

AVR TV Tennis

Play MahPong, the Single-Chip Edition!

Design by M. Hasenstab

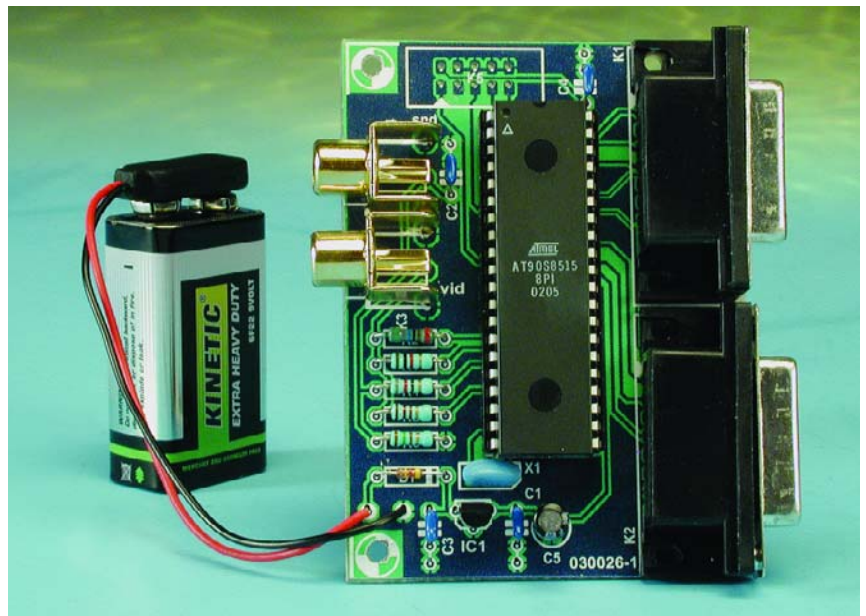
MAH@jkdesign.de

The circuit described here employs just a single microcontroller to put the classic MahPong video game (a.k.a. TV Tennis) on the TV screen. Guaranteed 1970's nostalgia!

In this day and age of gigahertz PCs with massive amounts of user memory, only extremely extensive games using all the latest 3-D video effects and ditto sounds appear to satisfy the younger generation. If such a game runs too slow, well that simply means your PC is sluggish (again) and needs to be replaced a.s.a.p. only to discover adverts telling you that the machine you've just unpacked from the box is... outdated! Forever gone are days of the first home computers for which highly optimised software was written that pushed computing power to levels never dreamed of when these early machines were first sold.

The challenge and excitement of the present project is in its minimalist approach to obtain, in a word, nostalgia. Today, instead of a box full of complex, purpose-designed hardware, a single 8-bit AVR microcontroller is pushed for maximum performance. The original version of the TV Tennis game, published in Elektor November 1976, contained 13 TTL ICs and 50-odd discrete parts. Follow-up articles published in 1976 added dozens more ICs that reportedly made the power supply too hot to touch, at least in the prototype used for lunchtime amusement in the Elektor lab.

Returning to 2003, the only storage capacity available on the AVR chip we'll be using is 8 kbytes of program memory and 512 bytes of RAM. The video signals are mixed together from a couple of port pins and resistors, while a Timer is used to generate the primitive sounds used in the MahPong game. During the dark (actually, blanked) periods at the top and bottom edges of the screen, the vexed microcontroller is allowed to handle the rules of the game, interpret the user input, as well as do various chores. Nothing even vaguely



resembling an operating system or similar imposes itself between the object code burned into the micro and the naked silicon on the chip!

The rules of the MahPong game probably only require clarification to the younger among Elektor readers. Each player employs a bat to bounce a ball back to the opponent. If the ball gets past your bat and bounces against the wall behind it, your opponent gains one point. The score is prominently displayed in the top left hand corner of the screen. The first player to reach 21 points is the winner and is awarded a little ceremony consisting of a victory tune!

Hardware? What hardware?

