**Ping pong becomes bing bong – a clock with a Big Ben melody**

**Here ping pong is converted to bing bong: a clock that shows the time on an LED display and plays the well-known melody and chimes of Big Ben, the famous clock tower in London, every quarter of an hour. Via additional hardware connected to a ping pong circuit board with simple components, the melodies are played on the so-called ‘Simple SD Audio Player’.**

by Michael Gaus

[](http://www.elo-web.de/ximage/1003bingbong.jpg)Here ping pong is converted to bing bong: a clock that shows the time on an LED display and plays the well-known melody and chimes of Big Ben, the famous clock tower, every quarter of an hour.

Via additional hardware connected to a ping pong circuit board with simple components, the melodies are played on the so-called ‘Simple SD Audio Player’.

The ‘Simple SD Audio Player’ can play WAV files from an SD card, and requires only an ATtiny85 with a few external components. It has been modified so that via an I2C bus, a certain sound can be played back. A total of five WAV files are required to play Big Ben.

**Additional hardware required**

a) One extra button (1-pole close contact), connected between PortD.3 (i.e. solder point D3) and GND. The necessary pull-up resistance is triggered in the controller internally.

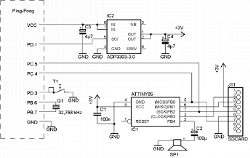
b) Watch crystals with 32,768 kHz must be soldered to pins PB6 and PB7. So that the oscillator in Atmega8 becomes active, the fusebit CKOPT must be set. But the clocking of the ATmega8 remains unchanged thanks to the internal 8 MHz oscillator, the 32,768 kHz quartz only clocks Timer2.

c) Components for 'Simple SD Audio Player': ATTiny85, SD-card holder, 4 capacitors, speakers, voltage regulator ADP3303-3.0

A circuit diagram can be found in the file bingbong\_schaltplan.png.

Because the SD card works with a voltage level between 2.7V to 3.3V, a 3.0V LowDrop linear voltage regulator ADP3303-3.0 is used, which has an additional shutdown pin. If shutdown is drawn on GND, then the 3V are switched off and the voltage regulator in standby mode only uses a few µA. The shutdown pin is connected to the port pin D.1 of the Ping Pong circuit board, so that the Atmega8 can switch the power supply for the Simple SD Audio Player on and off.

Because only a minimum number of components is intended to suffice, the low-pass filter that is actually required on the PWM output of the ATtiny85 has been done away with. Instead, a speaker has been connected directly via an electrolytic capacitor. Please note that no amplifier and no active speakers should be connected, as the lack of a low-pass filter means that these could be destroyed.

[](http://www.elo-web.de/ximage/1003bingbong2.jpg)

**Simple SD Audio Player**

The Attiny85 reads the data of a WAV file from the SD card and generates a sound output via an integrated PWM unit.

Files in RIFF-WAVE format/LPCM, 8/16-bit, mono/stereo and up to a sampling rate of 48kHz are supported.

Despite the small amount of hardware, surprisingly good sound quality is achieved if the corresponding WAV files are of good quality, e.g. a sampling rate of 44kHz.

The original project, with its description, was conceived by a man Japan and can be found here:**<http://elm-chan.org/works/sd8p/report.html>**

The software has been modified so that using an I2C bus, a certain sound can be played back from the SD card. Five sounds are required to do this: sound files 1-4 are the sounds for quarter past, half past, quarter to and on the hour; sound file 5 is the sound that comes after sound 4 on the hour and is played as a chime x number of times and is the ‘bong’. Thus the Simple SD Audio Player is a I2C slave, that has been realised via USI. As the I2C pins of the ATtiny85 are already in use to drive the SD card, a switch has been made via the software: after PowerOn, these are switched as I2C and there is a wait until the sound number has been received. Then the pins are used as an SPI interface to drive the SD card. The Atmega8 of the Ping Pong circuit board is the I2C master and uses the on-chip I2C interface. The required pullups for I2C are triggered in the ATmega8 internally. After the sound number has been transferred, the I2C interface in the Atmega8 is switched off and both pins SCL and SDA are activated as inputs, as level changes are no longer present on the CLK line of the SD card.

