AN10722

RC5 decoder using the LPC2000

Rev. 01 — 16 July 2008 Application note

Document information

Info	Content
Keywords	LPC2000, ARM7, RC5 decoder, Infrared Remote Control
Abstract	This application note demonstrates the use of a low cost ARM7 based NXP microcontroller for receiving and decoding RC5 commands.

Revision history

Rev	Date	Description
01	20080716	Initial version.

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The RC5 protocol has been developed to offer a unified infrared (IR) remote control system for equipment used in and around domestic environments.

To ensure immunity to interference from other IR sources such as the sun, lamps and IR sound transmissions (for example to headphones), bi-phase encoding (also called Manchester encoding) is used for RC5 code words. As shown in Fig 1 each bi-phase encoded bit is a symbol comprising two logic levels with a transition in the middle. The bi-phase code words modulate a 36 kHz carrier, before being transmitted via the IR LED. Since the repetition period of the 36 kHz carrier is 27.778 us and the high part of each bit of the RC5 code word contains 32 carrier pulses, 1 bit period is 64 x 27.778 us = 1.778 ms. A complete RC5 code word (one message) contains 14 bits, so it takes 24.889 ms to transmit. Each 14 bit RC5 code word consists of:

- a start bit (S) which is always logic 1
- a field bit (F) which denotes command codes 0 to 63 or 64 to 127
- a control bit (C) which toggles and initiates a new transmission
- five system address bits for selecting one of 32 possible systems
- six command bits representing one of the 128 possible RC5 commands

Table 1 below, shows an overview of the data pulse-width tolerances, used in this application note.

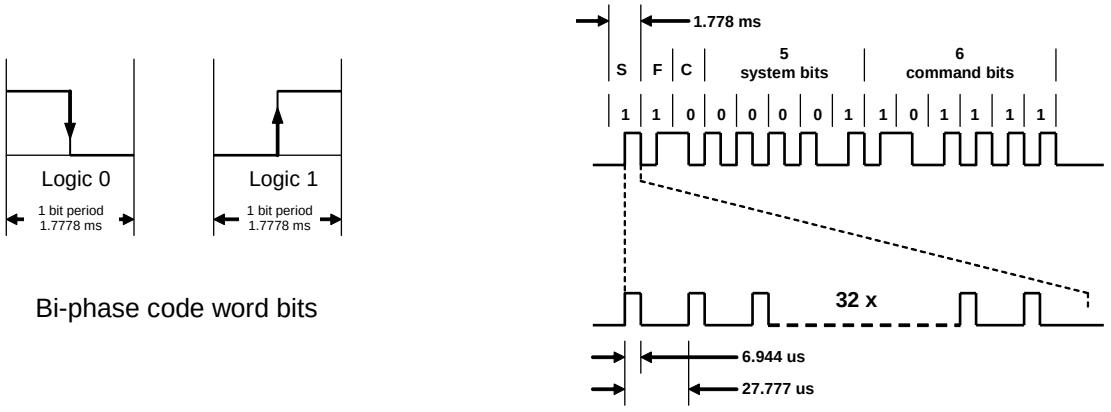

Fig 1. RC5 code word example

Table 1. RC5 pulse-width tolerances

Description	Min	Typical	Max	Unit
RC5 Half bit period	640	889	1140	µsec
RC5 Full bit period	1340	1778	2220	µsec
RC5 message time	23.644	24.889	26.133	msec
RC5 message repetition time	108.089	113.778	119.467	msec
Carrier pulse bit time	27.233	27.778	28.345	µsec

2. Hardware

The hardware setup to test the RC5 decoder is very easy. Timer 0, capture 0 input of an LPC2141 (see Fig 2) is used. This input can capture the current timer value both at falling and rising edges as well as generate an interrupt on both edges. This feature makes it easy to measure the RC5 pulse high and low times. Furthermore, the RC5 input is connected to a general purpose input pin, used to determine whether a rising or falling edge interrupt has occurred.

