**Etch-A-Sketch with Haynes Retro Arcade Kit (AKA Pong)**

**Using the program ‘Paint’, it is possible to draw pictures, whereby the left wheel controls the cursor position on the Y axis (height); while the wheel on the right controls the position on the X axis. The longer the cursor is left at a certain position, the brighter the pixel is illuminated afterwards.**

by Patrick Kaiser

[](http://www.elo-web.de/ximage/1003Malen1.jpg)Who doesn’t remember these drawing pads from their childhood – where two wheels were used to draw a picture and then shaking the tablet would erase the picture again?

The Retro Arcade assembly kit, with its two control wheels, is perfect for use as a drawing tool – and so is interesting even for the little ones.

Using the program ‘Paint’, it is possible to draw pictures, whereby the left wheel controls the cursor position on the Y axis (height); while the wheel on the right controls the position on the X axis.

The longer the cursor is left at a certain position, the brighter the pixel is illuminated afterwards.

After setting the first position, it takes about half a second for the pixel to reach its first level of brightness. If you want to switch off the game, you turn both wheels inwards – here it is slightly different from its predecessor. ;-)

The fact that the cursor flickers is due to it being especially bright and standing out against the dark background. The less it flickers, the brighter the pixel afterwards. This makes it possible to also hide the cursor in a light area, after if you’ve drawn an advertising sign, for example. I hope you have lots of fun getting creative with your drawing and I’d love to receive more feedback on a work of art created in this way.

[](http://www.elo-web.de/ximage/1003Malen4.jpg)The game was written in the Arduino environment in C++ language.

In the description below, I explain how to set up the environment and compile the game yourself.

I hope this gives you the basics to set your own Pong environment, as the environment offers a few advantages – which I will come to later.

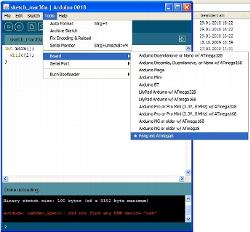
Now let’s look at setting it up. In my case, the operating system is Windows XP. First you download the latest version of the environment from [**arduino.cc**](http://arduino.cc/en/Main/Software):

Then you unzip/extract it into your favourite Pong folder. If you want to save the settings in program files (rather than in a user folder), you have to switch to the subfolder ‘lib’, go to the file ‘preferences.txt’ and change the entry ‘#settings.path=data’ to ‘settings.path=./settings’ (without the ‘#’ !) to save the configuration file in the ‘Settings’ subfolder. Here you can also amend the entry ‘#build.path=build’ if you want to have the temporary data (including the compiled \*.hex file) in a certain folder. This makes sense if you experience difficulties with an integrated Flash program.

The next step is to go into the subfolder ‘hardware/arduino’ and edit the file ‘programmers.txt’ – in my case, I have simply added the following section:

##############################################################  
pong.name=Pong with ATmega8  
pong.upload.protocol=stk500  
pong.upload.maximum\_size=8192  
pong.upload.speed=19200  
#pong.bootloader.low\_fuses=0xe4  
#pong.bootloader.high\_fuses=0xd9  
pong.bootloader.path=atmega8  
pong.bootloader.file=ATmegaBOOT.hex  
#pong.bootloader.unlock\_bits=0xFF  
#pong.bootloader.lock\_bits=0xFF  
pong.build.mcu=atmega8  
pong.build.f\_cpu=8000000L  
pong.build.core=arduino

The Bootloader details should simply be ignored – but to be safe, we nonetheless comment these out with a ‘#’. The speed was adjusted here to the 8MHz of our Pong. Now the environment has to be restarted and the newly added entry ‘Pong with ATmega8’ selected in the Menu ‘Tools ->Board’.

[](http://www.elo-web.de/ximage/1003Malen3.jpg)Then we close the environment again, as only now is the final file created with the settings. In my case, it is saved in my subfolder ‘Settings’ and has the name ‘preferences.txt’, though it has nothing in common with the previous file with the same name!

Roughly in the middle of the file (line 26), there is the entry ‘upload.using=bootloader’. If you want to operate your programmers and don’t want a Bootloader for the serial upload, the entry has to be changed.

