

---

School of Electrical, Electronic & Computer Engineering



---

# Multiple rail phase encoding circuits

Andrey Mokhov, Crescenzo D'Alessandro, Alex Yakovlev

Technical Report Series  
NCL-EECE-MSD-TR-2008-133

---

May 2008

Contact:

Andrey.Mokhov@ncl.ac.uk

Crescenzo.DAlessandro@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grant EP/C512812

NCL-EECE-MSD-TR-2008-133

Copyright © 2008 University of Newcastle upon Tyne

School of Electrical, Electronic & Computer Engineering,  
Merz Court,

University of Newcastle upon Tyne,  
Newcastle upon Tyne, NE1 7RU, UK

<http://async.org.uk/>

# Multiple rail phase encoding circuits

Andrey Mokhov, Crescenzo D'Alessandro, Alex Yakovlev

May 2008

## Abstract

Multiple rail phase encoding data communication protocol introduced recently has several unexploited advantages over traditional encodings. The main difficulties in using it arise from the absence of practical and scalable implementations of controllers for phase encoded data transmission.

This paper presents analytical justification for the practical benefits of using multiple rail phase encoding protocol and focuses on techniques for generating efficient circuits for multiple rail phase encoders, decoders and repeaters. The circuits are specified and synthesised using Conditional Partial Order Graph model which provides robust and scalable area-efficient gate-level implementation of the controllers.

## 1 Introduction

The design of the on-chip interconnect fabric is a crucial part of the design of large complex Systems-on-Chip (SoCs). The many-layered set of design requirements imposed by the ever-increasing performance constraints, coupled with the difficult task of ensuring timing closure in newer technology nodes, require the identification of fast, reliable and scalable data/control signalling techniques. Networks-on-Chip (NoCs) are one such method, responding to the need of modularity and scalability, and also adaptability: the network can be designed a priori, freeing the designer team from the task of designing ad-hoc solutions for their designs. Mentioning layers in the preceding text is apt in the context of NoC: these are typically designed using a layered approach (see, for instance, [1]), which identify and separate the requirements and characteristics of physical communication, node-to-node interaction all the way to application-specific requirements. The physical layer of a NoC, and more generally on-chip signalling, is the underlying motivation of this paper.

D'Alessandro et al. in [5] introduced the concept of phase encoding for on-chip signalling, where the information is encoded into the sequence of events over a number of lines: this provides a way to concentrate information into symbols more than by using binary encoding, with the added advantage of reliability to single-event upsets [4, 6]. However, the previous work does not describe a satisfactory method to generate encoders and decoders for this communication scheme: the structures described are limited to small number of wires (*rails*), and the scalability of these encoders/decoders (in terms of logic per number of wires in the channel) is not clearly described.

Phase encoding is a signalling technique that belongs to a class of *self-synchronous* (cf. *mesochronous* [7]) protocols, where the validity of data (i.e. clocking) is transmitted together with the data itself. The class of *delay insensitive* data transfer protocols [12] is a subclass of self-synchronous schemes. This paper presents asymptotic comparison between phase encoding and several well-known delay insensitive encodings in terms of information capacity, power consumption etc.

While conventional control logic specification and synthesis methods based on Petri nets/Signal Transition Graphs [3, 11] or on Burst-mode Finite State Machines [10] have certain advantages, they cannot be directly applied to the problem of phase encoders circuitry synthesis as shown in [9]. In particular, the specification size of *matrix phase encoder* (see Section 5.2) is exponential w.r.t. the number of output rails in these models. To overcome this, the paper defines and solves the problem of specification and

synthesis of multiple rail phase encoding circuits using the model of *Conditional Partial Order Graphs* which was recently introduced in [9], providing efficient gate-level implementations for the circuits.

## 2 Phase encoding essentials

*Phase encoding* protocol was introduced by D'Alessandro *et al.* in [5]. The initial idea was to encode an information bit into phase difference between two switching signals. The idea was further extended into *multiple-rail phase encoding* [6] which uses several wires for communication and data is encoded in the order of occurrence of transitions on the communication lines. Figure 1 shows an example of 4-wire phase encoding communication channel. The order of rising signals on wires  $\{a, b, c, d\}$  indicates that permutation  $abdc$  is being sent. In total it is possible to send  $n!$  different permutations over an  $n$ -wire channel. This makes the multiple rail phase encoding protocol very attractive for its information efficiency.

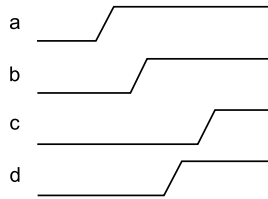


Figure 1: Data symbol in multiple-rail phase encoding channel

Table 1 contains several important characteristics of multiple-rail phase encoding protocol. The amount of information that can be sent in a symbol in  $n$ -wire channel grows faster than linearly. This is due to the fact that

$$\log_2(n!) = \sum_{k=1}^n \log_2 k \approx \int_1^n \log_2 x dx = x(\log_2 x - \ln 2)|_1^n = \Theta(n \log_2 n) \quad (1)$$

number of wires	number of permutations	bits per data symbol	bits per wire/time slot	transitions per bit
2	2	1	1/2	2
3	6	2	2/3	3/2
4	24	4	1	1
5	120	6	6/5	5/6
6	720	9	3/2	2/3
$n$ (asymptotic)	$n!$	$\Theta(n \log_2 n)$	$\Theta(\log_2 n)$	$\Theta(\frac{1}{\log_2 n})$

Table 1: Phase encoding protocol characteristics

We use the standard notation for describing the asymptotic behaviour of functions:

- $f(n) \in O(g(n))$  means that  $f$  is bounded above by  $g$  (up to a constant factor) asymptotically i.e.

$$\exists C > 0, n_0 : \forall n > n_0, |f(n)| \leq |Cg(n)|$$

- $f(n) \in \Theta(g(n))$  means that  $f$  is bounded both above and below by  $g$  asymptotically i.e.

$$\exists C_1 > 0, C_2 > 0, n_0 : \forall n > n_0, |C_1g(n)| \leq |f(n)| \leq |C_2g(n)|$$

- $f(n) \in \Omega(g(n))$  means that  $f$  is bounded below by  $g$  asymptotically i.e.

$$\exists C > 0, n_0 : \forall n > n_0, |Cg(n)| \leq |f(n)|$$

Asymptotic behaviour of other characteristics is based on (1), e.g. number of bits that can be sent in a time slot (the time separation between signals switching) is

$$\frac{\log_2(n!)}{n} = \frac{\Theta(n \log_2 n)}{n} = \Theta(\log_2 n)$$

Protocol	number of data combinations	bits per symbol	time slots per symbol	bits per wire	bits per time slot	transitions per bit
phase encoding	$n!$	$\Theta(n \log_2 n)$	$n$	$\Theta(\log_2 n)$	$\Theta(\log_2 n)$	$\Theta(\frac{1}{\log_2 n})$
dual rail	$2^{\lfloor \frac{n}{2} \rfloor}$	$\Theta(n)$	1	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
1-of- $n$ encoding	$n$	$\Theta(\log_2 n)$	1	$\Theta(\frac{\log_2 n}{n})$	$\Theta(\log_2 n)$	$\Theta(\frac{1}{\log_2 n})$
$\lfloor \frac{n}{2} \rfloor$ -of- $n$ encoding	$\binom{n}{\lfloor \frac{n}{2} \rfloor}$	$\Theta(n)$	1	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$

