Delay-Insensitive, Point-to-Point Interconnect using m-of-n Codes

W.J. Bainbridge, W.B. Toms, D.A. Edwards, S.B. Furber
Dept. of Computer Science, The University of Manchester,
Oxford Road, Manchester M13 9PL, UK.
{wjb, tomsw, sfurber, doug}@cs.man.ac.uk

Abstract

m-of-n codes can be used for carrying data over self-timed on-chip interconnect links. Such codes can be chosen to have low redundancy, but the costs of encoding/decoding data is high. The key to enabling the cost-effective use of m-of-n codes is to find a suitable mapping of the binary data to the code.

This paper presents a new method for selecting suitable mappings through the decomposition of the complex m-of-n code into an *incomplete* m-of-n code constructed from groups of smaller, simpler m-of-n and 1-of-n codes.

The circuits used both for completion detection and for encoding/decoding such incomplete codes show reduced logic size and delay compared to their full m-of-n counterparts. The improvements mean that the incomplete m-of-n codes become attractive for use in on-chip interconnects and network-on-chip designs.

1. Introduction

Quasi-Delay Insensitive (QDI) logic [7] is an attractive asynchronous design style for many reasons, but especially for the simple timing closure and analysis that it allows. There is growing interest in using self-timed or delay-insensitive (DI) approaches for system-level interconnect, with the ITRS roadmap [13] suggesting this will become the prevalent approach.

There are many possible DI data encodings, but very few of these are practical for CMOS logic design. Dual-rail encoding, a 1-of-2 code, has been the dominant style to date, but recently there has been interest in 1-of-4 signalling[1,3,4] which uses half the number of signal transitions to convey two bits of data with signalling activity on just one wire.

Beyond 1-of-4, one-hot 1-of-n codes are significantly more expensive than a dual-rail encoding, e.g. the 3-bits of data that can be carried in a 1-of-8 code requires only 6 wires in dual-rail, 4-bits requires a 1-of-16 code, but only 8 wires in dual-rail. Other more complex codes which offer better efficiency have been proposed for use in specialist applications, predominantly involving chip-to-chip communication [6].

2. Long-distance, on-chip connectivity

The issue of long distance on-chip communication is becoming more significant as CMOS feature sizes shrink and the relative costs of wire delays and logic delays changes. One of the consequences is that longer connections are now often broken into multiple segments separated by buffers to repower the signal, as in figure 1, thus minimizing the propagation latency. For improved throughput, the buffers can be replaced by latches.

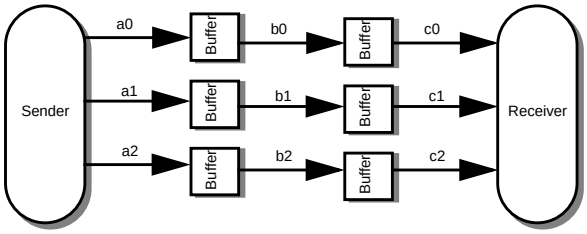

Figure 1: Buffering signals to minimize latency

Typically a point-to-point interconnect as illustrated in figure 1 will be many bits wide, requiring multiple self-timed code-groups as illustrated in figure 2 to carry the data word to give DI operation. At every pipeline latch repeater stage (two shown), each of the g groups has to be completion-detected on the latch outputs, and then a g-input C-element tree is used to gather the group acknowledges together into a single acknowledge signal for the latch stage. This is fed back to the preceding stage, through suitable buffering to give the drive necessary to fan-out the acknowledge to all of the latches in the preceding stage.

Since asynchronous circuits are effectively a series of interlocked ring-oscillators, their performance is determined by the delays in one loop of the slowest stage. Here measured in inverting logic stages, this is made up from (by

following the shaded grey loop): Delay=

2x2 (2 inversions in each of the two high drive-strength datapath C-element latches)
+ 2 w (two long wires, 1 forwards, 1 acknowledge)
+ $\log_2(g)$ (1 inversion in each level of the tree of low drive-strength C-elements in acknowledge path)
+ $\log_2(g)$ (buffer in acknowledge path prior to fan-out to many C-elements in preceding latch stage)
+ completion-detector
+ (1+ completion-detector + 2 $\log_2(g)$) mod 2

The final term arises because every loop must contain an odd number of logical inversions for correct operation/ oscillation to occur and thus an additional inverter may be required – dependent on the logic depth of the completion detector, C-element tree and buffer.

