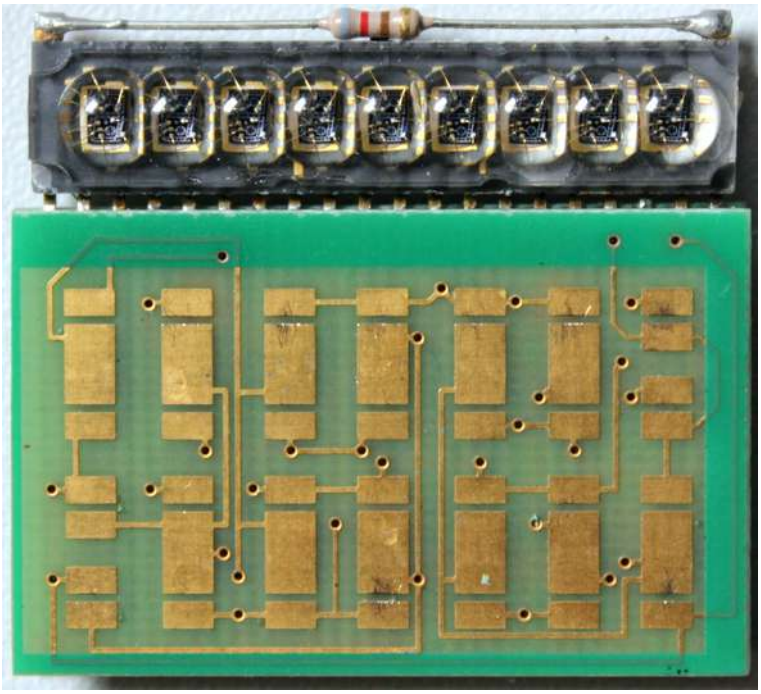
Inside the HP-01

This document gives some special technical information about the „new HP-01“ circuit, which replaces the forty year old original HP-01 Hybrid module and makes repairs possible, which were unthinkable before.

How does it work? What were the tricks? How do the components play together?



The new HP-01

The Hardware

The new HP-01 Hardware had to be designed as simple as possible and had to contain as few components as imaginable for two reasons. First, there is not much space inside the HP-01 watch, and last, the more components to use, the more current consumption will empty the small battery cells. Even the modern variety of hundred thousands of electronic components offer only few possibilities.

I chose a Low Power PIC processor PIC16LF1619, which is in the nanoampere range when sleeping, and offers up to 36 IO pins, for which every single of them needed to be used. And with its only 4x4mm size, it was perfectly fitting for the application.

Another crucial component was the real time clock. After comparing many options I chose a highly accurate chip PCF2127 , which is originally designed for stand alone electronic heat cost allocators.

The LED display is driven directly by the PIC processor , there was no room for driver arrays, thus only two active components are used.

Why is the new HP-01 circuit green and not white ?

Originally I wanted to produce the printed circuit board in white, to imitate the original ceramic board as close as possible. But the manufacturer couldn't guarantee the fine pitch soldering of the processor on a board with white solder mask. Only green solder mask was approved for these small pads. With a heavy heart I agreed to produce the normal green color, because functionality has priority before design.

And to be honest, if the circuit is installed once, and the watch case is closed, nobody ever will see the green printed circuit board. :)

How to sleep?

The important task of running the watch for month or years with the same batteries, can only be achieved by switching off the display after some seconds and switching the processor into deep sleep mode.

