Using the analog-to-digital converter of the 8XC552 microcontroller

EIE/AN93017

Author: Theo van Daele, Philips Semiconductors Product Concept & Application Laboratory Eindhoven, the Netherlands

SUMMARY

On the 80C552 microcontroller, an 8-input 10-bit ADC is available. To get correct results from the ADC, the slew-rate of the input signal during sampling must be limited. 10 Bit accuracy will be obtained if the layout of the 80C552 application is done correctly. EMC measures must be taken into account. Some software examples are given on how to use the ADC.

1.0 INTRODUCTION

The 80C552 microcontroller has an on-chip ADC. The converter consists of an 8 input analog multiplexer, and a 10-bit binary successive approximation ADC. A conversion takes 50 machine cycles (is 20µs at 30MHz oscillator frequency). The ADC has dedicated analog supply and reference voltages to minimize influence form digital circuitry. The DAC of the successive approximation ADC is a resistor ladder network. This ensures that there no missing codes.

To obtain the 10-bit accuracy, it is important to pay attention to the design of the application. First the operation of the ADC will be described. Then design and layout subjects are described that can influence the accuracy of the conversion result.

References:

1. *80C51-based 8-bit microcontrollers* (Data Handbook IC20 1994)
2. *Electro Magnetic Compatibility and Printed Circuit Board (PCB) Constraints* (ESG89001)

2.0 INTERNAL OPERATION OF THE ADC

2.1 General Description

Figure 1 shows a general block diagram of the ADC.

The inputs of P5 are connected to a multiplexer and an input buffer with Schmitt-trigger inputs.

When the digital value on P5 must be read (e.g., with a MOV A,P5 instruction), the output of the Schmitt-trigger is taken. This output can be used for further processing.

An analog input signal on P5 that must be converted is selected by the input multiplexer. The bits ADCON.0 . . ADCON.2 of the ADCON special function register select the input signal. The output of the multiplexer is connected to the input of a comparator. The sampling capacitor is included in the comparator. The ADC control block of the ADC controls the timing of the sampling and conversion.

After the input signal is sampled, the actual analog-to-digital conversion starts. The comparator compares the input signal V_{IN} with the output of the 10-bit DAC V_{DAC}. The output voltage of the DAC is determined by the output of the successive approximation register (SAR). The range of the DAC signal varies between AV_{REF-} and $AV_{REF\_}$. These two signal levels also define the voltage range of the input signal.

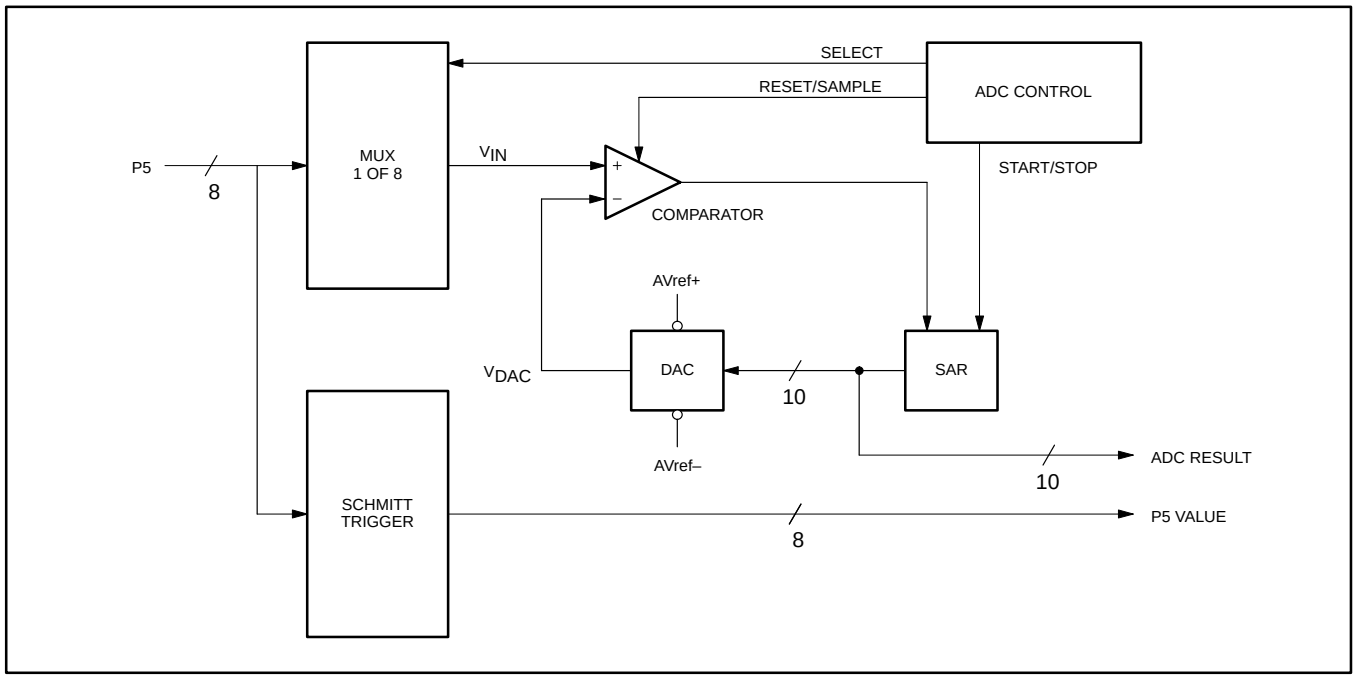


Figure 1.

2.2 Conversion Process

Figure 2 shows an example of the conversion principle with 3 bit resolution.

The SAR will make its output bits SAR2 . . SAR0 successively high from MSB to LSB. Every time a SAR-line is made HIGH, a DA-conversion will take place. If the output of the DAC (V_{DAC}) is higher than the input voltage (V_{IN}), the SAR output bit that was made HIGH the last time will be made LOW. If V_{DAC} is smaller than V_{IN}, the SAR output bit will remain HIGH. The process will proceed for the subsequent SAR output bits. At the end of the conversion, V_{DAC} has converged to a value of $V_{IN} \pm {}^1/_2$LSB.

Example: V_{IN} is 11/16*V_{REF}. The conversion sequence is shown in Table 1.

After STEP 3 the conversion is finished. The SAR register contains the result of the AD-conversion.

The ADC in the 80C552 has 10 bits resolution. The conversion in this ADC will take 10 conversion steps.

Table 1.

	SAR Value (SAR2.SAR1.SAR0)	V_{DAC} (*V_{REF})	Output Comparator	Action by SAR
START	000	0	0	SAR2=1
STEP 1	100	4/8	0	SAR1=1
STEP 2	110	6/8	1	SAR1=0, SAR0=1
STEP 3	101	5/8	0	

