

CONSTRUCTING DRIVERS FOR HEADER RULES COMPLIANCE (Kieras, 2012)

NEW: ADC_DR1 MODULE on CVAVR 3 / CL2 – Rev. 10-02-2018 / R.Oliva

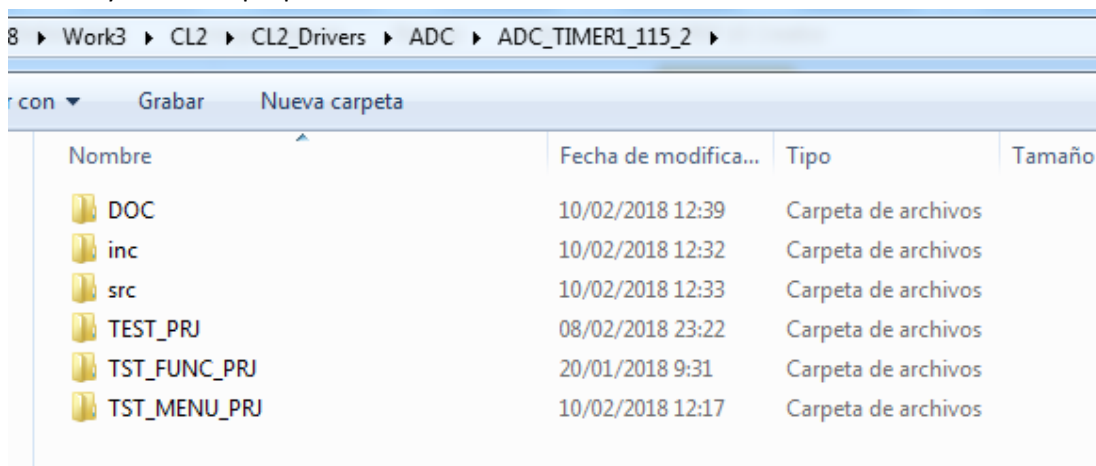
TIMER 1 TRIGGERED VERSION @ 115.2 kHz - Updated with test results 11- 02-2018

Reformatted – printout in PDF form

1. CREATION 10-02-2018 – ADC_DR1 - (Files ADC_DR1.c / .h, Only for ATmega1284P 10 bit ADC)

This document in: "C:\cvavr328\Work3\CL2\CL2_Drivers\ADC\ADC_TIMER1_115_2\DOC
\ADC_DR1(TIMER1_115)_ASSY+TEST_v10-02-2018.docx"

Directory structure proposed:



Nombre	Fecha de modifica...	Tipo	Tamaño
DOC	10/02/2018 12:39	Carpeta de archivos	
inc	10/02/2018 12:32	Carpeta de archivos	
src	10/02/2018 12:33	Carpeta de archivos	
TEST_PRJ	08/02/2018 23:22	Carpeta de archivos	
TST_FUNC_PRJ	20/01/2018 9:31	Carpeta de archivos	
TST_MENU_PRJ	10/02/2018 12:17	Carpeta de archivos	

2. BASIC FUNCTIONALITY:

2.1 Aim and Source of code: This driver for CL2bm1 board is aimed to bring the internal 10-bit successive approximation ADC to a programmer's interface. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND). The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion.

2.2 Features of the ADC on ATmega1284P (Atmel/Microchip Datasheet 2016, 42719C):

- 10-bit Resolution
- 0.5 LSB Integral Non-Linearity
- ± 2 LSB Absolute Accuracy
- 13 - 260 μ s Conversion Time
- Up to 15kSPS at Maximum Resolution
- 8 Multiplexed Single Ended Input Channels
- Differential mode with selectable gain at 1x, 10x or 200x(1)
(Not available on PDIP modules such as the one use in CL2bm1)
- Optional Left Adjustment for ADC Result Readout
- 0 - VCC ADC Input Voltage Range
- 2.7V - VCC Differential ADC Voltage Range
- Selectable 2.56V or 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

2.3 Block diagram and operation in CL2bm1 board: A block diagram of the ADC is shown in Figure 1. The ADC has a separate analog supply voltage pin, AVCC. AVCC is connected on the VCC on CL2bm1 thru a simple RC Filter as shown in Figure 2, where also the main ADC connector JA1 is visible:

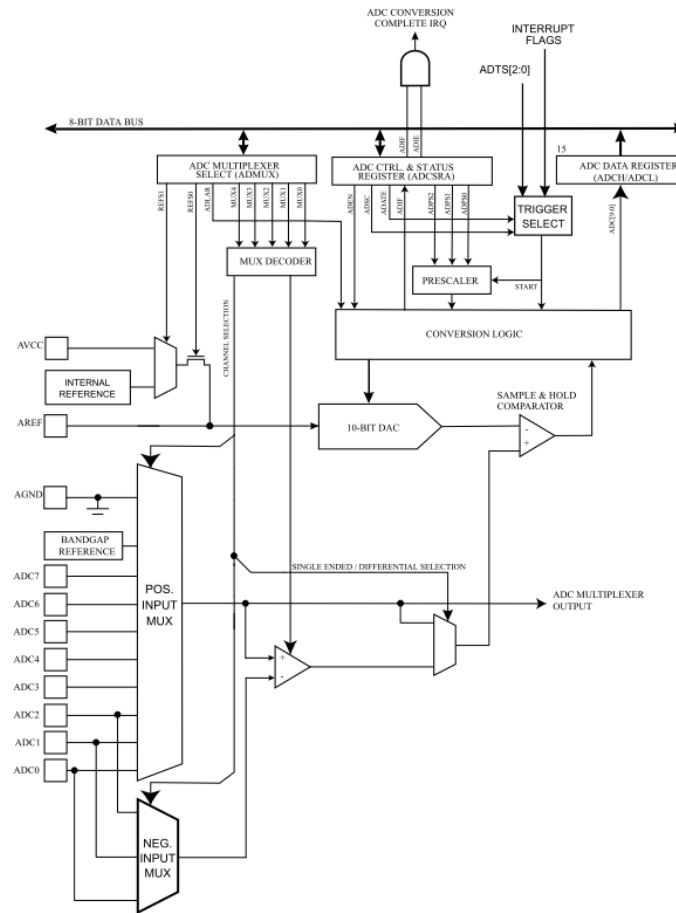


Figure 1 – Internal ADC of the ATmega1284P

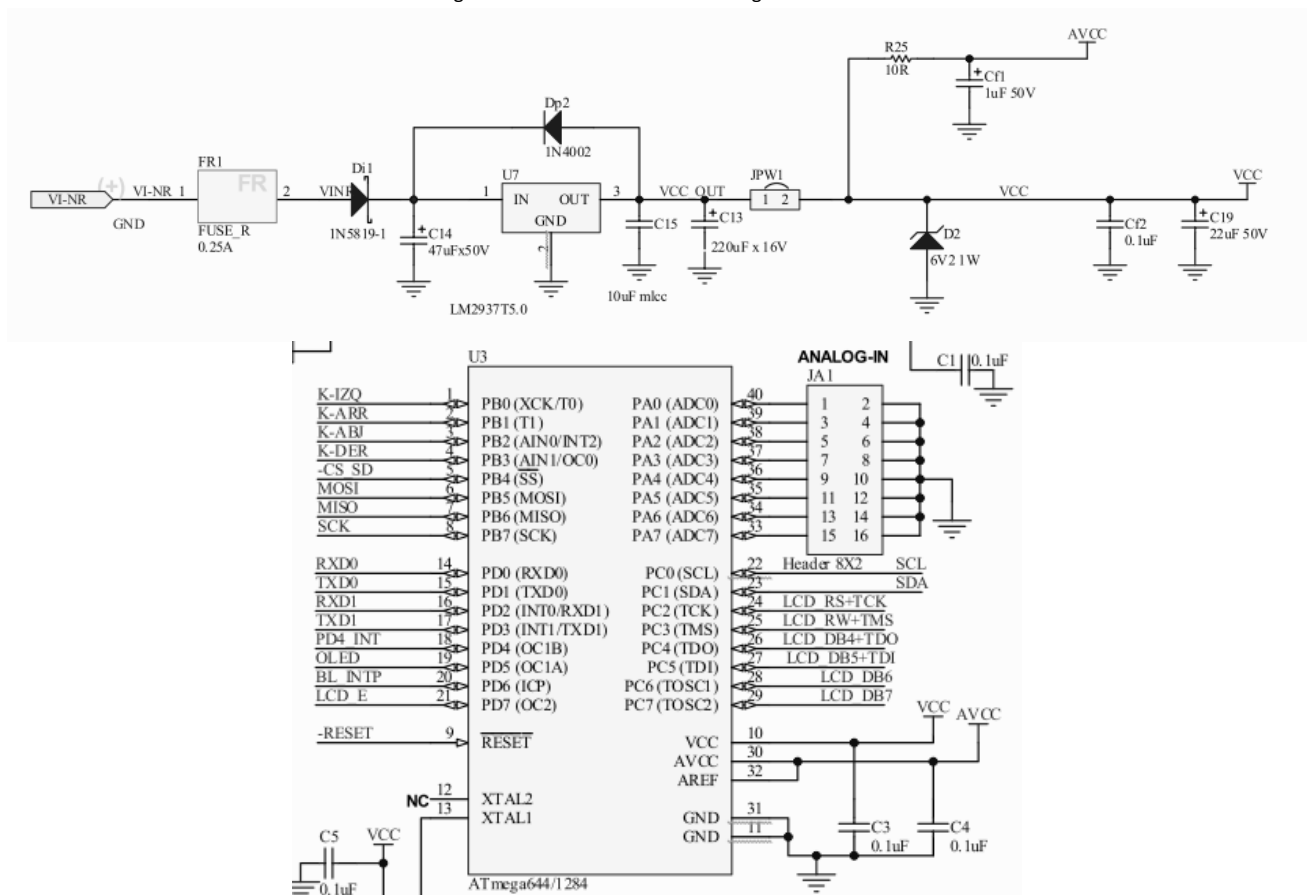
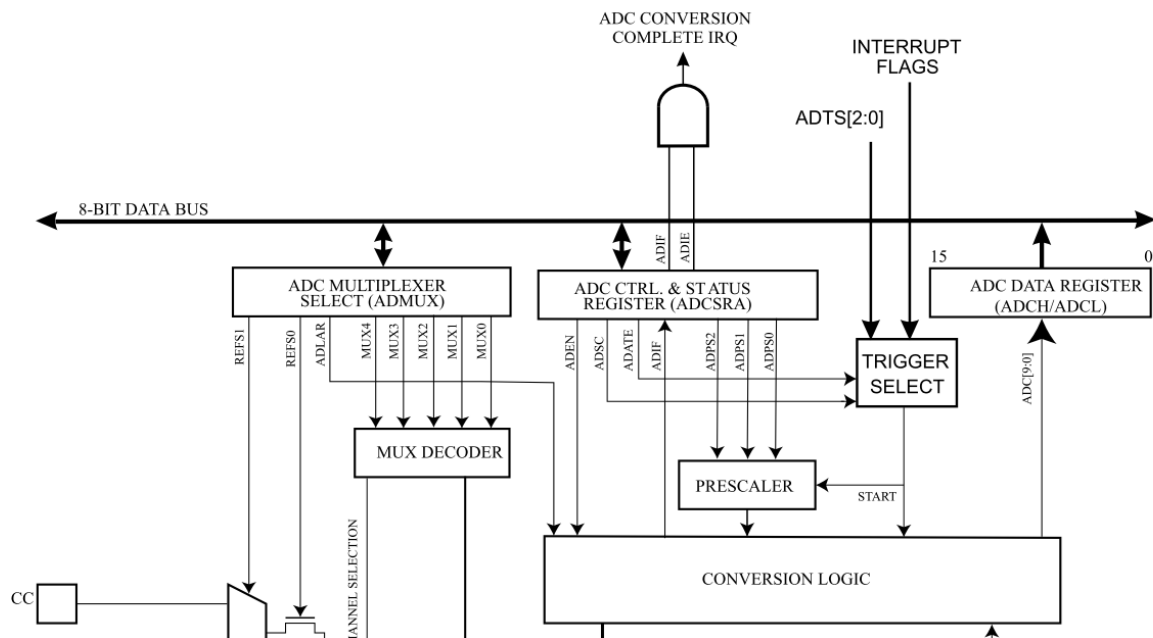