Just for showing what a bunch of tasks had to be accomplished before going to sleep and another bunch after wakeup, the following code shows the GotoSleep() procedure of the new HP-01. If you are familiar with C and with PIC peripherals you can understand what is going on, if not, just read the comments and have a slight imagination of it.

```
Enter Sleep mode, select all keyboard rows simultaneously and activate PORTB
Change Interrupt.
if stopwatch is running activate Timer 1 Overflow Interrupt

void GotoSleep()
{
  GIE=0; // Global Interrupts disable, stop display running
  LATD=0; // switch 8 LED segments off
  COLON=0; // switch colon dot off
  LATA=0xC0; // /CE=High /TS=High  select all keyboard rows ROW1-6 low

  flags &=~F_SHOWINFO;         // end text display
  act_flags&=~F_DISPLAY_ON;    // tell emulator, that display is switched off

  char buf[1];  // send command to RTC to disable second interrupt
  buf[0]=0x00;  // normal operation, 24h mode, Second Interrupt disabled
  RTCWrite(0x00,buf,1); // write to Control register 1, disable second interrupt

  if(TMR1ON) // is Stopwatch running ?
  {
    TMR1IE=1;  // enable Timer1 interrupt for wakeup from sleep at overflow
  } else       // Stopwatch not running
  {
    buf[0]=7; // 5=1024 Hz 6=1 Hz 7=High Z, High Z consumes 0 uA saves power
    RTCWrite(0x0f,buf,1); // write to CLKOUT register, disable 1024 Hz clock
  }
  TMR2IE=0;  // disable Timer2 Interrupt to avoid wake up by Timer 2 overflow
  TMR2IF=0;  // reset Timer 2 pending bit

L1:
// setup Interrupt On Change configuration

  IOCBN=0x3f; // negative edge interrupts for all 5 columns and /INT for alarm
  IOCBF=0x00; // reset IOC Flags
  IOCIE=1;  // enable Interrupt On Change

// all peripheral interrupts must be disabled, because they could
// accidentally wakeup from sleep

#asm
      sleep   ; enter deep sleep mode, FOSC stopped, processor halted
      nop     ; this operation is executed after wakeup from sleep
#endasm



// wakeup from sleep by either key pressed or Timer 1 overflow
```

```
  if(TMR1IF)  // can only be set if TMR1ON=1 SWSTARTED, every 8,5 Minutes
  {
    TMR1IF=0;
    LastTMR1H=TMR1H;  // must be 0

    if(!(hp01_flags & F_SWDEC))  // counting upwards
    {
      SWStartTime++;  // increment overflow counter, add 512 seconds
    } else
    {
      if(SWStartTime==0)  // countdown zero ?
      {
        hp01_flags &= ~F_SWDEC; // count upwards after reaching zero
        AlarmCnt=ALARMTIME;     // Start Alarm Buzzer when reaching zero
        SleepCnt=SleepTime;     // keep awake as long as alarm is active
        goto L2;
      } else
        SWStartTime--;    // decrement Stopwatch counter, subtract 512 seconds
    }
      goto L1; // goto sleep again
  }

      SleepCnt=SleepTime; // failsafe, if unwanted wakeup without keycode or
alarm

  if(IOCIF)  // keyboard interrupt occurred or alarm ?
  {
      if(IOCBF & 1)  // Alarm during sleep?
      {
// noting to do
      }
      else
      {
        if(!PressKey())
        {
//          ShowError('3');  // can happen if key bounce wake up
//          goto L1;
        }
      }
  } else
  {
// this should never occur if all other interrupts are disabled and
// pending flags reset prior to sleep
    ShowError('9');
  }

L2:
  TMR1IE=0; // disable Timer1 interrupt
  IOCIE=0;  // disable Interrupt on Change
  IOCBF=0x00; // reset all IOC Flags, clears also IOCIF

  RTCReadDate();  // read actual Time before display is activated

  buf[0]=0x01;  // normal operation, 24h mode, Second Interrupt enabled
  RTCWrite(0x00,buf,1); // write to Control register 1, enable second interrupt
  buf[0]=5; // 5=1024 Hz
  RTCWrite(0x0f,buf,1); // write to CLKOUT register, 1024 Hz

  TMR2IE=1;  // enable Time2 Interrupt
  PEIE=1; // enable peripheral interrupts
  GIE=1; // enable global interupts, start display running
}
```

The running Stopwatch during sleep mode

For having a very low power consumption and achieving years of battery lifetime, the sleep mode is entered as soon as the display shuts off. In power down mode, the processor only needs 20nA, which is nothing, and all clocks are stopped and it is not possible to display anything.