You can get an idea of the entries in the file ‘hardware/arduino/programmers.txt’ before the dot in each case. In my case, I’d like to flash Pong as normal with the programmer, and my program is an AVR ISP MKII, so for me the entry is as follows: ‘upload.using=avrispmkii’. Now you can launch the environment and try to flash a ‘int main(){while(1);}’ – if this doesn’t work, you presumably have to install the libusb filter (libusb-win32-filter-bin-0.1.12.2.exe ) because avrdude (the flasher) has not found the USB programmer. Alternatively you can flash the hex file from the build path manually (discussed at the beginning).

Why all this effort? Those familiar with the Arduino platform appreciate the environment, the surrounding community and the many free libraries: there are libraries for playing back sounds via connected speakers (on any port pins) or for driving servos or displays or 12C and more. This makes it possible, for example, to write a Pong class or a Pong display class and make a library from it, which can be added at the click of a mouse. (Unfortunately in an argument with the compiler, this time I had to back down because my C++ skills are too weak – otherwise I would have supplied that, of course …). Besides this, the environment works across all platforms – this means that Linux users are also included, once again.

However, when working with the environment, you should also be aware of its peculiarities. Firstly, the environment numbers all the pins and differentiates between analogue and digital pins. If you’d like to use analogue pins digitally, you can address them with the numbers they would have if they were the last digital pins. You can find an overview of this at: www.arduino.cc

Secondly, you need to know that Arduino programs usually have a ‘void setup(){...}’ function and a ‘void loop(){...}’ function. The setup function is used to initialise everything that has to be initialised, and the loop function is invoked by the system automatically (in practically the same was as for a while(1) block). It is not compulsory to use these functions, though you should use them, as the delay function works with timers and the libraries assume that the environment is being used in the way it was conceived. Note: The timer0 is then taboo and should only be relaunched with ‘init();’ after a ‘sleep’. But in principle (if you omit the io.h), you can also bring in normal AVR studio code or C+ + code – however, you should then use the delay functions of the gcc: comment out the corresponding delay functions in the file ‘hardware\arduino\cores\arduino\wiring.h’ and insert the following:

#define F\_CPU 8000000UL // CPU speed (for delay routine)  
#include <util/delay.h> // Definition of delay functions (\_delay\_ms)  
#define delayMicroseconds(x) \_delay\_us(x)  
#define delay(x) \_delay\_ms(x)

In addition, you should then comment out the contents of the file ‘wiring.c’ practically in their entirety. Supposedly it is also much easier to get to grips with enclosed functions. Hopefully soon, I’d like to find a Pong library that is easy to use – so that you no longer have to be proficient in electronics in order to program with the Pong assembly kit.

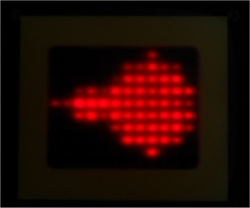
Download: [**Arduino project and hex file**](http://www.elo-web.de/xattachment/1003malen.zip)

