Formal Modelling of Burst-Mode Specifications in a Distributed Environment

Alex Chan[1] (a.chan@ncl.ac.uk), Danil Sokolov[2], Victor Khomenko[1] and Alex Yakovlev[1]
[1]*Newcastle University, UK;* [2]*Dialog Semiconductor (Renasas), UK*

***Abstract*—Generalised fundamental mode is an important timing assumption for implementing digital circuits, where the environment is assumed to wait for the circuit to stabilise before producing new inputs. In particular, Burst-Mode (BM) timing assumption states that the circuit must wait until a complete input burst has arrived and the environment must wait until a complete output burst is produced. However, this timing assumption may be difficult to enforce in a distributed environment, if each part only observes a subset of the circuit's output burst.**

In this paper, we address the above by proposing two formal modelling methodologies: 1) Design by Signal Transition Graphs (STGs), and 2) Design by our new model called Burst Automata (BAs). STGs are flexible as they express many behaviours, while BAs extends the BM methodology and enables interoperability between many different models. Our experimental results show improved synthesis success rates and significant reduction in literal count.

***Index Terms*—burst-mode, speed-independence, distributed environment, signal transition graphs, burst automaton**

I. Introduction

Asynchronous circuits are a promising type of digital circuit that removes the use of a global clock signal in favour of local synchronisation between their components [1], resulting in many benefits including lower latency and lower power consumption [2], as well as resolving a host of clock-related issues like clock skew [3]. Notably, the operation mode of asynchronous circuits can be classified based on the timing assumptions made about the delays between their components and their interaction with the environment.

Generalised fundamental mode is a well-known circuit operation mode, where the environment is assumed to wait for the circuit to stabilise before producing any new inputs. In particular, fundamental mode can be classified as single input change or multiple input change, where the former forces sequential inputs but restricts the circuit's operation speed, while the latter allows one or more input changes after circuit stabilisation but makes the circuit more difficult to implement.

As a trade-off between the two modes, the BM timing assumption [4] was proposed, where signals may change in groups called bursts and are allowed to arrive in any order and time. Each burst consists of a non-empty set of inputs that precedes a finite set of outputs, such that the circuit must wait until a complete input burst has arrived before it produces any outputs and transitions to a different state. In turn, the environment must also wait until a complete output burst has been produced before it can send any new inputs. Note that BM-designed specifications must be well-formed where:

- Bursts must not contain an empty set of inputs.
- A given state must always be entered with the same input burst (unique entry condition).
- No input burst from a state can be a subset of another burst from the same state (maximal set property).

However, there are some caveats with the BM design. Namely, BMs cannot specify neither input-output concurrency nor choices between outputs (e.g. mutex), and their timing assumption may be difficult to enforce in a distributed environment, if each part only observes a subset of the circuit's output burst. This in particular makes composition and decomposition of BM specifications difficult, as signals are produced in bursts and the environment is assumed to be one. Moreover, while BM's maximal set property ensures specifications cannot enter an undetermined state, it also forbids non-deterministic (in the sense of language and automata theory) specifications.

Extended Burst-Mode (XBM) [5] was later proposed to enable input-output concurrency for BMs by introducing conditionals and directed "don't cares". Conditionals are level-sensitive inputs that determine a system's control flow based on their sampled values, while directed "don't cares" are monotonic signals (i.e. signals that only changes once), which are relatively slowly and may take several cycles to switch but enables input-output concurrency. Like BM-designed specifications, XBM-designed specifications must also be well-formed where the above and following properties are held:

- Bursts from the same state must have unique conditional values, or their input set (including "don't cares") is not a subset of another burst (distinguishability constraint).
- Every burst must contain a compulsory input (i.e. an input that does not appear nor terminate as a "don't care").
- All directed "don't cares" must toggle the signal when it terminates (i.e. if the signal was $+/-$ before appearing as a "don't care" then it must next terminate with $-/+$).

Still, XBMs cannot specify choices between outputs, and the issue with their inherent BM timing assumption in a distributed environment remains. Furthermore, their distinguishability constraint also forbids non-deterministic specifications.

Another well-known circuit operation mode is input-output mode, where inputs and outputs are said to be concurrent, and the environment is allowed to respond to a circuit's

This work was funded by Dialog Semiconductor (Renasas) via A×A project and Alex Chan's studentship, with partial support from EPSRC EP/N023641/1.

978-1-6654-7332-3/22/$31.00 ©2022 IEEE.

outputs without any timing constraints. In particular, Speed-Independent (SI) circuits [6] are an important class of asynchronous circuits that work correctly in input-output mode regardless of gate delays, and all wire delays are assumed to be negligible (or shorter than any gate delay). In Quasi-Delay-Insensitive (QDI) circuits [7], the assumption about wire delays is relaxed by requiring that (some) wire forks are isochronic, i.e. the maximal difference in delays from the root of the fork to the ends of its branches is negligible (or shorter than any gate delay). Intuitively, the isochronic fork assumption means that the fork can be characterised by a single delay, which conceptually can be appended to the driving gate's delay, making QDI very similar to SI. Thus, these two modes do not have to be distinguished for this paper.