UART 0 of the LPC2141 is used to send incoming RC5 messages, out via an RS232 interface (19200 baud), to for example a PC (HyperTerminal).

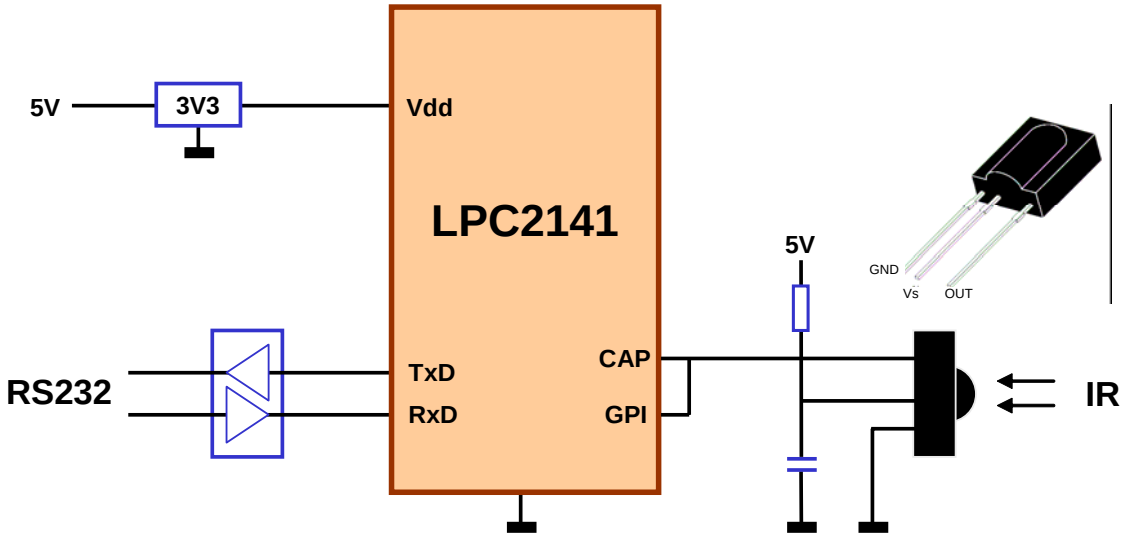

Fig 2. RC5 decoder test application

3. Software

The RC5 decoder code example is written in C language and compiled using Keil's uVision (ARM7 RealView, V3.2) free demo compiler. It performs following main tasks:

- Initialization: for LPC2141 configuration the standard startup code from Keil was used and set as CCLK = PCLK = 60 MHz (startup.s not listed)
- Receiving RC5 messages using Timer 0. Input clock to the timer is set to 1 µsec (using the prescale register). See rc5.c module listed below
- Sending of received RC5 messages to PC terminal using UART0 at 19200 baud, see main.c and uart.c, listed below

3.1 main.c

```
1   #include <LPC214x.H>                          // LPC214x definitions
2   #include "main.h"
3
4   int main (void)
5   {
6       UART0_Init(19200);
7       RC5_Init();
8
9       PrintString("\f\nLPC2148-RC5 test June 2008\n\n");
10
11      while (1)
12      {
13          if (RC5_flag)                         // wait for RC5 code
14          {
15              RC5_flag = 0;
16              PrintString("RC5 = ");            // and print it
17              PrintByte(RC5_System);
18              PrintString(" ");
19              PrintByte(RC5_Command);
20              PrintString("\n");
21          }
22      }
23  }
```

