Microcontroller Basics Course (2)

part 3: port characteristics and port accesses

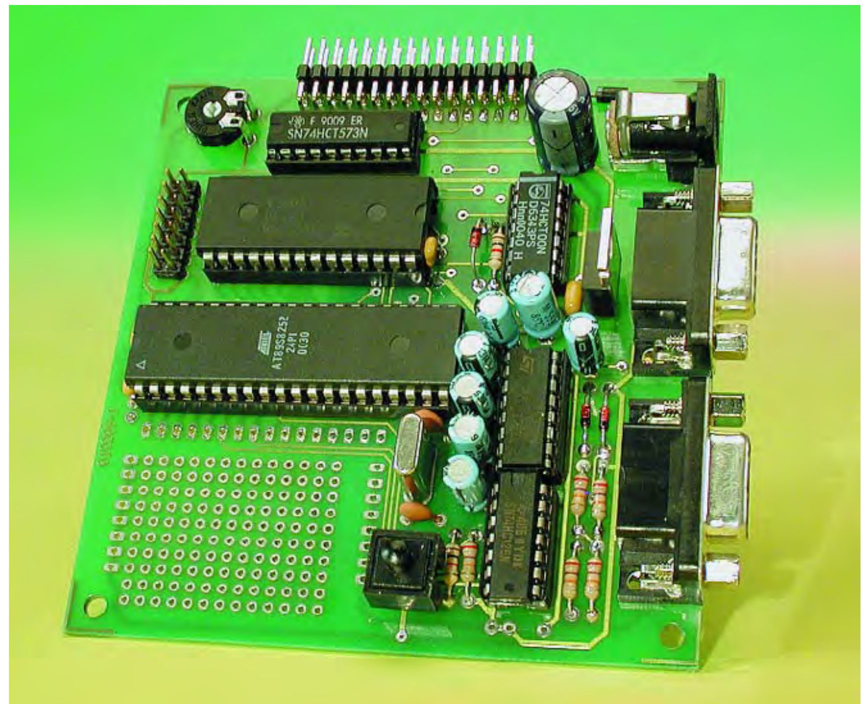
bahramelectronic.tk

By B. Kainka

In the first instalment of this basics course, we introduced the assembler that we will be using. Now we come to the first practical applications using the processor ports. First, though, we have to take a look at the physical characteristics of the port connections.

Anyone who already has a bit of experience with digital electronics knows that there are many different types of outputs. Each type has quite specific characteristics, regardless of whether it is a TTL, CMOS, tri-state, open-collector or open-drain output, and if you want to connect something to the output you need to be aware of these characteristics.

In any case, the Port 1 outputs of an 8051 microcontroller do not fall in line with any of the know logic families, but instead employ a rather unique solution. These ports are what is known as 'quasi-bidirectional', which means that they can be used as inputs or outputs without having to be specially switched over. You should keep in mind that the ports of a microcontroller represent a sort of door to the outside world. Depending on the task to be performed, inputs or outputs are needed. Some microcontrollers use tri-state buffers that must be switched to the high-impedance state to allow them to be used as inputs. This naturally requires a special switching signal or special instructions to switch the data direction. This is not necessary with an 8051, since all ports can be used as both inputs and outputs without any switching.



The port in detail

A glance at the detailed circuit diagram of a port (**Figure 1**) shows how a quasi-bidirectional port is built. There is a single FET with a pull-up resistor located at the output. In the High state, the FET is cut off and the pull-up resistor alone defines the internal resistance of the port. Consequently, it is certainly possible to connect any desired logic output here or change the signal level by means of a switch connected to ground. Even a logic input with high input impedance, such as that of a CMOS IC, will not have any difficulty recognising a High state.