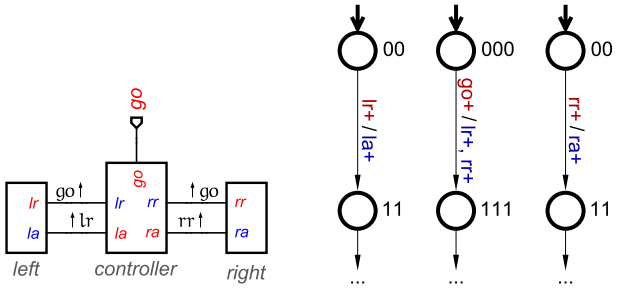
Signal Transition Graphs (STG) [8], [9] are a formal model that can be synthesised into SI circuits. STGs are Petri Nets [10] whose transitions are labelled by the rising and falling edges of signals, where signals can be subdivided into inputs, outputs, internals, and dummies with the latter behaving like the empty word ε in language theory. STGs are known for their flexibility as they can specify input-output concurrency, choices between any types of signals, and non-determinism. STGs are easy to compose and decompose, especially when using high-level asynchronous concepts [11], and enjoy good tool support for synthesis and verification from established tools like PETRIFY [12] and MPSAT [13], which are both integrated in the visual framework WORKCRAFT [14].

In this paper, we address the issues of BM timing assumption in a distributed environment, and extend the BM methodology to handle distributed systems. To achieve this, BMs have been generalised to a model called Burst Automata (BAs). Unlike BMs, BAs are a proper extension of Finite State Machines (FSMs), which allows them to capture input-output concurrency and other behaviours that are inherent in distributed systems. The proposed methodology has been implemented as a plugin for WORKCRAFT. By developing a translation from BAs to STG, we enable interoperability between many types of models including BMs, BAs, STGs, and FSMs.

The contents of this paper includes a simple example that highlights the issues of BM timing assumption in a distributed environment (Section II), design of the simple example using an STG and a BA with details on the translation from BAs to STGs (Section III), and experimental results that show a significant reduction in literal count when specifications are designed as an STG rather than as an XBM (Section IV).

II. MOTIVATION

In this section, we highlight the issues of BM timing assumption in a distributed environment. Here, we provide a simple example consisting of a controller with a two-part environment called left and right respectively in Figure 1. Part 1a shows the example as a circuit block diagram, while part 1b shows it as three separate XBM models that communicate with each other via common signals. For simplicity, we have omitted details about the reset phases of all signals, and assume input go is asserted by another part of the environment.



(a) Circuit block diagram (b) BM fragments

Fig. 1: Simple example of controller and two-part environment

Firstly, let us consider the scenario where the environment is one, i.e. the environment is not distributed and consist of both the left environment and right environment. When the controller receives input go+, it can produce its outputs lr+ and rr+. Following the BM timing assumption, the controller must then wait until it receives the next input burst (i.e. inputs la+ and ra+) from the environment, which the environment only produces after receiving lr+ and rr+ from the controller.

Now, let us consider the scenario described above where the environment is split into two, i.e. the environment is distributed, and both the left environment and right environment are no longer aware of each other. Suppose the controller has already received its input go+ and only produces output lr+ from its output burst lr+, rr+ due to rr being slower than lr. Then, it is possible for the left environment to receive lr+ early and produce la+ before the right environment even receives rr+, as the left environment does not know anything about the right environment. This in particular causes two issues:

1) Violation of BM timing assumption: Given that the left environment can receive lr+ and produce la+ before the controller finishes producing rr+ for the right environment, this contradicts the BM timing assumption where it is assumed the environment must wait until the circuit stabilises (i.e. finishes producing its output burst) before producing a new input burst.

2) Early arrival of inputs that are not yet expected: Let us assume that the controller's output rr was not delayed, and that the right environment has received its input rr+ but has not yet produced its output ra+. If the controller receives its next input burst from another part of the environment (e.g. la+ from left) and produces a subsequent output burst that can trigger the right environment to change (e.g. rr-), then this will cause right to unexpectedly receive another input before it even completes producing its output ra+ causing an incorrect behaviour. Note that the controller is aware of all parts of the environment, meaning it can receive signal(s) from one part of the environment and send signal(s) to another part.

To avoid the two issues that are described above, one might consider globalising the BM timing assumption assumption for the whole environment. However, there is no easy way to force parts of the environment to wait for other parts of the environment, as these parts could have been designed independently from each other and may not even necessarily

be an XBM. This means one cannot simply synthesise the XBM and all parts of the environment separately, and then just connect them without requiring some higher level of synchronisation. Alternatively, one might consider decomposing the problematic signals (e.g. the controller's output burst lr+, rr+) such that signals from one part of the environment is no longer expected to wait for signals from another part of the environment. However, this decomposition cannot be done for the BM methodology, as it assumes the environment is one where signals must be produced in bursts and output bursts can only be produced after the arrival of an input burst. Note that the method in [15] can decompose XBMs, although this decomposition is based on the number of reachable cycles that can be found meaning bursts are not decomposed and the environment is still assumed to be one.