#include <avr/interrupt.h> /\* Interrupt treatment routines (for timer interrupt) \*/  
#include <avr/sleep.h>  
#define WIDTH 12 /\* Width of the display – corresponds to x\*/  
#define HEIGHT 10 /\* Height of the display – corresponds to y\*/  
#define disp\_pin0 14  
#define disp\_pin1 15  
#define disp\_pin2 16  
#define disp\_pin3 17  
#define disp\_pin4 4  
#define disp\_pin5 5  
#define disp\_pin6 6  
#define disp\_pin7 7  
#define disp\_pin8 8  
#define disp\_pin9 9  
#define disp\_pin\_data 12  
#define disp\_pin\_clock 11  
#define disp\_pin\_strobe 10  
#define INT0pin 2 //queries the coin slot and supplies power for the potentiometers  
#define frame\_slowness 0x8; //How long to wait until the next column is displayed. – Don’t make this value too small!  
volatile uint8\_t leds[WIDTH][HEIGHT]; //Height = the 10 (of the 16) bits  
volatile uint8\_t aktZeile; //column-sampling-counter  
volatile uint8\_t brightness\_cnt;  
volatile uint16\_t poti\_a;  
volatile uint16\_t poti\_b;  
volatile uint8\_t adc\_switch;  
volatile uint16\_t permanent\_cnt; //Helps with time functions  
//save old position to recognise movement  
uint8\_t pa\_alt;  
uint8\_t pb\_alt;  
void one\_reg();  
void null\_reg();  
void empty\_reg();  
void fill\_reg(); //=Erase screen because negative switch with register  
void set\_disp\_ports();  
void clr\_disp\_ports();  
void adc\_init();  
void component\_inits();  
void power\_off();  
void power\_on();  
ISR(ADC\_vect);  
ISR(TIMER2\_COMP\_vect);  
ISR(INT0\_vect);  
void setup()  
{  
cli();  
aktZeile=0;  
brightness\_cnt=0;  
permanent\_cnt=0;  
//Bildchen init/generieren  
for(int x=0;x<WIDTH;x++){  
for(int y=0;y<HEIGHT;y++){  
leds[x][y]=0;  
}  
}  
/\*---------------------------------------------------  
\* Configure ports (inputs/outputs)  
\*---------------------------------------------------\*/  
pinMode(disp\_pin0,OUTPUT);  
pinMode(disp\_pin1,OUTPUT);  
pinMode(disp\_pin2,OUTPUT);  
pinMode(disp\_pin3,OUTPUT);  
pinMode(disp\_pin4,OUTPUT);  
pinMode(disp\_pin5,OUTPUT);  
pinMode(disp\_pin6,OUTPUT);  
pinMode(disp\_pin7,OUTPUT);  
pinMode(disp\_pin8,OUTPUT);  
pinMode(disp\_pin9,OUTPUT);  
pinMode(disp\_pin\_data,OUTPUT);  
pinMode(disp\_pin\_clock,OUTPUT);  
pinMode(disp\_pin\_strobe,OUTPUT);  
pinMode(INT0pin,OUTPUT);  
digitalWrite(INT0pin,LOW);  
fill\_reg(); //with HIGH to switch of the LEDs  
component\_inits();  
set\_sleep\_mode(SLEEP\_MODE\_ADC);  
sei();  
//especially good for performance if it is permanently left on:  
digitalWrite(disp\_pin\_strobe, HIGH);  
/\*  
//save old position to recognise movement  
uint8\_t pa\_alt;  
uint8\_t pb\_alt;  
\*/  
}  
void loop(){  
//selects potentiometers  
if((ADCSRA & \_BV(ADSC)) == 0){ //\_bv = bit\_value  
sleep\_mode();  
}  
//calculates the potentiometer value to display dimensions  
uint8\_t pa=poti\_a/84;  
uint8\_t pb=poti\_b/96;  
if(pa>=WIDTH) pa=WIDTH-1;  
if(pb>=HEIGHT) pb=HEIGHT-1;  
int t=leds[WIDTH-pa-1][pb];//temporarily save brightness value  
leds[WIDTH-pa-1][pb]=5;//brightness set to very high -> position cursor  
if((pa==pa\_alt)&&(pb==pb\_alt)){  
if(permanent\_cnt>1000){  
if(t<5) t++;  
//checks if it is being switched off  
if((poti\_a>1010)&&(poti\_b>1010)){  
power\_off();  
power\_on();  
t=0;  
}  
permanent\_cnt=0;  
}  
} else permanent\_cnt=0;  
pa\_alt=pa;  
pb\_alt=pb;  
delay(1);  
leds[WIDTH-pa-1][pb]=t;  
}  
ISR(TIMER2\_COMP\_vect){  
cli();  
if(aktZeile==0){  
clr\_disp\_ports(); //anti-flirr ;-)  
null\_reg();  
set\_disp\_ports();  
aktZeile++;  
} else {  
clr\_disp\_ports(); //anti-flirr  
one\_reg();  
set\_disp\_ports();  
aktZeile++;  
if(aktZeile>=WIDTH){  
aktZeile=0;  
brightness\_cnt++;  
if(brightness\_cnt>5) brightness\_cnt=0;  
}  
}  
permanent\_cnt++;  
sei();  
}  
void one\_reg()  
{  
digitalWrite(disp\_pin\_data, HIGH); /\* send one (PB4 = HIGH) \*/  
//digitalWrite(disp\_pin\_strobe, HIGH);  
digitalWrite(disp\_pin\_clock, HIGH);  
digitalWrite(disp\_pin\_clock, LOW);  
//digitalWrite(disp\_pin\_strobe, LOW);  
digitalWrite(disp\_pin\_data, LOW); /\* reset data to LOW (PB4 = LOW) \*/  
}  
void null\_reg()  
{  
digitalWrite(disp\_pin\_data, LOW);  
//digitalWrite(disp\_pin\_strobe, HIGH);  
digitalWrite(disp\_pin\_clock, HIGH);  
digitalWrite(disp\_pin\_clock, LOW);  