Figure 2 – AVCC and ADC connection on CL2bm1 board (2010)

Operation of the ADC on CL2bm1 board: The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register (Figure 3). The internal voltage reference must be decoupled by an external capacitor at the AREF pin to improve noise immunity.



Voltage reference and input channel selections will not take effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

- By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADC Left Adjust Result bit ADMUX.ADLAR.
- If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion: Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a second conversion completes before ADCH is read, neither register is updated and the result from the second conversion is lost.
- When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.
- The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

- If Auto Triggering is enabled, single conversions can be started by writing ADCSRA.ADSC to '1'. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as '1' during a conversion, independently of how the conversion was started. If Auto Triggering is enabled, single conversions can be started by writing ADCSRA.ADSC to '1'. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as '1' during a conversion, independently of how the conversion was started.

Figure 25-2. ADC Auto Trigger Logic

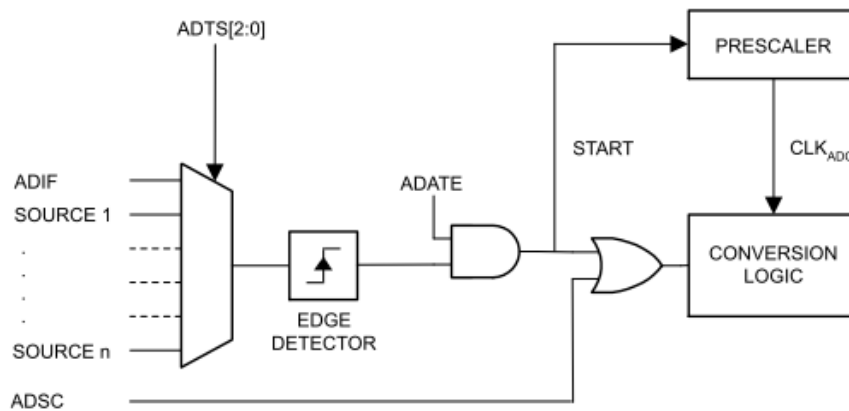


Figure 4 –AutoTrigger logic / ADC on ATmega1284P

- By default, the successive approximation circuitry requires an input clock frequency between 50kHz and 200kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200kHz to get a higher sample rate. The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100kHz. The prescaling is selected by the ADC Prescaler Select bits in the ADC Control and Status Register A (ADCSRA.ADPS) (Figure 5).

Prescaling and Conversion Timing

Figure 25-3. ADC Prescaler

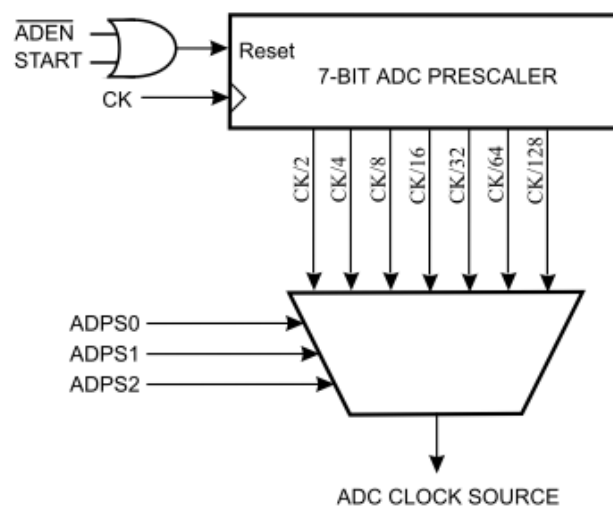


Figure 5 – Prescaler for ADC_Clock / ADC on ATmega1284P

When initiating a single ended conversion by writing a '1' to the ADC Start Conversion bit (ADCSRA.ADSC), the conversion starts at the following rising edge of the ADC clock cycle. A normal conversion takes 13 ADC clock cycles.

- The first conversion after the ADC is switched on (i.e., ADCSRA.ADEN is written to '1') takes 25 ADC clock cycles in order to initialize the analog circuitry.
- The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC Data Registers (ADCL and ADCH), and the ADC Interrupt Flag (ADCSRA.ADIF) is set.
- In Single Conversion mode, ADCSRA.ADSC is cleared simultaneously. The software may then set ADCSRA.ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

- When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place two ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic. (Figure 6 – top)
- In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADCRSA.ADSC remains high. (Figure 6 – bottom)

Timing Diagrams of ADC ATmega1284P - on CL2bm1 board:

Figure 25-6. ADC Timing Diagram, Auto Triggered Conversion

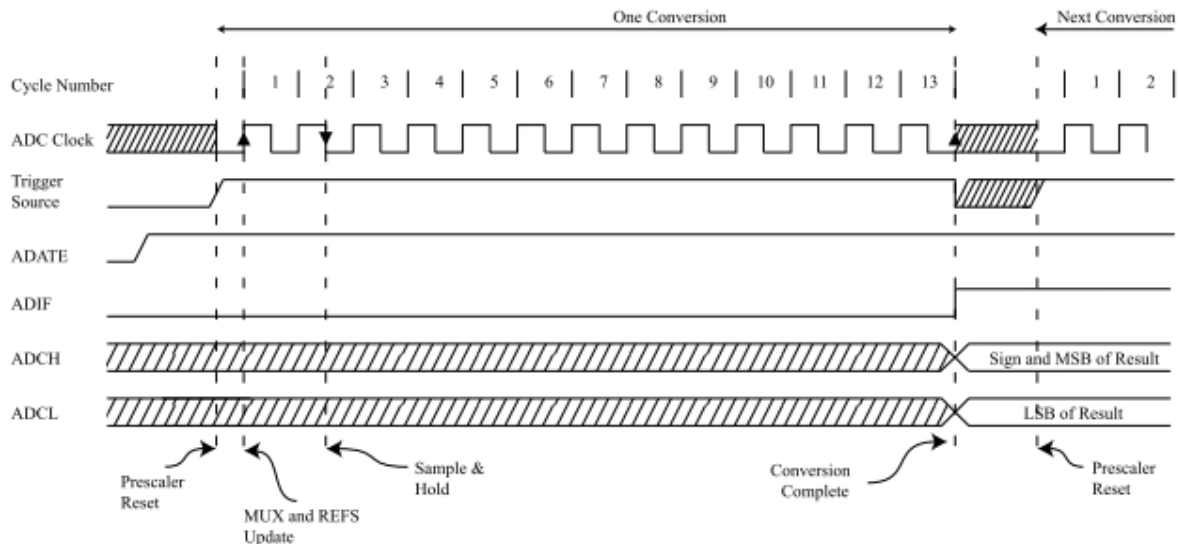


Figure 25-7. ADC Timing Diagram, Free Running Conversion

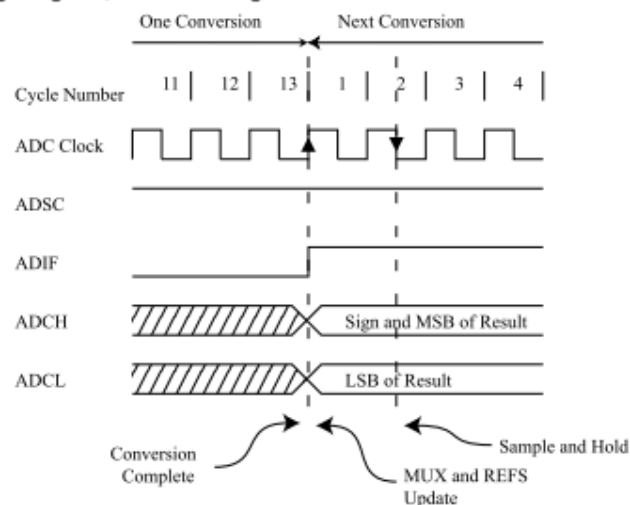


Figure 6 – Timing Diagram AutoTrigger (top) and Free Running (bottom), ADC_Clock / ADC on ATmega1284P

2.4 Implementation on CL2bm1: The registers related to ADC on Mega1284P are as follows:

25.8.1. ADC Multiplexer Selection Register

Name: ADMUX
Offset: 0x7C
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin
10	Internal 1.1V Voltage Reference with external capacitor at AREF pin
11	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Bit 5 – ADLAR: ADC Left Adjust Result

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see the [ADCL](#) and [ADCH](#).