Despite one may argue that it is too troublesome to use a distributed environment rather than its centralised counterpart and that one can simply just design the environment as a single model, there are still many benefits that a distributed environment can provide over their centralised counterpart. For example, suppose that the distributed environment behaves correctly and that it confirms with its given specification (i.e. the environment will not produce any unexpected signal for the specification). Then, the benefits that one can achieve includes:

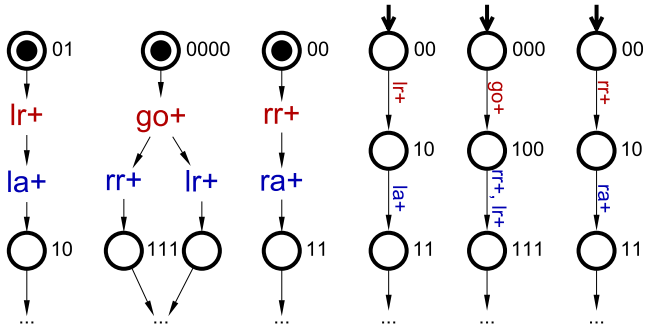
1) Smaller-sized Models: Like correctness of a model, it is also important to ensure that the model is easy to understand and manageable for a designer, especially when an issue arises. For example, if there was an issue with one part of an extremely large environment, then the designer must check every part of the environment to identify the problematic part. Thus, by splitting the environment into several parts, it becomes easier for the designer to identify the problematic part as each part becomes independent of each other. Additionally, if one still requires a centralised environment, then one may compose all parts of the environment accordingly.

2) Localised timing assumptions: By splitting the environment, each environment part now only manages the timing of its own signal changes without needing to wait for the signal changes by the other parts of the environment. This in particular enables input-output concurrency, as different parts of the environment may begin producing their own outputs without waiting for the inputs by other parts of the environment. Moreover, this can help optimise the speed of the environment, as there is no longer a timing constraint between the environment parts that may only be partially dependent on each other.

III. Debursting Burst-Mode Specifications

In this section, we address the two issues that were identified in our simple example, where it was possible that the BM timing assumption can be violated and that inputs may arrive too early such that they are not yet expected, by proposing to:

1) Design specifications by using the traditional STG model.
2) Design specifications by using our proposed BA model, which extends the BM methodology.



(a) STG fragments (b) BA fragments

Fig. 2: Simple example designed with other models

A. Design by Signal Transition Graphs

As described in Section I, STGs are known for their flexibility as they can easily specify input-output concurrency, choices between any types of signals, and non-determinism, while also being supported by well-established tools PETRIFY and MPSAT. Additionally, STGs are also easy to compose and decompose, making synchronisation and desynchronisation of signals, like outputs lr+ and rr+ in the simple example, trivial.

To illustrate the benefits of the STG model, Figure 2a shows the same simple example from Section II designed as an STG. Visually, one may notice the structure of the STG model strongly resembles their XBM counterpart, with the only differences being that STG transitions are not grouped into bursts (though, one can interpret them as explicit bursts), and that places corresponding to states are made implicit, if the place has exactly one incoming arc and one outgoing arc (e.g. the places corresponding to state $s1$ was made implicit). Semantically, STGs have richer behaviour than XBMs, as they can express behaviours that XBMs can express (e.g. XBM's well-formedness properties) and behaviours that XBMs cannot express (e.g. non-determinism and output choices). Thus, this makes STGs more desirable than XBMs, as they have a simpler design and can express more behaviours. Moreover, STGs are not constrained by the BM timing assumption meaning it can avoid the two issues that are described above.

B. Design by Burst Automata

Before discussing the design of BAs, it is important that we first provide their formal definition.

A BA can be defined as a tuple $B = (\Sigma, S, A, s_0)$ where:

- Σ is an alphabet of atomic actions.
- S is a finite set of states.
- $A \subseteq S \times 2^{\Sigma} \times S$ is the set of arcs.
- $s_0 \in S$ is the initial state.

Unlike XBMs, BAs can be interpreted as Finite State Machines (FSMs) [16] with arcs that label sets of actions including the empty set ($\varnothing$). In the definition above, the alphabet is not partitioned into inputs and outputs, and no directions ($+$ or $-$) are assigned to the actions. Instead, they can be viewed as refinements of the BA, where the BA can just be seen as a generic model. Additionally, there are no

restrictions like the maximal set property from BMs nor the distinguishability constraint from XBMs, and arbitrary non-determinism is allowed, i.e. it is possible for two distinct arcs to originate from the same state and be labeled by sets of actions, where these sets are in the subset relation or are equal. Moreover, it is possible to have an empty set of actions as an arc label, which can be interpreted as an ε transition. Thus, one can see that BAs naturally extend FSMs, where the latter are just BAs with its arcs labelled by singletons or $\varnothing$.

