ADConverter
|
ADConverter: an open project for analog to digital convertion

Robin Siemiatkowski

21st January 2014

Contents

Chapter 1

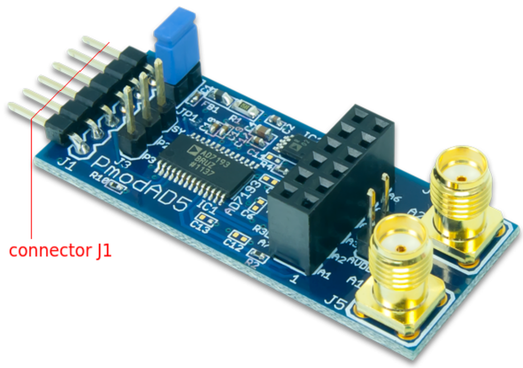
Presentation

ADConverter project is based on PmodAD5,This is a high resolution analog-to-digital converter built around the Analog Devices AD7193 Sigma-Delta ADC. PmodAD5 can be used with Arduino devices or with chipkit devices that are fitted with SPI communication (that project was develloped with Cerebot Mx3CK(chipkit), Arduino UNO/MEGA/DUE).

1.1 Coponents

1.1.1 PmodAD5

The PmodAD5 has ten analog inputs that correspond to eight data lines. The two SMA female connectors route to inputs one and two, while eight standard female header pins route to inputs one through eight. Customers can set the PmodAD5 into single or continuous conversion mode. The PmodAD5 powers up by default in continuous conversion mode. You can set the mode to start a conversion by either writing to the appropriate registers or on the rising edge of SYNC. PmodAD5 is composed of AD7193 microcontroller.



Spi connection on PmoAd5 :

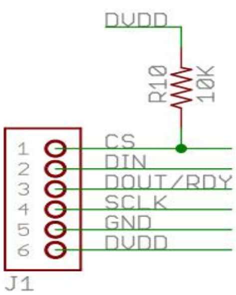


Figure 1.1:
Note:The voltage applied to DVDD must be kept between 3.0V and 5.25V in order to avoid damaging the parts used in this circuit

1.1.2 AD7193

The AD7193 is a low noise, complete analog front end for high precision measurement applications. It contains a low noise, 24-bit sigma-delta (Îč-ÎŤ) analog-to-digital converter (ADC). The on-chip low noise gain stage means that signals of small amplitude can interface directly to the ADC.
The device can be configured to have four differential inputs or eight pseudo differential inputs. The on-chip channel sequencer allows several channels to be enabled simultaneously, and the AD7193 sequentially converts on each enabled channel, simplifying communication with the part. The on-chip 4.92 MHz clock can be used as the clock source to the ADC or, alternatively, an external clock or crystal can be used. The output data rate from the part can be varied from 4.7 Hz to 4.8 kHz. The device has a very flexible digital filter, including a fast settling option. Variables such as output data rate and settling time are dependent on the option selected. The AD7193 also includes a zero latency option. The part operates with a power supply from 3 V to 5.25 V. It consumes a current of 4.65 mA, and it is available in a 28-lead TSSOP package and a 32-lead LFCSP package.

Applications

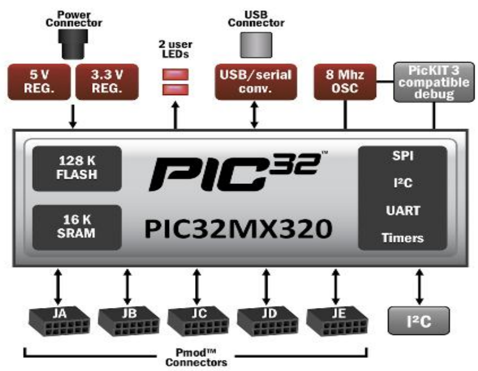
- PLC/DCS analog input modules
- Data acquisition
- Strain gage transducers
- Pressure measurement
- Temperature measurement
- Flow measurement
- Weigh scales
- Chromatography
- Medical and scientific instrumentation