The ZIP file has a subfolder containing the bingbong\_attiny85, the hex file sd8p\_mo.hex, as well as a screenshot – attiny85\_fusebits.jpg – with the necessary fusebits for the ATtiny85. Programming the ATtiny85 can be done via ISP. To do this, no SD card should be inserted to do this yet.

After switching on the power, the clock display appears: a circle with 12 LEDs to symbolise the hours, whereby the current hour is displayed in full brightness and the other 11 LEDs are dimmed. Inside the circle, the minutes are displayed as a dimmed two-digit decimal value. Immediately after switching on, the clock starts with that time at 0:00.

By briefly pressing the operating button, the decimal display can be switched between minutes and seconds.

Pressing the operating button for longer engages the mode to set the time. The button must be held down until the decimal display inside the circle disappears. The clock is stopped and the hour value set by the right potentiometer is displayed with full brightness. The potentiometer can now be used to set the current hour value. To do this, the setting range is divided into 12 steps. Then pressing the button for a short time confirms this.  
In the centre of the circle, corresponding to both potentiometers, the minutes set are displayed in full brightness. Using the left potentiometer above the coin slot, the tens digit (0-5) can be set; with the right potentiometer used to set the ones (0-9) of the minute value. The latter should be set to the current minute plus 1 minute. The selection is once again confirmed by holding down the button for a short time.  
In the decimal display, the value 00 is displayed and is dimmed, denoting the seconds. If the minute value set previously has been reached, then this must be confirmed by holding the button down for a short time. Releasing the button starts the clock according to the hour and minute values set, with the seconds value at zero. This completes the procedure of setting the time. In the decimal display, the seconds are displayed; holding the button down for a short time changes this to display the minutes.

Every quarter of an hour a sound is played corresponding to the sound files 1-4 (quarter past, half past, quarter to and on the hour) on the SD card. Every hour on the hour sound file 5 is played as a chime x number of times according to the hour value to indicate the current hour.

**Sound files**

Five sound files are required on the SD card: bigben1.wav, bigben2.wav, bigben3.wav, bigben4.wav and bong.wav  
bigben1-4.wav contain the sounds for quarter past, half past, quarter to and on the hour; bong.wav contains the sound for the chime that sounds x number of times according to the current hour value.

The files must be in the following format: RIFF-WAVE format/LPCM, 8/16-bit, mono/stereo and have a sampling rate of up to 48 kHz.

When playing the files in the Simple SD Audio Player, the files are numbered in the order they were copied. The best method is to copy the five files to the SD card individually in the following order: bigben1.wav to bigben4.wav and then bong.wav. By playing your own files, you could, for example, have the sound of a cuckoo clock playing.

Download: **All sound files**

**Source code**

The code for the ATmega8 on the Ping Pong circuit board was created with the C-Compiler CodeVision AVR (Version 2.04.6 Evaluation). This Evaluation version may be used free of charge for private, non-commercial purposes and limited to a code size of 3KB, which is sufficient for this application. The zip file contains the entire project (the project file for CodeVison AVR is bingbong.prj).

The code for the Simple SD Audio Player on the ATtiny85 was created with the freeware WinAVR (Version WinAVR-20100110.exe). For this purpose, the original code for the Bing Bong project has been modified so that after PowerOn, there is an initial waiting period until the number of the sound to be played back (WAV file) has been received. Then, if it is available, the desired sound is played back and then stopped.