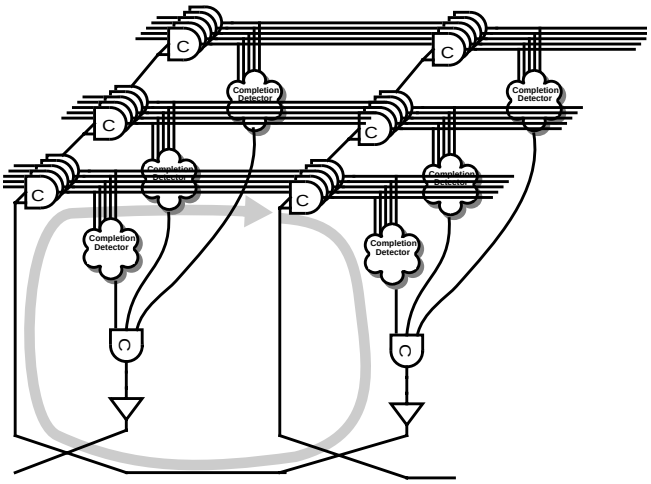

Figure 2: Self-timed multiple-group pipeline

Some of these terms are fixed – the repeater latches are essential; wire delays are determined by the space between repeaters which is chosen to minimise latency. The others can be varied.

In earlier work [2], we suggested using narrow links each with their own acknowledge signal, operating at higher speeds in a time-division multiplexed manner to provide a fatter virtual pipe that allowed eight 1-of-4 encoded channels operating in parallel to provide comparable throughput to an equivalent 32-bit dual-rail channel, but using much less resource. That approach trades an increased number of wires (additional acknowledge signals) for reduced cycle-times (no C-element tree and little buffering in the acknowledge path) allowing higher frequency operation.

Here, we present a new approach for using more complex delay-insensitive codes, to increase the "code-density" beyond that possible with 1-hot codes such as dual-rail or 1-of-4.

3. Delay Insensitive Codes

Unordered codes are codes in which no code word is contained in any other. They possess the property that the time of arrival of individual bits does not affect the interpretation of the code word. Consequently, these codes have many applications in error correction and delay-insensitive communication.

DI codes can be characterised by three main factors [12]:

- Efficiency
- Membership test
- Encoding/Decoding complexity

Efficiency is determined by a code's *rate R*, the number of binary bits per wire given by:

$$R = \frac{\log_2 M}{n}$$

where M is the *size* of the code (the number of data values represented) and *n* is *length* of the code (the number of wires occupied).

Membership test is the completion detection function to determine the arrival of valid data. Encoding/Decoding complexity is the circuitry required to extract the dataword from the code word. Both concepts are difficult to quantify.

Costs given in this paper will be based on implementing these functions using the Delay-Insensitive Minterm Synthesis (DIMS) technique [10]. DIMS functions are generated using the complete set of valid products for the function. Each product is formed using a C-element (to give DI operation on the return-to-zero phase) across each of the inputs as illustrated in figure 3.

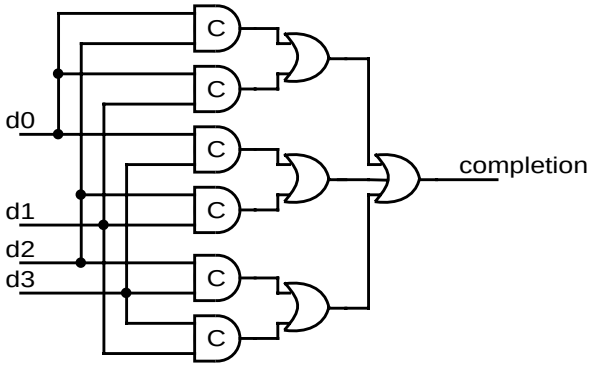

Figure 3: DIMS 2-of-4 Completion Detection

While DIMS circuits are often far from optimal, they are useful in a delay-insensitive environment as they simplify the creation of logic functions with respect to delay assumptions. They are employed here because the size of such circuits can easily be determined as a function of the valid products. All costs are given in terms of transistors per bit, based on the circuitry required to represent 32-bit data-

paths. In all cases, the minimum integer number of code-groups required to represent the 2^{32} symbols is used. Encoding and decoding costs are for converting between dual-rail and the relevant code since it is assumed that the data will arrive at the interface from a DI dual-rail datapath.

The costs are calculated as follows: C-elements and OR gates of up to 3 inputs are used. 2-input C-elements contain 10 transistors, and 3-input 12, 2-input or gates contain 6 transistors and 3-input 8. For codes where $\log_2$(no of symbols) is not an integer, the encode and decode costs are infeasibly large due to the need to form 2^{32} products representing all of the symbols in the datapath. In these cases the costs in table 1 assume that the number of valid symbols has been reduced to the nearest power of two.

There are many different types of DI codes. The most important of these are described below and characteristics of examples of these codes are summarised in table 1.

