

branch: master Arduino / hardware / arduino / cores / arduino / WInterrupts.c

Fede85 5 months ago Added support to INT6 on Leonardo.

5 contributors

file 335 lines (294 sloc) 8.97 kb

Open Edit Raw Blame History Delete

```
1  /* -*- mode: jde; c-basic-offset: 2; indent-tabs-mode: nil -*- */
2
3  /*
4   Part of the Wiring project - http://wiring.uniandes.edu.co
5
6   Copyright (c) 2004-05 Hernando Barragan
7
8   This library is free software; you can redistribute it and/or
9   modify it under the terms of the GNU Lesser General Public
10  License as published by the Free Software Foundation; either
11  version 2.1 of the License, or (at your option) any later version.
12
13  This library is distributed in the hope that it will be useful,
14  but WITHOUT ANY WARRANTY; without even the implied warranty of
15  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
16  Lesser General Public License for more details.
17
18  You should have received a copy of the GNU Lesser General
19  Public License along with this library; if not, write to the
20  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
21  Boston, MA 02111-1307 USA
22
23  Modified 24 November 2006 by David A. Mellis
24  Modified 1 August 2010 by Mark Sproul
25  */
26
27  #include <inttypes.h>
28  #include <avr/io.h>
29  #include <avr/interrupt.h>
30  #include <avr/pgmspace.h>
31  #include <stdio.h>
32
33  #include "wiring_private.h"
34
35  static volatile voidFuncPtr intFunc[EXTERNAL_NUM_INTERRUPTS];
36  // volatile static voidFuncPtr twiIntFunc;
37
38  void attachInterrupt(uint8_t interruptNum, void (*userFunc)(void), int mode) {
39    if(interruptNum < EXTERNAL_NUM_INTERRUPTS) {
40      intFunc[interruptNum] = userFunc;
41
42      // Configure the interrupt mode (trigger on Low input, any change, rising
43      // edge, or falling edge). The mode constants were chosen to correspond
44      // to the configuration bits in the hardware register, so we simply shift
45      // the mode into place.
46
47      // Enable the interrupt.
48
49      switch (interruptNum) {
50      #if defined(__AVR_ATmega32U4__)
51        // I hate doing this, but the register assignment differs between the 1280/2560
52        // and the 32U4. Since avrlib defines registers PCMSK1 and PCMSK2 that aren't
53        // even present on the 32U4 this is the only way to distinguish between them.
54      case 0:
55        EICRA = (EICRA & ~(1<<ISC00 | 1<<ISC01)) | (mode << ISC00);
56        EIMSK |= (1<<INT0);
```

```

57     break;
58 case 1:
59     EICRA = (EICRA & ~((1<<ISC10) | (1<<ISC11))) | (mode << ISC10);
60     EIMSK |= (1<<INT1);
61     break;
62 case 2:
63     EICRA = (EICRA & ~((1<<ISC20) | (1<<ISC21))) | (mode << ISC20);
64     EIMSK |= (1<<INT2);
65     break;
66 case 3:
67     EICRA = (EICRA & ~((1<<ISC30) | (1<<ISC31))) | (mode << ISC30);
68     EIMSK |= (1<<INT3);
69     break;
70 case 4:
71     EICRB = (EICRB & ~((1<<ISC60) | (1<<ISC61))) | (mode << ISC60);
72     EIMSK |= (1<<INT6);
73     break;
74 #elif defined(EICRA) && defined(EICRB) && defined(EIMSK)
75 case 2:
76     EICRA = (EICRA & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
77     EIMSK |= (1 << INT0);
78     break;
79 case 3:
80     EICRA = (EICRA & ~((1 << ISC10) | (1 << ISC11))) | (mode << ISC10);
81     EIMSK |= (1 << INT1);
82     break;
83
84 case 4:
85     EICRA = (EICRA & ~((1 << ISC20) | (1 << ISC21))) | (mode << ISC20);
86     EIMSK |= (1 << INT2);
87     break;
88 case 5:
89     EICRA = (EICRA & ~((1 << ISC30) | (1 << ISC31))) | (mode << ISC30);
90     EIMSK |= (1 << INT3);
91     break;
92 case 0:
93     EICRB = (EICRB & ~((1 << ISC40) | (1 << ISC41))) | (mode << ISC40);
94     EIMSK |= (1 << INT4);
95     break;
96 case 1:
97     EICRB = (EICRB & ~((1 << ISC50) | (1 << ISC51))) | (mode << ISC50);
98     EIMSK |= (1 << INT5);
99     break;
100 case 6:
101     EICRB = (EICRB & ~((1 << ISC60) | (1 << ISC61))) | (mode << ISC60);
102     EIMSK |= (1 << INT6);
103     break;
104 case 7:
105     EICRB = (EICRB & ~((1 << ISC70) | (1 << ISC71))) | (mode << ISC70);
106     EIMSK |= (1 << INT7);
107     break;
108 #else
109 case 0:
110     #if defined(EICRA) && defined(ISC00) && defined(EIMSK)
111     EICRA = (EICRA & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
112     EIMSK |= (1 << INT0);
113     #elif defined(MCUCR) && defined(ISC00) && defined(GICR)
114     MCUCR = (MCUCR & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
115     GICR |= (1 << INT0);
116     #elif defined(MCUCR) && defined(ISC00) && defined(GIMSK)
117     MCUCR = (MCUCR & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
118     GIMSK |= (1 << INT0);
119     #else
120     #error attachInterrupt not finished for this CPU (case 0)
121     #endif
122     break;
123
124 case 1:
125     #if defined(EICRA) && defined(ISC10) && defined(ISC11) && defined(EIMSK)
126     EICRA = (EICRA & ~((1 << ISC10) | (1 << ISC11))) | (mode << ISC10);
127     EIMSK |= (1 << INT1);
128     #elif defined(MCUCR) && defined(ISC10) && defined(ISC11) && defined(GICR)
129     MCUCR = (MCUCR & ~((1 << ISC10) | (1 << ISC11))) | (mode << ISC10);
130     GICR |= (1 << INT1);
131     #elif defined(MCUCR) && defined(ISC10) && defined(GIMSK) && defined(GIMSK)

