Power-efficient Delay-insensitive Codes for Data Transmission

P. Patra and D. S. Fussell

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188, USA

Abstract

We have introduced and formalized the notion of dynamic delay-insensitive codes *for data communication. We describe several* codes *and* protocols *designed to optimize switching energy expended at the data pins during data transmission in asynchronous systems. These include adaptations of some existing communication methods as well as some new techniques for reducing energy used in dynamic data communication between* delay-insensitive *circuits.*

1 Introduction and background

Current generations of CMOS VLSI systems have reached the point that power consumption more often than not is the limiting factor in the size and speed that can be obtained. This is especially true in chips designed for use in battery-powered portable systems, but it is increasingly true for high performance conventional systems as well, where exotic and expensive means to dissipate the heat generated on chip may be required to enable the system to operate at all. As a result, low voltage systems with built-in power management features are becoming common for both portable and desktop use.

Power management generally involves power down techniques to reduce dissipation by disabling portions of a circuit not currently in use. Dynamic power is the dominant source of dissipation in digital CMOS circuits (which do not use energy recovery techniques [1]). Dynamic power for clocked circuits is often approximated ([2]) as:

$$\sum_{all\ \ gates} p_t \cdot C_l \cdot V_{dd}^2 \cdot f_{clk}$$

where p_t is probability of a transition at a gate output during a clock cycle, C_l is the total loading capacitance of the gate, V_{dd} is the supply voltage, and f_{clk} is the clock frequency. Power management techniques aim to reduce p_t, particularly at nodes with large C_l such as I/O pins or internal bus interfaces. The effectiveness of this approach has led to increased interest in the use of asynchronous circuit design techniques, since asynchronous circuits lack a global clock and thus naturally have the property that switching only occurs in a portion of a circuit when some computation is being performed.

The class of "delay-insensitive (DI) circuits"—circuits whose external behaviors make no explicit reference to time and are independent of any (non-negative) delays in their internal components and wires—is a subclass of the class of asynchronous circuits. This class holds a great, although largely undemonstrated, potential for low-power applications since there are no global clock trees to be powered and switching activity occurs only in conjunction with useful computation or communication. For an application where particularly impressive power savings have been demonstrated using delay-insensitive circuitry, see [3].

DI systems can be expected to employ DI communication protocols not only within the computational logic of the system but also on busses used for communication between major subsystems. It has been observed that (DI) communication seems to provide better reliability and throughput ([4]) than alternative protocols for parallel transmission over large distances and between systems clocked by different clocks. Moreover, the non-synchronized transitions, inherent to DI protocols, tend to even out power drains, thereby improving overall noise characteristics.

Previous work on DI protocols for parallel bus communication has not explicitly dealt with the issue of minimizing the signal transitions required and thus the power consumed, although the importance of minimizing I/O signal transitions by means of coding was observed by both [5] and [6]. However, the former offered only an initial attempt at the problem, and the latter focused on synchronous data transmission only.

In this paper we consider ways to reduce transitions in DI data communication over a parallel bus through efficient protocols and codes. Codes and protocols are devised to reduce the number of voltage transitions at pins between chips, boards and systems and at bus interface points–possibly at the expense of higher complexity in encoding and decoding circuitry.

2 Theory of delay-insensitive codes

In our model, a transmitter (sender) circuit and a receiver circuit mutually communicate only through a set of wires of two types between them: *data wires* and *control wires*. We assume that arbitrary, undetermined non-negative delays are possible in the wires. Because only causal, but not any arbitrary temporal, relationships among events are preserved in a system in our model, one needs to encode the data-validity within the data itself in order to communicate delay-insensitively. The receiver must computationally determine if a new piece of valid data is available before consuming/accepting it. Other approaches to correct asynchronous data communication use a *delay-sensitive* control wire or signal ([7]), (1) which tracks the delay in the signal path of the data and goes active whenever data becomes valid, or (2) which goes active sufficiently late to 'practically' ensure that data has become valid. The former approach is often physically impracticable and the latter is often unacceptable for performance. Therefore, in this paper we consider purely delay-insensitive (but only bit-parallel) data communication.

