Bulgarian Academy of Sciences. Space Research and Technology Institute.
Aerospace Research in Bulgaria. 30, 2018, Sofia

DOI: https://doi.org/10.3897/arb.v30.e14

DATA LOGGING BY AD7656 ANALOG TO DIGITAL CONVERTER

Konstantin Metodiev

Space Research and Technology Institute – Bulgarian Academy of Sciences
e-mail: komet@space.bas.bg

Abstract

In the paper hereby is presented, an exemplary algorithm of data logging by means of Analog Devices' AD7656 analog to digital converter. Converted data are transmitted further to ATmega644P microcontroller unit through Serial Peripheral Interface. The microcontroller, in turn, sends data to a PC through FTDI's FT232 UART to USB chip.

What motivates the current study is a source code development for the microcontroller unit. The source code is available for download at the quoted link. The main technique used in the code is external interrupt. The development environment used is MikroC Pro for AVR.

The presented study is a part of software developed for space project "Resonance".

1. Introduction

Analog Devices' AD7656 is six channel, bipolar, 16-bit analog to digital converter capable of simultaneous parallel sampling at rate up to 250 kSPS. It can accept bipolar input signals and handle input frequencies up to 12 MHz. The AD7656 has a high speed parallel and serial interface that let the device connect with microprocessors or DSPs. The AD7656 also accepts true bipolar input signals within $\pm 4 \times V_{REF}$ range and $\pm 2 \times V_{REF}$ range and maintains an on-chip 2.5 V reference voltage, [1].

The AD7656 has been used by the project "Resonance" research team as a circuit element included in the control and measuring equipment. As a whole, readily available software for AD7656 application is difficult to encounter which is why both presented study and source code in the appendix section may serve to fill up the gap.

2. Materials and methods

The experimental setup is depicted in Fig. 1. The AD7656 is powered by unipolar voltage of +5 V DC, so is the microcontroller unit. In addition, pin selectable bipolar voltage range of ±12 V DC is provided by AIMTEC's AM3D-0515DH30Z at 100 mW DC/DC converter and 78L12, 79L12 voltage regulators.

A 20 MHz external crystal oscillator clocks the MCU. Auxiliary equipment used is USBASP v.2.0, [2], in-system serial programmer as well as AVRDUDE utility, [3], to write to flash memory of the AVR chip.