Table 2: Asymptotic comparison of DI communication protocols

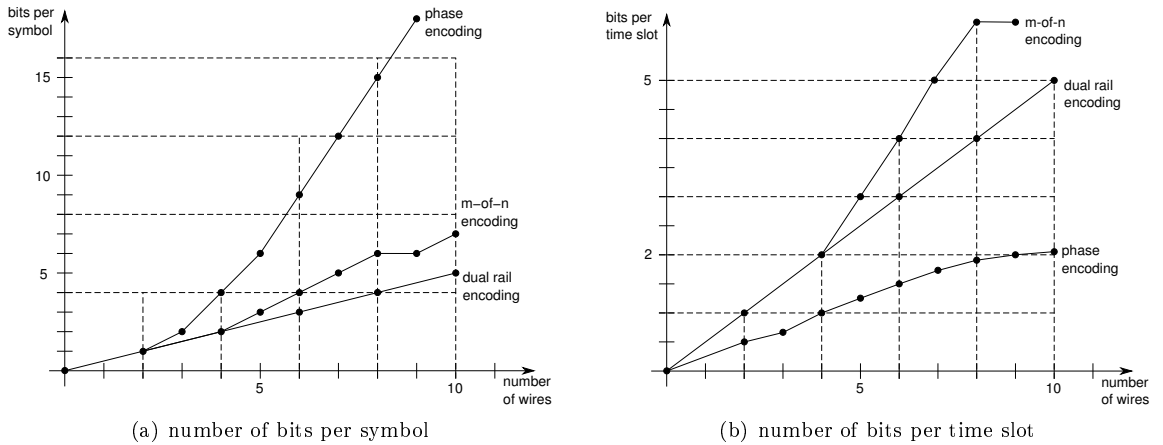


Figure 2: Numeric comparison of DI communication protocols: information efficiency

Table 2 shows asymptotic comparison of characteristics of phase encoding, *dual rail* and *m-of-n encoding* protocols [12] for  $n$  wires. You can see that phase encoding loses only in one parameter - bits per time slot:  $\Theta(n)$  (dual-rail) and  $O(n)$  (*m-of-n encoding*) vs  $\Theta(\log_2 n)$  (phase encoding). This is because it needs  $n$  time slots to send one data symbol. Phase encoding is a clear winner in all the other parameters. The last one is particularly interesting: number of signal transitions per data bit. Phase encoding protocol needs only  $\Theta(\frac{1}{\log_2 n})$  transitions per data bit so the more wires the channel has the cheaper (in terms of power consumption) the bits are. Theoretically if we had infinite number of wires we could send a bit of information for free. A special case of *m-of-n encoding* with  $m = 1$  (*one hot encoding* [12]) also needs only  $\Theta(\frac{1}{\log_2 n})$  transitions per data bit but its information efficiency is very low so it is not reasonable to use it for large values of  $n$ .

*m-of-n encoding* [8, 12] cannot principally beat dual-rail asymptotically in terms of number of bits per symbol: it reaches its maximum information efficiency when  $m = \lfloor \frac{n}{2} \rfloor$ . The number of bits in symbol in this case is

$$\log_2 \binom{n}{\lfloor \frac{n}{2} \rfloor} = \log_2 \left( \frac{n!}{\lfloor \frac{n}{2} \rfloor! \lfloor \frac{n}{2} \rfloor!} \right) \approx \log_2 \left( \frac{\sqrt{2\pi n} n^n e^{-n}}{\pi n n^{2-n} e^{-n}} \right) = \log_2 \left( \frac{2^n}{\sqrt{\pi n}} \right) = \Theta(n) \quad (2)$$

Here we used *Stirling approximation* of factorial  $n! \approx \sqrt{2\pi n} n^n e^{-n}$ . Equation (2) gives the upper bound of information efficiency for  $m$ -of- $n$  encoding protocol. The lower bound is achieved for one hot encoding and is equal to  $\log_2 \binom{n}{1} = \log_2 n$ . For other values of  $0 < m < n$  information efficiency varies but remains within  $\Omega(\log_2 n) \cap O(n)$  interval. (Intersection  $\Omega(f(n)) \cap O(g(n))$  denotes a set of functions bounded by  $f(n)$  below and by  $g(n)$  above.)

So, asymptotically  $m$ -of- $n$  encoding is equivalent to dual-rail in terms of information efficiency though it is better by constant factor approximately equal to 2:

$$\lim_{n \rightarrow \infty} \frac{\log_2 \left( \frac{2^n}{\sqrt{\frac{\pi n}{2}}} \right)}{\log_2 2^{\lfloor \frac{n}{2} \rfloor}} = \lim_{n \rightarrow \infty} \frac{\log_2 2^n - \log_2 \sqrt{\frac{\pi n}{2}}}{\log_2 2^{\lfloor \frac{n}{2} \rfloor}} = \lim_{n \rightarrow \infty} \frac{n - \Theta(\log_2 n)}{\lfloor \frac{n}{2} \rfloor} = 2$$

In terms of power consumption  $m$ -of- $n$  encoding can beat dual-rail and approach phase encoding. The lower bound of power consumption is achieved for one hot encoding ( $m = 1$ ) and is equal to

$$\frac{m}{\log_2 n} = \frac{1}{\log_2 n}$$

But the upper bound (when  $m = \lfloor \frac{n}{2} \rfloor$ ) is the same as of dual-rail:

$$\frac{m}{\Theta(n)} = \frac{\lfloor \frac{n}{2} \rfloor}{\Theta(n)} = \Theta(1)$$

So, power consumption of  $m$ -of- $n$  encoding protocol is in interval  $\Omega(\frac{1}{\log_2 n}) \cap O(1)$ . Interestingly the interval spans exactly between phase-encoding and dual rail protocols.

The numeric comparison of the three protocols for up to 10-wire communication channels is shown in Figure 2. The results of  $m$ -of- $n$  encoding are calculated for the most informative case when  $m = \lfloor \frac{n}{2} \rfloor$ . Subfigure (a) shows information efficiency with respect to a symbol size, while Subfigure (b) - with respect to a time slot. Notice that phase encoding is dominating on the first graph and shows rather bad results on the second. However this should not be misleading: although  $n$ -wire phase encoding protocol needs  $n$  time slots to send a data symbol these time slots can be significantly shorter than that of dual rail or  $m$ -of- $n$  protocols because each wire switches only once in these  $n$  time slots. Therefore the sending and receiving circuitry of a particular wire can work at a speed  $n$  times slower than the communication channel as a whole. It allows the time slots to be compressed much more than for dual rail and  $m$ -of- $n$  encoding protocols and achieve higher information density over time. Thus phase encoding is potentially optimal in terms of area (number of bits per wire), speed (number of bits per time interval) and power (number of signal transitions per bit).

