Synthesis of multiple rail phase encoding circuits

Andrey Mokhov, Crescenzo D'Alessandro, Alex Yakovlev
Microelectronics System Design Group, School of EECE, Newcastle University, UK

***Abstract*—Multiple rail phase encoding communication protocol has several unexploited advantages over traditional encodings. The main impediment to its use is the absence of practical and scalable implementations of controllers for phase encoded data transmission.**

The paper shows that phase encoding controllers belong to a wide class of circuits which convert combinatorial codes to partial orders of events and vice versa. The conventional methods of control logic synthesis are not directly applicable to this class due to the combinatorial explosion of the controllers specification. The Conditional Partial Order Graph model introduced recently is inherently suitable for specification and synthesis of these controllers as demonstrated in this work.

The main focus of the paper is generation of robust and scalable area-efficient circuits for multiple rail phase encoders, decoders and repeaters. However, the proposed methodology can be applied to the larger class of controllers operating in the same code-to-sequence mode, e.g. CPU controllers, NoC routers etc.

I. 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 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 controllers (in terms of logic per number of wires in the channel) is not clearly described.

While conventional control logic specification and synthesis methods based on Petri Nets/Signal Transition Graphs [3], [16] or on Burst-Mode Finite State Machines [13] have certain advantages, they cannot be directly applied to the problem of phase encoders circuitry synthesis as shown in [11]. In particular, the size of the specification of *matrix phase encoder* (see Section V-B) 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* (*CPOGs*) which was recently introduced in [11], providing efficient gate-level implementations for the circuits. The detailed formal definitions of the CPOG algebraic structures and proofs of the theorems used in this paper can be found in [10].

The paper is organised as follows. Section II provides a short overview of the phase encoding protocol. Sections III and IV introduce Conditional Partial Order Graph model and the general synthesis method of CPOGs. The application of the method to synthesis of the phase encoding controllers is presented in Sections V through VII. An example of the *speed-independent* [12] controller synthesis within the presented methodology is studied in Section VIII. It is followed with the benchmarks discussion and conclusions in Section IX.

II. Phase encoding essentials

The *phase encoding* protocol was introduced by D'Alessandro *et al* in [5]. The initial idea was to encode an information bit into the phase difference between two switching signals. The idea was further extended into the *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. Fig. 1 shows an example of a data packet transmission over a 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. Note also that phase encoding 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 – no wire is dedicated for the clock signal.

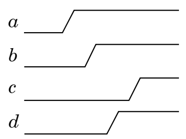


Figure 1. Data symbol in multiple rail phase encoding channel

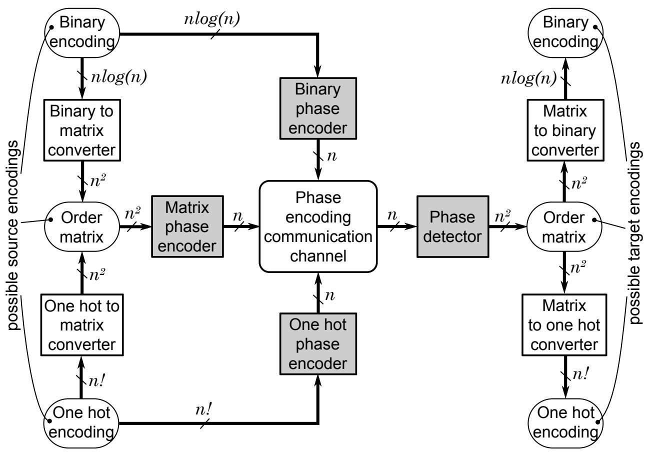


Figure 2. Phase encoding communication circuitry (numbers of wires are shown beside communication channels; implementation of functional units that are drawn tinted is covered in the paper; rounded rectangles represent different data encodings)

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

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

To exploit these attractive asymptotic characteristics of the protocol it is necessary to have a scalable approach for the design of controllers for large numbers of wires, however, the 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 set of techniques for generating circuits for multiple rail phase encoding senders/receivers/repeaters.

Fig. 2 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 the tinted boxes. Note that all of the target controllers convert data between two conceptually different information domains. The first one corresponds to the combinatorial data encoding, e.g. binary or m-of-n encoded data symbols. The second domain is comprised of sequences of events ordered in time, and these orders (in general, *partial orders* [9]) correspond to data symbols in the same way as different STGs correspond to different signal transition scenarios. Conditional Partial Order Graphs [11] are capable of specifying such controllers without the explicit representation of all the contained behavioural scenarios and thus avoiding the combinatorial explosion of the specification.

The methods presented in this paper can be applied whenever a controller has to react to different control codes by initiating different event sequences. The most natural

Table I
PHASE ENCODING PROTOCOL CHARACTERISTICS

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)	$n!$	$\Theta(n \log_2 n)$	$\Theta(\log_2 n)$	$\Theta(\frac{1}{\log_2 n})$

examples include: CPU instruction decoders, which receive an instruction code and activate a set of data path operational units (adders, multipliers etc.) in a proper partial order; and NoC routers, which receive a routing code (source/destination address) and perform a set of routing procedures in the requested sequence.

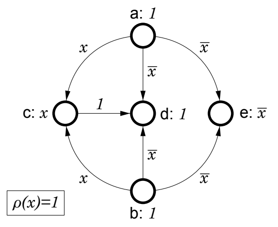
III. CONDITIONAL PARTIAL ORDER GRAPHS

Conditional Partial Order Graph (further called *CPOG* or *graph* for short) is a quintuple $H(V,\ E,\ X,\ \rho,\ \phi)$ where:

- V is a set of *vertices* corresponding to the events in the modelled system. V defines the system's *event domain.*
- $E \subseteq V \times V$ is a set of ordered pairs of vertices, or *arcs*, representing the dependencies between the events.
- *Control vector* X is a finite set of Boolean variables (also called *control variables* or *signals*).
- $\rho \in \mathcal{F}(X)$ is a *restriction function,* where $\mathcal{F}(X)$ is the set of all Boolean functions over the control variables in X. ρ defines the *operational domain* of the graph: control vector X is allowed to have only those values $(x_1,\ x_2,\ ...,\ x_{|X|}) \in \{0,\ 1\}^{|X|}$ which satisfy the restriction function: $\rho(x_1,\ x_2,\ ...,\ x_{|X|}) = 1$. Graph is called *singular* iff its operational domain is empty i.e. function ρ is a contradiction: $\rho = 0$.
- Function $\phi\ :\ (V \cup E) \rightarrow \mathcal{F}(X)$ assigns a Boolean *condition* $\phi(z) \in \mathcal{F}(X)$ to every vertex and arc $z \in V \cup E$ in the graph. Let us also define $\phi(z) = 0$ for $z \notin V \cup E$ in order to simplify some of the further computations.