The situation is quite different when the output is conducting and thus forces the signal level to be Low. In this case, the port impedance is relatively low. Anyone who attempts to force the output level of the port to a High state when it is in this state can only have bad intentions, since he or she is trying to force the microcontroller into the dig-

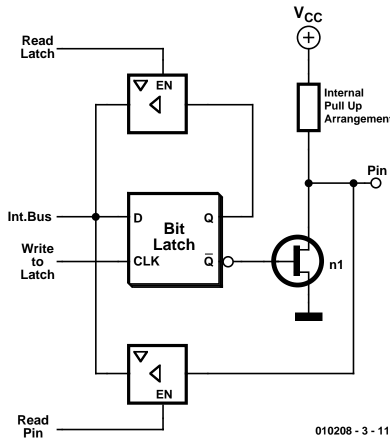


Figure 1. Internal structure of a port.

ital 'happy hunting ground'. We thus have the following rule: outputs must never be connected to other outputs. You must always remember that if you want to use a port pin as an input, it must be in the High state! This is always the case following a reset.

The internal circuit diagram of the port pin contains yet another simplification, since the pull-up resistor is in fact also a FET. Consequently, it acts like a constant-current source. This naturally raises the question of how large this constant current is. A measurement on the AT89S8252 shows that a short-circuit current of around 19 µA flows when the port is pulled to ground. This is not very much. It is far less than what is needed to drive an LED, for example. Is it actually enough to change the signal level on a lead reasonably quickly? Actually, it isn't, since every lead has an intrinsic capacitance that must first be charged or discharged. Here again there is a refinement in the internal circuit of the port. Strictly speaking, the pull-up resistor consists of two (in fact, three) FETs. One of them provides the normal, very weak pull-up current. A second one provides a much stronger current, but it is active only for a short interval when the port is switched from the Low state to the High state (or when an address must be output). The port pin thus changes state very quickly, even with a certain amount of capacitive loading. However, it can still be used as an input and can be actively pulled to ground, since any possible short circuit lasts less than a microsecond.

A quasi-bidirectional port can also directly drive an LED, but only if series resistor for the LED is connected to V_{CC} rather than ground. **Figure 2** shows how an LED and a switch can be connected. For the first program of the previous instalment of our course, the LED must be connected to one of pins P1.4 through P1.7 for it to be illuminated, since only these pins are switched Low. In this case, the switch should be connected to one of pins P1.0 through P1.3, since they are in the High state and can thus act as inputs. By the way, here you can operate the switch as often as you like, but nothing will happen, since we must first write a program that polls the input and evaluates the result.

Our first program loop

After this introduction, it's time for something practical. What we want to do is to automatically switch the output levels on the port pins. To do so, we will modify the program from the first instalment of the course to obtain the program shown in **Listing 1**. This program first outputs the port value 0Fh and then outputs the port value F0h. When hexadecimal notation is used, the first character must always be a numeral. This is why the listing shows a '0' in front of the second value, which is thus '0F0h' instead of 'F0h'.

Listing 1.
A program loop with port outputs.

```
;flash2.asm, fast loop

#include 8051.H
         .org 0000H

main     mov   a,#0Fh     ;1 a = 15
         mov   P1,a       ;1 P1 = a
         mov   a,#0F0h    ;1 a = 240
         mov   P1,a       ;1 P1 = a
         sjmp  main       ;2
         .end
```

Another feature of this listing is that the register p1 is no longer defined in the text. Instead, we have added an 'include file' (`8051.h`) that contains all important definitions, including much more than just Port 1. Besides this, the starting address is explicitly specified using the `.org` statement. The microcontroller always starts its programs at address 0000h following a reset. Finally, the critical change is that the loop has been expanded. It now encompasses the entire program, which is executed over and over again.

This small program helps answer a quite important question, which is how fast such a microcontroller can actually run programs. All we have to do is to touch the input probe of an oscilloscope to one of the port pins. Here we will see a rectangular waveform with a period of around 150 kHz. This can also be demonstrated using a radio. A short piece of wire attached to the port pin can serve as an antenna. In the long-wave band, you will find the signal at around 150 kHz. The fifth harmonic can be received at roughly 750 kHz in the medium-wave band. This gives us more insight into the fundamental significance of the EMC directives. Whenever high frequencies and steep edges occur, special provisions must be taken to prevent the circuit in question from acting as a transmitter.