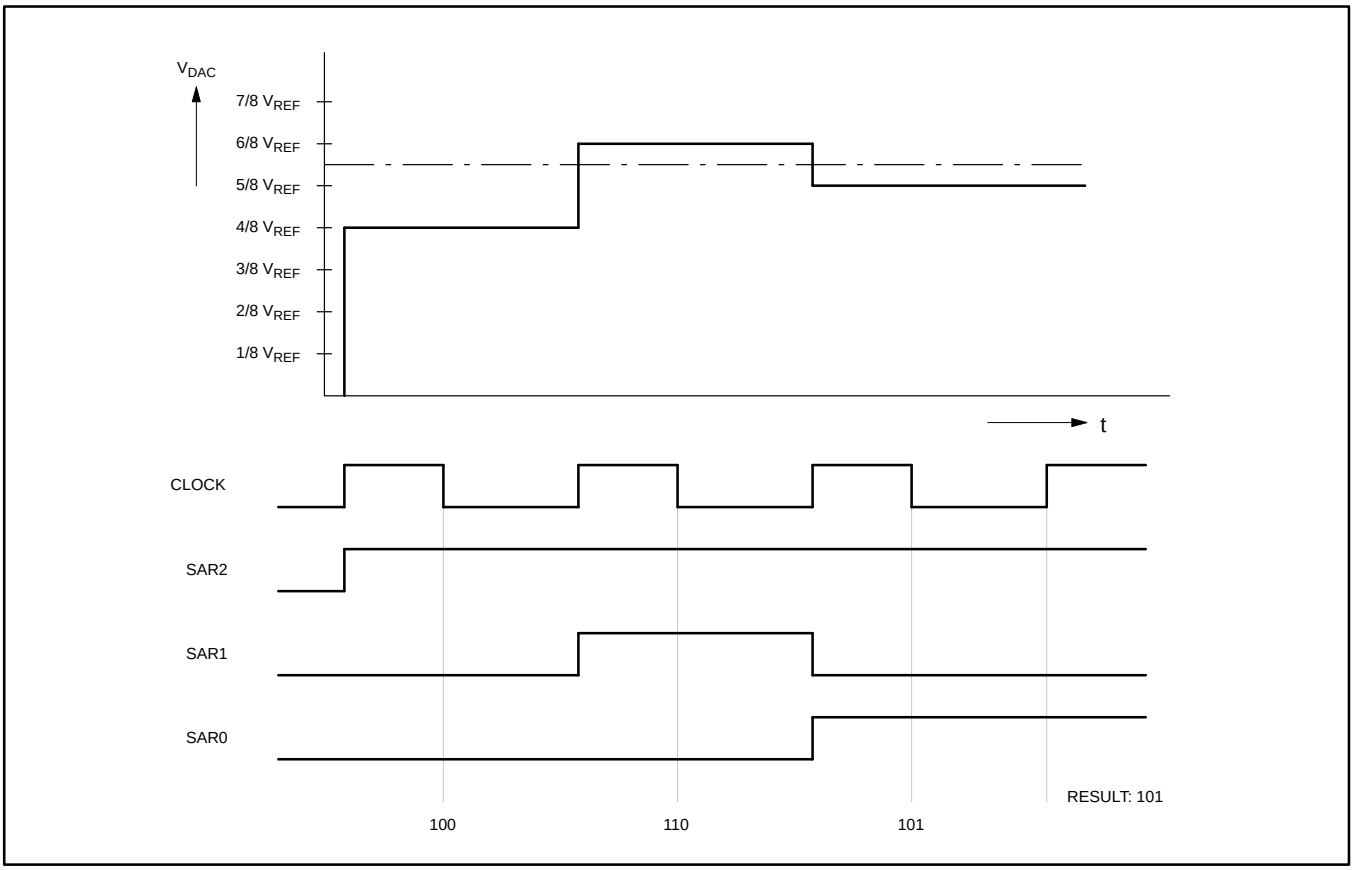


Figure 2.

2.3 The ADC in the 80C552

Figure 3 shows a block diagram of the implementation of the ADC in an 80C552 microcontroller.

The analog input signal V_{IN} is connected to the non-inverting input of the comparator via switch S1 during the sampling interval. Internally the comparator consists of 3 serially connected sampled-data-comparator stages A_1 . . A_3. The stages are capacitively coupled. The coupling capacitor of the first comparator that is connected to S1 will also act as sample capacitor for V_{IN}.

Sampled-data-comparators are used to minimize the effect of offsets and temperature drive. During the sampling interval, the value of the offset voltage of the comparator stages are stored on the coupling capacitors. This voltage will have the opposite sign of the comparators stages' offset, so it will cancel this offset voltage. This process is called auto-zeroing, and will be explained in 2.3.2.

The non-inverting input of the comparator is connected to $^1/_2V_{REF}$ via switch S2. S2 consists of 2 parallel switches. There is always 1 switch closed, so the voltage on this input is always $^1/_2V_{REF}$. Although S2 looks superfluous from a functional point of view, it assures that, for instance, switching glitches of S1 and S2 appear on both inputs of the comparator and will cancel each other.

When the sampling is finished, the actual conversion will start. S1 will connect the inverting input of the comparator with the output of the DAC. At this moment, the output of the DAC is connected to the center tap of the resistor network. The voltage on the inverting input of the comparator will be $^1/_2V_{REF}$ (V_{REF} is defined as $^1/_2[V_{ref+} – V_{ref–}]$). During conversion, the output of the SAR will determine which tap of the ladder-network will be connected to the inverting input. Using a ladder-network guarantees a monotonic characteristic of the DAC. This in turn will result in an ADC-characteristic without missing codes. The relative deviations of the resistor values result in a non-linear transfer characteristic.

The following three phases in the ADC conversion can be determined and will be described in more detail:

– Start phase
– Sampling phase
– Conversion phase.

Timing of these phases is shown in Figure 4.

2.3.1 Start Detection Phase

An ADC conversion can be started by software or by a hardware trigger on the STADC pin.

Software start

When an ADC start is initiated by software (set ADCS in ADCON register), the internal start signal will immediately be active at S6P2 (for state timing, see [Reference]). The value of ADCS can be read by software. However, there is a delay of 2 machine cycles between the internal start signal and the ability of reading a '1' from ADCS.

Hardware start

A hardware start of an ADC conversion is initiated by a rising edge on STADC. The 80C552 samples STADC every machine cycle during S6P2. When a valid edge is detected, the internal start signal will be active at S1P2 in the subsequent machine cycle. To ensure that the edge is detected, the high and low time should be at least 1 machine cycle each. When a valid edge is detected, 'ADCS' is set.

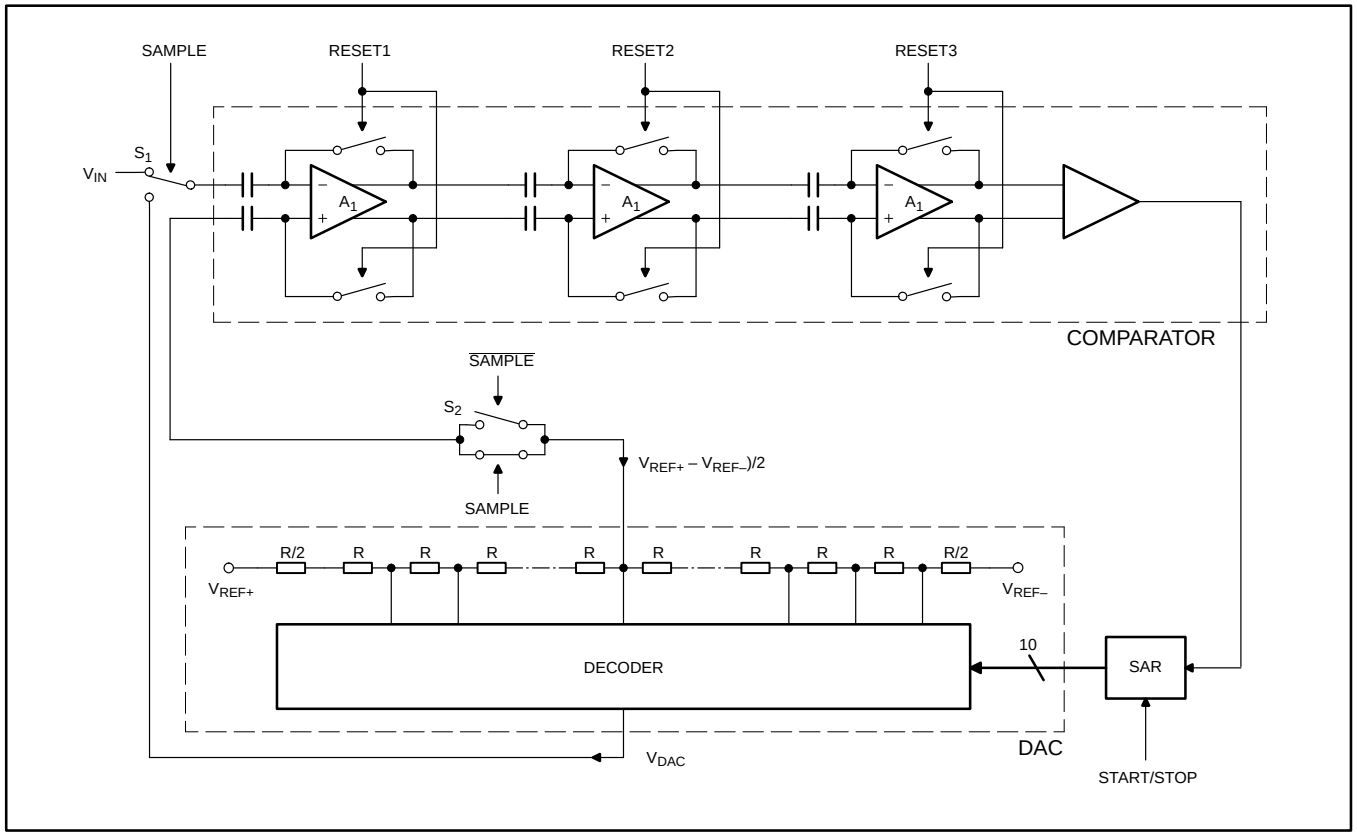


Figure 3.