Download: [**Source texts and hex files**](http://www.elo-web.de/xattachment/1003BingBong.zip)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Compiler : CodeVisionAVR 2.04.6 Evaluation  
Chip type : ATmega8  
Clock frequency : 8 MHz (int. RC-OSC)  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Bing Bong – a clock with a Big Ben melody  
// based on Ping Pong by Franzis  
// http://www.elo-web.de/ping-pong-start  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <mega8.h>  
#include <stdio.h>  
#include <delay.h>  
#include <sleep.h>

#define P\_CLK PORTB.3  
#define P\_DATA PORTB.4  
#define P\_STROBE PORTB.2

#define PIN\_COIN PIND.2  
#define PULLUP\_COIN PORTD.2  
#define DIR\_COIN DDRD.2  
#define COIN\_INSERTED (PIN\_COIN == 0)

#define PIN\_KEY PIND.3  
#define PULLUP\_KEY PORTD.3  
#define KEY\_PRESSED (PIN\_KEY == 0)

#define P\_SHUTDOWN\_SD PORTD.1 // Portpin for /Shutdown of SD card player  
#define DP\_SHUTDOWN\_SD DDRD.1  
#define SD\_PLAYER\_OFF (P\_SHUTDOWN\_SD = 0)  
#define SD\_PLAYER\_ON (P\_SHUTDOWN\_SD = 1)

#define P\_SDA PORTC.4 // Portpin for SDA  
#define DP\_SDA DDRC.4  
#define PIN\_SDA PINC.4

#define P\_SCL PORTC.5 // Portpin for SCL  
#define DP\_SCL DDRC.5  
#define PIN\_SCL PINC.5

#define WIDTH 12 // number of fields in horizontal direction  
#define HEIGHT 10 // number of fields in vertical direction

unsigned char ledState[WIDTH\*HEIGHT]; // current state of each LED

enum { LED\_OFF = 0, LED\_ON, LED\_DIM }; // possible states of a LED  
#define BLINKING 0x80 // to set a LED blinking, it is ORed with 0x80

unsigned char seconds = 0; // current time: seconds  
unsigned char minutes = 0; // current time: minutes  
unsigned char hours = 0; // current time: hours  
bit bSound = 0; // 1=play sound

flash unsigned char ledHours[12] = // LED numbers for all 12 hours (0..11)  
{  
50, 81, 103, 105, 107, 89, 59, 29, 7, 5, 3, 21  
};

flash unsigned char decimals[] =  
{ // Font for decimal values  
0x1F, 0x11, 0x1F, // 0  
0x04, 0x02, 0x1F, // 1  
0x1D, 0x15, 0x17, // 2  
0x15, 0x15, 0x1F, // 3  
0x07, 0x04, 0x1F, // 4  
0x17, 0x15, 0x1D, // 5  
0x1F, 0x15, 0x1D, // 6  
0x01, 0x01, 0x1F, // 7  
0x1F, 0x15, 0x1F, // 8  
0x17, 0x15, 0x1F // 9  
};

// clear the whole screen, all LEDs off  
void clearScreen(void)  
{  
unsigned char i;  
  
for(i = 0; i < (WIDTH\*HEIGHT); i++)  
{  
ledState[i] = LED\_OFF;  
}  
}

// wait until key is pressed  
void waitUntilKeypressed(void)  
{  
while(!KEY\_PRESSED);  
delay\_ms(20);  
}

// wait until key is released  
void waitUntilKeyreleased(void)  
{  
while(KEY\_PRESSED);  
delay\_ms(20);  
}

// check if key is pressed  
// return: 0=key not pressed, 1=key pressed  
unsigned char keyPressed(void)  
{  
unsigned char result = 0;  
  
if(KEY\_PRESSED)  
result = 1;  
delay\_ms(20);  
  
return result;  
}

// Timer 0 overflow interrupt service routine  
// multiplexing of LEDs  
interrupt [TIM0\_OVF] void timer0\_ovf\_isr(void)  
{  
#define FRAME\_MASK 0x03  
static unsigned char count = 0;  
static unsigned char framecount = 0;  
static unsigned int blinkcounter = 0;  
static bit bBlinkOn = 1;  
bit bLed;  
unsigned char state, i, value1, value2, value3;  
unsigned char \*pState;  
  
blinkcounter++;  
if(blinkcounter == 1000)  
{  
bBlinkOn = ~bBlinkOn;  
blinkcounter = 0;  
}  
  
count++;  
if(count >= 12)  
{  
count = 0;  
P\_DATA = 0;  
framecount++;  
}  
else  
{  
P\_DATA = 1;  
}  
P\_CLK = 1;  
P\_CLK = 0;  
  
