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

```
/* -*- mode: jde; c-basic-offset: 2; indent-tabs-mode: nil -*- */

/*
  Part of the Wiring project - http://wiring.uniandes.edu.co

  Copyright (c) 2004-05 Hernando Barragan

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General
  Public License along with this library; if not, write to the
  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
  Boston, MA  02111-1307  USA

  Modified 24 November 2006 by David A. Mellis
  Modified 1 August 2010 by Mark Sproul
*/

#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdio.h>


#include "wiring_private.h"

static volatile voidFuncPtr intFunc[EXTERNAL_NUM_INTERRUPTS];
// volatile static voidFuncPtr twiIntFunc;

void attachInterrupt(uint8_t interruptNum, void (*userFunc)(void), int mode) {
  if(interruptNum < EXTERNAL_NUM_INTERRUPTS) {
    intFunc[interruptNum] = userFunc;

    // Configure the interrupt mode (trigger on low input, any change, rising
    // edge, or falling edge).  The mode constants were chosen to correspond
    // to the configuration bits in the hardware register, so we simply shift
    // the mode into place.

    // Enable the interrupt.

    switch (interruptNum) {
#if defined(__AVR_ATmega32U4__)
        // I hate doing this, but the register assignment differs between the 1280/2560
        // and the 32U4.  Since avrlib defines registers PCMSK1 and PCMSK2 that aren't
        // even present on the 32U4 this is the only way to distinguish between them.
    case 0:
        EICRA = (EICRA & ~((1<<ISC00) | (1<<ISC01))) | (mode << ISC00);
        EIMSK |= (1<<INT0);
```

```
        break;
    case 1:
        EICRA = (EICRA & ~((1<<ISC10) | (1<<ISC11))) | (mode << ISC10);
        EIMSK |= (1<<INT1);
        break;
    case 2:
        EICRA = (EICRA & ~((1<<ISC20) | (1<<ISC21))) | (mode << ISC20);
        EIMSK |= (1<<INT2);
        break;
    case 3:
        EICRA = (EICRA & ~((1<<ISC30) | (1<<ISC31))) | (mode << ISC30);
        EIMSK |= (1<<INT3);
        break;
    case 4:
        EICRB = (EICRB & ~((1<<ISC60) | (1<<ISC61))) | (mode << ISC60);
        EIMSK |= (1<<INT6);
        break;
#elif defined(EICRA) && defined(EICRB) && defined(EIMSK)
    case 2:
      EICRA = (EICRA & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
      EIMSK |= (1 << INT0);
      break;
    case 3:
      EICRA = (EICRA & ~((1 << ISC10) | (1 << ISC11))) | (mode << ISC10);
      EIMSK |= (1 << INT1);
      break;

    case 4:
      EICRA = (EICRA & ~((1 << ISC20) | (1 << ISC21))) | (mode << ISC20);
      EIMSK |= (1 << INT2);
      break;
    case 5:
      EICRA = (EICRA & ~((1 << ISC30) | (1 << ISC31))) | (mode << ISC30);
      EIMSK |= (1 << INT3);
      break;
    case 0:
      EICRB = (EICRB & ~((1 << ISC40) | (1 << ISC41))) | (mode << ISC40);
      EIMSK |= (1 << INT4);
      break;
    case 1:
      EICRB = (EICRB & ~((1 << ISC50) | (1 << ISC51))) | (mode << ISC50);
      EIMSK |= (1 << INT5);
      break;
    case 6:
      EICRB = (EICRB & ~((1 << ISC60) | (1 << ISC61))) | (mode << ISC60);
      EIMSK |= (1 << INT6);
      break;
    case 7:
      EICRB = (EICRB & ~((1 << ISC70) | (1 << ISC71))) | (mode << ISC70);
      EIMSK |= (1 << INT7);
      break;
#else
    case 0:
    #if defined(EICRA) && defined(ISC00) && defined(EIMSK)
      EICRA = (EICRA & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
      EIMSK |= (1 << INT0);
    #elif defined(MCUCR) && defined(ISC00) && defined(GICR)
      MCUCR = (MCUCR & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
      GICR |= (1 << INT0);
    #elif defined(MCUCR) && defined(ISC00) && defined(GIMSK)
      MCUCR = (MCUCR & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
      GIMSK |= (1 << INT0);
    #else
      #error attachInterrupt not finished for this CPU (case 0)
    #endif
      break;

    case 1:
    #if defined(EICRA) && defined(ISC10) && defined(ISC11) && defined(EIMSK)
      EICRA = (EICRA & ~((1 << ISC10) | (1 << ISC11))) | (mode << ISC10);
      EIMSK |= (1 << INT1);
    #elif defined(MCUCR) && defined(ISC10) && defined(ISC11) && defined(GICR)
      MCUCR = (MCUCR & ~((1 << ISC10) | (1 << ISC11))) | (mode << ISC10);
      GICR |= (1 << INT1);
    #elif defined(MCUCR) && defined(ISC10) && defined(GIMSK) && defined(GIMSK)
```