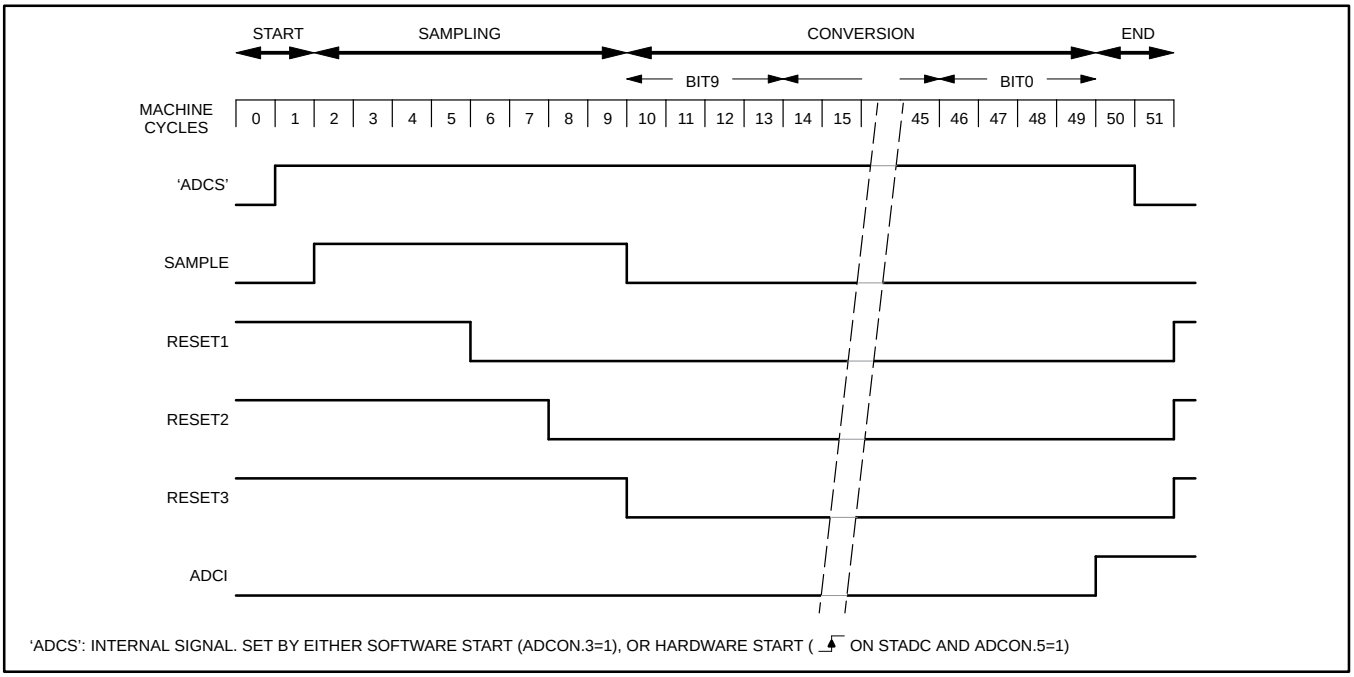


Figure 4.

2.3.2 Sample Phase

The 8 machine cycles following the start detection is the sample interval (Figure 4). In this time interval, the input signal is sampled and the 3 comparator stages are auto-zeroed.

The actual sampling of the analog input signal on the input capacitor starts at machine cycle '2' (Figure 4). The sample capacitor is connected between V_{IN} and the output of the first comparator state (Figure 3). The sampling is finished at the end of machine cycle '5'. After this machine cycle, the sample capacitor is connected between V_{IN} and the input of the first comparator stage. Since this is a very high impedance input, no extra charge will be stored in the sampling capacitor.

At the start of the sample phase, the inverting input of the comparator is connected to V_{IN} via a coupling capacitor. This coupling capacitor also serves as sampling capacitor. The non-inverting input is connected to the DAC via a coupling capacitor to a voltage of $^1/_2V_{REF}$.

Figure 5 shows the sampling and auto-zeroing of the first comparator stage.

When the RESET1=1, switches will connect the outputs of the individual comparator stages to their inputs. The outputs will settle to the unity-gain output voltage V_{UG}.

The differential voltage (error voltage) on the inputs of the first comparator stage will be:

$$V_{IL} = V_{OL} = -V_{OS1} \times \frac{A_1}{A_1 + 1}$$

V_{IN} = Input voltage of ADC
V_{IL} = Differential input voltage of first comparator stage
V_{OL} = Differential output voltage of first comparator stage
V_{OS1} = Offset voltage of first comparator stage
A_1 = Open loop gain of first comparator stage

The switches are opened when RESET1=0. The differential voltage on the comparator inputs is still V_{IL} because of the stored charge on the coupling capacitors.

The resulting offset voltage $V_{OS1,I}$ seen on the input of the comparator stage is obtained by adding this differential voltage V_{IL} to the input offset voltage V_{OS1}.

$$V_{OS1,I} = V_{OS1} + V_{IL} = V_{OS1} \times \left(\frac{A_1}{1 + A_1}\right)$$

The effective offset voltage at the input of the comparator stage is reduced with a factor $(1+A_1)$.

The auto-zeroing procedure described above will be repeated successively for the following 2 stages. After auto-zeroing the third comparator stage, the differential output voltage of the total comparator (all 3 comparators in series will be:

$$V_{O,3} = A_3 \times \frac{V_{OS3}}{1 + A_3}$$

This voltage can be translated to an effective input offset voltage by dividing it by the total gain of the comparator:

$$V_{OS,I} = \frac{V_{OS3}}{A_1 \times A_2 \times (1 + A_3)}$$

If auto-zeroing was not used, and all comparator stages were DC-coupled, the differential output voltage of the comparator would be:

$$V_{O3} = A_1 \times A_2 \times A_3 \times V_{OS1} + A_2 \times A_3 \times V_{OS2} + A_3 \times V_{OS3}$$

The effective input offset voltage in this case is:

$$V_{OS,I} = V_{OS1} + \frac{V_{OS2}}{A_1} + \frac{V_{OS3}}{A_1 \times A_2}$$

As can be seen, the auto-zeroing reduces the effect of the individual comparator stages considerably.

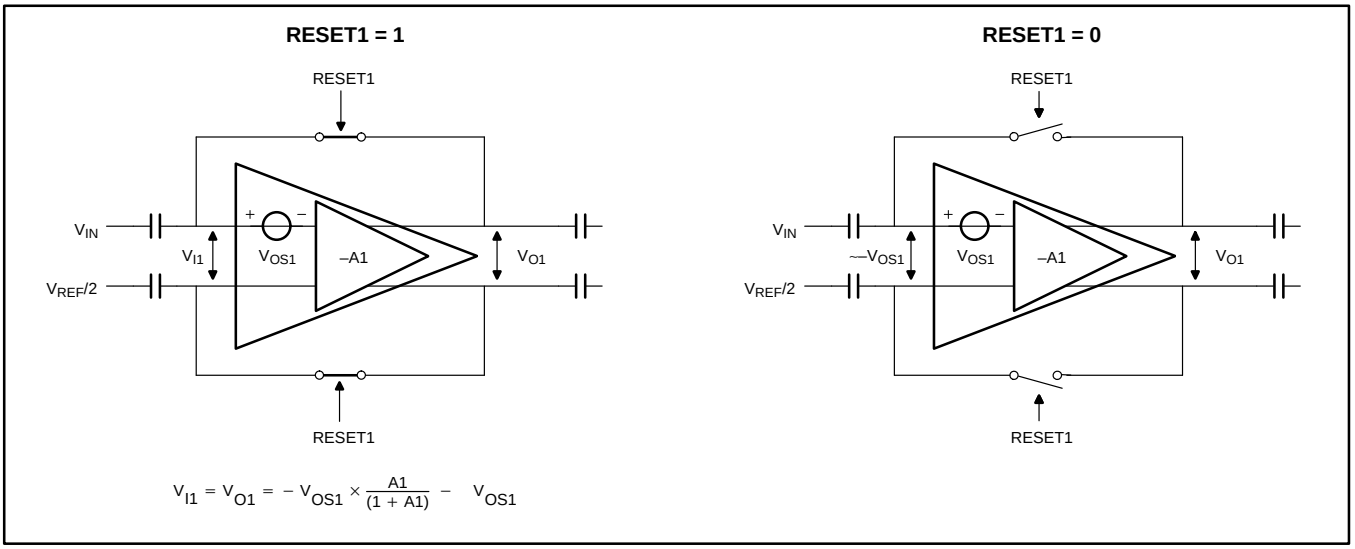


Figure 5.