Conditional Partial Order Graphs are represented graphically by drawing a labelled circle ○ for every vertex $v \in V$, and drawing a labelled arrow $\longrightarrow$ for every arc $e \in E$. Label of a vertex $v \in V$ consists of the vertex name, semicolon and the vertex condition $\phi(v)$, while every arc $e \in E$ is labelled with the corresponding arc condition $\phi(e)$. The restriction function ρ is depicted in a box next to the graph; control vector X can therefore be observed as the parameters of ρ.

Fig. 3(a) shows an example of a graph containing $|V| = 5$ vertices and $|E| = 7$ arcs. The restriction function is $\rho(x) = 1$, and the control vector consists of a single variable $X = \{x\}$. Vertices $\{a,\ b,\ d\}$ have constant $\phi = 1$ conditions and are called *unconditional*, while vertices $\{c,\ e\}$ are *conditional* and have conditions $\phi(c) = x$ and $\phi(e) = \overline{x}$ respectively. Arcs also fall into two classes: *unconditional* (arc $(c,\ d)$) and *conditional*


(a) Full notation

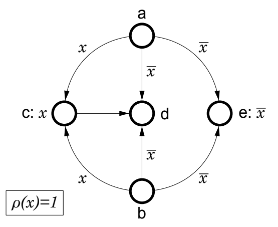

(b) Simplified notation

Figure 3. Graphical representation of conditional partial order graphs

(all the rest). As CPOGs tend to have many unconditional vertices and arcs it is reasonable to use a simplified notation in which conditions equal to 1 are not depicted in the graph. This is demonstrated in Fig. 3(b).

The purpose of the vertex and arc conditions is to 'switch off' some of the vertices and arcs in the graph according to the control variables. This makes CPOGs capable of containing multiple *projections* (cf. definition in Subsection III-A), as shown in Fig. 4. The leftmost projection is obtained by keeping in the graph only those vertices and arcs whose conditions evaluate to Boolean 1 after substitution of the control variable x with Boolean 1. Hence, vertex e disappears (denoted as a dashed circle ◌), because its condition evaluates to 0: $\phi(e) = \overline{x} = \overline{1} = 0$. Arcs $\{(a,\ d),\ (a,\ e),\ (b,\ d),\ (b,\ e)\}$ disappear for the same reason (denoted as dashed arrows ⇢). The rightmost projection is obtained in the same way with the only difference that the control variable x is set to 0. Note also that although the condition of arc $(c,\ d)$ evaluates to 1 (in fact it is constant 1) the arc is still excluded from the resultant graph because one of the vertices it connects (vertex c) is excluded and obviously an arc cannot appear in a graph without one of its vertices. The restriction function of the graph does not affect anything in this particular case because it evaluates to 1 for both possible control vector assignments ($x = 1$ and $x = 0$): $\rho(0) = \rho(1) = 1$.

Each of the obtained projections can be treated as a directed graph which specifies a *partial order* [9] of events in a particular behavioural scenario of the modelled system (see [10], [11] for more examples). Potentially, a CPOG $H(V,\ E,\ X,\ \rho,\ \phi)$ can specify an exponential number of different scenarios comprised of events in V according to one of $2^{|X|}$ different possible assignments of control variables X.

A. Projections

A *projection* of graph $H(V,\ E,\ X,\ \rho,\ \phi)$ under constraint $x = \alpha$ (where $x \in X$, $\alpha \in \{0,\ 1\}$) is denoted as $H|_{x=\alpha}$ and is equal to graph $H'(V,\ E,\ X \setminus \{x\},\ \rho|_{x=\alpha},\ \phi|_{x=\alpha})$ where notations $\rho|_{x=\alpha}$ and $\phi|_{x=\alpha}$ mean that variable x is substituted with constant Boolean value α in ρ and all functions $\phi(z),\ z \in V \cup E$, hence $\rho|_{x=\alpha}$ and $\phi|_{x=\alpha}(z)$ belong to $\mathcal{F}(X\setminus\{x\})$.

Projection is a *commutative operation* i.e. $(H|_{x=\alpha})|_{y=\beta} = (H|_{y=\beta})|_{x=\alpha}$ so the following short notation can be used without any ambiguity: $H|_{x=\alpha,\ y=\beta}$.

A *complete projection* of graph H is such a projection that

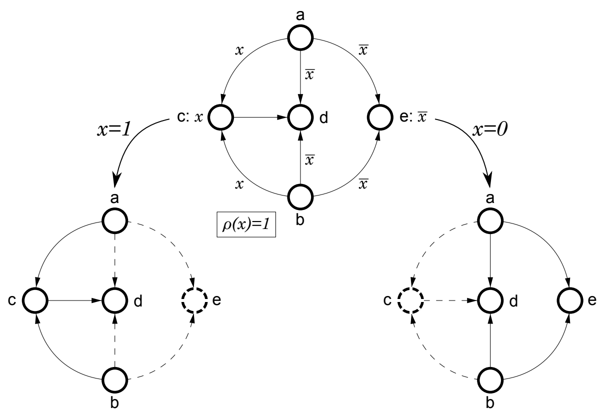


Figure 4. Multiple projections contained within a single CPOG

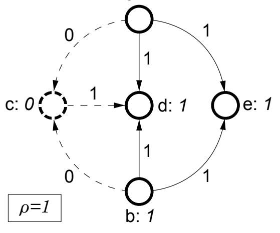
all the variables in X are constrained to constants. It is denoted as $H|_\psi$ where $\psi:\ X \to \{0,\ 1\}$ is an *assignment function* (or *encoding*) that assigns a Boolean value to every variable in X. Complete projection is a graph whose restriction function and vertex/arc conditions are only Boolean constants $\rho|_\psi$ and $\phi|_\psi$ (either 0 or 1), and control signals set is empty: $X = \emptyset$.

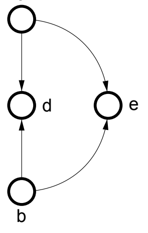
A (complete) projection is called *singular* iff the resultant graph is singular.

