN_OFF      600.0              // usec

#define   SIRC_DATA_OFF            600.0              // usec
#define   SIRC_DATA_1_ON           1200.0             // usec
#define   SIRC_DATA_0_ON           600.0              // usec

void SIRC_bits(unsigned char data, unsigned char bits)
{
    do {
        if (data & 0x01)
            ModClockOn(SIRC_DATA_1_ON);         // log. '1' mark
        else
            ModClockOn(SIRC_DATA_0_ON);         // log. '0' mark

        WaitEOC();                              // wait for CMT to load values
        data >>= 1;

    } while (--bits);
}

void SIRC_protocol(unsigned char address, unsigned char command)
{
unsigned int cmdCnt;

    cmdCnt = 0;     /* command counter */

    // set up Carrier generator
    CGClockOn(SIRC_CARRIER_FREQUENCY, SIRC_IR_RATIO);
    CGClockOff(SIRC_CARRIER_FREQUENCY, SIRC_IR_RATIO);

    do {
        TimerResAndGo(SIRC_REPEAT_PERIOD);

        // train pulse
        ModClockOn(SIRC_PULSETRAIN_ON);         // on-time train pulse
        ModClockOff(SIRC_PULSETRAIN_OFF);       // off-time train pulse

        CMTEnable(CMTMSC_MCGEN_MASK);  // enable & start CMT
        WaitEOC();                              // wait for CMT to load values

        ModClockOff(SIRC_DATA_OFF);             // off-time always the same

        // send adress and data
        SIRC_bits(command, 7);
        SIRC_bits(address, 5);