The extremely simple circuit diagram shown in **Figure 1** is typical for a turnkey microcontroller application like the MahPong game. As always in these cases, daunting functions become possible thanks to ingenious software and the functional blocks available inside the microcontroller. Consequently, hardly anything appears in the way of external components.

Resistors R1, R2 and R3 between the AVR micro and connector K3, together with the 75-Ω terminating

resistance inside the TV set, form a potential divider. The 90S8515 micro is capable of supplying about 18 mA through its port pins, although it must be noted that due to its internal resistances, the output voltage is rather current-dependent. If you want to calculate accurately, you first have to establish the currents required for a number of discrete signal levels to appear across the 75- Ω input impedance represented by the CVBS (composite video input) on the monitor or TV set. Such calculations yield the following results:

R1: Sync

Black level: 0.3 V

Current from Sync Pin 14:

$$0.3 \text{ V} / 75 \Omega = 4 \text{ mA}$$

Output voltage at Port:

$$4.5 \text{ V} \text{ at } 4 \text{ mA}$$

Required total resistance:

$$4.5 \text{ V} / 4 \text{ mA} = 1125 \Omega$$

Series resistor:

$$1125 \Omega - 75 \Omega = 1050 \Omega (1 \text{ k}\Omega)$$

R2: PixelOut1

Level for bright grey: 0.8 V

Current from Pin 28:

$$(0.8 \text{ V} - 0.3 \text{ V}) / 75 \Omega = 6.7 \text{ mA}$$

Output voltage at Port:

$$4.0 \text{ V} \text{ at } 6.7 \text{ mA}$$

Required total resistance:

$$4 \text{ V} / 6.7 \text{ mA} = 600 \Omega$$

Series resistor:

$$600 \Omega - 75 \Omega = 525 \Omega (576 \Omega, 1\%)$$

R3: PixelOut0

Level for dark grey: 0.5 V

Current from Pin 24:

$$(0.5 \text{ V} - 0.3 \text{ V}) / 75 \Omega = 2.7 \text{ mA}$$

Output voltage at Port:

$$4.6 \text{ V} \text{ at } 2.7 \text{ mA}$$

Required total resistance:

$$4.6 \text{ V} / 2.7 \text{ mA} = 1700 \Omega$$

Series resistor:

$$1700 \Omega - 75 \Omega = 1625 \Omega (1.2 \text{ k}\Omega)$$

This value causes dark grey to appear a little brighter but still dis-

tinguishable from bright grey.

Potential divider R4/R5 on the sound output of the AVR chip reduces the voltage level from 'digital' (5 V) to about 1 volt. This allows a sufficient volume range on the TV set or monitor. If you want a slightly less aggressive sound, then capacitor C2 may be connected in parallel with R5.

Digital joysticks are connected to sockets K1 and K2. A digital joystick has a switch for each direction. When actuated, each switch connects its own control line to ground. Together with resistors integrated in the AVR micro, these lines allow joystick movements to be recognised reliably.

Interface connector K5 is a 10-way box-header of the same type used on AVR evaluation boards. It allows the microcontroller to be programmed in-circuit using one of the well-known AVR programmers.

Power-wise the microcontrollers from the Atmel AVR series are marked by a rather Spartan lifestyle. The 78L05 voltage regulator in the circuit reduces the battery input voltage to +5 V. The circuit draws less than 30 mA which is easily furnished by a 9-volt battery. An 8-MHz ceramic resonator is used to help generate the microcontroller clock signal. If you happen to have a low-power stabilised 5-V supply available, then the 78L05 may be omitted from the circuit.

Powerful software

The software developed for the project may be thought of as consisting of the following modules:

Reset and initialisation routine

This module initialises the directions and levels of the I/O ports, plus organises the controller's inner life.

Main program

This bit of software not only awaits user action for the game selection, but also calls the relevant game routines while the game is being played.