Given a non-singular complete projection $H(V,\ E,\ \emptyset,\ 1,\ \phi)$ operation $G = \mathbf{dg}\ H$ generates *directed graph* [9] $G(V_G,\ E_G)$ such that

$$\begin{cases} V_G = \{v \in V,\ \phi(v) = 1\} \\ E_G = \{e = (a,\ b) \in E,\ \phi(a)\phi(b)\phi(e) = 1\} \end{cases}$$

In other words G includes only those vertices and arcs whose conditions in H are constant 1. Note, that exclusion of a vertex also leads to exclusion of all its adjacent arcs. The inverse operation is $H' = \mathbf{dg}^{-1}\ G$. Here $H'(V,\ E,\ X,\ \rho,\ \phi)$ is defined in terms of $G(V_G,\ E_G)$ as follows: $V = V_G$, $E = E_G$, $X = \emptyset$, $\rho = 1$ and $\phi(z) = 1,\ z \in V \cup E$. Note, that $\mathbf{dg}^{-1}$ is a *right inverse* operation i.e. $\mathbf{dg}(\mathbf{dg}^{-1}\ G) = G$ but $\mathbf{dg}^{-1}(\mathbf{dg}\ H)$ is not necessarily equal to H. This is demonstrated in Fig. 5 which shows an example of complete projection H (Fig. 5(a)), its conversion into directed graph $G = \mathbf{dg}\ H$ (Fig. 5(b)), and complete projection $H' = \mathbf{dg}^{-1}\ G$ (Fig. 5(c)). One can see, that $H \neq H'$ but both $\mathbf{dg}\ H$ and $\mathbf{dg}\ H'$ are the same and equal to G.


(a) Complete projection H


(b) $G = \mathbf{dg}\ H$

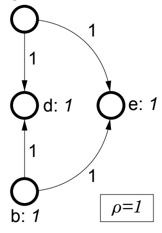
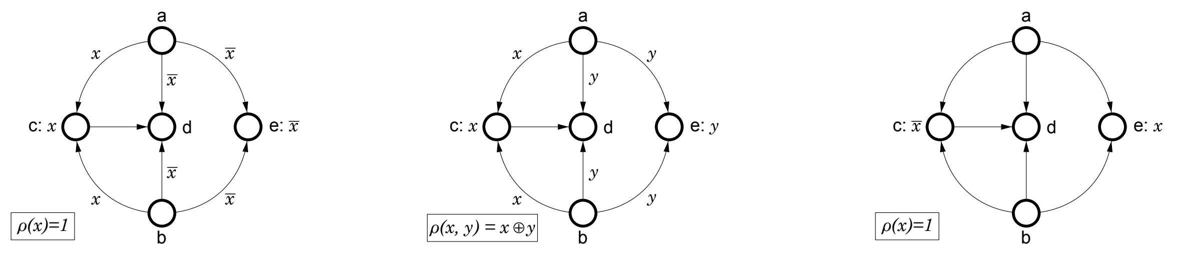

(c) $H' = \mathbf{dg}^{-1}\ G$

Figure 5. Operation $\mathbf{dg}$ and its inverse



(a) Example graph (8 literals) (b) Graph with two control variables (10 literals) (c) No redundant conditional arcs (2 literals)

Figure 6. Three equivalent graphs

A complete projection $H|_\psi$ is called *valid* iff it is not singular and its corresponding directed graph $\mathbf{dg}\ H|_\psi$ is acyclic (DAG). Hence, an assignment function ψ is called *valid* w.r.t. graph H iff complete projection $H|_\psi$ is valid.

Graph $H(V,\ E,\ X,\ \rho,\ \phi)$ is *well-formed* iff its every non-singular complete projection $H|_\psi$ is valid. In other words, every complete projection $H|_\psi$ which is allowed by the restriction function ($\rho|_\psi = 1$) must produce DAG $\mathbf{dg}\ H|_\psi$. Let the set of all well-formed CPOGs be denoted as $\mathcal{W}$.

Given a DAG $G(V_G,\ E_G)$ operation $P = \mathbf{po}\ G$ generates a *strict partial order* [9] $P(V_G,\ \prec)$ such that $a \prec b$ iff G contains an oriented path between vertices $a,\ b \in V_G$. The (right) inverse operation is $\mathbf{po}^{-1}$: $G = \mathbf{po}^{-1}\ P$.

Using operations $\mathbf{dg}$ and $\mathbf{po}$ it is possible to write equations operating over CPOGs, DAGs and partial orders, e.g. the partial order defined by the rightmost projection of graph H in Fig. 4 can be denoted as $\mathbf{po}(\mathbf{dg}\ H|_{x=0})$:

$$\mathbf{po}(\mathbf{dg}\ H|_{x=0}) = \begin{cases} V_G = \{a,\ b,\ d,\ e\} \\ \prec = \{a \prec d,\ b \prec d,\ a \prec e,\ b \prec e\} \end{cases}$$

The set of all partial orders defined by a well-formed graph H is denoted as $\mathcal{P}(H)$ and is formally defined as:

$$\mathcal{P}(H) \overset{\text{df}}{=} \{P = \mathbf{po}(\mathbf{dg}\ H|_\psi),\ \rho|_\psi = 1\}$$

The set of all partial orders defined by graph in Fig. 4 is

$$\mathcal{P}(H) = \{P_1,\ P_2\} = \{\mathbf{po}(\mathbf{dg}\ H|_{x=0}),\ \mathbf{po}(\mathbf{dg}\ H|_{x=1})\}$$

This definition provides the background for a natural *equivalence relation* [9] $\sim$ over the set of well-formed graphs $\mathcal{W}$. Graphs $H_1 \in \mathcal{W}$ and $H_2 \in \mathcal{W}$ are called *equivalent* (denoted as $H_1 \sim H_2$) iff they define the same set of partial orders:

$$(H_1 \sim H_2) \overset{\text{df}}{=} \mathcal{P}(H_1) = \mathcal{P}(H_2)$$

Fig. 6 shows three equivalent graphs $H_a \sim H_b \sim H_c$. Graph H_a in Fig. 6(a) is taken from the previous example. Fig. 6(b) shows graph H_b with the modified control set. It contains two control variables $X = \{x,\ y\}$ which are restricted in the *one hot encoding* manner: only encodings (0, 1) and (1, 0) are allowed with the restriction function $\rho(x,\ y) = x \oplus y$. Graph H_c in Fig. 6(c) does not contain any arc conditions (which are in fact redundant) and it also has inverted encodings compared to H_a. In spite of the seeming difference between the three graphs, they are equivalent as they define the same set of two partial orders $\mathcal{P}(H_a) = \mathcal{P}(H_b) = \mathcal{P}(H_c) = \{P_1,\ P_2\}$:

$$P_1 = \mathbf{po}(\mathbf{dg}\ H_a|_{x=0}) = \mathbf{po}(\mathbf{dg}\ H_b|_{\substack{x=0\\y=1}}) = \mathbf{po}(\mathbf{dg}\ H_c|_{x=1})$$

$$P_2 = \mathbf{po}(\mathbf{dg}\ H_a|_{x=1}) = \mathbf{po}(\mathbf{dg}\ H_b|_{\substack{x=1\\y=0}}) = \mathbf{po}(\mathbf{dg}\ H_c|_{x=0})$$

It is useful to introduce a measure of complexity of graphs in order to be able to compare them within the same equivalence class. For instance, graph H_c in Fig. 6 has the simpler description in comparison with graphs H_a and H_b and is preferred in most cases.