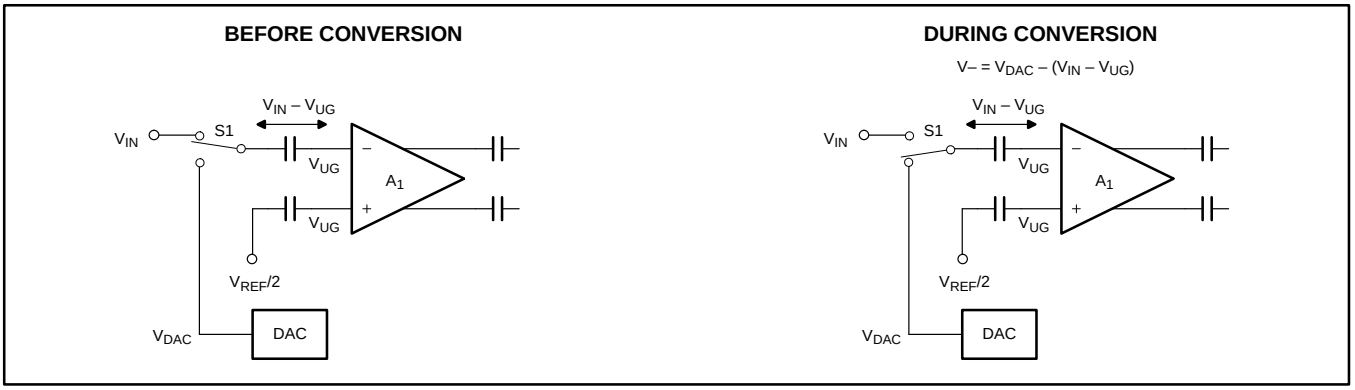


Figure 6.

2.3.3 Conversion Phase

Just before the sampling phase is finished, the following voltages are present over the coupling capacitors of the first comparator stage:

Capacitor on inverting input: $V_{IN} - V_{UG}$

Capacitor on non-inverting input; $^1/_2V_{REF} - V_{UG}$

For clarity, the offset voltages are neglected.

When the conversion phase is started, S1 (Figure 6) will connect the coupling capacitor of the inverting input to the output of the DAC. The effective voltage on the comparator input is the voltage applied to the coupling capacitor minus the voltage that was stored on the capacitor during the sampling phase.

The following voltages are present on the input of the first comparator stage:

Inverting input: $V_{DAC} - V_{IN} + V_{UG}$

Non-inverting input: V_{UG}

The comparator stage amplifies the differential voltage between its inputs. The output voltage of the first comparator stage will be:

$$V_{OL} = A_1 \times (V_{UG} - (V_{DAC} - V_I + V_{UG})) = A_1 \times (V_I - V_{DAC})$$

After amplification by the 3 comparator stages the input signal for the SAR is $|A_1 \times A_2 \times A_3 \times (V_{IN} - V_{DAC})|$. Depending on the sign of this signal, the SAR will set or clear the MSB. In the following cycles of the conversion, the other bits of the SAR will be updated. At the end of the conversion V_{DAC} will have a value of $V_{IN} \pm 0.5$LSB. The contents of the SAR that generates this V_{DAC} is the result of the AD-conversion.

3.0 APPLICATION INFORMATION

Although the ADC in the 80C552 has a resolution of 10 bits, the user must be careful in the design of the application to really get this resolution. The constraints can be divided in 2 categories:

– Constraints on the analog input signal and the input signal source

– Layout constraints of the design.

3.1 Analog Input Signal Constraints

3.1.1 Range of Analog Input Signal

The value of the analog input signal must be between V_{REF+} and V_{REF-}. The span of the analog input signal is $V_{REF} = (V_{REF+} - V_{REF-})$.

There is a minimum limit to the span. This limit depends on the gain of the comparators. A differential voltage of 1LSB (1LSB = $V_{REF}/1024$) on the inputs of the comparator should be able to generate a logic ‘1’ or ‘0’ level on the input of the SAR. If not, the resolution of 10 bits for the ADC will not be met.

the comparator in the 80C552 needs a minimum differential input voltage of 0.3mV to generate a valid logic output level. For the 10-bits ADC in the 80C552, this means that V_{REF} should be at lease 1024*0.3mV=0.31V to get 10-bit resolution. The absolute values of V_{REF+} and V_{REF-} that determine this span may not exceed AV_{SS} and AV_{DD}.

3.1.2 Slew Rate of Analog Input Signal

A distinction must be made between 2 different slew-rate constraints. The first slew-rate constraint deals with the required accuracy during sampling. The second constraint to prevent wrong readings deals with a limitation on the slew rate that may otherwise lead to a conversion result that has no relation at all with the analog input signal.

1: To obtain a stable reading from the ADC, the analog input signal should be stable during the sampling time. The sampling may be triggered by an external event (via ADEX pin). From this trigger point until machine cycle ‘5’ (see Figure), the input signal is sampled and should not change more than the desired accuracy.

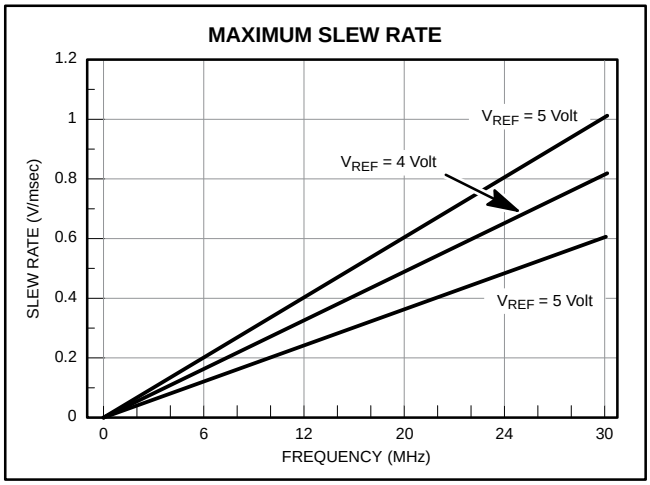
Example: If a stability of 0.5LSB is required, then the analog input signal should not change more than 0.5LSB in 6 machine cycles. In that situation the maximum slew-rate of the analog input signal is:

$$\frac{dV}{dt} = \frac{0.5LSB}{6T}$$

where T is the machine cycle time.

The following graph gives the maximum slew-rate as function of the operating frequency for various values of V_{REF} and a required stability of $^1/_2$LSB for a 10-bit conversion.

When the slew-rate of the input signal is more than the maximum slew-rate as determined above, the read-out stability will decrease. The conversion result will be the digital value of the input signal somewhere between machine cycle ‘0’ and machine cycle ‘5’. Consecutive conversions of a signal that consists of a DC-value with an AC-component that has high slew-rates as mentioned above, and that has the same amplitude between machine cycle ‘0’ and machine cycle ‘1’, may give different read-outs. However, the accuracy of the sampled signal will not be affected.



2: If the slew-rate exceeds a certain value, the accuracy of the conversion will decrease rapidly. The result of the conversion will not have any result anymore with the analog input signal. Tests have shown that the most probable conversion result is 0x3ff (result bits ADS.0 . . ADC.9 are ‘1’).

This error situation will occur when the slew rate is too high in the time frame from machine cycle ‘2’ to machine cycle ‘9’ of the conversion. In this time frame, the comparators are auto-zeroing their offsets. For proper auto-zeroing, the comparator stages must work in their linear region.

If the input signal is changing rapidly, the voltage change may couple through the coupling capacitors to the input of the comparator stage. If this voltage change is sufficiently high, it may saturate the comparator stage. The comparator stage is not working in its linear region anymore, and the saturation voltage (equal to about the supply voltage) is stored on the coupling capacitors.