Having decided to encode validity within the data, we also have to ensure 'data separation'—i.e. the receiver needs to know when a new piece of valid data is available. We assume that the receiver is capable of observing any changes in its input wires. But, whether a particular logic value pattern on its input wires is a new data value or not is defined by a *protocol* between the sender and the receiver, and the associated *code*.

Definition 1: A *code* C is a set of subsets of a finite set W. Each member of C is a *codeword*. By identifying each member of W with a unique wire, we can mechanistically interpret each subset of W to be a logic state of the data wires, $|W|$ in all, between a receiver and a transmitter. Thus, if the empty set $\emptyset$ denotes a fixed initial state, e.g., the state where all data wires are logically low, then a subset $w \in 2^W$ denotes a bit-vector (equivalently, a state of the data wires) where exactly those bits corresponding to the wires represented in w are logically opposite to their values in the initial state vector.

Example 1: Let $W = \{a, b, c\}$. If $\emptyset$ denotes state 000 of the three wires, a and c identified with the first and last wires, respectively, then $\{a, c\}$ denotes the state 101. If one is interested in only two data values, a possible code C is $\{\{a, c\}, \{a, b\}\}$. □

We will henceforth use 'subset' (of W), 'state' (of data wires), or *'word'* (transmitted) interchangeably depending on the context.

Definition 2: Suppose there exists an *onto* mapping M from C to the set, V, of all the data values we are interested in transmitting. Code C is then said to be *statically* delay-insensitive (SDI) if each codeword can be unambiguously received in the presence of arbitrary wire delays when the transmitter switches the wires to the codeword from a fixed initial state/word assumed not part of C. Equivalently ([8]), $(\forall \quad x, y \in C \ : \ x \subseteq y \ : \ x = y)$ holds.

However, one is usually concerned with transmission of sequences of data. Thus, we introduce and formalize the concept of a *dynamically* delay-insensitive (DDI) code as follows:

Let $D, S \subset 2^W$ such that $D \cap S = \emptyset$, $C = D \cup S$ and $I = (2^W - C)$. D and S stand for the sets of codewords called *datawords* and *spacers*, respectively. I is the set of all words that are 'invalid' or non codewords. Suppose also that an onto function M exists from D to V, the set of data values to be communicated. Furthermore, let graph G be the natural $|W|$-dimensional hypercube induced by the powerset of W, i.e. an edge exists between vertices $u, v \in 2^W$ iff $u \setminus v$ is a singleton.

Definition 3: A *hyperedge* in G exists between $x, y \in C$ if the following holds:
$(\forall \quad z \in C \ : \ x \cap y \subseteq z \ \wedge \ z \subseteq x \cup y \ : \ z = x \ \vee \ z = y)$

Definition 4: A *hyperpath* exists between $x, y \in C$ if there is a sequence of one or more hyperedges in G between x and y such that all the intermediate vertices (between hyperedges) are from S.

Definition 5: A data value v is said to be *transmissible* at a dataword x if there exists a dataword y such that there is a hyperpath from x to y, and $M(y)$ is v.

Definition 6: C is a DDI code if each data value $v \in V$ is transmissible at each dataword $x \in D$.

Definition 7: A hyperpath consisting of a single hyperedge is *monotone*. A hyperpath between $x, y \in D$ is *monotone* if for some $z \in S, x \cap y \subseteq z, z \subseteq x \cup y$, there exist monotone hyperpaths between x and z, and between z and y.

Definition 8: C forms a *monotone* DDI code if each data value $v \in V$ is transmissible at each dataword $x \in D$ via monotone hyperpaths only.

It is noteworthy that DI codes have some unidirectional error-detection properties as well.

