A new A/D converter design. Bit-block converters.

Andras Tantos

This paper presents a new analog to digital converter implementation. The technique described here based on the well-known process to convert base-10 fractional numbers between 0 and 1 to their base-2 representation. Different configurations featuring different speed, complexity and accuracy will be introduced. Converters with continuous time analog domain and logarithmic output will also presented.

Introduction

In the past various different methods has been invented to covert analog values to their digital representation. Devices used various electric or mathematic approaches to convert an analog voltage or current value to their digital representation. The class of converters introduced in this paper use a mathematical approach. All share one common central idea: a building block, that produces one bit of digital information (or in one variant one digit of information) at a time. Using this building block many different configurations can be constructed, with different trade offs in speed, complexity and accuracy. I will describe a configuration which can achieve comparable speed to successive approximation converters with constant complexity (i.e. the complexity of the converter is independent of the resolution). I will also describe a converter that performs comparable to (pipelined) flash converters with $\log_2(n)$ complexity (the complexity is proportional to the number of output bits, not the number of output values). Further modifications of the basic idea lead to a continuous-time A/D converter where the sampling occurs in the digital domain, and to a converter that performs logarithmic conversion.

The basic idea

It is a well-known technique to convert fractional base-10 numbers between 0 and 1 into their base-2 representation. I will first describe it shortly just for completeness:

Let's have a number N_1 between 0 inclusive and 1 exclusive in it's base-10 representation. We need it's base-2 digits. It's first digit is off course 0 and than comes the fractional sign, '.'. The first digit after the fractional sign is '1' if $N_1 >= 0.5$ and '0' otherwise. Lets subtract 0.5 from N_1 if it is greater than 0.5, and multiply this by 2. This way we get a new number, N_2 between 0 and 1 just as N_1 was. It's first digit after the fractional sign is the second of N_1, so we can deduce N_1 's second digit with the same method only use N_2 instead of N_1. We can continue this method until we reach N_i equals to 0 in which case all other digits are zeros, or we reach the desired accuracy. In formal:

1. Let's have a number $0 <= N_1 < 1$. Let's call our output number's i'th digit after the fractional sign O_i. Let's define an iteration counter i and initialize it with 1.
2. In iteration i, let $O_i = 1$ if $N_i >= 0.5$, 0 otherwise. Also let $N_{i+1} = 2* (N_i - O_i/2)$, which is: $N_{i+1} = 2*N_i - O_i$
3. Increment the iteration counter i with one and continue with step 2 until N_i becomes 0 or the desired accuracy is reached.

An example:

i	***N_i***	***O_i***	***N_{i+1}***
1	0.41	0	2 * 0.41 - 0
2	0.82	-1	2 * 0.82 - 1
3	2.64	-1	2 * 0.64 - 1
4	6.28	-1	2 * 0.28 - 0
5	13.56	-1	2 * 0.56 - 1
6	28.12	-1	...

So the first 6 digits of 0.41 in base-2 is 0.011010.

The main point is that we didn't need much knowledge about the source representation. We didn't actually use that the source representation was base-10. We didn't even use that the representation is a number. What we did use is a method to decide weather a value is greater than 0.5 or not, a method to multiply a value by two, and a method to decrease a value by 1 if desired. If these functions available in any source representation, we can do the conversion. All described functionality is available in the analog domain so there is a possibility to design a circuit that implements the above algorithm. This construct will be introduced in the next chapter.

The basic circuitry

The core of the algorithm described in the previous chapter is point 2. It can be considered as a black box, with one analog input (N_i), one analog output (N_{i+1}) and one digital output (O_i). For easier reference lets rename these signals as follows:

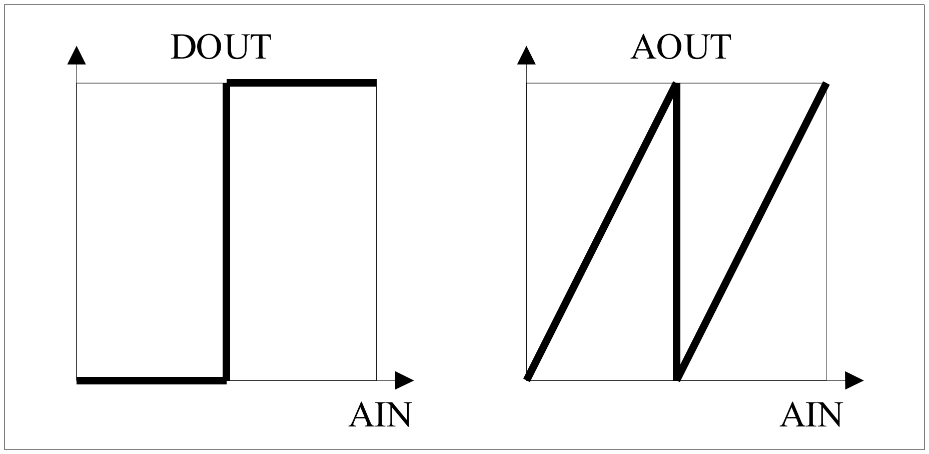
Old name	***New name***
N_i	**AIN**
N_{i+1}	**AOUT**
O_i	**DOUT**

We have the transfer functions also defined:

DOUT = (**AIN**>0.5)?1:0

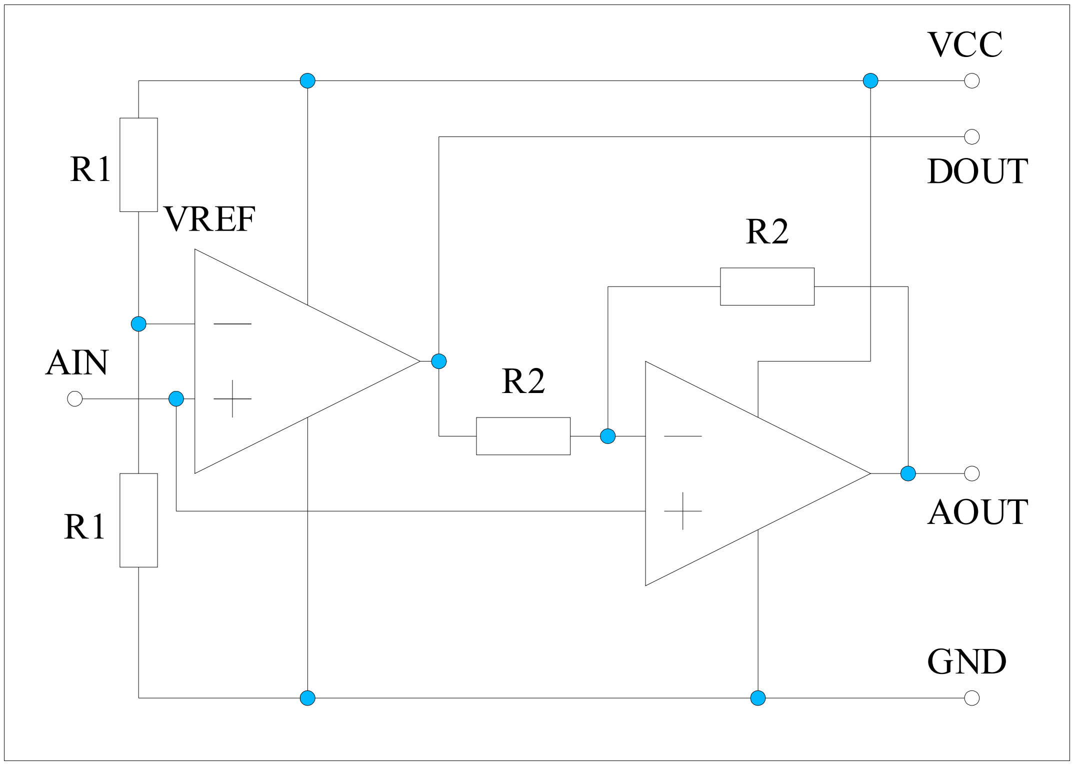
AOUT = 2***AIN** - **DOUT**

We can construct a plot for these functions:



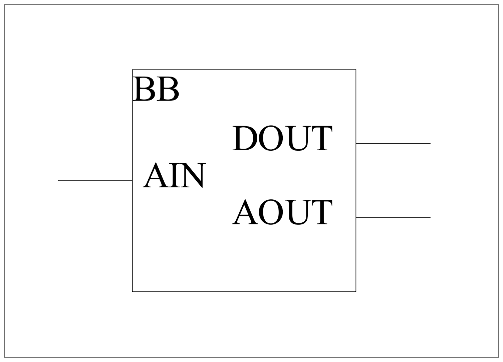
Bit-block's transfer function

And we can easily create a circuit that can implement those transfer functions:



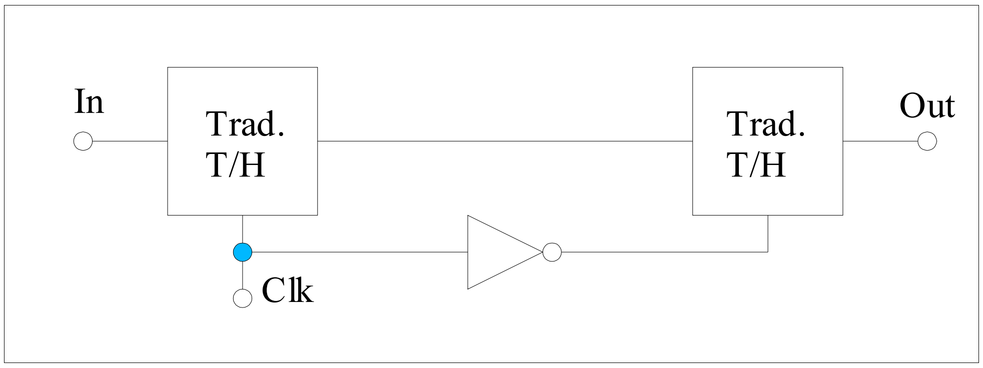
A simple bit-block implementation

In this design the operational amplifiers considered to be ideal except that their output saturates to the power lines. One possible interpretation of this circuit is a 1-bit A/D converter with a conversion error output amplified to full-scale. Lets call this building block as **bit-block**. Next, we will use this block in larger constructs, so it is practical to have a symbol represent of the above circuit:



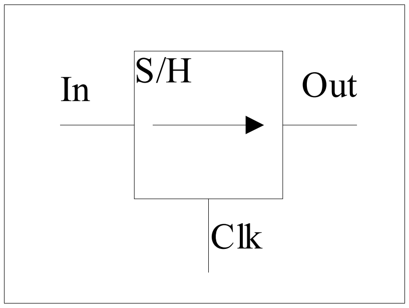
Bit-block symbol

We will also need a special track and hold circuit (a sample and hold circuit in its strict sense). Traditional T/H devices have two states. In state one (track) the input theoretically equals with the output. In state two (hold) the output equals the last input value of the track state. We will need a different behavior. The output should change only on state 1-2 transition and should keep it's value constant in both state one and two. The simplest implementation of this could be to cascade two traditional track and hold devices with inverted control input:



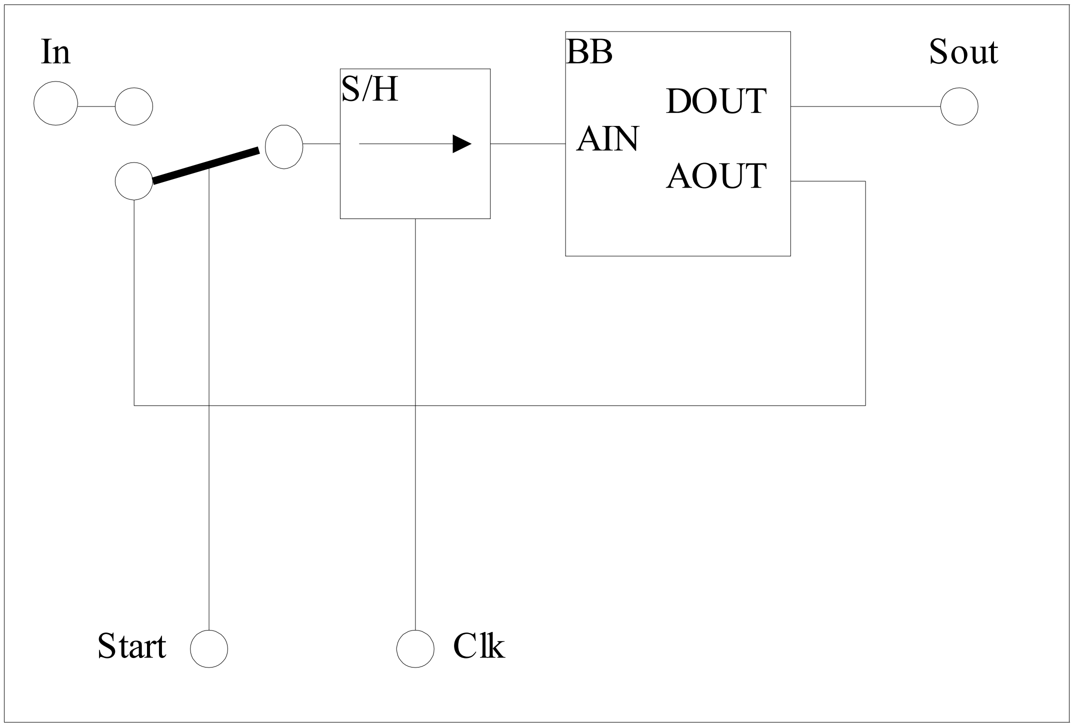
A simple sample-and-hold circuit

In the following figures this symbol will refer to the above circuit:



Sample-and-hold circuit symbol

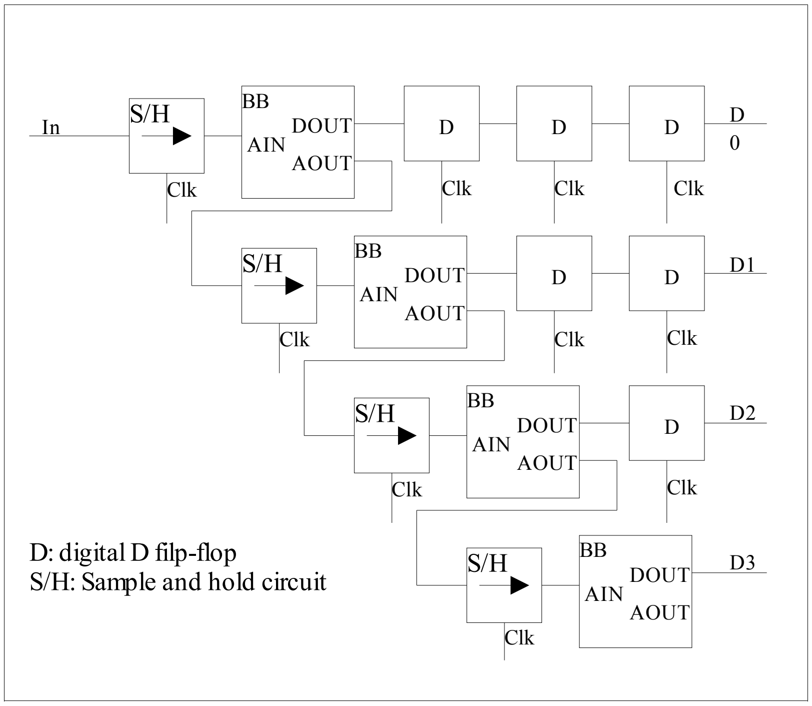
With these components at hand we can now construct the complete circuit that implements the described algorithm thus reassembling an A/D converter. Actually we can construct at least three basic structures. In the first case we use the strict algorithm definition above:



Converter design A

This circuit will work as an MSB first serial A/D converter. The analog input of the converter is **`In`**. The converter produces one bit on the output (**`Sout`**) for each clock pulse on **`Clk`**. The conversion starts with a one-clock wide pulse on the Start pin. The switch is shown in the position when Start is not active. This converter type provides more or less the same features for the outside world as a serial output successive approximation converter. Note however that the converter's complexity is independent of the precision of the conversion. One consequence of this feature is that the precision (the number of bits per sample) can be adjusted by control only and the same device can be used as an 8-bit or a 12-bit converter for example.