The ADC has the highest sensitivity to these high slew-rate signals in the time frame from machine cycle ‘8’ to machine cycle ‘9’. In this time frame the RESET switches of comparator stage 1 and 2 are open; the RESET switch of comparator stage 3 is closed for auto-zeroing. An analog input signal with sufficient slew rate may couple through to comparator stage 3 via the coupling capacitors of stage 1 and 2. The high sensitivity comes from the fact that the signal is amplified by comparator stages 1 and 2 before it reaches the input of comparator stage 3.

When the saturation voltage is stored on the coupling capacitors, the following comparator stage is not useful anymore to determine the sign of $|V_{IN} - V_{DAC}|$. Suppose the coupling capacitors on the input of the second comparator stage are charged to the saturation voltage V_{SAT}. The differential output voltage of the first comparator stage will be $A_1 \times (V_{IN} - V_{DAC})$. This signal is fed to the input of the second comparator stage whose output signal will be $A_2 \times [A_1 \times (V_{IN} - V_{DAC}) \pm V_{SAT}]$. Since the differential output voltage of the comparator stages can never be higher than $\pm V_{SAT}$, the output of this comparator stage will stay at its saturation level, independent of the value of $(V_{IN} - V_{DAC})$.

The only way to avoid the error mentioned above is to limit the slew rate during the sampling interval (machine cycle 2 to 9). For the ADC in the 80C552 the slew rate of the analog input signal must be lower than 10V/ms.

From the discussion above, it becomes evident that it is essential that the slew-rate of the input signal is limited to 10V/ms. Although 'clean' DC signals may be applied to the ADC, noise spikes or crosstalk from neighboring signals may still result in signal components with a slew-rate >10V/ms on the DC signal during the sampling interval.

The following measures can be taken to reduce the slew rate on analog input signals:

- Supply the analog signal from a source with low output impedance. This will reduce the sensitivity to cross-talk.
- Keep the analog input signal lines away from digital signal lines. Analog signals may be screened from digital signal lines with a grounded guard ring on the PCB.
- Do not mix analog and digital signals on P5 pins.
- Connect an RC filter to the analog inputs. The time constant should be ≥500µs.

For a $5V_{PP}$ input sine-signal, the slew rate constraint of 10V/ms results in a maximum input frequency of 637Hz. When we use the Nyquest criterion ($f_{SAMPLE} \geq 2*f_{SIGNAL}$), the maximum input signal frequency is 1kHz when the 80C552 runs on 1.2MHz (1 conversion every 500µs).

This shows that the maximum input signal frequency for the ADC of the 80C552 is determined by the **slew-rate**. So, increasing the XTAL frequency on which the 80C552 is operating, does not automatically imply that the maximum input signal frequency also scales to a higher value. This scaling is only allowed for signals with a slew rate <10V/ms.

3.1.3 Analog Signal Drive

The output resistance of the analog signal source should be small enough not to add a significant error to the conversion result. The output impedance has two effects on the accuracy of the conversion, that is, the voltage drop over the source resistance and the time constant to charge the sampling capacitor.

1: The voltage drop over the output impedance due to the input current of the ADC. In the 80C552, the input current is a leakage current. this leakage current is specified as less than 1µA. Practically, however, the leakage current will be less than 100nA.

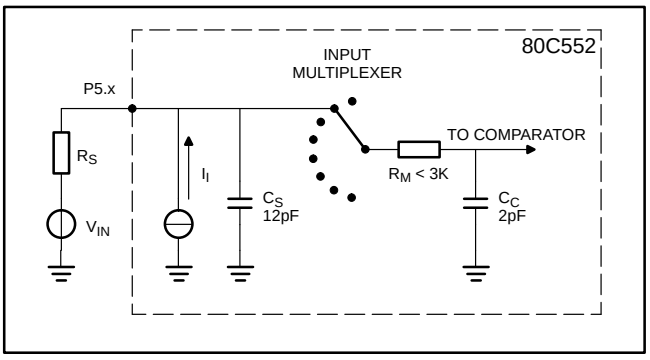


Figure 7.

Figure 7 shows the input circuit. The leakage current comes mainly from circuitry directly connected to the P5.x pin. Compared with this leakage current, the input current of the comparator can be neglected. C_S represents the contribution of the stray capacitances; C_C represents the sampling capacitor. Before the sampling capacitor, there is the series resistance R_M of the analog multiplexer.

The output resistance R_S of the input signal source will cause a voltage drop due to the input leakage current. The voltage that will be converted is the voltage on the sampling capacitor. This voltage is the input voltage V_{IN} minus the voltage drop over R_S. This voltage drop will give an error contribution in the conversion result of V_{IN}.

When an accuracy of $^1/_2$LSB is required, the maximum source resistance R_S is:

$$R_S \quad \frac{0.5LSB}{I_I}$$

Example: If V_{REF} = 5.12V and I_I = 1µA, the source resistor should be less than 2.5kΩ.

When this constraint on output resistance of the signal source cannot be met, the analog signal should be buffered with a buffer of sufficiently low output resistance. this buffer should be placed as close as possible to the analog source. the longer leads from buffer to the ADC input will be less sensitive to cross-talk (low impedance source resistance) than long leads from signal source to buffer (high impedance source resistance). Filtering may be included in the buffer stage to limit the slew-rate of the signal to <10V/ms.

2: The ability to charge the sampling capacitor within the sampling interval. Figure 7 also shows the dimensions of the capacitances as seen from the analog input. The capacitances consist of stray-capacitances and the actual sample capacitance. These capacitances must be charged within 4 machine cycles (machine cycle '2' until '5'), which will put a constraint on the maximum source resistance.

Example: An input signal with a slope of 10V/ms applied to the ADC input. For simplicity, assume that the capacitance that must be charged via R_S is 14pF and the 80C552 is running on 30MHz. At this frequency, the available charging time for the capacitor is 1.6µs. With the given slew-rate and the charging time of the capacitor, the analog voltage V_{IN} has changed 16mV. The response of an RC-network on an input ramp signal is:

$$A\ t - RC\left(1 - e^{-\frac{t}{RC}}\right)$$

A is slew-rate of input signal; RC is time constant of input RC-network.

The term:

$$ARC\left(1 - e^{-\frac{t}{RC}}\right)$$

represents the deviation of the capacitor voltage from V_{IN}.

If this deviation must be less than $^1/_2$LSB (is 2.5mV at V_{REF} = 5.12V) after 1.6µs, then the RC-time must be less than 0.25µs. Given that R_S = 2.5kΩ and C = 124pF, the RC time of the input circuit of the 8XC552 is 35ns. Hence, there will be no significant error contribution because of the charging time of the input capacitance.

The two constraints on R_S mentioned above show that the effect of the input leakage current determines the maximum value for R_S.

Conclusion: R_S should not exceed 2.5kΩ.

3.2 Layout Considerations

Although this application note handles subjects related to the ADC, the following layout considerations are also valid for applications that do not use the ADC. For more general information on PCB layout design, see Reference 2.

3.2.1 Decoupling

The analog and digital circuit parts in the 80C552 have their own set of power supply pins. Mutual inductance between on-chip AV_{DD} and AV_{SS} signal lines will cause the analog AC current I_A to flow in the AV_{DD} and out of the AV_{SS} pin. The same is true for the digital AC current I_D in the V_{DD} and V_{SS} pins (Figure 8).

Because this mutual inductance is harder to realize off-chip, a low-impedance signal path must be created for both I_A and I_D. This is realized with decoupling capacitors between AV_{DD} and AV_{SS} for the analog signal part; a decoupling capacitor between V_{DD} and V_{SS} will decouple the digital signal part.

To ensure a low impedance ground path, the use of a ground plane is recommended. The decoupling capacitors (for example, 100nF ceramic capacitors) must be placed as close as possible to the AV_{DD} and V_{DD} to minimize the loop area of the supply currents. Series inductors in the power supply lines may be used to improve decoupling (for example, 1..5µH). Using this decoupling scheme, both analog and digital supply connections can be connected together to a single (stable) +5V.