MUX[4:0]	Single Ended Input	Positive Differential Input	Negative Differential Input
11110	1.1V (V_{BG})	N/A	
11111	0V (GND)		

MUX[4:0]	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001	N/A	ADC1	ADC0	10x

25.8.2. ADC Control and Status Register A

Name: ADCSRA
Offset: 0x7A
Reset: 0x00
Property: -

Bit 4 – ADIF: ADC Interrupt Flag

This bit is set when an ADC conversion completes and the Data Registers are updated.

Bit 3 – ADIE: ADC Interrupt Enable

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – ADEN: ADC Enable

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off.

Bit 6 – ADSC: ADC Start Conversion

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

Bit 5 – ADATE: ADC Auto Trigger Enable

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

Bits 2:0 – ADPSn: ADC Prescaler Select [n = 2:0]

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

ADPS[2:0]	Division Factor
000	2
001	2
ADPS[2:0]	Division Factor
010	4
011	8
100	16
101	32
110	64
111	128

Name: ADCSRB
Offset: 0x7B
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
		ACME				ADTS2	ADTS1	ADTS0
Access		R/W				R/W	R/W	R/W
Reset		0				0	0	0

ADTS[2:0]	Trigger Source
000	Free Running mode
001	Analog Comparator
010	External Interrupt Request 0
011	Timer/Counter0 Compare Match A
100	Timer/Counter0 Overflow
101	Timer/Counter1 Compare Match B
110	Timer/Counter1 Overflow
111	Timer/Counter1 Capture Event

Data Register ADCH:ADCL

Name: ADCL and ADCH
Offset: 0x78
Reset: 0x00
Property: ADLAR = 0

Bit	15	14	13	12	11	10	9	8
							ADC9	ADC8
Access							R	R
Reset							0	0

Bit	7	6	5	4	3	2	1	0
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Access	R	R	R	R	R	R	R	R
Reset	0	0	0	0	0	0	0	0

11.12.3. Power Reduction Register 0

Name: PRR0
Offset: 0x64
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	PRTWI	PRTIM2	PRTIM0	PRUSART1	PRTIM1	PRSPI0	PRUSART0	PRADC
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 0 – PRADC: Power Reduction ADC

Writing a logic one to this bit shuts down the ADC. The ADC must be disabled before shut down. The analog comparator cannot use the ADC input MUX when the ADC is shut down.

```

/*****
** power_adc_enable() macro definition.(from /avr/power.h)
** #if defined(__AVR_HAVE_PRR_PRADC)
** #define power_adc_enable() (PRR &= (uint8_t)~(1 << PRADC))
** #define power_adc_disable() (PRR |= (uint8_t)(1 << PRADC))
** #endif
** Other definitions:
** #define _BV (bit) (1 << (bit))
**
**/

```

```

/*****
**
* @file adc.c
*
* @brief implementation of ADC API
*
* This file provides an API implementation of the ADC routines
*
*/

#include <avr/power.h>

void adc_init(e_adc_mode a_mode) {

    // enable power to ADC
    power_adc_enable();

    // default prescaler = 128. The ADC clock = F_CPU/128 = 125 kHz
    ADCSRA = (_BV(ADEN) | 0x07); // Sets ADEN=b7 and ADPS=111

    // configure mode
    if (E_SINGLE_SHOT == a_mode) {
        ADCSRB = 0x00; // ADTS =000 is Free running mode..
    }
    else {
        // set the appropriate mode
        ADCSRB = (a_mode - 1); // ADTS = b2b1b0 alternative auto Trigger source..
        // enable auto-trigger mode
        ADCSRA = _BV(ADATE); // ADATE=b5 (A) Enables Auto triggering..
    }

    // set data format, Vcc reference and channel zero by default
    ADMUX = 0x00;
    ADMUX &= ~_BV(ADLAR);
    adc_reference_set(E_ADC_EXTERNAL_AVCC);
}

```

The parameter a_mode can be one of the following, defined in adc.h

```

/*
**
* @brief possible ADC operational modes
*/
typedef enum _e_adc_mode {
    /// single shot mode, ADC won't be re-triggered
    E_SINGLE_SHOT = 0,

    /// free running mode. ADC will be re-triggered constantly after every conversion
    E_AT_FREERUN,

    /// analog comparator re-trigger
    E_AT_AC,

    /// external interrupt re-trigger
    E_AT_EINT0,

    /// Timer0 Compare A re-trigger
    E_AT_TIMER0_COMP_A,

    /// Timer0 Overflow re-trigger
    E_AT_TIMER0_OVFL,

    /// Timer1 Compare B re-trigger
    E_AT_TIMER1_COMP_B,

    /// Timer1 Overflow re-trigger
    E_AT_TIMER1_OVFL,

    /// Timer1 Capture event re-trigger
    E_AT_TIMER1_CAPTURE_EVENT,

    /// enumeration counter
    E_AT_MODE_LAST
} e_adc_mode;

```

A predefined macro to fix the ADC Ref Voltage:

```

/**
* @brief set ADC analog reference voltage source

```



```

*
* @param a_ref reference voltage source
*/
#define adc_reference_set(__ref) \
    ADMUX &= 0x3f; \
    ADMUX |= ((__ref & 0x03) << 6)

```

Possible values of the ADC Ref Voltage

```

/**
* @brief enumeration defining possible analog reference voltage sources
*/
typedef enum _e_adc_ref {
    // voltage provided to external AREF pin
    E_ADC_EXTERNAL_AREF = 0x00,

    // use analog voltage Vcc as reference (5V)
    E_ADC_EXTERNAL_AVCC,

    /// internal 1.1 analog reference will be used
    E_ADC_REF_INTERNAL_11 = 0x03
} e_adc_ref;

```

A predefined macro to fix the ADC MPX channel is:

```

/**
* @brief set ADC multiplexer to a specified channel
*
* @param a_channel channel (0-8)
*/
#define adc_channel_set(__channel) \
    ADMUX &= 0xf0; \
    ADMUX |= (__channel & 0x0f)

```

2.5 ADC_Dr1.c / .h on CL2bm1: This second implementation is oriented to **Timer1 Triggered 115.2 kHz** ADC clock, and the files ADC_dr1.c/.h are located in:

C:\cvavr328\Work3\CL2\CL2_Drivers\ADC\ADC_TIMER1_115_2\src; \inc; \TEST_PRJ

NOTE: The only difference is in ADC_Init(), and the timing of the ADC ISR, which is controlled by overflow of Timer1. Since ElmChan's FFS implementation on CVAVR uses Timer1 for DiskProc() routine, this OVF is set to 100Hz, giving a sampling frequency for all 8 channels of 12.5 Hz. A document about the Testing and a sample project of this configuration can be found in:

C:\cvavr328\Work3\CL2\CL2_Tests_withWizard\CL2_ADC_wiz\ADC_Timer1Trigg_115kHz\

→ Project document: Test3_Timer1Trig_at115_2kHz_PantallasADC_7-2-18.docx

→ Sample project;: ADC_Timer1_Trig07022018.prj

Also a brief description in this document in item: **APP.2.4 TEST PROJECT -3**

Rule #1 – OK Groups functional operations

Rule #2 OK – “Include guards”, used here:

```

#ifndef ADC_INCLUDED
#define ADC_INCLUDED
...
#endif

```

Rule #3 All required declarations to use the module appear in the ADC_DR1.h file

Rule #4 OK → .h file only contains declarations, and is included by the .c file

```

.C FILE:
// *****
// local functions
// *****

```

NO LOCAL FUNCTIONS IN .C FILE

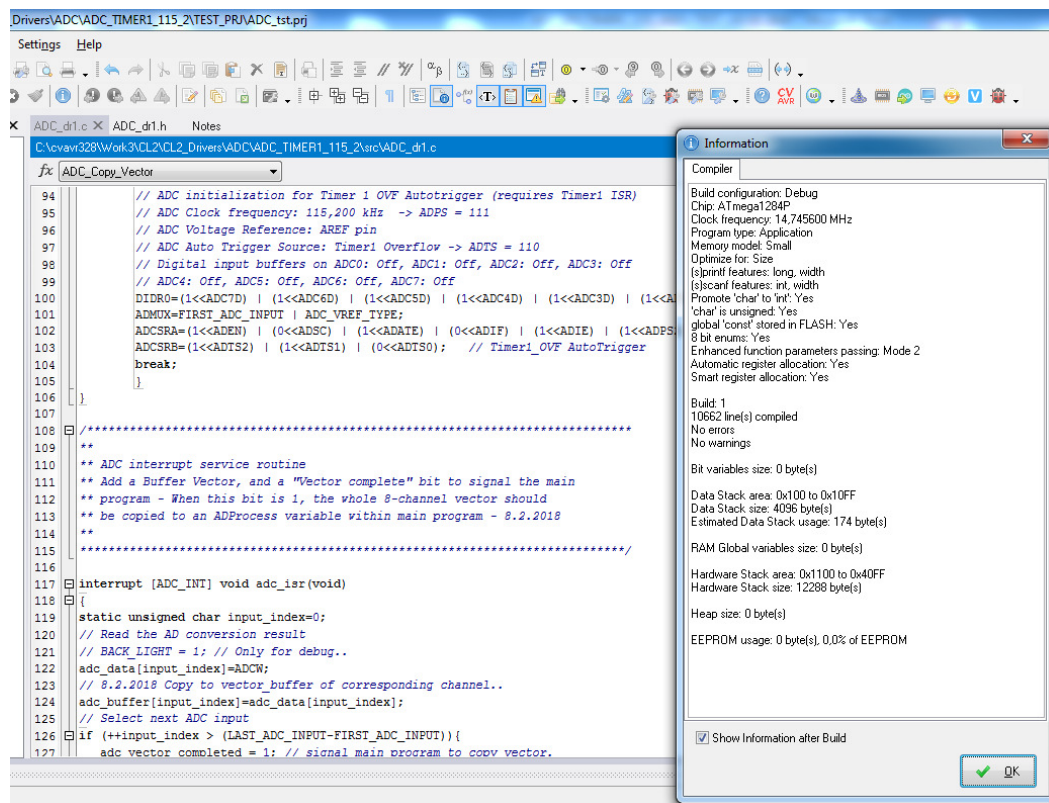


Figure 7 – Auto compilation of drivers – ok, Timer1Trig ADC on ATmega1284P

Rule #10 OK: ADC_DR1.c includes at beginning the file ADC_DR1.h, other files not applicable, since no external .h functions are required for this module.