The *complexity* (or *size*) $C(H)$ of graph $H(V,\ E,\ X,\ \rho,\ \phi)$ is measured in the number of literals contained in the restriction function ρ and conditions $\phi(z),\ z \in V \cup E$:

$$C(H) \overset{\text{df}}{=} C(\rho) + \sum_{v \in V} C(\phi(v)) + \sum_{e \in E} C(\phi(e))$$

where $C(f),\ f \in \mathcal{F}(X)$ denotes the literal count [17] of a Boolean function f. Looking at graphs in Fig. 6 one can see that $C(H_a) = 0 + 2 + 6 = 8$, $C(H_b) = 2 + 2 + 6 = 10$, and $C(H_c) = 0 + 2 + 0 = 2$. So, graph H_c can be called *optimal* in this context. Methods for graphs size optimisation are addressed in [11].

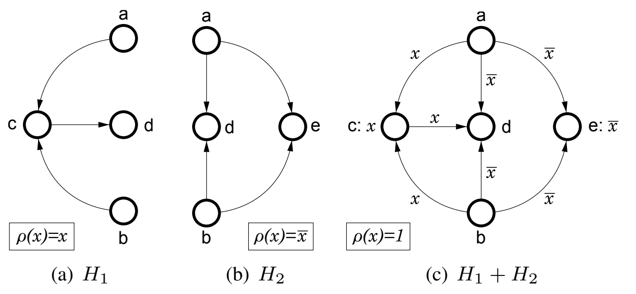
Two well-formed graphs H_1 and H_2 are said to be *in conflict* w.r.t. their restriction functions ρ_1 and ρ_2 iff $\rho_1\rho_2 \neq 0$. A conflict implies the existence of an encoding ψ such that both the restriction functions are satisfied: $\rho_1|_\psi = \rho_2|_\psi = 1$. This leads to an ambiguity in some cases (e.g. in case of graph addition introduced in Subsection III-B), when two graphs describe different behaviour under the same encoding ψ.

B. Addition

The result of *addition* of graphs $H_1(V_1,\ E_1,\ X_1,\ \rho_1,\ \phi_1)$ and $H_2(V_2,\ E_2,\ X_2,\ \rho_2,\ \phi_2)$ is graph $H(V_1 \cup V_2,\ E_1 \cup E_2,\ X_1 \cup X_2,\ \rho_1 + \rho_2,\ \phi)$ where the vertex/arc conditions ϕ are defined as

$$\forall z \in V_1 \cup V_2 \cup E_1 \cup E_2,\ \phi(z) \overset{\text{df}}{=} \rho_1\overline{\rho_2}\phi_1(z) + \overline{\rho_1}\rho_2\phi_2(z)$$

Addition is denoted using the standard notation $H = H_1 + H_2$.



(a) H_1 (b) H_2 (c) H_1+H_2

Figure 7. Graph addition

Theorem 1: Pair $(\mathcal{W},\ +)$ is a commutative semigroup [9], i.e. set of well-formed graphs $\mathcal{W}$ is closed under addition $+$, which is an associative and commutative operation [10].

Corollary 1: When adding more than two graphs the redundant brackets can be omitted without any ambiguity: $H_1+H_2+H_3$.

In the same way as graphs H_1 and H_2 are considered to be specifications of certain behavioural scenarios over event domains V_1 and V_2, graph H_1+H_2 is considered to be specification of the scenarios from both the graphs over the joint event domain $V=V_1\cup V_2$. This is formally stated in the following theorem.

Theorem 2: If H_1 and H_2 are well-formed graphs that are not in conflict then $\mathcal{P}(H_1+H_2)=\mathcal{P}(H_1)\cup\mathcal{P}(H_2)$ [10], i.e. graph H_1+H_2 contains partial orders from both H_1 and H_2, preserving their encodings.

Consider an example of addition in Fig. 7. Each of graphs H_1 and H_2 specifies a single scenario. The graphs are not in conflict ($\rho_1\rho_2=x\overline{x}=0$), the result of their addition H_1+H_2 is shown in Fig. 7(c). It contains both of the scenarios (as was demonstrated in Fig. 4). Another example of graph addition is shown in Fig. 8.

C. Scalar multiplication

Graph $H(V,\ E,\ X,\ \rho,\ \phi)$ can be *multiplied* by a Boolean function $f\in\mathcal{F}(Y)$ (which in our context can be called *scalar*). The resultant graph is $H'(V,\ E,\ X\cup Y,\ f\rho,\ \phi)$. The standard notation will be used for scalar multiplication: $H'=fH$.

Theorem 3: For every Boolean function f and well-formed graph H, graph $H'=fH$ is also well-formed and $\mathcal{P}(H')\subseteq\mathcal{P}(H)$ [10].

A *linear combination* of $n\geq 1$ graphs $H_1,\ H_2,\ ...,\ H_n$ and scalars $f_1,\ f_2,\ ...,\ f_n$ is

$$\sum_{1\leq k\leq n} f_kH_k=f_1H_1+f_2H_2+...+f_nH_n$$

Note that any linear combination of well-formed graphs is also well-formed due to the closure of addition and scalar multiplication operations over $\mathcal{W}$ (Theorems 1 and 3).

Fig. 8 shows linear combination $H=f_1H_1+f_2H_2$ of graphs $(H_1,\ H_2)$ w.r.t. scalars $(f_1,\ f_2)$.