All these comparisons are theoretical and need experimental refinement. However existing implementations of multiple rail phase encoding circuitry are area inefficient and usually designed by hand for a particular number of wires. This work presents a number of techniques for generating circuits for multiple-rail phase encoding senders/receivers/repeaters. Figure 3 shows the overall phase encoding communication circuitry. Rectangular boxes represent functional units for conversion between different data encodings. The paper covers implementation of the units in tinted boxes. Most of the phase encoding circuits presented in the work are synthesised using Conditional Partial Order Graphs [9] which are introduced in the next two sections.

### 3 Conditional Partial Order Graphs

*Conditional partial order graph* (CPOG) is a tuple  $H(V, E, X, \phi)$  where  $V$  is the set of *vertices* (or *nodes*),  $E \subseteq V \times V$  is the set of ordered pairs of vertices, called *arcs*,  $X$  is the set of Boolean variables, and function  $\phi : V \cup E \rightarrow \mathcal{F}(X)$  assigns a *condition* to every vertex and arc in the graph. A condition on a vertex

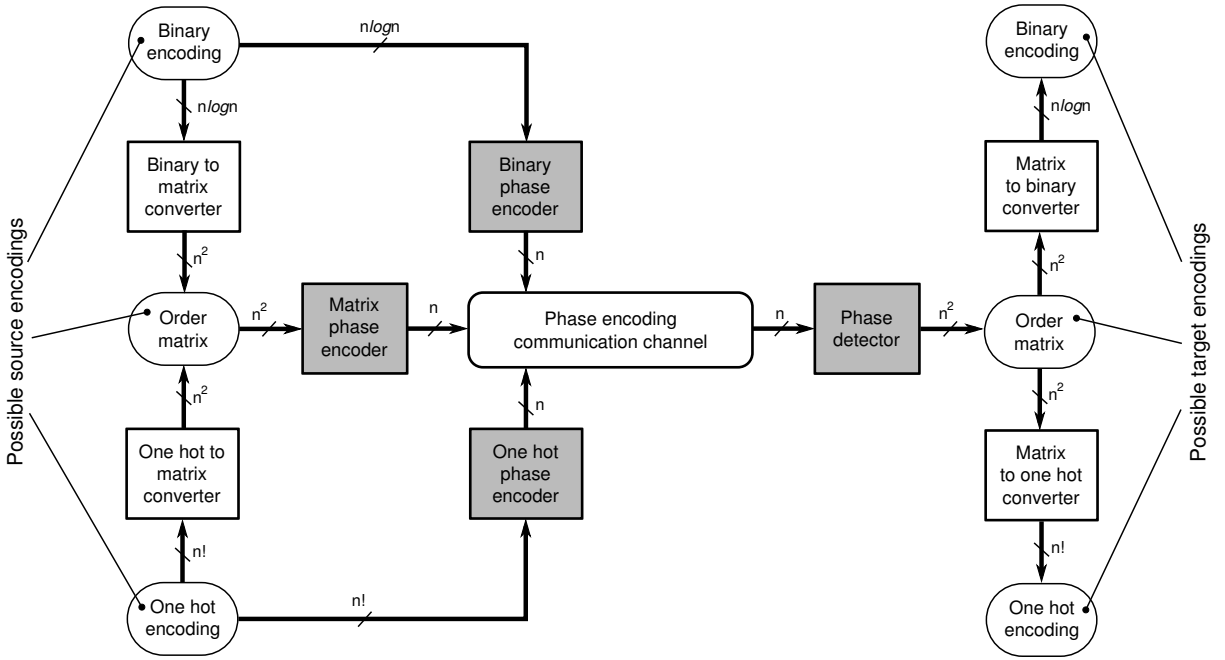


Figure 3: Phase encoding communication circuitry (numbers of wires are shown beside communication channels; implementation of functional units that are drawn tinted is covered in the paper)

or arc  $z \in V \cup E$  is a Boolean function  $\phi(z) \in \mathcal{F}(X)$  where  $\mathcal{F}(X)$  is the set of all Boolean functions over variables in  $X$ . Let's also define  $\phi(z) = 0$  for  $z \notin V \cup E$  in order to simplify some of the further computations.

The following subsections introduce an algebra over CPOGs.

### 3.1 Addition

The result of *addition* of  $H_1(V_1, E_1, X_1, \phi_1)$  and  $H_2(V_2, E_2, X_2, \phi_2)$  is CPOG  $H(V_1 \cup V_2, E_1 \cup E_2, X_1 \cup X_2, \phi)$ , where  $\forall z, \phi(z) = \phi_1(z) + \phi_2(z)$ . Here  $\phi_1 + \phi_2$  stands for Boolean disjunction of functions  $\phi_1$  and  $\phi_2$ . We will use standard notation for addition:  $H = H_1 + H_2$ .

CPOG addition is commutative ( $H_1 + H_2 = H_2 + H_1$ ) and associative ( $(H_1 + H_2) + H_3 = H_1 + (H_2 + H_3)$ ) and thus redundant brackets can be omitted when more than two CPOGs are added.

### 3.2 Scalar multiplication

A CPOG  $H(V, E, X, \phi)$  can be *multiplied* by a Boolean function  $f \in \mathcal{F}(Y)$  (which in our context can be called *scalar*). The resultant CPOG is  $H'(V, E, X \cup Y, \phi')$  where  $\forall z, \phi'(z) = f\phi(z)$  ( $f\phi$  stands for Boolean conjunction of functions  $f$  and  $\phi$ ). We will use standard notation for scalar multiplication:  $H' = fH$ .

Scalar multiplication and addition have the following common properties:

- Left distributivity:  $(f + g)H = fH + gH$ ;
- Right distributivity:  $f(H_1 + H_2) = fH_1 + fH_2$ ;
- Associativity:  $f(gH) = (fg)H$ ;

### 3.3 Projection

A *projection* of a CPOG  $H(V, E, X, \phi)$  under constraint  $x = \alpha$  ( $x \in X$ ) is denoted as  $H|_{x=\alpha}$  and is equal to CPOG  $H'(V, E, X \setminus \{x\}, \phi|_{x=\alpha})$  where notation  $\phi|_{x=\alpha}$  means that variable  $x$  is substituted with constant Boolean value  $\alpha$  in all the functions  $\phi(z), z \in V \cup E$ . Projection is a *commutative operation* i.e.  $(H|_{x=\alpha})|_{y=\beta} = (H|_{y=\beta})|_{x=\alpha}$ .