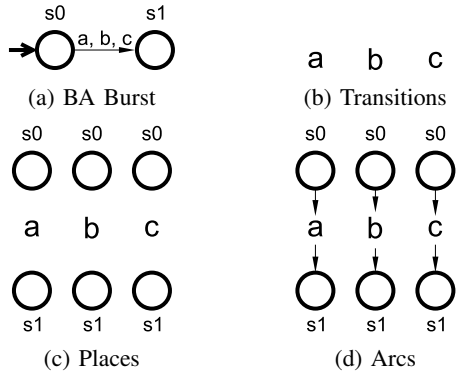
In XBMs, bursts are comprised of a set of inputs followed by a set of outputs, while in BAs, these bursts can be modelled by explicitating the intermediate state between when all inputs have arrived and no outputs have been produced, i.e. a BA needs two steps to fire an XBM-like burst. Note that the graphical notation could hide this intermediate state and be similar to the XBM notation. On the other hand, this allows the possibility to mix inputs and outputs in the BA arc labels, which in particular enables input/output concurrency.

To also illustrate the benefits of the BA model, Figure 2b shows the same simple example from Section II designed as a BA. Visually, one can see that the structure of the BA model is exactly the same as their XBM counterpart. Indeed, the only exception is that the intermediate state between the XBM's input burst and the XBM's output burst is made explicit, as explained above. Semantically, BAs are different from XBMs and are in fact closer to the semantics of STGs, e.g. BAs can express non-determinism (due to no maximal set property nor distinguishability constraint, and modelling of $\varnothing$), and both input-output concurrency and choices between signals (due to their alphabet not being partitioned into inputs and outputs, and their segregated bursts). Thus, this also makes BAs more desirable than XBMs, as they can express much more behaviours than XBMs, naturally extends FSM without complicated semantics, and provides the same design as XBMs with the option to fall back to XBM semantics as needed. Furthermore, BA's modelling of bursts allows outputs to be properly desynchronised like in STGs, enabling composition and decomposition of BAs.

C. Translating Burst Automata to Signal Transition Graphs

In addition to the benefits that BAs provide, it also provides a framework that enables interoperability between different models, e.g. one may translate FSMs like BMs and XBMs into equivalent STGs and vice versa. Below, we present a translation from XBMs to STGs through our BA framework, which produces an STG that is exponential in size to the original XBM in the worst case. Note that this worst case scenario is rare for practical specifications, and that this exponential explosion only occurs when there are many arcs incoming to and/or originating from the same state and are labelled with large bursts, e.g. if there are m incoming arcs and n outgoing arcs labelled with non-empty bursts of cardinalities $k_1, k_2, \ldots, k_m$ and $k'_1, k'_2, \ldots, k'_n$ then $k_1 \cdot k_2 \cdot \ldots \cdot k_m \cdot k'_1 \cdot k'_2 \cdot \ldots \cdot k'_n$ STG places are created. Furthermore, this translation does not include any dummy transitions, which may be particularly beneficial for verification and synthesis tools as there are fewer or no ε-transitions to handle, e.g. MPSAT uses STG unfoldings and preserves ε-transitions.

To begin, we provide an example of a simple BA that contains a generic burst of a, b, c in Figure 3a to be translated. Note that the steps for translating bursts with signals are exactly the same as demonstrated in the following example, but signals are instead translated into corresponding signal transitions (e.g. inputs as input signal transitions and outputs as output signal transitions).



(a) BA Burst (b) Transitions (c) Places (d) Arcs

Fig. 3: Translation from BAs to STGs

Transitions (Figure 3b) For each arc $a = (u, D, v) \in A$ in the BA, a transition is created for every burst label element, such that:

- If the arc's label is $\{\sigma_1, \sigma_2, ..., \sigma_k\}$, where $k \geqslant 1$, then k STG transitions labelled σ_1, σ_2, ..., σ_k are created.
- If the burst's label is $\varnothing$, then an STG transition labelled ε is created.

Thus, this defines

$$T_a := \begin{cases} \{(a, d) \mid d \in D\} & \text{if } D \neq \varnothing \\ \{(a, \varepsilon)\} & \text{if } D = \varnothing \end{cases}$$

as the set of transitions that corresponds with a.

Places (Figure 3c) For every state s in the BA, a set of places in the STG is created as follows. Suppose that the bursts labelling the arcs incoming to s are $In_1, In_2, \ldots, In_k$, and the bursts labelling the arcs outgoing from s are $Out_1, Out_2, \ldots, Out_{k'}$, where $k, k' \geqslant 0$ and empty bursts are encoded as $\{\varepsilon\}$, a new place is created for each tuple in the Cartesian product $In_1 \times In_2 \times \cdots \times In_k \times Out_1 \times Out_2 \times \cdots \times Out_{k'}$. If $k = 0$ and $k' = 0$ then this Cartesian product contains the empty tuple as the only element, and if s is the initial state of the BA then the set of places that corresponds with s are initially marked.

Arcs (Figure 3d) For each place p that corresponds with a BA state s and a tuple $(i_1, \ldots, i_k, o_1, \ldots, o_{k'})$, the following arcs are created:

1) From the i_j-labelled transition (which may be ε-labelled transition in case of an empty burst) in the jth incoming burst to p, for each $j \in \{1, \ldots, k\}$.
2) From p to the o_j-labelled transition (which may be ε-labelled transition in case of an empty burst) in the jth outgoing burst, for each $j \in \{1, \ldots, k'\}$.