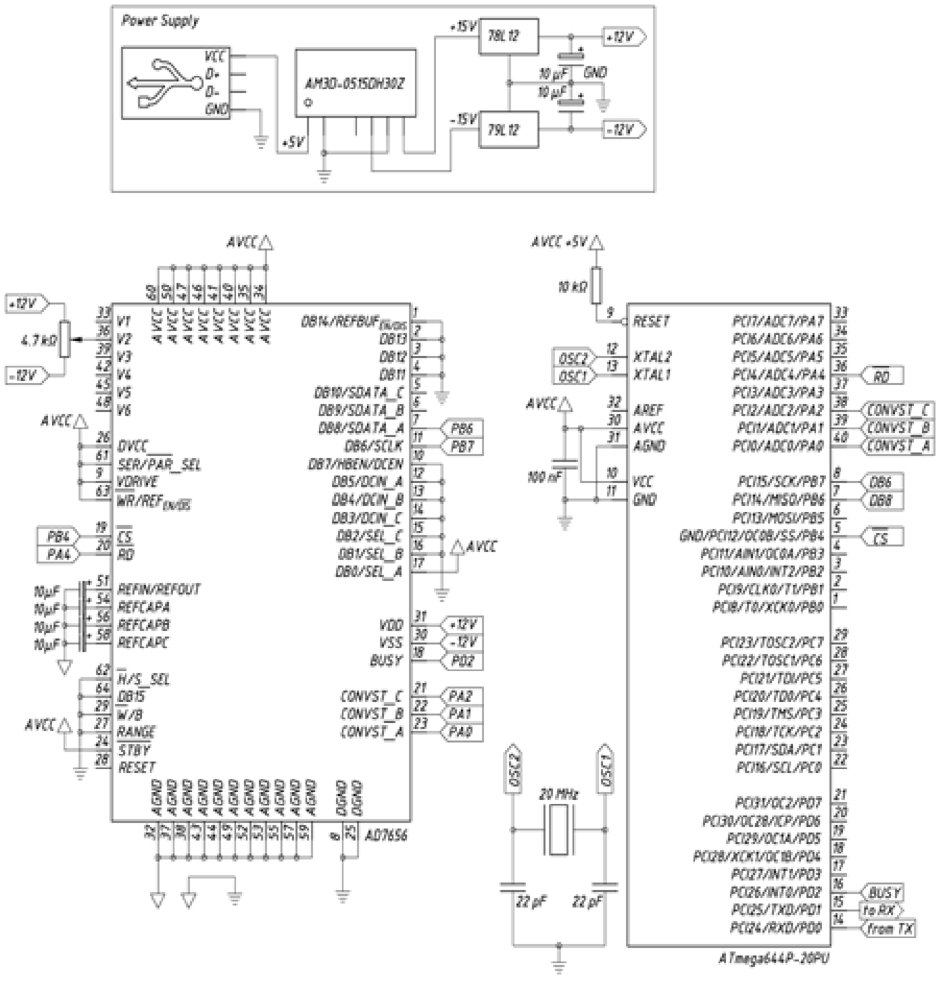


Fig. 1. AD7656 and ATmega644P wiring diagram, serial hardware mode, single output

During the initialization step, the MCU pins PA0, PA1, and PA2 are set to output. They are connected to CONVST_A, _B, _C pins on the AD7656 chip respectively. The CONVST pins are used to start conversions on the channel pairs. Initially, they are set to 1, i.e. idle state HIGH. Having set as input, the MCU pin

PA3/INT0 is connected to BUSY pin on the ADC. The latter returns state LOW whenever conversion is complete; hence, user is allowed to read data from the ADC, [4]. The Interrupt Sense Control bits ISC01 and ISC00 of the External Interrupt Control Register EICRA are set to 1 and 0 respectively, so that the falling edge of INT0 pin generates asynchronously an interrupt request, [5]. The external interrupt request is enabled by setting INT0 bit of the External Interrupt Mask Register EIMSK. Finally, global interrupt is enabled by setting I bit of the Status Register SREG.

In order to get AD7656 to start sampling, a one μs wide negative logic rectangular pulse, must be sent to CONVST pins. During the conversion process, BUSY pin is set to HIGH automatically which prevents the user from accessing the ADC data registers. It takes the ADC about three μs to succeed in converting voltages at all six channels. The conversion completes as soon as the BUSY pin goes back to state LOW. Thus, the falling edge arriving at INT0 pin, Fig. 2, triggers external interrupt request and an Interrupt Service Routine (ISR) starts. Essentially, the ISR reads converted data through SPI interface at frequency Fosc/128 ≈ 160 kHz. Finally, a data processing function sends text strings further via UART to the PC.

Data reading process, performed by the ISR, is implemented by means of SPI interface as follows. The CS (chip select) and the RD (ready) pins are set to state LOW to address ADC. The MCU exchanges data with the ADC like shift register. For this reason, SPI Data Register SPDR has to be written to initiate data transmission. After transmission completion, SPIF bit is set to HIGH by the hardware. The result is brought from the ADC and stored back in the SPDR register. This sequence is repeated twice to obtain data two bytes long (function "write2bytes" in the Appendix). Data transfer initiates by simply writing to the SPDR register twice in a row. It is not necessary to connect the MOSI line. Finally, both CS and RD pins are restored to state HIGH to deselect the ADC.

The reader is advised to examine closely the included source code at the Appendix section so as to apprehend the underlying idea.

