Introduction

Most audio and video systems are equipped with an infrared remote control. This application note describes a receiver for the frequently used Philips/Sony RC5 coding scheme.

Features

- Low-cost
- Compact design, only one external component
- Requires only one controller pin, any AVR® device can be used
- Size-efficient code
- Complementary of the Atmel® AVR415 RC5 IR Remote Control Transmitter on Atmel tinyAVR® and megaAVR® devices
# Table of Contents

Introduction......................................................................................................................1

Features.......................................................................................................................... 1

1. RC5 Coding Scheme.................................................................................................3

2. Timing........................................................................................................................ 4

3. Software.....................................................................................................................5

4. Example Program......................................................................................................8

5. Revision History.......................................................................................................12
1. **RC5 Coding Scheme**

The RC5 code is a 14-bit word bi-phase coded signal as seen in the figure below. The first two bits are start bits, always having the value “1”. The next bit is a control bit, which is toggled every time a button is pressed on the remote control transmitter. This gives an easy way of determining whether a button is pressed and held down, or pressed and released continuously.

**Figure 1-1. RC5 Frame Format**

| S1 | S2 | Ctrl | S4 | S3 | S2 | S1 | S0 |

Five system bits hold the system address so that only the right system responds to the code. Usually, TV sets have the system address 0, VCRs the address 5, and so on. The command sequence is six bits long, allowing up to 64 different commands per address. The bits are transmitted in bi-phase code (also known as Manchester code) as shown below, along with an example where the command 0x35 is sent to system 5.

**Figure 1-2. Bi-phase Coding**

```
| 1 | 0 |
```

**Figure 1-3. Example of Transmission**

```
1 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1
```

Note that the figures above show the signal that enters the Atmel ATtiny28 hardware modulator. The actual signal emitted by the IR-LED will be modulated with a certain carrier frequency as shown in the figure below.

**Figure 1-4. Signal Before and After Modulation**

```
Carrier Frequency
```

```
1 1 0 0
```

```
1 1 0 0
```

\[\text{HM}\]
2. **Timing**

The bit length is approximately 1.8ms. The code is repeated every 114ms. To improve noise rejection, the pulses are modulated at 36kHz. This can be seen in the RC5 IR Receiver figure below. The easiest way to receive these pulses is to use an integrated IR-receiver/demodulator like the Siemens SFH 506-36. This is a 3-pin device that receives the infra-red burst and gives out the demodulated bit stream at the output pin. Note that the data is inverted compared to the transmitted data (i.e., the data is idle high). The output of the demodulator device is connected to PD2 on the AVR device. The decoded instructions in this case will be output on PORTB, but the chosen port is easily reconfigurable.

**Figure 2-1. RC5 Receiver**
3. **Software**

The assembly code found in AVR410.ASM contains the RC5 decode routine. In addition, it contains an example program which initializes the resources, decodes the RC5 data and outputs the received command on PORTB.

**The Detect Subroutine**

When the detect subroutine is called, it first waits for the data line to be idle high for more than 3.5ms. Then, a start bit can be detected. The length of the low part of the first start bit is measured. If no start bit is detected within 131ms, or if the low pulse is longer than 1.1ms, the routine returns indicating no command received.

The measurement of the start bit is used to calculate two reference times, ref1 and ref2, which are used when sampling the data line. The program uses the edge in the middle of every bit to synchronize the timing. 3/4 of a bit length after this edge, the line is sampled. This is in the middle of the first half of the next bit (see the figure below). The state is stored and the routine waits for the middle edge. Then, the timer is synchronized again and the steps are repeated for the following bits. If the synchronizing edge is not detected within 5/4 bit times from the previous synchronizing edge, this is detected as a fault and the routine terminates. When all the bits are received, the command and system address are stored in the “command” and “system” registers. The control bit is stored in bit 6 of “command”.