A *complete projection* of a CPOG  $H$  is such a projection that all the variables in  $X$  are constrained to constants. It is denoted as  $H|_\psi$  where  $\psi : X \rightarrow \{0, 1\}$  is an *assignment function* that assigns a Boolean value to every variable in  $X$ . Complete projection is a CPOG whose vertex and arc conditions are only Boolean constants  $\phi|_\psi$  (either 0 or 1).

Let  $H(V, E, \emptyset, \phi)$  be a complete projection ( $\forall z, \phi(z) \in \{0, 1\}$ ). We can construct a graph  $G(V_G, E_G)$  such that

$$V_G = \{v \in V | \phi(v) = 1\}$$

$$E_G = \{e = (a, b) \in E | \phi(a)\phi(b)\phi(e) = 1\}$$

In other words  $G$  contains only those vertices and arcs whose conditions in  $H$  are constant 1.

A complete projection  $H$  is *valid* iff its corresponding graph  $G$  is a *directed acyclic graph* [2].

The obtained DAG  $G(V_G, E_G)$  can be further converted into a corresponding *partial order* [2]  $P(V_G, \prec)$  such that  $a \prec b$  iff  $G$  contains an oriented path between vertices  $a$  and  $b$ . Let this operation of partial order construction from a valid CPOG complete projection  $H$  be shortly denoted as  $\mathbf{po}$ :  $P = \mathbf{po}(H)$  and the inverse operation as  $\mathbf{po}^{-1}$ :  $H = \mathbf{po}^{-1}(P)$ . Note, that  $\mathbf{po}^{-1}$  is a *right inverse* operation i.e.  $\mathbf{po}(\mathbf{po}^{-1}(PO)) = PO$  but  $\mathbf{po}^{-1}(\mathbf{po}(H))$  is not necessarily equal to  $H$ .

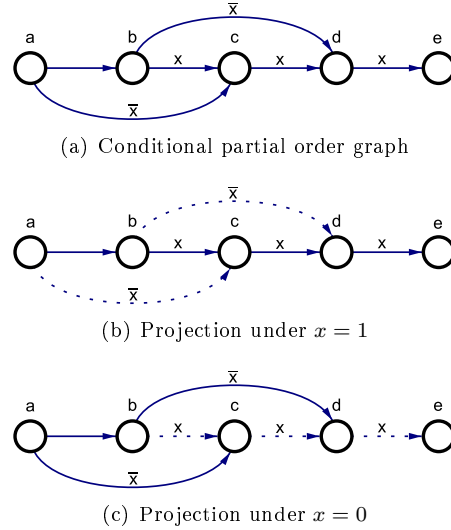


Figure 4: CPOG and its projections

An example of a CPOG and its projections is shown in Figure 4. Subfigure (a) shows the initial graph. The conditional arc functions are indicated over the arcs: arcs  $(b, c)$  and  $(c, d)$  have conditional function  $f = x$ ; the function on arcs  $(a, c)$  and  $(b, d)$  is  $f = \bar{x}$ ; arcs  $(a, b)$ ,  $(d, e)$  and vertices  $a \dots d$  are *unconditional* i.e. their functions are constant Boolean 1. Such functions are not shown on diagrams for simplicity. The only conditional vertex  $e$  has condition  $f = x$  which is shown next to its label separated by a colon.

Figure 4(b) shows the complete projection under  $x = 1$ . The dotted arcs are those that turn to have constant 0 conditions after the projection and therefore will be excluded from the resultant partial order. The solid arcs have constant 1 conditions. The partial order defined with the projection is a simple series of events:  $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$ .

Complete projection under condition  $x = 0$  (Figure 4(c)) results in the following partial order. Events  $b$  and  $c$  can happen only after  $a$ . There is no constraint between them, thus they can be concurrent. Event  $d$  can happen only after event  $b$ . Event  $e$  is excluded from the partial order; note, that this implies exclusion of arc  $(d, e)$  as well.

### 3.4 $\Psi$ -equivalence

Let *assignment set*  $\Psi = \{\psi_1, \psi_2, \dots, \psi_n\}$  be the set of  $n$  assignment functions  $\psi_k : X \rightarrow \{0, 1\}$ . Two Boolean functions  $f, g \in \mathcal{F}(X)$  are  $\Psi$ -*equivalent* iff they evaluate to the same values over the assignment set  $\Psi$ :

$$\forall \psi_k \in \Psi, f|_{\psi_k} = g|_{\psi_k}$$

A CPOG  $H$  is  $\Psi$ -*well-formed* iff every complete projection  $H|_{\psi}, \psi \in \Psi$  is valid.  $\Psi$ -well-formed CPOGs  $H_1$  and  $H_2$  are  $\Psi$ -equivalent iff they produce the same partial orders:

$$\forall \psi_k \in \Psi, \mathbf{po}(H_1|_{\psi_k}) = \mathbf{po}(H_2|_{\psi_k})$$

We will use the following notation for  $\Psi$ -equivalence:  $f \stackrel{\Psi}{\sim} g$  or  $H_1 \stackrel{\Psi}{\sim} H_2$ .  $\Psi$ -equivalence is a proper *equivalence relation* [2] as it satisfies the following properties:

- Reflexivity:  $a \stackrel{\Psi}{\sim} a$ ;
- Symmetry:  $a \stackrel{\Psi}{\sim} b \Rightarrow b \stackrel{\Psi}{\sim} a$ ;
- Transitivity:  $a \stackrel{\Psi}{\sim} b \wedge b \stackrel{\Psi}{\sim} c \Rightarrow a \stackrel{\Psi}{\sim} c$ .

## 4 CPOG Synthesis

The previous section showed that a CPOG can contain several partial orders in a compressed form and thus can be used to specify a system with several behavioural scenarios. [9] showed that it is possible to synthesise a compact CPOG system specification given its description as a set of partial orders corresponding to different scenarios in the modelled system.

Formally, let  $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$  be the set of  $n$  given partial orders. The objective is to synthesise CPOG  $H(V, E, X, \phi)$  and *assignment set*  $\Psi = \{\psi_1, \psi_2, \dots, \psi_n\}$  such that  $\psi_k$  are assignment functions and

$$\forall \psi_k \in \Psi, \mathbf{po}(H|_{\psi_k}) = P_k$$

The idea behind the synthesis approach presented in [9] is to represent  $H$  as the following sum of given partial orders:

$$H = f_1 \mathbf{po}^{-1}(P_1) + \dots + f_n \mathbf{po}^{-1}(P_n) = \sum_{k=1}^n f_k \mathbf{po}^{-1}(P_k) \quad (3)$$

Now control signals  $X$ , functions  $f_k \in \mathcal{F}(X)$  and  $\psi_k$  should be selected so that  $f_k|_{\psi_k} = 1$  and  $f_k|_{\psi_j} = 0, j \neq k$ . This can be done in different ways depending on the chosen encoding scheme. The following three encoding schemes will be used for synthesis of phase encoding senders in this paper.

