CpE 213
Digital Systems Design

Lecture 20

Tuesday 11/4/2003

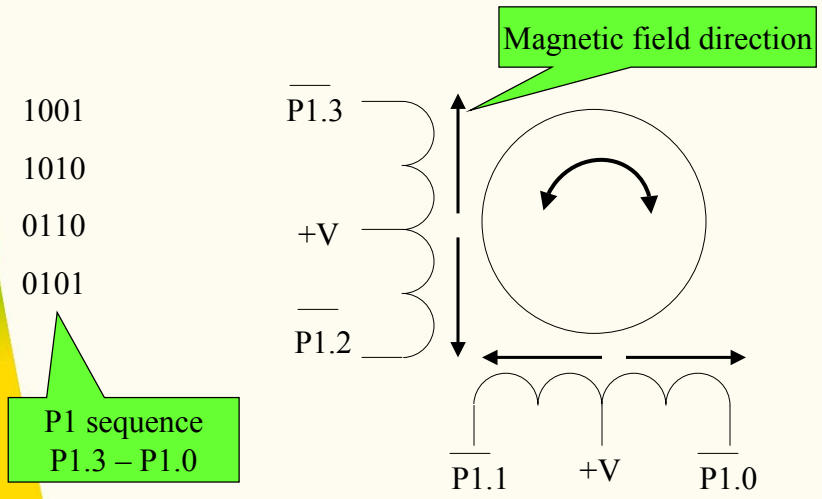
UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

Announcements

- Exam 2: Thursday Nov. 13th
  - Topics to be covered and practice exams have been posted.
  - Will cover everything up to, but not including interrupts.
  - Emphasis will be on material not covered in exam 1.
- Assignment 7 has been posted.
  - Due: Nov. 11th at 11 am.
  - Will be solved in review session.
- Review session: Tuesday Nov. 11th, from 7 to 9 pm in EECH G31 (this room).
- Suggest review topics in addition to HW 7.