3.2 uart.c

```
1   #include <LPC214x.h>
2   #include "main.h"
3
4   const char ascii[] = "0123456789ABCDEF";
5
6   void UART0_Init(unsigned int baudrate)
7   {
8     unsigned int brd = (Fpclk / (baudrate << 4));
9
10      PINSEL0 |= 0x00000005;                    // Select UART0 RXD / TXD
11
12      U0FCR = 7;                                // Enable and clear FIFO's
13      U0LCR = 0x83;                             // Set DLAB and set word format to 8-N-1
14      U0DLL = (brd & 0xFF);                     // Set baud rate dividers
15      U0DLM = (brd >> 8);                       //
16      U0LCR = 3;                                // Disable Divisor latch bit
17  }
18
19  static void ua_outchar(char c)
20  {
21      U0THR = c;
22      while (!(U0LSR & 0x40));
23  }
24
25  void PrintByte(unsigned char b)
26  {
27      ua_outchar(ascii[b >> 4]);
28      ua_outchar(ascii[b & 0x0f]);
29  }
30
31  void PrintString(const char *s)
32  {
33      while (*s)
```

```
34      {
35          if (*s == '\n')
36              ua_outchar('\r');                   // output a '\r' first
37          ua_outchar(*s);
38          s++;
39      }
40  }
```