### 4.1 One-hot encoding scheme

In this scheme we use  $n$  control signals  $X = \{x_1, x_2, \dots, x_n\}$ . Functions  $f_k$  and  $\psi_k$  ( $k = 1 \dots n$ ) are trivial:  $f_k = x_k, \psi_k(x_k) = 1, \psi_k(x_j) = 0, j \neq k$ .

One-hot scheme provides a simple and intuitive way of encoding but it is inefficient because of the large size of control signals set  $X$ :  $|X| = n$ .

Here is an example of synthesis of a CPOG containing two partial orders  $P_1 = \{a \prec b, a \prec c, b \prec c\}$  and  $P_2 = \{b \prec a, b \prec c\}$ . The control signals set is  $X = \{x_1, x_2\}$ . The complete projections obtained from the partial orders are  $H_1 = \mathbf{po}^{-1}(P_1) = \overset{\overset{a}{\circ}}{\circ} \xrightarrow{\quad} \overset{\overset{b}{\circ}}{\circ} \xrightarrow{\quad} \overset{\overset{c}{\circ}}{\circ}$  and  $H_2 = \mathbf{po}^{-1}(P_2) = \overset{\overset{a}{\circ}}{\circ} \xrightarrow{\quad} \overset{\overset{b}{\circ}}{\circ} \xrightarrow{\quad} \overset{\overset{c}{\circ}}{\circ}$ . The result of application of (3) to partial orders  $\mathcal{P} = \{P_1, P_2\}$  is CPOG  $H$ :

$$\begin{aligned}
 H &= x_1 \cdot \begin{array}{c} \text{a} \\ \circ \end{array} \xrightarrow{\quad} \begin{array}{c} \text{b} \\ \circ \end{array} \xrightarrow{\quad} \begin{array}{c} \text{c} \\ \circ \end{array} + x_2 \cdot \begin{array}{c} \text{a} \\ \circ \end{array} \xrightarrow{\quad} \begin{array}{c} \text{b} \\ \circ \end{array} \xrightarrow{\quad} \begin{array}{c} \text{c} \\ \circ \end{array} = \\
 &= \begin{array}{c} \text{a}:x_1 \\ \circ \end{array} \xrightarrow{x_1} \begin{array}{c} \text{b}:x_1 \\ \circ \end{array} \xrightarrow{x_1} \begin{array}{c} \text{c}:x_1 \\ \circ \end{array} + \begin{array}{c} \text{a}:x_2 \\ \circ \end{array} \xrightarrow{x_2} \begin{array}{c} \text{b}:x_2 \\ \circ \end{array} \xrightarrow{x_2} \begin{array}{c} \text{c}:x_2 \\ \circ \end{array} = \begin{array}{c} \text{a}:x_1+x_2 \\ \circ \end{array} \xrightarrow{x_1} \begin{array}{c} \text{b}:x_1+x_2 \\ \circ \end{array} \xrightarrow{x_1+x_2} \begin{array}{c} \text{c}:x_1+x_2 \\ \circ \end{array}
 \end{aligned}$$

Bearing in mind the assignment set  $\Psi = \{\psi_1, \psi_2\} = \{(1, 0), (0, 1)\}$  it is possible to optimise the obtained result into a simpler  $\Psi$ -equivalent CPOG:

$$\begin{array}{c} \text{a}:x_1+x_2 \\ \circ \end{array} \xrightarrow{x_1} \begin{array}{c} \text{b}:x_1+x_2 \\ \circ \end{array} \xrightarrow{x_1+x_2} \begin{array}{c} \text{c}:x_1+x_2 \\ \circ \end{array} \sim \begin{array}{c} \text{a} \\ \circ \end{array} \xrightarrow{x_1} \begin{array}{c} \text{b} \\ \circ \end{array} \xrightarrow{x_2} \begin{array}{c} \text{c} \\ \circ \end{array}$$

Details of the logic optimisation technique that reduces the size of CPOGs based on the notion of  $\Psi$ -equivalence can be found in [9].

## 4.2 Binary encoding scheme

In binary scheme only  $m = \lceil \log_2 n \rceil$  control variables  $X = \{x_1, x_2, \dots, x_m\}$  are used which is the theoretical minimum. Let  $b_{jk}$  denote  $j$ -th bit of integer number  $k$ . Then we can define functions  $\psi_k$  and  $f_k$  ( $k = 1 \dots n$ ) as:

$$\psi_k(x_j) = b_{(j-1)(k-1)}, \quad f_k = \bigwedge_{j=1}^m (x_j \Leftrightarrow \psi_k(x_j))$$

For example, if  $n = 3$  we will get  $\psi_1 = (0, 0)$ ,  $\psi_2 = (1, 0)$  and  $\psi_3 = (0, 1)$ . Functions  $f_k$  are:  $f_1 = (x_1 \Leftrightarrow 0)(x_2 \Leftrightarrow 0) = \bar{x}_1 \bar{x}_2$ ,  $f_2 = x_1 \bar{x}_2$  and  $f_3 = \bar{x}_1 x_2$ .

If we apply binary encoding scheme for synthesis of a CPOG containing  $P_1 = \{a \prec b, a \prec c, b \prec c\}$  and  $P_2 = \{b \prec a, b \prec c\}$  (as in Section 4.1) we get

$$H = x \cdot \begin{array}{c} \text{a} \\ \circ \end{array} \xrightarrow{\quad} \begin{array}{c} \text{b} \\ \circ \end{array} \xrightarrow{\quad} \begin{array}{c} \text{c} \\ \circ \end{array} + \bar{x} \cdot \begin{array}{c} \text{a} \\ \circ \end{array} \xrightarrow{\quad} \begin{array}{c} \text{b} \\ \circ \end{array} \xrightarrow{\quad} \begin{array}{c} \text{c} \\ \circ \end{array} = \begin{array}{c} \text{a}:x+\bar{x} \\ \circ \end{array} \xrightarrow{x} \begin{array}{c} \text{b}:x+\bar{x} \\ \circ \end{array} \xrightarrow{x+\bar{x}} \begin{array}{c} \text{c}:x+\bar{x} \\ \circ \end{array} \sim \begin{array}{c} \text{a} \\ \circ \end{array} \xrightarrow{x} \begin{array}{c} \text{b} \\ \circ \end{array} \xrightarrow{\bar{x}} \begin{array}{c} \text{c} \\ \circ \end{array}$$

As one can see the selected encoding scheme does not affect the structure of the optimised CPOG. However, it affects the complexity of the functions and the size of the physical controller implementation.