D. Implementation of Extended Burst-Mode Features

Now, let us consider the translation of XBM components described in [17]. There, the translation method shows that conditionals are translated into elementary cycles with places, which acts like locks to prevent the signal transitions from firing after a compulsory input transition has fired, and "don't cares" are translated into explicitly delayed STG transitions, where they are connected from the first burst they appear in and are then connected to the burst they terminate in.

In BAs, we do not include conditionals and "don't cares" as they can be seen as XBM-exclusive features. Instead, we independently translate both conditionals and "don't cares" using the method in [17] from our BA to STG translation, and attach them to the resulting STG by connecting them to the appropriate transitions and/or places of the corresponding burst. Note that these connections are connected in the same way that has been described in [17], and that the examples shown there included dummy transitions, which our BA to STG translation does not include. Indeed, by following the transition contraction methods described in [18], one can see that our BA to STG translation is a contracted variant of the translation in [17], where all dummy transitions labelled as forks and joins have been contracted.

One important exception to consider is the contraction of dummy transitions that include read arcs, e.g. the read arcs that are connected between the elementary cycle's lock places and the fork-labelled dummy transitions in [17]. Read arcs are a shorthand expression for an arc connecting from a component x to a component y and an arc connecting from y to x, which is drawn as a line between two components (e.g. a line between places/transitions and transitions/places in STGs). In [18], it is stated one cannot contract a transition if it contains a read arc. This means that one must modify the connection of these read arcs to be connected to and from the corresponding input transitions (which will not affect the firing order of these inputs) instead of these fork-labelled transitions, before they can be contracted. For an example of these translations, Figure 4a shows the translation of conditionals and Figure 4b shows the translation of "don't cares", where the top models are XBMs and bottom models are STGs.

Another point of interest is the XBM's empty output bursts. In [4], [5], it is said that bursts with an empty set of outputs are interpreted as "dummy outputs" to signify a system change internally. For the XBM to STG translation in [17], empty output bursts are translated into so-called fake output transitions, where these fake outputs are inserted at every input-only burst (usually starting with a + transition), and its opposite-edged transition is inserted after one of the inputs in the input-only burst has toggled its value. However in BAs, there is no need to translate empty output bursts. Indeed, these empty output bursts are equivalent to the $\varnothing$-labelled arcs, which can simply be interpreted as ε transitions and so, can be excluded when the BA is translated into an STG (i.e. if an input-only burst is followed by an input-output burst in the XBM, then the input transitions of the input-only burst are connected to the input transitions of the input-output burst in the STG). Although, there are still some scenarios where these fake outputs are needed to avoid some potential complete state coding (CSC) conflicts, e.g. when a signal transition $x+$ ($x-$) is directly followed by its opposite-edged transition $x-$ ($x+$). Note that there must be no CSC conflicts for an STG to be synthesised into a circuit. Thus, one can optionally include these fake output transitions after the BA is translated into an STG (though, it is also possible to include these fake outputs during the translation process). Unlike [17], these fake outputs have been mapped to one of the input transitions in the input-only burst, such that the fake output will always appear in burst that the mapped input appears in to ensure consistency. For example, Figure 4c shows the translation of an input-only burst where $fake0+$ ($fake-$) is mapped to every $in0+$ ($in0-$).