2.1 Communication protocol

With each code is associated a *protocol* of transmission : To send c after d ($c, d \in D$), the sender follows a hyperpath from c to d, handshaking (synchronizing) with the receiver at each intermediate vertex. The signal transitions from one vertex to the next on the hyperpath is also monotonic in the sense that a wire changes its logic value at most once while wires switch from one vertex/state to the next —a general requirement on delay-insensitive logic. The receiver acknowledges each received spacer or dataword on the hyperpath that the sender follows, when ready for the next—perhaps using a separate feedback line to the sender. This way the two communicating parties synchronize with each other. Note that self-timed circuits (such as micropipelines [7]) are quite compatible with this notion of spacers, as a receiver can receive a codeword and throw it out, if it is a spacer, without affecting the underlying computation.

We observe that if the physical signals representing a codeword are sent on the wires and then consumed (i.e. received and removed) by the receiver, then a code which is SDI is also DDI. This is so because the very act of reception leaves the wires logically at their initial state represented by $\emptyset$—there is no need for additional synchronization to reset. Another way to turn a statically DI code into a dynamic one is to treat $\emptyset$ as a spacer between every two consecutive datawords transmitted.

We will use names of the sets such as W to denote their sizes, to avoid clutter, when no confusion may arise. From here on, we speak synonymously of a word sent over the wires W and its mechanistic interpretation as a state bit-vector of size $|W|$.

The map M from D to V is assumed to be one-to-one and onto in rest of the paper except in section 4.1.

3 Energy usage properties of DI transmission schemes

For large loads and long communication lines, we adopt the reasonable notion that the energy consumed in a transmission is primarily due to transitions on signal wires (or transmission lines). The computations necessary to encode or decode raw data are assumed to consume negligible power. Consequently, given a set of data values, we define the energy efficiency of a DI transmission scheme (code plus protocol) as inversely proportional to the average number of transitions on the signal wires (hence, energy consumed) per data value when a random sequence of data values is to be transmitted.

The time taken for a data value is measured in units of 'handshakes' between the sender and the receiver. The average time is the inverse ratio of the length of a random sequence of data values to the length of the corresponding sequence of codewords transmitted.

The space resource for a scheme is the number of wires used.

The possibility of multiple spacers between two data codewords allows us to use a scheme similar to 'Huffman encoding' such that the average area or space requirement can be minimized.

3.1 Some well-known DI codes

One-hot code: The sender sets to high the wire mapped to the data value being sent. After the receiver acknowledges by toggling the 'feedback' wire, the sender resets the data wire last turned high. This is a spacer condition, and the receiver acknowledges it by again toggling the lone feedback wire. At this point new data can be sent by repeating the protocol just described. The feedback wire is necessity for synchronization between the communicating parties. In the following figures for various metrics, we include the feedback transitions as well as the feedback wire. The number of data wires is W.

For this scheme, Size (number of distinct data values) $= W$; Space (total number of wires used) $= W+1$. Time (to transmit one data value) $= 2$ which is the number of synchronizations needed. Energy (used per transmission) $= 2+2$ units where one unit is the energy used to make a logic transition on a communication wire. Space used is $D+1, D = W$ because D wires are needed to permit transmission of D different data values, and one wire is used for acknowledgement.

Although this scheme is quite efficient in energy, it is often unacceptable because of very high space inefficiency.

Dual-rail code: To transmit an n-bit data value, $W = 2n$ wires are used—each bit of a data value is encoded by a pair of wires. One of the protocols used is called 4-phase handshaking, where each dataword is followed by an 'all-zero' spacer. Each dataword has ones in exactly $W/2$ bit positions. For this scheme:

$$Size = 2^{W/2};\ Space = W + 1;$$

$$Time = 2;\ Energy = W + 2$$

Sperner code: A Sperner code of dimension W is the largest static DI code for code length W. Each word with ones at exactly $\lfloor W/2 \rfloor$ bit positions is a dataword and represents a unique data value ([8]). The following protocol makes the DI code dynamic: Initially, the wires are in the 'all-zero' (reset) state. The following steps are repeated to communicate one or more data values: The sender switches exactly $\lfloor W/2 \rfloor$ wires, as appropriate, to send a data value (dataword). The receiver acknowledges upon receiving the current dataword. The sender, then, returns the wires to the 'all-zero' or reset state. The receiver acknowledges seeing this state which in our terminology is a 'spacer'. Then, the sender is ready again to transmit the next data value, if any—by changing the present (reset) state of the wires to the next dataword/state.