3. Results

An exemplary scanning of all six channels by a logic analyzer is depicted in Fig. 2. In addition to generic SPI channels – MISO, CLK, and CS, the ADC pins BUSY, CONVSTA, and RD are also examined. The start pulse and the falling edge triggering an external interrupt are visible in the zoomed window.

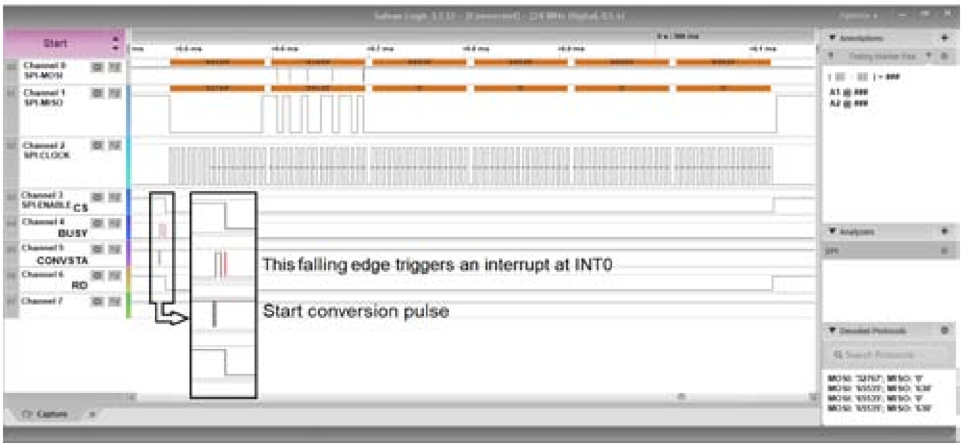
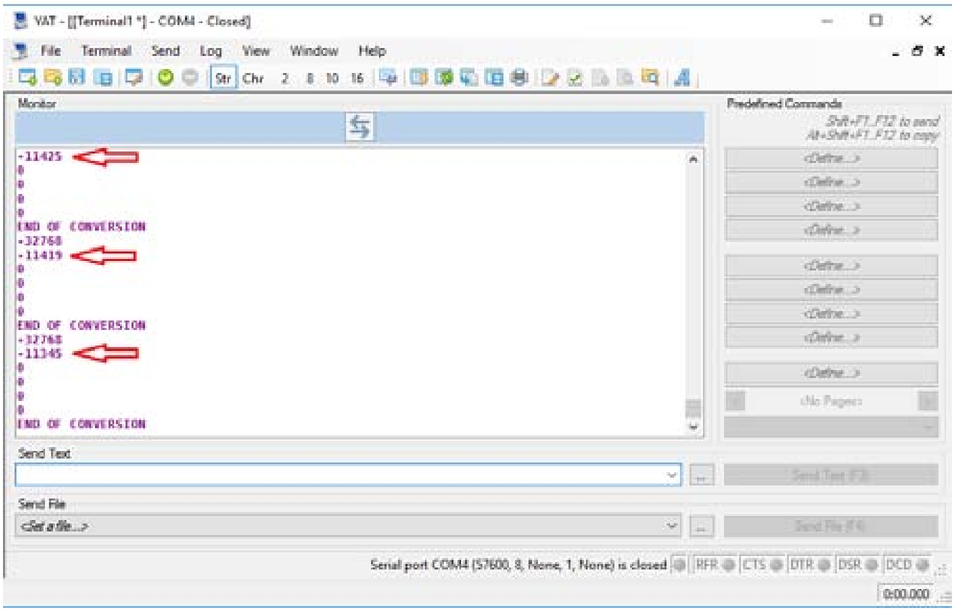


Fig. 2. Reading all six channels (two bytes each) of AD7656 via SPI, Saleae Logic session