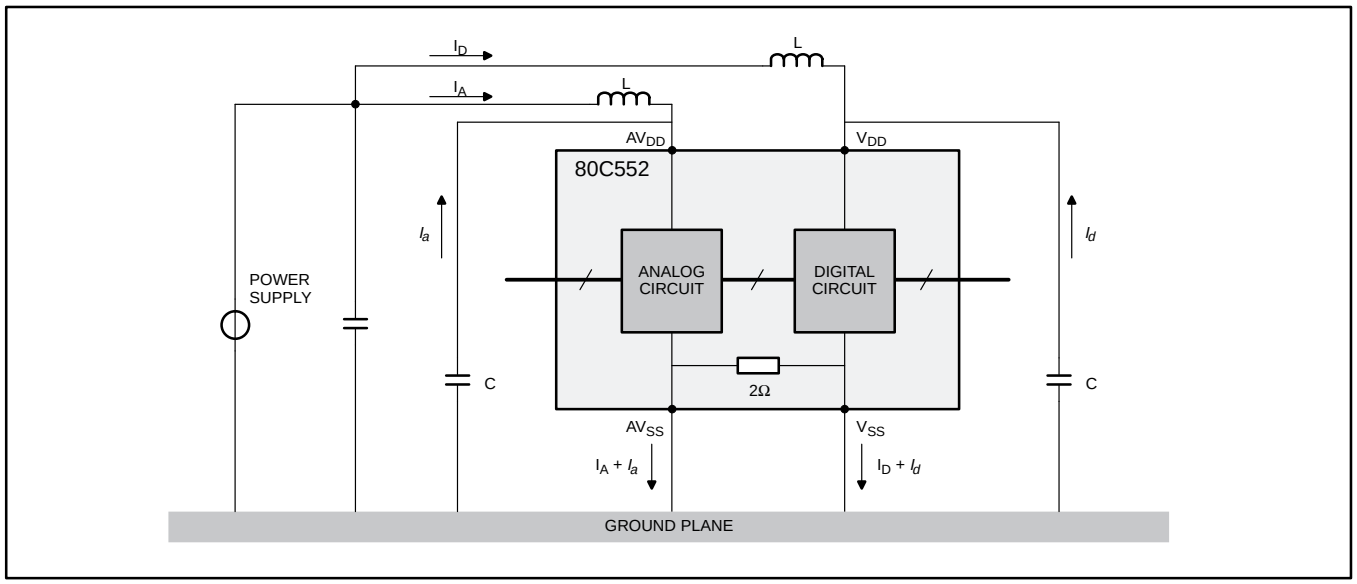


Figure 8.

3.2.2 Grounding

When using both analog and digital circuits on the same PCB, it is common practice to isolate the analog power supply and ground as much as possible from the digital power supply and ground. This will reduce crosstalk via common ground impedances. Common impedances may result in noise in the (sensitive) analog circuitry caused by crosstalk of noise that originates from the digital circuitry.

At some point however, both analog and digital grounds must be tied together. When this takes place far from the microcontroller, it will increase impedance between the analog and digital ground connections. This can create a considerable differential voltage between the analog and digital ground lines. Analog and digital circuitry inside the microcontroller will operate at different ground levels and improper functioning may be the result. A second effect of large ground impedance is that the crosstalk between the analog and digital circuits does not take place via the common ground impedance anymore, but via internal parasitic capacitances and substrate contacts.

To prevent differences in ground levels, the analog and digital ground inside the 80C552 are connected together via an impedance of ±2Ω (Figure 8). This will keep both grounds at the same DC level. This impedance does not mean that now a common ground pin for analog and digital ground currents can be used. Impedance of the common bondwire will cause ground bounce, and thus crosstalk between digital and analog circuits.

When the supply lines are properly decoupled, mutual inductance between the on-chip supply traces will assure that the AC-part of the supply current that flows inside AV_{DD} and V_{DD} will leave via the AV_{SS} and V_{SS} pin even though AV_{SS} and V_{SS} are connected together via a 2Ω impedance. The best place on the PCB to connect AV_{SS} and V_{SS} to each other is directly outside the microcontroller. This is also the best place for the ground connection of the decoupling capacitors. If the ground connection is to a ground plane, a ground plane on the component side is preferred.

Two separate grounds are needed if the application has more analog ICs on the PCB, apart from the 80C552. In that case, the AV_{SS} and V_{SS} of the 80C552 should be connected to the analog ground. The digital ground may be the return path for, for instance, line drivers. This will result in a ground that is less ‘clean’ than the analog ground. For the digital circuits this less ‘clean’ ground is less of a problem because they always have a certain noise margin. An alternative is to create a ‘star-point’. This is a connection between AV_{SS} and V_{SS} at the ADC area of the 80C552. Care should be taken to avoid ground loops in the other circuit parts up to the power supply.

If the 80C552 application also uses external digital circuits, noise margin may be lost due to possible different ground levels. This can be reduced by the connecting two anti-parallel diodes close to the ground pins of the 80C552 and the connected digital circuits (Figure 9).

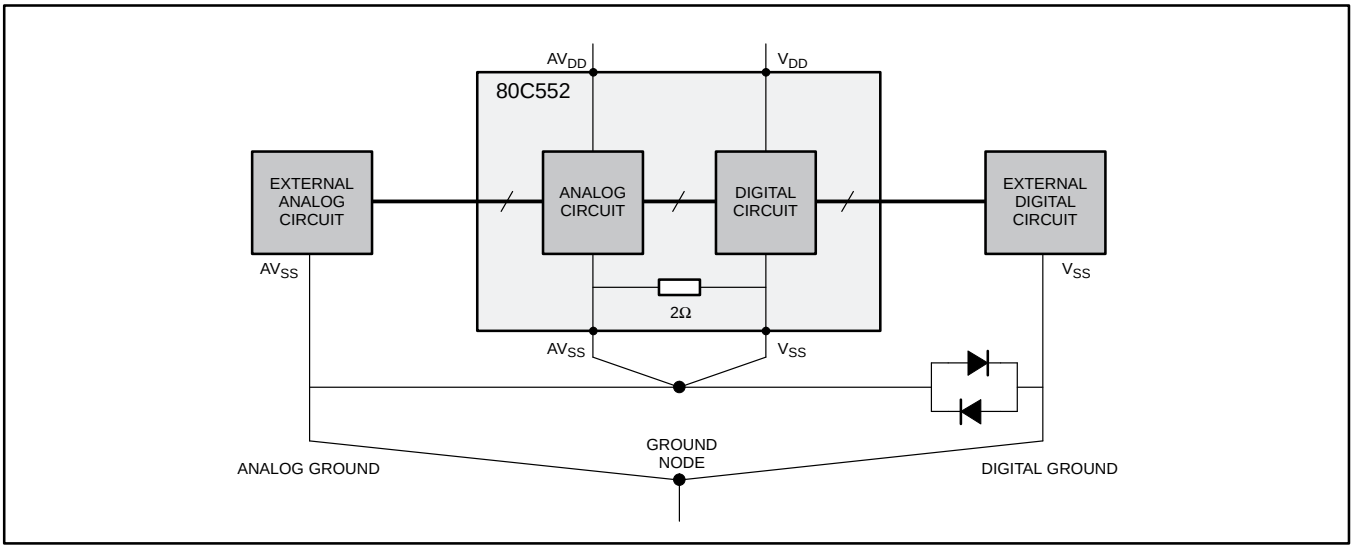


Figure 9.

3.2.3 Placement of External Components

External circuits connected to the microcontroller should be p;aced as close as possible to the microcontroller. This will reduce signal loops and common impedances on which high frequent signals may occur. These signals may come from an external source or may be generated by the microcontroller itself. When the disturbing signals are coming from an external source, they may cause improper functioning of the microcontroller. If originating from the microcontroller, they will cause unwanted radiation.

When a common ground plane cannot be implemented, the microcontroller and circuits directly connected to the 80C552 must have a local groundplane (Figure 10). This local ground should have a connection to the main ground of the application at some point.

If, for layout reasons, it is not possible to place the external circuitry close the the microcontroller, an RC-filter may be placed between the microcontroller and the external circuit (Figure 10). When microcontroller lines are connected to Off-PCB circuits, it is also advised to connect an RC network to the I/O line. The reason is sensitivity of the microcontroller to short (4 . . 5ns) high-voltage (30 . . 40V) spikes. Although these pulses will not damage the microcontroller, they may be responsible for incorrect functioning. With microcontrollers becoming faster, the on-chip I/O drivers will have increased bandwidths, hence become more sensitive for short spikes.

The resistor of the filter should be connected as close as possible to the output of the driving device. The capacitor of the filter should be connected as close as possible to the input of the receiving device. Also the ground connection of the capacitor must be connected as close as possible to the ground of the receiving device.