Only four things must be handled in sleep mode:

1.) a counting clock with date and time,

2.) a running stopwatch if started

3.) checking the alarm time

4.) wake up by any key press.

Fortunately the RTC chip can handle the counting clock date and time and the alarm. And by a special feature of the PIC chip, called "Interrupt On Change", the processor will wake up by any key pressed.

But none of them can handle a stopwatch! It is not possible to emulate a running stopwatch if the RTC has no stopwatch registers, nor with a sleeping processor. There was only one solution. They have to help each other.

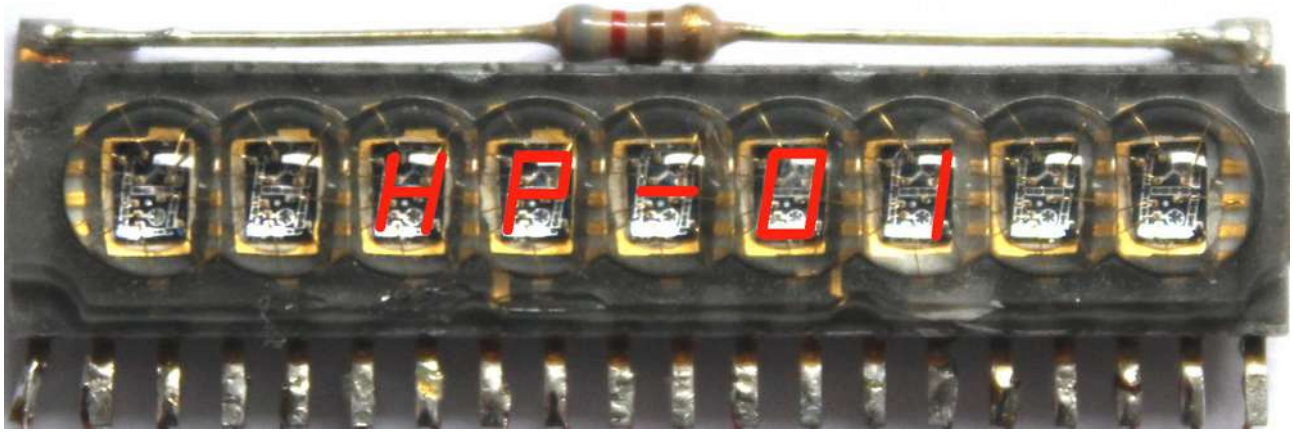
If the PIC processor has shutdown all its clocks, it anyhow accepts an external clock signal at one of its input pins, which does count an internal 16-bit timer upwards. And the RTC is able to deliver that accurate clock signal, which is connected to this external clock input.

So both together can build a counting unit, which does not need to wake up the processor. As the stopwatch counts in 1/100 seconds, it would be desirable to have a 100 Hz clock from the RTC. But according to the data sheet this is not possible, it can generate either a 1 Hz or 1024 Hz signal. 1 Hz is too slow, 1024 Hz is too fast. But by using a 1:8 prescaler in the PIC timer module, the 1024 Hz can be divided to 128 Hz and voila, this is a good approximation. 128 pulses represent one second of stopwatch count. Now it is easy to calculate the 1/100 seconds from 1/128 seconds just by making a multiplication. In fact, there have to be done some more calculations to get the actual stopwatch displayed from the timer value, as the stopwatch is able also to count downwards and the 16-bit hardware timer can only count upwards and it can count only 512 seconds (8 minutes 32 seconds) before it overflows. Fortunately the overflow can be programmed to wake up the processor. If a wakeup occurs by timer overflow, which happens every 512 seconds (about 8 minutes) if stopwatch is running, the processor just adds or subtracts 512 seconds to the stopwatch base value and goes to sleep again.