In Fig. 3, an exemplary terminal session is shown. The input voltage feeds channel V2. The converted value, denoted by a red arrow, varies within –32 768 to +32 767 quantization levels, so does the input voltage, i.e. from –12 V to +12 V DC. In this study case, the first two input channels solely produce meaningful results according to following states of pins SEL_A = HIGH, SEL_B = LOW, SEL_C = LOW, SER/PAR = HIGH, H/S = LOW, Fig. 1 [4].



```
-11425
0
0
0
0
END OF CONVERSION
-32768
-11419
0
0
0
0
END OF CONVERSION
-32768
-11345
0
0
0
0
END OF CONVERSION
```

Serial port COM4 (57600, 8, None, 1, None) is closed

Fig. 3. Yet Another Terminal session, SDATA_A output channel

4. Discussion

The external interrupt technique used in the current study proved to be versatile enough. It lets user read data as soon as the conversion completes. The advantage of the presented interrupt technique over the so-called polling technique is evident. In the latter ("polling") case, the user is compelled to scan the MISO pin on a regular basis whilst, in the former ("interrupt") case, data are transferred to the MCU automatically whenever the conversion is done.

In the Appendix section, reader may find source code, which is uploaded in the MCU. The development environment is Mikroelektronika's MikroC Pro for AVR, v.7.0.1 [6]. The presented source code might be downloaded at link [7].

A 3D model of sensing element and data logger used in project "Resonance" is shown in Fig. 4. The model is created in Autodesk Inventor IDE v.2018, [8]. The AD7656 used in the current study is an essential part of the corresponding control and measuring equipment.

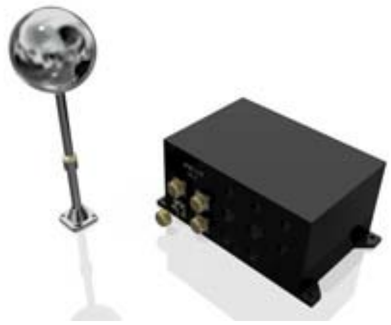


Fig. 4. Electric potential measurement equipment, project "Resonance"

According to experience, it is very rare for the AD7656 to go silent. The reason this happens is that the BUSY pulse does not occur. Should it happen, it is necessary to reset the AD7656 (pin 28 to GND, Fig. 1) to restore its functionality.

Further project development might be using multiple ADCs configured in daisy-chain mode as it is described in [4].

References

1. AD7656/AD7657/AD7658 datasheet, Analog Devices
2. https://www.fischl.de/usbasp/
3. https://www.nongnu.org/avrdude/
4. Croke, C., Configuring the AD7656/AD7657/AD7658 for Serial and Daisy-Chain Interface Modes of Operation, Application Note AN-893, Analog Devices
5. ATmega644P/V, datasheet complete, Atmel
6. https://www.mikroe.com/mikroc-avr
7. https://libstock.mikroe.com/projects/view/2015/atmega644p-ad7656-in-serial-mode
8. https://www.autodesk.com/products/inventor/overview