![Synchronizing and Sampling of the Data](image)

**Table 3-1. "Decode" Subroutine Performance Figures**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code size</td>
<td>72 words</td>
</tr>
<tr>
<td>Register usage</td>
<td>Low registers used: 3</td>
</tr>
<tr>
<td></td>
<td>High registers used: 6</td>
</tr>
<tr>
<td></td>
<td>Global registers: 6</td>
</tr>
<tr>
<td></td>
<td>Pointers used: None</td>
</tr>
</tbody>
</table>

**Table 3-2. "Detect" Register Usage**

<table>
<thead>
<tr>
<th>Register</th>
<th>Internal</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>&quot;inttemp&quot; - Used by TIM0_OVF</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>&quot;ref1&quot; - Hold timing information</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>&quot;ref2&quot; - Hold timing information</td>
<td></td>
</tr>
<tr>
<td>R16</td>
<td>&quot;temp&quot; - Temporary Register</td>
<td></td>
</tr>
<tr>
<td>R17</td>
<td>&quot;timerL&quot; - Timing Register</td>
<td></td>
</tr>
<tr>
<td>R18</td>
<td>&quot;timerH&quot; - Timing Register</td>
<td></td>
</tr>
</tbody>
</table>
Timer/Counter 0 Overflow Interrupt Handler
The function of the timer interrupt is to generate a clock base for the timing required. The routine increments the “timerL” Register every 64μs, and the “timerH” every 16.384ms.

Table 3-3. "TIM0_OVF" Interrupt Handler Performance Figures

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code size</td>
<td>7 words</td>
</tr>
<tr>
<td>Execution cycles</td>
<td>6 + reti</td>
</tr>
<tr>
<td>Register usage</td>
<td>Low registers used: 2</td>
</tr>
<tr>
<td></td>
<td>High registers used: 2</td>
</tr>
<tr>
<td></td>
<td>Global registers: 0</td>
</tr>
<tr>
<td></td>
<td>Pointers used: None</td>
</tr>
</tbody>
</table>

Table 3-4. "TIM0_OVF" Register Usage

<table>
<thead>
<tr>
<th>Register</th>
<th>Internal</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>&quot;S&quot; - Temporary Storage of Sreg</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>&quot;inttemp&quot; - Used by TIM0_OVF</td>
<td></td>
</tr>
<tr>
<td>R17</td>
<td>&quot;timerL&quot; - Incremented every 64μs</td>
<td></td>
</tr>
<tr>
<td>R18</td>
<td>&quot;timerH&quot; - Incremented every 16.384ms</td>
<td></td>
</tr>
</tbody>
</table>

Example Program
The example program initializes the ports, sets up the timer, and enables interrupts. Then, the program enters an eternal loop, calling the detect routine. If the system address is correct, the command is output on PORTB.

Table 3-5. Overall Performance Figures

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code size</td>
<td>79 words - &quot;detect&quot; and &quot;TIM0_OVF&quot;</td>
</tr>
<tr>
<td></td>
<td>96 words - Complete example code</td>
</tr>
<tr>
<td>Register usage</td>
<td>Low registers: 4</td>
</tr>
<tr>
<td></td>
<td>High registers: 6</td>
</tr>
<tr>
<td></td>
<td>Pointers: None</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Interrupt usage</td>
<td>Timer/Counter 0 Overflow Interrupt</td>
</tr>
<tr>
<td>Peripheral usage</td>
<td>Timer/Counter 0</td>
</tr>
<tr>
<td></td>
<td>Port D, pin 2</td>
</tr>
<tr>
<td></td>
<td>Port B (example program only)</td>
</tr>
</tbody>
</table>
4. Example Program

```assembly
.rjmp reset

.TMRO_OVF:
    in S,sreg ; Store SREG
    inc timerL ; Updated every 64us
    inc inttemp
    brne TMRO_OVF_exit

    inc timerH ; if 256th int inc timer

.TMRO_OVF_exit:
    out sreg,S ; Restore SREG
    reti
```

* Example program
* Version :1.0
* Initializes timer, ports and interrupts.