IV. 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. [11] showed how 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 $\{P_1,\ P_2,\ ...,\ P_n\}$ be the set of n given partial orders. The objective is to synthesise CPOG $H(V,\ E,\ X,\ \rho,\ \phi)$ such that

$$\mathcal{P}(H)=\{P_1,\ P_2,\ ...,\ P_n\} \tag{1}$$

The idea behind the synthesis approach presented in [11] is to represent H as the following linear combination of complete projections $H_k=\mathbf{dg}^{-1}(\mathbf{po^{-1}}\ P_k)$:

$$H=f_1H_1+f_2H_2+...+f_nH_n=\sum_{1\leq k\leq n} f_kH_k \tag{2}$$

where *encoding functions* $f_k\in\mathcal{F}(X)$ are *orthogonal* i.e. $f_jf_k=0,\ 1\leq j<k\leq n$ and are not contradictions: $f_k\neq 0,\ 1\leq k\leq n$. According to Theorems 2 and 3 this guarantees that $\mathcal{P}(H)=\mathcal{P}(H_1)\cup\mathcal{P}(H_2)\cup...\cup\mathcal{P}(H_n)=\{P_1\}\cup\{P_2\}\cup...\cup\{P_n\}$. Thus, (2) satisfies the synthesis requirement (1).

Control signals X and functions f_k can be selected 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.

A. One hot encoding scheme

In this scheme n control signals $X=\{x_1,\ x_2,\ ...,\ x_n\}$ are used to select a particular scenario. Functions f_k are set to

$$f_k=x_k\bigwedge_{\substack{1\leq j\leq n\\ j\neq k}}\overline{x_j}$$

establishing one hot encodings of the scenarios: P_1 is encoded as $(x_1,\ x_2,\ x_3,\ ...)=(1,\ 0,\ 0,\ ...)$, P_2 — as $(x_1,\ x_2,\ x_3,\ ...)=(0,\ 1,\ 0,\ ...)$ etc.

Fig. 8 shows an example of synthesis of a CPOG containing partial orders $P_1=\{a\prec b\}$ and $P_2=\{b\prec a\}$. The control signals set is $\{x_1,\ x_2\}$ and the encoding functions

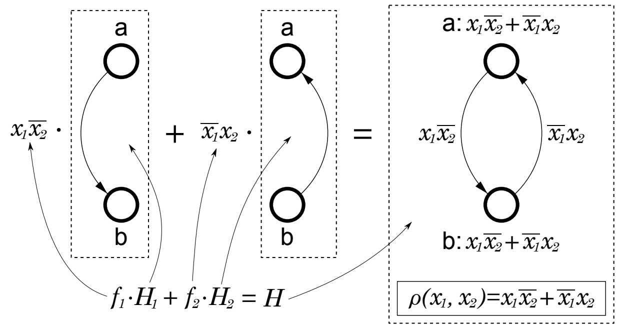
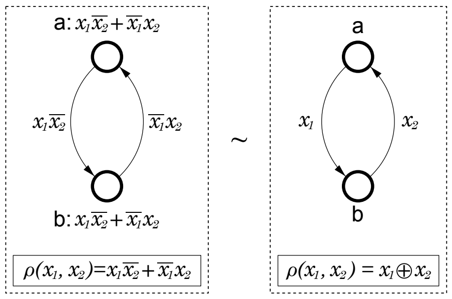

$x_1\overline{x_2}\cdot$
a
b
+
$\overline{x_1}x_2\cdot$
a
b
=
a: $x_1\overline{x_2}+\overline{x_1}x_2$
$x_1\overline{x_2}$
$\overline{x_1}x_2$
b: $x_1\overline{x_2}+\overline{x_1}x_2$
$f_1\cdot H_1+f_2\cdot H_2=H$
$\rho(x_1,\ x_2)=x_1\overline{x_2}+\overline{x_1}x_2$

Figure 8. One hot CPOG synthesis

are $f_1 = x_1\overline{x_2}$ and $f_2 = \overline{x_1}x_2$. The result $H = f_1H_2 + f_2H_2$ contains both partial orders as projections $H|_{x_1=1,\ x_2=0}$ and $H|_{x_1=0,\ x_2=1}$. It is possible to optimise it reducing the literal count from 16 to 4 using the methods presented in [11]:



a: $x_1\overline{x_2}+\overline{x_1}x_2$
$x_1\overline{x_2}$
$\overline{x_1}x_2$
b: $x_1\overline{x_2}+\overline{x_1}x_2$
$\rho(x_1, x_2)=x_1\overline{x_2}+\overline{x_1}x_2$
~
a
x_1
x_2
b
$\rho(x_1, x_2) = x_1\oplus x_2$

(3)

One hot scheme provides a simple and intuitive way of encoding partial orders but it is inefficient because of the large size of control signals set: $|X| = n$. It is not practical for synthesis of CPOGs containing large number of projections. In this work it was used for synthesis of controllers with up to $5! = 120$ different behavioural scenarios (one hot phase encoding senders, Section VI).

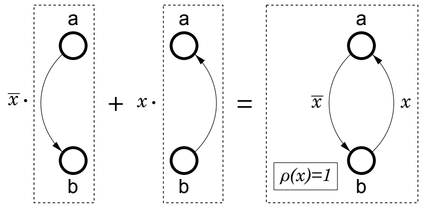
B. Binary encoding scheme

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

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

For example, if $n = 3$ we get $f_1 = (x_1 \Leftrightarrow 0)(x_2 \Leftrightarrow 0) = \overline{x_1}\,\overline{x_2}$, $f_2 = (x_1 \Leftrightarrow 1)(x_2 \Leftrightarrow 0) = x_1\overline{x_2}$ and $f_3 = (x_1 \Leftrightarrow 0)(x_2 \Leftrightarrow 1) = \overline{x_1}x_2$ resulting in a natural binary encodings of the three partial orders: $\psi_1 = (0,\ 0)$, $\psi_2 = (1,\ 0)$ and $\psi_3 = (0,\ 1)$.

Application of the binary encoding scheme to synthesis of a CPOG containing partial orders $P_1 = \{a \prec b\}$ and $P_2 = \{b \prec a\}$ leads to a very compact (only 2 literals) specification:



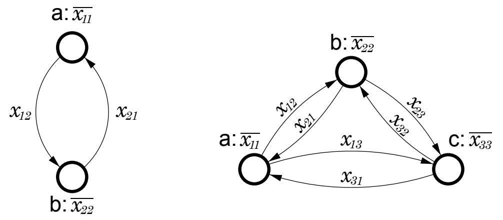
Observe the difference between this result and the optimised version of the one hot solution (3). As one can see the selected encoding scheme does not affect the structure of the optimised CPOG. However, it affects the complexity of the functions and the size of the physical controller implementation.