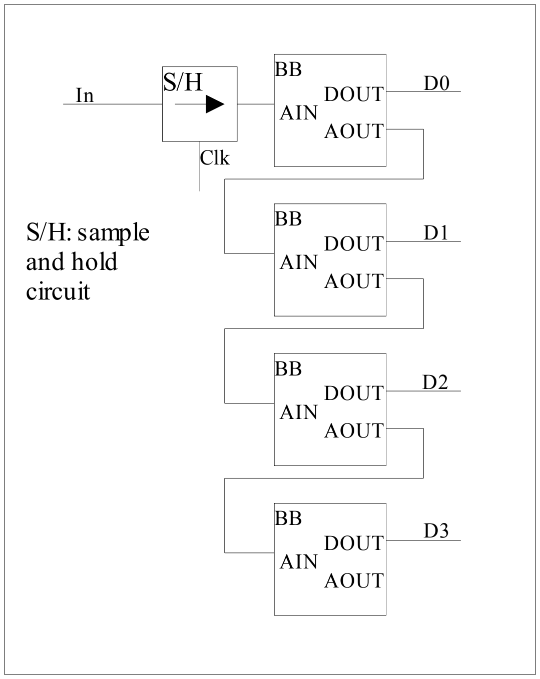
If we unfold the loop of the algorithm and implement each iteration with a unique set of components we get another layout:



Converter design B

This circuit will provide a parallel output value on **`D[0..3]`** of each pulse on **`Clk`**. This output will be the converted value of **`In`** four clock cycles before. Naturally the circuit can be expanded to any number of bits. This circuit reassembles a pipe-line A/D converter scheme. The complexity of the converter however is proportional to the number of output bits – except for the digital delay line which has a cubic complexity over the number of output bits - as opposed to the number of output values as in conventional flash-converters.

If we sacrifice some performance for simplicity we can leave most of the sample and hold circuits and D filp-flops out of the design. This will lead us to the third A/D converter implementation:



Converter design C

This converter scheme will produce one set of digital output on **`D[0..3]`** for each clock pulse on **`Clk`**. The output will correspond to the analog value on **`In`** at the time of last clock pulse. Thus this converter behaves like a flash converter with no pipe-lining effect.

Performance considerations

Design A has the unique feature to produce as many bits as desired. It also produces direct serial output, compatible with many current DSP's serial port. It can be used with such intelligent devices with nearly no additional logic. It's maximal conversion speed determined by the delay of the bit-block and the settling time of the sample and hold circuit. Of course it is also determined by the number of required bits:

$T_{min} = n * (T_{bb}+T_{sh})$, where *n* is the number of output bits per sample, T_{bb} is the delay of the bit-block and T_{sh} is the settling time of the S/H circuit.

It's *minimal* conversion speed is determined by the required precision and the fall of the output of the S/H circuit. The circuit's precision affected by the precision of the bit-block and the precision of the S/H circuits used.

Design B's speed is determined by the delay of the bit-block and the settling time of the S/H circuit. Digital delay lines will probably be much faster than S/H-s so their delay isn't a factor.

$T_{min} = T_{bb}+T_{sh}$

This converter will perform a complete conversion under this time, so it's n times faster than design A. The minimal conversion time is determined by the same featured as design A but it also dependent on the output word length (the number of cascaded S/H circuits). This means that this converter cannot be used at such slow conversion rates as design A. It's precision is affected by the precision of the bit-block and the S/H circuits, but also depends upon the similarity of the many bit-blocks and S/H's used.

Design C has a maximum conversion speed between design A and B. It's conversion time is basically determined by the sum of the delay of the bit-block:

$T_{min}= n * T_{bb}+T_{sh}$

It's minimal conversion time is equal to design A because both determined by the fall-time of the sample and hold circuit. The precision of this implementation is a function of the precision of one S/H circuit only and the precision and similarity of the bit-blocks. As a result it's precision is also between design A and B.