1.2 Plateforms

The goal of this project was to allow the use of the PmodAD5 on several platforms. For this reason, we tried to develop this project on several micro-controller :

- chipkit Cerebot Mx3ck
- Arduino uno / mega
- arduino DUE

1.2.1 Cerebot Mx3ck



Presentation of Cerebot MX3ck Specifications

- Microcontroller: PIC32MX320F128H
- Flash Memory: 128K
- RAM Memory: 16K
- Operating Voltage: 3.3V
- Max Operating Frequency: 80Mhz
- Typical operating current: 75mA
- Input Voltage (recommended): 7V to 15V
- Input Voltage (maximum): 20V
- I/O Pins: 42 total
- Analog Inputs: 12
- Analog input voltage range: 0V to 3.3V
- DC Current per pin: +/-18mA

This section presents the steps for developing a chipKIT application that will run on the Digilent Cerebot MX3cK development board for controlling and monitoring the operation of the ADI part. To develop our application, we chose to use MPIDE. This is a modifed IDE of Arduino IDE. The reason is that the development of the program with MPIDE (with the library SPI of Arduino and not that of chipkit, the DSPI) is going to bring us to a compatible program with arduino (the most obvious case to have a compatible and simple program), we thus had to install Mpide.

How to install MPIDE

- Open a Web browser and navigate to the github for MPIDE
  Presently held at: `https://github.com/chipKIT32/chipKIT32-MAX/downloads`
- download mpide-0023-linux-20120903.tgz
- extract the file on a folder
- Open a terminal, navigate to the location of the extract file
- lauch MPIDE with ./mpide command

To be able to compile a program on this card, we must reprogram the bootloader.

To change the boot loader, you must install MplabX:

- Open a Web browser and navigate to `https://www.microchip.com/pagehandler/en-us/family/mplabx/`
- download MPLAB X IDE v1.95
- open a terminal, navigate to the location of the file downloaded.
- execute this command : chmod +x MPLABX-v1.95-linux-installer.run
- if you are in 64bit, it will be impossible to execute the program of installation of MPLAB. It is necessary to install drivers 32 bits: sudo apt-get install ia32-libs
- Now,you can lauch the installation: sudo ./MPLABX-v1.95-linux-installer.run

How to change Bootloader

/!\ you need to have a Chipkit Programmer 

To reprogram the boot loader using MPLAB, perform the following steps:

- download Bootloader folder on Github(RobinSI/ADConverter)
- launch MPLAB IPE
- in "Slect Device and Tool", select Family: 32-bit MCUs (PIC32) and Device: PIC32MX320F128H.
- select the source (browse chipKIT_Bootloader_MX3ck.hex)
- Apply and connect.
- Click on Program

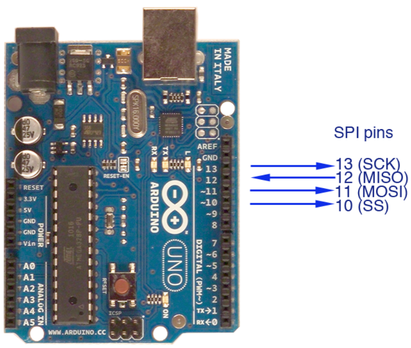
1.2.2 Arduino UNO/MEGA

To developp our application on an Arduino UNO or an Arduino mega, we use the classic IDE of Arduino. But you must have been changing connection to SPI.

Arduino UNO

Arduino UNO presentation

- Microcontroller ATmega328
- Operating Voltage 5V
- Input Voltage (recommended) 7-12V
- Input Voltage (limits) 6-20V
- Digital I/O Pins 14 (of which 6 provide PWM output)
- Analog Input Pins 6
- vDC Current per I/O Pin 40 mA
- DC Current for 3.3V Pin 50 mA
- Flash Memory 32 KB (ATmega328) of which 0.5 KB used by bootloader
- SRAM 2 KB (ATmega328)
- EEPROM 1 KB (ATmega328)
- Clock Speed 16 MHz



SPI Connection on Arduino UNO:

- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).