3.3 rc5.c

```
1   /*****************************************************************************
2   ;  LPC2000 - RC5 decoder
3   ;
4   ;  This package uses T0-CAP0 input (capture and interrupt on both edges)
5   ;  CAP0.0 (P0.30) is connected to P0.16 (to check high / low level by software)
6   ;  RC5 format:
7   ;
8   ;      | S | F | C |   5 system bits   |    6 command bits     |
9   ;      | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
10  ;
11  ;   -----+ +-+   +-+ +-+ +-+ +-+ +---+ +-+   +---+ +-+ +-+ +-+ +----
12  ;        | | |   | | | | | | | | | |   | | |   |   | | | | | | | |
13  ;        +-+ +---+ +-+ +-+ +-+ +-+   +-+ +---+   +-+ +-+ +-+ +-+
14  ;
15  *****************************************************************************/
16  #include <LPC214x.h>                            // LPC214x definitions
17
18  #define MIN_HALF_BIT       640                  // 640 us
19  #define HALF_BIT_TIME      889                  // 889 us
20  #define MAX_HALF_BIT       1140                 // 1140 us
21  #define MIN_FULL_BIT       1340                 // 1340 us
22  #define FULL_BIT_TIME      1778                 // 1778 us
23  #define MAX_FULL_BIT       2220                 // 2220 us
24
25  unsigned char RC5_System;                       // Format 1 E/N t  s4 s3 s3 s1 s0
26  unsigned char RC5_Command;                      // Format 0  0  c5 c4 c3 c2 c1 c0
27  unsigned char RC5_flag;
28
29  static signed int    low_time;
30  static signed int    high_time;
31  static unsigned char half_bit;
32  static unsigned char sys;                       // temp system byte
33  static unsigned char cmd;                       // temp Command byte
34
35
36  static void RC5_Shift_Bit(char val)
37  {
38      if (sys & 0x80)
39      {
40          if (cmd & 0x80)                         // command full ?
41          {
42              sys = 0;                            // yes, ERROR
43              cmd = 0;
44          }
45          else
46              cmd = (cmd << 1) | val;             // shift command
47      }
48      else
49          sys = (sys << 1) | val;                 // shift system
50  }
51
52
```

```
/*****************************************************************************
; RC5_Decode (we only take action at a rising edge)
; Half(prev) Bit   Low Time        High Time      Action     New Half Bit
; ----------------------------------------------------------------------
;     0              0               0          Shift 0         0
;     0              0               1          Shift 1         1
;     0              1               0          -ERROR-         *
;     0              1               1          Shift 1,0       0
;     1              0               0          Shift 1         1
;     1              0               1          -ERROR-         *
;     1              1               0          Shift 1,0       0
;     1              1               1          -ERROR-         *
*****************************************************************************/
static void RC5_Decode(void)
{
  unsigned char action;

    action = half_bit << 2;

    if ((high_time > MIN_FULL_BIT) && (high_time < MAX_FULL_BIT))
        action = action | 1;                    // high_time = long
    else if (!((high_time > MIN_HALF_BIT) && (high_time < MAX_HALF_BIT)))
    {
        sys = 0;                                // RC5 ERROR
        cmd = 0;
        return;
    }

    if ((low_time > MIN_FULL_BIT) && (low_time < MAX_FULL_BIT))
        action = action | 2;                    // low_time = long
    else if (!((low_time > MIN_HALF_BIT) && (low_time < MAX_HALF_BIT)))
    {
        sys = 0;                                // RC5 ERROR
        cmd = 0;
        return;
    }

    switch (action)
    {
      case 0:  RC5_Shift_Bit(0);                // short low, short high, shift 0
               break;
      case 1:  RC5_Shift_Bit(1);                // short low, long high, shift 1
               half_bit = 1;                    // new half bit is true
               break;
      case 2:  sys = 0;                         // long low, short high, ERROR
               cmd = 0;
      case 3:  RC5_Shift_Bit(1);                // long low, long high, shift 1,0
               RC5_Shift_Bit(0);
               break;
      case 4:  RC5_Shift_Bit(1);                // short low, short high, shift 1
               break;
      case 5:  sys = 0;                         // short low, long high, ERROR
               cmd = 0;
               break;
      case 6:  RC5_Shift_Bit(1);                // long low, short high, shift 1,0
               RC5_Shift_Bit(0);
               half_bit = 0;                    // new half bit is false
               break;
      case 7:  sys = 0;                         // long low, long high, ERROR
               cmd = 0;
      default: break;                           // invalid
    }
}
```

```
116
117 __irq void RC5_Isr(void)
118 {
119     T0TC = 0;                                   // Reset timer
120
121     if (T0IR & 1)                               // Timeout ? to guarantee a 12 msec
122     {                                           // idle time after last RC5 pulse
123         if (cmd & 0x80)                         // command full ?
124         {
125             RC5_Command = cmd & 0x7F;           // OK! Save command byte
126             RC5_System = sys;                   // save system byte
127             RC5_flag = 1;                       // set event to application
128         }
129         sys = 0;
130         cmd = 0;
131         T0IR = 0x01;                            // clear MR0 interrupt flag
132     }
133     else                                        // capture interrupt
134     {
135         if (IO0PIN & 0x00010000)                // check P0.16, rising or falling edge
136         {
137             if (sys == 0)                       // First pulse ?
138             {
139                 low_time  = HALF_BIT_TIME;      // assume short low time
140                 high_time = HALF_BIT_TIME;      // assume short high time
141                 half_bit  = 1;                  // assume half bit is true
142                 cmd = 0x02;                     // = 00000010, prepare command byte
143             }
144             else
145                 low_time = T0CR0;               // rising, so capture low time
146
147             RC5_Decode();
148         }
149         else
150             high_time = T0CR0;                  // falling, so capture high time
151
152         T0IR = 0x10;                            // reset interrupt flag
153     }
154     VICVectAddr = 0;                            // Acknowledge interrupt by reseting VIC
155 }
156
157 void RC5_Init(void)
158 {
159     VICVectAddr0  = (unsigned int) &RC5_Isr;
160     VICVectCntl0  = 0x24;                       // Channel0 on Source#4 ... enabled
161     VICIntEnable |= 0x10;                       // Channel#4 is the Timer 0
162
163     PINSEL1 |= 0x30000000;                      // P0.30 as CAP0.0
164
165     T0PR  = 60;                                 // presc 60, timer runs at 1 MHz
166     T0MR0 = 12000;                              // 12 ms High (idle) Timeout
167     T0MCR = 3;                                  // Int on Match0, reset timer on match
168     T0CCR = 0x0007;                             // Capture and interrupt on both edges
169     T0TC  = 0;                                  // Reset timer
170     T0TCR = 1;                                  // start timer
171 }
```

4. Terminal output

Received RC5 messages are send out over an RS232 interface using UART0 of the LPC2141. Connected to a PC running HyperTerminal (19200 baud) the output screen is as show in Fig 3 below. The first value represents the RC5 System byte, the second value gives the RC5 Command byte.