Play routine PLAYPONG

This module is synchronised to the picture output and on being called first waits for the VSYNC signal that marks the start of a new picture. Once VSYNC is present, the players' input is examined to enable the position of the bats to be computed. Next, the routine searches for obstacles in the trajectory of the ball. If none is found, the ball keeps moving along a straight line. If the ball encounters an object, it bounces back into the playing field

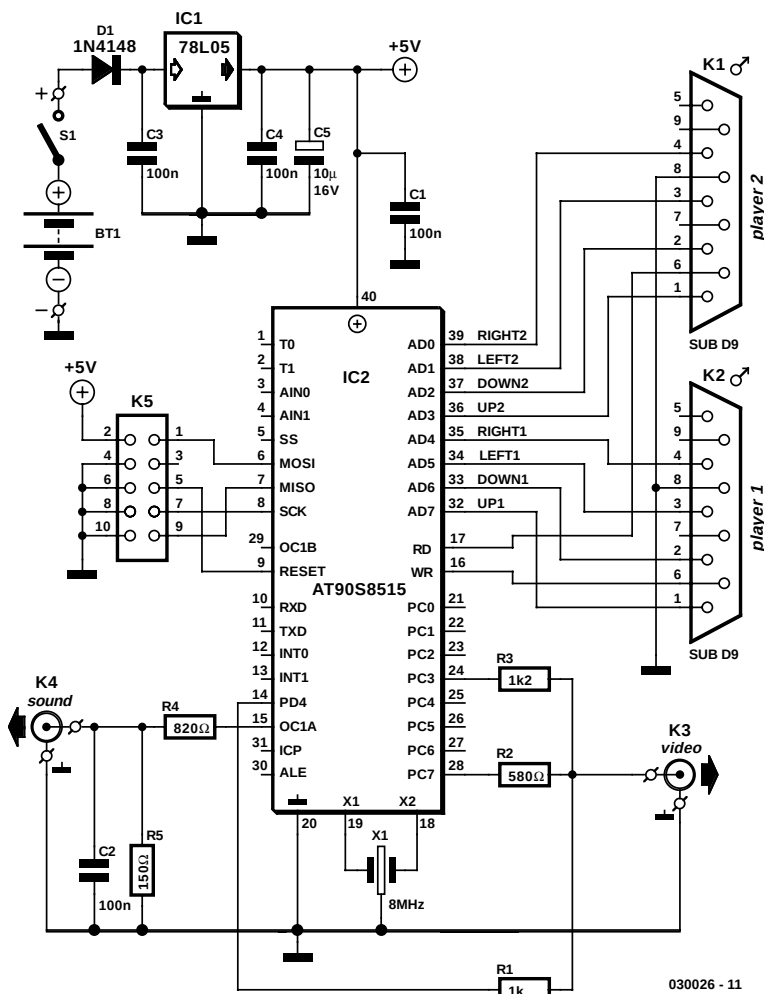


Figure 1. The minimalist approach: circuit diagram of the TV Tennis game, 2003-style.

according to the simple rule *angle of incidence = angle of reflection* and moves on in the new direction. Also, the software checks if the ball touched the left-hand or right-hand wall. If so, the score is updated. Each time the ball is in contact with a bat or a wall, an object-specific sound is generated. Once a player reaches the number of points required to win the game, a “Winner” screen is displayed and the game is ended.

TMRO Interrupt routine

This piece of software for the video signal generating routines runs quasi in parallel with the main program. The timing of the system is crucial to the way the video generation works. For example, the video routine may never be interrupted or called with a delay. Consequently, only one interrupt routine is allowed (and that's the video generator proper), and the rest of the user program actually has its available computing time allotted by the video routine.

In addition to these larger modules we have:

Read/Write utilities for operations in the picture memory.

Sound utilities for starting sounds, playing and stopping them.

Composite video

The CVBS signal (composite video, blanking, sync) is intended to generate a monochrome image and forms the basis of extensions and standards defining colour TV, like PAL and NTSC. A CVBS picture may have 625 lines with a duration of $64\mu\text{s}$ each. A complete picture takes 40 ms to build, which implies a frame frequency of 25 Hz. Interlacing is used to reduce picture flicker — in this system the lines of successive rasters are not superimposed on one another but are interlaced, two rasters constituting one picture or ‘frame’ (**Figure 2**).