Rule #11 OK no .c files #included.

TESTING:

A) 10.02.2018 TEST ADC_Dr1 WITH TSTMENU_ADC_PRJ,
C:\cvavr328\Work3\CL2\CL2_Drivers\ADC\ADC_TIMER1_115_2\TST_MENU_PRJ\
Test_Menu_ADCDr.prj

This Project uses the Timer1_115.2 ADC driver and includes a simple Menu System borrowed from PWRC2, to test the reading.

Mostly is the same as FR115_2 (changes in yellow): We have a main Tst_ADCMenu_main.c file, which contains main(). An annex file called Basic_Menu.c contains the UART0_Menu() routine (from PWRC2, simplified) and auxiliary functions. Initially we do not use the Menu, we simply call the ADC_Copy_Vector() function within the loop when the adc_vector_completed is set. This function is as follows:

```

/*****
** Protected Copy of ADC Buffer to Vector Buffer FPSChannel[]
** 8.2.2018
*****/
void ADC_Copy_Vector(void)
{
    unsigned char j_cnt;
    CLI();
    for(j_cnt=0; j_cnt<(LAST_ADC_INPUT-FIRST_ADC_INPUT+1); j_cnt++){
        FPSChannel[j_cnt] = adc_buffer[j_cnt];
    }
    adc_vector_completed = 0; // Reset signal to copy vector..
    SEI();
}

```

```

(ISR taken from Wizard, Tests, Timer1_autotrigger):
// Original ADC interrupt service routine
// with auto input scanning, Timer 1 Trig
interrupt [ADC_INT] void adc_isr(void)
{
    static unsigned char input_index=0;
    BACK_LIGHT = 1; // Signal in ISR.. 7.2.2018
    // Read the AD conversion result
    adc_data[input_index]=ADCW;
    // Select next ADC input
    if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT))
        input_index=0;
    ADMUX=(FIRST_ADC_INPUT | ADC_VREF_TYPE)+input_index;
    // Delay needed for the stabilization of the ADC input voltage
    delay_us(10);
    BACK_LIGHT = 0; // Signal out ISR.. 7.2.2018
    // No autostart..
}

New ISR 10.2.2018:
/*****
**
** ADC interrupt service routine
** Add a Buffer Vector, and a "Vector complete" bit to signal the main
** program - When this bit is 1, the whole 8-channel vector should
** be copied to an ADProcess variable within main program - 8.2.2018
**
*****/
interrupt [ADC_INT] void adc_isr(void)
{
    static unsigned char input_index=0;
    // Read the AD conversion result
    BACK_LIGHT = 1; // Only for debug..
    adc_data[input_index]=ADCW;
    // 8.2.2018 Copy to vector_buffer of corresponding channel..
    adc_buffer[input_index]=adc_data[input_index];
    // Select next ADC input
    if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT)){
        adc_vector_completed = 1; // signal main program to copy vector.
        input_index=0;           // Reset input_index
    }
    ADMUX=(FIRST_ADC_INPUT | ADC_VREF_TYPE)+input_index;
    // Delay needed for the stabilization of the ADC input voltage
    delay_us(10);
    // Start the AD conversion - comment out 10.2.2018
    // ADCSRA|=(1<<ADSC); comment out the AutoConversion..
    BACK_LIGHT = 0; // Only for debug..
}

ADC Initialization - REMEMBER to set the correct parameter..!!!
/*****
**
** Initializes the ADC
** void ADC_Init(uint8_t ADC_param)
** Parameters: uint8_t ADC_param
** Select (A) FreeRunning or (B) Timer10VF as trigger,
** ADC clock is 115.2 kHz @ 14.7456 MHz Clk
** (A)#define ADC_INIT_FR_115K2 0
** (B)#define ADC_INIT_TMR1_115K2 1
**
** Returns: NONE
** If ADC_param == ADC_INIT_TMR1_115K2
** then definition of Timer1 OVF ISR (used by ElmChanFFS)
** will be required, at 100Hz by default.
** (use (B) for this version)
*****/

void ADC_Init(uint8_t ADC_param)
{
    switch (ADC_param)
    {
        case ADC_INIT_FR_115K2:
            // ADC initialization Free Running at 115.2 kHz = 14.7456E06 / 128 Hz
            // ADC Clock frequency: 115,200 kHz -> ADPS = 111
            // ADC Voltage Reference: AREF pin
            // ADC Auto Trigger Source: Free Running -> ADTS = 000
            // Digital input buffers on ADC0: Off, ADC1: Off, ADC2: Off, ADC3: Off
            // ADC4: Off, ADC5: Off, ADC6: Off, ADC7: Off
            DIDR0=(1<<ADC7D) | (1<<ADC6D) | (1<<ADC5D) | (1<<ADC4D) | (1<<ADC3D) | (1<<ADC2D) | (1<<ADC1D) | (1<<ADC0D);
            ADMUX=FIRST_ADC_INPUT | ADC_VREF_TYPE;
            ADCSRA=(1<<ADEN) | (1<<ADSC) | (1<<ADATE) | (0<<ADIF) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
            ADCSRB=(0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0); // Free Running -> ADTS = 000
            break;

        case ADC_INIT_TMR1_115K2:

```

```

// ADC initialization for Timer 1 OVF Autotrigger (requires Timer1 ISR)
// ADC Clock frequency: 115,200 kHz -> ADPS = 111
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: Timer1 Overflow -> ADTS = 110
// Digital input buffers on ADC0: Off, ADC1: Off, ADC2: Off, ADC3: Off
// ADC4: Off, ADC5: Off, ADC6: Off, ADC7: Off
DIDR0=(1<<ADC7D) | (1<<ADC6D) | (1<<ADC5D) | (1<<ADC4D) | (1<<ADC3D) | (1<<ADC2D) | (1<<ADC1D) | (1<<ADC0D);
ADMUX=FIRST_ADC_INPUT | ADC_VREF_TYPE;
ADCSRA=(1<<ADEN) | (0<<ADSC) | (1<<ADATE) | (0<<ADIF) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
ADCSRB=(1<<ADTS2) | (1<<ADTS1) | (0<<ADTS0); // Timer1_OVF AutoTrigger
break;
}
}

```

AND.. We need to add the Timer1 Initialization and ISR (in final program, we will use the slightly different approach used in ELM_FFS driver – use different prescaler so requires different load values, but the TIM1_OVF is similar in resulting OVF frequency – 100Hz)

```

/*****
**
** Timer1 overflow interrupt service routine - Added for testing, from Wizard
** for Version Timer1_115_2 10.2.2018
**
*****/

interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
// Reinitialize Timer1 value
TCNT1H=0xB800 >> 8;
TCNT1L=0xB800 & 0xFF;
// Place your code here
}

/*****
** Initialize Timer1 for overflow interrupt - Added for testing, from Wizard
** for Version Timer1_115_2 10.2.2018
**
** Timer/Counter 1 initialization
** Clock source: System Clock
** Clock value: 1843,200 kHz (in TCCR1B, CS12:CS10 = 010 -> CLK/8)
** Mode: Normal top=0xFFFF
** OC1A output: Disconnected
** OC1B output: Disconnected
** Noise Canceler: Off
** Input Capture on Falling Edge
** Timer Period: 10 ms
** Timer1 Overflow Interrupt: On
** Input Capture Interrupt: Off
** Compare A Match Interrupt: Off
** Compare B Match Interrupt: Off
** In ELM_FFS Timer1 initialization value after overflow (equiv. to 0x00FF70 )
** #define T1_INIT (0x10000L-(MCU_CLOCK_FREQUENCY/(T1_PRESC*T1_OVF_FREQ)))
** ..BUT! in TCCR1B, CS12:CS10 = 101 (-> CLK/1024 = 14.4kHz..-> T=69.44us)
** .. so 0x10000-0xFF70 = 0x090 (144 dec)-> 144*T = 10ms, -> FT1= 1/10ms=100Hz
*****/

void Timer1_Initialize(void){
TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) | (0<<WGM11) | (0<<WGM10);
TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) | (0<<CS12) | (1<<CS11) | (0<<CS10);
TCNT1H=0xB8;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
// Timer/Counter 1 Interrupt(s) initialization
TIMSK1=(0<<ICIE1) | (0<<OCIE1B) | (0<<OCIE1A) | (1<<TOIE1);
}

```

Also, the Three driver functions (UART0, ADC and TWI12A) are called within the PRJ. The loop in main() looks like the following:

```

while (1)
{
// Place your code here
delay_ms(1); // Minimal delay 8.2.2018
//Check_UART0_Menu(MSG_opt);
if (adc_vector_completed == 1){
ADC_Copy_Vector();
}

if (Flag_Sec_Change == 1){
Flag_Sec_Change = 0;
}
}

```



```

RTC_result = (int8_t)(rtc_get_time(&urTCHour, &urTCMin, &urTCSec));
sprintf(s, "\n\r ADC Read at %02d:%02d:%02d ", urTCHour, urTCMin, urTCSec);
printf("%s", s);
for(j_1=0; j_1<(LAST_ADC_INPUT-FIRST_ADC_INPUT+1); j_1++){
    sprintf(s, "Ch(%d):%04d ", j_1, FPSChannel[j_1]);
    printf("%s", s);
}
}

} // end while(1)
}

```

10.02.2018 Tested With AOIP CP6632 Calibrator Signal, 1.000 V applied on Ch0 and Ch1

```

.Drivers\ADC\ADC_TIMER1_115_2\TST_MENU_PRA\Test_Menu_ADCDr.prj
Settings Help
Hex Code:
ADC Read at 19:21:08 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:09 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:10 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:11 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:12 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:13 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:14 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:15 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:16 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:17 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:18 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:19 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:20 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:21 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:22 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:23 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:24 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:25 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:26 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:27 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:28 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:29 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:30 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:31 Ch(0):0202 Ch(1):0202 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
ADC Read at 19:21:32 Ch(0):0202 Ch(1):0203 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000

```

Figure T.1 – Terminal output of testing, Timer1Trig ADC on ATmega1284P – see channel readings are corrected now
Problem Corrected! – (channel offset of one, observed in FreeRun mode..)