This was one of the last puzzles to solve for the new HP-01

The constant LED display brightness

Driving a LED display with several digits is always done by multiplexing. That is, only one digit is shown at a time for about 1 ms, then the next digit and so on. For the human eye it appears as if the display digits are constantly illuminated.



When displaying different digits, some need more current than others. The digit “8” for example has all segments on, and the digit “1” only 2 segments or a single dot with only one segment, will be displayed in different brightness, because the same current flows through 7 segments, or through only 1 or 2 segments. This effect is clearly visible and cannot be tolerated.

There was no way of inserting a constant current source for each digit or segment like in the original HP-01, neither were current limiting chips for 1,5 Volt available not would they have fit into the limited space of the watch. The original HP-01 had a unique display driver for this purpose, but I could not place an order for developing a display driver silicon chip of my own.

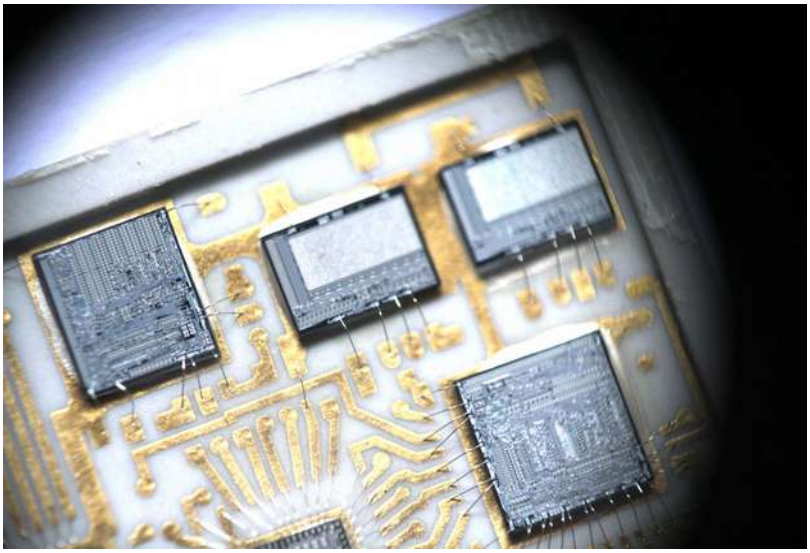
The solution I found was to show each multiplexed digit or character not at equal time intervals, but specifically different times, to compensate the brightness difference. i.e. the digit “8” is shown longer, that the digit “1”. One of the miracles of software with embedded systems, is that this kind of tasks sometimes really are possible. With a table of different times for each character, I managed to program the display multiplexing timer afer every digit to achieve a constant brightness for each character. The multiplexing frequency is in the range of 400 to 1200 Hz, instead of a constant 800 Hz or 1kHz as usual.

The Alarm Buzzer

The HP-01 back cover contains a piezo buzzer, which beeps for 5 seconds if an alarm is triggered. The new HP-01 does control the buzzer by issuing a square wave frequency of about 800 Hz in intervals.

But there was a difficulty. The 800 Hz signal had to be generated by the same unit that drives the LED display multiplexer. Don't ask why, but there was no other way. As a consequence when the alarm buzzer was used, the intelligent brightness adjust procedure had to be deactivated as long as the buzzer sound was on. Indeed, it is visible in the display during alarm beed, that the brightness now is different for the short intervals of beeping. Although this does not occur in the original HP-01 it is a nice feature, because the display does assist the alarm by a kind of slightly blinking.

