ADC_Interrupt_Nano

1.0

Generated by Doxygen 1.8.11

Contents

1 Overview

Reading AD-Valures, which are generated by interrupts
ATMega328p: UNO, NANO, ...

Sketch uses 3872 bytes (12%) of program storage space. Maximum is 30720 bytes. Global variables use 388 bytes (18%) of dynamic memory, leaving 1660 bytes for local variables. Maximum is 2048 bytes.

Introduction

Realtime Operating systems are the prefered tool for most measurements.
ChibiOS and derivatives are running on the Arduino UNO.
But sometimes a TickTime of 1 ms is too long.

Interrupts

Here I show another approach using Interrupts. Three curves are red from the AD-converter using AD-Interrupt-↩ Conversion-Ready
About 6000 conversions per second are done.
The red values are filtered an may be displayed using "Serial Plotter"

Multitasking

Besides "loop" 2 tasks are running in the backgrund"
One of them precisly ever 1 ms !

Filter

AD-Values for CHNUM channel are read using AD-Ready-Interrupts. One Channel after the other is read starting with channel 0 again.

This is done in the background without any intervention of the LOOP

The values are averaged for IRQ_SAMPLES samples

This is the code for averaging:

```
avv=(float)analogVal;
av[Channel] = (Alpha[Channel]*oldVal[Channel]) + ((1-Alpha[Channel])*avv);
oldVal[Channel]=av[Channel];
```

The sense is to get most samples possible, build an average over IRQ_SAMPLES value and set the IrqReadyFlag. Then the Measurement is stopped and the "Busy" Flag is set.

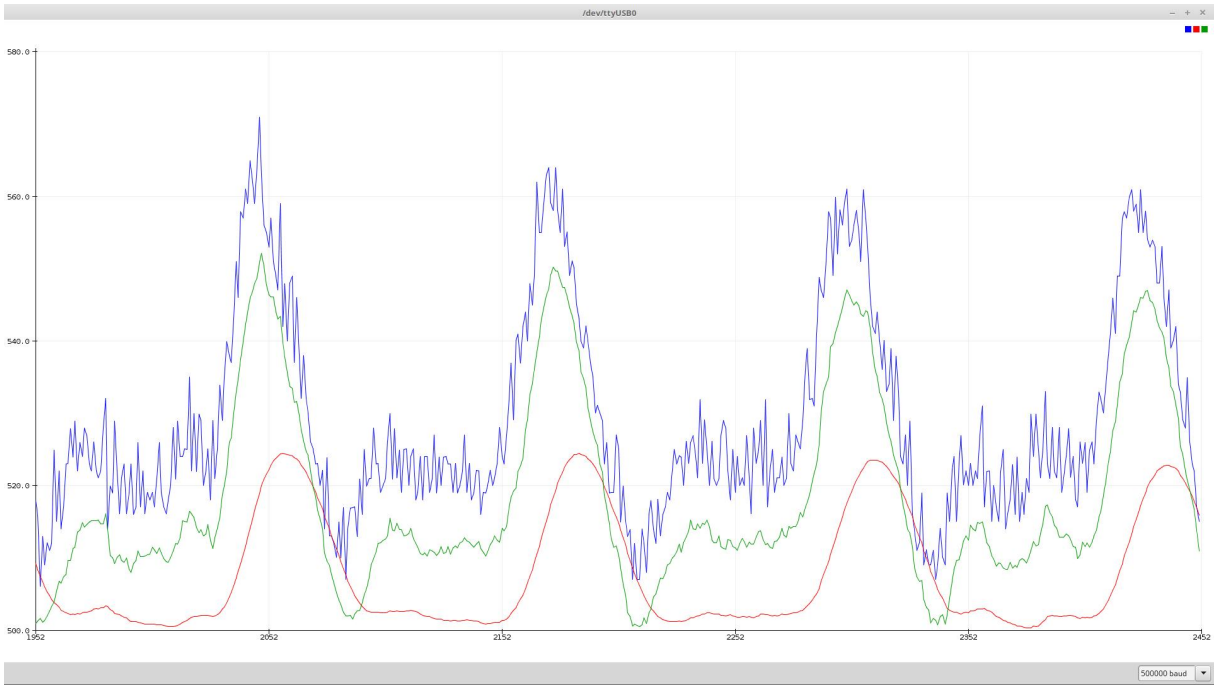
LOOP or any other function has to read (and print/ plot) the values.