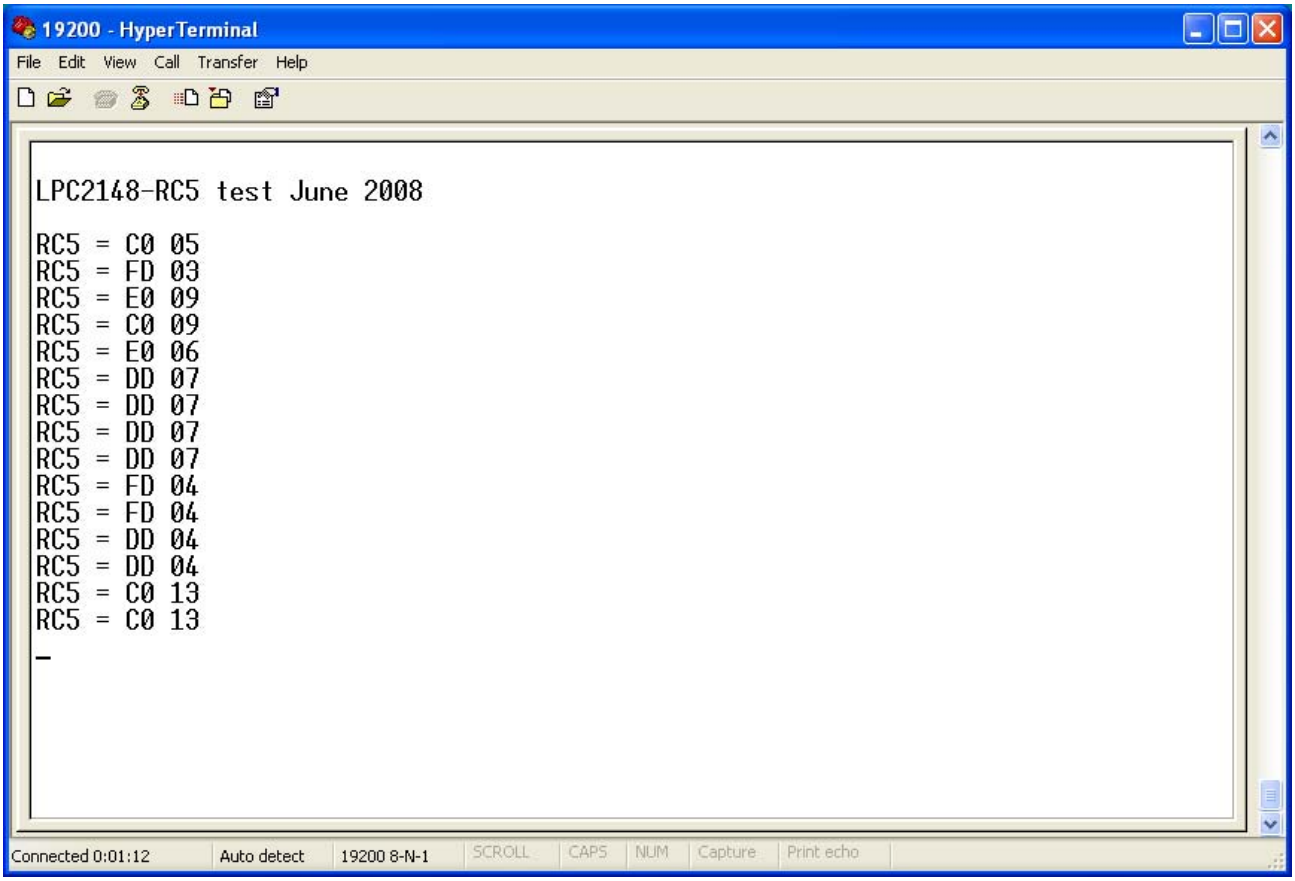

Fig 3. RC5 output at PC HyperTerminal

5. Legal information

5.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

5.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

5.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

6. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, email to: salesaddresses@nxp.com

Date of release: 16 July 2008
Document identifier: AN10722_1