The number of instruction cycles is shown in the listing comments. An instruction cycle takes 12 oscillator clock cycles, which means it has a period of (12 ÷ 11.059 MHz) = 1.095 µs. Most instructions have a duration of one instruction cycle, while the jump instruction takes two cycles. In total, the sum of the instruction times is six instruction cycles. This means that the loop takes 6.51 µs, which gives a frequency of 153 kHz.

There is yet another interesting observa-

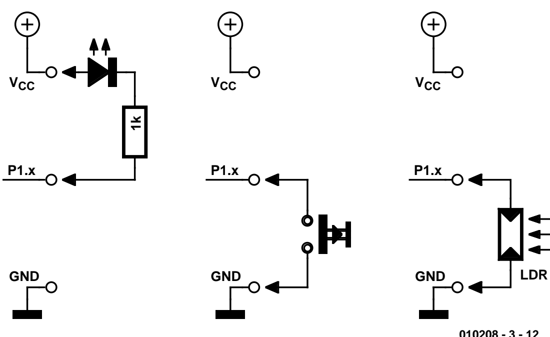


Figure 2. Port connections for inputs and outputs.

tion that we can make with this program. We can use it to watch the two pull-up FETs working 'live'. To do so, we connect a 33-kΩ resistor from one of the port pins to ground. With this resistance value, the smaller FET is no longer able to pull the signal level to V_{CC}, but the larger FET can still do so. In the oscillogram, we can now see how long (or better, how short) the current level is increased. The initial edge is very fast. This is followed by a high plateau with a duration of around 100 ns (one quarter of a clock period), and finally the voltage drops to a low level.

The port evidently has yet another characteristic that is not mentioned in most data sheets. The smaller FET is also divided into two and behaves differently when the port state is High than when it is Low. With a load resistance of 6.8 kΩ, we see a second kink in the curve at a voltage of roughly 1.5 V. Evidently, the current that flows in the region above this input voltage is greater than the current that flows below this level. Altogether, this acts like a form of current feedback and results in a certain amount of input hysteresis. It can be easily measured using a multimeter. At a voltage greater than 1.5 V, the port supplies up to 200 µA, but below this voltage the current is only 10 µA. Thanks to this port behaviour, connecting a simple resistance to an input always results in an unambiguous input state. It is even possible to connect a potentiometer or an LDR, which will then be read with well-defined hysteresis. In the original Intel and Philips data books, the division into three FETs can still be seen, but the data sheets for the more recent 8051 derivatives from Atmel neglect this detail.

Figure 4 shows a small circuit for our first experiment with input connections for a port. Here P1.0 can be set to zero either by means of a switch or by a shining a sufficiently bright light on the LDR. A matching program must read the port state and switch on the LED attached to port 1.1 only when a High signal is found at the input. Here we must take into account the fact that the LED is connected to V_{CC}, which means that it has an inverted function; it is 'on' when P1.1 is in the Low state. S2 allows us to provide feedback between the output and input of our test setup. What happens if we actuate S1 and S2 together and thereby short the P1.1 output to ground? Nothing serious, since a connection to ground is always allowed for a quasi-bidirectional port. The only thing is that in this case the LED is always on.

Listing 2.
Responding to an input state

```
;flash3.asm, input/output

#include 8051.H
          .org 0000H

main     jb    P1.0,ON   ;P1.0 = ?
         setb  P1.1      ;P1.1 = 1
         sjmp  OFF
ON       clr   P1.1      ;P1.1 = 0
OFF      sjmp  main
         .end
```

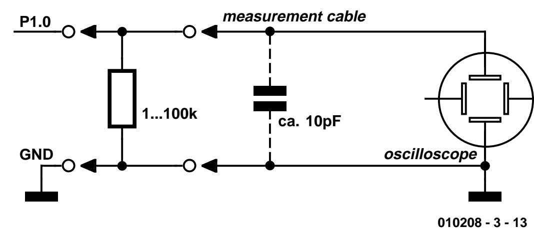


Figure 3. Measuring the port signal under load.