(10.2.2018 Problem was: on printing, the channels are offset by One: Printed Ch1 -> is hardware Ch0, Printed Ch 2 -> is hardware Ch 1, Printed Ch3 -> is hardware Ch2,Printed Ch7 -> is hardware Ch0 // We tried changing the adc_vector_complete flag from bit to uint8_t, but to no effect..

Problem then is Out_of_Phase error in FreeRun Mode → avoid

10.02.2018, 19:48 Tested With AOIP CP6632 Calibrator Signal, 2.048 V applied on Ch0 and Ch7

```

.Drivers\ADC\ADC_TIMER1_115_2\TST_MENU_PRA\Test_Menu_ADCDr.prj
Settings Help
Hex Code:
ADC Read at 19:48:09 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:10 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:11 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:12 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:13 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:14 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:15 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:16 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:17 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:18 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:19 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:20 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:21 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:22 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:23 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:24 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:25 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:26 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:27 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:28 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:29 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:30 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:31 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:32 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418
ADC Read at 19:48:33 Ch(0):0418 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0418

```

Figure T.2 – Terminal output of testing, Timer1Trig ADC on ATmega1284P – Ch0 and Ch7 with 2.048 V

```

/*****
** Protected Copy of ADC Buffer to Vector Buffer FPSChannel[]
** 8.2.2018 -Rev 10.2.2018 -Add toggle of BackLight PD.6 (pin20)
** Duration of copy: 26us (scope)
*****/
void ADC_Copy_Vector(void)
{
    unsigned char j_cnt;
    CLI();
    BACK_LIGHT = 1; // Only for debug..
    for(j_cnt=0; j_cnt<(LAST_ADC_INPUT-FIRST_ADC_INPUT+1); j_cnt++){
        FPSChannel[j_cnt] = adc_buffer[j_cnt];
    }
    adc_vector_completed = 0; // Reset signal to copy vector..
    BACK_LIGHT = 0; // Only for debug..
    SEI();
}

```

As seen in the code, the BackLight pin is set on entering ADC_Copy_Vector(), which happens only when all 8 channels have been read. The calculated, expected period 12.5 Hz.. OK!! This can be seen in the figures T.3 and T.4 below. The copy process takes 26us approximately.

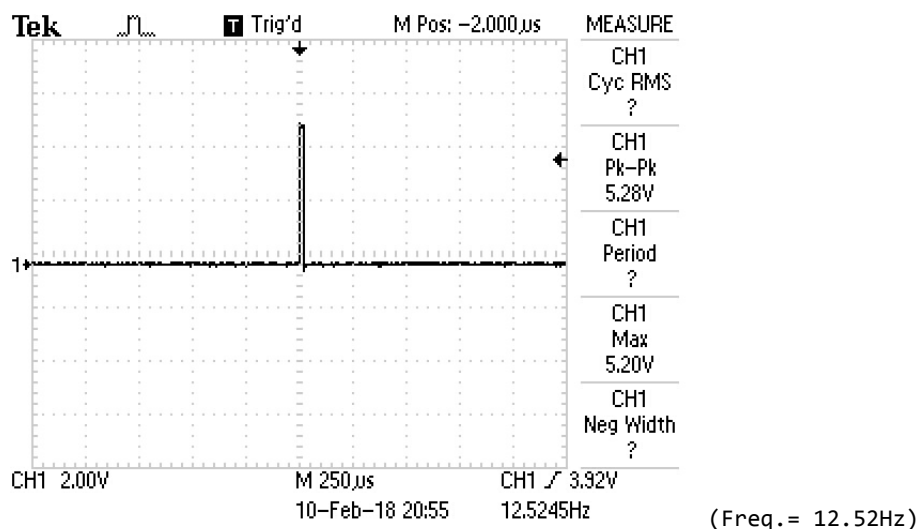


Figure T.3 – BACK_LIGHT = PD.6 Pin set within the complete vector-copy for loop, executed when 8 channels are sampled. - Timer1Trig ADC on ATmega1284P

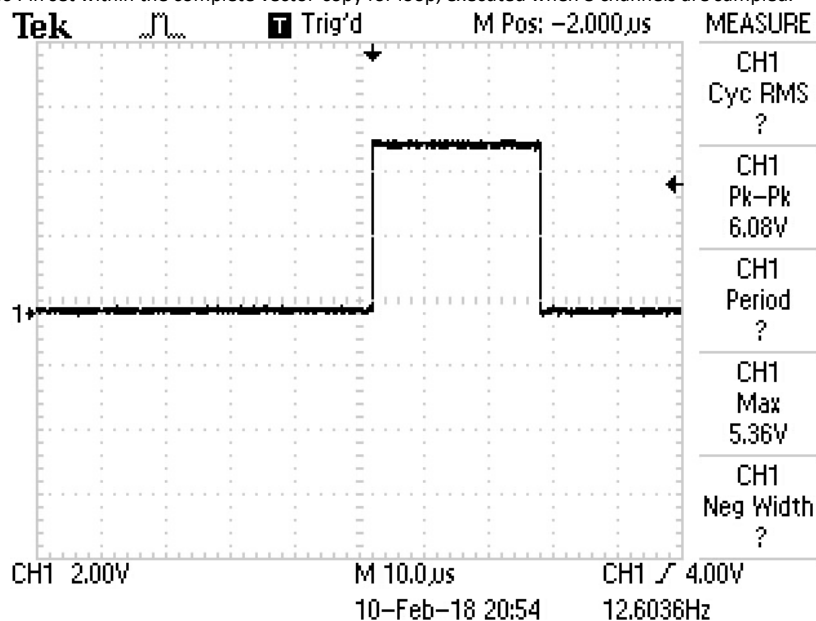


Figure T.4 – Expanded image of BACK_LIGHT = PD.6 Pin set within the complete vector-copy for loop, executed when 8 channels are sampled. Shows 26 us duration of complete copy - Timer1Trig ADC on ATmega1284P

B) 11.02.2018 READING VALUES AGAINST VOLTAGE INJECTED BY CALIBRATOR:

The values read were taken with an AOIP 6632 calibrator, injecting known voltage values into each of the analog channels. All of the channels were taken to GND via a 4.7K resistor, to avoid noise in the absence of signal. Minimum readings were 0000, and maximum 1023 with 4.999 V applied. The table values were copied on a paper, shown in next figure:

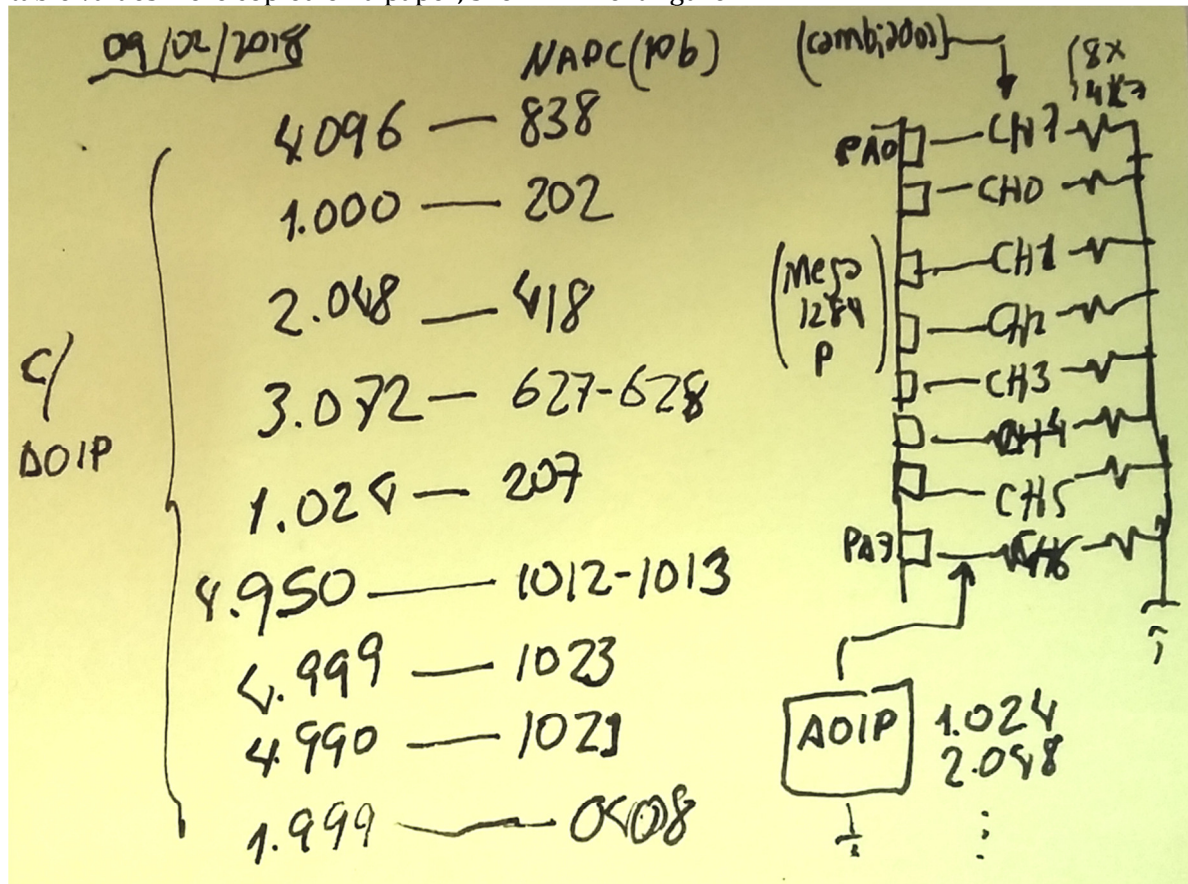


Figure T.5 – Raw ADC readings for different values of input voltage on one of the channels (CH0) of CL2bm1, with AOIP 6632 Calibrator