Arduino MEGA

Arduino MEGA presentation

- Microcontroller ATmega1280
- Operating Voltage 5V
- Input Voltage (recommended) 7-12V
- Input Voltage (limits) 6-20V
- vDigital I/O Pins 54 (of which 15 provide PWM output)
- Analog Input Pins 16
- DC Current per I/O Pin 40 mA
- DC Current for 3.3V Pin 50 mA
- Flash Memory 128 KB of which 4 KB used by bootloader
- SRAM 8 KB
- EEPROM 4 KB
- Clock Speed 16 MHz

SPI Connection on Arduino MEGA:

- SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).

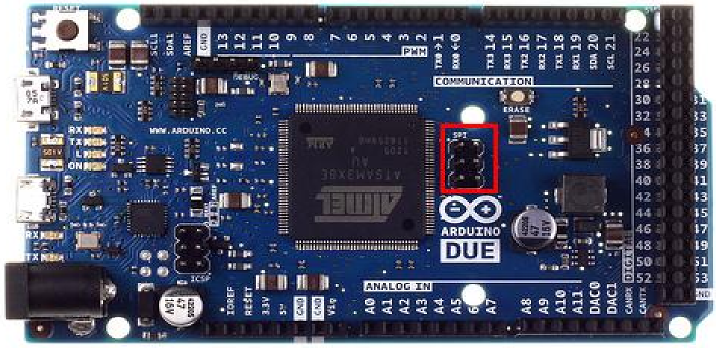
1.2.3 Arduino DUE

Arduino DUE presentation

- Microcontroller AT91SAM3X8E
- Operating Voltage 3.3V
- Input Voltage (recommended) 7-12V
- Input Voltage (limits) 6-16V
- Digital I/O Pins 54 (of which 12 provide PWM output)
- Analog Input Pins 12

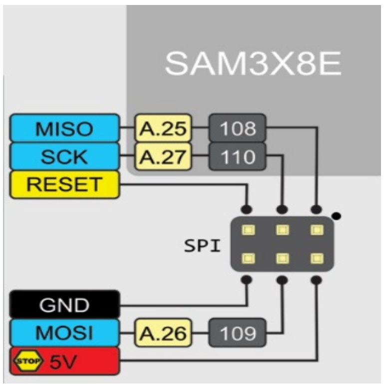
- Analog Outputs Pins 2 (DAC)
- Total DC Output Current on all I/O lines 130 mA
- DC Current for 3.3V Pin 800 mA
- DC Current for 5V Pin 800 mA
- Flash Memory 512 KB all available for the user applications
- SRAM 96 KB (two banks: 64KB and 32KB)
- Clock Speed 84 MHz

/!\ For using Arduino DUE you need to install Arduino IDE-1.5.4



SPI Connection on Arduino DUE:

- The extended API can use pins 4, 10, and 52 for CS.
- SPI: SPI header (ICSP header on other Arduino boards):



Chapter 2

Basic Program

2.1 AD7193

At the begining, we recovered the generic driver of AD7193, and after debugging, just three functions had to be implemented:

- SPI_Init
- SPI_Read
- SPI_Write

But those three functions should have been implemented for each micro-controller that we would to use, (each micro-controller have separate register). that is why we decided to replace those three functions by functions of the SPI library.

2.1.1 SPI Library

- **SPI.begin()** : Initializes the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.

- **SPI.transfer(val)** : Transfers one byte over the SPI bus, both sending and receiving.(val:the byte to send out over the bus)

- **SPI.setBitOrder(order)** : Sets the order of the bits shifted out of and into the SPI bus, either LSBFIRST (least-significant bit first) or MSBFIRST (most-significant bit first).

