Binary Access Memory: An Optimized Lookup Table for Successive Approximation Applications

Benjamin Hershberg*, Skyler Weaver*, Seiji Takeuchi†, Koichi Hamashita†, Un-Ku Moon*
*School of Electrical Engineering and Computer Science, Oregon State University
†Asahi Kasei EMD Corporation, Atsugi, Japan

***Abstract*—An optimized memory structure, Binary Access Memory (BAM), is presented for successive approximation applications that employ an error correction lookup table. Unlike true random-access memory, the probability of different codes occurring in a binary successive approximation access pattern is not uniformly distributed. BAM exploits this fact in several ways to reduce the number of sub-block switches, the average and worst-case access latency, and power consumption compared to a conventional SRAM lookup table. A simple technique for using BAM in an asynchronous successive approximation design is also presented.**

I. Introduction

The binary search algorithm is a widely applicable search technique that has been used since ancient times to determine the value of an unknown quantity. In the realm of modern analog and mixed-signal integrated circuit design, a very popular application of this algorithm is the successive approximation register analog-to-digital converter (SAR ADC). The basic structure of a voltage-domain SAR ADC is shown in Fig. 1(a). The ADC determines the digital representation of the unknown input voltage V_{in} by successively decreasing the error between the input voltage and the DAC voltage in binary steps over several cycles [1]. Each cycle, the SAR code is updated with a new, more accurate approximation of V_{in}. While several of the key ideas presented in this paper are applicable to binary search applications in a more general sense, they will be presented from within the context of SAR ADCs in this discussion.

In a practical SAR ADC implementation, mismatch of the DAC elements cause INL and DNL errors. Even with careful layout and high precision analog process technologies, the upper bound on DAC element matching will be limited by overall power, area, and input loading requirements. To push beyond this matching limit, calibration is often employed. A very popular method is to determine the precise bit weights of a binary controlled DAC using sub-radix-2 DAC elements, such as in [2]. This approach allows small DAC elements sized only for noise requirements, but does require additional digital processing in order to apply the bit-weight corrections to the final SAR code.

Bit-weight calibration works under the assumption that the DAC element values are static. For example, a MOS capacitor would be unsuitable as a capacitive DAC element because its capacitance value is a non-linear function of the voltage across its two terminals. In some processes, particularly exotic and emerging semiconductor technologies, highly linear capacitors

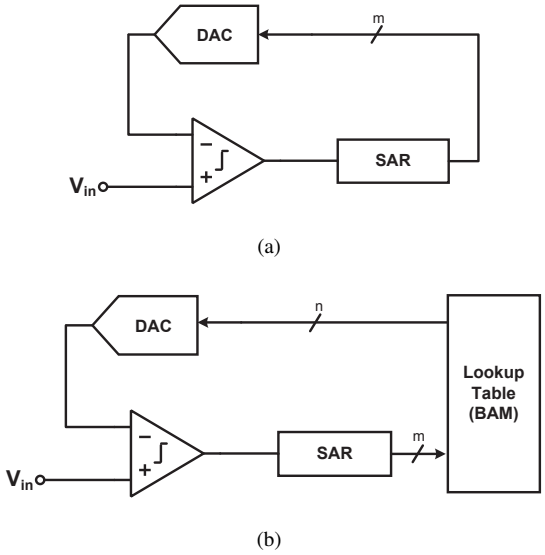

Fig. 1. a) The fundamental structure of a voltage-domain SAR ADC. b) A SAR ADC which uses a lookup table to correct for non-idealities. Unlike most SAR calibration techniques, it can correct highly non-linear and signal dependent errors.

are simply not available. Furthermore, from a topological viewpoint the ability to perform non-linear calibration would enable new design approaches and successive approximation techniques which have never been seriously considered until now.

For a fully generalized form of error correction, we can use a one-to-one lookup table to translate between each possible SAR code and a corresponding DAC code which will generate the correct analog reference. This is shown in Fig. 1(b). The DAC must still be able to accurately generate every analog reference described by the SAR, but the codes which generate these references no longer need to be linear or co-dependent in any way. The SAR code is the address provided to the lookup table, and the output code used to control the DAC can be a different length than the SAR code. This allows the DAC to be designed with many extra levels of redundancy in order to ensure that there will be at least one suitable DAC code for every SAR code even in the presence of mismatch, non-linearity, and signal dependence.