This scheme yields:

$$Size \approx \frac{2^W}{\sqrt{\pi W/2}};\ Space = W + 1;$$

$$Time = 2;\ Energy = W + 2$$

Figure 1 shows the six datawords as vertices with filled circles and the "all-zero" spacer as a vertex with empty circle on the pictured 4-cube. Recall that each vertex of a cube corresponds to a unique state of the data wires W. There exists a bijection from the set of vertices marked as datawords to the set of (six) data values.

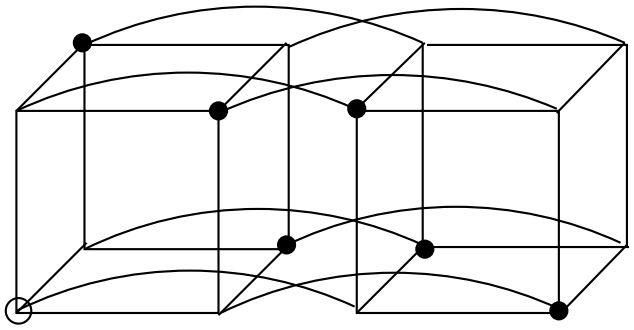

Figure 1: Sperner code: length 4 and size 4

4 Spacerless codes

The fastest way to transmit data delay-insensitively is to have no intermediate handshakes—that is, no 'spacers'. We have observed that the size of the a DDI code which is spacerless is proportional to the number of wires used, and hence, greatly limited.

DI Code	Size	Space	Time	Energy
One-hot	W	$W+1$	2	4
Dual-rail	$2^{W/2}$	$W+1$	2	$W+2$
Sperner	$\approx \frac{2^W}{\sqrt{\pi W/2}}$	$W+1$	2	$W+2$

Table 1: Summary of some well-known DI Codes

4.1 Spacerless code: type I

One approach to designing a spacerless DDI code is the following:
Given W wires, consider the W-dimensional cube all of whose vertices are datawords, i.e., each word is a dataword. These vertices are partitioned such that a one-to-one mapping is established between the partitions and the set of data values to be transmitted. (That is, map M from D to V is not necessarily one-to-one.)

We can show that one can form a spacerless DDI code to transmit W distinct data values using W wires, where $|W|$ is a power-of-two. Abstractly, we color the vertices of a W-dimensional cube with W colors such that each vertex is adjacent to all the different W colors. Each color stands for a distinct data value and the coloring scheme is the convention for both reception (decoding) and transmission (encoding). It can be further shown that such a coloring is not possible when W is not a power-of-two.

Theorem 1: Vertices of a W-cube is colorable with W colors s.t. each vertex is adjacent to exactly W differently colored vertices, iff $(\exists\ \ k\ :\ k \text{ natural}\ :\ W = 2^k)$. □

Example 2: Figure 2 is an example for $W = 4$ with R, G, B and Y being the four "colors." □

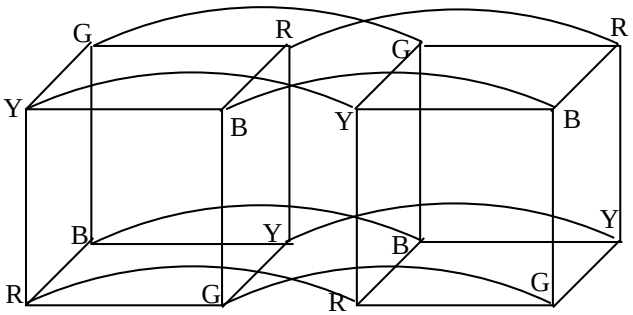

Figure 2: Spacerless code: type I, length 4 and size 4

4.2 Spacerless code: type II

Yet another approach to a spacerless DDI code is to associate exactly one dataword to each data value—in contrast to Spacerless Code I. The goal is to build the largest code of length W such that the smallest subcube containing any two datawords does not cover a third dataword—meaning that a direct transmission of an dataword following any other is possible. This eliminates the intermediate handshakes (synchronizations) associated with spacers. Now, each dataword, except one, is associated with a distinct data value. The lone special dataword represents a "repetition," i.e. when the sender needs to repeat the last data value sent via an ordinary dataword, it simply sends this special "repetition" dataword.