The CVBS signal employs just one wire to carry synchronisation as well as brightness information. The instantaneous level of a CVBS signal is between 0 V and 1 V. The required terminating resistance of 75Ω is contained in the monitor or TV set. The rear porch (black reference level) is defined as 0.3 V, so all levels of grey, right up to white, are comprised between 0.3 V and 1 V. The monitor interprets an instantaneous level of 0 V as the sync pulse. The horizontal sync pulse (HSYNC) lasts $4.7\mu\text{s}$ and marks the start of a picture line. Likewise, a $160\mu\text{s}$ long signal of 0 V is taken to mean the vertical sync pulse. Each half image consists of 312.5

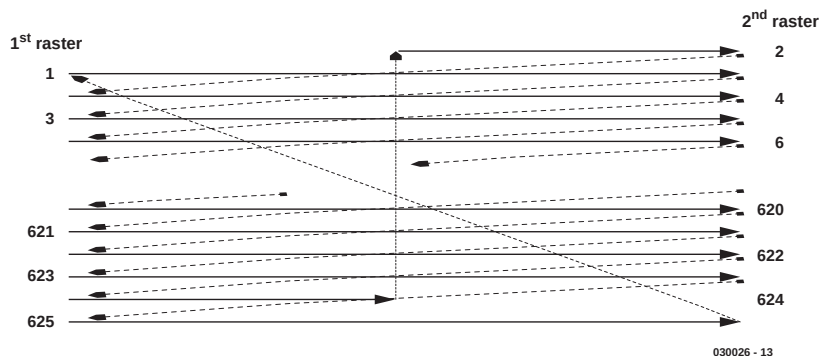


Figure 2. The CVBS video signal.

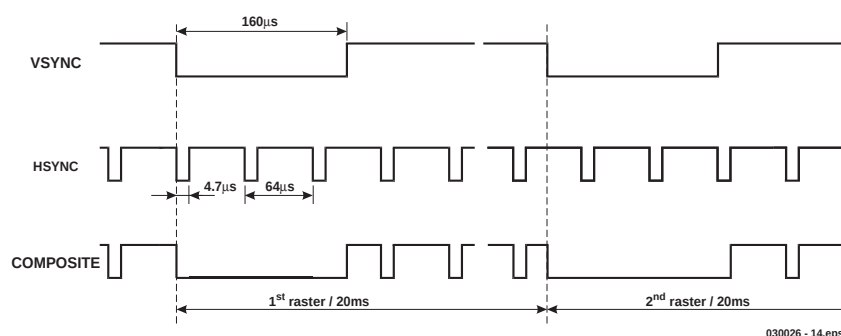


Figure 3. Composite sync signal.

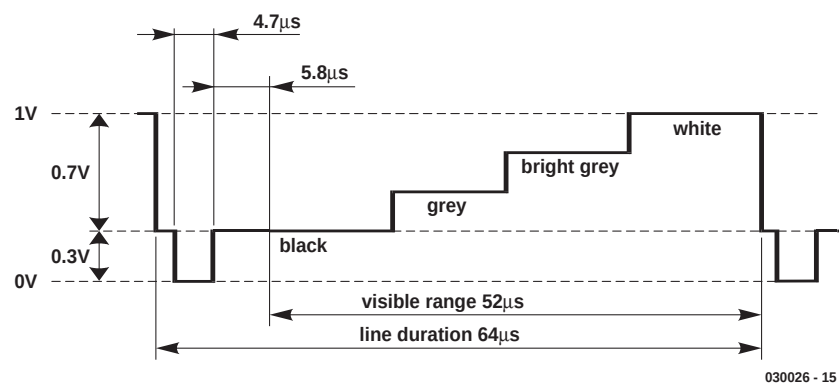


Figure 4. CVBS picture line signal.

lines. The timing diagram in **Figure 3** shows that VSYNC and HSYNC coincide for the first half image, but are offset by $32\mu\text{s}$ for the second half. This allows the monitor to discriminate between the first and second half image and display them correctly.