After that the next points will be generated in the background again with:

```
IrqReadyFlag = 0;
Busy = false;
ADCSRA |= B01000000; // Start next conversion
```

Example-HEART-BEAT

Here is an example of a HEART-BEAT-Sensor.



Blue - original

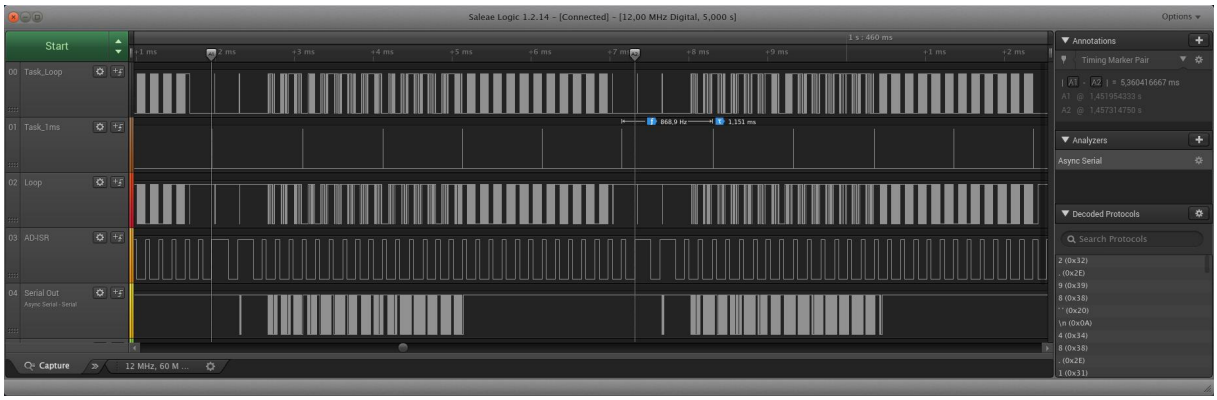
Green - soft filtered

Red - hard filtered<br

The channels got an offset for better display.

Filtereing implies a phase shift !

Timing



We get about 6000 (filtered) samples per second - 2000 per channel

- with 10 IRQ_SAMPLES there are 200 Time Points per second (180 with Serial.print)

Enough to show the details of HEART-BEAT

The output in LOOP needs 1,6 ms every 5,6 ms for a Time Point.

There is plenty of room to do other things in LOOP !

- Build pdf: in latex: pdflatex refman

Author

Helmut Weber

2 AD-Interrupts

The AD-Converter is initialized in "setup" and the first conversion is started
When AD is ready an interrupt to "ISR(ADC_vect)" is executed.
The Interrupt

- Reads the value from the actual channel (starting with channel 0)
- Does Filtering:
- > avv=(float)analogVal;
- > av[Channel] = (Alpha[Channel]∗oldVal[Channel]) + ((1-Alpha[Channel])∗avv);
- > oldVal[Channel]=av[Channel];

Alpha[] for each channel must be set in "setup" befor

Then the AD is prepared for the next channel and another conversion is started.
If all channels got "IRQ_SAMPLES" values ReadyFlag is set to last channels+1
This is done to test "ReadyFlag>=" in "loop"
The "Busy" flag is set, which disallow further sampling.

3 Loop

"loop" test if there is a "ReadyFlag", which is the signal that a channel got
"IRQ_SAMPLES" (filtered) values. Because all channels get the same number of values the
After reading the filtered value of the channel the "Busy" Flag is set to 0 to enable
further AD-conversions and the ADC ist started again.

"ReadyFlag" 's for all channels should send the "ReadyFlag" one after the other -
starting with channel 0.
The results are converted to an Ascii-String and a flag for output is set.

if NO "ReadyFlag" is set "loop" will:

- Test, if there is an outputstring to send
- Calls "Task_Loop" "Task_Loop" is called very frequently (up to 50 kHz), but there are gaps of up to 1.6 ms.

4 Task_Loop

This task is called very often from "loop" but with great jitter.
The execution time must be (in this example) less than 100 µs

5 Task_1ms

This task is called precisely ever 1ms with very low jitter.
It is called from the Interrupt-Routine. The execution time must be (in this example) less than 100 µs

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

7 File Documentation

7.1 ADC_Interrupt_Nano.ino File Reference

Macros

- #define CHNUM 3

  Number of AD-channels (1..5) to use.
- #define IRQ_SAMPLES 10

  Number of IRQ-Samples to average befor setting "IrqReadyFlag".