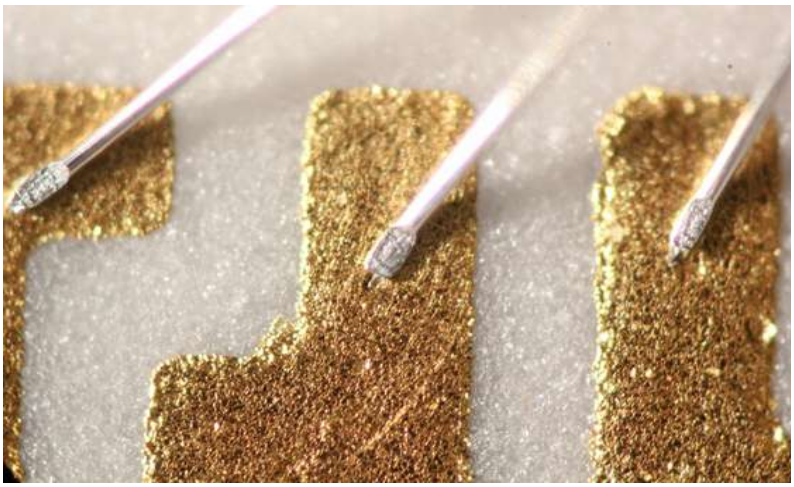
How I nearly failed!



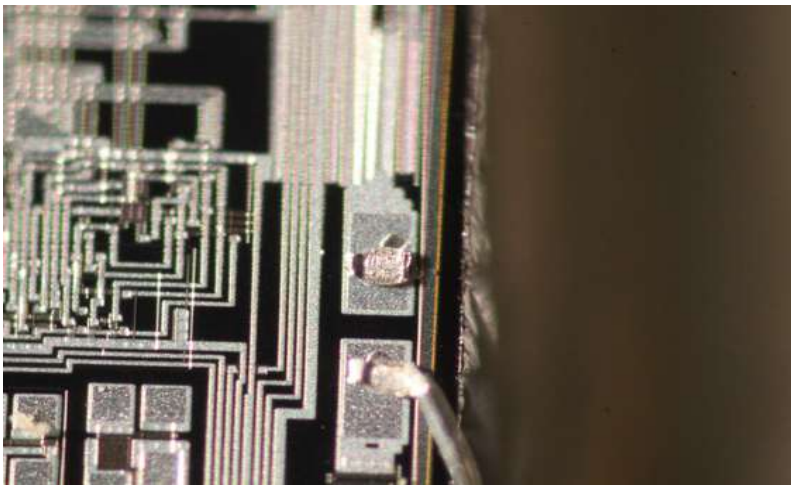
The HP-01 hybrid module with the two ROM chips visible on the upper right.

Below is the arithmetic chip, left is the display driver die.

For making a replacement of the original HP-01 it was essential to read the original ROM code from the original hybrid module. There was one critical moment, when I nearly lost my last chance to read the ROM code, when attaching my oscilloscope probe to the small circuits. I ripped off one of a small bond wires without the chance ever to reconnect it.



The very small bond wires are attached to the golden circuit paths of the hybrid module. The other end is connected to the silicon dies.



In the middle of the image you can clearly see the empty pad, where the bond wire originally was attached.

But I was very lucky! The wire was not related to the ROM signals, it was the carry bit of the arithmetic circuit.

Thus finally I could read the ROMs and the new HP-01 was given birth.

The Firmware

How is the display driven? How is the keyboard read?

They are driven together! The first 6 digits of the display are directly connected to the 6 keyboard rows. During display multiplexing the keyboard is read at the same time. If the keyboard rows had to be separate lines, the processor would not have had enough pins for both.

The HP-01 keyboard has 6 rows of 5 columns, which theoretically allows to distinguish 30 buttons, 28 buttons are present. The new HP-01 has the same keyboard layout than the original keyboard, as well as its pads are gold plated.

The Emulator

The program part, that executes the original HP-01 firmware instructions to calculate the basic arithmetic functions and show the time date and stopwatch, is called an „Emulator.“ This emulator is a derivative of Eric Smith's famous „nonpareil“ emulator, which he gave to the public domain some years ago. Although many of the secrets of the HP-01 firmware could be revealed by carefully reading the HP patent US4158285, and many others I could find and solve by myself, still all HP emulators benefit from his work directly or indirectly now and in the future. Many many credits to him.

Bernhard Emese
PANAMATIK

Sept. 14th 2016

HP-01 Schematic