An example of the contents of a CVBS video line is shown in **Figure 4**. The line starts with the rear porch (i.e., the invisible area to the right of the image). The appearance of the HSYNC pulse tells the monitor that a new picture line is about to

start. Next comes the front porch (left-hand invisible area). This signal level is the reference for the darkest (blackest) content of the upcoming video signal. The active range contains the visible contents of the line (here, a staircase signal).

Video signal generation inside the AVR

The graphics system, which does most of its work largely unnoticed by the user program, requires a significant portion of the available com-

```

mov    HCOUNT, XL          ; shift Low Byte of picture memory pointer to
subi   HCOUNT, -13         ; HCOUNT and add 13 to value

                                ; cycles; comment
ld     REGA, X+              ; 2 ; Load picture data from SRAM into REGA
                                ; ; The following code takes 16*12 cycles: 192 cycles = 24 us
STREAMLINE_SHIFTER:
out     PIXELPORT, TMPL_I    ; 1 ; Write pixel 0 onto pixel port (PORTC)
                                ; ; note that only PC7 (bright grey) and
                                ; ; PC3 (dark grey) are used, hence
                                ; ; any values may be present on other pins.
mov     REGB, REGA           ; 1 ; Copy REGA into register REGB
lsl     REGB                  ; 1 ; Shift REGB 1 bit left
nop                                           ; 1 ; Idle

out     PIXELPORT, REGB      ; 1 ; Write pixel onto pixel port
lsl     REGB                  ; 1 ; Shift REGB left
nop                                           ; 1
nop                                           ; 1

out     PIXELPORT, REGB      ; 1 ; Write pixel onto pixel port
ld     REGA, X+              ; 2 ; Load new pixel data into REGA
lsl     REGB                  ; 1 ; Shift REGB left

out     PIXELPORT, REGB      ; 1 ; Write pixel onto pixel port
cp      VRPOINTL, HCOUNT    ; 1 ; Compare picture memory pointer with
                                ; ; byte counter
brne    STREAMLINE_SHIFTER   ; 2 ; If unequal, go to STREAMLINE_SHIFTER'
; end shifter;

```

Figure 5. Source code snippet.

puting power. If, however, the picture 'assembly' is assigned to an interrupt routine, the user program is simply 'pushed aside', without noticing the serious number crunching that's done in the background.

We will first have a look at the picture memory. The AVR has available just 512 bytes of RAM which, we're glad to note, is still sufficient to store 48×30 pixels with up to four grey levels (2-bit coding: 00 = black; 01 = dark grey; 10 = bright grey; 11 = white). In fact, the RAM has a spare capacity of 152 bytes for use as a stack area and for variables storage. The image memory is organised as illustrated in **Table 1**.

Counting starts in the left-hand top corner at coordinate 0/0. One byte always contains four pixels, with the High/Low values repre-

sending the brightness distributed across the two nibbles. The picture memory is cyclic (continuous).

The active (visible) part of a video line lasts 52 μ s. Consequently, at a horizontal resolution of 48 pixels, only 1 μ s is available per pixel. Because the AVR micro has a cycle time of 125 ns, the following tasks should be completed within eight cycles: (1) establish picture memory address; (2) load pixel brightness information from picture memory; (3) set appropriate port pins. The piece of assembly code shown in **Figure 5** arguably presents the fastest way of effectively transferring pixel data to a port pin. The code shows how different execution times of individual instructions are combined to write exactly one pixel in each 4-cycle block. This code snippet proves that

the AVR micro is capable of continuously displaying a pixel every four machine cycles (500 ns). Theoretically this 'throughput' would allow a horizontal resolution of 104 pixels. To obtain the desired (lower) resolution, four NOPs (no-operation instructions) have been added to each read/write block to slow down the shift logic to 1 μ s per pixel.