## 4.3 Matrix encoding scheme

The size of the control signals set in this scheme does not depend on the number of scenarios  $|\mathcal{P}|$ . It depends only on the number of different events in the system  $|V|$ . In particular,  $X = \{x_{jk} | j = 1 \dots |V|, k = 1 \dots |V|\}$ , so  $|X| = |V|^2$  and thus  $X$  can be called *control matrix*. Control matrix has enough information capacity to describe any partial order  $P(V', \prec)$  of any subset  $V' \subseteq \{e_1, e_2, \dots, e_{|V|}\}$  of  $|V|$  events:

$$\psi(x_{jk}) = \begin{cases} 1 & \text{if } (e_j \in V') \wedge (e_k \in V') \wedge (e_j \prec e_k) \\ 0 & \text{otherwise} \end{cases} \quad (j \neq k)$$

$$\psi(x_{kk}) = \begin{cases} 1 & \text{if } (e_k \notin V') \\ 0 & \text{otherwise} \end{cases}$$

Matrix encoding scheme is general in the sense that it can be used to encode any possible behavioural scenario of a system with  $n$  events in a reasonably compact and understandable way. It is a trade-off between one-hot encoding which is straightforward but inefficient in terms of the number of control signals and binary encoding which has the least possible number of control signals but more complicated encoding functions which are not affordable in some cases as will be demonstrated later.

The encoding matrices for the example from the previous sections are shown in Table 3. Instead of applying (3) for the resultant CPOG synthesis it is possible to use generic solution (see Figure 5(a)).

$\psi(x_{jk})$	$k = 1$	$k = 2$	$k = 3$
$j = 1$	0	1	1
$j = 2$	0	0	1
$j = 3$	0	0	0

 (a)  $\psi_1$  for  $P_1 = \{a \prec b, a \prec c, b \prec c\}$ 

$\psi(x_{jk})$	$k = 1$	$k = 2$	$k = 3$
$j = 1$	0	0	0
$j = 2$	1	0	1
$j = 3$	0	0	0

 (b)  $\psi_2$  for  $P_2 = \{b \prec a, b \prec c\}$ 

Table 3: Assignment functions for the matrix encoding example

This generic solution can be optimised taking into account assignment set  $\Psi = \{\psi_1, \psi_2\}$  from Table 3 producing a simpler CPOG in Figure 5(b). Note that again it is structurally similar to the CPOGs obtained using different encoding schemes from the previous sections.



Figure 5: CPOG synthesis and optimisation using matrix encoding scheme

## 5 Phase encoding repeater

The first multiple rail phase encoding circuit that we are going to synthesise is *phase encoding repeater* [4] – a circuit able to regenerate the deteriorating phase difference between signals in phase encoding communication channel.

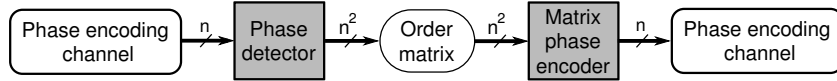


Figure 6: Phase encoding repeater circuitry

Phase encoding repeater consists of two functional parts: a receiver (a *phase detector*, which determines the order of the incoming transitions) and a sender (a *phase encoder*, which has to generate a series of transitions in the order they were received) as shown in Figure 6.

It should be noted that we assume here that the phase encoded symbols arriving via the communication channel to the repeater are correct i.e. all transitions are ordered with appropriate time slot condition. The issues of error behaviour and noise tolerance have been addressed in [4].

### 5.1 Phase detector

Phase detector for  $n$ -wire communication channel consists of  $\binom{n}{2}$  *mutual-exclusion (mutex) elements* [4]: each for every pair of wires. A possible implementation of a mutex is shown in Figure 7(a): it consists of a pair of cross-coupled NAND gates (an SR-latch) and a simple metastability filter constructed from two inverters. To determine the order of  $n$  transitions it is possible to compare their arrival times pairwise (see Figure 7(b) for an example of 3-wire phase detector).

The result of phase detection can be seen as a control matrix from Section 4.3 with diagonal elements set to 0. Therefore the subsequent phase encoder should be synthesised using matrix encoding scheme to avoid additional encoding conversion circuitry.

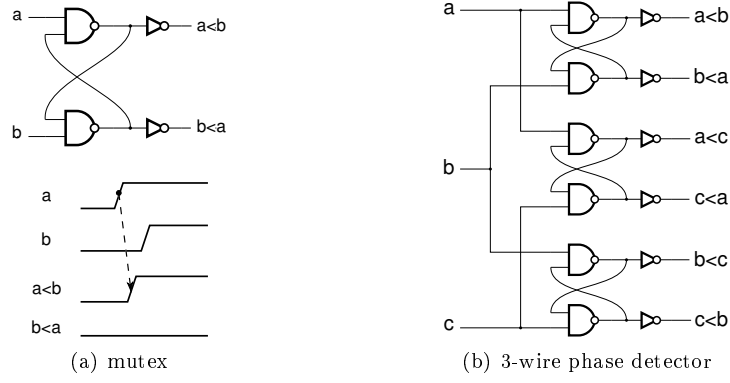


Figure 7: Phase detection

## 5.2 Matrix phase encoder

Given control matrix  $X = \{x_{jk} | j = 1 \dots n, k = 1 \dots n, j \neq k\}$  containing pairwise comparisons of arrival times of  $n$  transitions *matrix phase encoder* should generate  $n$  output transitions in the specified order.

The control matrix  $X$  coming from the phase detector has  $n!$  different possible values assignments  $\psi_k : X \rightarrow \{0, 1\}, k = 1 \dots n!$  each of them corresponding to a partial order of a particular scenario. Conditional partial order graph  $H(V, E, X, \phi)$  containing all of them as its projections has the following generic description:

$$\begin{aligned}
 V &= \{e_j | j = 1 \dots n\} \\
 E &= \{(e_j, e_k) | j = 1 \dots n, k = 1 \dots n, j \neq k\} \\
 X &= \{x_{jk} | j = 1 \dots n, k = 1 \dots n, j \neq k\} \\
 \phi(e_j) &= 1, j = 1 \dots n \\
 \phi((e_j, e_k)) &= x_{jk}, j = 1 \dots n, k = 1 \dots n, j \neq k
 \end{aligned} \tag{4}$$

Example of such a CPOG for the case of 3 wires is shown in Figure 8(a).