More precisely, this spacerless DDI code has $S = \emptyset$, and satisfies the following:

$$(\forall x, y, z \in D \ : \ x \cap y \subseteq z \ \wedge \ z \subseteq x \cup y \ : \ z = x \ \vee \ z = y)$$

The above may be viewed as a coloring problem where *some* of the vertices of the W-cube are colored. The idea is that the remaining vertices of the smallest subcube containing a pair of colored vertices must all be uncolored, and we wish to maximize the number of colored vertices in the cube.

The upper and lower bound on the size $|D|$ of the largest spacerless DDI code of length W in this approach is conjectured to be $W + 1$, for all $W \neq 2$. For $W = 2$, the size is 2. An example of such a code for $W > 2$ is where each word with exactly $W - 1$ ones is a dataword. (The all-zero codeword is special—it may be interpreted as to stand for whatever the previous data value was. This requires storage of the last data value. Alternatively, two codewords may be assigned for each data value in order to allow back-to-back repetition of a data value. See next subsection.)

Unfortunately the size of such a code is very small—linear in code length—just as the one-hot code and the code in the previous subsection are.

For an illustration of this approach, see Figure 3. Four vertices of the 4-cube are marked R, G, B, or Y to indicate four datawords representing the four different data values. The all-zero spacer is indicated by an empty-circled vertex. Note that there is a natural equivalence relation among codes of a particular type, size and length that we are not dealing with here.

4.3 Further discussion

One need not assign two codewords for each data value (anticipating back-to-back transmission of a data value). In stead, just one codeword may be allocated to each (distinct) data value, and either of the following approaches may be taken to allow back-to-back repetition.

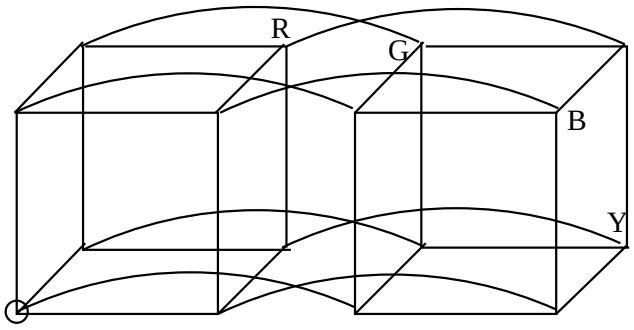

Figure 3: Spacerless code: type II, length 4 and size 4

- An extra control wire is employed to indicate whether the last data value is being repeated. This control information can be 'event' or 'level' based, i.e. a 2-phase or a 4-phase handshake protocol, respectively, may be adopted for the control wire.

- A codeword is designated as special (not bound to a fixed data value), and it is sent only when any dataword is to be repeated.

In case of the one-hot code, both the approaches above essentially call for an extra data wire—keeping the set of data values V fixed.

5 Adaptive Sperner code

We have seen how the Sperner Code introduces the "reset" handshake to allow code sizes exponential in the number of data wires used. In the following, we exploit this idea for high code size efficiency while trying to reduce the individual voltage switchings on the wires to minimize energy loss. The key to the following is the clever use of spacers (i.e., intermediate handshakes).

First note that the (minimum) energy (hence, the number of signal transitions) to switch the wires from one dataword to another is directly proportional to the *Hamming distance* ([9]) between them. When at any step all data values are equally likely to occur, the following lemma gives the mean Hamming distance between two consecutive datawords. The mean Hamming distance is also the *expected* Hamming distance when datawords are uniformly distributed (i.e., each dataword is equally likely to be the next one for transmission.)