A conditional jump

The assembler program for **Listing 2** reads the port state at P1.0 and then executes a conditional jump. The instruction `jb` (jump if bit set) belongs to the group of special instructions for single-bit processing. Although most instructions work with byte values, these instructions evaluate, set or clear individual bits. Here 'P1.0' refers to a single port pin, while 'P1' in the first example refers to the complete port with all eight of its leads. The header file (`8051.h`) defines all necessary bit addresses, such as P1.0 and P1.1. You can view this ASCII file using a simple text editor.

In the program, if a '1' state is read at P1.0 in the first line, a jump to the destination 'ON' is executed. At this location, port P1.1 is placed into the Low state by means of `clr P1.1` ('clr' = 'clear'), which switches on the LED. On the other hand, if a Low state is read at P1.0 in the first line of the program, the jump is not executed. Instead, the program quite properly executes the next following instruction. Here we find `setb P1.1` ('setb' = 'set bit'). This is an instruction to enable port P1.1, which effectively means that the LED is switched off. In order to prevent this from being counteracted by the lines with the label 'ON', it is necessary to jump over this part of the program. An additional jump instruction thus leads to the destination 'OFF', where the loop is closed by jumping back to the very beginning of the program.

This small example shows several interesting results. For one thing, we have practically built a logical inverter. A Low state on the input results in a High state on the output, and vice versa. Since the LED (Low = on) and the input switch (on = Low) have inverting functions, the overall operation is inverting. This means that if you press S1, the LED goes off, while if you release the switch the LED goes on again.

This experiment will show the input hysteresis of the port if a resis-

Figure 4. Port loaded with 33 kΩ.

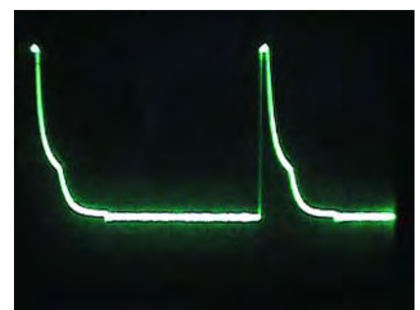


Figure 5. Port loaded with 6.8 kΩ.

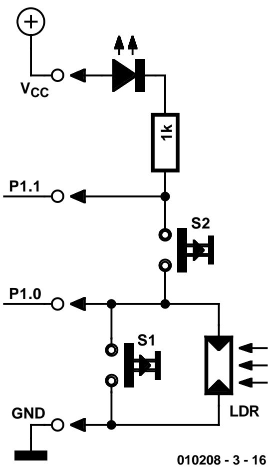


Figure 6. A circuit with an input and an output.

tor potentiometer or LDR is connected to the input. There is a definite gap between the switch-on and switch-off brightness of the light falling on the LDR.

If S2 is closed, feedback gives rise to a rapidly alternating succession of signal levels. The only visible effect of this is that the brightness of the LED is reduced, but an oscilloscope reveals the true situation and shows a sequence of fast rectangular signals. Naturally, this is exactly what should be expected, since the input state is only polled once at the beginning of the program. With feedback, the microcontroller will always find the output state resulting from the previous pass through the loop at this location, and that is exactly the opposite of the previous input state. The microcontroller thus has no other choice than to repeatedly switch the port state back and forth.

If we attempt something like this using a logic gate, in this case an inverter, the result can look much different. Usually the signal assumes an average voltage level, by means of which the gate reveals its analogue roots, since it actually an inverting dc amplifier. A sequential logic circuit (such as a microcontroller), by contrast, allows no possibility of analogue behaviour. Unambiguous yes or no decisions are always taken.