Datos de lectura ADC - CL2bm1

Calibrador AOIP y lecturas Fluke 115

11.02.2018

Codigo de placa:

Instrum: AOIP 6632 S/n WEM41020-000 2003R R41 0898A, Fluke F115 / S/N15720249

	Vdis (AOIP)	Terminal Read
	0.000	0000
	1.000	0202
	1.024	0207
	1.999	0408
	2.048	0418
	3.072	0627
	4.096	0838
	4.950	1012
	4.990	1021
	4.999	1023

Table T.1 – Raw ADC readings to Vdis AOIP

Measurement setup is shown in next photo (Figure T.6):



Figure T.6 – MEaurement setup for Calibrator signal injection to CL2bm1 ADC inputs (10.02.2018)

Datos de lectura ADC - Placa CL2bm1 - Driver ADC_Timer1 Trig115.2kHz
Calibrador AOIP + Fluke 115 - 11-02-2018

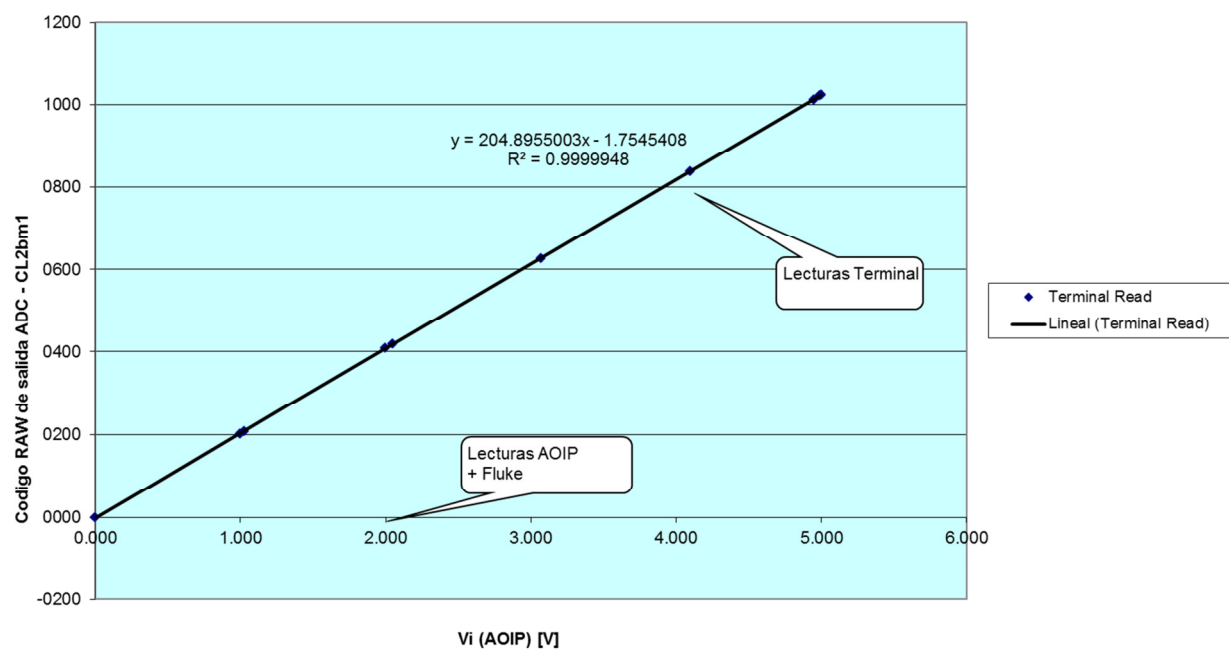
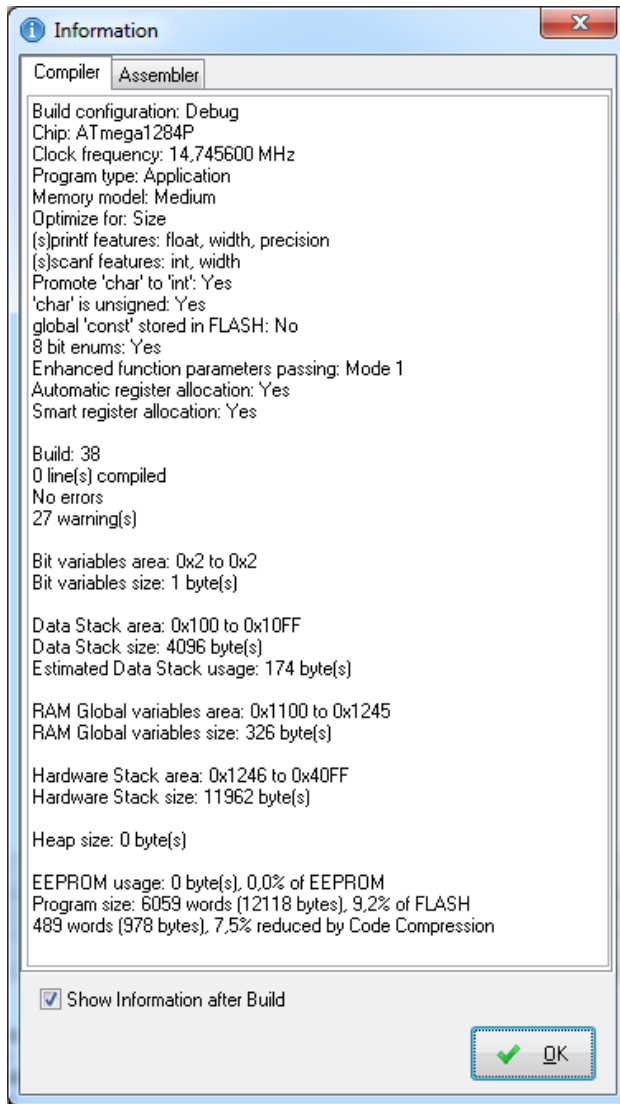


Figure T.7 – Meaurement resulting data + Linear adjustment

CONCLUSION: IF POSSIBLE: SELECT THE TIMER1_TRIGGERED VERSION..

APPENDIX 1: As usual, we use the following programming settings:



APPENDIX 2 / CURRENT ADC INITIALIZATION for CL2bm1 boards

The initialization history on CL2bm1 boards for ADC has been problematic. Probably some inherent mistake somewhere, but the Wizard-generated values were used most of the time. Data sheet clearly recommends using a clock source between 50 and 250 kHz, but the working results occurred only with 921.6kHz, no further testing was made (since important values were taken @INT1 from the other 13-bit ADC on the M4E) .

APP.2.1

This is the initialization (A) on PWRC2, v9e (2012):

```
// ADC initialization v128-1 / Leave as proposed by CV-Wiz 23.7.10
// ADC Clock frequency: 921.600 kHz
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: Free Running
// Digital input buffers on ADC0: On, ADC1: On, ADC2: On, ADC3: On
// ADC4: On, ADC5: On, ADC6: On, ADC7: On
// DIDR0=0x00;
// ADMUX=FIRST_ADC_INPUT | (ADC_VREF_TYPE & 0xff);
// ADCSRA=0xEB;
// ADCSRB=0xF8;
// ***** Replace 2-10-2010 *****
// ADC initialization - Rev 12D (slower) 2-10-2010
// ADC Clock frequency: 115.200 kHz - dont work, 460.8kHz = 0xEC in ADCSRA
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: Free Running
// Digital input buffers on ADC0: Off, ADC1: Off, ADC2: Off, ADC3: Off
// ADC4: Off, ADC5: Off, ADC6: Off, ADC7: Off
// DIDR0=0xFF;
// ADMUX=FIRST_ADC_INPUT | (ADC_VREF_TYPE & 0xff);
```

```

// ADCSRA=0xEB;      // Try Back with 921.6kHz..
// ADCSRA=0xEC;      // Try 460.8kHz.. - No va..
// ADCSRB=0xF8;

// ADC initialization - XTAL 14.76MHz 22-3-2012
// ADC Clock frequency: 921.600 kHz
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: Free Running
// Digital input buffers on ADC0: On, ADC1: On, ADC2: On, ADC3: On
// ADC4: On, ADC5: On, ADC6: On, ADC7: On
DIDR0=0x00;
ADMUX=FIRST_ADC_INPUT | (ADC_VREF_TYPE & 0xff);
ADCSRA=0xEC;      // 11101100
ADCSRB=0xF8;

