MPWG ’Energy Meter’ Testbed
(ADC PROJECT)

INTRODUCTION

Following suggestions in a recent MPWG to investigate a digital method to linearise the output from LHC MB DCCT signals, the following work has been done.

The basic idea requires a free running ADC to digitise the DCCT output and with minimal delay, output a corrected/linearised value corresponding to the LHC energy. To achieve this, some hardware to control the ADC reading process, followed by a method to rapidly derive the corrected value is needed. The fastest correction method would be to use a look-up table containing a non-linear function derived, say from magnet test-bench measurements.

We proposed to explore the use of a modern 8-bit RISC microcontroller and an independent 16bit ADC to measure the DCCT output voltage (+/-10v) and to convert this into a ‘corrected’ value derived from an internal look-up table stored in the microcontroller Flash memory. This would give the cheapest and most reliable product, while being fully programable for individual correction tables

HARDWARE DESCRIPTION

The microcontroller chosen was the Atmel Atmega103 device. Atmel's AVR microcontrollers have a RISC core running single cycle instructions and a well-defined I/O structure that limits the need for external components. Internal oscillators, timers, UART, SPI, pull-up resistors, pulse width modulation, ADC, analog comparator and watch dog timers are some of the features in AVR devices. AVR instructions are tuned to decrease the size of the program whether the code is written in C or Assembly.

The Atmega103 has internally, 128 kbytes programmable Flash, 4 kbytes SRAM and 4 kbytes of EEPROM. The processor runs at 4 Mhz, has 32 8-bit general purpose registers and 32 programmable I/O lines, 8 input lines and 8 output lines.

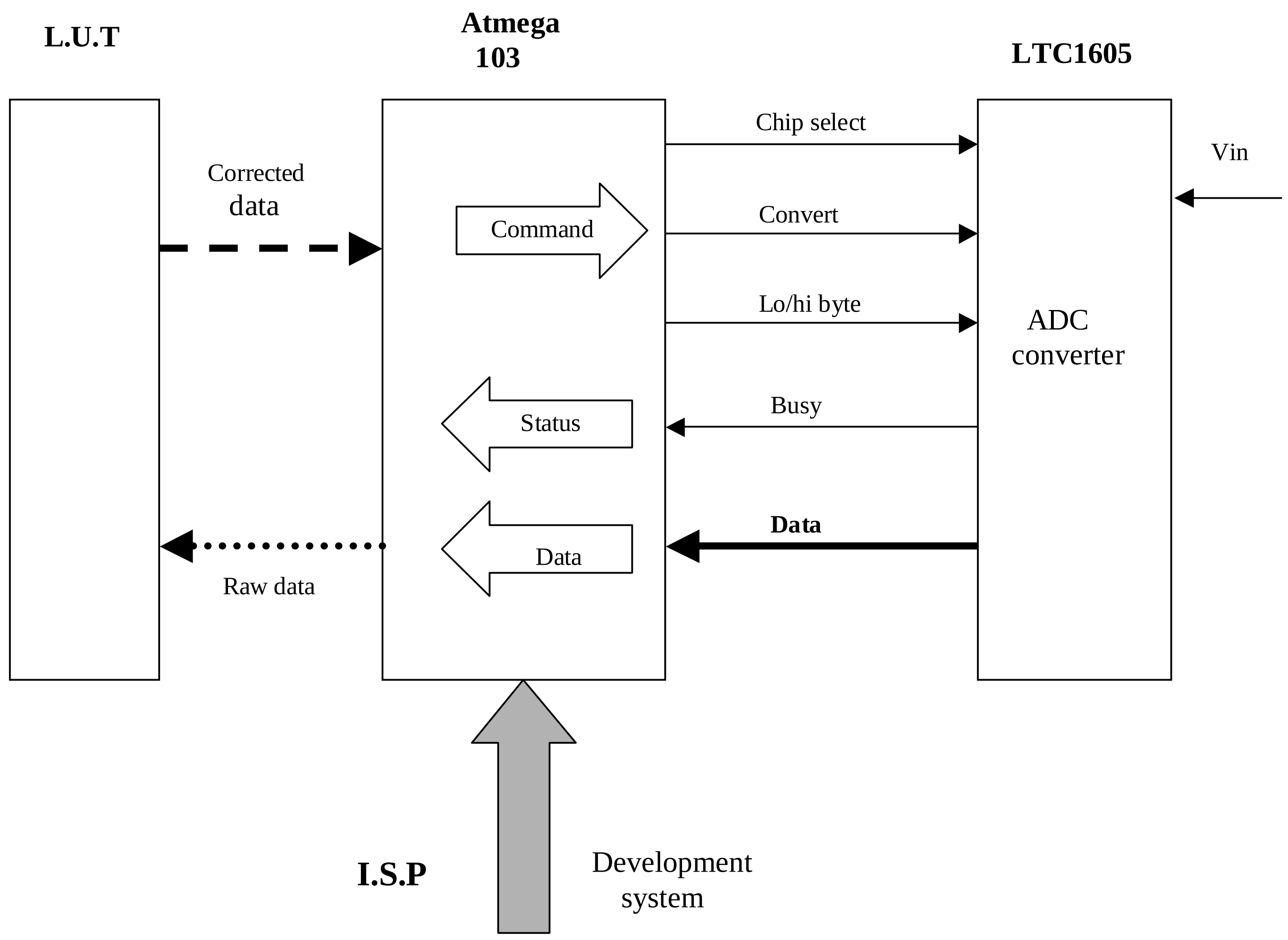
The ADC chosen was the Linear Technologies LTC1605, a 16-bit device with a 8 usec conversion time. This chip is easily interfaced to an 8-bit processor, as it is ‘byte addressable’ using a dedicated control line (lo/hi byte select).