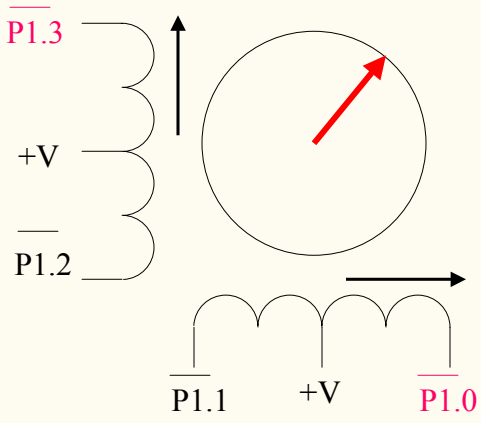
Stepper Motor Review

Unipolar Stepper Motor Basics



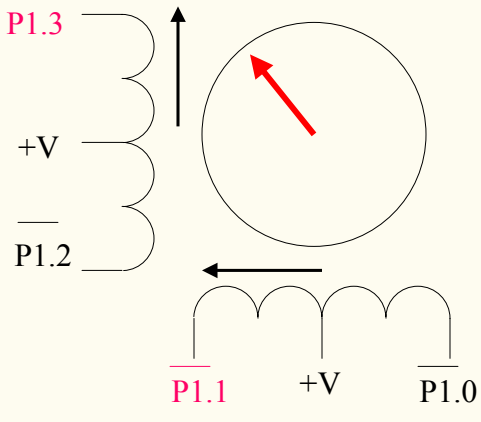
Stepper Motor Basics

1001



Stepper Motor Basics

1010



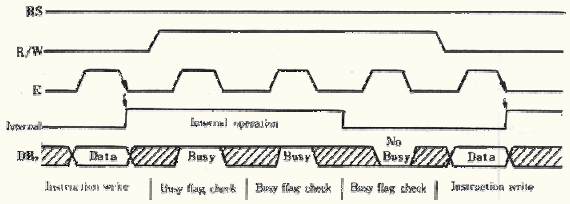
LCD Display Programming

LCD Modules

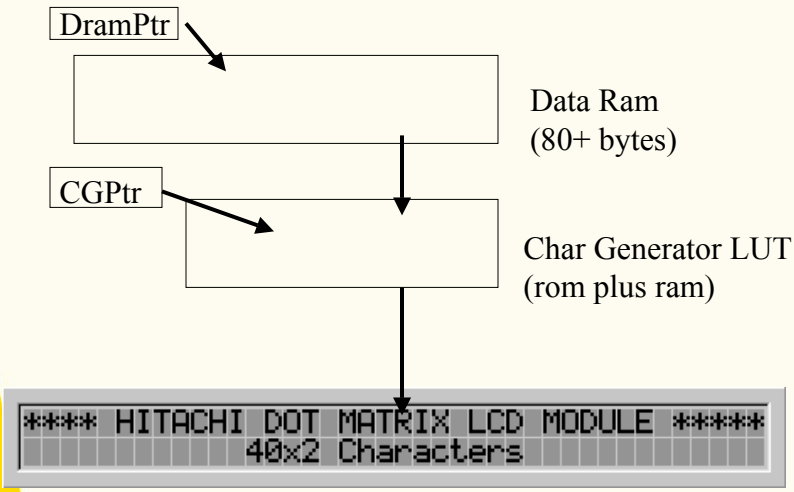
- LCD = Liquid Crystal Display
- Info from the LCD FAQ at:
  - http://www.repairfaq.org/filipg
- Module = LCD panel plus controller
  - panel = polarizers, glass plates, LC
  - Twisted Nematic Field Effect
  - Controller: HD44780 microprocessor

HD44780 Controller

- RS (4) Register Select 1=data, 0=instruction
- R/W* (5) Read/Write 1=toMCU, 0=toLCD
- E (6) Enable (Clk) Falling edge triggered
- DB(7:0) Data bus. Also four bit versions.



HD44780 Controller

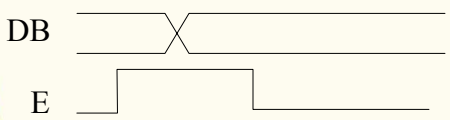


Control Procedure

- Set 8 bit data mode (write 0x30)
- Turn on display (write 0x0e)
- Set dram pointer (write 0x80+val)
- Write characters (ascii code set)

Command write

```
void lcdcmdwr(uchar dbyte){
   rs=0;      //sel instr reg
   ena=1;     // raise clock high
   P1= dbyte; //output a byte
   ena= 0;    // lower clock and
              // latch data
}
```



Data Write

```
void lcddatwr(uchar dbyte){
   rs=1;      //sel data reg
   ena=1;     // raise clock high
   P1= dbyte; //output a byte
   ena= 0;    // lower clock and
              // latch data
}
```

Display data

```
void display (uchar pos, uchar *s){
 lcdcmdwr(0x80|pos); msec(1);
 while(*s) lcddatawr(*s++); msec(1);
}
void main(void){
...
  display(0,"a message");
...
```

Pulse Width Modulation

- One bit Digital to Analog converter
- Commonly used in
  - DC motor drives
  - Power supplies
  - Many micro-controllers have PWM hardware
- Square wave output
  - variable duty cycle (pwm)
  - fixed frequency

Pulse Width Modulation

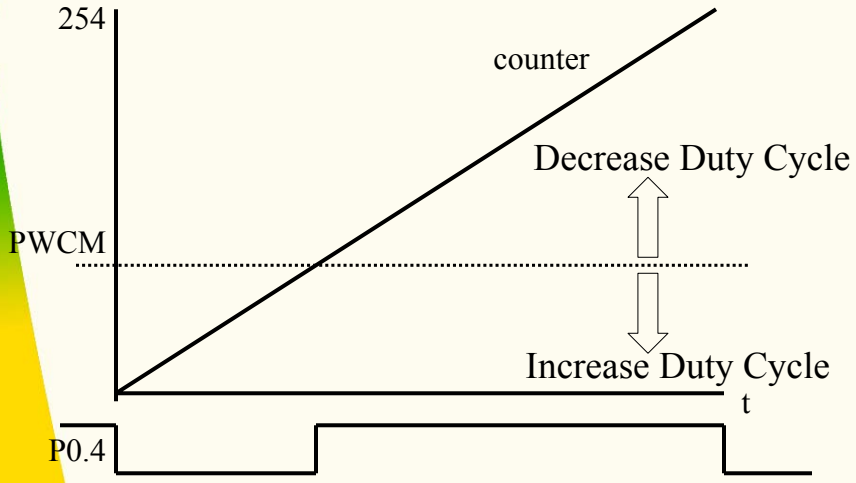
PWM

- Counter plus comparator
- Algorithm
  - Set bit = 1
  - increment count
  - if count = duty cycle then set bit = 0
  - continue until count = period
- 0 < duty cycle < period