```

And the initialization Definitions & ISR routine on CL2bm1 were:

```

/*****
**
** Standard Input/Output functions
** ADC - define last channel = 7
*****/
//
#include <stdio.h>

#define FIRST_ADC_INPUT 0
#define LAST_ADC_INPUT 7
unsigned int adc_data[LAST_ADC_INPUT-FIRST_ADC_INPUT+1];
// v12B - Make a Buffer for adc_data, to avoid conflicts with ISR. All calcs to
// be performed on adc_buffer[], read from adc_data with CLI/SEI..
unsigned int adc_buffer[LAST_ADC_INPUT-FIRST_ADC_INPUT+1]; // Added 28-9-2010

#define ADC_VREF_TYPE 0x00

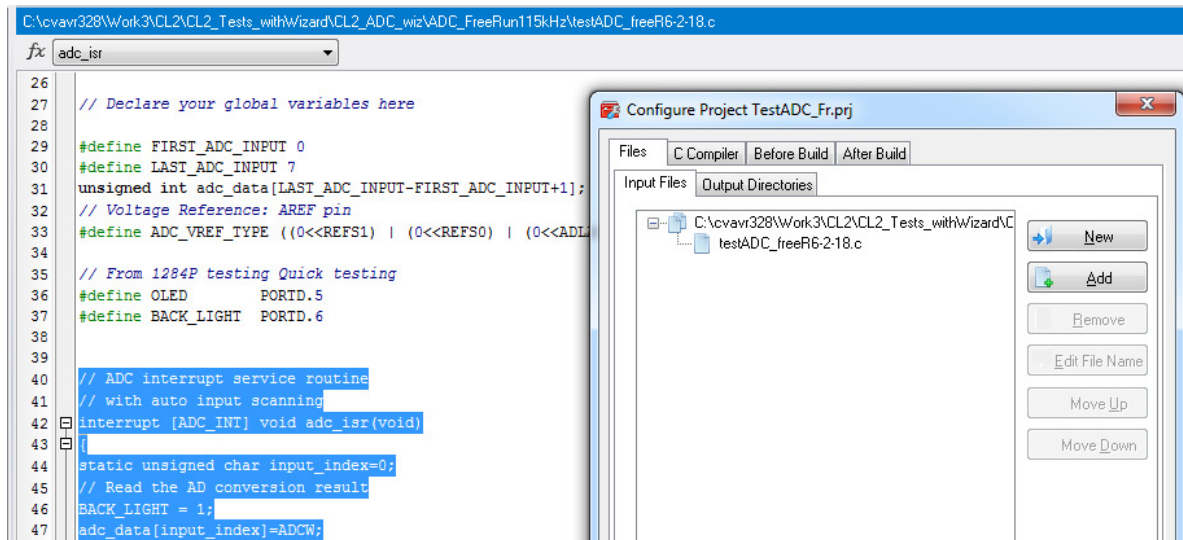
// From 1284P testing Quick testing - Added 6.2.18 for testing
#define OLED          PORTD.5
#define BACK_LIGHT    PORTD.6

/*****
**
** ADC interrupt service routine
** with auto input scanning
** Note 6.2.2018 From Data sheet- If done at 921 kHz, the conversion time
** will take 13.5 x 1.06us = 14.67 us, this is ** apparently too fast..
** (considering the ISR is inserting a 10us delay, see (A) in code..)
** At 115 kHz, the conversion time will be around 117 us. We will verify this
** Experimentally with the Wizard generated code, see testADC_freer6-2-18.c
** and we add a Toggle Pin for testing..
*****/

interrupt [ADC_INT] void adc_isr(void)
{
    static unsigned char input_index=0;
    // Read the AD conversion result
    adc_data[input_index]=ADCW;
    // Select next ADC input
    if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT))
        input_index=0;
    // 24.7.2010 ADC REF Type is 0x00= External on AREF pin, as in CL2bm1
    ADMUX=(FIRST_ADC_INPUT | (ADC_VREF_TYPE & 0xff))+input_index;
    // (A) Delay needed for the stabilization of the ADC input voltage
    delay_us(10);
    // Start the AD conversion
    // by setting only the AD Conversion start bit
    ADCSRA|=0x40; // ADSC -> 1
}

```

APP.2.2 TEST PROJECT -1:



Now 7.2.18 in testADC_freeR6-2-18.c we add the two test pins, (OLED & BACK_LIGHT) and use the first for life detection, and the other one to signal in-out of ADC ISR. The ADC is initialized with the Wizard to 115.2KHz, as shown:

```
// ADC initialization
// ADC Clock frequency: 115,200 kHz = = 14.7456e06 / 128
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: Free Running
// Digital input buffers on ADC0: Off, ADC1: Off, ADC2: Off, ADC3: Off
// ADC4: Off, ADC5: Off, ADC6: Off, ADC7: Off
DIDR0=(1<<ADC7D) | (1<<ADC6D) | (1<<ADC5D) | (1<<ADC4D) | (1<<ADC3D) | (1<<ADC2D) | (1<<ADC1D) | (1<<ADC0D);
ADMUX=FIRST_ADC_INPUT | ADC_VREF_TYPE;
ADCSRA=(1<<ADEN) | (1<<ADSC) | (1<<ADATE) | (0<<ADIF) | (1<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
// Divider is now ADPS2:ADPS0 = 111
ADCSRB=(0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0);
```

..and the ADC ISR is modified with addition of the signal pin BACK_LIGHT = PD.6, which is observed with the scope..

```
// ADC interrupt service routine
// with auto input scanning
interrupt [ADC_INT] void adc_isr(void)
{
    static unsigned char input_index=0;
    // Read the AD conversion result
    BACK_LIGHT = 1;
    adc_data[input_index]=ADCW;
    // Select next ADC input
    if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT))
        input_index=0;
    ADMUX=(FIRST_ADC_INPUT | ADC_VREF_TYPE)+input_index;
    // Delay needed for the stabilization of the ADC input voltage
    delay_us(10);
    // Start the AD conversion
    ADCSRA|=(1<<ADSC);
    BACK_LIGHT = 0;
}
```

Main is modified to show life signal on OLED..

```
// 6.2.2018 BACK_LIGHT is toggled on ADC ISR calls, starts = 0;
// OLED is life indication, toggled in main loop.
BACK_LIGHT = 0;
OLED = 0;
// Globally enable interrupts
#asm("sei")
```

```
while (1)
{
    // Place your code here
    OLED = 1;
    delay_ms(500);
    OLED = 0;
    delay_ms(500);
}
```

```
}
}
```

This produces a signal showing the ISR (pin 20 = PD.6) on the scope as shown, with ISR duration of 12.4 to 12.6 μs And a “low period” of 100.4 μs , giving a total fo 112.8 to 113 μs -> the 8 channels will be updated every 900 μs , or an update frequency of 1106 Hz (Correct, but we need to verify readings..). Note that the ON time is a bit more than 10 μs (which is the artificial delay inserted, to stabilize input) – at the end of this ON time, at OFF the conversion starts and takes about 100 μs , once completed the ISR is triggered and signal goes ON again.

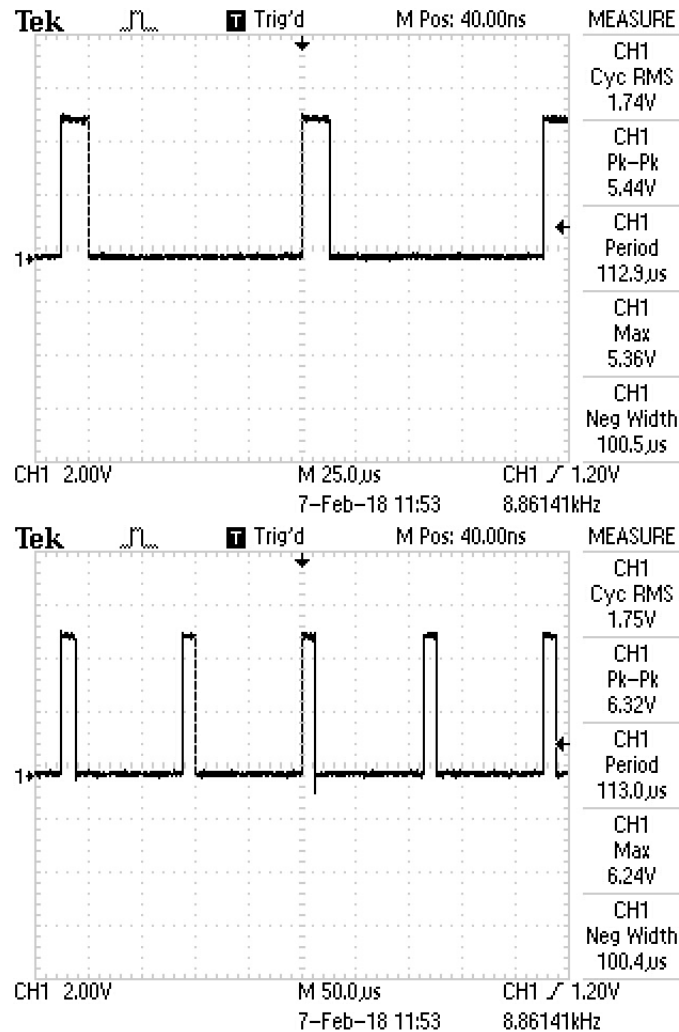


Figure App.2.1 – Timing 115.2 kHz Free Running ($T=50\mu\text{sec}$ / bottom), ADC ISR PinD.6 on ATmega1284P, CL2bm1 board

APP.2.3 TEST PROJECT -2:

If changing to original 921.6kHz settings,

```
// ADC initialization
// ADC Clock frequency: 921,600 kHz = 14.7456e06 / 16
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: Free Running
// Digital input buffers on ADC0: Off, ADC1: Off, ADC2: Off, ADC3: Off
// ADC4: Off, ADC5: Off, ADC6: Off, ADC7: Off
DIDR0=(1<<ADC7D) | (1<<ADC6D) | (1<<ADC5D) | (1<<ADC4D) | (1<<ADC3D) | (1<<ADC2D) | (1<<ADC1D) | (1<<ADC0D);
ADMUX=FIRST_ADC_INPUT | ADC_VREF_TYPE;
// Divider is now ADPS2:ADPS0 = 100
ADCSRA=(1<<ADEN) | (1<<ADSC) | (1<<ADATE) | (0<<ADIF) | (1<<ADIE) | (1<<ADPS2) | (0<<ADPS1) | (0<<ADPS0);
ADCSRB=(0<<ADTS2) | (0<<ADTS1) | (0<<ADTS0);
```

the same ISR shows overlapping and time hogging by the ISR which is probably why the detected readings were not very consistent in the first PB/2 systems.. Still, lower fCK/ADPS frequencies in those systems did not produce ADC readings..