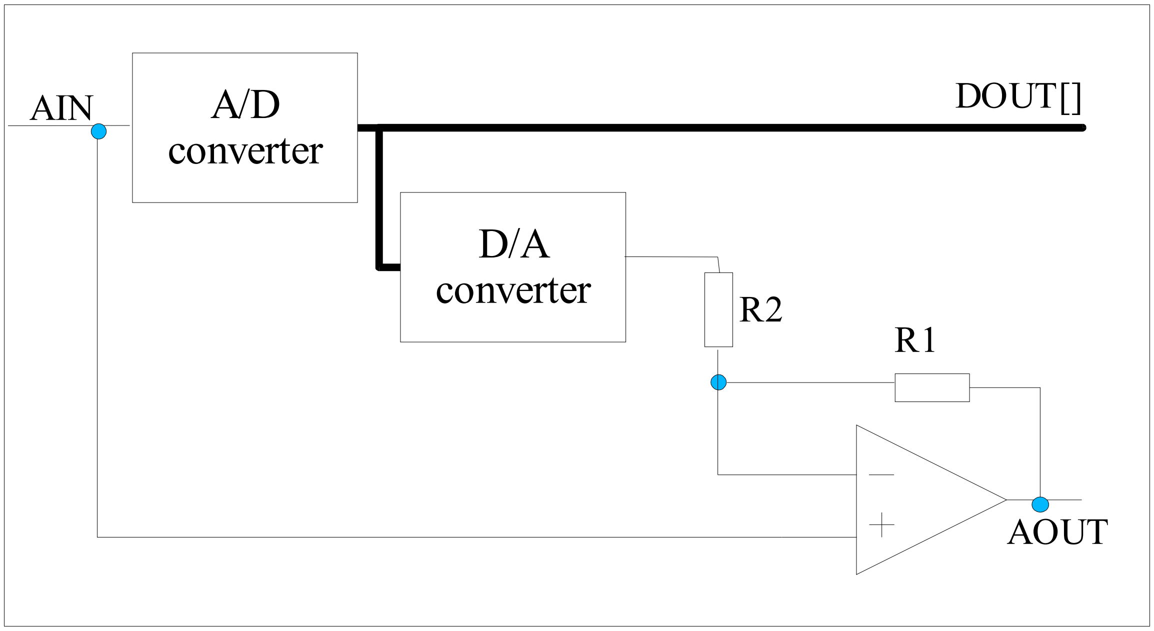
We can summarize the main features in the following table:

	Complexity	***Speed***	***Accuracy***
Design A	low	low	high
Design B	high	high	low
Design C	medium	medium	medium

Derivatives

Direct base-n converters

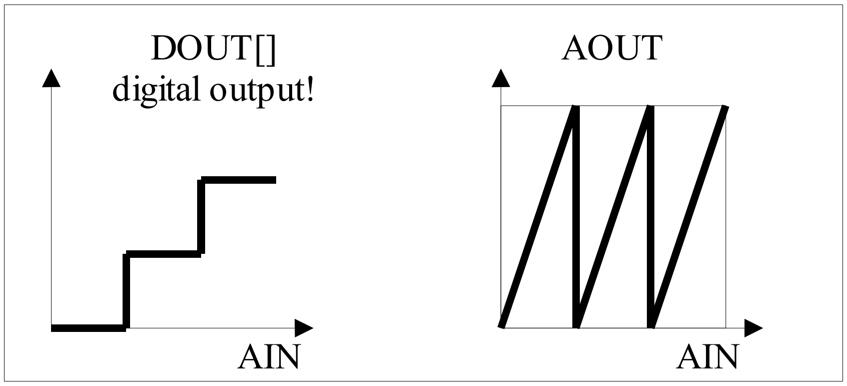
As mentioned earlier the bit-block can be considered as a one-bit A/D converter with an amplified conversion error output. Having this in mind one can generalize the bit-block to other than base-2 converters:



Base-n bit-block

The resistor values **`R1`** and **`R2`** can be calculated from the criteria that **`AOUT`**'s span must be equal to **`AIN`**'s span. Naturally the digital output **`DOUT`** now consists of more than 1 wires. You can use this bit-block in any of the three basic designs. One benefit is that you convert more than one bit in one step.

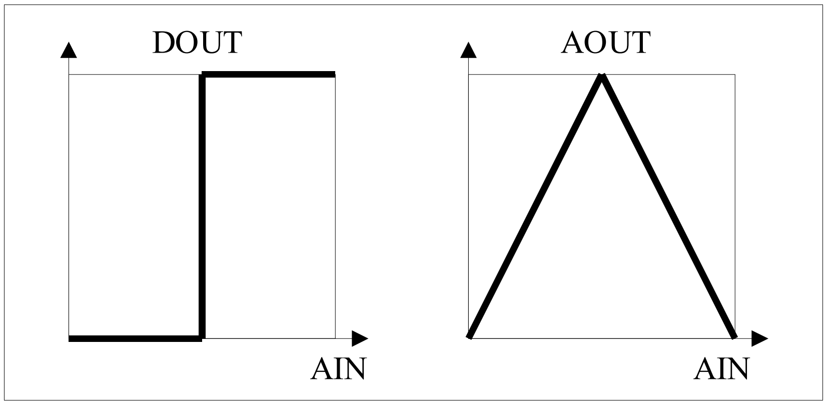
Another feature is that you are not tied to base-2 any longer. You can create for example a decadic flash converter at the A/D convert stage of the bit-block (9 comparators) and thus getting direct base-10 (for example BCD coded) output. As an example, let's see the transfer functions of a base-3 bit-block:



Base-n bit-block's transfer functions

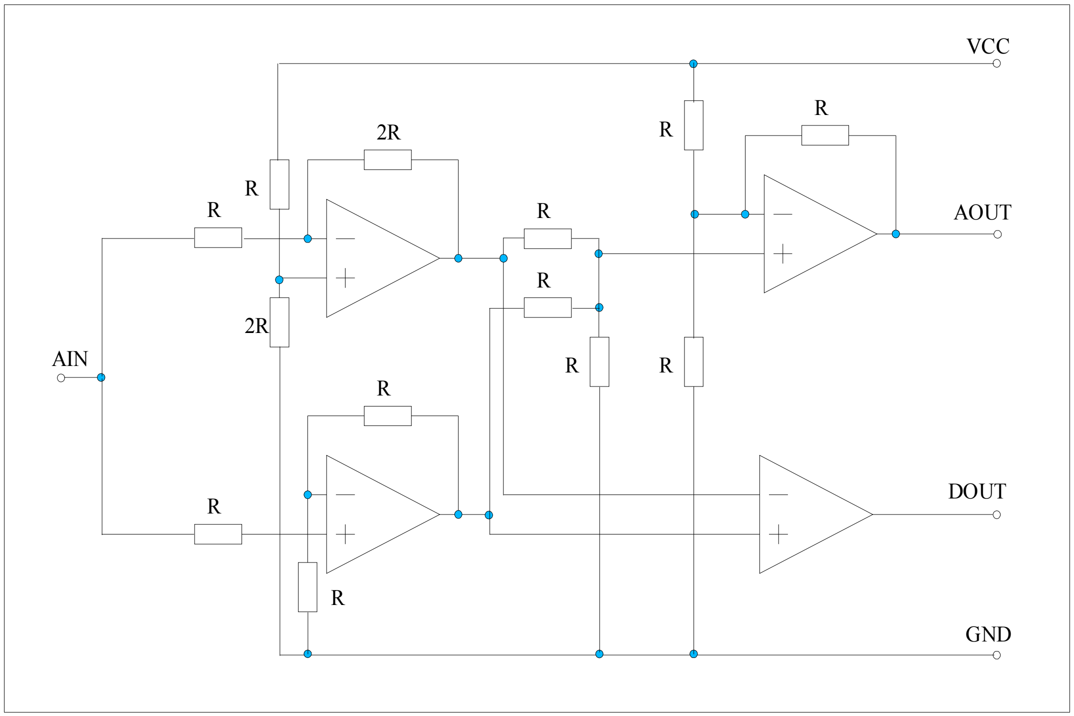
Continuous-time converters

With further modification of the transfer function of the bit-blocks we can construct a converter that can have a continuous-time analog and a discrete-time digital domain. To achieve this, we need to use a special coding of numbers, called Gray-codes. The coding has the feature that neighboring codes differ only in one digit. This code is widely used in places where asynchronous signals have to be sampled in a synchronous part of the system, like in positional encoders or signals crossing clock-domains. A Gray-coded number can be easily created from it's base-2 representation: If a bit is one than invert all lower bits of the number. Starting from the MSB bit and applying the previous modification to all bits downwards one can get the Gray-coded version of the original number. A small modification to the transfer characteristics of the bit-block can give us the same results:



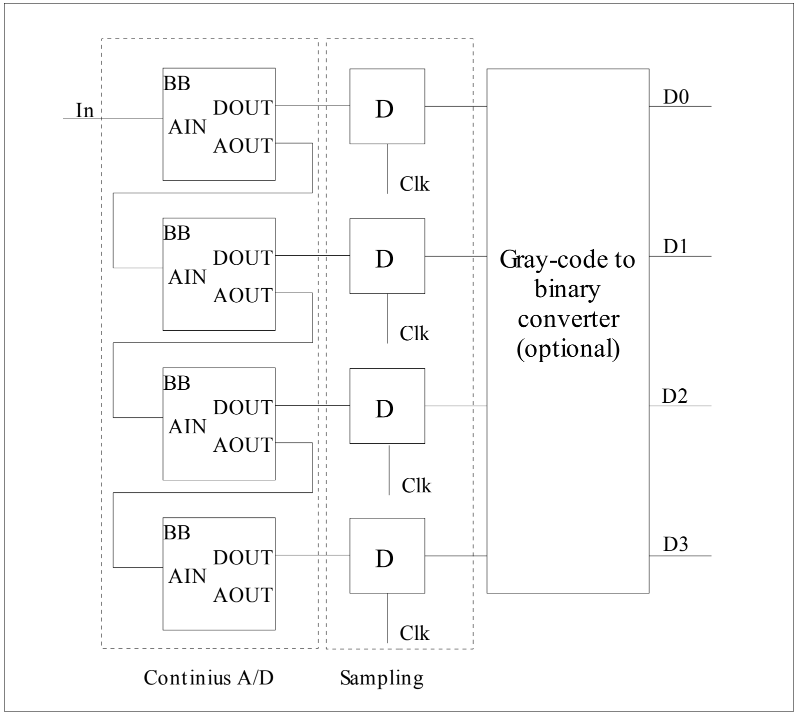
Gray-coded bit-block's transfer function

Note, that the modified scheme does the same thing. If the bit-block's digital output is one, the analog output and thus all lower bits are inverted. Also, the analog output's transfer function became continuous. This is another implementation of the main idea behind Gray-code, that is, a little change in the represented value causes a little change in the bits coding that value. A possible implementation using ideal, but saturating operational amplifiers (same as before) would be like this:



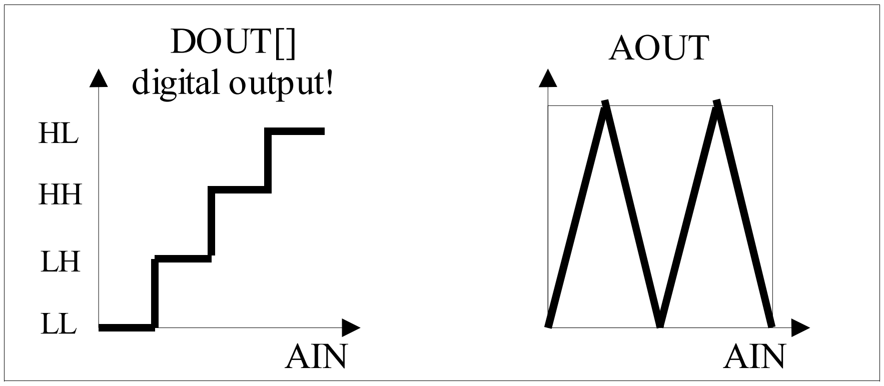
A simple Gray-coded bit-block implementation