We used the Atmega103 micro on an Atmel STK 300 development card. The LTC1605 ADC chip was connected to the A,C & E ports of the Atmega103.

- Port A (input) is used for status –
  A6 is ‘busy’ bit (converting)

- Port C (output) is used for commands –
  C0 is read/convert command
  C1 is the chip select
  C4 is lo/hi byte select line

- Port E is input data from the ADC chip (8 bits)



SOFTWARE

The program sends the start conversion sequence and waits for the ‘end-of-conversion’ signal (‘busy’ bit). It then gets the data in a two byte read sequence.

The read and convert should be done as fast as possible – preferably the conversion should be done on the last value while the ADC is doing the next acquisition (8 usec). The fastest way to do this is means of a ***look-up table***.

The look-up table (LUT) is a simple one-to-one table i.e each input value (read) has a corresponding output value (corrected value). For our simple test, the restricted LUT was in bytes to limit size to 256 8-bit entries, as 16-bits would need 64k of 16-bit entries !!! The LUT in our example is just the (data value / 2). Note:- that for this simple test, care must be taken not to send *negative* values to the L.U.T !

For a final version, some data smoothing/filtering, followed by correct handling of the signed value before use of the LUT, would be mandatory. Equally, it would be sensible to limit the data from the ADC to 12 to 14 bits and thus reduce the size of the LUT.

THE LOOK-UP TABLE

This is defined as :

```
char lut[] char
```

The high byte is sent to the function and the return value is the ‘corrected’ value. So if the high byte is 10 (0x0A) the return value is 0x05, which is in the 10th position in the table.

```
char lut[] = {
0x00,0x00,0x01,0x01,0x02,0x02,0x03,0x03,0x04,0x04,0x05,
etc……
,0xfc,0xfc,0xfd,0xfd,0xfe,0xfe,0xff,
};
```

Evidently, if the resolution required is more than 8 bits, the table must be ‘wider’ (16 bits). In this case, the table would be defined as:

```
int lut[] int

int lut[] = {
0x0000,0x0000,0x0001,0x0001,0x0002,0x0002,0x0003,0x0003,0x0004,0x0004,0x0005,
etc……
,0xfffc,0xfffc,0xfffd,0xfffd,0xfffe,0xfffe,0xffff,
   };
```

This is obviously unsatisfactory, as the table would be 16 bits wide x 64k deep!! Restricting the ADC data to less bits would then allow the code plus the LUT to reside in the 128K Flash memory.

DATA DISPLAY

For our rudimentary test, data was displayed on the STK 300 development card LEDs (port B). The choice of data is determined by a push-button (port D). If PB1 is pushed, the LEDs shows data from the look-up-table (LUT) (corresponding to the value of the hi-byte of data), else the hi-byte itself. (For ease of viewing, the data is inverted before being output to the LEDs – lit = 1, off = 0)

DEVELOPMENT SYSTEM DESCRIPTION

Hardware:

Atmel STK 300 development card from ANATEK	CHF 180
Atmel Atmega103 microcontroller	CHF 32*
Linear Technologies LTC1605 ADC chip	CHF 45*

* small quantity prices

Software:

AVR Studio (free)
AVR ISP (In System Programmer) program for programming the Flash Memory (free)

AVR MegaBuilder for generating the correct port configuration file. (Free)

IAR Embedded Workbench (compiler, assembler, linker) CHF 3200.-

The ***program*** was written in ‘C’ and compiled with the IAR system. A Borland-C program, generating a file compatible with the IAR compiler, generated the l.u.t ***data***. While this procedure is OK for a test environment, another method of look-up table generation would be needed for the final version. The data file (lut.h) was ‘included’ in the adc.c source file. The whole file was compiled & programmed into the flash memory using the ISP facility.

- **Note that the flash memory of the processor can only be programmed/erased using the ISP facility: the processor itself cannot (re)program its’ own flash memory! This is of interest when considering the possibility of down-line loading of new parameters.**

Files used

Program	C:\1\Atmel_IAR\Ian\ADC\src\adc.c
LUT	C:\1\Atmel_IAR\Ian\ADC\src\lut.h

Object file for downloading using AVR ISP pusher

Object	C:\1\Atmel_IAR\Ian\ADC\Debug\Exe\adc.a90

This file is also copied to D:\AVRTOOLS\adc.a90 for easier loading by the ISP program.

RESULTS

The development was ‘relatively’ straightforward as far as the program implementation was concerned. The IAR embedded workshop is a user-friendly system. One problem encountered during the development was related to the set-up time for reading the (2 times 8-bit) data from the ADC - there is no real-time debug available for the Atmega series of microprocessors, as the program is downloaded into flash memory. A real-time debugger would simplify such problems.

Although the data treatment and display were kept simple, the project did show that it is quite feasible to do the required job using a modern, relatively cheap, 8-bit processor and 16bit ADC.

For a final project in the MPWG context, much more definition of data accuracy, output hardware and LUT data input mechanisms would be required. However, the Flash memory size should allow more data processing etc without restricting the size of the LUT (14bits = 32K*2bytes out of 128Kbytes). Equally, the speed of ADC reading plus processing can very easily guarantee a few KHz data output rate.

CONCLUSIONS

This rather simple demonstration has shown that a 'two chip' hardware design can provide an adequate performance for the LHC Energy Meter ideas. SL/PO will not take this work any further, since it is outside the mandate of the group. Due to the imminent departure of Ian Barnett, who has executed this work, anyone wishing to obtain more detail for future use should do so rapidly.