1-of-n codes are 1-hot codes of size n. Wider datapaths are created by concatenating many such codes together. The simplest, and most efficient of the 1-of-n codes are 1-of-2 (dual-rail) and 1-of-4 codes, both of which have a rate of 0.5. The main advantage of the 1-of-n code is the ease of completion detection. Because data is represented by a single '1' within a code group, completion detection can be performed with a simple OR of the wires.

Systematic codes are those in which some bits represent valid binary encoded data symbols and others, the so-called check bits, are introduced to make the code delay-insensitive. A dual-rail encoding is an example of a systematic code. Encoding and decoding of systematic codes to binary values is straightforward with little overhead. It should be noted that table 1 presents results for transforming to/from dual rail interfaces. In the case of a dual-rail code, the encoding and decoding costs are obviously zero.

No 1-of-n codes for n > 2 are systematic and therefore data encoded in a 1-of-4 scheme incurs encoding and decoding costs. For n > 4, the efficiency of the code decreases dramatically, consequently these codes are rarely used.

m-of-n codes are the superset of 1-of-n codes. Here the presence of m '1's on n wires represents the arrival of a data word. As in 1-of-n codes, small code groups may be concatenated to form wider datapaths. The size of an m-of-n code is given by *nCm*.

The most optimal of the m-of-n codes, the Sperner code, is also the most optimal DI code, here m = n/2. However the completion detection and encoding/decoding overheads for m-of-n codes with n > 2 are considerable. Example costs for 2-of-4, 2-of-7 and 3-of-6 codes are presented in table 1. DIMS completion detection methods involve large sets of products that are of exponential order with increasing n. As m-of-n codes with n > 2 are not systematic, encoding and decoding complexity may be great. The costs in table 1 are for a simple DIMS implementation using a complete set of products. Although more optimal techniques are possible, the circuits depend on the code and the ordering chosen.

Systematic codes, as previously described, are codes where the information content can be distinguished from the check bits required to make the code unordered. The most optimal systematic code, the Berger code consists of I data bits and k check bits, where $I = 2^k-1$. The checkbits, k, record the number of zeros in the I-bit data word. The length of a Berger code is: $I + \log_2 I$ giving a rate of

$$R = \frac{\log_2 I^I}{I + \log_2 I} = \frac{I}{I + \log_2 I}$$

As table 1 shows, DIMS techniques are unsuitable for Berger codes due to the need to use a complete set of products. This makes completion detection and encoding very expensive. It should be noted that the high cost of decoding is due to the need for completion detection to signify a valid code word. Realistic completion detection methods for Berger codes and other systematic codes are explained by Piestrak [8] and Akella [1]. Applying Piestrak's technique reduces the complexity of completion detection of the I=32,K=5 berger code to 109 transistors/bit. This also reduces the encoding costs to 124.6 and decoding cost to 131.9 transistors/bit.

Code	M	Rate	Completion detection	Encoding	Decoding
			transistors-per-bit		
Dual-Rail	2	0.5	11.9	0	0
1-of-4	4	0.5	9.9	20	12
2-of-4	6	0.65	35.6	30	32
3-of-6	20	0.72	80.9	116	100
2-of-7	21	0.63	73.9	133	108
Berger I=32, K=5	2^{32}	0.86	$2.6\text{x}10^{10}$	$2.0\text{x}10^{10}$	$2.6\text{x}10^{10}$

Table 1: Properties of DI-codes and DIMS implementations

4. Incomplete Codes

As discussed in sections 2 and 3 the rate of 1-of-n codes deteriorates rapidly beyond a 1-of-4 code and the complexity of the logic circuits necessary to use n-of-m codes is too great.

In this section, a method is presented to combine small 1-of-n codes together so as to increase the number of symbols beyond the products of their constituent component

codes without drastically increasing the completion detection complexity.

When two codes are concatenated, the number of symbols in the resultant code is given by the product of the two code sizes. This number can be increased by moving ‘1’s between the code groups, allowing one of the groups to adopt ordered or invalid values. These values can, however be detected by the data value in the other code group thus preserving the delay-insensitivity of the code as a whole. This method can be applied to any concatenation of m-of-n codes. If any ‘1’ is permitted to move between code groups so that all groups are allowed to adopt unordered values, a regular M-of-N code is created (where M is the sum of all weightings (m) of constituent codes and N is the combined number of wires (n) in the code). When movement is only permitted in a single direction, the code size is increased beyond that of the simple concatenation of the codes, but not to the full M-of-N code. Such a code is an *incomplete* [8] m-of-n code, represented here as m-of-n*, and offers opportunities for simpler completion detection and encoding/decoding as discussed later.

To further illustrate the composition and use of the incomplete m-of-n* code, this paper focuses on two examples where movement is only permitted in a single direction. In each case one group is chosen as the *control* group that adopts ordered values by acquiring or donating ‘1’s to or from the other *body* groups. It should be noted that the number of ‘1’s active on the wires carrying a code is kept constant at m and therefore the number of transitions contributing to energy consumption is not increased by this technique.