This generalized approach to SAR-ADC error correction is often avoided because the associated latency, power, calibration, and complexity penalties of implementing the lookup

978-1-4244-9474-3/11/$26.00 ©2011 IEEE

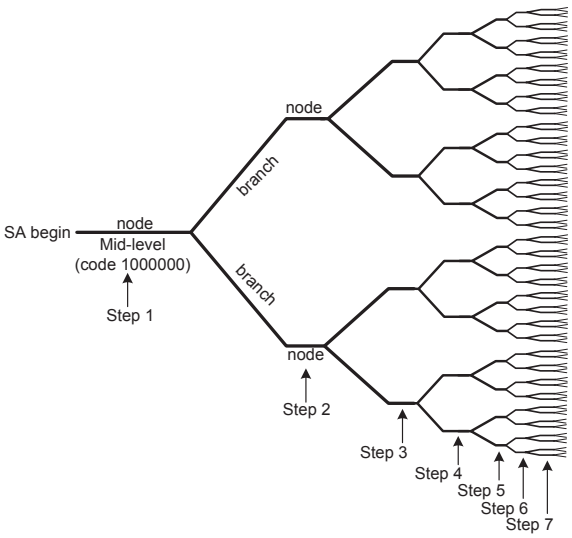

Fig. 2. A binary tree describing all possible paths for a 7 step binary search. BAM sub-blocks are organized in a tree structure, and sub-blocks store sub-trees.

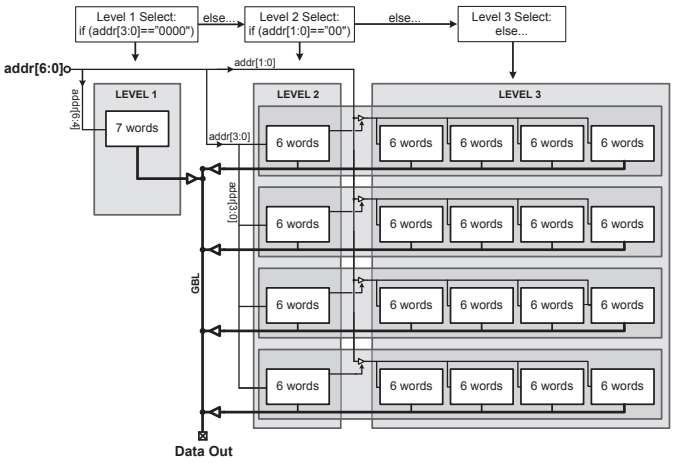

Fig. 3. Basic structure of a three level, 127 word BAM. SA steps 1-3 access Level 1, steps 4-5 access Level 2, and steps 6-7 access Level 3. The tree-like structure of BAM places the most frequently accessed sub-blocks closest to the inputs and outputs, with the fewest intermediate buffers.

table as a RAM or ROM are often too high. The very name RAM (random access memory) provides a key insight into how we can optimize the lookup table for binary-search applications - a binary-search is not an entirely random operation. The address decode and data read-out hardware of a RAM is specifically structured with the assumption that all memory locations have an equal probability of being accessed on any given cycle. Quite to the contrary, during any cycle within the binary search there are only two possible memory locations out of the entire range of addresses which might be accessed on the following cycle. In this paper we present the concept of binary access memory (BAM), which can be used to reduce the latency and power requirements of the lookup table.

II. Useful Properties of Binary Search

As visualized in the decision tree of Fig. 2 for a 7-bit SAR ADC, the structure of possible memory accesses patterns during a successive approximation (SA) conversion is a binary tree, with each node branching into exactly two children nodes (each horizontal "node" line in Fig. 2 corresponds to a memory address in the lookup table). A SA conversion consists of many steps, with step 1 beginning at the left most node and moving one branch to the right on each successive step until reaching the right-most nodes, where the final conversion result is obtained. There are some key properties of this tree structure that we can exploit in order to optimize the memory access:

1) Once a search goes down a specific branch, the probability that any nodes which are not descendants of this branch being accessed later on in the search becomes exactly zero.
2) The probability of any specific node being visited during a SA conversion is a non-uniform distribution. The further to the left of Fig. 2 the decision level is, the higher the probability is that it will be visited during an SA conversion. At the left most node (the mid-level code), the probability of being visited equals 1.
3) For any node on the tree currently being visited, its two children nodes each have a probability of 0.5 of being visited on the next step, and all other nodes in the tree have zero probability of being visited on the next step.
4) There is only one step number which a node can be visited during, and this step number is known for all nodes. For example, the left-most node in Fig. 2 will always be visited on step 1, its two children can only be visited on step 2, their children can only be visited on step 3, and so on.