Lemma 1: The mean Hamming distance between two datawords is

$$\left\lceil \frac{\sum_{i=0}^{n} \left(2i \times \binom{n}{i}^2 \right)}{\binom{2n}{n}} \right\rceil$$

Lemma 2: The expected Hamming distance between two consecutive datawords is $\lceil |W|/2 \rceil$.
Proof: We consider the case where $|W| = 2n$ and leave out the very similar proof for $|W| = 2n + 1$.

By the binomial theorem we get the following two equations:

$$(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i} \quad (1)$$

$$(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^{i} \quad (2)$$

Differentiating and then multiplying both sides of Eqn 2 by x,

$$nx(1+x)^{n-1} = \sum_{i=0}^{n} i \binom{n}{i} x^{i} \quad (3)$$

Multiplying the corresponding sides of Eqns. 1 and 3,

$$nx(1+x)^{2n-1} = \sum_{k=0}^{2n} \sum_{i=0}^{k} i \binom{n}{i}^2 x^{k} \quad (4)$$

Equating the coefficients of x^n on the two sides of Eqn 4,

$$n \binom{2n-1}{n-1} = \sum_{i=0}^{n} i \binom{n}{i}^2 \quad (5)$$

Therefore,

$$\frac{\sum_{i=0}^{n} \left(2i \times \binom{n}{i}^2 \right)}{\binom{2n}{n}} = \frac{2n \binom{2n-1}{n-1}}{\binom{2n}{n}} = n = |W|/2 \quad (6)$$

The claim follows from the above and Lemma 1 when we note that the expected Hamming distance is the mean distance between any two datawords.

(*End of Proof*)

5.1 Reworking the code

Using the original Sperner scheme, the number of signal transitions made to complete transmission of one data value is $W + 2$, for even W. (We will henceforth, without loss of generality, consider only even W.) The much lower value of the expected Hamming distance provides the motivation to devise a new scheme to conserve signal transitions. In the following, we will often abbreviate the Hamming Distance function of two words as HD.

First, we observe that wires representing a dataword can be switched to another dataword at Hamming distance 2 without a risk of incorrect reception, i.e. there is no need for synchronizing on an intermediate spacer. Second, we propose to include a few more spacers in C so that while going from a dataword to the next, one does not always have to use the 'all-zero' spacer. One method to achieve this is to augment the original Sperner code with spacers that are all the words with exactly $W/2 + K$ ones in their bit-vector representations, assuming $W > 3$. We can now state:

Lemma 3: The augmented code above is a monotone DDI code for $K = 2$.

Suppose that this specific ($K = 2$) *adaptive* Sperner code is used and that the sender uses hyperpaths without the all-zero spacer. Then, it can be shown that $x + x/2$ signal transitions and $x/2$ handshakes are necessary and sufficient to transmit any dataword b after a with $HD(a, b) = x$, $x > 2$. A possible protocol for the sender, in pseudo-code:

IF $HD(a, b) = 0$ (* the next data value is same as the current *)
THEN Send a spacer l with $HD(l, a) = 2$; then send a.
ELSE Follow a monotone hyperpath from a to b *never* using the 'all-zero' spacer.

For Hamming distance x, $x > 2$, this scheme implies $x/2$ handshakes hence, $\lceil x/2 \rceil$ acknowledgement transitions from the receiver. So, the total number of signal transitions involved is $Max(2 + 1, x + x/2)$, for $x > 0$. For $x = 0$, the energy used is that for $2 + 1 + 2 + 1 = 6$ signal transitions.

The adaptive Sperner scheme for $K = 2$ makes a simpler case in a hierarchy of codes with diverse time and energy tradeoffs. Relative to the original Sperner scheme, the expected time is increased by $W/4 - 1$, while expected switching energy is decreased by $W/4$ even when feedback transition from the receiver is counted in.

Lemma 4: For $x < (2W+4)/3$, the protocol above always uses fewer signal transitions than the Sperner scheme.