4.1. Example A1: the incomplete 2-of-7* code

A seven wire code can be composed from a 1-of-3 and a 1-of-4 code. Twelve values can be formed from the concatenation of these two codes. Using the 1-of-3 code as control, an extra six values can be generated by using the idle-state value (000), also known as *spacer* and adding an extra ‘1’ to the body group thereby converting these 6 values from a 1-of 4 code to a 2-of-4 code. This gives 18 values in total as shown in figure 4 compared with 21 for a complete 2-of-7 code.

A valid codeword is detected when a valid 1-of-3 code and a valid 1-of-4 code are detected or when a valid 2-of-4 code is detected. Although this is more complicated than completion detection for a 1-hot code, it is considerably more simple than that required for a complete 2-of-7 code.

The completion detection circuitry (discussed in more detail in section 5) can be further simplified by treating the two-of-four code as two dual-rail codes (body0 and body1), avoiding the illegal ‘11’ symbols (marked *i* on fig 4), leaving 16 usable symbols.

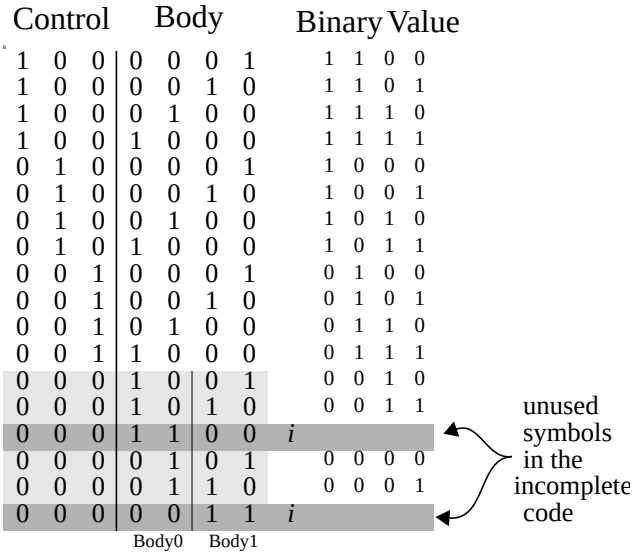

Figure 4: Incomplete 2-of-7 code

The incomplete 2-of-7 code now has a rate of 0.57, a 14% improvement over both a 1-of-2 dual rail code and a 1-of-4 code. The mapping of the binary values to code symbols shown in figure 4, and other possible mappings, are discussed in section 6 since their principal affect is on the encoding and decoding complexity.

4.2. Example B1: the incomplete 3-of-6* code

A similar code can be generated in 6 wires using a dual-rail and a 2-of-4 code group. Concatenating these two codes creates 12 symbols. Here the dual-rail code is used as a control group, allowing extra values to be generated by “borrowing” a ‘1’ from the 2-of-4 code, and using the “11” symbol of the dual-rail code as illustrated in figure 5.

It should be noted that the spacer symbol of the dual-rail code could have been adopted as in the previous example, and a 3-of-4 code used; however, this makes encoding/decoding the symbols more complex. If both the spacer symbol and the “11” symbol are used the code becomes a regular 3-of-6 code. The incomplete code shown has a rate of 0.67.

This method can be extended to any concatenation of m-of-n codes. However, with the increase in size, particularly in the control group, comes an increase in completion detection and encoding/decoding overheads. Additional examples of its application are shown in table 2. In the remainder of this document codes created using this method will be denoted by m-of-n*, to differentiate them from regular m-of-n codes. In table 2 each encoding adopted by each group is listed, the spacer values are marked ‘spacer’, and the all-one values are marked with an

Control		Body				Binary Value			
0	1	0	0	1	1	0	1	0	0
0	1	0	1	0	1	1	0	0	0
0	1	0	1	1	0	1	0	0	1
0	1	1	0	0	1	1	1	0	0
0	1	1	0	1	0	1	1	0	1
0	1	1	1	0	0	1	1	1	0
1	0	0	0	1	1	0	0	0	1
1	0	0	1	0	1	0	0	1	0
1	0	0	1	1	0	0	1	1	0
1	0	1	0	0	1	0	0	1	1
1	0	1	0	1	0	0	1	1	1
1	0	1	1	0	0	1	0	1	1
1	1	0	0	0	1	0	0	0	0
1	1	0	0	1	0	0	1	0	1
1	1	0	1	0	0	1	0	1	0
1	1	1	0	0	0	1	1	1	1

Figure 5: Incomplete 3-of-6 code

n-of-n entry. The ‘useful symbols factor’ column gives the number of symbols per constituent code combination with the ‘total’ column showing the total number of symbols representable by the m-of-n* code. The number of information bits this can represent is shown in the rightmost column.