Values of the resistor of the filter depend on the drive capability of the outputs and the input impedances/levels of the inputs connected to the filter. The series resistor will reduce some noise margin. Typical capacitor values are 470pF. When connected to 80C552 inputs, the resistor value may be 1k; for 80C552 outputs 100Ω may be used.

NO (filter)-capacitors should be connected directly to outputs! This will cause (dis)charge currents to flow in the supply and ground lines. This can cause severe noise problems in the application.

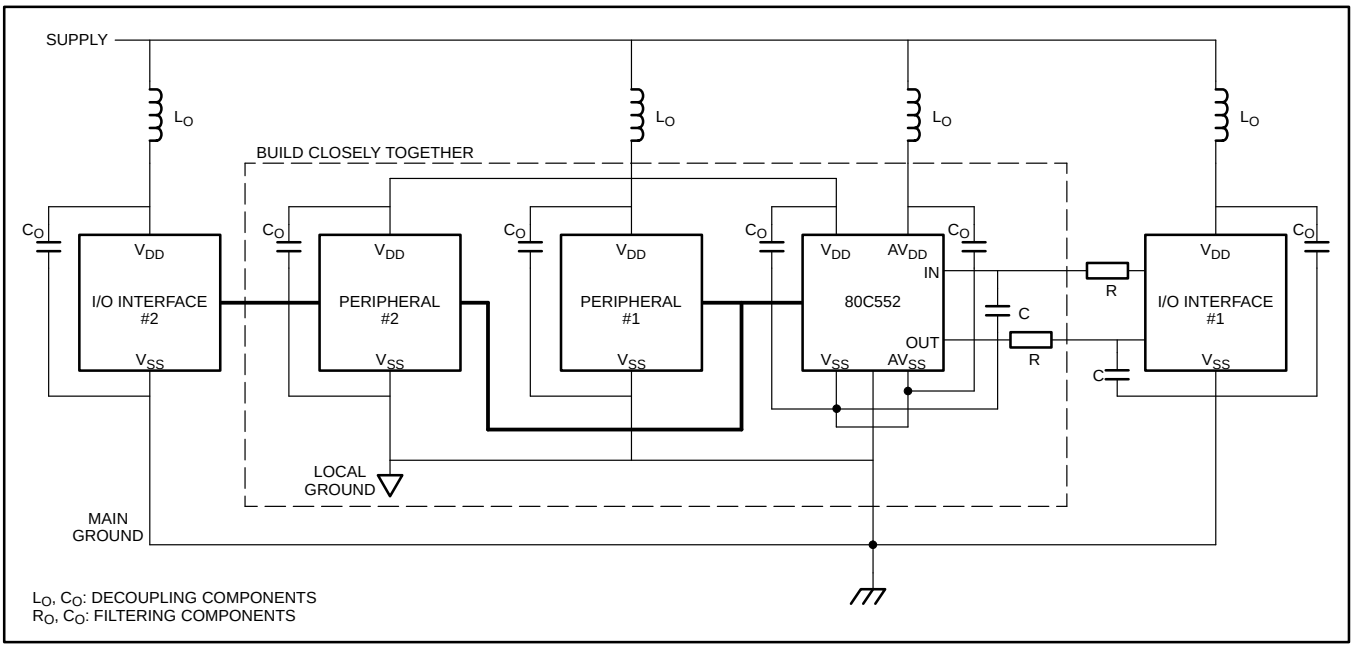


Figure 10.

4.0 PROGRAM EXAMPLES

Two program examples are given that show how to operate the ADC with software. The sources are written in 'C'. The program 'ADC_pol.c' works on polling basis; the program 'ADC_int.c' works on interrupts.

The programs will start scanning all ADC inputs when a rising edge appears on the STADC pin. This can be realized with a resistor from ground to STADC and a switch from STADC to V_{DD}. If STADC is connected to P4.0, a conversion of all channels will start every 1.14ms. Timer T2 controls the timing of P4.0. When all analog input signals on P5 are converted, the results will be sent to the UART and be made visible on a terminal. The communication part of the program is given in the file 'output.c'. After compiling and assembling 'output.c', it should be linked to 'ADC_pol' or 'ADC_int'.

The required terminal settings are:

- 8 data bits
- no parity
- 1 STOP-bit
- 19200 baud.

The 80C552 must run on 11.0592MHz.

The following points are important when writing code for the ADC:

- ADCS and ADCI must be cleared, before programming AADC0..AADR2.
- Channel selection bits AADR0..AADR2 must be programmed before setting ADCS (software start) or ADEX (enabling hardware start)
- When working in polling mode, use ADCI to test if the conversion is finished.
  - **Do not** test 'ADCS=0' for this purpose.
  - When and ADC conversion is initiated by software, there is a delay of 2 machine cycles between the moment of writing a '1' to ADCS and reading a '1' from the ADCS-bit in ADCON.
  - When an ADC conversion must be initiated by a rising edge on ADEX, it is not known when ADCS becomes '1'. This depends on the external signal that starts the conversion.

The following development tools were used on a PC (DOS 5.0):

TOOL	TYPE	PHILIPS NUMBER
C-compiler	BSO/Tasking V2.1	OM4136
Assembler	BSO/Tasking V3.3	OM4142
Emulator	SDS+80C552 probe	OM4120S +OM1092 +OM1095
Debugger	BSO/Tasking XRAY51 V1.4d	OM4129

ADC_pol.c

```
/*************************************************************************
*
***
*0*     MODULE             : adc_pol.c
***
*0*     FILENAME           : adc_pol.c
***
*0*     APPLICAITON        : Demo code for ADC of 8xC552 polling
*0*                          mode
***
*0*     PROGRAMMER         : T. v. Daele
***
*0*     DESCRIPTION        : After rising edge on STADC–pin, all
*0*                          ADC channels are scanned.
*0*                          Rising edges are available on P4.7 at a
*0*                          repetition rate of 1.14ms. This timing is
*0*                          controlled by T2.
*0*                          Results are sent to UART.
***
*************************************************************************
/


#define ADEX     0x20
#define ADCI     0x10
#define ADCS     0x08

void write_UART (unsigned int *, unsigned int);
```

```
void main(void)
{
 unsigned int conversion,result_ADC[8];
 unsigned char ADC_Channel;

 S0CON=0x40;                          /* 8 bits, no parity, 1 STOPbit */
 TH1+TL1+0xfd;                        /* 19200 Baud @11.0592MHz */
 PCON=0x80;
 TMOD=0x20;
 TR1=1;

 TM2CON=0x0d;                         /* Source T2: osc/96 */
 RTE=0x80;                            /* Overflow rate: 0.569ms
                                         P4.7 toggles every 0.569ms
                                         ADC conversion on rising edge STADC
                                         P4.7/STADC: 1.14ms conversion rate
*/
 conversion=0;

 while(1)
 {
  for (ADC_Channel=0;ADC_Channel<8;ADC_Channel++)
  {
   ADCON=0;                           /* Make sure ADCI and ADCS are cleared
*/
   ADCON=ADC_Channel;                 /* before ADC channel is selected */

   if (ADC_Channel==0)
      ADCON=ADEX;                     /* ADC0: External start */
   else
      ADCON+ADCON|ADCS;               /* ADC1..ADC7: Software start */
   while((ADCON&ADCI)==0);            /* Wait till conversion finished
                                         by checking ADCI */
   result_ADC[ADC_Channel]=5*((256*ADCH+(ADCON&0xc0))>>6);
                                      /* Calculate 10-bits binary result
                                         relative to 5.12V ref */
  }
  write_UART(&result_ADC,conversion++); /*Output results to UART */
  if (conversion==10000)
     conversion=0;
 }
}
```

