

```
1 /**
2 * Arduino DCF77 decoder v0.2
3 * Copyright (C) 2006 Mathias Dalheimer (md@gonium.net)
4 *
5 * This program is free software; you can redistribute it and/or modify
6 * it under the terms of the GNU General Public License as published by
7 * the Free Software Foundation; either version 2 of the License, or
8 * any later version.
9 *
10 * This program is distributed in the hope that it will be useful,
11 * but WITHOUT ANY WARRANTY; without even the implied warranty of
12 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
13 * GNU General Public License for more details.
14 *
15 * You should have received a copy of the GNU General Public License along
16 * with this program; if not, write to the Free Software Foundation, Inc.,
17 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
18 */
19
20 /**
21 * Where is the DCF receiver connected?
22 */
23 #define DCF77PIN 2
24 /**
25 * Where is the LED connected?
26 */
27 #define BLINKPIN 13
28 /**
29 * Turn debugging on or off
30 */
31 #define DCF_DEBUG 1
32 /**
33 * Number of milliseconds to elapse before we assume a "1",
34 * if we receive a falling flank before - its a 0.
35 */
36 #define DCF_split_millis 140
37 /**
38 * There is no signal in second 59 - detect the beginning of
39 * a new minute.
40 */
41 #define DCF_sync_millis 1200
42 /**
43 * Definitions for the timer interrupt 2 handler:
44 * The Arduino runs at 16 Mhz, we use a prescaler of 64 -> We need to
45 * initialize the counter with 6. This way, we have 1000 interrupts per second.
46 * We use tick_counter to count the interrupts.
47 */
48 #define INIT_TIMER_COUNT 6
49 #define RESET_TIMER2 TCNT2 = INIT_TIMER_COUNT
50 int tick_counter = 0;
51 int TIMSK;
52 int TCCR2;
53 int OCIE2;
54 /**
55 * DCF time format struct
56 */
57 struct DCF77Buffer {
58     unsigned long long prefix :21;
59     unsigned long long Min :7; // minutes
60     unsigned long long P1 :1; // parity minutes
61     unsigned long long Hour :6; // hours
62     unsigned long long P2 :1; // parity hours
63     unsigned long long Day :6; // day
64     unsigned long long Weekday :3; // day of week
65     unsigned long long Month :5; // month
66     unsigned long long Year :8; // year (5 -> 2005)
```

```

67     unsigned long long P3 :1; // parity
68 }
69 struct {
70     unsigned char parity_flag :1;
71     unsigned char parity_min :1;
72     unsigned char parity_hour :1;
73     unsigned char parity_date :1;
74 } flags;
75 /**
76 * Clock variables
77 */
78 volatile unsigned char DCFSignalState = 0;
79 unsigned char previousSignalState;
80 int previousFlankTime;
81 int bufferPosition;
82 unsigned long long dcf_rx_buffer;
83 /**
84 * time vars: the time is stored here!
85 */
86 volatile unsigned char ss;
87 volatile unsigned char mm;
88 volatile unsigned char hh;
89 volatile unsigned char day;
90 volatile unsigned char mon;
91 volatile unsigned int year;
92
93
94 /**
95 * used in main loop: detect a new second...
96 */
97 unsigned char previousSecond;
98
99 /**
100 * Initialize the DCF77 routines: initialize the variables,
101 * configure the interrupt behaviour.