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

    #else
      #warning attachInterrupt may need some more work for this cpu (case 1)
    #endif
      break;

    case 2:
    #if defined(EICRA) && defined(ISC20) && defined(ISC21) && defined(EIMSK)
      EICRA = (EICRA & ~((1 << ISC20) | (1 << ISC21))) | (mode << ISC20);
      EIMSK |= (1 << INT2);
    #elif defined(MCUCR) && defined(ISC20) && defined(ISC21) && defined(GICR)
      MCUCR = (MCUCR & ~((1 << ISC20) | (1 << ISC21))) | (mode << ISC20);
      GICR |= (1 << INT2);
    #elif defined(MCUCR) && defined(ISC20) && defined(GIMSK) && defined(GIMSK)
      MCUCR = (MCUCR & ~((1 << ISC20) | (1 << ISC21))) | (mode << ISC20);
      GIMSK |= (1 << INT2);
    #endif
      break;
#endif
    }
  }
}

void detachInterrupt(uint8_t interruptNum) {
  if(interruptNum < EXTERNAL_NUM_INTERRUPTS) {
    // Disable the interrupt.  (We can't assume that interruptNum is equal
    // to the number of the EIMSK bit to clear, as this isn't true on the
    // ATmega8.  There, INT0 is 6 and INT1 is 7.)
    switch (interruptNum) {
#if defined(__AVR_ATmega32U4__)
    case 0:
        EIMSK &= ~(1<<INT0);
        break;
    case 1:
        EIMSK &= ~(1<<INT1);
        break;
    case 2:
        EIMSK &= ~(1<<INT2);
        break;
    case 3:
        EIMSK &= ~(1<<INT3);
        break;
    case 4:
        EIMSK &= ~(1<<INT6);
        break;
#elif defined(EICRA) && defined(EICRB) && defined(EIMSK)
    case 2:
      EIMSK &= ~(1 << INT0);
      break;
    case 3:
      EIMSK &= ~(1 << INT1);

      break;
    case 4:
      EIMSK &= ~(1 << INT2);
      break;
    case 5:
      EIMSK &= ~(1 << INT3);
      break;
    case 0:
      EIMSK &= ~(1 << INT4);
      break;
    case 1:
      EIMSK &= ~(1 << INT5);
      break;
    case 6:
      EIMSK &= ~(1 << INT6);
      break;
    case 7:
      EIMSK &= ~(1 << INT7);
      break;
#else
    case 0:
```

```
    #if defined(EIMSK) && defined(INT0)
      EIMSK &= ~(1 << INT0);
    #elif defined(GICR) && defined(ISC00)
      GICR &= ~(1 << INT0); // atmega32
    #elif defined(GIMSK) && defined(INT0)
      GIMSK &= ~(1 << INT0);
    #else
      #error detachInterrupt not finished for this cpu
    #endif
      break;

    case 1:
    #if defined(EIMSK) && defined(INT1)
      EIMSK &= ~(1 << INT1);
    #elif defined(GICR) && defined(INT1)
      GICR &= ~(1 << INT1); // atmega32
    #elif defined(GIMSK) && defined(INT1)
      GIMSK &= ~(1 << INT1);
    #else
      #warning detachInterrupt may need some more work for this cpu (case 1)
    #endif
      break;
#endif
    }

    intFunc[interruptNum] = 0;
  }
}


/*
void attachInterruptTwi(void (*userFunc)(void) ) {
  twiIntFunc = userFunc;
}
*/

#if defined(__AVR_ATmega32U4__)
ISR(INT0_vect) {
        if(intFunc[EXTERNAL_INT_0])
                intFunc[EXTERNAL_INT_0]();
}

ISR(INT1_vect) {
        if(intFunc[EXTERNAL_INT_1])
                intFunc[EXTERNAL_INT_1]();
}

ISR(INT2_vect) {
    if(intFunc[EXTERNAL_INT_2])
                intFunc[EXTERNAL_INT_2]();
}

ISR(INT3_vect) {
    if(intFunc[EXTERNAL_INT_3])
                intFunc[EXTERNAL_INT_3]();
}

ISR(INT6_vect) {
    if(intFunc[EXTERNAL_INT_4])
                intFunc[EXTERNAL_INT_4]();
}

#elif defined(EICRA) && defined(EICRB)

ISR(INT0_vect) {
  if(intFunc[EXTERNAL_INT_2])
    intFunc[EXTERNAL_INT_2]();
}

ISR(INT1_vect) {
  if(intFunc[EXTERNAL_INT_3])
    intFunc[EXTERNAL_INT_3]();
}

ISR(INT2_vect) {
```

```
278   if(intFunc[EXTERNAL_INT_4])
279     intFunc[EXTERNAL_INT_4]();
280 }
281
282 ISR(INT3_vect) {

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

333 */
334
```