Of course, the vertical resolution could be the number of visible lines that appear on the display. However the available memory being limited, we need to leave this at 48×30 pixels and perhaps aim for higher resolution in a future project. Here, the pixels are squares — eight cycles (1 μ s) wide and 16 lines high. The upshot is that the visible image has a width of 48 μ s (of 52 μ s) and a height of 480 lines (of 625). The resulting blank areas around the picture allow the game to appear reliably on most, if not all, (TV) screens and monitors.

Having solved the time-critical aspect of transferring the video signal, we still need to

add the synchronisation signals to the pixel data. A look at **Figure 6** shows that during the video line the processor is fully occupied with the pixel data and has no time for other tasks. Conse-

Table 1. Picture memory contents

BYTE0:	H00,00	H01,00	H02,00	H03,00	L00,00	L01,00	L02,00	L03,00
BYTE1:	H04,00	H05,00	H06,00	H07,00	L04,00	L05,00	L06,00	L07,00
...								
BYTE359:	H44,29	H45,29	H46,29	H47,29	L44,29	L45,29	L46,29	L47,29

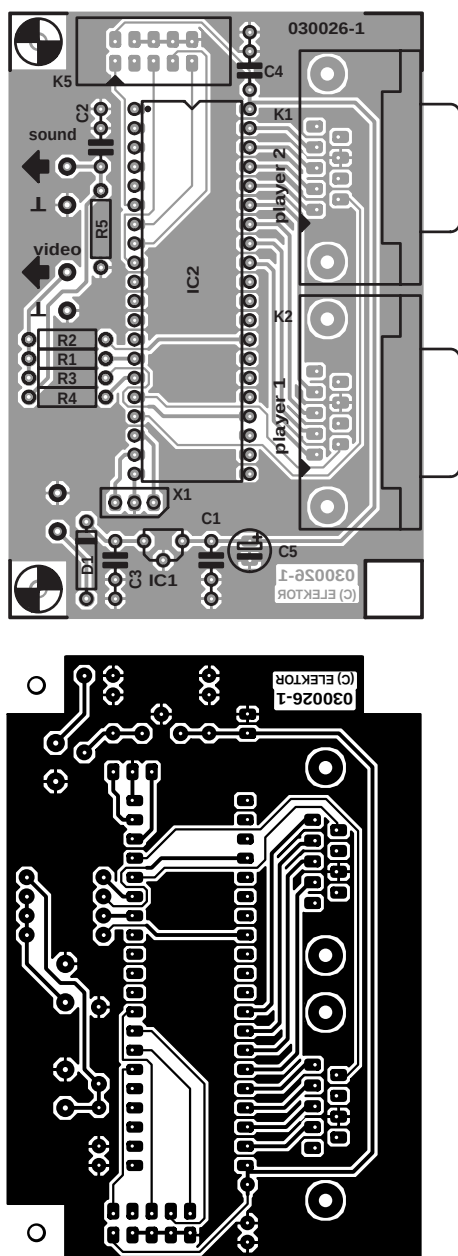


Figure 6. Single-sided board for the game.

quently the program control during the image assembly is landed with the TMR0 interrupt service routine (T0-ISR). However, when generating the black lines at the top and bottom of the picture, T0-ISR actually generates just

COMPONENTS LIST

Resistors:

R1 = 1k Ω
R2 = 576 Ω , 1%
R3 = 1k Ω
R4 = 820 Ω
R5 = 150 Ω

Capacitors:

C1-C4 = 100nF
C5 = 10 μ F 16V radial

Semiconductors:

D1 = 1N4148
IC1 = 78L05
IC2 = AT90S8515, programmed,
order code **030026-41**

Miscellaneous:

X1 = 8MHz ceramic resonator
Bt1 = 9V battery (6F22) with clip-on leads
K1, K2 = 9-way sub-D plug (male), PCB mount, angled pins
K3, K4 = cinch socket
K5 = 10-way boxheader
S1 = on/off switch, 1 contact
Enclosure, e.g., Hammond 1591BT
PCB, order code **030026-1** (main board)
PCB, order code **030026-2** (pushbutton board)
Disk, AVR source code, order code **030026-11** or Free Download.

the HSYNC signals. On completion, T0-ISR is left and the program control is given to the main program for about 59 μ s. Because the timer is programmed for 64 μ s (i.e., a full line), T0-ISR is called in time for the next HSYNC.