pState = &ledState[count\*HEIGHT];  
  
value1 = 0;  
for(i = 0; i < 4; i++)  
{  
value1 >>= 1;  
state = (\*pState) & (~BLINKING);  
bLed = 0;  
if(state == LED\_ON)  
{  
bLed = 1;  
}  
else if(state == LED\_DIM)  
{  
if((framecount & FRAME\_MASK) == FRAME\_MASK)  
bLed = 1;  
}  
if((\*pState) & BLINKING)  
{  
if(!bBlinkOn)  
bLed = 0;  
}  
if(bLed)  
{  
value1 |= 0x08;  
}  
pState++;  
}

value2 = 0;  
for(i = 0; i < 4; i++)  
{  
value2 >>= 1;  
state = (\*pState) & (~BLINKING);  
bLed = 0;  
if(state == LED\_ON)  
{  
bLed = 1;  
}  
else if(state == LED\_DIM)  
{  
if((framecount & FRAME\_MASK) == FRAME\_MASK)  
bLed = 1;  
}  
if((\*pState) & BLINKING)  
{  
if(!bBlinkOn)  
bLed = 0;  
}  
if(bLed)  
{  
value2 |= 0x80;  
}  
pState++;  
}

value3 = 0;  
for(i = 0; i < 2; i++)  
{  
value3 >>= 1;  
state = (\*pState) & (~BLINKING);  
bLed = 0;  
if(state == LED\_ON)  
{  
bLed = 1;  
}  
else if(state == LED\_DIM)  
{  
if((framecount & FRAME\_MASK) == FRAME\_MASK)  
bLed = 1;  
}  
if((\*pState) & BLINKING)  
{  
if(!bBlinkOn)  
bLed = 0;  
}  
if(bLed)  
{  
value3 |= 0x02;  
}  
pState++;  
}  
  
PORTC &= (~0x0F);  
PORTD &= (~0xF0);  
PORTB &= (~0x03);  
P\_STROBE = 1;  
P\_STROBE = 0;  
PORTC |= (value1);  
PORTD |= (value2);  
PORTB |= (value3);  
  
}

// initialization of Timer0  
void initTimer0(void)  
{  
// Timer/Counter 0 initialization  
// Clock source: System Clock  
// Clock value: 1000,000 kHz  
TCCR0=0x02;  
TCNT0=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization  
TIMSK |= 0x01;  
}

// Timer 2 overflow interrupt service routine  
// clock interrupt  
// occurs every 1 second  
interrupt [TIM2\_OVF] void timer2\_ovf\_isr(void)  
{  
if(seconds < 59)  
{  
seconds++;  
}  
else  
{  
seconds = 0;  
if(minutes < 59)  
minutes++;  
else  
{  
minutes = 0;  
if(hours < 23)  
hours++;  
else  
hours = 0;  
}  
if((minutes % 15) == 0)  
{ // play sound every 15 minutes  
bSound = 1;  
}  
}  
}

// initialization of Timer2  
void initTimer2(void)  
{  
// Timer/Counter 2 initialization  
// Clock source: TOSC1 pin  
// Clock value: PCK2/128  
// Mode: Normal top=FFh  
// OC2 output: Disconnected  
ASSR=0x08;  
TCCR2=0x05;  
TCNT2=0x00;  
OCR2=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization  
TIMSK |= 0x40;  
}

// stop Timer2  
void stopTimer2(void)  
{  
TCCR2=0x00;  
TCNT2=0x00;  
}

// External Interrupt 0 service routine  
interrupt [EXT\_INT0] void ext\_int0\_isr(void)  
{  
}

// initialization of INT0  
void initInt0(void)  
{  
// External Interrupt(s) initialization  
// INT0: disabled  
// INT0 Mode: Low level  
GICR &= (~0x40); //disable INT0  
MCUCR &= 0xFC;  
GIFR=0x40;  
}