//digitalWrite(disp\_pin\_strobe, LOW);  
}  
//fills with LOW  
void empty\_register()  
{  
for(int i=0;i<16;i++){  
null\_reg();  
}  
}  
//fills with HIGH  
void fill\_reg()  
{  
for(int i=0;i<16;i++){  
one\_reg();  
}  
}  
void set\_disp\_ports()  
{  
//-----------------------------------------  
if(leds[aktZeile][0]<=brightness\_cnt) digitalWrite(disp\_pin0,LOW); else digitalWrite(disp\_pin0,HIGH);  
if(leds[aktZeile][1]<=brightness\_cnt) digitalWrite(disp\_pin1,LOW); else digitalWrite(disp\_pin1,HIGH);  
if(leds[aktZeile][2]<=brightness\_cnt) digitalWrite(disp\_pin2,LOW); else digitalWrite(disp\_pin2,HIGH);  
if(leds[aktZeile][3]<=brightness\_cnt) digitalWrite(disp\_pin3,LOW); else digitalWrite(disp\_pin3,HIGH);  
if(leds[aktZeile][4]<=brightness\_cnt) digitalWrite(disp\_pin4,LOW); else digitalWrite(disp\_pin4,HIGH);  
if(leds[aktZeile][5]<=brightness\_cnt) digitalWrite(disp\_pin5,LOW); else digitalWrite(disp\_pin5,HIGH);  
if(leds[aktZeile][6]<=brightness\_cnt) digitalWrite(disp\_pin6,LOW); else digitalWrite(disp\_pin6,HIGH);  
if(leds[aktZeile][7]<=brightness\_cnt) digitalWrite(disp\_pin7,LOW); else digitalWrite(disp\_pin7,HIGH);  
if(leds[aktZeile][8]<=brightness\_cnt) digitalWrite(disp\_pin8,LOW); else digitalWrite(disp\_pin8,HIGH);  
if(leds[aktZeile][9]<=brightness\_cnt) digitalWrite(disp\_pin9,LOW); else digitalWrite(disp\_pin9,HIGH);  
//-----------------------------------------  
}  
void clr\_disp\_ports()  
{/  
/-----------------------------------------  
digitalWrite(disp\_pin0,LOW);  
digitalWrite(disp\_pin1,LOW);  
digitalWrite(disp\_pin2,LOW);  
digitalWrite(disp\_pin3,LOW);  
digitalWrite(disp\_pin4,LOW);  
digitalWrite(disp\_pin5,LOW);  
digitalWrite(disp\_pin6,LOW);  
digitalWrite(disp\_pin7,LOW);  
digitalWrite(disp\_pin8,LOW);  
digitalWrite(disp\_pin9,LOW);  
//-----------------------------------------  
//The registry could also be deleted here...  
}  
ISR(ADC\_vect){  
cli();  
if(adc\_switch==6){  
poti\_b=ADCW;  
ADMUX =0b01000111;  
adc\_switch=7;  
} else {  
poti\_a=ADCW;  
ADMUX =0b01000110;  
adc\_switch=6;  
}  
sei();  
}  
void adc\_init(){  
adc\_switch=6; //adc6  
//start ADC6+ADC7  
//ref-voltage to AVCC, adjust left for 10bit result, at the end 0110=ADC6 0111=ADC7  
//which can be changed during conversion and remains until it is completed  
ADMUX =0b01000110; //ADC6  
ADCSRA = \_BV(ADEN) | \_BV(ADIE) | \_BV(ADPS2) | \_BV(ADPS1) | \_BV(ADPS0);  
//ADCSRA=0b11101111;  
}  
void component\_inits(){  
aktZeile=0;  
brightness\_cnt=0;  
permanent\_cnt=0;  
//initialise/generate picture  
for(int x=0;x<WIDTH;x++){  
for(int y=0;y<HEIGHT;y++){  
leds[x][y]=0;  
}  
}  
// Timer0 is unfortunately already in use if the corresponding sections in wiring.c and  
wiring.h are commented out  
// so consequently, the delay routines from the avr-gcc-lib have to be adopted  
// however, it still has to be init()ialised, otherwise it won’t wake up from sleep anymore (this  
routine is also invoked after switching on)  
init();  
//Timer 2 (8bit) for multiplexing of LEDs  
TCCR2 = (1<<CS22)|(1<<CS21)|(1<<WGM21); // 1/256 prescaler, CTC  
OCR2= frame\_slowness; // simply an init value  
TIMSK |= (1<<OCIE2); // compare-irq enabled  
adc\_init(); //ADC on  
}  
void power\_off()  
{  
//set sleep  
set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN);  
ADCSRA=0; //ADC complete conversion  
ADCSRA=0; //ADC off  
//deactivate timer  
TCCR0=0;  
TCCR1B=0;  
TCCR2=0;  
fill\_reg();  
clr\_disp\_ports();  
//somehow it also works without the following line \*shrugs shoulders\*  
pinMode(INT0pin,INPUT);  
digitalWrite(INT0pin,HIGH); //PullUP = take voltage of potentiometers and INT0 make sensitive for gnd switch  
//activate int0 (to low-level sensitive)  
MCUCR &= ~(0x3); // level-controlled Interrupt to INT0  
GICR |= (1 << INT0); // release external Interrupt  
//sleep until int0 (switch) wakes up  
sleep\_mode();  
}  
void power\_on()  
{  
pinMode(INT0pin,OUTPUT); // deactivate 5V again (->potentiometers activated)  
digitalWrite(INT0pin,LOW);  
digitalWrite(disp\_pin\_strobe, HIGH); //so that reg adopts the data  
set\_sleep\_mode(SLEEP\_MODE\_ADC);  
component\_inits();  
sei();  
}  
ISR(INT0\_vect) {  
GICR &= ~(1 << INT0); // block external Interrupt  
}