C. Matrix encoding scheme

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

$$\psi(x_{jk})_{j\neq k} = \begin{cases} 1 & if\ (e_j \in V') \wedge (e_k \in V') \wedge (e_j \prec e_k) \\ 0 & otherwise \end{cases}$$
$$\psi(x_{kk}) = \begin{cases} 1 & if\ (e_k \notin V') \\ 0 & otherwise \end{cases} \quad (4)$$

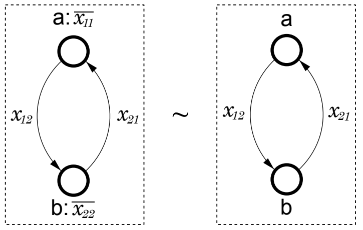
Instead of direct application of this encoding scheme to (2) we can use a generic solution with its subsequent optimisation taking into account the given scenarios. Generic solutions for systems with two and three events are given below:



For example, using (4) to encode partial orders $P_1 = \{a \prec b\}$ and $P_2 = \{b \prec a\}$ gives us these encoding matrices:

$$\psi_{1,2} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} : \quad \psi_1 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix},\ \psi_2 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

Diagonal elements x_{kk} are constant zeros, and in this case the generic matrix graph can be reduced to one hot solution (up to control variables renaming – see (3)):



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 intuitive 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 efficiency of matrix encoding scheme allowed us to specify and synthesise phase encoding controllers for up to 10 wires having $10! = 3628800$ different behavioural scenarios.

V. Phase encoding repeater

The first multiple rail phase encoding circuit that we are going to synthesise is a *phase encoding repeater* [4] – a circuit

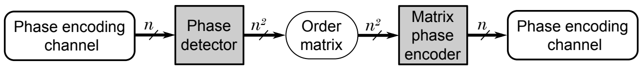


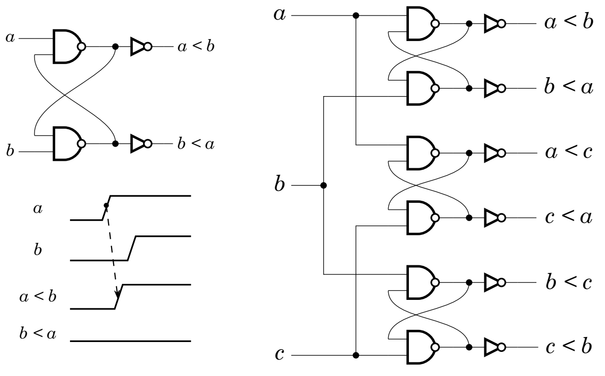
Figure 9. Phase encoding repeater circuitry

able to regenerate the deteriorating phase difference between signals in the phase encoding communication channel.

A 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* generating a series of transitions in the order they were received) as shown in Fig. 9. 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].

A. Phase detector

Phase detector for an 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 Fig. 10(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 Fig. 10(b) for an example of 3-wire phase detector).



(a) Mutual exclusion (b) 3-wire phase detector

Figure 10. Phase detection

The result of phase detection can be seen as a control matrix (4) with zero diagonal elements. Therefore the subsequent phase encoder should be synthesised using the matrix encoding scheme to avoid additional encoding conversion circuitry.

B. Matrix phase encoder

Given control matrix $X = \{x_{jk},\ j = 1...n,\ k = 1...n,\ j \neq k\}$ 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 value assignments $\psi_k :\ X \to \{0,\ 1\},\ k = 1...n!$, each of them corresponding to a particular scenario. CPOG $H(V,\ E,\ X,\ \rho,\ \phi)$ containing all of them as its projections has the following generic description:

$$V = \{e_j,\ j = 1...n\}$$

$$E = \{(e_j, e_k),\ j = 1...n,\ k = 1...n,\ j \neq k\}$$

$$X = \{x_{jk},\ j = 1...n,\ k = 1...n,\ j \neq k\}$$

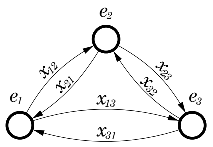
$$\rho = \bigwedge_{1\le j<k\le n} x_{jk} \oplus x_{kj} \bigwedge_{\substack{1\le i,j,k\le n \\ i\neq j,\ i\neq k,\ j\neq k}} x_{ij}x_{jk} \Rightarrow x_{ik} \qquad (5)$$

$$\phi(e_j) = 1,\ j = 1...n$$

$$\phi((e_j, e_k)) = x_{jk},\ j = 1...n,\ k = 1...n,\ j \neq k$$

The complexity $C(H)$ of this graph is dominated by the restriction function ρ which has a cubic size w.r.t. the number of wires n: $C(\rho) = \Theta(n^3)$. It restricts the operational domain of the graph to $n!$ assignments corresponding to $n!$ different *total orders* [9]. Total order is 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 (this is equivalent to the condition of all the mutexes being resolved).

Example of a CPOG specification of 3-wire matrix phase encoder based on (5) is shown below:



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

$$t_k = \phi(e_k) \cdot \bigwedge_{\substack{1\le j\le n \\ j\neq k}} (\phi(e_j) \cdot \phi((e_j, e_k)) \Rightarrow t_j) \qquad (6)$$

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

This generic equation can be simplified taking into account the particular CPOG specification (5):

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

Another optimisation opportunity is to exploit the fact that the control matrix X specifies a total order. In our case it means that $\overline{x_{jk}} = x_{kj}$:

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

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

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

where t_j^{Δ} represents signal t_j delayed for Δ 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:

$$t_k = go \cdot \bigwedge_{\substack{1 \le j \le n \\ j \ne k}} (x_{kj} + t_j^{\Delta}) \qquad (7)$$

The gate-level implementation of the controller specified with equation (7) is shown in Fig. 11.

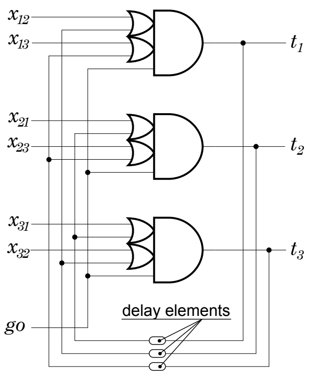


Figure 11. 3-wire matrix phase encoder

The implementation of phase encoding repeater consisting of phase detector and phase encoder is shown in Fig. 12. Signal go can be generated 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].