ADC_int.c

```
/*************************************************************************
*
***
*0*     MODULE             : adc_int.c
***
*0*     FILENAME           : adc_int.c
***
*0*     APPLICATION        : Demo code for ADC of 8xC552 in interrupt
*0*                          mode
***
*0*     PROGRAMMER         : T. v. Daele
***
*0*     DESCRIPTION        : After rising edge on STADC–pin, all
*0*                          ADC channels are scanned.
*0*                          Rising edges are available on P4.7 at a
*0*                          repetition rate of 1.14ms. This timing is
*0*                          controlled by T2.
*0*                          Results are sent to UART.
***
*************************************************************************
/

#define ADEX    0x20
#define ADCI    0x10
#define ADCS    0x08
#define ADCIn   0xref
#define FALSE   0
#define TRUE    1

void write_UART(unsigned int *, unsigned int);
bit conversion_finished;

void main(void)
{
 unsigned int conversion,result_ADC[8];
 unsigned char ADC_channel;

 S0CON=0x40;                          /* 8 bits, no parity, 1 STOPbit */
 TH1=TL1=0xfd;                        /* 19200 Baud @11.0592MHz */
 PCON=0x80;
 TMOD=0x20;
 TR1=1;

 TM2CON=0x0d;                         /* Source T2; osc/96 */
 RTE=0x80;                            /* Overflow rate: 0.569ms
                                         P4.7 toggles every 0.569ms
                                         ADC conversion on rising edge STQADC
                                         P4.7/STADC: 1.14ms conversion rate
*/

 EAD=1;                               /* Enable ADC interrupt */
 EA=1;

 conversion_finished=FALSE;
 ADC_channel=conversion=0;

 ADCON=0;                             /* First conversion; external start */

 ADCON=ADEX;
 while(1)
 {
  if (conversion_finished==FALSE)
     {
      /* User code executed while conversion is in progress  */
     }
```

```
  else
     {
      result_ADC[ADC_channel]=5*((256*ADCH+(ADCON&0xc0))>>6); /* Store
result */
      if (ADC_channel!=7)
        {
         /* Prepare conversion of next channel */
         ADCON=++ADC_channel;
         ADCON+ADCON|ADCS;
        }
      else
        {
         /* ADC0..ADC7 is converted. Send results to UART */
         write_UART(&result_ADC,conversion++);
         if (conversion==10000)
            conversion=0;
         ADC_channel=0;
         ADCON=0;                        /* Prepare next scan */
         ADCON=ADEX;
        }
     conversion_finished=FALSE;
    }
 }
}


interrupt 10 using 1 void ADC(void)
{
 ADCON=ADCON&ADCIn;                      /* Clear ADCI flag */
 conversion_finished=TRUE;
}
```

output.c

```
/*************************************************************************
***
*0*     MODULE             : output.c
***
*0*     FILENAME           : output.c
***
*0*     APPLICATION        : Example program for 80C552 ADC
***
*0*     PROGRAMMER         : T. van Daele
***
*0*     DESCRIPTION        : The results of the conversion are written
*0*                          to the 80C552 UART.
***
*0*     FUNCTIONS          : <name>               <description>
*0*                          write_UART           entry point
*0*                          send_byte            trx byte
*0*                          decode               trx binary nibble
*0*                          send_bin_byte        trx binary byte
*0*                          send_dec_int         trx decimal integer
*0*                          send_string          trx aSCII string
***
*************************************************************************/

rom char string_0[]    =       "Conversion #";
rom char string_1[]    =       ": (Ref is 5.12V)";
rom char string_2[]    =       "ADC_Channel # ";
rom char string_3[]    =       "mV";
rom char string_4[]    =       ": ";
rom char new_line[]    =       "\r\n";

/*************************************************************************
***
*1*     FUNCTION           : send_byte
***
*2*     SYNOPSYS           : send_byte(src_byte)
***
*3*     ARGUMENTS          : type                 name
*3*                          char                 src_byte
***
*4*     RETURNS            : nothing
***
*5*     MODIFIES           : nothing
***
*6*     DESCRIPTION        : Send byte to terminal via UART
*6*                          Wait till transmission is finished
***
*7*     HISTORY            : data          who             description
*7*                          05-02-93      tvd             initial
***
*************************************************************************/

void send_byte(char src_byte)
{
 S0BUF = src_byte;      /* Byte to transmit */
 while (TI == 0);       /* Wait till byte is transmitted */
 TI = 0;                /* Clear transmit flag */
}
```

```
/*************************************************************************
***
*1*     FUNCTION           : decode
***
*2*     SYNOPSYS           : decode(char src_nibble)
***
*3*     ARGUMENTS          : type                 name
*3*                          char                 src_nibble
***
*4*     RETURNS            : nothing
***
*5*     MODIFIES           : nothing
***
*6*     DESCRIPTION        : Decode least significant nibble to
*6*                          ASCII and transmit
***
*7*     HISTORY            : data          who             description
*7*                          05–02–93      tvd             initial
***
*************************************************************************/
void decode(char src_nibble)
{
 if ( src_nibble < 0x0a)
     send_byte(src_nibble + 0x30);
 else
     send_byte(src_nibble + 0x41 – 0x0a);
}

/*************************************************************************
***
*1*     FUNCTION           : send_bin_byte
***
*2*     SYNOPSYS           : send_bin_byte(char src_byte)
***
*3*     ARGUMENTS          : type                 name
*3*                          char                 src_byte
***
*4*     RETURNS            : nothing
***
*5*     MODIFIES           : nothing
***
*6*     DESCRIPTION        : Split a binary byte in nibbles, decode
*6*                          to ASCII and transmit
***
*7*     HISTORY            : data          who             description
*7*                          05–02–93      tvd             initial
***
*************************************************************************/
void send_bin_byte(char src_byte)
{
 decode((src_byte>>4) & 0x0f);      /* Get ms_nibble */
 decode(src_byte & 0x0f);           /* Get ls_nibble */
}
```

```
/*************************************************************************
***
*1*     FUNCTION           : send_dec_int
***
*2*     SYNOPSYS           : send_dec_int(unsigned int src_wrd)
***
*3*     ARGUMENTS          : type                  name
*3*                          unsigned              src_wrd
***
*4*     RETURNS            : nothing
***
*5*     MODIFIES           : nothing
***
*6*     DESCRIPTION        : Decode binary integer to decimal and
*6*                          transmit
***
*7*     HISTORY            : data          who             description
*7*                          07-07-93      tvd             initial
***
*************************************************************************/
void send_dec_int(unsigned int src_wrd)
{
 unsigned char a,b,c,d,e;

 a=src_wrd/1000;                        /* a='thousands' */
 b=((src_wrd%1000)/100);                /* b='hundreds' */
 c=((src_wrd%100)/10;                   /* c='tens' */
 d=src_wrd%10;                          /* d='units' */
 e=16*c+d;                              /* Print value for tens and units */

 /* Print integer without leading zero's */
 if (a==0)
    {
     send_byte(0x20);
     if (b==0)
        {
         send_byte(0x20);
         if (c==0)
           {
            send_byte(0x20);
            decode(d);
           }
         else
            send_bin_byte(e);
        }
     else
       {
        decode(b);
        send_bin_byte(e);
       }
    }
 else
    {
     send_bin_byte((16*a)+b);
     send_bin_byte(e);
    }
}
```

```
/*************************************************************************
***
*1*     FUNCTION           : send_string
***
*2*     SYNOPSYS           : send_string(rom char *str_ptr)
***
*3*     ARGUMENTS          : type                  name
*3*                          rom char *            str_ptr
***
*4*     RETURNS            : nothing
***
*5*     MODIFIES           : nothing
***
*6*     DESCRIPTION        : Send a string of characters from ROM to
*6*                          terminal.
***
*7*     HISTORY            : data          who             description
*7*                          05-02-93      tvd             initial
***
*************************************************************************/
void send_string(rom char *str_ptr)
{
 while (*str_ptr != 0)
       send_byte(*(str_ptr++));    /* Send byte */
}

/*************************************************************************
***
*1*     FUNCTION           : write_UART
***
*2*     SYNOPSYS           : write_UART(unsigned int *ADC_result,
***                                     unsigned int conversion_cnt)
***
*3*     ARGUMENTS          : type                  name
*3*                          unsigned int *        src_ptr
*3*                          unsigned int          msg_ptr
***
*4*     RETURNS            : nothing
***
*5*     MODIFIES           : nothing
***
*6*     DESCRIPTION        : Decode results to correct format and send
*6*                          to UART
***
*7*     HISTORY            : data          who             description
*7*                          30-06-93      tvd             initial
***
*************************************************************************/
void write_UART(unsigned int *result_ptr, unsigned int conversion_cnt)
{
 unsigned char cnt;

 send_string(new_line);
 send_string(new_line);
 send_string(string_0);            /* Send message number */
 send_dec_int(conversion_cnt);
 send_string(string_1);
 for (cnt=0;cnt<8;cnt++)
 {
  send_string(new_line);
  send_string(string_2);
  decode(cnt);                     /* Send channel number */
  send_string(string_4);
  send_dec_int(*(result_ptr++));   /* Send result to UART */
  send_string(string_3);
 }
}
```