80C752 PWM

- Alternative function for P0.4
- 8 bit Prescaler (PWMP=sfr 0x8F)
- 8 bit modulo 255 counter (0 to 254)
- 8 bit compare register (PWCM=sfr 0x8E)
- P0.4 (pwmout) initially 0
- pwmout set high when counter=PWCM
- PWCM analogous to duty cycle

80C752 PWM



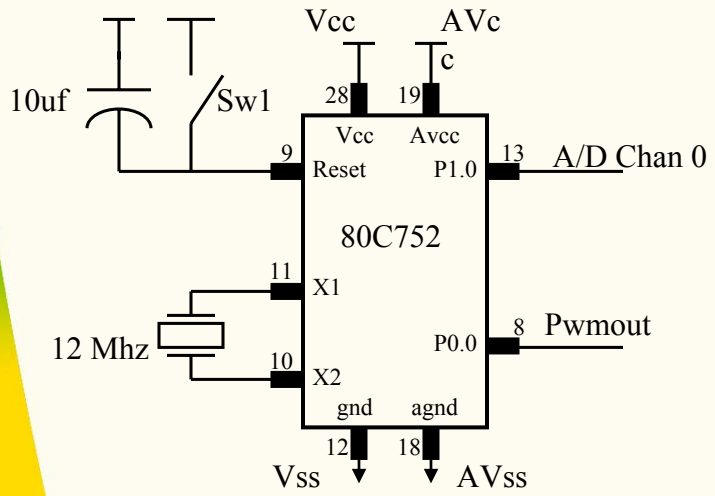
80C752 PWM

- PWM Period = (2*(1+PWMP)*255)/12E6
- Min Freq = 92Hz (PWMP=255)
- Max Freq = 23.5 Khz (PWMP=0)
- Example PWMPer = 10E-3 (1 kHz)
  - PWMP= 12E-3/(255*2) - 1 = 22 to 23
- See 80C752 data sheet for details

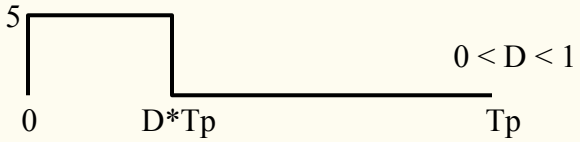
Example: A2DtoPwm

- Low Pass Filter PWM to get Avg Value
- Many mechanical devices are natural LPF's
- PWM makes a pretty good one bit digital to analog converter (D/A or DeeTooA)
- Square wave frequency must be lower than LPF cutoff.
- What is minimum period for software PWM?

A/D to PWM



Average Value of Square Wave



$$V_{avg} = \frac{1}{T_P}\int_0^{D\cdot T_P} 5dt$$

$$= \frac{1}{T_P} 5t\Big|_0^{D\cdot T_P} = 5\cdot D$$

Software PWM period

- About 1.5 mSec for full 8 bit duty cycle
- Can get about 1.0 mS for 7 bit duty cycle
- Use hardware PWM for greater range
- 80C752's PWM range is 92 Hz to 23.5kHz with 12 Mhz XTAL

Analog to Digital Conversion (ADC)

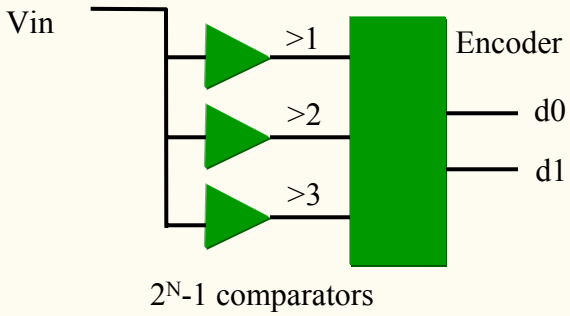
ADC Summary

- Direct conversion (Flash converter)
- Successive approximation
  - similar to binary search (20 questions)
- Dual slope; integrating
- Sigma-delta; oversampling
- Pipeline; subranging quantizer
- ADC model: sampler + quantizer
- Analog mux plus ADC combo is common