5. Completion detection

Detecting the arrival of data in delay-insensitive circuits is an important and often non-trivial problem [1,8].

From table 1 it can be seen that 1-of-n codes are the easiest to detect, with the complexity reduced for increasing sizes of n. Unfortunately, m-of-n codes are more complicated to detect: the complexity is proportional to the number of symbols and is also dependent on the size of *m* because of the limited fan-in of CMOS gates.

Small m-of-n codes can be detected using *threshold logic*, such as that proposed by Theseus Logic [5]. Threshold logic functions assign to each input a weighting, and to the function a threshold. If the product of the inputs and their weightings is greater than the threshold the output transitions. Null Convention Logic devices [9] developed by Theseus Logic, exhibit hysteresis behaviour to control the return-to-zero phase of their inputs in the same way as C-elements. However the fan-in of these gates is limited to around 3 in CMOS processes under 0.2 µm due to the limit on the number of series PMOS transistors. Using a combination of NCL threshold gates can reduce the complexity of a 2-of-4 code completion detection circuit to 8.6 transistors-per-bit. The complexity of larger code groups increases exponentially.

Piestrak [8] described a general purpose solution to completion detection for m-of-n codes, using multiple-output threshold functions, implemented using sorting networks. Using this method the complexity of completion detection for 2-of-7 and 3-of-6 codes presented in table 1 can be reduced to 30.4 and 39.9 transistors/bit respectively.

code	decomposition			useful symbols		bits
	control	body0	body1	factor	total (M)	
2-of-4*	1-of-2	1-of-2		2x2	4	2
2-of-5*	1-of-2 1-of-2 spacer	1-of-2 1-of-3 2-of-3		2x2x1 2x1x1 2x1x1	9	3
2-of-7*	1-of-3 spacer	1-of-4 2-of-4*		3x4 1x4	16	4
3-of-6*	1-of-2 2-of-2	2-of-4 1-of-4		2x6 1x4	16	4
3-of-7*	2-of-3 1-of-3 spacer	1-of-4 2-of-4 3-of-4		3x4 3x6 1x4	34	5
3-of-8*	1-of-2 spacer spacer	1-of-3 2-of-3 1-of-3	1-of-3 1-of-3 2-of-3	2x3x3 1x3x3 1x3x3	36	5
3-of-9*	2-of-5* 1-of-5 spacer	1-of-4 2-of-4 3-of-4		4x8 6x5 4x1	66	6
4-of-8*	3-of-4 2-of-4 1-of-4	1-of-4 2-of-4 3-of-4		4x4 6x6 4x4	66	6
4-of-9*	2-of-4 1-of-4	2-of-5* 3-of-5*		4x8 6x8	80	6
5-of-11*	1-of-3 2-of-3 2-of-3 3-of-3	2-of-4 1-of-4 2-of-4 1-of-4	2-of-4 2-of-4 1-of-4 1-of-4	3x6x6 3x4x6 3x6x4 1x4x4	268	8
5-of-12*	2-of-4 2-of-4 1-of-4	1-of-4 2-of-4 2-of-4	2-of-4 1-of-4 2-of-4	6x4x6 6x6x4 4x6x6	432	8

Table 2: Example formation of incomplete m-of-n codes

Completion detection for the incomplete m-of-n* codes introduced in the previous section is made considerably easier due to their being composed from smaller m-of-n codes. The 2-of-7* code from example A consists of three single 1-hot codes, concatenated together. Data is valid when any two of the code groups contain valid data. Completion detection can be implemented by ORing each code group and combining the result with either a 2-of-3 thresh-

old gate (3 inputs of unity weight and a threshold of 2), or with a set of C-elements covering the possible products as in figure 6.

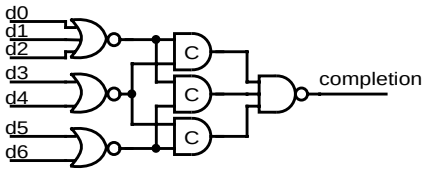

Figure 6: 2-of-7 completion detection

The code of example B requires more complex completion detection circuitry. Here the code is decomposed into 2-of-4 and dual-rail code groups. Completion detection for each sub-code is required as well as completion detection for the 1-of-4 code assumed when the control group transmits the "11" value, (which also must be detected). An example circuit is given in figure 7. It should be noted that in this circuit, several terms have been shared between the various constituent circuits, reducing the size of the overall circuit.Table 3 shows the completion detection complexity for some of the codes presented in the previous section, using these techniques.