Unfortunately, when calling T0-ISR we have to take into account the so-called interrupt latency. In practice, this means that an interrupt can not cause the CPU to actually stop the execution of machine code at any instant. In fact, the current machine code will be finished to completion, the address of the next instruction is saved on the stack (to act as a return address) and only then will the CPU jump to the start of the ISR. That the AVR micro has instructions with different execution times is a complicating factor. The calling of the ISR can be delayed by an amount of four to seven machine cycles. Because our pixels have a width of eight cycles, the upper picture line would vary in length by up to half a pixel width. Such jitter is prevented by staying inside the interrupt routine from the start of the last two black lines before the visible (half) picture up to their end. All times within this range are mutually dependent using accurately adjusted instruction execution durations.

Our active video image consists of 480 lines, a complete CVBS image, of 625. Four lines are used to compensate the interrupt latencies. Of the 40 ms it takes to build a picture, about $(625-480-4) \times 59 \mu$ s =

8319 μ s is available to the main user program.

For picture memory access the user program has three main functions at its disposal. Using PLOTSCREEN an image contained in Flash memory (title screen or empty playing area) may be copied into the graphics memory. The function SETPIXEL allows a pixel to be given a certain shade of grey in order to draw the ball or the bats on the screen. GETPIXEL, finally, enables the grey level of any pixel to be sampled. This enables, for example, collision of the ball with an obstacle to be recognised.

Sound generator

The project includes a simple sound generator that employs the sound output of the Output Compare A (OC1A) pin of 16-bit timer 1. If the counter state matches the value in compare register A, the sound output is inverted and the counter gets reset to 0. The sound system contains an expandable list of musical notes (well, frequency/length information).

Once during the assembly of a half image, that is, exactly at 20-ms intervals, the user program calls the function DOSOUND, which compares a VSYNC counter with the length of the currently played note and, if necessary, moves on to the next note. Functions STARTSOUND and STOPSOUND allow new musical 'pieces' to be started or stopped at any time.

Free Downloads

AVR microcontroller software (source code). File number: **030026-11.zip**
PCB layouts in PDF format. File number: **030026-1.zip**
www.elektor-electronics.co.uk/dl/dl.htm, select month of publication.



Figure 7. TV Tennis welcome screen.

Construction

The circuit is built on a single-sided PCB of which the design is shown in **Figure 6**. With so few parts involved, the construction of the AVR TV Tennis project is unlikely to present problems and you should be able to build the game in less than half an hour. The only two points to note are that the microcontroller should be fitted in a good quality IC socket, and that connector K5 may be omitted if you

do not have the equipment to program the AVR controller 'in-circuit'.