        CMTDisable();                           // disable CMT entirely
        WaitTmrFlag();
    } while (CheckButton(++cmdCnt));// see whether the key is still pressed (to send repeats)
}
```

5.4 Manchester Protocol (RC5)

RC5 protocol has been developed by Philips and is one of the most popular among hobbyists. It uses the Manchester (Biphase) encoding and amplitude modulation. The data payload consists of 5 bits address and 6 bits command.

The data is preceded by two start log. ‘1’ bits (S1 and S2) and one toggle bit (T). The toggle bit changes between ‘1’ and ‘0’ between separate key presses in order for the receiver to distinguish a long key press from several short key presses.

Log. ‘1’ is denoted as a 889 μs space period followed by a 889 μs mark period.

Log. ‘0’ is denoted as a 889 μs mark period followed by a 889 μs space period.

Each bit is 1778 μs long. Carrier frequency is 36 kHz.

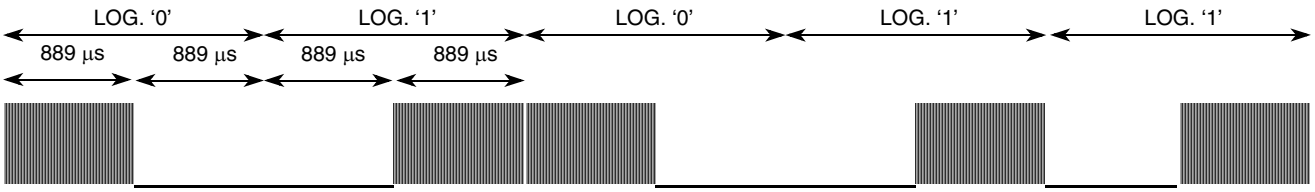

Figure 19. RC5 Protocol, Bit Encoding

Overall data frame structure is shown in Figure 20.

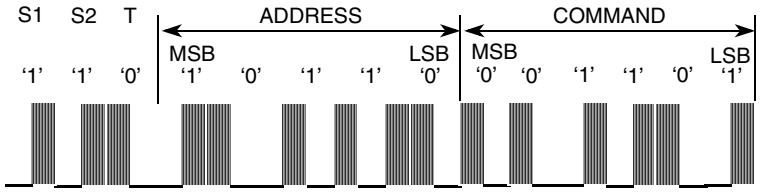

Figure 20. RC5 Protocol, Data Frame Structure

Autorepeat function for this protocol is handled by repeat data frames with the same toggle bit. If ever the key is pressed again, the next transmission holds the same data but the toggle bit is changed from ‘1’ to ‘0’ or vice versa.

There is also an extended version of this protocol, where the S2 start bit is no longer fixed but is interpreted rather as an inverted 6th address bit. The full sequence is shown in Figure 21.

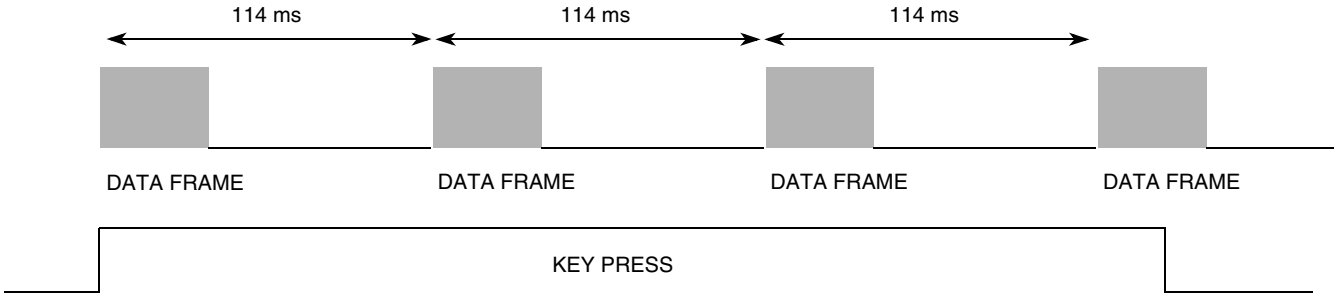

Figure 21. RC5 Protocol, Full Sequence Structure

5.4.1 Manchester Encoding Implementation

Since the CMT module allows generating one full cycle (see Figure 10) that is composed from a mark period followed by a space period, an extended space feature of the CMT module needs to be used to generate a Manchester log. ‘1’ (space followed by mark). The EXSPC bit in the CMTMSC register must be set to use this feature. The extended space cycle is the same as the regular cycle but no carrier is generated during the mark period. Thus, to generate a Manchester log. ‘1’, two cycles are required:

- First cycle has the extended space bit set, mark period equal to half-bit time, space period of zero.
- Second cycle has the extended space bit cleared, mark period equal to half-bit time, space period of zero too.

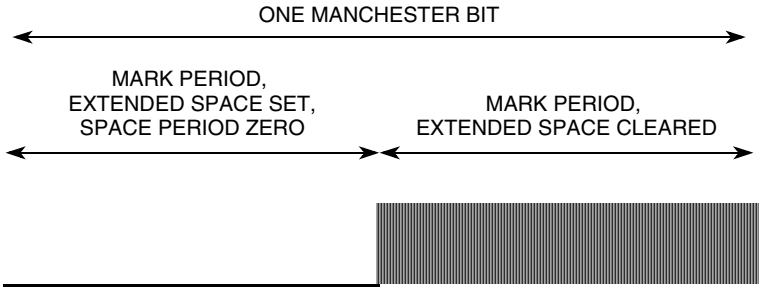

Figure 22. Manchester Encoding, Generation Space Followed by Mark

Log. ‘0’ is generated as a regular time-mode cycle with both mark and space periods equal to a half-bit time,

5.4.2 RC5 Protocol Source Code

```
#define   RC5_REPEAT_PERIOD       TPMCLK_MS(114.0)
#define   RC5_CARRIER_FREQUENCY   36000              // [Hz]
#define   RC5_IR_RATIO            33                 // [%]

#define   RC5_HALF_BIT            889.0              // usec
#define   RC5_MODE                0                  // no enable until first bit


void RC5_bits(unsigned char data, unsigned char bits)
{
    do {
        if (data & 0x80)
        {   /* Manchester bit 0 - ie. 01 '_M', extended space + second mark */
            ModClockOn(RC5_HALF_BIT);
            ModClockOff(0);
            CMTMSC_EXSPC = 1;       // extended space for half bit
            CMTMSC_MCGEN = 1;       // start CMT if needed
            WaitEOC();