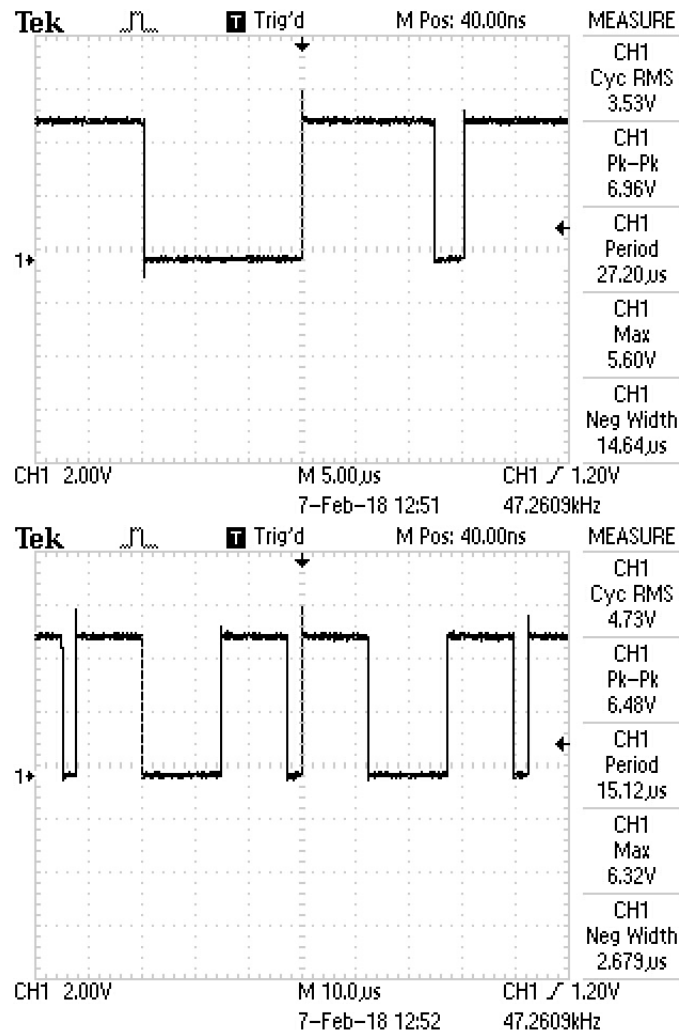


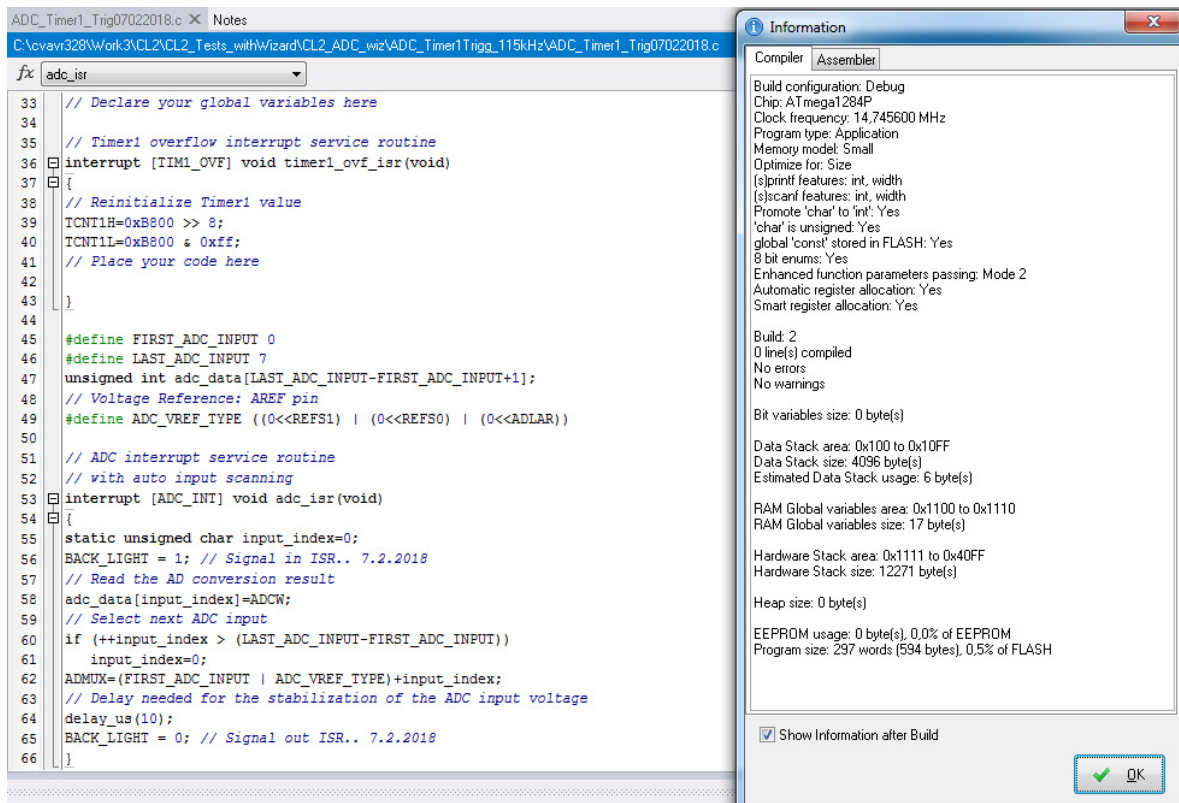
Figure App.2.2 – Timing 921.6 kHz Free Running (T=5usec top / 10us bottom, overlapping), ADC ISR PinD.6 on ATmega1284P, CL2bm1 board

APP.2.4 TEST PROJECT -3:

If we set the wizard for some of the other modes, for example Timer1 triggered conversions, we have the options of Timer0 or Timer1, but not 2 or 3, as seen in 2.4 – ADCSRB listing (table follows):

ADTS[2:0]	Trigger Source
000	Free Running mode
001	Analog Comparator
010	External Interrupt Request 0
011	Timer/Counter0 Compare Match A
100	Timer/Counter0 Overflow
101	Timer/Counter1 Compare Match B
110	Timer/Counter1 Overflow
111	Timer/Counter1 Capture Event

Since Timer0 will be used by UART3, and Timer1 is in OVF at 100Hz for DiskProc() if ElmChanFFS is used, then it might be possible to use Timer1OVF as Trigger. The code generated by the wizard and the compilation is shown below (note that we added BACK_LIGHT toggling to signal in/out of ADC ISR).



NOTE: Remember to set PD.5,PD.6 as outputs (if not, as we did, the outputs appear but only a weak 0-2V or so pulse, not a clear on-off of OLED or BACK_LIGHT)

The resulting wave capture is shown below (ADC still takes :

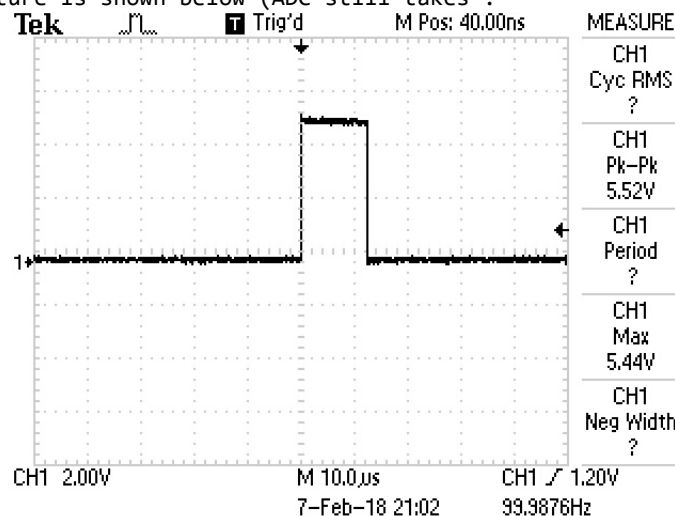


Figure App.2.3 – Timer1 Triggered at 100Hz OVF (T=10usec), ADC ISR PinD.6 on ATmega1284P, CL2bm1 board

Note that, as before, the ON time is a little more than 10 μs (which is the artificial delay inserted, to stabilize input) – at the end of this ON time, at OFF the conversion starts and will probably take about 100 μs (no signal here), once completed we have to wait until the Timer1 OVF (100 ms later..) when the ISR is triggered and signal goes ON again. This can be seen better in the new scope output at longer (2.5msec) horizontal setting (Figure App.2.4). The resulting signal period is 100Hz, as required by FFS-ELMChan. Updating all 8 channels will require $8 \times 10 \text{ ms} = 80 \text{ ms}$, or an updating frequency of 12.5 Hz. This may or may not be sufficient, but it certainly leaves plenty of application time for other CPU tasks.

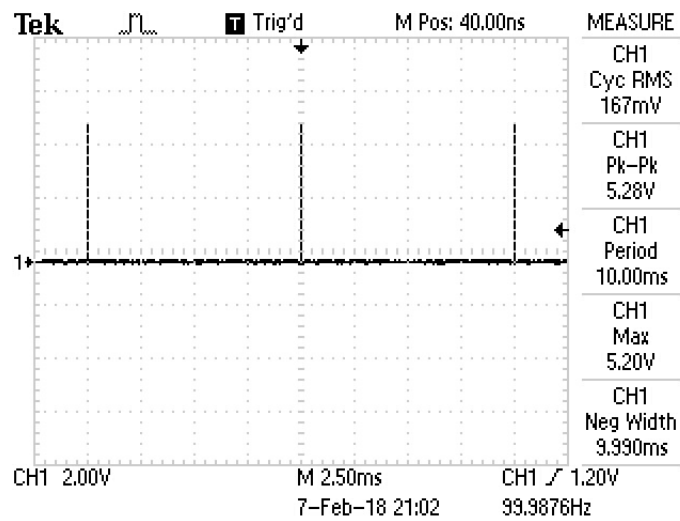


Figure App.2.4 – Timer1 Triggered at 100Hz OVF (Tsc=2.5msec), ADC ISR PinD.6 on ATmega1284P, CL2bm1 board