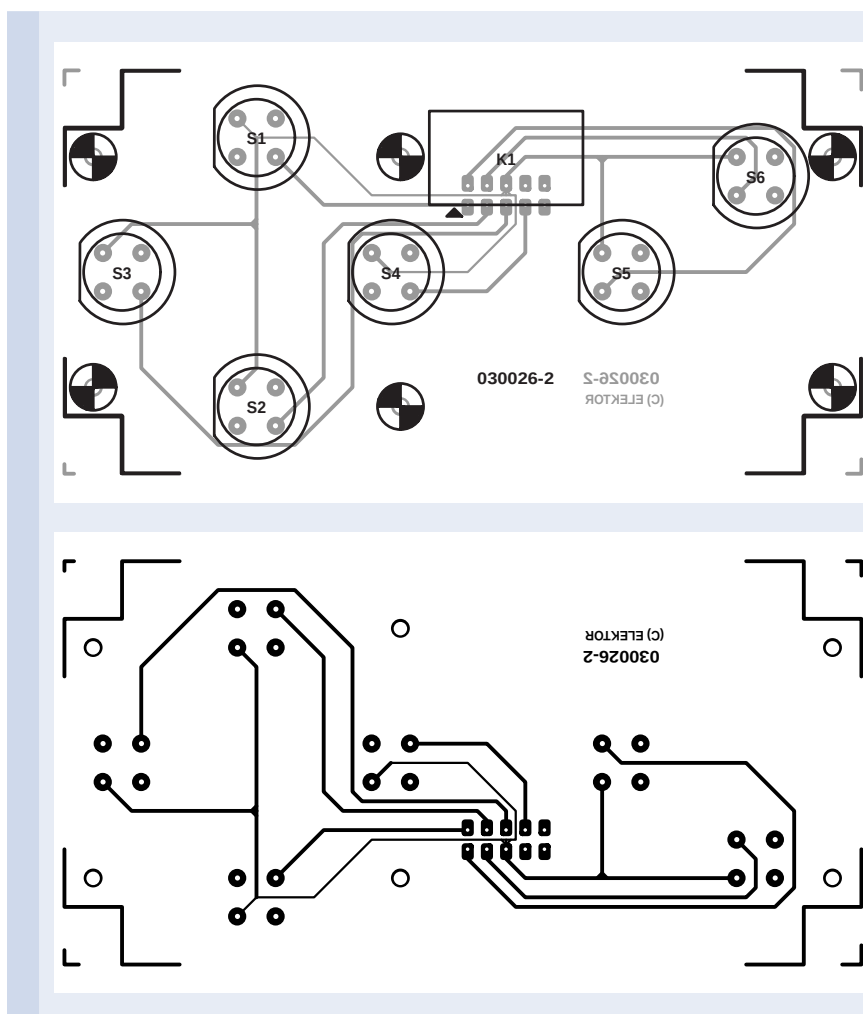
After a thorough inspection of your solder work, the digital joysticks are connected up and the video/sound cable to the TV set (SCART or composite video input) is plugged in. The title screen (**Figure 7**) should appear immediately when the supply voltage is applied. Joystick #1 even allows a game variant to be selected — if you press the right-hand pushbutton, you play TV Tennis. The left-hand button, however, allows the bats to be moved horizontally, too, while the ball will move much slower. Although the resulting game has been called Soccer by the author, admittedly the resemblance with the real game is very remote indeed. Whichever game is played, it is started by both players moving their joysticks upwards. If the ball bounces against one of the vertical walls, one point is gained by the player at the opposite side. The first player to collect 21

Software extensions

Only the main program and the play routines are specifically for the MahPong game. The remaining modules in the software may also be used for other games and projects. For example, the free port pins on the micro should allow serial communications to be implemented without problems. Alternatively, one more joystick could be connected up, or a digital thermometer. When an AVR controller with ADC inputs is applied, for example, the AT90S8535, it is even possible to connect analogue (PC) joysticks. The pots in these joysticks are then used as part of a R-C network, allowing the bat positions to be determined by means of the charge/discharge times. When the ATmega32 is used (AT90S8515-compatible, but 16 MHz and 2 Kbytes internal SRAM), then a significant increase in the screen resolution is within easy reach.

points is awarded the 'Winner' screen, which may be left by both players moving their joystick downward. Have fun!

(030026-1)



Digital joysticks

Digital joysticks, very common in the Commodore C64 age, are currently hard to get. If you fail to obtain a pair despite visits to second hand computer stores, flea markets, car boot sales and radio amateur rallies, you should not despair of ever being able to play an exiting game of TV Tennis. The good news is that a digital joystick is easily made at home, the circuit consisting of no more than four Direction buttons, two Fire buttons (not used by TV Tennis) and a connector. We have even designed a small printed circuit board for this mini project — designed to fit perfectly in a Hammond 1591BT case.