---

Atmel AV410: RC5 IR Remote Control Receiver on tinyAVR and megaAVR devices [APPLICATION NOTE]

Atmel-1473C-RC5-IR-Remote-Control-Receiver-on-tinyAVR-and-megaAVR-Devices_AV410_Application_Note-08/2016
**Calls "detect" in an endless loop and puts the result out on port B.**

**Number of words: 16**

**Low registers used: 0**

**High registers used: 3**

**Pointers used: 0**

```
reset:
    ;ldi temp,low(RAMEND) ;Initialize stackpointer for parts with SW stack
    ;out SPL,temp
    ;ldi temp,high(RAMEND) ; Commented out since 1200 does not have SRAM
    ;out SPH,temp
    ldi temp,1 ;Timer/Counter 0 clocked at CK
    out TCCR0,temp
    ldi temp,1<<TOIE0 ;Enable Timer0 overflow interrupt
    out TIMSK,temp
    ser temp ; PORTB as output
    sei ;Enable global interrupt

main:
    rcall detect ;Call RC5 detect routine
    cpi system,SYS_ADDR ;Responds only at the specified address
    brne release
    andi command,0x3F ;Remove control bit
    out PORTB,command
    rjmp main

release:
    clr command ;Clear PORTB
    out PORTB,command
    rjmp main
```
start1:
cpi            timerH,8          ;If no start bit detected
brge          fault             ;within 130ms then exit
sbic          PIN0,INPUT        ;Wait for start bit
rjmp          start1
clr            timerL            ;Measure length of start bit

start2:
cpi            timerL,17        ;If startbit longer than 1.1ms, 
brge          fault             ;exit
sbis          PIN0,INPUT        ;Positive edge of 1st start bit
rjmp          start2             ;Wait for falling edge start bit 2
mov            temp,timerL       ;timer is 1/2 bit time
clr            timerL
mov            ref1,temp
lsl            ref1
add            ref1,temp         ;ref1 = 3/4 bit time
lsr            temp
add            ref2,temp         ;ref2 = 5/4 bit time

start3:
cp            timerL,ref1       ;If high period St2 > 3/4 bit time
brge          fault             ;exit
sbic          PIN0,INPUT        ;Wait for falling edge start bit 2
rjmp          start3

ldi          bitcnt,12          ;Receive 12 bits
clr            command
clr            system

sample:
cp            timerL,ref1       ;Sample INPUT at 1/4 bit time
brlo          sample
sbic          PIN0,INPUT        ;Jump if line high

bit_is_a_0:
clc            command            ;Store a '0'
rol            command
rol            system            ;Synchronize timing

bit_is_a_0a:
cp            timerL,ref2       ;If no edge within 3/4 bit time
brge          fault             ;exit
sbis          PIN0,INPUT        ;Wait for rising edge
rjmp          bit_is_a_0a       ;in the middle of the bit
clr            timerL
rjmp          nextbit

bit_is_a_1:
sec            command            ;Store a '1'
rol            command
rol            system            ;Synchronize timing

bit_is_a_1a:
cp            timerL,ref2       ;If no edge within 3/4 bit time
brge          fault             ;exit
sbic          PIN0,INPUT        ;Wait for falling edge
rjmp          bit_is_a_1a       ;in the middle of the bit
clr            timerL

nextbit:
dec          bitcnt              ;If bitcnt > 0
brne          sample            ;get next bit
mov            temp,command     ;All bits sucessfully received!
rol            temp             ;Place system bits in "system"
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rol</td>
<td>system</td>
<td></td>
</tr>
<tr>
<td>rol</td>
<td>temp</td>
<td></td>
</tr>
<tr>
<td>rol</td>
<td>system</td>
<td></td>
</tr>
<tr>
<td>bst</td>
<td>system, 5</td>
<td>;Move toggle bit</td>
</tr>
<tr>
<td>bld</td>
<td>command, 6</td>
<td>;to &quot;command&quot;</td>
</tr>
<tr>
<td>andi</td>
<td>command, 0b0111111</td>
<td>;Clear remaining bits</td>
</tr>
<tr>
<td>andi</td>
<td>system, 0x1F</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fault:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ser</td>
<td>command</td>
<td>;Both &quot;command&quot; and &quot;system&quot;</td>
</tr>
<tr>
<td>ser</td>
<td>system</td>
<td>;0xFF indicates failure</td>
</tr>
<tr>
<td>ret</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. **Revision History**

<table>
<thead>
<tr>
<th>Doc Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1473C</td>
<td>08/2016</td>
<td>New template and some minor changes</td>
</tr>
<tr>
<td>1473B</td>
<td>05/2002</td>
<td></td>
</tr>
<tr>
<td>1473A</td>
<td>08/2016</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>