Flash converter

- Very fast; low resolution; large size, power
- Video conversion rates; 100Msps+

000 00
100 01
110 10
111 11

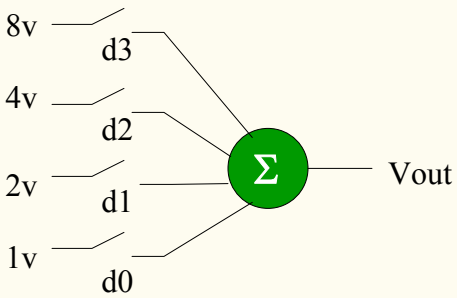


2^N-1 comparators

Digital to analog converter (DAC)

- Sum N weighted voltages (currents; charge)
- Simple example:

0000 0v
0001 1v
0010 2v
0011 3v
…
1111 15v

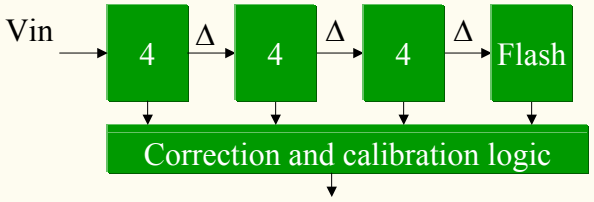


Sigma-Delta ADC

- Also called oversampling converters
- Sigma-delta modulator followed by digital decimation filter
- High rate (2Mhz+) one bit ADC followed by digital filter reduces sample rate to 8kHz+ and higher resolution (16bits+)
- Used mainly in DSP applications

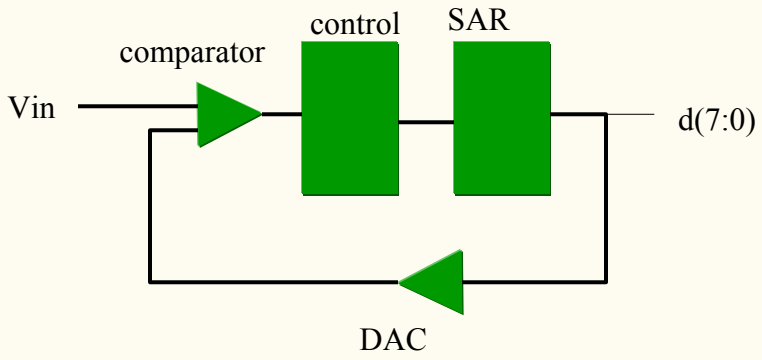
Pipeline ADC

- Fast, high resolution
- Applications include communications and CCD imaging
- Cascaded coarse ADC plus difference stages