            CMTMSC_EXSPC = 0;       // no extended space for half bit
            ModClockOff(0);
        }
        else
        {   /* Manchester bit 1 - ie. 10 'M_', normal mark/space */
```

```
            ModClockOn(RC5_HALF_BIT);
            ModClockOff(RC5_HALF_BIT);
            CMTMSC_MCGEN = 1;   // start CMT if needed
        }

        data <<= 1;                          // MSB first
        WaitEOC();
    } while (--bits);
}

void RC5_protocol(unsigned char address, unsigned char command)
{
unsigned int cmdCnt;
static unsigned char toggle;

    cmdCnt = 0;     /* command counter */
    toggle ^= 0xFF; /* toggle byte for this session */

    // set up Carrier generator
    CGClockOn(RC5_CARRIER_FREQUENCY, RC5_IR_RATIO);
    CGClockOff(RC5_CARRIER_FREQUENCY, RC5_IR_RATIO);

    do {
        TimerResAndGo(RC5_REPEAT_PERIOD);

        CMTEnable(RC5_MODE);                 // no-enable & just start CMT

        // send start bits & toggle
        RC5_bits(0xC0 | (toggle & 0x20), 3);
        // b7, b6 are constant 1 startbits, b5 is toggle bit
        // send adress and data
        RC5_bits(address<<3, 5);
        RC5_bits(command<<2, 6);

        CMTDisable();                       // disable CMT entirely

        WaitTmrFlag();
    } while (CheckButton(++cmdCnt));         // see whether the key is still pressed
                                               (to send repeats)
}
```

5.5 Flash Protocol

Flash protocol was developed by ITT. It uses Flash (‘Pulse’) modulation. The data payload consists of 4 bits address and 6 bits command. The data is preceded by a lead-in pulse and a start pulse (log. ‘0’), and finalized by a lead-out pulse. All pulses are 10 μs long, the distance between pulses are 100 μs for log. ‘0’, 200 μs for log. ‘1’, and 300 μs for lead-in and lead-out pulses.

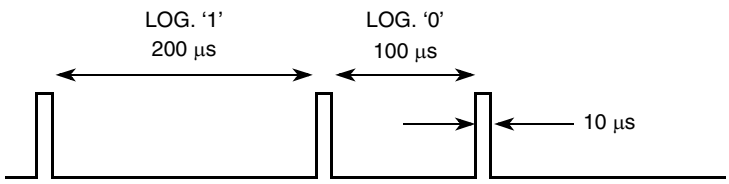

Figure 23. Flash Protocol, Bit Encoding

Overall data frame structure is shown in Figure 24.

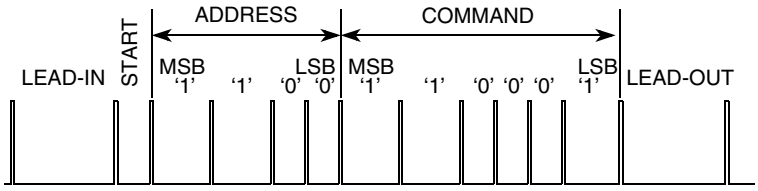

Figure 24. Flash Protocol, Data Frame Structure

Autorepeat function for this protocol is handled by repeating the data frames The frames are repeated every 130 ms until the same key is pressed on the remote control. The full sequence is shown in Figure 25

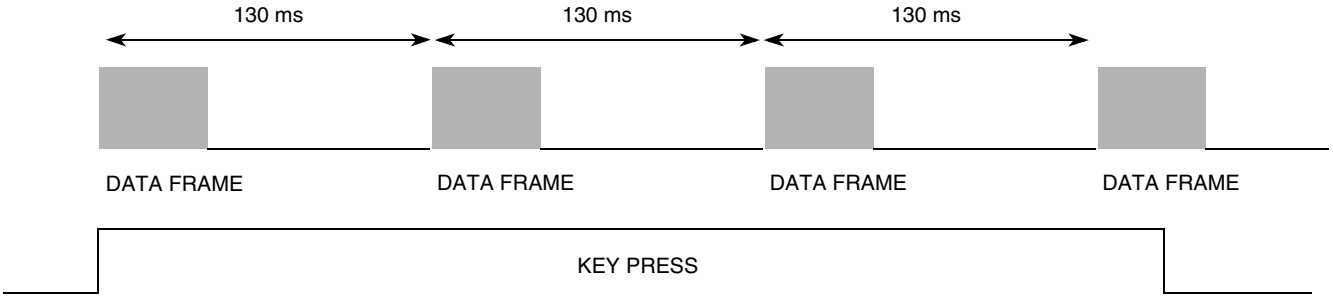

Figure 25. Flash Protocol, Full Sequence Structure

5.5.1 Flash Modulation Implementation

The CMT module also allows generating also Flash modulation. In CMT terminology this is called a baseband mode and the configuration is very similar to the time mode. The carried generator is not used, so its registers are not configured. To use baseband mode, the BASE bit in the CMTMSC register must be set.

5.5.2 Flash Protocol Source Code