Counting loops

The final program for this instalment uses only outputs. In this case, we want to have all eight pins of port 1 output symmetrical square waves at different frequencies. Our model for this (in digital electronics) is an 8-stage binary counter. A clock signal is applied to the input, and its frequency is divided by exactly 2 for each stage of the counter. The microcontroller generates the clock signal itself using its program. The divider chain can be obtained very simply by incrementing a binary number. This can be done either by using an addition instruction or an `inc` (increment) instruction. The instruction `inc a` increases the value stored in the accumulator by 1 each time. The program must execute this instruction repeatedly in a loop and repeatedly output the value of the accumulator to the port.

The program shown in **Listing 3** first loads an initial value of zero into the accumulator. Following this, the accumulator value is output to the port, the value in the accumulator is incremented, a jump is made to `next` for the next port output and so

Listing 3. A counting loop.

```
;flash4.asm  port outputs
#include 8051.H
         .org 0000H

main     mov   a,#00
next     mov   P1,a      ;1
         mov   r1,#255   ;1
loop     djnz  r3,loop   ;2 * 255
         inc   a         ;1
         sjmp  next      ;2
         .end
```

on. The program also contains a second counting loop to cause everything to run a bit slower. The objective is to allow at least the lowest-frequency signal on port P1.7 to be directly observed using an LED.

The counting loop uses a register (r1). There are eight registers in total (r0 through r7). It does not matter here exactly which register is used. The register is loaded with a value of 255, which means that in this case a decimal number is used. We could have just as easily written this as 0FFh. The actual counting loop employs the complex assembler instruction `djnz` (decrement and jump if not zero). During the first pass through the loop, the value in r1 is reduced to 254. Since this is greater than zero, a jump is made back to `loop`, where the same instruction is again executed. During this pass, the value is reduced to 253. After a total of 255 passes through the loop, the value of zero is reached. Now the jump is no longer executed, but instead, the program continues with the next following line. In this manner a total of 255 loop cycles, amounting to 510 instruction cycles or 558 µs, are consumed. If we add the remaining instructions, we have a total period of approximately 564 µs, or a frequency of 1.77 kHz. This is thus the clock frequency at the input to the counter chain. Consequently, on P1.0 we find a square-wave signal with a frequency of 885 kHz, on P1.1 443 kHz and so on, down to around 7 Hz on P1.7. This frequency can be observed using an LED. It is worthwhile to perform a number of additional experiments using different loop parameters. The highest frequency results when the starting value is 1. By the way, the greatest reduction in the clock rate can be achieved with a value of 0, rather than 255, since when 0 is decremented it rolls over to a value of 255, resulting in 256 passes through the loop.

(010208-3)

This completes our introduction to assembler. The next instalment will start with BASIC-52.

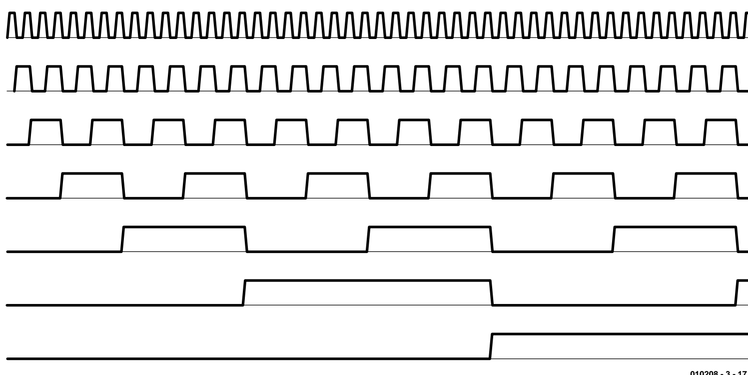


Figure 7. Output signals on P1.0 through P1.7.