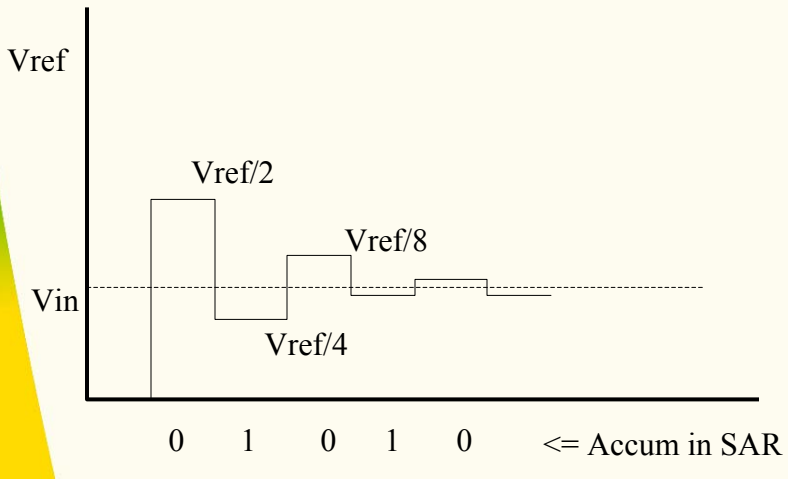


Successive Approximation

- Control does a binary search for digital representation of Vin

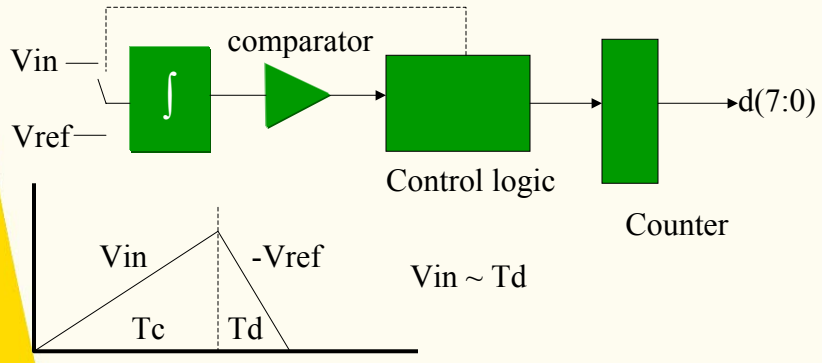


Binary search algorithm



Integrating ADC

- Cheap and simple
- Measure Cap charge/discharge times



An Example A/D Converter

- Philips 8xC752 Microcontroller A-D
  - Five channel (5 multiplexed inputs)
  - Eight bit
  - Successive Approximation (binary search)
  - Conversion rate:
  - Input voltage range:

8xC752 SFR's

- A-D control register
  - sfr ADCON= 0xA0
  - Enable, status, start bits and channel select
- A-D data register
  - sfr ADAT= 0x84;
  - 8 bit converted analog signal
- Neither are bit addressable!
- Complete example in following pages.

CpE 213 Example
80C752 Analog input, PWM output
May 7, 2001

Background

Pulse width modulation (PWM) is commonly used in microcontroller applications to drive continuous signal systems which are natural integrators such as motors and other electromechanical devices, lamps, heaters, capacitive loads such as switching power supplies, etc. A PWM output can be generated with a single bit while the alternative digital to analog converter requires several bits, normally 8 to 12. A PWM signal can be defined as a square wave with a constant period (T) and amplitude, and a duty cycle which is proportional to the output signal f(t). Like all sampled data signals, the frequency of the square wave (f=1/T) must be chosen so that it is at least twice the highest frequency in f(t). A square wave whose period is 0.001 sec can be used to produce an analog output signal which varies at about 10-100 Hz max. There may be other considerations as well such as making the frequency high enough so that it won't be annoying to humans. The duty cycle is generally understood to be the ratio of the square wave's on time to its period.

The data sheet for the Philips 87C752 shows that it has a five channel 8-bit A/D converter, is packaged in a 28 pin DIP, has 2k ROM and 64 bytes of RAM on chip, is not expandable, and has an 8 bit PWM output/timer. For part 1 example we will use channel 0 of the A/D converter. We will NOT use the PWM device but will program a *software* PWM instead. An A/D converter simply converts an analog input signal to a digital value which is proportional to the ratio of the input signal to its maximum range (0 to 5 volts in this case). Understanding the details of successive approximation A/D converters isn't required for this exercise but if you are interested, any good text on digital electronics should be of help. Horowitz and Hill's *The art of electronics* is a good source.

Part 2 of the example will modify the code to use the 87C752's hardware pwm device.

Problem:

Write a C program for the 87c752 which will sample channel 0 of the A/D and produce a PWM output whose duty cycle is proportional to the sampled analog signal. Do NOT use the 752's hardware PWM! The PWM output should have a period of about 1 mSec. Use a software wait loop to generate a square wave with the appropriate period and duty cycle on P0.4 (port 0 bit 4). Compile your program for the '752 and test it with dScope. Note that dScope doesn't directly support the '752. There is no support for either the ADC or the PWM device. This means you will need to exercise some ingenuity in order to adequately test your program.

Solution

The following listing is a *driver* for the 752's ADC.

```
/* 87c752 A-D driver */
#define ENADC 0x20            /* ADCON(5) enable ADC */
#define ADCI 0x10             /* ADCON(4) A/D done flag. */
#define ADCS 0x08             /* ADCON(3) A/D converter status bit */

typedef unsigned char uchar;
sfr ADAT= 0x84;               /* A/D converter data register */
sfr ADCON= 0xA0;              /* A/D control register */

void init_a2d(void){
 ADCON= ENADC; /* enable a/d inputs on pins P1(4:0) */
 }
```

```
uchar get_a2d(uchar chan){
  ADCON= ENADC | ADCS | (chan & 0x07);    /* select chan, enab a/d, and
start */
  while ((ADCON & ADCI)==0);        /* wait until ADCI=1 */
  return ADAT;                      /* read ADAT and return */
}
```

Although there is an include file for the 87C752, it doesn't appear to be complete so the required new sfr's are simply defined inline. This module includes an initialization routine that simply turns on the ADC and a *get* routine that starts a conversion, waits for it to complete, and returns the digitized value. If this code is to be used as part of a closed loop control system, it would need to be modified to provide a well defined sample period.