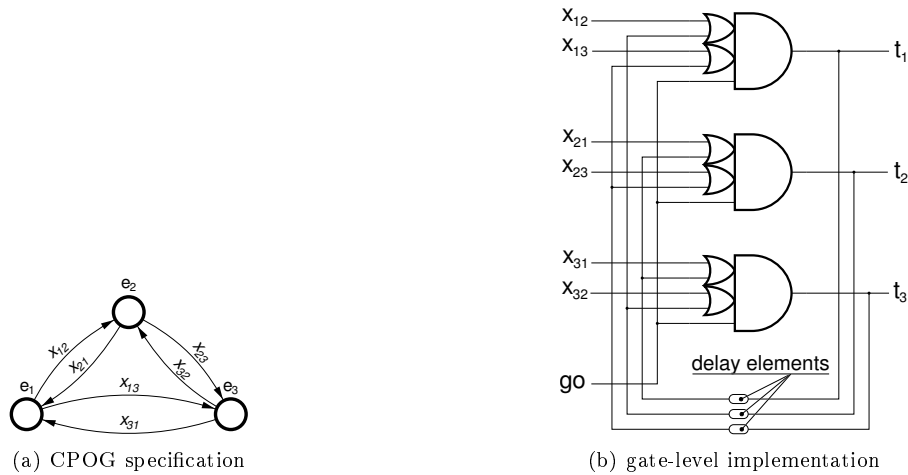


Figure 8: 3-wire matrix phase encoder specification and implementation

Having synthesised the CPOG it is possible to derive Boolean equations for physical controller implementation. The controller should have  $2^{\binom{n}{2}} = n^2 - n$  inputs  $X = \{x_{jk} | j = 1 \dots n, k = 1 \dots n, j \neq k\}$  and  $n$  outputs  $T = \{t_1, t_2, \dots, t_n\}$ . Output transition  $t_k$  is enabled to fire if all the preceding (w.r.t. to the partial order specified by control matrix  $X$ ) transitions have already fired i.e.

$$t_k = \phi(e_k) \cdot \bigwedge_{\substack{1 \leq j \leq n \\ j \neq k}} (\phi(e_j) \cdot \phi((e_j, e_k)) \Rightarrow t_j) \tag{5}$$

(Here  $a \Rightarrow b$  stands for *Boolean implication* [2] indicating 'b if a' relation. It shouldn't be mixed with  $a \Leftrightarrow b$  which is *Boolean equivalence* [2] indicating 'b if and only if a' relation.)

This generic equation can be simplified taking into account the particular CPOG specification (4):

$$t_k = \bigwedge_{\substack{1 \leq j \leq n \\ j \neq k}} (x_{jk} \Rightarrow t_j) = \bigwedge_{\substack{1 \leq j \leq n \\ j \neq k}} (\overline{x_{jk}} + t_j)$$

Another optimisation opportunity is to exploit the fact that the control matrix  $X$  specifies a *total order* [2] (a special case of partial order  $P(V, \prec)$  such that  $(a \prec b) \Leftrightarrow \neg(b \prec a)$  i.e. every pair of elements in  $V$  is ordered). In our case it means that  $\overline{x_{jk}} = x_{kj}$ :

$$t_k = \bigwedge_{\substack{1 \leq j \leq n \\ j \neq k}} (x_{kj} + t_j)$$

As the phase encoder should maintain a certain time separation  $\Delta$  between the generated transitions it is necessary to modify the above equation to take this fact into account:

$$t_k = \bigwedge_{\substack{1 \leq j \leq n \\ j \neq k}} (x_{kj} + t_j^\Delta)$$

where  $t_j^\Delta$  represents signal  $t_j$  delayed for  $\Delta$  time units. For the purpose of resetting the controller into the initial state after generating the desired sequence of transitions we should also add control signal  $go$  that would serve as an initiating and resetting signal:

$$t_k = go \cdot \bigwedge_{\substack{1 \leq j \leq n \\ j \neq k}} (x_{kj} + t_j^\Delta) \quad (6)$$

The gate-level implementation of the controller specified with equation (6) is shown in Figure 8(b).

Implementation of phase encoding repeater consisting of phase detector and phase encoder is shown in Figure 9. Generation of signal  $go$  can be done in a number of ways depending on whether the repeater should be *early-propagative* or not as well as on several other criteria which are out of the scope of this paper and are discussed in details in [4].

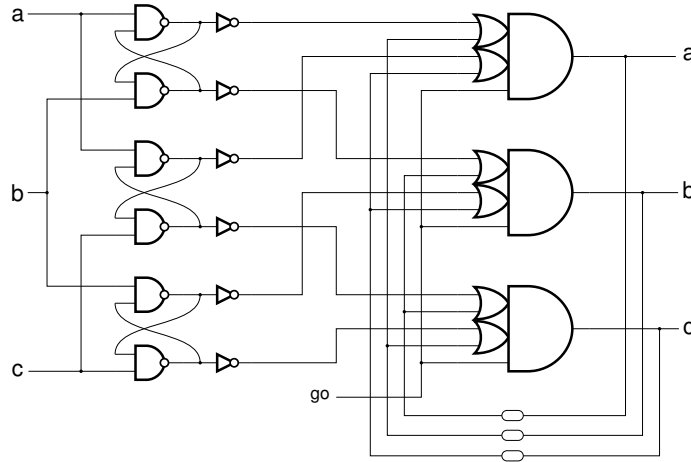


Figure 9: Phase encoding repeater

## 6 One hot phase encoder

One hot encoding can be used to specify the order of signal transitions for small values of  $n$  (for large values of  $n$  the method is inappropriate because it needs  $n!$  wires). To send data presented in one hot

encoding it is possible to convert it first into matrix form using *one hot code to matrix converter* and then to send the result using matrix phase encoder. Alternatively, to avoid unnecessary conversions it is possible to send one hot data directly using *one hot phase encoder* as shown in Figure 10.

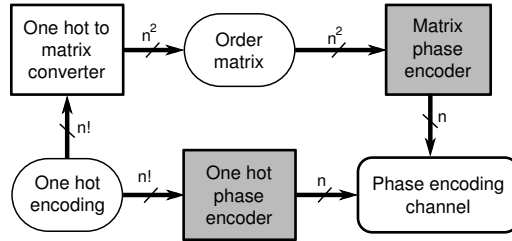


Figure 10: One hot phase encoder

There are  $n!$  partial orders  $\mathcal{P} = \{P_1, P_2, \dots, P_{n!}\}$  specifying the  $n!$  scenarios. Using one hot encoding scheme (Section 4.1) it is possible to synthesise CPOG containing all of them.

Consider the following example of synthesis of 3-wire one hot phase encoder. There are 6 one hot control signals  $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$  and 6 partial orders corresponding to the possible permutations of output transitions  $T = \{a, b, c\}$ :

#	permutation	one hot encoding	partial order
1	$(a, b, c)$	$\psi_1 = (1, 0, 0, 0, 0, 0)$	
2	$(a, c, b)$	$\psi_2 = (0, 1, 0, 0, 0, 0)$	
3	$(b, a, c)$	$\psi_3 = (0, 0, 1, 0, 0, 0)$	
4	$(b, c, a)$	$\psi_4 = (0, 0, 0, 1, 0, 0)$	
5	$(c, a, b)$	$\psi_5 = (0, 0, 0, 0, 1, 0)$	
6	$(c, b, a)$	$\psi_6 = (0, 0, 0, 0, 0, 1)$	

The synthesised CPOG (using approach from Section 4.1) is shown in Figure (a). Using logical optimisation it is possible to simplify it into slightly smaller CPOG shown in Figure (b).