Using this modified bit-block, you can leave out even the last S/H circuit from converter design C. This will give us a continuous-time A/D converter, whose output can be sampled in the digital domain:



Converter design D

The main benefit of this layout is that digital domain sampling is much easier to do and much more accurate than the a S/H circuit can be. Further more this design is able to work on any low conversion frequencies which none of the previous designs could do. There is also a possibility to create "semi Gray-coded" base-n converters using the same technique. However in this case the base (n) should be an even number. For example a base-4 output converter's semi Gray-coded version would have the following transfer functions:



Gray-coded base-n bit-block's transfer functions

Naturally **`DOUT`** now should be Gray-coded too.

Logarithmic converters

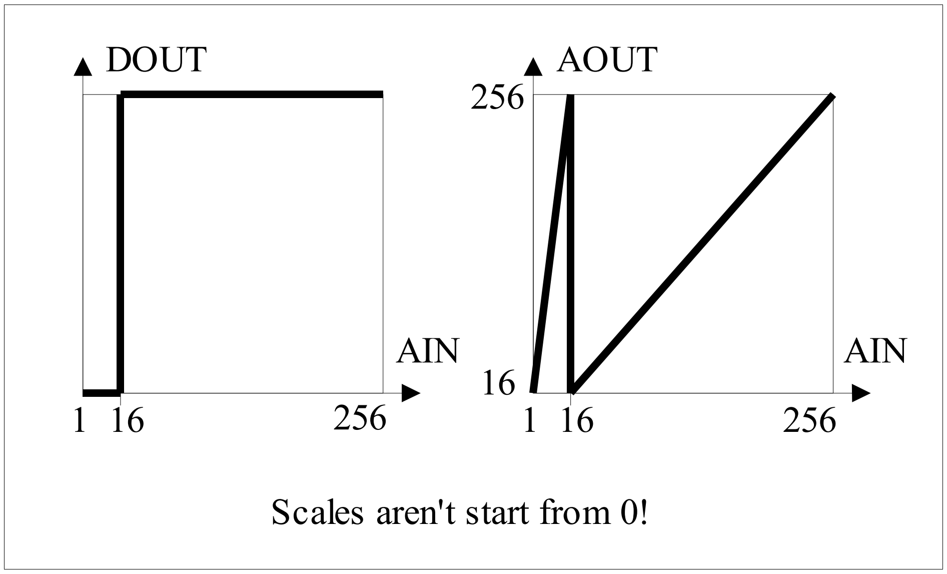
Until now we've used the same bit-block for generating all digits of the converted number. If we use different bit-blocks at different bit-positions we can create converters with other than linear transfer functions. For example we can construct a direct-logarithmic A/D converter. In the following this technique will be introduced with a 3-bit logarithmic converter which can convert an input value from 1mV (inclusive) to 256mV (exclusive). The output will be $\log_2$(In/[mV]) in our example. For a three bit converter the output values will correspond to the following input ranges:

input value range (mV)	***output value (binary)***
(1;2]	000
(2;4]	001
(4;9]	010
(8;16]	011
(16;32]	100
(32;64]	101
(64;128]	110
(128;256]	111

For the following explanation it's convenient to extend the input range of our converter to include the upper bound also, in our case 256mV. For the extended input range the converter's theoretical transfer characteristic should be like this:

input value range (mV)	*output value (binary)*
(1;2]	000
(2;4]	001
(4;9]	010
(8;16]	011
(16;32]	100
(32;64]	101
(64;128]	110
(128;256]	111
256	111

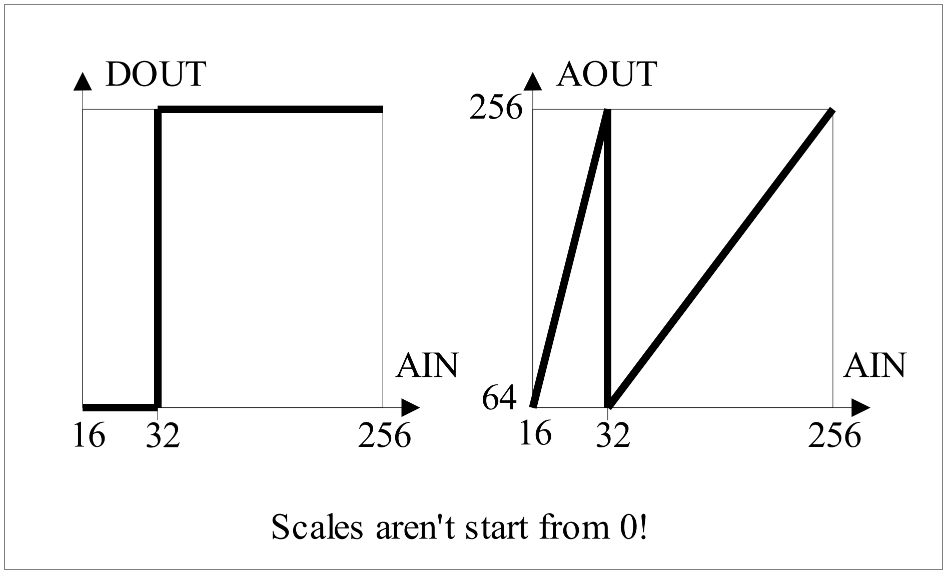
The most significant bit changes it's state at 16. This means that the MSB bit's bit-block's transfer functions should look like this:



Logarithmic converter's bit-2 bit-block's transfer functions

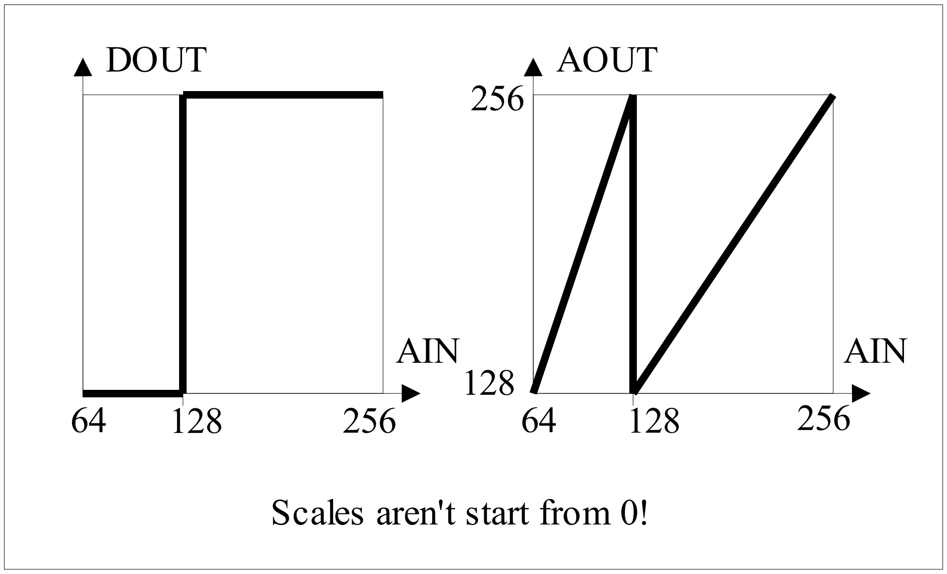
Note that, the **`AIN-AOUT`** transfer function has two different slope parts. It complicates a bit the implementation of the bit-block, but it is still possible. It also can be noticed that it maps range 1-16 and range 16-256 into the same output range (16-256).

The second bit changes it's state at values 1, 4, 16, 64 in the original input. But we want to connect this bit's bit-block over the MSB's bit-block which has the previously described mapping behavior. In detail it maps 1 to 1, 4 to 32, 16 to 1, 64 to 32. The two 0-to-1 transitions are mapped to the same value (32) and 1-to-0 transitions are also mapped to either 16 or 256. Now, we can construct the bit-block transfer functions for this bit:



Logarithmic converter's bit-1 bit-block's transfer functions

As can be seen these transfer functions are not the same as the previous ones. The third bit changes it's state at every integral power of two. You can check that the first and the second bit-blocks together maps all 1-to-0 transitions to 128 and all 0-to-1 transitions to either 64 or 256. The transfer functions can be designed as follows:



Logarithmic converter's bit-0 bit-block's transfer functions

Having these bit-blocks you can construct the complete 3-bit direct logarithmic A/D converter. You only have to choose the proper configuration. Because the bit-blocks are different it calls for design B or C where each bit has it's own dedicated bit-block. There's one important note to make here: the input span (along with the output span) of each stage decreases and though the upper bound remains the same the lower bound increases.

As a summary here are the general equations to implement an n-bit logarithmic A/D converter:

Design inputs:

- the lower bound of the input scale: $A_0>0$
- the number of bits of the converter: n
- the quotient of the converter: $q>0$.
  (This means that output value i'th change corresponds to input value $A_0\ q^i$)

Design outputs:

- the full-scale input value is: $A_0 \cdot q^{2^n-1}$
- the upper bound of the input range is: $A_0 \cdot q^{2^n}$
- for bit-block *i* (generating bit i) the input scale's lower limit is: $L_i = A_0 \cdot q^{(2^n - 2^{(n-i)})}$
- for bit-block *i* the input scale's higher limit is: $H_i = A_0 \cdot q^{2^i}$
- for bit-block *i* the output switch point is at: $S_i = A_0 \cdot q^{(2^n - 2^{(n-i-1)})}$

Conclusions

The basic idea explained at the beginning of this article can be the source of a group of A/D converters. They differ in complexity, speed and accuracy. They also differ in output code (Gray-code, binary code, or base-n code including BCD). Their interface can be serial or parallel and their transfer function can be linear or logarithmic. They can perform similar to existing converter classes with reduced complexity. Some has unique features, like direct logarithmic output, continuous time analog sampling or variable precision, not customary in current techniques.

Future work

Each of the building blocks and converter layouts should be tested against various real-world effects, like the non-ideal behavior of the operational amplifiers, non-perfect resistor values, parasitic effects, etc. Because both design A, B and C are basically discrete time analog circuits it calls for a switched capacitor circuit implementation.