// initialization of ports  
void init(void)  
{  
PORTB = 0;  
DDRB = 0x1F;  
PORTC = 0;  
DDRC = 0x0F;  
PORTD = 0;  
DDRD = 0xF4;  
PULLUP\_KEY = 1;  
P\_SHUTDOWN\_SD = 0;  
DP\_SHUTDOWN\_SD = 1;  
// Analog Comparator initialization  
// Analog Comparator: Off  
// Analog Comparator Input Capture by Timer/Counter 1: Off  
ACSR=0x80;  
SFIOR=0x00;  
}

#define ADC\_VREF\_TYPE 0x00

// initialization of ADC  
void initADC(void)  
{  
// ADC initialization  
// ADC Clock frequency: 125,000 kHz  
// ADC Voltage Reference: AREF pin  
ADMUX=ADC\_VREF\_TYPE & 0xff;  
ADCSRA=0x86;  
}

// Read the AD conversion result  
// adc\_input: ADC channel (0..7)  
// return: ADC value (0..1023)  
unsigned int read\_adc(unsigned char adc\_input)  
{  
ADMUX=adc\_input | (ADC\_VREF\_TYPE & 0xff);  
// Delay needed for the stabilization of the ADC input voltage  
delay\_us(10);  
// Start the AD conversion  
ADCSRA|=0x40;  
// Wait for the AD conversion to complete  
while ((ADCSRA & 0x10)==0);  
ADCSRA|=0x10;  
  
return ADCW;  
}

// select minute lower digit  
// return: selected minute lower digit (0...9)  
unsigned char selectMinutes1(void)  
{  
unsigned char x;   
  
x = 9 - (read\_adc(7) / (1023/10 + 1));

return(x);  
}

// select minute upper digit  
// return: selected minute upper digit (0...5)  
unsigned char selectMinutes10(void)  
{  
unsigned char x;   
  
x = (read\_adc(6) / (1023/6 + 1));  
  
return(x);  
}

// select hour  
// return: selected hour (0...11)  
unsigned char selectHour(void)  
{  
unsigned char x;   
  
x = 11 - (read\_adc(7) / (1023/12 + 1));  
  
return(x);  
}

// initialize I2C interface  
void initI2C(void)  
{  
// Two Wire Bus initialization  
// Bit Rate: 25,000 kHz  
TWBR=0x98;  
// Two Wire Bus Slave Address: 0h  
// General Call Recognition: Off  
TWAR=0x00;  
// Generate Acknowledge Pulse: On  
// TWI Interrupt: Off  
TWCR=0x44;  
TWSR=0x00;   
}

// disable I2C interface  
void disableI2C(void)  
{  
TWCR=0x00;  
TWSR=0x00;  
}

#define TWINT 7  
#define TWSTA 5  
#define TWSTO 4  
#define TWEN 2

// I2C start condition  
void i2cStart(void)  
{  
TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);  
  
while (!(TWCR & (1<<TWINT)));  
  
}

// I2c stop condition  
void i2cStop(void)  
{  
TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);  
}

// I2C send a byte  
// c: byte to send  
void i2cSend(unsigned char c)  
{  
TWDR = c;  
TWCR = (1<<TWINT) | (1<<TWEN);

while (!(TWCR & (1<<TWINT)));  
}

// show decimal value on display  
// d: value to be displayed (0...99)  
// state: display as dimmed or ON (LED\_DIM or LED\_ON)  
void showDecimal(unsigned char d, unsigned char state)  
{  
unsigned char i, j, k, value, col, divisor, pattern;  
  
col = 2;   
divisor = 10;  
for(k = 0; k < 2; k++)  
{  
value = d / divisor;  
for(j = 0; j < 3; j++)  
{ // show decimal using stored font  
pattern = decimals[value\*3 + j];  
for(i = 0; i < 5; i++)  
{   
if( pattern & (0x01 << i) )  
ledState[col\*HEIGHT + 3 + i] = state;  
else  
ledState[col\*HEIGHT + 3 + i] = LED\_OFF;  
}   
col++;  
}   
col++;  
d = d % divisor;  
divisor = divisor / 10;  
}