- **SPI.setClockDivider(divider)** : Sets the SPI clock divider relative to the system clock. On AVR based boards, the dividers available are 2, 4, 8, 16, 32, 64 or 128. The default setting is SPI_CLOCK_DIV4, which sets the SPI clock to one-quarter the frequency of the system clock (4 Mhz for the boards at 16 MHz).

  <u>divider:</u>

  - SPI_CLOCK_DIV2
  - SPI_CLOCK_DIV4
  - SPI_CLOCK_DIV8
  - SPI_CLOCK_DIV16
  - SPI_CLOCK_DIV32
  - SPI_CLOCK_DIV64
  - SPI_CLOCK_DIV128

- **SPI.setDataMode(mode)** : Sets the SPI data mode: that is, clock polarity and phase.
  mode:
  - SPI_MODE0
  - SPI_MODE1
  - SPI_MODE2
  - SPI_MODE3

2.1.2 final AD7193 library

On the generic driver, four functions need to communicate through the SPI connection :

- AD7193_Init()
- AD7193_GetRegisterValue()
- AD7193_Reset()
- AD7193_SetRegisterValue()

for example, AD7193_SetRegisterValue() function for Cerebot Mx3ck :

```
void AD7193_SetRegisterValue(unsigned char registerAddress,
                             unsigned long registerValue,
                             unsigned char bytesNumber,
                             unsigned char modifyCS)
{//setregistervalue
    unsigned char writeCommand[5] = {0, 0, 0, 0, 0};
    unsigned char* dataPointer    = (unsigned char*)&registerValue;
    unsigned char bytesNr         = bytesNumber;

    writeCommand[0] = AD7193_COMM_WRITE |
                      AD7193_COMM_ADDR(registerAddress);

    while(bytesNr > 0)
    {
        writeCommand[bytesNr] = *dataPointer;
        dataPointer ++;
        bytesNr --;
    }
    SPI_Write(AD7193_SLAVE_ID * modifyCS, writeCommand, bytesNumber + 1);
    // write data to SPI
}//setregistervalue}
```

This code has been modified using the SPI library.

code for all devices :

```
void AD7193_SetRegisterValue(unsigned char registerAddress,
                              unsigned long registerValue,
                              unsigned char bytesNumber,
                              unsigned char modifyCS)
{//setregistervalue
    unsigned char commandByte = 0;
    unsigned char txBuffer[4] = {0, 0, 0 ,0};

    commandByte = AD7193_COMM_WRITE | AD7193_COMM_ADDR(registerAddress);
        txBuffer[0] = (registerValue >> 0)  & 0x000000FF;
        txBuffer[1] = (registerValue >> 8)  & 0x000000FF;
        txBuffer[2] = (registerValue >> 16) & 0x000000FF;
        txBuffer[3] = (registerValue >> 24) & 0x000000FF;
        if(modifyCS == 1)
        {
                digitalWrite(PMOD1_CS_PIN, LOW);
        }
        SPI.transfer(commandByte);
    while(bytesNumber > 0)
    {
```

```
        SPI.transfer(txBuffer[bytesNumber - 1]);
        bytesNumber--;
    }
        if(modifyCS == 1)
        {
                digitalWrite(PMOD1_CS_PIN, HIGH);
        }
}//setregistervalue
```

The other functions are implemented on sketches available here :

`https://github.com/RobinSi/ADConverter`

2.1.3 AD7193 library for Arduino DUE

The Arduino Due's SPI interface works differently than any other Arduino boards. On the Arduino Due, the SAM3X has advanced SPI capabilities. It is possible to use these extended methods, or the AVR-based ones.
The extended API can use pins 4, 10, and 52 for CS.

/!\ You must specify each pin you wish to use as CS for the SPI devices.