**Addendum 31.3.:**  
I have tried it out again and for simple compiling I have found an easier solution for a temporary installation (however, the timer then doesn’t work properly, the settings are in the user folder and you have less space for the compiled file because the Bootloader is included – vlt nevertheless also compiles a larger amount). You can simply load the program in any Arduino environment, as a board, select ‘Arduino/NG or older w/ATmega8’ (this is the last point; see also the screen shot in the submitted report) and press the Play button. Then you can retrieve the hex file from the Temporary Folder and flash with AVR Studio, for example. But I think it’s cooler if you can do everything from the environment! ;-) The differences of the hex files created in this way don’t seem to be so big (attached are three different compilations and disassembly – ‘WinMerge’ shows the differences clearly, by marking both files in Explorer and right-clicking on WinMerge to select it).

Download: [**Paint2**](http://www.elo-web.de/xattachment/1003Malen2.zip)

**Addendum 31.3.: Apple man**

[](http://www.elo-web.de/ximage/1003Malen5.jpg)

Because I am not very good at drawing, I’ve taken a photo of an apple man on Pong.

However, I have cheated a bit and have slightly rewritten the processing code from http://www.cognitiones.de/doku.php/mandelbroetchen (the Arduino environment is based on processing and can be easily combined) and incorporated it into the framework of the Paint program.

What’s special about this is that I have used a type of anti-aliasing:

I work with double the resolution that Pong has and include this as brightness in the surrounding pixels.