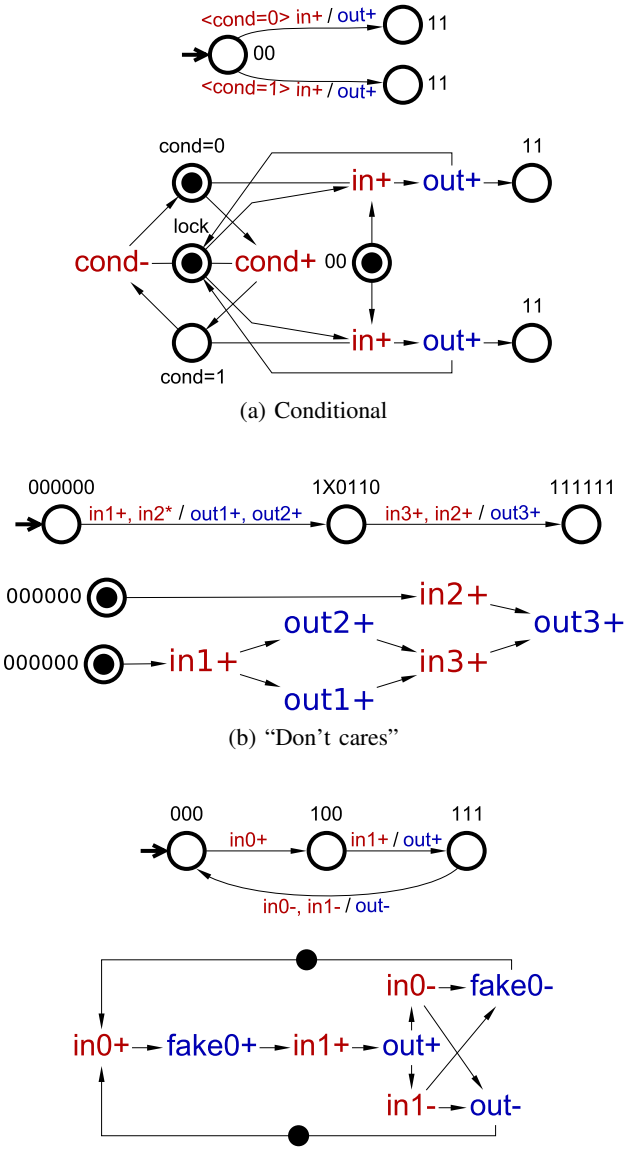


Fig. 4: Translation of XBM components

TABLE I: Literal Count Comparison

Specification	Sigs	Literal Counts (+Overhead%)	
		MINIMALIST/3D	PETRIFY/MPSAT
ack-xbm-si	7	X / 26 (+37%)	24 (+27%) / **19**
biu-dma2fifo	6	X / 44 (+92%)	24 (+5%) / **23**
biu-fifo2dma	6	X / F	23 (+5%) / **22**
c-element	3	5 / 5	5 / 5
concur-mixer	6	15 / 15	15 / 15
diffeq-alu1	8	49 (+59%) / 39 (+26%)	**31** / 32 (+4%)
diffeq-mul1	6	X / 28 (+34%)	**21** / 27 (+29%)
diffeq-mul2	6	X / 13	13 / 13
dme	6	21 (+50%) / 20 (+43%)	**14** / 15 (+8%)
dme-fast	7	27 (+80%) / 27 (+80%)	20 (+34%) / **15**
dram-ctrl	14	41 (+25%) / 40 (+22%)	37 (+13%) / **33**
fifocellctrl	4	X / 10 (+12%)	11 (+23%) / **9**
gcd-controller	10	X / 177 (+91%)	F / **93**
hp-ir	5	8 / F	8 / 8
hp-ir-it-ctrl	12	46 (+18%) / **39**	40 (+3%) / 48 (+24%)
hp-ir-rf-ctrl	11	37 (+24%) / F	F / **30**
imec-alloc-outb	7	23 (+44%) / 21 (+32%)	**16** / 17 (+7%)
martin-qelement	4	9 (+29%) / 9 (+29%)	**7** / **7**
nowick-basic	5	10 (+25%) / 10 (+25%)	9 (+13%) / **8**
token-distributor	8	42 (+62%) / 39 (+50%)	28 (+8%) / **26**
pe-send-ifc	8	81 (+66%) / 52 (+7%)	50 (+3%) / **49**
po-sbuf-send-ctl	6	28 (+8%) / 31 (+20%)	**26** / **26**
pscsi-ircv	7	27 (+59%) / 27 (+59%)	19 (+12%) / **17**
pscsi-isend	7	55 (+67%) / 61 (+85%)	**33** / **33**
pscsi-trcv	7	23 (+28%) / 23 (+28%)	19 (+6%) / **18**
pscsi-trcv-bm	8	38 (+32%) / 35 (+21%)	**29** / 33 (+14%)
pscsi-tsend	7	43 (+35%) / 45 (+41%)	46 (+44%) / **32**
pscsi-tsend-bm	8	52 (+45%) / 50 (+39%)	40 (+12%) / **36**
scsi-isend-bm	9	47 (+31%) / 50 (+39%)	43 (+20%) / **36**
scsi-isend-csm	9	47 (+31%) / 50 (+39%)	42 (+17%) / **36**
scsi-trcv-bm	9	55 (+38%) / 45 (+13%)	**40** / 41 (+3%)
scsi-trvc-csm	9	46 (+65%) / 38 (+36%)	32 (+15%) / **28**
scsi-tsend-bm	9	76 (+112%) / 50 (+39%)	**36** / 39 (+9%)
scsi-tsend-csm	9	41 (+71%) / 36 (+50%)	33 (+38%) / **24**
tangram-mixer	6	**8** / **8**	**8** / 10 (+25%)
two-ticks-if	6	13 (+19%) / 11	12 (+10%) / 11
Average Overhead		38% / 33%	8% / 3%

IV. EXPERIMENTAL RESULTS

In our experiment, we used the listed benchmarks to compare the literal counts of their synthesised result between the XBM tools (MINIMALIST and 3D) and the STG tools (PETRIFY and MPSAT via WORKCRAFT). All results that have been synthesised by XBM tools are BM circuits, while all results synthesised by STG tools are SI circuits. Our benchmarks include specifications found in MINIMALIST [19], [20] (*concur-mixer*, *dme*, *dram-ctrl*, *hp-ir*, *token-distributor*, *pe-send-ifc*, *pscsi*, *scsi*, *tangram-mixer*) and in 3D [5] (*biu-dma2fifo*, *diffeq*, *fifocellctrl*), as well as other specifications from various publications (*ack-xbm-si* [21], *biu-fifo2dma* [5], *c-element* [22], *gcd-controller* [23], *imec-alloc-outbound* [24], *nowick-basic* [4], *po-office-sbuf-send-ctl* [25]). Note that all published XBMs were translated into STGs using our BA model, and all XBMs of published STGs were newly designed using our WORKCRAFT plugin [17]. To ensure fairness, we have also factorised the literal counts produced by MINIMALIST and 3D using LOGIC FRIDAY (a graphical frontend to the ESPRESSO logic minimiser [26]) before counting them, as they were originally in the sum-of-products form.