SPI library	**SPI library for Arduino DUE**
SPI.begin()	SPI.begin(slaveSelectPin)
SPI.end()	SPI.end(slaveSelectPin)
SPI.setClockDivider(divider)	SPI.setClockDivider(slaveSelectPin, divider)
SPI.setDataMode(mode)	SPI.setDataMode(slaveSelectPin, mode)
SPI.setBitOrder(order)	SPI.setBitOrder(slaveSelectPin, order)
SPI.transfer(val)	SPI.transfer(slaveSelectPin, val)

Chapter 3

How to use PmodAD5

3.1 AD7193 functions

The AD7193 library provides several functions for using the PmodAD5.

Resets the device. `AD7193_Reset(void)`
Checks if the AD7139 part is present `AD7193_Init(void)`
Selects the channel to be enabled. `AD7193_ChannelSelect(unsigned short channel)`
Performs the given calibration to the specified channel. `AD7193_Calibrate(unsigned char mode, unsigned char channel)`
Selects the polarity of the conversion and the ADC input range. `AD7193_RangeSetup(unsigned char polarity, unsigned char range)`
Returns the average of several conversion results. `AD7193_ContinuousReadAvg(unsigned char sampleNumber)`
Converts 24-bit raw data to volts. `AD7193_ConvertToVolts(unsigned long rawData, float vRef)`
Read data from temperature sensor and converts it to Celsius degrees. `AD7193_TemperatureRead(void)`
Returns the result of a single conversion. `AD7193_SingleConversion(void)`
Waits for RDY pin to go low. `AD7193_WaitRdyGoLow(void)`
Set device to idle or power-down. `AD7193_SetPower(unsigned char pwrMode)`
Selects the AD7193's operating mode. `AD7193_SetMode(unsigned char mode)`

Reads the value of a register.
`AD7193_GetRegisterValue(unsigned char registerAddress,`
`unsigned char bytesNumber,`
`unsigned char modifyCS)`

Writes data into a register
`AD7193_SetRegisterValue(unsigned char registerAddress,`
`unsigned long registerValue,`
`unsigned char bytesNumber,`
`unsigned char modifyCS)`

3.2 Sample application

The following code allows to measure a voltage (0 to 3.3V) on channel 1.

```
void setup()
{//setup
    unsigned long regValue = 0;
    Serial.begin(9600);
```

```
   delay(7000);
  if(AD7193_Init())
  {
      Serial.print("AD7193 OK");
  }
  else
  {
      Serial.print("AD7193 Error");
  }
    /*! Resets the device. */
    AD7193_Reset();
    Serial.print("\n");
    Serial.print("reset OK");
    Serial.print("\n");
    /*! Select Channel 0 */
    AD7193_ChannelSelect(AD7193_CH_0);
    Serial.print("chanel OK");
    Serial.print("\n");
    /*! Calibrates channel 0. */
    AD7193_Calibrate(AD7193_MODE_CAL_INT_ZERO, AD7193_CH_0);
    Serial.print("calibrate OK");
    Serial.print("\n");
    /*! Selects unipolar operation and ADC's input range to +-2.5V. */
    AD7193_RangeSetup(0, AD7193_CONF_GAIN_1);
    Serial.print("range OK");
    Serial.print("\n");
    /*Set the pseudo bit in configuration register */
    regValue = AD7193_GetRegisterValue(AD7193_REG_CONF, 3, 1);
    regValue |= AD7193_CONF_PSEUDO;
    AD7193_SetRegisterValue(AD7193_REG_CONF, regValue, 3, 1);

}//setup
void loop()
{
  unsigned long data = 0;
  /*retur an average of 1000 convertion on selected channel*/
  data = AD7193_ContinuousReadAvg(1000);
  /*convert binari data to volt*/
  float volt=AD7193_ConvertToVolts(data,3.3);
  // wait a while
  delay(1000);
  // print the voltage of selected channel
  Serial.print("volt=");
  Serial.println(volt,DEC);

}
```