Figure 11: CPOGs for 3-wire one hot phase encoder

The gate-level implementation of 3-wire one hot phase encoder specified with the obtained optimal CPOG is shown in Figure 12.

## 7 Binary phase encoder

Binary encoding is traditionally used for data transmission. To send a binary encoded data symbol it is possible to convert it first into matrix form using *binary code to matrix converter* and then to send the result using matrix phase encoder. But to avoid unnecessary conversion we can synthesise

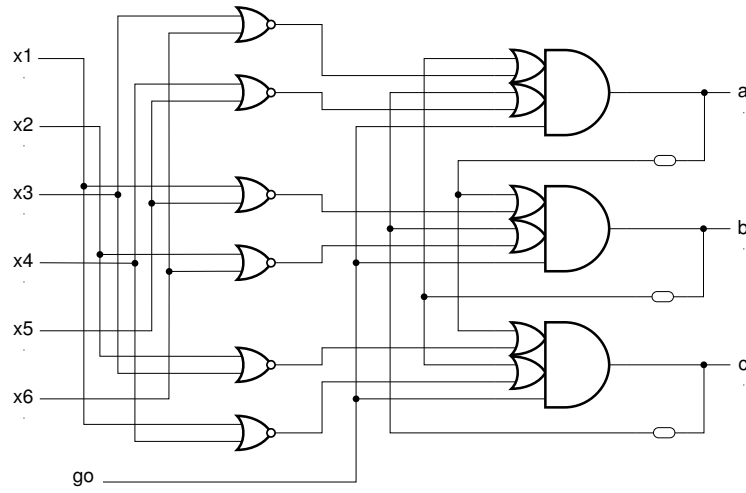


Figure 12: 3-wire one hot phase encoder circuit

customised *binary phase encoder* using the same principle as in the previous section for one hot encoding (cf. Figure 10).

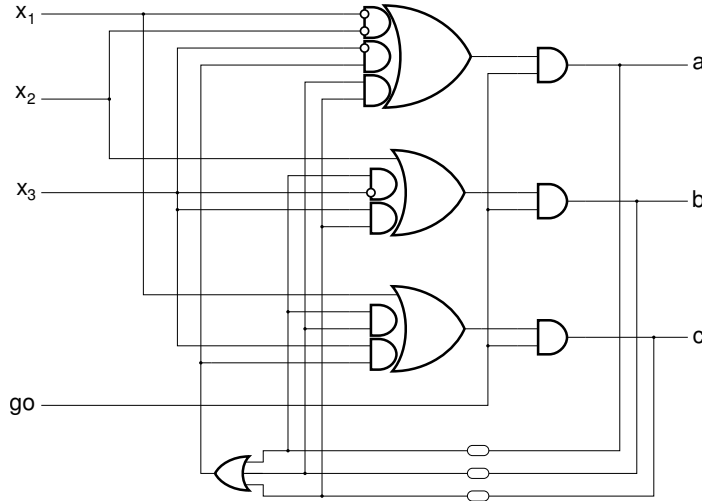


Figure 13: Binary phase encoder

The CPOG synthesis process is the same as for one hot phase encoding with the only exception that the binary encoding scheme is used (see Section 4.2). For the case of 3-wire binary phase encoder, the following set of Boolean equations for output signals  $T = \{a, b, c\}$  is eventually derived:

$$\begin{cases} a = ((\bar{x}_1 x_2 \bar{x}_3 + x_1 \bar{x}_2 x_3) \Rightarrow b^\Delta) ((\bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 \bar{x}_3) \Rightarrow c^\Delta) \\ b = ((\bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{x}_2 \bar{x}_3) \Rightarrow a^\Delta) ((\bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 x_3) \Rightarrow c^\Delta) \\ c = ((\bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3) \Rightarrow a^\Delta) ((\bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 x_2 x_3) \Rightarrow b^\Delta) \end{cases}$$

Taking into account binary assignment set  $\Psi = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1)\}$  the above equations can be simplified into

$$\begin{cases} a = \bar{x}_1 \bar{x}_2 + b^\Delta c^\Delta + \bar{x}_3 (b^\Delta + c^\Delta) \\ b = x_2 + \bar{x}_3 a^\Delta + x_3 c^\Delta \\ c = x_1 + a^\Delta b^\Delta + x_3 (a^\Delta + b^\Delta) \end{cases}$$

These resultant equations can now be mapped into gates to produce physical implementation of the

binary phase encoder as shown in Figure 13 (signal *go* is added for start/reset purposes).

## 8 Conclusions

The paper discusses the benefits of using multiple rail phase encoding protocol and compares it with the some other self-synchronous communication protocols. It also presents a CPOG model based approach for specification and synthesis of phase encoding multiple-rail controllers (phase detector, repeater, variety of phase encoders for different source encodings). The obtained solutions are more robust and scalable than in the previously published approaches.

## References

- [1] William J. Bainbridge. *Asynchronous System-on-Chip Interconnect*. PhD thesis, University of Manchester, 2000.
- [2] G. Birkhoff. *Lattice Theory*. Third Edition, American Mathematical Society, Providence, RI, 1967.
- [3] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. *Logic synthesis of asynchronous controllers and interfaces*. Advanced Microelectronics. Springer-Verlag, 2002.
- [4] Crescenzo D'Alessandro, Andrey Mokhov, Alex Bystrov, and Alex Yakovlev. Delay/Phase Regeneration Circuits. In *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2007.
- [5] Crescenzo D'Alessandro, Delong Shang, Alexandre V. Bystrov, and Alexandre Yakovlev. PSK signalling on NoC buses. In *PATMOS*, pages 286–296. Springer, 2005.
- [6] Crescenzo D'Alessandro, Delong Shang, Alexandre V. Bystrov, Alexandre Yakovlev, and Oleg V. Maevsky. Multiple-rail phase-encoding for NoC. In *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 107–116, 2006.
- [7] William J. Dally and John W. Poulton. *Digital systems engineering*. Cambridge University Press, New York, NY, USA, 1998.
- [8] Victor I. Varshavsky (Editor). *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, 1990.
- [9] Andrey Mokhov and Alex Yakovlev. Conditional Partial Order Graphs and Dynamically Reconfigurable Control Synthesis. In *Proceedings of Design, Automation and Test in Europe (DATE) Conference*, 2008.
- [10] Steven Nowick. *Automatic Synthesis of Burst-Mode Asynchronous Controllers*. PhD thesis, Stanford University, 1993.
- [11] Jens Sparsø and Steve Furber. *Principles of Asynchronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, 2001.
- [12] Tom Verhoeff. Delay-insensitive codes - an overview. *Distributed Computing*, 3(1):1–8, 1988.