In the following sections, we will explore the practical application of these four properties in the BAM structure.

III. Binary Access Memory

In a typical SRAM, the data words are subdivided into many sub-blocks, as in [3]. In a three-level decode scheme, the correct sub-block is selected by decoding the MSBs of the address into column select and row select signals. The remaining LSBs are passed on to the sub-block itself, which then decodes and reads the requested data word. Properties 1 and 2 of Section II indicate that this is not the optimal structuring for a BAM. Property 1 suggests that we can minimize the number of sub-block decodes and block-switches which must be performed during a SA-conversion by storing sub-trees in the sub-blocks rather than storing words according to common MSB codes (as in the typical SRAM case). Furthermore, Property 2 suggests that we should make the data words closest to the trunk of the tree "easier" to access than words with a lower probability of being read out. A three-level, 127-word BAM structure for a 7-bit SAR ADC which implements these two considerations is presented in Fig. 3. The Level 1 sub-block contains all of the words with `addr[3:0]` equal to `'0000'`, which in Fig. 2 is all of the nodes in steps 1-3. The four Level 2 sub-blocks contain all of the words where `addr[3:2]` is not `'00'` but

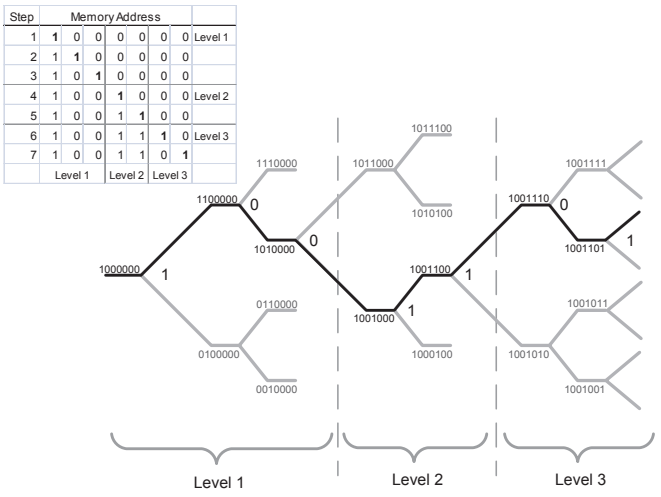

Fig. 4. An example SA conversion without pre-fetch. Unlike the many block switches that would be necessary for an SRAM, a BAM only requires as many block switches as it has levels (3 in this case).

`addr[1:0]` is `'00'`; these are the four 6-node sub-trees that form steps 4 and 5 in Fig. 2. The sub-sub-trees in steps 6 and 7 of Fig. 2 make up the sub-blocks of Level 3. For simplicity, this configuration is presented; however, in practice, a three level BAM would be better suited for a memory depth of 1024 words or more.

A hypothetical BAM memory access pattern is presented in Fig. 4. As highlighted in the table at the upper left of the figure, the address' LSBs are always a `'1'` followed by $step - 7$ zeroes. This makes the address decode hardware very straightforward - not only does the number of zeroes tell us which level we are currently on, but the position of the least-significant `'1'` in the address tells us which step we are currently on. Once this information is used to select the proper sub-block, decoding within any sub-block is simply a matter of providing it with the correct subset of address bits (the subset used depends on which level the sub-block resides).

The number of sub-block switches during a SA conversion for a three-level BAM is always three. By contrast, the average number of sub-block switches for an analogous three-level SRAM is $n - (m + 0.5)$ where n is the total address length and m is the sub-block address length. In a practical design, this will result in speed and power benefits. For example, for a 12-bit SAR ADC, the BAM will switch sub-blocks 3 times per SA conversion whereas an SRAM would switch an average of 7.5 times per conversion.

Further speed and power savings are obtained with the BAM structure by positioning the most commonly accessed words nearest to the address inputs and data outputs. For example, the Level 1 sub-block will always be accessed during a SA-conversion, and this block is connected to address-in and data-out by the fewest number of buffers and routing. The power-per-bit-accessed will be lowest for this sub-block, second lowest for the Level 2 sub-blocks, and the most for the Level 3 blocks. This "easy" access to the most commonly accessed words also results in a lower average access time, which can improve the overall speed of an asynchronous BAM (discussed further in Section VI).