```

```

131     MCUCR = (MCUCR & ~((1 << ISC10) | (1 << ISC11))) | (mode << ISC10);
132     GIMSK |= (1 << INT1);

133 #else
134 #warning attachInterrupt may need some more work for this cpu (case 1)
135 #endif
136     break;
137
138 case 2:
139 #if defined(EICRA) && defined(ISC20) && defined(ISC21) && defined(EIMSK)
140     EICRA = (EICRA & ~((1 << ISC20) | (1 << ISC21))) | (mode << ISC20);
141     EIMSK |= (1 << INT2);
142 #elif defined(MCUCR) && defined(ISC20) && defined(ISC21) && defined(GICR)
143     MCUCR = (MCUCR & ~((1 << ISC20) | (1 << ISC21))) | (mode << ISC20);
144     GICR |= (1 << INT2);
145 #elif defined(MCUCR) && defined(ISC20) && defined(GIMSK) && defined(GIMSK)
146     MCUCR = (MCUCR & ~((1 << ISC20) | (1 << ISC21))) | (mode << ISC20);
147     GIMSK |= (1 << INT2);
148 #endif
149     break;
150 #endif
151 }
152 }
153 }
154
155 void detachInterrupt(uint8_t interruptNum) {
156     if(interruptNum < EXTERNAL_NUM_INTERRUPTS) {
157         // Disable the interrupt. (We can't assume that interruptNum is equal
158         // to the number of the EIMSK bit to clear, as this isn't true on the
159         // ATmega8. There, INT0 is 6 and INT1 is 7.)
160         switch (interruptNum) {
161 #if defined(__AVR_ATmega32U4__)
162             case 0:
163                 EIMSK &= ~(1<<INT0);
164                 break;
165             case 1:
166                 EIMSK &= ~(1<<INT1);
167                 break;
168             case 2:
169                 EIMSK &= ~(1<<INT2);
170                 break;
171             case 3:
172                 EIMSK &= ~(1<<INT3);
173                 break;
174             case 4:
175                 EIMSK &= ~(1<<INT6);
176                 break;
177 #elif defined(EICRA) && defined(EICRB) && defined(EIMSK)
178             case 2:
179                 EIMSK &= ~(1 << INT0);
180                 break;
181             case 3:
182                 EIMSK &= ~(1 << INT1);
183
184                 break;
185             case 4:
186                 EIMSK &= ~(1 << INT2);
187                 break;
188             case 5:
189                 EIMSK &= ~(1 << INT3);
190                 break;
191             case 0:
192                 EIMSK &= ~(1 << INT4);
193                 break;
194             case 1:
195                 EIMSK &= ~(1 << INT5);
196                 break;
197             case 6:
198                 EIMSK &= ~(1 << INT6);
199                 break;
200             case 7:
201                 EIMSK &= ~(1 << INT7);
202                 break;
203 #else
204             case 0:

```

```

204     #if defined(EIMSK) && defined(INT0)
205         EIMSK &= ~(1 << INT0);
206     #elif defined(GICR) && defined(ISC00)
207         GICR &= ~(1 << INT0); // atmega32
208     #elif defined(GIMSK) && defined(INT0)
209         GIMSK &= ~(1 << INT0);
210     #else
211         #error detachInterrupt not finished for this cpu
212     #endif
213     break;
214
215     case 1:
216     #if defined(EIMSK) && defined(INT1)
217         EIMSK &= ~(1 << INT1);
218     #elif defined(GICR) && defined(INT1)
219         GICR &= ~(1 << INT1); // atmega32
220     #elif defined(GIMSK) && defined(INT1)
221         GIMSK &= ~(1 << INT1);
222     #else
223         #warning detachInterrupt may need some more work for this cpu (case 1)
224     #endif
225     break;
226 #endif
227 }
228
229 intFunc[interruptNum] = 0;
230 }
231 }
232
233 /*
234 void attachInterruptTwice(void (*userFunc)(void) ) {
235     twiIntFunc = userFunc;
236 }
237 */
238
239 #if defined(__AVR_ATmega32U4__)
240 ISR(INT0_vect) {
241     if(intFunc[EXTERNAL_INT_0])
242         intFunc[EXTERNAL_INT_0]();
243 }
244
245 ISR(INT1_vect) {
246     if(intFunc[EXTERNAL_INT_1])
247         intFunc[EXTERNAL_INT_1]();
248 }
249
250 ISR(INT2_vect) {
251     if(intFunc[EXTERNAL_INT_2])
252         intFunc[EXTERNAL_INT_2]();
253 }
254
255 ISR(INT3_vect) {
256     if(intFunc[EXTERNAL_INT_3])
257         intFunc[EXTERNAL_INT_3]();
258 }
259
260 ISR(INT6_vect) {
261     if(intFunc[EXTERNAL_INT_4])
262         intFunc[EXTERNAL_INT_4]();
263 }
264
265 #elif defined(EICRA) && defined(EICRB)
266
267 ISR(INT0_vect) {
268     if(intFunc[EXTERNAL_INT_2])
269         intFunc[EXTERNAL_INT_2]();
270 }
271
272 ISR(INT1_vect) {
273     if(intFunc[EXTERNAL_INT_3])
274         intFunc[EXTERNAL_INT_3]();
275 }
276
277 ISR(INT2_vect) {

```

```
278     if(intFunc[EXTERNAL_INT_4])
279         intFunc[EXTERNAL_INT_4]();
280 }
281
282 ISR(INT3_vect) {
283
284     if(intFunc[EXTERNAL_INT_5])
285         intFunc[EXTERNAL_INT_5]();
286 }
287
288 ISR(INT4_vect) {
289     if(intFunc[EXTERNAL_INT_0])
290         intFunc[EXTERNAL_INT_0]();
291 }
292
293 ISR(INT5_vect) {
294     if(intFunc[EXTERNAL_INT_1])
295         intFunc[EXTERNAL_INT_1]();
296 }
297
298 ISR(INT6_vect) {
299     if(intFunc[EXTERNAL_INT_6])
300         intFunc[EXTERNAL_INT_6]();
301 }
302
303 ISR(INT7_vect) {
304     if(intFunc[EXTERNAL_INT_7])
305         intFunc[EXTERNAL_INT_7]();
306 }
307
308 #else
309
310 ISR(INT0_vect) {
311     if(intFunc[EXTERNAL_INT_0])
312         intFunc[EXTERNAL_INT_0]();
313 }
314
315 ISR(INT1_vect) {
316     if(intFunc[EXTERNAL_INT_1])
317         intFunc[EXTERNAL_INT_1]();
318 }
319
320 #if defined(EICRA) && defined(ISC20)
321
322 ISR(INT2_vect) {
323     if(intFunc[EXTERNAL_INT_2])
324         intFunc[EXTERNAL_INT_2]();
325 }
326 #endif
327
328 #endif
329
330 /*
331 ISR(TWI_vect) {
332     if(twiIntFunc)
333         twiIntFunc();
334 }
335
336 */
```