For $x >= (2W+4)/3$, the Sperner scheme of using the all-zero spacer is preferred for better time and energy performance. With this in mind, the sender's protocol now is:

IF $HD(a,b) = 0$ (* the next data value is same as the current *)
THEN Send any spacer l with $HD(l,a) = 2$; then send a.
ELSE Follow a shortest, monotone hyperpath from a to b.

In Fig 4, we show the above scheme for $W = 6$. The datawords are indicated by filled circles and the spacers by empty circles on the vertices of the hypercube. To transmit the dataword 001110 after the word 111000, the adaptive scheme requires 6 signal transitions as opposed to $6+2$ transitions in the non-adaptive case.

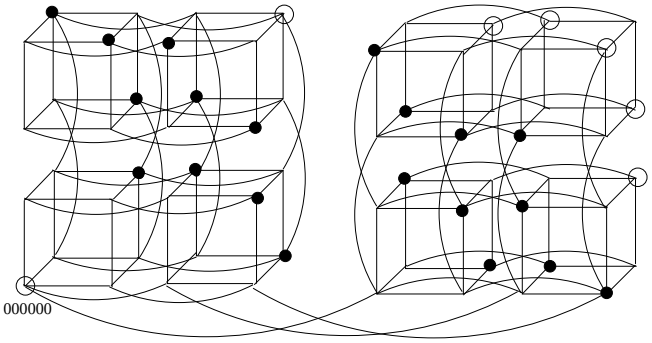

Figure 4: An adaptive Sperner code of dimension 6 (some cube edges not shown)

One can design a family of such schemes for $2 \leq K \leq W/2$.

6 Separable codes

Most algorithms for arithmetic circuits assume the input values to be in binary format. Hence, using a code such as Sperner's involves complex encoding and decoding circuitry. One approach to address this problem is to encode a condition for data validity outside the data bits. This way complexity for data validity may actually increase somewhat, but data decoding is almost trivial. For example, we can take the binary representation (x) of a data and concatenate another set of bits (y) encoding the data-validity condition to give a codeword ($x : y$). Decoding such a codeword involves stripping off the y bits that encode the validity condition. The Berger code [10] is an example of such a 'separable' code (it is a statically DI code, too.)

6.1 Code interpretation in terms of bit-transitions

We have two choices as to how we transmit a codeword. So far we have implicitly assumed "level-encoding," i.e., a wire is set to High (Low) if the corresponding bit in the codeword to be transmitted is 1 (0). The other method, which is emphasized in this section, is to toggle the state of a wire if and only if the corresponding bit of the codeword is 1. This approach to transmission is called "transition-encoding." For example, assuming the initial state is 10101, the final state upon (complete) reception of the codeword 11000 is the state 01101. As before, because of arbitrary transmission delays in wires, no particular order of reception among causally unrelated transitions is guaranteed. The receiver acknowledges upon reception of a codeword and enters into the 'clear' state, ready for a new data value. Conceptually, the receiver removes a codeword from the channel (wires) upon reception. The sender does not have to 'erase' the previous data, by resetting the state of the data wires, in order to transmit the next data value.

Under the interpretation above, two datawords of a Sperner code can be transmitted back to back without a spacer, and each value transmitted accounts for exactly $\lfloor W/2 \rfloor$ transitions. Two disadvantages of such a scheme are: (1) the scheme cannot exploit any locality property within the data stream to reduce the total number of transitions, and (2) the sender and receiver circuits tend to be more complex because of the 'event or transition logic' implied in the scheme.

6.2 A separable DDI code

We propose a scheme, based on Berger code, for a DDI code that has the nice property of 'separability' mentioned above while the code length is still logarithmic in code size.

Suppose the sender wants to send data value b after a, both binary values being of length L (i.e. $|a|=|b|=L$). A subcode y, in binary, is formed such that y, $y = L - HD(a,b)$, is the number of bit positions where the two data values match. The binary subcode y is transition-encoded, while the data part b is level-encoded for concurrent transmission on their respective wires. Note that the length of a subcode is $\log(L+1)$ if the number of data wires is L.