```
#define FLASH_REPEAT_PERIOD      TPMCLK_MS(130.0)
#define FLASH_LEAD_IN_ON         10.0              // usec
#define FLASH_LEAD_IN_OFF        300.0             // usec
#define FLASH_LEAD_OUT_ON        10.0              // usec
#define FLASH_LEAD_OUT_OFF       300.0             // usec

#define FLASH_DATA_ON            10.0              // usec
#define FLASH_DATA_1_OFF         200.0             // usec
#define FLASH_DATA_0_OFF         100.0             // usec
```

```
void FLASH_bits(unsigned char data, unsigned char bits)
{
  do
  {
        if (data & 0x01)
            ModClockOff(FLASH_DATA_1_OFF);      // log. '1' space
        else
            ModClockOff(FLASH_DATA_0_OFF);      // log. '0' space

        WaitEOC();                            // wait for CMT to load values
        data >>= 1;

    } while (--bits);
}

void FLASH_protocol(unsigned char address, unsigned char command)
{
unsigned int cmdCnt;

    cmdCnt = 0;     /* command counter */

    do {
        TimerResAndGo(FLASH_REPEAT_PERIOD);

        // lead-in pulse
        ModClockOn(FLASH_LEAD_IN_ON);// on-time lead-in pulse
        ModClockOff(FLASH_LEAD_IN_OFF);// off-time lead-in pulse

        CMTEnable(CMTMSC_BASE_MASK | CMTMSC_MCGEN_MASK);  // enable base mode & start CMT
        WaitEOC();                                          // wait for CMT to load values

        ModClockOn(FLASH_DATA_ON);    // on-time always the same

        // send adress and data
        FLASH_bits(0, 1);                // start bit, zero
        FLASH_bits(address, 4);
        FLASH_bits(command, 6);

        // lead-out pulse
        ModClockOn(FLASH_LEAD_OUT_ON);// on-time lead-out pulse
        ModClockOff(FLASH_LEAD_OUT_OFF);// off-time lead-out pulse
        WaitEOC();                                          // wait for CMT to load values

        // tail pulse
        ModClockOn(FLASH_LEAD_OUT_ON);// on-time tail pulse
        ModClockOff(0);                // off-time tail pulse, zero
        WaitEOC();                          // wait for CMT to load values

        CMTDisable();                     // disable CMT entirely

        WaitTmrFlag();
    } while (CheckButton(++cmdCnt));// see whether the key is still pressed (to send repeats)
}
```

5.6 CMT Interrupts

The examples above are based on a polling method (the software polls the various flags until another write to the registers is possible). Such a method is more descriptive so it was selected for the purpose of the application note. Often, this method is sufficient for most implementations.

Anyway, a CMT module allows generating interrupts at the end of each cycle and creating software that is completely interrupt driven. The interrupt is generated whenever the end of cycle flag (EOCF) is set. A description of interrupt driven software is out with the scope of this document. The full description of the CMT interrupts can be found in the MC9S08RC/RD/RE/RG data sheet.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2005, 2008. All rights reserved.

AN3053
Rev. 1, 03/2008

freescale™
semiconductor