102 */
103 void DCF77Init() {
104     previousSignalState=0;
105     previousFlankTime=0;
106     bufferPosition=0;
107     dcf_rx_buffer=0;
108     ss=mm=hh=day=mon=year=0;
109 #ifdef DCF_DEBUG
110     Serial.println("Initializing DCF77 routines");
111     Serial.print("Using DCF77 pin #");
112     Serial.println(DCF77PIN);
113     pinMode(BLINKPIN, OUTPUT);
114     pinMode(DCF77PIN, INPUT);
115 #endif
116     pinMode(DCF77PIN, INPUT);
117 #ifdef DCF_DEBUG
118     Serial.println("Initializing timerinterrupt");
119 #endif
120     //Timer2 Settings: Timer Prescaler /64,
121     TCCR2 |= (1<<CS22); // turn on CS22 bit
122     TCCR2 &= ~((1<<CS21) | (1<<CS20)); // turn off CS21 and CS20 bits
123     // Use normal mode
124     TCCR2 &= ~((1<<WGM21) | (1<<WGM20)); // turn off WGM21 and WGM20 bits
125     // Use internal clock - external clock not used in Arduino
126     ASSR |= (0<<AS2);
127     TIMSK |= (1<<TOIE2) | (0<<OCIE2); //Timer2 Overflow Interrupt Enable
128     RESET_TIMER2;
129 #ifdef DCF_DEBUG
130     Serial.println("Initializing DCF77 signal listener interrupt");
131 #endif
132     attachInterrupt(0, int0handler, CHANGE);

```

```

133 }
134
135 /**
136 * Append a signal to the dcf_rx_buffer. Argument can be 1 or 0. An internal
137 * counter shifts the writing position within the buffer. If position > 59,
138 * a new minute begins -> time to call finalizeBuffer().
139 */
140 void appendSignal(unsigned char signal) {
141 #ifdef DCF_DEBUG
142 // Serial.print(", appending value ");
143 // Serial.print(signal, DEC);
144 // Serial.print(" at position ");
145 // Serial.println(bufferPosition);
146 #endif
147 dcf_rx_buffer = dcf_rx_buffer | ((unsigned long long) signal << bufferPosition);
148 // Update the parity bits. First: Reset when minute, hour or date starts.
149 if (bufferPosition == 21 || bufferPosition == 29 || bufferPosition == 36) {
150     flags.parity_flag = 0;
151 }
152 // save the parity when the corresponding segment ends
153 if (bufferPosition == 28) {flags.parity_min = flags.parity_flag;};
154 if (bufferPosition == 35) {flags.parity_hour = flags.parity_flag;};
155 if (bufferPosition == 58) {flags.parity_date = flags.parity_flag;};
156 // When we received a 1, toggle the parity flag
157 if (signal == 1) {
158     flags.parity_flag = flags.parity_flag ^ 1;
159 }
160 bufferPosition++;
161 if (bufferPosition > 59) {
162     finalizeBuffer();
163 }
164 }
165
166 /**
167 * Evaluates the information stored in the buffer. This is where the DCF77
168 * signal is decoded and the internal clock is updated.
169 */
170 void finalizeBuffer(void) {
171     if (bufferPosition == 59) {
172 #ifdef DCF_DEBUG
173         Serial.println("Finalizing Buffer");
174 #endif
175         struct DCF77Buffer *rx_buffer;
176         rx_buffer = (struct DCF77Buffer *)((unsigned long long)&dcf_rx_buffer);
177         if (flags.parity_min == rx_buffer->P1 &&
178             flags.parity_hour == rx_buffer->P2 &&
179             flags.parity_date == rx_buffer->P3)
180     {
181 #ifdef DCF_DEBUG
182         Serial.println("Parity check OK - updating time.");
183 #endif
184         //convert the received bits from BCD
185         mm = rx_buffer->Min-((rx_buffer->Min/16)*6);
186         hh = rx_buffer->Hour-((rx_buffer->Hour/16)*6);
187         day= rx_buffer->Day-((rx_buffer->Day/16)*6);
188         mon= rx_buffer->Month-((rx_buffer->Month/16)*6);
189         year= 2000 + rx_buffer->Year-((rx_buffer->Year/16)*6);
190     }
191 #ifdef DCF_DEBUG
192     else {
193         Serial.println("Parity check NOK - running on internal clock.");
194     }
195 #endif
196 }
197 // reset stuff
198 ss = 0;

```

```

19   bufferPosition = 0;
20   dcf_rx_buffer=0;
21 }
22
23 /**
24 * Dump the time to the serial line.