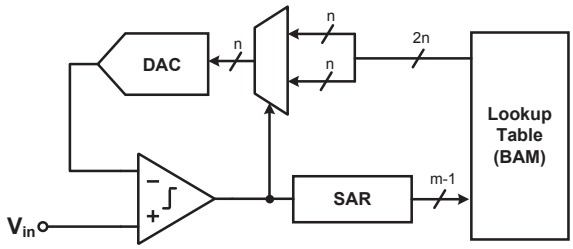

Fig. 5. System level view of BAM with pre-fetch. The two possible words which could be requested on the following cycle are retrieved in advance, and the correct one is chosen by the comparator.

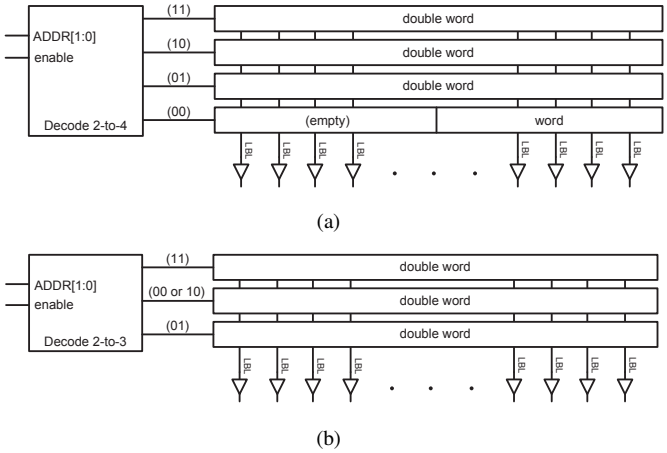

Fig. 6. Sub-block structures for BAM with pre-fetch and pre-charge for a) the Level 1 sub-block and b) all Level 2 and Level 3 sub-blocks.

IV. Global Pre-fetching

Property 3 of Section II hints at a way to dramatically reduce the effective latency of memory accesses. In a SA conversion, the children of the current reference level are the only two possible levels which might be visited on the next step (step n). If we begin pre-fetching these two words on step $n - 1$, we can simply select the correct word to use as soon as both the comparator has determined the branch to follow for step $n-1$ and the two possible choices for step n have been fetched. Pre-fetching reduces the memory access time (t_{BAM}) by the step period (T_{step}). If T_{step} is greater than t_{BAM}, the total effective latency is only the short time required to select the correct pre-fetched word and provide it to the DAC.

Fig. 5 is the system level implementation of pre-fetching. Although the global sub-block organization in Fig. 3 is unaffected by the addition of pre-fetching, the sub-blocks themselves must be modified. The most efficient way to pre-fetch is to pair sibling words together as a double length word and store them at the address of their parent (so that the SAR code during step $n - 1$ will automatically pre-fetch the double-word for step n). The pre-fetch enabled sub-block for Level 1

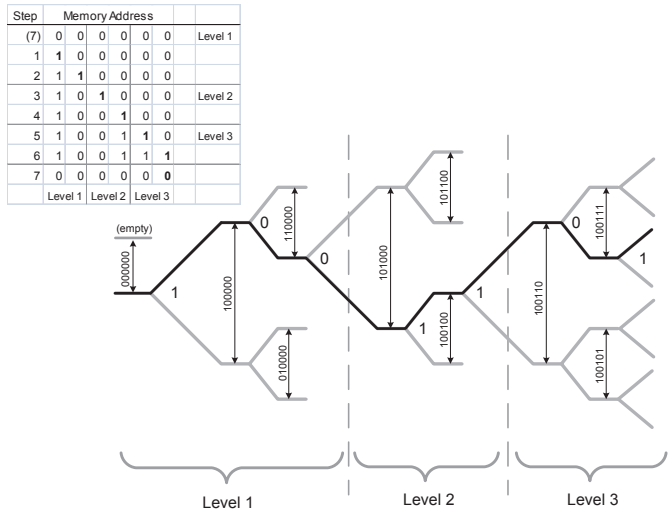

Fig. 7. Example SA conversion with pre-fetch. Data is now stored as double-words, which reduces the number of address bits by one. The first double-word accessed in a sub-block immediately after a level transition is always the same, which allows all sub-blocks to be pre-charged with their initial double-words.