Correctness of the scheme above follows from these observations: While receiving b, only those wires

where a and b differ switch. On the other hand, the transitions on the subcode part are made to indicate, in binary, how many data wires *won't* switch. If transmission is complete, eventually the number (count) of transitions on the data wires will monotonically match up with the monotonically increasing (transition-encoded, binary) number represented by the subcode wires. Upon match-up, the codeword's validity for reception is complete. An example should clarify the process:

Example 3: Suppose $b = 1100101$ is the next data while $a = 0100011$ is the present. Moreover, suppose that the present state of the subcode wires is 101. The new (to be transition-coded) value of the subcode to go with b is 100, because $|a| - HD(a, b) = 100$. Therefore, the sender needs to set the data and subcode wires to 1100101 and $101 \oplus 100 = 001$, respectively. □

The space used for this scheme is $W = L + \log(L + 1) + 1$ wires. The expected Hamming distance between two datawords of length L is $L/2$, when each dataword is equally likely to be transmitted next. Consequently, the expected transitions on the subcode bits is no more than $\log L$ (not a tight bound). The expected energy per transmission of a data value in this scheme is bounded from above by $\lceil (L/2 + \log L) \rceil$. The size metric is 2^L and the time metric equals 1.

6.2.1 An easy extension

If the circuit complexity of transition-encoding of a large subcode y is undesirable, then one may recursively apply the separable coding scheme, described above, to encode y. When a subcode is small enough in length (e.g. $|y| = 2$), no further recursive application of the coding scheme is made. Note that each application of the coding scheme increases the total number of wires used to transmit a fixed set of data values.

7 Concluding remarks

We introduced & formalized *dynamic* codes for DI data transmission Efficiencies of several codes and protocols were characterized. We exploited two ideas: (1) trade in additional complexity in 'local computation' for energy-efficiency in 'non-local communication' and (2) trade off time by increasing synchronizations to reduce communication energy. Several new codes and extensions such as spacerless codes, adaptive Sperner, and delay-insensitive Berger code, etc. were also provided.

If a *dataword* is interpreted in light of the spacer(s) preceding it, further optimization in transmission energy is possible while keeping the space and time metrics nearly the same. In such case, encoding/decoding of a dataword is dependent on the context—the type or identity of spacers used. These observations are preliminary and need further research.

Relevant empirical and theoretical data is much needed to measure the complexity of transition-based circuits vis-a-vis level-based ones to enable a designer to choose a right coding scheme. Moreover, combinations of coding schemes discussed here need to be explored.

Acknowledgements

We thank the anonymous referees for their helpful comments.

References

[1] W. Athas, J. Koller, and L. Svensson, "An energy-efficient cmos line driver using adiabatic switching," report ACMOS-TR-2, Information Sciences Institute, July 1993.

[2] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power cmos digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473–483, Apr. 1992.

[3] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Schalij, "A fully-asynchronous low-power error corrector for the DCC player," in *International Solid State Circuits Conference*, pp. 88–89, Feb. 1994.

[4] M. Greenstreet, *STARI: A Technique for High-Bandwidth Communication*. PhD thesis, Princeton University, January 1993.

[5] P. Patra, "Design of efficient and robust asynchronous circuits – a dissertation proposal (section on power-efficient DI codes)," Jan. 1994. Dept. of Comp. Sci., Univ of Texas at Austin.

[6] M. Stan and W. Burleson, "Limited-weight codes for low-power I/O," in *Proceedings of the 1994 International Workshop on Low Power Design*, ACM/IEEE, April 1994.

[7] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, pp. 720–738, June 1989.

[8] T. Verhoeff, "Delay-insensitive codes—an overview," *Distributed Computing*, vol. 3, no. 1, pp. 1–8, 1988.

[9] R. Hill, *A First Course in Coding Theory*. Clarendon Press, Oxford, 1986.

[10] C. V. Freiman, "Optimal error detection codes for completely asymmetric binary channels," *Inf Control 5*, pp. 64–71, 1962.