25 */
26 void serialDumpTime(void){
27   Serial.print("Time: ");
28   Serial.print(hh, DEC);
29   Serial.print(":");
30   Serial.print(mm, DEC);
31   Serial.print(":");
32   Serial.print(ss, DEC);
33   Serial.print(" Date: ");
34   Serial.print(day, DEC);
35   Serial.print(".");
36   Serial.print(mon, DEC);
37   Serial.print(".");
38   Serial.println(year, DEC);
39 }
40
41
42 /**
43 * Evaluates the signal as it is received. Decides whether we received
44 * a "1" or a "0" based on the
45 */
46 void scanSignal(void){
47   if (DCFSignalState == 1) {
48     int thisFlankTime=millis();
49     if (thisFlankTime - previousFlankTime > DCF_sync_millis) {
50 #ifdef DCF_DEBUG
51       serialDumpTime();
52       Serial.println("####");
53       Serial.println("#### Begin of new Minute!!!");
54       Serial.println("####");
55 #endif
56       finalizeBuffer();
57     }
58     previousFlankTime=thisFlankTime;
59 #ifdef DCF_DEBUG
60     //Serial.print(previousFlankTime);
61     //Serial.print(": DCF77 Signal erkannt, ");
62 #endif
63   }
64   else {
65     /* or a falling flank */
66     int difference=millis() - previousFlankTime;
67 #ifdef DCF_DEBUG
68     // Serial.print("Impulsdauer ms: ");
69     // Serial.print(difference);
70     Serial.print(" ");
71     Serial.print(hh, DEC);
72     Serial.print(":");
73     Serial.print(mm, DEC);
74     Serial.print(":");
75     //Serial.print(ss, DEC);
76     // Serial.print(" Date: ");
77     Serial.print(bufferPosition);
78     Serial.print(" --- ");
79     Serial.print(day, DEC);
80     Serial.print(".");
81     Serial.print(mon, DEC);
82     Serial.print(".");
83     Serial.println(year, DEC);
84 #endif

```

```

265     if (difference < DCF_split_millis) {
266         appendSignal(0);
267     }
268     else {
269         appendSignal(1);
270     }
271 }
272 }
273 /**
274 * The interrupt routine for counting seconds - increment hh:mm:ss.
275 */
276 ISR(TIMER2_OVF_vect) {
277     RESET_TIMER2;
278     tick_counter += 1;
279     if (tick_counter == 1000) {
280         ss++;
281         if (ss==60) {
282             ss=0;
283             mm++;
284             if (mm==60) {
285                 mm=0;
286                 hh++;
287                 if (hh==24)
288                     hh=0;
289             }
290         }
291     }
292     tick_counter = 0;
293 }
294 };
295 /**
296 * Interrupt handler for INT0 - called when the signal on Pin 2 changes.
297 */
298 void int0handler() {
299     // check the value again - since it takes some time to
300     // activate the interrupt routine, we get a clear signal.
301     DCFSignalState = digitalRead(DCF77PIN);
302 }
303 }
304
305
306 /**
307 * Standard Arduino methods below.
308 */
309
310 void setup(void) {
311     // We need to start serial here again,
312     // for Arduino 007 (new serial code)
313     Serial.begin(9600);
314     DCF77Init();
315 }
316
317 void loop(void) {
318     if (ss != previousSecond) {
319         serialDumpTime();
320         previousSecond = ss;
321     }
322     if (DCFSignalState != previousSignalState) {
323         scanSignal();
324         if (DCFSignalState) {
325             digitalWrite(BLINKPIN, HIGH);
326         } else {
327             digitalWrite(BLINKPIN, LOW);
328         }
329         previousSignalState = DCFSignalState;
330     }

```

```
331     //delay(20);  
332 }
```