Functions

- char * ftoa (double f, int digits)

  "ftoa"
  This is the selfmade conversion from float to ascii-string
  digits are the number of digits behind the"."
- void setup ()

  "Setup" sets the Alpha's for the channels (filtering, s.o) and
  initialises the ADC to run with 125 kHz - no autorun, no freerun

- void Task_Loop ()
- void Task_1ms ()

  "Task_1ms"
  is called every ms with VERY LOW jitter!!!
  It is called from ISR of the ADC - so it should be very short.

- void loop ()

  "loop"
  checks:
  if "ReadyFlag" (a channel got SAMPLE_NUM values)

- ISR (ADC_vect)

  "ISR)ADC_Vector)"

Variables

- volatile int IrqReadyFlag

  "IrqReadyFlag" is set after IRQ_SAMPLES for each channel
- volatile int analogVal
- int Channel =0
- float Alpha [CHNUM]

  Alpha[channel] is the Filter-Parameter for each channel.
  Is set in "setup". "Should be greater than 0.75 to see an effect.
  MUST be less than 1.0 !!!
- float av [CHNUM]

  av[channel] contains the last filtered value of a channel
  It is build in ISR and used by loop to generate ouput-value for the plotter
  "oldVal[ ]" is the last filtered value of a channel Read Only
- float oldVal [CHNUM]
- volatile bool Busy =false

  "Busy" is set together with ReadyFlag[channel] and disable reading new AD values in ISR
  until it is reset in "loop"
- char out [100]

  "out[ ]" is the buffer of the line to send to the plotter. Only internal usage !
- char * pt =out

  internal usage
- char * strpt

  internal usage
- float val

  internal usage
- volatile unsigned long lastTaskTime =micros()

  "lastTaskTime" is used by ISR to call Task_1ms br>
- bool output = false

  "Task_Loop"
  is called from "loop"r
  It should be very short because loop should be able to react fast Often called, but with jitter !

7.1.1 Macro Definition Documentation

7.1.1.1 #define CHNUM 3

Number of AD-channels (1..5) to use.

Definition at line 182 of file ADC_Interrupt_Nano.ino.

7.1.1.2 #define IRQ_SAMPLES 10

Number of IRQ-Samples to average befor setting "IrqReadyFlag".

Definition at line 188 of file ADC_Interrupt_Nano.ino.

7.1.2 Function Documentation

7.1.2.1 char* ftoa (double *f,* int *digits*)

"ftoa"
This is the selfmade conversion from float to ascii-string
digits are the number of digits behind the"."

Definition at line 271 of file ADC_Interrupt_Nano.ino.

```
271                                 {
272 static char b[31];
273 static char const digit[] = "0123456789";
274 char* p = b;
275 uint32_t i;
276
277 int d,j;
278   d=digits;
279   while (d) {
280     f*=10.0;
281     d--;
282   }
283
284   i=(uint32_t)f;
285
286   p=b+28;
287   j=0;
288   *p = 0;
289   *(p+1)=0;
290
291   do { //Move back, inserting digits as u go
292       if (j == digits) { p--; *p='.'; }
293       p--;
294       *p = digit[i % 10ll];
295       i = i/10ll;
296       j++;
297   } while(i);
298
299   return p; // return result as a pointer to string
300 }
```

7.1.2.2 ISR (ADC_vect)

"ISR)ADC_Vector)"

get fired (once), when an AD conversion is ready.
It

- calls Timer_1ms
- reads and filters actual channel
- adds channel∗10 to read value for better display
- if SAMPLE_NUM samples for each channel are reached:
- ∗ sets Busy and IrqReday for loop
- increments channel
- starts next conversion

Definition at line 540 of file ADC_Interrupt_Nano.ino.

```
540             {
541 static unsigned int Counter[CHNUM];
542 float avv;
543
544     //digitalWrite(6, HIGH);
545     PORTD |= (1<<6);
546
547     // we use the ADC-Interrupt to do another task every 1ms
548     // with low jitter:
549     if ((micros() - lastTaskTime ) >=1000) {
550       lastTaskTime=micros();
551       Task_1ms();
552     }
553
554   if (Busy) return;                         // Block measurement if BUSY: Loop is working
555
556    // Output is running:
557
558
559   // Must read low first
560   analogVal = ADCL | (ADCH << 8);
561
562   analogVal+=Channel * 10;                  // Channels are better seen with offset
563
564   avv=(float)analogVal;
565   av[Channel] = (Alpha[Channel]*oldVal[Channel]) + ((1-
      Alpha[Channel])*avv);
566   oldVal[Channel]=av[Channel];
567
568
569   Counter[Channel]++;
570   if (Counter[Channel]==IRQ_SAMPLES) {      // one channel has enough samples
571     // Done reading
572     Busy=true;                              // stop measurement
573     IrqReadyFlag = Channel + 1;             // mark this channrl READY
574     Counter[Channel]=0;
575   }
576
577   Channel++;                                // next channel
578   if (Channel==CHNUM) Channel=0;
579
580   ADMUX &= B11111000; // Channel=0;
581   ADMUX |= Channel;   // Set Channel
582
```

```
583
584   // Needed because free-running is disabled
585   // Set ADSC in ADCSRA (0x7A) to start another ADC conversion
586   // if Free Running is disabled:
587    ADCSRA |= B01000000; // Start next conversion
588    //digitalWrite(6, LOW);
589    PORTD &= ~(1<<6);
590 }
```