The next routine is a simple software PWM output. It generates a square wave with a period of at least 1500 μsec and a duty cycle proportional to the parameter *duty*. A PWM device is simply a counter and a comparator. An output bit is asserted when the counter goes through zero and toggled when the counter reaches the compare value (duty in this case). There is usually a simple provision made to vary the period of the resulting square wave. In this example, extra delay could be inserted in the loop to produce a pwm with a longer period.

```
#include <reg51.h>
typedef unsigned char uchar;
sbit Pwmout= P0^4; //P0.4 is used for pwm output

void init_pwm(void){} //null init routine

void put_pwm(uchar duty){
uchar i;
Pwmout= 1;
for (i=0;i<255;i++){
  if (i==duty) Pwmout= 0 ;
  };
 Pwmout= 1;
}
```

The next and final listing is a main program that calls the two driver functions. The result is a simple application that samples an analog signal and produces a pulse width modulated version of the signal at P0.4.

```
/* a2d to pwm */
/*typedef similar to define except interpreted by compiler */
typedef unsigned char uchar;

void put_pwm(uchar duty);
uchar get_a2d(uchar chan);
void init_a2d(void);

void main(void){
  init_a2d();
  init_pwm();
 while(1){
  uchar duty= get_a2d(0);
  put_pwm(duty);
  }
```

```
}
```

Hardware PWM

The 87C752 includes a hardware pwm device. According to the Philips datasheet for the 752, there are three special function registers associated with the device as shown in Table 2 below.

SFR	Function	Location
PWMP	Prescaler	8F
PWCM	Comparator	8E
PWENA	Enable (bit 0)	FE

The prescaler determines the repetition rate and thus the period according to the equation:

$$f_{pwm} = \frac{f_{clk}}{2(1 + PWMP)255}$$

The repetition rate can vary between 92 Hz and 23.5 Khz with a 12 Mhz clock. PWMP for a 10 kHz repetition rate is 22. To use the pwm device, the code must enable it by setting bit 0 of PWENA to 1, set the desired PWMP value, and vary the PWCM value as required. The following listing accomplishes this:

```
/* simple hardware pwm driver for 87C752 */
typedef unsigned char uchar;
//P0.4 is used for pwm output
/* define 752 pwm sfr's */
sfr PWMP= 0x8F;  //prescaler
sfr PWCM= 0x8E;  //duty cycle comparator
sfr PWENA= 0xFE; //enable is bit 0

/* init sets up pwm for a 10Khz rep rate.
   minimum is 0 (92 Hz), max is 255 (23.5 kHz)
*/
void init_pwm(void){
 PWMP=22; /* approx 10 kHz pwm rep rate */
 PWENA=1; /* enable pwm feature */
 }
void put_pwm(uchar duty){
 PWCM= duty;
 }
```

Testing

Unfortunately μvision 2 does not support the 752's peripherals. The debugger simply uses a generic set of 8051 I/O devices. Consequently the ADC's status bit and data register will need to be set manually. In order to make this a bit easier, the following dscope signal function can be used.

```
signal void adc (void) {
 while(1){
  while ((ADCON & 0x28)!= 0x28)
    {twatch(20);}; /* wait until started */
  twatch(40); /* ADC takes 40 cycles to complete */
  ADCON=  (ADCON & 0xf7 ) | 0x10; /* clear adcs, set adci */
 }
```

```
}
```

Signal functions are like C functions that can be run inside the debugger. Signal functions run in parallel with everything else and can be used to simulate various external devices while your program is running. This one simple polls the ADCON control register every 20 instruction cycles (twatch(20)) and sets the conversion complete status bit 40 instruction cycles after a conversion is started. The data register ADAT must be set manually using the dscope command line. The command adat=255 corresponds to an analog value of 5v while adat=127 corresponds to an analog value of about 2.5v.