VI. 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 Fig. 13.

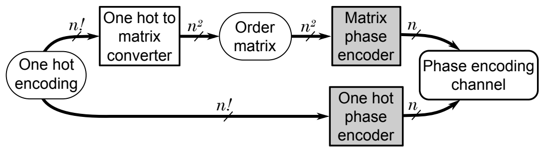


Figure 13. One hot phase encoder circuitry

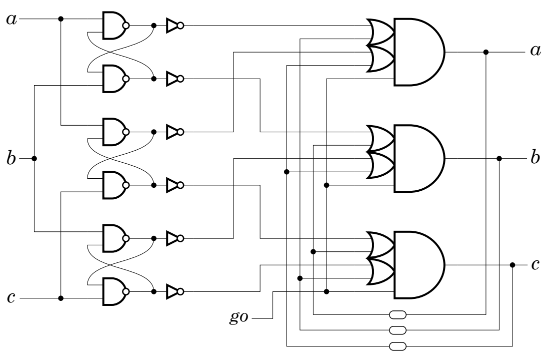
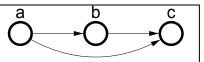
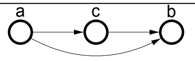
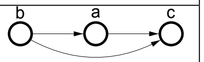
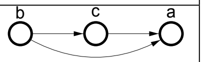
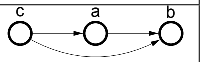
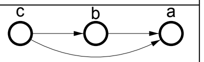
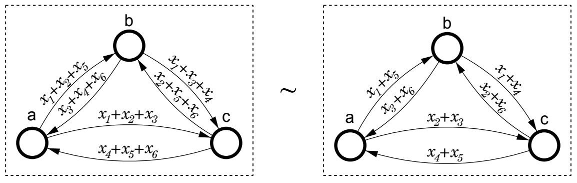


Figure 12. 3-wire phase encoding repeater

All the $n!$ different scenarios of n output wire transitions can be specified with $n!$ partial orders $\mathcal{P} = \{P_1,\ P_2,\ ...,\ P_{n!}\}$. It is possible to synthesise a CPOG containing all of them using one hot encoding scheme (Section IV-A). For example, there are 6 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 $T = \{a,\ b,\ c\}$ for the case of $n = 3$ wires:

#	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 is shown below (to the left); it is possible to simplify it into a slightly smaller CPOG using the *transitive conditions reduction* [11] (to the right):



The final gate-level implementation of the 3-wire one hot phase encoder specified with the obtained optimal CPOG is shown in Fig. 14.

VII. Binary phase encoder

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

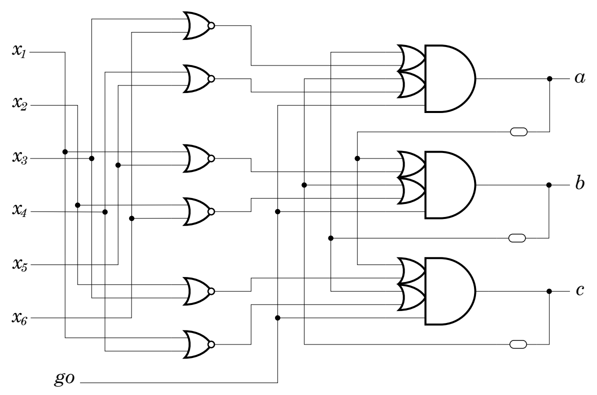


Figure 14. 3-wire one hot phase encoder

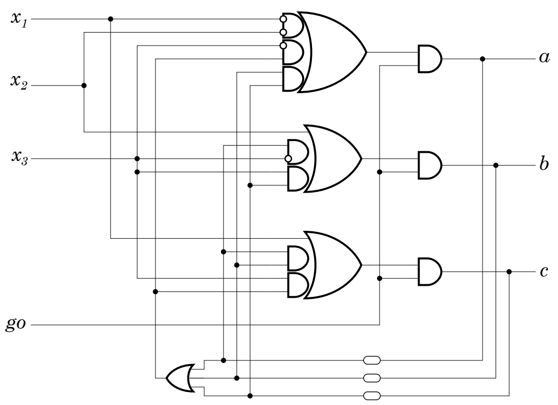


Figure 15. 3-wire binary phase encoder

binary phase encoder using the same principle as in the previous section for one hot encoding (cf. Fig. 13).

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 IV-B). 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:

$$\begin{cases} a = ((\overline{x_1}x_2\overline{x_3} + x_1\overline{x_2}x_3) \Rightarrow b^{\Delta})((\overline{x_1}x_2x_3 + x_1\overline{x_2}\,\overline{x_3}) \Rightarrow c^{\Delta}) \\ b = ((\overline{x_1}\,\overline{x_2}\,\overline{x_3} + x_1\overline{x_2}\,\overline{x_3}) \Rightarrow a^{\Delta})((\overline{x_1}\,\overline{x_2}x_3 + x_1\overline{x_2}x_3) \Rightarrow c^{\Delta}) \\ c = ((\overline{x_1}\,\overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}) \Rightarrow a^{\Delta})((\overline{x_1}\,\overline{x_2}\,\overline{x_3} + \overline{x_1}x_2x_3) \Rightarrow b^{\Delta}) \end{cases}$$

Taking into account the binary assignment set $\{000,\ 001,\ 010,\ 011,\ 100,\ 101\}$ the above equations can be simplified (e.g. by using ESPRESSO [15] logic minimisation tool) into

$$\begin{cases} a = \overline{x_1}\,\overline{x_2} + b^{\Delta}c^{\Delta} + \overline{x_3}(b^{\Delta} + c^{\Delta}) \\ b = x_2 + \overline{x_3}a^{\Delta} + x_3c^{\Delta} \\ c = x_1 + a^{\Delta}b^{\Delta} + x_3(a^{\Delta} + b^{\Delta}) \end{cases}$$

These resultant equations can now be mapped to gates to produce the physical implementation of the controller as shown in Fig. 15 (go is added for start/reset purposes).

VIII. Speed-independent synthesis

The presented method of synthesis produces a set of Boolean equations as a result. In general it is not always possible to implement a large function using a single complex gate: libraries are often limited to 2- and 3-input elementary logic gates due to the technological constraints. This brings us to one of the key problems of circuit synthesis — the *logic decomposition* [3] of a given complex gate into an equivalent network of library gates satisfying certain correctness requirements. These requirements may vary depending on the target class of the controller being synthesised. For example, the controllers presented in Sections VI - VII are not speed-independent and operate correctly only under the timing assumptions imposed on control signals X and request signal go allowing a simpler decomposition technique to be used. However, the presented CPOG based methodology can also be used for synthesis of speed-independent controllers, provided that special care is taken while mapping the resultant Boolean equations into the gate netlist (see [3] for a thorough analysis of this problem arising in a similar context of STG based logic synthesis).