7.1.2.3 void loop ( )

"loop"
checks:
if "ReadyFlag" (a channel got SAMPLE_NUM values)

- prepare string "out" for output
- reset "Busy" and "IrqReady"
- starts next conversion

else

- Send "out" string
- call "Task_Loop"

Definition at line 455 of file ADC_Interrupt_Nano.ino.

```
455           {
456 static int cnt;
457    cnt++;
458
459    //digitalWrite(5, HIGH);
460    PORTD |= (1<<5);
461
462   // Check to see if the value has been updated
463   if (IrqReadyFlag){  // == Channel+1
464     // Busy is TRUE, we have to start conversion again
465
466     // Enable Interrupts as fast as possible:
467     val=av[IrqReadyFlag-1];
468     ADCSRA |= B01000000; // Start next conversion
469     Busy = false;
470     IrqReadyFlag = 0;
471
472     // Interrupts are ready again, DAC is started again
473
474     // Output:
475     // -- the simple way
476 //    Serial.print(ftoa(val,2)); // Channel 0....
477 //    //Serial.print(val); // Channel 0....
478 //    Serial.print(", ");
479 //    if (IrqReadyFlag==CHNUM) {
480 //      Serial.println();
481 //    }
482
483     // -- the fast way
484     // we use our own FTOA conversion and build a string from all channel values
485     strpt=ftoa(val,2);                  // calling fast own ftoa()
486     memcpy(pt,strpt,strlen(strpt));     // add string vslue from this channel
487     pt+=strlen(strpt);                  // store (add) string to out[]
488     *pt++=' ';                          // Separator for next value
489     if (IrqReadyFlag==CHNUM) {          // this is the last value of all channels
490       *pt++='\n';                       // End Of Line
```

```
491       *pt=0;                            // End of String
492       pt=out;                           // Reset out-ptr
493     }
494
495
496
497   }
498   else {  // Print the line of values as text and do other tasks
499     cnt++;
500     if (cnt >10) {
501       cnt=11;
502       if (*out)  {
503         Serial.print(out);                // Interrrupts are running
504         out [0]=0;
505       }
506     }
507
508
509     // other Tasks
510     //Task_Loop();
511   } // else
512
513   Task_Loop();
514
515   //digitalWrite(5, LOW);
516   PORTD &= ~(1<<5);
517 }
```

7.1.2.4 void setup ( )

"Setup" sets the Alpha's for the channels (filtering, s.o) and
initialises the ADC to run with 125 kHz - no autorun, no freerun

pinModes are for output to oscilloscope only

Definition at line 314 of file ADC_Interrupt_Nano.ino.

```
314             {
315   pinMode(3,OUTPUT);
316   pinMode(4,OUTPUT);
317   pinMode(5,OUTPUT);
318   pinMode(6,OUTPUT);
319
320   Alpha[0]=0.995;
321   Alpha[1]=0.98;
322   Alpha[2]=0.01;
323
324   Serial.begin(500000);
325
326   // Disable global interrupts
327   cli();
328
329   // clear ADLAR in ADMUX to right-adjust the result
330   // ADCL will contain lower 8 bits, ADCH upper 2 (in last two bits)
331   ADMUX &= B11011111;
332
333   // Set REFS1..0 in ADMUX (0x7C) to change reference voltage to the
334   // proper source (01)
335   ADMUX |= B01000000;
336
337   // Clear MUX3..0 in ADMUX (0x7C) in preparation for setting the analog
338   // input
339   ADMUX &= B11110000;
340
341   // Set MUX3 = channel=0 in ADMUX
342   Channel=0;
343   ADMUX |= Channel;
344   // ADMUX |= B00001000; // Binary equivalent
345
346   // Set ADEN in ADCSRA to enable the ADC.
347   ADCSRA |= B10000000;
348
349   // Set ADATE in ADCSRA (0x7A) to enable auto-triggering.
352
353
354   // free running.
355   // This means that as soon as an ADC has finished, the next will be
```

```
356   // immediately started.
359
360   // Set the Prescaler to 128 (16000KHz/128 = 125KHz)
361   ADCSRA |= B00000111;
362
363   // Set ADIE in ADCSRA to enable the ADC interrupt.
364   // Without this, the internal interrupt will not trigger.
365   ADCSRA |= B00001000;
366
367   // Enable global interrupts
368   sei();
369
370   // Kick off the first ADC
371   IrqReadyFlag = 0;
372   // Set ADSC in ADCSRA (0x7A) to start the ADC conversion
373   ADCSRA |=B01000000;
374 }
```