Appendix 1. Source code.

```
typedef unsigned short uint8_t;
typedef unsigned int uint16_t;
typedef signed int int16_t;
sbit Chip_Select at PORTB4_bit;
sbit Chip_Select_Direction at DDB4_bit;
int16_t VBs[] = {0, 0, 0, 0, 0, 0};

int16_t write2bytes(uint16_t command) {
uint8_t higherByte, lowerByte;
int16_t result;

higherByte = (uint8_t)((command &
              0xFF00) >> 8);
lowerByte = (uint8_t)(command & 0x00FF);
SPDR = higherByte; // sending high byte
while(!(SPSR & (1<<SPIF))); // wait
result = SPDR & 0xFF;
result <<= 8;
SPDR = lowerByte;  // sending low byte
while(!(SPSR & (1<<SPIF))); // wait
result |= SPDR;

return result;
}//write2bytes

uint8_t SPI_init(void) {

DDB6_bit = 0; // Set PB6 pin as input MISO
DDB5_bit = 1; // Set PB5 pin as output MOSI
Chip_Select = 1; // Deselect ADC
Chip_Select_Direction = 1; // Set CS# pin
SPI1_Init_Advanced(_SPI_MASTER,
     _SPI_FCY_DIV128, _SPI_CLK_LO_LEADING);

return 0;
}//SPI_init

uint8_t sendThroughUART(char *text_) {

UART1_Write_Text(text_);
UART1_Write(13); UART1_Write(10);

return 0;
}//sendThroughUART

uint8_t startSampling(void) {

PORTA.F0 = 0; // CONVSTA = LOW
// PORTA.F1 = 0; // CONVSTB = LOW
// PORTA.F2 = 0; // CONVSTC = LOW
Delay_us(1);
PORTA.F0 = 1; // CONVSTA = HIGH
// PORTA.F1 = 1; // CONVSTB = HIGH
// PORTA.F2 = 1; // CONVSTC = HIGH

return 0;
}//startSampling

uint8_t readADC(void) {
uint8_t j;

Chip_Select = 0; // Select chip
PORTA.F4 = 0; // Select RD pin
for (j = 0; j < 6; j++)
   VBs[j] = write2bytes(0x0000);
PORTA.F4 = 1; // Deselect RD pin
Chip_Select = 1; // Deselect chip

return 0;
}//readADC

uint8_t processData(void) {
uint16_t result = 0;
uint8_t j;
char buffer[16];

for (j = 0; j < 6; j++) {
   result = VBs[j];
   sprintf(buffer, "%d", result);
   sendThroughUART(buffer);
}//for_j
UART1_Write_Text("End of conversion");
UART1_Write(13); UART1_Write(10);

return 0;
}//processData

void interrupt_ISR () org IVT_ADDR_INT0 {

SREG_I_bit = 0; // Disable interrupts
readADC();
SREG_I_bit = 1; // Enable interrupts

return;
}//interrupt_ISR

uint8_t init(void) {

DDA0_bit = 1; // PA0 as output (CONVSTA)
DDA1_bit = 1; // PA1 as output (CONVSTB)
DDA2_bit = 1; // PA2 as output (CONVSTC)
// DDA3_bit = 0; // PA3 as input (BUSY)
DDA4_bit = 1; // Set PA4 as output (RD)
SPI_init();
UART1_Init(57600);
Delay_ms(100);
PORTA.F0 = 1; // CONVSTA = idle HIGH
// PORTA.F1 = 1; // CONVSTB = idle HIGH
// PORTA.F2 = 1; // CONVSTC = idle HIGH
EICRA = 0b00000010; // Falling edge of INT0
EIMSK = 0b00000001; // External interrupt
SREG_I_bit = 1; // Enable interrupts

return 0;
}//init

void main() {

     init();
     while (1) {
        startSampling();
        processData();
        Delay_ms(100);
     }//while

return;
}//main
```

СЪБИРАНЕ НА ДАННИ ПОСРЕДСТВОМ АНАЛОГО-ЦИФРОВ ПРЕОБРАЗУВАТЕЛ AD7656

К. Методиев

Резюме

В настоящата статия е представен примерен алгоритъм за събиране на данни с помощта на аналого-цифров преобразувател AD7656. Преобразуваните данни се предават към микроконтролер ATmega644P по сериен интерфейс SPI. Микроконтролерът, на свой ред, предава данните към компютър по USB чрез преобразувател FT232.

Това, което мотивира настоящето изследване, е създаването на програмно обезпечение за микроконтролера. Изходният код може да бъде изтеглен от цитираната препратка. Основната програмна техника, използвана в кода, е външно прекъсване. Развойната среда е MikroC Pro for AVR.

Представеното изследване е част от програмното обезпечение, разработвано за космически проект „Резонанс“.