The speed-independent synthesis can be demonstrated on 3-wire one hot phase encoder from Section VI. The controller interface should be changed in order to establish a proper speed-independent communication protocol between the controller and the environment. Signal go is not needed anymore (the start and reset functions are delegated to one hot control signals $x_1...x_6$). Instead a new signal $done$ should be introduced to prompt the environment that the controller has sent the phase encoded data and is ready for the next symbol. The delay elements should also be moved outside of the controller and become part of the environment. The implementation of the controller is shown in Fig. 16 (the controller is separated from the environment with a dotted line). The complex gates generating the output signals are decomposed into 2- and 3-input logic gates with a subsequent negative logic optimisation. The delayed output transitions are synchronised with a C-element to produce signal $done$. The circuit is formally verified for the compliance with the environment interface and the absence of hazards using WORKCRAFT [14] framework.

In the general case a speed-independent controller has to issue signal $done$ only after all the internal gates have switched and the communication with the environment has finished. Similarly, during the reset phase as soon as all the gates and external handshakes have been reset to the initial states the falling transition of signal $done$ enables the environment to initiate the next working cycle. So, the role of signal $done$ is equivalent to that of *completion detection* [8] signal in asynchronous data path, which informs the control path about the completion of computation within the combinational logic.

IX. Benchmarks and Conclusions

The paper presents a CPOG model based approach for specification and synthesis applied to phase encoding circuits for different number of wires and source encodings. Table II summarises the obtained results. The leftmost four columns specify

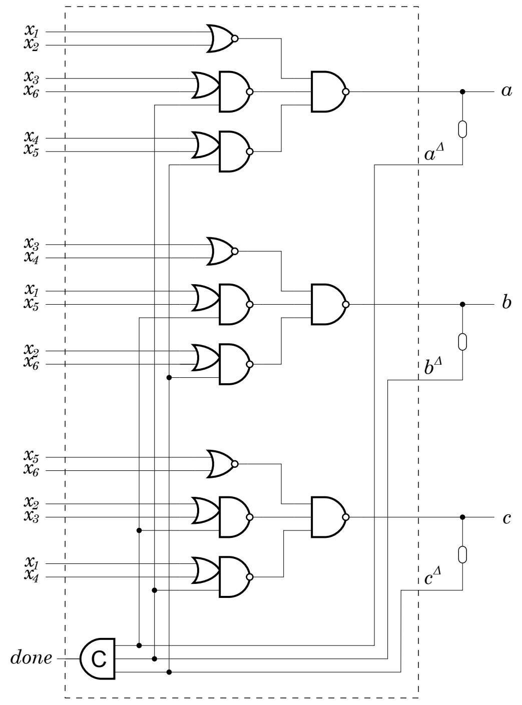


Figure 16. 3-wire one hot phase encoder (a speed-independent solution)

Table II
SYNTHESISED PHASE ENCODING CONTROLLERS

Circuit family	# output wires	# data symbols	# control signals	# literals in solution	synthesis time
One hot phase encoders	3	6	6	21/27*	<10 ms
	4	24	24	160	220 ms
	5	120	120	1225	160 s
Binary phase encoders	3	6	3	21	<10 ms
	4	24	5	107	20 ms
	5	120	7	477	420 ms
Matrix phase encoders	3	6	3	15	<10 ms
	4	24	6	28	<10 ms
	5	120	10	45	<10 ms
	6	720	15	66	<10 ms
	7	5040	21	91	<10 ms

* refers to the speed-independent decomposition (Section VIII)

a particular synthesis problem while the last two columns describe the solution size (in the number of literals, which closely correlates with area of the decomposed gate-level implementation) and the total synthesis time (which mostly consists of logic minimisation performed by ESPRESSO [15]).

The largest synthesis instance among the presented is 5-wire one hot phase encoder: it took almost 3 minutes to synthesise. This can be explained by the fact that the controller has $5! = 120$ one hot control signals blowing out the optimisation search space. It is also reflected in the huge physical implementation size making the controller hardly practical.

The most efficient controllers are synthesised using the matrix encoding scheme (Subsection IV-C). The specification and implementation sizes grow cubically w.r.t. the number of wires, even though the number of controller's behavioural scenarios grows exponentially. Synthesis times are non-measurable because the obtained CPOGs do not require any logical optimisation. This presents a practical justification of theoretical claims of CPOG model efficiency [11].

The family of binary phase encoders occupies the middle-ground: controller sizes and synthesis times grow linearly w.r.t. the number of scenarios (the number of different phase encoded symbols). Existence of a generic binary solution whose size grows polynomially w.r.t. the number of wires is still an open question for future research.

The benchmarks demonstrate that the presented generic solutions are more scalable than the previously published. Another advantage of the proposed methodology is an opportunity to improve the robustness of the solution by its speed-independent decomposition at the expense of the resultant controller area.

Acknowledgement

This work was supported by EPSRC grants EP/F016786/1 and EP/C512812/1.

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, 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] Alex Kondratyev and Kelvin Lwin. Design of asynchronous circuits using synchronous CAD tools. *IEEE Design and Test of Computers*, 2002.

[9] Art Lew. *Computer Science: A Mathematical Introduction.* Prentice-Hall, 1985.

[10] Andrey Mokhov and Alex Yakovlev. Conditional partial order graphs algebra. Technical report, Newcastle University, September 2008.

[11] 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.

[12] D. Muller and W. Bartky. A Theory of Asynchronous Circuits. In *Proc. Int. Symp. of the Theory of Switching*, pages 204–243, 1959.

[13] Steven Nowick. *Automatic Synthesis of Burst-Mode Asynchronous Controllers*. PhD thesis, Stanford University, 1993.

[14] Ivan Poliakov, Andrey Mokhov, Ashur Rafiev, Danil Sokolov, and Alex Yakovlev. Automated Verification of Asynchronous Circuits Using Circuit Petri Nets. In *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2008.

[15] Richard L. Rudell and Alberto L. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 6(5):727–750, 1987.

[16] Jens Sparsø and Steve Furber. *Principles of Asynchronous Circuit Design: A Systems Perspective.* Kluwer Academic Publishers, 2001.

[17] Ingo Wegener. *The Complexity of Boolean Functions*. Johann Wolfgang Goethe-Universitat, 1987.