In the table, from the leftmost column to the rightmost column, we have included the name of the specification, the number of signals in the specification, the literal counts of the synthesised circuits produced by the XBM tools, and the literal counts of the synthesised circuits produced by the STG tools. Note that under the column **MINIMALIST/3D**, all literal counts on the left side of the forward slash symbol (/) were from the circuits produced by MINIMALIST and all literal counts on the right side of the forward slash symbol were from circuits produced by 3D. Similarly, under the column **PETRIFY/MPSAT**, all literal counts on the left side of the forward slash symbol were from the circuits produced by PETRIFY and all literal counts on the right side of the forward slash symbol were from circuits produced by MPSAT.

For each row, the smallest literal count is highlighted in bold, and if a literal count is not the smallest in their row then they are included with a percentage overhead over the smallest literal count in their respective row. Additionally, a result marked with an 'X' means that the tool was not able to read the specification due to incompatible syntax, e.g. MINIMALIST was developed to support both BMs and XBMs but it only received support for the former and so it cannot read the XBM extension, and a result marked with an 'F' means that there was a synthesis failure. For example, let us consider the third row that includes the specification *biu-fifo2dma*: Here, the specification has 6 signals, and it could not be synthesised with MINIMALIST due to an incompatible syntax nor 3D due to a synthesis failure. On the other hand, the specification could be synthesised with PETRIFY and MPSAT with a literal count of 23 and 22 respectively. As the synthesis result by PETRIFY was not the smallest, a percentage overhead is also included. This is calculated by dividing the difference between the non-smallest and smallest literal counts with the smallest literal count, i.e. $\frac{(23-22)}{22}$.

For the final row, the average overhead of each synthesis tool is included. This is calculated by totalling the percentage overhead of each column and dividing them by the tool's total number of successful synthesis results. For example, let us consider the average overhead for 3D: Here, we total up all the percentage overheads and divide them by the total number of non-X and non-F results, i.e. $\frac{(37+92+\ldots+39+50)}{(36-3)}$.

When evaluating our experimental results, we found that most of the specifications that were synthesised with PETRIFY and MPSAT had a lower literal count than when they were synthesised with MINIMALIST and 3D, despite many of them originally being XBMs. In particular, simpler specifications like *c-element*, *concur-mixer*, and *nowick-basic* shown little to no improvement, as they were already the most optimal result. However, more complicated specifications like *dme-fast*, *scsi-tsend-bm*, *biu-dma2fifo*, and *gcd-controller* all shown substantial improvements. Note that WORKCRAFT supports both PETRIFY and MPSAT, so one can synthesise the specification with both tools and select the more optimal result.

Additionally, there were only a few specifications that required the implementation of fake outputs to be synthesiseable with PETRIFY and MPSAT. These included *biu-fifo2dma*, *pe-send-ifc*, *po-sbuf-send-ctl*, and *two-ticks-if* where they all had a signal transition $x+$ ($x-$) followed by an immediate signal

transition of $x-$ ($x+$), which causes a CSC conflict. This suggests that fake outputs do not have to be implemented for every empty output burst like it was stated in in [17].

Finally, all of the benchmarks could be synthesised as an STG, when we used MPSAT. This suggests that most XBMs can be designed as an STG and remain synthesiseable without needing the BM timing assumption.

V. Conclusion

In this paper, we addressed the issues with BM timing assumption in a distributed environment by proposing to: 1) design specifications by using the traditional STG model, and 2) design specifications by using our proposed BA model, which extends the BM methodology.

In our motivation, we provided a simple example of a controller with a two-part environment and highlight the benefits of a distributed environment over a singular environment. In the simple example, the BM timing assumption was shown to be violated if the controller's output to the right environment was slow, such that the left environment can proceed without waiting for right. Additionally, we also highlight that it was still possible that the right environment may not be ready to receive its next input (e.g. if the controller's output to right was fast, but right was still producing its previous output burst).

Next, we then shown an STG and a BA of the simple example, and highlighted their benefits over XBMs. STGs are more flexible than XBMs providing both a simpler design and more behaviours like input-output concurrency and non-determinism, while BAs provides the same behaviours as STGs and a framework that enables interoperability between many types of models, e.g. translation from XBMs to STGs. Details of translation from BAs to STGs were also covered including step-by-step translations of each BA component and each XBM-exclusive component, where the latter are translated independently and then subsequently attached to the resulting STG. Moreover, unlike XBMs, both BAs and STGs are a true extension to FSMs, allowing them to easily specify input-output concurrency and other behaviours from distributed systems.