It is worth mentioning at this point that the completion detection method described by Piestrak [8], is especially suitable for these codes as all the functions with thresholds from 1 to m are calculated, and so detection of the code adopted during reallocation of '1's can easily be obtained.

code	completion detection
	transistors-per-bit
2of5*	17.7
2of7*	13.9
3of6*	23.4
4of8*	46.1
5of12*	32.2

Table 3: Incomplete m-of-n* code completion detection complexity

6. Ordering

Delay-Insensitive codes, by definition, are unordered. For systematic codes and some simple codes such as 1-of-n codes, an ordering scheme is implied or easily determined. However, for complex codes such as the m-of-n code, encoding and decoding complexity can be reduced by selecting a suitable ordering scheme.

A standard method of assigning binary values to m-of-n codes is described by Overveld [11]. Table 1 shows the

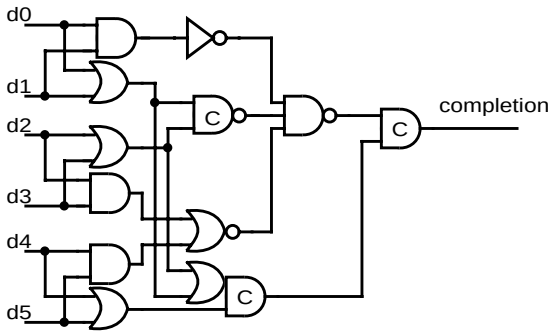

Figure 7: 3-of-6 completion detection

encoding/decoding complexities of various m-of-n codes adopting this ordering and table 4 shows the same metrics for the incomplete m-of-n* codes discussed in this paper.

Encoding and decoding complexity can be significantly reduced for these m-of-n* codes using the control group to determine the ordering, making the codes *semi-systematic*[8]. The following two examples show how different approaches to ordering can significantly reduce the complexity.

code	encode	decode
	transistors-per-bit	
2of5*	42.5	51.25
2of7*	116	100
3of6*	133	108
4of8*	497	456
5of12*	2108	1788

Table 4: Incomplete m-of-n code circuit costs

6.1. Example A2

From table 4, it can be seen that the cost of encoding a 2-of-7* code (from dual-rail) is 116 transistors-per-bit.

This cost can be significantly reduced by using the 1-of-3 control group to coordinate the encoding. An example ordering is given in figure 8. Here when the control group adopts the spacer (000) value, the 2-of-4 body code values map straight onto the message vector. When the control group assumes the 0 (001) value the message vector has 3 zeros (in the bottom four bits) and the 1-of-4 body code determines the position of the 1 value. The 3 (100) value of the control code covers the cases when the message vector contains 3 ones; here the 1-of-4 code represents the position of the zero in the message vector. The final four cases are covered by the 2 ("010") control value, where the four values have to be generated. As can be seen in table 1 encoding

Control			Body				Binary Value				
0	0	0	0	0	1	1	0	0	0	1	1
0	0	0	0	1	0	1	0	0	1	0	1
0	0	0	0	1	1	0	0	0	1	1	0
0	0	0	1	0	0	1	0	1	0	0	1
0	0	0	1	0	1	0	0	1	0	1	0
0	0	0	1	1	0	0	0	1	1	0	0
0	0	1	0	0	0	1	0	0	0	0	1
0	0	1	0	0	1	0	0	0	0	1	0
0	0	1	0	1	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	1	0	0	0
0	1	0	0	0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0	1	1	1	1
0	1	0	0	1	0	0	1	0	0	0	0
0	1	0	1	0	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	1	1	1	0
1	0	0	0	0	1	0	0	1	1	0	1
1	0	0	0	1	0	0	0	1	0	1	1
1	0	0	1	0	0	0	0	0	1	1	1

Figure 8: Example ordering for incomplete 2-of-7

and decoding codes that do not readily map to binary bits is infeasibly complex. For this reason the encoding complexity of the scheme outlined above is given in terms of *transistors-per-symbol* for a *single code group*, it is compared with a similar metric for regular 2-of-7 codes. Using this encoding scheme gives an encoding complexity and decoding complexity of 14.6 and 18.7 transistors per symbol. A complete 2-of-7 code has encoding and decoding complexities of 31.3 and 28.1 respectively.

As mentioned in the previous section this complexity can be reduced still further by treating the 2-of-4 code as two dual-rail code groups. In this case the ordering can be determined by using the regular ordering of 1-of-n codes to produce the ordering shown in figure 4. This ordering gives an encoding cost of 26 transistors/bit and a decoding cost of 32 transistors/bit.