}

// show clock on display:   
// 1 LED for each hour, current hour LED is ON, other 11 LEDs are dimmed  
// minutes or seconds are displayed as 2 digits decimal  
// cShowMinutes: 1=show minutes, 0=show seconds  
// cDecimalState: display decimals as dimmed or ON (LED\_DIM or LED\_ON)  
void showClock(unsigned char cShowMinutes, unsigned char cDecimalState)  
{  
unsigned char i, cValue, cHour, state;  
  
if(cShowMinutes)  
cValue = minutes;  
else  
cValue = seconds;

cHour = hours;  
if(cHour >= 12)  
cHour -= 12;   
  
for(i = 0; i < 12; i++)  
{  
if(i == cHour)  
state = LED\_ON;  
else  
state = LED\_DIM;  
ledState[ledHours[i]] = state;  
}   
showDecimal(cValue, cDecimalState);   
  
}

// main routine  
void main(void)  
{  
unsigned char d, d10, index=0, i, bongCounter=0;  
unsigned int iCountSCL;  
bit bShowMinutes = 1;  
bit bCheckSoundEnd = 0;  
  
init();  
initTimer0();  
initTimer2();  
initADC();  
initInt0();

clearScreen();

#asm("sei") // Global enable interrupts

while(1)  
{  
if(KEY\_PRESSED)  
{  
delay\_ms(20);   
i = 50;   
while( keyPressed() && (i != 0) ) // check for short or long keypress  
{  
i--;  
}  
if(i == 0)  
{ // long keypress => set time  
stopTimer2();  
showClock(1, LED\_OFF);  
waitUntilKeyreleased();

do  
{  
d = selectHour(); // select hour (0-11)  
hours = d;  
showClock(1, LED\_OFF);  
}  
while(!keyPressed());  
waitUntilKeyreleased();

do  
{  
d10 = selectMinutes10(); // select minutes upper digit (0-5)  
d = selectMinutes1(); // select minutes lower digit (0-9)  
minutes = d10\*10 + d;  
showClock(1, LED\_ON);  
}  
while(!keyPressed());  
waitUntilKeyreleased();

seconds = 0;  
showClock(0, LED\_DIM);  
bShowMinutes = 0; // show seconds in display after setting clock  
waitUntilKeypressed();  
waitUntilKeyreleased();  
initTimer2();  
}   
else  
{ // short keypress => toggle between display of minutes and seconds  
bShowMinutes = ~bShowMinutes;  
}  
}  
  
showClock(bShowMinutes, LED\_DIM); // show current time on display  
  
if(bSound) // check if sound has to be played  
{   
index = minutes / 15; // sound 1-4: 1/4, 1/2, 3/4, 4/4 hour   
if(index == 0)  
{  
index = 4;  
}  
bSound = 0;  
if(index == 4) // each hour, "bong" current hours  
{  
bongCounter = hours;  
if(bongCounter > 12)  
bongCounter -= 12;  
if(bongCounter == 0)  
bongCounter = 12;  
}  
}  
  
if(index) // check if SD card player has to be started  
{   
P\_SCL = 1; // enable pullups for I2C  
P\_SDA = 1;  
  
SD\_PLAYER\_ON;

delay\_ms(100);  
initI2C();  
i2cStart();  
i2cSend(0xAA); // I2C address  
i2cSend(index); // I2C data  
i2cStop();

disableI2C();

delay\_ms(50);  
index = 0;  
iCountSCL = 0;  
bCheckSoundEnd = 1;  
}  
  
if(bCheckSoundEnd)  
{ // check for sound end, if SCL is permanent HIGH  
if(PIN\_SCL)  
{  
iCountSCL++;  
}  
else  
{  
iCountSCL = 0;  
}   
  
if(iCountSCL >= 1000)  
{  
bCheckSoundEnd = 0;  
  
if(bongCounter)  
{   
index = 5; // sound 5: "bong"  
bongCounter--;  
}  
else  
{  
P\_SCL = 0; // disable pullups for I2C  
P\_SDA = 0;  
  
SD\_PLAYER\_OFF; // switch OFF SD card player  
}   
}  
}  
  
}   
}