Lastly, experimental results demonstrate an improvement to the synthesis success rate and a significant reduction in literal count, when specifications were designed as STGs instead of as XBMs. Furthermore, all specifications were still synthesiseable as an STG (with MPSAT) suggesting most BM specifications can be designed as an STG and remain synthesiseable without the BM timing assumption.

Future Directions: We are considering further research into the modelling of distributed environments, where we investigate the possibility of allowing multiple timing assumptions to co-exist.

Acknowledgment

We would like to thank Prof. Steve Nowick for his advice with MINIMALIST, and the anonymous reviewers for their feedback on this paper.

References

[1] J. Sparso and S. Furber, *Principles of Asynchronous Circuit Design*. Springer, 2002.
[2] C. J. Myers, *Asynchronous Circuit Design*. John Wiley & Sons, 2001.
[3] C. V. Berkel, M. Josephs, and S. Nowick, "Applications of Asynchronous Circuits," *Proc. IEEE*, vol. 87, no. 2, pp. 223–233, 1999.
[4] S. Nowick and D. Dill, "Synthesis of Asynchronous State Machines Using a Local Clock," in *Int. Conf. Computer Design (ICCD)*, 1991, pp. 192–197.
[5] K. Yun and D. Dill, "Automatic Synthesis of Extended Burst-Mode Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 101–132, 1999.
[6] D. Muller and W. Bartky, "A Theory of Asynchronous Circuits," in *Int. Symp. Theory of Switching*, 1959, pp. 204–243.
[7] A. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Computing*, vol. 1, pp. 226–234, 2005.
[8] L. Rosenblum and A. Yakovlev, "Signal Graphs: From Self-Timed to Timed Ones," in *Int. Workshop on Timed Petri Nets*, 1985, pp. 199–206.
[9] T. Chu, "Synthesis of Self-Timed VLSI Circuits From Graph-Theoretic Specifications," Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Tech. Rep., 1987.
[10] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
[11] J. Beaumont, A. Mokhov, D. Sokolov, and A. Yakovlev, "High-level asynchronous concepts at the interface between analog and digital worlds," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 61–74, 2018.
[12] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers," *IEICE Trans. Information and Systems*, vol. E80-D, pp. 315–325, 1997.
[13] V. Khomenko, M. Koutny, and A. Yakovlev, "Logic Synthesis for Asynchronous Circuits Based on Petri Net Unfoldings and Incremental SAT," in *Int. Conf. Application of Concurrency to System Design (ACSD)*, 2004, pp. 16–25.
[14] "Workcraft," [Online]. Available: https://workcraft.org/, 2006, accessed: 15-07-2022.
[15] M. Agyekum and S. Nowick, "A cycle-based decomposition method for burst-mode asynchronous controllers," in *Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2007, pp. 129–142.
[16] G. H. Mealy, "A method for synthesizing sequential circuits," *The Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.
[17] A. Chan, D. Sokolov, V. Khomenko, D. Lloyd, and A. Yakovlev, "Synthesis of SI Circuits from Burst-Mode Specifications," in *Design, Automation and Test in Europe Conf. (DATE)*, 2021, pp. 366–369.
[18] V. Khomenko, M. Schaefer, W. Vogler, and R. Wollowski, "STG Decomposition Strategies in Combination with Unfolding," *Acta Informatica*, vol. 46, pp. 433–474, 2009.
[19] S. Nowick, "Minimalist homepage," [Online]. Available: http://www.cs.columbia.edu/~nowick/minimalist/minimalist.html, 1999, accessed: 15-07-2022.
[20] "CaSCADE tools," [Online]. Available: http://www.cs.columbia.edu/~nowick/asynctools/, 2007, accessed: 15-07-2022.
[21] H. Jacobson, C. Myers, and G. Gopalakrishman, "Achieving fast and exact hazard-free logic minimization of extended burst-mode gc finite state machines," in *IEEE/ACM Int. Conf. on Computer Aided Design (ICCAD)*, 2000, pp. 303–310.
[22] "Design of c-element," [Online]. Available: https://workcraft.org/tutorial/design/c_element/start, 2014, accessed: 15-07-2022.
[23] D. L. Oliveira, D. Bompean, L. Faria, T. Curtinhas, and N. Alles, "Design of asynchronous digital systems using two-phase bundled-data protocol," in *2012 VI Andean Region Int. Conf.*, 2012, pp. 73–76.
[24] J. De San Pedro, T. Bourgeat, and J. Cortadella, "Specification mining for asynchronous controllers," in *Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2016, pp. 107–114.
[25] A. Davis, B. Coates, and K. Stevens, "The post office experience: Designing a large asynchronous chip," in *Proc. of the Twenty-sixth Hawaii Int. Conf. on System Sciences*, vol. i, 1993, pp. 409–418 vol.1.
[26] R. Brayton, A. Sangiovanni-Vincentelli, C. McMullen, and G. Hachtel, "Logic minimization algorithms for vlsi synthesis."