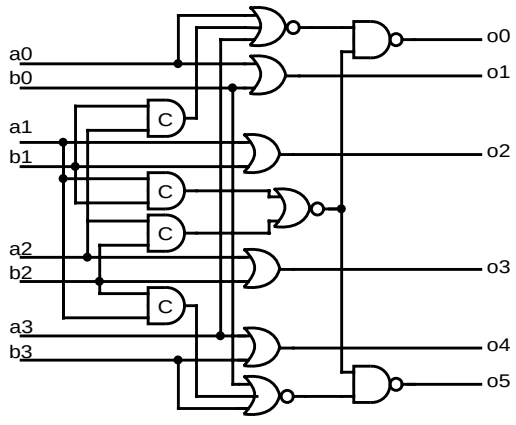

Figure 9: 1-of-4 to 3-of-6 encode

6.2. Example B2

Another example of using the control group to orchestrate the encoding process involves the incomplete 3-of-6 code of example A2. Here the 2-of-4 body code, can be represented as two 1-of-4 codes superimposed (ORed) on top of each other, the control group can then be used to determine which of the '1's in the code symbol determines the least significant two bits (each 1-of-4 code represents two binary bits). The cases where both 1-of-4 codes contain the same value are covered by the "11" value of the code group since in this case both sets of message bits are determined by the 1-of-4 code. For example, the code word 01 1001 maps to the message vector 1100, as the "01" control symbol decrees the leftmost active wire to represent the least significant bits (0001 equates to 00 and 1000 to 11). In the case of 10 0110 the message vector becomes 0110. Figure 5 shows the complete ordering.

This scheme gives an encoding complexity of 42 transistors/bit and a decoding complexity of 73.5 transistors/bit. Suitable encode and decode circuits are shown in figures 9 and 10..

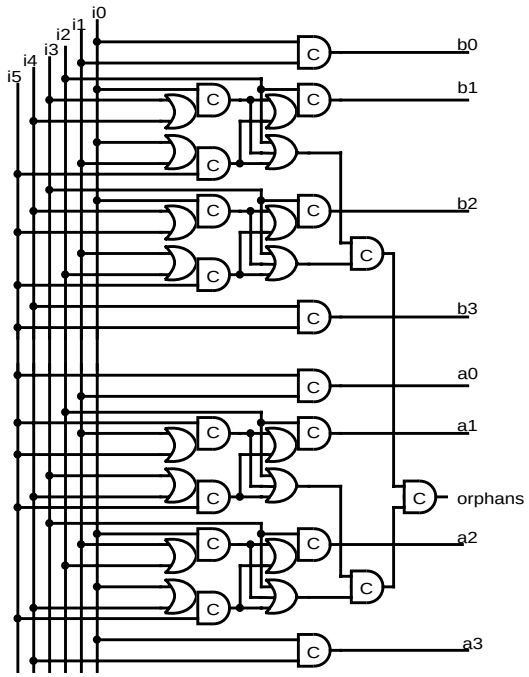

Figure 10: 3-of-6 to 1-of-4 decode

7. Code evaluation

The decomposition of m-of-n* codes into smaller, simpler codes is applicable to many such codes as illustrated above. However, whilst using a more complex code reduces the data-path width for a given point-to-point connection, it also affects the complexity of the completion detection

used at repeater stages and the complexity of the encoder and decoder at each end of the link.

The cost of the completion detectors is the most important factor here since this is typically paid multiple times in a long link, whereas the encode and decode costs only occur once for a point-to-point link. The completion detector is also the limiting factor upon performance since the encoder and decoder can often be further pipelined.

Whilst previous sections described the costs of the logic blocks required to use the incomplete m-of-n codes, they do not easily facilitate a direct comparison between them. For this, an additional metric is required that takes into account the number of bits transmitted by the code.

Here we compare:

- the delay (in logic inversions, i) around the loop illustrated in figure 2 (excluding the wire delay which is the same in each case) ;
- the size of the circuit (in transistors, t) required to implement the repeater stage;
- the number of wires (w) used for the connection, including the acknowledge signals.

Table 5 shows a comparison of these costs and performances for an isolated link, and for a 32-bit datapath constructed from such links both when each of the links runs independently with its own acknowledge, and when a single acknowledge is used for the entire 32-bit datapath - requiring the extra logic mentioned in section 2 to gather individual acknowledges and provide buffering to drive the many gates all connected to the same acknowledge wire.

In cases where the datapath width does not exactly match an integer number of instances of the chosen code, then the deficit is made up using simpler codes. Such situations are marked in the g column of table 5 which shows the number of code groups used for the 32-bit datapath.

This table shows:

- with each individual code group having its own acknowledge, the smaller the group size n, the faster it operates;
- the logic overhead of the m-of-n* codes varies between zero and 30% relative to the simple 1-hot codes;
- the wire saving provided by the m-of-n* codes varies between 0 and 30% relative to the simple 1-hot codes;
- the speed overhead of the complex m-of-n* codes is small when considered for wide, conventional datapaths where the individual group completion signals must be gathered to generate a single acknowledge.