7.1.2.5 void Task_1ms ( )

"Task_1ms"
is called every ms with VERY LOW jitter!!!
It is called from ISR of the ADC - so it should be very short.

Usage: (examples)

- Driving a Stepper Motor
- Set signals at Digital Pins depending on measurements (for external devices) with precision timing

Definition at line 424 of file ADC_Interrupt_Nano.ino.

```
424                 {
425 static float _old, _val;
426 static int count;
427 static int test;
428
429   //digitalWrite(4,HIGH);
430   PORTD |= (1<<4);
431
432   //digitalWrite(4,LOW);
433   PORTD &= ~(1<<4);
434
435 }
```

7.1.2.6 void Task_Loop ( )

Definition at line 398 of file ADC_Interrupt_Nano.ino.

```
398                 {
399 char c;
400   //digitalWrite(3,HIGH);
401   PORTD |= (1<<3);
402
403   //digitalWrite(3,LOW);
404   PORTD &= ~(1<<3);
405
406 }
```

7.1.3 Variable Documentation

7.1.3.1 float Alpha[CHNUM]

Alpha[channel] is the Filter-Parameter for each channel.
Is set in "setup". "Should be greater than 0.75 to see an effect.
MUST be less than 1.0 !!!

Definition at line 212 of file ADC_Interrupt_Nano.ino.

7.1.3.2 volatile int analogVal

Definition at line 201 of file ADC_Interrupt_Nano.ino.

7.1.3.3 float av[CHNUM]

av[channel] contains the last filtered value of a channel
It is build in ISR and used by loop to generate ouput-value for the plotter
"oldVal[ ]" is the last filtered value of a channel Read Only

Definition at line 221 of file ADC_Interrupt_Nano.ino.

7.1.3.4 volatile bool Busy =false

"Busy" is set together with ReadyFlag[channel] and disable reading new AD values in ISR
until it is reset in "loop"

Definition at line 228 of file ADC_Interrupt_Nano.ino.

7.1.3.5 int Channel =0

Definition at line 203 of file ADC_Interrupt_Nano.ino.

7.1.3.6 volatile int IrqReadyFlag

"IrqReadyFlag" is set after IRQ_SAMPLES for each channel

Definition at line 195 of file ADC_Interrupt_Nano.ino.

7.1.3.7 volatile unsigned long lastTaskTime =micros()

"lastTaskTime" is used by ISR to call Task_1ms br>

Definition at line 256 of file ADC_Interrupt_Nano.ino.

7.1.3.8 float oldVal[CHNUM]

Definition at line 221 of file ADC_Interrupt_Nano.ino.

7.1.3.9 char out[100]

"out[ ]" is the buffer of the line to send to the plotter. Only internal usage !

Definition at line 236 of file ADC_Interrupt_Nano.ino.

7.1.3.10 bool output = false

"Task_Loop"
is called from "loop"r
It should be very short because loop should be able to react fast Often called, but with jitter !

Usage: (examples) as human interface

- Driving a NeoPixel showing results as Color
- Set signals at Digital Pins depending on measurements (HEART-BEAT)

Definition at line 396 of file ADC_Interrupt_Nano.ino.

7.1.3.11 char* pt =out

internal usage

Definition at line 241 of file ADC_Interrupt_Nano.ino.

7.1.3.12 char* strpt

internal usage

Definition at line 246 of file ADC_Interrupt_Nano.ino.

7.1.3.13 float val

internal usage

Definition at line 250 of file ADC_Interrupt_Nano.ino.

7.2 Demo.jpg File Reference

7.3 Timing.jpg File Reference