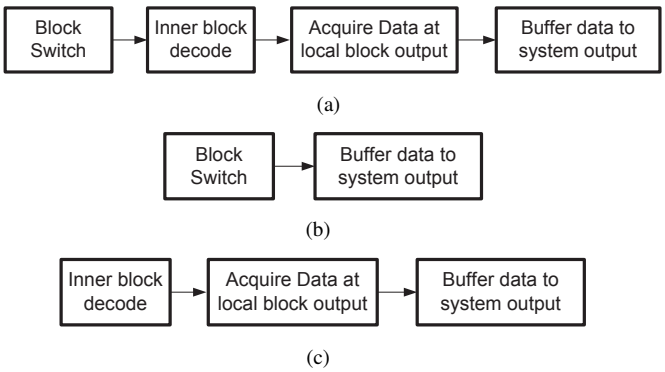

Fig. 8. a) Without pre-charging, the worst-case timing scenario will occur during steps with block switches and will also include output buffer glitching (for directly buffered designs). b) With pre-charging, the timing for steps with block switches is greatly improved and glitches are eliminated. c) The timing scenario for steps which do not have a block switch.

is shown in Fig. 6(b) and for Levels 2 and 3 in Fig. 6(a). A sample SA-conversion with pre-fetch is shown in Fig. 7 for the same access pattern of Fig. 4. As noted in the SAR code table of Fig. 7, the overall address length of the BAM has been reduced by one bit and is only addressed by the six MSBs of the SAR. This is because the double-words are stored in their parent's addresses. The "missing" 7th bit is now fed directly from the comparator output to the double-word selection MUX in Fig. 5. The amount of data stored remains the same, because the word length has doubled while the address range has halved. Since half of all bits that are pre-fetched are discarded, from a power-delay product perspective pre-fetching may not have much benefit. However, from a total conversion speed standpoint, it is very helpful in minimizing the speed bottleneck that the BAM would otherwise present to the overall SAR ADC operation.

V. Local Pre-charging

A subtle but useful change that pre-fetching brings to the BAM structure is that the `'10'` double-word in each Level 2 and 3 sub-block and the `'00'` double-word in the Level 1 sub-block is always the first double-word to be requested when that respective sub-block is initially selected (as seen in Fig. 7). This is a helpful trait of pre-fetching because the worst-case access time occurs on steps with sub-block transitions, as shown in Fig. 8(a). If the output is buffered directly (instead of with sense amps), this worst-case access time becomes even worse because there will be speed-reducing glitches on the output buffers due to the held value on the local bit lines of previously requested data sent from that sub-block. By pre-charging the local bit lines within the sub-block to the first double-word to be accessed (which will now be the same word every time), the latency of block-switching steps can be reduced to that of Fig. 8(b). The worst-case timing scenario now happens during steps where there is no block transition (Fig. 8(c)). The decoders of Fig. 6 are designed so that whenever the sub-block is not enabled, the local bit lines are pre-charged with the first double-word that will be requested the next time the sub-block is accessed.

VI. Asynchronous BAM

A fully asynchronous BAM can be built by considering Property 4 of Section II. If we add an additional bit to each double-word, and this bit is set to a `'1'` for every double-word which would be accessed on an odd numbered step and a `'0'` for every double-word on an even numbered step, then we can use this toggling bit as a clock signal to asynchronously detect when the data is ready at the output of the BAM. A fully asynchronous BAM (when used with an asynchronous SAR) can yield significant speed improvements, because the access latency for different SA steps varies considerably due to the structural variations discussed throughout this paper.

VII. Conclusion

The concept of binary access memory has been presented for use as an optimized lookup table for applications with binary-search memory access patterns. It exploits the unique properties of binary search to increase access speeds with tree-based memory structuring, pre-fetching and pre-charging. Power is decreased by placing the most frequently accessed words closest to the inputs and outputs and by minimizing the number of block switches. A simple solution for enabling fully asynchronous operation offers additional speed improvements.

References

[1] D. A. Johns and K. Martin, *Analog Integrated Circuit Design*. John Wiley And Sons, 1997.

[2] W. Liu, P. Huang, and Y. Chiu, "A 12b 22.5/45MS/s 3.0mW 0.059mm2 CMOS SAR ADC achieving over 90dB SFDR," *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 380 –381, feb. 2010.

[3] S. Cosemans, W. Dehaene, and F. Catthoor, "A 3.6 pJ/Access 480 MHz, 128 kb On-Chip SRAM With 850 MHz Boost Mode in 90 nm CMOS With Tunable Sense Amplifiers," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 7, pp. 2065 –2077, jul. 2009.