The 2-of-7* code is a particularly interesting example having the same repeater stage logic size and performance as a 1-of-4 code but with 10% fewer wires required.

A final point to note is that when using 1-hot codes for constructing on-chip networks[3], additional wires are required to provide additional symbols for control signalling. With the complex m-of-n codes, the extra symbols can be provided from the pool of symbols that are not used for the normal data traffic, and although these may have to be treated as a special case in the completion detection logic this can usually be allowed without impeding the overall performance. In such cases, the wire-cost improvements over the 1-hot codes are even more substantial.

		Isolated link			32-bit datapath						
code	M	1 ack/group			groups used	1 ack/group			1 ack/32-bits		
		i	t	w	g	i	t	w	i	t	w
1of2	2	5	24	3	32	5	768	96	15	1214	65
1of4	4	7	54	5	16	7	864	80	15	1054	65
2of4*	4	7	110	5	16	7	1760	80	15	1950	65
2of5*	8	9	98	6	10+1of4	9	1034	65	15	1144	55
2of7*	16	9	122	8	8	9	976	64	15	1054	57
3of6*	16	11	152	7	8	11	1216	56	17	1286	49
4of8*	64	13	322	9	5+1of4	13	1664	50	19	1710	45
5of12*	256	15	378	13	4	15	1512	52	19	1558	49

Table 5: m-of-n* repeater stage costs

8. Conclusions

Throughout this paper we have ignored wire delay in our measurements and calculations since it is a constant in our comparison. However wire delay is becoming increasingly significant and when added to the delay in the loop, further reduces the impact of the extra completion detection complexity necessary to use the m-of-n* codes.

For maximum throughput whilst retaining delay-insensitive operation, 1-hot codes with separate acknowledges for each code-group deliver the best results. However the incomplete m-of-n (m-of-n*) codes allow a trade-off between performance and wire cost.

For wide datapaths using a single, common acknowledge, the m-of-n* codes can deliver almost the same throughput as the 1-hot codes at substantially reduced interconnect fabric wiring cost.

Both cases do however require the added expense of an encoder and decoder at the sender and receiver and slightly larger repeater latch stages.

Thus, whilst full m-of-n codes are too expensive, the incomplete codes presented here provide an attractive method for trading cost versus performance when constructing on-chip interconnect systems.

9. References

[1] Akella, V., Vaidya, N. H., Redinbo, G. R., "Limitations of VLSI implementations of delay-insensitive codes", Proc. 26th Int. Symp. on Fault-Tolerant Computing, Sendai, June 1996, pp. 208-217.

[2] Bainbridge, W.J., Furber, S.B., "Delay Insensitive System-on-Chip Interconnect using 1-of-4 Data Encoding", Proc. Async'01, Utah, April 2001 pp. 118-126

[3] Bainbridge, W.J., Furber, S.B., "CHAIN: A Delay-Insensitive Chip Area Interconnect", IEEE Micro, Sep/Oct 2002 pp16-23

[4] Craft, D.J., Improved CMOS Core Interconnect Approach for Advanced SoC Applications, In IP99 Europe, pp233-246, (November 1999)

[5] Fant, K.M., Brandt, S.A., NULL Convention Logic. *Tech. Report*, Theseus Logic Inc. 140, 485 N. Keller Rd. Maitland, FL 32751. 1997.

[6] Furber, S.B., Efthymiou, A., and Singh, M., "A Power-Efficient Duplex Communication System", Proc. AINT'2000, Delft, The Netherlands, 19-20 July 2000

[7] Martin, A.J,, The Limitations to Delay-Insensitivity in Asynchronous Circuits. *6th MIT Conference on Advanced Research in VLSI Processes*, 1990.

[8] Piestrak, S.J., Membership test logic for Delay-Insensitive codes, Proc Async '98, San Diego, California, April 1998, pp194-204

[9] Sobelman,G.E., Fant. K.M., CMOS Circuit Design of Threshold Gates with Hysteresis. Proc. International Symposium on Circuits and Systems, 1998.

[10] Sparsø, J.Staunstrup. J., Delay Insensitive Multi Ring Structures. *Integration, the VLSI Journal*. Vol. 15. 1993.

[11] van Overveld, W.M.C.J., On arithmetic operations with M-out-of-N codes, Computing Science Notes, no 85/02, Dept of Mathematics and Computing Science, Eindhoven University of Technology

[12] Verhoeff ,T. Delay-insensitive codes- an overview. Distributed Computing, 3(1):1-8, 1988.

[13] Semiconductor Industry Association, International Technology Roadmap for Semiconductors (ITRS) 2001, http://public.itrs.net